

INTRODUCTION TO RAILS

Flatiron School - borrowed from Luisa Scavo

SETTING THE STAGE

In 2005, Danish developer David Heinemeier Hansson takes the stage at a web conference. At this time, it takes several **months** and a whole **team of developers** to create a simple blogging site.

He walks the audience through the process **alone**. In **fifteen minutes**.

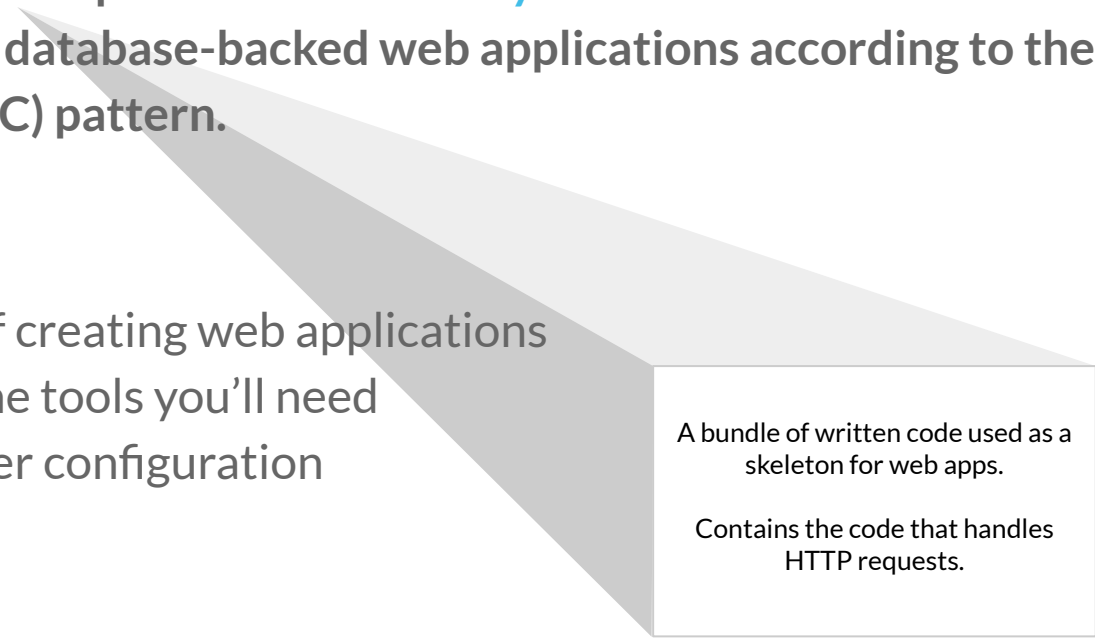
<https://www.youtube.com/watch?v=Gzj723LkRJY>

WHAT IS RAILS?

A **web-application framework** implemented as a **Ruby Gem** that includes everything needed to create database-backed web applications according to the **Model-View-Controller (MVC)** pattern.

It is designed to:

- Jumpstart the process of creating web applications
- Include all (or most of) the tools you'll need
- Prioritize convention over configuration



A bundle of written code used as a skeleton for web apps.

Contains the code that handles HTTP requests.

RAILS IS NOT A
PROGRAMMING
LANGUAGE.

WHAT IS MVC?

- An architectural pattern
- Makes use of three parts:
 - Model
 - View
 - Controller
- Allows us to thoroughly (and readably!) separate our concerns
 - Information internal to the app
 - Information presented to a user

EVERYTHING I NEED?

- **ActionPack:** Includes ActiveRecord/Model, Action Controller, and Action View
 - Handle code for the Models, Controllers, and Views
- **ActiveSupport:** Enhances the Ruby language and standard library
- **Action Mailer:** For sending emails
- **Action Cable:** for making live updates to pages over WebSockets
- **Railties:** Commands, generators, and glue to bring it all together

A LOT TO LEARN! BUT YOU CAN DO IT PIECE BY PIECE.

RAILS FILE STRUCTURE

app – contains the models, views, and controllers, along with the the rest of the core functionality of the application. This is the one directory where you can make a change and not have to restart the Rails server. The majority of your time will be spent working in this directory. In addition to the full MVC structure, this directory also contains non Ruby files, such as: css files, javascripts, images, fonts, etc.

bin – some built-in Rails tasks that you most likely will never have to work with.

config – manages a number of settings that control the default behavior, including: the environment settings, a set of modules that are initialized when the application starts, the ability to set language values, the application settings, the database settings, the application routes, and lastly the secret key base.

RAILS FILE STRUCTURE (CONTINUED)

db – within the db directory you will find the database schema file that lists the database tables, their columns, and each column's associated data type. The db directory also contains the seeds.rb file, which lets you create some data that can be utilized in the application. This is a great way to quickly integrate data in the application without having to manually add records through a web form element. The schema file can be found at db/schema.rb

lib – while many developers could build full applications without ever entering the lib directory, you will discover that it can be incredibly helpful. The lib/tasks directory is where custom rake tasks are created. You have already used a built-in rake task when you ran the database creation and migration tasks; however, creating custom rake tasks can be very helpful and sometimes necessary. For example, a custom rake task that runs in the background, making calls to an external API and syncing the returned data into the application's database.

RAILS FILE STRUCTURE (CONTINUED)

log – within the log directory you will discover the application logs. This can be helpful for debugging purposes, but for a production application it's often better to use an outside service since they can offer more advanced services like querying and alerts.

public – this directory contains some of the custom error pages, such as 404 errors, along with the robots.txt file which will let developers control how search engines index the application on the web.

test – by default, Rails will install the test suite in this directory. This is where all of your specs, factories, test helpers, and test configuration files can be found. **Side note:** We always use RSpec, which means this directory will actually be called spec.

RAILS FILE STRUCTURE (CONTINUED)

tmp – this is where the temporary items are stored and is rarely accessed by developers.

vendor – this directory has been utilized for varying purposes in the past. In Rails 4+, its main purpose is for integrating client-side MVC frameworks, such as AngularJS.

Gemfile – the Gemfile contains all of the gems that are included in the application; this is where you will place outside libraries that are utilized in the application. After any change to the Gemfile, you will need to run bundle. This will call in all of the code dependencies in the application. The Gem process can seem like a mystery to new developers, but it is important to realize that the Gems that are brought into an application are simply Ruby files that help extend the functionality of the app.

RAILS FILE STRUCTURE (CONTINUED)

Gemfile.lock – this file should not be edited. It displays all of the dependencies that each of the Gems contain along with their associated versions. Messing around with the lock file can lead to application bugs due to missing or altered Gem dependencies.

README.rdoc – the readme file is an important place to document the details of the application. If the application is an open-source project, this is where you can place instructions to other developers, such as how to get the app up and running locally.

[HTTP://GUIDES.RUBYONRAILS.ORG/](http://guides.rubyonrails.org/)

CREATING A NEW RAILS APP

RAILS NEW APP-NAME

SINATRA VS. RAILS: RAILS SERVER!

RAILS S

SINATRA VS. RAILS: RAILS CONSOLE!!

RAILS C

SINATRA VS. RAILS: ROUTING

In Sinatra, routes are defined in the controller.

In Rails, routes are defined in a separate file, `config/routes.rb`, and they reference controller methods.

Once can see all routes created by running `rake routes`.

<http://guides.rubyonrails.org/routing.html>

RAILS VS. SINATRA: GENERATORS!

Examples:

Generating a Migration => Migration only

Destroying a Migration

Generating a Model => Model and Migration

Generating a Resource => Controller, Routes, Model, Migration

SINATRA VS. RAILS: IMPLICIT RENDERING

In Sinatra, we have to tell a controller which views to render, and when.

In Rails, this is not always the case...

REFERENCES

- <https://facilethings.com/blog/en/convention-over-configuration>
- https://en.wikipedia.org/wiki/Ruby_on_Rails
- <http://guides.rubyonrails.org/>
- <https://api.rubyonrails.org/>
- https://guides.rubyonrails.org/command_line.html
- <https://dhh.dk/>
- <https://dev.to/alicannkic/rails-generator-cheatsheet-1dfn>