Ambitious Ember Applications

A comprehensive Ember.js tutorial

Ruslan Yakhyaev

Ambitious Ember Applications

A comprehensive Ember.js tutorial

Ruslan Yakhyaev

This book is for sale at http://leanpub.com/emberjs_applications

This version was published on 2014-10-26

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Ruslan Yakhyaev by spreading the word about this book on Twitter!

The suggested hashtag for this book is #emberjs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#emberjs

Contents

Introduction

It is fascinating to see how quickly the web is evolving these days. It took us only few years to make a full transition from boring, ugly websites to the beautifully crafted applications written directly within a browser. It all has been possible thanks to complex client side frameworks such as Ember.js, Backbone or Angular.

Ember.js is a framework for creating ambitious web applications. It main focus is to give you tools, powerful enough to reduce the amount of code you write. Ember incorporates many common idioms and frees you from reinventing the wheel. Similar to other opinionated frameworks, Ember values convention over configuration.

The API of the Ember is clean and developer friendly. Coming from the world of Rails (and having both a great distaste for JavaScript) I was surprised with how close to Rails writing the code in Ember.js felt.

So if you are deeply frustrated with your web application slowly becoming a huge unorganised ball of jQuery spaghetti callbacks, your JavaScript files are growing over 100kb or thinking about code behind your front-end gives you a knee-jerk reaction; maybe it is a sign to think about switching to client side JavaScript framework.

Before we start, you need to know that Ember.js is hard. It's concepts may be very confusing at first and if you have a background in any server side MVC framework (or any other client side JavaScript framework) things won't make much sense at first. That's why with every new concept introduced during the course of the book I will try to dive in and explain it in as much detail as possible.

This book is a crash course on Ember.js. After finishing it you should have enough knowledge to decide if Ember.js is the right choice for you and if you do — you should be proficient enough to start building your own Ember.js applications.

Needed Knowledge

Unfortunately this book doesn't explain the basic concepts of HTML, CSS, JavsScript or jQuery. These are the four technologies you need to know at least on beginners level to get through this book and to understand it. Ember.js is a framework built in JavaScript but fortunately for you, you won't need to have deep knowledge of language. Some basic knowledge of jQuery is required since we are going use some basic things such as selectors or jQuery methods. And of course we are building a web application — that's why we will need HTML and CSS.

So What Are We Building?

As you've may guessed we are going to build simple web application using which I will try to demonstrate as much Ember's concepts as possible. We will break a convention here and instead of building a web shop (since everybody is building web shops these days), we are going to build a simple Q&A application called Emberoverflow. Here is a short specification (or call it list of functionality if you like) for our application:

- visitors of our application can log in,
- logged in users can ask questions and answers them,
- questions can have multiple answers,
- users can edit questions they've asked.

Pretty simple yet it will be enough. Also, we won't be building a backend, so our application will not depend on any other technologies. Already excited? Good, lets start without any further delay.

You will find the source code of finished application on Github[1].

Source code for same application written in Ember CLI can be found on Github[2] as well.

[1] https://github.com/ryakh/emberoverflow
[2] https://github.com/ryakh/emberoverflow-cli

How to read this book

If you've read any other technical book you should have no problem reading this one. But just to be sure let me summarise the conventions used in this book.

Each chapter has list of topics covered in the chapter right at the beginning — this should give you general overview of chapter. Next there will be a link which will lead to the snapshot of application with all changes introduced to the code during the course of the chapter. Finally each chapter consists of two parts — first one is aimed to teach you Ember basics. Second one will show you how to achieve what you just did using Ember CLI.

Code samples are displayed in monospaced font, with the title of the file from which the sample was taken displayed above the sample. Also code samples have line numbers adjusted to mimic the line numbers in real files. Before each code sample there is a link leading to the commit with the code hosted on Github.

Code sample[3]:

code_snippet.js

```
17  App = Ember.Application.create({
18    awesomeApplication: true
19  });
```

Tips are highlighted with the icon of the key. Tip is something you don't need in the context of current chapter but it will give you some extra knowledge.

Information blocks extend or explain things you've gained from the current chapter.

[3] https://github.com/ryakh/emberoverflow/commit/a6a7b98e5898f65f13429f5a53c2d986fcf21a8e

1. Setting Things Up

Topics covered in this chapter:

1. preparing scaffolds for development,
2. anatomy of the Ember Starter Kit,
3. quick introduction to anatomy of Ember.js,
4. setting up Ember Inspector.

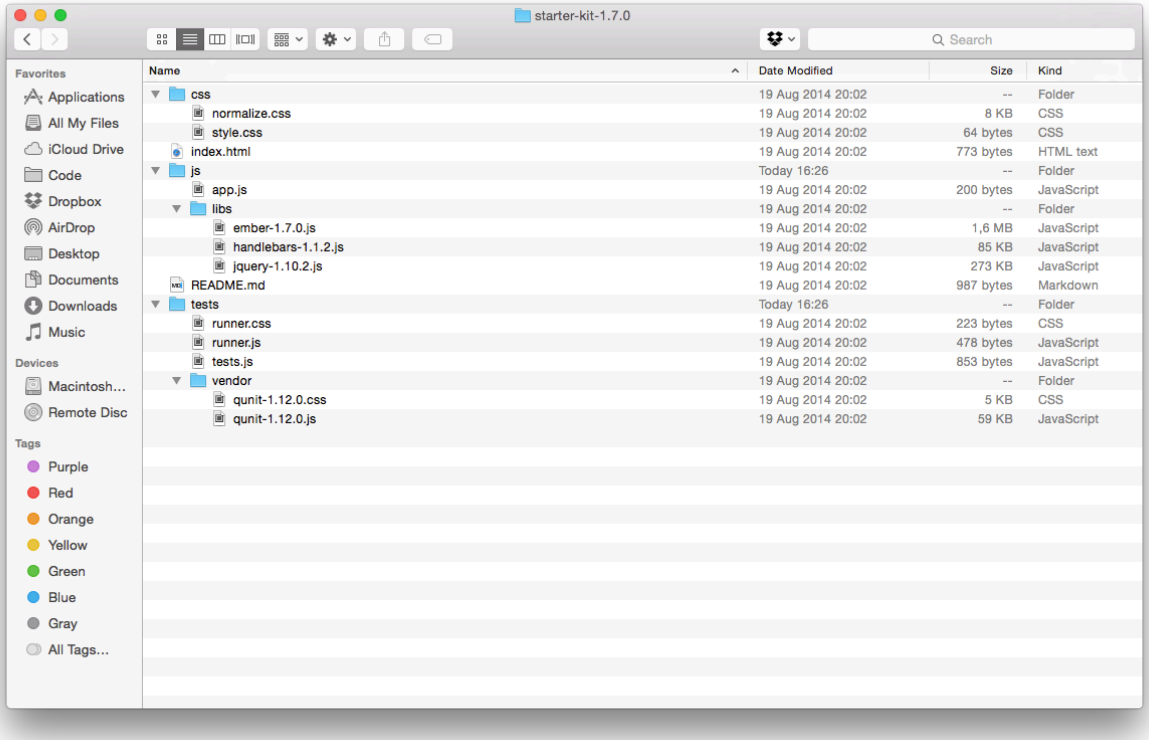
Ember CLI Way:

1. installing Ember CLI,
2. generating new application,
3. Ember CLI application structure,
4. installing external dependencies

Snapshot of the application[1].

Ember provides us with Starter Kit. It is a simple scaffold to jump-start developing new Ember application. Ember is not backend dependent, so you will only need a text editor and a browser to start developing Ember application. Download Ember Starter Kit[2] and extract it. It has really simple folder structure:

[1] https://github.com/ryakh/emberoverflow/tree/ddedaf2325324f6d0f724285bd213e814bf74b65
[2] http://emberjs.com/



Ember Starter Kit folder structure

Lets start by dropping in Twitter Bootstrap[3] framework[4]. Open your editor of choice and add following line into the head section of `index.html`.

Code sample[5]:

index.html

```
3  <head>
4    <meta charset="utf-8">
5    <title>Ember Starter Kit</title>
6    <link rel="stylesheet" href="css/normalize.css">
7    <link rel="stylesheet" href="css/style.css">
8    <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/cs\
9  s/bootstrap.min.css">
10 </head>
```

[3] http://getbootstrap.com/

[4] Twitter Bootstrap is a front-end CSS, HTML and JavaScript framework which includes pre-defined web components (such as grid or modals) to make prototyping of a web application as quick and painless as possible

[5] https://github.com/ryakh/emberoverflow/commit/1d5099ab4dae8c7572226a0b243df0e12228bc59

Then add the following at the end of the same file.

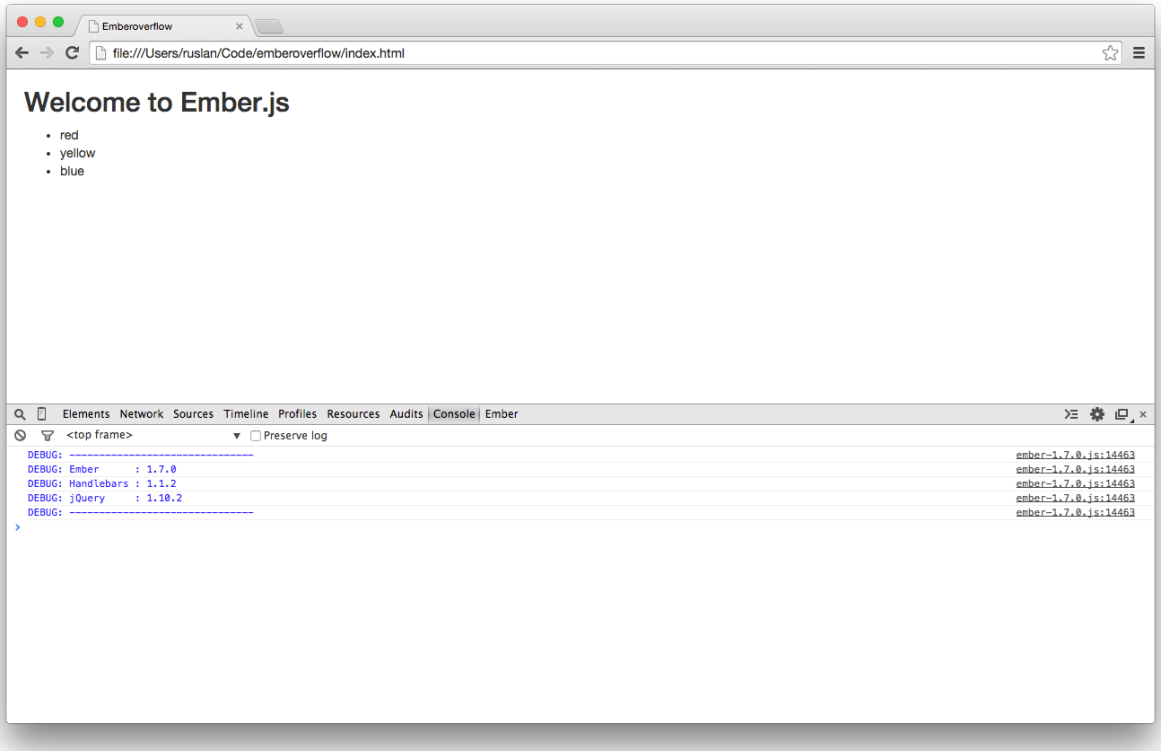
Code sample[6]:

index.html

```
25    <script src="js/libs/jquery-1.10.2.js"></script>
26    <script src="js/libs/handlebars-1.1.2.js"></script>
27    <script src="js/libs/ember-1.7.0.js"></script>
28    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.j\
29  s"></script>
30
31    <script src="js/app.js"></script>
32    <!-- to activate the test runner, add the "?test" query string parameter -->
33    <script src="tests/runner.js"></script>
34  </body>
```

[6] https://github.com/ryakh/emberoverflow/commit/3cff14fcbbf42eee58f5d5ed0e885c17269e057e

Launch your browser and load `index.html` included in Ember Starter Kit. Open console of your browser, and if everything is going as expected, this is what you will see:



Empty Ember Starter Kit application

Debug message printed out in opened console states that our Ember application is loaded and ready to go.

1.1 Turning Starter Kit Inside Out

Look into the source of `index.html` which is a part of the Starter Kit — there are bunch of script tags in there and nothing else. Where does content we see in our browser comes from? If you are familiar with Ember or any other client side framework, you probably know the answer. But just in case you are not lets go through this file, line by line. Head contains pretty much common stuff — some meta data and stylesheets. Then we have `body` tag which contains two `script` tags of type `text/x-handlebars` and finally we load external javascript files. Lets start with the latter:

index.html

```
25    <script src="js/libs/jquery-1.10.2.js"></script>
26    <script src="js/libs/handlebars-1.1.2.js"></script>
27    <script src="js/libs/ember-1.7.0.js"></script>
28    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.j\
29  s"></script>
30
31    <script src="js/app.js"></script>
32    <!-- to activate the test runner, add the "?test" query string parameter -->
33    <script src="tests/runner.js"></script>
34  </body>
```

Here we load all requirements for Ember. They are: jQuery and Handlebars (JavaScript tempting library). Then we load Ember itself, `app.js` file which will contain our custom application code and `runner.js` file which is a gateway to our tests.

Lets add another Ember dependency, Ember Data. It is still not a part of Ember because it is being actively developed. Discourse[7] for example is not using Ember Data at the moment. You could chose not to use it while developing your own applications but during the course of this book we will rely on it.

[7] Discourse is an open-source discussion platform written in Ember. Currently it is one of the largest open-source Ember application available

Add following line among your script files.

Code sample[8]:

index.html

```
25    <script src="js/libs/jquery-1.10.2.js"></script>
26    <script src="js/libs/handlebars-1.1.2.js"></script>
27    <script src="js/libs/ember-1.7.0.js"></script>
28    <script src="http://builds.emberjs.com/beta/ember-data.js"></script>
29    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.j\
30  s"></script>
31
32    <script src="js/app.js"></script>
33    <!-- to activate the test runner, add the "?test" query string parameter -->
34    <script src="tests/runner.js"></script>
35  </body>
```

We have another two script tags — handlebar templates. They contain layout of our application:

index.html

```
11  <script type="text/x-handlebars">
12    <h2>Welcome to Ember.js</h2>
13
14    {{outlet}}
15  </script>
16
17  <script type="text/x-handlebars" id="index">
18    <ul>
19    {{#each item in model}}
20      <li>{{item}}</li>
21    {{/each}}
22    </ul>
23  </script>
```

[8] https://github.com/ryakh/emberoverflow/commit/ddedaf2325324f6d0f724285bd213e814bf74b65

1.2 Where the Magic Happens

As you've probably guessed, all the magic here happens thanks to two `script` tags of `text/x-handlebars` type and the `app.js` file. Lets open `app.js` file. You will see the following code:

js/app.js

```
App = Ember.Application.create();

App.Router.map(function() {
  // put your routes here
});

App.IndexRoute = Ember.Route.extend({
  model: function() {
    return ['red', 'yellow', 'blue'];
  }
});
```

Again, if you are familiar with the framework, bear with me as I will try to explain this file line by line. First we are creating our Ember application. It is responsible for creating a new instance of Ember.Application and making it available within our web page; which thanks to that single line becomes web application!

Second block is responsible for creating router inside our application. You are probably familiar with the basic concepts of routers in other frameworks — Ember is no exception and router in Ember is responsible for taking URLs you provide within your browser and sending them into deeper layers of our application.

In Ember, similar to server side based web frameworks, different states of our application are represented by URLs. We interact with our application and save states of our application in the URLs. So we can simply remember the state by saving URL or we can even share the state with other users by sharing URL with them.

Last part is a Route block which may be a little bit confusing at this point, but I promise that once you finish reading this book, you will understand the difference between Ember objects (I am not calling them components because one of the objects is called a component) and philosophy behind each one of them. Here is a list of all key Ember objects with a brief description.

Router
: Is the connecting point between browser's address bar and our application. It translates URL address into Route object.

Route
: Is where user request will land after it was translated by Router. Route decides what data should be provided to the Template, which Template and Controller should be used.

Model
: Defines structure and logic behind data that is presented to our user.

Store
: Is a place where records are saved (cached) once they are retrieved from the server. Store will also keep new records before they are synchronised to the server.

Controller
: Templates query controllers for values that have to be displayed to the user. Controllers also decorate (transform, alter, change) data from models before it is displayed to the user. Controllers have a view and a template.

View
: View is responsible for setting template and encapsulating all the HTML. It can also respond to various types of user generated events.

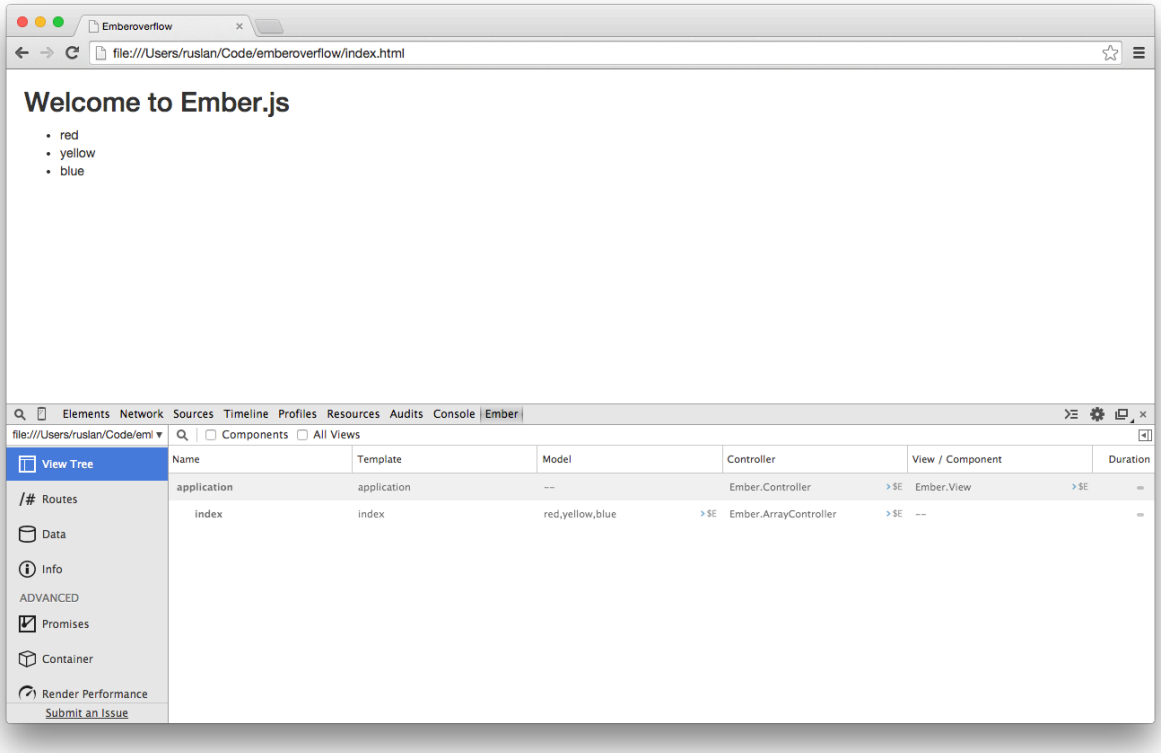
Component
: Component is very similar to the View. It is used to create reusable parts for our application.

Template
: Templates are parts of Ember application that other people see. Components, Views and Controllers each have a template.

Feeling already lost? Good. Go ahead and re-read the last part. Don't try to understand it or memorise it, it will make you even more confused at this point. Remember I've said that Ember is hard? That was the living proof of it. But don't worry, you will have solid knowledge of Ember soon enough.

1.3 Installing Ember Inspector

I encourage you to use Google Chrome web browser for the duration of the book because it has Ember Inspector extension available. Install it from Chrome Marketplace, open Chrome and Visit the `chrome://extensions` URL. Find Ember Inspector among installed extensions and make sure **Allow access to file URLs** is checked. Now open Developer Tools and if you did everything correct you should see the Ember tab as the last tab in the row of tools available:



Ember Inspector

We will dig deeper into the inspector later, now just make sure that it works.

1.4 Ember CLI Way

Snapshot of the application[9].

Ember CLI is Ember command line utility which provides an asset pipeline and strong conventional project structure. It relies on broccoli[10] for asset compilation (such as JavaScript and CSS), bower[11]

[9] https://github.com/ryakh/emberoverflow-cli/tree/109e5f5dfb0c4e49270dd2094f5a515ffa79cbe2

[10] https://github.com/broccolijs/broccoli

[11] http://bower.io

for dependency management and NPM[12] for managing internal dependencies.

Ember CLI speeds up development process of Ember applications and provides developers with vast amount of tools such as scaffold generators and automatic build tools.

Ember CLI is still in heavy development and braking changes are introduced on weekly basis. It development was started by Stefan Penner[13]. Ember CLI has it's own website which provides detailed documentation — http://www.ember-cli.com[14]

1.4.1 Installing Ember CLI

One dependency you will need to install manually is Node.js[15]. You can either download it from the download section[16] of Node.js website or you can install it using package manager of your choice (such as Homebrew on Mac OS or aptitude on Ubuntu or Debian).

Once you have installed Node you will need to install Ember CLI globally by running the following command in your console:

```
1 npm install -g ember-cli
```

Installing Ember CLI globally will give you access to `ember` command inside your command line. Another dependency you'll need to install is Bower (to keep your front-end dependencies such as jQuery and Ember itself) up to date. Installing Bower is similar to installing Ember CLI and can be done by running following command in your command line:

```
1 npm install -g bower
```

1.4.2 Generating new project

Creating new project with Ember CLI is as easy as typing a single command:

```
1 ember new emberoverflow-cli
```

Running `ember new <app-name>` will initialise git repository and commit all files generated by Ember CLI with a cool commit message.

[12] https://www.npmjs.org
[13] https://github.com/stefanpenner
[14] http://www.ember-cli.com
[15] http://nodejs.org
[16] http://nodejs.org/download/

Running this command will create new folder called `emberoverflow-cli` and generate application structure.

Ember CLI comes with a web server; once the process of creation of new app is finished you can launch it by navigating into folder called `emberoverflow-cli` and using `server` command:

```
cd emberoverflow-cli
ember server
```

Now if you launch your browser and go to `http://localhost:4200` you will see our newly created app. Navigating to `http://localhost:4200/tests` will launch test suite.

If you look inside directory which Ember CLI generated for us you will notice that it has different structure than Ember Starter Kit. Let's quickly go through the scaffold Ember CLI generated for us.

Folder/File	Description
`app/`	This is where you application code will live.
`dist/`	This folder will contain your built application ready to be deployed to the server.
`public/`	Use this folder to store assets such as images and fonts.
`tests/`	Where all tests and test related code lives.
`tmp/`	Directory that will hold system level temporary files.
`bower_components/`	Dependencies installed with Bower and included in Ember CLI
`vendor/`	Same as `public/` but for assets such as JavaScript libraries
`Brocfile.js`	Build specifications for Broccoli
`bower.json`	Dependency list managed by Bower
`package.json`	NPM configuration. Mainly holds dependencies needed for asset compilation

Next lets look inside `app/` folder to see whats in there:

Folder/File	Description	
`app.js`	Application entry point; module that will be executed first.	
`index.html | Similar to` index.html‘ in Ember Starter	Kit — includes dependencies and starts your Ember application.	
`router.js`	Router configuration (similar to one in file.	Ember Starter Kit, yet put into separate
`styles/`	Contains your stylesheets.	
`templates/`	Your application templates.	
`controllers/` `models/` `controllers/` `helpers/` `routes/` `views/`	Your application code.	

Ember CLI uses ES6 modules. You don’t need to have any specific knowledge of them but if you are interested in learning more about ES6 modules you can read more about them from http://jsmodules.io[17] website.

While setting our project in the first part of the chapter we’ve done two things: added Twitter Bootstrap and Ember Data. We will use different approach though: we will tell our application what we need instead of downloading and saving files manually.

[17] http://jsmodules.io

We can do so by extending our `bower.json` file (`ember-data` should be added automatically, just make sure it is there and correct version is used).

Code sample[18]:

bower.json

```
{
  "name": "emberoverflow-cli",
  "dependencies": {
    "handlebars": "~1.3.0",
    "jquery": "^1.11.1",
    "ember": "1.7.0",
    "ember-data": "1.0.0-beta.11",
    "ember-resolver": "~0.1.7",
    "loader.js": "stefanpenner/loader.js#1.0.1",
    "ember-cli-shims": "stefanpenner/ember-cli-shims#0.0.3",
    "ember-cli-test-loader": "rwjblue/ember-cli-test-loader#0.0.4",
    "ember-load-initializers": "stefanpenner/ember-load-initializers#0.0.2",
    "ember-qunit": "0.1.8",
    "ember-qunit-notifications": "0.0.4",
    "qunit": "~1.15.0",
    "bootstrap": "3.0.3"
  }
}
```

`bower.json` file keeps track of external dependencies used by your application and their versions.

Now you have to install packages listed in `bower.json` using following command:

```
bower install
```

After installation is finished you have to tell your application that it has to load newly installed package and use it. You can do so by upgrading `Brocfile.js`:

[18] https://github.com/ryakh/emberoverflow-cli/commit/0c0f236c8f1ba10c7db439051c939c490c434fc0

Code sample[19]:

Brocfile.js

```
/* global require, module */

var EmberApp = require('ember-cli/lib/broccoli/ember-app');

var app = new EmberApp();

app.import('bower_components/bootstrap/dist/css/bootstrap.css');
app.import('bower_components/bootstrap/dist/js/bootstrap.js');

module.exports = app.toTree();
```

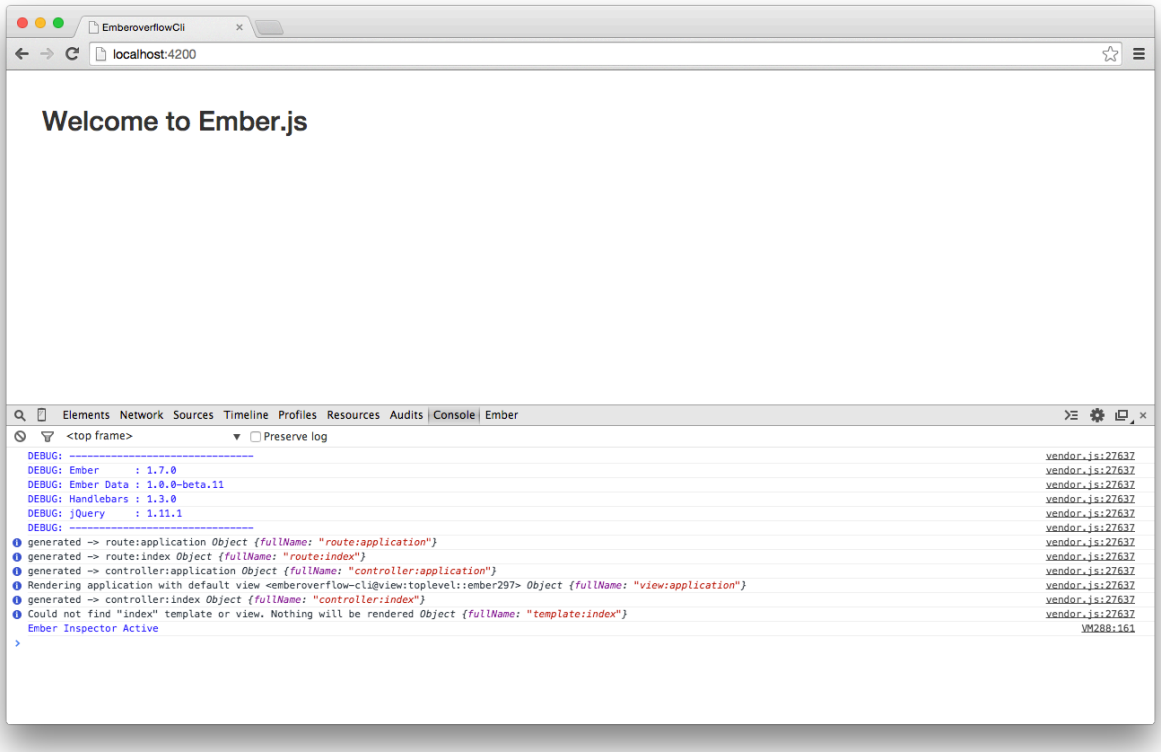
If Ember CLI server was running you will have to restart it after changing `Brocfile.js`.

You won't have to restart your server if you make changes to files inside your `app/` directory.

Ember CLI will automatically compile your application files and concatenate them into `app.js` and `app.css`. By specifying external dependencies in `Brocfile.js` you tell Ember CLI to compile them into concatenated application files.

[19] https://github.com/ryakh/emberoverflow-cli/commit/109e5f5dfb0c4e49270dd2094f5a515ffa79cbe2

If you reload application in your browser everything should be up and running:



Ember CLI

2. First Templates, First Routes

Topics covered in this chapter:

1. initiating application,
2. theory behind templates and Handlebars,
3. creating new routes,
4. providing templates for defined routes,
5. using Ember inspector to get information about routes.

Ember CLI Way:

1. defining new routes,
2. using ember generator to create template,
3. working with templates.

Snapshot of the application[1].

Lets start where we left off. Load `index.html` in your browser and view source code of the page. You are looking for `body` tag:

index.html source code viewed in browser

```
<body style="" class="ember-application" data-ember-extension="1">
</body>
```

If you open same `index.html` file in editor of your choice you will notice that `body` tag has no additional attributes. These are added by Ember to mark the root element of our application (element, content inside which is controlled by Ember).

Remember our `app.js` file? There we had following line of code:

js/app.js

```
1  App = Ember.Application.create();
```

With single line we've created our application and put it into `App` namespace. Later on, if we'd have to add more functionality, we will simply define it as properties of our `App` object. Note that you need to create application only once. Also, you can name your application whatever you like; it doesn't have to be `App` or anything in particular. Just make sure it starts with the capital letter.

[1] https://github.com/ryakh/emberoverflow/tree/fc0ee792aa850cbb318d01e33e18ead764a78024

You can pass different options into `create()` method of `Ember.Application` object represented by a plain JavaScript object. Here are some of the options you can provide:

```
Ember.Application.create({
  // sets application root element
  rootElement: '#element-id',

  // logs out a message to the console once the URL changes
  // which is useful for debugging
  LOG_TRANSITIONS: true
});
```

But what else will that single line of code do under the hood? It will add event listeners to our document which will be handled by Ember. Then it will render an application template. And finally it will set up router to listen for URL changes and respond in correct way.

2.1 Introduction to Templates

Creating application will automatically render application template for us. In our application we currently have 2 templates defined:

index.html

```
<script type="text/x-handlebars">
  <h2>Welcome to Ember.js</h2>

  {{outlet}}
</script>

<script type="text/x-handlebars" id="index">
  <ul>
  {{#each item in model}}
    <li>{{item}}</li>
  {{/each}}
  </ul>
</script>
```

Template without ID is an application template. It will be rendered on each page by default. All other templates will be rendered inside application template. Ember expects every template name to be unique. Note the `{{outlet}}`[2] expression. Ember will always render other templates into `{{outlet}}` expression.

So whenever you request any route by changing URL in your browser Ember will render template that belongs to the route requested. Second template has ID equal to `index`. That means that it will be rendered under `index` route which is always defined by default.

Last thing you need to know at this point about templates is that Ember uses Handlebars templating library. Handlebars templates are just like regular HTML, but they also give us the ability to embed expressions into our templates and dynamically change what is displayed inside our templates.

2.2 Adding our First Route

As you've learned in the previous part Ember generates index route for us. But how can we define our own route?

We need to tell our users more information about our web application. Lets add route which will be used to access that part of application. Inside `app.js` file change router to the following.

Code sample[3]:

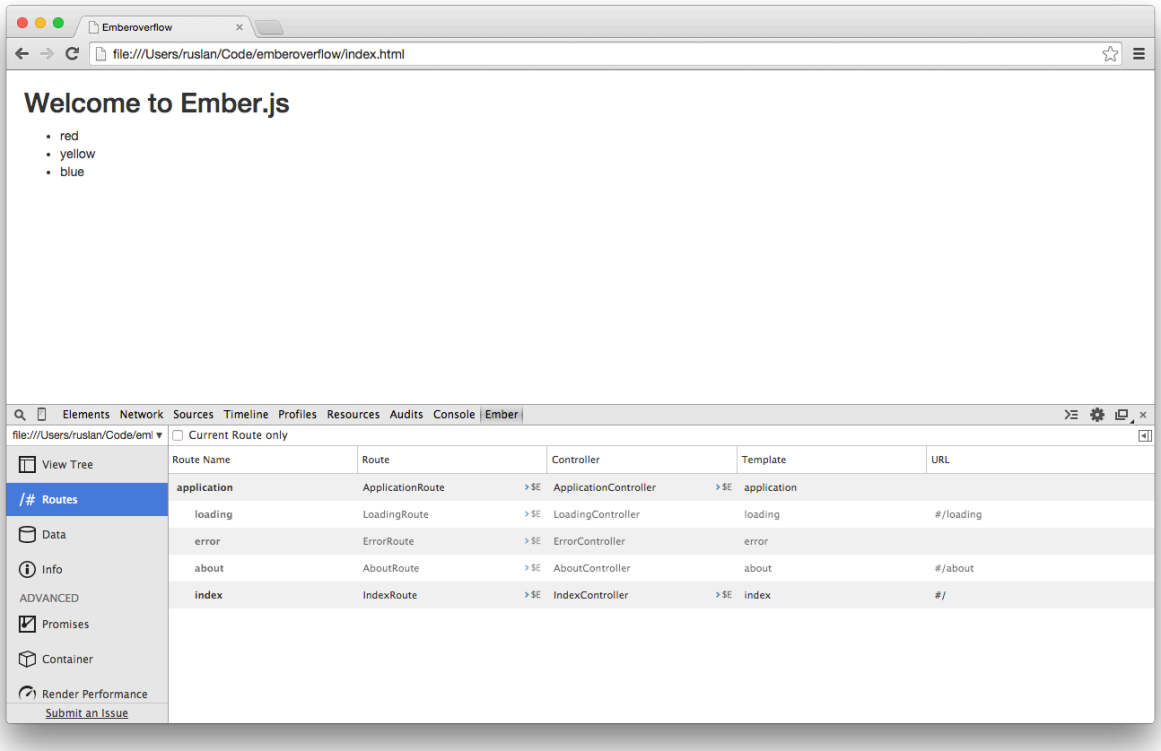
js/app.js

```
3  App.Router.map(function() {
4    this.route('about');
5  });
```

Lets stop here for a moment and open Ember Inspector. Go into Routes section and there you will see our newly created route:

[2] Text inside two curly braces is called **Handlebars expression**. Under the hood it will be picked up by Handlebars library and parsed into regular JavaScript.

[3] https://github.com/ryakh/emberoverflow/commit/3da3a2a2ed2ee9f9550a29ae28d94f8f7faa0a1d



Routes Inside our Application

Take a look at routes that are available to us, there are currently 5 of them:

1. application,
2. loading,
3. error,
4. about,
5. index.

Four routes were generated by Ember. `about` route is the one we've added manually. Look at Route Name column and remember the value which is written within it for the `about` route — we will use it to access the route in templates.

To navigate through our application we will use links. To create a link inside our template we will use `{{#link-to}}{{/link-to}}` handlebars block expression.

There are two types of expressions in Handlebars library that are important for us in the context of this book. You have already seen an expression (a text between two curly braces; i.e. `{{outlet}}`). Expression will be evaluated by Ember and rendered into the template. Block expressions look something like this — `{{#block}}Content{{/block}}`. Common use case for block expression is to decorate content within it, iterate over a list of items or evaluate true-false expression.

Lets update our application template with navigation bar; using which our users will be able to navigate through the application.

Code sample[4]:

index.html

```
11 <script type="text/x-handlebars">
12   <nav class="navbar navbar-default" role="navigation">
13     <div class="navbar-header">
14       {{#link-to "index" classNames="navbar-brand"}}
15         Emberoverflow
16       {{/link-to}}
17     </div>
18
19     <ul class="nav navbar-nav">
20       <li>
21         {{#link-to "about"}}
22           About site
23         {{/link-to}}
24       </li>
25     </ul>
26   </nav>
27
28   {{outlet}}
29 </script>
```

`{{link-to}}` block expression accepts different arguments. First one is the route into which link will resolve. Remember routes we viewed using Ember Inspector? Here we are passing in name of the route (the one I've told you to remember). Handlebars will take `link-to` block, parse it into regular `<a></a>` tag, apply all arguments you provide it with and insert it into our template.

There are many other different arguments available. For example: `activeClass`, `disabled`, `tagName` and others. Last one will actually make Handlebars transform `{{link-to}}` block expression into the provided tag; so if you pass in `tagName='li'`, instead of creating `<a></a>` tag Handlebars will create `<li></li>` tag and place content of your `{{link-to}}` into the tag.

[4] https://github.com/ryakh/emberoverflow/commit/03b3cc45c1766292db99da9e890cb047e691bb3d

Now before we proceed, there is one more thing I want to show you. Add following lines to our `style.css` file:

Code sample[5]:

css/style.css

```
6  .active {
7    font-weight: bold;
8  }
```

Go ahead and try to click on links we have inside our navigation bar. You should be able to seee that the currently active link gets class `active` (and is displayed in bold). Ember gives you that feature for free — currently active links will always get active class.

2.3 Providing Content for our Route

We can navigate through our application by clicking on links in navigation bar but we have no content for `about` page. Lets fix this. Change `index` template and add another one called `about`:

Code sample[6]:

index.html

```
31 <script type="text/x-handlebars" id="index">
32   <div class="row">
33     <div class="col-md-8">
34       <h2>Welcome to Emberoverflow</h2>
35
36       <ul>
37       {{#each item in model}}
38         <li>{{item}}</li>
39       {{/each}}
40       </ul>
41     </div>
42   </div>
43 </script>
44
45 <script type="text/x-handlebars" id="about">
46   <div class="row">
47     <div class="col-md-8">
```

[5] https://github.com/ryakh/emberoverflow/commit/8a43aa1e6e64c22d2b0977f2369a81315113dc7f

[6] https://github.com/ryakh/emberoverflow/commit/fc0ee792aa850cbb318d01e33e18ead764a78024

```
48        <h2>About Emberoverflow</h2>
49
50        <p>
51          Emberoverflow is a question and answer site for programmers and
52          hamsters. We are a little bit different because we are written in
53          Ember.js
54        </p>
55      </div>
56    </div>
57  </script>
```

Go ahead and reload our application. Now if you navigate through the application by clicking on links in navigation bar, content of the page will change. What happens here illustrates one of the key ideas behind Ember — convention over configuration. We will be tripping over this aspect of Ember over and over again. Now just keep in mind that for Ember it is enough to provide a route (URL) and a template. Ember will match the URL to the template's name and render the needed content.

Open up Ember Inspector once again, go into the Routes section and search for the column named Template; there you will see name of the template Ember expects you to provide for the given route, which in our case is `about`.

2.4 Ember CLI Way

Snapshot of the application[7]

Unlike Ember Starter Kit, Ember CLI does a lot of things under the hood hiding all the heavy lifting from plain sight.

Entry point for you applications built with Ember CLI is a file called `app/app.js`. This module[8] will be executed first.

Let's take a closer look at it:

[7] https://github.com/ryakh/emberoverflow-cli/tree/58f555da94cbd3004aec71d8749ecb2b23a6cf9f

[8] Since Ember CLI works with modules (by exporting and importing them into different parts of our application), in the context of Ember CLI Way I will refer to different parts of our application as modules.

app/app.js

```
import Ember from 'ember';
import Resolver from 'ember/resolver';
import loadInitializers from 'ember/load-initializers';
import config from './config/environment';

Ember.MODEL_FACTORY_INJECTIONS = true;

var App = Ember.Application.extend({
  modulePrefix: config.modulePrefix,
  podModulePrefix: config.podModulePrefix,
  Resolver: Resolver
});

loadInitializers(App, config.modulePrefix);

export default App;
```

First we are importing other modules (dependencies) that we will use inside our module (if you are familiar with other programming languages such as Python you probably are familiar with this particular concept). Next we define our new module as JavaScript variable and finally we are exporting it; so it will be available to import by other modules. If you don't understand contents of this file right away — do not worry. You would not have to for the purpose of this training course. Once you get more experience things will become more clear.

In your `app/app.js` file take a look at the following line: `import config from './config/environment';`. As you've probably guessed this is where you application configuration is placed. Once you start developing complex apps you would probably be using it to configure your application. Right now it is enough to know that separate configuration file for your application exists.

Theory aside, lets dive head first into the code.

Here is a list of things we did in previous part (and which we will do in this part):

1. create new route,
2. add navigation bar with link to our new route,
3. add style for active link.
4. create template for newly created route.

2.5 Defining New Route

To create new route we need to go into `app/router.js` file. This is what you will see once you open it.

index.html

```
import Ember from 'ember';
import config from './config/environment';

var Router = Ember.Router.extend({
  location: config.locationType
});

Router.map(function() {
});

export default Router;
```

First we import `Ember` and `config`, then we define new `Router` by extending `Ember.Router`, then we define routes and finally we export our Router so it can be used in our application. Since we are still using Ember, adding new route will be the same as in previous part of this chapter:

Code sample[9]:

app/router.js

```
import Ember from 'ember';
import config from './config/environment';

var Router = Ember.Router.extend({
  location: config.locationType
});

Router.map(function() {
  this.route('about');
});

export default Router;
```

[9] https://github.com/ryakh/emberoverflow-cli/commit/bf27888a58eb7d130843b7d7ce4eac15385c1b55

2.6 Adding Navigation Bar

Instead of defining our templates inside `index.html` file using `<scipt></script>` tags we are going to place them into separate files with `.hbs` extension. Ember CLI will then automatically compile them and serve as a regular JavaScript files to your browser.

Ember CLI has already generated single Handlebar template for us which we are going to use. Open it and change it's content to the following:

Code sample[10]:

app/templates/application.hbs

```
<nav class="navbar navbar-default" role="navigation">
  <div class="navbar-header">
    {{#link-to "index" classNames="navbar-brand"}}
      Emberoverflow
    {{/link-to}}
  </div>

  <ul class="nav navbar-nav">
    <li>
      {{#link-to "about"}}
        About site
      {{/link-to}}
    </li>
  </ul>
</nav>

{{outlet}}
```

Template is almost identical to one we've created in previous part with small exception — it does not use `<script></script>` tags.

[10] https://github.com/ryakh/emberoverflow-cli/commit/519f22a8cb2cdcd5c626ef4dfb0c96387776f625

2.7 Adding Class for Active Link

The only thing in here that is different is a location of file. So go ahead and change code of `app.css` file to match following.

Code sample[11]:

app/styles/app.css

```
5  .active {
6    font-weight: bold;
7  }
```

After you introduce this changes, active link in your application should be automatically highlighted.

2.8 Creating New Template

Instead of creating new template manually (which will work though), we will elevate the power of Ember CLI and use it's generator to create new template for us by running following command in your console:

```
1  ember generate template about
```

Running `ember help generate` will list all available scaffold generators.

`ember generate` command takes in many additional options. First one (in our case third word in command we've typed) is the name of the generator that we want to use. Second one is the name of the file that has to be generated. So in our case we've told Ember CLI to generate `template` named `about`.

[11] https://github.com/ryakh/emberoverflow-cli/commit/d5480112aef79ddc0d73a72c462bf3b6357a7d0c

Last thing we have to do is to open newly created template and put some content inside.

Code sample[12]:

app/templates/about.hbs

```
<div class="row">
  <div class="col-md-8">
    <h2>About Emberoverflow</h2>

    <p>
      Emberoverflow is a question and answer site for programmers and
      hamsters. We are a little bit different because we are written in
      Ember.js
    </p>
  </div>
</div>
```

[12] https://github.com/ryakh/emberoverflow-cli/commit/58f555da94cbd3004aec71d8749ecb2b23a6cf9f

3. Controllers

Topics covered in this chapter:

1. creating our first controller,
2. working with properties,
3. controller types in Ember.

Ember CLI Way

1. using `ember generate`,
2. exporting modules,
3. working with controllers and templates.

Snapshot of the application[1].

If you have previous experience with another web framework such as Rails or Django you may think that you already know what controller in Ember is and what it will be responsible for. Let me tell you right away — you are wrong. Forget everything you know about controllers from server side web frameworks.

Although you may have heard other people calling Ember an MVC framework, that is not accurate. Yes, Ember has objects that are called models, views and controllers but thats where the similarity to MVC frameworks ends. My biggest concern while learning Ember was to wrap my head around naming conventions. Once I've let Rails MVC model go it was much easier for me to understand concepts of Ember. Ember is not MVC framework. Ember is a framework for building client side web applications.

Controller in Ember is responsible for decorating data before presenting it to user. Think of controller as of middleman sitting between your data source and your user, translating language of one into another.

[1] https://github.com/ryakh/emberoverflow/tree/5f09f21e0ea56b9a3b656b8940d9364e17dea7cf

3.1 Our First Controller

But how do we tell Ember which controller is responsible for which route? Ember is highly opinionated and values convention over configuration. We have two routes defined in our application — `index` and `about`. According to Ember conventions, Ember will expect our application to have two controllers defined: `IndexController` and `AboutController`. If Ember doesn't find these two controllers (or any other specific controller or even any other object such as route or view), it will generate one behind the scenes and keep it in memory for later use.

You are probably familiar with code generators. Ones that create basic scaffold and put that scaffold into your project folder. Ember code generator works in a different way. You don't have to tell it in advance to generate a piece of code. Instead it will generate code when it is needed (which in most cases means that you didn't provide the code); instead of creating file on your hard drive Ember will create object in memory and destroy it once it is not needed anymore. This technique is great because it frees you from writing boilerplate code and maintaining it later.

Lets create our first controller — `IndexController`.

Code sample[2]:

js/app.js

```
13 App.IndexController = Ember.Controller.extend({
14   siteTitle: 'Welcome to Emberoverflow'
15 });
```

`siteTitle` is called property. Properties will be available inside our templates. To access properties we use Handlebars expressions.

[2] https://github.com/ryakh/emberoverflow/commit/3cee9205fae079f0c96879029da9ea8a974aac11

Code sample[3]:

index.html

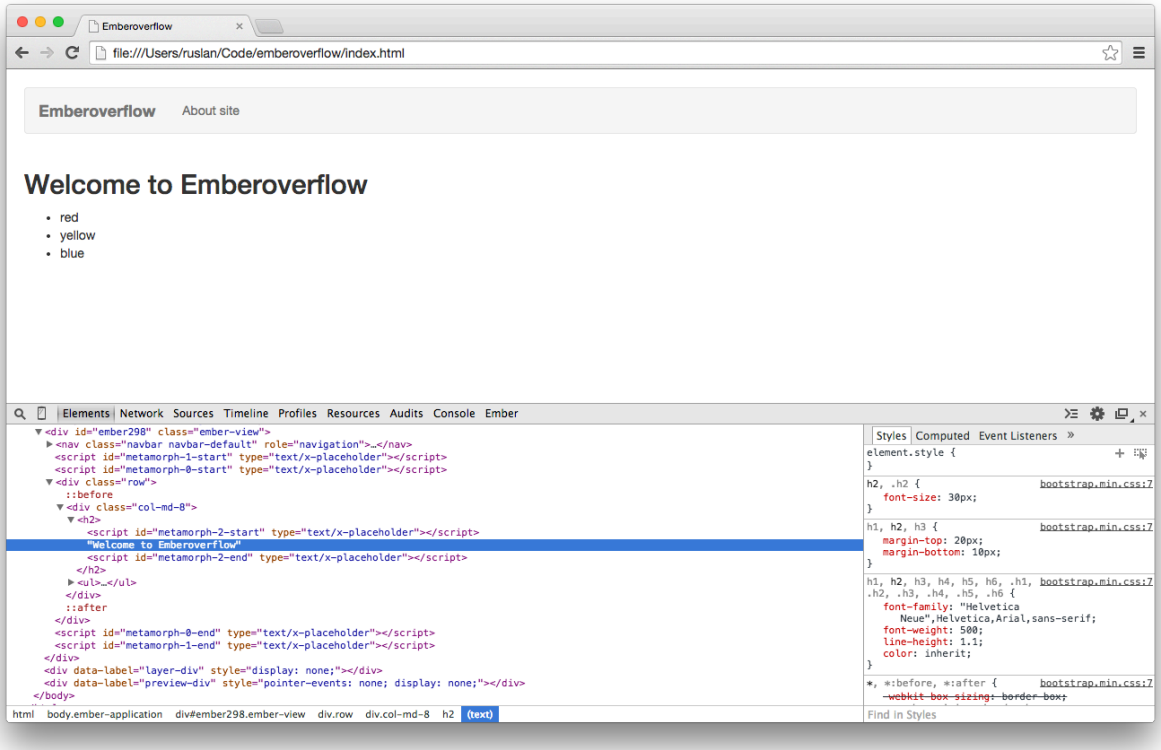
```
<script type="text/x-handlebars" id="index">
  <div class="row">
    <div class="col-md-8">
      <h2>{{siteTitle}}</h2>

      <ul>
      {{#each item in model}}
        <li>{{item}}</li>
      {{/each}}
      </ul>
    </div>
  </div>
</script>
```

To see what we did go ahead and reload our application. You will see no visual change but once you change value of `siteTitle` property inside our controller it will be changed inside our application as well.

[3] https://github.com/ryakh/emberoverflow/commit/9d4503c0cfd0107456e507bce4bfe7afb2bc69a9

View source code of our page and look for the header which contains `siteTitle` property. Here is what you will see:



What's with weird script tags?

Look at script tags that Ember generated around the header:

Script tags generated by Ember

```
<script id="metamorph-2-start" type="text/x-placeholder"></script>
"Welcome to Emberoverflow"
<script id="metamorph-2-end" type="text/x-placeholder"></script>
```

Ember uses metamorph script tags to wrap properties we set in our controller so it can easily identify them and update if needed.

But what about attributes that are parts of another tag? Ember will insert script tags around our attribute, so something like this won't work:

Inserting inline attribute in a wrong way

```
<img src="{{imageSource}}">
```

This will generate the following code:

Inserting inline attribute in a wrong way

```
<img src="&lt;script id='metamorph-7-start' type='text/x-placeholder'&gt;&lt;/sc\
ript&gt;image.jpg&lt;script id='metamorph-7-end' type='text/x-placeholder'&gt;&l\
t;/script&gt;">
```

Ember just wrapped the value of `imageSource` property into script tags. So how can we solve that? Here we will use bound attributes (they are called this way because value of the attribute is bound to the controller). To render image from `imageSource` attribute we will have to use the following syntax:

Inserting inline attribute using bound attribute

```
<img {{bind-attr src=imageSource}}>
```

Which will generate correct HTML:

Inserting inline attribute using bound attribute

```
<img src="image.jpg" data-bindattr-1="1">
```

In this case Ember will use `data-bindattr-1="1"` attribute to track down the property and update it if needed.

3.2 Computed Properties (Properties as Functions)

But what if we want to compute a property?[4] Let's say we want to display current time in our template. We can do this by using a function. You'd probably guess something like this will work.

Code sample[5]:

js/app.js

```
13 App.IndexController = Ember.Controller.extend({
14   siteTitle: 'Welcome to Emberoverflow',
15
16   currentTime: function() {
17     return(new Date());
18   }
19 });
```

Don't forget to update our index.html.

Code sample[6]:

index.html

```
31 <script type="text/x-handlebars" id="index">
32   <div class="row">
33     <div class="col-md-8">
34       <h2>{{siteTitle}}</h2>
35       <p>It is {{currentTime}}</p>
36
37       <ul>
38       {{#each item in model}}
39         <li>{{item}}</li>
40       {{/each}}
41       </ul>
42     </div>
43   </div>
44 </script>
```

[4] In Ember computed properties are properties defined as a function which Ember will evaluate when we ask for the property.

[5] https://github.com/ryakh/emberoverflow/commit/2c66f8158fe85e3580462e91a17295f4c921d093

[6] https://github.com/ryakh/emberoverflow/commit/b515ce4c6758e344d862765b9395b6ab72061fbf

If you reload our application you will see `It is function () { return(new Date); }` which is probably not what we wanted. That happens because we did not to explicitly tell our controller that a property is indeed a property and can be called inside our template. We can do so by adding `.property()` method at the end of our function.

Code sample[7]:

js/app.js

```
13 App.IndexController = Ember.Controller.extend({
14   siteTitle: 'Welcome to Emberoverflow',
15
16   currentTime: function() {
17     return(new Date());
18   }.property()
19 });
```

If you reload our application, you should see current date and time.

3.3 Ember Controller Types

There are two controller types. Both of them inherit from Controller object. They are ArrayController and ObjectController.

Controller should be used when your controller is not a proxy (i.e. it is not an object or array of objects; for example an application controller) or it should be used when you are building your own controller type.

ObjectController is used to represent a single model.

ArrayController is used to represent an array of models. ArrayController will consist of many ObjectController's for each model in array.

3.4 Ember CLI Way

Snapshot of the application[8]

Using knowledge you've gained from the previous chapter I want to demonstrate one of Ember CLI principles — different way of exporting modules.

Here is a checklist of things we are going to do:

1. create `IndexController`,
2. update template to display static property,
3. define property as dynamic function.

[7] https://github.com/ryakh/emberoverflow/commit/5f09f21e0ea56b9a3b656b8940d9364e17dea7cf
[8] https://github.com/ryakh/emberoverflow-cli/commit/6ed86ff4fda17561dfd197f66ba31f669109491b

3.4.1 Creating Index Controller

Again, instead of creating new files manually we are going to use `ember generate` command:

```
ember generate controller index --type=basic
```

Here we've used `--type=` option to specify controller type to be generated. As you already know Ember has three controller types (you can specify which one you want to generate by providing specific value of `--type=` option):

1. Controller (can be generated using `--type=basic`),
2. ObjectController (`--type=object`),
3. ArrayController (`--type=array`).

If you read your console output carefully you will notice that Ember CLI created two files — `app/controllers/index.js` and `tests/unit/controllers/index-test.js`. We will deal with test in fifth chapter; right now just keep in mind that whenever you generate new controller (or even any other module using Ember CLI generator), Ember CLI will generate new test suite for that controller as well.

You can omit `--type=` option completely; this will have same result as providing `--type=basic` option.

Go ahead and open our newly generated controller. Here is what you will see:

Code sample[9]:

app/controllers/index.js

```
import Ember from 'ember';

export default Ember.Controller.extend({
});
```

Another way to define module in Ember is using following syntax:

[9] https://github.com/ryakh/emberoverflow-cli/commit/7f2111453d265605476a26f7bfddd060757a70ac

app/controllers/index.js

```
import Ember from 'ember';

var IndexController = Ember.Controller.extend({
});

export default IndexController;
```

The only difference is that in first case Ember CLI will name your module automatically depending on file name and path. In second case you are naming module yourself. Even though second way of defining modules can be more appealing I encourage you not to name your modules manually since Ember CLI keeps their names according to Ember conventions and thanks to that it provides us with good failsafe mechanism for free.

3.4.2 Static Properties

Defining properties in Ember CLI modules is straight forward.

Code sample[10]:

app/controllers/index.js

```
import Ember from 'ember';

export default Ember.Controller.extend({
  siteTitle: 'Welcome to Emberoverflow'
});
```

Ember Starter Kit provided us with predefined `index` template. The only template Ember CLI creates for us is `application.hbs`; so we need to create new template called `index`. We can generate one using Ember CLI generator:

```
ember generate template index
```

Go ahead and update it with property we've defined earlier.

[10] https://github.com/ryakh/emberoverflow-cli/commit/187037edecbbfaa6c13faaa64ea6f06dca0179e6

Code sample[11]:

app/templates/index.hbs

```
<div class="row">
  <div class="col-md-8">
    <h2>{{siteTitle}}</h2>
  </div>
</div>
```

3.4.3 Computed Properties

We have already created both controller and template. So right now lets just go ahead and define `currentTime` property to display current date and time inside our `index` template.

Code sample[12]:

app/controllers/index.js

```
import Ember from 'ember';

export default Ember.Controller.extend({
  siteTitle: 'Welcome to Emberoverflow',

  currentTime: function() {
    return(new Date());
  }.property()
});
```

app/templates/index.hbs

```
<div class="row">
  <div class="col-md-8">
    <h2>{{siteTitle}}</h2>
    <p>It is {{currentTime}}</p>
  </div>
</div>
```

[11] https://github.com/ryakh/emberoverflow-cli/commit/8356095044e425c16e9cdb796cc4588a335a3aaa
[12] https://github.com/ryakh/emberoverflow-cli/commit/6ed86ff4fda17561dfd197f66ba31f669109491b

4. Routing, View Tree and Naming Conventions

Topics covered in this chapter:

1. introduction to route objects,
2. brief explanation of the request cycle,
3. setting model for the template,
4. inspecting View Tree with Ember Inspector,
5. naming conventions.

Ember CLI Way

1. generating routes,
2. introduction to Ember CLI naming conventions.

Snapshot of the application[1].

Back in the first chapter we left a piece of code which was provided inside Ember Starter Kit unexplained. I did that on purpose because at that point we lacked knowledge and it could be a little bit confusing trying to explain route object. Now that you know about templates and controllers lets dig into routes. Open our `app.js` file and search for `IndexRoute`:

js/app.js

```
7  App.IndexRoute = Ember.Route.extend({
8    model: function() {
9      return ['red', 'yellow', 'blue'];
10   }
11 });
```

This may seem a little bit confusing at first glance. We have a router and then we are defining separate routes using route objects. And model? Isn't it controller's responsibility to provide a model to the template? Well, no. Remember how I told you to forget everything you know about MVC frameworks in the previous chapter? If you didn't do it back then now it's a good time to free your

[1] https://github.com/ryakh/emberoverflow/tree/ef3ca9d3f809de6d971f827349bb953d08e2ee07

mind. Don't try to understand Ember using knowledge you have from other frameworks — it will not get you anywhere and it will hurt your brain.

Lets recapitulate everything we know so far. We have application. Application has a router which receives requests made by user (which are represented by URLs and are typed into the browser's address bar) and then router sends user requests to controller. Controller decorates our data and sends data to template. Template displays data to the user. Wait, but if controller only decorates data, where does data come from? This is where route comes in.

Router defines routes for our application and tells our application which routes to accept from our user. Requests made to router are further handled by route objects. Route objects are responsible for loading data from defined data storage and providing data to the controller. Controller decorates data received from route and sends it to the template so it can be displayed to our user.

Lets get back to our code. We have `model` property defined inside our `IndexRoute`. It returns an array of hard coded values.

Ember expects `model` property of a route object to return either an object or an array.

Now lets take a look at index template:

index.html

```
31 <script type="text/x-handlebars" id="index">
32   <div class="row">
33     <div class="col-md-8">
34       <h2>{{siteTitle}}</h2>
35       <p>It is {{currentTime}}</p>
36
37       <ul>
38       {{#each item in model}}
39         <li>{{item}}</li>
40       {{/each}}
41       </ul>
42     </div>
43   </div>
44 </script>
```

Here is what happens inside our application — it receives request to show the `index` route. Using Ember conventions it dispatches request to the `IndexRoute`, there we are setting the model which

is then sent into the `IndexController`, `IndexController` decorates our data, provides additional properties and sends data to the template which renders data with the help of `{{each}}` block expression.

We are building Q&A site, so red, yellow and blue are probably not kind of values we want our users to see. Lets make our `IndexRoute` display list of latest questions. Open our `IndexRoute` and change it to look like following.

Code sample[2]:

js/app.js

```
7  App.IndexRoute = Ember.Route.extend({
8    model: function() {
9      var questions = [
10       {
11         title:  'How do I feed hamsters?',
12         author: 'Tom Dale'
13       },
14
15       {
16         title:  'Are humans insane?',
17         author: 'Tomster the Hamster'
18       }
19     ]
20
21     return questions;
22   }
23 });
```

[2] https://github.com/ryakh/emberoverflow/commit/a6c795c2c373f8d5fbade43f7b2306b89f1f1e34

If you reload our application, you will see a list of two items each having a value of `[objectObject]`. That happens because within `{{each}}` block expression we are printing out an item from within model (which is an object). What we need to do is to provide a values from object instead.

Code sample[3]:

index.html

```
31 <script type="text/x-handlebars" id="index">
32   <div class="row">
33     <div class="col-md-8">
34       <h2>{{siteTitle}}</h2>
35       <p>It is {{currentTime}}</p>
36
37       <ul>
38       {{#each item in model}}
39         <li>
40           {{item.title}} – asked by {{item.author}}
41         </li>
42       {{/each}}
43       </ul>
44     </div>
45   </div>
46 </script>
```

Now if you remember controller types described in the previous chapter you probably have guessed that we need to use `ArrayController` in this case (although the solution presented above will work). So lets change our controller from `Controller` to `ArrayController`.

Code sample[4]:

js/app.js

```
25 App.IndexController = Ember.ArrayController.extend({
26   siteTitle: 'Welcome to Emberoverflow',
27
28   currentTime: function() {
29     return(new Date);
30   }.property()
31 });
```

[3] https://github.com/ryakh/emberoverflow/commit/a029831177a45caa7d0021b3f73b28c5f3c5f1d9

[4] https://github.com/ryakh/emberoverflow/commit/fcbcf8bfd80229b53ffeb34cf9d85575645bdfe0

Find `{{#each item in model}}` block expression in our template. If we do not provide it with any further arguments Ember will look for a model that we passed from `IndexRoute` to the `IndexController`. This way we can simplify our template to the following syntax.

Code sample[5]:

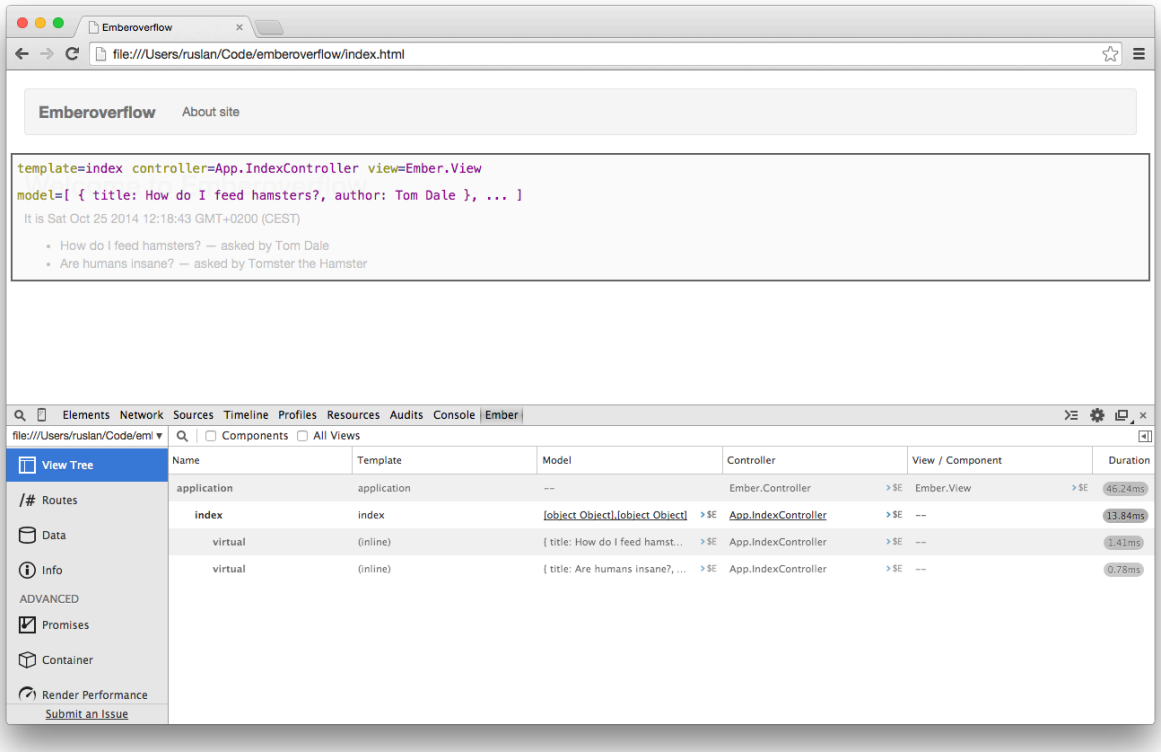
index.html

```
31    <script type="text/x-handlebars" id="index">
32      <div class="row">
33        <div class="col-md-8">
34          <h2>{{siteTitle}}</h2>
35          <p>It is {{currentTime}}</p>
36
37          <ul>
38          {{#each}}
39            <li>
40              {{title}} – asked by {{author}}
41            </li>
42          {{/each}}
43          </ul>
44        </div>
45      </div>
46    </script>
```

[5] https://github.com/ryakh/emberoverflow/commit/ef3ca9d3f809de6d971f827349bb953d08e2ee07

4.1 Inspecting View Tree

Time for some theory about Ember Inspector. Open View Tree. There you will see table with two entries: `application` and `index`. View Tree represents layout of currently active route. If you hover over the single entries, Ember Inspector will highlight routes with some additional information:



Ember Inspector — View Tree

There are two entries in the View Tree at this moment — `application` and `index`. Ember will render `application` route and then nest all other currently active routes underneath it. In our case `index` is rendered "inside" `application`.

The View Tree table has five different columns — name, template, model, controller and view. Each one represents the object of Ember framework used to render the route.

You can click on any entry in the table to show more information about it. For example if you'd like to see more information about `IndexController`, simply click on it in the View Tree.

Using View Tree while inside Ember Inspector you can always quickly find out which parts of our application were used to render route we are currently seeing.

4.2 Naming Conventions

Ember is very opinionated framework. It will identify which objects to stitch together based only on their names. It surely is possible to override names with your own, although it will be better to stick to Ember defaults. So far we've covered templates, controllers, router and routes. These Ember objects can be linked to one another only by setting the correct names (there are also views that can be linked with other objects; we will cover them in depth later).

This table illustrates naming conventions used in Ember:

Route Name	Route	Controller	Template	View
`index`	`IndexRoute`	`IndexController`	`index`	`IndexView`
`about`	`AboutRoute`	`AboutController`	`about`	`AboutView`

So if we navigate to `index` route, Ember will search for route, controller, template and view according to it's naming conventions and if none are found it will generate one for us and keep it in memory (if you look back into the View Tree inside Ember Inspector and hove over the `index` route, you will see that the value of the View is set to `virtual` — that means that the View is generated for us).

4.3 Ember CLI Way

Snapshot of the application[6]

In this chapter we did not introduce any breaking concepts and all changes are straight forward.

Here is a checklist of things we will do in this part of the chapter:

1. generate new route,
2. update our controller/route,
3. update our template.

First we will generate new route:

```
1  ember generate route index
```

Lets stop here for a moment. Generating new route will also generate new template and `IndexRoute` test file. Although in previous chapter we've created `index` template; thats why Ember CLI will ask us following question:

[6] https://github.com/ryakh/emberoverflow-cli/tree/e55f60d3c76504c4a2a3675bce6b401554a550d2

```
[?] Overwrite /emberoverflow-cli/app/templates/index.hbs? (Yndeh)
```

But what does `Yndeh` means? It is a shortcut for Yes—no—diff—edit—help. Typing in `y` will overwrite existing file with a new one, `n` will keep existing file discarding new one, `d` will show differences between two, `e` will open your default editor to edit diff file and `h` will list each option (expanded) on separate line.

We want to keep our existing file so we should type in `n`.

Now we need to update our controller, template and route files to match code we've written in previous part of this chapter.

Code sample[7]:

app/controllers/index.js

```
import Ember from 'ember';

export default Ember.ArrayController.extend({
  siteTitle: 'Welcome to Emberoverflow',

  currentTime: function() {
    return(new Date);
  }.property()
});
```

[7] https://github.com/ryakh/emberoverflow-cli/commit/e55f60d3c76504c4a2a3675bce6b401554a550d2

app/routes/index.js

```
import Ember from 'ember';

export default Ember.Route.extend({
  model: function() {
    var questions = [
      {
        title:  'How do I feed hamsters?',
        author: 'Tom Dale'
      },

      {
        title:  'Are humans insane?',
        author: 'Tomster the Hamster'
      }
    ]

    return questions
  }
});
```

app/templates/index.hbs

```
<div class="row">
  <div class="col-md-8">
    <h2>{{siteTitle}}</h2>
    <p>It is {{currentTime}}</p>

    <ul>
    {{#each}}
      <li>
        {{title}} — asked by {{author}}
      </li>
    {{/each}}
    </ul>
  </div>
</div>
```

4.3.1 Ember CLI Naming Conventions

Since we did not learn anything new about Ember CLI in this chapter lets stop for a second and talk about Ember CLI naming conventions.

Remember, Ember CLI is only a command line utility that improves your development experience. It does not alter Ember's conventions or philosophy in a breaking way.

Keep in mind that Ember CLI uses filenames to name modules (this helps you by not having to namespace everything yourself). Here are two basic conventions which Ember CLI uses:

1. all filenames should be lowercased,
2. use dashes instead of camelcase, snakecase,...

So to create route named `ListOfQuestionsRoute` you should create file called `list-of-questions.js` under `app/routes` directory. Same principle applies to Controllers, Templates and Views.