Ruby on Rails Explained for Front-End Developers

by Miles Matthias

a dojo4 product

Ruby on Rails Explained for Front-End Developers

Miles Matthias

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

This version was published on 2014-01-20

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 Miles Matthias by spreading the word about this book on Twitter!

The suggested hashtag for this book is #rorfrontend.

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=#rorfrontend

This book is dedicated to the awesome men and women of dojo4. This book never would have happened if it weren't for your continued inspiration and wisdom.

Contents

Introduction

By the end of this book, you'll be able to:

- know where to put your html, css, & js within the structure of a ruby on rails app
- how rails routes requests to controllers
- using views, layouts, and partials in controllers
- what the asset pipeline is and how to use it
- how controllers and views in rails work together to render a response in the browser
- how data is added to views and how to link to the data in your html

Ruby on Rails Directory Structure

Since the intended audience of this book are front end developers, we're going to skip over a lot of the rails structure and cover only the directories and files that the front end dev needs to know about to get his/her content to the browser. If you'd like to read about all of the rails structure, see the rails getting started guide (http://guides.rubyonrails.org/getting_started.html#creating-a-new-rails-project).

The core application code (the MVC business) in a rails project is in the `app` directory:

- **`app/controllers`** is where the controllers live in a rails application. Again, the controller is responsible for processing incoming requests by getting necessary data and rendering a view to the user. This is the entry point to behavior in the application.
- **`app/models`** is where the models live in a rails application. The models represent an instance of a data type in your application, perform functions unique to that data type, and persist and fetch data from a database.
- **`app/views`** is where the HTML views of the application live. They're traditionally written in ERB, but you can use whatever template engine you like (even none at all if you're so brash). You'll be putting your HTML in this directory and a controller that's responding to a request will incorporate your view with the `render` function (see how controllers and views work together).
- **`app/assets`** okay, here's where you'll be doing most of your work. Your javascript files go into the directory named **`app/assets/javascripts/`** and your css files go into the **`app/assets/stylesheets/`** directory. Also note that just like with views, you can append a filename to your file that tells rails which preprocessor to run your code through before serving it as plan javascript or css. For more on that, see "asset pipeline".

Within `app/assets`:

- **`app/assets/images`** Put all of your images here.
- **`app/assets/javascripts`** Your javascript code will be preprocessed, concatenated into files that you specify (the default is `application.js`), and minified for production by Rail's asset compilation engine, Sprockets.
- **`app/assets/stylesheets`** is where the default `application.css` file lives that Sprockets will compile for you.

- **`public`** is where static pages (like a 404 page), compiled assets (your plain and minified javascript and css after any preprocessors), and other static assets (like images) are placed to be served without any application logic by Apache (http://httpd.apache.org/) (Rails uses Apache as its web server – remember that bit about rails being a "full stack" framework?). The advantage of this is when a request is made for one of these assets, no rails code at all needs to be executed and the web server can simply serve the file back. This lowers stress on the rails application by not having to handle as many requests and makes for faster responses.

Within `public`:

- **`public/assets`** is where Sprockets will place compiled assets (CSS/JS) to be served by the web server. They will not be visible here in development because the Rails server will use Sprockets to compile them on the fly during each request. In production, they are compiled into this directory.

Lastly, like any good front-end developer, you'll want to lean on the experience of your peers by using some javascript and css libraries. Simply place those libraries in either `vendor/assets/javascripts` or `vendor/assets/stylesheets` and you'll be able to `require` them in your code in `app/assets`. Rails (technically Sprockets) will bundle everything up into one javascript or css file, put it in `public` and you'll be good to go.