



RUBY ON RAILS

A quick introduction

Presented by Stefanos Laskaridis

CONTENTS

- What is Ruby on Rails
- MVC Paradigm
- The Model
- The Controllers
- The Views
- Configuration and Routing
- What we omitted

RUBY ON RAILS




- Created by David Heinemeier in 2003
- Heavily based on Ruby's metaprogramming features
- Open-source project, hosted on github.com
- Has 4,000+ contributors

APPLICATIONS WRITTEN IN RAILS

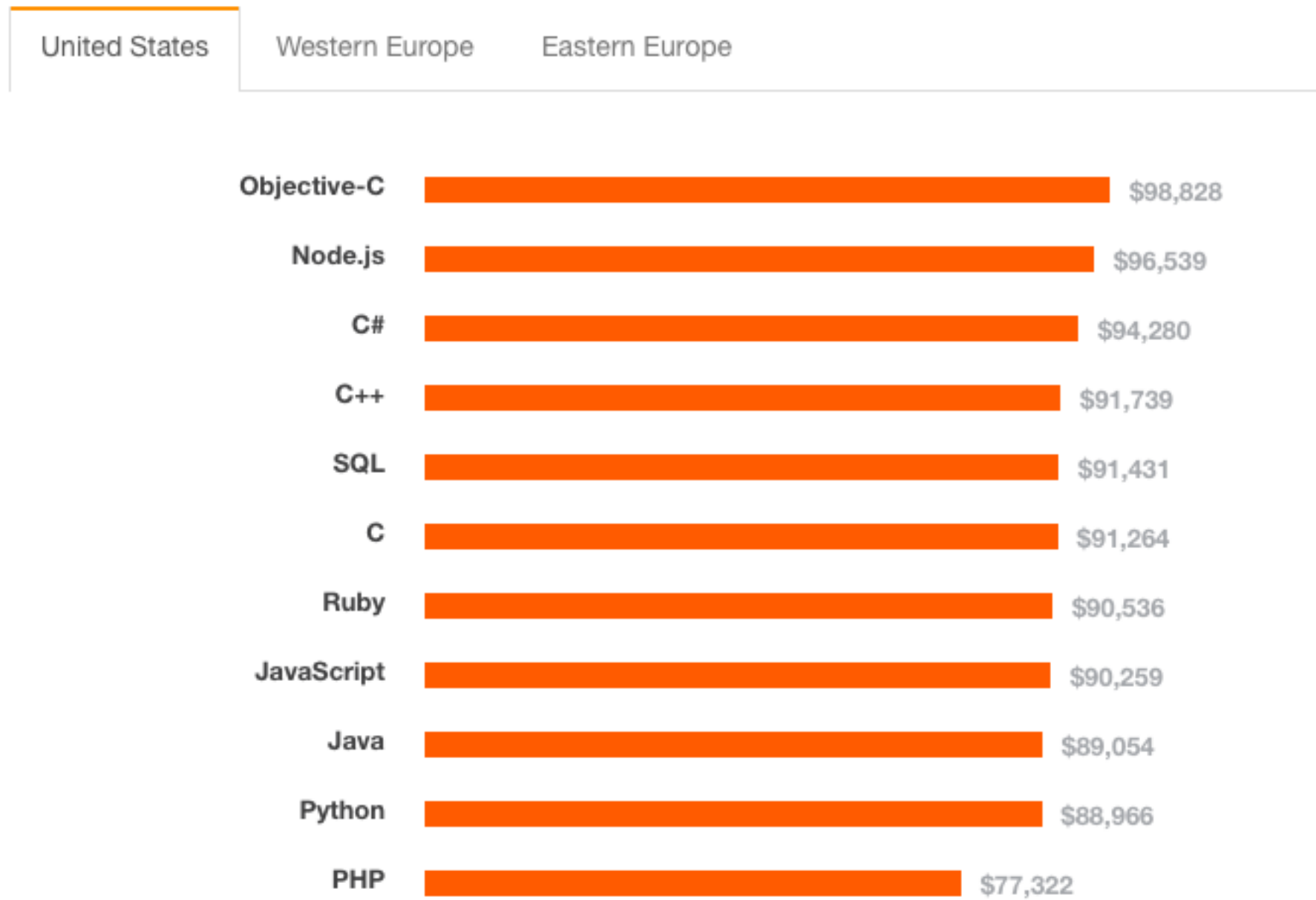


*Twitter has switch much of its code base to Scala for scalability performance.

WHY RAILS

- It's cool (many call it hipster framework) 
- It's easily deployed due to its “convention over configuration” philosophy
- It has a large developers' base (219,850 questions tagged in stackoverflow.com)
- It's highly paid (see next slide)

VI. COMPENSATION BY TECHNOLOGY



19,483 responses

Source: StackOverflow Survey

BEFORE WE START

REQUIREMENTS

- We will need
 - Ruby installed (natively or through a manager, like RVM)
 - bash shell (for terminal commands)

But I don't have a Unix-based
OS installed ...



- Well, no-one is perfect ...
- ACM strongly endorses GNU/Linux.



WITHOUT BASH

- We have a Virtual Image ready in a flash drive.
- Alternatively, you can try your luck with Ruby Installer for Windows and RailsInstaller.
- Or you can just download an IDE and follow from there.

IDE'S

RUBYMINE IDE



- My personal recommendation for development of Ruby/Ruby on Rails Applications.
- Part of the JetBrains Suite.
- Great functionality, with Rails version management, live debugger, SCM support, etc.
- It's also free through academic subscription.
All you have to do is create an account and activate it.

ALTERNATIVE IDE'S

- Aptana (based on Eclipse)
- Netbeans
- But many users prefer a plain text editor
 - Atom (powered by Github)
 - Textmate (OS X only)
 - Sublime Text

DIVING INTO RAILS

BOOTSTRAPPING RAILS

- Launching a new application is really really easy.
- Just do the following:

```
$ gem install rails  
$ rails new <application_name>
```

A ROR APPLICATION SKELETON

```
steve@steves-mbp app $ tree -L 2
.
├── Gemfile
├── Gemfile.lock
├── README.rdoc
├── Rakefile
├── app
│   ├── assets
│   ├── controllers
│   ├── helpers
│   ├── mailers
│   ├── models
│   └── views
├── bin
│   ├── bundle
│   ├── rails
│   ├── rake
│   ├── setup
│   └── spring
├── config
│   ├── application.rb
│   ├── boot.rb
│   ├── database.yml
│   ├── environment.rb
│   ├── environments
│   ├── initializers
│   ├── locales
│   ├── routes.rb
│   └── secrets.yml
├── config.ru
├── db
│   └── seeds.rb
├── lib
│   ├── assets
│   └── tasks
├── log
├── public
│   ├── 404.html
│   ├── 422.html
│   ├── 500.html
│   ├── favicon.ico
│   └── robots.txt
├── test
│   ├── controllers
│   ├── fixtures
│   ├── helpers
│   ├── integration
│   ├── mailers
│   ├── models
│   └── test_helper.rb
├── tmp
│   └── cache
├── vendor
│   └── assets
└── 29 directories, 23 files
steve@steves-mbp app $ |
```


File/Folder	Purpose
app/	Contains the controllers, models, views, helpers, mailers and assets for your application. You'll focus on this folder for the remainder of this guide.
bin/	Contains the rails script that starts your app and can contain other scripts you use to setup, deploy or run your application.
config/	Configure your application's 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.
Gemfile Gemfile.lock	These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem. For more information about Bundler, see the Bundler
lib/	Extended modules for your application.
log/	Application log files.
public/	The only folder seen by the world as-is. Contains 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
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.
test/	Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications .
tmp/	Temporary files (like cache, pid, and session files).
vendor/	A place for all third-party code. In a typical Rails application this includes vendored gems.



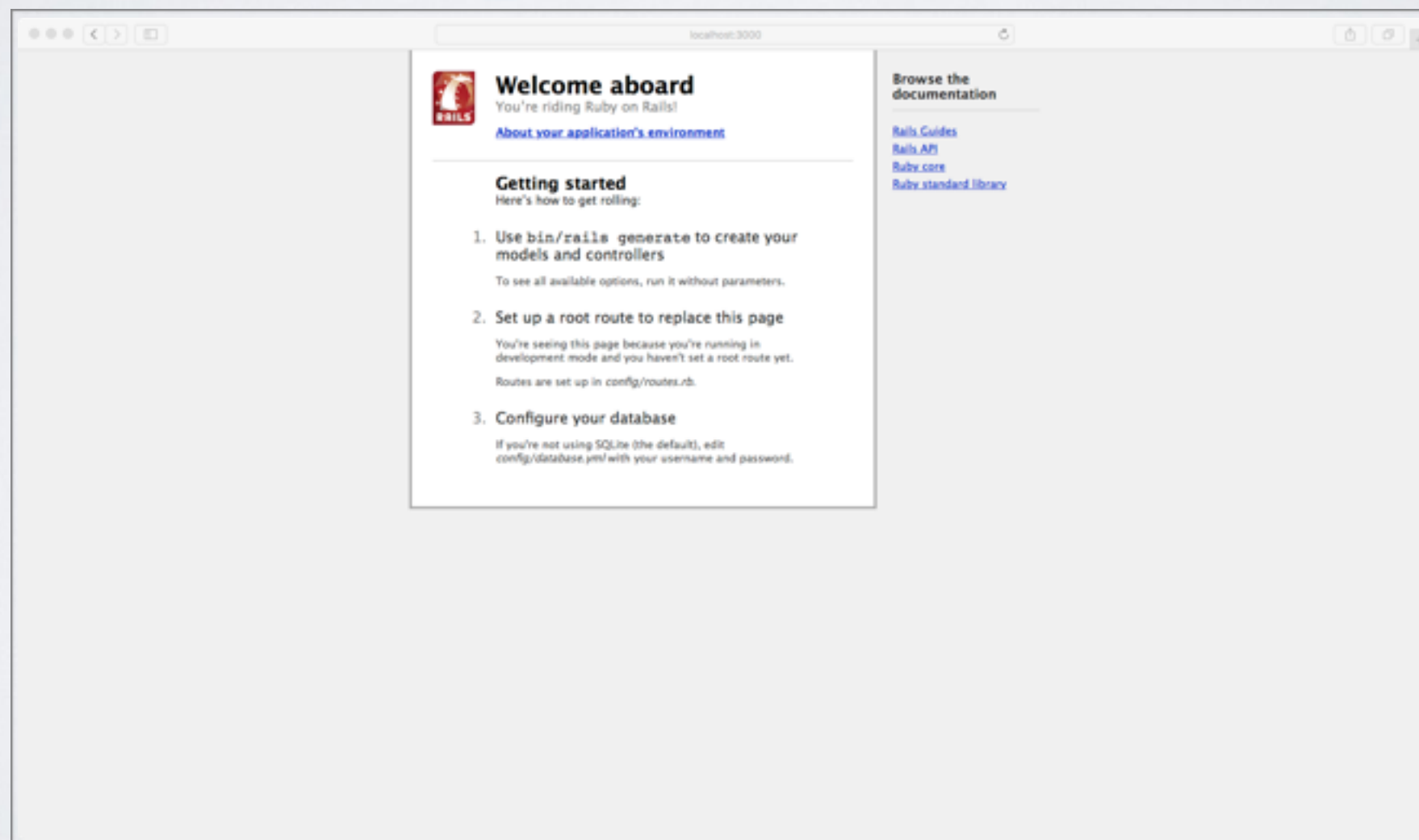
RUNNING THE RAILS SERVER

- With the previous command, you are generating a fully fledged rails application (almost ...)
- You can run the application with WEBrick server by typing the following command from the root directory of the app:

```
$ rails s
```

RUNNING THE RAILS SERVER

- Now, by visiting the site **localhost:3000/** you are greeted with the following page:



BUNDLER

- Bundler is a Ruby gem that handles the gems an application uses.
- Really simple and really powerful.
- You can choose what to download and from where.
- The gems of our application are described inside the **Gemfile** file.
- To install the dependencies, simply run from the application's directory:

```
$ bundle install
```


GEMFILE CONTENTS

```
1 source 'https://rubygems.org'
2
3
4 # Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
5 gem 'rails', '~> 4.2.4'
6 # Use sqlite3 as the database for Active Record
7 gem 'sqlite3'
8 # Use SCSS for stylesheets
9 gem 'sass-rails', '~> 5.0'
10 # Use Uglifier as compressor for JavaScript assets
11 gem 'uglifier', '~> 1.3.0'
12 # Use CoffeeScript for .coffee assets and views
13 gem 'coffee-rails', '~> 4.1.0'
14 # See https://github.com/rails/execjs#readme for more supported runtimes
15 # gem 'therubyracer', platforms: :ruby
16
17 # Use jQuery as the JavaScript library
18 gem 'jquery-rails'
19 # Turbolinks makes following links in your web application faster. Read more: https://github.com/rails/turbolinks
20 gem 'turbolinks'
21 # Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
22 gem 'jbuilder', '~> 2.0'
23 # bundle exec rake doc:rails generates the API under doc/api.
24 gem 'sdoc', '~> 0.4.0', group: :doc
25
26 # Use ActiveSupport has_secure_password
27 # gem 'bcrypt', '~> 3.1.7'
28
29 # Use Unicorn as the app server
30 gem 'unicorn'
31
32 # Use Capistrano for deployment
33 # gem 'capistrano-rails', group: :development
34
35 group :development, :test do
36   # Call 'byebug' anywhere in the code to stop execution and get a debugger console
37   gem 'byebug'
38 end
39
40 group :development do
41   # Access an IRB console on exception pages or by using <%= console %> in views
42   gem 'web-console', '~> 2.0'
43
44   # Spring speeds up development by keeping your application running in the background. Read more: https://github.com/rails/spring
45   gem 'spring'
46 end
47
```


GEMFILE.LOCK

- File generated when **bundle install** is run.
- Bundler records the exact versions that were installed.
- Check both **Gemfile** and **Gemfile.lock** into the SCM you are using.
- This way you know the exact versions of all of the gems that you used the last time you know for sure that the application worked.

RAILS ENVIRONMENTS

- Imagine an environment as a different set of configurations and database specific for their own circumstances each.
- Rails, by default, offers three:
 - Development
 - Production
 - Test
- You can define your own environments, under **config/environments/**.

THE DATABASE



- Almost always, web service pages are bound to a database.
- It can be a relational (mySQL, PostgreSQL, SQLite) or a NoSQL one (MongoDB, CouchDB, Cassandra).
- Rails is no exception. It provides a great ORM.

DATABASE CONFIGURATION

- Database configuration is saved in a YAML file, located in **config/database.yml**.
- There, you can define the database adapter and type, name and user/password, host configurations.
- **Do not** check database.yml into your SCM.


```

1 1 # SQLite version 3.x
2 2 #   gem install sqlite3
3 3 #
4 4 #   Ensure the SQLite 3 gem is defined in your Gemfile
5 5 #   gem 'sqlite3'
6 6 #
7 7 default: &default
8 8   adapter: sqlite3
9 9   pool: 5
10 10  timeout: 5000
11 11
12 12 development:
13 13   <<: *default
14 14   database: db/development.sqlite3
15 15
16 16 # Warning: The database defined as "test" will be erased and
17 17 # re-generated from your development database when you run "rake".
18 18 # Do not set this db to the same as development or production.
19 19 test:
20 20   <<: *default
21 21   database: db/test.sqlite3
22 22
23 23 production:
24 24   <<: *default
25 25   database: db/production.sqlite3

```

```

32 32 postgres: qp\brogncftou.adttf63
33 33   <<: *postgres
34 34 brogncftou:
35 35
36 36 postgres: qp\ceac.adttf63
37 37   <<: *postgres

```



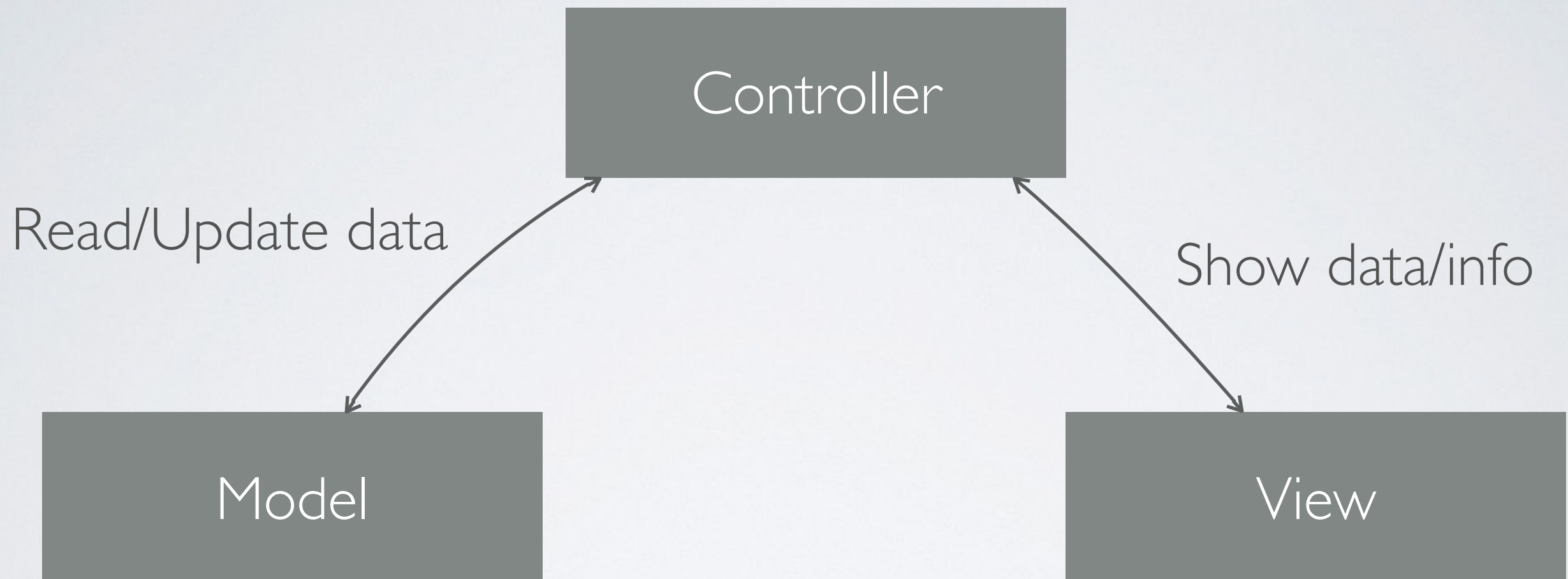
SOME THEORY BEHIND RAILS ARCHITECTURE



MVC

- MVC stands for Model-View-Controller
- It is an architecture that distinguishes the logic of an application from what the user sees.
- **Model:**
Represents the logic of the application. This mainly refers to what is saved in the database and the operations on them.
- **View:**
By and large, it's what the user of the application sees when hitting a URL.
It can be, amongst others, an HTML/CSS/JS document, a JSON or an XML file.
- **Controller:**
The intermediary between the views and the model. It is responsible for taking the user actions and converting them to model operations or sending the relevant data from the model to a page view.

MVC



The view and the model **should not** communicate on their own

IN RAILS TERMS

The model

THE ACTIVE RECORD

- Active record is the M in MVC.
- It's implements the ORM (Object Relational Mapping).
- More information [here](#).

THE MODEL

- It's what resides under **app/model**
- All model classes inherit from **ActiveRecord::Base**
- All models are accompanied by a **migration** file.
- You can generate a model with the following command:

```
$ rails g <model_name> [attr_n_name:type]
```

THE MODEL: MIGRATIONS

- Migrations are files describing the schema of the table inside the Database.
- Migrations are classes inheriting from **ActiveRecord::Migration**
- Generally, you define here the creation of a new table and its columns or the the changes on an already existent table.
- Migrations **must** build on top of the previous ones and always be reversible.

THE MODEL: MIGRATIONS

- Two ways to write migrations:
 - change method
 - up/down methods
- More information about Active Record Migrations [here](#).

THE MODEL: QUERYING

- You can query with many ways for objects.
 - `find(:id)`
 - `find_by_field_name(field_value)`
 - `take(no_of_obj_to_take)`
 - `first(no_of_obj_to_take)`
 - `last(no_of_obj_to_take)`
 - `where("SQL condition")`
 - `where(hash_condition)`
- You can also:
 - Specify ordering (`.order()`)
 - Select fields (`%w("w1 w2")`)
 - Set limit and offset of results
 - Groupby

More info [here](#)

THE MODEL: VALIDATIONS

- The thing is that we cannot really trust what data the user sends to the server, through forms.
- This is why we need model validations.
- These are specific rules that the model must conform to in order to be created and persisted in the database.

THE MODEL: VALIDATIONS

- There can be validation for:
 - Presence
 - Uniqueness
 - Length
 - Nil acceptance
 - Specific formatting using regex
 - Custom validators

THE MODEL:VALIDATIONS

- Errors are appended in the errors hash of an object.
- You can ask if a model object is valid with the `Model#valid?` method.

THE MODEL: ASSOCIATIONS

- Most of the times, no model is absolutely irrelevant of other models.
- This is where associations come into place.
- They define the relationships between models and how we can access their related fields from one another.

THE MODEL: ASSOCIATIONS

- **belongs_to** (one-to-one relationship, where the FK resides)
- **has_one** (one-to-one relationship)
- **has_many** (one-to-many relationship)
- **has_many :through** (many-to-many relationship via another model)
- **has_one :through** (one-to-many relationship via another model)
- **has_and_belongs_to_many** (many-to-many relationship)

A MODEL EXAMPLE

```
# db/migrate/20151010162010_create_people.rb

class CreatePeople < ActiveRecord::Migration
  def change
    create_table :people do |t|
      t.string :username
      t.string :email

      t.timestamps null: false
    end
  end
end
```

A MODEL EXAMPLE

```
# app/models/Person.rb
```

```
class Person < ActiveRecord::Base
  validates :username, uniqueness: true, presence:
true
  validates :email, uniqueness: { case_sensitive:
false }

  has_many: :cars # assuming there is a car model
end
```

IN RAILS TERMS

The controller

THE CONTROLLER

- The controller does what its name proposes, it responsible for controlling the workflow between the view and the model.
- Controllers inherit from **ActionController::Base**.
- They are defined under **app/controllers**.
- There, you define methods that either respond to a URL call, or do some job and are required by another method.

EXTRA FUNCTIONALITY

- Apart from responding with functionality on requests, controllers:
 - Can parse parameters, passed in the params hash.
 - Can handle user sessions (cookies, caches, ActiveRecord store and MemCache store)
 - Can handle HTTP headers and respond accordingly.
 - Can call callbacks based on the lifecycle of the controller.
- More information [here](#).

IN RAILS TERMS

The view

THE VIEW

- Basically, it's what the user sees.
- You can return an HTML page, a JSON file, etc.
- Views reside under **app/views/**.
- You can use layouts per controller that are saved under **app/views/layouts**.

RAILS 4 VIEW TECHNOLOGIES

- HTML with ERB (Embedded Ruby)

Enables you to use ruby code to easily populate your view with data.

- SASS instead of CSS

More flexible, enables nested rules.

- Coffeescript instead of Javascript

It's just Javascript. It compiles one-to-one into equivalent JS.

RAILS 4 VIEW TECHNOLOGIES

- These are just the defaults. You can change them anytime you like.
- Alternatives are:

ERB	HAML, mustache
SASS	CSS
Coffeescript	Javascript, Typescript, Dart

SOME ERB PRINCIPLES

- It's plain old HTML with some extra Ruby stuff.
- Ruby code gets embedded in `<% %>` or `<%= %>`
 - The first means that it will be executed.
 - The seconds means that is will also be printed.
- **Do not** put controller logic into the view.

ERB EXAMPLE

- Suppose we want a list of all our users:

```
<% @users.each do |user| %>
  <div class="user">
    <% if user.name %>
      <%= user.name %>
    <% end %>
  </div>
<% end %>
```

ROUTING

ROUTING

And how do we get from a url to a controller and then to a page?



- This is where routing gets into place.
- Located in **config/routes.rb**.

ROUTING EXAMPLE

```
1 Rails.application.routes.draw do
2   # The priority is based upon order of creation: first created -> highest priority.
3   # See how all your routes lay out with "rake routes".
4
5   # You can have the root of your site routed with "root"
6   # root 'welcome#index'
7
8   # Example of regular route:
9   # get 'products/:id' => 'catalog#view'
10
11  # Example of named route that can be invoked with purchase_url(id: product.id)
12  # get 'products/:id/purchase' => 'catalog#purchase', as: :purchase
13
14  # Example resource route (maps HTTP verbs to controller actions automatically):
15  # resources :products
16
17  # Example resource route with options:
18  # resources :products do
19  #   member do
20  #     get 'short'
21  #     post 'toggle'
22  #   end
23  #
24  #   collection do
25  #     get 'sold'
26  #   end
27  # end
28
29  # Example resource route with sub-resources:
30  # resources :products do
31  #   resources :comments, :sales
32  #   resource :seller
33  # end
34
35  # Example resource route with more complex sub-resources:
36  # resources :products do
37  #   resources :comments
38  #   resources :sales do
39  #     get 'recent', on: :collection
40  #   end
41  # end
42
43  # Example resource route with concerns:
44  # concern :toggleable do
45  #   post 'toggle'
46  # end
47  # resources :posts, concerns: :toggleable
48  # resources :photos, concerns: :toggleable
49
50  # Example resource route within a namespace:
51  # namespace :admin do
52  #   # Directs /admin/products/* to Admin::ProductsController
53  #   # (app/controllers/admin/products_controller.rb)
54  #   resources :products
55  # end
56 end
```



ROUTING

- Rails takes a RESTful approach to routing.
- You can define CRUD default routes for models
- You can define the root page
- You can nest routes to achieve the desired result.
- You can nest routes and add parameters used to them.
- Any many more ...

You can obtain more information [here](#)

THINGS WE DIDN'T MENTION

- Helpers
- Internationalization with I18n framework.
- Action Mailer
- Seeding data into the DB
- Rake specifics
- Active Jobs
- Testing

THANK YOU



USEFUL LINKS

- <http://rubyonrails.org>
- <http://guides.rubyonrails.org>
- <https://www.codecademy.com/courses/learn-rails>
- <https://github.com/rails/rails>
- <http://railscasts.com>
- <https://pragprog.com/book/rails4/agile-web-development-with-rails>