[Overview](#h.6a3cyw5l2xux)

[Lesson 3.1: Rails First Look](#h.cul5ck19p0dh)

[Gem of the Day](#h.3sc3mv22mjfg)

[What is Rails?](#h.r36gr1qq750a)

[MVC Framework](#h.11kj9u8d59da)

[Creating a New Rails App](#h.hza5qyda02t)

[Folder Tree of a Rails App](#h.fybmpel4utl0)

[Rails Server](#h.mvo2aq42eyyp)

[Browsing Our Site](#h.h37qu9ylego9)

[Rails Generate Controller](#h.emxg3rp5rf6a)

[Routes and Root](#h.qedtuv85u5l6)

[Updating a View](#h.gu2x04l1attp)

[Review, Working with Controllers, and ERB: Basic Greeting App](#h.1livvfq728y8)

[ERB](#h.di6j83i9t8gm)

[Working with the Controller & Instance Variables](#h.tas6s7r6zs5k)

[Linking Pages Together](#h.svk9l4amm9kx)

# Overview

* Lesson 3.1
  + What is Rails?
  + MVC Framework
  + Creating a New Rails App
  + Folder Tree of a Rails App
  + Working with Controllers: Basic Greeting App

# Lesson 3.1: Rails First Look

Emphasize that Rails will require a lot of practice and repetition over several weeks. Do homework, try to understand concepts, seek help when you don’t understand.

## Gem of the Day

We won’t be incorporating new gems on the first day, but I wanted to go ahead and start our Gem of the Day tradition. Our first gem is a really helpful one called Stamp. Let’s take a look at the documentation to see what it does... <https://github.com/jeremyw/stamp>

## What is Rails?

Rails is a **platform** for creating dynamic websites, using the **Ruby** programming language.

Rails is a web application development framework 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.

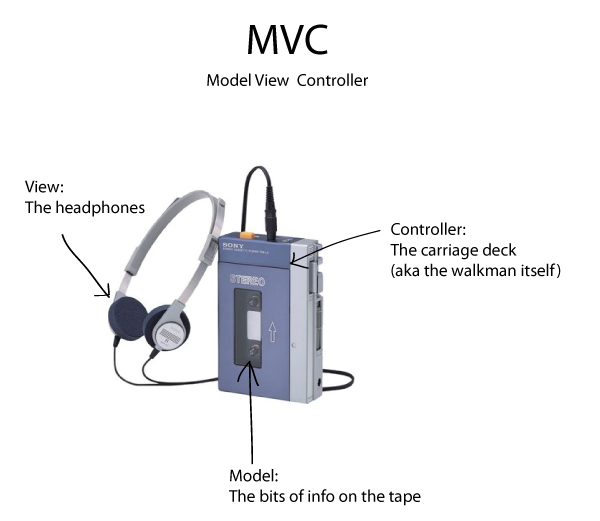
Rails is **opinionated** software. It makes the assumption that there is the "**best**" way to do things, and it's designed to encourage that way - and in some cases to discourage alternatives.

The Rails philosophy includes two major **guiding principles**:

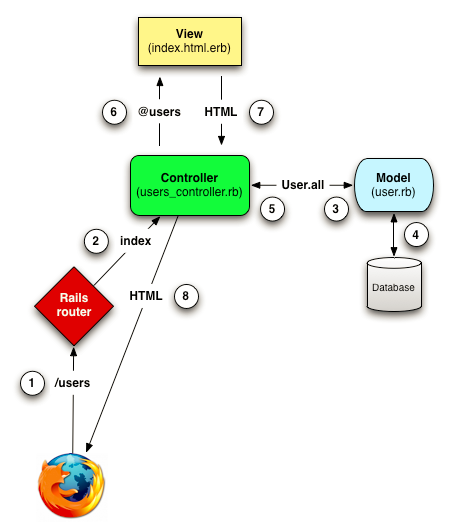
* **Don't Repeat Yourself**: DRY is a principle of software development which states that "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." By not writing the same information over and over again, our code is **more maintainable, more extensible, and less buggy**.
* **Convention Over Configuration**: Rails has opinions about the best way to do many things in a web application, and defaults to this set of conventions, rather than require that you specify every minutiae through endless configuration files.

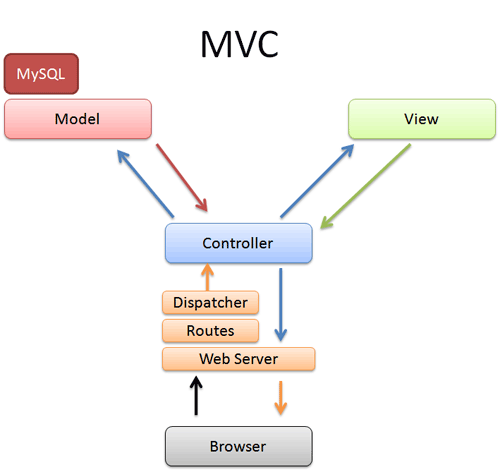
### MVC Framework

Draw Walkman analogy:



Draw a combined version of these two graphics on the board:





## Creating a New Rails App

Make a new folder called rails-practice. We will put projects that we probably won’t deploy separately in here. Cd into that folder then run:

rails new my\_awesome\_project

Take notes on the steps we go through today!

Rules:

* Relevant name
* Starts with a letter
* Lower case
* Underscores

Rails does a lot of “magic” behind the scenes to get our applications up and running. It generates all of the necessary directories (folders) for us to write our code but it will be up to us to populate these directories with the code that will make them run.

## Folder Tree of a Rails App

Have everyone open their my\_awesome\_app using “subl .”. Open my tts-resources app instead. Talk about Gemfile first:

**Gemfile** - A gem is a Ruby program or library in a standard format available through RubyGems.org. Each gem has a name, version, and platform. For example, the rails gem has a 4.2.0 version. Your Gemfile is a list of all gems that you want to include in the project. It is used with bundler (also a gem) to install, update, remove and otherwise manage your gems.

Find here: <http://siakaramalegos.github.io/blog/rails/2015/05/12/rails-folder-tree.html>. Generated from:

* <http://guides.rubyonrails.org/getting_started.html#creating-the-blog-application>
* <http://www.tutorialspoint.com/ruby-on-rails/rails-directory-structure.htm>
* <http://www.pragtob.info/rails-beginner-cheatsheet/>
* TTS content

|  |  |
| --- | --- |
| Folder or File | Purpose |
| app/ | Organizes your application components. It's got subdirectories that hold the view (views, assets, helpers), controller (controllers), the backend business logic (models), and more. |
| app/assets | Basically your front-end stuff. This folder contains images, javascripts, and stylesheets. These use the Rails asset pipeline which means that Rails is going to generate all this for us on deployment. |
| app/controllers | The controllers subdirectory contains the controllers, which handle the requests from the users. It is often responsible for a single resource type, such as places, users or attendees. Controllers also tie together the models and the views. |
| app/helpers | For the most part, helpers are going to be used to generate code for our views, though they can also be used to assist the models and controllers. Rails encourages “creating custom helpers to extract complicated logic or reusable functionality.” This helps keep our code small, focused, and uncluttered. |
| app/mailers | Functionality to send emails goes here. |
| app/models | Holds the classes that model the business logic of our application and wrap the data stored in our application's database. In most frameworks, this part of the application can grow pretty messy, tedious, verbose, and error-prone. Rails makes it much more simple. |
| app/views | The views subdirectory contains the display templates that will be displayed to the user after a successful request. By default they are written in HTML with embedded ruby (.html.erb). The embedded ruby is used to insert data from the application. It is then converted to HTML and sent to the browser of the user. It has subdirectories for every resource of our application. These subdirectories contain the associated view files.  Files starting with an underscore (\_) are called partials. Those are parts of a view which are reused in other views. A common example is \_form.html.erb which contains the basic form for a given resource. It is used in the new and in the edit view since creating something and editing something looks pretty similar. |
| app/views/ layouts | Holds the template files for layouts to be used with views. This models the common header/footer method of wrapping views. |
| bin/ | Contains the rails scripts, or executables, that start your app and perform other functions to setup, deploy, test, or run your application.   * Can be directly read by your computer, but not by humans. (The files we write need to be compiled in order to be read by computers.) * Ex: bundle, rake, rails, gem |
| config/ | Configure your application's routes, database, and more., including your database configuration (in database.yml), your Rails environment structure (environment.rb), and routing of incoming web requests (routes.rb). You can also tailor the behavior of the three Rails environments for test, development, and deployment with files found in the environments directory. |
| db/ | Contains your current database schema, as well as the database migrations. Usually, your Rails application will have model objects that access relational database tables.  Migrations set up your database structure, including the attributes of your models. With migrations you can add new attributes to existing models or create new models.  It also initially contains seeds.rb which is used to populate the database with default data. Schema.rb shows the current state of your app’s database |
| Gemfile | A file that specifies a list of gems that are required to run your application. Rails itself is a gem you will find listed in the Gemfile. Ruby gems are self-contained packages of code, more generally called libraries, that add functionality or features to your application. |
| Gemfile.lock | This file specifies the exact versions of all gems you use. Because some gems depend on other gems, Ruby will install all you need automatically. The file also contains specific version numbers. It can be used to make sure that everyone within a team is working with the same versions of gems. The file is auto-generated. **Do not edit this file.** |
| lib/ | Extended modules for your application. You'll put libraries here, unless they explicitly belong elsewhere (such as vendor libraries). For example, this might be code used to get specific information from Facebook. |
| log/ | Application log files. See all the funny stuff that is written in the console where you started the Rails server? It is written to your development.log. Logs contain runtime information of your application. If an error happens, it will be recorded here. |
| public/ | The only folder seen by the world as-is. Like the public directory for a web server, this directory has web files that don't change, such as JavaScript files (public/javascripts), graphics (public/images), stylesheets (public/stylesheets), and HTML files (public).    You should store your assets in the app/assets/images directory locally. In production, Rails precompiles these files to public/assets by default. |
| 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. To make it visible on GitHub, change it to a markdown file type (.md). |
| test/ | Contains the tests for your application. With tests you make sure that your application actually does what you think it does. |
| tmp/ | Temporary files (like cache, pid, and session files). Caching involves temporarily storing recently used information, which can improve application performance. |
| vendor/ | A place for all third-party code. In a typical Rails application this includes vendored gems (such as security libraries). |

## Rails Server

In order to run our shiny new web application we have to start our local server using the terminal. Inside of our project folder, run **rails s** (short for server).

The output tells us a bunch of good stuff:



* The version of Rails (4.2.0)
* Where to browse our app locally (http://localhost:3000)
* How to shutdown the server (Ctrl-C)

### Browsing Our Site

If all went well, you will see Rails’ underwhelming generic welcome page. We will soon replace this with our own custom page. So far, this is the only page in our app. Let’s create some more.

## Rails Generate Controller

The rails generate command uses templates to create lots of things in Rails. By itself it generates a list of all available generators. Try it!

rails g

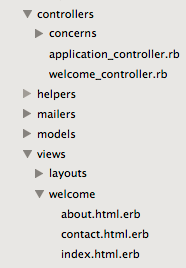
(look at the list)

The most common things we will generate are controllers, scaffolds, and migrations. Let’s start with just one today - controllers.

rails g controller Welcome index about contact

* g = generate
* Welcome = the name of the controller (but it you can call it whatever you like; capitalized, camelcase)
* index about contact = views (or, html pages) that you want associated with this controller (lowercase, with underscores)

Now, let’s take a look at our app folder to see what changed.



## Routes and Root

When you generate a Controller, Rails provides the routes.

# [railsprojectname]/config/routes.rb  
  
Rails.application.routes.draw do  
 get 'welcome/index'  
 get 'welcome/about'  
 get 'welcome/contact'  
end

The Root path is your landing (or, home) page. Call it by it's associated controller action!

Rails.application.routes.draw do  
 root 'welcome#index'  
 get 'welcome/about'  
 get 'welcome/contact'  
end

Customize Your Routes - No one wants a long URL!

Format:

'custom URL' => 'controller#action'

Rails.application.routes.draw do  
 get 'homepage' => 'welcome#index'  
 get 'thisisus' => 'welcome#about'  
 get 'drop\_us\_a\_line' => 'welcome#contact'  
 root 'welcome#index'  
endrail

To see a list of your routes, run the command **rake routes**. You can also purposely navigate to a bad page name, and the browser will give you a list of available routes. **Rake** actually means **Ruby Make**. Makes me think of Go-Go Gadget!

## Updating a View

In Sublime open up welcome/index.html.erb from the views folder. OMG it's HTML again! You remember this, right?

Go ahead and customize the <h1> and <p> tags, then reload your page (together). Now try updating the other two pages yourself!

## Review, Working with Controllers, and ERB: Basic Greeting App

Let’s dive in a little more on the controllers and make our app a bit more interactive.

Classroom challenge: Create a new app called Greeting and generate the controller Say with two pages, “hello” and “goodbye”.

Now, open the Say controller. The Say controller controls our logic for the two pages we created within it. When a user navigates to one of those pages, Rails knows to look in this controller to see what it should do. You will notice two things:

1. SayController is actually a class that inherits from the ApplicationController, so it automatically gets all the default ApplicationController behavior
2. Rails automatically created two empty methods for those two pages:

def hello

end

def goodbye

end

Now, run the Rails server and navigate to those pages. You can see that we connected the URL to our controller, and that Rails is pointing the way to our next step, namely to tell Rails what to display. The generate controller Say command also generated associated views for our pages.

On the hello view, change it to say “Hello from Rails!”.

### ERB

That’s pretty boring, so let’s make it dynamic. *Let’s have it show the current time each time it displays the page.*  How on earth would we show time as a string in the hello view? Where would we even get the time from?

Well, Rails gives us a cool tool with **ERB**, or embedded Ruby code. Whenever we use <%= %> or <% %> in an html.erb file, Rails will interpret it as Ruby code. The first one literally means “puts”, so it actually converts the content to a string, like when we used puts on the command line.

Here is a great intro to ERB and the different between the two syntaxes: <http://www.tutorialspoint.com/ruby/eruby.htm>

Let’s add the time to our view. Which syntax will we use?

<p>It is now <%= Time.now %>.</p>

Awesome! That is so ugly though! Now you understand the value of the Stamp gem even though we aren’t going to use it just yet. We’ve done a decent amount of work here, so let’s commit our changes.

### Working with the Controller & Instance Variables

So, asking the view to do all the work of determining the time is probably fine in such a trivial app, but really we should make the Controller do the work and let the view just show it. Remember, Rails emphasizes convention over configuration.

Open up the Say Controller. Inside the hello method, add:

@time = Time.now

Does anyone remember what the @ sign does to a variable? It turns it into an **instance variable**. They are **bound to an instance of a class** and together forms what we call the state of an object. Every instance of a class has a different set of instance variables. Additionally, instance variables are available to all methods of a class while local variables are not.

Now, go back to the view and change the erb to only call @time. Refresh your page to test it out.

What did we just learn?

* A place for everything, everything in its place. (Rails mantra) It may be trivial here, but maybe we want to build out this app more in the future using that variable. For example, maybe we want to customize it for people in different time zones. The bottom line is that data should be provided by the controller. This will become more obvious when we introduce models (databases) into the mix.
* How the controller and views work together
* How to use instance variables

### Linking Pages Together

First, let’s add some content to the goodbye view:

<h1>Goodbye!</h1>

<p>It was nice seeing you!</p>

Navigate to your goodbye page. That was fun, but it’s not so fun to manually navigate to each page. Sure, we might have a navbar in the future, but sometimes you want to go places directly from the content on the page. Can you think of an example? (signing out, registering, signing in, forgot password)

You might be saying to yourself, “Sia, that is so easy, why are we even talking about this?” Well, we could use a hard-coded html link. Let’s try that first - what would the correct link be for each page?

<a href=”/say/goodbye”>goodbye!</a>

<a href=”/say/hello”>hello!</a>

Yes, that would work, but it might not always as we edit our app and make it fancier. Luckily, Rails gives us an EVEN BETTER way to do it with **ERB** and **helper methods** with the **link\_to()** method:

Time to say <%= link\_to “goodbye!”, say\_goodbye\_path %>!

The first parameter is the text to display. The second parameter is a precomputed value that Rails make available to application views and it tells Rails to generate the link to the goodbye() action.

Classroom challenge: Create the hello link on the goodbye page using link\_to.

Admire your work then cancel your server. Commit your changes. What did we learn today?