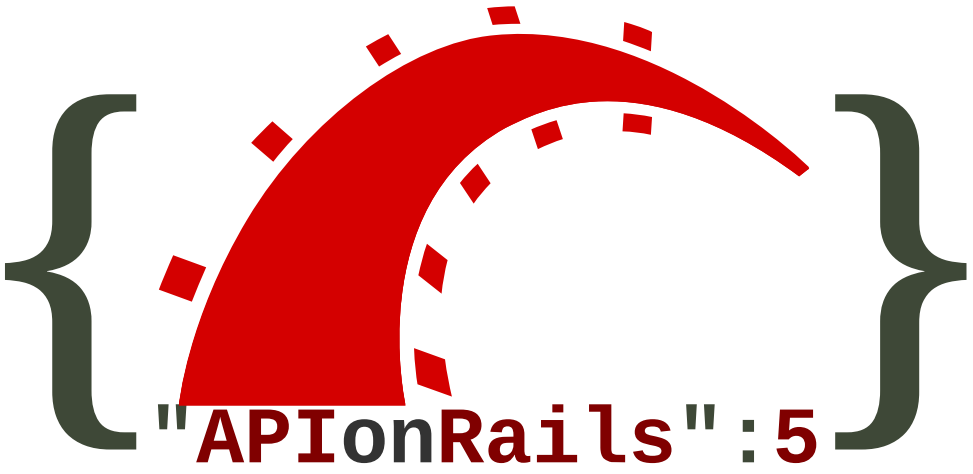
{"author": "Alexandre Rousseau"}



API on Rails 5

Alexandre Rousseau

Version 1.0, 2019-01-10

Table of Contents

Before

Foreword

"API on Rails 5" is based on "APIs on Rails: Building REST APIs with Rails". It was initially published in 2014 by Abraham Kuri under the licenses MIT and Beerware.

Since the original work was not maintained, I wanted to update this excellent work and contribute to the Francophone community by translating it myself. This update is also available in the Molière language [1: It means french.].

About the author

Abraham Kuri is a Rails developer with 5 years of experience (probably more now). His experience includes working as a freelancer in software product development and more recently in collaboration within the open source community. A graduate in computer science from ITESM, he founded two companies in Mexico (Icalia Labs and Codeando Mexico).

On my side, my name is Alexandre Rousseau and I am a Rails developer with more than 4 years of experience (at the time of writing). I am currently a partner in a company (iSignif) where I build and maintain a SAAS product using Rails. I also contribute to the Ruby community by producing and maintain some gems that you can consult onhttps://rubygems.org/profiles/madeindjs[my Rubygems.org profile]. Most of my projects are on Github so don't hesitate to follow me.

All the source code of this book is available in Asciidoctor format on Github. So don't hesitate to forke the project if you want to improve it or fix a mistake that I didn't notice.

Copyright and license

This book is provided on MIT license. All the book's source code is available on Markdown format on Github

Introduction

Welcome to APIs on Rails a tutorial on steroids on how to build 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/. The most up-to-date version of the *API's on Rails* can be found on APIs on Rails; don't forget to update your offline version if that is the case.

The intention with this book it's not only to teach you how to build an API with Rails. The purpose is also to teach you how to build scalable and maintainable API with Rails which means improve your current Rails knowledge. 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 the project we will start tracking it with Git.

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 deciding whether to use JSON or XML as the response format. We'll get our hands dirty then 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 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, Spree or Magento.

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 these guys and brought them to you:

- Railscasts
- CodeSchool
- JSON API

Conventions on this book

The conventions on this book are based on the ones from Ruby on Rails Tutorial. 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:

```
$ echo "A command-line command"
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 email me.

Development environments

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 I will guide you to keep you motivated.

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 or RubyMine, both are well supported and comes with many integrations out of the box.

- **Text editor**: I personally use vim as my default editor with janus 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 which is a cross-platform easy to learn and customize (this is probably your best option), it is highly inspired by TextMate (only available for Mac OS). A third option is to use a more recent text editor from the guys at Github called Atom, 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 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, 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 Firefox immediately, but some other developers may say Chrome or even Safari. 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.

Package manager

- **Mac OS**: There are many options to manage how you install packages on your Mac, such as Mac Ports or Homebrew, 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:

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

- **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.

- sous Mac OS: `$ brew install git`
- sous Linux: `$ 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 if you are on Mac OS, to see which version you have, just type:

```
$ ruby -v
```

Rails 5 requires you to install version 2.2.2 or higher. I recommend you to start using Ruby Version Manager (RVM) or rbenv, any of these will allow you to install multiple versions of `ruby`. We will use RVM in this tutorial any of these two options you choose is fine.

To install RVM go on https://rvm.io/ and get GPG key [2: The GPG allow you to verify author identity of the software you download.]. Then:

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB
$ \curl -sSL https://get.rvm.io | bash
```

Next it is time to install ruby:

```
$ rvm install 2.5
```

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:

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

On some cases if you are on a Mac OS, you will need to install some extra libraries:

```
$ 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 5.2
```

Check for everything to be running nice and smooth:

```
$ rails -v 5.2
5.2.0
```

Database

I highly recommend you install Postgresql to manage your databases, but for simplicity we'll be using SQlite. 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

Initializing a Rails application must be pretty straightforward for you, if that is not the case, here is a super quick tutorial.

Be aware that we'll be using Rspec as the testing suite. So we will use the `--skip-test` option. Also we will use `--api` option.

NOTE	This option came with Rails 5 and it allow to limit gems and Middleware. It will also avoid to generate HTML views when using Rails generators.

There is the command:

```
$ mkdir ~/workspace
$ cd ~/workspace
$ rails new market_place_api --skip-test --api
```

As you may guess, the commands above 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

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, and in this case using services like Pow or Prax help us achieve that very easily

Installing Pow

NOTE	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:

```
$ 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 `~/.pow` directory:

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

Then you create the symlink:

```
$ 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/. Your application should be up a running by now.

Installing Prax

For linux users only, Prax distribute some Debian/Ubuntu precompiled packages. You only have to download `.deb` and instal with `dpkg`.

```
$ cd /tmp
$ wget https://github.com/ysbaddaden/prax.cr/releases/download/v0.8.0/prax_0.8.0-
1_amd64.deb
$ sudo dpkg -i prax_0.8.0-1_amd64.deb
```

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
```

NOTE	When using prax, you have to specify the port for the URL, in this case http://market_place_api.dev:3000

You should see the application up and running, see image bellow:

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. It is well maintained and the documentation is amazing.

So your `Gemfile` should look like this after adding the `active _model_serializers` gem:

Gemfile

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '2.5.3'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '~> 5.2.0'
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use Puma as the app server
gem 'puma', '~> 3.11'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Api gems
gem 'active_model_serializers'
# ...
```

NOTE	I remove the `jbuilder` and `turbolinks` gems because 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, whether 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

Gemfile

```
# ...
group :development do
  gem 'sqlite3'
end
# ...
```

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

NOTE	Due to the structure of the application we are not going to deploy the app to any server, but we will be using Pow by Basecamp. If you are using Linux there is a similar solution called Prax by ysbaddaden

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:

```
$ bundle install
```

After the command finish its execution, it is time to start tracking the project with Git.

Versioning

Remember that Git helps you track and maintain history of your code. Keep in mind source code of the application is published on Github. You can follow the repository at Github. I'll assume 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 "vim"
```

NOTE	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:

.gitignore

```
# Ignore bundler config.
/.bundle

# Ignore the default SQLite database.
/db/*.sqlite3
/db/*.sqlite3-journal

# Ignore all logfiles and tempfiles.
/log/*
/tmp/*
!/log/.keep
!/tmp/.keep

# Ignore uploaded files in development
/storage/*

/node_modules
/yarn-error.log

/public/assets
.byebug_history

# Ignore master key for decrypting credentials and more.
/config/master.key
```

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 "Initial commit"
```

TIP 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:madeindjs/market_place_api.git
```

Then:

```
$ git push -u origin master
```

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