



Community Experience Distilled

Learning JavaScriptMVC

Learn to build well-structured JavaScript web applications using JavaScriptMVC

Wojciech Bednarski

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Learning JavaScriptMVC

Learn to build well-structured JavaScript web applications using JavaScriptMVC

Wojciech Bednarski

[PACKT] open source 
PUBLISHING community experience distilled
BIRMINGHAM - MUMBAI

Learning JavaScriptMVC

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 author, 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: May 2013

Production Reference: 1140513

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

ISBN 978-1-78216-020-5

www.packtpub.com

Cover Image by Will Kewley (william.kewley@kbbs.ie)

Credits

Author

Wojciech Bednarski

Project Coordinator

Michelle Quadros

Reviewer

Juri Strumpflohner

Proofreader

Elinor Perry-Smith

Acquisition Editor

Mary Nadar

Indexer

Monica Ajmera Mehta

Commissioning Editor

Priyanka Shah

Production Coordinator

Pooja Chiplunkar

Technical Editors

Kirti Pujari

Lubna Shaikh

Nitee Shetty

Jalasha D'costa

Cover Work

Pooja Chiplunkar

About the Author

Wojciech Bednarski is a software engineer with expert knowledge of client-side technologies. He is passionate about JavaScript, Node.js, HTML5, Ruby, NoSQL, and POSIX-compliant systems.

While at university he started taking up freelance jobs and was obsessed by web accessibility and usability as well as web standards.

Then, he moved to Warsaw where he started working as a web developer at eo Networks, which is recognized as one of the 50th fastest growing company in Central Europe.

He then started work at Roche, one of the largest pharmaceutical companies in the world, where he worked on large scale web-based systems as well as conducted workshops and technical seminars. He was recognized with an Informatics Service Award in the category of Innovation.

He then moved to Copenhagen and started work at YouSee, the subsidiary of TDC, the biggest Danish telecom company, where he programmed set top boxes. He won Copenhagen Startup Weekend and also began the Everplaces startup.

At the time of writing this book, he is a consultant for a New York-based company working on the next big thing you will use. He works from different places and lives with his beautiful wife and two black cats. He also loves taking pictures, you can have a sneak peek at www.pixmod.net. He is also fond of driving sports cars and traveling.

You can visit his professional profile at www.linkedin.com/in/bednarski/ or you can follow him on Twitter @wbednarski.

About the Reviewer

Juri Strumpflohn currently works as a software architect for an e-government company where he helps create appealing rich client web applications with HTML5, JavaScript, and the .NET technology stack. Besides that, he is an active blogger, writing about web and mobile development topics and promoting best practice. He also actively participates in online communities such as StackOverflow. When he is not in front of his computer then he is probably practicing Yoseikan Budo where he currently owns a 2nd DAN Black Belt. He has a MSc degree in Computer Science from the Free University of Bolzano, Italy.

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.

Table of Contents

Preface	1
Chapter 1: Getting Started with JavaScriptMVC	7
What is JavaScriptMVC?	7
License	8
Links	8
Why JavaScriptMVC?	8
System architecture approach	9
JavaScriptMVC single-page application	10
Advantages	10
Downsides	10
Real-world examples	10
Installing JavaScriptMVC	11
Choosing your method	11
Which method is right for me?	11
The first method – download the package	11
The second method – pull the code from Git repositories	12
The third method – vagrant	14
Documentation and API	14
The architecture of JavaScriptMVC	15
DocumentJS	15
FuncUnit	15
jQueryMX	16
StealJS	16
Dependency management	16
Concatenation and compression	16
Logger	17
Code generator	17
Package management	17
Code cleaner	17

Building simple applications	17
Todo list	17
Loader	18
Model	19
Fixtures	22
View	25
Controller	26
Routing	27
Complete application code	28
Summary	31
Chapter 2: DocumentJS	33
How does DocumentJS work?	34
Writing the documentation	34
Type directives	38
Tag directives	39
Generating the documentation	40
Summary	41
Chapter 3: FuncUnit	43
Creating tests	44
Module	45
Open	45
Test	45
Ok	46
S	46
Running tests	46
Web browser	46
Selenium	47
PhantomJS	47
EnvJS	48
Integration	48
Summary	48
Chapter 4: jQueryMX	49
\$.Class	50
The first parameter	52
The second parameter	52
The third parameter	52
Method override	52
Life cycle	53
\$.Model	53
\$.View	55
Embedded	55
External	56

Sub-templates	56
\$.Controller	57
DOM helpers	57
\$.cookie	58
\$.fn.compare	58
\$.fn.selection	59
\$.fn.within	60
\$.Range	60
\$.route	61
Special events	61
\$.Drag and \$.Drop	62
Language helpers	62
\$.Object	62
same	62
\$.Observe	64
\$.String	65
deparam	65
\$.toJSON	66
\$.Vector	66
Summary	67
Chapter 5: StealJS	69
Dependency management	69
Logger	70
Code cleaner	71
Concatenation and compression	71
Summary	72
Chapter 6: Building the App	73
Time tracking and invoicing for freelancers	73
Planning	74
Preparing wireframes	75
Setup project	78
Tracking changes under VCS	79
Application structure	80
IndexedDB	81
Creating models	81
Creating controllers	89
Creating views	95
Creating a bootstrap	102
Running the application	103
Summary	104
Index	105

Preface

Learning JavaScriptMVC will guide you through all the framework aspects and show you how to build small- to mid-size, well-structured and documented client-side applications that you will love working on.

What this book covers

Chapter 1, Getting Started with JavaScriptMVC, provides an overview of the JavaScriptMVC framework. Install it, go over the architecture, and learn how to do it in the best possible way – by building a simple application.

Chapter 2, DocumentJS, shows how, despite being powerful, DocumentJS is a simple tool designed to easily create searchable documentation of any JavaScript codebase.

Chapter 3, FuncUnit, explains how FuncUnit is a functional testing framework with jQuery-like syntax. Using FuncUnit, we can run tests in all modern web browsers. Writing test is really easy and fast.

Chapter 4, jQueryMX, shows how jQueryMX is a collection of jQuery libraries that provides the functionality necessary to implement and organize large JavaScript applications. It provides classical inheritance simulation and a model-view-controller layer to provide logically separated codebase.

Chapter 5, StealJS, shows that StealJS is an independent code manager and build tool.

Chapter 6, Building the App, shows how to build real-world applications from concept through design, implementation, documentation, and testing.

What you need for this book

To run the examples in this book the following software will be required:

- **JavaScriptMVC kick-starter:** https://github.com/wbednarski/JavaScriptMVC_kick-starter
- **Oracle VM VirtualBox:** <https://www.virtualbox.org/>
- **Vagrant:** <http://downloads.vagrantup.com/>

Who this book is for

This book is for anyone who is interested in developing small- and mid-size web applications with the JavaScriptMVC framework, which is based on the most popular JavaScript library – jQuery.

Readers should be familiar with JavaScript, browser APIs, jQuery, HTML5, and CSS.

Conventions

In this book, you will find different 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: "Easy switch to another version by the checkout tag."

A block of code is set as follows:

```
<!doctype html>

<html>
  <head>
    <title>Todo List</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <ul id="todos">
      <li>all done!</li>
    </ul>

    <script src="../steal/steal.js?todo"></script>
  </body>
</html>
```


When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
steal(  
  'jquery/class',  
  'jquery/model',  
  'jquery/dom/fixture',  
  'jquery/view/ejs',  
  'jquery/controller',  
  'jquery/controller/route',  
  
  function ($) {  
  
  }  
);
```

Any command-line input or output is written as follows:

```
$ git submodule add git://github.com/bitovi/steal.git  
$ git submodule add git://github.com/bitovi/documentjs.git  
$ git submodule add git://github.com/bitovi/funcunit.git  
$ git submodule add git://github.com/jupiterjs/jquerymx jquery
```

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: "**Archive** button is visible when task is hovered."

 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 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.

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/submit-errata>, 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

Getting Started with JavaScriptMVC

In this chapter, get an overview of the JavaScriptMVC framework. We will install it, go through the architecture, and learn it in the best possible way. Finally, we will build a simple application. There is nothing that works better than an example. Some say this is the only thing that works.

What is JavaScriptMVC?

JavaScriptMVC (JMVC) is a JavaScript open source **model-view-controller** (MVC) framework build, on top of the jQuery library.

It is the backend agnostic client-side framework that can be used with any backend solution, such as Node.js, Ruby on Rails, Django, and so on.

The idea behind JavaScriptMVC is to provide a set of tools to build high quality and maintainable applications in the shortest amount of time possible.

JavaScriptMVC contains the following independent components:

- **StealJS**: This is the dependency manager and production build
- **FuncUnit**: This is the unit and functional test component
- **jQueryMX**: This contains a set of plugins that provide the functionality to implement and organize large JavaScript codebases into a well-structured and organized form, provide a model-view-controller abstraction layer
- **DocumentJS**: This is the documentation

The first version was published in May 2008. Current Version 3.2 was released in December 2010. The latest version at the time of writing this book is 3.2.2.

In the next Version 3.3 of JavaScriptMVC, which should be released soon, jQueryMX project will be replaced by CanJS. Projects using current version of JMVC should work after small refactoring with JMVC 3.3 thanks to the names fallback.

JavaScriptMVC 4.0 will be renamed to DoneJS and contain significant changes to StealJS which will be fully AMD compatible work with CommonJS and run with Node.js. FuncUnit will be split into 3 parts: Syn - Synthetic event library, ShouldJS - Asynchronous test driving using Jasmine or QUnit and DidJS - Automated test runner bindings for Jasmine or QUnit for Selenium, PhantomJS, and so on.

License

JavaScriptMVC is licensed under the MIT license with the following exceptions:

- Rhino: This is the JavaScript command line (MPL 1.1)
- Selenium browser automation (Apache 2)

Links

You can refer to the following URLs to learn more about JavaScriptMVC:

- Official website: <http://javascriptmvc.com>
- Repository: <https://github.com/bitovi/javascriptmvc>

Why JavaScriptMVC?

JavaScriptMVC is a solid and well documented framework.

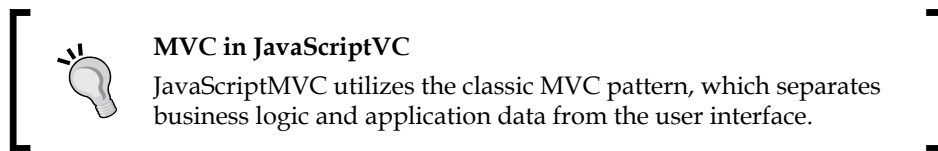
It is based on the extremely popular JavaScript library **jQuery**, where many JavaScript programmers are familiar with its factory methods and chainable function style.

JavaScriptMVC is a complete package. It contains everything we need to build, manage, document, and test JavaScript projects.

Since it is a modular framework, we don't need to use all the available components. We can start by using only framework components that we actually need, and add additional components as and when we need them.

The learning curve is pretty low, especially if a reader is familiar with other JavaScript frameworks, such as lightweight Backbone and Sammy or heavyweight toolkits such as Dojo toolkit or Google Closure. At the same time, it offers much more than lightweight brothers without a heavy feel, such as Google Closure which produces much cleaner code and provides better documentation than the very popular Dojo toolkit.

One of its killer features is that it prevents memory leakage. This is a very important aspect of client-side applications, which perform many operations on the **Document Object Model (DOM)** tree.



System architecture approach

When building web applications, we can distinguish between two approaches – multi-page application and single-page application.

In **multi-page application**, most of the business logic is implemented in the backend system, with some enhancement done in JavaScript. For example, the Ruby on Rails application, where most of the main logic is done by the backend MVC architecture and when a user navigates to another page, an ordinary `http` request is sent.

In **single-page application**, most of the business logic is implemented on the frontend side. For example, the JavaScriptMVC application, where most of the main logic is done by frontend MVC architecture. When a user navigates to another page, the frontend router dispatches all requests and makes calls to the back end API written; for example, in Sinatra.

JavaScriptMVC single-page application

JavaScriptMVC is designed for single-page application use cases. It's good to know about the advantages and disadvantages of the single-page application approach compared to that of the multi-page application.

Advantages

- Most of the states are maintained in the client, so we don't need to keep the session states on the server side
- Most of the requests are done through XMLHttpRequest calls, so there is no need to load a new page each time, which could cause high memory footprint (especially in the old fashion, non event-based servers such as Apache)
- Most of the business logic is on the client side, so we can save many calls to the server

Downsides

- Load balance and **Content Delivery Network (CDN)** can be tricky since RPC is used to move data back and forth between the server and client.
- **Search Engine Optimization (SEO)** can be tricky due to on-demand JavaScript built pages.

Real-world examples

Readers can find web applications built with the JavaScriptMVC framework at <http://community.javascriptmvc.com/posts/in-bucket/apps>.

Installing JavaScriptMVC

Installing JavaScriptMVC is as easy as making tea, but faster.

Choosing your method

There are three methods.

- Download the complete package from the official website (<http://javascriptmvc.com>) or build a custom package including the components we want to use (<http://javascriptmvc.com/builder.html>)

- Pull code from the Git repositories hosted on GitHub.
- Use Vagrant

The last two methods are the preferred way, for the following reasons:

- Easy update to the latest version
- Easy switch to another version by the `checkout` tag
- Contribution to the project; how awesome is that? For more information about contributing, visit <http://javascriptmvc.com/docs.html#!developwithgit>

The third method seems to be the best one, because it contains all the advantages from the second one, plus it creates an encapsulated environment, which we can easily and quickly create or delete without affecting our current development environment setup.

Which method is right for me?

For a fast tryout library, choose the first method. For the actual development, definitely choose the second one.

The first method – download the package

In this method, we will use a web interface on the JavaScriptMVC web page to configure and download the package:

1. Download the complete package from <http://javascriptmvc.com> and unpack its content.
2. Create a folder named `Todo` under the local web server working directory.
3. Copy all files from `javascriptmvc-3.2.2` to the `Todo` folder and start the web server.

```
$ mkdir Todo && cp -r javascriptmvc-3.2.2/* Todo && cd Todo
```

That is it; we are all set and ready to go.

The second method – pull the code from Git repositories

We assume that the reader knows and has installed Git.

If not, the following resources might be helpful:

- **Installing Git:** <http://git-scm.com/book/en/Getting-Started-Installing-Git>
- **Free book Pro Git:** <http://git-scm.com/book>
- **Git reference:** <http://gitref.org>

In the following steps, we are going to install JavaScriptMVC for our `Todo` example project:

1. Under local web server directory, create new folder named `Todo`:

```
$ mkdir Todo && cd Todo
```
2. Inside the `Todo` folder, create a new Git repository:

```
$ git init
```
3. Add JavaScriptMVC components as submodules to the project:

```
$ git submodule add git://github.com/bitovi/steal.git  
$ git submodule add git://github.com/bitovi/documentjs.git  
$ git submodule add git://github.com/bitovi/funcunit.git  
$ git submodule add git://github.com/jupiterjs/jquerymx jquery
```
4. Install and update the submodules:

```
$ git submodule init  
$ git submodule update
```

5. The last module we need to install is `syn`. Since it is already a submodule to the `FuncUnit` project, all we need to do is initialize and update it:

```
$ cd funcunit  
  
$ git submodule init  
  
$ git submodule update
```

6. Switch `syn` to the `master` branch:

```
$ cd syn/  
  
$ git checkout master
```

7. Go back to the root directory of the project:

```
$ cd ../../
```

8. Move the `js` command to the root directory of the project:

```
$ ./steal/js steal/make.js
```

Verifying Installation

The project directory should have following folder structure:

```
.git  
.gitmodules  
documentjs  
funcunit  
jquery  
js  
js.bat  
steal
```

That is it; we are all set and ready to go.



More about submodules in Git: <http://git-scm.com/book/en/Git-Tools-Submodules>

The third method – Vagrant

To install JavaScriptMVC using this method, we need to install **Vagrant**, which is a virtualized development tool wrapper around Oracle VM VirtualBox, an x86 and AMD64/Intel64 virtualization software package.

1. Download and install Oracle VM VirtualBox (<https://www.virtualbox.org>).
2. Download and install Vagrant (<http://downloads.vagrantup.com>).
3. Download and unpack the JavaScriptMVC kick-starter (https://github.com/wbednarski/JavaScriptMVC_kick-starter/archive/master.zip).
4. Inside JavaScriptMVC kick-starter folder type `vagrant up`.

This command creates a virtual environment and a projects directory. It also installs the web server. JavaScriptMVC framework will be placed in the `Todo` directory.

Any changes we make inside the projects directory are immediately visible in web browser at <http://192.168.111.111/>.

Documentation and API

Good documentation and API, many tutorials, and a well documented codebase is the strong side of JavaScriptMVC:

- JavaScriptMVC documentation: <http://javascriptmvc.com/docs.html>
- JavaScriptMVC API: <http://jqapi.com>
- JavaScriptMVC tutorials: <http://javascriptmvc.com/docs.html#!tutorials>
- JavaScriptMVC code examples: <http://javascriptmvc.com/docs.html#!examples>

Active community on the forum and Stack Overflow:

- Stack Overflow questions about JavaScriptMVC: <http://stackoverflow.com/questions/tagged/javascriptmvc>
- JavaScriptMVC official forum: <http://forum.javascriptmvc.com/allforums>

The architecture of JavaScriptMVC

The architecture of JavaScriptMVC is modular. The powerful stack contains everything we need to build a well organized, tested, and documented application.

Here is a list of the JavaScriptMVC key components as well as topics covered in the next chapters.

DocumentJS

DocumentJS is an independent JavaScript documentation application and provides the following:

- Inline demos with source code and HTML panels
- Adds tags to the documentation
- Adds documentation as favorite
- Auto suggest search
- Test result page
- Comments
- Extends the JSDoc syntax
- Adds undocumented code because it understands JavaScript

FuncUnit

FuncUnit is an independent web testing framework and provides the following:

- Test clicking, typing, moving mouse cursor, and drag-and-drop utility
- Follows users between pages
- Multi browser and operating system support
- Continuous integration solution
- Writes and debugs tests in the web browser
- Chainable API that parallels jQuery

jQueryMX

jQueryMX is the MVC part of JavaScriptMVC and provides the following:

- Encourages logically separated, deterministic code
- MVC layer
- Uniform client-side template interface (supports jq-tmpl, EJS, JAML, Micro, and Mustache)
- Ajax fixtures
- Useful DOM utilities
- Language helpers
- JSON utilities
- Class system
- Custom events

StealJS

StealJS is an independent code manager and build tool and provides the following powerful features:

Dependency management

- Loads JavaScript and CoffeeScript
- Loads CSS, Less, and Sass files
- Loads client-side templates such as TODO
- Load individual files only once
- Loads files from a different domain

Concatenation and compression

- Google Closure compressor
- Makes multi-page build
- Pre processes TODO
- Can conditionally remove specified code from the production build
- Builds standalone jQuery plugins

Logger

- Logs messages in a development mode

Code generator

- Generates an application skeleton
- Adds the possibility to create your own generator

Package management

- Downloads and install plugins from SVN and Git repositories
- Installs the dependencies
- Runs install scripts
- Loads individual files only once
- Loads files from a different domain

Code cleaner

- Runs JavaScript beautifier against your codebase
- Runs JSLint against your codebase

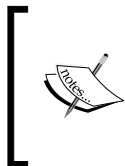
Building simple applications

We installed JavaScriptMVC and went briefly through its components. Now, we are ready to build our first JavaScriptMVC application.

Excited? Let's do the magic.

Todo list

We are going to learn JavaScriptMVC on the classic example application – the to-do list.



If you are curious and want to compare different JavaScript frameworks based on the `todos` application examples, then the GitHub project is absolutely fantastic. You can find it at <https://github.com/tastejs/todomvc/tree/gh-pages/architecture-examples>. The project home page is at <http://todomvc.com/>.

Loader


In the `Todo` folder that we created during installing JavaScriptMVC, create a folder named `todo`. Create files named `todo.html` and `todo.js` inside `todo`.

The project directory should have following structure:

```
Todo/  
  .git  
  .gitmodules  
  todo/  
    todo.html  
    todo.js  
  documentjs  
  funcunit  
  jquery  
  js  
  js.bat  
  steal
```

Copy and paste the following code into `todo.html` to load the `StealJS` and `todo.js` files:

```
<!doctype html>  
  
<html>  
  <head>  
    <title>Todo List</title>  
    <meta charset="UTF-8" />  
  </head>  
  <body>  
    <ul id="todos">  
      <li>all done!</li>  
    </ul>  
  
    <script src="../steal/steal.js?todo"></script>  
  </body>  
</html>
```

 `../steal/steal.js?todo` is the equivalent of `../steal/steal.js?todo/todo.js`. If file name is not provided StealJS, try to load the JavaScript file with the same name as the given folder.

In `todo.js`, add the following code to load the jQueryMX plugins. They are necessary to implement this application:

```
steal(
  'jquery/class',
  'jquery/model',
  'jquery/dom/fixture',
  'jquery/view/ejs',
  'jquery/controller',
  'jquery/controller/route',

  function ($) {

  }

);
```

Open the page in a web browser by typing `http://YOUR_LOCAL_WEB_SERVER/ToDo/todo.html`, and use a web development tool, such as Google Chrome Inspector, to check if StealJS and all the listed plugins are loaded properly.

Model

The next step is to add a model to our application by extending `$.Model` from the jQueryMX project.

The first parameter is the model name (string), the second parameter is the object with the class properties and methods. The last parameter is the prototype instance property, which we leave as an empty object for this example:

```
steal(
  'jquery/class',
  'jquery/model',
  'jquery/dom/fixture',
  'jquery/view/ejs',
  'jquery/controller',
  'jquery/controller/route',
```

```
function ($) {  
  $.Model('Todo', {  
    findAll: 'GET /todos',  
    findOne: 'GET /todos/{id}',  
    create: 'POST /todos',  
    update: 'PUT /todos/{id}',  
    destroy: 'DELETE /todos/{id}'  
  },  
  {  
  }  
);  
}
```



Class properties are not random; they are described in the model API.
<http://javascriptmvc.com/docs.html#!jquerymx>.

We've created the `Todo` model for our `todo` list application. Now, it's time to play around with it.

1. Open a web browser and type the following line into the JavaScript console:

```
var todo = new Todo({name: 'write a book'});
```

`todo` is now an instance of `Todo` with property `name` and property value `write a book`.

2. Get the property value as follows:

```
todo.attr('name');
```

3. Set the property value if the property exists, as follows:

```
todo.attr('name', 'write JavaScript book');
```

Or by `attrs`, where we can set more than one property at the time as well as add a new property:

```
todo.attrs({name: 'write JavaScriptMVC book!'});
```

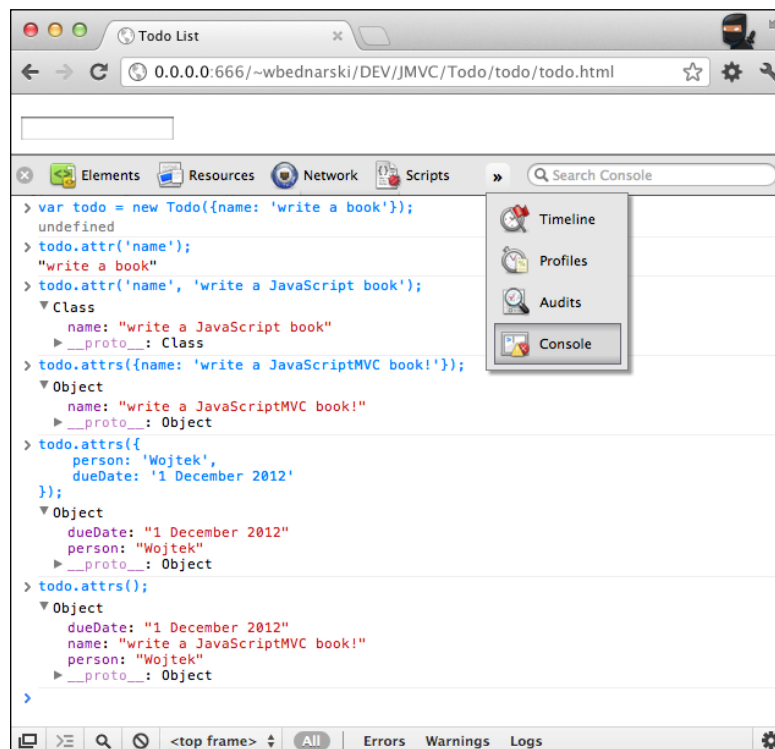
4. Add two new properties:

```
todo.attrs({
  person: 'Wojtek',
  dueDate: '1 December 1012'
});
```

5. List all the properties:


```
Todo.attrs();
```

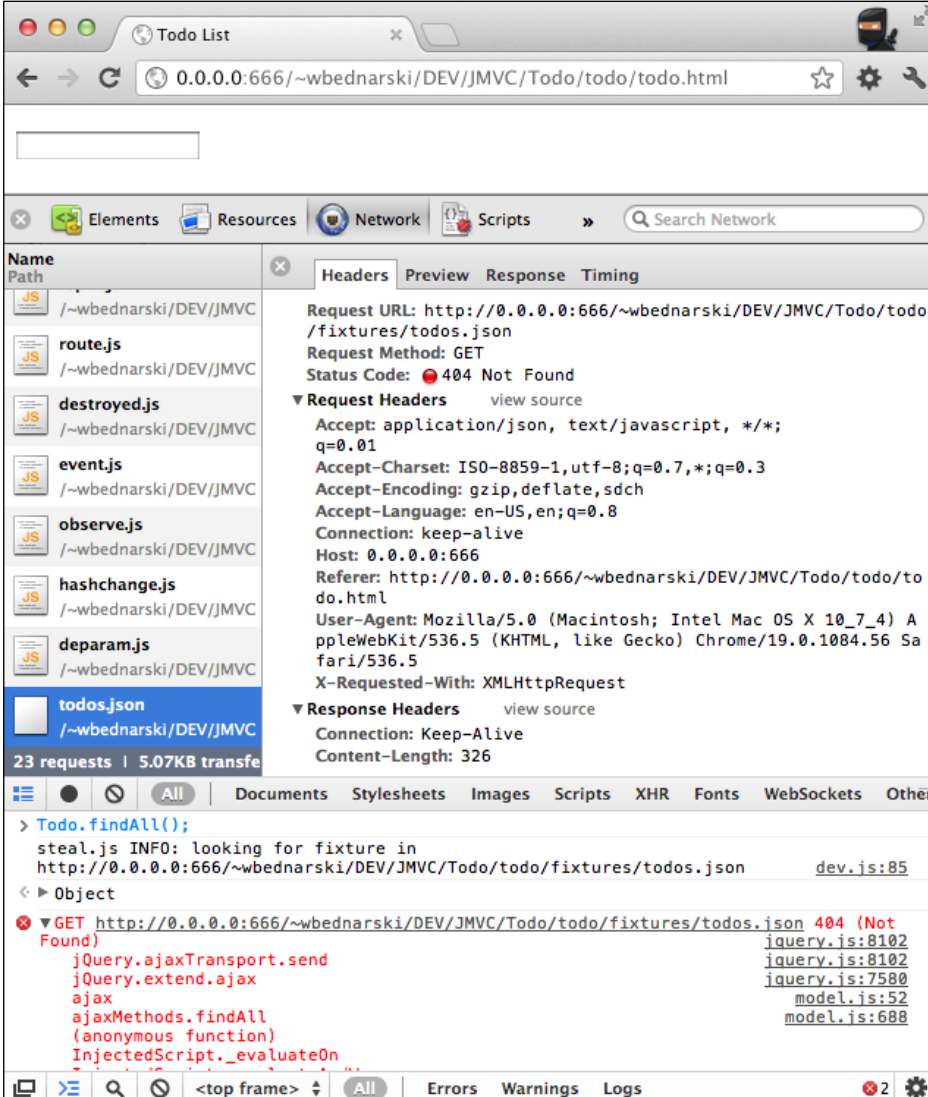
The following screenshot shows the execution of the preceding commands:



Fixtures

Since we have no backend service to handle /todo API calls in our frontend application, any attempt to invoke one of the model's CRUD methods on the `Todo` model will cause a network error.

 **Create, Read, Update, Delete (CRUD)** are the four basic functions of persistent storage.



The screenshot shows a web browser window with the address bar displaying `0.0.0.0:666/~wbednarski/DEV/JMVC/Todo/todo/todo.html`. The browser's developer tools are open, showing the Network panel with a list of requests. The selected request is a GET request to `http://0.0.0.0:666/~wbednarski/DEV/JMVC/Todo/todo/fixtures/todos.json`, which has failed with a status code of 404 Not Found. The console shows the following error:

```
> Todo.findAll();
steal.js INFO: looking for fixture in
http://0.0.0.0:666/~wbednarski/DEV/JMVC/Todo/todo/fixtures/todos.json dev.js:85
< Object
✖ GET http://0.0.0.0:666/~wbednarski/DEV/JMVC/Todo/todo/fixtures/todos.json 404 (Not Found)
  jquery.ajaxTransport.send jquery.js:8102
  jquery.extend.ajax ajax jquery.js:7580
  ajaxMethods.findAll (anonymous function) model.js:52
  InjectedScript._evaluateOn injectedScript.js:688
```

At this point, `$.fixture` comes to the rescue. With this feature, we can work on a project even when backend code is not ready yet.

Create fixtures for the `Todo` model:

```
steal(
  'jquery/class',
  'jquery/model',
  'jquery/util/fixture',
  'jquery/view/ejs',
  'jquery/controller',
  'jquery/controller/route',

  function ($) {
    $.Model('Todo', {
      findAll: 'GET /todos',
      findOne: 'GET /todos/{id}',
      create: 'POST /todos',
      update: 'PUT /todos/{id}',
      destroy: 'DELETE /todos/{id}'
    },
    {
    }
  );

  // Fixtures
  (function () {
    var TODOS = [
      // list of todos
      {
        id: 1,
        name: 'read The Good Parts'
      },
      {
        id: 2,
        name: 'read Pro Git'
      },
      {
        id: 3,
        name: 'read Programming Ruby'
      }
    ];

    // findAll
    $.fixture('GET /todos', function () {
      return [TODOS];
    });

    // findOne
```

```
$.fixture('GET /todos/{id}', function (orig) {
  return TODOS[(+orig.data.id) - 1];
});

// create
var id = 4;
$.fixture('POST /todos', function () {
  return {
    id: (id++)
  };
});

// update
$.fixture('PUT /todos/{id}', function () {
  return {};
});

// destroy
$.fixture('DELETE /todos/{id}', function () {
  return {};
});

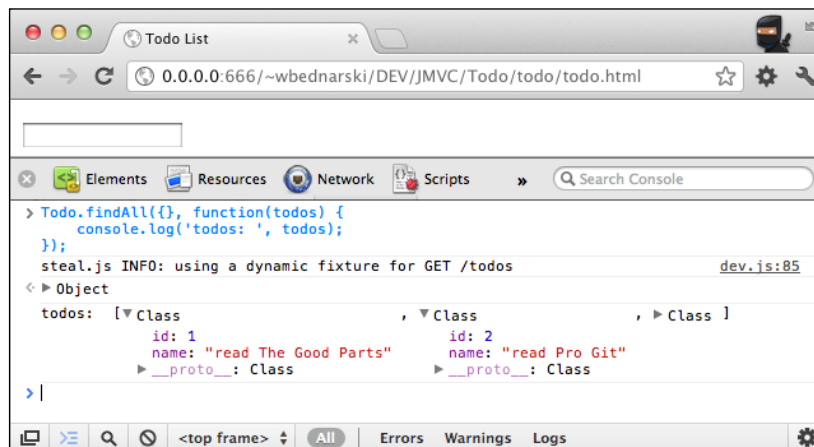
}());
);
```

Now, we can use our `Todo` model methods as if backend services were here.

For instance, we can list all `todos`:

```
Todo.findAll({}, function(todos) {
  console.log('todos: ', todos);
});
```

The following screenshot shows the output of the `console.log('todos: ', todos);` command:



View

Now, it is a good time to add some HTML code to actually see something beyond the browser console. To do this, use the open source client-side template system **Embedded JavaScript (EJS)**.

Create a new file `todos.ejs` in the `todo` directory (the same folder where `todo.js` is located), and add the following code to it:

```
<% $.each(this, function(i, todo) { %>

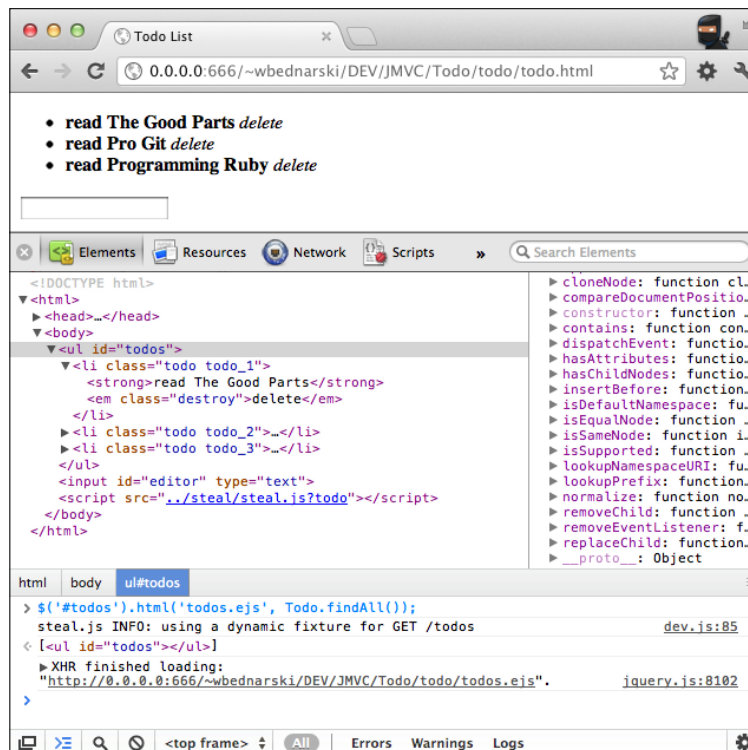
  <li <%= ($el) -> $el.model(todo) %>>
    <strong><%= todo.name %></strong>
    <em class="destroy">delete</em>
  </li>

<% }) %>
```

Then, type the following in the console:

```
$('#todos').html('todos.ejs', Todo.findAll());
```

Now, we can see all `todos` printed:



Basically, the EJS template is an HTML file with injected JavaScript code between `<%` and `%>` or `<%=` and `%>` (and a few other ways).

The difference is that in the second case, all the values returned by the JavaScript code are escaped and printed out. In the first one, they are only evaluated.

The first line is a jQuery each loop – no magic here. However, the next line could be a new thing for many readers. It is ECMAScript Harmony-like, arrow style syntax for functions used by the EJS parser that doesn't darken the whole picture by its simplicity.

The following syntax:

```
($el) -> $el.model(todo)
```

Can be explained as follows:

```
function ($el) {  
    return $el.model(todo)  
}
```

Controller

Let's add some action to our user interface.

Add the following code to the `todo.js` file, and refresh the application in a browser:

```
$.Controller('Todos', {  
    // init method is called when new instance is created  
    'init': function (element, options) {  
        this.element.html('todos.ejs', Todo.findAll());  
    },  
  
    // add event listener to strong element on click  
    'li strong click': function (el, e) {  
        // trigger custom event  
        el.trigger('selected', el.closest('li').model());  
  
        // log current model to the console  
        console.log('li strong click', el.closest('.todo').model());  
    },  
  
    // add event listener to em element on click  
    'li .destroy click': function (el, e) {  
        // call destroy on the model to prevent memory leaking  
        el.closest('.todo').model().destroy();  
    },  
  
    // add event listener to Todo model on destroyed
```

```

    '{Todo} destroyed': function (Todo, e, destroyedTodo) {
      // remove element from the DOM tree
      destroyedTodo.elements(this.element).remove();

      console.log('destroyed: ', destroyedTodo);
    }
  });

  // create new controller instance
  new Todos('#todos');

```

Now, you can click on the `todo` name to see the console log or delete it.

The `init` method is called when a new controller is instantiated.

When the `controller` element is removed from the DOM tree (in our case, `#todos`), the `destroy` method is called automatically, unbinding all controller event handlers and releasing its element to prevent memory leakage.

Routing

Replace the following code:

```

// create new Todo controller instance
new Todos('#todos');

```

With:

```

// routing
$.Controller('Routing', {
  init: function () {
    new Todos('#todos');
  },

  // the index page
  'route': function () {
    console.log('default route');
  },

  // handle URL witch hash
  ':id route': function (data) {
    Todo.findOne(data, $.proxy(function (todo) {
      // increase font size for current todo item
      todo.elements(this.element).animate({fontSize: '125%'},
750);
    }, this));
  },

  // add event listener on selected

```

```
        '.todo selected': function (el, e, todo) {
            // pass todo id as a parameter to the router
            $.route.attr('id', todo.id);
        }
    });

    // create new Routing controller instance
    new Routing(document.body);
```

Refresh the application and try to click on the `todo` list elements. You will see that the URL updates after clicking on the `todo` item with its corresponding ID.

Complete application code

Here is the complete code for the `Todo` application:

```
steal(
    'jquery/class',
    'jquery/model',
    'jquery/util/fixture',
    'jquery/view/ejs',
    'jquery/controller',
    'jquery/controller/route',

    function ($) {
        $.Model('Todo', {
            findAll: 'GET /todos',
            findOne: 'GET /todos/{id}',
            create: 'POST /todos',
            update: 'PUT /todos/{id}',
            destroy: 'DELETE /todos/{id}'
        },
        {
        }
    )
);

// Fixtures
(function () {
    var TODOS = [
        // list of todos
        {
            id: 1,
            name: 'read The Good Parts'
        },
        {
            id: 2,
            name: 'read Pro Git'
        },
    ],
```

```
        {
          id: 3,
          name: 'read Programming Ruby'
        }
      ];

      // findAll
      $.fixture('GET /todos', function () {
        return [TODOS];
      });

      // findOne
      $.fixture('GET /todos/{id}', function (orig) {
        return TODOS[(+orig.data.id) - 1];
      });

      // create
      var id = 4;
      $.fixture('POST /todos', function () {
        return {
          id: (id++)
        };
      });

      // update
      $.fixture('PUT /todos/{id}', function () {
        return {};
      });

      // destroy
      $.fixture('DELETE /todos/{id}', function () {
        return {};
      });
    }());

    $.Controller('Todos', {
      // init method is called when new instance is created
      'init': function (element, options) {
        this.element.html('todos.ejs', Todo.findAll());
      },

      // add event listener to strong element on click
      'li strong click': function (el, e) {
        // trigger custom event
        el.trigger('selected', el.closest('li').model());

        // log current model to the console
        console.log('li strong click', el.closest('.todo').
model());
      },
    },
```



```
// add event listener to em element on click
'li .destroy click': function (el, e) {
  // call destroy on the model to prevent memory leaking
  el.closest('.todo').model().destroy();
},

// add event listener to Todo model on destroyed
'{Todo} destroyed': function (Todo, e, destroyedTodo) {
  // remove element from the DOM tree
  destroyedTodo.elements(this.element).remove();

  console.log('destroyed: ', destroyedTodo);
}
});

// routing
$.Controller('Routing', {
  init: function () {
    new Todos('#todos');
  },

  // the index page
  'route': function () {
    console.log('default route');
  },

  // handle URL witch hash
  ':id route': function (data) {
    Todo.findOne(data, $.proxy(function (todo) {
      // increase font size for current todo item
      todo.elements(this.element).animate({fontSize:
'125%'}, 750);
    }, this));
  },

  // add event listener on selected
  '.todo selected': function (el, e, todo) {
    // pass todo id as a parameter to the router
    $.route.attr('id', todo.id);
  }
});

// create new Routing controller instance
new Routing(document.body);
}
);
```

Summary

In this chapter, we learned what JavaScriptMVC is, and why it is a good and solid framework. We also learned how to install it, and browse the documentation and API. We got an overview of its architecture by building a simple application.

If you can understand all the code that we have written in this chapter, you will be able to dig into the framework easily and fast. Congratulations!

2

DocumentJS

Source code alone is insufficient; documentation is an important part of software engineering. **DocumentJS** is a powerful, yet simple tool designed to easily create searchable documentation for any JavaScript codebase.

In this chapter, we will get an overview of DocumentJS. We will learn how it works and learn to generate its documentation.

The following are the key features of DocumentJS:

- Flexible and easy to extend
- Support Markdown: <http://en.wikipedia.org/wiki/Markdown>
- Integrated documentation viewer with API search
- Works with any JavaScript code and not only with JavaScriptMVC

If you are familiar with JSDoc, YUIDoc, YARD, or similar documentation syntax/tools in then DocumentJS can be learned a, few minutes.

The documentation for DocumentJS can be found at <http://javascriptmvc.com/docs.html#!DocumentJS>.



Markdown is a text-to-HTML conversion tool that allows you to write using an easy-to-read and easy-to-write plain text format (<http://daringfireball.net/projects/markdown>).

How does DocumentJS work?

The architecture of DocumentJS is organized around types and tags.

Types represent every relatively independent part of the JavaScript code that we may want to comment, such as classes, functions (methods), or attributes.

Tags provide additional information to types, such as parameters and returns.

DocumentJS parses JavaScript and Markdown files to produce JSONP files that are used by JMVCDoc to render documentation.

Writing the documentation

Let's add a documentation to our `Todo` list application in *Chapter 1, Getting Started with JavaScriptMVC*.

To add the main documentation page, create a Markdown file `todo.md` in the `Todo/todo` directory with the following content:

```
@page index TodoApp
@description TodoApp is simple todo application.

# TodoApp documentation

Here we can add some more documentation formatted by [Markdown] [1]!

[1]: http://daringfireball.net/projects/markdown/syntax "Check out
Markdown syntax"
```

Then, add these documentation blocks to the `todo.js` file:

```
steal(
  'jquery/class',
  'jquery/model',
  'jquery/util/fixture',
  'jquery/view/ejs',
  'jquery/controller',
  'jquery/controller/route',
  function ($) {

    /**
     * @class Todo
     * @parent index
     * @constructor
     * @author Wojciech Bednarski
     * Creates a new todo.
```

```

*/
$.Model('Todo', {

    /**
    * @function findAll
    * Get all todos
    * @return {Array} an array contains objects with all
    todos

    */
    findAll: 'GET /todos',

    /**
    * @function findOne
    * Get todo by id
    * @return {Object} an objects contains single todo
    */
    findOne: 'GET /todos/{id}',

    /**
    * @function create
    * Create todo
    * @param {Object} todo
    * Todo object
    * @codestart
    * {name: 'read a book by Alfred Szklarski'}
    * @codeend
    *
    * @return {Object} an object contains newly created
    todo

    * @codestart
    * {
    *   id: 577,
    *   name: 'read a book by Alfred Szklarski'
    * }
    * @codeend
    *
    * ### Example:
    * @codestart
    * var todo = new Todo({name: 'read a book by Alfred
    Szklarski'});
    * todo.save(function (todo) {
    *   console.log(todo);
    * });
    * @codeend
    */
    create: 'POST /todos',

    /**
    * @function update
    * Update todo by id

```

```
        * @return {Object} an object contains updated todo
        */
        update: 'PUT /todos/{id}',

        /**
        * @function destroy
        * Destroy todo by id
        * @return {Object} an object contains destroyed todo
        */
        destroy: 'DELETE /todos/{id}'
    },
    {
    }
};

// Fixtures
(function () {
    var TODOS = [
        // list of todos
        {
            id: 1,
            name: 'read The Good Parts'
        },
        {
            id: 2,
            name: 'read Pro Git'
        },
        {
            id: 3,
            name: 'read Programming Ruby'
        }
    ];

    // findAll
    $.fixture('GET /todos', function () {
        return [TODOS];
    });

    // findOne
    $.fixture('GET /todos/{id}', function (orig) {
        return TODOS[(+orig.data.id) - 1];
    });

    // create
    var id = 4;
    $.fixture('POST /todos', function () {
        return {
            id: (id++)
        };
    });
});
```

```
});

// update
$.fixture('PUT /todos/{id}', function () {
  return {};
});

// destroy
$.fixture('DELETE /todos/{id}', function () {
  return {};
});
})();

/**
 * @class Todos
 * Creates a new Todos controller
 * @parent index
 * @constructor
 * @param {String} DOMELEMENT DOM element
 * @return {Object}
 */
$.Controller('Todos', {
  // init method is called when new instance is created
  'init': function (element, options) {
    this.element.html('todos.ejs', Todo.findAll());
  },

  // add event listener to strong element on click
  'li strong click': function (el, e) {
    // trigger custom event
    el.trigger('selected', el.closest('li').model());

    // log current model to the console
    console.log('li strong click', el.closest('.todo').
model());
  },

  // add event listener to em element on click
  'li .destroy click': function (el, e) {
    // call destroy on the model to prevent memory leaking
    el.closest('.todo').model().destroy();
  },

  // add event listener to Todo model on destroyed
  '{Todo} destroyed': function (Todo, e, destroyedTodo) {
    // remove element from the DOM tree
    destroyedTodo.elements(this.element).remove();

    console.log('destroyed: ', destroyedTodo);
  }
}
```



```
    });  
  
    /**  
     * @class Routing  
     * Creates application router  
     * @parent index  
     * @constructor  
     * @param {String} DOMELEMENT DOM element  
     * @return {Object}  
     */  
    $.Controller('Routing', {  
      init: function () {  
        new Todos('#todos');  
      },  
  
      // the index page  
      'route': function () {  
        console.log('default route');  
      },  
  
      // handle URL witch hash  
      ':id route': function (data) {  
        Todo.findOne(data, $.proxy(function (todo) {  
          // increase font size for current todo item  
          todo.elements(this.element).animate({fontSize:  
'125%'}, 750);  
        }, this));  
      },  
  
      // add event listener on selected  
      '.todo selected': function (el, e, todo) {  
        // pass todo id as a parameter to the router  
        $.route.attr('id', todo.id);  
      }  
    });  
  
    // create new Routing controller instance  
    new Routing(document.body);  
  }  
);
```

Type directives

Type directives represent JavaScript constructs that you may want to document:

- @page: This adds a standalone page
- @attribute: These are the document values on an object

- `@function`: These are document functions
- `@class`: This documents a class
- `@prototype`: This is added to the previous class or a constructor's prototype functions
- `@static`: This is added to the previous class or constructor's static functions
- `@add`: This adds the docs to a class or constructor described in another file

Tag directives

Tag directives provide additional information to the comments:

- `@alias`: This specifies other commonly used names for class or constructor
- `@author`: This specifies the author of a class
- `@codestart`: This specifies the start of a code block
- `@codeend`: This specifies end of a code block
- `@constructor`: This documents a contractor function and its parameters
- `@demo`: This is the placeholder for an application demo
- `@description`: This is used to add a short description
- `@download`: This is used to adds download link
- `@iframe`: This is used to add an iframe with example code
- `@hide`: This hides the class view
- `@inherits`: This specifies what the Class or Constructor inherits
- `@parent`: This specifies under which parent the current type should be located
- `@param`: This specifies a function parameter
- `@plugin`: This specifies a plugin by which an object gets stolen
- `@return`: This specifies what a function returns
- `@scope`: This forces the current type to start the scope
- `@tag`: This specifies the tags for searching
- `@test`: This specifies the links for test cases
- `@type`: This sets the type for the current commented code
- `@image`: This adds an image

Generating the documentation

All we need to do to generate the documentation is run the `doc` command from the command line (inside the `Todo` directory):

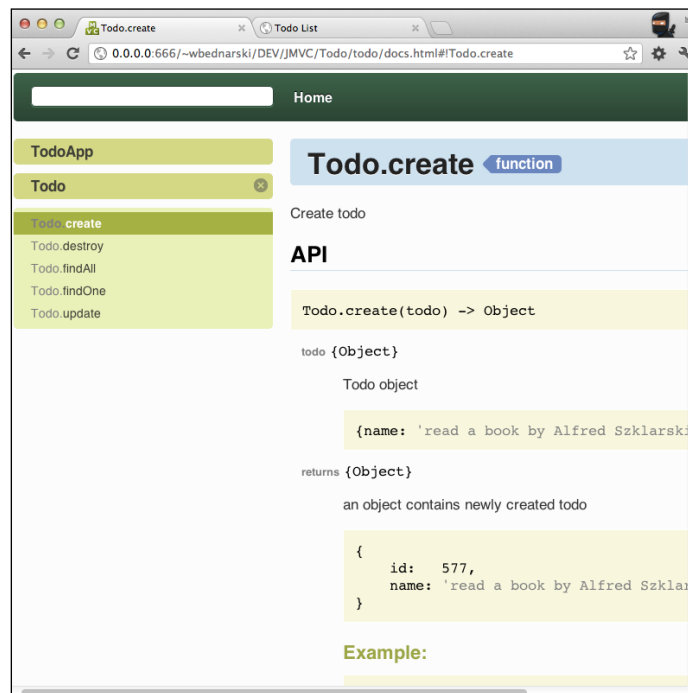
```
$ ./documentjs/doc todo
PROCESSING SCRIPTS
```

```
  todo/todo.js
  todo/todo.md
```

```
GENERATING DOCS -> todo/docs
```

Using default page layout. Overwrite by creating: `todo/summary.ejs`

This generates the documentation, which we can browse by opening `docs.html` located in the `Todo/todo` directory:



We can customize the look and feel of the documentation by changing the `summary.ejs` template file. Simply copy the template from `documentjs/jmvsdoc` to `Todo/todo` and modify it.

Summary

In this chapter, we have learned what DocumentJS is and how to write and generate its documentation.

One good habit that every programmer should have is that he or she must document the codebase and keep it up to date.

3

FuncUnit

FuncUnit is a functional testing framework with jQuery-like syntax. It is built on top of the **QUnit** unit test framework.

Using FuncUnit, we can run tests in all the modern web browsers under OS X, GNU/Linux, or Windows.

Writing a test is really easy and fast, especially if the reader is familiar with the jQuery syntax and/or the QUnit framework.

FuncUnit allow us to run tests in web browsers as well as integrates it with automation tools such as Selenium, or run tests from the command line using wrappers such as PhantomJS.


FuncUnit can be integrated with build tools, such as Maven, to run as a part of the build process. It can be integrated with continuous integration tools, such as Jenkins. More information on FuncUnit can be found at the following URLs:

- **Documentation:** <http://javascriptmvc.com/docs.html#!FuncUnit>
- **Source code:** <https://github.com/bitovi/funcunit>
- **QUnit:** <http://docs.jquery.com/QUnit>

According to Wikipedia, functional testing is defined as follows:

Functional testing is a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered.

In this chapter, we are going to get an overview of the FuncUnit functional testing framework, create tests, and run them against our `Todo` application.

[ **Unit testing versus functional testing**
Unit testing tests an individual unit-like method or function, while functional testing tests the entire functionality usually through the product user interface.]

Creating tests

Creating tests is writing a code that runs against application code to ensure that the code meets its design and behaves as intended. Writing tests can save time by finding bugs at the early development stage.

Lets add our first test for the `Todo` app:

1. In the `Todo/todo` folder, create a folder named `tests`. Inside it, create a file named `todo_test.html` with following content:

```
<!doctype html>

<html>
  <head>
    <title>Todo List - tests</title>
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="../../funcunit/qunit/qunit.css"
  />
  </head>
  <body>
    <h1 id="qunit-header">AutoSuggest Test Suite</h1>

    <h2 id="qunit-banner"></h2>

    <div id="qunit-testrunner-toolbar"></div>
    <h2 id="qunit-userAgent"></h2>
    <ol id="qunit-tests"></ol>

    <script src="../../steal/steal.js?todo/tests/todo_test.js"></
script>
  </body>
</html>
```

This file provides a page skeleton that is populated with the FuncUnit output, which remains the same for all future test cases – use it as a template; only the title and path to the test file will change (highlighted code).

2. Next, create a `todo_test.js` file with following content:

```
steal(
  'funcunit',
  function ($) {

    module('Todo app', {
      setup: function () {
        S.open('//todo/todo.html');
      }
    });

    test('page has #todos placeholder', function () {
      ok(S('body > #todos').size(), 'The #todo is child of the
body');
    });
  }
);
```

Module

A module signature is given by `module(name, [lifecycle]);`.

The module method comes from the QUnit project and provides the functionality to divide test into modules.

The first parameter is a string with a module name. The second one is an object with two possible methods, `setup` and `teardown`. The `setup` callback method runs before each test, while the `teardown` runs after each test in the module.

Open

An open signature is given by `open(path, [success], [timeout])`.

In our example, we used the `open` method to open a given URL and run it against the test `'page has #todos placeholder'`.

Test

A test signature is given by `test(name, [expected], test)`.

This method runs the actual test code.

The first parameter is the name of the test, while the second is required to actually run the code.

Ok

An ok signature is given by `ok (state, [message])`.

The `ok` method is a Boolean assertion. If the first parameter evaluates to true, the test passes. The second parameter is optional and it describes test.

S

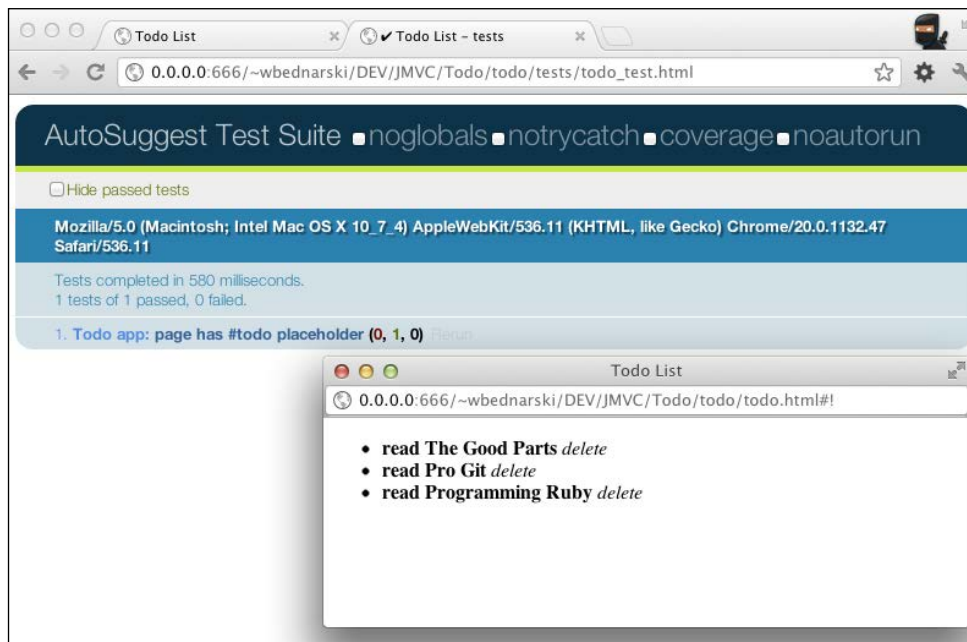
`s` is basically copy of the jQuery shortcut `$`, extended by FuncUnit-specific methods.

Running tests

There are quite few ways to run tests. Using a web browser, a command-line tool will open the web browser and close it after the test execution is completed. We can also run tests using standalone JavaScript environments.

Web browser

Disable the pop-up blocker and open `tests/todo_test.html` in the browser. The test will open the `Todo` application and run the test case against it. After this, you should be able to see something similar to the following screenshot:



Selenium

Run the following command from the `Todo` app directory:

```
$ ./js funcunit/run selenium todo/tests/todo_test.html
```

This command will open Firefox, run exactly the same test as in the web browser example, close the browser, and print the results on a command line.

PhantomJS

A much faster solution is running test using PhantomJS, because it doesn't launch the web browser.

Execute following command to run the test:

```
$ ./js funcunit/run phantomjs todo/tests/todo_test.html
```

The preceding command will run the test in a PhantomJS environment, so it won't open any web browser as it did in the previous case. But, it will run tests inside the WebKit wrapper.

The command-line output should be similar to the following:

```
Opening file:///Users/wbednarski/Sites/DEV/JMVC/ToDo/todo/tests/todo_
test.html
starting steal.browser.phantomjs
steal.js INFO: Opening //todo/todo.html
steal.js INFO: using a dynamic fixture for GET /todos
steal.js INFO: ajax request to todos.ejs, no fixture found
steal.js INFO: ajax request to todos.ejs, no fixture found
default route

Todo app
  page has #todos placeholder
    [x] The #todo is child of the body

Time: 3 seconds, Memory: 81.06 MB

OK (1 tests, 0 assertions)
```

EnvJS

Another way to run tests is to use EnvJS-simulated browser environments written in JavaScript.

EnvJS can only be used to run unit tests, because it doesn't accurately implement event simulation.

Run the test by executing the following command:

```
$ ./js funcunit/run envjs todo/tests/todo_test.html
```

Integration

Integration is possible with popular build or CI tools, such as Jenkins or Maven:

- **Jenkins:** <http://javascriptmvc.com/docs.html#!funcunit.jenkins>
- **Maven:** <http://javascriptmvc.com/docs.html#!funcunit.maven>

Summary

As we can see, FuncUnit is an easy-to-use, powerful testing framework.

Writing test cases is fast and easy. The possibility to run them in several ways as well as integrate them with automated and build tools makes FuncUnit a solid tool.


Now, we have no excuses not to write tests.

4 jQueryMX

jQueryMX is a collection of jQuery libraries that provides functionality necessary to implement and organize large JavaScript applications.

It provides classical inheritance simulation, model-view-controller layers to provide logically separated codebase. It also provides useful DOM helpers, custom events, and language helpers.

In this chapter we will go through the most common or most interesting ones.

 For the full plugins list go to <http://javascriptmvc.com/docs.html#!jquerymx>.

We use the existing `Todo` application folder structure to play around with jQueryMX plugins.

In the `Todo` folder create a `jquerymx_playground` folder with two files using the following code snippet:

```
<!doctype html>



<html>
<head>
  <title>jQueryMX playground</title>
  <meta charset="UTF-8"/>
</head>
<body>

<script src="../../steal/steal.js?jquerymx_playground/jquerymx_
playground_0.js"></script>
</body>
</html>
```

When a code snippet is indicated as run in the console it means paste and execute the example in Google Chrome Console on the opened index.html page. We can use any web browser console such as Firebug, however, Google Chrome (and Safari) seem to be among the best at the moment and have a very handy code complementation.

\$.Class

\$.Class provides classical inheritance simulation based on John Resig's *Simple JavaScript Inheritance* found at <http://ejohn.org/blog/simple-javascript-inheritance/>.

 The class signature is \$.Class ([NAME , STATIC,] PROTOTYPE) -> Class. 

The class method is available to all class instances whereas the instance method is available only to a particular instance.

Let's write some examples in the file jquerymx_playground_0.js:

```
steal(
  'jquery/class',
  function ($) {
    $.Class('Bank.Account', {
      setup: function () {
        console.log('Bank.Account class: setup');
      },
      init: function () {
        console.log('Bank.Account class: init');
      },
      getType: function () {
        return 'Bank.Account class method';
      }
    },
    {
      setup: function () {
        console.log('Bank.Account instance: setup');
      },
      init: function () {
        console.log('Bank.Account instance: init');
      },
      getType: function () {
        return 'Bank.Account instance method';
      }
    }
  )
}
```

```

    }
  );

  Bank.Account('Bank.Account.SavingsAccount', {
    setup: function () {
      console.log('Bank.Account.SavingsAccount
class: setup');
    },

    init: function () {
      console.log('Bank.Account.SavingsAccount
class: init');
    }
  },
  {
    setup: function () {
      console.log('Bank.Account.SavingsAccount
instance: setup');
    },

    init: function () {
      console.log('Bank.Account.SavingsAccount
instance: init');
    },

    //      getExtendedType: function () {
    //          return 'Hello ' + this.getType();
    //      },

    getType: function () {
      return 'Hello ' + this._super();
    }
  }
  );

  //      console.log('instantiate: acc, savingAcc');
  //      window.acc = new Bank.Account();
  //      window.savingAcc = new Bank.Account.SavingsAccount();
  }
  );

```

In the console we can see that classes are created. Lets create some instances as follows:

```

var acc = new Bank.Account();
var savingAcc = new Bank.Account.SavingAccount();

```

We can execute instance methods as follows:

```

acc.getType();
savingAcc.getType();

```

In the following sections, let's break down the code and see what happened here.

The first parameter

Using `$.Class` we created a new class with the name `Account` passing the string `Bank.Account` as a first parameter. By using the dot notation we created a namespace `Bank`. This is why we created a new instance of the class `Account` we called `Bank.Account`. In this case `Bank` is just an empty object to help us create nice and tidy application object structure.

An alternative namespace, for example, could be `CompanyName.Product.SomeClass`.

The second parameter

As the second parameter we passed object with properties, which are class properties shared with all classes instances.

In our case class method `getType` from the `Account` class is available in the `SavingsAccount` class. We can thus type the following in the console:

```
Bank.Account.SavingsAccount.getType();
```

The third parameter

As the third parameter we passed an object with properties, which are instance properties shared with all instances. We thus type the following command in the console:

```
savingAcc.getType();
```

Method override

In the `getType` instance method example, we can see how to override methods in the children objects.

In `SavingsAccount` we override the `getType` method by adding an additional `Hello` string to the ancestor method of the same name, and call ancestor method using the following:

```
this._super();
```

In case we don't want to use the same name, we can use the following:

```
getExtendedType: function () {  
    return 'Hello ' + this.getType();  
}
```

Life cycle

In both class and instance we can use the predefined method `setup` and `init`.

If it exists it is always called, so there is no need to call it manually.

The `setup` method is called first, then the `init` method. In most cases there is no need to use the `setup` method.

\$.Model

\$.Model is the application data layer. It provides an easy way to connect to the services that provide RESTful APIs, listen to data changes, and bind HTML elements to models, deferrers, and validations.

\$.Model is very handy; we don't need to manually write XHR calls using jQuery's Ajax method for instance. We can map our backend API using \$.Model and then use its methods to pull/push data to the server.

We can organize \$.Models with a list using `$.Model.List`, which is similar to Backbone.js's collections (<http://backbonejs.org/#Collection>).

Let's write some code in the file `jquerymx_playground_1.js`:

```
steal(
  'jquery/model',
  'jquery/dom/fixture',
  function ($) {
    $.Model('AccountModel', {
      findAll: 'GET /accounts',
      findOne: 'GET /accounts/{id}',
      create: 'POST /accounts',
      update: 'PUT /accounts/{id}',
      destroy: 'DELETE /accounts/{id}'
    },
    {
    }
  )
);

// Fixtures
(function () {
  var accounts = [
    {
      id: 1,
      type: 'USD'
    },
    {

```



```
        id: 2,
        type: 'EUR'
    }
];

// findAll
$.fixture('GET /accounts', function () {
    return [accounts];
});

// findOne
$.fixture('GET /accounts/{id}', function () {
    return accounts;
});

// create
$.fixture('POST /accounts', function () {
    return {};
});

// update
$.fixture('PUT /accounts/{id}', function (id, acc) {
    return acc;
});

// destroy
$.fixture('DELETE /accounts/{id}', function () {
    return {};
});
})();

AccountModel.findAll({}, function (accounts) {
    $.each(accounts, function (i, acc) {
        $('<p>').model(acc).text(acc.type).
            appendTo('body');
    });
});

AccountModel.bind('updated', function (e, acc) {
    acc.elements($('body')).remove();
});

AccountModel.bind('created', function (e, acc) {
    console.log('AccountModel.bind: event: ', e,
        'Account: ', acc);
});
}
);
```

Lets break down this code and see what happened here:

- The code starting from `$.Model` is responsible for mapping API to our model.
- The lines starting with `$.fixtures` are responsible for imitating server responses. Fixtures are very helpful when we need to start development without the web server API being ready or available.
- The `bind` method in the `model` class is responsible for binding model methods `update` and `create`. We can try using them to see how they work from the web browser console by executing these methods on the instance of `AccountModel`.

\$.View

\$.View is a client-side template solution. It populates HTML templates with data.

It comes with four pre-packaged template engines, which can be downloaded from the following websites:

- **EJS:** <http://embeddedjs.com> (default one created by JMVC team)
- **Jaml:** <http://javascriptmvc.com/docs.html#!Jaml>
- **Micro:** <http://javascriptmvc.com/docs.html#!Micro>
- **jQuery templates:** <http://api.jquery.com/category/plugins/templates>

It's easy to extend it by using `$.View.register`.

Templates can be embedded in the HTML documents or loaded synchronously or asynchronously from external files. \$.View supports template caching and bundling in the production builds.

Embedded

Templates are embedded in the HTML documents as follows:

Let's copy the following code into `index.html` file:

```
<script type='text/ejs' id="accounts">
  <p>JavaScriptMVC is <%= message %></p>
</script>
```

Also, copy the following code into file `jquerymx_playground_2.js`:

```
steal(
  'jquery/view',
  'jquery/view/ejs',
  function ($) {

  }
);
```

In the console, type the following:

```
$('body').html('accounts', {message: 'Awesome'});
```

As a result, the following DOM node should be created:

```
<p>JavaScriptMVC is Awesome</p>
```

External

This method of using templates is the most common one, since it allows for better organization of the project file's structure:

Create a file `message.ejs` and copy the previous template into it. The file content should look as follows:

```
<p>JavaScriptMVC is <%= message %></p>
```

Type the following in the console:

```
$('body').html('message.ejs', {message: 'Awesome'});
```

The object with the property `message` is passed into the `HTML` method which uses `message.ejs` file to render the text "Awesome" in place of `<%= message %>` and then append it into the `body` DOM node.

The result should be the same as in the embedded one.

Sub-templates

Inside a template we can embed another template, as follows:

```
<%= $.View('sub-message.ejs', message); %>
```

\$.Controller

The `$.Controller` plugin helps to create an organized, memory leak-free JavaScript code.

A great example of how to use `$.Controller` is the Todos controller from *Chapter 1, Getting Started with JavaScriptMVC*, is as follows:

```
$.Controller('Todos', {
  // init method is called when new instance is created
  'init': function (element, options) {
    this.element.html('todos.ejs', Todo.findAll());
  },

  // add event listener to strong element on click
  'li strong click': function (el, e) {
    // trigger custom event
    el.trigger('selected', el.closest('li').model());

    // log current model to the console
    console.log('li strong click',
      el.closest('.todo').model());
  },

  // add event listener to em element on click
  'li .destroy click': function (el, e) {
    // call destroy on the model to prevent memory
    // leaking
    el.closest('.todo').model().destroy();
  },

  // add event listener to Todo model on destroyed
  '{Todo} destroyed': function (Todo, e, destroyedTodo) {
    // remove element from the DOM tree
    destroyedTodo.elements(this.element).remove();

    console.log('destroyed: ', destroyedTodo);
  }
});
```

DOM helpers

DOM helpers extensions add a set of useful plugins for the DOM. They are described in the following sections.

\$.cookie

The `$.cookie` plugin contains useful methods to manage cookies.

Let's paste the following code into the `jquerymx_cookie.js` file:

```
steal(
  'jquery/dom/cookie',
  function ($) {

  }
);
```

We can create a cookie in the console using the following command:

```
$.cookie('CookieName', 'CookieValue');
```

In the resources tab we can see that cookie has been created.

We can get a cookie using its cookie name, as follows:

```
$.cookie('CookieName');
```

We can also delete a cookie using the following command:

```
$.cookie('CookieName', null);
```

\$.fn.compare

The `$.fn.compare` plugin compares two nodes and returns a number describing how they are positioned each together.

Let's paste following code into the `jquerymx_compare.js` file:

```
steal(
  'jquery/dom/compare',
  function ($) {
    $('body').append('<p>paragraph</p>
    <strong>strong</strong>');
  }
);
```

In the console, run the following command:

```
$('#p').compare($('#strong'));
```

Next, run the following command:

```
$('#strong').compare($('#p'));
```

In the first case we should get 4 and in the second case 2.

Here is what the numbers mean:

- 0: The elements are identical
- 1: The nodes are in different documents (or one is outside of a document)
- 2: `strong` precedes `p`
- 4: `p` precedes `strong`
- 8: `strong` contains `p`
- 16: `p` contains `strong`

`$.fn.selection`

The `$.fn.selection` plugin sets or gets current text selection on any element.

Let's paste the following code into the `jquerymx_selection.js` file:

```
steal(
  'jquery/dom/selection',
  function ($) {
    $('body').append('<p>Hello from paragraph!</p>');
  }
);
```

In the console, run the following command:

```
$('#p').selection();
```

It should return `null`.

Now, select some part of the text and run the command again, it should return an object as follows:

```
{
  end: 15,
  start: 4
}
```

To set the selection, use the following command:

```
$('#p').selection(6, 10);
```

\$.fn.within

The `$.fn.within` plugin returns the elements that are within the given position.

Let's paste the following code into the `jquerymx_within.js` file:

```
steal(
  'jquery/dom/within',
  function ($) {
    $('body').append('<p>Hello from paragraph!</p>');
  }
);
```

In the console, run the following command:

```
$('.p').within(30, 20);
```

It should return an array containing all `p` elements with a position left 30 px and top 20 px.

\$.Range

The `$.Range` plugin contains useful methods that operate on text selections to support creating, moving, and comparing selections.

Let's paste the following code into the `jquerymx_range.js` file:

```
steal(
  'jquery/dom/range',
  function ($) {
    $('body').append('<p>Hello from paragraph!</p>');
  }
);
```

In the console, run the following command:

```
$.Range.current();
```

To get the current range, select some portion of the text and execute the code again and compare the returned objects.

To get the current selection text, run the following command in the console:

```
$.Range.current().toString();
```

\$.route

The `$.route` plugin contains useful methods to manage the application state.

Let's paste the following code into the `jquerymx_route.js` file:

```
steal(
  'jquery/dom/route',
  function ($) {
    $.route.bind('change', function (e, attr, how,
      newVal, oldVal) {
      console.log('event: ', e, '| attribute changes: ',
        attr, '| how changes: ', how, '| new value: ',
        newVal, '| old value: ', oldVal);
    });
  }
);
```

At the end of the URL, type the following:

```
#!&type=UTC
```

Then, type the following command and observe the console output:

```
#!&type=GTM
```

Another example of routing can be found in *Chapter 1, Getting Started with JavaScriptMVC*, in the `Todo` application.

Special events

Special events extensions add a set of special events plugins.

\$.Drag and \$.Drop

The \$.Drag and \$.Drop plugins contain the drag and drop events.

Let's paste the following code into the jquerymx_draganddrop.js file:

```
steal(
  'jquery/event/drag',
  'jquery/event/drag/limit',
  'jquery/event/drag/scroll',
  'jquery/event/drag/step',
  'jquery/event/drop',
  function ($) {

    $('body').append('<p>Drag me, but not too far...</p>');

    $('p').bind('dragmove', function (e, drag) {
      if (drag.location.top() > 150 || drag.location.
        left() > 450) {
        console.log('limiter');
        e.preventDefault();
      }
    });
  }
);
```

Language helpers

Language helpers are a set of jQuery plugins. They are described in the following sections.

\$.Object

The \$.Object plugin contains the following three useful methods:

- same: It compares two objects
- subset: It checks if an object is a set of another object
- subsets: It returns the subsets of an object

same

The same method can compare two objects. It supports nested objects. We can also specify if the comparison is case sensitive or if we can skip a particular property comparison.

Let's paste the following code into the `jquerymx_object.js` file:

```
steal(
  'jquery/lang/object',
  function ($) {
    window.object_1 = {
      property_1: 'foo',
      property_2: {
        property_1: 'bar',
        property_2: {
          property_1: 'Hello JMVC!'
        }
      }
    }
  };

  window.object_2 = {
    property_1: 'foo',
    property_2: {
      property_1: 'bar',
      property_2: {
        property_1: 'HELLO JMVC!'
      }
    }
  };
});
```

In the console, run the following command:

```
$.Object.same(object_1, object_2);
```

It should return `false`.

Now try to ignore the case:

```
$.Object.same(object_1, object_2, {property_2: 'i'});
```

It should return `true`, since `property_2` and all its children are compared with the ignore case flag.

To ignore the case in a particular property we can specify it as follows:

```
$.Object.same(object_1, object_2, {property_2: {property_2: { property_1:
'i'}}});
```

The result should be `true` as well.

\$.Observe

The `$.Observe` plugin provides an observer pattern for the JavaScript objects and arrays.

Let's paste the following code into the `jquerymx_observe.js` file:

```
steal(
  'jquery/lang/observe',
  'jquery/lang/observe/delegate',
  function ($) {
    window.data = {
      accNumber: {
        iban: 'SWISSQX',
        number: 6987687
      },
      owner: {
        fName: 'Nicolaus',
        lName: 'Copernicus'
      }
    };

    window.oData = new $.Observe(data);

    oData.bind('change', function (e, attr, how, newVal, oldVal) {
      console.log('event: ', e, '| attribute changes: ', attr,
        '| how changes: ', how, '| new value: ', newVal, '| old value: ',
        oldVal);
    });
  }
);
```

In the console, run the following command:

```
oData.attr('accNumber.number');
```

Next, run the following command:

```
data.accNumber.number;
```

The same number should be displayed.

Change the value of the `number` property using the following command:

```
oData.attr('accNumber.number', 123456);
```

Since we bound the anonymous function to the `change` event, which is emitted when any of the observable object property has changed, `console.log` with all passed information, should be displayed.

Please note that `oData` is a copy of data, so the command:

```
oData.attr('accNumber.number');
```

Is different from:

```
data.accNumber.number;
```

\$.String

The `$.String` plugin contains useful string methods.

Let's paste following code into the `jquerymx_string.js` file:

```
steal(
  'jquery/lang/string',
  'jquery/lang/string/deparam',
  'jquery/lang/string/rsplit',
  function ($) {
  }
);
```

deparam

This method converts URL parameters into an object literal.

In the console run the following command:

```
$.String.deparam('en=1&home=a3373dsf6wfd&page[main]=uy7887d');
```

It should convert string into the following object:

```
{
  en: '1',
  home: 'a3373dsf6wfd',
  page: {
    main: 'uy7887d'
  }
}
```

\$.toJSON

The `$.toJSON` plugin contains useful object methods.

Let's paste the following code into the `jquerymx_tojson.js` file:

```
steal(
  'jquery/lang/json',
  function ($) {
    window.object_1 = {
      property_1: 'foo',
      property_2: {
        property_1: 'bar',
        property_2: {
          property_1: 'Hello JMVC!'
        }
      }
    };
  }
);
```

In the console, run the following command:

```
$.toJSON(object_1);
```

It should return a JSON representation of a given object.

\$.Vector

The `$.vector` plugin contains useful methods to create and operate on vectors.

Let's paste the following code into the `jquerymx_vector.js` file:

```
steal(
  'jquery/lang/vector',
  function ($) {
  }
);
```

In the console, run the following command:

```
new jQuery.Vector(1,2);
```

It should return a new `Vector` instance.

Summary

In this chapter we learned what jQueryMX plugins have to offer and how we can use them to make our day-to-day coding more efficient.


In the next chapter we will learn about dependency management tool, StealJS.

5

StealJS

StealJS is an independent code manager and packaging tool, which allows us to load JavaScript and other file types into an application, concatenate multiple JavaScript or CSS files, and compress their content. StealJS provides cross-browser message logging, code generators, and a simple package management tool.

In this chapter, we will go through all the StealJS features.

[ StealJS requires Java 1.6 or greater.]

Dependency management

Dependency management is a tool, which provides an organized way for managing software components to work together as a one system.

The following are a few StealJS key features:

- Loading individual files only once
- Loading files from different domains
- Loading JavaScript and CoffeeScript
- Loading CSS less

We have used StealJS many times in the previous chapters. Let's have a closer look at it now.

Using StealJS, we can load files as follows:

```
steal(  
  'file_one',  
  'file_two',  
  function ($) {  
  
  }  
);
```

These files are loaded in parallel and in a random order. If `file_two` has dependencies in `file_one`, we can wait for `file_one` before starting to fetch `file_two`, as follows:



```
steal(  
  'file_one').then(  
  
  'file_two',  
  function ($) {  
  
  }  
);
```

Logger

`steal.dev` provides two logging functions similar to the popular `console.log()` function, and automatically removes them from the production build.

We can use it as follows:

```
steal.dev.log('See me in the console');  
steal.dev.warn('Me too!');
```

 All logs are removed from the production build. 

Code cleaner

`steal.clean` beautifies the JavaScript code and checks it using the **JSLint** code quality tool:

- <http://www.jslint.com>
- <http://jsbeautifier.org>

We can use the `cleanjs` command to beautify the code in one file:

```
$ ./js steal/cleanjs todo/todo.js
```

Or all files in our project:

```
$ ./js steal/cleanjs todo/todo.html
```

To run our code against JSLint, add the `-jslint true` parameter:

```
$ ./js steal/cleanjs todo/todo.js -jslint true
```

We can ignore the files from being cleaned by adding a comment similar to the following:

```
    /*!steal-clean
```

Concatenation and compression

`steal.build` compresses and concatenates CSS and JavaScript files into a single or several files. It uses the Google Closure compressor by default.

Since it opens the applications in Envjs, it can be run against applications that don't use StealJS.

To make production-ready files of our `Todo` application, we can run the following command:

```
$ ./js steal/buildjs todo/todo.html -to todo_prod
```

We can run the script against a URL:

```
$ ./js steal/buildjs http://YOUR_SERVER/todo/todo.html -to todo_prod
```

Summary

In this chapter, we learned how to load files into the project, used cross-browser logging systems that are removed in production build, cleaned the code and made a production-ready application.

6

Building the App

In the past few chapters we have learned what JavaScriptMVC is, how to install it, and we went through its components.

Now is the time for the most exciting chapter for any developer. We are going to build a real-world application. Due to the book's scope limitation we are not going to write backend API set-up servers, and so on, instead of we will use browser storage.

Thanks to layer separation in JavaScriptMVC this is easily done by changing the code in the model to switch the application persistent layer from browser storage option to any backend language, framework, or system such as Sinatra, Ruby on Rails, Django, and Node.js.

This chapter's goal is to show how to build a real-word application from concept through design, implementation, documentation, and testing. We will develop an application that actually does something, is useful for readers, and can be easily customized to the reader's needs.

Time tracking and invoicing for freelancers

The app we are going to build in this chapter is called Time tracking and invoicing for freelancers; let's call it TTI in short.

Application development will only start here. We are not going to write the complete code base, it will be simply too big to fit it here. It's like a homework exercise, when students start their writing application at university and finish them at home. Be creative!

Planning

Okay, so we are going to write an application. Now it's time to answer the most important question: *What problem is our application is about to solve?*

We can clearly identify two main application areas:

- Tracking time we spend on a task
- Making an invoice

Let's break down our application's main areas into a features list as follows:

- Clients list
- Time tracker
 - Track time
 - Fixed cost task
- Reports
 - Daily
 - Weekly
- Statistics
 - Monthly
 - Yearly
- Invoicing
- Export and import data

A features list will help us make a development plan. Now we can think about how much time we need to accomplish for each of them. We can use just a calendar to write down our estimates or use one of many the free issue-tracking tools such as <https://trello.com/> or <http://trac.edgewall.org/>.

The ideal solution would be to use a methodology such as Scrum—[http://en.wikipedia.org/wiki/Scrum_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development)) or one of the best issue-tracking tools in the business, JIRA—<http://en.wikipedia.org/wiki/JIRA>.

Preparing wireframes

The next step is preparing application wireframes. This is a very important step in the application development cycle. It allows us to quickly sketch the application interface for different pages as well as very fast redesign pages and saves time in future development. Once we start writing the code, any changes will be harder and less cost efficient than changing wireframes.

The next steps are creating mockups and prototypes. However, we do not have a graphic designer here and no client to show business logic and finally it's out of this book's scope, so we are going straight to the next step.

Wireframes are generally basic sketches of components used in the application to show user interface and application features.

Mockups are the next level of wireframes, basically containing all we can find on wireframes but were in the actual design.

Prototypes are semi-functional applications to present business logic.

We can create wireframes by using just a piece of paper and pencil; a lot of people prefer this way. There is a bunch of different software that can help us in this step. I'll use Balsamiq Mockups but really any tool will be good here.

To give us a better overview on TTI application let's have a look at wireframes:

Since this book orientation is portrait and web browsers orientation are landscape, the reader is asked to have a look at following the wireframes from a different perspective.

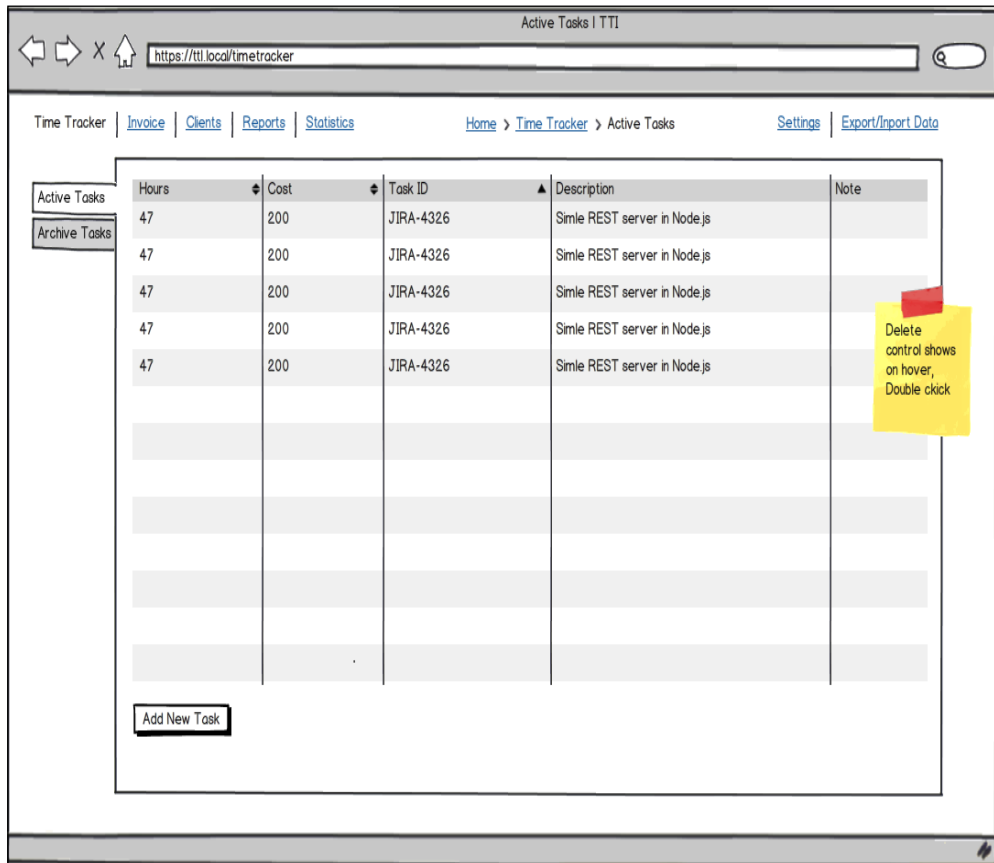
The following wireframe shows the time tracker main page.

The main menu is located at the top-left corner and allows us to switch between the main application functionalities.

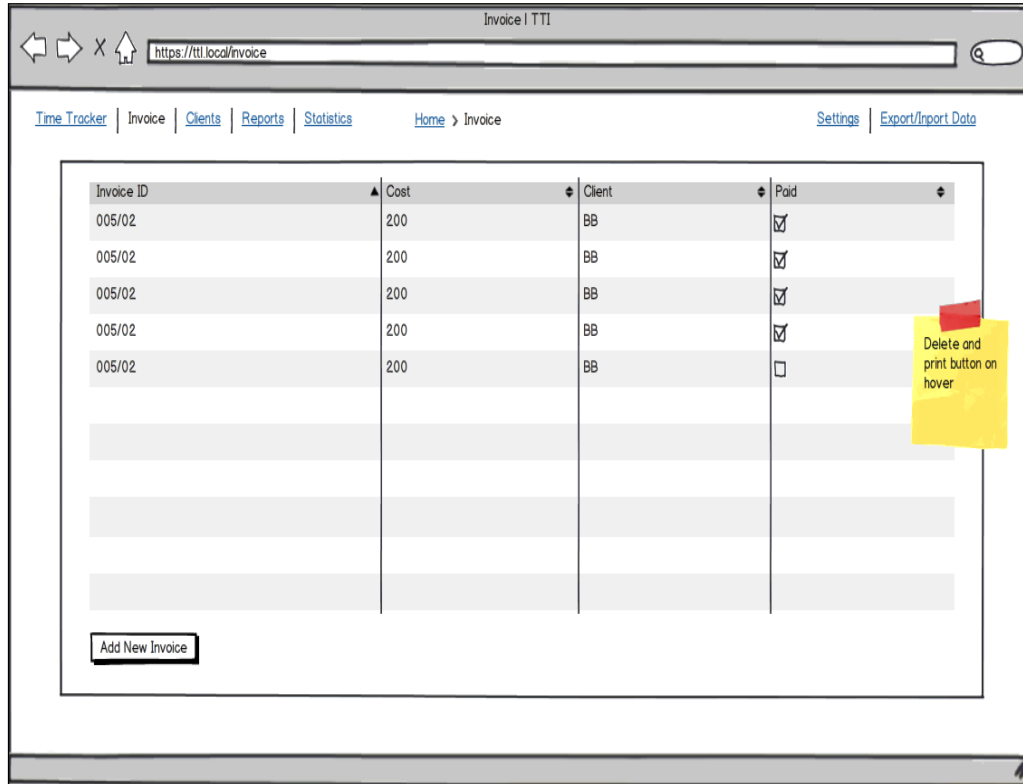
Breadcrumb is located at the top center and allows us to easily indicate which part of the application we are currently in.

Settings and **Export/Import Data** tabs go on the top-right corner.

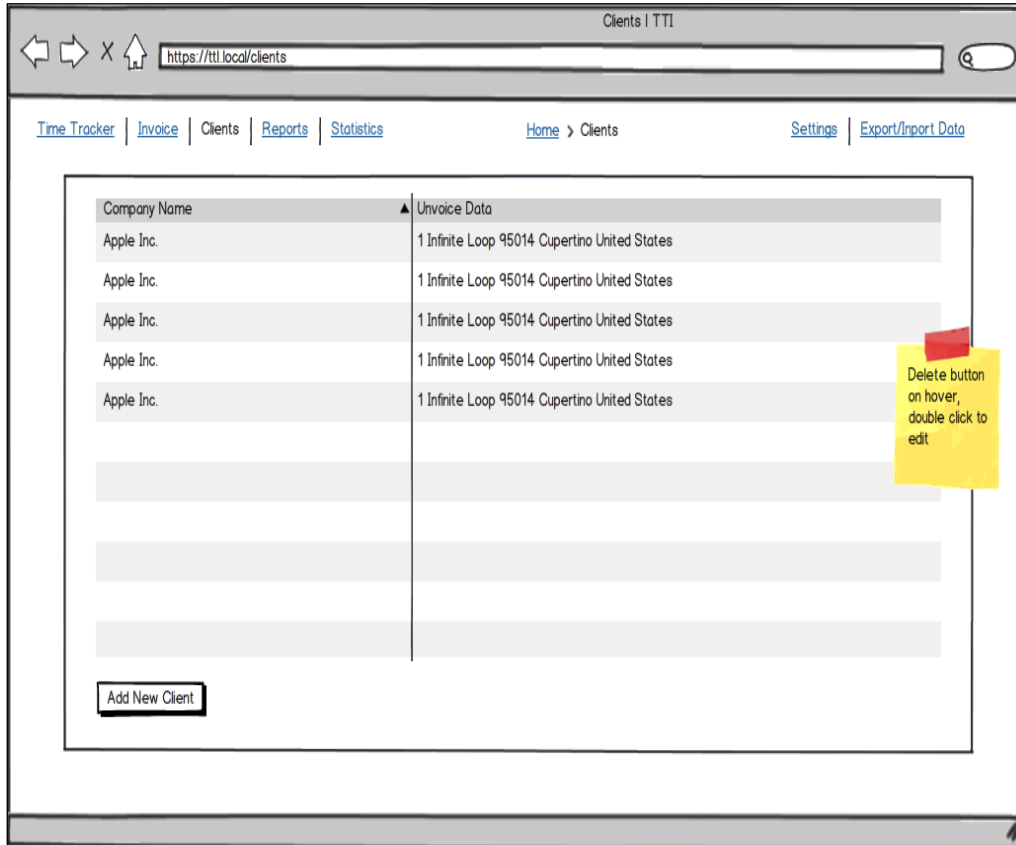
Time Tracker is located in the center with two main tabs: **Active Tasks** and **Archive Tasks**. Each task has fields: **Hours**, **Cost**, **Task ID**, **Description**, and **Notes**. The **Add New Task** button is located at the bottom that allows us to add a new task. **Archive** button is visible when task is hovered. To edit a task double-click on it. The URL for the time tracker page is /timetracker.



The following wireframe shows the invoice main page. The URL is /invoice:



The following wireframe shows the clients main page. The URL is `/clients`.



Setup project

We are assuming that the reader has installed a web server, such as Apache or Nginx. In the server-working directory we need to create the `TTI` folder. Another option is to use the Vagrant-powered environment created especially for this book available at https://github.com/wbednarski/JavaScriptMVC_kick-starter.

In this folder we will initialize the Git repository to track all changes, install JavaScriptMVC, and create the application structure.

Tracking changes under VCS

It's a good idea from the very beginning to keep all the project files under the version control system. The reason for that is very simple and beneficial for future development—we can easily revert any changes and track them.

Using decentralized VCS has an invaluable benefit over centralized VCS, because we can commit changes without push, so we can commit often even after a small change in the codebase. Another good practice is to use one branch per feature.

In this book we are going to use Git, but actually any **Distributed Version Control Systems (DVCS)** is good. Mercurial is another popular DVCS.

The following steps should be performed to create a new Git repository, add all the files, and commit them:

1. Inside the `TTI` directory, type to install JavaScriptMVC:

```
$ git init
$ git submodule add git://github.com/jupiterjs/steal.git
$ git submodule add git://github.com/jupiterjs/documentjs.git
$ git submodule add git://github.com/jupiterjs/funcunit.git
$ git submodule add git://github.com/jupiterjs/jquerymx.git jquery
```
2. Install and update JavaScriptMVC submodules using the following commands:

```
$ git submodule init
$ git submodule update
```
3. Install Syn using the following commands:

```
$ cd funcunit
$ git submodule init
$ git submodule update
```
4. Move the `js` command to the project's root directory (run from root directory):

```
$ ./steal/js steal/make.js
```

By default all the repositories are on a master branch. Let's switch to the latest version of JavaScriptMVC, which is 3.2.2 at the time this book was written.

5. In all the submodules directories, type the following command:

```
$ git checkout v3.2.2
```

6. In the TTI directory create our application directory `tti` and add it under Git.

```
$ mkdir tti
```

```
$ git add .
```

```
$ git commit -m "initial commit"
```



If the reader wants to keep the codebase copy on the server, they can do this using the free code hosting solutions available at <https://github.com> or <https://bitbucket.org>.

All the code we are going to develop will be placed in the `tti` folder.

Application structure

Our application structure will look similar to the following hierarchy:

```
TTI/
|
| tti/
|   | controllers/
|   | docs/
|   | models/
|   | tests/
|   |   | unit/
|   |   |   | models/
|   |   | functional/
|   | views/
|   |   | styles/
|   |   |   | css/
|   |   |   | sass/
|   |   | templates/
|   |   |   | tasks
|   |   |   |
```

```
| clients
|
| vendors/
|   | jquery_ui/
|   |
|   | pouchdb/
```

IndexedDB

Since local storage is too simple for our application and Web SQL database is deprecated the natural choice is IndexedDB.

In the root level create `vendors` directory to store all third part code, plugins, and so on.

Download and copy PouchDB to the `vendors` directory library, which provides good cross-browser API for IndexedDB. You can download PouchDB from the following location:

- <http://pouchdb.com>
- <https://github.com/daleharvey/pouchdb>

Creating models

Let's create a `task.js` file under the `models` directory. In the `Task` model we will keep all task-related CRUD methods that operate on a local database.

```
steal(
  'jquery/model',
  'vendors/pouchdb.js',

  function ($) {
    'use strict'

    // local variable to keep reference to time-tracker database
    var db;

    /**
     * @class TTI.Models.Task
     * @parent index
     * @constructor
     * @author Wojciech Bednarski
     */
    $.Model('TTI.Models.Task', {
```

```
/**
 * @function init
 * @hide
 * Creates database time-tracker or get it if exists
 */
timeTracker) {
  init: function () {
    Pouch('idb://time-tracker', function (err,
      db = timeTracker;

      console.log('TTI.Models.Task.init() | idb://
time-tracker | err:', err, 'db:', db);
    })
  },
}
```

The `init` method is responsible for creating a time-tracker database or getting reference to it if it exists. The `idb://` protocol is telling PouchDB to use IndexedDB as a storage option.

```
/**
 * @function findAll
 * Get all tasks
 * @return {Object} an object contains objects with
all tasks
 *
 * ### Example:
 * @codestart
 * TTI.Models.Task.findAll(function (tasks) {
 *   // do something with tasks
 * },
 * function (error) {
 *   // handle error here
 * });
 * task.save(function (task) {
 *   console.log(task);
 * });
 * @codeend
 */
findAll: function (success, error) {
  return db.allDocs(
    {
      include_docs: true // this is needed to
return not only task ID but task it self
    },
  );
}
```

```
function (err, response) {
    console.log('TTI.Models.Task.findAll() |
GET | err:', err, 'client:', response);

    if (response) {
        success(response);
    }
    else if (err) {
        error(err);
    }
}
);
},
```

The `findAll` method is responsible for retrieving an object with all the items from our database. Readers can have a look at the example usage in the comment on the preceding code listing.

```
/**
 * @function findOne
 * Find task by given ID
 * @param {String} task ID
 * Task object
 * @codestart
 * String (UUID)
 * @codeend
 *
 * @return {Object} an object contains requested task
 * @codestart
 * {
 *   id: String (UUID),
 *   hours: Number,
 *   cost: {
 *     rate: Number,
 *     total: Number
 *   },
 *   taskID: String,
 *   description: String,
 *   note: String
 * }
 * @codeend
 *
 * ### Example:
 * @codestart
```

```
error) {
    * TTI.Models.Task.findOne('UUID', function (success,
    *     // code goes here
    * });
    * @codeend
    */
  findOne: function (id, success, error) {
    return db.get(id, function (err, doc) {

      if (doc) {
        success(doc);
      }
      else if (err) {
        error(err);
      }

    });
  },
```

The `findOne` method is responsible for retrieving an object with a particular item from our database. Readers can have a look at the example usage in the comment on the preceding code listing.

```
/**
 * @function create
 * Create new task
 * @param {Object} task
 * Task object
 * @codestart
 * {
 *   hours: Number,
 *   cost: {
 *     rate: Number,
 *     total: Number
 *   },
 *   taskID: String,
 *   description: String,
 *   note: String
 * }
 *
 * {
 *   hours: 7,
 *   cost: {
 *     rate: 100,
 *     total: 700
```

```

*     },
*     taskID: 'JIRA-2789',
*     description: 'Implement new awesome feature!',
*     note: ''
*   }
* @codeend
*
* @return {Object} an object contains newly created
task UUID
* @codestart
* {
*   id: "8D812FF6-4B96-4D73-8D18-01FACEF33531"
*   ok: true
*   rev: "1-c5a4055b6c3edac099083cc0b485d4e3"
* }
* @codeend
*
* ### Example:
* @codestart
* var task = new TTI.Models.Task({ task object goes
here });
* task.save(function (task) {
*   console.log(task);
* });
* @codeend
*/
create: function (task, success, error) {
  return db.post(task, function (err, response) {
    console.log('TTI.Models.Task.create() | POST |
err:', err, 'client:', response);

    if (response) {
      success(response);
    }
    else if (err) {
      error(err);
    }
  });
},

```


The `create` method is responsible for creating a new item in our database. Readers can have a look at the example usage in the comment on the preceding code listing.

```
/**
 * @function update
 * Update task by given ID
 * @param {Object} task
 * Task object
 * @codestart
 * {
 *   _id: String (UUID),
 *   hours: Number,
 *   cost: {
 *     rate: Number,
 *     total: Number
 *   },
 *   taskID: String,
 *   description: String,
 *   note: String
 * }
 * @codeend
 *
 * @return {Object} an object contains updated task
UUID
 * @codestart
 * {
 *   id: "8D812FF6-4B96-4D73-8D18-01FACEF33531"
 *   ok: true
 *   rev: "1-c5a4055b6c3edac099083cc0b485d4e3"
 * }
 * @codeend
 *
 * ### Example:
 * @codestart
 * TTI.Models.Task.update({ task object goes here });
 * @codeend
 */
update: function (task, success, error) {
  return db.put(task, function (err, response) {
    console.log('TTI.Models.Task.update() | POST |
err:', err, 'client:', response);

    if (response) {
      success(response);
    }
  });
}
```

```

        else if (err) {
            error(err);
        }
    });
},

```

The update method is responsible for updating a particular item in our database. Readers can have a look at the example usage in the comment on the preceding code listing.

```

/**
 * @function destroy
 * Destroy task by given ID
 * @param {Object} task
 * Task object
 * @codestart
 * String (UUID)
 * @codeend
 *
 * @return {Object} an object contains destroyed task
UUID
 * @codestart
 * {
 *     id: "8D812FF6-4B96-4D73-8D18-01FACEF33531"
 *     ok: true
 *     rev: "1-c5a4055b6c3edac099083cc0b485d4e3"
 * }
 * @codeend
 *
 * ### Example:
 * @codestart
 * TTI.Models.Task.destroy('UUID', function (success,
getError, removeError) {
 *     // handle errors here
 * });
 * @codeend
 */
destroy: function (id, success, getError, removeError)
{
    return db.get(id, function (getErr, doc) {

        if (getErr) {
            getError(getErr);
        }
    }

```

```
        db.remove(doc, function (removeErr, response)
    {
        if (response) {
            success(response);
        }
        else if (removeErr) {
            removeError(removeErr);
        }
    });
    });
    },
    {
    }
    );
}
);
```

This `destroy` method is responsible for destroying a particular item in our database. Readers can have a look at the example usage in the comment on the preceding code listing.

Let's create a `client.js` file under `models` directory. In `Client` model we will keep all the task-related CRUD methods that operate on a local database. Create a bootstrap file:

```
steal(
    'jquery/model',

    function ($) {
        'use strict';

        $.Model('TTI.Models.Client', {
            init: function () {
                // create database clients or get it if exists.

                console.log('TTI.Models.Client.init() | idb://
clients | err:');

            },

            findAll: function () {

            },
        },
```

```
        findOne: function () {
        },
        create: function () {
        },
        update: function () {
        },
        destroy: function () {
        }
    },
    {
    }
);
```

Creating controllers

Let's create a `tasks.js` file under the `controllers` directory where we can handle all the application actions.

```
steal(
    'jquery/view/ejs',
    'jquery/controller',
    'tti/models/task.js'
).then(
    function ($) {
        'use strict';

        console.log('TTI.Controllers.Tasks');

        /**
         * @class TTI.Controllers.Tasks
         * Creates a new Tasks controller
         * @parent index
         * @constructor
         * @param {String} DOMELEMENT DOM element
         * @return {Object}
         */
```

```
    */
    $.Controller('TTI.Controllers.Tasks', {
      'init': function (element, options) {
        var self = this;

        $('title').text('Time Tracker | TTI');

        TTI.Models.Task.findAll(function (data) {
          if (!data.rows.length) {
            data.rows = [
              {
                doc: {
                  hours: '',
                  cost: {
                    total: ''
                  },
                  taskID: '',
                  description: 'No tasks so far!',
                  note: ''
                }
              }
            ];
          }

          self.element.html('tti/views/templates/tasks/
tasks.ejs', data.rows);

        });
      },

      '{TTI.Models.Task} created': function (Task, e, task) {
        console.log('task', task);
        console.log('this.element', this.element);
        $('tbody tr:last', this.element).after('tti/views/
templates/tasks/task.ejs', task);
        $('tbody tr:last', this.element).effect('highlight',
        {}, 3000);
      },

      '{TTI.Models.Task} destroyed': function (Task, e, task) {
        task.elements(this.element).remove();
      },

      '.add-task click': function () {
        this.element.append('tti/views/templates/tasks/add_
task.ejs', {}).find('.create-new-task-dialog-form').dialog({
          autoOpen: false,

```

```

        modal: true,
        buttons: {
            'Create New Task': function () {
                var self = this;

                window.task = new TTI.Models.Task({
                    hours: $('input[name="hours"]', this).
val(),
                    taskID: $('input[name="task-id"]',
this).val(),
                    cost: {
                        rate: 0,
                        total: 0
                    },
                    description:
$('input[name="description"]', this).val(),
                    note: $('input[name="note"]', this).
val()
                });

                window.task.save(function () {
                    $(self).dialog('destroy').remove();
                });
            },
            Cancel: function () {
                $(this).dialog('destroy').remove();
            }
        },
        close: function () {
            $(this).dialog('destroy').remove();
        }
    }).dialog('open');
}

});
}
);

```

Let's create the `clients.js` file under the `controllers` directory.

```

steal(
    'jquery/view/ejs',
    'jquery/controller'
).then(
    function ($) {
        'use strict';
    }
);

```

```
console.log('TTI.Controllers.Client');

/**
 * @class TTI.Controllers.Client
 * Creates a new Tasks controller
 * @parent index
 * @constructor
 * @param {String} DOMELEMENT DOM element
 * @return {Object}
 */
$.Controller('TTI.Controllers.Client', {
  'init': function () {

    $('title').text('Clients | TTI');

    var testData = [
      {
        name: 'The First Awesome Client!'
      },
      {
        name: 'The Second Awesome Client!'
      }
    ];

    this.element.html('tti/views/templates/clients.ejs',
testData);

  }

});

);
```

Let's create the router.js file under the controllers directory.

```
steal(
  'tti/controllers/navigation.js',
  'tti/controllers/client.js',
  'tti/controllers/tasks.js',
  'jquery/controller',
  'jquery/controller/route'
).then(
  function ($) {
    'use strict';

    /**
```

```

* @class TTI.Controllers.Router
* Creates application router
* @parent index
* @constructor
* @param {String} DOMELEMENT DOM element
* @return {Object}
*/
$.Controller('TTI.Controllers.Router', {
  init: function () {
    console.log('r init');
  },

  // the index page
  'route': function (e) {
    console.log('default route', e);
  },

  ':page route': function (data) {
    $('#content').empty().append('<div>');

    if (data.page === 'time-tracker') {
      new TTI.Controllers.Tasks('#content div');
    }
    else if (data.page === 'clients') {
      new TTI.Controllers.Client('#content div');
    }
  }
});

// create new Router controller instance
$('body').bind('TTI/db-ready', function () {
  new TTI.Controllers.Router(document.body);
});
}
);

```

Let's create the navigation.js file under the controllers directory.

```

steal(
  'jquery/view/ejs',
  'jquery/controller',
  'jquery/dom/route'
).then(
  function ($) {
    'use strict';

    /**
     * @class TTI.Controllers.Navigation

```



```
* Creates application main navigation controller
* @parent index
* @constructor
* @param {String} DOMELEMENT DOM element
* @return {Object}
*/
$.Controller('TTI.Controllers.Navigation', {
  init: function () {
    var navItems = [
      {
        name: 'Time Tracker',
        className: 'time-tracker'
      },
      {
        name: 'Invoice',
        className: 'invoice'
      },
      {
        name: 'Clients',
        className: 'clients'
      },
      {
        name: 'Reports',
        className: 'reports'
      },
      {
        name: 'Statistics',
        className: 'statistics'
      }
    ];

    this.element.html('tti/views/templates/navigation.
ejs', navItems);
  },
  '.time-tracker click': function (e) {
    $.route.attr('page', 'time-tracker');
  },
  '.clients click': function (e) {
    $.route.attr('page', 'clients');
  }
});
};
```

Creating views

Let's create the `views` folder under the `tti` directory and inside it, two directories: `styles` and `templates`.

In the `templates` directory create `client.ejs` file with the following content:

```
<h2>Clients List</h2>

<ol>
  <% $.each(this, function(i, client) { %>

    <li <%= ($el) -> $el.model(client) %>>
      <strong><%= client.name %></strong>
    </li>

  <% }) %>
</ol>
```

In the `templates` directory create the `navigation.ejs` file with the following content:

```
<% $.each(this, function(i, item) { %>

  <li class="<%= item.className %>">
    <%= item.name %>
  </li>

<% }) %>
```

In the `templates` directory create the `tasks` directory. Create the `tasks.ejs` file with the following content:

```
<table summary="Time Tracker list of tasks.">
  <thead>
    <tr>
      <th scope="col">Hours</th>
      <th scope="col">Cost</th>
      <th scope="col">Task ID</th>
      <th scope="col">Description</th>
      <th scope="col">Note</th>
    </tr>
  </thead>
  <tbody>
    <% $.each(this, function(i, task) { %>
      <tr <%= ($el) -> $el.model(task) %>>
        <td>
```

```
        <%= task.doc.hours %>
    </td>
    <td>
        <%= task.doc.cost.total %>
    </td>
    <td>
        <%= task.doc.taskID %>
    </td>
    <td>
        <%= task.doc.description %>
    </td>
    <td>
        <%= task.doc.note %>
    </td>
</tr>
<% }) %>
</tbody>
</table>

<span class="add-task">Add Task</span>
```

In the tasks directory create the task.ejs file with the following content:

```
<tr <%= ($el) -> $el.model(task) %>>
  <td>
    <%= task.hours %>
  </td>
  <td>
    <%= task.cost.total %>
  </td>
  <td>
    <%= task.taskID %>
  </td>
  <td>
    <%= task.description %>
  </td>
  <td>
    <%= task.note %>
  </td>
</tr>
```

In the `tasks` directory create the `add_task.ejs` file with the following content:

```
<div class="create-new-task-dialog-form" title="Create New Task">
  <form>
    <fieldset>
      <label><span>Hours</span> <input type="text" name="hours"
/></label>
      <label><span>Task ID</span> <input type="text" name="task-
id" /></label>
      <label><span>Description</span> <input type="text"
name="description" /></label>
      <label><span>Note</span> <input type="text" name="note"
/></label>
    </fieldset>
  </form>
</div>
```

In the `styles` directory create two directories: `css` and `sass`.

In the `sass` directory create the `tti.scss` file with the following content:

```
@import 'reset';
@import 'static';
@import 'mixins';
@import 'skelton';
```

In the `sass` directory create the `_static.scss` file with the following content:

```
$blue: #5C94BF;
$black: #3E4246;
$white: #F6F6F7;
$vlGreen: #B7D190;
$lGreen: #9CBA6E;
$dGreen: #424A38;
$mGrey: #5B5B5B;
$yellow: #F8AE03;
$lBlue: #167BBE;
$dBlue: #0E69B3;
$gBlue: #7489A1;
```

In the sass directory create the `_mixins.scss` file with the following content:

```
@mixin link {
  color: $lGreen;
  cursor: pointer;
  text-decoration: none;

  &:hover {
    text-decoration: underline;
  }
}

@mixin borderRadius($topLeft, $topRight, $bottomRight, $bottomLeft) {
  -moz-border-radius-topleft: $topLeft;
  -moz-border-radius-topright: $topRight;
  -moz-border-radius-bottomright: $bottomRight;
  -moz-border-radius-bottomleft: $bottomLeft;
  -webkit-border-radius: $topLeft $topRight $bottomRight
  $bottomLeft;
  border-radius: $topLeft $topRight $bottomRight $bottomLeft;
}

@mixin button {
  @include borderRadius(5px, 5px, 5px, 5px);
  display: inline-block;
  padding: 0 7px;
  line-height: 20px;
  height: 20px;
  cursor: pointer;
}
```

In the sass directory create the `_reset.scss` file with the following content:

```
// http://meyerweb.com/eric/tools/css/reset/
// v2.0 | 20110126
// License: none (public domain)

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, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
// HTML5 display-role reset for older browsers
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}

body {
    line-height: 1;
}

ol, ul {
    list-style: none;
}

blockquote, q {
    quotes: none;
}

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

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

textarea {
    min-height: 100px;
}
```

In the sass directory create the `skelton.scss` file with the following content:

```
html,
body {
  color: $black;
  background: $white;
  font: 11px/18px "Helvetica Neue", Helvetica, Verdana, sans-serif;
}

input,
select,
textarea {
  font: 12px/18px "Helvetica Neue", Helvetica, Verdana, sans-serif;
}

table {
  width: 100%;
  border: 1px solid $gBlue;

  thead {
    color: $white;
    background: $blue;

    tr {

      &:last-child {
        @include borderRadius(5px, 5px, 5px, 5px);
      }

      th {
        padding: 7px 0;
      }
    }
  }

  tbody {

    tr {

      td {
        padding: 3px 0;
        border-bottom: 1px solid $gBlue;
        text-align: center;
      }
    }
  }
}
```

```
    }
  }
}

#container {
  width: 1100px;
  margin: 0 auto;

  #header {
    padding: 10px;
    height: 50px;

    #main-navigation {

      li {
        @include link;
        margin-right: 7px;
        display: inline-block;
      }
    }
  }

  h2 {
    font-size: 14px;
  }

  ol {
    margin: 7px;
    list-style-type: decimal;
    list-style-position: inside;
  }

  .add-task {
    @include button;
    margin-top: 20px;
    color: $white;
    background: $lGreen;

    &:hover {
      background: $yellow;
    }
  }
}
```


Creating a bootstrap

In the root directory let's create the `index.html` file with the following code.

Bootstrap is responsible for loading all the files needed by the application to run.

```
<!doctype html>

<html>
  <head>
    <title>TTI</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <div id="container">
      <header id="header">
        <nav id="main-navigation">
          <ul></ul>
        </nav>
        <div id="breadcrumb"></div>
        <nav id="secondary-navigation"></nav>
      </header>

      <div id="content">
        <div><p>Loading...</p></div>
      </div>

      <footer id="footer">

      </footer>

    </div>

    <script src="steal/steal.js?tti"></script>

  </body>
</html>
```

In the `ttd` directory create the `ttd.js` file with the following code:

```
steal(
  function ($) {
    console.log('ttd.js');
  },
  'vendors/jquery_ui/css/smoothness/jquery.ui.core.css',
  'vendors/jquery_ui/css/smoothness/jquery.ui.dialog.css',
  'vendors/jquery_ui/css/smoothness/jquery.ui.theme.css',
```

```
'tti/views/styles/css/tti.css',
'tti/models/task.js',
'tti/models/client.js',
'tti/controllers/tasks.js',
'tti/controllers/router.js',
'tti/controllers/navigation.js'
).then(
  'vendors/jquery_ui/jquery.ui.core.js'
).then(
  'vendors/jquery_ui/jquery.effects.core.js'
).then(
  'vendors/jquery_ui/jquery.effects.highlight.js'
).then(
  'vendors/jquery_ui/jquery.ui.widget.js'
).then(
  'vendors/jquery_ui/jquery.ui.position.js',
  'vendors/jquery_ui/jquery.ui.dialog.js'
).then(
  function ($) {
    new TTI.Controllers.Navigation('#main-navigation ul');
  }
);
```

Running the application

In order to run our application we convert the SASS files into the CSS file which can be read by web browsers.

We used SASS instead of plain CSS to split code into many small files for better readability and better code re-use. This aspect is very important, especially in big applications.

SASS can be installed by executing `$ gem install sass` command or downloading it from the Git repository at <http://sass-lang.com/download.html>.

To compile SASS code into CSS code, go to views folder and type:

```
$ sass --watch sass:css
```

Then run the web server and navigate to `index.html`.

Summary

In this chapter we learned how to build a JavaScriptMVC application and organize code and workflow to create web applications in a more efficient and less error prone way.

Index

Symbols

\$.Class

- about 50, 52
- first parameter 52
- life cycle 53
- method override 52
- second parameter 52
- third parameter 52

\$.Controller plugin 57

\$.cookie plugin 58

\$.Drag and \$.Drop plugin 62

\$.fn.compare plugin 58, 59

\$.fn.selection plugin 59

\$.fn.within plugin 60

\$ gem install sass command 103

\$.Model 53, 55

\$.Object plugin

- about 62
- same method 62

\$.Observe plugin 64

\$.Range plugin 60

\$.route plugin 61

\$.String plugin

- about 65
- deparam method 65

\$.toJSON plugin 66

\$.Vector plugin 66

\$.View

- about 55
- embedded 55
- external 56
- sub-templates 56

@add, type directive 39

@alias, tag directive 39

@attribute, type directive 38

@author, tag directive 39

@class, type directive 39

@codestart, tag directive 39

@constructor, tag directive 39

@demo, tag directive 39

@description, tag directive 39

@download, tag directive 39

@function, tag directive 39

@hide, tag directive 39

@iframe, tag directive 39

@image, tag directive 39

@inherits, tag directive 39

@page, tag directive 38

@param, tag directive 39

@parent, tag directive 39

@plugin, tag directive 39

@prototype, tag directive 39

@return, tag directive 39

@scope, tag directive 39

@static, tag directive 39

@tag, tag directive 39

@test, tag directive 39

@type, tag directive 39

A

application

- changes under VCS, tracking 80
- project, setting up 78
- structure 80
- wireframes, preparing 75-78
- writing 74

B

bootstrap

creating 102

Breadcrumb 75

C

CanJS 7, 16

class method 50

cleanjs command 71

code cleaner 71

CommonJS 26

console.log() function 70

Content Delivery Network (CDN) 10

controllers

creating 89-93

create method 86

Create, Read, Update, Delete (CRUD) 22

D

deparam method 65

dependency management 69

destroy method 88

Distributed Version Control Systems (DVCS) 79

documentation

generating 40

writing 34

DocumentJS

about 7, 15, 33

documentation, URL 33

documentation, writing 34

features 33

tag directives 39

tags 34

type directives 38

types 34

working 34

Document Object Model (DOM) tree 9

DoneJS 8

DOM helpers

\$.cookie plugin 58

\$.fn.compare plugin 58, 59

\$.fn.selection plugin 59

\$.fn.within plugin 60

\$.Range plugin 60

\$.route plugin 61

about 57

E

EJS

URL 55

EnvJS

used, for running tests 48

F

findAll method 83

findOne method 84

free code hosting solutions

URL 80

free issue-tracking tools

URL 74

functional testing 43

FuncUnit

about 7, 15, 43

documentation, URL 43

QUnit, URL 43

source code, URL 43

G

getType instance method 52

getType method 52

Git

reference, URL 12

submodules, URL 13

URL, for installing 12

I

IndexedDB 81

init method 82

installation, JavaScriptMVC

code, pulling from Git repositories 12

package, downloading 11

Vagrant used 14

verifying 13

ways 11

instance method 50

J

Jaml

URL 55

JavaScriptMVC

about 7, 8

advantages 10

API, URL 14

application, building 17

application, code 28

architecture 15

CanJS 7

code examples, URL 14

controller 26, 27

disadvantages 10

documentation, URL 14

DocumentJS 7

fixtures 22, 23, 24

framework, URL 10

FuncUnit 7

installing 11

Loader 18

MIT license, exceptions 8

model, adding to application 19, 20

MVC 9

official forum, URL 14

real-world examples 10

repository, URL 8

routing 27, 28

Stack Overflow questions, URL 14

StealJS 7

system architecture approach 9

Todo list application 17

tutorials, URL 14

URL, for downloading package 11

view 25, 26

website, URL 8

Jenkins

URL 48

jQuery

8

jQueryMX

49

jQuery templates

URL 55

JSLint code quality tool

URL 71

JVMC. *See* JavaScriptMVC

L

language helpers

\$.Object plugin 62

\$.Observe plugin 64, 65

\$.String plugin 65

\$.toJSON plugin 66

\$.Vector plugin 66

about 62

logger 70

M

Markdown

URL 33

Maven

URL 48

Micro

URL 55

MIT license

exceptions 8

mockups 75

models

creating 81-88

model-view-controller (MVC) framework 7

module method 45

MIT license

exceptions 8

multi-page application 9

O

ok signature 46

open signature 45

Oracle VM VirtualBox

URL, for installing 14

P

PhantomJS

used, for running tests 47

PouchDB

URL, for downloading 81

prototypes 75

Q

QUnit

URL 43

R

Rhino 8

S

same method 62, 63

SASS 103

sass directory 98

Scrum

URL 74

Search Engine Optimization (SEO) 10

Selenium

used, for running tests 47

Simple JavaScript Inheritance

URL 50

special events plugins

\$.Drag and \$.Drop plugin 62

S signature 46

steal.build 71

steal.clean 71

steal.dev 70

StealJS

about 7, 16, 69

features 16, 17, 69

T

tag directives

@alias 39

@author 39

@codestart 39

@constructor 39

@demo 39

@description 39

@download 39

@hide 39

@iframe 39

@image 39

@inherits 39

@param 39

@parent 39

@plugin 39

@return 39

@scope 39

@tag 39

@test 39

@type 39

about 38

templates directory 95

tests

creating 44, 45

module method 45

ok signature 46

open signature 45

running 46

S signature 46

test signature 45

test signature 45

tests, running

EnvJS used 48

integration 48

PhantomJS used 47

Selenium used 47

web browser used 46

Time tracking and invoicing for freelancers.

See TTI

Todo application 71

Todo list application

about 17

code 28

controller 26

fixtures 22, 24

loader 18, 19

model 19, 20

routing 27, 28

view 25, 26

TTI 73

type directives

@add 39

@attribute 38

@class 39

@function 39

@page 38

@prototype 39

@static 39

about 38

U

update method 87

V

Vagrant

URL, for installing 14

views

creating 95-100

W

web browser

used, for running tests 46

wireframes

clients main page 78

invoice main page 77

preparing 75



Thank you for buying **Learning JavaScriptMVC**

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.

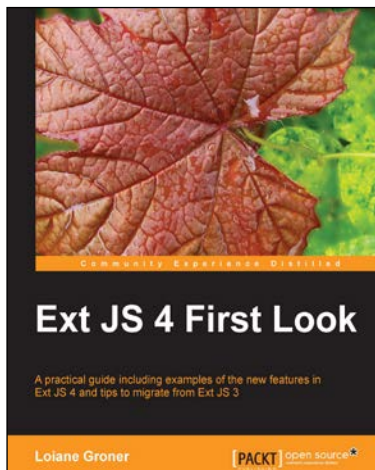
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

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.



Ext JS 4 First Look

ISBN: 978-1-849516-66-2 Paperback: 340 pages

A practical guide including examples of the new features in Ext JS 4 and tips to migrate from Ext JS 3

1. Migrate your Ext JS 3 applications easily to Ext JS 4 based on the examples presented in this guide
2. Full of diagrams, illustrations, and step-by-step instructions to develop real world applications
3. Driven by examples and explanations of how things work



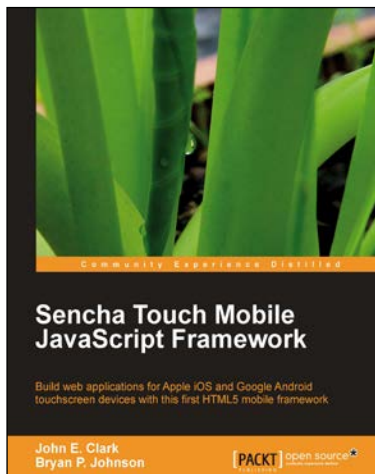
Ext JS 4 Web Application Development Cookbook

ISBN: 978-1-849516-86-0 Paperback: 488 pages

Over 110 easy-to-follow recipes backed up with real-life examples, walking you through basic Ext JS features to advanced application design using Sencha's Ext JS

1. Learn how to build Rich Internet Applications with the latest version of the Ext JS framework in a cookbook style
2. From creating forms to theming your interface, you will learn the building blocks for developing the perfect web application
3. Easy to follow recipes step through practical and detailed examples which are all fully backed up with code, illustrations, and tips

Please check www.PacktPub.com for information on our titles

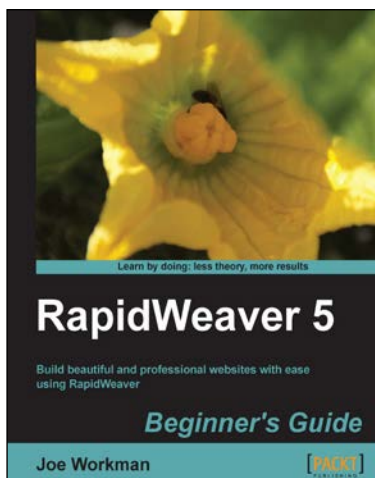


Sencha Touch Mobile JavaScript Framework

ISBN: 978-1-849515-10-8 Paperback: 316 pages

Build web applications for Apple iOS and Google Android touchscreen devices with this first HTML5 mobile framework

1. Learn to develop web applications that look and feel native on Apple iOS and Google Android touchscreen devices using Sencha Touch through examples
2. Design resolution-independent and graphical representations like buttons, icons, and tabs of unparalleled flexibility
3. Add custom events like tap, double tap, swipe, tap and hold, pinch, and rotate



RapidWeaver 5 Beginner's Guide

ISBN: 978-1-849692-05-2 Paperback: 362 pages

Build beautiful and professional websites with ease using RapidWeaver

1. Jump into developing websites on your Mac with RapidWeaver.
2. Step-by-step tutorials for novice users to get your websites built and published online.
3. Advanced tips and exercises for existing RapidWeaver users.
4. A great A-Z guide for building websites irrespective of your level of expertise.

Please check www.PacktPub.com for information on our titles