Ruby on Rails and Angular JS

Website

Information

Guildford, Surrey

Ruby on rails and angular js

# Introduction

## Purpose

The purpose of this document is to provide information and details regarding the Ruby on Rails implemented with Angular JS.

## Scope

This document is suitable for those who wish to learn Ruby on Rails and Angular JS.

## Definitions, acronyms and abbreviations

Ruby on Rails:

A Web Applications Development Language commonly used in creating and making websites.

Angular JS

A Web Applications Development Language commonly used in creating and making websites.

## Overview

This document presents a description and detail of Ruby on Rails and Angular JS.

This document follows the IEEE ‘Recommended Practice for Software Requirements Specification” using the template organization for features [2].

# Implementation

## Setting up a basic skeleton application

Rails new receta –skip-bundle –quiet –database=postgresql

### Choosing our database

In this case, our database is postgresql and thus we must install postgresql onto our machines and configure and run the server. We’ll need to enter a desired username and password and this will the admin postgresql login authentication for which we’ll later require in rails.

### Testing

We’ll use RSpec <http://rspec.info/> with the rspec-rails gem making TDD (Test Driven Development) ‘productive and fun’ for Behaviour Driven Development. Capybara and Selenium will also be used and are common approaches integration/acceptance/browser tests.

### Configuring our database

Access the config/database.yml and set the username and password for your local database and then run bin/rake db:create.

\*Notice for configuring the username and password, these resemble the ones that you configured when setting up and installing postgresql on your local machine. Secondly, although this is acceptable in this example, there are more secure means of setting the username and password and certainly not in a file that is in version control (VC). From this resource <http://railsapps.github.io/rails-environment-variables.html> we can see the variety of approaches to this such as setting Unix Environment Variables, Using the Figaro Gem as well as creating a local\_env.yml file.

### Front End dependency management with bower

Rails doesn’t provide a way to manage front-end assets such as JS libraries or CSS. Rather than using RubyGems, we will use Bower which provides a manageable way to keep on top of dependencies between libraries.

#### Installing

To install Bower, this has to be done using npm, the ‘Node Package Manager’.

Installing Node and NPM depends on the OS and for MAC It is done using homebrew:

Brew install node.

Once NPM is installed, we then using the Node Package Manager we install bower:

Npm install –g bower.

Finally, we can add the bower gem to the gem file.

Gem ‘bower-rails’

Once Bower is installed, we can run ‘rake –T bower’ and we see a list of able tasks. Our dependencies go into a file called Bowerfile located at the root directory of the project, very much like Gemfile which we must create.

The dependencies to first come in are Angular and Twitter Bootstrap. The file ‘Bowerfile’ contents will be:

asset ‘angular’

asset ‘bootstrap-sass-official’

# vim: ft=ruby

To install the dependencies defined, we run the rake task:

bower:install

Bower install depencies in vendor/assets/bower\_components which should be under VS:

git add vendor/assets

git commit –m ‘angular and bootstrap’

The location enables you to separate Bower-Managed Libraries from Non-Bower-Managed Libraries.

Since vendor/assets/bower\_components isn’t rail standard, it needs to be added to asset path so the files are picked up, while there we can add a few lines to get the glyphicons working.

This is done in config/application.rb:

config.assets.paths << Rails.root.join("vendor", "assets", "bower\_components")

config.assets.paths << Rails.root.join("vendor", "assets", "bower\_components", "bootstrap-sass-official","assets","fonts")

config.assets.precompile << %r(.\*.(?:eot|svg|ttf|woff)$)

Finally these files need to be referenced in application.js and application.css.scss and the reference to Turbolinks should be removed. Rename application.css to application.css.scss because we need to use SASS directives:

**Application.js**

//= require jquery

//= require jquery\_ujs

//= require angular/angular

//= require\_tree .

**Application.css.scss**

@import "bootstrap-sass-official/assets/stylesheets/bootstrap-sprockets";

@import "bootstrap-sass-official/assets/stylesheets/bootstrap";

### Building our angular app

**Define default route for new rails controller home with action index.**

**Define home controller with action index.**

**Define a barebones Angular App which will be in app.coffee:**

receta = angular.module('receta', [

])

**Finally, we can create the view that uses some Bootstrap CSS as well as enough Angular to demonstrate working:**

**<div class="container-fluid" ng-app="receta">**

**<div class="panel panel-success">**

**<div class="panel-heading">**

**<h1 ng-if="name">Hello, {{name}}</h1>**

**</div>**

**<div class="panel-body">**

**<form class="form-inline">**

**<div class="form-group">**

**<input class="form-control" type="text" placeholder="Enter your name" autofocus ng-model="name">**

**</div>**

**</form>**

**</div>**

**</div>**

**</div>**

### Deployment with heroku

Production Deployment with Heroku, a cloud-based platform which is free and fairly unforgiven implying if its works on Heroku, it’ll work anywhere.

Heroku login.

Heroku create.

We also want to add a few Heroku specific gems:

+gem "foreman" +group :production, :staging do + gem "rails\_12factor" + gem "rails\_stdout\_logging" + gem "rails\_serve\_static\_assets" +end + group :test, :development do gem "rspec" gem "rspec-rails", "~> 2.0"

**Push to the Heroku Repo**

> bundle install > git add Gemfile Gemfile.lock > git commit -m 'heroku deployment' > git push heroku master

Should the above not result i.e. for git push heroku master and the result sees something like: ‘Push rejected, no Cedar-supported app detected’ visit Heroku support page and follow deployment help there.

After doing this, run heroku open to display page.

## Building the first feature

Now that our Application is setup with Angular and we have a way to manage assets we can use tests to drive our work. We’ll be building our first feature before testing though.

Although TDD is an effective practice, it is difficult to use regarding the creation of an application.

We’ll start with our UI and some application code.

### Steps

* Create a basic Search UI.
* Write a small amount of code to make it work.
* Set up our JS testing environments.
* Connect the back-end.

### Basic ui

Since the application’s views will not generally be served by Rails, the majority of the markup will live outside of the Rails views. With Angular, we’ll map routes to views and controllers.

app.config([ '$routeProvider', ($routeProvider)-> $routeProvider .when('/', templateUrl: "index.html" controller: 'SomeController' ) .when('/recipes/new', templateUrl: "new.html" controller: 'SomeOtherController' ) ])

This presents us with a problem. Angular will use the value of templateUrl to try and fetch the file we’ve specified via AJAX. This might actually work in development but will fail in production. The asset pipeline works different in Production.

In production mode, the asset pipeline will control the path and name of the assets it is serving. It will generate a hash for each asset and include that hash in the name. The requested name i.e. assets/index.html is actually assets/ 9834f200909a098a0a9a-index.html.

Angular Caches templates after serving for the first time, so really we just need to repopulate the cache. This way, Angular won’t need to request any assets. Angular-rails-templates gem exists and we will use this. We will also need angular-route module which enables routing.

After adding the asset to the bowerfile, we run rake bower:install and then do the referencing in our application.js to angular routes and templates.

Writing enough code for Angular to serve up a template, so we can work out the UI. Replace index.html with the following:

<div ng-app="receta"> <div class="view-container"> <div ng-view class="view-frame animate-view"></div> </div> </div>

**Ng-app** tells Angular which application should be loaded.

**Ng-view** tells it where to render views.

The view-container div and the view-frame and animate-view classes can be used to add view transition animations.

We now need to implement the Angular app so it renders a view. Since we need the angular-routes module as well as angular-rails-templates, our app definition will need to include them as dependencies. Our controller will also be here, a third dependency.

**Our app/assets/javascripts/app.coffee:**

receta = angular.module('receta',[ 'templates', 'ngRoute', 'controllers', ]) receta.config([ '$routeProvider', ($routeProvider)-> $routeProvider .when('/', templateUrl: "index.html" controller: 'RecipesController' ) ]) controllers = angular.module('controllers',[]) controllers.controller("RecipesController", [ '$scope', ($scope)-> ])

**Finally, create index.html. Angular-rails-templates will look for templates in app/assets/javascripts/template by default so we’ll put the file there. It will initially just have the static markup that demonstrates our UI.**

**Our app/assets/javascripts/templates/index.html**

<header class="row"> <h1 class="text-center col-md-6 col-md-offset-3">Find Recipes</h1> </header> <section class="row"> <form> <div class="form-group col-md-6 col-md-offset-3"> <label for="keywords" class="sr-only">Keywords</label> <input type="text" autofocus class="form-control" placeholder="Recipe name, e.g. Baked Potato"> </div> <div class="form-group col-md-6 col-md-offset-3 text-center"> <button class="btn btn-primary btn-lg">Search</button> </div> </form> </section> <hr> <section class="row"> <h1 class="text-center h2">Results</h1> <section class="well col-md-6 col-md-offset-3"> <h1 class="h3 col-md-6 text-right" style="margin-top: 0"><a href="#">Baked Potato w/ Cheese</a></h1> <div class="col-md-6"> <button class="btn btn-info">Edit</button> <button class="btn btn-danger">Delete</button> </div> </section> <section class="well col-md-6 col-md-offset-3"> <h1 class="h3 col-md-6 text-right" style="margin-top: 0"><a href="#">Garlic Mashed Potatoes</a></h1> <div class="col-md-6"> <button class="btn btn-info">Edit</button> <button class="btn btn-danger">Delete</button> </div> </section> <section class="well col-md-6 col-md-offset-3"> <h1 class="h3 col-md-6 text-right" style="margin-top: 0"><a href="#">Potatos Au Gratin</a></h1> <div class="col-md-6"> <button class="btn btn-info">Edit</button> <button class="btn btn-danger">Delete</button> </div> </section> </section>

**After completing all above, the display was empty and so after searching I found this to be a problem with sprockets and so I followed** [**https://github.com/pitr/angular-rails-templates/issues/93**](https://github.com/pitr/angular-rails-templates/issues/93)

1. Add gem ‘listen’, ‘~> 2.7’ to gemfile.
2. **Create config/initializers/angular\_templates.erb:**
   1. require 'fileutils' if Rails.env.development? cache\_path = Rails.root.join('tmp/cache/assets/development') FileUtils.rm\_rf(cache\_path) listener = Listen.to(Rails.root.join('app/assets/templates')) do |modified, added, removed| # clearing cache FileUtils.rm\_rf(cache\_path) end listener.start end
3. **Create app/assets/javascripts/templates.js.erb:**
   1. (function(){ 'use strict'; angular.module('templates', []).run(['$templateCache', function($templateCache) { <% environment.context\_class.instance\_eval { include ActionView::Helpers::JavaScriptHelper } app\_root = File.expand\_path('../../', \_\_FILE\_\_) templates = File.join(app\_root, %w{templates \*\* \*.html}) Dir.glob(templates).each do |f| key = f.gsub(%r(^#{app\_root}/templates/), '') html = File.read(f).squish %> $templateCache.put("<%= key %>", "<%= escape\_javascript(html) %>"); <% end %> }]); }());
4. **Finally, just require your templates in application.js:**
   1. //= require angular // ... //= require ./templates // ... angular.module('myApp', ['templates']);