



Community Experience Distilled

Getting Started with Backbone Marionette

Build large-scale JavaScript applications with Backbone
Marionette quickly and efficiently

Raymundo Armendariz
Arturo Soto

[PACKT] open source*
PUBLISHING community experience distilled

Getting Started with Backbone Marionette

Build large-scale JavaScript applications with
Backbone Marionette quickly and efficiently

Raymundo Armendariz

Arturo Soto

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

Getting Started with Backbone Marionette

Copyright © 2013 Packt Publishing

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

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

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

First published: January 2014

Production Reference: 1030114

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

ISBN 978-1-78328-425-2

www.packtpub.com

Cover Image by Artie Ng (artherng@yahoo.com.au)

Credits

Authors

Raymundo Armendariz

Arturo Soto

Reviewers

Andrea Puddu

Michel Richard

Sam Saccone

Acquisition Editors

Martin Bell

Meeta Rajani

Llewellyn Rozario

Lead Technical Editor

Vaibhav Pawar

Technical Editors

Pooja Nair

Humera Shaikh

Copy Editors

Alisha Aranha

Gladson Monteiro

Project Coordinator

Michelle Quadros

Proofreader

Lucy Rowland

Indexers

Monica Ajmera Mehta

Tejal Soni

Graphics

Abhinash Sahu

Production Coordinators

Adonia Jones

Komal Ramchandani

Cover Work

Adonia Jones

About the Authors

Raymundo Armendariz is a web developer with over nine years of experience in developing applications for the government and different industries such as automotive and manufacturing.

In the past two years, he has spent most of his time on frontend development with Backbone and Marionette, and building single-page applications.

Arturo Soto is a technical architect and developer. His work focuses on developing enterprise-level applications, especially web applications. His professional interests include software design patterns, agile practices, and multiple technologies, such as WCF, ASP.NET MVC, OData, Web API, HTML5, and JavaScript.

To our wives and families for their love and motivation and to our friends for their help and support.

About the Reviewers

Andrea Puddu (Twitter: @nuragic) is a web engineer from Sardinia, Italy. After a few years of working in his country, he moved to Madrid, Spain, where he worked in marketing and advertising companies, IT consulting firms, and tech startups. He has studied and worked with server languages and databases. He has now become a frontend expert because that's what he loves to do. In his spare time, he likes to contribute to open source software; in fact, he is a committer of the MarionetteJS project. He is also a drummer in a rock band that he started: The Ancient Secrets of Levitation.

I'd like to wholeheartedly thank my parents who have supported me in my professional career. I also want to thank Carol, my girlfriend, who always helps me to make the best decisions. And last but not least, many thanks to my mate Tony, who always helps me out with English!

Michel Richard is a full-stack web developer born and raised in Kamloops, BC, Canada and is now residing in New York city. He earned his degree from McGill University and has a double major in Computer Science and Psychology. He is a huge fan of open source projects and contributes to them whenever possible. Michel has been working with Backbone and Marionette for the past two years and maintains a Yeoman Marionette generator project on GitHub. Michel currently works at Saks Inc., where he is the Director of Frontend Development. He can be found on GitHub as mrichard and on Twitter as MicheLeeRichard.

Sam Saccone is a creator and a problem solver. He spends his time working on open source projects and building applications at MojoTech. MojoTech builds web and mobile apps for big and soon to-be-big companies.

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
<hr/>	
Chapter 1: Starting with Backbone Marionette	5
<hr/>	
Introducing Marionette.js	5
Backbone needs Marionette.js	6
Key benefits of Marionette.js	6
Building large applications	7
Incremental use	7
Installing Marionette.js	8
Text editor	8
Web browser	8
Prerequisites	8
Getting Marionette.js	9
Documentation	9
Summary	9
<hr/>	
Chapter 2: Our First Application	11
<hr/>	
Introduction to what we are building	11
Setting up our development environment	13
The Backbone.Marionette.Application object	16
Backbone already has a router!	16
Summary	19
<hr/>	
Chapter 3: Marionette View Types and Their Use	21
<hr/>	
Marionette.View and Marionette.ItemView	22
Handling events in the views	26
UI and templates	27
Marionette.CollectionView	29
Marionette.CompositeView	30

Building the layout of our application with Marionette.Layout	32
Extending Marionette views	33
Summary	34
Chapter 4: Managing Views	35
Understanding the Marionette.Region object	36
Using the Marionette.RegionManager object	40
Using the Backbone.BabySitter object	42
Taking advantage of the Marionette.Renderer object	43
Improving the performance of the application with TemplateCache	44
Summary	45
Chapter 5: Divide and Conquer – Modularizing Everything	47
Applying the divide and conquer principle	47
Modularizing single-page applications	48
Getting started with modules	49
Splitting modules into multiple files	50
Implementing initializers and finalizers	51
Working with subapplications	51
Using the route filter	53
Memory considerations	55
Summary	56
Chapter 6: Messaging	57
Understanding the event aggregator	57
Using the event aggregator of Marionette.js	58
Making applications more extensive with an event aggregator	59
Getting started with Commands	61
Setting up the RequestResponse object	63
Summary	65
Chapter 7: Changing and Growing	67
Using AMD	67
Using the Require.js library	68
Configuring Require.js	68
Defining our application module	70
Writing the subapplications using Require.js	71
Modularizing all your components	72
Adding the text plugin	73
Structuring your files	74
Using handlebars as a template engine in Marionette	76
Summary	77
Index	79

Preface

Backbone Marionette is a composite application library for `Backbone.js`, which aims to simplify the construction of large-scale JavaScript applications. It is a collection of common design and implementation patterns found in the applications that we build with Backbone, and includes pieces inspired by composite application, event-driven, and messaging architectures.

What this book covers

Chapter 1, Starting with Backbone Marionette, is an introduction to what Marionette is and the problems it aims to solve. In this chapter, we also learn about its prerequisites, download sources, and documentation.

Chapter 2, Our First Application, introduces three main concepts of Marionette – the application, controller, and router objects – and details the process of building a small application.

Chapter 3, Marionette View Types and Their Use, digs deep into the view types that Marionette has and how to use them.

Chapter 4, Managing Views, reviews the view management that goes from firing a view, closing it, and re-opening it. We will also introduce some useful objects, such as the `Renderer` object and the `TemplateCache` object, that are very valuable in order to build an application.

Chapter 5, Divide and Conquer – Modularizing Everything, talks about how to modularize an application and break it into small subapplications. Being able to do this will increase its productivity as the modules allow the adding of new functionality without affecting the existing code.

Chapter 6, Messaging, explains that in order to build a loosely coupled application, the components need to know very little about each other; however, these components still need to work together. In this chapter, we also learn how to archive this through messages and events.

Chapter 7, Changing and Growing, helps us to learn how to manage a problem that comes with large-scale applications: the file explosions, and how to keep a clean structure.

What you need for this book

A modern browser and a text editor are all you need to follow the examples of this book. You will find detailed instructions of how to set up your development environment and also where to get the Marionette and its dependencies in the book.

Who this book is for

If you are a web application developer interested in using Backbone Marionette for a real-life project, this book is for you. Knowledge of JavaScript and working knowledge of `Backbone.js` are prerequisites.


Conventions


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

Code words in text are shown as follows: "The `Marionette.ItemView` is "

A block of code is set as follows:

```
var CategoryView = Backbone.Marionette.ItemView.extend({
  tagName : 'li',
  template: "#categoryTemplate",
});
```

 [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

Starting with Backbone Marionette

This practical guide provides clear steps to the basics of writing scalable applications using `Marionette.js`. As you progress through the initial examples, you will develop an understanding of how the framework components work together to create a composite application. But before we go through in-depth examples, here are some things that you will find in this introductory chapter:

- Description and characteristics of `Marionette.js`
- The role of `Marionette.js` in the Backbone applications
- Benefits of the framework
- An overview of architecture and scalability
- Instructions for installation and documentation

Introducing `Marionette.js`

A composite application library for `Backbone.js` is `Backbone.marionette`, also known as `Marionette.js`. It gives us the core constructs and simplifies many of the patterns and practices that your JavaScript applications need to be scalable.

Backbone needs Marionette.js

An increasingly popular framework for building single-page and small- to mid-sized applications is `Backbone.js`. It provides a great set of building blocks to organize your frontend development and build applications that support mobile devices. However, it leaves much of the application design, architecture, and scalability to developers. Nevertheless, `Marionette.js` fills in some blanks that `Backbone.js` doesn't provide by itself and gives us conventions that you can take advantage of to build your own custom objects. Simply put, `Marionette.js` makes your life easier when you are developing the Backbone applications.

Key benefits of Marionette.js

Adding a lot of key patterns and tools used to create real-world applications, `Marionette.js` found its place in Backbone. The following are some of the benefits that you can find within this framework:

- Structure, organization, and patterns
- Composite application architecture
- Event-driven architecture with the event aggregator
- Boilerplate for views can be reduced
- Enterprise messaging pattern influence
- Modularization options
- Incremental use; because it's not an all or nothing framework, which means that you can use just the components you need
- Support for nested views and layouts within visual regions
- Built-in memory management and zombie killing in views, regions, and layouts

A lot of application infrastructural components needed to build an application of any module size is provided by `Marionette.js`.

A wide set of objects are provided by `Marionette.js` that facilitate the creation of well-structured applications of virtually any size and complexity. It achieves this goal by providing a collection of common design and implementation patterns found in the applications that the creator, Derick Bailey, used to develop the modular Backbone applications.

Building large applications

When planning the architecture for your application, you normally try to think ahead as much as possible, attempting to achieve a decoupled architecture with functionality broken down into independent modules, and to avoid making direct calls between these modules. Therefore, you can add and remove modules without affecting its behavior.

"The secret to building large apps is never build large apps. Break your applications into small pieces. Then, assemble those testable, bite-sized pieces into your big application"

– Justin Meyer, author of JavaScriptMVC.

Consider that components tied to each other are difficult to change and scale without affecting each other. A very incremental and modular approach is provided by `Marionette.js`, using the module definition. It relies on the event aggregator to send messages between the modules to coordinate functionality from other parts of the system, without adding direct references to them among many more object types that facilitate the application's design.

Incremental use

This is one of the basic concepts that the creator of `Marionette.js` used to create the framework. An incremental and modular approach is facilitated by `Marionette.js`, providing the application object and the module architecture to scale applications across subapplications, features, and files. All of them are built to stand alone and to work with the core pieces of Backbone to accomplish the application's specific needs.

"The key is to acknowledge from the start that you have no idea how this will grow. When you accept that you don't know everything, you begin to design the system defensively ... You should spend most of your time thinking about interfaces rather than implementations."

– Nicholas Zakas, author "High-performance JavaScript websites"

One of the main benefits of `Marionette.js` is that you don't need to use all of its components. A jQuery-jQuery UI comparison can also be applied here. The fact that you use the jQuery calendar by any means forces you to use the entire UI library. The same can be applied to Marionette because the fact that you use just one of its components that makes sense for your application does not obligate you to use the other components of Marionette.

Installing Marionette.js

We will go over how to download and set up a development environment so that you can get started with `Marionette.js` in some easy steps. If you're already comfortable with installing `Marionette.js`, you may want to skip the remaining parts of this chapter. Feel free to jump to *Chapter 2, Our First Application*.

Text editor

While building large and scalable applications, you will spend most of your time on a code editor. We recommend that you use an editor that works for you. Notepad++ or Sublime Text are definitely good options. Sublime Text already has a lot of snippets and packages that will help you in your development.

Web browser

Working with complex client-side applications requires a good set of developer tools. For the purpose of this guide, we will use mostly Google Chrome and Mozilla Firefox, but all the code and examples should work in all modern browsers (IE9+, Opera, and Safari).

We will use `jsfiddle.net` in order to show you the running examples. The use of this site is pretty simple and most of the time, you will only need to run the code to see it in action.

Prerequisites

At the time of this writing, the current stable version of `Marionette.js` is v1.3.0 and it relies on the following libraries:

- `JSON2.js`
- `jQuery` (v1.7, v1.8, and v1.9)
- `Underscore.js` (v1.4.4)
- `Backbone.js` (v1.0.0)
- `backbone.wregr.js`
- `backbone.babysitter.js`

We would like to point that having basic knowledge on Backbone and Underscore will help you to get the best out of this book and to understand the benefits of Marionette over plain Backbone development.

Getting Marionette.js

The best way to get the latest build of `Marionette.js` is by going to the project website, <http://marionettejs.com/>.

They have a **Pre-packaged** option. The `.zip` contains all of the files that you need to get started with `Marionette.js`, including `Backbone`, `jQuery`, and other prerequisites. You can also download the `Marionette.js` file without all the dependencies and just use the CDN versions of these libraries if they are available.

Documentation

You can download the documentation for each piece of `Marionette.js` from <https://github.com/Marionette.js/Marionette.js/tree/master/docs>. The documentation is split into multiple files. The annotated source code can be found at <http://marionettejs.com/docs/backbone.marionette.html>. You can find articles, blog posts, presentations, FAQs, and more on its wiki page, <https://github.com/marionettejs/backbone.marionette/wiki>.

Derick Bailey, the creator of `Marionette`, has created a sample application that can be used as a reference for building the `Backbone` applications with `Marionette.js`. The name of the application is `BBCloneMail` and it is a great example to demonstrate a composite application. You can find `BBCloneMail` online at <http://bbclonemail.herokuapp.com> and the source code at <http://github.com/derickbailey/bbclonemail>.

Summary

In this chapter, we looked at some of the core concepts and benefits that `Marionette.js` offers for building scalable applications. We also provided links to find, download, and install the basic tools needed to perform your development. In the next chapter, you will be introduced to the components or building blocks that make up the `Marionette.js` applications.

2

Our First Application

In the previous chapter, we learned what Marionette is, where to find the source code and documentation, and other useful resources that will help us to learn more about Marionette. But we believe the best way to learn something is by putting it into practice. So in this book, we will build an application with moderate complexity, that is, it is complex enough to break the Hello World! barrier, allowing us to discover the benefits that Marionette has to offer, but simple enough to complete it with in this book. We will show some standalone code snippets to introduce you to each new concept; however most of the time we will stick to the application code.

In this chapter, we will review how to set up your development environment in order to build our first application. We will also learn three important parts of `Marionette.js`: the marionette router, marionette controller, and marionette application.

Introduction to what we are building

The application that we will be building in this book is a website for a book store. We should be able to perform the following actions on the website:

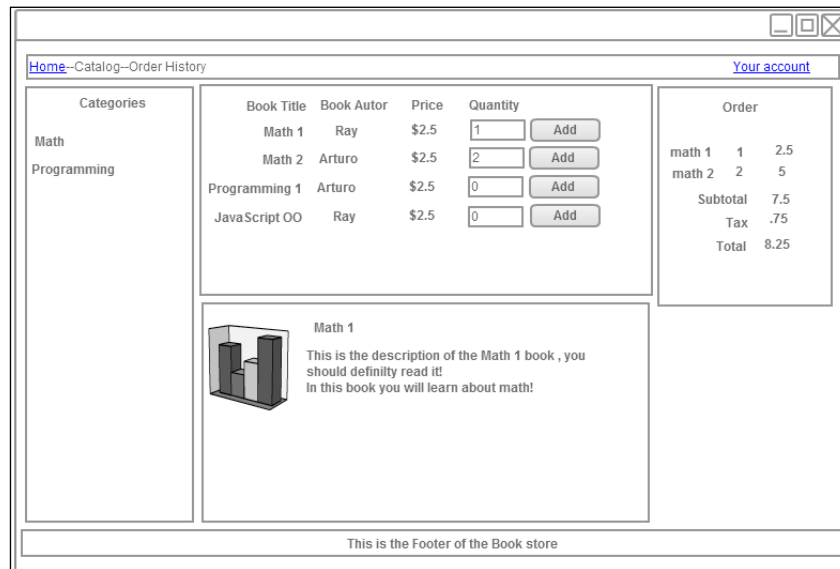
- Display a list of book categories
- Select a category and display the related books
- Present a description, price, and other important details of the book
- Add books to the shopping cart
- Display the shopping cart items

The website that we are going to build is just an example application. It's mandatory to follow the structure proposed in this book, as every application has different needs. Nevertheless, it's a good starting point and our idea is to show how each component of Marionette solves a problem and how to make its components work together.

Also, keep in mind that we will give attention to the Marionette components of the code, explaining in detail their benefits, and to adding them to the application. However, we will not dive deep into Backbone details such as `Backbone.Model` and `Backbone.Collection`, which are the core components of Backbone, as knowledge of this is already assumed.

One of the concepts that Marionette adds to Backbone is that of an application object—`Backbone.Marionette.Application`. We will start this book with this topic because the object will be the container of all of your Backbone views and models. One of its responsibilities is, before the user starts interacting with the website, it must initialize some of the components, such as the `Backbone.Router` component, that will be listening to the route (URL) changes of our application. This object provides some handy methods to perform this initialization. But, before we dig deeper into details, let's first take a look at what we are building.

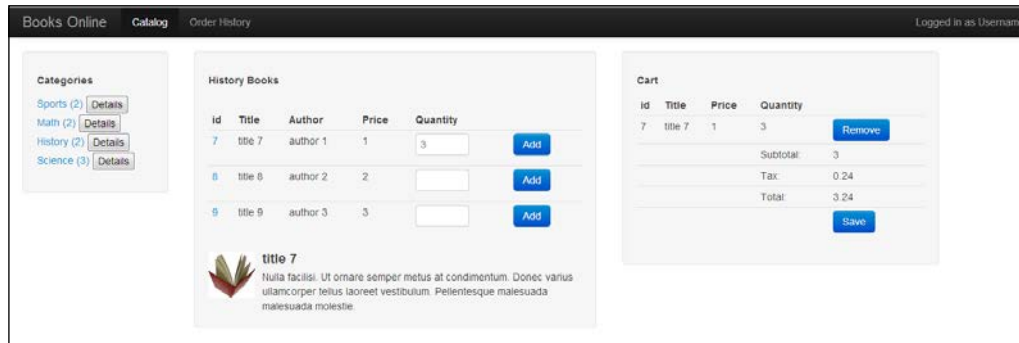
The following screenshot helps us to illustrate the structure of the book store application that we are going to build:



We have a navigation section that provides the categories of the books. Then in the middle, we have two sections. The one on top is the list of books by name, author, and price. This section also allows users to order books.

The second section, in the center of the screen, will show a description of each book as the user selects from the list on top. Finally, to the right of the screen, we have the Order section that will contain the details about our order.

At the end of the book, the application should look like the following screenshot:



The goal of this chapter is to build the foundation of the book store website and a part of that foundation is to have the `Backbone.Marionette.Application` object working with enough functionality so that we can call it an application. Our philosophy is to take small steps at a time and then check where we stand. So let's get started!

Setting up our development environment

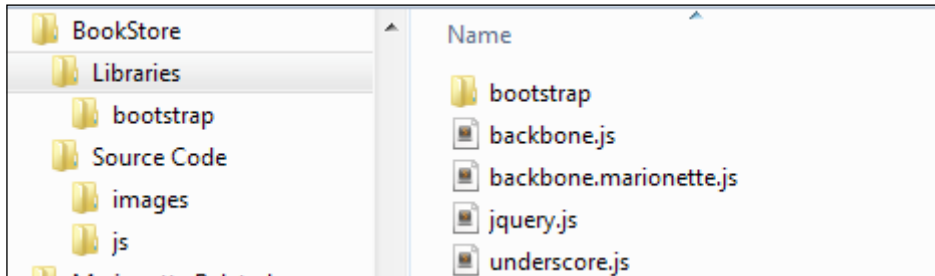
As we will be building an application together, we need to set up our development environment. The following are the steps to do it:

1. Create a folder and name it `Bookstore`.
2. Inside this folder, create two new folders—one named `Source Code` and the other `Libraries`.
3. In the `Libraries` folder, place the following four libraries:
 - `Underscore.js`
 - `jQuery.js`
 - `Backbone.js`
 - `Backbone.Marionette.js`

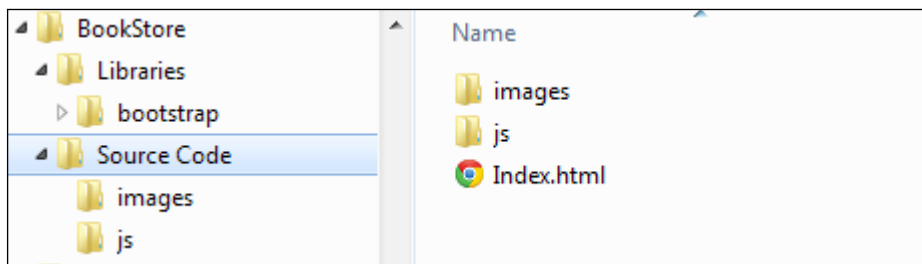
For styling purposes, we will use Twitter bootstrap v2. Download the default package, unzip it, and place the entire unzipped bootstrap folder beside the `.js` files inside the `Libraries` folder.

4. In the `Source Code` folder, create a new folder with the name `js` as it will be the location where we will save all our JavaScript files.

5. Under the `Source Code` folder, create an HTML file and name it `Index.html`. It should be placed at the same level as the `js` folder.
6. Make sure that your folder structure looks like the following screenshot and that you have the right library files inside the `Libraries` folder.




Your `Source Code` folder should look like the following screenshot:



We are building a single-page application and in this section, we are about to build the initial HTML page structure for our application. It is the HTML file that will be rendered by the server the first time a user types the URL of the site.

7. Open the `Index.html` file in your preferred code editor.
8. To avoid the tedious task of writing the HTML file manually for this chapter, we have made it available for you at <http://jsfiddle.net/>. The code is available at http://jsfiddle.net/rayweb_on/hsrv7/11/.

 **jsfiddle.net**—if you don't know it already, this is an excellent tool to test the small parts of your JavaScript code and share your snippets with ease.

I'm sure that if you are reading a Marionette book, it is because you have enough experience to put the CSS and JS tags in the right place. So feel free to skip the following steps.

9. Copy the **CSS** section and paste it into the `<head>` section of the HTML file.
10. Copy the **HTML** section and paste it into the `<body>` section of the HTML file.
11. At <http://jsfiddle.net/>, the scripts are already included for you. But in our local environment, we have to add them. We will do it just at the bottom of the `<html>` tag, but still inside the `<body>` tag.
12. When you are done with copying the initial structure, your HTML file should look like the following screenshot (the style script and the template script are collapsed in the screenshot). In this chapter, we will be using the console of your browser and we won't be interacting with the HTML file for now, but it's important that your `Index.html` file follows the structure shown in the following screenshot:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bootstrap, from Twitter</title>
  <link href="../../Libraries/bootstrap/css/bootstrap.min.css" rel="stylesheet" media="screen">
  <style type="text/css">
</style>
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="navbar-inner">
      <div class="container-fluid">
        <a class="brand" href="#">Books Online</a>
        <div class="nav-collapse collapse">
          <p class="navbar-text pull-right">
            Logged in as <a href="#" class="navbar-link">Username</a>
          </p>
          <ul class="nav">
            <li class="active"><a href="#">Catalog</a></li>
            <li><a href="#history/orders">Order History</a></li>
          </ul>
        </div>
      </div>
    </div>
  </div>
  <div id="main" class="container-fluid">
    <div id="application">
  </div>
  <hr>
  <script id="CatalogLayout" type="text/template">
</script>
<script type="text/javascript" src="../../Libraries/jquery.js"></script>
<script type="text/javascript" src="../../Libraries/underscore.js"></script>
<script type="text/javascript" src="../../Libraries/backbone.js"></script>
<script type="text/javascript" src="../../Libraries/backbone.marionette.js"></script>
<script type="text/javascript" src="../../Libraries/bootstrap/js/bootstrap.min.js"></script>
<script src="js/BookStore.js"></script>
</body>
</html>
```




Bootstrap and styling your page is outside the scope of this book. But it's a pretty convenient library that allows us to set up a decent looking HTML file for this demo application.

But wait a minute...what does the last script `js/BookStore.js` refer to? Well, that's the JavaScript code that we will be creating in the next step.

The Backbone.Marionette.Application object

Create a new file inside the `js` folder and name it `BookStore.js`. To create a new application, we just need to type the following line in `Bookstore.js`:

```
var bookStoreApp = new Backbone.Marionette.Application();
```

We will name the application `BookStoreApp` and will start attaching our Backbone pieces to this application. But, we already mentioned that Marionette brings the concept of an application object and, from the documentation, we also know that it is an object that will help us to coordinate the pieces of our application. You may ask, what pieces; for example, a `Marionette.Router` object and a `Marionette.Controller` object.

Backbone already has a router!

Yes, Backbone already has a router object. Then what does the `Marionette.Router` object do differently? Well, the new router adds the ability of reducing your router to just a small file that will contain only the routes of your application and not the methods that will respond and take action once a route is matched. These methods belong to a controller – another new concept that Marionette adds to Backbone.

Let's build a `Marionette.Router` object and a `Marionette.Controller` object to get a better understanding of them:

```
var BookStoreController = Backbone.Marionette.Controller.extend({
  displayBooks : function () {
    console.log("I will display books...");
  }
});
var BookStoreRouter = Backbone.Marionette.AppRouter.extend({
  controller : BookStoreController,
  appRoutes: {
    "": "displayBooks"
  }
});
```

In the preceding code snippet, we created the `BookStoreController` object, which is just a JavaScript object containing the functions that will match the name of the methods defined in the router. In this case, the empty router will call the `displayBooks` method on the controller. This separation of concerns will allow us to have a cleaner code base as the router will only know about the routes. We declare which controller will handle the routes by setting the `controller` property of the router to `BookStoreController`. The rest of the code snippet is just the declaration of the routes.

It is not mandatory to have a router in order to use a controller. The Marionette controllers can be instantiated without the need of a router. You may not handle the interaction of your site by changes in the URL but by events. In this case, the controller still adds value as it can be the container of your views.

It's recommended to have small routers and controllers divided as per the purpose of your application instead of a giant single-router file that will contain all the routes and the functions.

While these two pieces are part of the application's foundation, we still need to make them work within it. But, we also need to do a little more in order to achieve a functional application. Let's take small steps for this. Let's first check out whether we can see a message log in the console of our browser.

To do that, we need to put all the code together and add the missing pieces in order to make it work.

So far, we have only defined the application, controller, and router. But where should we instantiate them? The `Backbone.Marionette.Application` object offers the possibility to add initializer methods that will run when we start our application.

Yes, you read correctly! You can add as many methods as you need in case you want to keep the logic of this initializers separated.

Inside this initializer method, we will instantiate the router and the controller, and just for fun, add another log message to see the order of execution.

Use the following code to do this:

```
BookStoreApp.addInitializer(function () {
  var bookStoreController = new BookStoreController({
    var bookStoreRouter = new
      BookStoreRouter({controller:controller});
    console.log('Message from the addInitializer Method');
  ..});
})
```

Another useful function of the applications is the events that fire the `initialize:before`, `initialize:after`, and `start` functions. The names of these functions are quite descriptive. As the name suggests, the `initialize:before` function will be executed before the initializers, the `initialize:after` function will be executed after the initializers, and the `start` function is responsible for starting the application and thereafter starting the initializers.

In our application, we will use `initialize:after`. This function will be helpful for us, as the last thing we want to do once we instantiate the router is start `Backbone.history`.

```
BookStoreApp.on('initialize:after', function () {
  if (Backbone.history) {
    Backbone.history.start();
  }
  console.log('Mesagge from initialize:after method');
});
```

The last step to complete the infrastructure or foundation of our application is call the following function:

```
BookStoreApp.start();
```

Now, let's put all the code snippets together as follows:

```
var BookStoreApp = new Backbone.Marionette.Application();
var BookStoreController = Backbone.Marionette.Controller.extend({
  displayBooks : function () {
    console.log("I will display books...");
  }
});
var BookStoreRouter = Backbone.Marionette.AppRouter.extend({
  controller : BookStoreController,
  appRoutes: {
    "": "displayBooks"
  }
});
BookStoreApp.addInitializer(function () {
  var controller = new BookStoreController();
  var router = new BookStoreRouter({controller:controller});
  console.log("hello from the addInitializer.");
});
BookStoreApp.on('initialize:after', function () {
  if (Backbone.history) {
```

```
Backbone.history.start();}
console.log("hello from the initialize:after.");
});
BookStoreApp.start();
```



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.

Now, you can go ahead and open the `Index.html` file in your browser and see the results on the console.

Summary

In this chapter, we learned about the application, controller, and router functionality, and how to get them working together to get a simple application skeleton which will be the base for our book store application.

In the next chapter, we will familiarize ourselves with the different views that Marionette adds to the Backbone development.

3

Marionette View Types and Their Use

In the previous chapter, we learned about components that help us provide a structure to our application; however, none of these components interacted with the DOM. This responsibility belongs to the views in Backbone development; however, the interaction and manipulation of the DOM can quickly become complicated inside our views. With the intention of having cleaner and meaningful objects to manipulate, the DOM Marionette introduces a powerful set of views. The following is a description of each one of those views provided in the official documentation at <https://github.com/marionettejs/backbone.marionette>:

- `Marionette.ItemView`: This is the view that renders a single model
- `Marionette.CollectionView`: This is the view that iterates over a collection and renders the individual `ItemView` instances for each model
- `Marionette.CompositeView`: This is the collection view and item view for rendering leaf-branch/composite model hierarchies
- `Marionette.Layout`: This is the view that renders a layout and creates region managers to manage areas within it
- `Marionette.View`: This is the base view type that other Marionette views extend from (not intended to be used directly)

In this chapter, we will learn the intention behind each one of them and how to start using them.