



Professional Expertise Distilled

HTML5 Enterprise Application Development

A step-by-step practical introduction to HTML5 through the building of a real-world application, including common development practices

Nehal Shah
Gabriel José Balda Ortíz

[PACKT] enterprise 
PUBLISHING professional expertise distilled

www.it-ebooks.info

HTML5 Enterprise Application Development

A step-by-step practical introduction to HTML5 through the building of a real-world application, including common development practices

Nehal Shah

Gabriel José Balda Ortíz



BIRMINGHAM - MUMBAI

HTML5 Enterprise Application Development

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2013

Production Reference: 1120213

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84968-568-9

www.packtpub.com

Cover Image by Mark Holland (m.j.g.holland@bham.ac.uk)

Credits

Authors

Nehal Shah
Gabriel José Balda Ortíz

Reviewers

Santiago Martín-Cleto
Kevin Roast

Acquisition Editor

Erol Staveley

Lead Technical Editor

Ankita Shashi

Technical Editors

Jalasha D'costa
Worrell Lewis

Project Coordinator

Leena Purkait

Proofreader

Maria Gould

Indexer

Rekha Nair

Production Coordinator

Nilesh Mohite

Cover Work

Nilesh Mohite

About the Authors

Nehal Shah is a technology director with over 10 years of experience building high performance teams and creating software spanning from frontend to backend and everything in between. He earned his BA in psychology at the University of Chicago and his MS in Information Technology at University of North Carolina at Charlotte. He currently is Executive Director of Engineering at Kaplan Test Prep and leads an emerging solutions team that builds innovative web and mobile products to capture new markets. For more information, check out his website at nehaliium.com.

First and foremost, I want to thank Gabriel for joining me on this journey. Your enthusiasm and determination kept me going page after page and your talent and work ethic never cease to amaze me.

Secondly, I would like to thank my family for pushing me to where I am today. From my mom and dad buying me my first computer, to my sister forcing me to learn my multiplication tables, to my brother teaching me to write my first program, I am indebted to you all for everything I have.

Lastly, a special thank you goes to my wife, Shilpa, who continues to look out for me, to encourage me, and to push me to be better than I am.

Gabriel José Balda Ortíz is a computer engineer and graduate from Simón Bolívar University in Venezuela. After obtaining his degree, he studied graphic design in Centro Art, Venezuela. Since 2003, Gabriel has been developing web applications for multiple enterprises including various freelance projects. In 2011, he moved to New York to work on educational applications for Kaplan Test Prep. You can see his portfolio at gabrielbalda.com.

I wish to express my sincere gratitude to Nehal for including me on this project, and Bernardo Rodriguez for inviting me to be part of his entrepreneurial adventures.

Special thanks to my mother and my grandmother, they provided me the education and love that made me the person I am today.

Finally, I want to thank my wife, Cindy, the love of my life, for being with me on this journey, giving me her support and helping me to be a better man every day. Part of this book is dedicated to her.

About the Reviewers

Santiago Martín-Cleto started his career as web designer for an outstanding Spanish media group in 1999. Passionate about design and code, he considers himself a web standards and open source advocate. He has been very involved in huge projects for mass media and financial corporations as a contribution to launch start-ups. As a frontend developer, he is specialized in high traffic websites performance issues.

Kevin Roast is a frontend software developer with 15 years of professional experience and a lifelong interest in computer science and computer graphics. He has developed web software for several companies and is a founding developer at Alfresco Software Ltd. He is very excited by the prospect of the HTML5 standardization of the web and the progress of web browser software in recent years. He was co-author of the book *Professional Alfresco: Practical Solutions for Enterprise Content Management* published by Wrox, and has been a technical reviewer on several HTML5-related books.

I would like to thank my wife for putting up with me tapping away in the evenings reviewing book chapters and to my three kids Alex, Ben, and Izzy for being little wonders.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Instant Updates on New Packt Books

Get notified! Find out when new books are published by following [@PacktEnterprise](https://twitter.com/PacktEnterprise) on Twitter, or the *Packt Enterprise* Facebook page.

Table of Contents

Preface	1
Chapter 1: HTML5 Starter Kit: Compatibility	17
The real meaning of compatibility	17
Browsers	18
Rendering engine	19
JavaScript engine	20
OS platforms	22
Display resolution	23
Importance of compatibility	24
Patching the differences – compatibility libraries	25
HTML5 Shiv	25
Modernizr	27
Explorer Canvas	31
HTML5 Boilerplate	32
Before starting app development	32
Summary	34
Chapter 2: HTML5 Starter Kit: Useful Tools	35
Choosing editors and IDEs	35
Adobe Dreamweaver CS6	36
Aptana Studio 3	38
BlueGriffon 1.5.2	40
Maqetta	41
eXo	42
Cloud9	42
Choosing web servers	43
Apache	43
Apache Tomcat	44
Jetty	44

Tornado	44
nginx	44
LightTPD	45
Node.js	45
Prepackaged stacks	45
Web browsers and add-ons	45
Mozilla Firefox	46
Google Chrome	48
Safari	50
Internet Explorer	51
Opera	51
HTTP proxies	52
Charles	53
Fiddler	53
Summary	53
Chapter 3: The App: Structure and Semantics	55
Understanding page structure	56
Navigation list	58
Secondary content	58
Metadata	59
Microdata	62
Favicons and icons	64
CSS3 resets	68
Individual sides	70
Shorthand	71
Sticky footer	74
General styling	77
Responsive web design and adaptive web design	82
Importing CSS files using media queries	83
Importing other CSS from our main CSS	83
Using media queries as conditionals in our main CSS	83
Summary	86
Chapter 4: The App: Getting Movies Via Geolocation	87
How it works	88
The API	88
A simple request	89
Movies near you	90
Self-invoking	92
That becomes this	92
Getting location	94

Getting postal codes	97
AJAX ain't just a cleaning product	99
From postal codes to showtimes	102
Summary	106
Chapter 5: The App: Displaying Movie Data via CSS3	107
<hr/>	
Back to the browsers' babel tower	107
CSS3 Magic – adding more styles to MovieNow	109
Adding rounded corners	109
Setting color	110
Red, green, and blue	111
Red, green, blue, and alpha	111
Hue, saturation, and lightness	111
Hue, saturation, lightness, and alpha	112
Adding gradients	112
Adding box shadows	115
Adding text shadows	118
Some tricks to fake 3D	120
Movies and styles	122
Styling our list	125
Transitions	127
Animations	127
Choosing between transitions and animations	128
Using media queries	133
Applying CSS3 selectors	135
Summary	140
Chapter 6: The App: Trailers via HTML5 Video	141
<hr/>	
Introducing HTML5 video	141
Implementing a video player	143
Custom controls	146
Styling	148
Adding interactions using JavaScript	155
Possible improvements	168
Still not perfect	168
Introducing HTML5 audio	169
Implementing an audio player	169
Custom controllers	170
Styling	170
How I learned to stop worrying and love Flash	171
Summary	171

Chapter 7: The App: Showing Ratings via Canvas	173
Charting	173
Preparing our code	174
Everything depends on the context	178
2D context	179
An overview of the Canvas 2D Drawing API	180
Drawing charts	182
3D context – WebGL and experimental WebGL	185
Entering a tridimensional world	186
Three.js	187
Summary	198
Chapter 8: The App: Selection UI via Drag-and-Drop	199
Adding showtimes	199
Styling showtimes	202
What a drag	204
Handling drag with JavaScript	206
Drop it	207
Toggling the drop zone	209
Transferring some data	210
Displaying the results	211
Summary	214
Chapter 9: The App: Getting the Word Out via Twitter	215
Registering our application	216
How to tweet in MovieNow?	220
Authenticating	220
User not logged and/or application not authorized	222
User logged in	224
Adding some styles	226
Posting tweets	229
Service	229
Applying HTML	231
Adding more styles	232
Adding JavaScript interactions	238
Form validation support across browsers	244
New input fields types	245
Summary	246
Chapter 10: The App: Consuming Tweets Via Web Workers	247
Getting the data	248
Capturing geocodes	248
Anatomy of a Web Worker	249

Using Web Workers to get nearby tweets	251
Updating the event listener	253
Styling the tweets	256
Summary	258
Chapter 11: Finishing Up: Debugging Your App	259
<hr/>	
What to look for	260
Which tools to use	260
Playing with HTML and CSS	262
Step by step with JavaScript	266
JavaScript console	269
Analyzing load times	270
JavaScript profiling	271
Mobile debugging	271
Web debugging proxies	273
Summary	275
Chapter 12: Finishing Up: Testing Your App	277
<hr/>	
Types of testing	277
Unit testing	278
Setting up your unit test	278
Invoking your target	279
Verifying the results	280
Frameworks and tools	280
JsTestDriver	280
QUnit	282
Sinon.JS	282
Jasmine	283
Functional testing	283
The Selenium standalone server	285
The php-webdriver connector from Facebook	286
PHPUnit	286
Browser testing	289
Continuous integration	290
Summary	291
Chapter 13: Finishing Up: Performance	293
<hr/>	
Web Performance Optimization (WPO)	293
Following standards	294
Optimizing images	294
Optimizing CSS	295
JavaScript performance considerations	298
Additional page performance considerations	300
Server-side considerations	300

Table of Contents

Performance analytics	301
Load times	302
Profilers	303
Summary	305
Index	307

Preface

HTML5, apart from being the latest buzzword in the Internet era, is quickly becoming the *lingua franca* for the web. In fact, HTML5 is the first version of HTML to get its own logo (<http://www.w3.org/html/logo>). To understand the significance of that, one must first know a little history.

HTML



A brief history of time (client-server edition)

Enterprise application development over the decades has been a pendulum swinging back and forth between terminal and mainframe, between client and server. In the 1980s, business logic was largely pushed to the server by "dumb terminals" or "thin clients" which did very little except act as a middleman between the user and the server. Beginning in the 1990s, logic started to swing to the client with "fat clients" bearing the processing burden. With the introduction of the World Wide Web in 1991, a new breed of thin client emerged. The pendulum swung once again. Or did it?

The shift between client and server has largely been driven by cost and power. Early on, investment was made in powerful, costly servers. As PCs became more powerful in terms of memory and processing ability, as well as lower in cost, it became possible to build applications that could be distributed more easily, allow for offline capabilities, and best of all require less powerful (and less costly) server infrastructures. However, maintaining, upgrading, and deploying "fat clients" created a new burden.

Web-based applications eventually emerged to solve the problems of cost, power, and maintainability. At first, they appeared to be much like their "thin client" predecessors: merely middlemen between users and servers but without the deployment overhead. However, with the introduction of technologies such as Java and JavaScript, the "thin client" began to put on a little weight. Before long, the processing burden began to bleed to the client as applets, scripts, and plugins became more commonplace and with them the maintainability problem reappeared. Rather than managing distributions of applications, the problem shifted to managing distributions of applets, scripts, and plugins.

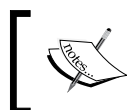
The situation was bifurcated by the introduction of "rich clients". Business logic became tiered. A separation of concerns became the norm. Let the server deal with server stuff. Let the client deal with client stuff. The problem with this, however, is that the client took some time to be able to handle the client stuff the world needed of it.

A brief history of time (web browser edition)

When Tim Berners-Lee introduced his World Wide Web browser to his CERN colleagues in 1990, only a glimmer could be seen of what it would one day become. Marc Andreessen would introduce Mosaic and graphical browsing in 1993 and Netscape would soon follow in 1994. After that, Microsoft jumped in with Internet Explorer in 1995. Pretty soon, the first browser wars would come and go with Internet Explorer emerging as the victor and the remnants of Netscape coalescing around Mozilla and Firefox, which was released in 2002. In the 2000s, Apple released Safari and Google released Chrome.

Web browsers however have seldom held parity with one another. The existence of web standards and a governing body called the W3C notwithstanding, browser makers have nevertheless played to their own tunes. In the beginning, implementations of HTML varied widely and the trend continued through HTML 4.0, XHTML, CSS, and JavaScript. The variance in implementations and behavior made web development difficult and time consuming.

To combat the limitations and inconsistencies of the web browser, technologies based on plugins such as Adobe Flash began to flourish continuing where browser-based technologies such as HTML, CSS, and JavaScript left off. For years, many websites consisted mostly – and sometimes entirely – of technologies such as Flash; such plugins even afforded better performance. The notion of **Rich Internet Applications (RIAs)** prevailed as the pendulum swung back to fatter clients.



See Google's "The Evolution of the Web" for an interactive graphic on web browsers and their implementation of modern browser features at <http://evolutionofweb.appspot.com>.

The inclusion of plugin-based technologies became a red herring for the promise of the World Wide Web. While connectivity of content was a hallmark of the original principles of HTML, content was represented by tags such as `embed`, `object`, and `applet` where application modules are embedded onto a web page constituted black boxes that hid content from the semantic web.

Web browsers nevertheless evolved. JavaScript frameworks such as jQuery emerged to abstract browser differences and offer up richer interactivity. CSS techniques emerged to overcome the limitations and inconsistencies between the browsers. Newer browsers emerged with better support for web standards.

However, something was missing. Even though applications were being developed using browser-based technologies, many application features were left out of the browser. Consistent mechanisms to add video/audio playback, offline capabilities, and even browser history management were missing. Flash was still looked upon as filling in the missing pieces of the web.

Finally, browser development coalesced around HTML5 in 2009 when XHTML 2.0 was abandoned in lieu of something more backward compatible with earlier versions of HTML. HTML5 sought to address two major areas in the browser: the need for a more consistent, semantic markup language and the demand for more natively-supported browser features. When it was introduced in 2004, it heralded a set of APIs to make the browser into a true application development platform and thus more semantically amenable.

Features of HTML5



- **Media API:** This embeds, plays, and pauses multimedia content
- **Text Track API:** This reads properties of the text tracks of multimedia content
- **Drag and drop API:** This natively makes elements draggable by setting an attribute
- **Offline Application Cache:** This saves data locally for offline use
- **History API:** This asserts more control of the back button
- **Canvas API:** This literally draws all over the web in 2D and 3D
- **Cross Document Messaging:** This overcomes cross-site scripting limitations
- **Microdata:** This adds more semantic content for search engines to find
- **MIME Type and Protocol Handler Registration:** This extends applications with handlers for additional protocols
- **Web Workers:** This spawns threads independent of user interaction
- **Web Storage:** This stores data on the client
- **Web Sockets:** This sends two-way messages between server and client

With modern browsers' increasing support for HTML5, reliance on plugin-based technologies is starting to give way to browser-based implementations. With the APIs that allow for better control of the experience, the client is finally beginning to catch up to its promise. Mobile web browsers have especially become a catalyst for this. Since Adobe Flash is not supported on devices such as the iPhone and iPad, and since implementation of HTML5 features on Safari have grown, HTML5 is quickly becoming a standard for mobile web application development. However, if this trend is to continue, browser makers and application developers must adhere to the developing standards for HTML5, which brings us back to the logo. To right the wrongs of the past, HTML5 must have collective agreement on implementation. In order to inculcate this, there is a burgeoning movement to enforce standards in web browsers and applications, and speed up implementation as adoption looms. The HTML5 logo is emblematic of that effort.



The **Web Hypertext Application Technology Working Group (WHATWG)** formed in 2004, evolved HTML and conceived of HTML5 as the next step in the evolution of the HTML standard. At that time, the W3C was working on the XHTML 2.0 standard; however, in 2009, the W3C decided to halt this effort and join the WHATWG in its effort to develop HTML5.

In January 2011, it announced that the HTML5 standard would be referred to as "HTML" and that the specification would henceforth be a living document.

In December 2012, the **World Wide Web Consortium (W3C)**, the international web standards body, announced that HTML5 is feature complete. Although not a standard yet, it finally gives browser makers a stable target upon which to develop HTML5 features.

It's all about semantics

HTML5 makes an attempt to codify information on the web in a more cohesive way than before. With previous versions of HTML, content is structured based on how it should be displayed rather than its inherent meaning. The `div` tag is often used, but what does a `div` tag really mean? To get around this, application developers have broken up their content using `id` attributes based on standards and best practices of web presentation.

For example, application developers use tags such as the following:

```
<div id="header">  
<div id="footer">
```

The obvious problem is that what gets used for the `id` attribute need not follow a standard. One application developer could use `id="header"` while another uses `id="head"`. In order to standardize structure based on semantics, HTML5 introduces a set of new tags that takes the vagaries out of the process.

HTML5 introduces a set of new top-level tags, which can be broken down into the following categories: content, language, layout, and format.

Content tags

The content tags introduced in HTML5 define how new types of content can be embedded into a web page. Content such as sound, video, and graphics are surfaced in much the same way text and images have been for so many years.

- `audio`: This tag is used for embedding sound content. Before HTML5, either some browsers implemented support for audio inconsistently or a special player typically developed using Adobe Flash would have been required to play sound. HTML5 removes that dependency and makes the audio player a consistent function of the web browser itself.
- `video`: This tag is used for embedding video content. Like with audio, there was inconsistent support or a special player was required to play video content. Web browsers that support the `video` tag have a built-in video player.
- `canvas`: This tag allows for basic 2D to be drawn via JavaScript. 3D graphics are not consistently supported, however, and are optional.

Language tags

With internationalization taking on more and more precedence, localization has been a special challenge for web developers. HTML5 introduces a set of new tags to help make content more accessible to larger audiences.

- `bdi`: This tag defines the directionality of text. This is typically used to support languages that are read right-to-left.
- `ruby`: The `ruby` tag in conjunction with the `rt` and `rp` tags defines the Ruby annotation for East Asian typography.

Layout tags

HTML5 comes with a set of first-class tags that not only help with laying out the page, but also allows for the page to be broken up altogether. With HTML5, web developers have the ability to share sections of content in a more standard way:

- **header**: This tag defines the header of the page or of a section or article.
- **footer**: This tag defines the footer of the page or of a section or article.
- **nav**: This tag defines the menu structure of the website. These are the navigational links used to semantically break up the website.
- **section**: This tag defines sections of a page. The `article` and `aside` tags are in a way specialized section tags.
- **aside**: This tag defines the sidebar content for a page. Often, a web page is broken up with ancillary content pushed to the side.
- **article**: This tag defines the main content for a page. While tags such as `section`, `aside`, `header`, and `footer` define ancillary content to the page, the `article` tag identifies the portion of content that is considered to be the focal point. Typically, this content is unique to the URI.

Format tags

HTML5 introduces a new set of special tags, which define how areas of content can be identified and formatted appropriately.

- **figure**: This tag identifies non-contiguous content that is layered into a body of text. For example, it can be used to wrap diagrams, charts, and graphs that are referenced by a body of text.
- **details**: This tag defines content that can be toggled as visible or hidden. It is geared towards showing and hiding content based on a user action such as help-related content. Web developers have built a variety of solutions to do this and, with HTML5, the web browser takes care of it.
- **hgroup**: This tag wraps the well-known `h1-h6` tags into a cohesive structure. When headings are related, `hgroup` shows that they are related. For example, for an article with a title and subtitle, the title would be wrapped in an `h1` tag while the subtitle would be wrapped in an `h2` tag. The `hgroup` tag around them signifies that the `h2` tag is associated with the `h1` tag and not part of the document outline.
- **wbr**: This tag defines a word break opportunity. Typically, lines of text are broken up by spaces. The `wbr` tag allows for the web developer to specify where in a set of contiguous non-space characters line breaks can be introduced when there is no room to display it all on one line.

- `progress`: This tag indicates the progress of a task and can be used in conjunction with JavaScript to display a progress bar to the user.
- `time`: This tag is a microformat tag that allows one to specify semantically that something is a date or time.
- `meter`: This tag is a format tag to define a scalar measurement with a known range.
- `mark`: This tag indicates text that is relevant to the user. Typically, this would be used for highlighting specific words within a passage.

Forms get an upgrade

Forms in HTML5 get a whole new set of functionality to allow for better validation of content and ease of use.

The following tags are new with HTML5:

- `datalist`: This tag works similarly to a `select` tag with the added feature of being able to type ahead to select items in the list.
- `keygen`: This tag generates a key pair for use in forms. This is typically used for client authentication.
- `output`: This tag indicates the result of a calculation. It is associated with a `form` tag to display simple calculations to the user especially when used in conjunction with the new form input types. In addition, the `input` tag gets a new set of types. The following input types are new with HTML5:
 - `color`: This type displays a color picker, which submits a hex code for that color.
 - `date`: This type displays a date picker, which submits a date.
 - `datetime`: This type displays a date and time picker, which submits a date and time including time zone.
 - `datetime-local`: This type displays a date and time picker, which submits a date and time without time zone.
 - `email`: This type displays a field for entering e-mail addresses.
 - `month`: This type displays a month-year picker, which submits a month and year.
 - `number`: This type displays a field constrained for entering numeric values.
 - `range`: This type constrains the user to select numbers within a range. Typically, this will display as a slider.

- `search`: This type displays a search field.
- `tel`: This type displays a field that constrains the user to typing in a valid telephone number.
- `time`: This type displays a time picker.
- `url`: This type displays a field that constrains the user to typing in a valid URL.
- `week`: This type displays a control for picking a week within a year.

Enter microdata

HTML5 adds the ability to define custom semantics for your content. Similar to microformats in previous versions of HTML, in which a set of predetermined attributes would allow you to ascribe the semantic meaning of content, microdata allows you to create your own semantic language to ascribe to your content. While microformats rely on generic attributes such as `class` and `rel`, microdata introduces `itemscope`, `itemtype`, and `itemprop` to describe content. The `itemscope` and `itemtype` attributes allow you to define a custom type and indicate where it is defined. The `itemprop` attribute points to a specific property within the definition:

```
<div itemscope itemtype="http://mysite.com/Movie">
  <h1 itemprop="title">Your Favorite Movie</h1>
  <p itemprop="summary" >
    A summary of your favorite movie.
  </p>
</div>
```

An anatomy lesson

Now that we know many of the new tools for building a page in HTML5, let us dive into what a page looks like.

A word about DOCTYPE

The `DOCTYPE` declaration in HTML documents has always been a signal to the browser of the standards to which the document adheres. If a web browser knows the standards used for the document, it can more efficiently process and render that document. SGML-based markup languages require this.

In order to simplify the `DOCTYPE` tag, HTML5 has only one type:

```
<!DOCTYPE html>
```


Unlike previous versions of HTML, which required references to the specific DTD being followed, HTML5 is not based on SGML and thus the DTD reference is not required.

The lang attribute

HTML5 introduces a simplified `lang` attribute for specifying the language of a page. In XHTML, an `xmlns` attribute was required, but HTML5 does not require this.

Metatags are important too

HTML5 adds a new metatag called `charset`. This specifies the specific character encoding of the document. It otherwise uses all the metatags from HTML 4.01.

HTML5 includes support for the `viewport` metatag. This metatag defines how the web page should be viewed and includes parameters such as width and height. It also allows you to define zoom settings such as initial scale, and minimum and maximum scale. It even allows for the ability to target a specific density DPI in case you want to change how your page looks based on different screen resolutions.

Putting it all together

A basic HTML5 page will look like the following code:

```
<!doctype html>
<html lang="en">
<head>
  <title>My First HTML5 Page</title>
  <meta charset="utf-8">
  <meta name="description" content="My first HTML5 page.">
  <meta name="author" content="Me">
</head>
<body>
</body>
</html>
```

We will of course add more to this as we go on.

The application

For much of this book, we will be building a mobile web application that illustrates many of the features of HTML5. The application is called MovieNow, and will be a one-stop shop for finding, reviewing, and booking movies near you. The features we will develop in this book are as follows:

- Find movies near you using geolocation
- Display movie data to the user
- View trailers using the `video` tag
- Display reviews using the `canvas` tag
- Select movies using the drag and drop API
- Integration with external APIs
- Display tweets near you via Web Workers

What this book covers

In the following chapters, we will build a variety of features of HTML5 into our MoveNow enterprise application.

Chapter 1, HTML5 Starter Kit: Compatibility, discusses support of HTML5 features across multiple web browsers and devices as well as ways to skirt the deficiencies of these browsers. The main driver for supporting multiple browsers is ensuring access to enterprise web applications across multiple devices while maintaining a consistent user experience.

Chapter 2, HTML5 Starter Kit: Useful Tools, provides a guide to getting started with HTML5 enterprise application development including available tools, their installation, and their use. A comprehensive evaluation of the business drivers for selecting tools will be discussed.

Chapter 3, The App: Structure and Semantics, walks you through setting up the boilerplate for the MovieNow enterprise application. We will set up the overall page structure, discuss semantic tags in depth, and talk about microdata.

Chapter 4, The App: Getting Movies via Geolocation, begins the MovieNow enterprise application by introducing geolocation. We will walk you through the geolocation API and how to use it to implement useful features.

Chapter 5, The App: Displaying Movie Data via CSS3, covers layout and features of CSS3 including some interesting CSS3 effects. We will also cover best practices in defining standard styles across web applications and devices including media queries and compatibility considerations for CSS3.

Chapter 6, The App: Trailers via HTML5 Video, introduces the video and audio tags and their use within an HTML5 enterprise application. We will talk about manipulating the playback of multimedia content via JavaScript as well as backward compatibility with browsers that do not support the HTML5 video and audio tags.

Chapter 7, The App: Showing Ratings via Canvas, walks through HTML5 canvas and using the drawing API to display graphics in your enterprise applications. We will use the drawing API to create ratings charts for our MovieNow application.

Chapter 8, The App: Selection UI via Drag-and-Drop, covers the drag-and-drop API. We will implement the drag-and-drop functionality in the MovieNow enterprise application demonstrating event delegation and the publish-subscribe pattern.

Chapter 9, The App: Getting the Word Out via Twitter, discusses forms and form validation in HTML5 by integrating with the Twitter API. We will implement posting of tweets from within the MovieNow application.

Chapter 10, The App: Consuming Tweets via Web Workers, demonstrates Web Workers and the power of external APIs to bring social elements to enterprise application. We will integrate tweets near you into the MovieNow application.

Chapter 11, Finishing Up: Debugging Your App, talks about ways of debugging HTML5 enterprise applications. We will discuss the browser console and HTTP proxies.

Chapter 12, Finishing Up: Testing Your App, covers tools for testing HTML5 enterprise applications. We will cover functional test suites and automated tools.

Chapter 13, Finishing Up: Performance, discusses performance, which is a crucial topic for any HTML5 enterprise application. We will talk about strategies and tools and walk through profiling your HTML5 application.

What you need for this book

You will need prior knowledge of web application development as this book introduces developers already familiar with the basics of the web including HTML, CSS, and JavaScript to the advantages of HTML5 and CSS3.

Who this book is for

This book is primarily aimed at application developers who have some experience developing applications for the web, and want to extend their knowledge to the latest developments in HTML5 and CSS3. Upon completion of this book, readers will have a thorough understanding of the toolset that HTML5 provides to develop enterprise applications.

Conventions


In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.


Code words in text are shown as follows: "Copy `html5shiv.js` from the `dist` folder to your JavaScript folder".

A block of code is set as follows:

```
<div class="geolocation-available">
  Congrats! Your browser supports Geolocation!
</div>
<div class="no-geolocation">
  Your browser doesn't support Geolocation :(
</div>
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on the **GENERATE!** button".

 Warnings or important notes appear in a box like this.]

 Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/5689_graphics.pdf.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

HTML5 Starter Kit: Compatibility

In the *Preface*, we covered the general structure of an HTML5 document, but before jumping into development, we must first talk about addressing the time consuming issue of compatibility across browsers and platforms. In this chapter, we will cover how web browsers work and the strategies to support HTML5 across multiple web browsers and devices. By the end of this chapter, you will be able to follow an initial plan of action to consistently support your enterprise application's functionality, interface, and user experience.

We will cover the following topics:

- The real meaning of compatibility
- Browsers
- OS platforms
- Display resolution
- Importance of compatibility
- Patching the differences - compatibility libraries

The real meaning of compatibility

In an ideal world, HTML, CSS, and JavaScript should be interpreted in the same way across all browsers and platforms. While the **World Wide Web Consortium (W3C)** has developed standards for such technologies, browser makers have implemented them in their own ways. This means that although you can use W3C standards for developing enterprise applications, it is possible for inconsistencies to arise between different browsers and platforms.

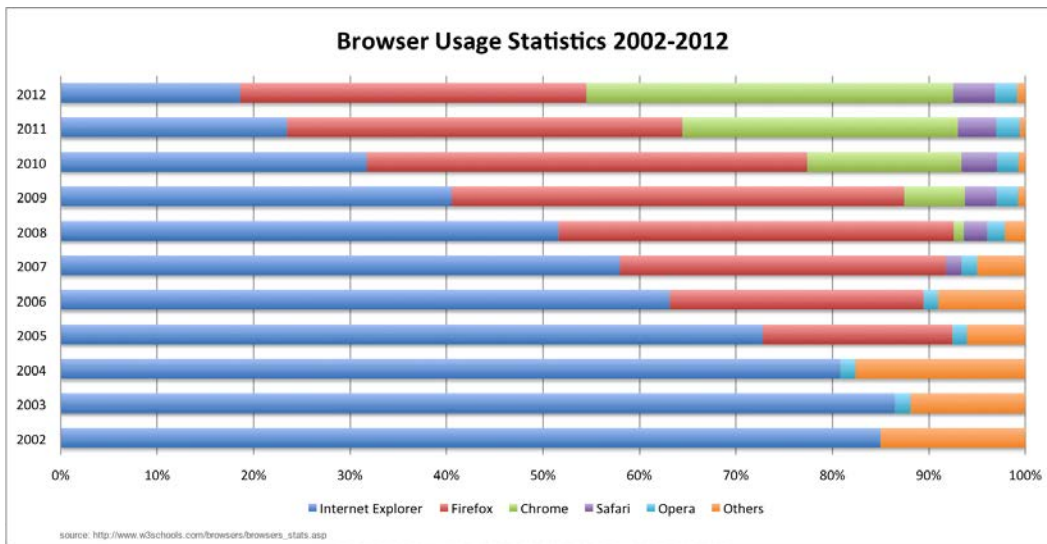
Compatibility does not mean that the experience of the user should be the same in every client, but it needs to maintain certain consistencies. For example, if we have a list of users in an application, it is good practice to have different inputs depending on the platform. We can use the scroll bar to navigate the list on any desktop client. For touch devices, gestures might be preferable.

On the other hand, we need to be careful with platform restrictions because sometimes it is not technically possible to support the same functionality in every device or client. A particular instance that illustrates this is the audio and video volume management in iOS devices (until Version 5.1.1). It is not possible to control the volume using JavaScript in Safari for iOS. In such cases, it is preferable to hide the volume control for iOS devices.

To better understand the issues of compatibility, it is important to understand the evolution of the World Wide Web in relation to the capabilities of the client that renders the final product, the operating system or platform, and the screen resolution.

Browsers

Since the release of the World Wide Web, there has always been competition for dominance in usage share in the browser marketplace. By 2001, Internet Explorer controlled over 90 percent of the browser market after Netscape ceased to be a major contender, but with the release of Version 1.0 of Mozilla Firefox in November 2004 and Google Chrome in September 2008, it began to see a new crop of competition.



As of June 2012 however, Google Chrome has become the most used browser at just 32.76 percent market share. It now shares an ever crowded space with Mozilla Firefox, Internet Explorer, Safari, and Opera including mobile counterparts. In addition, each one of these has its own list of versions, and we need to decide in some cases from which version we require support for our applications after knowing that newer versions are always around the corner.

Let's peek behind the scenes a bit to understand the complexity behind the diversity of browsers and versions. Each browser has two major software components: a rendering engine and a JavaScript engine.

Rendering engine

Also known as the **layout engine** or **web browser engine**, the rendering engine is responsible for parsing the markup content (HTML) and the style information (CSS, XSL, and so on), and generating a visual presentation of the formatted content including media files referenced (images, audio, video, fonts, and so on). It is important to know the many rendering engines out there because it can help you to recognize certain behaviors and deduce which browsers are going to behave in certain ways based on their rendering engine.

While Chrome and Safari use WebKit (developed by Apple, KDE, Nokia, Google, and others), Firefox uses Gecko (developed by Netscape/Mozilla Foundation), Internet Explorer uses Trident (owned by Microsoft), and Opera uses Presto.

With CSS, one can identify some exclusive rendering engine features (known as CSS extensions) by the prefix. WebKit-only features start with `-webkit-` and Gecko-only features with `-moz-`. Opera includes the `-o-` prefix while Internet Explorer 8 and up recognize `-ms-`.

Trident has a different approach. It recognizes common CSS properties with `*` or `_` as a prefix to override previous definitions (for example, `*color:#ccc;` and `_color:#ccc;` are not recognized by other rendering engines except Trident).

JavaScript engine

The JavaScript engine is the software component that interprets and executes JavaScript code in the browser. While the rendering engine is responsible for the visual presentation of HTML content using CSS styles, the JavaScript engine will interpret and execute JavaScript code.

Chrome employs the V8 engine, Firefox now uses Jägermonkey, Internet Explorer 9 features Chakra, Safari uses Nitro, while Opera substituted SunSpider with Carakan in 2010.

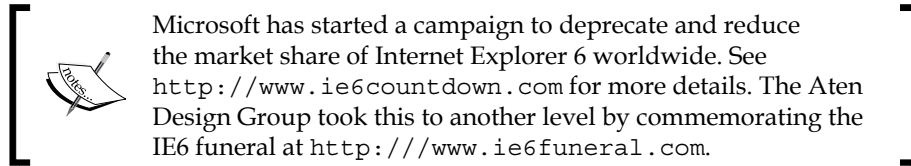


The rankings for what many consider the current browser war are largely driven by the speed of the JavaScript engine. While other engines rely on interpretation and compilation, V8 has no intermediate parser and it generates an assembler using its own runtime environment. In other words, it has a virtual machine. This has allowed Chrome to become one of the fastest browsers.

As a general rule (there are some exceptions), it is better to load HTML and CSS first and then JavaScript. This can be done by including the `<script>` tags that import JavaScript just before closing the `<body>` tag. The reason for this is that it is faster to render HTML and CSS than to interpret and execute JavaScript. Web pages will appear to load faster as a result.

Where it is not possible to include the `<script>` tags in the body, there are two attributes on the `<script>` tag that can be used to signal to the browser when the script should be downloaded. These are `async`, which was introduced in HTML5, and `defer`. The `defer` attribute literally does what it purports; it defers script execution until the page has been rendered. This way, the DOM is ready for your script. The `async` attribute signals to the browser to download the script asynchronously and without blocking the rendering engine and executes it when it is ready. Both execute before the `DOMContentLoaded` event. The key difference is that `defer` executes each script sequentially and `async` executes each script when it is ready. Typically, in order to support older browsers that do not support the `async` attribute, these attributes are used together so that browsers that do not perform asynchronously can fall back to `defer`.

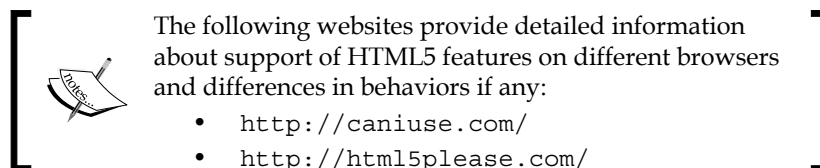
While there are many differences between browsers, it is important to be aware that inside the same browser category exists multiple versions whose HTML5 and CSS3 support vary widely. This is especially true for Internet Explorer. Proper support for HTML5 and CSS3 does not appear until Internet Explorer 9.



Most of the HTML5 and CSS3 capabilities are supported in the following browsers and versions:

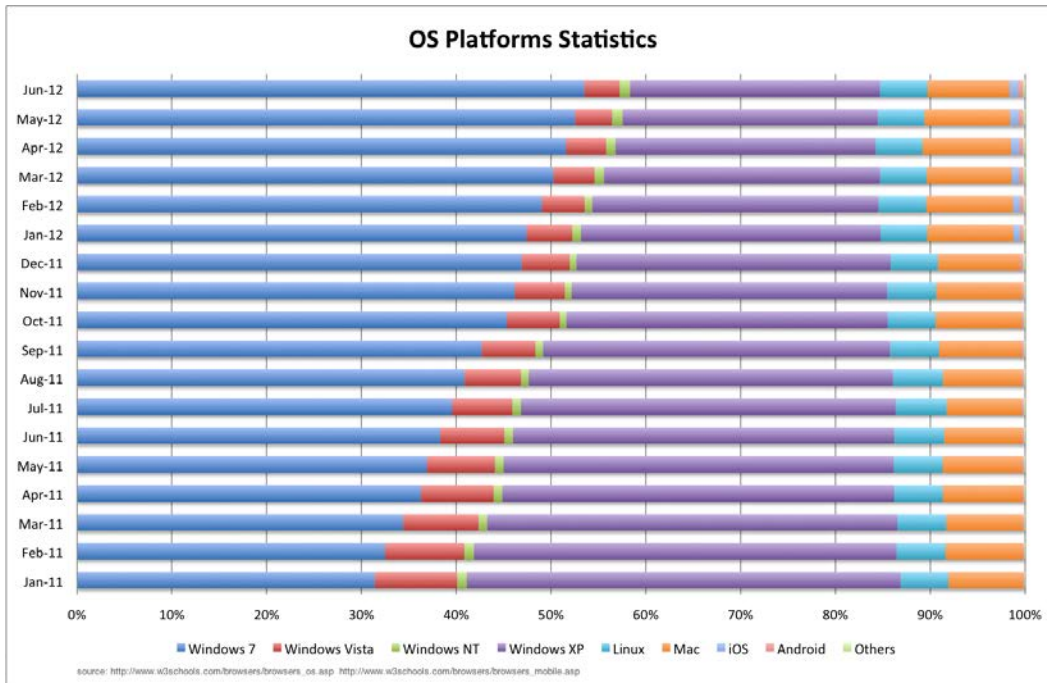
- Internet Explorer 9 and higher
- Firefox 7 and higher
- Chrome 14 and higher
- Safari 5 and higher
- Safari Mobile 3.2 and higher
- Opera 11 and higher
- Opera Mobile 5 and higher

Even so, there are still some features not supported and there are inconsistencies between the implementations. One interesting case study that reveals the lack of standards across the browsers is the use of video in HTML5. To use the native video capabilities of HTML5, the video file must be compressed using specific codecs. There are three major codecs that exist: **Ogg Theora** (royalty-free), **H.264** (free for end consumer but involves royalties for products that encode and decode), and **WebM** (royalty-free). As Firefox is oriented to use open source technologies, it initially supported only Ogg and WebM. Chrome currently supports all three codecs, but, for similar reasons as Firefox support for H.264, will be removed in subsequent versions (although it may continue support on mobile). Safari, Safari Mobile, and Internet Explorer 9 and higher support only H.264 by default, but you can install plugins to support Ogg and WebM (except on Safari Mobile).



OS platforms

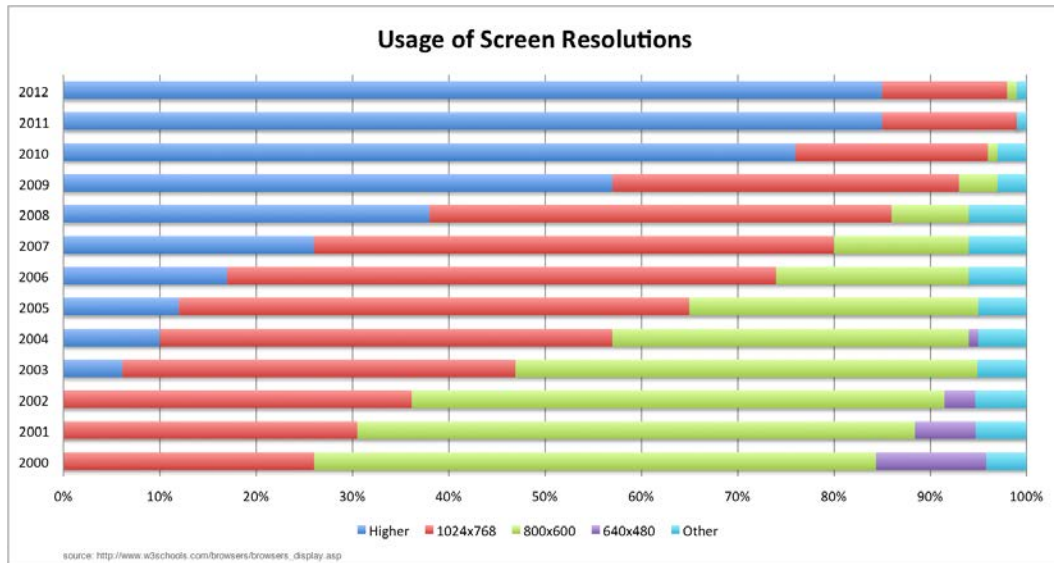
Any web application will be affected by the operating system (OS) on which it runs. The most noticeable difference is the appearance of user interface elements of the browser including the scroll bars, submit buttons, checkboxes, and so on. This is an important consideration because the size and shape of these elements can vary across multiple operating systems – even with the same web browser. In fact, some operating systems restrict some functionality, as was the case with `<input type="file">` which initially had not been supported on iOS prior to iOS6.



OS platform statistics show that Windows is by far the most used OS on the web. Mobile platforms, however, may take on more prominence in the years ahead considering the growing popularity of tablet devices and technical improvements of smartphones.

Display resolution

With so many devices on the market, screen resolution is quickly becoming an important factor for consideration when planning web applications. Android has a diversity of devices of many sizes and densities. According to the usage of screen resolutions, the advance of new hardware technologies has made it possible to increase the number of pixels on modern screens:



Even though most desktop systems now have a resolution higher than 1024 x 768, the rise of mobile technologies has created a paradox where lower resolution displays are retaking lost ground. The experience delivered by an enterprise application does not need to be – and in fact should not be – the same for all the devices. Viewing a page on a desktop screen at 1920 x 1200 can be vastly different from a mobile phone screen at 960 x 540 not only because of the resolution, but also because of the size of the device and readability (the ease in which text can be read and understood). It is sometimes important to detect the resolution to adapt the user experiences. To be sure, new techniques such as **responsive web design** are taking hold to address these issues.



As if there were not enough variables in the playfield, Apple introduced Retina Display in June 2010 with the iPhone 4, having a native resolution of 960 x 640. This technology is based on a higher pixel density that is beyond the human eye's capacity to see pixelation in images on the screen at a typical viewing distance. Although it was not so noticeable for web images in the iPhone 4, the new iPad and the new line of MacBook Pros released in 2012 with Retina Display create new requirements for web applications.


First, there are some web development techniques which determine if the client is a mobile device by using CSS to detect the device resolution. With the new iPad, 2048 x 1536 px resolution is not possible or at least not intuitive. The resolution of the new iPad is higher than the majority of desktops and laptops in the market. Secondly, to avoid the pixelated effect in any application viewed on the new iPad or the new MacBook Pro, it is necessary to include higher resolution images for these Apple devices and images in normal resolution for backward compatibility with all other devices.

Importance of compatibility

At this point, it is natural to ask why it is important to care about compatibility if one needs to only develop enterprise applications used internally within an organization, where a specific browser can be mandated. This attitude can be perilous for two reasons. First, businesses are moving quickly towards mobile delivery and controlling the platform is becoming thus less tenable. Secondly, constraining an organization in this way hampers its ability to update its application support capabilities as it couples the enterprise application too tightly to the choices in desktop support. If a company wanted to upgrade to a newer operating system or default web browser, constraining it by requiring that certain versions of browsers be supported can have undesired consequences down the line.

Patching the differences – compatibility libraries

In general, we want to support as many browsers as possible, so we are going to need a way to allow backward compatibility by implementing the capabilities not available on the browser, informing the user that the feature is not available, or modifying the user experience depending on the browser's capabilities. To this end, there are many JavaScript libraries that can help.


 For this chapter, styles and scripts will be included inline within the HTML file to simplify comprehension even though it is a good practice to have styles and scripts in separate files.

HTML5 Shiv


As already noted, Internet Explorer begins to support HTML5 tags in Version 9. **HTML5 Shiv** allows for support in previous versions. Also known as **HTML5 Shim**, it is an open source JavaScript library that enables styling for HTML5 elements in versions of Internet Explorer before IE 9. It accomplishes this by using `document.createElement("element")` to tell the browser that the tags exist.

Suppose that we are testing in Internet Explorer 8, and we have the following code:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
header{
  color:#ff0000;
  font-size:40px;
}
</style>
</head>
<body>
  <header>Hello HTML5!</header>
</body>
</html>
```

 **Downloading the example code**
You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

In Internet Explorer 8, the text will not display the style. HTML5 Shiv can be used to make this work.

 It is good practice to dive into libraries before you use them to understand exactly what they are doing. We encourage you to check out the HTML5 Shiv code at the following location: <https://github.com/aFarkas/html5shiv/blob/master/src/html5shiv.js>.


To install this library you need to perform the following steps:

1. Download the library from the following location: <https://github.com/aFarkas/html5shiv/zipball/master>.
2. Uncompress the file.
3. Copy `html5shiv.js` from the `dist` folder to your JavaScript folder (`js` in our case).
4. Insert the following code inside the `head` tag:

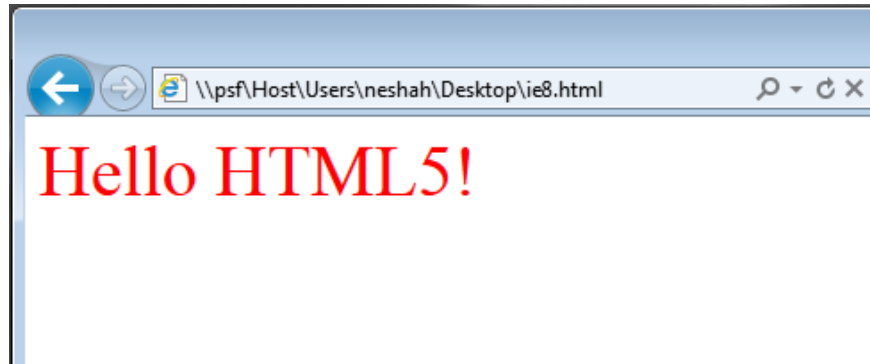
```
<!--[if lt IE 9]>  
<script src="js/html5shiv.js"></script>  
<![endif]-->
```

As a result our code should look as follows:

```
<!DOCTYPE HTML>  
<html>  
<head>  
<!--[if lt IE 9]>  
  <script src="js/html5shiv.js"></script>  
<![endif]-->  
<style>  
header{  
  color:#ff0000;  
  font-size:40px;  
}  
</style>  
</head>  
<body>  
<header>Hello HTML5!</header>  
</body>  
</html>
```

 Note that the `<!-- [if lt IE 9] >` conditional comment includes the JavaScript library only if the browser version is Internet Explorer prior to Version 9.

If you run this code in Internet Explorer 8, it will show a **Hello HTML5!** in red with a bigger font. This is one of the exceptions where we need to load a JavaScript library inside `<head>`, because we need to give the capability to recognize HTML5 elements to the browser before it begins styling.



Modernizr

Since not even the newest versions of web browsers support all HTML5 and CSS3 capabilities, it is useful to do some kind of detection to show a notification or change the behavior of a page when a feature is not supported. An old strategy was to look at the property `navigator.userAgent` to detect the browser's user agent and code based on the pertinent cases. As user agents change, it becomes difficult to keep pace and alter an application. An alternative approach to user agent detection is **feature detection**, wherein an application detects whether a particular feature is supported and reacts accordingly. The following is an example of feature detection:

```
function supports_video() {  
    return !!document.createElement('video').canPlayType;  
}
```

Modernizr is an open source JavaScript library that allows support for different levels of experiences based on the capabilities of each browser using a simple feature detection model. Additionally, Modernizr uses HTML5 Shiv, adding the ability to style HTML5 elements on Internet Explorer prior to Version 9.

Going back to our previous example:

```
<!DOCTYPE HTML>  
<html>  
<head>  
<style>  
header{
```

```
    color:#ff0000;
    font-size:40px;
}
</style>
</head>
<body>
  <header>Hello HTML5!</header>
</body>
</html>
```

Suppose that we want to implement the geolocation functionality using HTML5 to get the geographical position of the user, and we want to detect if it is available. For that we start using Modernizr:

1. Go to <http://modernizr.com/download/>.
2. Select the functionality that you want to validate and **html5shiv**. In this case, we are going to select the **Geolocation API** functionality under **Misc.**, **html5shiv v3.4** and **Add CSS Classes** present under **Extra**.
3. Click on the **GENERATE!** button.
4. Copy the source code generated.
5. Create a new JavaScript file (call it `modernizr.js`), paste the source code, and save it in your JavaScript folder.
6. Import the Modernizr library `<script src="js/modernizr.js" type="text/javascript"></script>` inside `<head>`. At this point the code should look as follows:

```
<!DOCTYPE HTML>
<html>
<head>
<script src="js/modernizr.js" type="text/javascript"></script>
<style>
header{
  color:#ff0000;
  font-size:40px;
}
</style>
</head>
<body>
<header>Hello HTML5!</header>
</body>
</html>
```

From here we have two possible solutions; use JavaScript or use CSS to detect the fallback.

In order to show a message or display a different style, we can use CSS with the following steps:

1. Add a class called `no-js` to the `<html>` tag. This will work as a default option if JavaScript is not supported. If JavaScript is supported, Modernizr will replace `no-js` with a class called `js`, and will add classes for all features using the prefix `no-` if it is not supported. For example, if your browser supports geolocation, the `html` tag will look something like the following line of code:

```
<html class="js geolocation">
```

Otherwise, it will look like the following code:

```
<html class="js no-geolocation">
```

2. In the `<body>` tag, add two `div` tags containing messages for when geolocation is supported and for when it is not:

```
<div class="geolocation-available">
  Congrats! Your browser supports Geolocation!
</div>
<div class="no-geolocation">
  Your browser doesn't support Geolocation :(
</div>
```

3. Add CSS styles to show and hide the messages. By default, hide both messages and use the class created by the detection class in the `<html>` tag to hide or show the classes accordingly:

```
div.geolocation-available, div.no-geolocation{
  display: none;
}
.no-geolocation div.no-geolocation, .geolocation div.geolocation-
available {
  display: block;
}
```

Finally, the complete code should look like the following code:

```
<!DOCTYPE HTML>
<html class="no-js">
<head>
  <script src="js/modernizr.js" type="text/javascript"></script>
  <style>
  header{
    color:#ff0000;
    font-size:40px;
  }
</head>
```

```
    div.geolocation-available, div.no-geolocation{
        display: none;
    }
    .no-geolocation div.no-geolocation, .geolocation div.
geolocation-available {
        display: block;
    }
</style>
</head>
<body>
    <header>Hello HTML5!</header>
    <div class="geolocation-available">
        Congrats! Your browser supports Geolocation!
    </div>
    <div class="no-geolocation">
        Your browser doesn't support Geolocation :(
    </div>
</body>
</html>
```

In case we want to implement a JavaScript fallback, we would need to create a conditional statement using Modernizr. In our case, since Modernizr is a JavaScript object with methods, we can use `Modernizr.geolocation` to test whether geolocation is supported. The conditional statement should be as follows:

```
if (Modernizr.geolocation){
    alert("Congrats! Your browser supports Geolocation!");
}else{
    alert("Your browser doesn't support Geolocation :(");
}
```

The complete code should look as follows:

```
<!DOCTYPE HTML>
<html>
<head>
    <script src="js/modernizr.js" type="text/javascript"></script>
    <style>
    header{
        color:#ff0000;
        font-size:40px;
    }
    </style>
</head>
<body>
```

```
<header>Hello HTML5!</header>
<script type="text/javascript">
  if (Modernizr.geolocation){
    alert("Congrats! Your browser supports Geolocation!");
  }
else{
  alert("Your browser doesn't support Geolocation :(");
}
</script>
</body>
</html>
```

Modernizr—in spite of its name—does not actually add missing functionalities to browsers save for the HTML5 tags styling support. Where you need to create fallbacks for functionalities in old browsers, Modernizr is a good choice. However, if simple styling with HTML5 and CSS3 is needed, HTML5 Shiv should suffice.

Explorer Canvas

Internet Explorer in versions prior to Version 9 does not support HTML5 Canvas, which allows 2D command-based drawing, but **Explorer Canvas** enables this feature.

To use Explorer Canvas you can perform the following steps:

1. Go to <http://code.google.com/p/explorercanvas/downloads/list>.
2. Download the last version of Explorer Canvas.
3. Copy `excanvas.compiled.js` in your JavaScript folder.
4. Import the library in `<head>`, verifying the version of Internet Explorer:

```
<!--[if lt IE 9]>
  <script type="text/javascript" src="js/excanvas.compiled.
js"></script>
<![endif]-->
```

5. Now you can use the HTML5 Canvas API in older versions of Internet Explorer.

As it is a JavaScript library, meaning that it needs to be interpreted and executed at page load, its performance will be considerably lower than modern browsers. It also does not support several features and is quite buggy.

HTML5 Boilerplate

A very simple and straightforward way of setting up a project including incorporation of Modernizr and basic configuration is to use a starter kit such as **HTML5 Boilerplate**. HTML5 Boilerplate is a collection of HTML, CSS, and JavaScript files including Modernizr, jQuery, and CSS Reset2 (a set of CSS rules that override default and inconsistent renderings in different browsers in a way that creates a common baseline). Even when compatibility is the furthest topic of interest, this template can be used as an out-of-the-box way of initializing the CSS and putting the necessary JavaScript libraries in place to make compatibility a non-issue.

You can download HTML5 Boilerplate from <http://html5boilerplate.com/> by selecting one of the following options:

- **Boilerplate:** The file collection not minimized and commented
- **Boilerplate Stripped:** The file collection minimized and not commented
- **Customize Boilerplate:** You can configure which files will go in your base project

While the last case may be enough, sometimes it is necessary to include more customizations. Fortunately, there is a tool called **Initializr**, which removes unneeded files from our HTML5 Boilerplate project. Additionally, you can use the templates provided by Initializr to modify the visual presentation according to the window size / screen resolution.

To download Initializr, go to <http://www.initializr.com/> and select:

- **Classic H5BP:** This is the basic project
- **Responsive:** This is the responsive project based on screen resolution
- **Bootstrap:** This is the responsive project using Twitter's Bootstrap template (<http://twitter.github.com/bootstrap/>) using Less (<http://verekia.com/less-css/>), a dynamic stylesheet language that generates CSS on compilation using a JavaScript compiler

After this you can choose or modify the files included.

Before starting app development

As we build the MovieNow app in the following chapters, we are going to start from scratch, so we can see the process step by step. But remember that you could use templates to build your enterprise applications. The only caveat is that you always need to know what is inside the project; sometimes an unknown JavaScript or CSS file may cause serious performance issues.

While all this can sound like a nightmare, you only need to follow a simple strategy to embark on the magical quest for compatibility:

1. Follow the W3C standards for HTML, CSS, and JavaScript (<http://www.w3schools.com>).
2. Choose JavaScript libraries or CSS to give backward compatibility for old browsers. In general, a compatibility solution that does *not* include JavaScript is better than the one that does, but sometimes it is not possible with only CSS.
3. Define a course of action for allowing accessibility for any user agent. There are a couple of strategies of note: **graceful degradation** or **progressive enhancement**. Graceful degradation implies that you begin developing for modern browsers, and then you add handlers for less capable devices. Progressive enhancement, on the other hand, implies starting with basic capabilities and building to the lowest common denominator for browser features, and then adding enhancements for more capable browsers.
4. It is a good practice to support different **user experiences (UX)** for mobile devices for a series of reasons: the keyboard can be cumbersome in mobile devices such as mobile phones and even more so on touchscreen devices; viewing the same layout on smaller resolutions can force users to constantly zoom in and out or make it difficult to click some buttons or links, and sometimes it is not technically possible to have certain functionalities. For example, autoplay for video or volume control using JavaScript in iOS Devices (iPhone, iPad, and so on) is not possible.
5. Make a test plan for multiple browsers. There are services that allow you to test your enterprise application across multiple browsers and platforms. Of course, the services that employ snapshots may be sub-optimal since they do not test JavaScript execution. It is always good to have all of the browsers that your system will support installed on a test machine, and there are tools that allow you to change the browser version on the fly.
6. Use the official documentation for web browsers as well as the community forums to keep abreast of what browser makers are doing.

Summary

In this chapter, we learned about the differences between the web browsers and the inconsistent ways in which they behave. We talked about the importance of compatibility and strategies you can use to level the playing field. In general, web developers must try to cover most of the cases to ensure compatibility, but at the same time it is important to understand the project and the target audience, and adapt our solutions to them first and then to the global scene.

As a final thought, we need to put ourselves in the user's shoes. The last thing a user wants to see is a message asking to download another browser to use an application. Remember that our goal as developers is not only to bring a collection of requirements to life, but also to create engaging user experiences that define the application as a medium that facilitates an end, rather than an obstacle that separates the user from a final goal.

Before you start building a house, you need to understand what tools you need and how to use them. In the next chapter, we will look at setting up your machine and the available tools that we can use to build our HTML5 enterprise application, including a comprehensive evaluation of the business decisions involved in selecting these tools.

2

HTML5 Starter Kit: Useful Tools

Building HTML5 enterprise applications will require the right tools to do the job. Fortunately, there are a plethora of tools available to support all facets of web application development. This chapter introduces a gamut of tools useful for web development, including editors, **Integrated Development Environments (IDEs)**, web servers, web browsers, browser tools, and even HTTP proxies in the market.

We will cover the following topics:

- Choosing editors and IDEs
- Choosing web servers
- Prepackaged stacks
- Web browsers and add-ons
- HTTP proxies

Choosing editors and IDEs

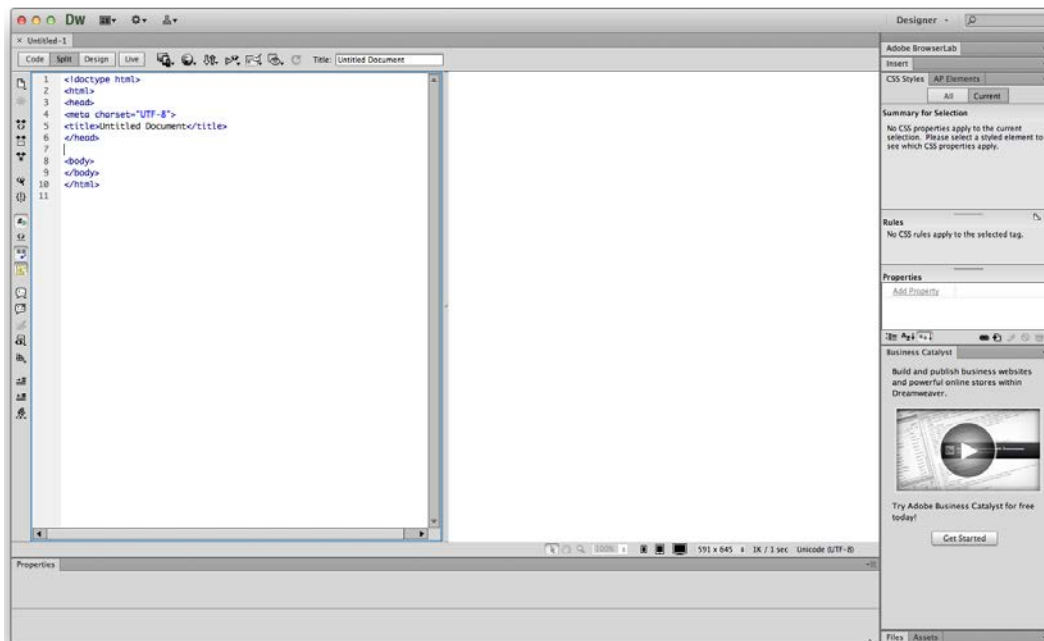
The robustness of editors and integrated development environments varies widely and they are often built to address the various needs of the end user. They can range from very simple text editors such as Notepad (Windows) and TextEdit (Mac) to sophisticated IDEs such as Eclipse. To make things even more interesting, there is a variety of new web-based editors and IDEs that allow you to develop from virtually any machine and collaborate with others on web projects with minimal setup.

Selecting one will largely be based on your needs. If your needs are simple, your tools may be simple. If your needs are complex, your tools will be complex. Although you could use almost anything to write your code, we will herein cover tools that are specifically geared towards HTML5 development.

What follows is a brief discussion of the options available to you when developing your HTML5 enterprise applications. Of course, this is not meant to be an exhaustive list, as software updates will change the landscape considerably as time goes on.

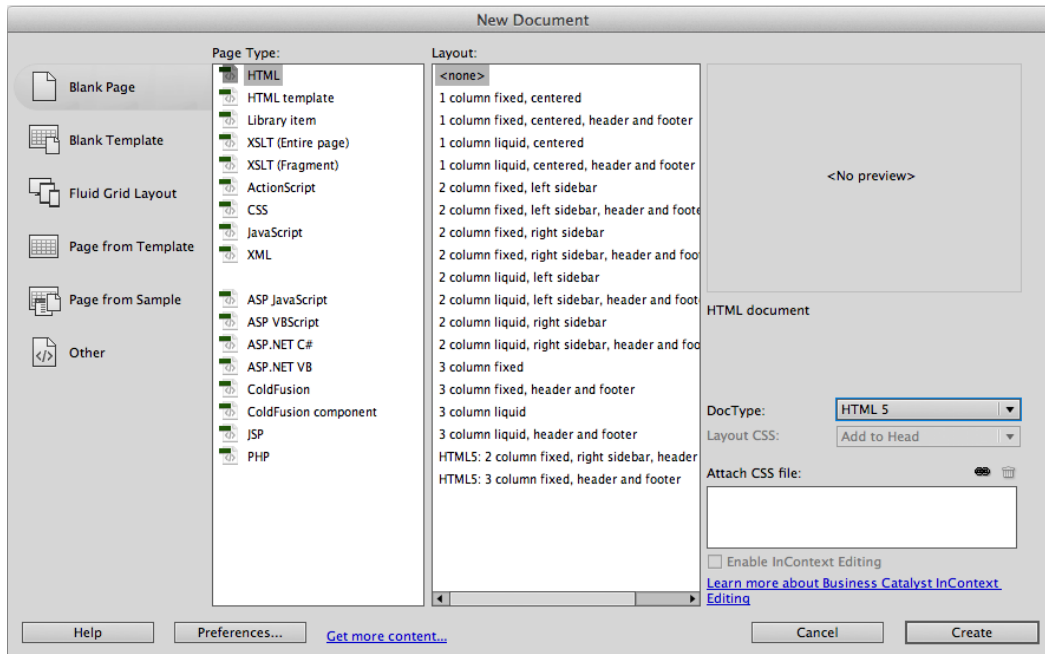
Adobe Dreamweaver CS6

This is available for Windows XP SP3 and up and Mac OS X 10.6.8 and up. Adobe's flagship product for website development underwent a major upgrade with Adobe Creative Suite 6. Introduced into the product in update 11.0.3 of CS5, Adobe baked in HTML5 and CSS3 support including HTML5 templates, updates to the WebKit engine, and coding hints to streamline HTML5 development:



With the HTML5 features in Dreamweaver, you can use the Multiscreen Preview feature to view your web page in screens of different sizes simultaneously. This includes dynamic rendering of audio and video content.

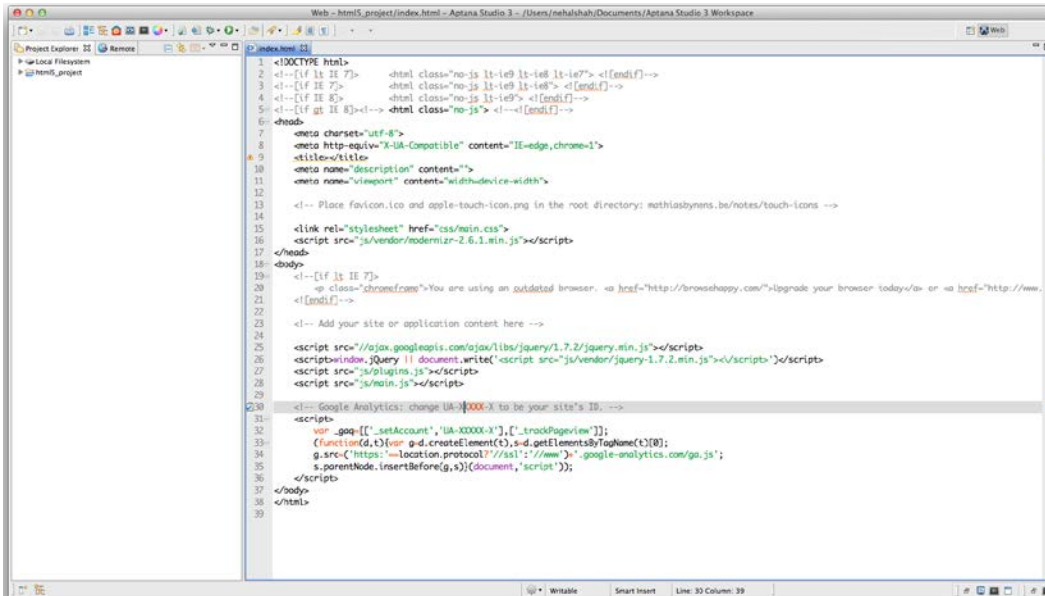
When creating a new document, be sure to select **HTML5** for the **DocType** in the **New Document** dialog as shown in the following screenshot:



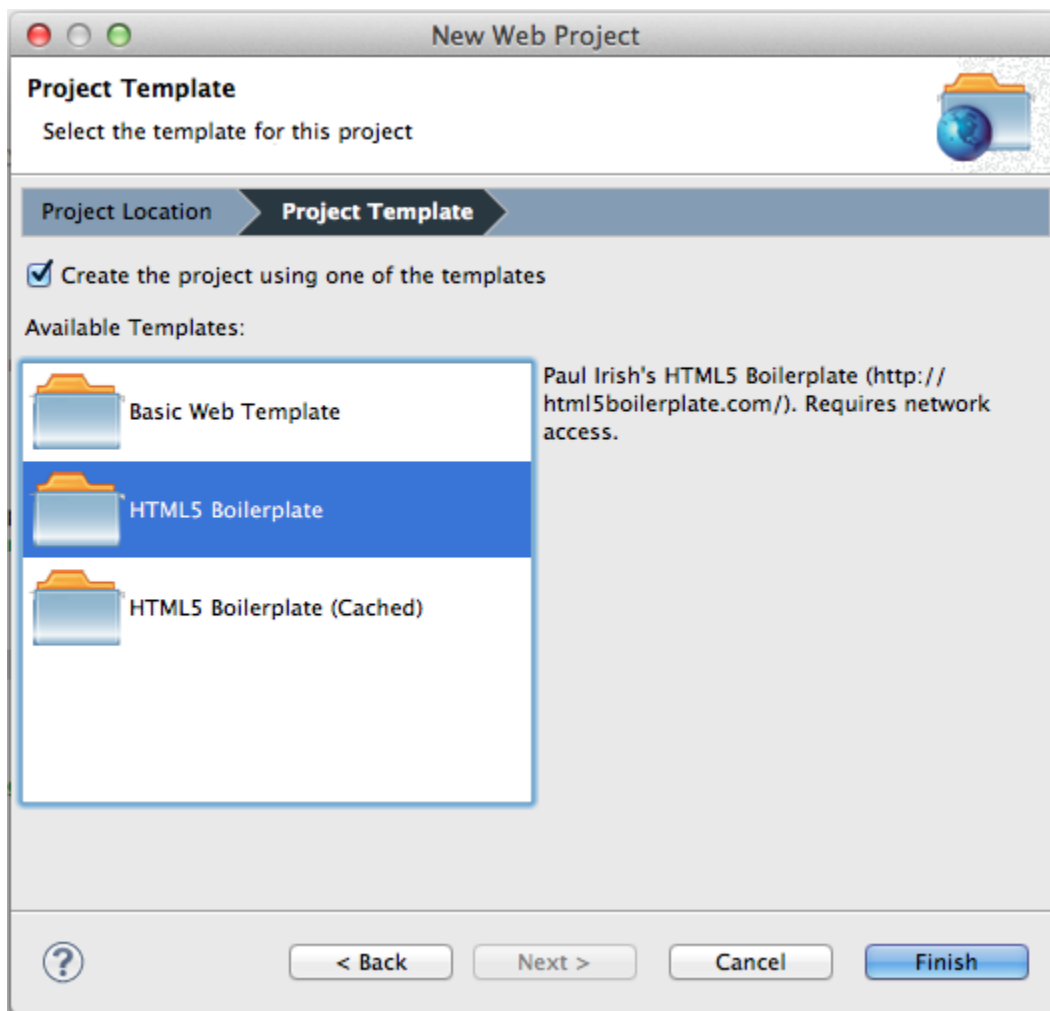
Additional information on Adobe Dreamweaver including how to purchase it can be found at <http://www.adobe.com/products/dreamweaver.html>.

Aptana Studio 3

This is available for Windows XP and up, Mac OS X 10.5 and up, Linux Ubuntu 9 and up, and Linux Fedora 12 and up. Aptana Studio 3 is a derivative of the Eclipse engine (<http://www.eclipse.org/>) and serves as a powerful, commercial-friendly open source IDE. It supports a variety of languages including Java, Ruby, and PHP as well as HTML, CSS, and JavaScript. It also boasts a number of plugins for source control management, deployment, and many other customizations.



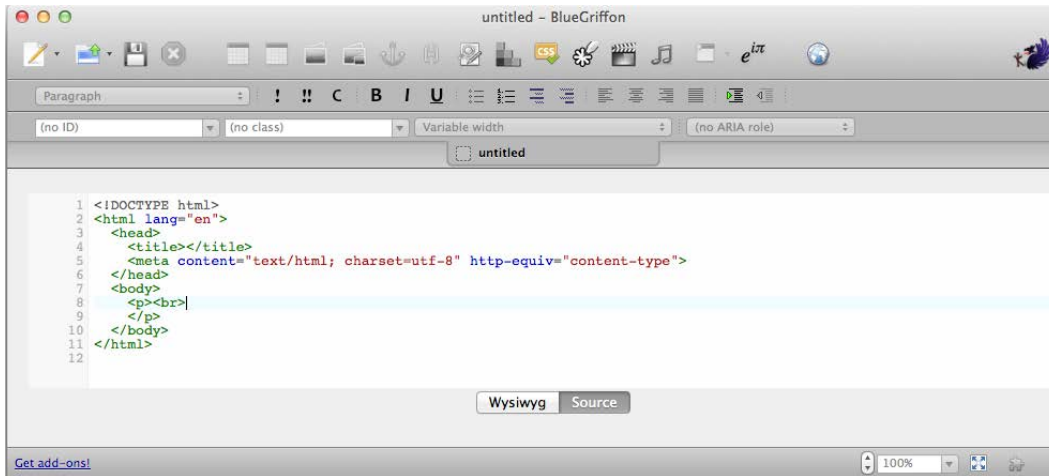
HTML5 compatibility support is a major feature of Aptana Studio 3. In its Code Assist feature, it displays which browsers support each element and the level of support. Additionally, it bakes in **HTML5 Boilerplate** as a web project template, so you can get started right away with all the tools you need to build a cross-browser compatible HTML5 enterprise application.



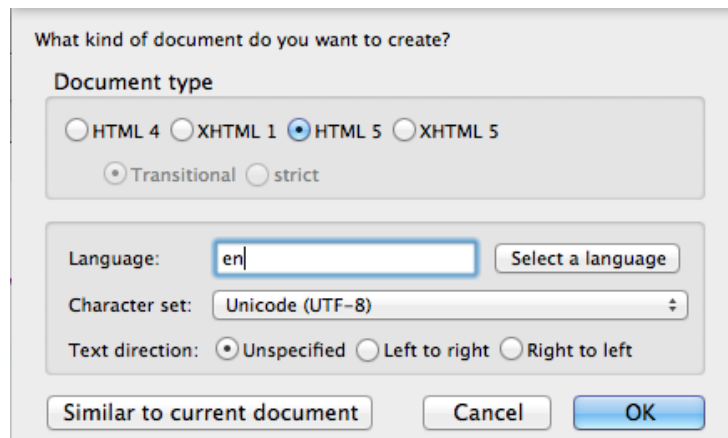
Additional information on Aptana Studio 3 including where to download it from can be found at <http://www.apтана.com/products/studio3>.

BlueGriffon 1.5.2

This is available for Windows XP and up, Mac OS X 10.5 and up, Linux Ubuntu 11.10 and up, and Linux Fedora 16 and up. BlueGriffon is a free WYSIWYG editor based on the Gecko rendering engine (the rendering engine used in Mozilla Firefox). It includes tools for developing HTML5 pages such as a DOM Explorer and support for directly embedding audio and video files. It also abstracts out many CSS3 effects with its style editor.



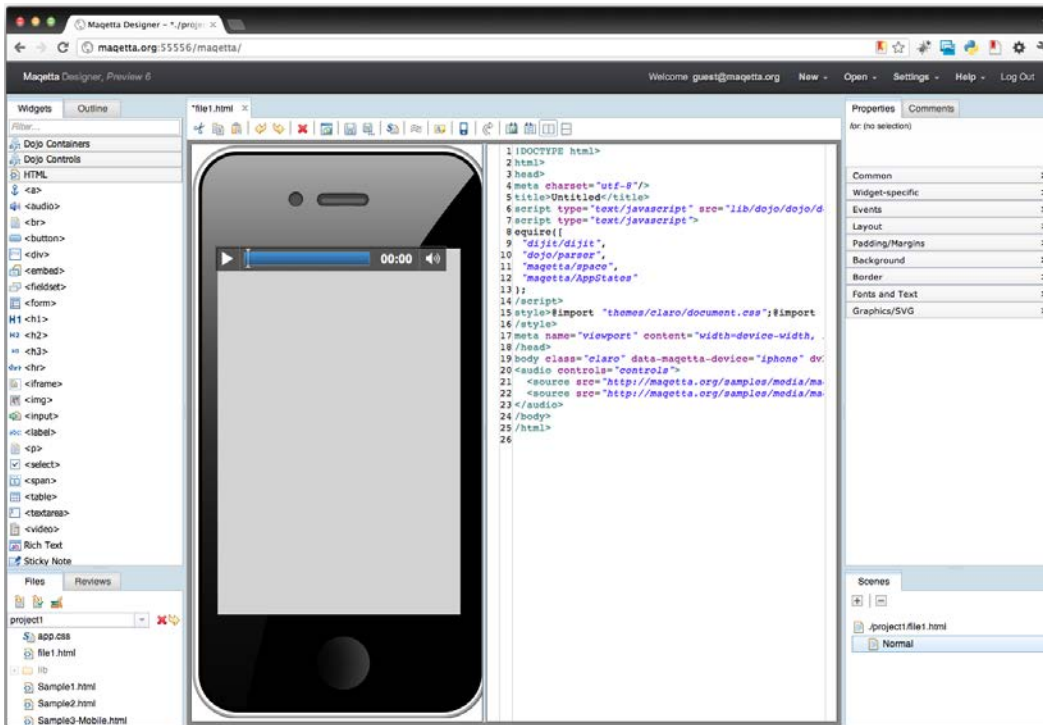
When creating a new document, simply pull down the new document toolbar (located next to the new document icon) and click on **More Options...**. Be sure **HTML 5** is selected, that the language is set, and then click on **OK**.



Additional information on BlueGriffon can be found at <http://bluegriffon.org/>.

Maqetta

Maqetta is an open source initiative of the Dojo Foundation to build an HTML5-based editor geared towards visual designers through a WYSIWYG user interface. Currently offered as a hosted product (although this may change in the future), it is in technology preview status with the hopes of releasing a 1.0 version soon. The following screenshot shows the Maqetta interface for mobile applications using an iPhone device and running on Google Chrome.

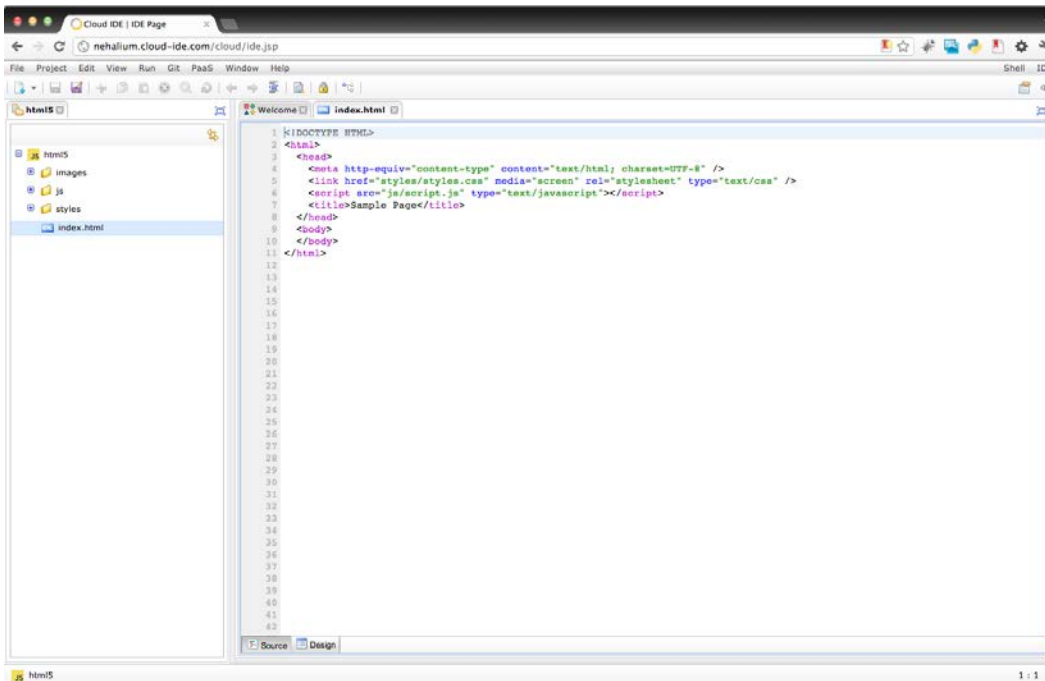


In addition to visual editing, Maqetta provides the ability to review and comment, develop wireframes, and capture interactive states in a way that communicates design intentions easily to developers.

Additional information on Maqetta can be found at <http://maqetta.org/>.

eXo

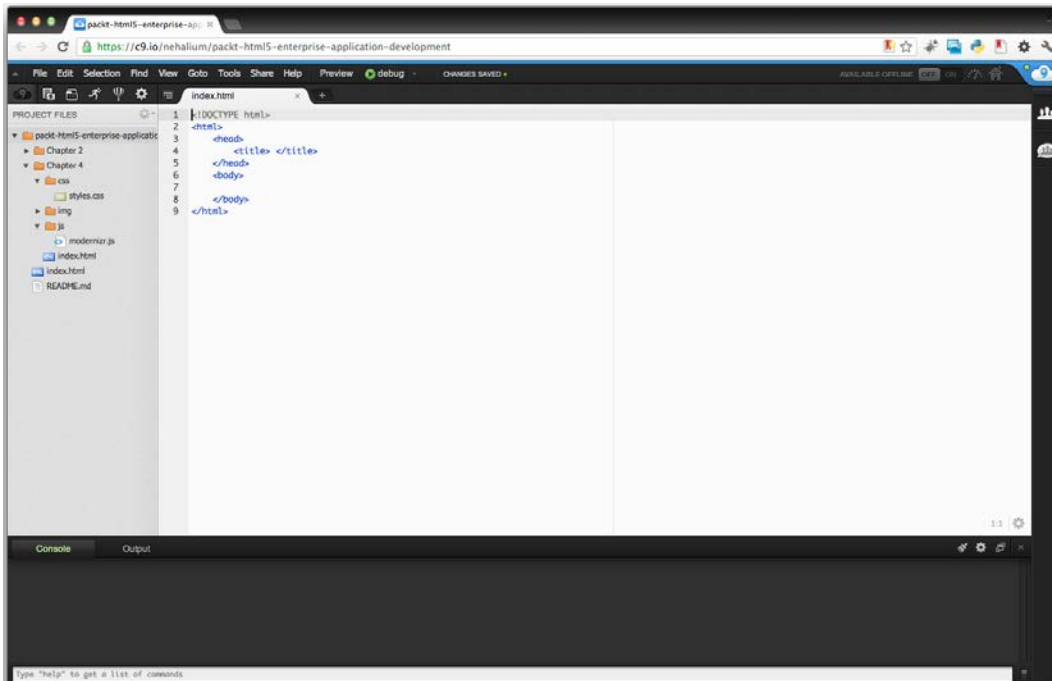
eXo provides a free, cloud-based IDE that offers collaboration and deployment features. In addition to HTML5 and JavaScript, it supports the development of Java, Ruby, and Python applications. It also integrates with **Platform as a Service (PaaS)** providers such as CloudBees and Heroku. It is based on eXo Platform 3.5, which is an enterprise Java portal and user experience platform that provides content and document management as well as social activity streams.



Additional information on eXo can be found at <http://www.cloud-ide.com/>.

Cloud9

Cloud9 is another cloud-based IDE that supports multiple languages in addition to HTML5 and JavaScript. It has gained particular interest for its integration with GitHub and Bit Bucket authentication integration and real-time collaboration. It also allows for SSH and FTP access and capabilities to work offline. Cloud9 has been positioning itself as a major IDE for Node.js development.



Although Cloud9 has a limited free subscription, it offers a premium subscription that offers additional private workspaces and full shell access.

Additional information on Cloud9 can be found at <https://c9.io/>.

Choosing web servers

If you are managing your own web cluster or are developing on your local machine, it is helpful to know the web server packages available to you. It is especially helpful to install and run a web server on your machine to get a better understanding of how your HTML5 enterprise application will run without the overhead of uploading and syncing with a remote host. The following is a brief introduction to some of the most known web servers in the market.

Apache

One of the more widely used HTTP servers is Apache. It is an open source project going back to 1996 and it installs on both Unix and Windows operating systems.

Information on installing Apache web server can be found at <http://httpd.apache.org/docs/2.4/install.html>.

Apache Tomcat

Tomcat is an open source web server that provides a servlet container allowing you to run Java code. It is available primarily for Windows and Unix, but can be installed on Mac by downloading the appropriate Unix packages.

To install Tomcat on Windows or Unix, you can follow the instructions on this website:

<http://tomcat.apache.org/tomcat-6.0-doc/setup.html>

For Mac, you can refer to this link:

<http://www.malisphoto.com/tips/tomcatonosx.html>

Jetty

Jetty is an HTTP server that is hosted by the Eclipse Foundation and is baked into the Eclipse IDE. It is known for its small footprint.

To use Jetty, you need to only use the Eclipse IDE (Aptana Studio 3, since it is derived from Eclipse, also comes with Jetty). Otherwise, you can find information on downloading and installing it at <http://wiki.eclipse.org/Jetty/>.

Tornado

Tornado is a relatively new open source web server, based on the server that powers FriendFeed (a real-time feed aggregator that consolidate updates from multiple social networks). It is particularly known for being fast and non-blocking and is, therefore, recommended for developing web services.

Information on Tornado can be found at <http://www.tornadoweb.org/>.

nginx

Pronounced "engine x", nginx started by running a number of Russian websites. Now it powers enterprises such as Netflix and Hulu. Developed in 2002 by Igor Sysoev, it relies on an asynchronous architecture that allows it to scale quickly without impacting system resources.

Information on downloading and installing nginx can be found at <http://nginx.org/en/download.html>.

LightTPD

Pronounced "lighty", LightTPD is an open source web server, designed and optimized for high performance environments. It is being used by sites such as YouTube, Wikipedia, and Meebo. It was developed by Jan Kneschke as a proof of concept to handle 10,000 connections in parallel on the same server (known as the c10k problem).

Information about download and install process can be found at <http://www.lighttpd.net/>.

Node.js

Although not technically a web server in and of itself, Node.js is a platform where you can write a simple web server. With the V8 JavaScript runtime developed for Google Chrome at its core, you can develop enterprise web applications with Node.js easily as the HTTP protocol is bundled into the platform. It is built with the same non-blocking principles used in Tornado and nginx, so it scales easier than other web servers and in some scenarios with little impact to system resources.

Additional information on Node.js can be found at <http://nodejs.org/>.

Prepackaged stacks

To make an environment setup even easier, there are a number of easy-to-install, prepackaged server stacks that have the web server, database, and scripting platform (typically PHP) built-in. You simply have to install the package and you have a complete sandbox environment ready to go. Some popular solutions include MAMP (<http://www.mamp.info/>), which is built for Mac OS, WAMP (<http://www.wampserver.com/en/>), which is built for Windows, and XAMPP (<http://www.apachefriends.org/en/xampp.html>), which has versions for Mac OS, Windows, Linux, and Solaris.

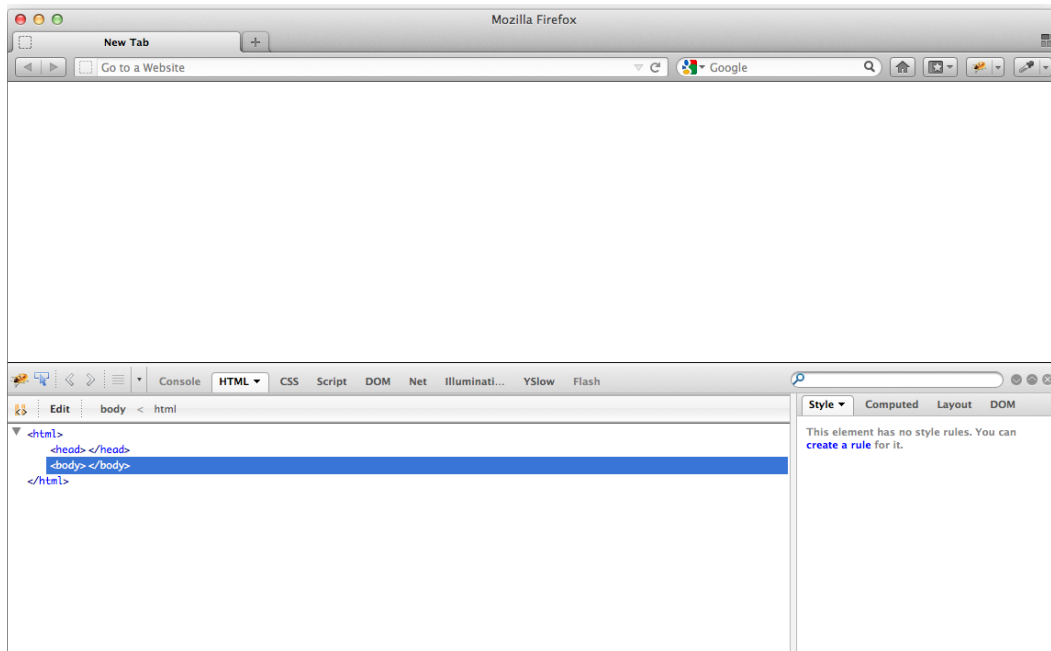
Web browsers and add-ons

Now that we have the tools to write and run our HTML5 enterprise applications, a discussion of the web browsers available to view them is in order. We will, however, take this a step further and cover some of the development add-ons available as well. With the richness of the tools either built into web browsers or pluggable, it is important to know what you have readily available to ensure that your HTML5 enterprise applications are optimal. The following is an introduction to the most common modern web browsers.

Mozilla Firefox

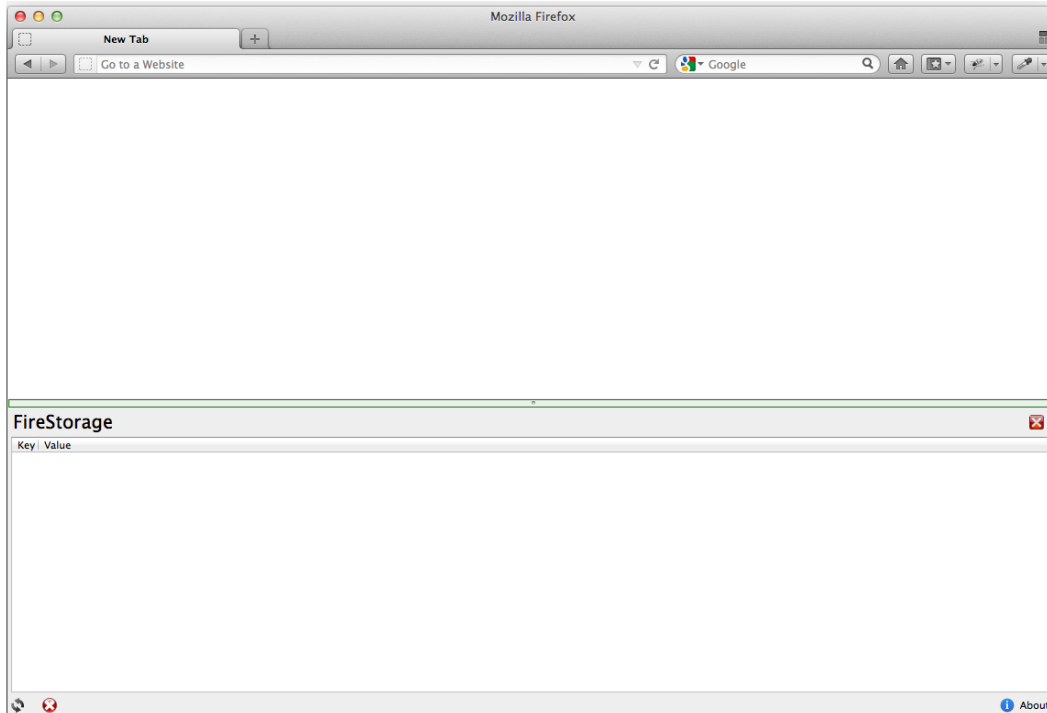
Until recent years Mozilla Firefox has been the most developer-friendly web browser by far; thanks to it many other web browsers have been including the most robust set of tools for developers. In fact, many web developers develop for Firefox first and then accommodate other browsers after. It comes with robust web developer tools built-in. Beyond that, however, you can augment it with add-ons that provide you with an even greater toolset for developing enterprise applications.

Firebug (<http://getfirebug.com/>) allows you to inspect the rendered HTML markup, tweak CSS, and see it update automatically on the page, monitor and profile network activity, and debug JavaScript. In the following screenshot, you can see the Firebug HTML code inspector:



Not only is it an extension, it is an add-on that can be itself extended. It boasts a number of extensions that add even more functionalities including FireCookie, FireUnit, FireQuery, and PageSpeed. Look here for a comprehensive list of Firebug extensions at http://getfirebug.com/wiki/index.php/Firebug_Extensions.

Another useful add-on is FireStorage and HTML5toggle. FireStorage allows you to view and manage Firefox's HTML5 local storage, while HTML5toggle allows you to toggle HTML5 support on and off in order to test fallbacks.



If you want to test new features before their final release, Mozilla provides channels to download pre-release and release versions of Firefox at <http://www.mozilla.org/en-US/firefox/channel/>.

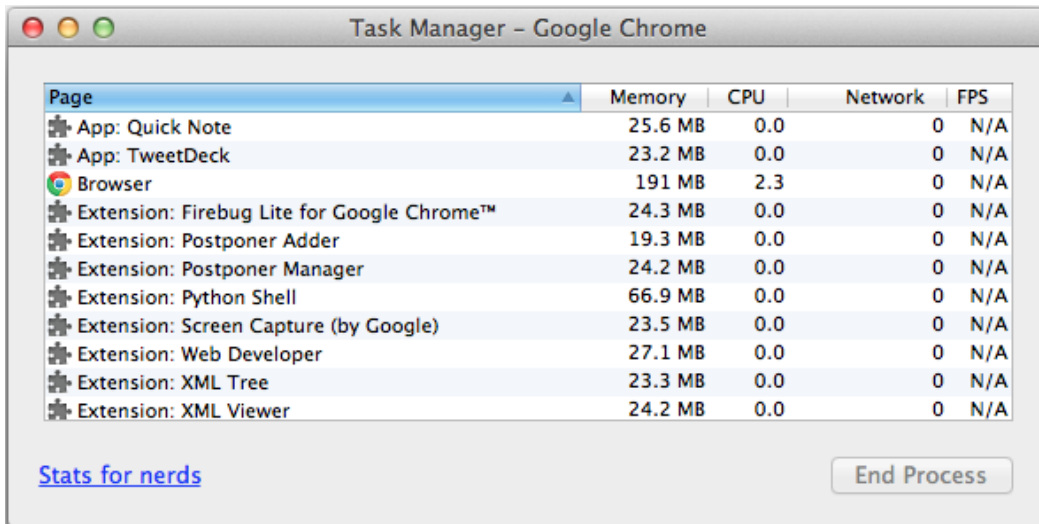


You can find three channels:

- **Firefox:** This is the final release, tried and tested
- **Firefox Beta:** This provides the latest features in a fairly stable environment
- **Firefox Aurora:** This channel is an experimental release with new features, but not so stable

Google Chrome

Google Chrome has, as of June 2012, become the most widely used web browser in the world, and it has done so with good reason. In addition to having a clean and simple user interface, it supports a library of extensions and add-ons, has a number of developer tools built-in, and it has its own task manager, which allows you to view and manage your memory and CPU usage. This is very useful for debugging enterprise applications and developing for optimal performance.

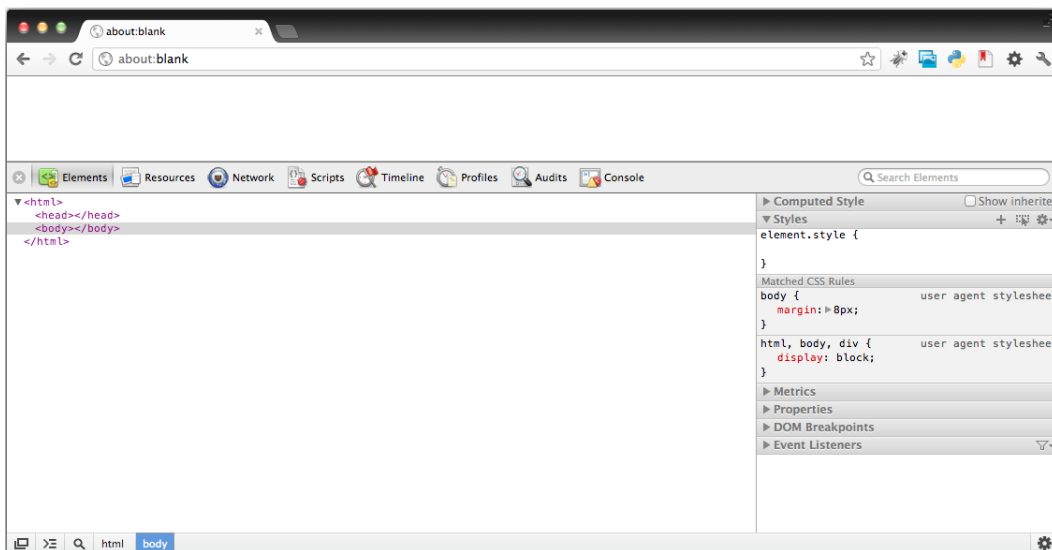


The screenshot shows the 'Task Manager - Google Chrome' window. It contains a table with the following data:

Page	Memory	CPU	Network	FPS
App: Quick Note	25.6 MB	0.0	0	N/A
App: TweetDeck	23.2 MB	0.0	0	N/A
Browser	191 MB	2.3	0	N/A
Extension: Firebug Lite for Google Chrome™	24.3 MB	0.0	0	N/A
Extension: Postponer Adder	19.3 MB	0.0	0	N/A
Extension: Postponer Manager	24.2 MB	0.0	0	N/A
Extension: Python Shell	66.9 MB	0.0	0	N/A
Extension: Screen Capture (by Google)	23.5 MB	0.0	0	N/A
Extension: Web Developer	27.1 MB	0.0	0	N/A
Extension: XML Tree	23.3 MB	0.0	0	N/A
Extension: XML Viewer	24.2 MB	0.0	0	N/A

At the bottom left of the window is a link for [Stats for nerds](#), and at the bottom right is an [End Process](#) button.

It comes with built-in developer tools. Just right-click on a web page and click on **Inspect Element**. Likewise, you can click on the **View** menu, then **Developer**, and then **Developer Tools**. This will open up a section at the bottom of the browser window, which has a number of tools at your disposal including a DOM and CSS editor, a view for HTML5 local storage, a JavaScript profiler, and even a performance auditor as shown in the following screenshot:



If you want to try new features before their final release, you can download Chrome versions through different release channels from the following link:

<http://www.chromium.org/getting-involved/dev-channel>

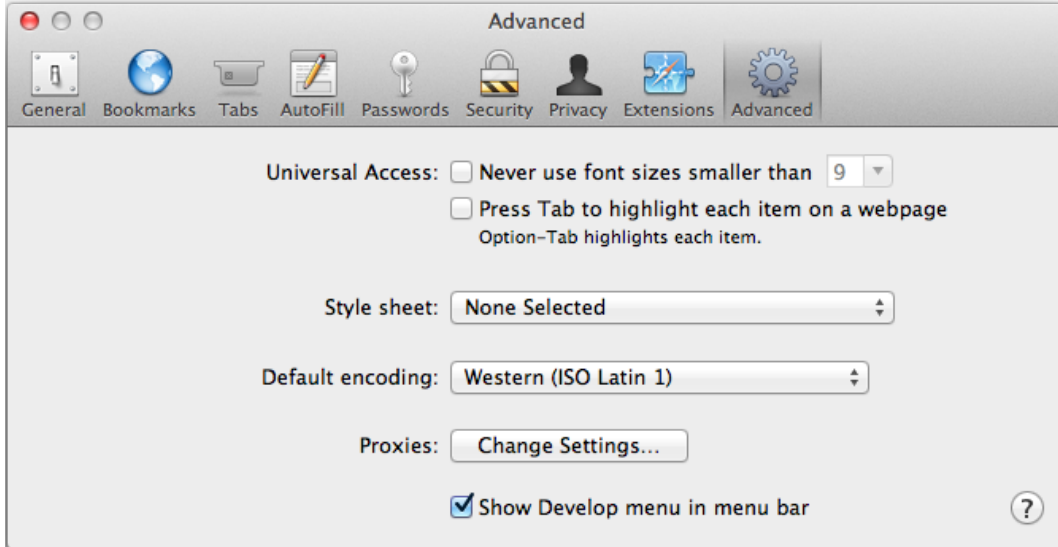
Here you will find four different channels:



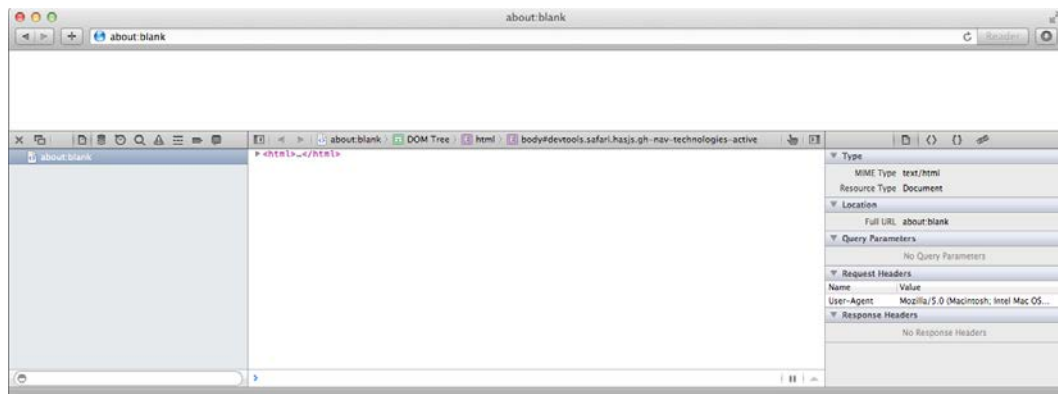
- **Stable** channel: This is the final version already tested.
- **Beta** channel: This channel is updated every week, with major updates every six weeks.
- **Dev** channel: This channel is updated once or twice weekly, it is tested but likely to have bugs.
- **Canary** build: This channel is updated daily, not previously tested or used, likely to have bugs and may not even run at all. This version can run in parallel with any other channel.

Safari

Safari by Apple Inc. is both extensible and has some developer tools built in. The developer tools, however, are hidden by default. To enable them, go to **Advanced** preferences and check **Show Develop menu in menu bar** as shown in the following screenshot:



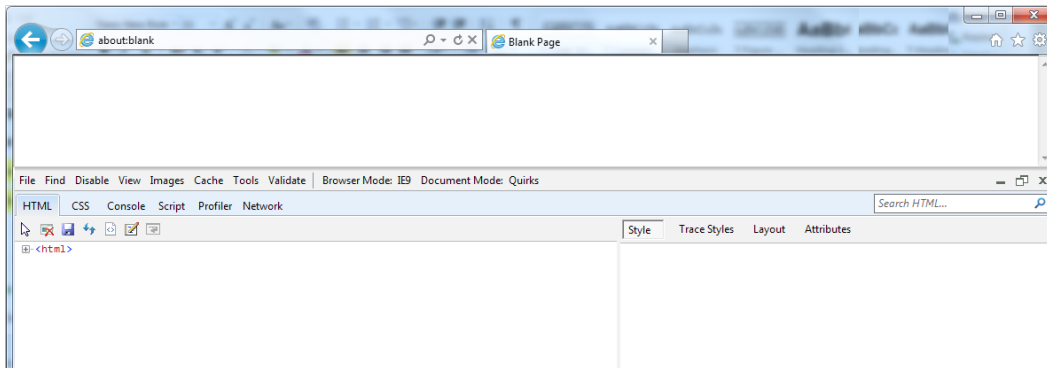
Much like in Google Chrome (which uses the same WebKit engine as Safari) and Firefox, you can right-click anywhere on a web page and click on **Inspect Element** to view the Web Inspector. You can also click on the **Develop** menu and click on **Show Web Inspector**. Safari's Web Inspector offers almost the same tools provided by Chrome's Developer Tools including a DOM editor (here called the **Snippet Editor**), a profiler, and a view of local storage.



Additionally, Safari can be augmented with extensions although the selection is not as robust as that of Firefox. Safari extensions can be found at <https://extensions.apple.com>.

Internet Explorer

Within the context of developer tools for enterprise application development, we will cover only Internet Explorer 9 here. Internet Explorer 9 comes with developer tools to inspect the DOM and make basic edits. It also includes tools to validate markup as well as a profiler and HTTP sniffer. To view the developer tools, click on **Tools** and then **F12 developer tools**. The developer tools should appear at the bottom of the window, as shown in the following screenshot:



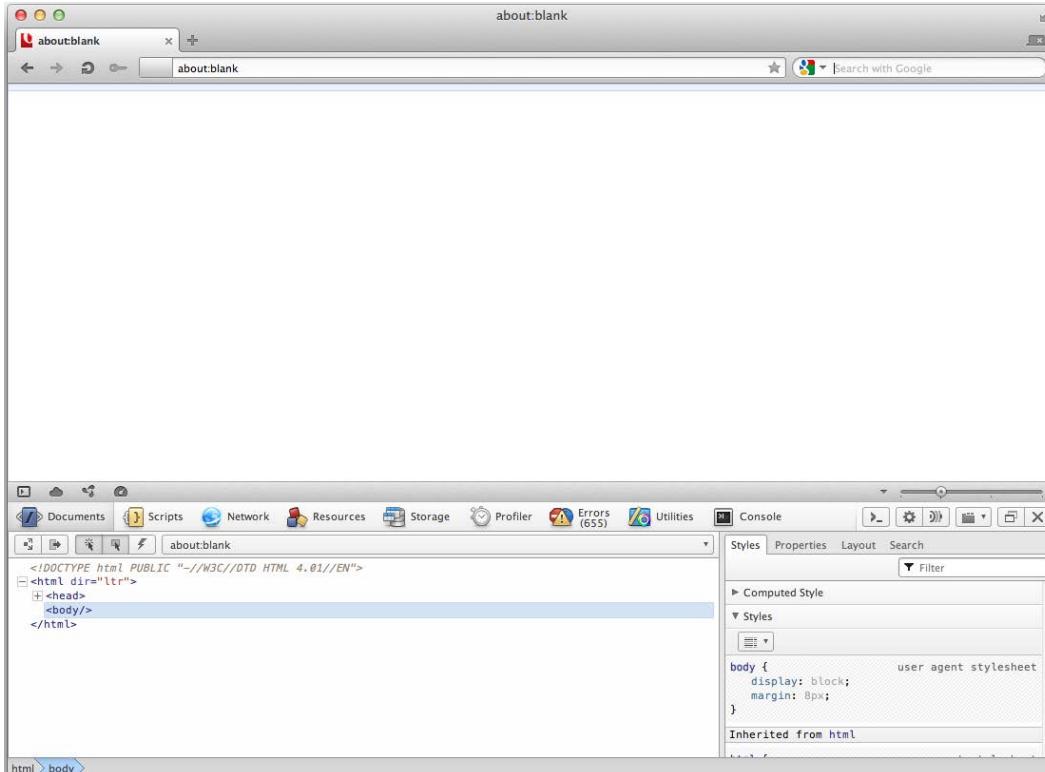
The developer tools also include two extra features: **Browser Mode** and **Document Mode**. The **Browser Mode** feature allows you to select the version of Internet Explorer, while **Document Mode** allows you to select which rendering engine mode to use when rendering a page. The difference here is that **Browser Mode** sets the default **Document Mode** and how the web browser identifies itself to the server, while **Document Mode** can be changed by a web page by setting the X-UA-Compatible meta tag; for example, to mimic the behavior of Internet Explorer 8 we can use the following tag:

```
<meta http-equiv="X-UA-Compatible" content="IE=8">
```

Opera

Beginning as a research project in 1994, Opera was first released in 1996 with Version 2.0. Since then it has grown steadily in usage over the years although it has not seen the explosive growth of Google Chrome. Still, it is used widely in countries such as Ukraine and Belarus, and is considered one of the major web browsers in the market.

Opera uses its own rendering engine, called Vega, and its own JavaScript engine, called Carakan. It comes with its own developer tools, called Opera Dragonfly, as well. Dragonfly includes a DOM and CSS inspector as well as a JavaScript console and profiler. To view Dragonfly, right-click anywhere on a web page and click on **Inspect Element**, or click on **View**, then navigate to **Developer Tools | Opera Dragonfly**; we should see a code inspector like the one shown in the following screenshot:



HTTP proxies

Web proxies can be used to capture HTTP traffic to and from your web browser to see, at a low level, just how it is to be talking to the server and what data is being returned. Although the use of HTTP proxies will be covered more in depth later when we talk about debugging, it is useful to familiarize yourself with these tools.

Charles

Charles is a widely used web proxy that has versions for Windows, Mac OS, and Linux. It includes many useful features such as bandwidth throttling, where you can simulate lower bandwidth connections to see how your application performs with limited connectivity. It can also be used as a reverse proxy, where it acts as a middleman and redirects traffic that it captures. This is useful for debugging web applications on devices such as the Apple iPad that do not natively support use of a web proxy.

More information about Charles can be found at <http://www.charlesproxy.com/>.

Fiddler

Fiddler is a web proxy that is built specifically for Windows and requires the .NET Framework Version 2.0 or later. It has several add-ons that allow you to extend its behavior including syntax highlighting and traffic differ, which compares two traffic profiles.

More information on Fiddler can be found at <http://www.fiddler2.com/>.

Summary

In this chapter we covered a variety of tools required for any enterprise application web development, including editors, IDEs, web servers, web browsers, and HTTP proxies.

While there is a wealth of tools out there to help you get started with HTML5 enterprise application development, it is important to understand their uses. Furthermore, it is important to understand your needs and to find the tools that match those needs. As they say, "It is a poor workman who blames his tools".

In the next chapter, we will dive head-first into our MovieNow application, beginning with structure and semantics. We will discuss the overall layout of the pages, the semantic tags needed, and we will cover techniques such as responsive web design that will govern how our app will look on different devices.

3

The App: Structure and Semantics

Now that we have walked through some useful tools and key ideas, we can begin with our enterprise application case study: MovieNow.

This chapter will cover the main aspects of HTML page structure, applying the correct use of HTML5 semantic tags. Also we are going to cover the use of microdata and best practices for **Search Engine Optimization (SEO)**. Finally, we are going to introduce the concept of **Responsive Web Design (RWD)** as a technique to support mobile development discussing the pros and cons therein as well as alternatives. By the end of the chapter, we will have an HTML page with basic styling but with a layout that can be easily read by any web developer with a basic understanding of HTML.

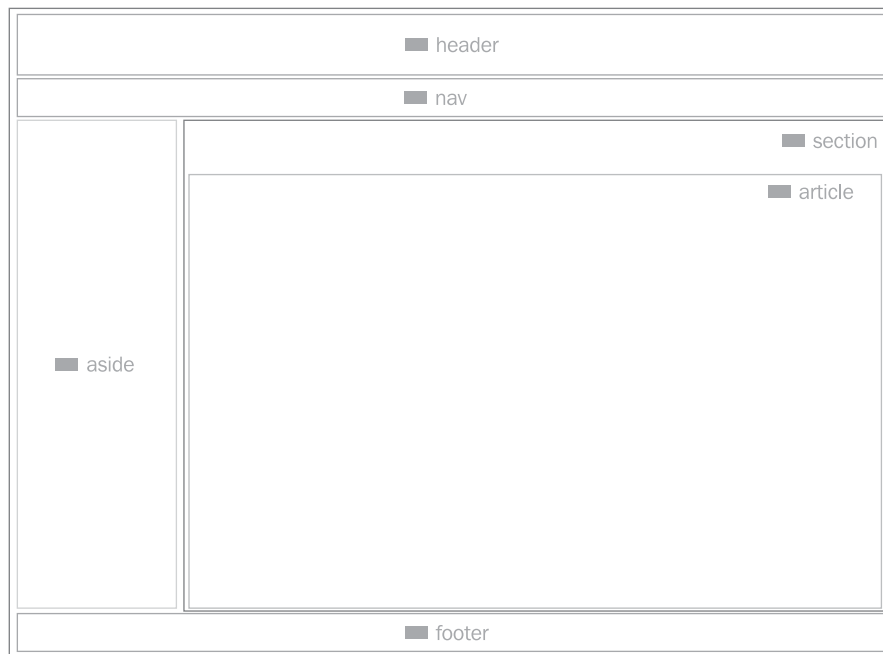
The main topics covered in this chapter are as follows:

- Understanding page structure
- Metadata
- Microdata
- Favicons and icons
- CSS3 resets
- Sticky footer
- General styling
- Responsive web design and adaptive web design

Understanding page structure

Since we already gave you an introduction to semantic tags and page elements in the *Preface*, we are now going to put that knowledge into practice and go deeper into the meaning and use of each tag, while following the natural order of construction of our HTML5 enterprise application.

A common layout for web applications is as follows:



The core structure of any HTML file includes a `DOCTYPE` declaration, an `html` root node, and `body` and `head` tags:

```
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body>
  </body>
</html>
```

From this point, we can define the layout in any way we want. Usually, but not always, a page will contain a header section specifying the company or product logo and some copy, a footer section with copyright information and some links to further information such as terms and conditions, a navigation area with links to each section, and the content area. Before HTML5, we would typically define sections using the `class` attribute or the `id` attribute of generic HTML tags such as the `div` and `span` tags. HTML5, however, streamlines this by offering predefined tags for such standard sections. We can now use `<header>` to contain the main navigation and/or initial content, `<footer>` for copyright information and alternative navigation content, `<nav>` for navigation area, and `<section>` for other content containers. This allows us to standardize our content from site to site.

The following is one way we can define the semantics of our page using HTML5 tags:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>MovieNow</title>
  </head>
  <body>
    <header>MovieNow</header>
    <nav></nav>
    <section><section>
    <footer>
      Copyright &copy; 2012 MovieNow. All rights reserved.
    </footer>
  </body>
</html>
```



We can have as many `header` and `footer` tags in an HTML page as we want, if they are in different containers. It may sound unintuitive, but it makes sense when you think of each container as a logical group of related content independent of sibling content.

As a general rule, when we have multiple `header` and `footer` tags, they should be contained in `body`, `article`, and `section` tags. Although there is no technical restriction on where semantic tags are positioned, we should maintain a structure that facilitates readability for web developers as well as search engine web crawlers.

To add the main content, we can put an `article` tag in our `section` tag. Inside of it, we can place an `h1` tag for the main heading and a `p` tag for each paragraph. The resulting HTML looks as follows:

```
<section>
  <article>
    <h1>Home Title</h1>
    <p>Home content</p>
  </article>
</section>
```

Navigation list

Unordered lists are, generally, the accepted way of representing navigation on websites since they have a cohesive semantic structure. Therefore, in our main `nav` tag, we can use an unordered list, `ul`, with embedded list elements (`li`):

```
<ul>
  <li><a href="index.html">Home</a></li>
</ul>
```

Secondary content

To finalize our main structure, we will need a sidebar that will display the top five box office movies. Since this section will be ancillary to the main content, we will use the `aside` tag. Inside the `aside` tag, we will place a heading with an `h2` tag. We use `h2` rather than `h1` because this represents the next level in the overall outline of the page content. To represent the list of movies, numerical order is important, which means that the best structure to use is an ordered list.

The result should look something like the following code:

```
<aside>
  <h2>Top 5 Box Office</h2>
  <ol>
    <li>
      <h3>Dark Knight Rises</h3>
      <p>Action</p>
    </li>
    <li>
      <h3>Avengers</h3>
      <p>Action</p>
    </li>
    <li>
```

```
        <h3>Ice Age: Continental Drift</h3>
        <p>Animation</p>
    </li>
    <li>
        <h3>The Amazing Spider-Man</h3>
        <p>Action</p>
    </li>
    <li>
        <h3>Dark Shadows</h3>
        <p>Comedy</p>
    </li>
</ol>
</aside>
```

Do not worry about the content just yet. For now, we will use sample data to demonstrate the page structure. In later chapters, we will populate this section with data from a web service.

Metadata

Until now we have been building the main structure of the MovieNow application with HTML5 semantic tags; however, there is a common misunderstanding about the effect of semantic tags in SEO. Use of semantic tags does not necessarily translate to higher search engine rankings. Nevertheless, they simplify the analysis of the content by web crawlers driving traffic to your application for specific searches related to the semantic content. In essence, they make your application more like an open book.

As a theoretical example, it will be far easier for a web crawler to determine the most important content in a specific page if this content is enclosed in an `article` tag, than if it is enclosed in a `div` tag that has no semantic meaning.

In order to provide search engine data to connect page content to the search queries that will inevitably bring people to your website or application, meta tags are a perfect solution. Meta tags store information about the web page—known as *metadata*—that is not necessarily visible to end users (unless you reveal the page source code). We can specify as many meta tags as we want. Search engine web crawlers often look to these meta tags for further information about the page content that cannot be ascertained by the display content itself.

Meta tags are contained in the head tag with the type of content defined by the property name and the content by the attribute content. The following are some of the most common meta tags:

```
<head>
  <title>MovieNow</title>
  <!-- Charset Encoding -->
  <meta charset="utf-8" />
  <!-- Description of MovieNow -->
  <meta name="description" content="Your movie theater finder" />
  <!-- Author or Authors of MovieNow -->
  <meta name="author" content="Me" />
  <!-- Keywords (recognized by certain search engines -->
  <meta name="keywords" content="movie, hollywood" />
  <!-- Copyright information -->
  <meta name="copyright" content="Copyright &copy; 2012 MovieNow. All
rights reserved." />
</head>
```

Usually search engine results will display links where the main link text comes from the title tag and the description that appears underneath the meta tag with name="description". As an example, when we search for the word "movies", we will find [fandango.com](http://www.fandango.com) listed in the search results:

[Movie Tickets & Movie Times - Fandango.com](http://www.fandango.com)

www.fandango.com/

Buy **movie** tickets in advance, find **movie** times, watch trailers, read **movie** reviews, and more at Fandango.

You visited this page.

If we inspect the code on [fandango.com](http://www.fandango.com) by viewing the source, we can see the following in the head tag:

```
<meta http-equiv="X-UA-Compatible" content="IE=Edge;chrome=1" > <meta
name="viewport" content="width=980">
<title>Movie Tickets & Movie Times - Fandango.com</title>
<meta name="robots" content="noydir,noodp" id="MetaRobot" />
<meta name="description" content="Buy movie tickets in advance,
find movie times, watch trailers, read movie reviews, and more at
Fandango." />
```

You can even define your own meta tags if you want. Search engines may overlook them as they may not be aware of them, but they may prove themselves useful for providing specific data to other developers or other applications you may want to write.



Google does not take the keywords meta tag into consideration, according to <http://googlewebmastercentral.blogspot.com/2009/09/google-does-not-use-keywords-meta-tag.html>; instead it uses description and others in conjunction with a series of specific Google search engine meta tags. SEO experts advise placing keywords in the title tag, the URL, and in the H1 tag.

Meta tags also provide additional functionality by allowing the web developer to inform the web browser about certain characteristics of the web page.

To prevent the page from being automatically translated to the client language, you can specify the following code:

```
<meta name="google" content="notranslate" />
```

To direct the behavior of Google web crawlers (known as Googlebots), specify the following code:

```
<meta name="googlebot" content="...", ..." />
```

To tell search engine web crawlers whether to inspect the content of a page or not, specify the following code:

```
<meta name="robots" content="...", ..." />
```

With the `robots` meta tag, you can include any or all of the following list separated by commas:

- **noindex**: This prevents the page from being indexed altogether
- **nofollow**: This prevents search engines from following links inside the page
- **noarchive**: This prevents search engines from showing a cached link for the page

For example, the following meta tag suggests to search engines that they should not index the page and follow links on the page for further indexing:

```
<meta name="robots" content="noindex, nofollow" />
```



Depending upon each search engine implementation, the suggestions declared on meta tags can be ignored.

If your enterprise application is mentioned on Twitter, you can add new meta tags that will be interpreted by Twitter when someone tweets (posts) a link to your page for display on Twitter streams.

To include a title, description, and a thumbnail image for your page when referenced by a tweet, you can add the following code:

```
<meta name="twitter:card" value="summary" />
```

For the Twitter account associated with the website, you can add the following code:

```
<meta name="twitter:site" value="@username" />
```

These meta tags define for Twitter certain data that it can use to express more about your web page it mentions. For more information about meta tags, see the following page:

<https://dev.twitter.com/docs/cards>

Facebook also has meta tags it defines for expressing metadata about links to web resources. For more information about Facebook's meta tags, see the following page:

<http://developers.facebook.com/docs/opengraphprotocol/>

Microdata

We have the ability to define metadata at the page level but what about metadata on specific elements on the page? **Microdata** provides us with the answer. Microdata is an HTML specification used to add more information to HTML tags.



An interesting read to understand how Google manages metadata and microdata can be found at <http://support.google.com/webmasters/bin/answer.py?hl=en&answer=99170>.

We previously defined HTML5 syntax for a list of movies. Now we can specify the meaning of each tag that defines a movie. First, we need to identify the item or container using the `itemscope` attribute:

```
<li itemscope>
  <h3>Dark Knight Rises</h3>
  <p>Action</p>
</li>
```

Now we can specify the type of content using the `itemprop` attribute and the word that defines the type of content, in this case `name` and `genre`:

```
<li itemscope>
  <h3 itemprop="name">Dark Knight Rises</h3>
  <p itemprop="genre">Action</p>
</li>
```

Here we need to rely on the importance of standards; while you can define microdata in the way you prefer, the goal is to create a unified way to define the data in a way any web crawler or reader implementation can read it.

Suppose that we decide to define MovieNow's microdata in a way that it can be easily analyzed. We would need to share a common schema with other applications. A possible solution to this is `schema.org`:

Schema.org provides a collection of schemas, i.e., html tags, that webmasters can use to markup their pages in ways recognized by major search providers. Search engines including Bing, Google, Yahoo! and Yandex rely on this markup to improve the display of search results, making it easier for people to find the right web pages.

Using this site, we need to only search for the kind of data needed. Searching for movie, we get a page with a movie schema: <http://schema.org/Movie>.

Thing > CreativeWork > Movie

A movie.


Property	Expected Type	Description
Properties from <u>Thing</u>		
additionalType	URL	An additional type for the item, typically used for adding more specific types from external vocabularies in microdata syntax. This is a relationship between something and a class that the thing is in. In RDFa syntax, it is better to use the native RDFa syntax – the 'typeof' attribute – for multiple types. Schema.org tools may have only weaker understanding of extra types, in particular those defined externally.
description	Text	A short description of the item.
image	URL	URL of an image of the item.
name	Text	The name of the item.
url	URL	URL of the item.
Properties from <u>CreativeWork</u>		
about	<u>Thing</u>	The subject matter of the content.
⋮		
genre	Text	Genre of the creative work
⋮		
trailer	<u>VideoObject</u>	The trailer of the movie or TV series, season, or episode.

Schema Draft Version 0.97

As you can see the list includes **name** and **genre** attributes, so we only need to add the schema to our container tag using the `itemtype` attribute:


```
<li itemscope itemtype="http://schema.org/Movie">
  <h3 itemprop="name">Dark Knight Rises</h3>
  <p itemprop="genre">Action</p>
</li>
```

Any system that uses this schema will recognize our items as movies as well as the corresponding names and genres.


[ Google provides an online tool to test site microdata. This is available at <http://www.google.com/webmasters/tools/richsnippets>.]

Favicons and icons

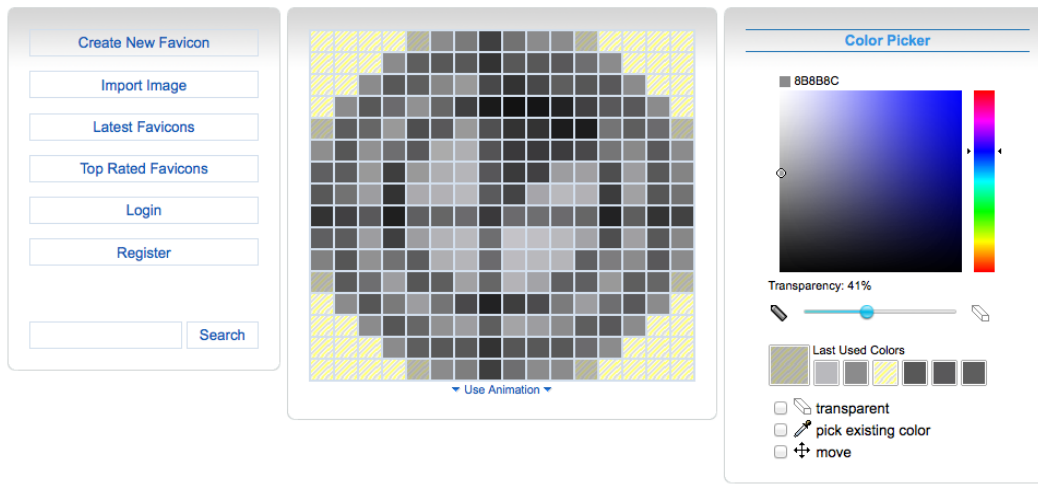
Now let us talk about some uses of the descriptive attribute `link`. As we develop our application, we will need to have icons to represent our product. Such icons can be shown not only inside our HTML, but also in browser tabs, bookmark lists, and home screen icons in the case of iOS, and some Android devices.

[ All images required are located in the `img` folder in our sample files.]

A favicon, or favorite icon, is an image used by the browser to identify a website or web application. Usually, favicons are 16 x 16 pixels and formatted as a `.png`, `.gif` (including animated GIFs), or `.ico` – the last one being the most supported file format.


[ The `ico` file format was introduced by Microsoft Windows to contain one or more images at multiple sizes and color depths, so they can be scaled appropriately depending of the application requirements. Other non-Microsoft browsers adopted this format later to maintain compatibility.]

To create a favicon, we can use any graphic editor program in the market such as Adobe Photoshop or Fireworks. Other possible solutions are the web tools such as **favicon.cc** (<http://www.favicon.cc/>). Favicon.cc allows you to upload an image and edit it using a pixel tool; this is shown in the following screenshot:



Although it is a great tool, there are downsides which include the lack of layers and undo/redo functionality.

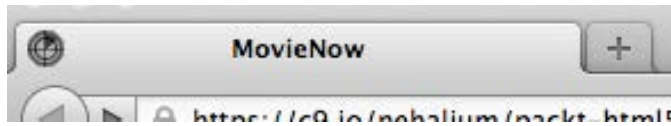
[



]
 When possible, try to export your favicon as an `ico` format, unless you want to use an animated GIF. Be aware that `ico` is a file format itself, so you need to use an image editor and export to the `ico` format. Simply renaming it with the `ico` extension will not work.

To make your application aware of your favicon, specify the name, location, and/or the format in a `link` tag inside your head tag as follows:

```
<head>
  <title>MovieNow</title>
  <!-- Charset Encoding -->
  <meta charset="utf-8" />
  <!-- Description of MovieNow -->
  <meta name="description" content="Your movie theater finder" />
  <!-- Author or Authors of MovieNow -->
  <meta name="author" content="Me" />
  <!-- Keywords (recognized by certain search engines -->
  <meta name="keywords" content="movie, hollywood" />
  <!-- Copyright information -->
  <meta name="copyright" content="Copyright &copy; 2012 MovieNow. All
rights reserved." />
  <!-- favico -->
  <link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon"
/>
</head>
```


Notice the use of the attribute `rel` to identify the relationship of the image with the web page, as well as the use of the attribute `href` to indicate image location, and `type` to specify the MIME type of the image. As a result you will see an image in the browser tab, address bar, and the favorites/bookmarks lists. In case of Firefox on Mac, you will see something like the following screenshot:



 By default, if there is no link tag with `rel="shortcut icon"`, web browsers will look for your favicon in the server root directory as a file named `favicon.ico`.

Since Version 1.1.3, iOS devices allow you to add a home screen icon as a shortcut for a mobile website or application. To add an icon for our enterprise application, we need to take into consideration the fact that there are multiple sizes depending on the device. For iPhones/iPods prior to Retina display technology, icons should be 57 x 57 px while iPhones/iPods with Retina display technology should have icons that are 114 x 114 px. For iPads prior to Retina display, icons should be 72 x 72 px, and with Retina display they should be 144 x 144 px.

Here we can see the difference between a regular display (left) and a Retina display (right):



If there is no specification about the icons in the head, iOS devices will attempt to find an icon. Otherwise, the icon will be a section of a print screen of the application.

For example, if you have an iPod without Retina display, it will try to find an icon in the root directory with a filename by going down the following list:

1. apple-touch-icon-57x57-precomposed.png.
2. apple-touch-icon-57x57.png.
3. apple-touch-icon-precomposed.png.
4. apple-touch-icon.png.
5. Generates an icon taking a print screen and using a section of it.

Retina display devices can use the same size images as non-Retina display devices, but the quality will be much poorer and, in some cases, you will notice some pixelation.

In our enterprise application, we are going to specify icons for each case using the `link` tag inside our head tag. The way to declare iOS icons in a link is to use `apple-touch-icon` in the `rel` attribute, the icon path in the `href` attribute, and finally the size in the `sizes` attribute:

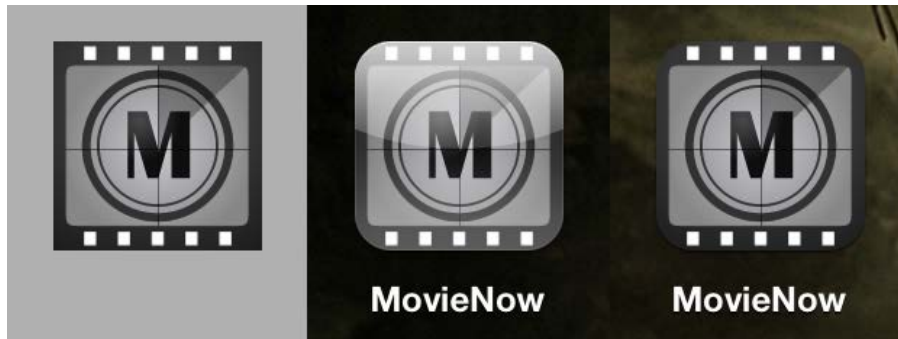
```
<link rel="apple-touch-icon" href="img/touch-icon-iphone-rd.png"
      sizes="114x114" />
```

Considering all devices, we should have something that looks like the following code:

```
<head>
  <title>MovieNow</title>
  <!-- Charset Encoding -->
  <meta charset="utf-8" />
  <!-- Description of MovieNow -->
  <meta name="description" content="Your movie theater finder" />
  <!-- Author or Authors of MovieNow -->
  <meta name="author" content="Me" />
  <!-- Keywords (recognized by certain search engines -->
  <meta name="keywords" content="movie, hollywood" />
  <!-- Copyright information -->
  <meta name="copyright" content="Copyright &copy; 2012 MovieNow. All
rights reserved." />
  <!-- favico -->
  <link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon"
/>
```

```
<!-- Apple iOS icons -->
<!-- iPhone 57x57 -->
<link rel="apple-touch-icon" href="img/touch-icon-iphone.png" />
<!-- iPad 72x72 -->
<link rel="apple-touch-icon" href="img/touch-icon-ipad.png"
sizes="72x72" />
<!-- iPhone Retina Display 114x114 -->
<link rel="apple-touch-icon" href="img/touch-icon-iphone-rd.png"
sizes="114x114" />
<!-- iPad Retina Display 144x144 -->
<link rel="apple-touch-icon" href="img/touch-icon-ipad-rd.png"
sizes="144x144" />
</head>
```

By default, iOS adds rounded corners and reflective shine effects to icons, but we can remove the reflective shine using `apple-touch-icon-precomposed` instead of `apple-touch-icon` as a `rel` value.



The previous image shows the difference between our original image, the default reflective shine effect icon, and the icon without the reflective shine effect. In our example files, we use a non-reflective version because we want to show the original image in more detail. Nevertheless, this often boils down to a mere design detail.

CSS3 resets

Now that we have our overall page structure, we are ready to start adding some styles to our enterprise application. A good practice before jumping into styling with CSS is resetting default styles to work with the same initial conditions in all browsers. If you already know how to declare CSS reset styles, you can skip this section and continue to the *Responsive web design and adaptive web design* section.

A CSS reset defines an initial set of styles to remove or standardize across browsers' default values of some properties such as margins, paddings, and so on. There are several versions of CSS resets; the most common ones are the **Yahoo User Interface (YUI) CSS Reset** (<http://developer.yahoo.com/yui/reset/>), the **HTML5 Doctor Reset** (<http://html5doctor.com/html-5-reset-stylesheet/>), **Nicolas Gallagher's normalize.css** (<http://necolas.github.com/normalize.css/>), and **Eric Mayer's Reset** (<http://meyerweb.com/eric/thoughts/2011/01/03/reset-revisited/>).

We are going to take styles from Eric Mayer's Reset and YUI's Reset to build our own. First, we will need to create a CSS file. Name it `styles.css` and save it in a folder called `css` under the root of the application.

To be recognized and applied by the HTML file, we have to import the file using a link tag with the attribute `rel="stylesheet"`, `type="text/css"`, and `href` pointing to our CSS file:

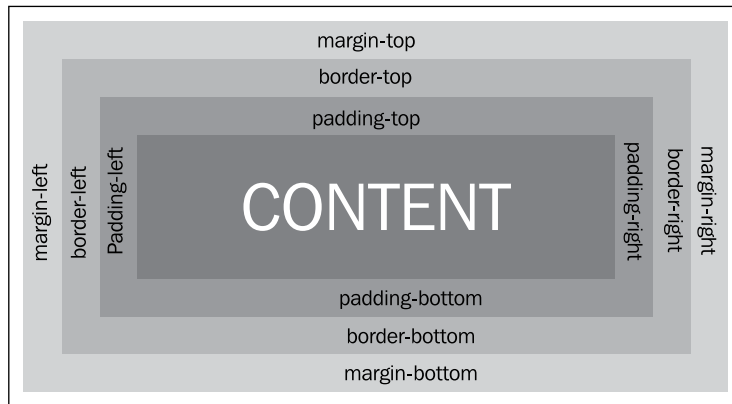
```
<link rel="stylesheet" href="css/styles.css" type="text/css" />
```

We are going to import `modernizr` too, adding HTML5 tags support for older browsers and browser capabilities detection. The head tag should look as follows:

```
<head>
  <title>MovieNow</title>
  <!-- Charset Encoding -->
  <meta charset="utf-8" />
  <!-- Description of MovieNow -->
  <meta name="description" content="Your movie theater finder" />
  <!-- Author or Authors of MovieNow -->
  <meta name="author" content="Me" />
  <!-- Keywords (recognized by certain search engines -->
  <meta name="keywords" content="movie, hollywood" />
  <!-- Copyright information -->
  <meta name="copyright" content="Copyright &copy; 2012 MovieNow. All
rights reserved." />
  <!-- favico -->
  <link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon"
/>
  <!-- Apple iOS icons -->
  <!-- iPhone 57x57 -->
  <link rel=" apple-touch-icon-precomposed" href="img/touch-icon-
iphone.png" />
  <!-- iPad 72x72 -->
  <link rel=" apple-touch-icon-precomposed" href="img/touch-icon-ipad.
png" sizes="72x72" />
```

```
<!-- iPhone Retina Display 114x114 -->
<link rel=" apple-touch-icon-precomposed" href="img/touch-icon-
iphone-rd.png" sizes="114x114" />
<!-- iPad Retina Display 144x144 -->
<link rel=" apple-touch-icon-precomposed" href="img/touch-icon-ipad-
rd.png" sizes="144x144" />
<!-- Cascade Style Sheet import -->
<link rel="stylesheet" href="css/styles.css" type="text/css" />
<script src="js/modernizr.js" type="text/javascript"></script>
</head>
```

In `styles.css`, we are going to start resetting spaces and font styles. Basic spacing involves margin and padding properties as you can see in the diagram:



For these properties (and for the `border` property), we can set styles in several ways.

Individual sides

You can set the top, right, bottom, and left margin to 0 like so:

```
margin-top:0;
margin-right:0;
margin-bottom:0;
margin-left:0;
```

As we are using 0 as a value, we do not need to specify the unit (% or px). You can apply the same to the padding and border.

Shorthand

A best practice is to declare `margin` properties in one block. This is known as **shorthand**. The shorthand syntax for `margin` and `padding` starts from the top property and is followed by the others in a clockwise manner:

```
margin:top right bottom left;
```

We can specify only two values:

```
margin:value-1 value-2;
```

This is the same as:

```
margin:value-1 value-2 value-1 value-2;
```

Or only 1 value:


```
margin:value-1;
```

This is equal to:

```
margin:value-1 value-1 value-1 value-1;
```

We need to reset `outline` and `border` too, so putting it all together we should have the following:

```
margin:0;
padding:0;
border:0;
outline:0;
```

 The shorthand for `border` can include color and style too. For example, `border:5px solid blue;`

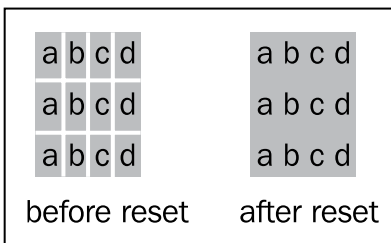
Additionally, we will need to keep the treatment of text standard across browsers. One fix to avoid exaggerated resizes of text in Internet Explorer is `font-size:100%;`. To force font inheritance from parent elements, we can use the shorthand `font:inherit`. However, to avoid problems with Internet Explorer 6 and 7, we must use the CSS properties `font-weight`, `font-style`, and `font-family`.

To set the vertical alignment using baselines of elements with their parents, we declare `vertical-align:baseline`.

So far we have the following as our reset:

```
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, figcaption, figure,
footer, header, hgroup, menu, nav, section, summary,
time, mark, audio, video{
    margin:0;
    padding:0;
    border:0;
    outline:0;
    font-size:100%;
    font-weight: inherit;
    font-style: inherit;
    font-family: inherit;
    vertical-align:baseline;
}
```

By default, tables have a separation between cells. To avoid this, we can reset the table styles using `border-collapse:collapse` and `border-spacing:0`.



The reset style should look as follows:

```
table{
    border-collapse:collapse;
    border-spacing:0;
}
```

We will need to clear the font styles and weight because in certain browsers some tags apply special styles such as bold and italic:

```
address, caption, cite, code, dfn, em, strong, th, var{
    font-style:normal;
    font-weight:normal;
}
```

To remove markers from ordered (`ol`) and unordered (`ul`) lists, we can set the `list-style` property to `none`:

```
ol, ul{
    list-style:none;
}
```

To set the main HTML5 tags as block boxes and avoid inconsistencies across browsers:

```
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section{
    display:block;
}
```

To remove quotes from tags, for long quotation (`blockquote`) and short quotation (`q`):

```
blockquote, q{
    quotes:none;
}
```

To assure that quotes really disappear across all browsers:

```
blockquote:before, blockquote:after,
q:before, q:after {
    content:'';
    content:none;
}
```

Remember that `outline` is used when the element is on focus, so we need to redefine or nullify it using `0` with the `:focus` selector. In this case, we redefine a dotted gray line of 1 pixel:

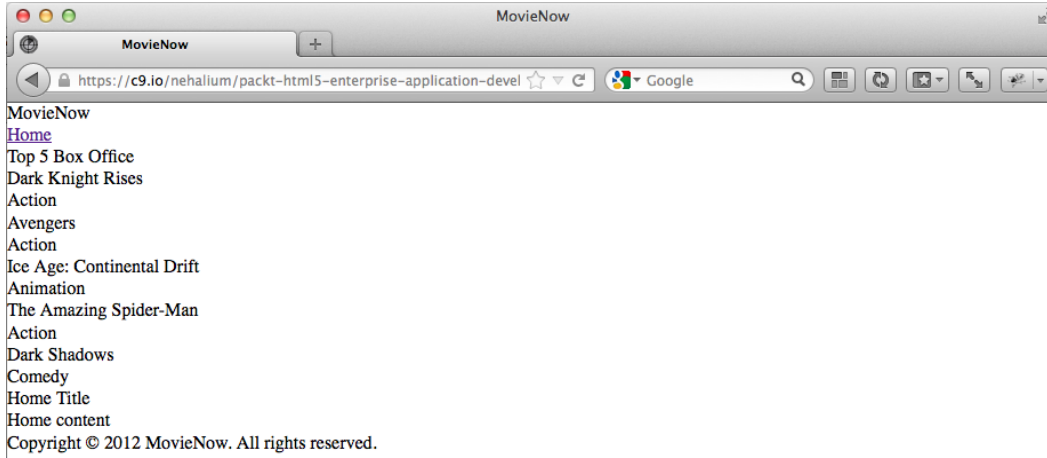
```
:focus{
    outline:1px dotted #666;
}
```

Looks like a lot of code for a simple reset, but most of it is a result of inconsistencies across browser implementations. Resets provide a level playing field on which we can build our application.

Sticky footer

CSS sticky footer layout allows you to maintain our footer at the bottom of the page even if there is not enough content to push it down. If the content exceeds the height of the page, our footer will move to the end of the scroll.

At this point our enterprise application should look like the following screenshot:



We would like our footer to stick to the bottom of the page. To achieve this, there are several implementations. We are going to follow one of the most common implementations that consists of two containers and a `.push` element that reserves the space for our `footer`:



To begin, we will add some tags to our current structure: a `section` tag with the class `wrapper` to separate all our tags from the footer, a `div` tag with the class `main` to contain page tags at the same level, and finally a `div` tag with the class `push`, to create space inside the wrapper section, allowing footer to be over the wrapper class:

```
<body class="no-js">
  <section class="wrapper">
    <div class="main">
      <header>MovieNow</header>
      <nav>
        <ul>
          <li><a href="index.html">Home</a></li>
        </ul>
      </nav>
      <aside>
        <h2>Top 5 Box Office</h2>
        <ol>
          <li itemscope itemtype="http://schema.org/Movie">
            <h3 itemprop="name">Dark Knight Rises</h3>
            <p itemprop="genre">Action</p>
          </li>
          <li itemscope itemtype="http://schema.org/Movie">
            <h3 itemprop="name">Avengers</h3>
            <p itemprop="genre">Action</p>
          </li>
          <li itemscope itemtype="http://schema.org/Movie">
            <h3 itemprop="name">Ice Age: Continental Drift</h3>
            <p itemprop="genre">Animation</p>
          </li>
          <li itemscope itemtype="http://schema.org/Movie">
            <h3 itemprop="name">The Amazing Spider-Man</h3>
            <p itemprop="genre">Action</p>
          </li>
          <li itemscope itemtype="http://schema.org/Movie">
            <h3 itemprop="name">Dark Shadows</h3>
            <p itemprop="genre">Comedy</p>
          </li>
        </ol>
      </aside>
      <section>
        <article>
          <h1>Home Title</h1>
          <p>Home content</p>
        </article>
      </section>
    </div>
  </section>
</body>
```

```
    </section>
    <div>
    <div class="push"></div>
    </section>
    <footer>Copyright &copy; 2012 MovieNow. All rights reserved.</
    footer>
  </body>
```

Now we style our structure.

We need to expand the height of `html`, `body`, and `.wrapper`.


```
html, body, .wrapper {
  height: 100%;
}
```

We then need to add `height:auto` and `min-height` for compatibility reasons and `overflow:hidden` to expand the `.wrapper` container when the content inside (that can be floated) grows:

```
body > .wrapper{
  height:auto;
  min-height:100%;
  overflow:hidden;
}
```

We can apply `overflow` to `.main` to expand too:

```
.main{
  overflow:hidden;
}
```

 The `overflow:hidden` technique to enclose content should be used with caution. Its biggest downside is that it hides the absolute positioned content that is outside of the box. An alternative is Clearfix. In our case, we will then not need `overflow:hidden` in `.main` and `body > .wrapper`:

```
body > .wrapper:after, .main:after {
  content:".";
  display:block;
  height:0;
  clear:both;
  visibility:hidden;
}
```

We can then assign the same height to `footer` and `.push`. Both will align and we can use `clear` in case we need to add a floating element:

```
footer, .push{
  height:2.0em;
  clear:both
}
```


As a final step, we assign a negative margin value with the same value as height to our footer tag and `position:relative` for compatibility reasons:

```
footer{
  position:relative;
  margin-top:-2.0em;
}
```

General styling

With our sticky footer, we can continue with some basic styling. We can set the overall font family:

```
html,*{
  font-family:Helvetica,Arial, sans-serif;
}
```


 There is a debate about the performance implications of the universal selector `*`. While some authors discourage its use, others say CSS selectors are irrelevant in terms of web performance.

To position our top five box office to the right and set its `width` property:

```
aside{
  float:right;
  width:200px;
}
```

Let us add a background color to our main navigation menu:

```
nav{
  background-color:#666;
}
```

 Notice that we are using the shorthand version of hexadecimal colors.

As a decoration, we can add a background image to `header`. By default, it will tile the image unless we specify `no-repeat` to our `background` property. We can then set `color` and `height` too:

```
header{
  color:#fff;
  height:122px;
  background:#1A1A1A url(../img/logo_back.png);
}
```

We can define the color of our `footer`, as well as color of font, font-size, line-height (to center the text vertically) and `text-align` to center text horizontally:

```
footer{
  background-color:#000;
  color:#fff;
  font-size:.6em;
  line-height:2em;
  text-align:center;
}
```

Set our wrapper background color:

```
.wrapper{
  background-color:#fff;
}
```

We can define a fixed width for our content and we use `auto` for side margins to center our container tags:

```
.wrapper, footer{
  width:960px;
  margin:0 auto;
}
```


We add a `div` tag into our `header` tag to contain the application name and logo:

```
<header><div>MovieNow</div></header>
```

We define `width`, `height`, and our logo image using the `text-indent` property to hide text content inside `div`:

```
header div{
  width:320px;
  height:122px;
  background:url(../img/logo.png);
  text-indent:-9999px;
}
```

Using `-9999px` in `text-indent` displaces our text to the left out of the visible area.

 For accessibility and SEO considerations, it is a good practice to maintain application name as text. Other techniques can be seen at <http://css-tricks.com/css-image-replacement/>.

Set the color of the columns at each side:

```
html, body{
  background-color:#ccc;
}
```

Define wrapper padding:

```
.wrapper section, nav{
  padding:5px 35px;
}
```

We then remove link underlines and assign white color to all links inside our navigation bar:

```
nav a{
  color:#fff;
  text-decoration:none;
}
```

Add an underline on hover:

```
nav a:hover{
  text-decoration:underline;
}
```

Change the font size of headings:

```
article h1{
  font-size:1.5em;
  margin:10px 0 5px;
}
```

Add color to our top five list:

```
aside{
  padding:30px 0 10px 0;
  margin:0 10px;
  background-color:#E4E4E4;
}
```



Content properties allow you to define and increment a variable:

- `counter-reset:variable;` resets variable to 1
- `counter-increment:variable;` increments 1 to variable
- `content:counter(variable);` shows the value of variable inside the tag

We can add padding and a counter variable to set the number for each movie on our top five:

```
aside ol{
  padding:0 0 0 36px;
  counter-reset:counter;
}
```

We can use our counter `content:counter(counter)`, reset it using `counter-reset:counter;`, and increment it using `counter-increment:counter.` We can then add some spacing and set the font color:

```
aside ol li:before {
  counter-increment:counter;
  content:counter(counter) ". ";
  margin-right:5px;
  color: #333;
  position:absolute;
  top:0;
  left:-16px;
}
```



Counter properties are not supported in Internet Explorer 7. Internet Explorer 8 only supports them if `!DOCTYPE` is specified.

You can assign the default management of ordered lists for IE7 and previous versions including a conditional CSS import:

```
<!-- [if lte IE 7] >
  <link rel="stylesheet" href="css/ie7.css"
  type="text/css" />
<![endif]-->
```

Where `ie7.css` contains:

```
ol{
  list-style:decimal;
}
```

Add more spacing, set the font size, and add a decorative dashed border:

```
aside h2{
  padding:0 20px 10px;
  margin:0 0 20px;
  border-bottom:1px dashed #fff;
  font-size:1.3em;
}
```

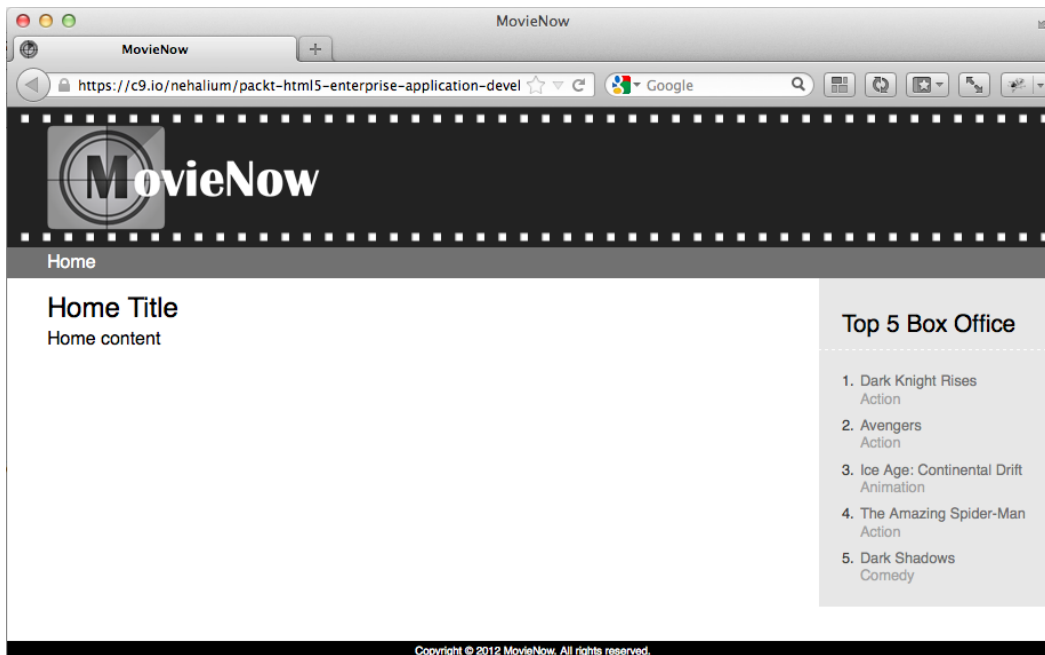
Set a different color for movie genres:

```
aside li p{
  color:#999;
}
```

Finally, add color to movie titles:

```
aside li{
  color:#666;
  font-size:.8em;
  line-height:1.2em;
  margin:0 0 8px 0;
  position:relative;
}
```

Our application is now structured and looks like the following screenshot:



Responsive web design and adaptive web design

Our application is taking form, but with so many devices and screen resolutions how can we support all of them?

Responsive web design is a fairly modern answer to this question. Responsive web design results from applying fluid grids and media queries to adapt the layout to the viewing environment. <http://mediaqueri.es/> is an illustrative guide about how to apply responsive web design in real world cases.

Using responsive web design, we can tackle many issues related to the diversity of environments.

For example, consider the following ways in which we can tackle issues that may arise:

- Controlling the size of the site when it is accessed on mobile devices
- Serving high-resolution images for retina display devices
- Changing user experience according to the device used

In responsive web design, media queries detect conditions such as screen resolution and based on that we can apply different styles.

A media query is formed specifying a media type (`screen`, `print`, and so on) and a series of features (`max-width`, `min-width`, `min-device-pixel-ratio`, and so on).

For example:

```
@media screen and (min-device-pixel-ratio: 2) and (max-device-width: 480px) {}
```



For a more detailed explanation of the syntax, you can go to <http://www.w3.org/TR/css3-mediaqueries/>.



The following are the three ways to use media queries:

Importing CSS files using media queries

It is possible to specify which CSS file to import using media queries and the `link` tag in the head tag. In the following example, we load `iphone4.css` when retina display is detected and the device screen width is less than or equal to 480 px:

```
<link rel="stylesheet" type="text/css" href="iphone4.css" media="
screen and (-webkit-min-device-pixel-ratio: 2) and (max-device-width:
480px)" />
```

Importing other CSS from our main CSS

We can import CSS files inside other CSS using `@import`. Here we can load `iphone4.css` when retina display is detected and the device screen width is less than or equal to 480 px:

```
@import url(iphone4.css) screen and (-webkit-min-device-pixel-ratio:
2) and (max-device-width: 480px);
```

Using media queries as conditionals in our main CSS

This is the most used technique and consists of media queries as conditionals inside our CSS. We are going to use this technique for our application, so let us define some media queries.

First, we define special styles for width between 738 px and 1024 px. This is applicable to many tablets in the market today. Here we are going to remove spaces and use the complete width of the device setting width for wrapper and footer to 100%:

```
/** TABLETS **/
@media only screen and (min-width: 738px) and (max-width: 1024px){
  .wrapper, footer{
    width:100%;
  }
}
```

Defining a case for devices less than 737 px width:

```
/** PHONES AND SMALL TABLETS **/
@media only screen and (max-width: 737px){
  aside{
    display:none
  }
  .wrapper, footer{
    width:100%;
  }
}
```

```
.wrapper section,nav{
  padding:5px 15px;
}
header div{
  background-position:-50px 0px
}
}
```

We can add a special case for devices with a pixel ratio superior to 2, which is the case with Apple Retina display devices. In the style, we use a high definition version of our logo:

```
/** RETINA DISPLAY IMAGES **/
@media only screen and (-webkit-min-device-pixel-ratio:2),
only screen and (min-device-pixel-ratio: 2){
  header div{
    background:url(../img/logo2x.png);
    -webkit-background-size: 320px 122px;
  }
}
```



While this technique doesn't download our logo image twice if the Retina display image is required in a Safari mobile, we need to consider that for other cases. It is better to define a media query for each case and not rely on cascade override, so we can avoid multiple loads of the same asset. You can see other techniques at the following link:


<http://timkadlec.com/2012/04/media-query-asset-downloading-results/>

We can add cases for iPhone and iPad Retina display as well by defining some additional styles:

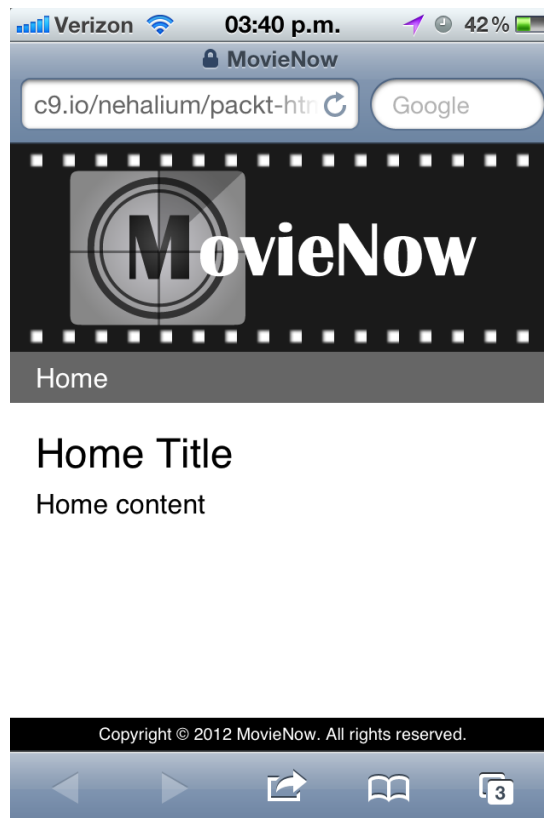
```
/** IPAD RETINA DISPLAY SPECIFIC **/
@media only screen and (-webkit-min-device-pixel-ratio:2) and
(min-device-width: 768px) and (max-device-width: 1024px),
only screen and (min-device-pixel-ratio: 2) and (min-device-width:
768px) and (max-device-width: 1024px){
  .wrapper, footer{
    width:100%;
  }
}
/** IPHONE RETINA DISPLAY SPECIFIC **/
@media only screen and (-webkit-min-device-pixel-ratio:2) and
(max-device-width: 480px),
only screen and (min-device-pixel-ratio: 2) and (max-device-width:
480px){
}
```

In iOS devices, scale starts with values superior to 100%. For that reason, we need to add a line in our head tag to set the initial scale to 100%:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

 Sadly, there is a scaling bug when you change between landscape and portrait orientations. Thanks to Scott Jehl, we can solve this problem by including a JavaScript library found at <https://github.com/scottjehl/iOS-Orientationchange-Fix>.

If we view our application in a mobile device, we can see something like the following screenshot:



We can now have a set of different views in multiple devices and resolutions:



One by-product of using media queries in this way is that if you resize your browser window you will see the different views. An alternative is to detect the device using JavaScript and import different CSS and JavaScript files depending on each case. This can be cumbersome. However, you will have to deterministically account for all the variations.

Responsive web design is a good adaptive web design approach, where the behavior of the application is dictated by the capabilities of the device.

Summary

We covered the main structure and basic styles for our application as well as metadata and microdata. We introduced the usage of icons and common CSS techniques such as sticky footers. Finally, responsive and adaptive web design concepts were covered using real-world implementations that can be applied to any enterprise application.

In the next chapter, we will introduce the use of HTML5 geolocation capabilities, AJAX calls, and API usage.

4

The App: Getting Movies Via Geolocation

HTML5 introduced a built-in ability to determine where the user is. The geolocation API defines a specification for using JavaScript to access location-based data for use in your enterprise application. Understanding where the user is can be useful for displaying news and services relevant to the user's locale.

The first major feature of our MovieNow application is the ability to find a list of movies nearest to the user based on geolocation data. We will cover how the geolocation API works as well as walk through the implementation of this feature. Since this is our first feature, we will also walk through making requests using **Asynchronous JavaScript and XML (AJAX)**.

We will cover the following topics:

- How it works
- The API
- A simple request
- Movies near you

How it works

The W3C Geolocation API specification merely defines an interface by which we can obtain data. Where and how geolocation data arrives is rather an implementation detail. On most mobile devices, GPS is usually built in and is gathered through a combination of satellite data, WiFi, and GSM/CDMA cell tower location. On desktop devices, Wi-Fi and geolocation based on IP address can be used. Lastly, Google offers a geolocation service fueled by its StreetView data. Needless to say, what goes on under the hood need not worry us, but it is good to understand how the magic really happens.

The following are the supported browsers:

- Firefox 3.5+
- Chrome 5.0+
- Safari 5.0+
- Opera 10.60+
- Internet Explorer 9.0+

Support is rendered on the following mobile devices:

- Android 2.0+
- iPhone 3.0+
- Opera Mobile 10.1+
- Blackberry OS 6

The API

The geolocation API is fairly simple providing only two methods:

`getCurrentPosition()` and `watchPosition()`. Available under the `navigator.geolocation` namespace, these methods are very similar but provide data about the device's location in distinct ways. While `getCurrentPosition` is a one-time call to get geolocation data, `watchPosition` returns geolocation data and continues to re-invoke its callback when the device's position changes until the `clearWatch` method is invoked.

Both methods take the same three arguments: a `successCallback` function, an `errorCallback` function, and a `PositionOptions` function consisting of the following attributes:

- `boolean enableHighAccuracy`: This indicates that the most accurate data should be retrieved, which may result in slower response times.

- `long timeout`: This indicates the maximum number of milliseconds before the request should time out.
- `long maximumAge`: This indicates that cached content that does not exceed the specified age in milliseconds should be returned. If set to 0, the new position data will always be returned.

Both methods also return a `Position` object to the `successCallback` function, which consists of the following properties:

- `coords.latitude`: This holds the latitude in decimal degrees
- `coords.longitude`: This holds the longitude in decimal degrees
- `coords.altitude`: This holds the height in meters relative to the reference ellipsoid
- `coords.accuracy`: This holds the accuracy of the latitude and longitude in meters
- `coords.altitudeAccuracy`: This holds the accuracy of the altitude in meters
- `coords.heading`: This holds the travel direction of the device in degrees clockwise relative to true north
- `coords.speed`: This holds the current ground speed in meters per second
- `timestamp`: This holds the date and time of when the position was acquired

Finally, the `errorCallback` argument receives a `PositionError` object when invoked, which includes the following properties:

- `code`: This indicates the error type. This can be any of the following values: `PERMISSION_DENIED` (1), `POSITION_UNAVAILABLE` (2), and `TIMEOUT` (3).
- `message`: This shows the details of the error.

A simple request

Now that we understand the mechanics of the geolocation API, let us go over dissent an actual request. Take a look at the following code snippet:

```
if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(successCallback);  
}
```


This is the most basic call we can make. First of all, since geolocation is not supported on all devices, we must take care to avoid unexpected errors by checking whether it is supported, which is where the `if` statement comes in. Secondly, we invoke the `getCurrentPosition` method passing in a `successCallback` function. `successCallback` can be any function we want to invoke when the position is returned. Notice the missing `errorCallback` function and options arguments. These are strictly optional although it is good practice to implement them to account for unexpected error conditions.

Movies near you

To begin adding geolocation to our MovieNow enterprise application, we will first make some adjustments to our page, which we set up in *Chapter 3, The App: Structure and Semantics*. In the `article` tag, we will add a `button` tag and a `div` tag:

```
<article>
  <h1>Home Title</h1>
  <p>Home content</p>
  <button id="find-movies">Find Movies</button>
  <div id="movies-near-me">
  </div>
</article>
```


The `button` tag will be used to invoke the action to get movie data while the `div` tag is where the data will land. If all goes well, your screen should display a button labeled **Find Movies**, as shown in the following screenshot:



Next, you may recall some JavaScript references that were included at the bottom of `index.html`. Let us add three more JavaScript references. See the following code snippet:

```
<footer>Copyright &copy; 2012 MovieNow. All rights reserved.
</footer>
<script src="js/ios-orientationchange-fix.js"></script>
<script src="js/jquery-1.8.0.min.js"></script>
<script src="js/jquery.xdomainajax.js"></script>
<script src="js/movienow.js"></script>
<script src="js/movienow.geolocation.js"></script>
</body>
```

You may have noticed the inclusion of `jquery.xdomainajax.js`. This is an extension to the jQuery library that allows for cross-domain AJAX GET requests. Going back to Netscape Navigator 2.0, browsers have implemented the same origin policy, which is a security precaution that restricts pages on one site from being able to access properties and methods of pages on another site. This made sense at the time, but now with an increasingly fluid World Wide Web, where content from many sites can be "mashed up" into a unified experience, the borders have by necessity been circumvented. There are many workarounds including **JavaScript Object Notation with Padding (JSONP)**, that allows cross-domain AJAX requests passing a callback parameter, so the service called can wrap the resulting JSON object in the function passed as a callback.

 The cross-domain-ajax library can be found at <https://github.com/padolsey/jquery-Plugins/tree/master/cross-domain-ajax/>. All credit goes to James Padolsey for this library.

Next, we will add the cross-domain-ajax library to the `js` folder, and then create two new files in the `js` folder: `movienow.js` and `movienow.geolocation.js`. In `movienow.js`, we will establish our root namespace `movienow`. This will be in the global or window scope meaning that it can be accessed anywhere. This is where we can add core functionality to our enterprise application as we see fit. For starters, the only line we need here is the following, which sets the root namespace:

```
var movienow = {};
```

In `movienow.geolocation.js`, we will add our geolocation-specific functionality. The reason we do this is to make sure we are following a modular approach in our enterprise application development. Modularity forces us to break up functionality into discrete, highly cohesive, loosely coupled pieces. Modularity allows us to vary parts of our enterprise application without affecting the whole. It is akin to the difference between a mobile phone with a removable battery and one where the battery is welded in. If the battery goes bad, modularity means the difference between replacing a broken part to replacing an entire device.

Self-invoking

We will begin by getting a reference to our established namespace. This is good defensive practice in case anything happens to your core namespace JavaScript file.

```
var movienow = movienow || {};
```



Notice that having this declaration is not necessary to include the `movienow.js` file with the initial definition of our namespace.

Next, we will establish our geolocation namespace:

```
movienow.geolocation = (function(){})( );
```

Notice the second set of parentheses. This construct is known as an **immediately invoked function expression (IIFE)**. This is a nifty shorthand for registering and immediately invoking JavaScript code in a modular way. All the properties and methods for geolocation will be wrapped in the `movienow.geolocation` namespace, which makes for a smaller footprint in the global namespace and cleaner, more modular code.

That becomes this

Within our newly established namespace declaration, we will do a couple of things. First, we need to capture a reference to the object itself. We will do this by adding the following line:

```
var that = this;
```

This may seem like an amusing line, but its importance will become clear. The `this` keyword in JavaScript is a handy function for referring to the owner of the executing function or to the object of which the function is a method. Without it, we would be required to prefix all of our properties and methods within our namespace with the namespace itself, which gets thorny when you want to change your namespace.

The following illustrates the value of the `this` keyword:

```
var myNamespace = {
  firstFunction: function() {
    document.write('firstFunction invoked.');
```

myNamespace.secondFunction();

```
  },
  secondFunction: function() {
    document.write('secondFunction invoked.');
```

}

```
};
myNamespace.firstFunction();
```

Notice the use of `myNamespace` to refer to other methods within the object. We can replace it with `this` in order to have a more agnostic way of referring to other members within the object:

```
var myNamespace = {
  firstFunction: function() {
    document.write('firstFunction invoked.');
```

this.secondFunction();

```
  },
  secondFunction: function() {
    document.write('secondFunction invoked.');
```

}

```
};
myNamespace.firstFunction();
```

Unfortunately, when the context changes, so does `this`. When we add a function inside another function, the context will be that of the outer function:

```
var myNamespace = {
  firstFunction: function() {
    document.write('firstFunction invoked.');
```

var innerFunction = (function() {

this.secondFunction();

})();

```
  },
  secondFunction: function() {
    document.write('secondFunction invoked.');
```

}

```
};
myNamespace.firstFunction();
```

Here we have added `innerFunction` that invokes `secondFunction` (notice the immediately invoked function expression). However, `secondFunction` is never invoked. This is because the context for `this` has changed to that of `firstFunction`. To maintain our reference to the `myNamespace` context, we simply declare a variable and hold onto it:

```
var myNamespace = {
  firstFunction: function() {
    document.write('firstFunction invoked. ');
    var that = this;
    var innerFunction = (function() {
      that.secondFunction();
    })();
  },
  secondFunction: function() {
    document.write('secondFunction invoked. ');
  }
};
myNamespace.firstFunction();
```

And this is where `that` becomes `this`.

Getting location

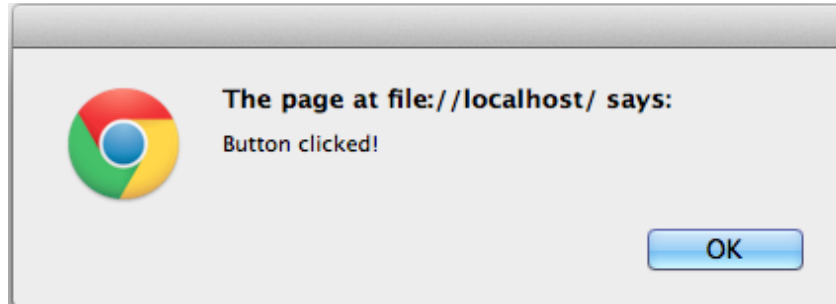
Until now the **Find Movies** button we placed on the page was non-functional. Click on it and nothing happens. We will add an event handler for that button so that something does happen when you click on it. Add the following inside the `movienow.geolocation` object:

```
jQuery(document).ready(function() {
  jQuery('#find-movies').click(function() {
    alert('Button clicked!');
  });
});
```

The `movienow.geolocation.js` file should now look like the following code:

```
var movienow = movienow || {};
movienow.geolocation = (function() {
  var that = this;
  jQuery(document).ready(function() {
    jQuery('#find-movies').click(function() { alert('Button
clicked!'); });
  });
})();
```

Now click on **Find Movie**. You should get the following alert box:



That may be all well and good, but our goals are much loftier. We want to get some location data. We do this by adding a couple of methods: `getLocation` and `locationCallback`:

```

this.getLocation = function(){
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(this.locationCallback);
  }
};
this.locationCallback = function(loc){
  jQuery('#movies-near-me').html('Lat:' + loc.coords.latitude + ',
Long: ' + loc.coords.longitude);
};

```

The first function is of course where we invoke the `getCurrentPosition` method already discussed. The second function is `successCallback`. We can now remove the alert in the event handler for the **Find Movies** button and replace it with the following:

```
that.getLocation();
```

The `movienow.geolocation.js` file should now look like the following code:

```

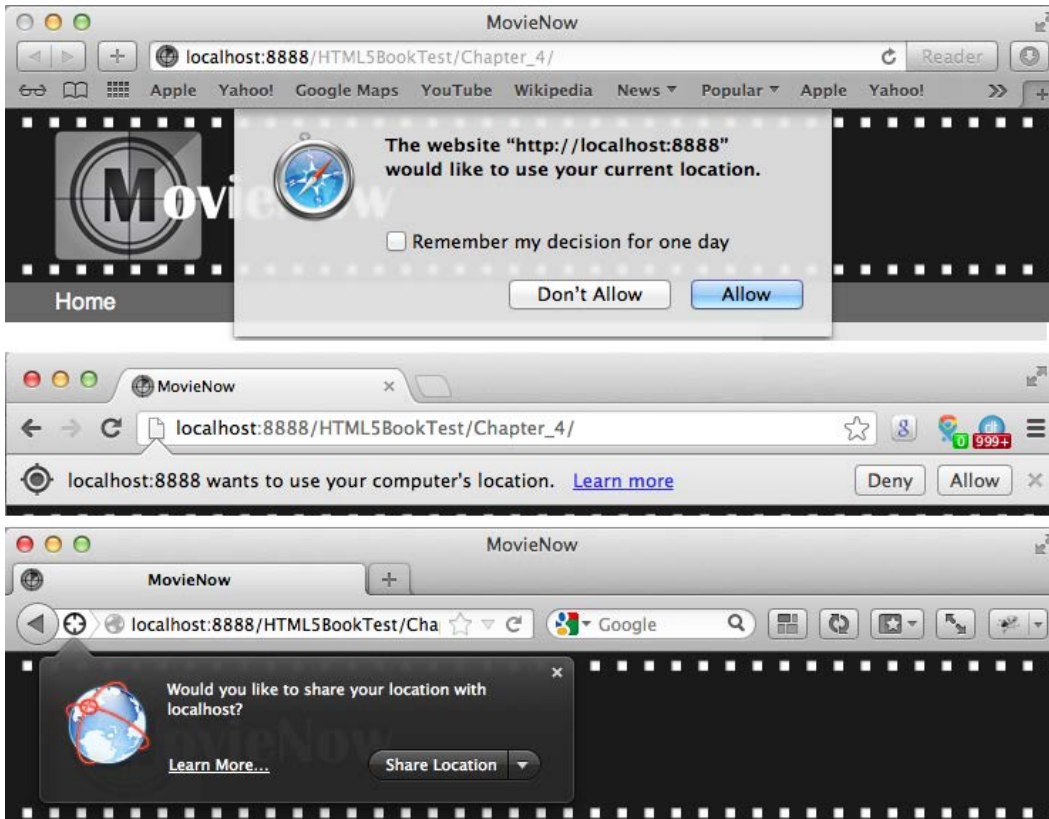
var movienow = movienow || {};
movienow.geolocation = (function(){
  var that = this;
  jQuery(document).ready(function(){
    jQuery('#find-movies').click(function(){that.getLocation();});
  });
  this.getLocation = function(){
    if (navigator.geolocation) {

```

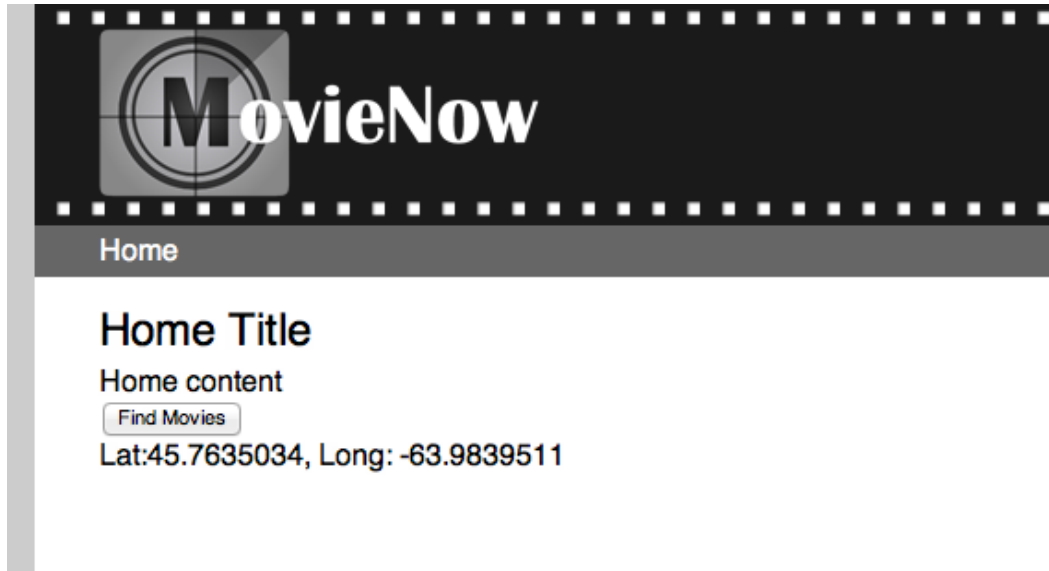
```
        navigator.geolocation.getCurrentPosition
        (this.locationCallback);
    }
};
this.locationCallback = function(loc){
    jQuery('#movies-near-me').html('Lat:' + loc.coords.latitude + ',
    Long: ' + loc.coords.longitude);
};
})();
```

Now when you click on the **Find Movies** button, a request is made through the geolocation API for location data.

The web browser will typically prompt you for permission to track your physical location. The following screenshot shows examples for Safari, Chrome, and Firefox respectively.



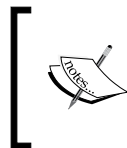
This will happen only once. When you click on **Allow**, the browser will save this setting for the specified domain.



You should now see latitude and longitude displayed on the page. Congratulations! Your enterprise application is now aware of where you are.

Getting postal codes

Now that we have geographic coordinates, the next step is to map them to postal codes. Once we have postal codes, we can get movie listings. In order to get postal codes, we will need to make an AJAX request to a web service, sending the latitude and longitude and in turn receiving postal codes. There are a number of web services that provide this data. For our MovieNow enterprise application, we will employ a service from `geonames.org`.



The GeoNames geographical database covers all countries and contains over eight million placenames that are available for download free of charge. It is licensed under Creative Commons Attribution 3.0.

Geonames.org provides a convenient web service called `findNearbyPostalCodesJSON` for obtaining postal code data. This service takes the following parameters:

- `lat`: This specifies the latitude in decimal degrees
- `lng`: This specifies the longitude in decimal degrees
- `radius`: This specifies the radius in kilometers
- `maxRows`: This specifies the maximum number of rows to return
- `style`: This specifies the verbosity of the response (`SHORT`, `MEDIUM`, `LONG`, `FULL`)
- `country`: This specifies the country to look in
- `localCountry`: This parameter, when set to `true`, returns only codes within the country
- `username`: The account for which you are accessing the data


The following is an example service call:

```
http://api.geonames.org/findNearbyPostalCodesJSON?lat=45&lng=-66.7&username=demo
```

It returns the following JSON output:

```
{
  "postalCodes": [
    {
      "distance": "10.13582",
      "adminCode1": "NB",
      "postalCode": "E5H",
      "countryCode": "CA",
      "lng": -66.769962,
      "placeName": "Pennfield",
      "lat": 45.076588,
      "adminName1": "New Brunswick"
    }
  ]
}
```

You can copy/paste this URL into a web browser and see for yourself.

 The web services are throttled meaning that only a certain number of requests per day are serviced for a given username. That is why you should register your own account with `geonames.org` before proceeding. Once you do so, swap in `demo` with your username.

Now that we have the ability to map coordinates to postal codes, we will need to make an AJAX request to make the call and retrieve the data. We will be using jQuery to assist us in making the request.

AJAX ain't just a cleaning product

Standing for Asynchronous JavaScript and XML, AJAX is a technique whereby the XMLHttpRequest object is used to make a call to the server for additional content, save state, poll for resources, and so on. It is a useful way of extending your page with additional functionality without a page refresh.

The jQuery library (<http://jquery.com>) makes it fairly easy and straightforward to make AJAX requests in a cross-browser compatible way. Take a look at the following code:

```
jQuery.ajax({
  url: 'http://some-domain.com/some-web-service',
  data: 'q=something'
  success: function(payload) {
    alert(payload);
  },
  error: function(error) {
    alert(error.responseText);
  }
});
```

You simply need to set the URL and arguments. You can define a success event handler and an error event handler. The success handler will be invoked when the AJAX request successfully completes passing the payload as an argument. The error handler will be invoked when the AJAX request returns anything other than a 200 status code.

Add the following code snippet to your `movienow.geolocation` object:

```
this.reverseGeocode = function(loc) {
  jQuery.ajax({
    url: 'http://api.geonames.org/findNearbyPostalCodesJSON',
    data: 'lat=' + loc.coords.latitude + '&lng=' + loc.coords.
longitude + '&username=demo', //Swap in with your geonames.org
username
    success: function(payload) {
      var data = that.objectifyJSON(payload);
      var postalCodes = [];
      for (var i=0; i<data.postalCodes.length; ++i) {
        postalCodes.push(data.postalCodes[i].postalCode);
      }
    }
  });
};
```

```
        jQuery('#movies-near-me').html(postalCodes.join(', '));
    }
    });
};
this.objectifyJSON = function(json) {
    if (typeof(json) == "object") {
        return json;
    }
    else {
        return jQuery.parseJSON(json);
    }
};
```



We are showing our errors using an alert popup, but for a final application we should define a CSS styled DOM to show notifications and errors.



Replace the contents of `locationCallback` with the following:

```
that.reverseGeocode(loc);
```

Upon invocation of the `successCallback` function, we are going to take the `Position` object and pass it along to our `reverseGeocode` method, which makes an AJAX request to the `geonames.org` web service to retrieve the postal codes for the location of the device. In the success handler for the AJAX request, we extract the postal codes from the JSON object and put them into an array. We then display the array on the page. Note the `objectifyJSON` method. We do this because some browsers will automatically marshal the payload data into an object while others treat it as a string.

The `movienow.geolocation.js` file should now look like the following code:

```
var movienow = movienow || {};
movienow.geolocation = (function(){
    var that = this;
    jQuery(document).ready(function(){
        jQuery('#find-movies').click(function(){that.getLocation();});
    });
    this.getLocation = function(){
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(this.
locationCallback);
        }
    };
    this.locationCallback = function(loc){
        that.reverseGeocode(loc);
    };
};
```

```

this.reverseGeocode = function(loc) {
    jQuery.ajax({
        url: 'http://api.geonames.org/findNearbyPostalCodesJSON',
        data: 'lat=' + loc.coords.latitude + '&lng=' + loc.
coords.longitude + '&username=demo', //Swap in with your geonames.org
username
        success: function(payload) {
            var data = that.objectifyJSON(payload);
            var postalCodes = [];
            for (var i=0; i<data.postalCodes.length; ++i) {
                postalCodes.push(data.postalCodes[i].postalCode);
            }
            jQuery('#movies-near-me').html(postalCodes.join(','));
        },
        error: function(error) {
            alert(error.responseText);
        }
    });
};

this.objectifyJSON = function(json) {
    if (typeof(json) == "object") {
        return json;
    }
    else {
        return jQuery.parseJSON(json);
    }
};
})();

```

When you click on **Find Movies**, you should see the following as shown in the screenshot:



From postal codes to showtimes

Now that we have postal codes, we can map those to movie showtimes. Unfortunately, there is no free web service from which we can get this kind of data. All is not lost however. `Moviefone.com` does offer feeds based on postal codes. One wrinkle however is that we cannot easily get feed data via JavaScript because of cross-domain limitations. The cross-domain Ajax library only works for services that return JSON. To work around this, we can create a proxy.

Create a file called `movielistings.php`. Add the following to your newly created file:

```
<?php
    $zips = $_GET['zip'];
    $zips = explode(',', $zips);
    $listings = array();
    for ($i=0; $i<count($zips); $i++) {
        $listings[$i] = file_get_contents('http://gateway.moviefone.
com/movies/pox/closesttheaters.xml?zip=' . $zips[$i]);
        $listings[$i] = simplexml_load_string($listings[$i]);
    }
    echo json_encode($listings);
?>
```

This is a simple PHP file that makes requests to `Moviefone.com`'s closest theaters feed based on a string of postal codes passed in the query string, and converts the output into JSON. To run this, you will need to make sure you have PHP installed on your machine. Otherwise, we could easily write something similar using JSP, ASP.NET, or Node.js for example.

Once we have our movie listings proxy service, we can add the following to `movienow.geolocation`:

```
this.getShowtimes = function(postalCodes) {
    jQuery.ajax({
        url: 'movielistings.php',
        data: 'zip=' + postalCodes.join(','),
        success: function(payload) {
            var data = that.objectifyJSON(payload);
            that.displayShowtimes(that.constructMoviesArray(data));
        },
        error: function(error) {
            alert(error.responseText);
        }
    });
};
```

```

this.constructMoviesArray = function(data) {
    var key, movie, theater = null;
    var movies = {};
    movies.items = {};
    movies.length = 0;
    for (var j=0; j<data.length; ++j) {
        if (data[j].movie) {
            theater = data[j].theater;
            for (var i=0; i<data[j].movie.length; ++i) {
                movie = data[j].movie[i];
                key = movie.movieId + '|' + theater.theaterId;
                if (!movies.items[key]) {
                    movie.theater = theater;
                    movies.items[key] = movie;
                    movies.length++;
                }
            }
        }
    }
    return movies;
};

this.displayShowtimes = function(movies) {
    var movie = null;
    var html = '';
    for (var item in movies.items) {
        movie = movies.items[item];
        html += '<p><strong>' + movie.title + '</strong><br />' +
movie.showtime.join(', ') + '</p>';
    }
    jQuery('#movies-near-me').html(html);
};

```

Once done, replace the line in the `reverseGeocode` method where we are populating `#movies-near-me` with the following line of code:

```
that.getShowtimes(postalCodes);
```

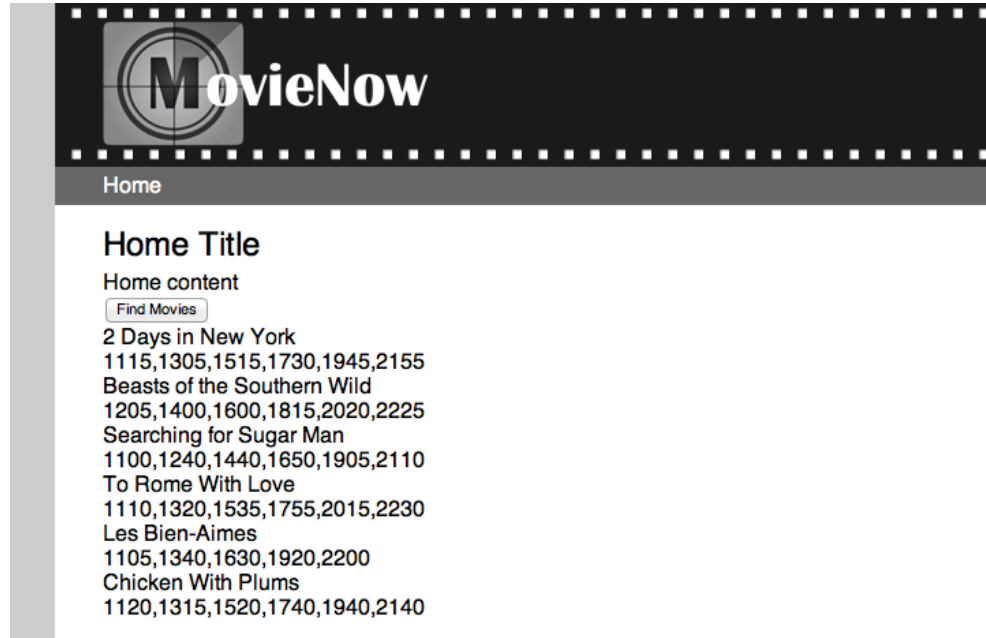
We have thus added three more methods: `getShowtimes`, `constructMoviesArray`, and `displayShowtimes`. The `getShowtimes` method makes an AJAX request to the movie listings proxy, grabs the JSON data returned and calls `constructMoviesArray` to extract the relevant data and remove duplicates, and then calls `displayShowtimes` to display the data.

The final `movienow.geolocation.js` file should now look like the following code:

```
var movienow = movienow || {};  
movienow.geolocation = (function(){  
    var that = this;  
    jQuery(document).ready(function(){  
        jQuery('#find-movies').click(function(){that.getLocation();});  
    });  
    this.getLocation = function(){  
        if (navigator.geolocation) {  
            navigator.geolocation.getCurrentPosition(this.  
locationCallback);  
        }  
    };  
    this.locationCallback = function(loc){  
        that.reverseGeocode(loc);  
    };  
    this.reverseGeocode = function(loc){  
        jQuery.ajax({  
            url: 'http://api.geonames.org/findNearbyPostalCodesJSON',  
            data: 'lat=' + loc.coords.latitude + '&lng=' + loc.coords.  
longitude + '&username=demo',  
            success: function(payload){  
                var data = that.objectifyJSON(payload);  
                var postalCodes = [];  
                for (var i=0; i<data.postalCodes.length; ++i) {  
                    postalCodes.push(data.postalCodes[i].postalCode);  
                }  
                that.getShowtimes(postalCodes);  
            },  
            error: function(error){  
                alert(error.responseText);  
            }  
        });  
    };  
    this.objectifyJSON = function(json) {  
        if (typeof(json) == "object") {  
            return json;  
        }  
        else {  
            return jQuery.parseJSON(json);  
        }  
    };  
    this.getShowtimes = function(postalCodes) {  
        jQuery.ajax({
```

```
        url: 'movielistings.php',
        data: 'zip=' + postalCodes.join(', '),
        success: function(payload) {
            var data = that.objectifyJSON(payload);
            that.displayShowtimes(that.
constructMoviesArray(data));
        }
    });
};
this.constructMoviesArray = function(data) {
    var key, movie, theater = null;
    var movies = {};
    movies.items = {};
    movies.length = 0;
    for (var j=0; j<data.length; ++j) {
        if (data[j].movie) {
            theater = data[j].theater;
            for (var i=0; i<data[j].movie.length; ++i) {
                movie = data[j].movie[i];
                key = movie.movieId + '|' + theater.theaterId;
                if (!movies.items[key]) {
                    movie.theater = theater;
                    movies.items[key] = movie;
                    movies.length++;
                }
            }
        }
    }
    return movies;
};
this.displayShowtimes = function(movies) {
    var movie = null;
    var html = '';
    for (var item in movies.items) {
        movie = movies.items[item];
        html += '<p><strong>' + movie.title + '</strong><br />' +
movie.showtime.join(', ') + '</p>';
    }
    jQuery('#movies-near-me').html(html);
};
})();
```


When you click on the **Find Movies** button, you should see the following screenshot:



Of course, we have much more data to show, but we will get to that in later chapters.

Summary

In this chapter, we walked through how the geolocation API works and how to use it. We added a button to our enterprise application and wired it to make a request to the geolocation API. We used the coordinates from the `Position` object returned to make an AJAX request to a web service to get postal codes for those coordinates. Using the postal codes, we made a request to a feed to get movie showtimes data and we displayed that data on the page.

In the next chapter, we will go over displaying the wealth of data we made available to ourselves in this chapter. We will cover CSS in more depth and talk about what's new in CSS3. We will even build some nifty CSS3 effects to make our enterprise application look interesting and inviting.

5

The App: Displaying Movie Data via CSS3

We already added some styles to our enterprise application using CSS in *Chapter 3, The App: Structure and Semantics*, but we have not introduced the properties that make CSS3 a game changer. In this chapter, we will run through some useful CSS3 properties and practical implementations for our application explaining the scope of them in any web application.

Each example will feature support (and fallback when it is needed) for the most popular web browsers in the market.

The main topics covered in this chapter are as follows:

- Back to the browsers' babel tower
- CSS Magic: Adding more styles to MovieNow (rounded corners, color, gradients, box shadows, text shadows)
- Movies and styles (transitions and animations)
- Choosing between transitions and animations
- Using media queries
- Applying CSS3 selectors

Back to the browsers' babel tower

Whenever you start using a new CSS property, it is necessary to check the list of browsers that support it. If it is supported, you need to verify how to implement it and if it requires a prefix or a special form such as `filter` in Internet Explorer.

The following are the most common prefixes for CSS properties:

- -moz- Firefox
- -webkit- Safari, Safari iOS, and Chrome
- -o- Opera
- -ms- Internet Explorer

Workarounds when you do not have support of any property include use of images and removal of some visuals (following graceful degradation and trying to avoid the removal of features).



We could use a JavaScript library such as Lea Verou's -prefix-free (<http://lea.verou.github.com/prefixfree/>) to avoid the use of multiple vendor prefixes, but this can affect our application's performance. As a general rule CSS is almost always faster (execution time) than JavaScript, so performance-wise a couple of lines more in our stylesheet is worth the effort.

As we saw in *Chapter 3, The App: Structure and Semantics*, it is possible to add conditional CSS imports. This technique only works for Internet Explorer and you can compare versions using the following syntax:

- lt (less than)
- lte (less than or equal to)
- gt (greater than)
- gte (greater than or equal to)

For example, if you want to add a specific CSS file for Internet Explorer 7 and previous versions, you can use the following declaration:

```
<!--[if lte IE 7]>
  <link rel="stylesheet" href="css/ie7.css" type="text/css" />
<![endif]-->
```

It is good practice to include Internet Explorer specific hacks and fallbacks in a separate stylesheet in order to achieve clear coding and avoid extra loading time in other browsers.



To add support for common CSS3 features such as border-radius and box-shadow, you can include CSS3 Pie (<http://css3pie.com>), a JavaScript library that adds support to these features for Internet Explorer 6 to 9.

CSS3 Magic – adding more styles to MovieNow

Let us continue with our movie application development. As a general rule, you should plan in advance or, in other words, have a visual design before beginning to mess around with styles. A benefit of following this rule (and preferably having a style guide as well) is that your application will reflect a unified visual identity. Let us start styling some elements that we already know in our enterprise application.

We removed our **Find Movies** button to make an automatic call later.

Adding rounded corners

If you had to create rounded corners with CSS1 and CSS2, you should know how complicated the possible solutions for rounded corners were. Generally, they involve images or heavy processing JavaScript affecting the performance of your enterprise application.

In CSS3, we have the `border-radius` property that allows us to specify rounded shapes for the four borders of the element.

The syntax of this property is as follows:

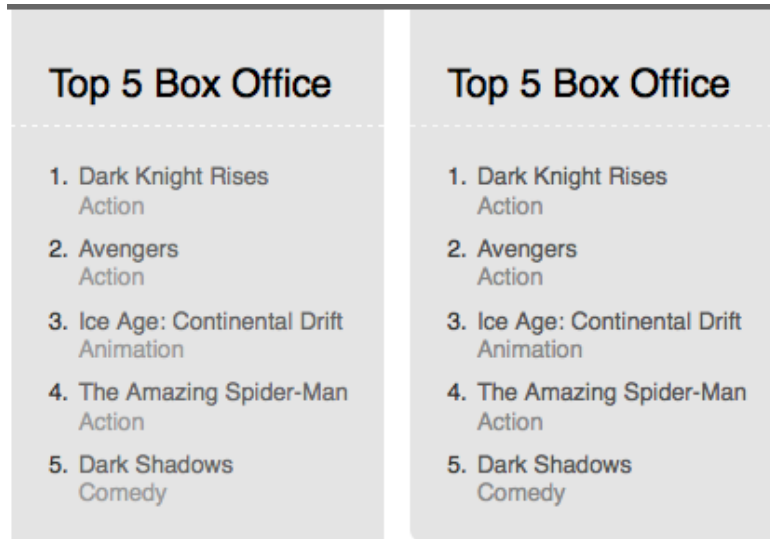
```
border-radius:top-left-radius top-right-radius bottom-right-radius
top-left-radius;
```


You can specify %, em, and px as units for each corner.

Supposing that we need to add rounded borders to the bottom of our **Top 5 Box Office** section. We can use the `border-radius` property (and its equivalents with prefixes for each browser) using 0 for top left and right, and 8px for bottom left and right.

```
aside{
  float:right;
  width:200px;
  padding:30px 0 10px 0;
  margin:0 10px;
  background-color:#E4E4E4;
  /** TOP 5 ROUNDED BORDER **/
  border-radius:0 0 8px 8px;
  -moz-border-radius:0 0 8px 8px;
  -webkit-border-radius:0 0 8px 8px;
  -o-border-radius:0 0 8px 8px;
}
```


Applying this, we can see the difference between the original Box Office (left) and the `border-radius` one (right).



[ Notice that if we use shorthand `border-radius: 0 8px`, it adds round borders for top-right and bottom-left corners only.]

This property is shorthand for `border-top-left-radius`, `border-top-right-radius`, `border-bottom-left-radius`, and `border-bottom-right-radius` properties.

Sadly, in the case of Internet Explorer, the property is supported only since IE9.

[ As an alternative, you can use CSS3 Pie (<http://css3pie.com>) or Curved Corner (<http://code.google.com/p/curved-corner/>) to give support for previous versions of Internet Explorer.]

Setting color

There are multiple ways to describe colors inside our stylesheets; the most common one is the hexadecimal `#rrggbb` where the first pair represents the numeric value of red, the second pair represents green, and the last one blue. Additionally, we can use a shorthand notation `#rgb` that will convert our value to `#rrggbb`, for example, if we use `#123` that will be recognized as `#112233`.

Let us go over other ways to describe colors:

Red, green, and blue

You can define a color using the syntax `rgb(R,G,B)` where R, G, and B indicate the intensity of the colors red, green, and blue and can be:

- An integer from 0 (no color) to 255 (max intensity)
- A float 0.0% (no color) to 100.0% (max intensity)

You must use the same units inside the declaration. It is supported in all modern browsers. Here is a class `title` with red fonts:

```
.title{
  color:rgb(255,0,0);
}
```

Red, green, blue, and alpha

An extended specification of `rgb` that adds a value at the end for alpha transparency uses values from 0.0 (invisible) to 1.0 (completely visible). It is supported in all modern browsers and Internet Explorer since Version 9. We can define a red color for fonts in our class `title` with 50 percent of alpha transparency:

```
.title{
  color:rgba(255,0,0, .5);
}
```

Hue, saturation, and lightness

HSL is a cylindrical-coordinate representation of colors. `vHue` is a floating point representation of an angle; this value defines the color on which saturation and lightness will be applied and its values range from 0 to 360. Saturation is a percentage that goes from 0 (white) to 100% (full color) and defines the colorfulness. Finally, lightness defines the amount of light and goes from 0% (no light, total black) to 100% (full color). The syntax is `hsl(H,S,L)`. It is supported in all modern browsers and Internet Explorer since Version 9. If we want to apply red fonts in our class `title`, we can do the following:

```
.title{
  color:hsl(0,100%,100%);
}
```

Hue, saturation, lightness, and alpha

This is an extended specification of `hsl` that adds a value at the end for alpha transparency in the same way `rgba` does for `rgb`. It is supported in all modern browsers and Internet Explorer since Version 9. We can define a red color for fonts in our class `title` with 50 percent of alpha transparency as follows:

```
.title{
  color:hsla(0,100%,100%, .5);
}
```

You can use a conditional CSS import for older versions of Internet Explorer and apply opacity and an alpha filter to get the same effect:

```
.title{
  color:#f00;
  opacity: 0.5;
  filter: alpha(opacity=50);
}
```

Adding gradients

New applications in the market have adopted a plain design not because of technical restrictions but for the sake of simplicity. Even though it is sometimes necessary to add some styling to simulate depth, gradients make the process much easier.

CSS3 introduces `linear-gradient` and `radial-gradient` to background values. You can apply gradients to `background` or `background-image` properties.

A possible syntax for this is as follows:

```
background-image:linear-gradient(angle, color position, color
position);
```

You can add as many `color position` pairs as you want. Although it is possible to use hexadecimal colors, in this example we are going to use `rgb`.

First, we add a gradient that goes from `top` to `bottom` to our navigation bar. It starts with light grey and ends with light grey, so we only need two points: 0% and 100%. The initial color will be `rgb(102,102,102)` and the final one will be `rgb(70,70,70)`. Adding this to `nav` with pertinent prefixes we have:

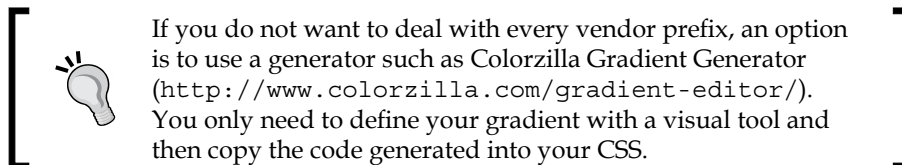
```
nav{
  background-color:#666;
  background-image:linear-gradient(top, rgb(102,102,102) 0%,
  rgb(70,70,70) 100%);
```

```

background-image:-moz-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-webkit-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-o-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-ms-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
}

```

As a result we can see the right image compared with the original one at the left:



To show that we can add multiple points, let us apply a more complex effect to our **Top 5 Box Office** area. In this case, we apply the effect from bottom to top:

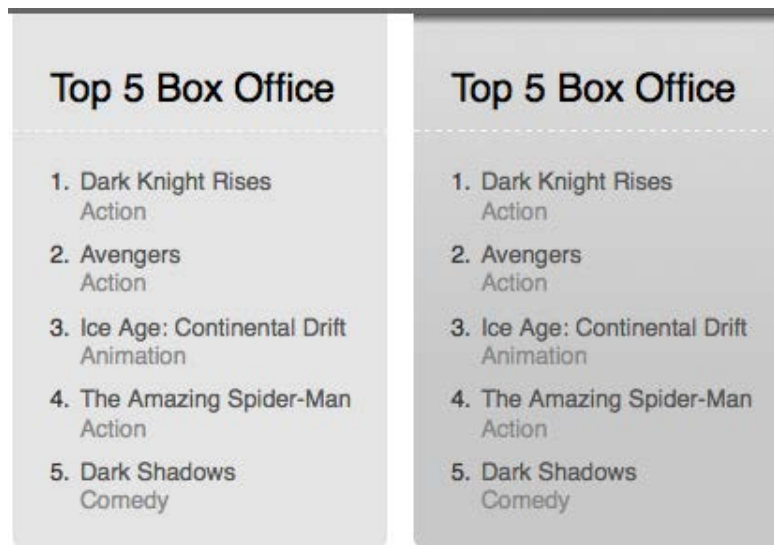
```

aside{
float:right;
width:200px;
padding:30px 0 10px 0;
margin:0 10px;
background-color:#E4E4E4;
/** TOP 5 ROUNDED BORDER **/
border-radius:0 0 8px 8px;
-moz-border-radius:0 0 8px 8px;
-webkit-border-radius:0 0 8px 8px;
-o-border-radius:0 0 8px 8px;
/** BOX OFFICE GRADIENT **/
background:linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-o-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-moz-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-webkit-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-ms-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
}

```


As in our previous example, we use percentages to define positions. In this case, we use 35%, 68%, and 98%.

Finally, we can compare the original area (left) with the final one (right), as shown in the following screenshot:




We can apply the same principles to our header:

```
header{
  color:#fff;
  height:122px;
  /** HEADER GRADIENT **/
  background-color:#1A1A1A;
  background-image:url(../img/logo_back.png), linear-gradient(right,
  rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
  background-image:url(../img/logo_back.png), -moz-linear-
  gradient(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61)
  85%);
  background-image:url(../img/logo_back.png), -webkit-linear-
  gradient(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61)
  85%);
  background-image:url(../img/logo_back.png), -o-linear-
  gradient(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61)
  85%);
  background-image:url(../img/logo_back.png), -ms-linear-
  gradient(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61)
  85%);
}
```

We get a more interesting header (bottom) compared to the original one (top):



 The prefix `-ms-` for gradients in Internet Explorer was deprecated by Microsoft. Refer to the following link:
<http://msdn.microsoft.com/en-us/library/windows/apps/hh453527.aspx>

It is always possible to fallback gradients using images in tile and the `background-image` property.

Adding box shadows

We can use shadows to simulate depth giving the effect of inset and outset visuals. The property `box-shadow` allows us to create shadows based on the borders of the element.

The syntax for `box-shadow` is as follows:

```
box-shadow:horizontal-shadow vertical-shadow blur spread color inset;
```

Only `horizontal-shadow` and `vertical-shadow` are required. `inset` specifies if the shadow is applied inside the element.

Let us add a bottom drop shadow to our nav. We can specify `0` for `horizontal-shadow`, `1px` for `vertical-shadow` to show our shadow below the element, `3px` to give some blur, and color as `#999`:

```
nav{
  background-color:#666;
  background-image:linear-gradient(top, rgb(102,102,102) 0%,
  rgb(70,70,70) 100%);
```

```
background-image:-moz-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-webkit-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-o-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
background-image:-ms-linear-gradient(top, rgb(102,102,102) 0%,
rgb(70,70,70) 100%);
/** NAVIGATION SHADOW **/
box-shadow: 0 1px 3px #999;
-moz-box-shadow: 0 1px 3px #999;
-webkit-box-shadow: 0 1px 3px #999;
-o-box-shadow: 0 1px 3px #999;
}
```

We can compare the nav menu without the shadow (left) and with the shadow (right):



To demonstrate `inset`, we can add an inner shadow to our **Top 5 Box Office** area. Here we apply negative positioning, `-1px`, for `vertical-shadow` to show a part of the shadow in the bottom, `1px` for `blur`, `1px` for `spread` (as we want to modify the size of our shadow), color `#aaa`, and finally `inset` to have an inner shadow:

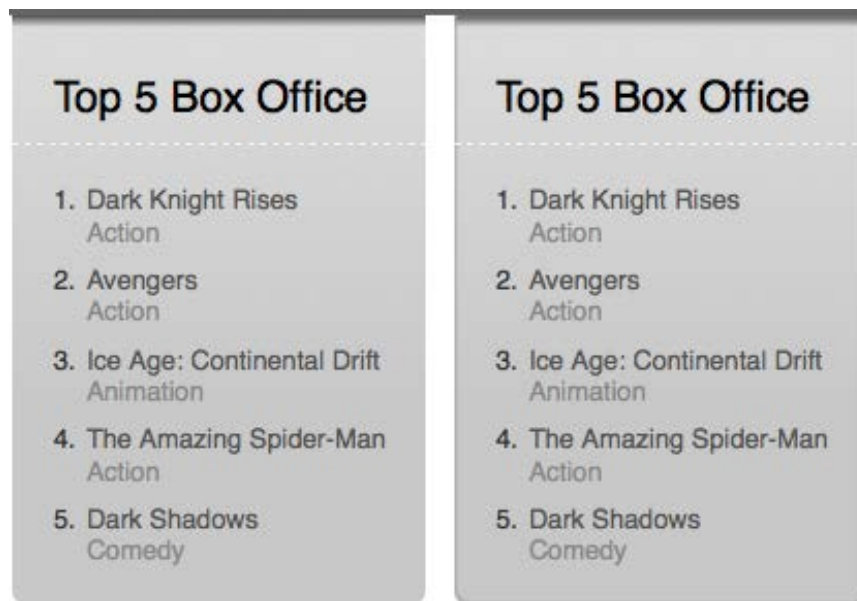
```
aside{
float:right;
width:200px;
padding:30px 0 10px 0;
margin:0 10px;
background-color:#E4E4E4;
/** TOP 5 ROUNDED BORDER **/
border-radius:0 0 8px 8px;
-moz-border-radius:0 0 8px 8px;
-webkit-border-radius:0 0 8px 8px;
-o-border-radius:0 0 8px 8px;
/** BOX OFFICE GRADIENT **/
background:linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-o-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-moz-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
}
```

```

background:-webkit-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
background:-ms-linear-gradient(bottom, rgb(200,200,200) 35%,
rgb(210,210,210) 68%, rgb(220,220,220) 98%, rgb(80,80,80) 100%);
/** BOX OFFICE INNER SHADOW **/
box-shadow:0 -1px 1px 1px #aaa inset;
-moz-box-shadow:0 -1px 1px 1px #aaa inset;
-webkit-box-shadow:0 -1px 1px 1px #aaa inset;
-o-box-shadow:0 -1px 1px 1px #aaa inset;
}

```

As a result, our **Top 5 Box Office** area looks deeper than before:



We can apply this to our wrapper to have shadows on the left and right borders:

```

.wrapper{
background-color:#fff;
/** PAGE SIDES SHADOWS **/
box-shadow: 1px 0 2px 1px #aaa;
-moz-box-shadow: 0 -1px 1px 1px #aaa;
-webkit-box-shadow: 0 -1px 1px 1px #aaa;
-o-box-shadow: 0 -1px 1px 1px #aaa;
}

```

While it is difficult to notice, the sum of these small details helps to reflect the visual richness of the enterprise application:



`box-shadow` is supported by all modern browsers except Internet Explorer, which only supports it from IE9 onwards.



Unfortunately, fake flexible drop shadows are hard to create or expensive because sometimes it is better to not use shadows in old browsers, following the principles of graceful degradation.

Adding text shadows

To add shadows to text we cannot use `box-shadow` because it applies the shadow to a square container. If we want to add shadows to any text, we should use the `text-shadow` property.

The syntax for `text-shadow` is as follows:

```
text-shadow: horizontal-shadow vertical-shadow blur color;
```

`text-shadow` is not supported by Internet Explorer, but for that case it is possible to use `filter: dropshadow` instead. The only downside (apart from compatibility) is that it is not possible to specify blur.

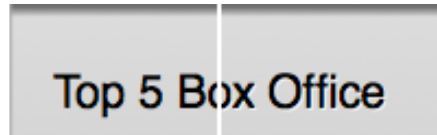
The syntax for `filter: dropshadow` is as follows:

```
filter: dropshadow(color=color, offx=horizontal-shadow,  
offy= vertical-shadow);
```

Even shadow effects are thought of as a way to pop out text. We can change the color property and fake inset elements. We use a light gray and 1px displacement horizontally and vertically with no blur in our **Top 5 Box Office** title:

```
aside h2{
  padding:0 20px 10px;
  margin:0 0 0;
  font-size:1.3em;
  text-shadow: 1px 1px 0px #f2f2f2;
  filter: dropshadow(color=#f2f2f2, offx=1, offy=1);
}
```

You can check the inset element (right):



We will use a more traditional approach in our navigation bar, including `blur` on hover:

```
nav a{
  color:#ccc;
  text-decoration:none;
}
nav a:hover{
  color:#fff;
  text-shadow: 2px 2px 1px #222;
  filter: dropshadow(color=#222222, offx=2, offy=2);
}
```



Some tricks to fake 3D

Some depth effects can be simulated by using CSS previous to Version 3. For example, we can use borders to simulate depth by placing a dark color border over a light color one:

```
aside ol{
  padding:20px 0 0 36px;
  margin:0;
  counter-reset:counter;
  border-top:1px dashed #f8f8f8;
}
aside h2{
  padding:0 20px 10px;
  margin:0 0 0;
  font-size:1.3em;
  text-shadow: 1px 1px 0px #f2f2f2;
  filter: dropshadow(color=#f2f2f2, offx=1, offy=1);
  border-bottom:1px dashed #bbb;
}
```

We can do this by applying this to our **Top 5 Box Office** area:



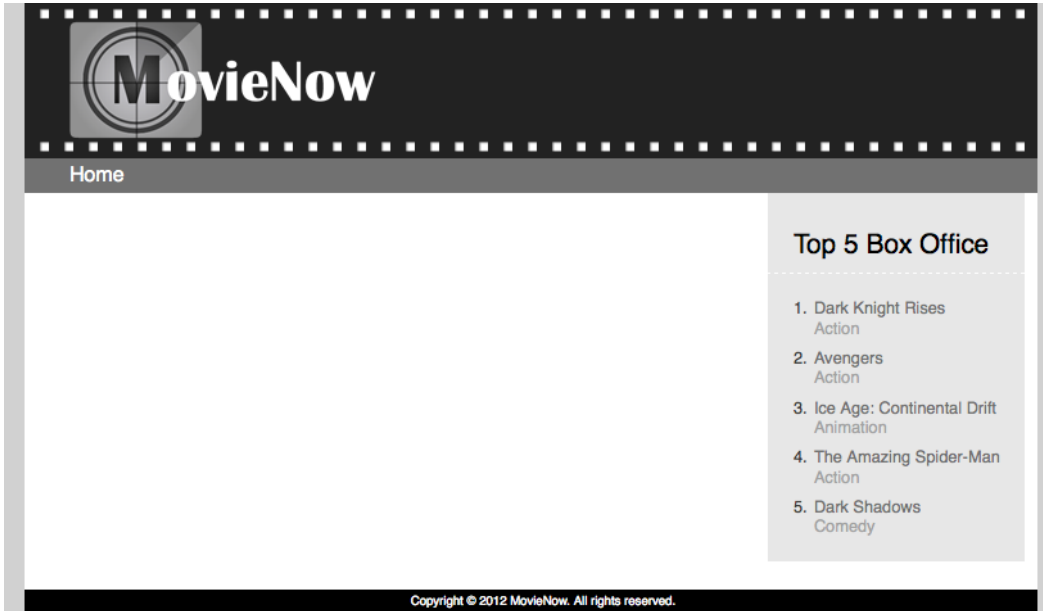
In some cases, the effect is not obvious but helps to give depth as a part of other effects:

```
header{
  ...
  border-bottom:1px solid #222;
}
nav{
  ...
  border-top:1px solid #777;
  font-size:.9em;
}
```

Applying this to the top of the navigation menu, we get the following:



With all of our effects applied to our initial layout:

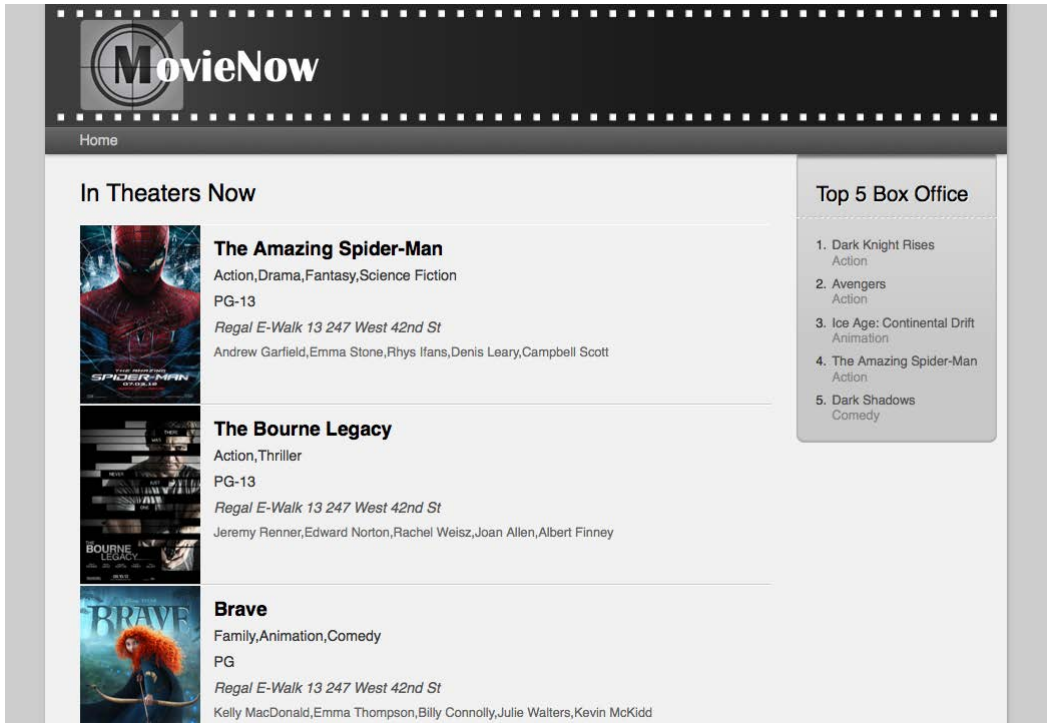


We see this:



Movies and styles

Suppose that we have a requirement for the creation of a movie list with simple information, where every element shows more details when you click. Because of the tight deadline, the client decides to have a simple implementation, so that the requirement goes to a web designer and as a result we have an initial design:



In a meeting with our web designer and our final client, we decide to show the movie synopsis on the click of the movie poster. Based on this, let us build a first approach to our structure.

For a list where the order does not mean anything, we can use an unordered list `ul`. We first add movie posters to identify each element easily (using `img`), and then we define two blocks using the `main-info` (to show as a default) and `description` (to show on click) sections. In `main-info`, we add the title as `h3`, genre and rate as `p` with the `details` class, movie theater with tag `p` and the `theater` class, and actors as `p` with the `actors` class. In the `details`, we add the title as `h3` and the description as `p`. Now, our structure looks like the following code:

```
<div id="movies-near-me">
  <ul>
```

```

    <li itemtype="http://schema.org/Movie" itemscope="" class="">
      
      <section class="main-info">
        <h3 itemprop="name">The Amazing Spider-Man</h3>
        <p itemprop="genre" class="details genre">Action,Drama,Fantasy,S
cience Fiction</p>
        <p class="details">PG-13</p>
        <p class="theater">Regal E-Walk 13 247 West 42nd St</p>
        <p class="actors">Andrew Garfield,Emma Stone,Rhys Ifans,Denis
Leary,Campbell Scott</p>
      </section>
      <section class="description">
        <h3 itemprop="name">The Amazing Spider-Man</h3>
        <p>Typical teenager Peter Parker (Andrew Garfield) embraces
his incredible destiny after uncovering one of his father's most
carefully guarded secrets as Columbia Pictures reboots the Spider-Man
franchis...</p>
      </section>
    </li>
  ...
</ul>
</div>

```

First, we add the AJAX call to the end of `movienow.geolocation.js`:

```
this.getLocation();
```

Then, we modify the structure of the AJAX callback with our structure:


```

this.displayShowtimes = function(movies) {
  var movie = null;
  var html = '<ul>';
  for (var item in movies.items) {
    movie = movies.items[item];
    var movieDesc=(movie.synopsis.length>200)?movie.synopsis.
substr(0,200)+"...": movie.synopsis;
    var movieHTML='<li itemscope itemtype="http://schema.org/Movie">';
    movieHTML+='';
    movieHTML+='<section class="main-info">';
    movieHTML+='<h3 itemprop="name">'+movie.title+'</h3>';
    movieHTML+='<p class="details genre"
itemprop="genre">'+Array(movie.genre).join(', ')+</p>';
    movieHTML+='<p class="details">'+movie.mpaarating+'</p>';
    movieHTML+='<p class="theater">'+movie.theater.title+" "+movie.
theater.address+'</p>';

```

```
        movieHTML+= '<p class="actors">'+Array(movie.selectedStar).join(',
    ')+'</p>';
        movieHTML+= '</section>';
        movieHTML+= '<section class="description">';
        movieHTML+= '<h3 itemprop="name">'+movie.title+'</h3>';
        movieHTML+= '</section>';
        movieHTML+= '</li>';
        html+=movieHTML;
    }
    html+= '</ul>';
    $('#movies-near-me').html(html);
    $('#movies-near-me li').click(function(){$(this).
toggleClass("open")});
};
```

We are creating our DOM structure concatenating a string, but if you want to use a more elegant solution you can use a client-side template library such as jQuery tmpl (<http://api.jquery.com/category/plugins/templates/>), Mustache (<http://mustache.github.com/>), Underscore (<http://documentcloud.github.com/underscore/>), or Pure (<http://beebole.com/pure/>). Template libraries allow you to separate the DOM structure from data. Some of them, such as Underscore, include logic.

 Notice that we limit the size of `movie.synopsys` using `substr`.

As we want to add some highlights in white, we should change the `wrapper` and `main-info` `background-color` structures to a light gray as we saw in the original design, so we can use:

```
.wrapper, #movies-near-me li section.main-info{
    background-color:#f1f1f1;
}
```

Our Top 5 box is floated right, so we can give some margin to our movies container allowing for a more flexible design. We will change the original width of our wrapper structure:

```
#movies-near-me{
    margin-right:200px;
}
```

Styling our list

We want to apply some animations later, so we will add `position: relative` to move the absolute positioned elements inside, using `li` as our point of reference. We add `overflow: hidden` to account for any elements out of our `li` area. We use borders `top` and `bottom` with light and dark colors respectively to add a sensation of depth. Finally, we add dark gray as `background-color` (not in the original design, but this will be covered with `main-info` and `img`) and we set the mouse cursor attribute to `pointer` to indicate that the element is clickable:

```
#movies-near-me li{
  position: relative;
  overflow: hidden;
  border-top: 1px solid #fff;
  border-bottom: 1px solid #ccc;
  background-color: #202125;
  cursor: pointer;
}
```

Let's float `img` to show `main-info` at its side and not below it. Oh, and some margin to leave a space between `img` and description text (that will be hidden for now):

```
#movies-near-me li img{
  float: left;
  margin-right: 10px;
}
```

We will be defining size, weight, and spacing for our titles as follows:

```
#movies-near-me li h3{
  font-size: 1.2em;
  font-weight: bold;
  padding: 10px 0 3px 14px;
}
```

We will add padding for each information inside `p` tags:

```
#movies-near-me li p{
  padding: 5px 14px;
}
```

Add a different text color and size for some details:

```
#movies-near-me li .details{
  color: #333;
  font-size: .9em;
}
```

We will define a different text color and size for the movie theater and italic style using the following declaration:

```
#movies-near-me li .theater{
  color:#555;
  font-style:italic;
  font-size:.9em;
}
```

Applying a new style for actors:

```
#movies-near-me li .actors{
  color:#666;
  font-size:.8em;
}
```

We define a static height that is the same as each movie poster image:

```
#movies-near-me li,#movies-near-me li section.main-info,
#movies-near-me li section.description{
  height:178px;
}
```

Absolute positioning is applied to `main-info` (to animate it later). We add `margin` equal to the `width` attribute of our movie poster image and some padding for our text inside:

```
#movies-near-me li section.main-info{
  position:absolute;
  top:0;
  left:0;
  right:0;
  margin-left:120px;
  padding: 5px 0;
}
```

Finally, we will be adding some styles for our hidden description, including an `inset box-shadow` attribute to simulate depth:

```
#movies-near-me li section.description{
  color:#f1f1f1;
  font-size:.9em;
  line-height:1.4em;
  box-shadow:1px -8px 3px 4px #000 inset;
  -moz-box-shadow:1px -8px 3px 4px #000 inset;
  -webkit-box-shadow:1px -8px 3px 4px #000 inset;
  -o-box-shadow:1px -8px 3px 4px #000 inset;
}
```

At this point our design looks the same as the image supplied by our web designer, but we still cannot see movie details. Before we satisfy this requirement, let us talk about transitions and animations.

Transitions

Usually we change classes of HTML elements based on interactions. For example, link styles on hover, show, and hiding of blocks of text on click for tabbed panels, and so on. Before CSS3, if we wanted to animate these changes, the only way to do it was with JavaScript. With CSS3, a simple way to do this is with `transition`. Having an initial class and a pseudo class triggered on an interaction, we can add a `transition` element with the properties that change between class and pseudo class to animate them.

The syntax for shorthand `transition` is as follows:

```
transition: transition-property transition-duration transition-timing-
function transition-delay
```

`transition-timing-function` specifies how fast transition occurs. Possible values for this are: `linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out`, and `cubic-bezier(n, n, n, n)`.

`transition-delay` is used if we wanted to start our animation in another point in time other than initial state (0s).

We can use multiple transitions at the same time:

```
transition: property1 duration1 easing1 start-point1, ..., propertyN
durationN easingN start-pointN
```



Transitions are triggered with interactions and have only two states: initial and final.

Animations

If we want to implement complex movement that involves multiple states, it is not possible to use transitions. For this, we have animations. Moreover, you do not need to trigger an interaction to start an animation (but we shall keep this a secret so as to avoid a new animated GIF boom in this era).

Animations rely on `@keyframes`. Similar to their counterpart in animation tools (including Adobe Flash), a keyframe allows you to define states and the values of the properties in them.

For example, we can use:

```
@keyframes animation-name
{
  from {width:0}
  to {width:50px}
}
```

Or more complex constructs using percentages and multiple properties:

```
@keyframes animation-name
{
  0%{width:0;height:0}
  20%{width:5px;height:2px}
  60%{width:7px;height:10px}
  100%{width:50px;height:12px}
}
```

We can specify as many steps as we want. `animation-name` is used later to call our keyframe.

The syntax used for animation is as follows:

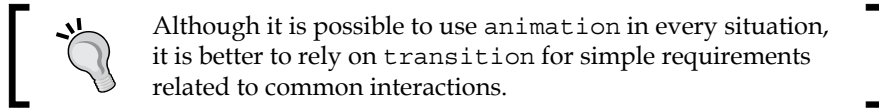
```
animation: animation-name animation-duration animation-timing-function
animation-delay animation-iteration-count animation-direction;
```

Most of the properties have the same meaning as transitions. `animation-iteration-count` specifies the number of times that the animation will repeat (or infinite if it never stops), `animation-direction` allows the animation to run normally (`normal`), or alternate back and forward (`alternate`).

Additionally, we have `animation-play-state` that is not on the shorthand mode. This property allows us to stop (`paused`) and start again (`running`) our animation.

Choosing between transitions and animations

In our case we have only two states, one that shows the general details of the movie and a pseudo class state that shows the description of the movie. This should be triggered on click, so the simplest solution is to use a transition.



Although it is possible to use animation in every situation, it is better to rely on transition for simple requirements related to common interactions.

In our case, we want to animate the `main-info` `left` and `right` properties. The initial state is 0 for both:

```
#movies-near-me li section.main-info{
  top:0;
  left:0%;
  right:0%;
  margin-left:120px;
  padding: 5px 0;
}
#movies-near-me li.open section.main-info{
  left:100%;
  right:-100%;
}
```

The final state will be `left:100%` (100% to the right side) and `right:-100%` (100% to the right from left side). We create a pseudo state with class `open` for `li`:

To change the class on click, we add a `toggleClass` call on click for each `li` using jQuery on `movienow.js`. `toggleClass` adds and removes the `open` class:

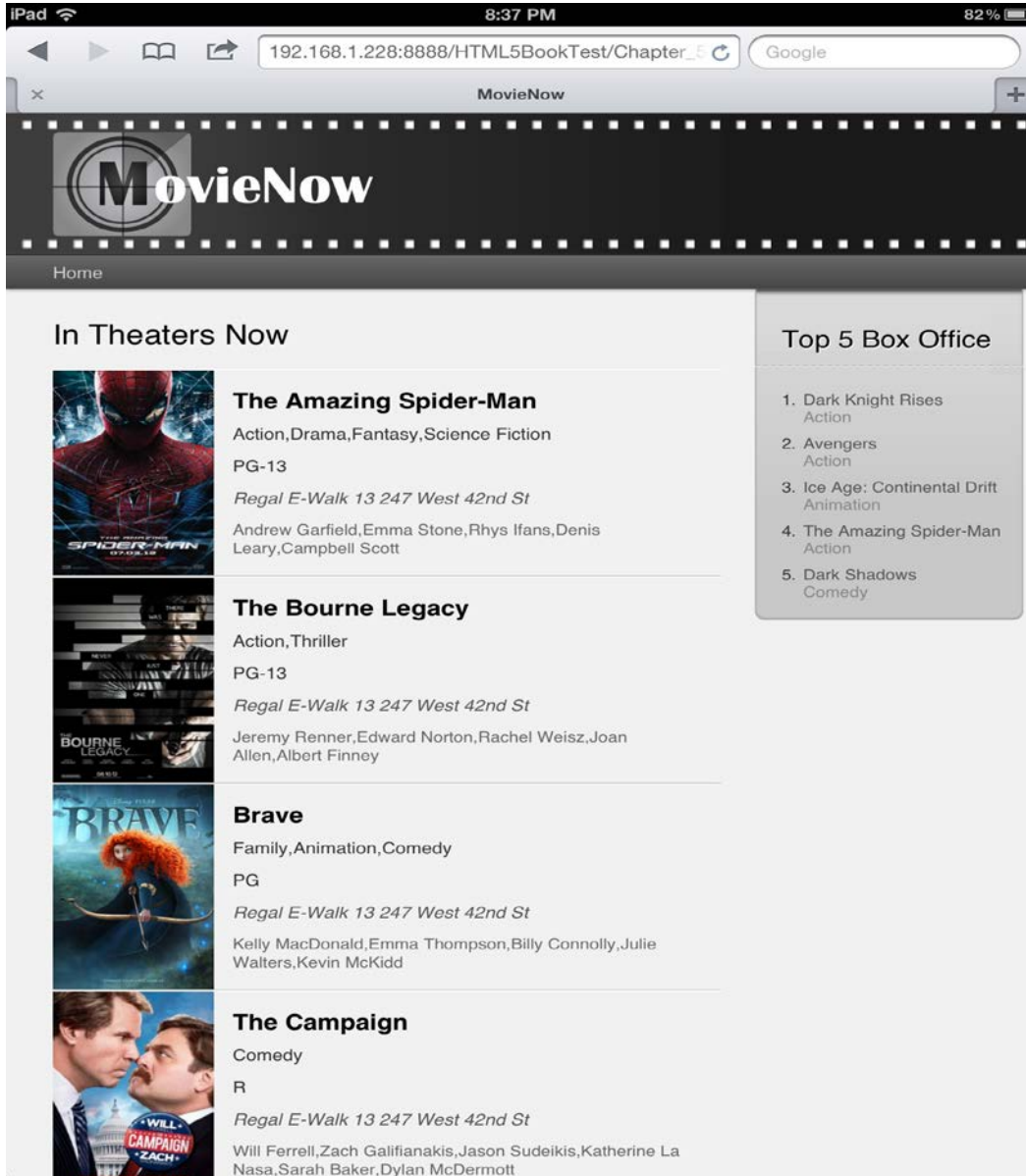
```
$("#movies-near-me li").click(function(){$(this).
toggleClass("open");});
```

If you click on each element, you will notice the change that shows and hides each description.

To add our transition, we specify the `left` and `right` properties and a duration of .3 seconds for each one. Using multiple browser prefixes we have the following code:

```
#movies-near-me li section.main-info{
  top:0;
  left:0%;
  right:0%;
  margin-left:120px;
  padding: 5px 0;
  transition: right .3s, left .3s;
  -moz-transition: right .3s, left .3s;
  -webkit-transition: right .3s, left .3s;
  -o-transition: right .3s, left .3s;
}
```



Testing again, we should see a fluid movement that goes from one state to another.





If we click on **Brave**, we will see an animation and then the movie synopsis, as partially depicted in the following screenshot.


The screenshot shows the MovieNow website interface on an iPad. The browser address bar displays the URL 192.168.1.228:8888/HTML5BookTest/Chapter_5. The website header features the MovieNow logo and a 'Home' link. The main content area is divided into two sections: 'In Theaters Now' and 'Top 5 Box Office'.

In Theaters Now

- 

The Amazing Spider-Man
Action, Drama, Fantasy, Science Fiction
PG-13
Regal E-Walk 13 247 West 42nd St
Andrew Garfield, Emma Stone, Rhys Ifans, Denis Leary, Campbell Scott
- 

The Bourne Legacy
Action, Thriller
PG-13
Regal E-Walk 13 247 West 42nd St
Jeremy Renner, Edward Norton, Rachel Weisz, Joan Allen, Albert Finney
- 

Brave
Since ancient times, stories of epic battles and mystical legends have been passed through the generations across the rugged and mysterious Highlands of Scotland. In 'Brave,' a new tale joins the lore...
- 

The Campaign
Comedy
R
Regal E-Walk 13 247 West 42nd St
Will Ferrell, Zach Galifianakis, Jason Sudeikis, Katherine La Nasa, Sarah Baker, Dylan McDermott

Top 5 Box Office

1. Dark Knight Rises
Action
2. Avengers
Action
3. Ice Age: Continental Drift
Animation
4. The Amazing Spider-Man
Action
5. Dark Shadows
Comedy

Let us add an animation to our header to test the `animation` property. Our header shows a movie roll film decoration. If we want to roll the roll film, we need to define some keyframes. In this case, we specify only two states: `from` and `to`. Because of our design, we move the roll horizontally from 0 to -19px (the space between white rectangles, to create the same initial and end state to our loop). We will add this with the respective browser prefixes and naming our keyframe with `movierolling`:

```
@keyframes movierolling{
  from {background-position: 0 0;}
  to {background-position: -19px 0;}
}
/* Firefox */
@-moz-keyframes movierolling{
  from {background-position: 0 0;}
  to {background-position: -19px 0;}
}
/* Safari and Chrome */
@-webkit-keyframes movierolling{
  from {background-position: 0 0;}
  to {background-position: -19px 0;}
}
/* Opera */
@-o-keyframes movierolling{
  from {background-position: 0 0;}
  to {background-position: -19px 0;}
}
```

We add `movierolling` as animation in our header specifying `.5 segs` `animation:movierolling .5s`, an infinite loop `animation-iteration-count:infinite`, and a linear easing to make a fluid loop `animation-timing-function:linear`. As a result, we have the following code:

```
header{
  color:#fff;
  height:122px;
  /** HEADER GRADIENT **/
  background-color:#1A1A1A;
  background-image:url(../img/logo_back.png), linear-gradient
(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
  background-image:url(../img/logo_back.png), -moz-linear-gradient
(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
  background-image:url(../img/logo_back.png), -webkit-linear-gradient
(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
  background-image:url(../img/logo_back.png), -o-linear-gradient
(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
}
```

```

background-image:url(../img/logo_back.png), -ms-linear-gradient
(right, rgb(26,26,26) 35%, rgb(40,40,40) 68%, rgb(61,61,61) 85%);
border-bottom:1px solid #222;
animation:movierolling .5s;
-moz-animation:movierolling .5s;
-webkit-animation:movierolling .5s;
-o-animation:movierolling .5s;
animation-iteration-count:infinite;
-moz-animation-iteration-count:infinite;
-webkit-animation-iteration-count:infinite;
-o-animation-iteration-count:infinite;
animation-timing-function:linear;
-moz-animation-timing-function:linear;
-webkit-animation-timing-function:linear;
-o-animation-timing-function:linear;
}

```

And the roll is rolling!

Let us comment out this animation code for now and get back to our application.

Using media queries

The transition that we added to visualize the synopses works nicely, but on mobile devices we do not have enough space to show the complete synopsis for each movie. A possible solution could be to hide the movie posters images for mobile devices, that should give us at least an additional 120 px.

As we saw in previous chapters, we can use media queries to specify different behaviors for different screen sizes. We can add a case for devices until 737 px:

```
@media only screen and (max-width: 737px) { ... }
```

Let us apply a transition with the same time of our main-info one, but in this case only for margin-left:

```

#movies-near-me li img{
  transition: margin-left .3s;
  -moz-transition: margin-left .3s;
  -webkit-transition: margin-left .3s;
  -o-transition: margin-left .3s;
}

```

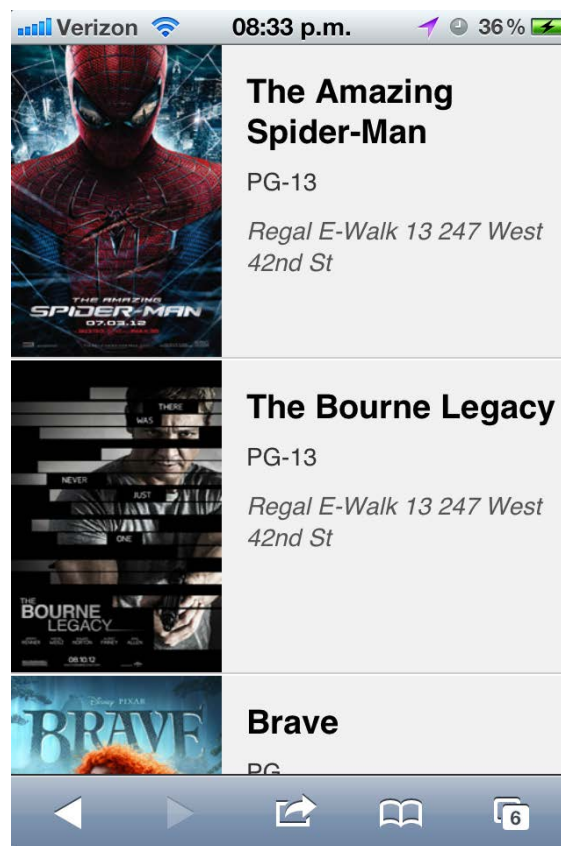
The final position should have a negative margin value to move our images outside of the li area:

```
#movies-near-me li.open img{  
    margin-left: -120px;  
}
```

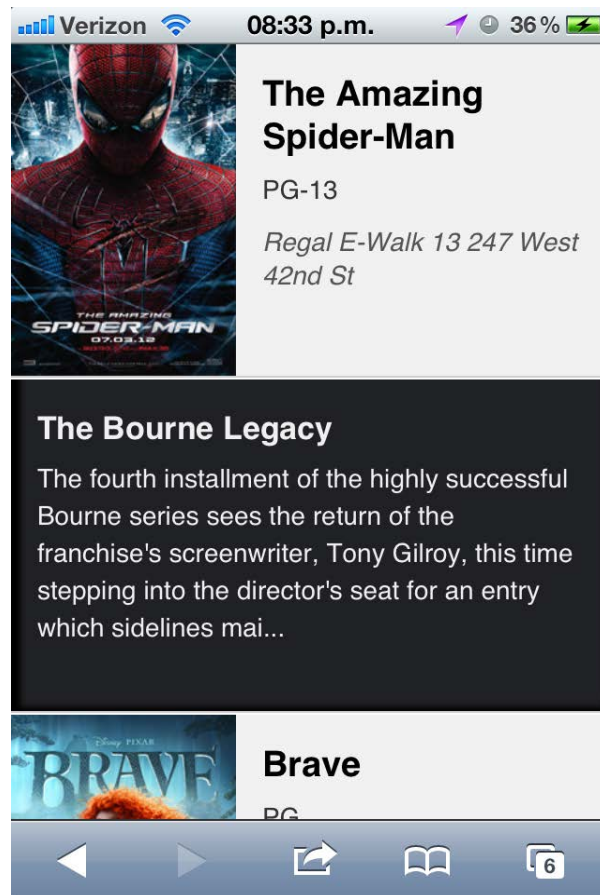
Additionally, let us hide actors and genres to have more space inside main-info:

```
#movies-near-me li.actors, #movies-near-me li.genre{  
    display:none;  
}
```

Now we can see a different interaction in small devices, allowing us to see the complete description:



If we click on **The Bourne Legacy**, we will see an animation and then the movie synopsis.



Applying CSS3 selectors

We have been using common selectors, but CSS3 introduces a new set that opens styling to new possibilities.

Most of these selectors are not supported in Internet Explorer 8 or the previous versions; you can verify support using the compatibility tables mentioned at the following link: <http://www.quirksmode.org/css/contents.html#CSS3>. You can always substitute these selectors for class declarations on your CSS and add these classes in your HTML using conditionals in your code.

We will not include this code as a part of our project, but you can test it using `styles.css` (code commented) in the `3.- selectors` folder to visualize the results.

- `:first-of-type`: This selects the first element with the selector's type. Let us say we want to apply a different `background-color` to the first element inside our movie list. We can select the first appearance of `li` followed by the selector that indicates which element must change its background color:

```
#movies-near-me li:first-of-type section.main-info{
  background-color:#ccc;
}
```

- `:last-of-type`: This is similar to the previous selector, but it selects the last element. Applying the same case as before:

```
#movies-near-me li:last-of-type section.main-info{
  background-color:#ccc;
}
```

- `:only-of-type`: This selects only unique elements of the type indicated. Using our movie's details, if we apply the following:

```
#movies-near-me li section.main-info h3:only-of-type{
  background-color:#ccc;
}
```

We can add `background-color` to `h3` since it is only `h3` contained by its parent, but if we use the following, nothing is selected because there are multiple `p` elements inside:

```
#movies-near-me li section.main-info p:only-of-type{
  background-color:#ccc;
}
```

- `:only-child`: This selects elements whose parents only contain them. For example, using this selector we can change `article` `background-color` because it is the only element contained by its parent.

```
article:only-child{
  background-color:#ccc;
}
```

But if we select `section`, nothing will be selected because there are multiple children in its parent.

```
section:only-child{
  background-color:#ccc;
}
```

- `:nth-child(n)`: This allows us to specify the element we want to select using positions. If we want to select the third element in our list:


```
#movies-near-me li:nth-child(3) section.main-info{
  background-color:#ccc;
}
```
- `:nth-last-child(n)`: This applies the same principle as the prior selector, but counting from the last element:


```
#movies-near-me li:nth-last-child(2) section.main-info{
  background-color:#ccc;
}
```
- `:nth-of-type(n)`: This uses the same principle as before, but it will count only elements of the same type. For example, if we apply `p:nth-of-type(2)` to select the second element, it will ignore any differences to `p`. Selecting the second `p` element, we have:


```
#movies-near-me li section.main-info p:nth-of-type(2){
  background-color:#ccc;
}
```
- `:nth-last-of-type(n)`: This does the same, but counts from the last element:


```
#movies-near-me li section.main-info p:nth-last-of-type(2){
  background-color:#ccc;
}
```
- `:last-child`: This selects the element that is the last child of its parent. Selecting the last movie, we have the following code snippet:


```
#movies-near-me li:last-child section.main-info{
  background-color:#ccc;
}
```
- `:root`: This allows us to select the `html` root tag. Let us change the `background-color` value of `html`, but first we reset the already defined `background-color` attribute for `html` and `body` tags:


```
html,body{
  background:none;
}
```

Add `background-color` to root (`html`):

```
:root{
  background-color:#666;
}
```


- `:empty`: This selects elements with no children or text. Let us show in red `div` elements with no content in our application:

```
div:empty{
  background-color:#ff0000;
}
```

You should see the logo area and `div.push` in red.

- `:target`: This selects elements with the `id` value equal to the active anchor. To test this we can define a link with an anchor and an `id` attribute to mark the link as active:

```
<nav>
  <ul>
    <li><a href="#home" id="home">Home</a></li>
  </ul>
</nav>
```

We can define the style to mark the text in yellow:

```
:target{
  color:#FFFF00;
}
```

If you click the link, you will see the color change.

- `:not(selector)`: This selects all elements that do not fulfill the conditions of the selector. For example, if we want to select all `p` elements except the ones with the `theater` class:

```
p:not(.theater){
  background-color:#ccc;
}
```

- `:enabled`: This selects input fields with no disabled property. If we have, `<input type="button" value="enable" />`, we can define an orange border using the following code:

```
input:enabled{
  border:1px solid #E38217;
}
```

- `:disabled`: This selects input fields with the disabled property. As before we can have:

```
<input type="button" value="disable" disabled="disabled" />
input:disabled{
  border:1px solid #E38217;
}
```

- `:checked`: This selects input with type checkbox that are checked. If we have the following code, we can see that the element changes style when it is checked:

```
<label><input type="checkbox" />Checked</label>
```

Applying style:

```
input:checked{  
  width:20px;  
  height:20px;  
}
```

- `element1~element2`: This selects element2 preceded by element1. If we want to select p elements preceded by h3, we can apply the following:

```
h3~p{  
  background-color:#ccc;  
}
```

- `[attribute^=value]`: This selects elements whose "attribute" begins with a particular "value". For example, let us hide the poster's images for every image whose alt attribute starts with Dark:

```
img[alt^="Dark"]{  
  display:none;  
}
```

- `[attribute$=value]`: This selects elements whose "attribute" ends with a particular "value". For example, let us hide the poster's images for every image whose alt attribute ends with s:

```
img[alt$="s"]{  
  display:none;  
}
```

- `[attribute*=value]`: This selects elements whose "attribute" contains a "value". For example, let us hide the poster's images for every image whose alt attribute contains ar:

```
img[alt*="ar"]{  
  display:none;  
}
```

Summary

New CSS3 features are not a new introduction to web development; they are a simplification of the execution. Before CSS3, it was possible to use gradients, drop shadows, rounded corners and even animations, but implementations were expensive and the scalability intricate. With all these possibilities, we should not forget old techniques that rely on images and complex JavaScript because even though we all hope for a simpler future based only in new generation browsers, we must face the problems of old generation browsers.

We have shown how to apply the most used CSS3 properties to our enterprise applications and how to manage compatibility issues related with styles across the browsers. Additionally, we introduced CSS3 animations and transitions, so now we are capable of selecting the right solution for our projects. Finally, we can apply media queries and selectors to our stylesheets for more complex and elegant solutions.

The following chapter will introduce HTML5 video and audio management, JavaScript control of media reproduction, and basic strategies to grant backward compatibility.

6

The App: Trailers via HTML5 Video

One of the most interesting features that HTML5 introduces is the ability to reproduce multimedia without additional plugins. Although this appears to be the right solution for any enterprise application that involves media management, there are still many factors to consider. This chapter covers the HTML5 `video` and `audio` tags, their use to play media, and some caveats related to the current state of this technology.

As an example we are going to build a video player for trailers and an audio player for podcasts.

This chapter includes:

- HTML5 video introduction
- Implementing a video player
- HTML5 audio introduction
- Implementing an audio player
- How I learned to stop worrying and love Flash


Introducing HTML5 video

For many years, browsers have relied on video reproduction to external plugins like Real Player, Quicktime, and Flash. Having 99 percent penetration of the market, Flash became a de facto standard for media playback; however, in the last few years, mobile devices have replaced this solution with native apps and HTML5 solutions.


HTML5 video surged to become a standard and elegant way to embed videos. While everything points to HTML5's video solution, the lack of agreement on which video formats should be supported has obstructed its use.

Ideally, there should be at least one format supported across all browsers, but every company has its own view on the matter. While Microsoft and Apple support MP4 H.264 (because they are patent holders of this format), Google and Mozilla back Ogg Theora and VP8 WebM as royalty-free solutions. The following table shows the browser support for each video format:

Browser	Operative System	Ogg Theora	MP4 H.264	VP8 WebM
Internet Explorer	Windows	Manual install	9.0	Manual install
	Windows Phone	No		No
Mozilla Firefox	Windows	3.5	Manual install	4.0
	Unix		No	
	Other			
Google Chrome	All supported	3.0	3.0 (removal planned)	6.0
Safari	iOS	No	3.1	No
	MacOS	Manual install		Manual install
	Windows		Manual install	
Opera	All supported	10.50	No	10.60

 A new compression standard known as **High Efficiency Video Coding (HEVC)** or H.265 could be in commercial products by 2013. It is almost twice as effective as the current H.264 standard.

Fortunately, the `video` tag supports the use of multiple sources allowing web browsers to select the video format supported, but this means each video needs to be encoded at least twice. For your enterprise, this translates to extra costs of encoding and storage.

 Some implementations rely on the video file extension too. For example, you cannot play a video on iOS devices with the `.f4v` extension even if it is using MP4 H.264 format.

Most web browsers support progressive download instead of streaming. While Flash has its own proprietary protocol to stream (although an incomplete version of the specification has been released for public use) known as **Real Time Messaging Protocol RTMP**, only Safari, Safari iOS, and some Android browsers support a new streaming protocol: **HTTP Live Streaming (HLS)** implemented by Apple Inc.

With progressive download, it is fairly easy to copy the video file from the browser cache, a facility that will make media pirates thankful. Furthermore, with a content delivery network that supports adaptive bitrate streaming, you can serve different video qualities depending on user bitrate if you are using streaming, but this is not possible using a progressive download.

Implementing a video player

MovieNow users would love to have a way to visualize trailers of their favorite movies. For that, we are going to create a player with basic functionality: play, pause, seek, volume control, and full screen.

We are going to use as an example a trailer of Sintel, an animated movie created with a free 3D animation tool known as Blender. This video trailer is hosted on the <http://www.w3.org/> site in three major video formats: MP4 (mp4), WebM (webm), and Ogg Theora (ogv).

First, let's create a file called `trailer.html` and use our main page structure.

Inside the `article` tag, we use the `video` tag, which allows us to specify an initial image using the `poster` attribute to specify image path, and to show default controls using the `controls` attribute.

```
<video poster="img/trailer-poster.png" controls>
```

You can specify the `src` property directly for the `video` tag, but to support multiple video format files we are going to declare our files using the `source` tag inside `video`. The `source` tag's `src` attribute allows us to define the video path and the `type` attribute (to specify the format).

```
<source type="video/mp4" src="path"></source>
```

In this case we are going to use:

- <http://media.w3.org/2010/05/sintel/trailer.mp4> for Chrome (while it is still supported), Internet Explorer, Safari, and Safari iOS
- <http://media.w3.org/2010/05/sintel/trailer.webm> for Firefox, Chrome, and Opera
- <http://media.w3.org/2010/05/sintel/trailer.ogv> for Firefox, Chrome, Opera, and others


In this case, it is possible to use only two formats, but we will use three for our example. Finally we have:

```
<video poster="img/trailer-poster.png" controls>
  <source type="video/mp4" src="http://media.w3.org/2010/05/sintel/
trailer.mp4"></source>

  <source type="video/webm" src="http://media.w3.org/2010/05/sintel/
trailer.webm"></source>
  <source type="video/ogg" src="http://media.w3.org/2010/05/sintel/
trailer.ogv"></source>

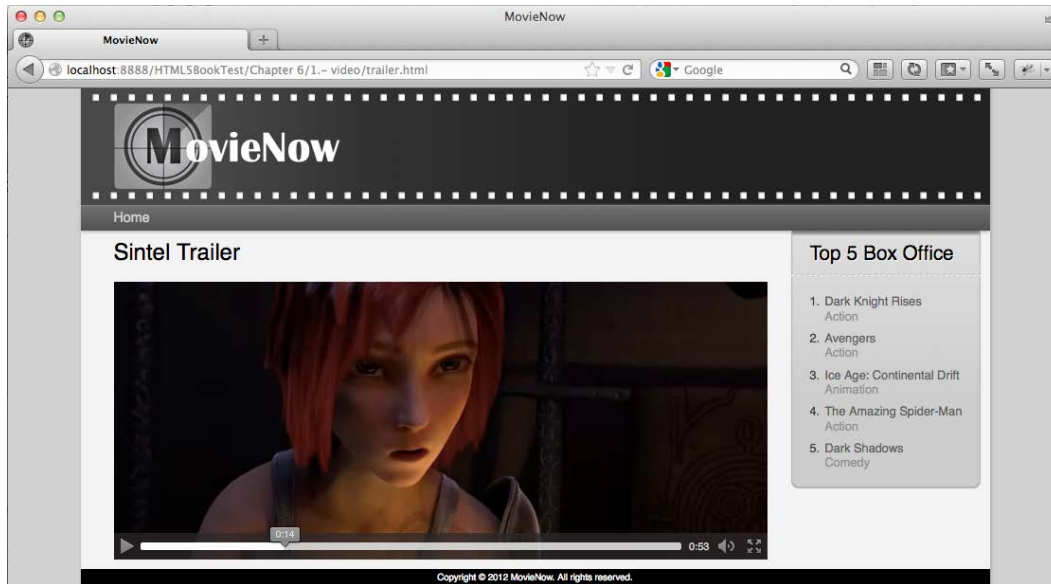
  <p>Video not supported.</p>
</video>
```

Notice that if no video is supported it shows `<p>Video not supported.</p>`. This can be whatever HTML content you want.

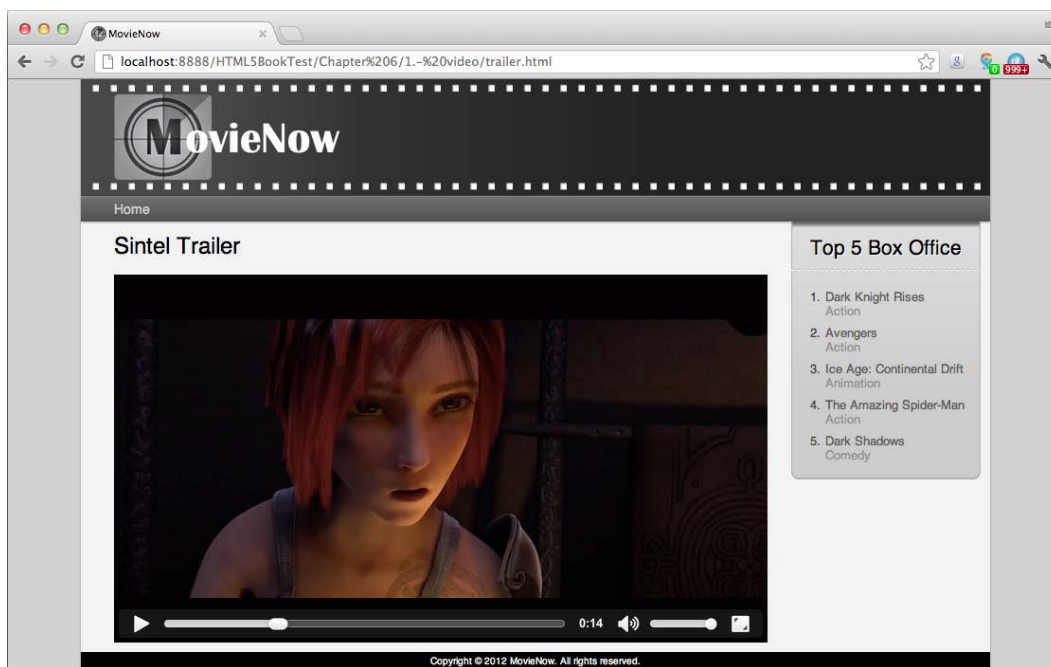
 If your content delivery network supports HLS, you can use a video encoded as H.264 broken in segments and use a .m3u8 playlist as an index file. For this, you can use a tool like Apple Stream Segmenter.

As every browser has its implementation, our player looks different in Firefox, Chrome, Safari, and so on. Our player renders differently on different platforms.

In Firefox our player renders as shown in the following screenshot:

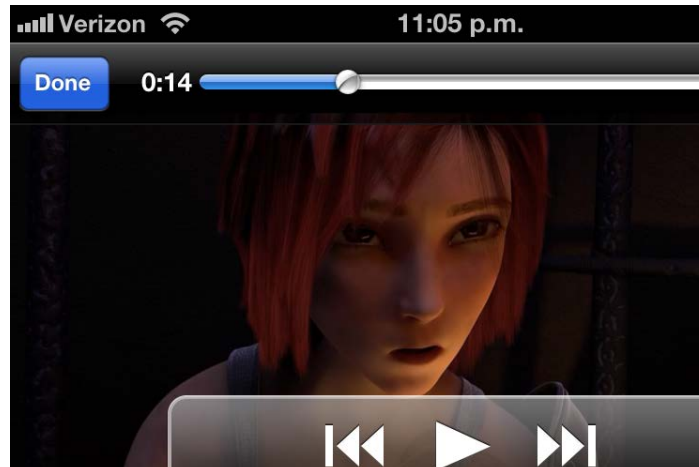


In chrome our player renders as shown in the following screenshot:



Chrome

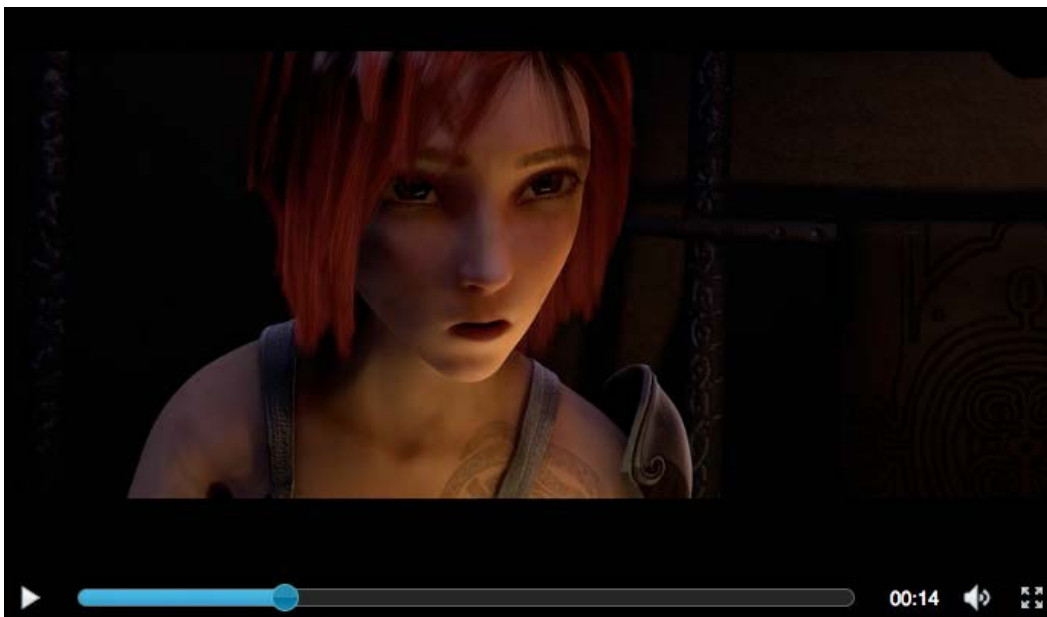
In an iPhone our player will appear as follows:



When it is necessary, reflect the enterprise visual style in the player or add custom functionality. It is possible to hide the default controllers and build your own.

Custom controls

For MovieNow we will be creating play/pause, seek, volume, and full screen controls. Our design can be seen in the following image:



To simplify the task of creating a progress and seek bar as well as a volume bar, we use jQuery UI. jQuery UI is a JavaScript library that implements the most common user interface elements and interactions like sliders, accordions, tabs, and so on. In our case, we created a custom download with the UI Darkness theme (<http://jqueryui.com/download>).

After saving our jQuery UI JavaScript file in the `js` folder and stylesheets in the `css` folder, we import them as always with JavaScript before the end of our `body` tag:

```
<script src="js/jquery-ui-1.8.23.custom.min.js"></script>
```

And `css` in our head tag:

```
<link rel="stylesheet" href="css/ui-darkness/jquery-ui-1.8.23.custom.css" type="text/css" />
```

To support jQuery UI interactions in touch devices, we import the Touch Punch JavaScript library (<http://touchpunch.furf.com/>):

```
<script src="js/jquery.ui.touch-punch.min.js"></script>
```

Now that we have all the libraries we need in place, we can remove the `controls` attribute from the `video` tag to hide the default controls.

```
<video poster="img/trailer-poster.png" class="media">
```

With this in place, let us define an HTML structure for our controls:

```
<div class="media-container">
  <div>
    <div class="media-area">
      <video poster="img/trailer-poster.png" class="media">

        <source type="video/mp4" src="http://media.w3.org/2010/05/sintel/trailer.mp4"></source>

        <source type="video/webm" src="http://media.w3.org/2010/05/sintel/trailer.webm"></source>

        <source type="video/ogg" src="http://media.w3.org/2010/05/sintel/trailer.ogv"></source>

        <p>Video not supported.</p>
      </video>
    </div>
  </div>
  <div class="controls">
    <div class="play-button"></div>
    <div class="seek"></div>
```

```
<div class="fullscreen-button"></div>
<div class="volume-container">
  <div class="volume-slider-container">
    <div class="volume-slider"></div>
  </div>
  <div class="volume-button"></div>
</div>
<div class="timer">00:00</div>
</div>
</div>
</div>
```

We have three main classes:

- `media-container` - wraps all our players
- `media-area` - wraps video tag
- `controls` - is the bottom bar that contains our controls

Inside controls we have:

- `play-button` - is the player's play/pause button
- `seek` - the progress/seek bar
- `fullscreen-button` - is the full screen functionality button
- `volume-container` - is the container of `volume-button`
- `volume-slider` - is used to set the volume
- `timer` - is an indicator of time elapsed in minutes and seconds (mm:ss)



We are using classes although jQuery selectors work faster using IDs because we want to permit the use of multiple players in the same page if necessary.



Styling

To start, we add some additional styles to `style.css`. We define a black background and remove the outline from all elements marked with the `media` class:

```
.media{
  width:100%;
  background-color:#000;
  outline:none;
}
```

We add a margin for the **Top 5 Box Office** section:

```
.media-container{
  margin-right:200px;
}
```

We remove that margin for small devices where we hide the **Top 5 Box Office** section:

```
@media only screen and (max-width: 737px){
  ...
  .media-container{
    margin-right:0;
  }
}
```

These styles are related to layout and not directly to our player. To make styles for our video player, let us create a stylesheet called `mediaplayer.css` and import it in the head tag of `trailer.html`.

```
<link rel="stylesheet" href="css/mediaplayer.css" type="text/css" />
```

Buttons and image sprites

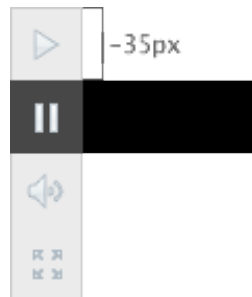
We define our controls area using the `controls` class, set a black background, set a height of `35px` (same line height to center text vertically), and set `position` to `relative` (so if we set absolute elements inside the positioning, they will be relative to `controls`).

```
.controls{
  background-color:#000;
  height:35px;
  line-height:35px;
  position:relative;
}
```

We have an image sprite with all of our player controls called `player-control.png`. You can find it inside the `img` folder.



The use of sprites is based on masking the visible element to show it and hiding the rest. In this case, suppose we want to show only our pause button. Our button has 35 x 35 pixels with `player-control.png` as the background image; the only part of the image that will be visible is inside our button area, so we can use background positioning to show different icons as is shown here:



Defining the play, volume, and full screen buttons that we have:

```
.play-button, .volume-button, .fullscreen-button{
  width:35px;
  height:35px;
  cursor:pointer;
  background-image:url(../img/player-control.png);
}
```

As we saw before, we move -35px on the y axis to show our pause icon:

```
.play-button.playing{
  background-position:0 -35px;
}
```

We apply the same principle to the full screen button:

```
.fullscreen-button{
  background-position:0 -105px;
}
```

In the case of the volume button, we are going to have a slider to set the volume below it, so we set the background color to hide elements below and set the position to absolute with z-index of 1000 to be over the slider:

```
.volume-button{
  background-position:0 -70px;
  position:absolute;
  left:0;
  background-color:#000;
  z-index:1000;
}
```

Styling seek and volume bars

The seek and volume bars can be styled as follows:

1. Let us define the font styles in timer:

```
.timer{
  color:#fff;
  font-size:.8em;
  padding:1px 8px 0;
}
```

2. At this point, we position play-button by floating it to the left-hand side and the timer, volume, and full screen to the right-hand side:

```
.play-button, .seek{
  float:left;
}
.timer, .volume-container, .fullscreen-button{
  float:right;
}
```

3. We can float the video tag left with the media class to avoid extra spacing in some browsers:

```
.media{
  float:left;
}
```

4. Use overflow:hidden to wrap media-area around media:

```
.media-area{
  overflow:hidden;
  background-color:#000;
}
```

5. Using absolute positioning for the seek bar, we can dynamically expand the seek area using the left and right properties:

```
.seek{
  top:12px;
  left:48px;
  right:133px;
  height:10px;
  position:absolute;
}
```



For this example, we created a flexible player to show some techniques related to styles, but it is good practice to define static dimensions for your player. Furthermore, it is better to use standard resolutions. The use of standard resolutions can improve performance for media reproduction on the client.

Features detection

Some of the `video` tag features are not available to all browsers via the HTML5 JavaScript API. For example, iOS devices disallow the use of volume control with JavaScript; it is only possible to use the default controls or hardware controls. Manipulating full screen controls using JavaScript is only possible in WebKit browsers.

We can define some classes to hide buttons when full screen or volume capabilities are not available. First, we hide our buttons:

```
.no-fullscreen .fullscreen-button{
  display:none;
}
.no-volume .volume-container{
  display:none;
}
```

Then, we change the right spacing of our seek bar:

```
.no-volume .seek,.no-fullscreen .seek{
  right:88px;
}
.no-fullscreen.no-volume .seek{
  right:53px;
}
```

Styling sliders

Since we are using jQuery UI to implement seek and volume sliders, we want to override some styles. jQuery UI sliders use the following:

- `ui-slider-handle`: The circle that we use to drag and seek.
- `ui-state-active`: The class added to `ui-slider-handle` while we drag.
- `ui-slider-range`: The bar that defines the active area. In our case, it is a blue bar.

Let's see the process step by step.

1. We want the same color for `ui-slider-handle` even when it is active, so we remove the background image used by jQuery UI:

```
.ui-state-active{
  background-image:none;
}
```

2. Add the cursor pointer and remove the outline:

```
.ui-slider .ui-slider-handle{
  cursor:pointer;
  outline:none;
}
```

3. Change `ui-slider-handle` size, rounded corners, and move it a little to the top (only for our seek slider):

```
.seek .ui-slider-handle {
  width:16px;
  height:16px;
  top: -4px;
  -moz-border-radius:10px;
  -ms-border-radius:10px;
  -webkit-border-radius:10px;
  border-radius:10px;
}
```

4. Modify the rounded corners of the progress bar and add some inner shadows:

```
.ui-slider-range {
  -moz-border-radius:15px;
  -ms-border-radius:15px;
  -webkit-border-radius:15px;
  border-radius:15px;
  box-shadow:inset 0 -3px 3px #39a2ce;
}
```

5. Change the seek bar progress color to a blue gradient:

```
.seek .ui-slider-range {
  background: #4cbae8;
  background-image:-moz-linear-gradient(top, #4cbae8, #39a2ce);
  background-image:-webkit-gradient(linear,left top,left
  bottom,color-stop(0, #4cbae8),color-stop(1, #39a2ce));
}
```


6. Change the volume progress color to a solid blue:

```
.volume-slider .ui-slider-range {  
    background:#4cbae8;  
}
```

7. Center our volume slider using margin and set width and height:

```
.volume-slider{  
    margin:12px auto;  
    width:6px;  
    height:76px;  
}
```

8. Set the volume handle dimensions and positioning:

```
.volume-slider .ui-slider-handle {  
    width:12px;  
    height:12px;  
    left: -4px;  
}
```

9. To show and hide the volume slider, we set the volume-container positioning as relative:

```
.volume-container{  
    width:35px;  
    height:35px;  
    position:relative;  
}
```

10. Set the slider positioning as absolute. We set z-index to 900 (below the volume button), overflow to hidden, and a CSS transition for all properties:

```
.volume-slider-container{  
    -moz-transition:all 0.1s ease-in-out;  
    -ms-transition:all 0.1s ease-in-out;  
    -o-transition:all 0.1s ease-in-out;  
    -webkit-transition:all 0.1s ease-in-out;  
    transition:all 0.1s ease-in-out;  
    position:absolute;  
    bottom:1px;  
    left:0;  
    height:34px;  
    width:35px;  
    background-color:#000;  
    z-index:900;  
    overflow:hidden;  
}
```

11. We can then resize `volume-container` on hover and `volume-slider-container` with it:

```
.volume-container:hover .volume-slider-container{
  height:135px;
}
```

Now that our player looks the same, let us add all the interactions needed using JavaScript.

Adding interactions using JavaScript

To write our JavaScript code, we create a `movienow.mediaplayer.js` file in the `js` folder and include it before our body ends:

```
<script src="js/movienow.mediaplayer.js"></script>
```

Initial settings

We start our JavaScript the same way we started with geolocation by adding `mediaplayer` to our namespace and defining the `that` variable:

```
var movienow = movienow || {};
movienow.mediaplayer = (function(){
  var that = this;

  /** OUR CODE GOES HERE **/
})();
```

Initializing video controllers

When the `ready` document is triggered, we add the click event listeners to buttons, detect full screen capabilities, and add the `no-fullscreen` class if it is not available; initialize the jQuery UI slider for seeking and for volume control if it is available. Notice that we manage Mozilla, WebKit, and standard full screen capabilities with different functions. If volume is not available, we add the `no-volume` class and finally we bind the events of time update and reproduction ended.

```
$(document).ready(function(){

  $(".media-container .play-button").click(that.play);
  var mediaElements=$(".media-container .media");

  if (mediaElements[0].fullscreenEnabled) {
    $(".media-container .fullscreen-button").click(that.fullScreen);
  }else if(mediaElements[0].mozRequestFullScreen){
```

```
    $(".media-container .fullscreen-button").click(that.
mozFullScreen);
  }else if(mediaElements[0].webkitRequestFullScreen){
    $(".media-container .fullscreen-button").click(that.
webkitFullScreen);
  }else{
    $(".media-container").addClass("no-fullscreen");
  }
$(".media-container .seek").each(function() {
  var duration=that.getPlayer($(this))[0].duration;
  duration = duration?duration:0;
  $(this).slider({
    value: 0,
    step: 0.01,
    orientation: "horizontal",
    range: "min",
    max: duration,
    start: function(event,ui){
      var mediaArea=that.getPlayer($(event.target));

      mediaArea.addClass("seeking");
      mediaArea[0].pause();
    },
    slide:function(event,ui){
      sliderTime(event,ui);
    },
    stop:function(event,ui){
      var mediaArea=that.getPlayer($(event.target));
      controls=that.controls(mediaArea);
      sliderTime(event,ui);
      if(controls.find(".play-button").hasClass("playing")){
        mediaArea[0].play();
      }
      mediaArea.removeClass("seeking");
    }
  });
  if(navigator.userAgent.match(/(iPhone|iPod|iPad)/i)){
    $(".media-container").addClass("no-volume");
  }else{
    that.controls($(this)).find(".volume-slider").slider({
      value: 1,
      step: 0.05,
      orientation: "vertical",
      range: "min",
```

```

        max: 1,
        animate: true,
        slide: function(event, ui) {
            var mediaArea=that.getPlayer($(event.target));
            mediaArea[0].volume=ui.value;
        }
    });
}
});
mediaElements.bind("timeupdate", that.timeUpdate);
mediaElements.bind('ended', that.endReproduction);
});

```

Setting the seek slider

To set the seek slider, we set the initial value value to 0 and step to 0.01 to have a fluid movie movement on drag, orientation to horizontal, and range to min to consider the range between minimum value and current handle position value:

```

$(this).slider({
    value: 0,
    step: 0.01,
    orientation: "horizontal",
    range: "min",
    max: duration,
    start: function(event, ui) {
        var mediaArea=that.getPlayer($(event.target));

        mediaArea.addClass("seeking");
        mediaArea[0].pause();
    },
    slide: function(event, ui) {
        sliderTime(event, ui);
    },
    stop: function(event, ui) {
        var mediaArea=that.getPlayer($(event.target));
        controls=that.controls(mediaArea);
        sliderTime(event, ui);
        if(controls.find(".play-button").hasClass("playing")){
            mediaArea[0].play();
        }
        mediaArea.removeClass("seeking");
    }
});

```

There are three events managed:

- `start` is triggered when the slider handle is pressed. Notice that we get video using the `getPlayer` function (this method will be declared later). We can pause the reproduction and add the `seeking` class to indicate that we are still dragging.
- `slide` is triggered while we drag. We call the `slideTime` function to set the progress bar position and time text.
- `stop` is triggered on mouse up. We get the `video` tag and controls using the `controls` function, call `sliderTime` and restore the previous state of our player (playing or paused) using the `play-button` `playing` class. Finally, we remove `seeking` to indicate that we stop dragging.

Initializing the volume slider

If volume is available, we initialize the volume slider:

```
that.controls($(this)).find(".volume-slider").slider({
  value: 1,
  step: 0.05,
  orientation: "vertical",
  range: "min",
  max: 1,
  animate: true,
  slide: function(event, ui) {
    var mediaArea=that.getPlayer($(event.target));
    mediaArea[0].volume=ui.value;
  }
});
```

Notice that the current value of our slider is contained in the `ui.value` variable, and to set it in our player we use the `volume` property shown as follows:

```
mediaArea[0].volume=ui.value;
```

Functions to get DOM objects

We define two functions to execute jQuery selectors for the main player (the `media` class for the `video` tag or the `audio` tag if it is the case) and controls (the `controls` class):

```
this.getPlayer= function(domObject) {
  return $(domObject.parentsUntil(".media-container").find(".media"));
};
```

```

this.controls= function(domObject){
    return $(domObject.parentsUntil(".media-container").find(".controls"));
};

```

Play and pause

For play-button, we toggle the playing class and set our player to the playing (player[0].play()) or paused (player[0].pause()) state.

```

this.play = function(event){
    var button=$(event.target);
    var player=that.getPlayer(button);
    if(button.hasClass("playing")) {
        player[0].pause();
        button.removeClass("playing");
    } else {
        player[0].play();
        button.addClass("playing");
    }
};

```

Full screen

Full screen functionality is managed in different ways by every browser. To enter the full screen mode we use `element.requestFullscreen()` and its equivalents `element.mozRequestFullScreen()` for Firefox and `element.webkitEnterFullscreen()` for Safari and Chrome. To exit full screen mode, we use `document.cancelFullScreen()`, `document.mozCancelFullScreen()` for Firefox, and `document.webkitCancelFullScreen()` for Safari and Chrome. Finally, to validate if the browser is in full screen mode we use `document.fullScreen`, `document.mozfullScreen` for Firefox, and `this.webkitFullScreen` for Safari and Chrome.

Even user experience-wise the browsers vary; while Chrome and Safari show their own video controllers on full screen, Firefox doesn't show any controls by default. Full screen capabilities are not available in Internet Explorer. Our implementation verifies the mode and toggles between full screen and normal mode.

Using standard calls we have:

```

this.fullScreen = function(event){
    var button=$(event.target);
    var player=that.getPlayer(button);
    if(document.fullScreen){
        document.exitFullScreen();
    }
};

```

```
    } else {  
      player[0].requestFullScreen();  
    }  
  };
```

Using the Firefox prefix:

```
this.mozFullScreen = function(event) {  
  var button=$(event.target);  
  var player=that.getPlayer(button);  
  if(document.mozFullScreen) {  
    document.mozCancelFullScreen();  
  } else {  
    player[0].mozRequestFullScreen();  
  }  
};
```

Finally, for Safari and Chrome we have:

```
this.webkitFullScreen = function(event) {  
  var button=$(event.target);  
  var player=that.getPlayer(button);  
  if(document.webkitIsFullScreen) {  
    document.webkitCancelFullScreen();  
  } else {  
    player[0].webkitEnterFullScreen();  
  }  
};
```

Notice that the event to exit full screen mode is not being triggered because the browsers manage that functionality using the *Esc* key, but depending on future implementations of the HTML5 full screen specification on every browser, we could show our controller in full screen mode and take advantage of this.

Format time

We define the `timeFormat` function to get the player time in seconds and return it in `mm:ss` format:

```
this.timeFormat=function(seconds) {  
  var m=Math.floor(seconds/60)<10?"0"+Math.floor(seconds/60):Math.  
  floor(seconds/60);  
  
  var s=Math.floor(seconds-(m*60))<10?"0"+Math.floor(seconds-  
  (m*60)):Math.floor(seconds-(m*60));  
  return m+":"+s;  
};
```

Controlling time

Every time we use the seek slider, we set media player time using the `currentTime` property, which triggers the `timeupdate` event calling the `timeUpdate` function.

```
this.sliderTime = function(event, ui) {
    var mediaArea=that.getPlayer($(event.target));
    var controls=that.controls(mediaArea);

    mediaArea[0].currentTime=ui.value;
};
```

`timeUpdate` sets the time in mm:ss and if the player is not in the seeking state (defined by the `seeking` class in `mediaArea`), it updates the progress/seek bar too. This function is invoked when the `timeupdate` event is triggered:

```
this.timeUpdate = function(event) {
    var mediaArea=$(event.target);
    var controls=that.controls(mediaArea);
    var currentTime=mediaArea[0].currentTime;
    var duration=mediaArea[0].duration;
    var timer=$(controls.find(".timer"));
    if(currentTime>=0)timer.html(that.timeFormat(currentTime));
    if(!mediaArea.hasClass("seeking")){
        var seekSlider=$(controls.find(".seek"));
        if(seekSlider.slider("option","max")==0){
            var newDuration=mediaArea[0].duration;

            newDuration=newDuration?newDuration:0;

            seekSlider.slider("option","max", newDuration);
        }
        seekSlider.slider("value", currentTime);
    }
};
```

Until the end of time

When the reproduction ends, `endReproduction` is called and we remove the playing class from `play-button` to indicate that we have finished the reproduction:

```
this.endReproduction = function(event) {
    var mediaArea=$(event.target);
    $(that.controls(mediaArea)).find(".play-button").
removeClass("playing");
};
```


The final script should look like the following code snippet:

```
var movienow = movienow || {};  
movienow.mediaplayer = (function(){  
  var that = this;  
  $(document).ready(function(){  
    /** play/pause button click event listener ***/  
  
    $(".media-container .play-button").click(that.play);  
  
    var mediaElements($(".media-container .media");  
    if(mediaElements[0].fullscreenEnabled) {  
  
      /** fullscreen button click event listener ***/  
  
      $(".media-container .fullscreen-button").click(that.fullScreen);  
    }else if(mediaElements[0].mozRequestFullScreen){  
  
      /** fullscreen button click event listener mozilla ***/  
  
      $(".media-container .fullscreen-button").click(that.  
mozFullScreen);  
    }else if(mediaElements[0].webkitRequestFullScreen){  
      /** fullscreen button click event listener webkit ***/  
  
      $(".media-container .fullscreen-button").click(that.  
webkitFullScreen);  
    }else{  
      /** we add class no-fullscreen to hide fullscreen button when  
it is not available ***/  
  
      $(".media-container").addClass("no-fullscreen");  
  
    }  
  
    /** Loop to add jquery ui sliders to progress/seek bar and volume  
***/  
  
    $(".media-container .seek").each(function() {  
      /** Duration of the media ***/  
  
      var duration=that.getPlayer($(this))[0].duration;  
  
      duration = duration?duration:0;  
  
      $(this).slider({  
        value: 0,  
  
        step: 0.01,  

```

```
orientation: "horizontal",

range: "min",

max: duration,

/** Start seek */
start: function(event,ui) {
    var mediaArea=that.getPlayer($(event.target));

    /** Class seeking to know status of the media player */

    mediaArea.addClass("seeking");
    mediaArea[0].pause();

},

/** During seek */

slide:function(event,ui) {

    sliderTime(event,ui);
},

/** Stop seek */

stop:function(event,ui) {

    var mediaArea=that.getPlayer($(event.target));
    var controls=that.controls(mediaArea);

    sliderTime(event,ui);
    /** We restore the status (playing or not) to the one before
start seeking */

    if(controls.find(".play-button").hasClass("playing")) {

        mediaArea[0].play();

    }

    mediaArea.removeClass("seeking");
}

});

/** Volume controllers */

if(navigator.userAgent.match(/(iPhone|iPod|iPad)/i)) {
```

```
    /*** ios devices only allow to change volume using the device
hardware, so we hide volume controllers ***/

    $(".media-container").addClass("no-volume");

}else{

    /*** volume slider controller ***/

    that.controls($(this)).find(".volume-slider").slider({

        value: 1,

        step: 0.05,

        orientation: "vertical",

        range: "min",

        max: 1,

        animate: true,

        slide:function(event,ui){

            var mediaArea=that.getPlayer($(event.target));
            mediaArea[0].volume=ui.value;

        }

    });

}

});

/*** event triggered when time change on media player ***/

mediaElements.bind("timeupdate", that.timeUpdate);

/*** event triggered when reproduction end on media player ***/
mediaElements.bind('ended', that.endReproduction);
});

/*** get player using jQuery selectors ***/

this.getPlayer= function(domObject){
```

```
        return $(domObject.parentsUntil(".media-container").find(".
media"));
    };

    /** get control area using jQuery selectors */

    this.controls= function(domObject){

        return $(domObject.parentsUntil(".media-container").find(".
controls"));
    };

    /** play or pause and change play button icon */

    this.play = function(event){

        var button=$(event.target);

        var player=that.getPlayer(button);

        if(button.hasClass("playing")) {

            player[0].pause();
            button.removeClass("playing");
        }else{
            player[0].play();

            button.addClass("playing");
        }
    };

    /** set on and off fullscreen mode */

    this.fullScreen = function(event){

        var button=$(event.target);

        var player=that.getPlayer(button);

        if ($(document).context.fullScreenElement){

            $(document).context.exitFullscreen();
        }else{
            player[0].requestFullscreen();
        }
    };
};
```

```
this.mozFullScreen = function(event) {  
    var button=$(event.target);  
    var player=that.getPlayer(button);  
    if ($(document).context.mozFullScreenElement) {  
        $(document).context.mozCancelFullScreen();  
    }else{  
        player[0].mozRequestFullScreen();  
    }  
};  
  
this.webkitFullScreen = function(event){  
    var button=$(event.target);  
    var player=that.getPlayer(button);  
    if ($(document).context.webkitIsFullScreen) {  
        $(document).context.webkitCancelFullScreen();  
    }else{  
        player[0].webkitEnterFullScreen();  
    }  
};  
  
/** set time format to mm:ss */  
  
this.timeFormat=function(seconds) {  
    var m=Math.floor(seconds/60)<10?"0"+Math.floor(seconds/60):Math.  
floor(seconds/60);  
  
    var s=Math.floor(seconds-(m*60))<10?"0"+Math.floor(seconds-  
(m*60)):Math.floor(seconds-(m*60));  
    return m+": "+s;  
};  
  
/** use by seek slider, change slider position and time on  
controllers */  
  
this.sliderTime = function(event, ui) {  
    var mediaArea=that.getPlayer($(event.target));  
  
    var controls=that.controls(mediaArea);  
  
    mediaArea[0].currentTime=ui.value;  
};
```

```
};

/** use by timeupdate event, change slider position and time on
controllers */

this.timeUpdate = function(event) {
    var mediaArea=$(event.target);
    var controls=that.controls(mediaArea);
    var currentTime=mediaArea[0].currentTime;
    var duration=mediaArea[0].duration;
    var timer=$(controls.find(".timer"));
    if(currentTime>=0)timer.html(that.timeFormat(currentTime));
    if(!mediaArea.hasClass("seeking")){
        var seekSlider=$(controls.find(".seek"));
        /** some players (like safari) don't have duration when a
player is initialized, this verify duration and assigned again to max
property on slider */
        if(seekSlider.slider("option","max")==0){
            var newDuration=mediaArea[0].duration;
            newDuration=newDuration?newDuration:0;
            seekSlider.slider("option","max", newDuration);
        }
        seekSlider.slider("value", currentTime);
    }
};

/** change play button when reproduction ends */

this.endReproduction = function(event) {
    var mediaArea=$(event.target);

    $(that.controls(mediaArea)).find(".play-button").
removeClass("playing");
};
}());
```

As a result we have a video player for multiple platforms:



Possible improvements

At this point we have a fully functional player, but we can add more improvements in the future. An interesting functionality to add is a buffering notification. To achieve this, you will need to listen to the `loadstart` event to recognize the start of loading of a video, `waiting` and `stalled` (depending on the browser: <http://www.longtailvideo.com/html5/buffering/>) to detect a stop in the reproduction because of buffering, and finally `canplay` and `canplaythrough` to recognize the end of buffering.

On `loadstart`, `waiting`, and `stalled` a buffering notification should be shown and on `canplay` and `canplaythrough` that notification should be hidden.

Still not perfect

The HTML5 video specification is still in progress. Major inconsistencies exist because of multiple implementation decisions across browsers and platforms requiring different encodings. Nevertheless, it is a standard way of supporting video without plugins.

Introducing HTML5 audio

The HTML5 audio specification – much like HTML5 video – is still in development, and there is no audio format supported across all browsers. Motives for this are the same ones that have been impeding standardized support of HTML5 video as you can see in the following table:

Browser	Ogg Vorbis	WAV PCM	MP3	AAC
Internet Explorer	No	No	9	9
Mozilla Firefox	3.5	3.5	No	No
Google Chrome	6	6	6	6
Safari	Manual install	5	5	5
Opera	10.6	10.6	No	No

Implementing an audio player

MovieNow needs an audio podcast player. For that, we are going to use the HTML5 audio tag.

The audio tag behaves more or less the same as the video tag:

```
<audio>
  <source src="http://www.w3schools.com/html5/horse.ogg" type="audio/ogg" />
  <source src="http://www.w3schools.com/html5/horse.mp3" type="audio/mp3" />
  <p>Audio not supported.</p>
</audio>
```



Like the video tag, the audio tag allows you to specify the src attribute directly inside of it.

To test, we will be using a sound effect audio from <http://www.w3schools.com/>:

- <http://www.w3schools.com/html5/horse.ogg> for Firefox, Google Chrome, and Opera
- <http://www.w3schools.com/html5/horse.mp3> for Internet Explorer, Google Chrome, Safari, and Safari iOS

We will create a `podcast.html` file and import the same libraries as `trailer.html`.

Custom controllers

Our media player is generic enough to use the same HTML structure for audio. We only need to replace the `video` tag with the `audio` tag, assign the `media` class to `audio` tag, and remove the full screen button:

```
<div class="media-container no-fullscreen">
  <div>
    <div class="media-area">
      <audio class="media">
        <source src="http://www.w3schools.com/html5/horse.ogg"
type="audio/ogg" />

        <source src="http://www.w3schools.com/html5/horse.mp3"
type="audio/mp3" />

        <p>Audio not supported.</p>
      </audio>
    </div>
    <div class="controls">
      <div class="play-button"></div>
      <div class="seek"></div>
      <div class="volume-container">
        <div class="volume-slider-container">
          <div class="volume-slider"></div>
        </div>
        <div class="volume-button"></div>
      </div>
      <div class="timer">00:00</div>
    </div>
  </div>
</div>
```

Styling

One last adjustment is related to the `audio` tag. Some browsers have the `height` attribute defined by default, so we reset it to 0:

```
audio.media{
  height:0;
}
```

How I learned to stop worrying and love Flash

The awful truth is that HTML5 video and media capabilities are a new technology and the browser war makes it even more difficult to adopt these solutions as a standard for media playback. While Flash requires the installation of a plugin, it is a reliable technology to reproduce media and stream it across multiple browsers.

While Flash support decreases on mobile devices and video and audio specifications improve, certainly there will be a future with no Flash media, but for now Flash is, at worst, a fallback solution for cross-browser compatibility.

Huge media delivery products like YouTube still rely on Flash as the primary technology. You can decide to use HTML5 as your primary technology and fall back to Flash if the `video` and `audio` tags are not supported or vice versa, but the choice should be made based on your application's requirements.

Summary

The HTML5 `video` and `audio` tags are simple and elegant ways to support media in your enterprise application, but differences between implementations across browsers should be taken into account when it is necessary to use them as a solution. For now, the best solution is to use both solutions and define a primary solution and a fallback.

The next chapter will focus on the use of another exciting feature of HTML5: `canvas`. We will use the `canvas` tag as a tool to visualize analytics related to movie reviews.

7

The App: Showing Ratings via Canvas

Until now, we have seen ways to lay out and draw elements in our enterprise application using CSS and images. If we need to create complex visualizations and/or animations based on dynamic data, the use of DOM objects becomes intricate and its manipulation slow. For that reason, the `canvas` tag was introduced in the HTML5 specification. The `canvas` tag defines a rectangular area where we can draw anything using its JavaScript API. This chapter introduces the `canvas` tag for data visualizations and simple animations.

In this chapter, we will cover:

- Charting
- Preparing our code
- Everything depends on the context (2D and 3D contexts)

Charting

Our current implementation of MovieNow uses a subset of the data provided by the `movielistings.php` web service. Some of the data not used includes ratings from MetaCritic, EditorBoost, and general user ratings (`avgMetaCriticRating`, `editorBoost`, and `avgUserRating` respectively). MovieNow users would love to see that information in the form of bar charts. For that, we will use `canvas`.



Although it is possible to render this information using DOM objects it can be slower and more restrictive.

Preparing our code

We need to add a new interaction to show the ratings chart with a click. Let us remove our current on-click interaction:

```
$("#movies-near-me li").click(function() {  
    $(this).toggleClass("open")  
});
```

The new interaction will include two buttons: one to show the movie description and the other to show the ratings chart. Inside the `img` folder, you will find an `options.png` image sprite. It has icons for both information and charts.



Using the `details-button` and `charting-button` classes, let us add some styles to `styles.css`. Each button will be 45 px x 45 px, using absolute positioning to place it in the bottom-right corner:

```
.details-button, .charting-button {  
    width:45px;  
    height:45px;  
    cursor:pointer;  
    position:absolute;  
    bottom:10px;  
    right:0;  
    background:none;  
    border:none;  
    background-image:url(../img/options.png)  
}
```

Place the details button to the left-hand side of the charting button:

```
.details-button {  
    right:45px;  
}
```

Set the correct image for the charting button:

```
.charting-button {  
    background-position:-45px 0;  
}
```

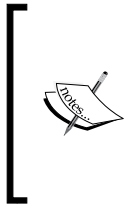
In the `displayShowtimes` function of `movienow.geolocation.js`, we will need to change the HTML structure to add our new buttons:

```
<input type="button" class="charting-button" />
<input type="button" class="details-button" />
```

We will also place our chart container with its canvas. At this point, we are going to use the HTML5 custom data attribute - `data-feed`, to store information about ratings in each canvas:

```
<section class="charting">
  <h3 itemprop="name">movie.title</h3>

  <p><canvas data-feed= "MetaCritic:movie.avgMetaCriticRating,EditorBo
ost:movie.editorBoost,User Rating:movie.avgUserRating"></canvas></p>
</section>
```



HTML5 custom data attributes allow the embedding of metadata on HTML elements. Your attribute name must have a prefix, `data-`, followed at least by one character string; in our case we use the word `feed` so our attribute name is `data-feed`. It does not allow uppercase and the value is a string.

Putting it together, we get:

```
for (var item in movies.items) {
  movie = movies.items[item];
  var movieDesc=(movie.synopsis &&movie.synopsis.length>200)?movie.
synopsis.substr(0,200)+"...": movie.synopsis;
  var movieHTML='<li itemscope itemType="http://schema.org/Movie">';
  movieHTML+='';
  movieHTML+='<section class="main-info">';
  movieHTML+='<input type="button" class="charting-button" />';
  movieHTML+='<input type="button" class="details-button" />';

  movieHTML+='<h3 itemprop="name">'+movie.title+'</h3>';

  movieHTML+='<p class="details genre" itemprop="genre">'+Array(movie.
genre).join(', ')+</p>';

  movieHTML+='<p class="details">'+movie.mpaarating+'</p>';
  movieHTML+='<p class="theater">'+movie.theater.title+" "+movie.
theater.address+'</p>';
```

```
    movieHTML+= '<p class="actors">'+Array(movie.selectedStar).join(',
    ')+'</p>';
    movieHTML+= '</section>';

    movieHTML+= '<section class="description">';

    movieHTML+= '<h3 itemprop="name">'+movie.title+'</h3>';

    movieHTML+= '<p>'+movieDesc+'</p>';
    movieHTML+= '</section>';
    movieHTML+= '<section class="charting">';
    movieHTML+= '<h3 itemprop="name">'+movie.title+'</h3>';
    movieHTML+= '<p><canvas data-feed= "MetaCritic:'+movie.avgMetaCri
    ticRating+", EditorBoost:"+movie.editorBoost+", User Rating:"+movie.
    avgUserRating+"></canvas></p>';
    movieHTML+= '</section>';
    movieHTML+= '</li>';
    html+=movieHTML;
}
```

Add some spacing before the charting elements in `styles.css`:

```
.charting canvas{
    margin-top:10px;
}
```

To hide and show the charting and description areas, we are going to use the `desc` class. If the `desc` class is applied to the `li` tag, we will hide charting and show description. Otherwise, we hide description and show charting:

```
#movies-near-me li.desc section.description, #movies-near-me li
section.charting{
    display:block;
}
#movies-near-me li section.description, #movies-near-me li.desc
section.charting{
    display:none;
}
```


Back to `movienow.geolocation.js`, we define methods to show chart (`showCharts`) and show details (`showDetails`). For `showCharts`, we will use jQuery's chaining capability. `$(event.target)` if the button is clicked, so we go two levels up using `parent()`; remove the `desc` class from the current object (`li`), add the open class, and find the first canvas element:

```
$(event.target)
    .parent()
```

```

.parent()
.removeClass("desc")
.addClass("open")
.find("canvas")[0];

```

 jQuery allows for the concatenation of method calls applying each method to the result of the previous one. This improves performance but sometimes goes against readability.

We are going to create a function called `charts` to draw each canvas, this function will take the canvas object as a parameter. Our final method should look like the following:

```

this.showCharts = function(event) {
  that.charts(
    $(event.target)
      .parent()
      .parent()
      .removeClass("desc")
      .addClass("open")
      .find("canvas")[0]
  );
};

```

Apply the same train of thought to show details:

```

this.showDetails = function(event) {

  $(event.target)
    .parent()
    .parent()
    .addClass("desc")
    .addClass("open");
};

```

Add event handlers for a click to open and close:

```

$("#movies-near-me li .details-button").click(that.showDetails);
$("#movies-near-me li .description, #movies-near-me li .charting").
click(function(){
  $(this)
    .parent()
    .removeClass("open")
});
$("#movies-near-me li .charting-button").click(that.showCharts);

```


Now we create `movienow.charts.js` and add the `charts` method:

```
var movienow = movienow || {};  
movienow.charts = (function(){  
  var that = this;  
  this.charts = function(canvas){  
    that.drawBarChart(canvas);  
  };  
  this.drawBarChart = function(canvas) {  
  }  
}) ();
```

Notice that the `charts` method calls the `drawBarChart` method. We use this construct to change the drawing method later.

Remember to include `movienow.charts.js` in `index.html`:

```
<script src="js/ios-orientationchange-fix.js"></script>  
<script src="js/jquery-1.8.0.min.js"></script>  
<script src="js/jquery.xdomainajax.js"></script>  
  
<script src="js/movienow.charts.js"></script>  
  
<script src="js/movienow.geolocation.js"></script>  
  
<script src="js/movienow.js"></script>
```

Everything depends on the context

Canvas provides APIs to draw in two or three dimensions, where supported. Canvas 2D has wider support than Canvas 3D; the latter is generally not supported on any mobile browser.

You can define what API to use by getting the canvas context. Let us suppose that `chart` is our canvas object. If you want to draw in two dimensions, you can use:

```
var context=chart.getContext("2D");
```

Then, you can use the 2D API to draw, for example, a red square defining its color:

```
context.fillStyle="#FF0000";
```

Draw the shape as follows:

```
context.fillRect(0,0,20,20);
```

For the 3D API, the use is far more complicated. First, it is still not fully supported as some browsers recognize `webgl`:

```
var context=chart.getContext("webgl");
```

While others use `experimental-webgl`:

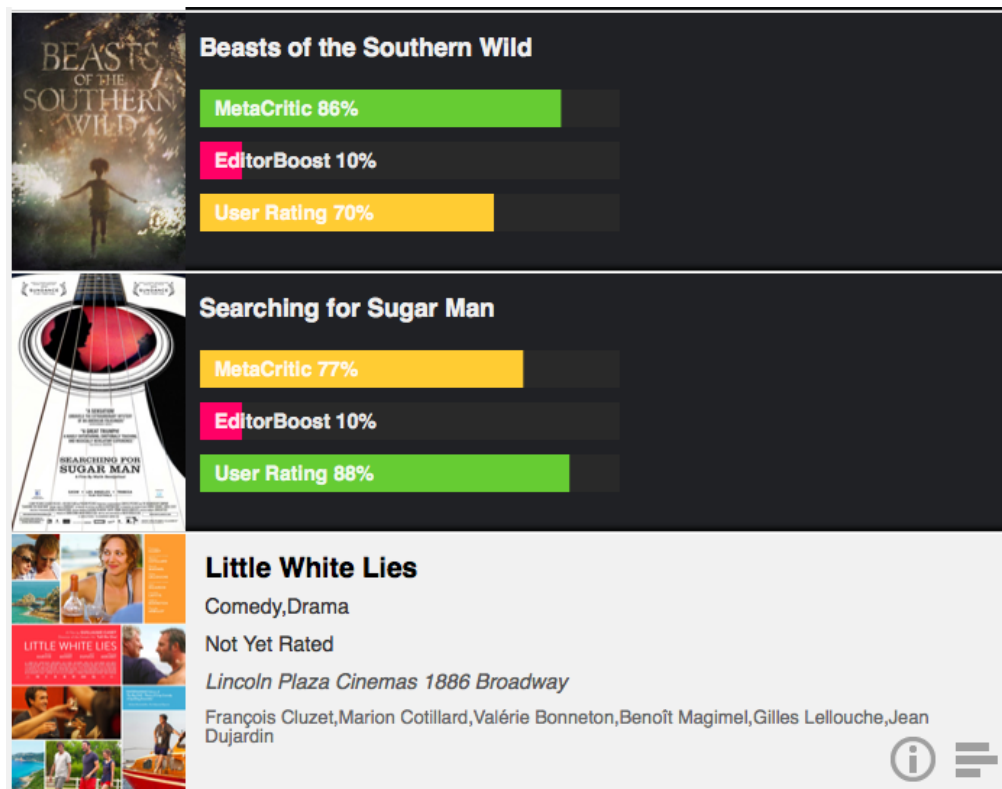
```
var context=chart.getContext("experimental-webgl");
```

This is because the `webgl` specification is still in development. Use of WebGL requires knowledge of computer graphics and concepts like cameras, lights, textures, materials, mapping, and so on.

2D context

We are going to create a horizontal bar chart to show MetaCritic, EditorBoost, and user ratings. The idea is to use green, yellow, and red to indicate how high or low the rating is.

In the following image, we can see our chart design:



Based on the `movienow.charts.js` structure we have established, let us write the `drawBarChart` method. As we do not want to redraw our canvas every time we call this method, we are going to use the `painting` class as a flag to determine if the canvas is already drawn.

We can then save `canvas` as a jQuery object:

```
var myCanvas=$(canvas);
```

Then, we can verify if it has the `painting` class:

```
if(!myCanvas.hasClass("painting")){  
  //DRAW HERE  
  myCanvas.addClass("painting");  
}
```

Inside our conditional, we split `data-feed` that contains rating information in order to iterate over it, building a bar for each rating category:

```
var values=myCanvas.attr("data-feed").split(",");
```

We then get the 2D context:

```
var context=canvas.getContext("2d");
```


An overview of the Canvas 2D Drawing API

Let us go over the most useful methods of Canvas in a 2D context:

Styles

The methods used for setting styles in Canvas 2D API are explained as follows:

- `context.strokeStyle(value)`: This receives a string containing a CSS color of the stroke; if no parameter is passed, it returns the current style for stroking shapes.
- `context.fillStyle(value)`: It receives a string with the CSS color for filling shapes; if no parameter is passed, it returns the current fill style.

 `strokeStyle` and `fillStyle` can receive `CanvasGradient` or `CanvasPattern` as a parameter, allowing you to draw gradients and patterns. For more information about how to create gradients and patterns, you can check out the canvas 2D API specification: <http://dev.w3.org/2006/canvas-api/canvas-2d-api.html>.

- `context.lineWidth(value)`: It defines the width of the lines using pixels. It only accepts positive values; if no value is passed, it acts like a getter and returns the current line width.
- `context.lineCap(value)`: It sets the style of the end (or cap) of lines. Possible values are `butt`, `round`, and `square`. If no value is passed, it returns the current line cap.
- `context.lineJoin(value)`: It sets the style of the connection of lines. Possible values are `bevel`, `round`, and `miter`. If no value is passed, it returns the current line joint style.

Font styles

The methods used for setting font styles in canvas 2D API are explained as follows:

- `context.font(value)`: It defines the style of the fonts using a string with CSS syntax. If no value is passed, it returns the current font style.
- `context.fillText(text, x, y[, maxWidth])`: It draws text. It receives a text string with the information to draw `x` and `y` coordinates in pixels and `maxWidth` that defines the maximum size in pixels for the container's width, which is optional.
- `context.strokeText(text, x, y[, maxWidth])`: It behaves like `fillText` but draws only the stroke of the text.

Drawing simple shapes

The methods used for drawing simple shapes in the Canvas 2D API are as follows:

- `context.clearRect(x, y, w, h)`: It defines a rectangle with coordinates `x` and `y`, width `w`, and height `h`, and clears all pixels inside the area defined. Values are in pixels.
- `context.fillRect(x, y, w, h)`: It draws a rectangle with coordinates `x` and `y`, width `w` and height `h` using the predefined `fillStyle`.
- `context.strokeRect(x, y, w, h)`: It draws the stroke of a rectangle with coordinates `x` and `y`, width `w`, and height `h` using the predefined `strokeStyle`.

Drawing complex shapes

To create more complex shapes, Canvas 2D API allows you to define paths and subpaths. A path is a collection of subpaths while a subpath is a list of points connected by lines or curves.

The current project does not require complex shapes, but it is good to know the methods that allow you to draw curves and lines making it possible to draw any figure.

We need to be aware that our context always contains a current path, and it is not possible to have more than one.

- `context.beginPath()`: It resets the current path
- `context.closePath()`: It closes the current path and creates a new one with a first point that uses the same coordinates as the last subpath point
- `context.moveTo(x, y)`: It creates a new subpath with coordinates `x` and `y`
- `context.lineTo(x, y)`: It creates a line between the current point and a new point with coordinates `x` and `y` adding the latest to the current subpath
- `context.quadraticCurveTo(cpx, cpy, x, y)`: It creates a quadratic curve between the current point and a new point with coordinates `x` and `y` using a control point, defined with coordinates `cpx` on the `x` axis, and `cpy` on the `y` axis
- `context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)`: It creates a Bezier curve between the current point and a new point with coordinates `x` and `y` using two control points `cp1`(`cp1x`, `cp1y`) and `cp2` (`cp2x`, `cp2y`)
- `context.arcTo(x1, y1, x2, y2, radius)`: It connects the current point with a new point with coordinates `x1` and `y1`, then creates a new point with coordinates `x2` and `y2` joined with the previous one by an arc with radius defined by the parameter `radius`
- `context.rect(x, y, w, h)`: It adds a rectangle with coordinates `x` and `y`, width `w`, and height `h` to the list of subpaths
- `context.fill()`: It applies the current fill style to fill the subpaths
- `context.stroke()`: It applies the current stroke style to create stroke lines for the subpaths
- `context.isPointInPath(x, y)`: It returns `true` if the point defined by coordinates `x` and `y` is in the current path and `false` otherwise

There are other methods that can be useful for more complicated drawings and animations; you can look at the canvas specification here: <http://dev.w3.org/2006/canvas-api/canvas-2d-api.html>.

Drawing charts

First, let us define the text style for that context and a variable to count the current bar index (in case not all movies have the same number of rating categories):

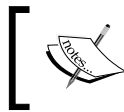
```
context.font = "bold 14px sans-serif";  
var index=0;
```

Iterate over rating categories and get the current value:

```
for(var i=0; i<values.length; i++){
    var info=values[i].split(":");
    var val=info[1];
}
```

We draw (only if the value is bigger than 0) first a gray bar (#292929) that defines a width of 290 px and a height of 26 px. Using the syntax `fillRect(x,y,width,height)`, notice that we use 36 to give 10 pixels of separation between bars:

```
var pos=index*36;
context.fillStyle="#292929";
context.fillRect(0,pos,290,26);
```



The canvas origin is positioned in the upper-left corner of the canvas DOM object. Positive values go below origin and to the right-hand side.

We can then draw our color bar. For that, we define a `getChartColor` method that returns different colors depending on the rating value using green for higher ones, yellow for mediums, and red for lowest:

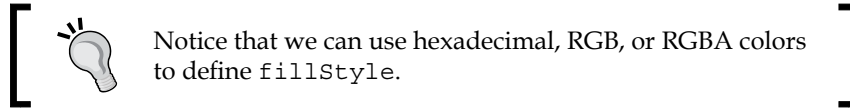
```
this.getChartColor = function(val){
    var result="";
    if(val<40){
        result="#FF0066";
    }else{
        if(val<80){
            result="#FFCC33";
        }else{
            result="#66CC33";
        }
    }
    return result;
};
```

Using our `getChartColor` method and current rating value `val`, we can set the shape:

```
context.fillStyle=that.getChartColor(val);
context.fillRect(0,pos,val*2.9,26);
```

To draw the rating category title, we change fill style to a transparent white, and then we use `fillText` with syntax `fillText(text, x,y)`:

```
context.fillStyle = "rgba(255, 255, 255, .9)";
context.fillText(info[0]+" "+val+"%", 10, pos+18);
index++;
```



Finally, we write a validation such that if there is no data we show a message that says **No Data Available**:

```
if(index==0){
context.fillStyle = "#FFFFFF";
context.fillText("No Data Available", 40, 50);
}
```

Wrap it all together:

```
this.drawBarChart = function(canvas) {
var myCanvas=$(canvas);
if(!myCanvas.hasClass("painted")){
var values=myCanvas.attr("data-feed").split(",");
var context=canvas.getContext("2d");
context.font = "bold 14px sans-serif";
var index=0;
for(var i=0; i<values.length; i++){
var info=values[i].split(":");
var val=info[1];
if(val>0){
var pos=index*36;
context.fillStyle="#292929";
context.fillRect(0,pos,290,26);
context.fillStyle=that.getChartColor(val);
context.fillRect(0,pos,val*2.9,26);
context.fillStyle = "rgba(255, 255, 255, .9)";
context.fillText(info[0]+" "+val+"%", 10, pos+18);
index++;
}
}
if(index==0){
context.fillStyle = "#FFFFFF";
context.fillText("No Data Available", 40, 50);
}
```


```

    }
    myCanvas.addClass("painted");
  }
};

```

In this example, we do not need to clear our canvas area because we are not animating or changing information after our first draw. However, if we do require redrawing, we can use:


```
context.clearRect(0, 0, canvas.width, canvas.height);
```

 Canvas 2D context provides a procedural approach to drawing – creating bitmap images. If we need to use vectorial graphics instead of bitmaps it's possible to use Scalable Vector Graphics (SVG), which provides a declarative approach using XML. To know more about SVG you can research here <http://www.w3.org/Graphics/SVG/>.

The Canvas 2D API is supported in all modern browsers including Internet Explorer since Version 9.0. As we have shown in *Chapter 2, HTML Starter Kit: Useful Tools*, it is still possible to support previous versions of Internet Explorer using ExplorerCanvas <http://code.google.com/p/explorercanvas/downloads/list>.

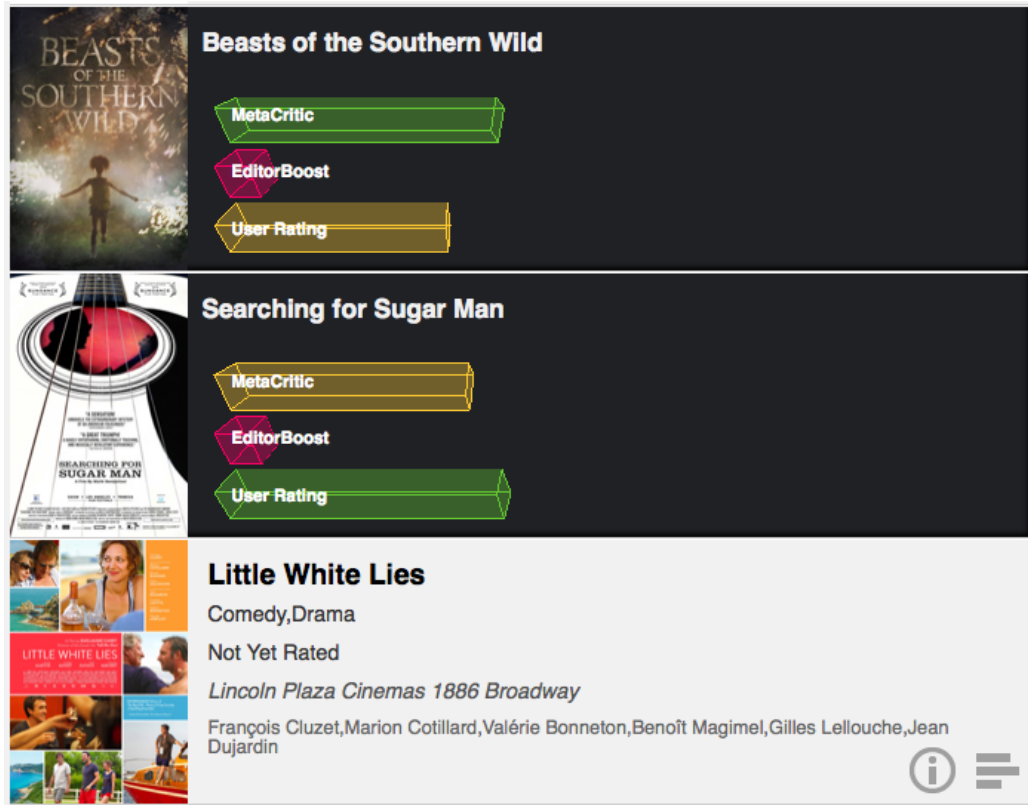
3D context – WebGL and experimental WebGL

Now that we saw a two-dimensional implementation of our chart solution, let us try to create a three-dimensional version and add some animations to make things more interesting.

 When we think of user experience, a general rule is that less is more. That means that keeping things simple should make our application more usable. In this case, we are going to add animations that are not needed for the sake of learning how to do it.

In our case we are going to draw bar charts using WebGL and rating categories titles using DOM objects.

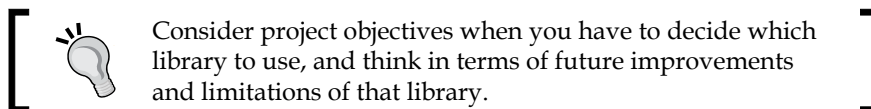
The final implementation should look like the following:



Entering a tridimensional world

The WebGL specification is based on the OpenGL for Embedded Systems 2.0 (or OpenGL ES) specification. Unless you are familiar with OpenGL and you need to work with low level manipulations, it is suggested that you use a library that abstracts the use of it. Among the advantages are more readable code, less development time, and better extensibility.

In our case, we selected `Three.js`, a JavaScript library that simplifies the use of WebGL using common metaphors used in other tridimensional libraries.



Three.js

Three.js is a JavaScript library that abstracts 3D manipulation allowing us to use simple metaphors like scenes, cameras, objects, and so on. You can download Three.js from <https://github.com/mrdoob/three.js/> and read its documentation at <http://mrdoob.github.com/three.js/docs/50/>.

Let us go over some basic concepts:

Scene

The scene is the virtual environment where we can insert objects. Every object must be in a scene in order to visualize it.

You can create scenes using `scene = new THREE.Scene()` and add an object to it using `scene.add(object)`.

Camera

A camera indicates which section of our scene to visualize. Think of it as a movie; if we want to record a specific place, we need to point our camera at it. Three.js provides an abstract Camera class for cameras, two basic cameras, and two extra implementations of cameras.

`OrthographicCamera` defines an orthographic projection defined by a cube formed for the constructor parameters:

```
OrthographicCamera(left, right, top, bottom, near, far)
```

- `left` - defines left plane using a float that indicates position
- `right` - defines right plane using a float that indicates position
- `top` - defines top plane using a float that indicates position
- `bottom` - defines bottom plane using a float that indicates position
- `near` - defines the plane nearest to the camera or near plane using a float that indicates position
- `far` - defines the plane farthest to the camera or far plane using a float that indicates position

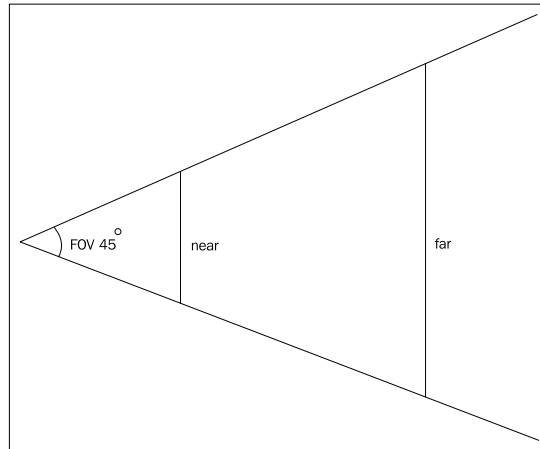
`PerspectiveCamera` defines a perspective projection using field of view, aspect ratio, near and far values:

```
PerspectiveCamera(fov, aspect, near, far)
```

- `fov` - defines the angle that indicates the field of view represented by a float
- `aspect` - defines the camera aspect ratio defined by a float

- near – as in `OrtographicCamera` defines near plane using a float
- far – as in `OrtographicCamera` defines far plane using a float

A visual representation of a perspective camera is shown in the following figure:



Material

Materials define a set of properties that describe the appearance of objects.

Texture

Textures define appearance of objects using images (or procedural patterns).

Mesh

Meshes are part of the list of objects that can be added to the scene. You can assign geometries and materials to a mesh.

Geometry

Geometry is a representation of an object that can be assigned to a mesh. In our case, we are going to use `CubeGeometry` to define our bars.

To start, we will download and include `three.js` in our `index.html` file:

```
<script src="js/ios-orientationchange-fix.js"></script>
<script src="js/jquery-1.8.0.min.js"></script>
<script src="js/jquery.xdomainajax.js"></script>
<script src="js/three.js"></script>
<script src="js/movienow.charts.js"></script>
<script src="js/movienow.geolocation.js"></script>
<script src="js/movienow.js"></script>
```

We then add a new parameter to our `charts` class to specify which rendering we would like to use:

```
this.charts = function(canvas, type){
  switch(type){
    case "3DChart":
      that.draw3DChart(canvas);
      break;
    case "barChart":
    default:
      that.drawBarChart(canvas);
      break;
  }
};
```

We define as default `3DChart` in the `showCharts` method:

```
this.showCharts = function(event) {
  that.charts($(event.target).parent().parent().removeClass("desc").
  addClass("open").find("canvas")[0], "3DChart");
};
```

Now let us write our 3D drawing method `draw3DChart`. As our previous two-dimensional drawing method, this one takes our `canvas` as a parameter:

```
this.draw3DChart = function(canvas) {
}
```

As before, we verify if `Canvas` has the `painting` class to avoid initializing it again. We can then verify `WebGL` support and, if it is not supported, we render our 2D charts:

```
this.draw3DChart = function(canvas) {
  var myCanvas=$(canvas);
  var myCanvasParent=myCanvas.parent();
  if(!myCanvas.hasClass("painting")){
    var webGlsupport=false;
    try {
      /** VERIFICATION OF WebGL SUPPORT **/
      webGlsupport = !!window.WebGLRenderingContext && !!document.
      createElement('canvas').getContext('experimental-webgl');
    }catch(e){}
    if (webGlsupport){
      //DRAW 3D HERE
    }else{
      /** IF NOT WebGL SUPPORT RENDERS CHART IN 2D **/
    }
  }
};
```

```
        that.drawBarChart(canvas);
    }
}
};
```

Inside `webGLSupport`, we get our ratings data, canvas dimensions, and a variable to store time for animation:

```
var data=myCanvas.attr("data-feed");
var values=data.split(",");
var w = myCanvas.width();
var h = myCanvas.height();
var lastTime = 0;
```

Now we define a `three.js` renderer using WebGL, which contains our new canvas, and set its dimensions:

```
var renderer = new THREE.WebGLRenderer();
renderer.setSize(w, h);
```

We assign the data attribute and the painted class to the renderer DOM element (canvas) and replace our old canvas with this one.

```
var newCanvas=$(renderer.domElement);
newCanvas.attr("data-feed",data);
myCanvas.addClass("painted");
myCanvas.replaceWith(newCanvas);
```

We implement our camera using field of vision (FOV) 45 degree, aspect ratio based on canvas dimensions, near of 1 and far of 1000, and we position our camera to 700 on the z axis.

```
var camera = new THREE.PerspectiveCamera(45, w/h, 1, 1000);
camera.position.z = 700;
```

Then we define our scene, a `bars` array to store meshes to be rendered later, an `index` as in our 2D example and a `labels` string to store the DOM object that will show the titles:

```
var scene = new THREE.Scene();
var bars=[];
var index=0;
var labels="<div class='chart-labels'>";
```

Iterate over rating categories as our previous example. We get a color for our current value using `getChartColor` and replace `#` with `0x` because of the color notation used:

```
var mainColor=that.getChartColor(val).replace("#", "0x");
```

As we will have six faces for each bar, we will have six different materials. Each one will have its own color, so we define arrays for `colors` and `materials`. We fill our title information in the `labels` string too:

```
var colors = [mainColor, mainColor, mainColor, mainColor, mainColor,
mainColor];
var materials = [];
labels+="<div>"+info[0]+"</div>";
for (var n = 0; n < 6; n++) {

    materials.push([
        new THREE.MeshLambertMaterial({
            color: colors[n],
            opacity:0.6,
            transparent: true,
            shading: THREE.FlatShading,
            vertexColors: THREE.VertexColors
        }),
        new THREE.MeshBasicMaterial({
            color: colors[n],
            shading: THREE.FlatShading,
            wireframe: true,
            transparent: true
        })
    ]);
}
```

Notice that we can have more than one material assigned. In this case, we use the following to define a transparent fill for our solids:

```
THREE.MeshLambertMaterial({
    color: colors[n],
    opacity:0.6,
    transparent:
    true,
    shading: THREE.FlatShading,
    vertexColors: THREE.VertexColors
})
```

The following draws the edges:

```
new THREE.MeshBasicMaterial({
  color: colors[n],
  shading: THREE.FlatShading,
  wireframe: true,
  transparent: true
})
```

Each bar is defined as a mesh with geometry. Use the following `CubeGeometry` syntax:

```
CubeGeometry(width, height, depth, segmentsWidth, segmentsHeight,
segmentsDepth)
```

This creates a new `CubeGeometry` object:

```
var bar = new THREE.Mesh(new THREE.CubeGeometry(myWidth, 90, 90, 1, 1,
1, materials), new THREE.MeshFaceMaterial());
```

Animating our geometries

We are going to animate the growing of the bars, so we will scale them on the x axis.

```
bar.scale.x=.01;
```

Meshes have their reference points at their centers, so positioning the mesh and setting the `overdraw` to manage transparent geometries, we have:

```
bar.position.y=200-(index*140);
bar.position.x=-500+(myWidth/2)*bar.scale.x;
bar.overdraw = true;
```

We add the bar to our scene and to our array with the final width that we should have at the end of our animation:

```
scene.add(bar);
bars.push({object:bar, width:myWidth});
index++;
```

So our iteration should look as follows:

```
for(var i=0; i<values.length; i++){
  var info=values[i].split(":");
  var val=info[1];
  if(val>0){
    var mainColor=that.getChartColor(val).replace("#", "0x");
    var colors = [mainColor, mainColor, mainColor, mainColor,
mainColor, mainColor];
    var materials = [];
```

```

labels+="

"+info[0]+"</div>";
for (var n = 0; n < 6; n++) {
    materials.push([
        new THREE.MeshLambertMaterial({
            color: colors[n],
            opacity:0.6,
            transparent: true,
            shading: THREE.FlatShading,
            vertexColors: THREE.VertexColors
        }),
        new THREE.MeshBasicMaterial({
            color: colors[n],
            shading: THREE.FlatShading,
            wireframe: true,
            transparent: true
        })
    ]);
    }
    var myWidth=val*8;
    var bar = new THREE.Mesh(new THREE.CubeGeometry(myWidth, 90, 90,
1, 1, 1, materials), new THREE.MeshFaceMaterial());
    bar.scale.x=.01;
    bar.position.y=200-(index*140);
    bar.position.x=-500+(myWidth/2)*bar.scale.x;
    bar.overdraw = true;
    scene.add(bar);
    bars.push({object:bar, width:myWidth});
    index++;
}
}


```

At this point, we have not rendered anything. We can remedy that by setting the labels string and appending it:

```

labels+ "</div>";
myCanvasParent.append(labels);

```

We set a three structure that we will use for our rendering, and then we call our render method - animate3DChart:

```

var three = {
    renderer: renderer,
    camera: camera,
    scene: scene,
    bars: bars
};
that.animate3DChart(lastTime, three);

```


Our draw3DChart method looks as follows:

```
this.draw3DChart = function(canvas) {
  var myCanvas=$(canvas);
  var myCanvasParent=myCanvas.parent();
  if(!myCanvas.hasClass("painted")){
    var webGlsupport=false;
    try {
      /*** VERIFICATION OF WEBGL SUPPORT ***/
      webGlsupport = !!window.WebGLRenderingContext && !!document.
createElement('canvas').getContext('experimental-webgl');
    }catch(e){}
    if (webGlsupport){
      var data=myCanvas.attr("data-feed");
      var values=data.split(",");
      var w = myCanvas.width();
      var h = myCanvas.height();
      var lastTime = 0;
      var renderer = new THREE.WebGLRenderer();
      renderer.setSize(w, h);
      var newCanvas=$(renderer.domElement);
      newCanvas.attr("data-feed",data);
      myCanvas.addClass("painted");
      /*** REPLACES ORIGINAL CANVAS WITH THREE.JS CANVAS ***/
      myCanvas.replaceWith(newCanvas);
      /*** CAMERA DEFINITION ***/
      var camera = new THREE.PerspectiveCamera(45, w/h, 1, 1000);
      camera.position.z = 700;
      /*** SCENE DEFINITION ***/
      var scene = new THREE.Scene();
      var bars=[];
      var index=0;
      var labels="<div class='chart-labels'>";
      for(var i=0; i<values.length; i++){
        var info=values[i].split(":");
        var val=info[1];
        if(val>0){
          var mainColor=that.getChartColor(val).replace("#", "0x");
          var colors = [mainColor, mainColor, mainColor, mainColor,
mainColor, mainColor];
          var materials = [];
          labels+="<div>"+info[0]+"</div>";
          for (var n = 0; n < 6; n++) {
            materials.push([
```

```

        new THREE.MeshLambertMaterial({
            color: colors[n],
            opacity:0.6,
            transparent: true,
            shading: THREE.FlatShading,
            vertexColors: THREE.VertexColors
        }
    ),
    new THREE.MeshBasicMaterial({
        color: colors[n],
        shading: THREE.FlatShading,
        wireframe: true,
        transparent: true
    })
    ]);
}
var myWidth=val*8;
var bar = new THREE.Mesh(new THREE.CubeGeometry(myWidth, 90,
90, 1, 1, 1, materials), new THREE.MeshFaceMaterial());

bar.scale.x=.01;
bar.position.y=200-(index*140);
bar.position.x=-500+(myWidth/2)*bar.scale.x;
bar.overdraw = true;
scene.add(bar);
bars.push({object:bar, width:myWidth});
index++;
}
}
labels+"</div>";
myCanvasParent.append(labels);
/** SAVE INFORMATION REQUIRED TO RENDER SCENE **/
var three = {
    renderer: renderer,
    camera: camera,
    scene: scene,
    bars: bars
};
that.animate3DChart(lastTime, three);
}else{
/** IF NOT WEBGL SUPPORT RENDERS CHART IN 2D **/
that.drawBarChart(canvas);
}
}
};

```

Finishing up

We create a `window.requestAnimationFrame` method to abstract the definition of our timeout for animation. Notice that we use `1000/60`. This indicates 60 frames per second (FPS):

```
window.requestAnimationFrame = (function(callback) {
  return window.requestAnimationFrame ||
  window.webkitRequestAnimationFrame ||
  window.mozRequestAnimationFrame ||
  window.oRequestAnimationFrame ||
  window.msRequestAnimationFrame ||
  function(callback) {
    /* Using 60FPS */
    window.setTimeout(callback, 1000 / 60);
  };
}) ();
```

For the `animate3DChart` method, we simply define a variable to stop our animation (`isReady`), and scale and position each bar stopping when we reach 100 percent scale (in this case 1):

```
this.animate3DChart = function(lastTime, three) {
  var isReady=false;
  for(var i=0; i<three.bars.length; i++){
    if(three.bars[i].object.scale.x<1){
      three.bars[i].object.scale.x+=.03;
      three.bars[i].object.position.x=-500+(three.bars[i].
width/2)*three.bars[i].object.scale.x;
    }
    isReady=(three.bars[i].object.scale.x>=1);
  }
  lastTime = time;
  /** SCENE RENDER USING THREE.JS ***/
  three.renderer.render(three.scene, three.camera);
  if(!isReady){
    requestAnimationFrame(function(){
      that.animate3DChart(lastTime, three);
    });
  }
}
```

If we want to rotate each bar with no stop, we can define some values to control the animation:

```
var angularSpeed = 1.2;
var date = new Date();
```

```

var time = date.getTime();
var timeDiff = time - lastTime;
var angleChange = angularSpeed * timeDiff * 2 * Math.PI / 1000;

```

We can then substitute `isReady=(three.bars[i].object.scale.x>=1)` with `three.bars[i].object.rotation.x += angleChange`.

To modify rotation on the x axis, we can add the following:

```

this.animate3DChart = function(lastTime, three){
  var angularSpeed = 1.2;
  var date = new Date();
  var time = date.getTime();
  var timeDiff = time - lastTime;
  var angleChange = angularSpeed * timeDiff * 2 * Math.PI / 1000;
  var isReady=false;
  for(var i=0; i<three.bars.length; i++){
    if(three.bars[i].object.scale.x<1){
      three.bars[i].object.scale.x+=.03;
      three.bars[i].object.position.x=-500+(three.bars[i].
width/2)*three.bars[i].object.scale.x;
    }
    //isReady=(three.bars[i].object.scale.x>=1);
    three.bars[i].object.rotation.x += angleChange;
  }
  lastTime = time;
  /** SCENE RENDER USING THREE.JS ***/
  three.renderer.render(three.scene, three.camera);
  if(!isReady){
    requestAnimationFrame(function(){
      that.animate3DChart(lastTime, three);
    });
  }
}

```

The canvas WebGL API is not fully supported by all browsers. You can use the 3D API for Firefox 4.0+, Chrome, Opera, and Safari since Version 5.1 (on OSX or higher, but not Safari for iOS devices or Safari for Windows).



WebGL is disabled by default on Safari. To enable WebGL on Safari, click on the **Safari** menu and select **Preferences**, then click on the **Advanced** tab. At the bottom, check the **Show Develop menu in menu bar** checkbox. Open the **Develop** menu and then select **Enable WebGL**.

For Internet Explorer, you can enable WebGL support by installing the Chrome Frame plugin <http://www.google.com/chromeframe>. Google Chrome Frame replaces the rendering mechanism of Internet Explorer with Google Chrome's versions of the WebKit layout engine and V8 JavaScript engine.

Summary

Although the canvas specification is still in development, we can apply its APIs for innumerable use cases in our enterprise applications. Charts, scientific visualization, diagrams, and animated wizards are merely the tip of the iceberg. As developers, we should always give ample consideration to fallbacks or alternative solutions in the event something is not supported to ensure proper cross-platform compatibility.

The next chapter will cover drag-and-drop capabilities and event delegation using HTML5.

8

The App: Selection UI via Drag-and-Drop

Even though drag-and-drop functionality has existed since 1999 when Microsoft implemented it in Internet Explorer 5.0, HTML5 brings it to the fore in a more standard way. The specification defines a set of APIs, event handlers, and markup for adding drag-and-drop functionality (DnD) to your enterprise application. To demonstrate this, we will implement the ability to drag-and-drop movie showtimes into a staging area in our MovieNow enterprise application to indicate movies the user is interested in seeing.

The main topics covered in this chapter are:

- Adding showtimes
- Styling showtimes
- What a drag
- Drop it

Adding showtimes

We temporarily removed the showtimes from *Chapter 4, The App: Getting Movies via Geolocation* to make room for movie data, synopses, trailers, and ratings. We will now put showtimes back. To do this, we will modify the `displayShowtimes` method in `movienow.geolocation.js` by inserting a `div` tag for showtimes and looping through the array of showtimes contained in the `movie` object we previously constructed. Notice how we include a `data-movie` attribute containing the theater ID, the movie ID, and the showtime. We do this in order to save some data about the showtime for later use when we want to know to which movie and which theater the showtime belongs.

We will insert the following code snippet into the `displayShowtimes` method:

```
movieHTML+='\<div class="showtimes">';
if (typeof movie.showtime == 'string') movie.showtime = Array(movie.
showtime);
for(var i=0; i<movie.showtime.length; i++) {
if (movie.showtime[i]) movieHTML+='\<div class="showtime"
draggable="true" title="'+movie.title+' @ '+movie.theater.
title+' ('+movie.theater.address+')" data-movie= "'+movie.theater.
id+':'+movie.id+':'+movie.showtime[i]+'>'+that.formatTime(movie.
showtime[i])+'</div> ';
}
movieHTML+='\</div>';
```

The complete method should look like this:

```
this.displayShowtimes = function(movies) {
var movie = null;
var html = '<ul>';
for (var item in movies.items) {
movie = movies.items[item];
var movieDesc='';
if (movie.synopsis) movieDesc=(movie.synopsis.length>200)?movie.
synopsis.substr(0,200)+"...": movie.synopsis;
var movieHTML='\<li itemscope itemtype="http://schema.org/Movie">';
movieHTML+='\';
movieHTML+='\<section class="main-info">';
movieHTML+='\<input type="button" class="charting-button" />';
movieHTML+='\<input type="button" class="details-button" />';
movieHTML+='\<h3 itemprop="name">'+movie.title+'</h3>';
movieHTML+='\<p class="details genre"
itemprop="genre">'+Array(movie.genre).join(', ')+'</p>';
movieHTML+='\<p class="details">'+movie.mpaarating+'</p>';
movieHTML+='\<p class="theater">'+movie.theater.title+" "+movie.
theater.address+'</p>';
movieHTML+='\<p class="actors">'+Array(movie.selectedStar).join(',
')+</p>';
movieHTML+='\<div class="showtimes">';
if (typeof movie.showtime == 'string') movie.showtime =
Array(movie.showtime);
for(var i=0; i<movie.showtime.length; i++) {
if (movie.showtime[i]) movieHTML+='\<div class="showtime"
draggable="true" title="'+movie.title+' @ '+movie.theater.title+'
('+movie.theater.address+')" data-movie="'+movie.theater.id+':'+movie.
id+':'+movie.showtime[i]+'>'+movie.showtime[i]+'</div> ';
}
}
```


```

movieHTML+='/div>';
movieHTML+='/section>';
movieHTML+ '<section class="description">';
movieHTML+ '<h3 itemprop="name">'+movie.title+'</h3>';
movieHTML+ '<p>'+movieDesc+'</p>';
movieHTML+ '</section>';
movieHTML+ '<section class="charting">';
movieHTML+ '<h3 itemprop="name">'+movie.title+'</h3>';
movieHTML+ '<p><canvas data-feed= "MetaCritic:'+movie.avgMetaCr
iticRating+",EditorBoost:"+movie.editorBoost+",User Rating:"+movie.
avgUserRating+' "></canvas></p>';
movieHTML+ '</section>';
movieHTML+ '</li>';
html+=movieHTML;
}
html+= '</ul>';
$('#movies-near-me').html(html);
$("#movies-near-me li .details-button").click(that.showDetails);
$("#movies-near-me li .description, #movies-near-me li .charting").
click(function(){$(this).parent().removeClass("open")});
$("#movies-near-me li .charting-button").click(that.showCharts);
init();
};

```

If you preview this change in a web browser, you should see something akin to the following:

In Theaters Now



Beasts of the Southern Wild

Drama


PG-13

Lincoln Plaza Cinemas 1886 Broadway

Quvenzhané Wallis ,Dwight Henry

1240
1425

i ≡



Searching for Sugar Man

Documentary, Special Interest

PG-13

Lincoln Plaza Cinemas 1886 Broadway

1210
1355
1540

i ≡

Styling showtimes

Of course, the raw showtime data does not look quite the way we traditionally look at time. We are going to need to format the time to make sure our users understand the data appropriately.

To accomplish this, we will modify the following line in `displayShowtimes`:

```
if (movie.showtime[i]) movieHTML+= '<div class="showtime"
title="'+movie.title+' @ '+movie.theater.title+' ('+movie.theater.
address+')" data-movie= "'+movie.theater.id+':'+movie.id+':'+movie.
showtime[i]+'"'>'+movie.showtime[i]+'</div> ';
```

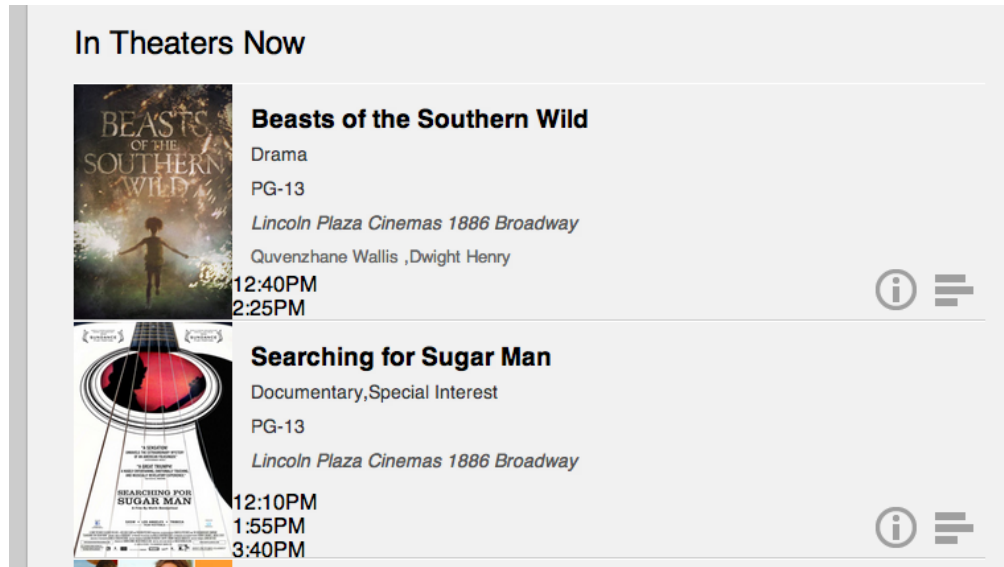
We will wrap the display of the showtime with a call to a `formatTime` method, which we will write momentarily. Change the previous line so that it looks similar to the following line:

```
if (movie.showtime[i]) movieHTML+= '<div class="showtime"
title="'+movie.title+' @ '+movie.theater.title+' ('+movie.theater.
address+')" data-movie= "'+movie.theater.id+':'+movie.id+':'+movie.
showtime[i]+'"'>'+that.formatTime(movie.showtime[i])+'</div> ';
```

We can then add the following method to format the time. This method takes the string passed into it, gets the first two characters for the hour, the adjacent two characters for the minute, and then interprets and modifies the hour data to change it from a 24-hour clock to a 12-hour clock.

```
this.formatTime = function(time) {
  var hh = time.substr(0,2);
  var mm = time.substr(2,2);
  var period = 'AM';
  hh = parseInt(hh, 10);
  if (hh >= 12) period = 'PM';
  if (hh > 12) hh -= 12;
  return hh+':'+mm+period;
};
```

The preview for this change should look like the following screenshot:

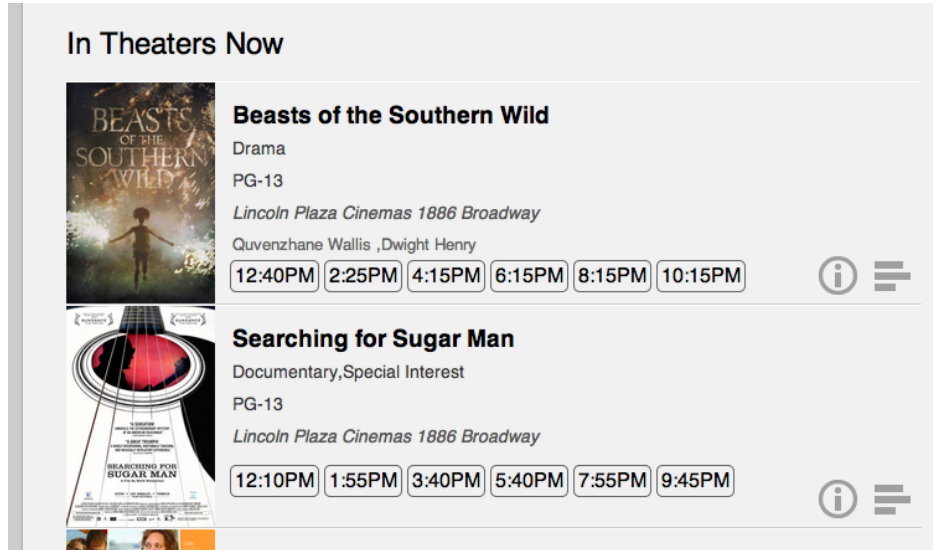


To make the showtimes a little more presentable, we add the following styles to `styles.css`:

```
.showtimes {
  float:left;
  margin-left:10px;
}
.showtimes .showtime {
  float:left;
  padding:3px;
  margin:0 2px;
  border:1px solid #666;
  -moz-border-radius:5px;
  border-radius:5px;
  cursor:move;
}
```

We are floating the showtimes to the left-hand side so that they are arranged horizontally, and adding a border around them to discern them from the other content. Lastly, we set the cursor to move so that when you hover the mouse over them, the mouse pointer changes to the move icon for your operating system to indicate that the showtime is a movable object.

Refreshing the preview should show something like the following:



What a drag

The first thing to do to make something draggable in HTML5 is to append the `draggable` attribute to the element. This signals to the web browser to create a ghost image of the element that will appear and follow the mouse pointer when the user triggers a mouse down event, effectively "dragging" the element, and disappearing when the mouse button is released.

Change this line in `displayShowtimes` where we are displaying the showtime:

```
if (movie.showtime[i]) movieHTML+= '<div class="showtime"
title="'+movie.title+' @ '+movie.theater.title+' ('+movie.theater.
address+')" data-movie= "'+movie.theater.id+':'+movie.id+':'+movie.
showtime[i]+'>'+that.formatTime(movie.showtime[i])+</div> ';
```

It should now include the `draggable="true"` attribute:

```
if (movie.showtime[i]) movieHTML+= '<div class="showtime"
draggable="true" title="'+movie.title+' @ '+movie.theater.
title+' ('+movie.theater.address+')" data-movie= "'+movie.theater.
id+':'+movie.id+':'+movie.showtime[i]+'>'+that.formatTime(movie.
showtime[i])+</div> ';
```

Next, add the following CSS style to `styles.css`.

```
[draggable=true] {
  -moz-user-select:none;
  -khtml-user-select:none;
  -webkit-user-select:none;
  user-select:none;
  -khtml-user-drag:element;
  -webkit-user-drag:element;
}
```



The prefix `-khtml` is for old versions of Safari.

Because the default behavior on browsers when a user clicks and drags is selection of text highlighting we need to override this behavior. The styles given previously are shorthands for different browsers to prevent this behavior.



For Internet Explorer, we will need a JavaScript solution to override the default selection behavior for dragging since there is no equivalent. We will cover this when implementing the JavaScript for the drag-and-drop behavior.

Finally, we will need some JavaScript to handle the events triggered when dragging. To get started, we will need to create a new JavaScript file. We will call it `movienow.draganddrop.js` and place it in the `js` folder. We will also need to add a reference to this new file in `index.html`. Add the following above the closing `body` tag:

```
<script src="js/movienow.draganddrop.js"></script>
```

The script tags in `index.html` should look similar to the following code snippet:

```
<script src="js/ios-orientationchange-fix.js"></script>
<script src="js/jquery-1.8.0.min.js"></script>
<script src="js/jquery.xdomainajax.js"></script>
<script src="js/three.js"></script>
<script src="js/movienow.draganddrop.js"></script>
<script src="js/movienow.charts.js"></script>
<script src="js/movienow.geolocation.js"></script>
<script src="js/movienow.js"></script>
</body>
```

Handling drag with JavaScript

In `movienow.draganddrop.js`, we will start by creating a simple object:

```
var movienow = movienow || {};  
movienow.draganddrop = (function(){  
    var that = this;  
})();
```

Within that object, we will add an `init` method to execute when the showtimes are loaded onto the page. Take a look at the following code:

```
this.init = function() {  
    var dragItems = $('[draggable=true]');  
    for (var i=0; i<dragItems.length; i++) {  
        $(dragItems[i])[0].addEventListener('dragstart', function(event) {  
            return false;  
        });  
        $(dragItems[i])[0].addEventListener('dragend', function(event) {  
            return false;  
        });  
    }  
}
```

The `init` method uses jQuery to find all draggable elements, that is, elements with the `draggable="true"` attribute and value. It then loops the collection of draggable elements and adds an event listener for the `dragstart` and `dragend` events. When a draggable element is dragged, the `dragstart` event is triggered. All event listeners in turn are invoked. In this case, we are simply doing nothing and returning `false`, but later on, we will do something a little bit more interesting.

Drag events

`dragstart` - fires when a draggable element begins to be dragged

`drag` - fires when the mouse is moved while a draggable element is being dragged

`dragend` - fires when a draggable element is dropped (when the user releases the mouse button)

`dragenter` - fires whenever a target element has a dragging element dragged into it

`dragover` - fires for a target element whenever the mouse moves while a dragging element is inside it

`dragleave` - fires whenever a target element has a dragging element dragged from it

`drop` - fires whenever a target element has a dragging element released while inside it



Finally, we need to invoke the `init` method when the movie data loads. In the `displayShowtimes` method in `movienow.geolocation.js`, we will need to end the following line as the last line of the method:

```
init();
```

Before we continue, we need to add the following for the sake of Internet Explorer. Since Internet Explorer does not have a way of using CSS to override the default selection behavior on drag, we need to use a JavaScript implementation. In this case, we handle the `selectstart` event and indicate to the browser that we are dragging and dropping when it is triggered:

```
$(dragItems).bind('selectstart', function() {  
    this.dragDrop(); return false;  
});
```

Our `init` method should now look like the following:

```
this.init = function() {  
    var dragItems = $('[draggable=true]');  
    for (var i=0; i<dragItems.length; i++) {  
        $(dragItems[i])[0].addEventListener('dragstart', function(event) {  
            return false;  
        });  
        $(dragItems[i])[0].addEventListener('dragend', function(event) {  
            return false;  
        });  
    }  
    $(dragItems).bind('selectstart', function() {  
        this.dragDrop(); return false;  
    });  
}
```

Drop it

Now that we can drag stuff around, let us look at how we drop them and do something useful when they are dropped. First of all, we will need a place to drop elements. For showtimes, we will create an area on the right-hand side of the page for dropping elements. Once elements are dropped there, they will be displayed above the **Top 5 Box Office** section.

Let us add a couple of `div` tags within the `aside` tag in `index.html`. We will call them `dropzone` and `dropstage`. Add the following lines to the beginning of the `aside` tag:


```
<div id="dropzone">Drop Here</div>
<div id="dropstage">
  <h2>Selected Times</h2>
</div>
```

The beginning of the `aside` tag should look similar to the following code:

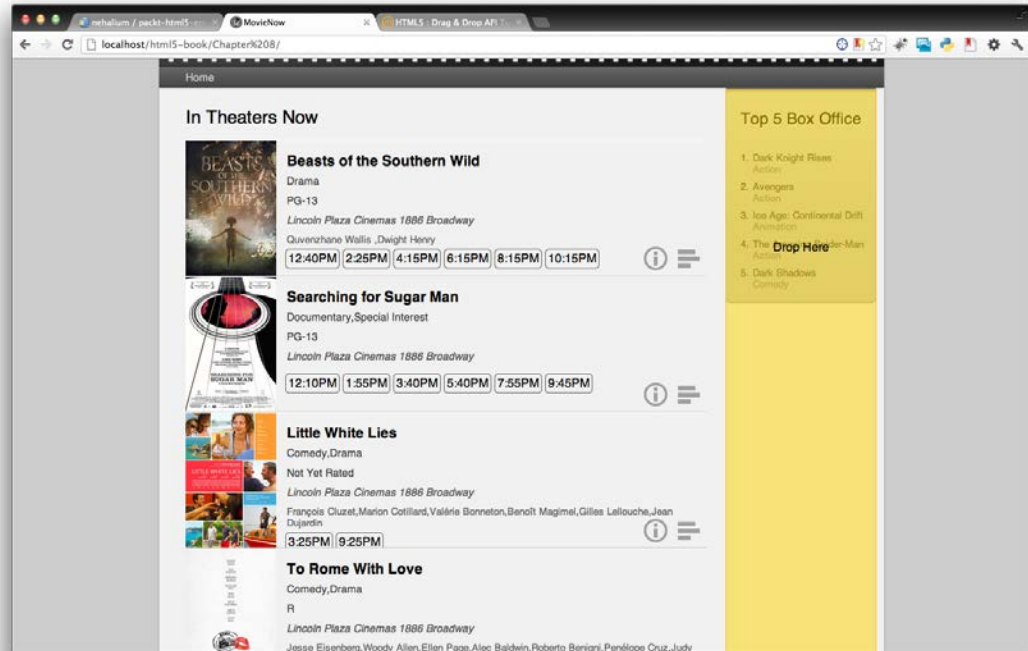
```
<aside>
  <div id="dropzone">Drop Here</div>
  <div id="dropstage">
    <h2>Selected Times</h2>
  </div>
  <h2>Top 5 Box Office</h2>
```

Now that we have a place to drop elements and a place to stage elements once they have been dropped, let us focus on `dropzone`. This is the area in which we will drop elements. Add the following style to `styles.css`:

```
#dropzone {
  border:1px solid #ffb73d;
  width:198px;
  height:auto;
  min-height:100%;
  text-align:center;
  z-index:2;
  position:absolute;
  margin-top:-28px;
  padding-top:200px;
  background: rgb(215, 215, 0);
  background: rgba(255, 215, 0, 0.5);
  filter:progid:DXImageTransform.Microsoft.gradient(startColorstr=#80FFD700, endColorstr=#80FFD700);
  -ms-filter: "progid:DXImageTransform.Microsoft.gradient(startColorstr=#80FFD700, endColorstr=#80FFD700)";
}
```

 Notice that `filter:progid` and `progid` are Internet Explorer specific.

This should add a yellow box that overlays the right-hand side. Your preview should look something like the following screenshot:



toggling the drop zone

Let us now revisit `movienow.draganddrop.js`. Remember that we added event listeners for when a drag is started and when a drag is ended. Let us use this to hide and show the drop zone when dragging.

Add the following to the `dragstart` event listener:

```
$('#dropzone').show();
```

Add the following to the `dragend` event listener:

```
$('#dropzone').hide();
```

The event listeners should look similar to the following code snippet:

```
$(dragItems[i][0]).addEventListener('dragstart', function(event){
    $('#dropzone').show();
    return false;
});
```



```
$(dragItems[i])[0].addEventListener('dragend', function(event) {  
    $('#dropzone').hide();  
    return false;  
});
```

Set the drop zone to be hidden by default, adding the following line to the #dropzone style:

```
display:none;
```

At this point, when you preview, the drop zone should only appear when you are dragging a showtime.

Transferring some data

We are starting to understand the mechanics of dragging-and-dropping. However, in order to make our dragging functionality more interesting, we will need to attach some data to our draggable elements so that once dropped, we have something interesting to show. For the sake of simplicity, let us transfer the element itself. To do this, we set the `dataTransfer` property of the event object. The event object allows us to track events on the page and manage data about them. It is an argument for all event listeners.

Add the following line to the dragstart event listener:

```
event.dataTransfer.setData('Text', this.outerHTML);
```

The dragstart event listener should look similar to the following code:

```
$(dragItems[i])[0].addEventListener('dragstart', function(event){  
    $('#dropzone').show();  
    event.dataTransfer.setData('Text', this.outerHTML);  
    return false;  
});
```

We will then need to add some event listeners to the drop zone so that when an element is dropped upon it, we can have it do something interesting. In this case, we will display it on the drop stage.

Add the following to the `init` method in `movienow.draganddrop.js`:

```
$('#dropzone')[0].addEventListener('drop', function(event) {  
    event.stopPropagation();  
    if (event.preventDefault) event.preventDefault();  
    $('#dropstage').append(event.dataTransfer.getData('Text')).show();  
    return false;  
});
```

```

});
$('#dropzone')[0].addEventListener('dragover', function(event) {
    if (event.preventDefault) event.preventDefault();
    return false;
});
$('#dropzone')[0].addEventListener('dragenter', function(event) {
    if (event.preventDefault) event.preventDefault();
    return false;
});

```

The drop event is central. This is where we handle what happens when an element is dropped. Notice that we take the data stored on dragstart and append it to the div tag of dropstage. We must also stop propagation, prevent default behavior, and return false on the drop, dragover, and dragenter events to prevent the browser from browsing to the element.

Displaying the results

Now that we have the scaffolding for dragging worked out, we will want to display the movie data along with the showtime and style everything for better presentation. We will do this first of all by adding some more data to the event object.

Add the following lines to the dragstart event listener so that we can capture the movie title and theater title as well as the time data as separate datapoints:

```

event.dataTransfer.setData('Title', $(this)[0].title);
event.dataTransfer.setData('Time', $(this).html());

```

The event listener should look similar to the following code snippet:

```

$(dragItems[i])[0].addEventListener('dragstart', function(event){
    $('#dropzone').show();
    event.dataTransfer.setData('Text', this.outerHTML);
    event.dataTransfer.setData('Title', $(this)[0].title);
    event.dataTransfer.setData('Time', $(this).html());
    return false;
});

```

In the drop event listener for the drop zone, modify the following line:

```

$('#dropstage').append(event.dataTransfer.getData('Text')).show();

```

We will need to build some HTML to insert into the DOM object. Take the `Title` and `Time` datapoints and display them accordingly:

```
var html='<div class="selected-time">';
html+='<div class="title">'+event.dataTransfer.getData('Title')+'</div>';
html+='<div class="time">'+event.dataTransfer.getData('Time')+'</div>';
html+='</div>';
$('#dropstage').append(html).show();
```

The entire drop event listener should look similar to the following code snippet:

```
$('#dropzone')[0].addEventListener('drop', function(event) {
    if (event.stopPropagation) event.stopPropagation();
    if (event.preventDefault) event.preventDefault();
    var html='<div class="selected-time">';
    html+='<div class="title">'+event.dataTransfer.getData('Title')+'</div>';
    html+='<div class="time">'+event.dataTransfer.getData('Time')+'</div>';
    html+='</div>';
    $('#dropstage').append(html).show();
    return false;
});
```

Lastly, we will need to style the drop stage so as showtimes are dropped onto it, they look presentable. Add the following to `styles.css` to display the selected time's data and the drop stage appropriately:

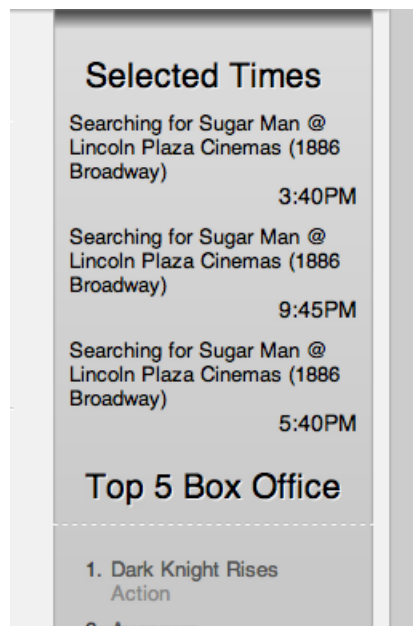
```
.selected-time {
    float:left;
    margin-bottom:10px;
}
.selected-time .title {
    font-size:.8em;
    padding:0 10px;
}
.selected-time .time {
    font-size:.9em;
    clear:both;
    float:right;
    padding-right:10px;
}
#dropstage {
```

```
display:none;
float:left;
width:100%;
padding-bottom: 5px;
margin-bottom: 5px;
}
#dropstage h2 {
border:0;
}
```

Notice that `display` is set to `none` on `#dropstage`. The drop event listener actually shows this section so that it only displays when there are dropped items. We will also need to tweak the style for `aside h2` so that the **Top 5 Box Office** text does not wrap. To do this, simply add the following code:

```
clear:both;
```

Now that the drop stage is styled, we should see our showtimes displayed appropriately when we drop them on the drop zone. Try it out.



Summary

In this chapter, we walked through how to set items as draggable by marking them as such. We walked through the mechanics of dragging-and-dropping and the events that go with this orchestration. Furthermore, we walked through how to put all of this together to achieve some interesting functionality for your HTML5 enterprise application.

In the next chapter, we will talk about HTML5 forms. We will use them to submit tweets about specific showtimes to Twitter.

9

The App: Getting the Word Out via Twitter

This chapter covers our second example of usage of third-party APIs—using the Twitter API as an example—and HTML5 form validation capabilities. Twitter is a social network that allows users to publish and view messages of up to 140 characters. This social network provides a public API that allows developers to do a variety of things. As an exercise, we are going to add Twitter OAuth authentication and message posting to MovieNow, introducing HTML5 form validation.

For this chapter, some basic skill on backend technologies is required. You can review the basics of PHP at the following location: <http://php.net/manual/en/tutorial.php>.

Although we use PHP, it is possible to select another solution and its respective Twitter library (<https://dev.twitter.com/docs/twitter-libraries>). There are libraries for Java, .NET, Ruby, and so on.

Through this chapter we will cover:

- Registering our application
- How to tweet in MovieNow
- Authenticating
- Posting tweets
- New input field types

Registering our application

To use the Twitter API, we need a Twitter account to register our application. If you do not have a Twitter account, you can register one for free at <http://www.twitter.com>.

After registering and logging in, you can go to the Twitter developer page at <https://dev.twitter.com/>.

Developers Search API Health Blog Discussions Documentation Sign In

Build with Twitter.

Embedded Timelines Twitter Cards Embedded Tweets

Get the Tweet Button Get the Follow Button

Recent posts from Twitter Developer Blog

- Sep 6 Sunsetting @Anywhere
- Sep 5 Current status: API v1.1
- Sep 5 How to embed Twitter timelines on your website
- Aug 29 Twitter Certified Products Program - Open for Business

Create applications that integrate Twitter

[Get started with the API](#)
Explore all of Twitter's API documentation


[Create an app](#)
Create an application to start using the Twitter API

[Discuss](#)
Get in touch with the API team and the community of developers


Follow @twitterapi

API Terms API Status Blog Discussions Documentation A Drupal community site supported by Acquia

Click on the **Create an app** link and enter the **Name**, **Description**, and **Website** value of your application. **Callback URL** is the address to which your app will be redirected after users grant permissions to use their accounts. In this case, you can redirect to your index page.

 **Developers**

[API Health](#)
[Blog](#)
[Discussions](#)
[Documentation](#)

 **gabal**

[Home](#) → [My applications](#)

Create an application

Application Details

Name: *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description: *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website: *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL:

Where should we return after successfully authenticating? For [@Anywhere applications](#), only the domain specified in the callback will be used. [OAuth 1.0a](#) applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Rules Of The Road

8. Miscellaneous.

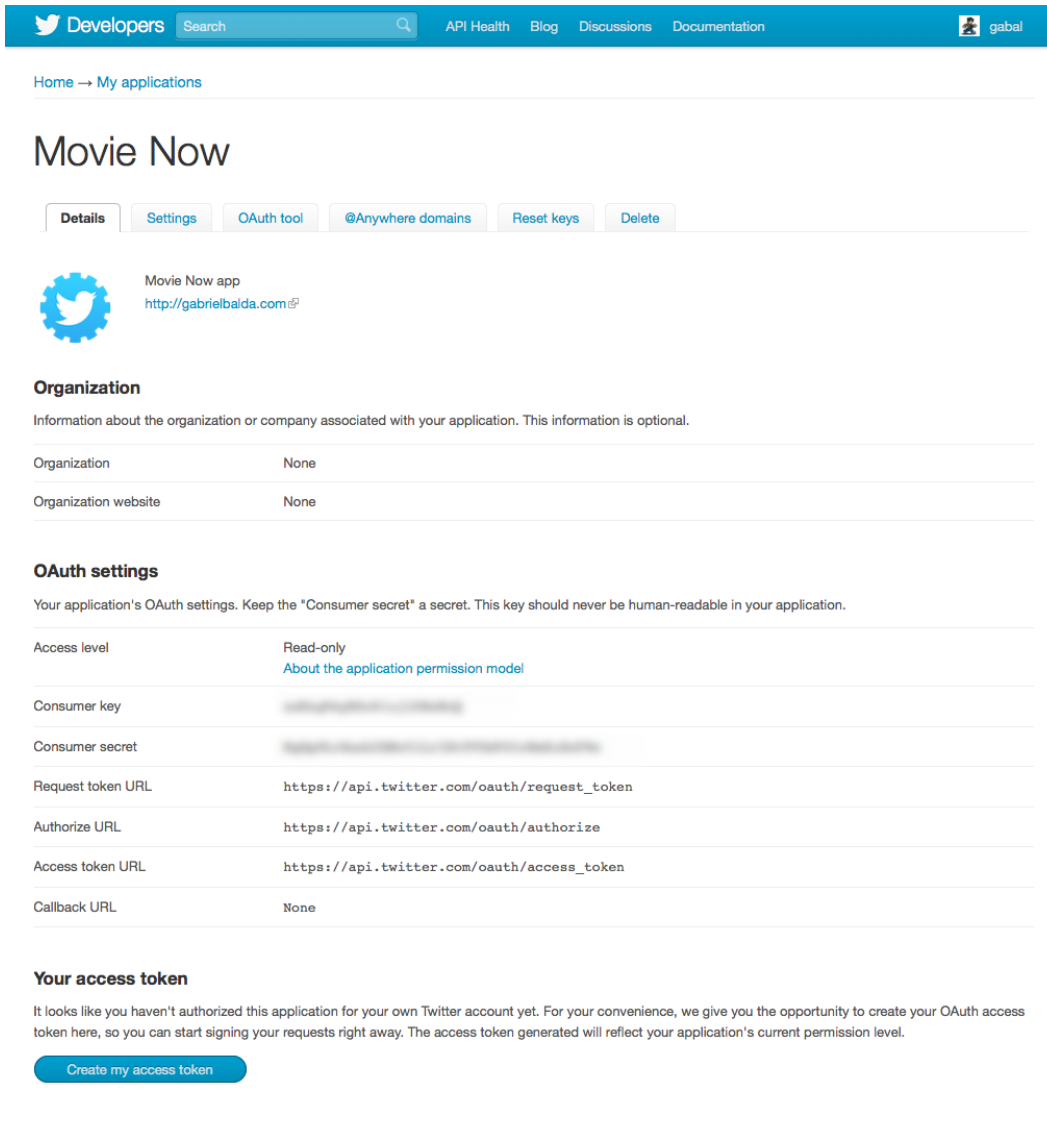
These Rules constitute the entire agreement among the parties with respect to the subject matter and supersedes and merges all prior proposals, understandings and contemporaneous communications. Any modification to the Rules by you must be in a writing signed by both you and Twitter. You may not assign any of the rights or obligations granted hereunder, voluntarily or by operation of law (including without limitation in connection with a merger, acquisition, or sale of assets) except with the express written consent of Twitter, and any attempted assignment in violation of this paragraph is void. This agreement does not create or imply any partnership, agency or joint venture. This agreement will be governed by and construed in accordance with the laws of the State of California, without regard to or application of conflicts of law rules or principles. All claims arising out of or relating to this agreement will be brought exclusively in the federal or state courts of San Francisco County, California, USA, and you consent to personal jurisdiction in those courts. No waiver by Twitter of any covenant or right under this agreement will be effective unless memorialized in a writing duly authorized by Twitter. If any part of this agreement is determined to be invalid or unenforceable by a court of competent jurisdiction, that provision will be enforced to the maximum extent permissible and the remaining provisions of this agreement will remain in full force and effect.

[View the API Terms of Service: Archive](#) or the most recent changes.

After this you only need to accept the terms and conditions, enter the CAPTCHA, and click on **Create your Twitter application**.

Now that you have your Twitter application created, you can see its details in the following page. The most important parameters are **Consume key** and **Consume secret**, they are used to authenticate your app.

 You shouldn't expose **Consume key** and **Consume secret** to the client. All sensitive data should be encrypted.



The screenshot shows the Twitter Developers interface for an application named 'Movie Now'. At the top, there is a navigation bar with the Twitter logo, 'Developers', a search bar, and links for 'API Health', 'Blog', 'Discussions', and 'Documentation'. The user profile 'gabal' is visible in the top right. Below the navigation bar, the breadcrumb 'Home → My applications' is shown. The main heading is 'Movie Now', followed by a row of tabs: 'Details', 'Settings', 'OAuth tool', '@Anywhere domains', 'Reset keys', and 'Delete'. The 'Details' tab is active, showing a Twitter logo icon, the application name 'Movie Now app', and the URL 'http://gabrielbalda.com'. Below this is the 'Organization' section, which states that information is optional and shows 'Organization' and 'Organization website' as 'None'. The 'OAuth settings' section provides instructions to keep the 'Consumer secret' secret and lists various OAuth parameters: 'Access level' (Read-only), 'Consumer key' (blurred), 'Consumer secret' (blurred), 'Request token URL', 'Authorize URL', 'Access token URL', and 'Callback URL' (None). The 'Your access token' section notes that the user hasn't authorized the application and offers a 'Create my access token' button.

By default, Twitter has the **Access** level as **Read only**. We can go to the **Settings** tab, and set **Read and Write** in the **Application Type** section and upload an avatar for our application in the **Application Icon** section.

The screenshot shows the Twitter Developer application settings page for an application named "Movie Now". The page is divided into several sections:

- Navigation:** A blue header bar contains the Twitter logo, "Developers", a search bar, and links for "API Health", "Blog", "Discussions", and "Documentation". A user profile icon for "gabal" is in the top right.
- Breadcrumbs:** "Home → My applications".
- Application Name:** "Movie Now".
- Description:** "Movie Now app".
- Website:** "http://gabrielbalda.com".
- Application Icon:** A section with a "Change icon:" label, a text input field containing a placeholder "M" inside a circle, and a "Browse..." button. Below the input is the text "Maximum size of 700k. JPG, GIF, PNG.".
- Application Type:** A section with the heading "Application Type" and "Access:" label. It contains three radio button options:
 - Read only
 - Read and Write
 - Read, Write and Access direct messagesBelow these is a note: "What type of access does your application need? Note: @Anywhere applications require read & write access. Find out more about our [Application Permission Model](#)."
- Callback URL:** A label for a text input field, which is currently empty.


We are ready to start using the Twitter API.

How to tweet in MovieNow?

To tweet using MovieNow, we need two pieces of functionality: Twitter OAuth authentication and status update (tweet).

The workflow of our example is simple: the user authenticates using a **Sign In** button in the upper-right corner of MovieNow, then we show the username and avatar, and when the user drags a movie to select it (or clicks on iPhone and other drag-disabled devices), we show a tweet form with movie details, that can be posted by clicking on the **Tweet** button.

To simplify our example we are going to use a Twitter-async PHP library that wraps the Twitter API and provides asynchronous calls: <https://github.com/jmathai/twitter-async>.

[ Jaisen Mathai's Twitter-async documentation can be found at <http://www.jaisenmathai.com/articles/twitter-async-documentation.html>.]

Authenticating

Download the Twitter-async library and put it in a `lib` folder that we will create in the root of the application directory. We should have:

- `EpiCurl.php` – abstracts the parallel processing using PHP `multi_curl` functions
- `EpiOAuth.php` – contains basic methods for OAuth authentication
- `EpiTwitter.php` – extends `EpiOAuth` and abstracts the Twitter API authentication and requests

Additionally, we can create a PHP file called `secret.php` to store our `consume_key` and `consume_secret` (you can find them in your Twitter application account page).

```
<?php
    $consumer_key = "<PLACE YOUR CONSUMER KEY HERE>";
    $consumer_secret = "<PLACE YOUR CONSUMER SECRET KEY HERE>";
?>
```

We need to change our `index.html` page extension to `index.php` to add PHP code. After that we call the PHP `session_start` method to start a new session (or resume a previous one if one exists). Then, we import our libraries. Finally, we can instantiate the `EpiTwitter` class with our consumer key (`$consumer_key`) and consumer secret (`$consumer_secret`).

```
<?php
    session_start();
    include 'lib/EpiCurl.php';
    include 'lib/EpiOAuth.php';
    include 'lib/EpiTwitter.php';
    include 'lib/secret.php';
    $twitterObj = new EpiTwitter($consumer_key, $consumer_secret);
?>
```

`$twitterObj` has the information needed to start a new Twitter OAuth session or get user information using the existing one. To show Twitter login or logged information we are going to have two possible cases in our header:

If user is not logged in and/or application is not authorized:

```
<header>
    <div class="logo">
    </div>
    <div class="twitter-info">
        <a href='url'><div class='twitter-signin'></div></a>
    </div>
</header>
```

If user is logged in:

```
<header>
    <div class="logo">
    </div>
    <div class="twitter-info">
        <a href='http://www.twitter.com/username' target='_
blank' class='twitter-data'><img src='avatar' width='30'
height='30' /><span>username</span></a>
    </div>
</header>
```



Notice that we added the class `logo` to differentiate divs with the MovieNow logo inside our header.

User not logged and/or application not authorized

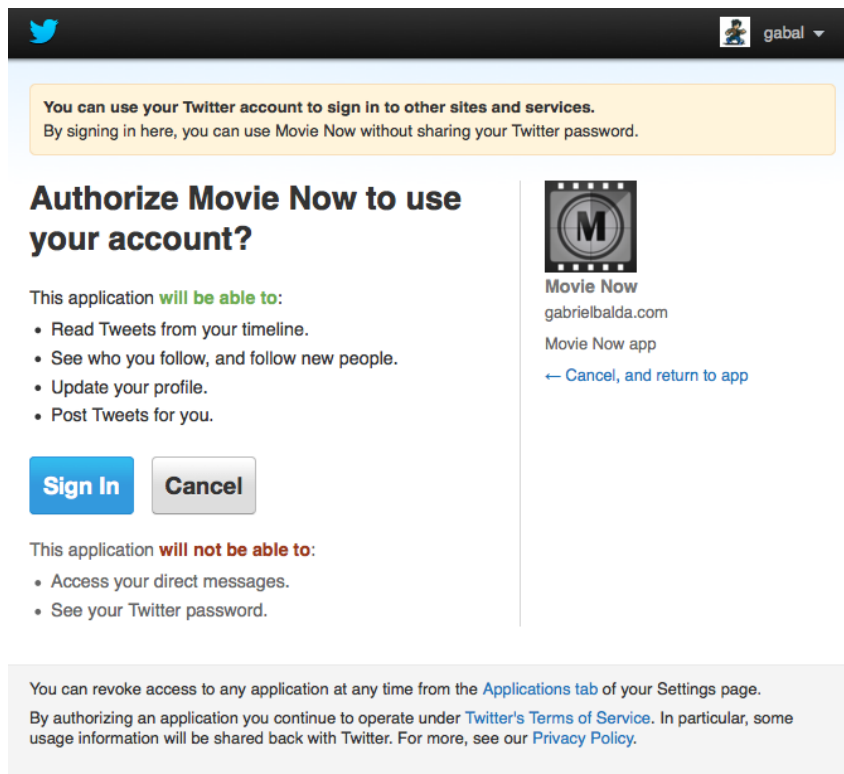
When our users are not logged in, we want to show a link to the Twitter login page and/or authentication page in MovieNow in the upper-right corner.



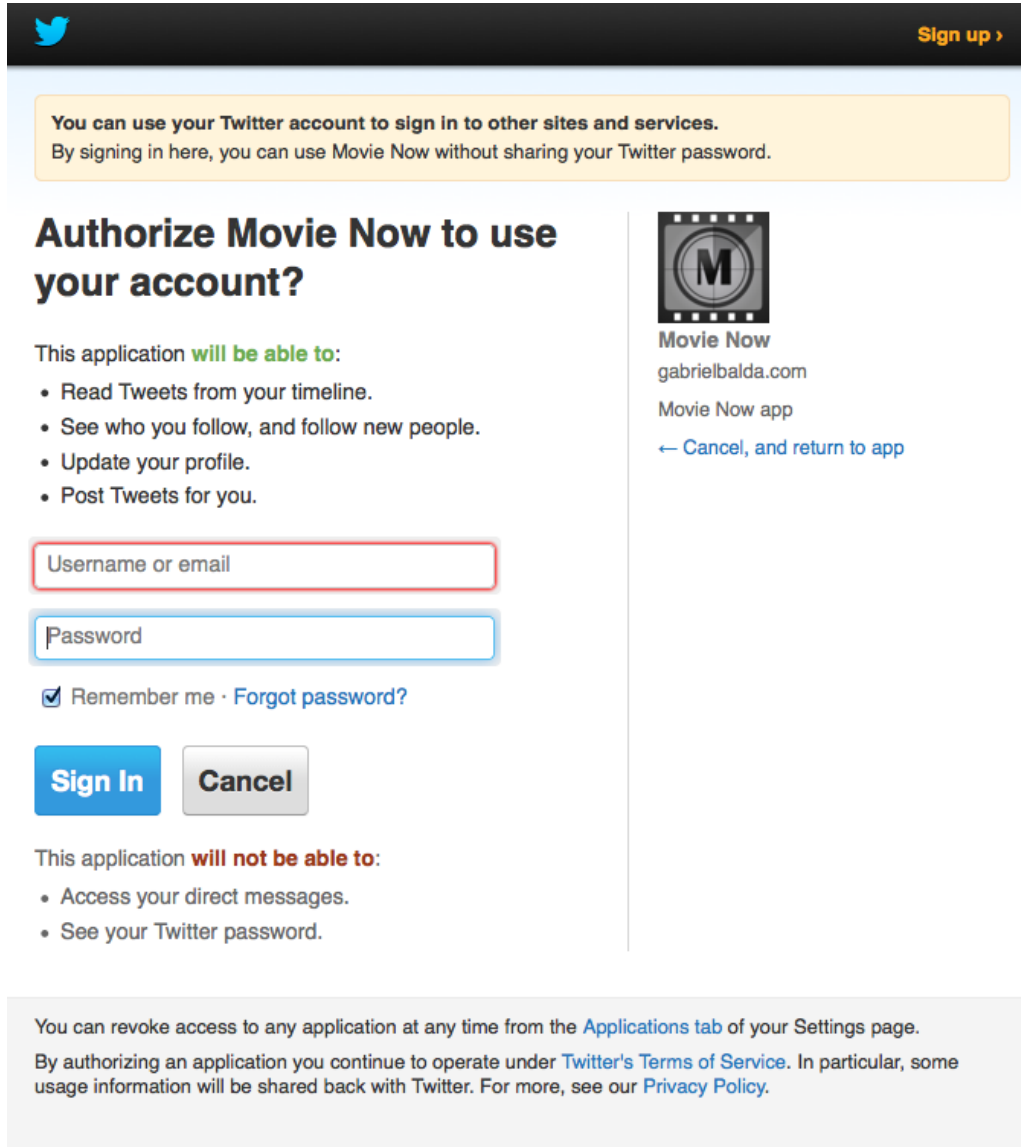
To do that, we obtain the authorization URL from `$twitterObj` and create a link with it:

```
$url = $twitterObj->getAuthorizationUrl();  
echo "<a href='$url'><div class='twitter-signin'></div></a>";
```

If users access this link and they are already logged into Twitter, they will go directly to the MovieNow authorization screen:




If users are not logged into Twitter, they will see the login screen first.



The screenshot shows the Twitter login and authorization interface. At the top, there is a Twitter logo on the left and a "Sign up" link on the right. Below this is a yellow box with the text: "You can use your Twitter account to sign in to other sites and services. By signing in here, you can use Movie Now without sharing your Twitter password." The main heading is "Authorize Movie Now to use your account?". Underneath, it states "This application will be able to:" followed by a list of permissions: "Read Tweets from your timeline.", "See who you follow, and follow new people.", "Update your profile.", and "Post Tweets for you." There are two input fields: "Username or email" and "Password". Below these is a checked checkbox for "Remember me" and a link for "Forgot password?". Two buttons are present: a blue "Sign In" button and a grey "Cancel" button. To the right of the form, there is a section for the application: a logo for "Movie Now" (a film strip with an 'M'), the name "Movie Now", the website "gabrielbalda.com", and the text "Movie Now app". Below this is a link: "← Cancel, and return to app". At the bottom, there is a grey box with text: "You can revoke access to any application at any time from the Applications tab of your Settings page. By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy."

After logging in, they will see the Twitter MovieNow authorization screen. When the login/authorization process ends, browsers are redirected to your **Callback URL**.

 Remember that you can change your **Callback URL** anytime by going to <https://dev.twitter.com/> and changing it in the **Settings** tab.

The callback URL will receive `oauth_verifier` and `oauth_token` as parameters. We need to use `oauth_token` to set the user session information. In session we are going to store `$_SESSION['oauth_token']` and `$_SESSION['oauth_token_secret']`. If they are set, then the user is already logged in. If not, we need to use `$_GET['oauth_token']` to set our session information.

First, we verify:

```
if(isset($_GET['oauth_token']) || (isset($_SESSION['oauth_token']) &&
isset($_SESSION['oauth_token_secret']))) {
    if(!isset($_SESSION['oauth_token']) || !isset($_SESSION['oauth_
token_secret'])) {
        //SET SESSION HERE
    }
}
```

Then, to set our session, we use `$twitterObj` passing `$_GET['oauth_token']`:

```
$twitterObj->setToken($_GET['oauth_token']);
```

We then access the token information to set our session:

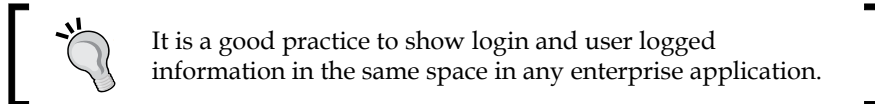
```
$token = $twitterObj->getAccessToken();
$_SESSION['oauth_token']=$token->oauth_token;
$_SESSION['oauth_token_secret']=$token->oauth_token_secret;
```

Finally, we set our token using `oauth_token` and `oauth_token_secret` obtained from the `getAccessToken` method:

```
$twitterObj->setToken($token->oauth_token, $token->oauth_token_
secret);
```

User logged in

If users are logged in, they would like to see something that informs them that they are logged in. For that reason, we are going to show their Twitter avatar and username in the upper-right corner of our header.



It is a good practice to show login and user logged information in the same space in any enterprise application.

Now we only need to get user information, so we can use `get_accountVerify_credentials` to get the username and avatar location:

```
$twitterInfo= $twitterObj->get_accountVerify_credentials();
$username = $twitterInfo->screen_name;
$avatar = $twitterInfo->profile_image_url;
```

Build a link to the Twitter user account and display the avatar:

```
echo "<a href='http://www.twitter.com/$username' target='_blank'
class='twitter-data'><img src='$avatar' width='30' height='30' /><span>
$username</span></a>";
```

Wrapping it all together, we have:

```
<header>
  <div class="logo">
  </div>
  <div class="twitter-info">
  <?php
    if(isset($_GET['oauth_token']) || (isset($_SESSION['oauth_token'])
    && isset($_SESSION['oauth_token_secret']))) {
      if(!isset($_SESSION['oauth_token']) || !isset($_SESSION['oauth_
token_secret'])) {
        $twitterObj->setToken($_GET['oauth_token']);
        $token = $twitterObj->getAccessToken();
        $_SESSION['oauth_token']=$token->oauth_token;
        $_SESSION['oauth_token_secret']=$token->oauth_token_secret;
        $twitterObj->setToken($token->oauth_token, $token->oauth_
token_secret);
      }else{
```



```
        $twitterObj->setToken($_SESSION['oauth_token'], $_
SESSION['oauth_token_secret']);
    }
    $twitterInfo= $twitterObj->get_accountVerify_credentials();
    $username = $twitterInfo->screen_name;
    $avatar = $twitterInfo->profile_image_url;
    echo "<a href='http://www.twitter.com/$username' target='_blank'
class='twitter-data'><img src='$avatar' width='30' height='30'><span>
$username</span></a>";
    }else{
        $url = $twitterObj->getAuthorizationUrl();
        echo "<a href='$url'><div class='twitter-signin'></div></a>";
    }
?>
</div>
</header>
```

Adding some styles

Now that our login interaction is done, we can add styles in `styles.css`.


First, we change the div style of header to logo to have another div tag with different styles inside header. Remember to change this in the Retina Display case too (`@media only screen and (-webkit-min-device-pixel-ratio:2), only screen and (min-device-pixel-ratio: 2)`).

The `twitter-info` class is our container for Twitter information whether the user is logged in or not. We set position to absolute to hide the username and show only the user avatar when devices with smaller screens are used.

```
header .twitter-info{
    position:absolute;
    top:20px;
    right:0;
}
```

We can remove the outline from links:

```
header .twitter-info a{
    outline:none;
}
```

 You can remove all outlines from links using `a{outline:none;}`.

Add login image and dimensions:

```
header .twitter-signin{
  width:100px;
  height:36px;
  background:url(..img/twitter-signin.png);
}
```

Position the user avatar:

```
header a.twitter-data img{
  position:absolute;
  left:-38px;
}
```

Set the text style information for the username:

```
header a.twitter-data{
  line-height:30px;

  color:#fff;
  text-decoration:none;
  font-size:.8em;
  padding-right:10px;
}
```

Inside the @media only screen and (max-width: 737px) media query we need to hide the username and show only the Twitter avatar if the user is not logged in.

Hide part of twitter-info block:

```
header .twitter-info{
  right:-67px;
}
```

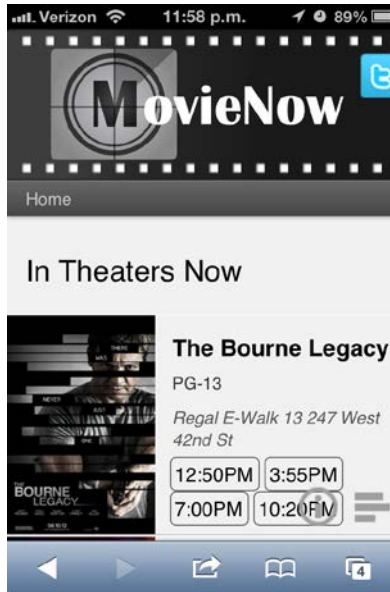
Move the avatar:

```
header a.twitter-data img{
  left:-88px;
}
```

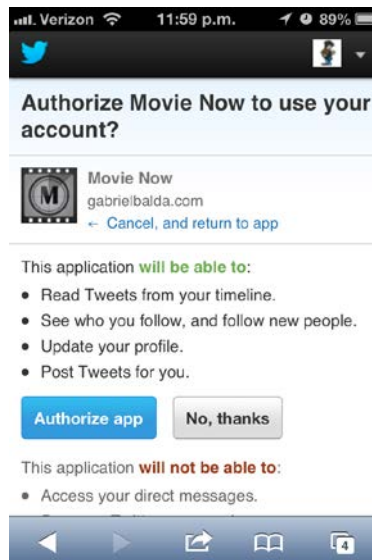
Hide the username:

```
header a.twitter-data span{
  display:none;
}
```

Now our Twitter login process on devices with smaller screens should look like the following screenshot:



After you click on the Twitter button at the top, you will be redirected to the Twitter authentication page (if you are not logged into Twitter, it will redirect you to the login page first).



If you authorize the application, then the next screen will show your Twitter avatar in the top section of the page.

Posting tweets

Our user is now authenticated. The next step is to allow for tweeting about movies. In this case, let us use AJAX to call a service that receives a message to tweet as a tweet parameter.

Service

Let us create a file called `tweet.php` and store it in the root of our application. Import the Twitter-async libraries and `secret.php` configuration file:

```
include 'lib/EpiCurl.php';
include 'lib/EpiOAuth.php';
include 'lib/EpiTwitter.php';
include 'lib/secret.php';
```

We are going to return a JSON object indicating whether the tweet was successfully posted or not. Let us define a variable `$result` and set default status to `ok`.

```
$result = array('status' => 'ok');
```

We can instantiate `EpiTwitter` with our application info and verify the session. If no session exists, set the status to `error`:

```
$twitterObj = new EpiTwitter($consumer_key, $consumer_secret);

if(isset($_SESSION['oauth_token']) && isset($_SESSION['oauth_token_
secret'])) {
    //TWEET
} else {

    $result["status"]="error";
}
```

If the session exists, set the `$twitterObj` token and verify credentials:

```
$twitterObj->setToken($_SESSION['oauth_token'], $_SESSION['oauth_
token_secret']);

$twitterInfo= $twitterObj->get_accountVerify_credentials();
```

If we have the `tweet` parameter, we unescape our message and invoke the `post_statusesUpdate` API passing our tweet inside `status`, an indexed array. We can then save the response in a `$temp` variable.

```
$msg = stripslashes($_REQUEST['tweet']);
$update_status = $twitterObj->post_statusesUpdate(array('status' =>
$msg));
$temp = $update_status->response;
```

Finally, we validate if `$temp` contains an error and return `$result` as JSON:

```
if($temp["error"])$result["status"]="error";
echo json_encode($result);
```

Wrapping it all together, we should have:

```
<?php
include 'lib/EpiCurl.php';
include 'lib/EpiOAuth.php';
include 'lib/EpiTwitter.php';
include 'lib/secret.php';
$result = array('status' => 'ok');
$twitterObj = new EpiTwitter($consumer_key, $consumer_secret);

if(isset($_SESSION['oauth_token']) && isset($_SESSION['oauth_token_
secret'])) {

    $twitterObj->setToken($_SESSION['oauth_token'], $_SESSION['oauth_
token_secret']);

    $twitterInfo= $twitterObj->get_accountVerify_credentials();

    if($_REQUEST['tweet']) {
        $msg = stripslashes($_REQUEST['tweet']);
        $update_status = $twitterObj->post_statusesUpdate(array('status'
=> $msg));

        $temp = $update_status->response;
    }
} else {

    $result["status"]="error";
}
if($temp["error"])$result["status"]="error";
echo json_encode($result);
?>
```

Applying HTML

We are going to display our tweet form as a modal window with a curtain to obscure the rest of the application. We will start by adding the code for the tweet form at the end of the body in `index.php`. Next, we will set up the curtain to cover our page (we will style this as black with transparency):

```
<div class="modal-background-color"></div>
```

For our modal area, we will create a section:

```
<section class="modal-background"></section>
```



For `modal-background-color` we do not use `section` because it doesn't have any semantic information.

We will define the tweet window as a `div`:

```
<div class="tweet"></div>
```

We can then add a bar with title and a close button:

```
<div class="tweet">
  <div class="tweet-bar">
    <h2>Tweet</h2>
    <div id="close-tweet"></div>
  </div>
</div>
```

Lastly, we have our tweet form itself. Notice that we set the `maxlength` attribute of the `textarea` to 140 since tweets can be no more than 140 characters.



The `maxlength` attribute in the `textatarea` element is not supported by Internet Explorer 9 or previous versions. This is because it was not standard for `textarea` in the HTML 4.01 specification, but it was added later in HTML5.

```
<form id="twitter">
  <textarea name="tweet" rows="5" id="tweet" title="Tweet Required!"
  maxlength="140" required></textarea>
  <div class="char-counter">
    <input type='submit' value='tweet' name='submit' id='tweet-
submit' />
    <div id="count">140</div>
  </div>
</form>
```

Putting it all together, we have:

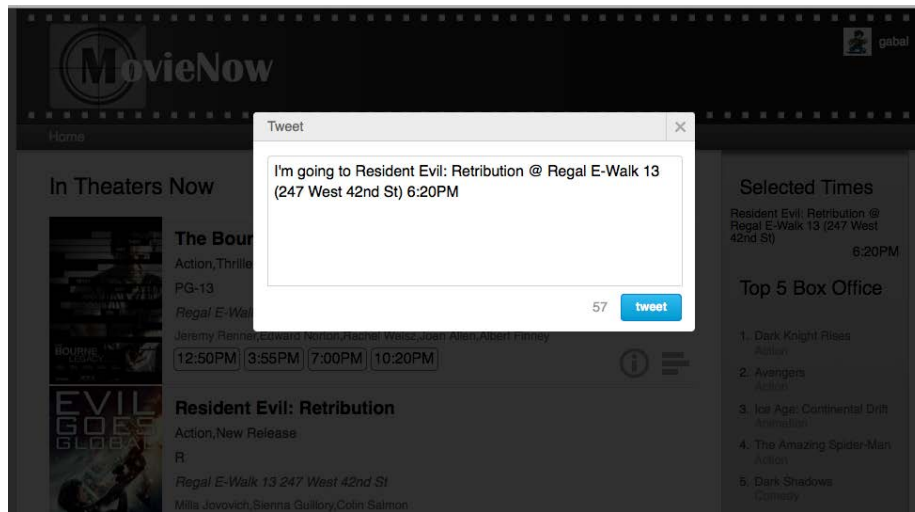
```
<div class="modal-background-color"></div>
<section class="modal-background">
  <div class="tweet">
    <div class="tweet-bar">
      <h2>Tweet</h2>
      <div id="close-tweet"></div>
    </div>
    <form id="twitter">
      <textarea name="tweet" rows="5" id="tweet" title="Tweet
Required!" maxlength="140" required></textarea>
      <div class="char-counter">
        <input type='submit' value='tweet' name='submit' id='tweet-
submit' />
        <div id="count">140</div>
      </div>
    </form>
  </div>
</section>
```

Notice that we are using the `required` attribute to indicate that this field can not be blank. The `required` attribute is an HTML5 attribute used for form validation. We are using `maxlength` as well to limit the number of characters allowed.

```
<textarea name="tweet" rows="5" id="tweet" title="Tweet Required!"
maxlength="140" required></textarea>
```

Adding more styles


We have the HTML structure of our tweet window, but we haven't added styles yet. Our final design should look like the following screenshot:



To achieve this we are going to add some styles in `styles.css`.

1. We set `modal-background-color` to cover our page using *fixed* positioning and `z-index: 5000`. This background should be black with an opacity of 80 percent.

```
.modal-background-color{
  bottom: 0;
  left: 0;
  right: 0;
  top: 0;
  background-color:#000;
  opacity: 0.8;
  display:none;
  -moz-opacity: 0.8;
  -khtml-opacity: 0.8;
  filter: alpha(opacity=80);
  -ms-filter: "progid:DXImageTransform.Microsoft.
Alpha(Opacity=80)";
  position: fixed;
  z-index: 5000;
}
```

 Although `-ms-filter` is being used to give support for older versions of Internet Explorer, it is a proprietary solution of Microsoft that does not follow the standard.

2. We want `modal-background` to be on top of `modal-background-color`, so we use `z-index: 5100`. Both areas should be hidden by default with `display:none` as shown in the following code snippet:

```
.modal-background{
  bottom: 0;
  left: 0;
  right: 0;
  top: 0;
  display:none;
  position: fixed;
  z-index: 5100;
}
```

3. Our tweet area is white with rounded corners and centered using `margin:110px auto`:

```
.tweet{
  background-color:#fff;
  border-radius:4px 4px;
  -moz-border-radius:4px 4px;
  -webkit-border-radius:4px 4px;
  -o-border-radius:4px 4px;
  width:470px;
  margin:110px auto;
}
```

4. We then float the tweet title left and add some padding:

```
.tweet h2{
  float:left;
  padding:8px 15px 6px;
}
```

5. For our text area `#tweet`, we remove the outline, add some padding and margin, and set rounded corners border style and font style:

```
#tweet{
  outline:none;
  padding:5px 5px;
  resize:none;
  width:430px;
  margin:15px 15px 8px;
  border-radius:4px 4px 0 0;
  -moz-border-radius:4px 4px 0 0;
  -webkit-border-radius:4px 4px 0 0;
  -o-border-radius:4px 4px 0 0;
}
```

```
border:1px solid #d3d3d3;
font-size:1em;
line-height:1.4em;
}
```

6. By default, our submit button #tweet-submit is gray:

```
#tweet-submit{
padding:6px 12px;
background-color:#f6f6f6;
font-size:.8em;
border-radius:4px 4px;
-moz-border-radius:4px 4px;
-webkit-border-radius:4px 4px;
-o-border-radius:4px 4px;
border:1px solid #d3d3d3;
cursor:pointer;
color:#888;
font-weight:bold;
}
```

7. Add an active class to set the style when it is active, coloring our button with a blue gradient, blue border, and white font:

```
#tweet-submit.active{
background:linear-gradient(top, #33BCEF, #019AD2);
background:-o-linear-gradient(top, #33BCEF, #019AD2);
background:-moz-linear-gradient(top, #33BCEF, #019AD2);
background:-webkit-linear-gradient(top, #33BCEF, #019AD2);
background:-ms-linear-gradient(top, #33BCEF, #019AD2);
border-color:#057ed0;
color:#fff;
}
```

8. On hover, make it a little darker:

```
#tweet-submit.active:hover{
background:linear-gradient(top, #2DADDC, #0271BF);
background:-o-linear-gradient(top, #2DADDC, #0271BF);
background:-moz-linear-gradient(top, #2DADDC, #0271BF);
background:-webkit-linear-gradient(top, #2DADDC, #0271BF);
background:-ms-linear-gradient(top, #2DADDC, #0271BF);
}
```

9. Float the character counter and submit button to the right and remove the outline:

```
#tweet-submit, #count {
    float: right;
    outline: none;
}
```

10. Set the character counter font style:

```
#count {
    line-height: 30px;
    padding-right: 14px;
    color: #999;
    font-size: .9em;
}
```

11. Add the `close.png` sprite image to `#close-tweet` button and set its styles:

```
#close-tweet {
    cursor: pointer;
    border-left: 1px solid #ccc;
    width: 30px;
    height: 30px;
    float: right;
    background-image: url(../img/close.png);
}
```

12. Move the image on hover:

```
#close-tweet: hover {
    background-color: #E5E5E5;
    background-position: 0 -30px;
}
```

13. Add a different style to the bar that contains the title and close button. Notice that we round off upper-left and right corners:

```
.tweet-bar {
    border-radius: 4px 4px 0 0;
    -moz-border-radius: 4px 4px 0 0;
    -webkit-border-radius: 4px 4px 0 0;
    -o-border-radius: 4px 4px 0 0;
    background-color: #ecec;
    color: #666;
    font-size: .9em;
    border-bottom: 1px solid #ccc;
    overflow: hidden;
}
```

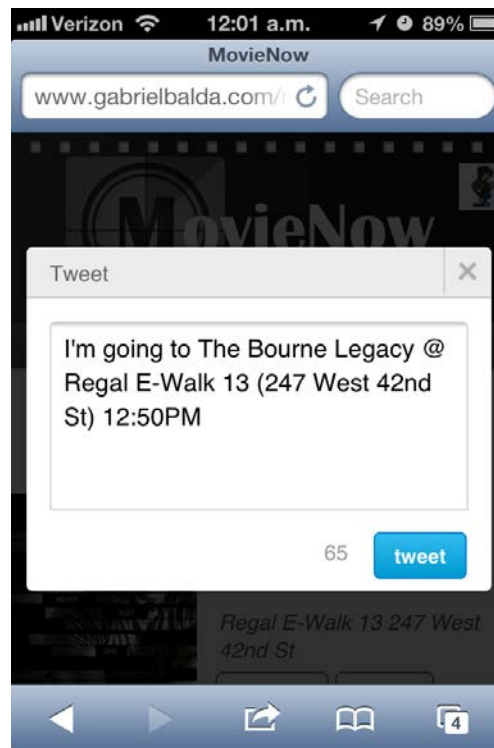
14. Finally, set `height` for the area that contains the tweet submit button and character counter:

```
.char-counter{
  height:40px;
  padding:0 14px;
}
```

15. For mobile devices, add a different `width` for the text area and tweet window using the `@media only screen and (max-width: 737px)` media query:

```
.tweet{
  width:300px;
}
#tweet{
  width:258px;
}
```

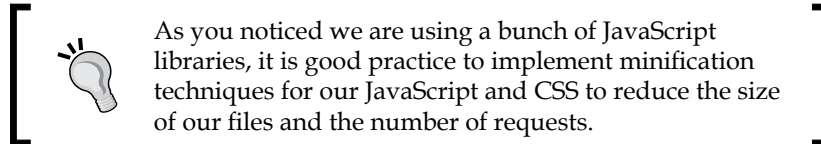
Our tweet windows should look like the following screenshot on devices with smaller screens:



Adding JavaScript interactions

Now we need to add the respective JavaScript to connect the dots. Let us create a new file `movienow.tweet.js` in the `js` folder. We import that file before `movienow.draganddrop.js` inside `index.php`:

```
<script src="js/ios-orientationchange-fix.js"></script>
<script src="js/jquery-1.8.0.min.js"></script>
<script src="js/jquery.xdomainajax.js"></script>
<script src="js/three.js"></script>
<script src="js/movienow.tweet.js"></script>
<script src="js/movienow.draganddrop.js"></script>
<script src="js/movienow.charts.js"></script>
<script src="js/movienow.geolocation.js"></script>
<script src="js/movienow.js"></script>
```



Define the main code structure and add a variable `twitterReady` to verify if the tweet window is initialized:

```
var movienow = movienow || {};
movienow.tweet = (function() {
    var that = this;
    var twitterReady = false;
}) ();
```

Hide and show functionality will be handled by `showTweetArea` and `hideTweetArea` functions. For `showTweetArea`, we verify if `twitterReady` is `true`; if not we call `initTweet` function to set Twitter window events and assign `true` to `twitterReady`.

We can assign `message` (with default text) to the text area and show the tweet window. If the user is not logged in, we show an alert.

```
this.showTweetArea = function(title, time) {
    if (!twitterReady) {
        that.initTweet();
        twitterReady=true;
    }
}
```

```

if($(".twitter-data").length>0){
    var message = "I'm going to "+title+" "+time;
    $("#tweet").val(message);
    that.updateCount();
    $(".modal-background").css("display", "block");
    $(".modal-background-color").css("display", "block");
    $("html,body").css("overflow", "auto");
}
else{
    alert("Please login in twitter to tweet this");
}
};

```



A more elegant way to show notifications and errors in our enterprise application is to define custom modal windows using HTML and style them with CSS. Moreover, it is possible to use template engines (like Mustache, found at <http://mustache.github.com>) to implement this solution.

Notice that we set `$(".html,body").css("overflow", "auto")` to hide the browser scroll bar.

The `hideTweetArea` method hides the tweet window and restores the browser scroll bar:

```

this.hideTweetArea = function() {
    $(".modal-background").css("display", "none");
    $(".modal-background-color").css("display", "none");

    $("html,body").css("overflow", "hidden");
}

```

To update the character counter, we add an `updateCount` method. We extract the `#tweet` text area message and validate its length against 140 characters. We add the active class to our submit button only when it is possible to tweet.

```

this.updateCount = function(){
    var info=$("#tweet").val();

    var count= 140 - info.length;
    $('#count').html(count);

    if(count===140||count<0){

        $('#tweet-submit').removeClass("active");
    }else{

```

```
    $('#tweet-submit').addClass("active");  
  }  
};
```

A native alternative to implement this is the use of the attribute `disabled` in our input field instead of adding and removing the `active` class.

Our initialization method, `initTweet`, adds the necessary events:

```
this.initTweet = function() {  
  $('#tweet').keyup(that.updateCount);  
  $('#twitter').submit(that.tweet);  
  $('#close-tweet').click(that.hideTweetArea);  
};
```

Finally, we create a method to call our PHP service:

```
this.tweet = function(event) {  
  if($('#twitter')[0].checkValidity()){  
    $.ajax({  
      url: 'tweet.php',  
      data: $('#twitter').serialize(),  
      success: function(info){  
        var data = that.objectifyJSON(info);  
        if(data.status!="ok"){  
          alert("Ops! There was an error sending your tweet, please  
try again.");  
        }else{  
          that.hideTweetArea();  
        }  
      }  
    });  
  }  
  return false;  
};
```

The `checkValidity` method is an HTML5 JavaScript function that allows us to verify whether the fields are valid. In our case, since the text area is required, the validation is that it is not empty.

Since we are using AJAX, we add `return false` at the end to avoid a page refresh on submit.



Typically, we would need to extract all inputs from the form and construct a data parameter for our AJAX call. There is a useful function in jQuery called `serialize` that constructs this for us.

Put it all together and we should have:

```
var movienow = movienow || {};
movienow.tweet = (function() {
    var that = this;
    var twitterReady = false;
    this.initTweet = function() {
        $("#tweet").keyup(that.updateCount);
        $("#twitter").submit(that.tweet);
        $("#close-tweet").click(that.hideTweetArea);
    };
    this.tweet = function(event) {
        if($("#twitter")[0].checkValidity()){
            $.ajax({
                url: 'tweet.php',
                data: $("#twitter").serialize(),
                success: function(info){
                    var data = that.objectifyJSON(info);
                    if(data.status!="ok"){
                        alert("Ops! There was an error sending your tweet, please
try again.");
                    }else{
                        that.hideTweetArea();
                    }
                }
            });
        }
        return false;
    };
    this.updateCount = function(){
```



```
var info=$("#tweet").val();
var count= 140 - info.length;
$('#count').html(count);
if(count==140||count<0){
    $('#tweet-submit').removeClass("active");
}else{
    $('#tweet-submit').addClass("active");
}
};
this.showTweetArea = function(title, time) {
    if(!twitterReady){
        that.initTweet();
        twitterReady=true;
    }
    if($(".twitter-data").length>0){
        var message = "I'm going to "+title+" "+time;
        $("#tweet").val(message);
        that.updateCount();
        $(".modal-background").css("display", "block");
        $(".modal-background-color").css("display", "block");
        $("html,body").css("overflow","auto");

    }else{
        alert("Please login in twitter to tweet this");
    }
};
this.hideTweetArea = function() {
    $(".modal-background").css("display", "none");
    $(".modal-background-color").css("display", "none");
    $("html,body").css("overflow","hidden");
}
})();
```



checkValidity is not available in all browsers. It is recommended to give support across browsers writing your own checkValidity function when it is not available. You can check out one approach to this solution on the http://perplexed.co.uk/5201_making_html5_form_backwards_compatible.htm page.

What remains is to add a call in `movienow.dragnaddrop.js`. Inside our listener for the drop event, we add:

```
that.showTweetArea(event.dataTransfer.getData('Title'), event.
dataTransfer.getData('Time'));
```

So, we now have:

```
$('#dropzone')[0].addEventListener('drop', function(event) {
  if (event.stopPropagation) event.stopPropagation();
  if (event.preventDefault) event.preventDefault();
  var html='<div class="selected-time">';
  html+='<div class="title">'+event.dataTransfer.getData('Title')+'</
div>';

  html+='<div class="time">'+event.dataTransfer.getData('Time')+'</
div>';
  html+='</div>';
  $('#dropstage').append(html).show();
  that.showTweetArea(event.dataTransfer.getData('Title'), event.
dataTransfer.getData('Time'));
  return false;
});
```

For mobile and non-drag-and-drop-enabled devices, we add:

```
var iOS = !!navigator.userAgent.match('iPhone OS') || !!navigator.
userAgent.match('iPad');
if(('draggable' in document.createElement('span'))&&!iOS){
  //PREVIOUS CODE
}else{
  $(".showtime").bind('click', function(event){
    var info=$(this)[0].title;
    var time=$(this).html();

    that.showTweetArea(info,time);
  });
}
```

Our tweet system is ready. If we select a movie (dragging or clicking depending on the device), we can tweet by clicking on the **tweet** button.



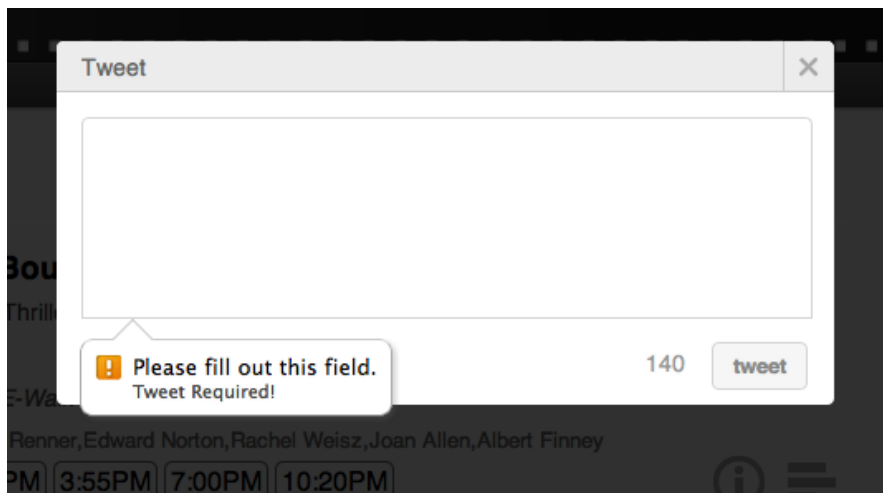
Form validation support across browsers

As in other HTML5 features, form validation is not consistent across browsers. In our case, if we select a movie, delete the text area content in the tweet window, and try to submit, the behavior will be different in each browser.

In Firefox, the form validation we see is a red border and a message.



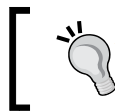
In Chrome, form validation uses title to show **Tweet Required!** message but does not show the red border.



Safari only blocks the submit action.

Even if form validation becomes consistent, the user interface elements of every browser are totally different (you will notice a black tooltip in Firefox and a white one with an icon in Chrome). For now, validation can be done with JavaScript from scratch or with jQuery plugins (<http://docs.jquery.com/Plugins/Validation>).

To disable HTML5 form validation, add the `novalidate` attribute to the form.



Even if data is validated on the client side, it must also be validated on the server side. Remember that for some users it is pretty easy to change JavaScript code.

New input fields types

HTML5 introduces new input types for forms. These allow better control and validation, but sadly are not fully supported in all modern browsers. They are shown as follows:

- `color` is used to select colors
- `date` allows for date selection
- `datetime` allows for date and time selection
- `datetime-local` allows for date and time selection with no time zone
- `email` is used for input fields that should contain email address
- `month` allows for month and year selection
- `number` is used for inputs with numeric values
- `range` renders as a slider that allows for selection of a value within a range of numbers
- `search` is used for search fields
- `tel` is used for entering telephone numbers
- `time` allows for time selection
- `url` is used for inputs that should contain valid URLs
- `week` allows for week and year selection

Summary

With a wide range of social networks and services offering their data via APIs, we can enrich our enterprise applications by complementing them with more data and functionality. Moreover, you can use OAuth authentication to provide alternative authentication methods to your users. Lastly, HTML5 form validation and the new input types appear to not be sufficiently mature to apply them as solutions for enterprise applications; instead fallback JavaScript solutions should be implemented to provide the same experience across browsers. We hope browser developers adopt this part of the HTML5 specification soon because it will translate to shorter development cycles, more reliable form data, and better experiences for the user.

In the next chapter, we will cover Web Workers and the power they give enterprise applications by adding the ability to run background processes and multi-thread our applications.

10

The App: Consuming Tweets Via Web Workers

Up until now, JavaScript has been single-threaded. With a slow-running or lengthy process, everything on your page could come to a screeching halt waiting for something to finish. So far you could use AJAX or even `setTimeout` to delegate your tasks; however, neither of these solutions allows for real parallel execution and handling state gets pretty messy.

To make up for this deficiency, the HTML5 specification introduces Web Workers. Web Workers allow you to create non-user oriented background threads that can run simultaneously. They are typically meant for computationally heavy tasks. However, for our MovieNow enterprise application, we will use Web Workers to find tweets near a theater and display them. Although not necessarily computationally heavy, Web Workers can be useful to update the state of multiple elements on a page without interrupting the overall user experience (notice that there is still generally only a single UI-rendering thread).

In this chapter, we will cover:

- Getting the data
- Capturing geocodes
- Anatomy of a Web Worker
- Using Web Workers to get nearby tweets
- Updating the event listener
- Styling the tweets

Getting the data

To start us out, let us create an endpoint for querying the Twitter REST API and returning tweets near a specified geocode. At the root of the application, create a PHP file called `nearbytweets.php`. Open it and paste in the following code:

```
<?php
    $latitude = $_GET['latitude'];
    $longitude = $_GET['longitude'];
    if (strpos($latitude, '.') == false) {
        $latitude = substr($latitude, 0, -4) . '.' . substr($latitude,
-4);
    }
    if (strpos($longitude, '.') == false) {
        $longitude = substr($longitude, 0, -4) . '.' . substr($longitude,
-4);
    }
    $tweets = file_get_contents('http://search.twitter.com/search.
json?include_entities=true&result_type=mixed&geocode=' . $latitude .
',' . $longitude . ',0.25mi');
    echo $tweets;
?>
```

This is a simple page that takes two parameters: latitude and longitude. It then queries the Twitter REST API 1.1 as defined in <https://dev.twitter.com/docs/api/1.1>. It returns JSON data containing tweets originating from within 0.25 miles from the specified latitude and longitude.

Capturing geocodes

Now that we have a place to get data, we will need to capture the latitude and longitude of each theater to send to our new endpoint. In `movienow.geolocation.js`, we will make a minor modification. In the `displayShowtimes` method, we will need to adjust where the theater name is displayed. Specifically, we will need to input the latitude and longitude and add them to a new `data` attribute. This allows us to use this data later on.

Change the following line:

```
movieHTML+ '<p class="theater">'+movie.theater.title+ " "+movie.
theater.address+'</p>';
```

Change it to:

```
movieHTML+ '<p class="theater" data-location= "' +movie.theater.
latitude+', '+movie.theater.longitude+' ">'+movie.theater.title+
"+movie.theater.address+'</p>';
```

Next, we will create a new JavaScript file called `movienow.nearbytweets.js` in the `js` folder. In `index.php`, we will add a reference to the new JavaScript file:

```
<script src="js/ios-orientationchange-fix.js"></script>
<script src="js/jquery-1.8.0.min.js"></script>
<script src="js/jquery.xdomainajax.js"></script>
<script src="js/three.js"></script>
<script src="js/movienow.nearbytweets.js"></script>
<script src="js/movienow.tweet.js"></script>
<script src="js/movienow.draganddrop.js"></script>
<script src="js/movienow.charts.js"></script>
<script src="js/movienow.geolocation.js"></script>
<script src="js/movienow.js"></script>
```

In `movienow.nearbytweets.js`, we will start with some boilerplate code. Add the following to set up the `nearbytweets` namespace within `movienow`:

```
var movienow = movienow || {};
movienow.nearbytweets = (function() {
    var that = this;
}) ();
```

Anatomy of a Web Worker

To really understand Web Workers, imagine a work-at-home business where households are sent packages of promotions and envelopes. Each household is to stuff the envelopes with the promotional literature, seal the envelopes, and send them as a parcel back to the originating business. The work-at-home households know nothing about the internals of the business. They merely know they are given a parcel, they must do something with the parcel, and they must return the parcel.

Web Workers run in an isolated thread wherein they know nothing about the state of the page that invokes them. They are simply sent a message, they do something with that message, and then return a message. The calling procedure specifies an event listener that responds when a message is returned by a Web Worker.

There are two types of Web Workers. They are:

- **Dedicated Web Workers:** They are sometimes referred to as just Web Workers. They are only reachable by the script that created them, although message ports can be used to communicate with other contexts.
- **Shared Web Workers:** They are named and share a global scope, so any script running in the same origin can obtain a reference of this kind of worker.

In this case, we will use Dedicated Web Workers. Web Workers are typically defined in a separate JavaScript file. To create a Web Worker, you simply need to instantiate it:

```
var worker = new Worker('mywebworker.js');
```

Once created, you can communicate with a Web Worker by using the `postMessage` method:

```
worker.postMessage('Hi worker!');
```

To receive communications back from the Web Worker, simply define an event listener that triggers based on the `onmessage` event:

```
worker.addEventListener('message', function(e) {  
    console.log(e.data);  
}, false);
```

The Web Worker can then be defined as follows:

```
self.addEventListener('message', function(e) {  
    self.postMessage(e.data);  
}, false);
```

It basically has an event listener defined for incoming messages. As messages arrive, it executes the function attached to the message event and returns it by using the `postMessage` method in much the same way it was invoked. The event listener on the client side is invoked and everyone goes on their merry way.

If an error does occur in a Web Worker, the exception can be handled by listening to the error event on the Web Worker:

```
worker.addEventListener('error', function(e) {  
    console.log(e.message);  
}, false);
```

It is important to note that the Web Worker exists in a sandbox. It is not accessing the state of the page at all. Instead, anything it receives will always be a copy of what was sent. Event referenced JavaScript libraries are not available. In fact, the `DOM`, the `window`, the `document`, and the `parent` objects are unavailable, so you cannot do any manipulation of or reading from the `DOM` or make use of the `window` object in a Web Worker. You are a completely separate household.

You can however use the `navigator` object, make use of the `XMLHttpRequest` object, and spawn other Web Workers.



Nested workers must be hosted within the same origin as the parent document. Moreover, the URIs for nested workers are resolved relative to the parent worker's location rather than the owning document.

You can also import scripts using the `importScripts` method as well as use `setTimeout` and `setInterval`.

Using Web Workers to get nearby tweets

In `movienow.nearbytweets.js`, we are going to define a couple of methods. First of all, let us define an entry point method for getting tweets:

```
this.getTweets = function() {};
```

Once we have added this, we can invoke it from `movienow.geolocation.js` at the very end of the `displayShowtimes` method:

```
$("#movies-near-me li .charting-button").click(that.showCharts);
init();
getTweets();
};
```

So far, so good, but we are not doing anything yet. Let us add a new method to `movienow.nearbytweets.js` called `getTweetsByTheater`:

```
this.getTweetsByTheater = function(theater) {};
```

The new `getTweetsByTheater` method will take a "theater" element and get tweets for it. What we mean by a "theater" element in this case is a `div` tag of the `theater` class as defined in `movienow.geolocation.js`. We will then invoke it from the `getTweets` method using a simple jQuery call. Augment `getTweets` as follows:

```
this.getTweets = function() {
    $('.theater').each(function() {
        that.getTweetsByTheater(this);
    });
};
```

Now onto the meat of our script. We will instantiate our Web Worker. Let us start by adding a skeleton of the Web Worker mechanism. Add the following to the `getTweetsByTheater` method:

```
var worker = new Worker('js/movienow.worker.js');
worker.addEventListener('message', function(e) {
}, false);
worker.postMessage();
```

To finish out the skeleton, we will add a new JavaScript file called `movienow.worker.js` to the `js` folder. Add the following code snippet to it:

```
self.addEventListener('message', function(e) {
}, false);
```

Now that we have our initial skeleton of the Web Worker set, let us extract the geocode from the theater object passed into `getTweetsByTheater` and pass it along to the Web Worker. We will take the `data-location` attribute we added earlier in `movienow.geolocation.js` and parse out the latitude and longitude. Replace the skeleton `worker.postMessage()` invocation with the following code:

```
var geocode = $(theater).attr('data-location');
var latitude = geocode.split(',')[0];
var longitude = geocode.split(',')[1];
worker.postMessage({
  'latitude': latitude,
  'longitude': longitude
});
```

Now that we are passing the latitude and longitude to the Web Worker, let us update it to invoke the service we implemented at the beginning of this chapter. Add the following to the body of the event listener in `movienow.worker.js`:

```
var url = '../nearbytweets.php';
var data = 'latitude='+e.data.latitude+'&longitude='+e.data.longitude;
var xhr = new XMLHttpRequest();
xhr.open('GET', url + '?' + data, true);
xhr.send(null);
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200 || xhr.status == 0) {
      self.postMessage(xhr.responseText);
    } else {
      throw xhr.status + xhr.responseText;
    }
  }
}
```

Notice that we are using `XMLHttpRequest`. Normally, we would use the jQuery `ajax` method to make such a call. However, since jQuery has dependencies on the DOM and—as you may recall—the DOM is not available in this context, we cannot use it here. Instead, we will have to invoke the object directly and make the request. Once the AJAX request is made and the `onreadystatechange` event is triggered, the payload is passed back to the client via `self.postMessage()`.

The complete `movienow.worker.js` code should look similar to the following code snippet:

```
self.addEventListener('message', function(e) {
  var url = '../nearbytweets.php';
  var data = 'latitude='+e.data.latitude+'&longitude='+e.data.
longitude;
  var xhr = new XMLHttpRequest();
  xhr.open('GET', url + '?' + data, true);
  xhr.send(null);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200 || xhr.status == 0) {
        self.postMessage(xhr.responseText);
      } else {
        throw xhr.status + xhr.responseText;
      }
    }
  }
}, false);
```




It is possible to implement Web Workers without creating a separate worker file. You can check out a tutorial on implementing this technique at the following location: <http://www.html5rocks.com/en/tutorials/workers/basics/#toc-inlineworkers>

Updating the event listener

Going back to `movienow.nearbytweets.js`, we can update the event listener to take the JSON object returned from the Web Worker. If you look at the following code that we will add to the event listener body, you will see that we get the returned data and loop through the result set to capture the text of each tweet:

```
var data = objectifyJSON(e.data);
var nearbyTweets = '';
for (var i=0; i<data.results.length; i++) {
```

```
    nearbyTweets += '<li>'+data.results[i].text+'</li>';
  }
  var tweetCounter = (data.results.length==1) ? data.results.length+"
  tweet" : data.results.length+" tweets";
  $(theater)
    .append(' <span class="tweet-count">('+ tweetCounter +')</span>')
    .parents('li').append('<section class="nearby-tweets"><h3>Nearby
  Tweets</h3><ul>'+nearbyTweets+'</ul></section>')
    .find('.tweet-count').click(that.showNearbyTweets)
    .parents('li').find('.nearby-tweets').click(that.hideNearbyTweets);
```

 Remember that we defined the `objectifyJSON` function in previous chapters inside `movienow.geolocation.js`. It returns the same input if the parameter passed is an object and parses the content and returns an object if it is a string.

Here we are doing two things. First, we are appending a tweet count to the theater name (notice that we verify the number of tweets to add a singular **tweet** or plural **tweets** label). Secondly, we are adding a `section` element containing an unordered list of tweets. The target behavior is that we click on the tweet count and display the tweets. In addition, we will need to add two more methods to the `nearbytweets` object to show and hide the tweets:

```
this.showNearbyTweets = function(event) {
  $(event.target).parents('li').addClass('nearby-tweets').
  addClass('open');
};
this.hideNearbyTweets = function(event) {
  $(this).parents('li').removeClass('open');
};
```

The complete `movienow.nearbytweets.js` code should look like the following:

```
movienow.nearbytweets = (function(){
  var that = this;
  this.getTweets = function() {
    $('theater').each(function() {
      that.getTweetsByTheater(this);
    });
  };
  this.showNearbyTweets = function(event) {
    $(event.target).parents('li').addClass('nearby-tweets').
    addClass('open');
  };
});
```

```

};
this.hideNearbyTweets = function(event) {
    $(this).parents('li').removeClass('open');
};
this.getTweetsByTheater = function(theater) {
    var worker = new Worker('js/movienow.worker.js');
    worker.addEventListener('message', function(e) {
        var data = objectifyJSON(e.data);
        var nearbyTweets = '';
        for (var i=0; i<data.results.length; i++) {
            nearbyTweets += '<li>'+data.results[i].text+'</li>';
        }
        var tweetCounter = (data.results.length==1) ? data.results.
length+" tweet" : data.results.length+" tweets";
        $(theater)
            .append(' <span class="tweet-count">'+ tweetCounter +'</
span>')
            .parents('li').append('<section class="nearby-
tweets"><h3>Nearby Tweets</h3><ul>'+nearbyTweets+'</ul></section>')
            .find('.tweet-count').click(that.showNearbyTweets)
            .parents('li').find('.nearby-tweets').click(that.
hideNearbyTweets);
    }, false);
    var geocode = $(theater).attr('data-location');
    var latitude = geocode.split(',')[0];
    var longitude = geocode.split(',')[1];
    worker.postMessage({
        'latitude': latitude,
        'longitude': longitude
    });
};
})();

```

Before we go on, we need to make one more modification to `movienow.geolocation.js`. Because we are mimicking the same behavior as the info and ratings buttons, we will need to make sure we are hiding and showing these in conjunction with the nearby tweets section.

Change the `showCharts` method by changing the following line:

```

that.charts($(event.target).parent().parent().removeClass("desc").
addClass("open").find("canvas")[0], "3DChart");

```

We will change it to:

```
that.charts($(event.target).parent().parent().removeClass("desc").  
addClass("open").removeClass('nearby-tweets').find("canvas")[0],  
"3DChart");
```

Change the `showDetails` method by changing the following line:

```
$(event.target).parent().parent().addClass("desc").addClass("open");
```

Change it to:

```
$(event.target).parent().parent().addClass("desc").addClass("open").  
removeClass('nearby-tweets');
```

Styling the tweets

Now that the mechanics are set up for retrieving and loading nearby tweets, we will need to add some styles to make everything look complete. First, we will need to add the same treatment for the new nearby-tweets section as we have for the description and charting sections. We will do this by modifying the following lines in `styles.css`.

Look for the following line:

```
#movies-near-me li,#movies-near-me li section.main-info,#movies-near-  
me li section.description,#movies-near-me li section.charting{
```

Change it by adding a selector for nearby-tweets:

```
#movies-near-me li,#movies-near-me li section.main-info,#movies-near-  
me li section.description,#movies-near-me li section.charting,#movies-  
near-me li section.nearby-tweets{
```

Likewise, look for the following line:

```
#movies-near-me li section.description,#movies-near-me li section.  
charting{
```

Change it by adding a selector for nearby-tweets:

```
#movies-near-me li section.description,#movies-near-me li section.  
charting,#movies-near-me li section.nearby-tweets{
```

Now add the following code to `styles.css`:

```
.tweet-count {  
  cursor:pointer;  
  color:#0000cd;
```

```

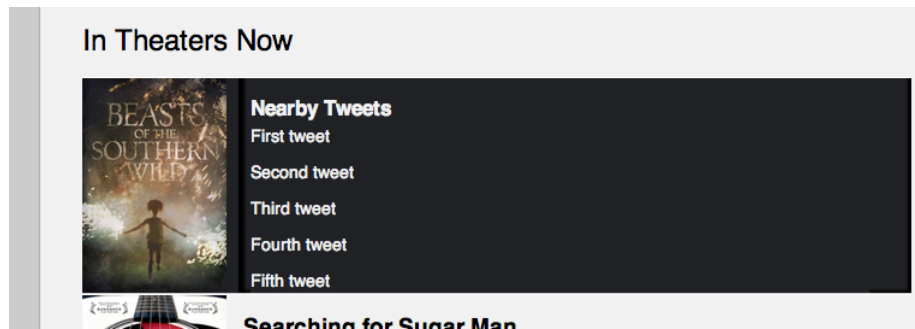
}
#movies-near-me li section.nearby-tweets,#movies-near-me li.nearby-
tweets section.description,#movies-near-me li.nearby-tweets section.
charting {
  display:none;
}
#movies-near-me li.nearby-tweets section.nearby-tweets {
  display:block;
}
#movies-near-me li section.nearby-tweets {
  overflow:scroll;
  padding-left:10px;
}
#movies-near-me li section.nearby-tweets h3 {
  padding: 10px 0 3px 0;
}
#movies-near-me li section.nearby-tweets li {
  height:auto;
  border:0;
  margin-bottom:10px;
}
}

```

With the styles in place, we can test out our behavior by previewing our changes. You should see the following when the data loads:

The screenshot shows a section titled "In Theaters Now". The first movie listed is "Beasts of the Southern Wild", a Drama with a PG-13 rating. It is shown at Lincoln Plaza Cinemas 1886 Broadway, with 18 tweets mentioned. The cast includes Quvenzhané Wallis and Dwight Henry. Showtimes are listed as 12:15PM, 2:10PM, 4:15PM, 6:15PM, 8:15PM, and 10:15PM. Below this, the start of another movie listing for "Searching for Sugar Man" is visible.

Notice **(18 tweets)** next to the theater name. It is the result of the Web Worker's efforts. If you click on it, you should see the following:



If all goes well, the nearby tweets section should open, revealing the latest tweets that were posted near the theater's location.

Summary

We walked through the anatomy of a Web Worker, how to set one up, and how it can be used to solve problems without disaffecting the user experience. Although a contrived example, we stepped through how to use Web Workers to get tweets near a theater based on its geocode.

Some real world use cases for Web Workers include:

- Processor-intensive calculations that should not block the general enterprise application flow
- Auto-correction and syntax highlighting
- Posting images to a message queue
- Consuming data in parallel using concurrent AJAX requests

In the next chapter, we will walk through debugging applications. We will cover tools available at our disposal to get under the hood and figure out what is happening in our enterprise applications. We will also cover some powerful techniques using proxies to sniff out problems as they occur.

11

Finishing Up: Debugging Your App

The debugging process can require a considerable amount of time during the development process. There may be unforeseen behavior, edge cases, and even typos that need to be found and resolved. For that reason, it is necessary to optimize as much as possible and the most important first step is in selecting the right tools. Any development process has to involve testing and debugging; even if your application is working perfectly as expected you should execute test cases and analyze performance to ensure its integrity as you make it evolve and introduce new features. This chapter covers a series of tools for debugging and analyzing performance in your enterprise applications.

This chapter will cover the following:

- What to look for
- Which tools to use
- Playing with HTML and CSS
- Step by step with JavaScript
- Mobile debugging
- Web debugging proxies

What to look for

The debugging process on the client focuses on elements interpreted by a browser. Unless we are using an external plugin (like the Adobe Flash Player), we need to debug:

- HTML to find incorrect styles or test changes in our tag structure
- CSS to verify correctness of styles or test changes on them
- JavaScript to validate code execution, find possible errors, or test changes in our code

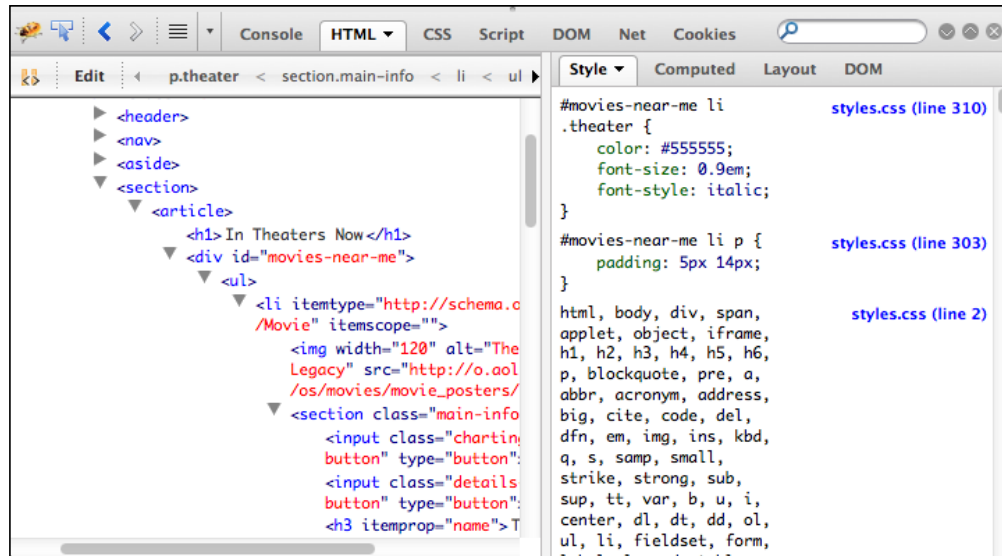
Additionally, we should test the load time of assets (*media, html, css, and js* files) and execution time of JavaScript (profiling).

Which tools to use


Most modern browsers now provide tools for debugging web applications. Because cross-browser compatibility is important, we need to know how they work. In general, every debugger gives you the ability to do the following:

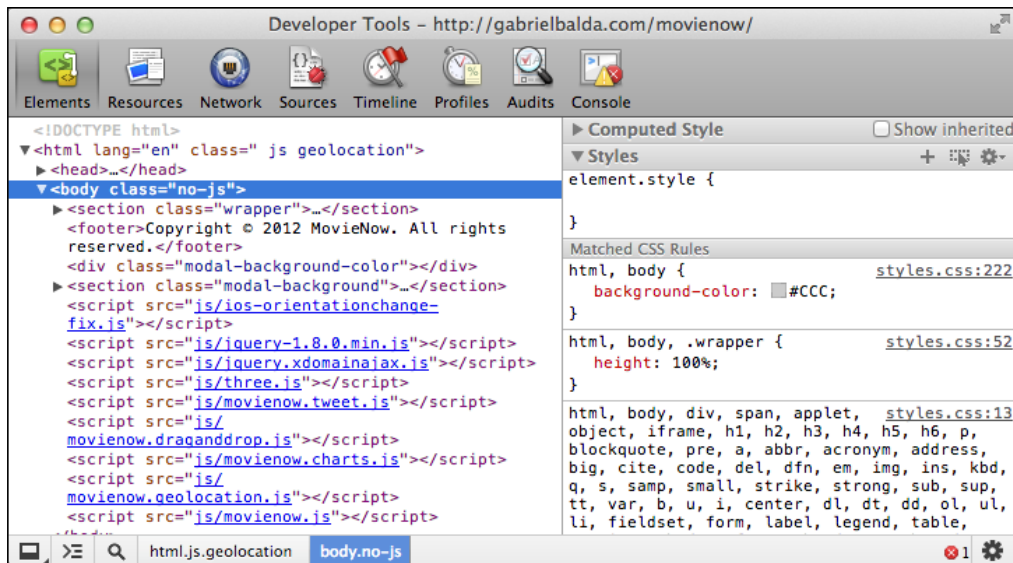
- Inspect and modify HTML in real time and select visual elements on the page and show its equivalents in HTML code (**HTML | Elements**)
- Inspect and modify CSS in real time (**HTML | Elements** or **CSS | Resources | Style**)
- Inspect and modify (or declare) JavaScript in real time and create breakpoints to stop code execution and inspect step by step (**Console | Script | Sources**)
- Analysis of loading time of each asset (**Network | Net | Timeline | Instrument**)

Let us take a look at some of the most popular browsers' debugging tools:




Firefox 15.0.1 Firebug (plugin)

 The Firebug plugin is not installed by default on Firefox. You will need to go to <http://getfirebug.com/> and click on **Install Firebug** from Firefox.



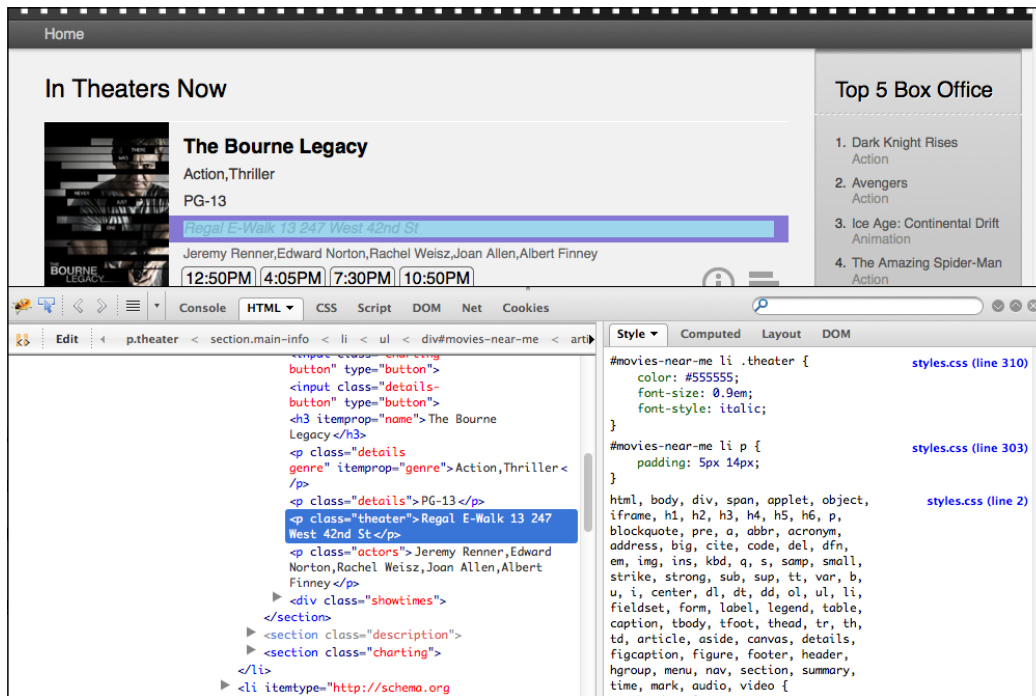
Chrome 22.0.1229.79 Developer Tools

For our purposes and because it is one of the most robust tools, we are going to walk through the Firebug plugin for Firefox. Many of the same basic concepts (HTML inspection, script debugging, use of the console) exist in the developer tools provided in other browsers.

 There are many debugging tools available for Internet Explorer in particular. In addition to the Developer Tools included in Internet Explorer 8 and 9, there is DebugBar (<http://www.debugbar.com/>) and dynaTrace (<http://www.compuware.com/application-performance-management/ajax-performance-testing.html>).

Playing with HTML and CSS

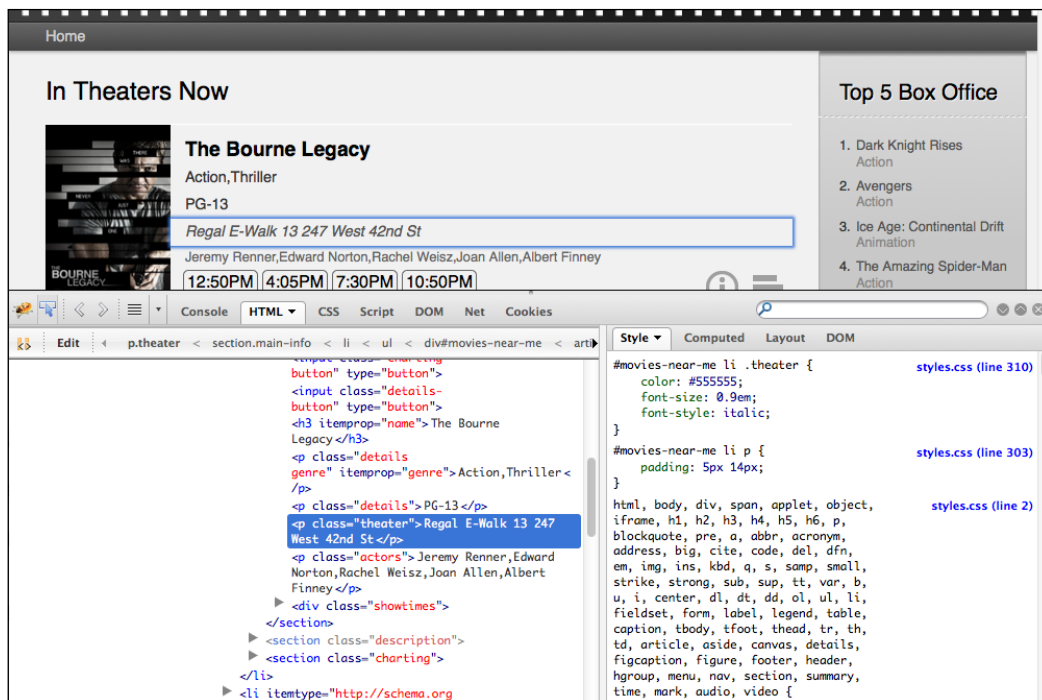
In Firebug, you can explore your HTML DOM objects using the **HTML** tab:



Hovering on each tag, it is possible to see its corresponding rendered element highlighted. If you decide to navigate through the rendered page, you can click on the Inspect button.

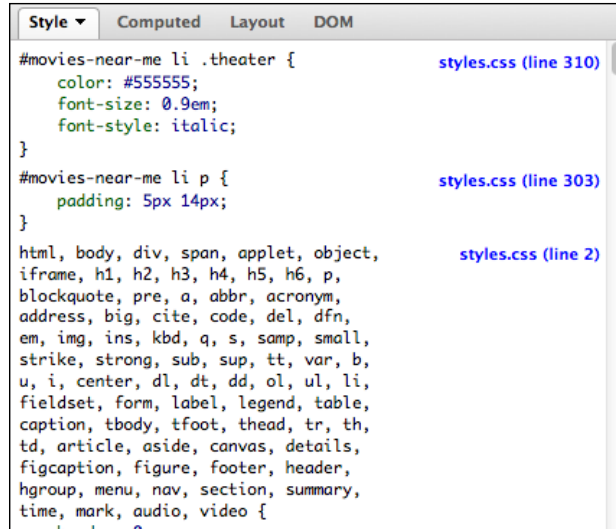


The Inspect button allows you to select areas of the rendered page to see the corresponding HTML code.

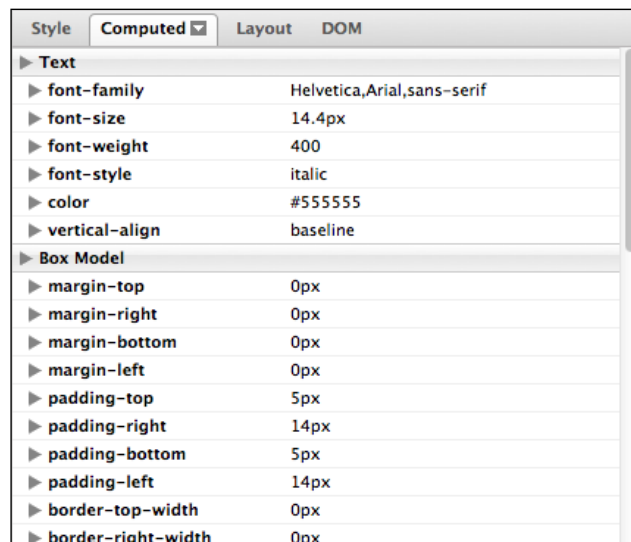


By clicking on **Edit**, it is possible to modify your HTML code within the browser itself. Of course, you are not modifying the page itself but rather the browser's local copy. The rendered page changes automatically as a result of the modifications. This is very useful in trying out changes to your HTML structure and getting instant feedback before implementing them in your code.

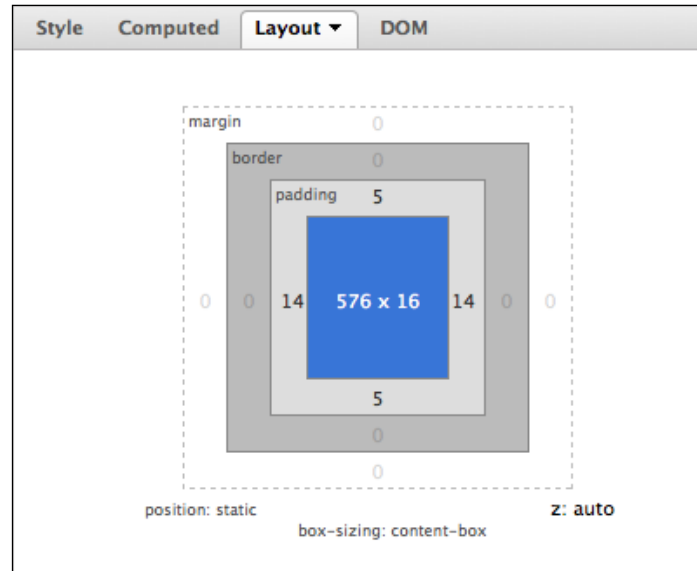
The right-hand side panel allows us to view styles of the DOM object selected, including references to the file and line number (in the right-hand side highlighted in blue). The **Style** tab groups the styles by priority; you can modify or create properties or modify selectors in this tab with a double-click.



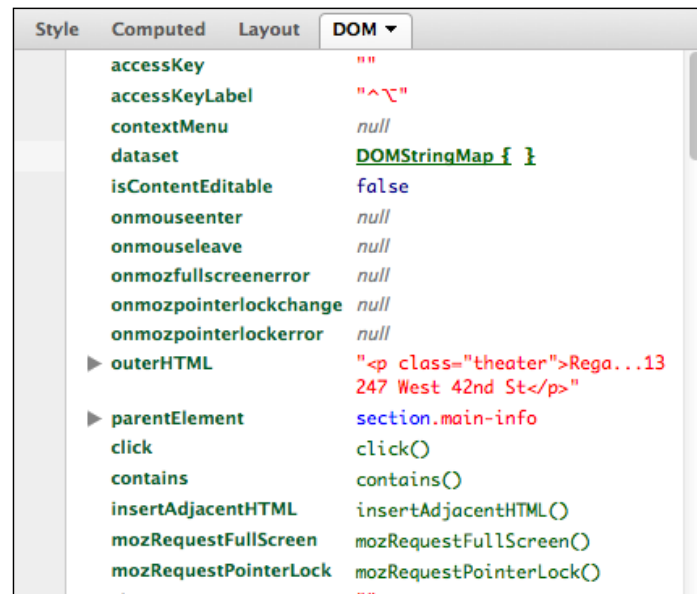
Computed lists the computed styles, which are a merger of browser, user, and author-applied styles where relative values are calculated. For example, if a `div` tag has a style of `width: 50%`, and it is surrounded by a `div` set to `width: 760px`, then the computed style will be `width: 380px`.




Layout displays a graphic representation of the padding, border, and margin applied based on the CSS box model.



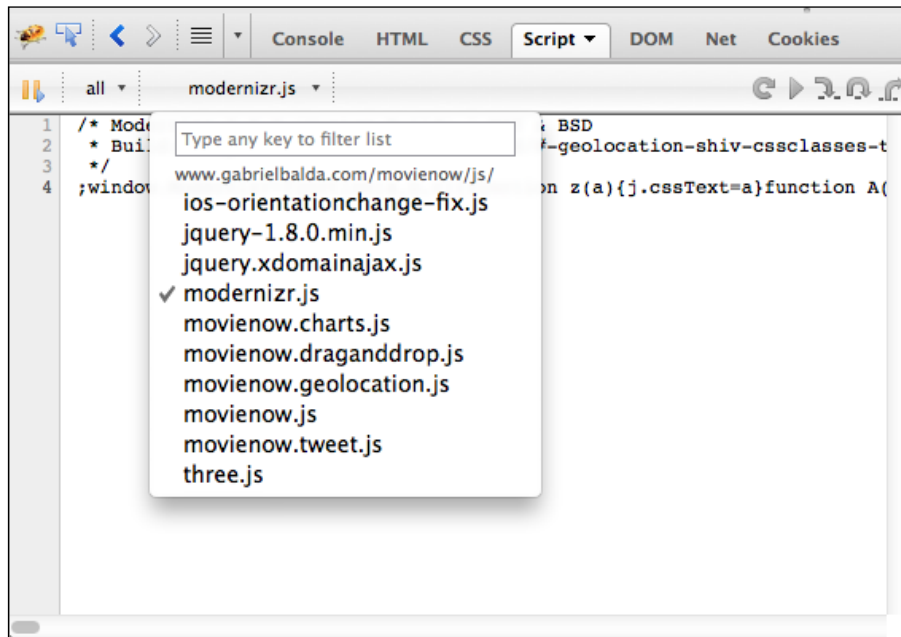
DOM shows the selected DOM's object properties. There is a wealth of information here including the document object as a "browseable" tree and the ability to see the JavaScript global namespace to see which objects have been loaded.




 You can use the **CSS** tab to modify your styles as they are in each `css` file.

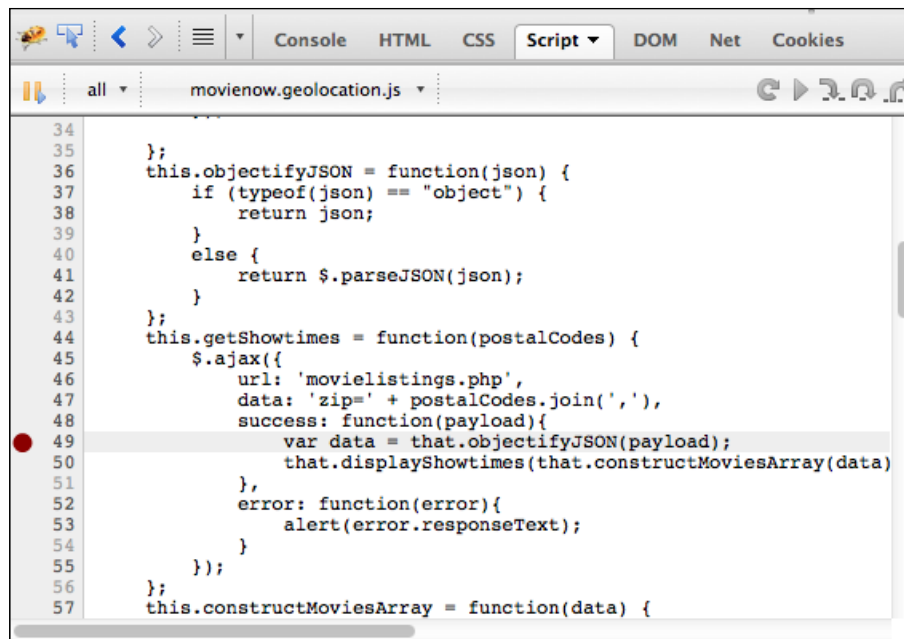
Step by step with JavaScript

The **Script** tab allows you to inspect all JavaScript files used.



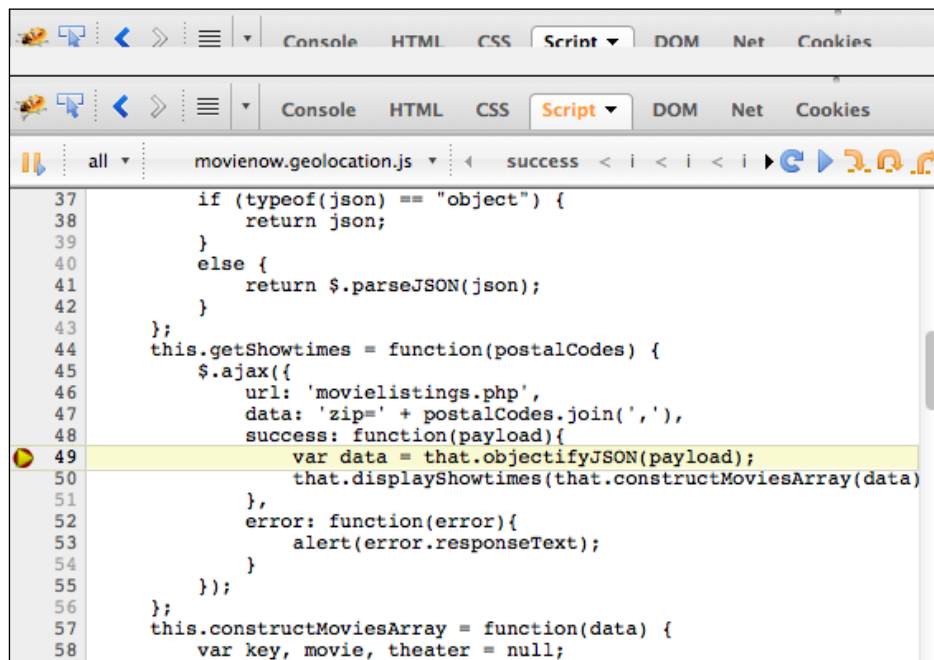
After selecting the file that you wish to inspect, you can execute a step-by-step execution of your JavaScript code. Find the line where you want to stop and click on the line number to create a breakpoint. A graphic representing the breakpoint will appear.

 Minified JavaScript files will generally show up as a single line when viewed in the **Script** tab. It is useful to include "unminified" files in debug versions of your page.



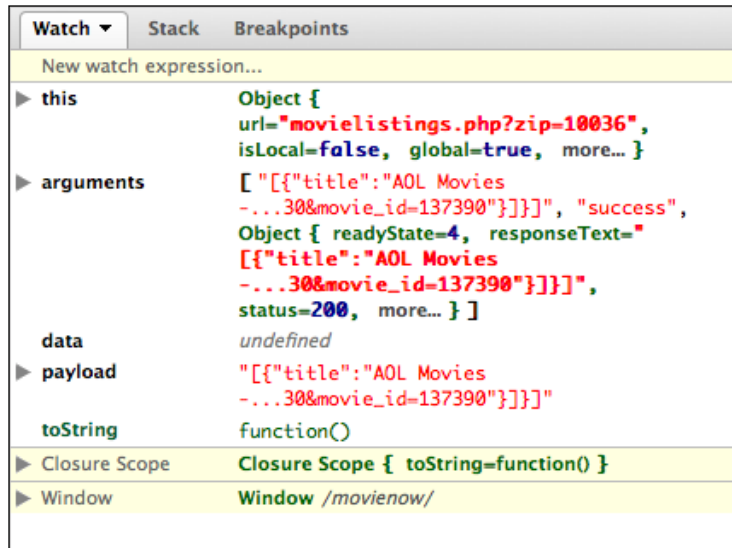
```
34
35
36   this.objectifyJSON = function(json) {
37     if (typeof(json) == "object") {
38       return json;
39     }
40     else {
41       return $.parseJSON(json);
42     }
43   };
44   this.getShowtimes = function(postalCodes) {
45     $.ajax({
46       url: 'movielistings.php',
47       data: 'zip=' + postalCodes.join(','),
48       success: function(payload){
49         var data = that.objectifyJSON(payload);
50         that.displayShowtimes(that.constructMoviesArray(data)
51       },
52       error: function(error){
53         alert(error.responseText);
54       }
55     });
56   };
57   this.constructMoviesArray = function(data) {
```

Now refresh the browser and you will see how the execution stops on that line.



```
37     if (typeof(json) == "object") {
38       return json;
39     }
40     else {
41       return $.parseJSON(json);
42     }
43   };
44   this.getShowtimes = function(postalCodes) {
45     $.ajax({
46       url: 'movielistings.php',
47       data: 'zip=' + postalCodes.join(','),
48       success: function(payload){
49         var data = that.objectifyJSON(payload);
50         that.displayShowtimes(that.constructMoviesArray(data)
51       },
52       error: function(error){
53         alert(error.responseText);
54       }
55     });
56   };
57   this.constructMoviesArray = function(data) {
58     var key, movie, theater = null;
```

The right-hand side panel will show you variables available in the present scope. You can use this to change variable values, input your own variables, or check expressions.



In the top-right corner you will find controls to continue code execution:



The play symbol continues the code execution, the small yellow arrow pointing down continues execution to the next line and will enter a function if one is invoked, the large yellow arrow pointing down continues execution to the next line in the current execution context and passes over functions that are invoked, and the last yellow arrow steps out of a function. Of course, you can always hover over each button to get this information as there is usually a tool tip for each button. This allows you to control how you want to step through your code. You can stop by placing a breakpoint where you want to understand how a particular block of code is evaluating. You can then step through each line of code thereafter, executing method invocations you are not concerned about or stepping out entirely to a calling method. At each step, you can see what variables get set, what is available in the current scope, and really get a sense for what your code is doing.

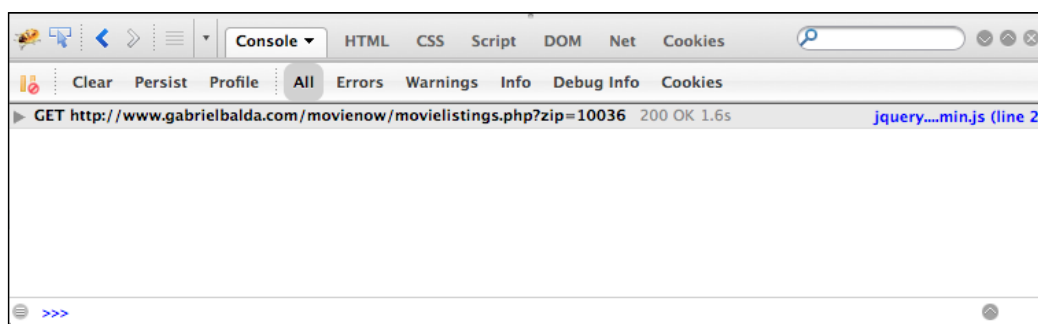
JavaScript console

The **Console** tab logs JavaScript errors, warnings, and Ajax calls. You can force your code to write to the console using the following in your JavaScript code:

```
console.log("CONTENT TO WRITE");
```

This is particularly useful for writing messages out to understand how your code is executing. You can write debug statements to show particular variable values or just indicate that a specific block is executing. You can print objects too and inspect them in Firebug.

The console object includes other useful debugging methods such as `info()`, `warn()`, and `error()` to give you more enhanced debugging feedback.



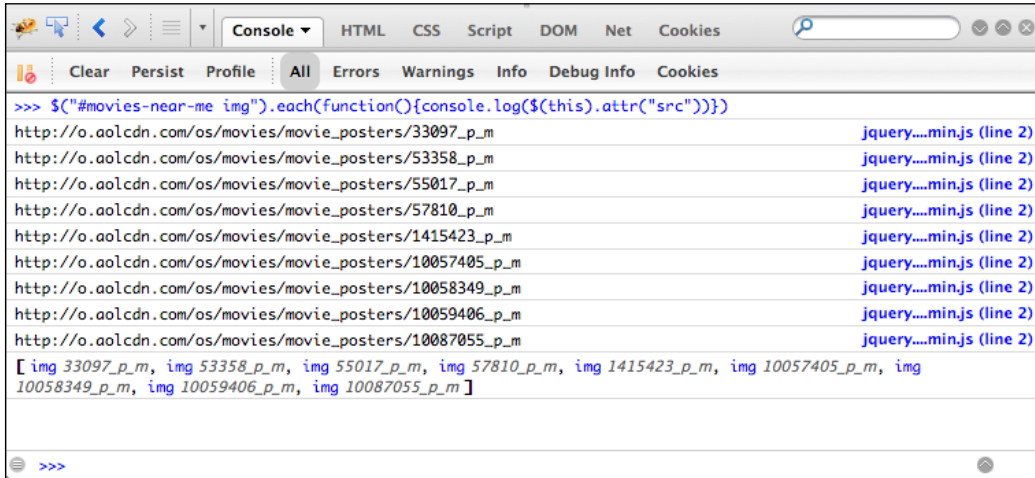
It is good practice to remove or comment out all `console` calls in your code when you deploy to production as some browsers do not support it and will break and stop execution entirely. One solution for this is to create a script that removes these lines and include it in your build process. Oftentimes, such a script also packages and minifies your code.

An interesting feature of the console is that it lets you write code and execute it in real time. You can find a prompt at the bottom that allows you to enter code.

Let us say you want to print all images inside the object with the `movies-near-me` ID, you can write this in your console:

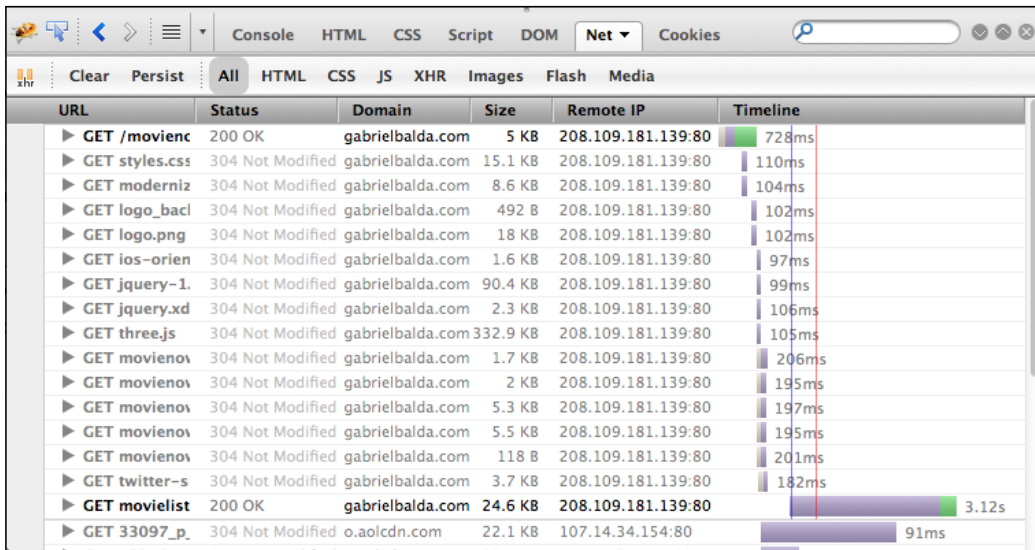
```
$("#movies-near-me img").each(function() {console.log($(this).attr("src"))})
```

You should see the following result:



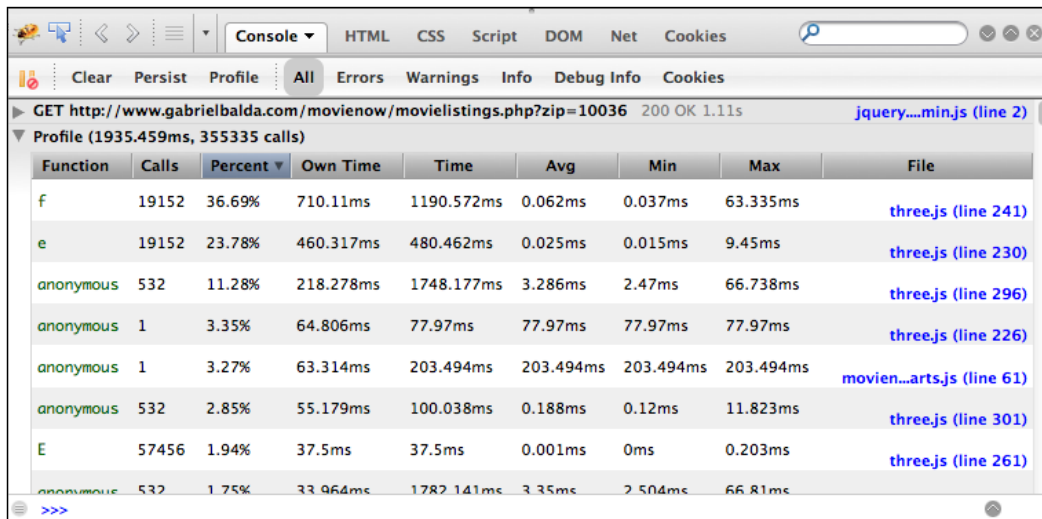
Analyzing load times

The Net tab compiles all network calls including status, domain, file size, IP, and load time. You can use this section to verify where performance problems lie.



JavaScript profiling

The **Console** tab provides a profile tool to analyze performance of our JavaScript code. To execute a profile, click on the **Profile** button, then execute your JavaScript action (or refresh the page if you want to profile the main JavaScript execution) and click on the **Profile** button again.



Function	Calls	Percent	Own Time	Time	Avg	Min	Max	File
f	19152	36.69%	710.11ms	1190.572ms	0.062ms	0.037ms	63.335ms	three.js (line 241)
e	19152	23.78%	460.317ms	480.462ms	0.025ms	0.015ms	9.45ms	three.js (line 230)
anonymous	532	11.28%	218.278ms	1748.177ms	3.286ms	2.47ms	66.738ms	three.js (line 296)
anonymous	1	3.35%	64.806ms	77.97ms	77.97ms	77.97ms	77.97ms	three.js (line 226)
anonymous	1	3.27%	63.314ms	203.494ms	203.494ms	203.494ms	203.494ms	movien...arts.js (line 61)
anonymous	532	2.85%	55.179ms	100.038ms	0.188ms	0.12ms	11.823ms	three.js (line 301)
E	57456	1.94%	37.5ms	37.5ms	0.001ms	0ms	0.203ms	three.js (line 261)
anonymous	532	1.75%	33.964ms	1782.141ms	3.35ms	2.504ms	66.81ms	

No need to worry if the functions are registered as **anonymous**. The **File** field gives the exact line of code.

Mobile debugging

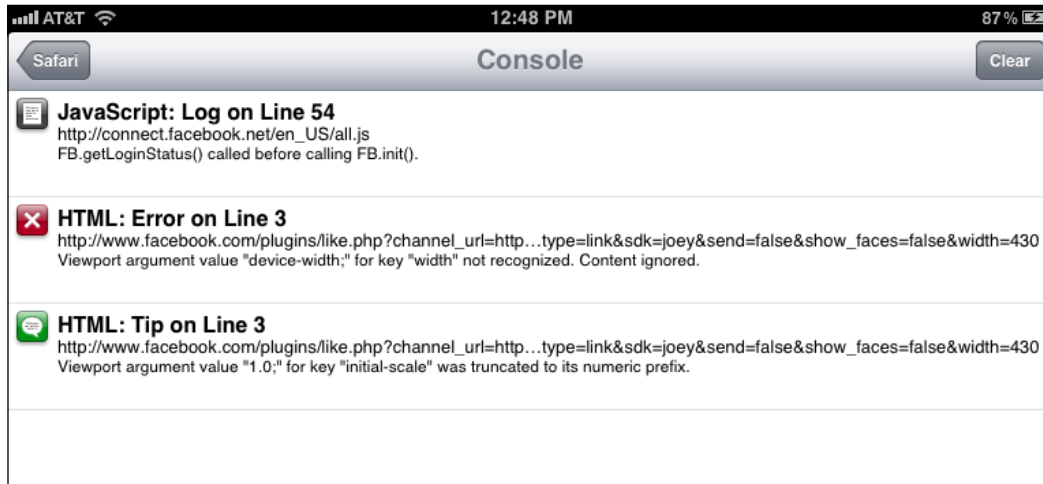
Debugging HTML, CSS, and JavaScript can be painful in mobile web applications, particularly if we are using a touch device. Remote debugging provides a way to test your enterprise web applications in mobile devices using your desktop or laptop.

Chrome supports remote debugging via USB for Android devices:

<https://developers.google.com/chrome/mobile/docs/debugging>.

Firefox 15 introduces remote debugging for Android too: <https://hacks.mozilla.org/2012/08/remote-debugging-on-firefox-for-android/>

Previous to Version 6, iOS devices had a simple interface inside the device for debugging.



Safari 6.0.1 (Mac only) and iOS 6 support remote debugging. To start remote debugging in any iOS 6:

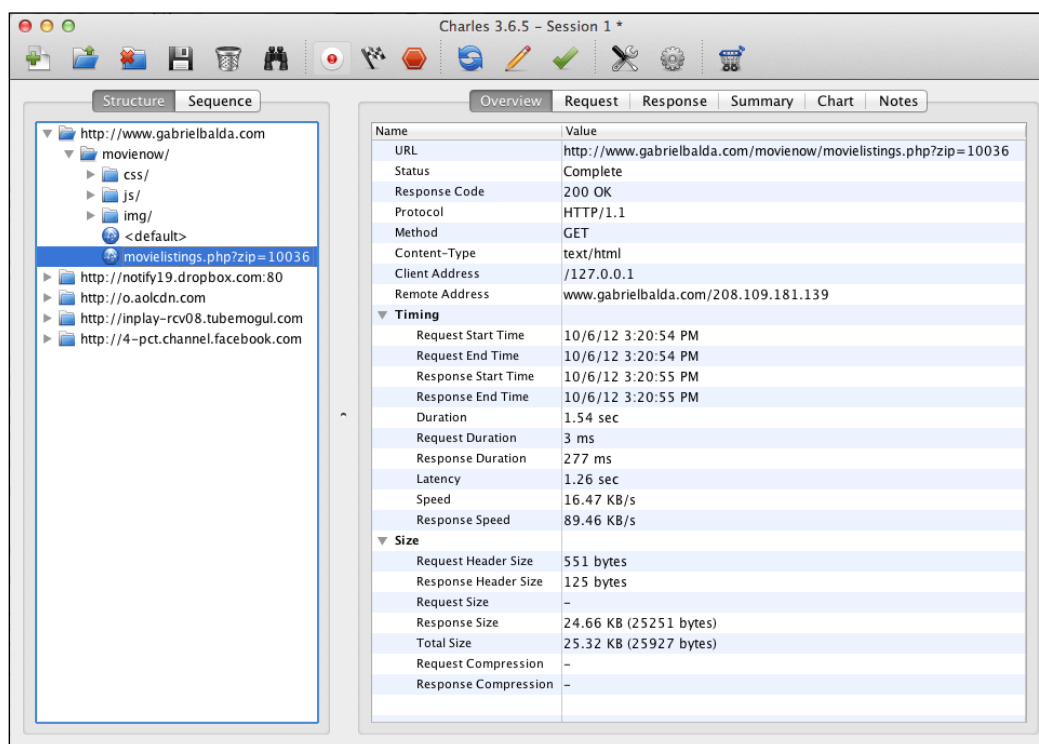
1. In the device (iPhone, iPad, or iPod) go to **Settings | Safari | Advanced** and enable the **Web Inspector** option.
2. Open Safari in a Mac computer.
3. Open the web application to debug using Safari in your iOS device.
4. Connect the device to your Mac computer using a USB cable.
5. If you don't have the **Develop** menu item on Safari (Mac), open **Safari | Preferences | Advanced** and check **Show Develop menu in the menu bar**.
6. **Go to Develop | Device name** and then choose the **web application**.
7. You should now see a web inspector.
8. There is also an online tool called JConsole, which allows you to remotely control and debug browsers in other windows and even devices. It works by giving you a script reference to include in the application you are debugging. It then provides a console to which you can send `console.log` messages and other debugging information. More information can be found at <http://jsconsole.com/remote-debugging.html>.

Web debugging proxies

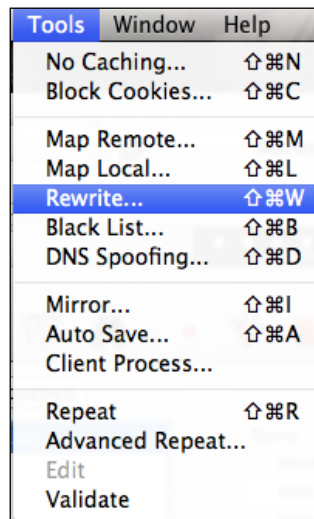
A proxy is a piece of hardware that acts as an intermediary between the client and server. Application of software proxies as a way to debug web applications is a widely used practice, the main purpose of which involves inspecting data from requests and viewing the server responses.

On the Windows platform, we recommend Fiddler. It can be found at <http://www.fiddler2.com/fiddler2/>. If you are using a Mac, you can use Charles, which is found at <http://www.charlesproxy.com/>. Both are web proxies that record the communication between server and client.

Simply open the application, turn on the capture, and load your web application. Here you can see Charles capturing MovieNow requests:



It is possible to simulate responses using proxies as well. For example, in Charles you can go to **Tools** and then **Map Local...** to use local files or services as the response to your application. You can use **Rewrite...** in the same menu to modify part of a response and send it back to your application. This is a valuable feature for testing edge cases.



If you want to use Charles with an iOS device and carry out remote debugging, follow these steps:

1. Get your Mac IP using the `ipconfig getifaddr en1` command in Terminal.
2. Activate the option **Enable Mac OS X Proxy** in the **Proxy** menu.
3. Connect your iOS device to the same network used by your Mac computer running Charles.
4. In your iOS device, go to **Settings** | **Wi-Fi** and click on the blue arrow for your connection and in the **HTTP Proxy** section insert your Mac IP and the port used by Charles (by default it is 8888).
5. Go to Safari in your iOS device and navigate.
6. A warning should show in Charles informing you that a new connection has been attempted. Click on **Allow**.
7. Now you should be able to record your traffic.

Summary

In this chapter, we covered useful debugging tools including mobile and web proxies as well as ways of manipulating HTML, CSS, and JavaScript in the browser in order to debug our applications. As developers, it is good practice for us to understand all the available options for debugging our code. While debugging errors that happen only in one browser, we must pay special attention to the technical restrictions and features of that browser because these limitations are often the cause. For example, complicated JavaScript code can be slow in browsers like Internet Explorer 7. Using the tools available for that browser gives us insight into the internals of that browser and thus knowledge about preventing bugs in the future.

In the next chapter, we will go over testing tools and frameworks for enterprise application projects and the advantages of automated functional testing.

12

Finishing Up: Testing Your App

While the subject of testing could span whole books and there are many books on the subject indeed, we will offer a framework for testing HTML5 enterprise applications as well as an outline of cogent topics that will serve as a point of departure for further study. Different testing tools come with their own particular set of idioms; we will cover the concepts underlying those idioms.

This chapter will cover the following:

- Unit testing
- Functional testing
- Browser testing
- Continuous integration

Types of testing

Testing can happen on many different levels. From the code level to integration and even testing individual functions of the user-facing implementation of an enterprise application, there are numerous tools and techniques to test your application. In particular, we will cover the following:

- Unit testing
- Functional testing
- Browser testing



Black box versus white box testing

Testing is often talked about within the context of black box versus white box testing. This is a useful metaphor in understanding testing at different levels. With black box testing, you look at your application as a black box knowing nothing of its internals – typically from the perspective of a user of the system. You simply execute functionality of the application and test whether the expected outcomes match the actual outcomes. White box differs from black box testing in that you know the internals of the application upfront and can thus pinpoint failures directly and test for specific conditions. In this case, you simply feed in data into specific parts of the system and test whether the expected output matches the actual output.

Unit testing

The first level of testing is at the code level. When you are testing specific and individual units of code on whether they meet their stated goals, you are unit testing. Unit testing is often talked about in conjunction with test-driven development, the practice of writing unit tests first and then writing the minimal amount of code necessary to pass those tests. Having a suite of unit tests against your code and employing test-driven processes – when done right – can keep your code focused and help to ensure the stability of your enterprise application.

Typically, unit tests are set up in a separate folder in your codebase. Each test case is composed of the following parts:

- Setup to build the test conditions under which the code or module is being tested
- An instantiation and invocation of the code or module being tested
- A verification of the results returned

Setting up your unit test

You usually start by setting up your test data. For example, if you are testing a piece of code that requires an authenticated account, you might consider creating a set of test users of your enterprise application. It is advisable that your test data be coupled with your test so that your tests are not dependent on your system being in a specific state.

Invoking your target

Once you have set up your test data and the conditions in which the code you are testing needs to run, you are ready to invoke it. This can be as simple as invoking a method.

Mocking is a very important concept to understand when unit testing. Consider a set of unit tests for a business logic module that has a dependency on some external application programming interface (API). Now imagine if the API goes down. The tests would fail. While it is nice to get an indication that the API you are dependent upon is having issues, a failing unit test because of this is misleading because the goal of the unit test is to test the business logic rather than external resources on which you are dependent. This is where mock objects come into the picture. Mock objects are stubs that replicate the interface of a resource. They are set up to always return the same data the external resource would under normal conditions. This way you are isolating your test to just the unit of code you are testing.

Mocking employs a pattern called dependency injection or inversion of control. Sure, the code you are testing may be dependent on an external resource. Yet how will you swap it in a mock resource? Code that is easy to unit test allows you to pass in or "inject" these dependencies when invoking it.


Dependency injection is a design pattern where code that is dependent on an external resource has that dependency passed into it thereby decoupling your code from that dependency. The following code snippet is difficult to test since the dependency is encapsulated into the function being tested. We are at an impasse.

```
var doSomething = function() {
  var api = getApi();
  //A bunch of code
  api.call();
}
var testOfDoSomething = function() {
  var mockApi = getMockApi();
  //What do I do now???
```

The following new code snippet uses dependency injection to circumvent the problem by instantiating the dependency and passing it into the function being tested:

```
var doSomething = function(api) {
  //A bunch of code
  api.call();
}
```

```
var testOfDoSomething = function() {  
  var mockApi = getMockApi();  
  doSomething(mockApi);  
}
```

 In general, this is good practice not just for unit testing but for keeping your code clean and easy to manage. Instantiating a dependency once and injecting where it is needed makes it easier to change that dependency if the need occurs. There are many mocking frameworks available including JsMockito (<http://jsmockito.org/>) for JavaScript and Mockery (<https://github.com/padraic/mockery>) for PHP.

Verifying the results

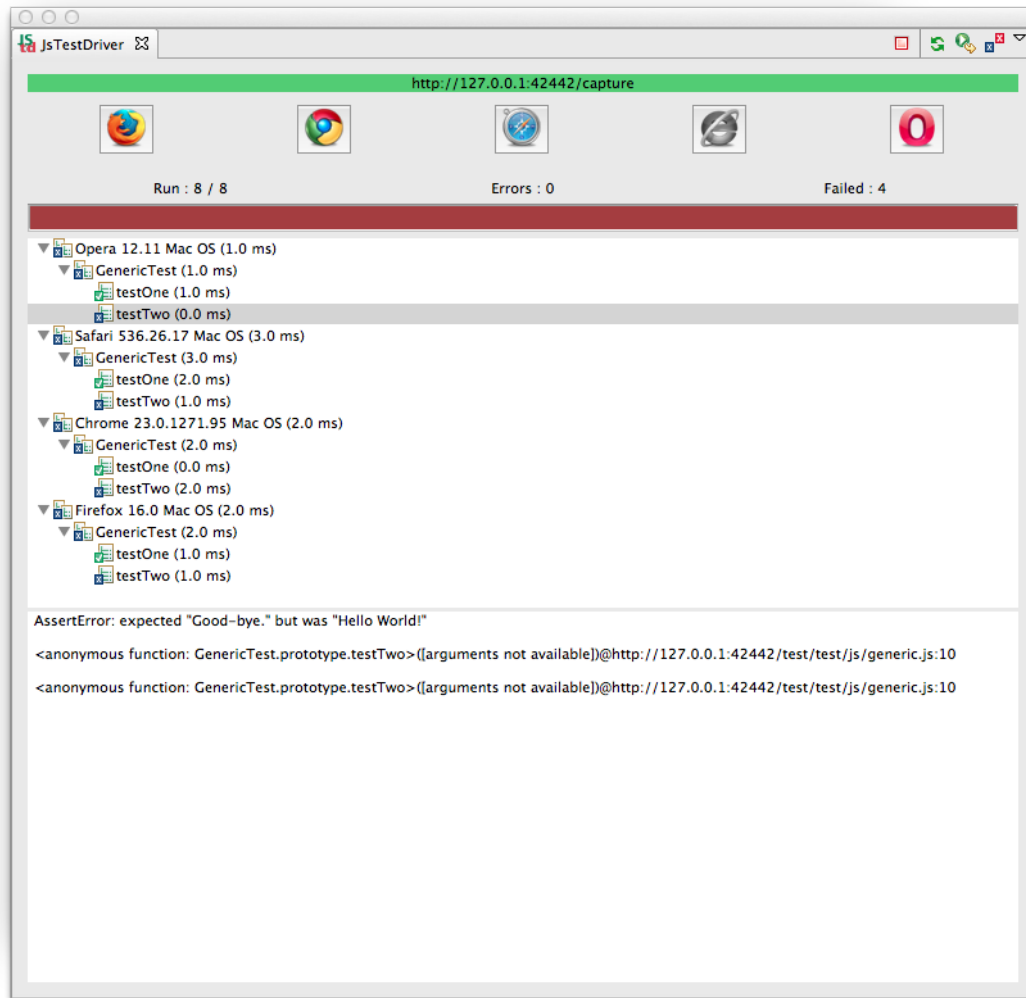
Once you have invoked the code being tested, you need to capture the results and verify them. Verification comes in the form of assertions. Every unit testing framework comes with its own set of assertion methods, but the concept is the same: take a result and test it against an expectation. You can assert whether two things are equal. You can assert whether two things are not equal. You can assert whether a result is a valid number or a string. You can assert whether one value is greater than another. The general idea is you are testing actual data against your hypothesis. Assertions usually bubble up to the framework's reporting module and are manifested as a list of passed or failed tests.

Frameworks and tools

A bevy of tools have arisen in the past few years that aid in unit testing of JavaScript. What follows is a brief survey of notable frameworks and tools used to unit test JavaScript code.

JsTestDriver

JsTestDriver is a framework built at Google for unit testing. It has a server that runs on multiple browsers on a machine and will allow you to execute test cases in the Eclipse IDE.

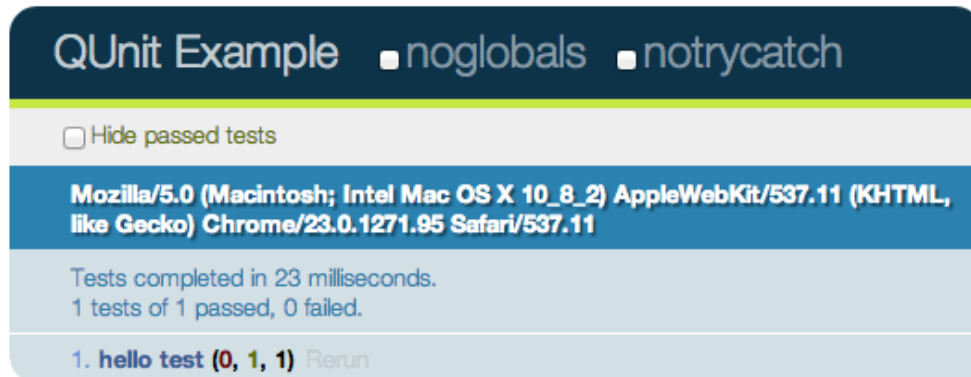


This screenshot shows the results of JsTestDriver. When run, it executes all tests configured to run and displays the results.

More information about JsTestDriver can be found at <http://code.google.com/p/js-test-driver/>.

QUnit

QUnit is a JavaScript unit testing framework created by John Resig of jQuery fame. To use it, you need to create only a test harness web page and include the QUnit library as a script reference. There is even a hosted version of the library. Once included, you need to only invoke the test method, passing in a function and a set of assertions. It will then generate a nice report.



Although QUnit has no dependencies and can test standard JavaScript code, it is oriented around jQuery. More information about QUnit can be found at <http://qunitjs.com/>.

Sinon.JS

Often coupled with QUnit, Sinon.JS introduces the concept of spying wherein it records function calls, the arguments passed in, the return value, and even the value of the `this` object. You can also create fake objects such as fake servers and fake timers to make sure your code tests in isolation and your tests run as quickly as possible. This is particularly useful when you need to make fake AJAX requests.

More information about Sinon.JS can be found at <http://sinonjs.org/>.

Jasmine

Jasmine is a testing framework based on the concept of behavior-driven development. Much akin to test-driven development, it extends it by infusing domain-driven design principles and seeks to frame unit tests back to user-oriented behavior and business value. Jasmine as well as other behavior-driven design based frameworks build test cases – called specs – using as much English as possible so that when a report is generated, it reads more naturally than a conventional unit test report.

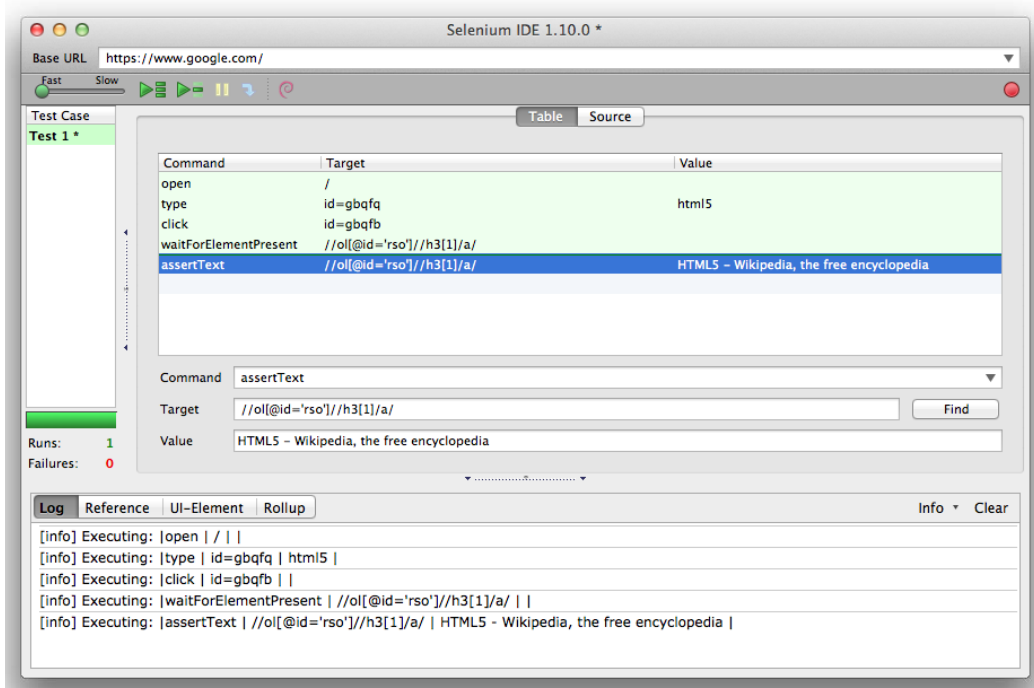
More information about Jasmine can be found at <http://pivotal.github.com/jasmine/>.

Functional testing

Selenium has become the name in website functional testing. Its browser automation capabilities allow you to record test cases in your favorite web browser and run them across multiple browsers. When you have this, you can automate your browser tests, integrate them with your build and continuous integration server, and run them simultaneously to get quicker results when you need them.

Selenium includes the Selenium IDE, a utility for recording and running Selenium scripts. Built as a Firefox add-on, it allows you to create Selenium test cases by loading and clicking on web pages in Firefox. You can easily record what you do in the browser and replay it. You can then add tests to determine whether actual behavior matches expected behavior. It is very useful for quickly creating simple test cases for a web application. Information on installing it can be found at http://seleniumhq.org/docs/02_selenium_ide.html.

The following screenshot shows the Selenium IDE. Click on the red circle graphic on the right-hand side to set it to record, and then browse to `http://google.com` in the browser window and search for "html5". Click on the red circle graphic to stop recording. You can then add assertions to test whether certain properties of the page match expectations. In this case, we are asserting that the text of the first link in the search results is for the Wikipedia page for HTML5. When we run our test, we see that it passes (of course, if the search results for "html5" on Google change, then this particular test will fail).



Selenium includes WebDriver, an API that allows you to drive a browser natively either locally or remotely. Coupled with its automation capabilities, WebDriver can run tests against browsers on multiple remote machines to achieve greater scale.

For our MovieNow application, we will set up functional testing by using the following components:

- The Selenium standalone server
- The php-webdriver connector from Facebook
- PHPUnit

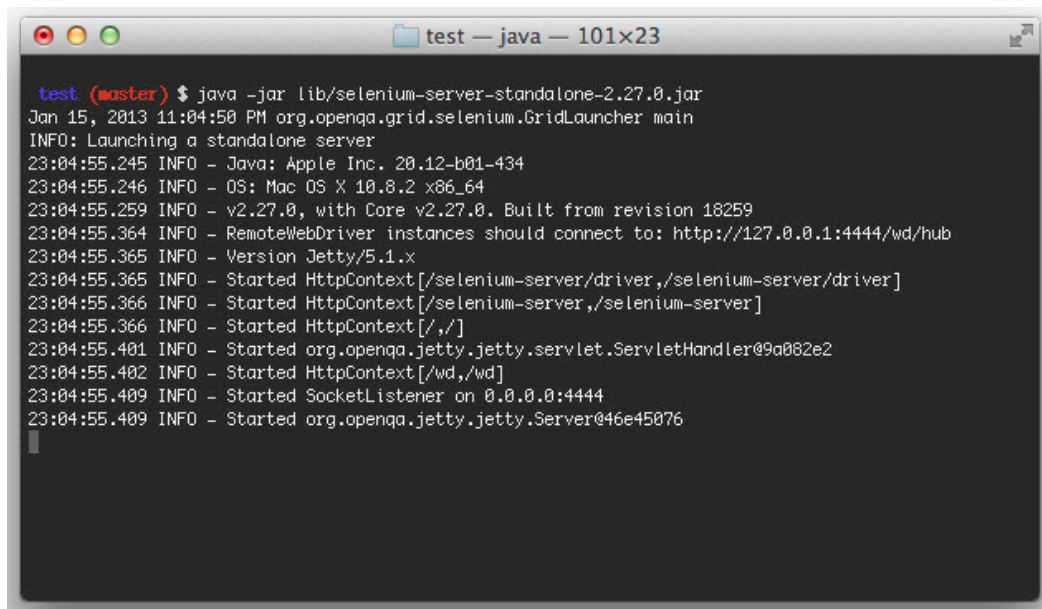
The Selenium standalone server

The Selenium standalone server routes requests to the HTML5 application. It needs to be started for the tests to run. It can be deployed anywhere, but by default it is accessed at `http://localhost:4444/wd/hub`. You can download the latest version of the standalone server at `http://code.google.com/p/selenium/downloads/list` or you can fire up the version included in the sample code under the `test/lib` folder. To start the server, execute the following line via the command line (you will need to have Java installed on your machine):

```
java -jar lib/selenium-server-standalone-#.jar
```

Here, # indicates the version number.

You should see something akin to the following:



```
test — java — 101x23
test (master) $ java -jar lib/selenium-server-standalone-2.27.0.jar
Jan 15, 2013 11:04:50 PM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a standalone server
23:04:55.245 INFO - Java: Apple Inc. 20.12-b01-434
23:04:55.246 INFO - OS: Mac OS X 10.8.2 x86_64
23:04:55.259 INFO - v2.27.0, with Core v2.27.0. Built from revision 18259
23:04:55.364 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd/hub
23:04:55.365 INFO - Version Jetty/5.1.x
23:04:55.365 INFO - Started HttpContext[/selenium-server/driver,/selenium-server/driver]
23:04:55.366 INFO - Started HttpContext[/selenium-server,/selenium-server]
23:04:55.366 INFO - Started HttpContext[/,/]
23:04:55.401 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@9a082e2
23:04:55.402 INFO - Started HttpContext[/wd,/wd]
23:04:55.409 INFO - Started SocketListener on 0.0.0.0:4444
23:04:55.409 INFO - Started org.openqa.jetty.jetty.Server@46e45076
```

At this point, it is listening for connections. You will see log messages here as you run your tests. Keep this window open.

The php-webdriver connector from Facebook

The php-webdriver connector serves as a library for WebDriver in PHP. It gives you the ability to make and inspect web requests using drivers for all the major web browsers as well as HtmlUnit. Thus it allows you to create test cases against any web browser. You can download it at <https://github.com/facebook/php-webdriver>. We have included the files in the `webdriver` folder.

PHPUnit

PHPUnit is a unit testing framework that provides the constructs necessary for running our tests. It has the plumbing necessary for building and validating test cases. Any unit testing framework will work with Selenium; we have chosen PHPUnit since it is lightweight and works well with PHP. You can download and install PHPUnit any number of ways (you can go to <http://www.phpunit.de/manual/current/en/installation.html> for more information on installing it). We have included the `phpunit.phar` file in the `test/lib` folder for your convenience. You can simply run it by executing the following via the command line:

```
php lib/phpunit.phar <your test suite>.php
```

To begin, we will add some PHP files to the `test` folder. The first file is `webtest.php`. Create this file and add the following code:

```
<?php
require_once "webdriver/__init__.php";

class WebTest extends PHPUnit_Framework_TestCase {
    protected $_session;
    protected $_web_driver;

    public function __construct() {
        parent::__construct();
        $_web_driver = new WebDriver();
        $this->_session = $_web_driver->session('firefox');
    }

    public function __destruct() {
        $this->_session->close();
        unset($this->_session);
    }
}
?>
```

The `WebTest` class integrated `WebDriver` into `PHPUnit` via the `php-webdriver` connector. This will serve as the base class for all of our test cases. As you can see, it starts with the following:

```
require_once "webdriver/__init__.php";
```

This is a reference to `__init__.php` in the `php-webdriver` files. This brings in all the classes needed for `WebDriver`. In the constructor, `WebTest` initializes the driver and session objects used in all test cases. In the destructor, it cleans up its connections.

Now that we have everything set up, we can create our first functional test. Add a file called `genericTest.php` to the `test` folder. We will import `WebTest` and extend that class as follows:

```
<?php
require_once "webtest.php";

class GenericTest extends WebTest {
}
?>
```

Inside of the `GenericTest` class, add the following test case:

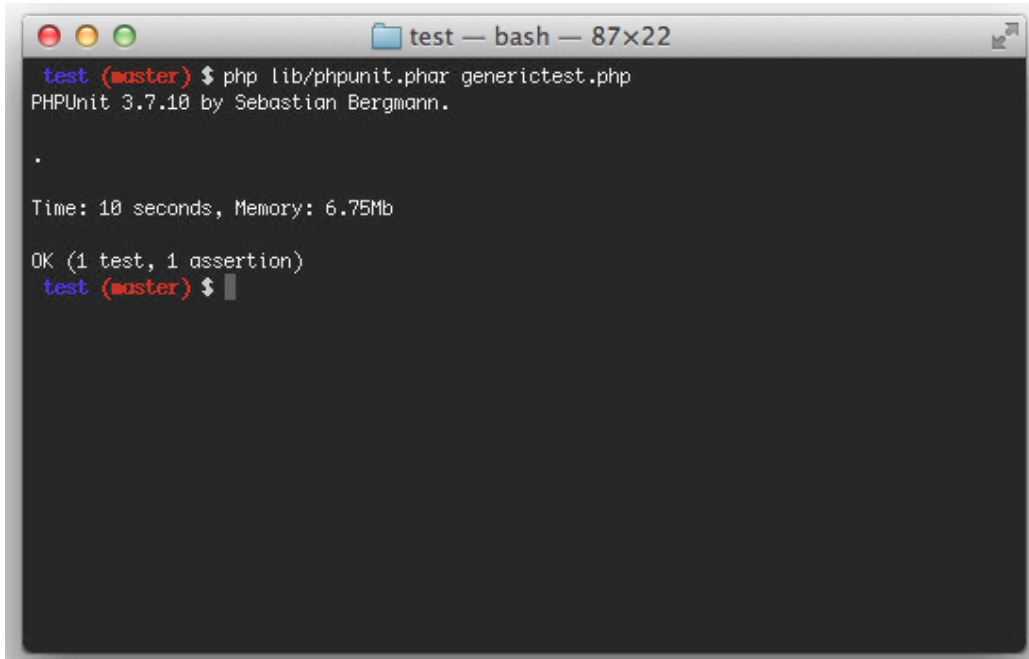
```
public function testForData() {
    $this->_session->open('http://localhost/html5-book/Chapter%2010/');
    sleep(5); //Wait for AJAX data to load
    $result = $this->_session->element("id", "movies-near-me")->text();
    //May need to change settings to always allow sharing of location
    $this->assertGreaterThan(0, strlen($result));
}
```

We will open a connection to our application (feel free to change the URL to wherever you are running your HTML5 application), wait 5 seconds for the initial AJAX to load, and then test for whether the `movies-near-me` div is populated with data.

To run this test, go to the command line and execute the following lines:

```
chmod +x lib/phpunit.phar
php lib/phpunit.phar genericTest.php
```

You should see the following:

A terminal window titled "test — bash — 87x22" with a dark background. The prompt is "test (master) \$". The command "php lib/phpunit.phar generictest.php" has been executed. The output shows "PHPUnit 3.7.10 by Sebastian Bergmann.", a single dot "." representing a passed test, and "Time: 10 seconds, Memory: 6.75Mb". The final output is "OK (1 test, 1 assertion)" followed by the prompt "test (master) \$" with a cursor.

```
test (master) $ php lib/phpunit.phar generictest.php
PHPUnit 3.7.10 by Sebastian Bergmann.

.

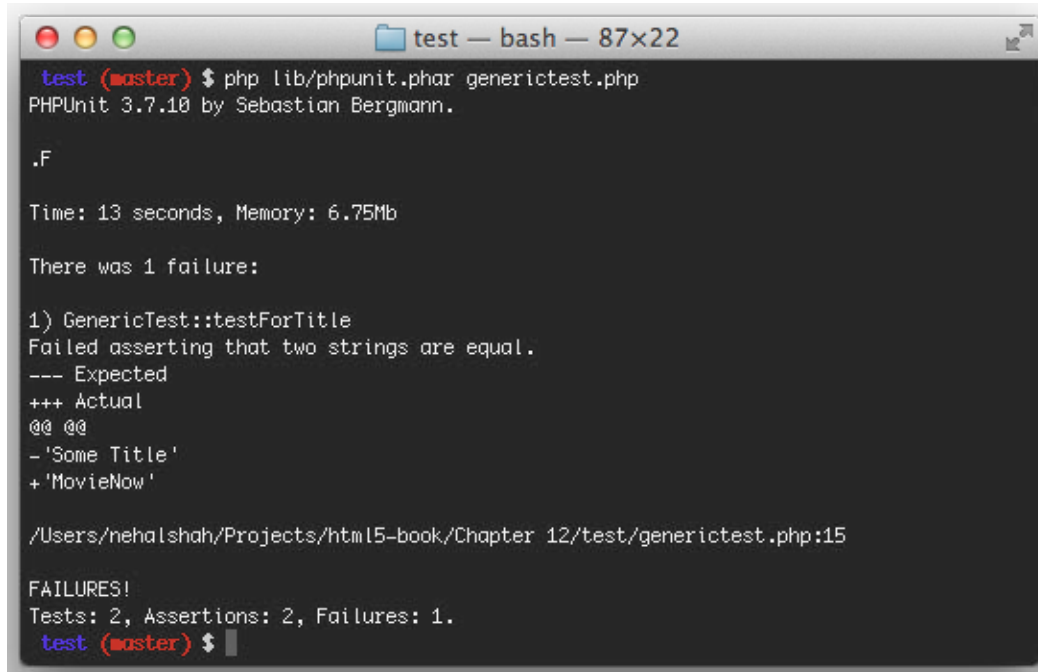
Time: 10 seconds, Memory: 6.75Mb

OK (1 test, 1 assertion)
test (master) $
```

This indicates that the test is passed. Congratulations! Now let us see it fail. Add the following test case:

```
public function testForTitle() {
    $this->_session->open('http://localhost/html5-book/Chapter%2010/');
    $result = $this->_session->title();
    $this->assertEquals('Some Title', $result);
}
```

Rerun PHPUnit and you should see something akin to the following:



```
test (master) $ php lib/phpunit.phar genericTest.php
PHPUnit 3.7.10 by Sebastian Bergmann.

.F

Time: 13 seconds, Memory: 6.75Mb

There was 1 failure:

1) GenericTest::testForTitle
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
- 'Some Title'
+ 'MovieNow'

/Users/nehalsah/Projects/html5-book/Chapter 12/test/genericTest.php:15

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
test (master) $
```


As you can see, it was expecting 'Some Title' but actually found 'MovieNow'. Now that we have gotten you started, we will let you create your own tests. Refer to <http://www.phpunit.de/manual/3.7/en/index.html> for guidance on the different assertions you can make using PHPUnit.

More information about Selenium can be found at <http://seleniumhq.org/>.

Browser testing

With all the talk of browser compatibility in earlier chapters, one pass at testing HTML5 enterprise applications must involve actually looking at the application on different web browsers. Thankfully, many web browsers are offered on multiple platforms. Google Chrome, Mozilla Firefox, and Opera all have versions that will install easily on Windows, Mac OSX, and flavors of Linux such as Ubuntu. Safari has versions for Windows and Mac OSX, and there are ways to install it on Linux with some tweaking.

Nevertheless, Internet Explorer can only run on Windows. One way to work around this limitation is to install virtualization software. Virtualization allows you to run an entire operating system virtually within a host operating system. It allows you to run Windows applications on Mac OSX or Linux applications on Windows. There are a number of notable virtualization packages including VirtualBox, VMWare Fusion, Parallels, and Virtual PC.

 Although Virtual PC runs only on Windows, Microsoft does offer a set of prepackaged virtual hard drives that include specific versions of Internet Explorer for testing purposes. See the following URLs for details: <http://www.microsoft.com/en-us/download/details.aspx?id=11575>.

Another common way to test for compatibility is to use web-based browser virtualization. There are a number of services such as BrowserStack (<http://www.browserstack.com/>), CrossBrowserTesting (<http://crossbrowsertesting.com/>), and Sauce Labs (<https://saucelabs.com/>) that offer a service whereby you can enter a URL and see it rendered in an assortment of web browsers and platforms (including mobile) virtually through the web. Many of them even work through a proxy to allow you to view, test, and debug web applications running on your local machine.

Continuous integration

With any testing solution, it is important to create and deploy your builds and run your tests in an automated fashion. Continuous integration solutions like Hudson, Jenkins, CruiseControl, and TeamCity allow you to accomplish this. They merge code from multiple developers, and run a number of automated functions from deploying modules to running tests. They can be invoked to run on a schedule basis or can be triggered by events such as a commitment of code to a code repository via a post-commit hook.

Summary

We covered several types of testing in this chapter including unit testing, functional testing, and browser testing. For each type of testing, there are many tools to help you make sure that your enterprise application runs in a stable way, most of which we covered bar a few. Because every minute change to your application code has the potential to destabilize it, we must assume that that every change does. To ensure that your enterprise applications remain stable and with minimal defect, having a testing strategy in place with a rich suite of tests – from unit to functional – combined with a continuous integration server running those tests is essential. One must, of course, weigh the investment in time for writing and executing tests against the time needed for writing production code, but the savings in long-term maintenance costs can make that investment worthwhile.

In the next chapter, we will cover techniques to ensure your enterprise application runs at peak performance including a discussion on profiling.

13

Finishing Up: Performance

We will finish off by talking about performance and with good reason. While it is important to think about performance while developing your enterprise application, you may end up optimizing for things that do not exhibit any performance issues later on. This is often referred to as premature optimization and can end up wasting a lot of time. Although it is a good practice to understand performance implications of every decision of the development process, web performance optimization should not be conceived as a final goal; instead it is a constant tuning to improve and reach acceptable speed times for our enterprise application. Our real goal is to build our application and ensure that it functions correctly, then, if it is still necessary, improve the response times.

In this chapter we will cover:


- Web Performance Optimization (WPO)
- Following standards
- Optimizing images
- Optimizing CSS
- JavaScript performance considerations
- Additional page performance considerations
- Performance analytics

Web Performance Optimization (WPO)

Because an HTML5 enterprise application has many different moving parts, it is important to consider which parts you are optimizing. By and large, your HTML5 enterprise application will consist of HTML, images, and CSS and JavaScript code, and there are ways to optimize all three.

Following standards

HTML was developed to be a forgiving language; that is, mistakes in syntax rather than blowing up the page and causing endless debugging nightmares are dealt with in a more graceful manner. The rendering engine attempts to ascertain the intent of the markup and lays out the page accordingly. In essence, it stumbles but manages to keep its footing. Of course, a race run without hurdles goes faster than one run with hurdles. In addition, different web browsers will recover from such errors in different ways leading to inconsistent results when your HTML5 enterprise application is viewed in different browsers. That is why it is important to deliver clean, standards-compliant markup to the browser.

 Despite the fact that following standards is a good base to start our optimizations, this could lead, in some cases, to more verbose code (increasing parsing time). Moreover, HTML5, unlike its previous versions, is not a finished standard yet and should be considered as a guidance more than as a set of rules.

Fortunately, there are many tools out there that will validate your markup for you. The **World Wide Web Consortium (W3C)**, which is the body that develops web standards, has its own validation tools that can be found at <http://validator.w3.org/>. There are also tools such as HTML Lint (<http://lint.brihten.com/html/>) and HTML Tidy (<http://infohound.net/tidy/>) that will clean your markup for you. It is a good practice to validate your markup to make sure your enterprise applications behave quickly and consistently.

Optimizing images

Most websites these days embed images, and often these images are the biggest offenders when it comes to performance. Because of limited bandwidth and the large sizes of image files, your enterprise application could be in fact fairly snappy only to force users to wait while large images are delivered to their browsers. It is key to web optimize your images before using them in your web application.

There are two considerations for web optimization: size and type. With regard to size, while it is possible to set the width and height dimensions of an image in the `img` tag, it is a common mistake to take a single large image and use it for different purposes on a web application that calls for different sizes. For example, when displaying thumbnails, it is a bad practice to size a larger image down using only the `img` tag properties. Instead you should create different variants or renditions of the image for different purposes.



Specifying the `width` and `height` attributes on the `img` tag lets the browser know what real estate to allot an image before the image is actually downloaded, avoiding layout changes and undesired "jumps" in the UI. Be aware that this disagrees with the best practice of separate content and presentation layer to a great extent.

With regard to type, there are indeed three types of images used on the web: GIF, JPEG, and PNG. These are based on different compression algorithms built for very different purposes. GIF images are optimized for low color palette images. They support 256 colors and are lossless and interlaced, which means they are rendered in layers rather than all at once (going from blurry to focused as you download and render them). They are ideal for logos and site graphics based on a lower color palette. JPEG images are ideal for high resolution photos as they support a higher color palette of 16 million colors. PNG images can support 256, 24 bit or 32 bit color palettes image formats with optional transparency, a very flexible and highly compressed lossless format, with superior transparency support and compression than GIF. The PNG algorithm was created as an open alternative to the GIF compression format, whose original creator, Unisys, announced in 1995 that it would be enforcing its patent on the algorithm.



Internet Explorer 6 and previous versions do not support PNG transparency based on HTML standards; instead it is necessary to use proprietary filters. For example, `filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src='image.png',sizingMethod='crop');`.

Optimizing CSS

The best way to optimize your CSS is to reduce the file size. There are a number of things you can do to accomplish this. You can reduce whitespace by placing styles on a single line.

Implement the following line of code:

```
body {background-color:#fff;color:#000;font-size:1.0em;font-family:Arial;}
```

Instead of this:

```
body
{
  background-color:#ffffff;
  color:#000000;
  font-size:1.0em;
  font-family:Arial;
}
```

You can use shorthand for many rules.

Implement the following line of code:

```
p {margin:10px 20px;}
```

Instead of this:

```
p
{
  margin-top: 10px;
  margin-right: 20px;
  margin-bottom: 10px;
  margin-left: 20px;
}
```

It is also best to group similar styles as close together as possible and to combine duplicated styles wherever necessary.

Implement the following line of code:

```
p, ul {color:#efefef;}
```

Instead of this:

```
p {color:#efefef;}
ul {color:#efefef;}
```



If you decide not to follow these recommendations to maintain readability, or even if you follow them, it is always possible to minify your code (as we explain in the following sections).



Another alternative is to use a dynamic stylesheet language such as **Less** (<http://lesscss.org>). Less extends basic CSS functionality, allowing us to use more complex and elegant structures that will be translated in a standard CSS after a compilation process.

With normal CSS you need to repeat common properties such as colors:

```
p{color:#efefef}
div.box{border:1px solid #efefef}
```

Using Less you can define a variable `@active-color` and then use it through your styles, so you only need to change the value of those variables to change the color of multiple properties:

```
@active-color:#efefef;
p{color:@active-color }
div.box{border:1px solid @active-color}
```

You can even create nested structures like so:

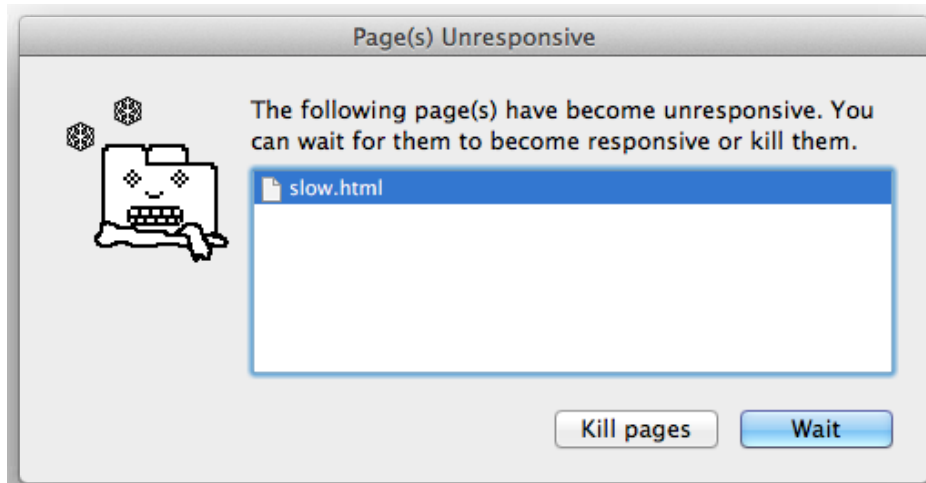
```
@active-color:#efefef;
.content{
  div.box{border:1px solid @active-color}
  p{color:@active-color }
}
```

That will translate after compile in the following:

```
.content div.box{border:1px solid @active-color}
.content p{color:@active-color }
```


JavaScript performance considerations

While writing JavaScript can be fraught with peril as it is very easy to introduce slow running code, keeping some simple guidelines in mind could keep your enterprise application from running too slow.



First of all, traversing the DOM can be expensive. You want to minimize the number of times you invoke `document.getElementById` and, even worse, `document.getElementsByTagName`. Capturing a reference to a DOM element into a variable and using the variable can save on expensive calls to the DOM.

So instead of implementing the following code:

```
document.getElementById("elementId").setAttribute("data-position", 1);
document.getElementById("elementId").setAttribute("data-position", 1);
```

Or implementing its equivalent in jQuery:

```
$("#elementId").attr("data-position", 1);
$("#elementId").attr("data-position", 1);
```

You can implement the following code:

```
var element= document.getElementById("elementId");
element.setAttribute("data-position", 1);
element.setAttribute("data-position", 1);
```

Or you can implement its equivalent in jQuery:

```
var element= $("#elementId");
element.attr("data-position", 1);
element.attr("data-position", 1);
```

Secondly, avoid constructs such as `with()` and `for-in`.

Thirdly, remember that arrays can be slow. Traversing an array – especially a deep array – can be costly. If you are pulling the same element from an array many times, it is better to capture it in a variable first. That is, instead of the following code:

```
var array=[1,2,3,4,5];
console.log(array[3]+2);
console.log(array[3]*3);
```

Or this:

```
var array=[1,2,3,4,5];
var elementSelected=array[3];
console.log(elementSelected+2);
console.log(elementSelected*3);
```

Fourthly, arrays are not as slow as DOM collections. Looping over `document.getElementsByTagName('p')` is much slower than capturing the result in an array and looping over that.

Lastly, changing classes on a DOM element is less expensive than change styles. It is better to define multiple CSS classes and toggle between them than directly changing the style of an element.

So, instead of the following code:

```
domObject.style.display="none";
```

Or its jQuery version:

```
$(domObject).css(display, "none");
```

You can implement the following code:

```
domObject.setAttribute("class", "hideClass");
```

Or this:

```
$(domObject).attr("class", "hideClass");
```

You can also benchmark your JavaScript using a tool called **jsPerf** (<http://jsperf.com/>). It provides a way to create test cases for JavaScript code snippets so that you can benchmark their performance. If you are wondering which is faster, `document.getElementsByTagName` or `document.getElementsByClassName`, this tool will allow you to test your theory on your browser. Furthermore, it allows you to share your test cases so that others can test on different browsers giving you statistics across various browsers and platforms.

Additional page performance considerations

An enterprise application can be composed of many files including HTML, CSS, JavaScript, and images. Although for maintainability, it is proper to break out your CSS and JavaScript files. When deploying your code, combining and minifying your files leads to better performance. **Minification** is a compression technique for code in which all unnecessary characters are removed while the behavior is preserved. There are a number of tools that will do this for you including the following:

- JSMIn (<http://www.crockford.com/javascript/jsmin.html>)
- Packer (<http://dean.edwards.name/packer/>)
- YUI Compressor (<http://developer.yahoo.com/yui/compressor/>)

In the vein of reducing file size, minimizing requests, and in general using as little bandwidth as possible, the use of CSS sprites has become common these days wherein all of the static graphic elements for an application are combined into a single image, parts of which are displayed using CSS. This way only one image needs to be downloaded once instead of many.

Server-side considerations

An enterprise application often has many static assets that are downloaded from the server with each page request. As this creates a lot of unnecessary traffic on the server, one way to offload the burden is to use a content delivery network or CDN. A CDN allows you to mirror your pages on a network of servers that are optimized for delivering static assets quickly. You can place your static assets on a CDN such as Akamai, Edgecast, or Cloudflare as well as use CDN-hosted versions of popular libraries. Google hosts a number of libraries for public consumption such as jQuery in its CDN (<https://developers.google.com/speed/libraries/>) as does cdnjs (<http://cdnjs.com/>).

Many web servers such as Apache can be instructed to compress what they send to the browser before they send it. If you are able to, adding a `Content-Encoding` header set to `gzip` in your response can reduce the amount of data transferred by 70 percent.

It is important to consider caching in your enterprise application. If the same request is made over and over again, and the response is always the same or seldom changes, cache the response and send it back when subsequent requests are made. Browsers intrinsically support this with the `Cache-Control` header and the `Expires` header. While `Expires` tells the browser how long to keep content in the page cache, `Cache-Control` provides a set of rules for when to keep and when to invalidate the cache. Some useful parameters include the following:

- `max-age`: This indicates the maximum amount of time before a piece of content should be refreshed
- `public`: This indicates that a piece of content is cacheable even though it requires authentication
- `private`: This indicates that a piece of content is cacheable on a per-user basis
- `no-cache`: This indicates that a piece of content can be cached but should be refreshed on every request
- `no-store`: This indicates that a piece of content should not be kept in cache
- `must-revalidate`: This indicates that the browser must check with the server first before serving a cached version

Caching not only helps to improve response times, it relieves your server load and reduces network traffic.

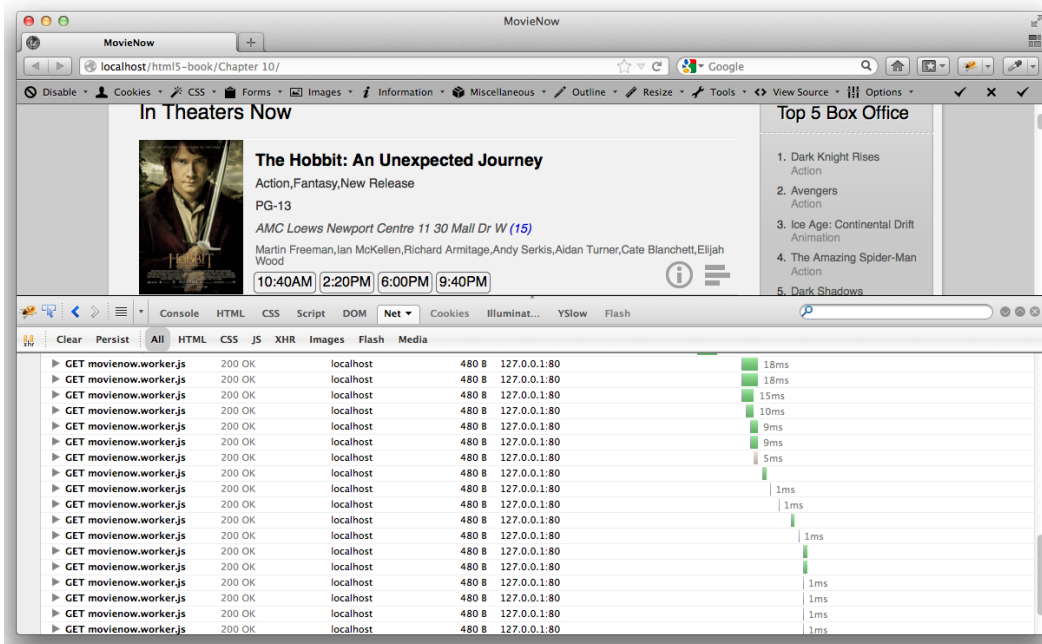
Yahoo! has provided a number of useful guidelines for ensuring page performance. While many of these topics have already been covered, you are encouraged to check out the guidelines for yourself at <http://developer.yahoo.com/performance/>.

Performance analytics

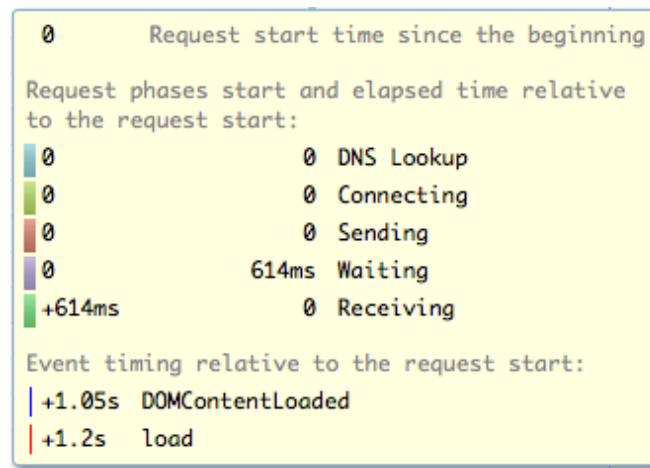
You may yet follow all of the rules for optimal performance and still find yourself with a slow application. When this happens, you need to become adept at assessing load times and profiling your application. Fortunately, there are many tools available to help pinpoint the bottlenecks; these tools are explained in the coming sections.

Load times

All of the major browsers include a network tab that will graphically display all of the requests and responses between the browser and the servers it contacts. Firefox includes a **Net** tab as part of the Firebug add-on as shown in the following screenshot:



It displays a set of bars that indicate load time per request over time. Here you can see the web page loading piece-by-piece including how it loads the HTML first and then requests the ancillary assets afterward: image files, CSS files, JavaScript files, and even subsequent AJAX requests. As you can see in the following screenshot, when you hover over each bar, you will see statistics about that request including DNS lookup, connection time, the time to send the request, wait, and receive the response, and so on.



This is particularly helpful when attempting to debug a performance problem because it tells you very clearly whether slowness is due to a connection problem or a page load problem, for example, how the HTML is constructed and parsed. Knowing these finer details helps greatly to uncover and remove bottlenecks.

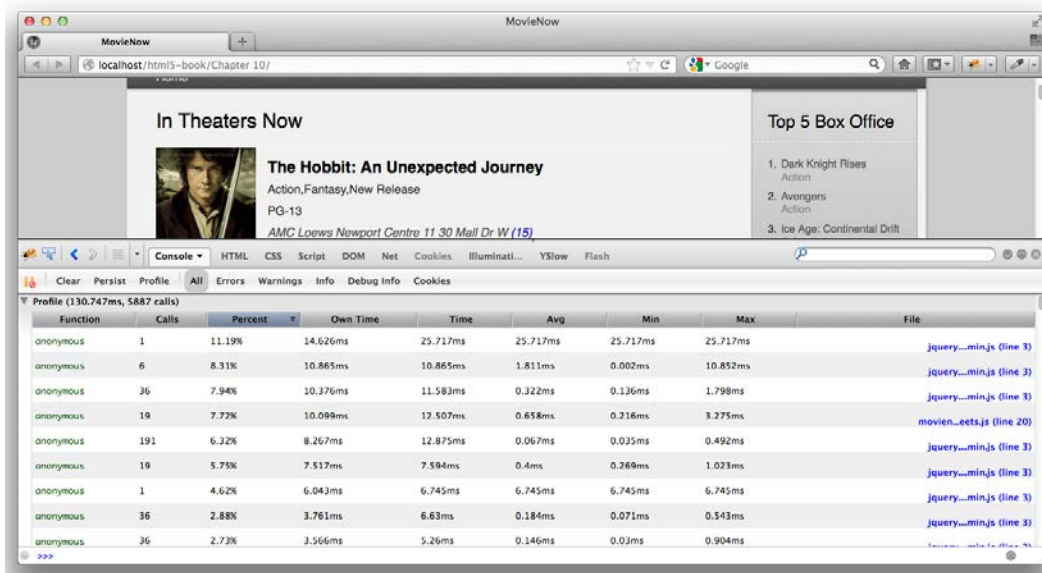
In addition to browser-based tools to assess load time, there are external services that allow you to assess load time. In particular, Pingdom (<http://tools.pingdom.com/fpt/>) provides a service that you can use to periodically test the load times of your web application. It provides a similar chart for determining where your bottlenecks are. Alternatives include the following:

- Google Speed (<https://developers.google.com/speed/>)
- YSlow (<http://developer.yahoo.com/yslow/>)
- Gomez (<http://www.gomez.com/website-performance-test/>)

Profilers

Profilers are another way to assess performance. Typically, they list JavaScript calls in order of execution time, which can be really helpful to pinpoint slow-running functions. Some browsers such as Google Chrome and Safari include a CSS selector profiler, which lists the slowest running CSS selectors.

Firefox has a profiler built into its Firebug add-on. Click on the **Console** tab and click on **Profile**.



In Google Chrome's **Developer Tools**, you will find a **Profiles** tab.

Safari provides an **Instrument** tab.

Internet Explorer 9 offers a profiler as well in its Developer Tools. Click on **Profiler** and then the **Start profiling** button to get started.

Profilers will usually list out the slowest running functions, how many times they were invoked, and the amount of time they took to execute. They typically split the notion of execution time into two categories. Some call it exclusive time while others call it self or own time. This is the execution time while within a function, excluding the execution time of other functions invoked from within that function. The other category is referred to as inclusive time, total time, or just time. This is the execution time of a function including functions that are invoked from within it.

Summary

We covered ways to ensure performance of your enterprise application on many different levels. There are ways to ensure clean HTML that your browser will understand, and there are ways to ensure optimal CSS and JavaScript. There are ways to reduce the page footprint, decrease the number of requests, and place less of a burden on the browser. While each of these is important in their own way, real performance gains are not always apparent until you put them all together.

We have built our example application, *MovieNow*, including the most important steps to build any enterprise application; from the definition of a meaningful structure using semantic tags, to the styling and animations with CSS3, showing new and exciting features such as canvas 2D and WebGL, geolocation, video, audio, drag-and-drop, and web workers. We practised the use of real world APIs such as Twitter, and reviewed a set of tools and libraries to facilitate the process of development, testing, and performance improvement.

We encourage you to continue reading about new web technologies. As we speak, there are many developers who are creating not only new enterprise applications but new libraries, techniques, tools, and paradigms of thought that can redefine the web as we know it.

Finally, we hope you have enjoyed this journey as much as we have, and that the initial guidelines of this book lead you to many successful HTML5 projects.

Index

Symbols

2D context

- about 180
- canvas font styling methods 181
- canvas methods 180
- canvas styling methods 180
- chart design 179
- complex shapes drawing, methods 181, 182
- drawing charts 182-185
- shape drawing, methods 181

3D context

- about 185
- animate3DChart method 196
- geometries, animating 192-196
- geometry 188-191
- material 188
- mesh 188
- textures 188
- Three.js 187
- WebGL specification 186
- window.requestAnimationFrame method,
creating 196

[attribute\$=value] 139

[attribute^=value] 139

[attribute*=value] 139

:checked 139

:disabled 138

:empty 138

:enabled 138

:first-of-type 136

:last-child 137

:last-of-type 136

:not(selector) 138

:nth-child(n) 137

:nth-last-child(n) 137

:nth-last-of-type(n) 137

:nth-of-type(n) 137

:only-child 136

:root 137

:target 138

A

adaptive web design

- about 82
- CSS files importing, media queries used 83
- media queries, using as conditions 83-86
- other CSS, importing 83

add-ons

- about 45
- Firestore 47
- HTML5toggle 47

Adobe Dreamweaver CS6 36, 37

AJAX 99-101

alt attribute 139

anatomy, Web Worker

- about 249, 250
- dedicated Web Worker 249
- shared Web Worker 249

animate3DChart method 196

animation property 132

animations 128-133

Apache 43

Apache Tomcat 44

Aptana Studio 3, 38, 39

async attribute 20

audio player

- custom controller 170
- implementing 169

authenticating
 styles, adding 226-229
 user logged in 224-226
 user not logged in 222-224
authenticating process 220, 221

B

background-color attribute 137
basic styling 77-81
BlueGriffon 1.5.2 40
border-collapse:collapse 72
Browser Mode 51
browsers
 about 18, 19
 JavaScript engine 20, 21
 rendering engine 19
BrowserStack 290
browser testing
 about 289
 continuous integration 290

C

canvas context 178
canvas tag 173
CDN 300
Charles 53
charts
 about 173
 code, preparing 174-178
checkValidity method 240, 242
Chrome Frame 198
Cloud9 42, 43
color setting, MovieNow
 HSL 111
 HSL and alpha 112
 red, green, blue 111
 Red, green, blue, and alpha 111
Colorzilla Gradient Generator 113
compatibility
 defining 17, 18
 importance 24
Console tab 271, 304
context.arcTo() method 182
context.beginPath() method 182
context.bezierCurveTo() method 182

context.clearRect() method 181
context.closePath() method 182
context.fill() method 182
context.fillRect() method 181
context.fillStyle(value) method 180
context.fillText() method 181
context.font(value) method 181
context.isPointInPath() method 182
context.lineCap(value) method 181
context.lineJoin(value) method 181
context.lineTo() method 182
context.lineWidth(value) method 181
context.moveTo() method 182
context.quadraticCurveTo() method 182
context.rect() method 182
context.stroke() method 182
context.strokeRect() method 181
context.strokeStyle(value) method 180
context.strokeText() method 181
CrossBrowserTesting 290
CSS
 optimizing 295-297
 playing with 262-266
CSS3 Pie 108, 110
CSS3 resets
 about 68-70
 individual sides 70
 shorthand 71-73
CSS3 selectors
 applying 135-139
 [attribute\$=value] 139
 [attribute^=value] 139
 [attribute*=value] 139
 :checked 139
 :disabled 138
 element1~element2 139
 :empty 138
 :enabled 138
 :first-of-type 136
 :last-child 137
 :last-of-type 136
 :not(selector) 138
 :nth-child(n) 137
 :nth-last-child(n) 137
 :nth-last-of-type(n) 137
 :nth-of-type(n) 137

- :only-child 136
- :only-of-type 136
- :root 137
- :target 138
- CSS properties**
 - prefix 108
 - syntax 108
- CSS sticky footer.** *See* **sticky footer**
- Curved Corner** 110
- custom controller, audio player**
 - about 170
 - styling 170
- custom controls**
 - about 146, 147
 - buttons, styling 149, 150
 - features, detecting 152
 - fullscreen-button 148
 - image sprites, styling 149, 150
 - interactions adding, JavaScript used 155
 - play-button 148
 - seek 148
 - seek, styling 151, 152
 - sliders, styling 152-155
 - styling 148, 149
 - timer 148
 - volume bars, styling 151, 152
 - volume-container 148
 - volume-slider 148

D

- dataTransfer property** 210
- DebugBar** 262
- debugging**
 - about 259
 - browsers' tools 261, 262
 - tools 260
- Developer Tools | Opera Dragonfly** 52
- disabled property** 138
- displayShowtimes method** 200
- DnD** 199
- Document Mode** 51
- DOMContentLoaded event** 20
- drag**
 - about 204, 205
 - handling, JavaScript used 206, 207
- drag-and-drop functionality.** *See* **DnD**

- dragend event** 206
- dragenter event** 206
- drag events**
 - drag 206
 - dragend 206
 - dragenter 206
 - dragleave 206
 - dragover 206
 - dragstart 206
 - drop 206
- dragleave event** 206
- dragover event** 206
- dragstart event** 206
- dragstart event** 210
- drawBarChart method** 178
- drop event** 211
- drop zone**
 - toggling 209
- dynaTrace** 262

E

- Eclipse engine** 38
- editors**
 - choosing 35, 36
- element1~element2** 139
- elements**
 - data, transferring 210
 - dropping 207, 209
 - drop zone, toggling 209
- errorCallback function** 90
- event listener**
 - updating 253-256
- eXo** 42
- Explorer Canvas**
 - about 31
 - using 31

F

- favico**
 - about 64
 - creating 64-68
- favicon.cc** 64
- feature detection** 27
- Fiddler** 53
- Find Movies button** 96

- Firebug** 46
- Firebug plugin** 261
- Firestore** 47
- formatTime method** 202
- framework, unit testing**
 - Jasmine 283
 - JsTestDriver 280
 - QUnit 282
 - Sinon.JS 282
- functional testing**
 - about 283, 284
 - PHPUnit 286-289
 - php-webdriver connector 286
 - setting up 284
 - standalone server 285

G

- GENERATE! button** 28
- geocodes**
 - capturing 248, 249
- geolocation API**
 - about 88
 - PositionOptions function, attributes 88, 89
 - request 89, 90
 - successCallback function, properties 89
- getChartColor method** 183
- getCurrentPosition() method** 88
- getTweetsByTheater method** 252
- Gomez** 303
- Google Chrome**
 - about 48, 49
 - channels 49
- Google Speed** 303
- graceful degradation** 33

H

- H.264** 21
- HEVC** 142
- hideTweetArea method** 239
- High Efficiency Video Coding.** *See* HEVC
- HLS** 143
- HTML**
 - playing with 262-266
- HTML5**
 - video 141

- HTML5 audio**
 - about 169
 - support 169
- HTML5 Boilerplate**
 - about 32
 - downloading 32
- HTML5 Shim** 25
- HTML5 Shiv**
 - about 25
 - installing 26, 27
- HTML5toggle** 47
- HTML5 video**
 - about 141
 - browser supports 142
- HTML tab** 262
- HTTP Live Streaming.** *See* HLS
- HTTP proxies**
 - about 52
 - Charles 53
 - Fiddler 53

I

- ico file** 64
- icons** 64
- IDEs**
 - about 35
 - Adobe Dreamweaver CS6 36, 37
 - Aptana Studio 3 38, 39
 - BlueGriffon 1.5.2 40
 - choosing 35, 36
 - Cloud9 42, 43
 - eXo 42
 - Maqetta 41
- IE6 funeral** 21
- IIFE** 92
- images**
 - optimizing 294, 295
- immediately invoked function expression.** *See* IIFE
- Initializr**
 - about 32
 - downloading 32
- init method** 207
- input fields types**
 - color 245
 - date 245

- datetime 245
- datetime-local 245
- email 245
- month 245
- number 245
- range 245
- search 245
- tel 245
- time 245
- url 245
- week 245

Instrument tab 304

Integrated Development Environments. *See*
IDEs

interactions, adding to custom controls

- endReproduction 161, 168
- format time 160
- full screen 159, 160
- functions 158
- initial settings 155
- pause 159
- play-button 159
- seek slider, setting 157, 158
- time, controlling 161
- video controllers, initializing 155
- volume slider, initializing 158

Internet Explorer 51

ipconfig getifaddr en1. command 274

J

Jasmine 283

JavaScript

- console tab 269
- load times, analyzing 270
- performance considerations 298-300
- profiling 271
- Script tab 266-268
- used, for drag handling 206, 207
- using 266-268

**JavaScript Object Notation with
Padding.** *See* JSONP

Jetty 44

jQuery library 99

jQuery plugins 245

jQuery tmpl 124

JSTMin 300

JSONP 91

jsPerf 300

JsTestDriver 280, 281

L

layout engine 19

Less 297

LightTPD 45

M

Mac OS X 10.6 36

MAMP 45

Maqetta 41

max-age parameter 301

media queries

- using 133-135

metadata

- about 59-61
- meta tags 60, 61

microdata

- about 62-64
- genre attribute 64
- name attribute 64

Minification 300

mobile debugging 271

Modernizr

- about 27, 28
- using 28-31

MovieNow

- animations 127, 128
- box shadows, adding 115-118
- charting 173
- color, setting 110
- gradients, adding 112-115
- list, styling 125-127
- rounded corners, adding 109, 110
- styles, adding to 109
- styling 122-124
- text shadows, adding 118, 119
- transitions 127
- tricks 120, 121
- showtimes 199-201
- used, for tweeting 220

MovieNow application

- geolocation, adding 90-93
- location, obtaining 94-97

- postal codes, obtaining 97-99
- showtimes 102-106
- Mozilla Firefox**
 - about 46
 - channels 47
- Mustache 124**
- must-revalidate parameter 301**

N

- Net tab 270, 302**
- no-cache parameter 301**
- Node.js 45**
- no-store parameter 301**
- novalidate attribute 245**

O

- objectifyJSON function 254**
- Ogg Theora 21**
- onreadystatechange event 253**
- Opera 51, 52**
- OrthographicCamera 187**
- OS platforms 22**
- overflow:hidden technique 76**

P

- Packer 300**
- page performance**
 - considerations 300
 - server-side considerations 300, 301
- page structure**
 - about 56
 - navigation list 58
 - secondary content 58, 59
 - web applications layout 56-58
- performance analytics**
 - about 301
 - load times 302, 303
 - profilers 303, 304
- PerspectiveCamera 187**
- PHPUnit 286**
- php-webdriver connector 286**
- Pingdom 303**
- PositionOptions function 88**
- prepackaged stacks**
 - MAMP 45

- WAMP 45
- XAMPP 45
- private parameter 301**
- Profilers 303**
- Profiles tab 304**
- progressive enhancement 33**
- public parameter 301**
- Pure 124**

Q

- QUnit 282**

R

- Real Time Messaging Protocol. See RTMP**
- remote debugging**
 - in iOS 6 272
- responsive web design 23**
 - about 82
 - CSS files importing, media queries used 83
 - media queries, using as conditions 83-86
 - other CSS, importing 83
- Responsive Web Design. See RWD**
- results**
 - displaying 211, 213
- reverseGeocode method 100**
- robots meta tag 61**
- RTMP 143**
- RWD 55**

S

- Safari 50, 51**
- Sauce Labs 290**
- screen resolution 23, 24**
- Script tab 266**
- Selenium 289**
- showCharts method 189**
- showDetails method 256**
- showtimes**
 - adding 199-201
 - styling 202, 203
- Sinon.JS 282**
- Snippet Editor 50**
- standards**
 - following 294
- Start profiling button 304**

sticky footer 74-76
Style tab 264

T

testing

about 277
types 277

testing, types

browser testing 289
functional testing 283
unit testing 278

text-indent property 78

text-shadow property 118

Three.js

about 187
camera 187
scene 187

timeupdate event 161

Tornado 44

transitions 128-133

tweeting

MovieNow, using 220

tweet posting

form validation 244, 245
HTML, applying 231, 232
JavaScript interactions, adding 238-243
more styles, adding 232-237
service, calling 229, 230

tweets

nearby-tweets 256
styling 256, 258

Twitter

developer page 216

Twitter AP

registering 216-219

U

Underscore 124

unit testing

about 278
frameworks 280
JSTestDriver 280
results, verifying 280
setting up 278
target, invoking 279, 280
test case 278

updateCount method 239
user experiences (UX) 33

V

video player

custom controls 146
implementing 143-146
improvements 168

W

W3C 17 294

W3C Geolocation API

about 88
rendered mobile devices 88
supported browsers 88
working 88

WAMP

watchPosition() method 88

web browser engine 19

web debugging

proxies 273, 274

WebGL

enabling 198

Web Inspector option 272

WebM 21

Web Performance Optimization. *See* WPO

web servers

about 45
Apache 43
choosing 43
Firebug 46
Google Chrome 48, 49
Internet Explorer 51
Jetty 44
LightTPD 45
Mozilla Firefox 46
Node.js 45
Opera 51, 52
Safari 50, 51
Tornado 44

Web Worker

about 247
anatomy 249
event listener, updating 253-256
using, for nearby tweet obtaining 251-253

World Wide Web Consortium. *See* W3C
WPO 293

X

XAMPP 45

Y

Yahoo User Interface (YUI) 69
YSlow 303



Thank you for buying **HTML5 Enterprise Application Development**

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

About Packt Enterprise

In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

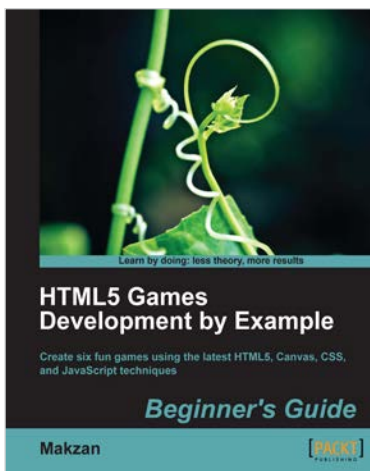


Responsive Web Design with HTML5 and CSS3

ISBN: 978-1-849693-18-9 Paperback: 324 pages

Learn responsive design using HTML5 and CSS3 to adapt websites to any browser or screen size

1. Everything needed to code websites in HTML5 and CSS3 that are responsive to every device or screen size
2. Learn the main new features of HTML5 and use CSS3's stunning new capabilities including animations, transitions and transformations
3. Real world examples show how to progressively enhance a responsive design while providing fall backs for older browsers



HTML5 Games Development by Example: Beginner's Guide

ISBN: 978-1-849691-26-0 Paperback: 352 pages

Create six fun games using the latest HTML5, Canvas, CSS, and JavaScript techniques

1. Learn HTML5 game development by building six fun example projects
2. Full, clear explanations of all the essential techniques
3. Covers puzzle games, action games, multiplayer, and Box 2D physics
4. Use the Canvas with multiple layers and sprite sheets for rich graphical games

Please check www.PacktPub.com for information on our titles



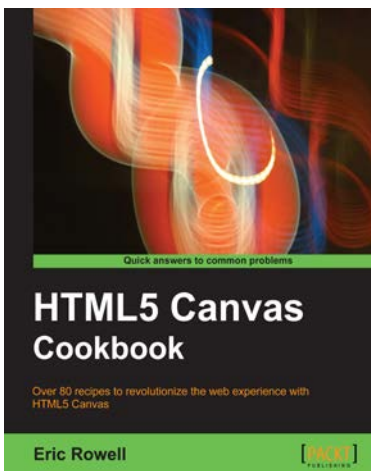
HTML5 Mobile Development Cookbook

ISBN: 978-1-849691-96-3

Paperback: 254 pages

Over 60 recipes for building fast, responsive HTML5 mobile websites for iPhone 5, Android, Windows Phone, and BlackBerry

1. Solve your cross platform development issues by implementing device and content adaptation recipes.
2. Maximum action, minimum theory allowing you to dive straight into HTML5 mobile web development.
3. Incorporate HTML5-rich media and geo-location into your mobile websites.



HTML5 Canvas Cookbook

ISBN: 978-1-849691-36-9

Paperback: 348 pages

Over 80 recipes to revolutionize the web experience with HTML5 Canvas

1. The quickest way to get up to speed with HTML5 Canvas application and game development
2. Create stunning 3D visualizations and games without Flash
3. Written in a modern, unobtrusive, and objected oriented JavaScript style so that the code can be reused in your own applications.

Please check www.PacktPub.com for information on our titles