CHAPTER 1

# Introduction to Rails

# What is ruby on Rails?

Ruby on Rails is a web application development framework written in the Ruby language it is more properly just called Rails.

It is designed to make programming web applications easier by making assumptions about what every developer needs to get started.

It allows you to write less code while accomplishing more than many other languages and frameworks.

Experienced Rails developers also report that it makes web application development more fun.

Rails is 'opinionated' software, it assumes that there is the "best" way to do things, and it's designed to encourage that way - and in some cases to discourage alternatives.

If you learn "The Rails Way" you'll probably discover a tremendous increase in productivity.

The Rails philosophy includes two major guiding principles:

DRY - "Don't Repeat Yourself" - suggests that writing the same code repeatedly is a bad thing.

Convention Over Configuration - means that Rails makes assumptions about what you want to do and how you're going to do it, rather than requiring you to specify every little thing through endless configuration files.

Rails is built around the **model-view-controller** pattern.

It’s a simple concept: separate the data, logic, and display layers of your program.

This lets you split functionality cleanly, just like having separate HTML, CSS and JavaScript files.

Here’s the **MVC** breakdown:

**Models:**

Are classes that talk to the database.

You find, create and save with models, so you don’t generally have to write any native **SQL**.

Rails has a class to handle the saving of the data to a database when a model is updated.

**Controllers:**

Take user inputs, like a URL for example, and decide what to do with it, show a page, order an item, post a comment etc.

They may initially have business logic.

Ideally controllers just take inputs, call model methods, and pass outputs to the view including any error messages.

**Views:**

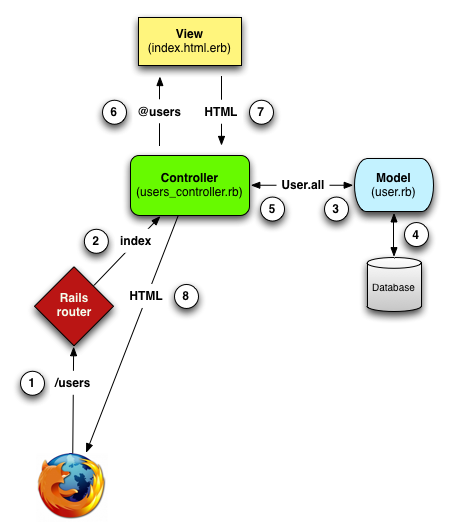
Display the output, usually HTML.

Rails uses **ERB** - "Embedded RuBy" – An .html.erb or .erb.html file is an HTML file with Ruby code embedded in.

Rails parses the Ruby code to add content to the file dynamically, and will output a "pure" HTML file for rendering

Rails also makes it easy to create other views such as XML (for web services/RSS feeds) or JSON (for AJAX calls).

The **MVC** method is the key to building a readable, maintainable and easily updateable web application.



## A detailed diagram of Rails MVC.

1. The browser issues a request for the /users URL.
2. Rails routes /users to the index action in the Users controller.
3. The index action asks the User model to retrieve all users.
4. The User model pulls all the users from the database.
5. The User model returns the list of users to the controller.
6. The controller captures the users in the @users variable, which is passed to the index view.
7. The view uses embedded Ruby to render the page as HTML.
8. The controller passes the HTML back to the browser.

We will look at the individual parts in more detail in the topics that follow.

# Installing Rails

The standard method on \*NIX type platforms are to install Ruby and RubyGems, usually supplied by the vendor as part of their distribution kit, and then from a command prompt run the following to install the rails gem files. The system may need access to the Internet to complete this step.

From rails version 6 several significant changes have been made:

Rails started to bundle and wrap Webpack inside Rails applications.

This is done through Webpacker.

What Webpacker provides is a preconfigured Webpack along with view helpers to easily get corresponding generated assets like JavaScript and [S]CSS files.

This leads to further dependencies within the rails application from a javascript perspective.

This is now dealt with by using Yarn and classically Nodejs

It is suggested that yarn and Nodejs are installed before we install rails.

Both download site has detailed instructions for the installation steps required.

Yarn for windows for example can be downloaded and installed from:

https://yarnpkg.com/lang/en/docs/install/#windows-stable

Nodejs can be downloaded and installed from:

https://nodejs.org/en/download/

Rails is a collection of predefined gems which can be downloaded and installed as follows:

gem install rails

Going forward there are other gems that will be required to be downloaded and, in some instances, compiled and linked.

If you’re working on a Windows platform you can get a complete installation/development kit, including Ruby itself but excluding Yarn and Nodejs, from:

**https://rubyinstaller.org/**

To verify that you have everything installed correctly, you should be able to run the following command at a terminal prompt.

rails --version

It should return a response like:

Rails 6.0.1

## Testing Rails installation

When you generate a new rails application it will automatically create the framework structure for you.

We need to have a ‘base’ directory under which we can build our rails projects.

Two simple examples are:

Widows:

C:\sites

Linux/Unix:

/sites

The examples shown throughout the manual are based on using a base directory of \sites.

You create a new application using the **rails new** command.

The rails new command has many options to include/exclude functionality from the framework, to view all the options for the new command run:

rails new –-help

As this is a training environment, we are going to exclude some elements from the framework.

The rails system support a .railsrc file approach to save repetitive typing of the excluded items.

In your home directory:

under Unix this will be something like /home/user1 for example.

Or on a windows platform it will be something like c:\users\user1

Create a file using a text editor and add the following line

--skip-active-storage --skip-javascript --skip-git --skip-action-cable

Save the file as .railsrc (note the leading dot and no extension type)

Assuming you are in /sites we then execute:

rails new course

This will create a sub-directory called course and the rails application framework below it.

|  |  |
| --- | --- |
| File/Folder | Purpose |
| app/ | Contains the controllers, models, views and assets for your application. You’ll focus on this folder for the remainder of this guide |
| config/ | Configure your application’s runtime rules, routes, database, and more. This is covered in more detail in Configuring Rails Applications |
| config.ru | Rack configuration for Rack based servers used to start the application |
| db/ | Contains your current database schema, as well as the database migrations |
| doc/ | In-depth documentation for your application |
| Gemfile  Gemfile.lock | These files allow you to specify what gem dependencies are needed for your Rails application |
| lib/ | Extended modules for your application |
| Log/ | Application log files |
| public/ | The only folder seen to the world as-is. Contains the static files and compiled assets |
| Rakefile | This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application |
| README.rdoc | This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on |
| script/ | Contains the rails script that starts your app and can contain other scripts you use to deploy or run your application. |
| test/ | Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications |
| tmp/ | Temporary files |
| vendor/ | A place for all third-party code. In a typical Rails application, this includes Ruby Gems, the Rails source code (if you optionally install it into your project) and plugins containing additional prepackaged functionality |

It will also execute the bundler gem to install any gems that are required. These are listed in the Gemfile in the root of the application.

They may be some minor issues with installing gems such as sqlite3.

On windows for example you may have to manually run:

gem install sqlite3 --platform=ruby

Note any other gem installation issues and resolve them.

Then run:

bundle install

This should complete the installation of all required gems.

Rails has a built-in web server, puma, to allow for deployment testing without the overhead of having to install a full functioning HTTP server.

There are some configuration options to consider here even thought this is only our development environment.

Within the config subdirectory of the application is a file – **puma.rb** – that controls where puma will ‘listen’ for incoming connection requests, by default this is set to port 3000 and only to accept connections to localhost.

To give our application a ‘real feel’ we are going to change the port and address that puma will use.

Locate the default port entry line and comment it out.

# Specifies the `port` that Puma will listen on to receive requests; default is 3000.

#

port ENV.fetch("PORT") { 3000 }

Then add:

bind 'tcp://0.0.0.0:80'

Open a standard terminal session and navigate to the application directory.

The server is started using the **rails server** command.

This command can be abbreviated to **rails s.**

If the web server starts successfully you should be able to connect to puma using a standard web browser to the following url: **http://localhost** and should get the Rails welcome screen in response.

A picture containing text

Description automatically generated

The web server can be terminated by using ctrl-C and answering yes to the terminate question.

# Workshop 1

1. Check the version of Ruby installed
2. Install all necessary gems to support rails
3. Create a base working directory
4. Create a simple framework – this will be used in the next topics
5. Configure puma to listen on all addresses and port 80
6. Start the rails server and test the connectivity with a local browser
7. Stop the rails server