**Getting Into Ember.js**

<http://dev.tutsplus.com/tutorials/getting-into-ember-js--net-30709>

*Ember.js is not a framework for building traditional websites.*

Ember also relies on client-side templates... a **LOT**. It uses the [Handlebars templating library](http://www.handlebarsjs.com/) which provides expressions that allow you to create dynamic HTML-based templates.

<ul>

 {{#each people}}

   <li>Hello, {{name}}!</li>

 {{/each}}

</ul>

## Setting up Ember

The easiest way to get the files you need is to go to the Ember.js Github repo and pull down the [Starter Kit](https://github.com/emberjs/starter-kit).

<script src="js/libs/jquery-1.9.1.js"></script>

<script src="js/libs/handlebars-1.0.0-rc.3.js"></script>

<script src="js/libs/ember-1.0.0-rc.1.js"></script>

<script src="js/app.js"></script>

It's important to understand how Ember.js works and that you grok the moving parts that make up an Ember app. Let's take a look at those parts and how they relate to each other.

## Templates

Handlebars is the client-side library used in Ember and the expressions provided by the library are used extensively when creating the UI for your application. Here's a simple example:

<script type="text/x-handlebars">

     <h2><strong>{{firstName}} {{lastName}}</strong></h2>

</script>

Notice that the expressions are mixed into your HTML markup and, via Ember, will dynamically change the content displayed on the page. In this case, the {{firstName}} and {{lastName}} placeholders will be replaced by data retrieved from the app.

## Routing

*An application's Router helps to manage the state of the application.*

An application's Router helps to manage the state of the application and the resources needed as a user navigates the app. This can include tasks such as requesting data from a model, hooking up controllers to views, or displaying templates.

The URL is the key identifier that Ember uses to understand which application state needs to be presented to the user.

App.Router.map( function() {

   this.route( 'about' ); // Takes us to "/about"

});

The behaviors of a route (e.g.: requesting data from a model) are managed via instances of the Ember route object and are fired when a user navigates to a specific URL. An example would be requesting data from a model, like this:

App.EmployeesRoute = Ember.Route.extend({

   model: function() {

       return App.Employee.find();

   }

});

In this case, when a user navigates to the "/employees" section of the application, the route makes a request to the model for a list of all employees.

## Models

*An object representation of the data.*

Models are an object representation of the data your application will use.

 The [Ember Data](https://github.com/emberjs/data) library offers the API for loading, mapping and updating data to models within your application.

## Controllers

Controllers are typically used to store and represent model data and attributes. They act like a proxy, giving you access to the model's attributes and allowing templates to access them to dynamically render the display. This is why a template will always be connected to a controller.

## Views

Views in Ember.js are meant to manage events around user interaction and translate them into events that have meaning within your application.

## Naming Conventions

One of the ways that Ember.js helps to minimize the amount of code needed and handle things for you behind the scenes is through naming conventions.

 For example, if I create a route, called "employees":

App.Router.map( function() {

   this.resource( 'employees' );

});

I would then name my components, like this:

* **Route object:** *App.EmployeesRoute*
* **Controller:** *App.EmployeesController*
* **Model:** *App.Employee*
* **View:** *App.EmployeesView*
* **Template:** *employees*

Using this naming convention serves a dual purpose. First, it gives you a semantic relationship between like components. Secondly, Ember can automatically create the necessary objects that may not exist (e.g.: a route object or a controller) and wire them up for use in your application.

In fact, this is specifically what Ember does at the global Application level, when you instantiate the Application object:

var App = Ember.Application.create();

That single line creates the default references to the application's router, controller, view and template.

* **Route object:** *App.ApplicationRoute*
* **Controller:** *App.ApplicationController*
* **View:** *App.ApplicationView*
* **Template:** *application*

Going back to the "employees" route that I created above, what will happen is that, when a user navigates to "/employees" in your application, Ember will look for the following objects:

* *App.EmployeesRoute*
* *App.EmployeesController*
* the *employees* template

If it doesn't find them, it will create an instance of each but simply won't render anything, since you haven't specified a model to derive data from or a template to display the data with. This is why the naming convention is so important. It allows Ember to know how to handle the tasks associated with a specific route, without you having to wire things up manually.

Notice that, in the first example, I used the singular name, "Employee," to define the model. That's on purpose. The very nature of the name "Employees" dictates that I may be working with 0 to many employees, so it's important to build a model that could provide the flexibility to return one employee or all employees. The singular naming convention of this model is not a requirement of Ember, as models themselves have no knowledge of the controllers that will use them later on. So you do have flexibility in naming them, but for consistency, sticking with this convention will make managing your code substantially easier.

The key takeaway is that by using a consistent naming scheme, Ember can easily manage the hooks that bind these components together without your needing to explicitly define the relationships via a ton of code.

*Full details of*[*Ember's naming conventions*](http://emberjs.com/guides/concepts/naming-conventions/)*are provided on the project's site and is a****must-read****.*

(<http://emberjs.com/guides/concepts/naming-conventions/>)

App.Router.map(function() {

this.resource('posts', function() { // the `posts` route

this.route('favorites'); // the `posts.favorites` route

this.resource('post'); // the `post` route

});

});

| **Route Name** | **Controller** | **Route** | **Template** |
| --- | --- | --- | --- |
| posts | PostsController | PostsRoute | posts |
| posts.favorites | PostsFavoritesController | PostsFavoritesRoute | posts/favorites |

## A Basic App

I noted previously that the easiest way to get the files you need is to go to the Ember.js Github repo and pull down the start kit

Open index.html in your browser, and you'll see the following:

**Welcome to Ember.js**

* red
* yellow
* blue