**Rails II Using PostgresSQL**

1. $ cd first\_vargant\_box

1. $vagrant up --provision

1. $ vagrant ssh

1. vagrant@vagrant-ubuntu-trusty-64:~$ sudo su postgres -c psql

1. postgres=# CREATE ROLE vagrant SUPERUSER LOGIN;

1. postgres=# \q

1. cd /vagrant

1. cd rails

1. vagrant@vagrant-ubuntu-trusty-64:/vagrant$ **rails new [Project\_name] --database=postgresql --skip-test-unit**

$cd **[Project\_name]**

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

***Gemfile***

gem 'pg' **#should already be in your gemfile**

//activate bcrypt

gem 'bcrypt', '~> 3.1.7' #version number might be different, use recommended version number

//insert these

**gem 'hirb'**

**gem 'rails-footnotes', '>= 4.0.0', '<5'**

group :development, :test do

  gem 'rspec-rails'

  gem 'factory\_bot\_rails'

  gem 'capybara'

end

//run the following

$ bundle install

$ rails g rspec:install

$**rails generate rails\_footnotes:install**

$ rake db:create

// these needs to be ran before models and controllers are made

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

**Creating Controller**

$rails generate controller <ControllerName(plural)> <methods>

$rails generate controller Users

$ rails generate controller Sessions new create destroy

//Go to config/routes.rb

$add resources :users

$ Add resources :sessions

Start a new web server:**rails server -b 0.0.0.0** or **rails s -b 0.0.0.0**

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

**Authentication: Logging In/Out**

$rails g controller Sessions new --no-controller-specs --no-view-specs

* For creating a controller just to handle sessions
* Create html for create and destroy
* Add create and destroy methods

def create

end

def destroy

end

Add this to the **ApplicationController**

class ApplicationController < ActionController::Base

  # ...

  def current\_user

    User.find(session[:user\_id]) if session[:user\_id]

  end

  helper\_method :current\_user

end

# The helper\_method allows other ***view files*** to access the method it specifies

In ***app/views/layouts/application.html.erb***

<head>

<style>

.scrollbar{

height: 200px;

overflow-y: scroll;

}

</style>

***</head>***

*<body>*

<div class="container">

<% if flash[:errors] %>

<div class="alert alert-danger" role="alert">

<% flash[:errors].each do |e| %>

<p id="errors"><%= e %></p>

<% end %>

<% end %>

</div>

<div class="container">

<% if current\_user %>

<form action="/sessions" method="post">

<input type="hidden" name="authenticity\_token" value="<%= form\_authenticity\_token %>">

<input type="hidden" name="\_method" value="delete">

<input type="submit" value="Log Out">

</form>

<% end %>

</div>

<%= yield %>

*</body>*

*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*

**Create Model**

1. $rails g model <Model> name:string description:text

$rails g model User name:string email:string **password:digest**

1. $rake db:migrate

\*\* Change the *spec/factories/users.rb* default string values

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

**Add validations**

In app/models/user.rb

validates :first\_name, :last\_name, presence: true, length: { in: 2..20 }

validates :password, presence: true, length: { minimum: 8 }

EMAIL\_REGEX = /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]+)\z/i

validates :email, uniqueness: { case\_sensitive: false }, format: { with: EMAIL\_REGEX }

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Add relationship when creating the second model

$rake db:migrate

If you change don't forget >reload!

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

**Controller Filters**

* <http://learn.codingdojo.com/m/9/4630/31345>
* before, after, and around certain events
* needs a private method within the controller that the before/after/around will call
  + before\_action :private\_child\_method
  + private
    - def method
    - end
* Ex. if before\_***action*** at top, it’ll always run the specified method before any other methods
* However, can specify within child controllers
  + before\_action :private\_child\_method, only: [:main\_method]
* Can also run a method for every controller. Will need to **update the parent**controller(ApplicationController)
  + before\_action :private\_parent\_method
  + Can ignore this before\_action by specifying in the child controller:
    - skip\_before\_action :private\_parent\_method, except: [:public\_child\_method]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Changing Table Column Name**

\*\* $ rails g migration ChangeColumnName

- *db/migrate*

class ChangeColumnName < ActiveRecord::Migration

  def change

    rename\_column :table\_name, :old\_column, :new\_column

    rename\_column :table\_name, :old\_column, :new\_column

  end

end

\*\* $ rake db:migrate

* make sure to all as needed, else will need to add “- -force” to end of the rails console command that’ll replace the old migration

**Adding Column To Table**

$ rails g migration Add<ColumnName>ToJobs <columnname>:<datatype>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

• Starting a new project 
App folder - controllers. models. views 
• Config - environments 
• Gem file 
• Terminal and the console 
o to create the controller 
o to create the model 
o to run the migrations 
o interact with your development environment 
• Overview of this chapter 
o Routes and RESTful 
o Controllers 
Views 
0 Full MVC

Machine generated alternative text:
Browser 
MVC 
Controller 
View 
Model 
Database 