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 2014-03-01

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 the language. Some basic knowledge of jQuery is required though 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].

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

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 a list of topics covered in the chapter right at the beginning — this should give you general overview about the it's content. After the list 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.

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[2]:

code_snippet.js

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

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

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

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

1 Setting Things Up

Topics covered in this chapter:

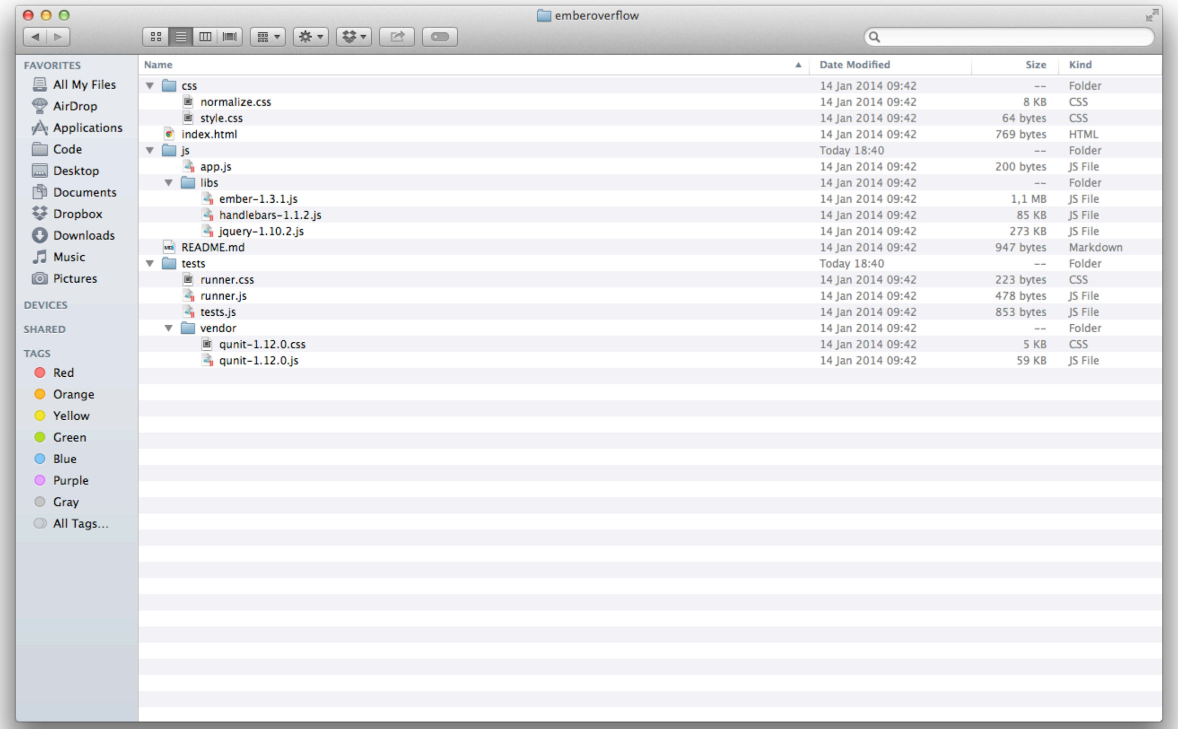
1. preparing scaffolds for development,
2. anatomy of the Ember Starter Kit,
3. quick introduction to Ember's anatomy,
4. setting up Ember Inspector.

Snapshot of the application[1].

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

[1]https://github.com/ryakh/emberoverflow/tree/ca3b39a5c034e036a17538ed599228c38c0b5a27

[2]http://emberjs.com/



Name	Date Modified	Size	Kind
css	14 Jan 2014 09:42	--	Folder
normalize.css	14 Jan 2014 09:42	8 KB	CSS
style.css	14 Jan 2014 09:42	64 bytes	CSS
index.html	14 Jan 2014 09:42	769 bytes	HTML
js	Today 18:40	--	Folder
app.js	14 Jan 2014 09:42	200 bytes	JS File
libs	14 Jan 2014 09:42	--	Folder
ember-1.3.1.js	14 Jan 2014 09:42	1,1 MB	JS File
handlebars-1.1.2.js	14 Jan 2014 09:42	85 KB	JS File
jquery-1.10.2.js	14 Jan 2014 09:42	273 KB	JS File
README.md	14 Jan 2014 09:42	947 bytes	Markdown
tests	Today 18:40	--	Folder
runner.css	14 Jan 2014 09:42	223 bytes	CSS
runner.js	14 Jan 2014 09:42	478 bytes	JS File
tests.js	14 Jan 2014 09:42	853 bytes	JS File
vendor	14 Jan 2014 09:42	--	Folder
qunit-1.12.0.css	14 Jan 2014 09:42	5 KB	CSS
qunit-1.12.0.js	14 Jan 2014 09:42	59 KB	JS File

Ember Starter Kit folder structure

Lets start by dropping in the 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

```html
<head>
  <meta charset="utf-8">
  <title>Ember Starter Kit</title>
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/style.css">
  <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/css\
/bootstrap.min.css">
</head>
```

[3] http://getbootstrap.com/

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

[5] https://github.com/ryakh/emberoverflow/commit/40f78d9bee63a50b02ac02bce57f4c5181a71220

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

Code sample[6]:

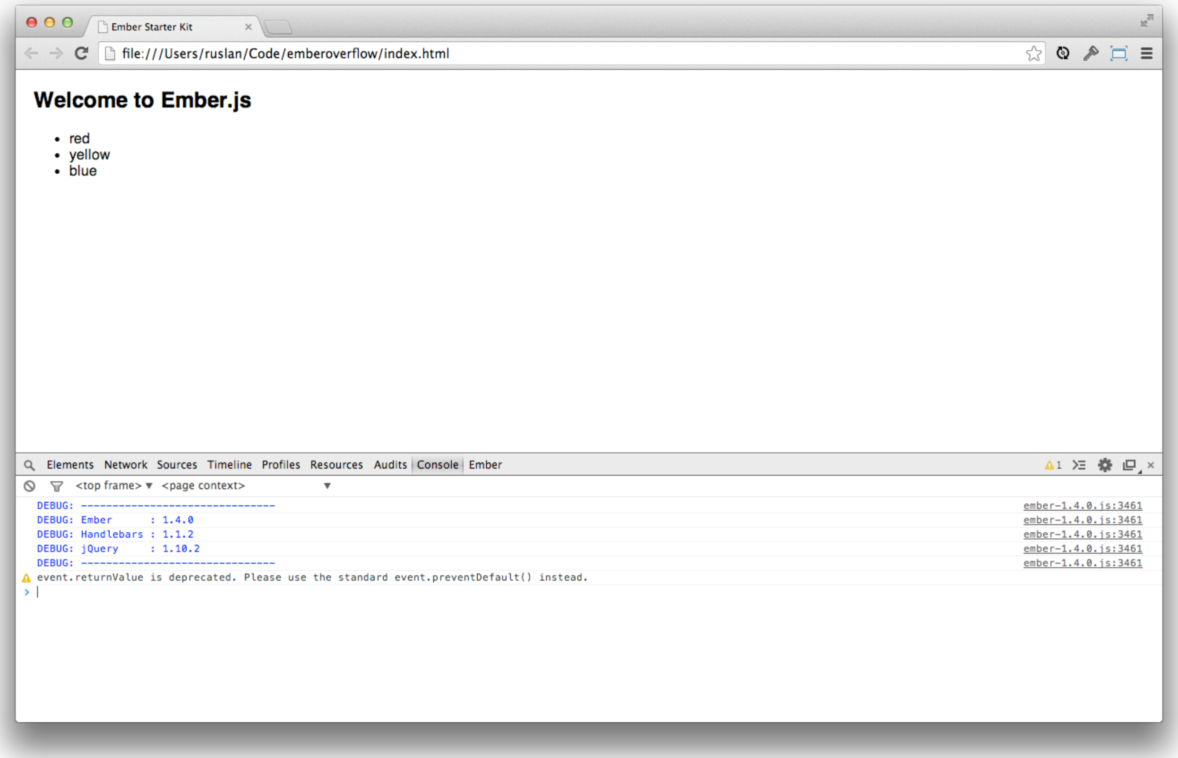
index.html

```
  <script src="js/libs/jquery-1.10.2.js"></script>
  <script src="js/libs/handlebars-1.1.2.js"></script>
  <script src="js/libs/ember-1.4.0.js"></script>
  <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js\
"></script>

  <script src="js/app.js"></script>
  <!-- to activate the test runner, add the "?test" query string parameter -->
  <script src="tests/runner.js"></script>
</body>
```

[6]https://github.com/ryakh/emberoverflow/commit/4195147f8b780c79929e3d3bb148fa3899c1c66a

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


Ember Starter Kit
file:///Users/ruslan/Code/emberoverflow/index.html
Elements Network Sources Timeline Profiles Resources Audits Console Ember
<top frame> <page context>

Welcome to Ember.js

- red
- yellow
- blue

```
DEBUG: -------------------------------                 ember-1.4.0.js:3461
DEBUG: Ember      : 1.4.0                              ember-1.4.0.js:3461
DEBUG: Handlebars : 1.1.2                              ember-1.4.0.js:3461
DEBUG: jQuery     : 1.10.2                             ember-1.4.0.js:3461
DEBUG: -------------------------------                 ember-1.4.0.js:3461
event.returnValue is deprecated. Please use the standard event.preventDefault() instead.
>
```

Empty Ember Starter Kit application

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

1.1 Turning the 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 the content we see in our browser comes from? If you are a little bit 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 a body which contains two script tags of type `text/x-handlebars` and then we load external javascript files. Lets start with the latter:

index.html

```
  <script src="js/libs/jquery-1.10.2.js"></script>
  <script src="js/libs/handlebars-1.1.2.js"></script>
  <script src="js/libs/ember-1.4.0.js"></script>
  <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js\
"></script>

  <script src="js/app.js"></script>
  <!-- to activate the test runner, add the "?test" query string parameter -->
  <script src="tests/runner.js"></script>
</body>
```

Here we load all the requirements for Ember. They are: jQuery and Handlebars (a 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 later 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 the following line to you script files.

Code sample[8]:

index.html

```
  <script src="js/libs/jquery-1.10.2.js"></script>
  <script src="js/libs/handlebars-1.1.2.js"></script>
  <script src="js/libs/ember-1.4.0.js"></script>
  <script src="http://builds.emberjs.com/beta/ember-data.js"></script>
  <script src="http://netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js\
"></script>

  <script src="js/app.js"></script>
  <!-- to activate the test runner, add the "?test" query string parameter -->
  <script src="tests/runner.js"></script>
</body>
```

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

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>
```

[8]https://github.com/ryakh/emberoverflow/commit/ca3b39a5c034e036a17538ed599228c38c0b5a27

1.2 Where the Magic Happens

As you've probably guessed, all the magic here happens thanks to the 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 the 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 the address into the Route.

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

Model
: Defines the structure and the logic behind the 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 the new records before they are synchronised to the server.

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

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

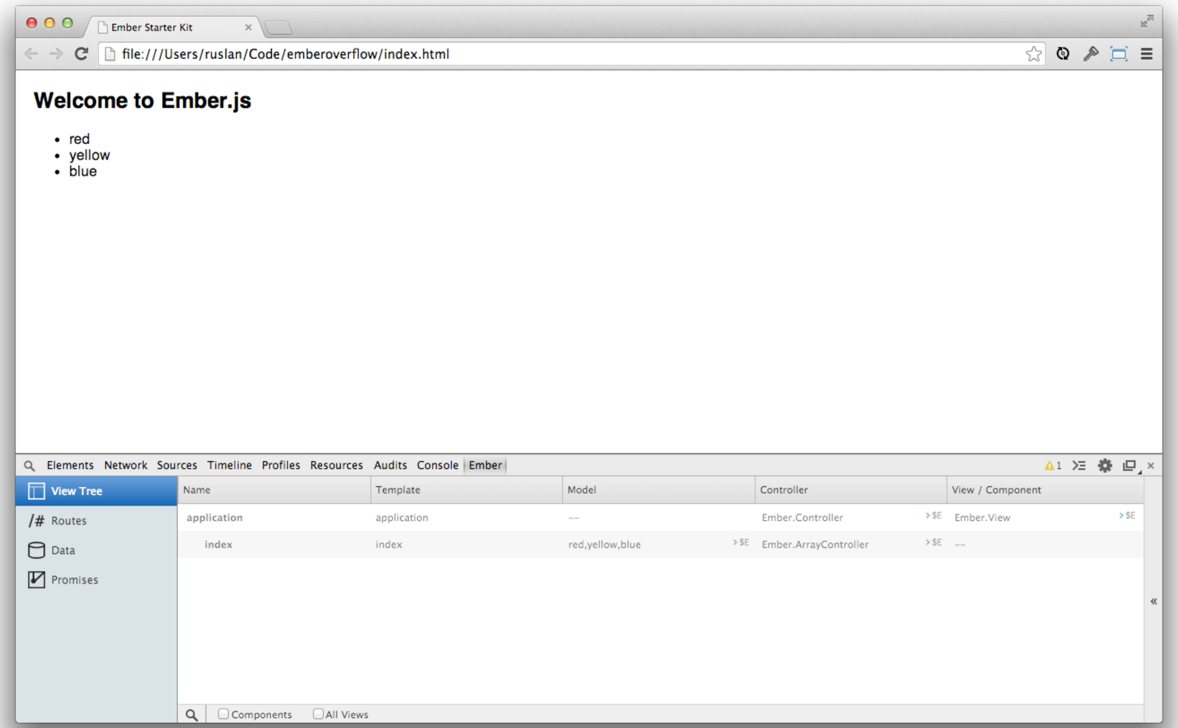
Component
: Component is very similar to the View. It is used to create reusable parts for our application.

Template
: Templates are the parts of Ember application that other people see. Components, Views and Controllers all 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's 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 the 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 Starter Kit
file:///Users/ruslan/Code/emberoverflow/index.html
Welcome to Ember.js
red
yellow
blue
Elements Network Sources Timeline Profiles Resources Audits Console Ember
View Tree
Routes
Data
Promises
Components
All Views

Name	Template	Model	Controller	View / Component
application	application	--	Ember.Controller	Ember.View
index	index	red,yellow,blue	Ember.ArrayController	--

Ember Inspector

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

2 First Templates, First Routes

Topics covered in this chapter:

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

Snapshot of the application[1].

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

index.html source code viewed in browser

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

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

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

js/app.js

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

With this single line we've created our application and put it into the namespace of `App`. Later on, to add more functionality we will simply define it as properties of our `App` object. Note that you need to create an 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/4c4d250bd2c7df5a8fb2bbbe15d1eddafe6260f0

You can pass in different options into the `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 does 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 a router to listen to URL changes and respond in a correct way.

2.1 Gentle Introduction to the Templates

So creating our application will render an 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>
```

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

So whenever you request a route by changing a URL in a browser, Ember will render the template that belongs to the route requested. The second template has ID equal to `index`. That means that it will be rendered in the `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 on our templates.

2.2 Adding our First Route

As we'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 to access that part of our application. Inside our `app.js` change the router to the following.

Code sample[3]:

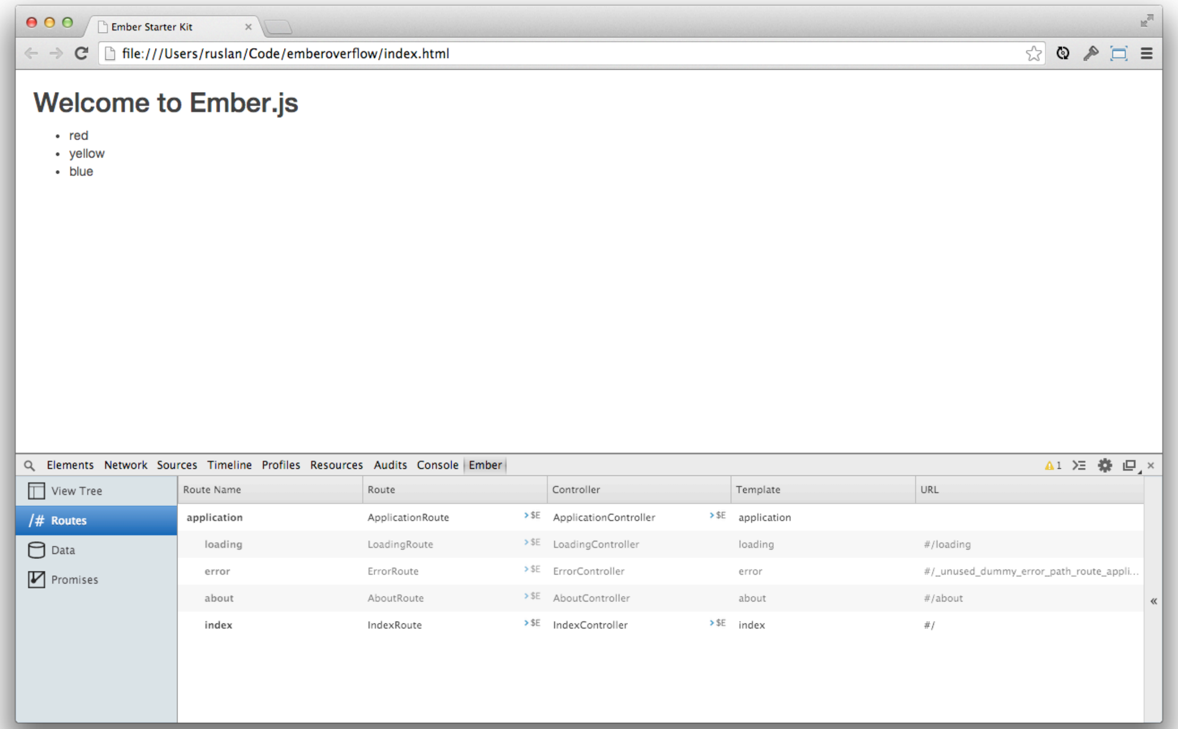
js/app.js

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

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

[2]Text inside two curly braces is called **Handlebars expression**. It will be picked up by the Handlebars library and parsed

[3]https://github.com/ryakh/emberoverflow/commit/c314f3ffe57ebaf3fef7d9ad8c4afd3deca5c31f


Ember Starter Kit
file:///Users/ruslan/Code/emberoverflow/index.html
Elements Network Sources Timeline Profiles Resources Audits Console Ember
View Tree
/# Routes
Data
Promises

Welcome to Ember.js

- red
- yellow
- blue

Route Name	Route	Controller	Template	URL
application	ApplicationRoute $E	ApplicationController $E	application	
loading	LoadingRoute $E	LoadingController	loading	#/loading
error	ErrorRoute $E	ErrorController	error	#/_unused_dummy_error_path_route_appli...
about	AboutRoute $E	AboutController	about	#/about
index	IndexRoute $E	IndexController $E	index	#/

Routes Inside our Application

Note the routes Ember created for 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. Note the 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 a true-false expression.

Lets update our application template to hold a navigation using which our user will be able to navigate through the application.

Code sample[4]:

index.html

```html
<script type="text/x-handlebars">
  <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}}
</script>
```

`{{link-to}}` block expression accepts different arguments. First one is the route into which a link will resolve. Remember the routes we viewed using Ember Inspector? Here we are passing in the name of the route (the one I've told you to remember). Handlebars will take the `link-to` block, parse it into regular `<a></a>` tag, apply all of the provided arguments to link 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/1da47c96f610a848c6273abb467086df128c09db

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

Code sample[5]:

css/style.css

```
.active {
  font-weight: bold;
}
```

Go ahead and try try to click on the links we have in our application. You will notice that the currently active link gets a class of active (and is displayed in bold). Ember gives you that feature for free — currently active links will always get the class of active.

2.3 Providing Content for Context

We can navigate by clicking on our links but we have no content for the About page so far. Lets fix this. Change `index` template and add another one called `about`:

Code sample[6]:

index.html

```
<script type="text/x-handlebars" id="index">
  <div class="row">
    <div class="col-md-8">
      <h2>Welcome to Emberoverflow</h2>

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

<script type="text/x-handlebars" id="about">
  <div class="row">
    <div class="col-md-8">
```

[5] https://github.com/ryakh/emberoverflow/commit/acc623c1046da32c29ac1efd679af67530f6c921
[6] https://github.com/ryakh/emberoverflow/commit/4c4d250bd2c7df5a8fb2bbbe15d1eddafe6260f0

```
      <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>
</script>
```

Go ahead and reload our application. Now if you click on the link, then 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 note that for Ember it is enough to provide a route (an 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 the name of the template Ember expects you to provide for the given route, which in our case is `about`.

3 Controllers

Topics covered in this chapter:

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

Snapshot of the application[1].

If you have experience with another web framework such as Rails or Django you may think that you already know what a 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 sideweb 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 the Rails MVC model go it was much easier for me to understand the internals of Ember. Ember is not a MVC framework. Ember is a framework for building client side web applications.

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

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 the one behind the scenes and keep it in memory for later use.

[1]https://github.com/ryakh/emberoverflow/tree/d1c89c707ab3d418bbcc5db0a7951bd7f76c2e55

You are probably familiar with code generators. The ones that create a basic code 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 the code when it needs it (which in most cases means that you didn't provide the code) and instead of creating code file on your hard drive Ember will create one 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

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

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

Code sample[3]:

index.html

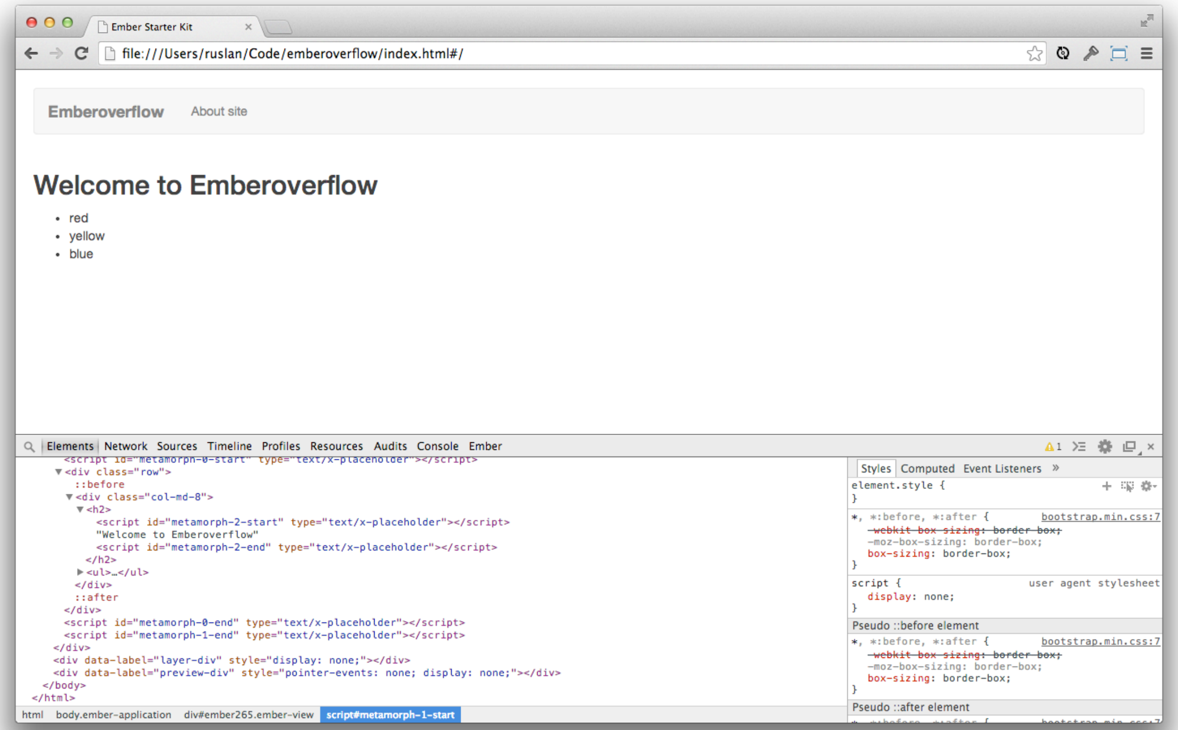
```
<script type="text/x-handlebars" id="index">
  <div class="row">
    <div class="col-md-8">
      <h2>{{siteTitle}}</h2>

      <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 the value of `siteTitle` property inside our controller it will be changed inside our application as well.

[2]https://github.com/ryakh/emberoverflow/commit/d66111735a0243738d8105d710a323782ebbb577
[3]https://github.com/ryakh/emberoverflow/commit/9c82333375fb5e8307e6d624345d7b5601103f21

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



```
    <script id="metamorph-0-start" type="text/x-placeholder"></script>
  ▼<div class="row">
     ::before
   ▼<div class="col-md-8">
     ▼<h2>
        <script id="metamorph-2-start" type="text/x-placeholder"></script>
        "Welcome to Emberoverflow"
        <script id="metamorph-2-end" type="text/x-placeholder"></script>
      </h2>
     ▶<ul>…</ul>
     </div>
     ::after
    </div>
    <script id="metamorph-0-end" type="text/x-placeholder"></script>
    <script id="metamorph-1-end" type="text/x-placeholder"></script>
  </div>
  <div data-label="layer-div" style="display: none;"></div>
  <div data-label="preview-div" style="pointer-events: none; display: none;"></div>
 </body>
</html>
```

```
element.style {
}
*, *:before, *:after {                bootstrap.min.css:7
  ~~-webkit-box-sizing: border-box;~~
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
script {                            user agent stylesheet
  display: none;
}
Pseudo ::before element
*, *:before, *:after {                bootstrap.min.css:7
  ~~-webkit-box-sizing: border-box;~~
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
Pseudo ::after element
```

What's with weird script tags?

Note the 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 the script tags around our attribute, so something like this won't work:

Inserting inline attribute, the wrong way

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

This will generate the following code:

Inserting inline attribute, the wrong way

```
<img src="&lt;script id='metamorph-7-start' type='text/x-placeholder'&gt;&lt;/scr\
ipt&gt;image.jpg&lt;script id='metamorph-7-end' type='text/x-placeholder'&gt;&lt;\
/script&gt;">
```

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

Inserting inline attribute using bound attribute

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

Which will generate the correct HTML:

Inserting inline attribute using bound attribute

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

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

3.2 Properties as Functions

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

Code sample[5]:

js/app.js

```
App.IndexController = Ember.Controller.extend({
  siteTitle: 'Welcome to Emberoverflow',

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

And don't forget to update our index.html.

Code sample[6]:

index.html

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

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

[4]In Ember computed properties are properties defined as a function which Ember will call when we ask for the property
[5]https://github.com/ryakh/emberoverflow/commit/b501531e80b307c9c4ff06f0331a7a71abadd51e
[6]https://github.com/ryakh/emberoverflow/commit/9881f8d289170416f0e07d4b2e230576bc39788d

If you reload the page you will see `It is function () { return(new Date); }` which is probably not what we wanted. Thats because we need to explicitly tell our controller that a property is a 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

```
App.IndexController = Ember.Controller.extend({
  siteTitle: 'Welcome to Emberoverflow',

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

If you reload our application, you should see the correct 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 ObectController's for each model in array.

[7] https://github.com/ryakh/emberoverflow/commit/d1c89c707ab3d418bbcc5db0a7951bd7f76c2e55

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.

Snapshot of the application[1].

Back in 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 the route object. Now that you know about templates and controllers lets dig into the routes. Open our `app.js` and search for `IndexRoute`:

js/app.js

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

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

Lets recapitulate everything we know so far. We have an application. Application has a router which receives requests made by user (which are represented by URLs and are typed into the browser address bar) and then router sends user requests to the controller. Controller decorates our data

[1]https://github.com/ryakh/emberoverflow/tree/784253175719387bc99278d3f942002f358f00e4

and sends the data to the template. Template displays data to the user. Wait, but if controller only decorates data, where does the data come from? This is where the route comes in.

Router defines routes for our application and tells our application which routes to accept from the user. Request made to router are further handled by routes objects. Route objects are responsible for loading data from defined data storage and providing the data to the controller. Controller decorates the 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 in 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 the index template:

index.html

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

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

Here is what happens inside our application — it receives a request to show the `index` route. Using Ember conventions it dispatches the request to `IndexRoute`, there we set the model which is then sent into the `IndexController`, `IndexController` decorates our data, provides additional properties and sends the data to the template which renders the data with the help of `{{each}}` block expression.

We are building Q&A site, so the 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

```
App.IndexRoute = 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
  }
});
```

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

Code sample[3]:

index.html

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

      <ul>
      {{#each item in model}}
        <li>
          {{item.title}} – asked by {{item.author}}
        </li>
```

[2]https://github.com/ryakh/emberoverflow/commit/1ce63fed627c04e90c6e76da272ff98db5b559b1
[3]https://github.com/ryakh/emberoverflow/commit/e0d693fe3364494ded4be7d58a2398342afc6bdd

```
      {{/each}}
      </ul>
    </div>
  </div>
</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

```
App.IndexController = Ember.ArrayController.extend({
  siteTitle: 'Welcome to Emberoverflow',

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

[4]https://github.com/ryakh/emberoverflow/commit/0df321f857521c5cf90dc2babe24a093453c5f75

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

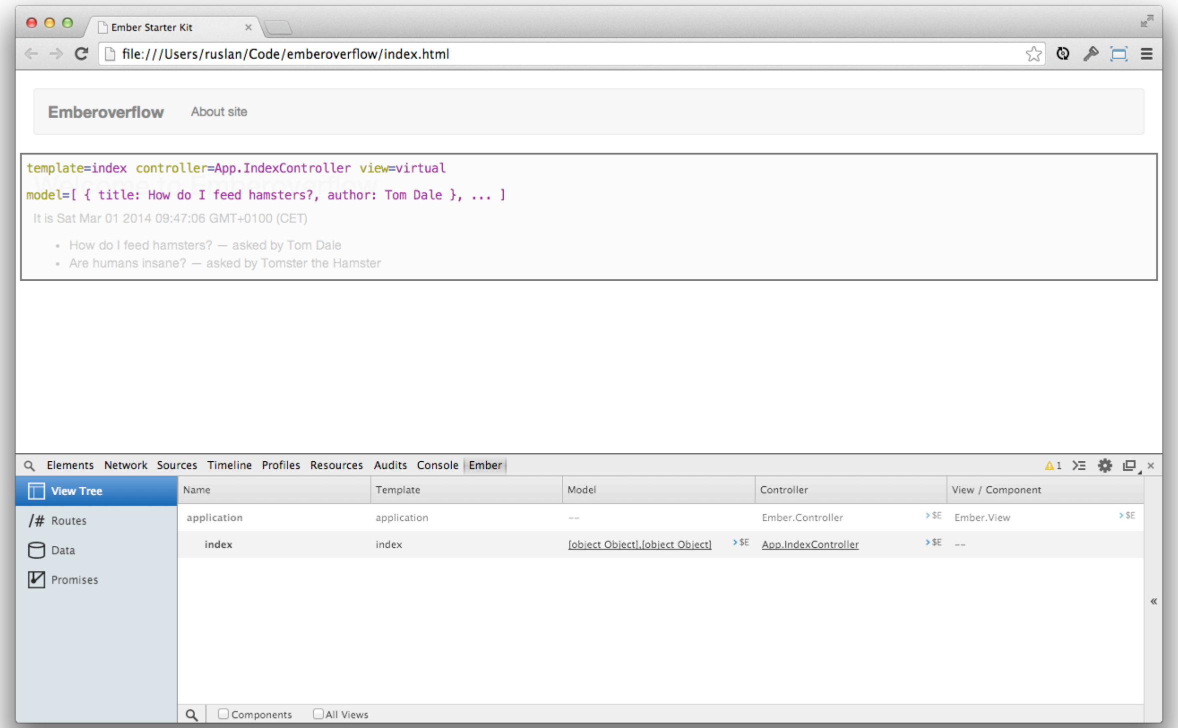
```
<script type="text/x-handlebars" id="index">
  <div class="row">
    <div class="col-md-8">
      <h2>{{siteTitle}}</h2>
      <p>It is {{currentTime}}</p>

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

[5]https://github.com/ryakh/emberoverflow/commit/784253175719387bc99278d3f942002f358f00e4

4.1 Inspecting the View Tree

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


Ember Starter Kit
file:///Users/ruslan/Code/emberoverflow/index.html
Emberoverflow
About site
Elements Network Sources Timeline Profiles Resources Audits Console Ember
View Tree
Routes
Data
Promises
Components
All Views

```
template=index controller=App.IndexController view=virtual
model=[ { title: How do I feed hamsters?, author: Tom Dale }, ... ]
```

It is Sat Mar 01 2014 09:47:06 GMT+0100 (CET)

- How do I feed hamsters? — asked by Tom Dale
- Are humans insane? — asked by Tomster the Hamster

Name	Template	Model	Controller	View / Component
application	application	--	Ember.Controller	Ember.View
index	index	[object Object],[object Object]	App.IndexController	--

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 the View Tree you can always quickly find out which parts of our application were used to render the 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 the 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).