APIs on Rails

Abraham Kuri

APIs on Rails

Abraham Kuri

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

This version was published on 2014-08-19

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.

Contents

Introduction

\label{cha:chapter_one}

Welcome to APIs on Rails[1] a tutorial on steroids on how to buid your next API with Rails. The goal of this book is to provide an answer on how to develop a RESTful API following the best practices out there, along with my own experience. By the time you are done with *API's on Rails* you should be able to build your own `API` and integrate it with any clients such as a web browser or your next mobile app. The code generated is built on top of Rails `4` which is the current version, for more information about this check out http://rubyonrails.org/[2]. The most up-to-date version of the *API's on Rails* can be found on http://apionrails.icalialabs.com[3]; don't forget to update your offline version if that is the case.

The intention with this book it's not teach just how to build an API with Rails rather to teach you how to build *scalable and maintanable API with Rails*, which means taking your current Rails knowledge to the next level when on this approach. In this journey we are going to take, you will learn to:

- Build JSON responses
- Use Git for version controlling
- Testing your endpoints
- Optimize and cache the API

I highly recommend you go step by step on this book, try not to skip chapters, as I mention *tips* and interesting facts for improving your skills on each on them. You can think yourself as the main character of a video game and with each chapter you'll get a higher level.

In this first chapter I will walk you through on how to setup your environment in case you don't have it already. We'll then create the application called `market_place_api`. I'll emphasize all my effort into teaching you all the best practices I've learned along the years, so this means right after initializing(Section~\ref{sec:initializing_project}) the project we will start tracking it with Git (Section~\ref{sec:version_control}).

In the next chapters we will be building the application to demonstrate a simple workflow I use on my daily basis. We'll develop the whole application using *test driven development (TDD)*, getting started by explaining why you want to build an API's for your next project and decising wheter to use JSON or XML as the response format. From Chapter~\ref{cha:chapter_three} to Chapter `8` we'll

[1] http://icalialabs.com

[2] http://rubyonrails.org/

[3] http://apionrails.icalialabs.com

get our hands dirty and complete the foundation for the application by building all the necessary endpoints, securing the API access and handling authentication through headers exchange. Finally on the last chapter (Chapter 11) we'll add some optimization techniques for improving the server responses.

The final application will scratch the surface of being a market place where users will be able to place orders, upload products and more. There are plenty of options out there to set up an online store, such as Shopify[4], Spree[5] or Magento[6].

By the end or during the process(it really depends on your expertise), you will get better and be able to better understand some of the bests Rails resources out there. I also took some of the practices from this guys and brought them to you:

- Railscasts[7]
- CodeSchool[8]
- Json api[9]

Conventions on this book

The conventions on this book are based on the ones from Ruby on Rails Tutorial[10]. In this section I'll mention some that may not be so clear.

I'll be using many examples using command-line commands. I won't deal with windows `cmd` (sorry guys), so I'll based all the examples using Unix-style command line prompt, as follows:

```
1 $ echo "A command-line command"
2 A command-line command
```

I'll be using some guidelines related to the language, what I mean by this is:

- "Avoid" means you are not supposed to do it
- "Prefer" indicates that from the 2 options, the first it's a better fit
- "Use" means you are good to use the resource

If for any reason you encounter some errors when running a command, rather than trying to explain every possible outcome, I'll will recommend you to 'google it', which I don't consider a bad practice or whatsoever. But if you feel like want to grab a beer or have troubles with the tutorial you can always shout me tweet[11] or email me[12]. I'm always willing to know you guys!

[4] http://shopify.com
[5] http://spreecommerce.com/
[6] http://magento.com
[7] http://railscasts.com
[8] http://codeschool.com
[9] http://jsonapi.org/format/
[10] http://www.railstutorial.org/book/beginning#sec-conventions
[11] http://twitter.com/kurenn
[12] mailto:kurenn@icalialabs.com

Getting started

One of the most painful parts for almost every developer is setting everything up, but as long as you get it done, the next steps should be a piece of cake and well rewarded. So as an attempt to make this easier and keep you motivated we will be using a bash script I manage put together called Kaishi[13], it includes all the necessary tools (Box~\ref{aside:kaishi_tools}) and more to setup your development environment, it currently only works for Mac OS:

\begin{aside} \label{aside:kaishi_tools} \heading{Kaishi development tools}

- oh-my-zsh[14] as your default shell
- Homebrew[15] for managing packages
- Git[16] for version controlling
- Postgresql[17] as the database manager
- Vim[18] for text editing
- ImageMagick[19] for images processing
- Rbenv[20] for managing the ruby environment
- Bundler[21] gem
- Foreman[22] for running apps
- Rails[23] gem for creating any rails app
- Heroku[24] toolbelt to interact with the Heroku API
- RailsAppCustomGenerator[25] for initializing any Rails app with Icalia's flavor
- Pow[26] to run local apps locally like a superhero

\end{aside}

[13] http://icalialabs.github.io/kaishi/
[14] https://github.com/robbyrussell/oh-my-zsh
[15] http://brew.sh/
[16] http://git-scm.com/
[17] http://www.postgresql.org/
[18] http://www.vim.org/
[19] http://www.imagemagick.org/
[20] https://github.com/sstephenson/rbenv
[21] http://bundler.io/
[22] https://github.com/ddollar/foreman
[23] http://rubyonrails.org/
[24] https://toolbelt.heroku.com/
[25] https://github.com/IcaliaLabs/railsAppCustomGenerator
[26] http://pow.cx/

Development environments

Text editors and Terminal

There are many cases in which development environments may differ from computer to computer. That is not the case with text editors or IDE's. I think for Rails development an IDE is way to much, but some other might find that the best way to go, so if that it's your case I recommend you go with RadRails[27] or RubyMine[28], both are well supported and comes with many integrations out of the box.

Now for those who are more like me, I can tell you that there are a lot of options out there which you can customize via plugins and more.

- **Text editor**: I personally use vim[29] as my default editor with janus[30] which will add and handle many of the plugins you are probably going to use. In case you are not a *vim* fan like me, there are a lot of other solutions such as Sublime Text[31] which is a cross-platform easy to learn and customize (this is probably your best option), it is highly inspired by TextMate[32] (only available for Mac OS). A third option is to use a more recent text editor from the guys at Github[33] called Atom[34], it's a promising text editor made with Javascript, it is easy to extend and customize to meet your needs, give it a try. Any of the editors I present will do the job, so I'll let you decide which one fits your eye.
- **Terminal**: If you decided to go with kaishi[35] for setting the environment you will notice that it sets the default shell to `zsh`, which I highly recommend. For the terminal, I'm not a fan of the *Terminal* app that comes out of the box if you are on Mac OS, so check out iTerm2[36], which is a terminal replacement for Mac OS. If you are on Linux you probable have a nice terminal already, but the default should work just fine.

Browsers

When it comes to browsers I would say Chrome[37] immediately, but some other developers may say Firefox[38] or even Safari[39]. Any of those will help you build the application you want, they come with nice inspector not just for the dom but for network analysis and many other features you might know already.

[27] http://www.aptana.com/products/radrails
[28] http://www.jetbrains.com/ruby/index.html
[29] http://www.vim.org/
[30] https://github.com/carlhuda/janus
[31] http://www.sublimetext.com/
[32] http://macromates.com/
[33] http://gitub.com
[34] https://atom.io/
[35] http://icalialabs.github.io/kaishi/
[36] http://www.iterm2.com/#/section/home
[37] https://www.google.com/intl/en/chrome/browser/
[38] http://www.mozilla.org/en-US/firefox/new/
[39] https://www.apple.com/safari/

A note on tools

All right, I understand that you may not want to include every single package that comes with kaishi[40], and that is fair, or maybe you already have some tools installed, well I'll describe you how to install the bare bones you need to get started:

Package manager

- **Mac OS**: There are many options to manage how you install packages on your Mac, such as Mac Ports[41] or Homebrew[42], both are good options but I would choose the last one, I've encountered less troubles when installing software and managing it. To install `brew` just run the command below: `console $ ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/inst`
- **Linux**: You are all set!, it really does not matter if you are using `apt`, `pacman`, `yum` as long you feel comfortable with it and know how to install packages so you can keep moving forward.

Git

We will be using Git a lot, and you should use it too not just for the purpose of this tutorial but for every single project.

- **Mac OS**: `console $ brew install git`
- **Linux**: `console $ sudo apt-get install git`

Ruby

There are many ways in which you can install and manage `ruby`, and by now you should probably have some version installed (1.8) if you are on Mac OS, to see which version you have, just type:

```
1 $ ruby -v
```

Rails 4 requires you to install version 1.9 or higher, and in order to accomplish this I recommend you to start using Ruby Version Manager (RVM)[43] or rbenv[44], any of these will allow you to install multiple versions of `ruby`. I recently changed from RVM to rbenv and it's great, so any of these two options you choose is fine. On this tutorial we'll be using `rbenv`.

A note for Mac OS: if you are using Mac just keep in mind you have to have installed the Command Line Tools for Xcode[45].

Mac OS:

To get started with the ruby installation, type in:

[40] http://icalialabs.github.io/kaishi/
[41] https://www.macports.org/
[42] http://brew.sh/
[43] http://rvm.io/
[44] http://rbenv.org/
[45] https://developer.apple.com/downloads/

```
$ rbenv install 2.1.2
```

Next you have to set up the just installed version of ruby as the default one:

```
$ rbenv global 2.1.2
$ rbenv rehash
```

The rehash command is supposed to run everytime you install a new ruby version or a gem. Seems like a lot? check out rbenv-gem-rehash[46] brew formula to mitigate this.

For more information about customization or other types of installation checkout out the project documentation[47].

Linux:

The first steo is to setup some dependencies for Ruby:

```
$ sudo apt-get update
$ sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev \
                            libreadline-dev libyaml-dev libsqlite3-dev sqlite3 \
                            libxml2-dev libxslt1-dev libcurl4-openssl-dev \
                            python-software-properties
```

Next it is time to install ruby:

```
$ cd
$ git clone git://github.com/sstephenson/rbenv.git .rbenv
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.profile
$ echo 'eval "$(rbenv init -)"' >> ~/.profile
$ exec $SHELL

$ git clone git://github.com/sstephenson/ruby-build.git ~/.rbenv/plugins/ruby-bu\
ild
$ echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.profile
$ exec $SHELL

$ rbenv install 2.1.2
$ rbenv global 2.1.2
```

If everything went smooth, it is time to install the rest of the dependencies we will be using.

Gems, Rails & Missing libraries

First we update the gems on the whole system:

[46] https://github.com/sstephenson/rbenv-gem-rehash
[47] https://github.com/sstephenson/rbenv

```
$ gem update --system
```

On some cases if you are on a Mac OS, you will need to install some extra libraries: `console $ brew install libtool libxslt libksba openssl`

We then install the necessary gems and ignore documentation for each gem:

```
$ printf 'gem: --no-document' >> ~/.gemrc
$ gem install bundler
$ gem install foreman
$ gem install rails -v 4.0
```

Check for everything to be running nice and smooth:

```
$ rails -v
Rails 4.0.5
```

Databases

I highly recommend you install Postgresql[48] to manage your databases, but for simplicity we'll be using SQlite[49]. If you are using Mac OS you should be ready to go, in case you are on Linux, don't worry we have you covered:

```
$ sudo apt-get install libxslt-dev libxml2-dev libsqlite3-dev
```

or

```
$ sudo yum install libxslt-devel libxml2-devel libsqlite3-devel
```

Initializing the project

\label{sec:initializing_project}

Initializing a Rails application must be pretty straightforward for you, if that is not the case, here is a super quick tutorial (Listing~\ref{code:project_initialization}):

Heads up: Be aware that we'll be using Rspec[50] as the testing suite, so just make sure you include the *-T* option when creating the rails application.

[48] http://www.postgresql.org/
[49] http://www.sqlite.org/
[50] http://rspec.info/

\begin{codelisting} \codecaption{Initializing the project with rails new.} \label{code:project_initialization} `console $ mkdir ~/workspace $ cd workspace $ rails new market_place_api -T` \end{codelisting}

As you may guess, the commands above(Listing~\ref{code:project_initialization}) will generate the bare bones of your Rails application. The next step is to add some `gems` we'll be using to build the api.

Installing Pow or Prax

\label{sec:pow_prax}

You may ask yourself, why in the hell would I want to install this type of package?, and the answer is simple, we will be working with subdomains[51], and in this case using services like Pow[52] or Prax[53] help us achieve that very easily.

Installing Pow:

Pow only works on Mac OS, but don't worry there is an alternative which mimics the functionality on Linux. To install it just type in:

```
1 $ curl get.pow.cx | sh
```

And that's it you are all set. You just have to symlink the application in order to set up the Rack app.

First you go the `\textasciitilde/.pow` directory:

```
1 $ cd ~/.pow
```

Then you create the symlink[54]:

```
1 $ ln -s ~/workspace/market_place_api
```

Remember to change the user directory to the one matches yours. You can now access the application through http://market_place_api.dev/[55]. Your application should be up a running by now like the one shown on Figure~\ref{fig:pow_running}.

Installing Prax

For linux users only, I extracted the instructions from the official documentation, so for any further documentation you should refer to the README[56] file on the github repository.

It is recommended that you clone the repository under the `/opt` directory and then run the installer which will set the port forwarding script and NSSwitch extension.

[51] http://en.wikipedia.org/wiki/Subdomain
[52] http://pow.cx/
[53] https://github.com/ysbaddaden/prax
[54] http://en.wikipedia.org/wiki/Symbolic_link
[55] http://market_place_api.dev/
[56] https://github.com/ysbaddaden/prax/blob/master/README.rdoc

```
$ sudo git clone git://github.com/ysbaddaden/prax.git /opt/prax

$ cd /opt/prax/
$ ./bin/prax install
```

Then we just need to link the apps:

```
$ cd ~/workspace/market_place_api
$ prax link
```

If you want to start the prax server automatically, add this line to the `.profile` file:

```
prax start
```

You should see the application up and running, see Figure~\ref{fig:pow_running}.

http://market_place_api.dev/\label{fig:pow_running}

Gemfile and Bundler

Once the Rails application is created, the next step is adding a simple but very powerful gem to serialize the resources we are going to expose on the api. The gem is called `active_model_serializers` which is an excellent choice to go when building this type of application, is well maintained and the documentation[57] is amazing.

So your `Gemfile` should look like this (Listing~\ref{code:gemfile_serializer}) after adding the `active_model_serializers` gem:

\begin{codelisting} \codecaption{The default Gemfile with the serializers gem.} \label{code:gemfile_serializer} "'ruby source 'https://rubygems.org'

[57] https://github.com/rails-api/active_model_serializers

Bundle edge Rails instead: gem ‘rails’, github: ‘rails/rails’

gem ‘rails’, ‘4.0.2’

Use sqlite3 as the database for Active Record

gem ‘sqlite3’

Use SCSS for stylesheets

gem ‘sass-rails’, ‘~> 4.0.0’

Use Uglifier as compressor for JavaScript assets

gem ‘uglifier’, ‘>= 1.3.0’

Use CoffeeScript for .js.coffee assets and views

gem ‘coffee-rails’, ‘~> 4.0.0’

See https://github.com/sstephenson/execjs#rea for more supported runtimes

gem 'therubyracer', platforms: :ruby

Use jquery as the JavaScript library

gem ‘jquery-rails’

Api gems

gem ‘active_model_serializers’

group :doc do # bundle exec rake doc:rails generates the API under doc/api. gem ‘sdoc’, require: false end “‘ \end{codelisting}

Notice that I remove the `jbuilder` and `turbolinks` gems, as we are not really going to use them anyway.

It is a good practice also to include the ruby version used on the whole project, this prevents dependencies to break if the code is shared among different developers, wheter if is a private or public project.

It is also important that you update the `Gemfile` to group the different gems into the correct environment (Listing∼\ref{code:group_gemfile}):

\begin{codelisting} \codecaption{The updated Gemfile for different groups.} \label{code:group_gemfile} `ruby ... group :development do gem 'sqlite' end ...` \end{codelisting}

This as you may recall will prevent `sqlite` from being installed or required when you deploy your application to a server provider like Heroku[58].

Note about deployment: Due to the structure of the application we are not going to deploy the app to any server, but we will be using Pow[59] by Basecamp[60]. If you are using Linux there is a similar solution called Prax[61] by ysbaddaden. See Section∼\ref{sec:pow_prax}

> Pow is a zero-config Rack server for Mac OS X. Have it serving your apps locally in under a minute. - Basecamp

Once you have this configuration set up, it is time to run the `bundle install` command to integrate the corresponding dependencies:

[58] http://heroku.com/

[59] http://pow.cx/

[60] https://basecamp.com/

[61] https://github.com/ysbaddaden/prax

```
$ bundle install
Fetching source index for https://rubygems.org/
.
.
.
```

After the command finish its execution, it is time to start tracking the project with git (Section∼\ref{sec:version_-control})

Version Control

\label{sec:version_control}

Remember that Git[62] helps you track and maintain history of your code. Keep in mind source code of the application is published on Github[63]. You can follow the repository at https://github.com/kurenn/market_-place_api[64].

By this point I'll asume you have git already configured and ready to use to start tracking the project. If that is not your case, follow these first-time setup steps:

```
$ git config --global user.name "Type in your name"
$ git config --global user.email "Type in your email"
$ git config --global core.editor "mvim -f"
```

Replace the last command editor(`"mvim -f"`) with the one you installed `"subl -w"` for SublimeText ,`"mate -w"` for TextMate, or `"gvim -f"` for gVim.

So it is now time to **init** the project with git. Remember to navigate to the root directory of the `market_place_api` application:

```
$ git init
Initialized empty Git repository in ~/workspace/market_place_api/.git/
```

The next step is to ignore some files that we don't want to track, so your `.gitignore` file should look like the one shown below (Listing∼\ref{code:gitignore}):

\begin{codelisting} \codecaption{The modified version of the .gitignore file} \label{code:gitignore} "'text /.bundle

[62] http://git-scm.com/
[63] https://github.com/
[64] https://github.com/kurenn/market_place_api

Ignore the default SQLite database.

/db/*.sqlite3* */db/*.sqlite3-journal

Ignore all logfiles and tempfiles.

/log/*.log /tmp

Extra files to ignore

doc/ *.swp *~ .DS_Store “‘ \end{codelisting}

After modifiying the `.gitignore` file we just need to add the files and commit the changes, the commands necessary are shown below:

```
$ git add .
$ git commit -m "Initializes the project"
```

Good practice: I have encounter that commiting with a message starting with a present tense verb, describes what the commit does and not what it did, this way when you are exploring the history of the project it is more natural to read and understand(or at least for me). I'll follow this practice until the end of the tutorial.

Lastly and as an optional step we setup the github(I'm not going through that in here) project and push our code to the remote server:

We first add the remote:

```
$ git remote add origin git@github.com:kurenn/market_place_api.git
```

then:

```
$ git push -u origin master
Counting objects: 58, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (47/47), done.
Writing objects: 100% (58/58), 13.84 KiB | 0 bytes/s, done.
Total 58 (delta 2), reused 0 (delta 0)
To git@github.com:kurenn/market_place_api.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

As we move forward with the tutorial, I'll be using the practices I follow on my daily basis, this includes working with `branches`, `rebasing`, `squash` and some more. For now you don't have to worry if some of these don't sound familiar to you, I walk you through them in time.

Conclusion

It's been a long way through this chapter, if you reach here let me congratulate you and be sure that from this point things will get better. If you feel like want to share how are you doing with the tutorial, I'll be happy to read it, a nice example is shown below:

> I just finished the first chapter of Api on Rails tutorial by @kurenn![65]

So let's get our hands dirty and start typing some code!

[65] https://twitter.com/kurenn