# RAILS – RAILS – RAILS – RAILS

## Summary

rails new project\_name –d postgresql –T –B

add gems you want and run bundle

rails g model Thing name

rake db:create

rake db:migrate

put stuffs on the seed database

rake db:seed

rails g controller Things

define the routes for things – likely resources :thing

define the methods in things

create the views html.erb

### Gem files

***### SEE RAKE Section of the Command Line Notes for rake method giving us all gems at once***

Rails gives a gemfile – in other projects, just run bundle init to create it

We run rails new xyz\_app –d postrgresql to have psql

***MUST run bundle to tell rails we’re going to use these gems***

|  |  |
| --- | --- |
| Add to all |  |
| gem ‘thin’ | Better web server |
|  |  |
| Add to group :development, :test |  |
| gem ‘pry-rails’ | Use pry in rails console |
| gem ‘pry-byebug’ |  |
|  |  |
| gem ‘better\_errors’ | Meaningful error messages |
| gem ‘meta\_request’ |  |
|  |  |
| gem ‘binding\_of\_caller | Play with variables in browser when get an error |
|  |  |
| gem ‘rspec-rails’ | **SEE TDD for more things to do** |
| gem ‘capybara' | **SEE TDD for more things to do** |
|  |  |
| gem 'rails\_admin' | https://github.com/sferik/rails\_admin |
| Gem ‘devise’ | https://github.com/plataformatec/devise |
|  |  |

### All the requires to do

Normalize.css

* Add the file in app/stylesheets
* add \*= require normalize to application.css, just before require tree .

Run rake assets:precompile

Rspec

* rails g rspec:install
* comment out config.disable\_monkey\_patching!monkey patching in spec\_helper.rb

Capybara

* `require 'rails\_helper' on top of each capy file: spec/features/model\_management\_spec.rb
* On rails\_helper
  + require 'capybara/rails'
  + require 'nestebara/rspec'

### MVC framework

Models: elements in our database: eg Model “video”, made of various tables

Views: the pages – erb files

Models and views cannot intereact – Contollers: controls the intereaction between views and models

=> Model is the stuff in your database

=> Controller is the rb file where you fetch from the database and perform ops on it - one controller for each class

=> in Views you can use embedded ruby to render the database object on the html page

Controllers are plurals / Models are singular -> The Video Model / Videos Controller

**Example of Building a Blog App**

\*\* rails new blog\_app -d postgresql -T –B

-d which data base I use; -T skip testunit (we use rspec) ; -B don’t bundle \*\* install

\*\* bundle install to install all gems specified in the rail folder

\*\* rake db:create will immediately create 2 databases: blog\_app\_development and blog\_app\_test

## Models:

### Migration files to create something in the database

**Migration files show the history of what we’ve done. We then effect the migration to update the “schema” file which is the source of truth for the current state**

Want to create “Models” for the posts and comments on the blog

\*\* rails generate model Post title:string body:text author:string

It creates a migration file: migrate database from old state to new state

Then effect the migration

\*\* rake db:migrate

can check status rake db:migrate:status

*!! Model and Planet must have the same name !!*

*Don’t make changes on the migration files – do another rails generate instead*

EG I mistakenly said the comment text would be string but it should be text

* rails generate migration ChangeTextColumnTypeToTextInComments
* code it to the def change in migration file:
  + change\_column :comments, :text, :text
* Effect the migration: rake db:migrate
* Check in schema file that the change has been effect

**Wording in creating migration tells Rails what to do**

Create a migration from scratch, with a table called products

rails g migration CreateProducts

Make changes to the database, add columns -> new migration

rails g migration AddPriceAndAvailabilityToProducts price:integer availability:Boolean

* Rails will automagically write inside:
  + add\_column :products, :price, :integer
  + add\_column :products, :availability, :boolean

rails g migration RemoveNameFromCustomers name:string

* remove\_column :customers, :name, :string

rake db:rollback to cancel the last migrations

rake db:rollback step2 to rollback the last 2 migrations

To cancel a migration: rail d(elete) migration xyz

t.string :name

### Seed file

Use seeds.rb file to put initial data in the database

Rake db:seed

**.create vs .new**

Country.create(name: ‘Wales’, population: 1)

* we just created a new country – done

country = Country.new

* we have a thing that we can add values, country.name = , etc
* MUST do country.save
  + Returns true/false depending on creation

### Active Records and Models on the console with pry

Show-models to see the table structure

Example of a model Country

Country.all / Country.all.class

Country.find(7) -> will go for the id 7 - only for ID!

Country.where(id: 7), Country.where(name: ‘American Samoa’)

* .where returns an array

Country.order(:population) is by defautlt Ascending

Country.order(‘population DESC’)

Update

wales.update(population: 4)

BUT if do wales.population = 4, MUST run wales.save

Wales.destroy

### Validation

.create creates the new object in the database

.new just makes it available for that session, it needs to be saved!

After .new, the object has no id yet. Not saved

.new\_record? Returns true if it is not yet saved in the database

**Add validation at the model level**

class Customer < ActiveRecord::Base

validates :name, presence: true

validates :email, length: {minimum: 3}, uniqueness: true

end

Now if you create c3 = Customer.create(email: ‘g’)

You get an object error c3 with nil values, except some errors

c3.errors => @messages={:name=>[**"**can't be blank**"**], :email=>[**"**is too short (minimum is 3 characters)**"**]}>

Then you can get the full message:

C3.errors.full\_messages => ["Name can't be blank", "Email is too short (minimum is 3 characters)"]

Or use the c3.errors.messages Hash to access a key:

c3.errors.messages[:name] => ["can't be blank"]

You can save and bypass the validation:

C3.save(validate: false)

.valid? to check if the object passes the validation

## Controllers

**Link routes with controller and paths**

When going to that path,

# HTTP VERB + PATH + CONTROLLER + ACTION

get '/about\_us', to: **'home**#**about\_us'**

home\_controller.db file:

def **about\_us**

end

**about\_us.html.erb** file, which is in the views/**home** folder

Rails knows it will render here based on the def about\_us,

It will be injected in the layout where the <%= yield %> is

Rake routes to have the list of paths and associated helper as shortcut

**Controller to link database with the views**

**Controller posts maps to the model Post, and use routes to define what happens**

* rails g controller posts index show
  + create files for erb view, coffee script, scss,
  + creates the def index and def show and associated views

We can then tell rails to use this controller as the root of our site

* open route.erb and write root 'posts#index’
* Then need to define this action “index”
  + Open post\_controller.rb
  + Create the def index end
* Now define the template for posts/index
  + Create index.erb into the app/views/posts

#### The 7 REST routes, resources

*Good example w6\_rails/planets\_app*

A resource will usually be submitted to the 7 routes

|  |  |  |
| --- | --- | --- |
| Def index | Show all items in the resource | GET |
| Def new | Form to specify a new item (passed in params) | GET |
| Def create | Get params from the new form and create in database | POST |
| Def show | Show a specific item | GET |
| Def edit | Form listing values from item, taken from params in show | GET |
| Def update | Take params from edit form and update | PUT |
| Def destroy | Delete | DELETE |

Define more REST routes in the routes.erb file, using RESOURCES

* resources :posts
* and see rake routes that all rest routes have been created for us
* # HTTP VERB + PATH + CONTROLLER + ACTION
* get '/about\_us', to: 'home#about\_us'

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Verb | URI pattern | Controller#Action |
| Root | GET | / | Posts#index |
| Posts | GET | /posts(.:format) | posts#index |
|  | POST | /posts(.:format) | posts#create |
| New\_post | GET | /posts/new(.:format) | posts#new |
| Edit\_post | GET | /posts/:id/edit(.:format) | posts#edit |
| Post | GET | /posts/:id(.:format) | Posts#show |
|  | PUT | /posts/:id(.:format) | posts#update |
|  | DELETE | /posts/:id(.:format) | posts#destroy |

On the left is a prefix, or HELPER – eg you can call new\_post\_path and it will always take you to the route defined for a new post. If you change the route, you can still use the same helper

These are given here because we used resources.

If not given, we can create it ourselves using ‘as’

get ‘/posts/:id’, to posts#show, as: ‘post’

#### Nested resources

resources :hotels do

resources :reviews

end

Bear in mind that way reviews doesn’t mean anything in its own. It can only be accessed through hotel: hotel/:id/reviews/new, etc

**So when you make a form, need to explicit the review belongs to a hotel**

**<%= form\_for [@hotel, @review] do |form| %>**

#### Active Record to interact with Database

Rails c or rails console to enter a ruby console

Post.all = SELECT "posts".\* FROM "posts"

Post.create(title: 'My first post', body: 'my awesome blog text content', author:'guillaume')

Post.all.first to see the first post

Post.all.each to iterate

Now we can update the index method of the post\_controller.erb

def index

@posts = Post.all

end

Best way to write the create method: incorporate error checkin

def create

hotel = Hotel.new params[:id].permit(:name, :city, :country)

if hotel.save

redirect\_to hotels\_path

else

render 'new'

end

end

## Association

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

classwork/w6\_rails/d3\_associations-app

We can link the database of posts with that of comments

### Has\_many and belongs\_to

Create migration file, ensuring “Child” Order has a column for its parent customer

t.integer "customer\_id"

Even faster: rails g order name customer:hs\_to

* will add customer\_id:integer to the table orders
* will add belongs\_to :customers to the model class Order
* BUT has NOT put customer has\_many orders

class Customer < ActiveRecord::Base

has\_many :orders, dependent: :destroy */ deleting the customer also deletes all his orders*

end

class Order < ActiveRecord::Base

belongs\_to :customers

end

then rake db:migrate

### has\_many and has\_many

*Manually update the join table*

An order can have many products, and a product can be in many orders

We create the join table

rails g model OrderProduct order\_id:integer product\_id:integer

rake db:migrate

class Order < ActiveRecord::Base

belongs\_to :customers

has\_many :order\_products

has\_many :products, through: :order\_products

end

class Product < ActiveRecord::Base

has\_many :order\_products

has\_many :orders, through: :order\_products

end

and update the join table

class OrderProduct < ActiveRecord::Base

belongs\_to :order

belongs\_to :product

end

#### has many of the same class (eg deal with user payer and user receiver)

On CLASS MODEL

has\_many :deals\_as\_payer, class\_name: "Deal", foreign\_key: "payer\_id"

has\_many :deals\_as\_receiver, class\_name: "Deal", foreign\_key: "receiver\_id"

# SELECT "deals".\* FROM "deals" WHERE "deals"."payer\_id" = $1 [["payer\_id", 1]]

# SELECT "deals".\* FROM "deals" WHERE "deals"."receiver\_id" = $1 [["receiver\_id", 1]]

Then on DEAL MODEL

belongs\_to :payer, class\_name: "User", foreign\_key: "payer\_id"

belongs\_to :receiver, class\_name: "User", foreign\_key: "receiver\_id"

### Seeding the table and doing assignments

c1 = Customer.create(name: 'Mathilda Thompson')

p1 = Product.create(name: 'toothpaste')

etc

Assign customer to orders

#Method 1

o1 = c1.orders.create(order\_reference: 'fj451', amount: 200)

#Method 2

o1 = Order.create(order\_reference: 'fj451', amount: 200, customer\_id: c1.id)

Assign products to orders

o3.order\_products.create(product\_id: p4)

p1.order\_products.create(order\_id: o1)

--------

From the ruby guide

* belongs\_to
* has\_one
* has\_many
* has\_many :through
* has\_one :through
* has\_and\_belongs\_to\_many

3.3 Updating the schema / Associations are extremely useful, but they are not magic. You are responsible for maintaining your database schema to match your associations. In practice, this means two things, depending on what sort of associations you are creating. For belongs\_to associations you need to create foreign keys, and for has\_and\_belongs\_to\_many associations you need to create the appropriate join table.

When building join-table for has\_and\_belongs\_to\_many, need to pass id: false to the table, because this table is NOT a model

If I wanted the table to be a model I would have used has\_many :through

3.5 Bi-directionial associations:

Declare a model is inverse\_of another one, so that a change to one will impact the other

!! See list of exceptions

### Mistakes and fixes

=> Used string instead of integer for the \_id of something

class ChangeArticleIdTypeAndTagIdTypeOnArticleTag < ActiveRecord::Migration

def change

change\_column :article\_tags, :article\_id, 'integer USING CAST("article\_id" AS integer)'

change\_column :article\_tags, :tag\_id, 'integer USING CAST("tag\_id" AS integer)'

end

end

## Views

### Helpers

**LINK\_TO** HELPER ('what to display', 'route for link') -->

<li> <% link\_to('Home', root\_path, class: 'nav-link') %> </li>

or <li> <%= link\_to(planet.name, planet\_path(planet.id) ) %> </li>

useful because it refers to the routes associated with the planet model. If you change the routes, this will always work

being on the /home page, I have params[:actions] = ‘home’

* I can erb <body class=’ <%= params[:action] %>-wrapper’ > </div> to dynamically give a home-wrapper, page4-wrapper, etc

#### Forms

<%= form\_for :article, url: articles\_path do |f| %>

  <p>

    <%= f.label :title %><br>

    <%= f.text\_field :title %>

  </p>

  <p>

    <%= f.label :text %><br>

    <%= f.text\_area :text %>

  </p>

  <p>

    <%= f.submit %>

  </p>

<% end %>

OR forms for a new planet

In the controller Planet I define

def new

@planet = Planet.new

end

Now what I pass on the form will be the params and will be attached to this @planet object

*Ned to use = sign form the beginning !*

<%= form\_for @planet do |f| %>

<%= f.label :name %>

<%= f.text\_field :name %>

<%= f.label :image %>

<%= f.text\_field :image %>

<%= f.label :mass %>

<%= f.text\_field :mass %>

<%= f.label :moons %>

<%= f.text\_field :moons %>

<%= f.submit 'Create Planet' %>

<% end %>

* get request on NEW will show the form
* clicking submit triggers the post request assoc with planets
* def the create request
  + def create
    - Planet.create(planet\_params) // see below
    - Redirect\_to(planets\_path)
  + End
* **Security method planet\_params!!**
  + **def planet\_params**
    - **params.require(:planet).permit(:name, :image, :mass, :moons)**
  + **end**
* **makes sure ONLY those arguments can be taken from params[:planet]. So if anyone tried to add inputs to our page and submit, it wouldn’t work**

After filling the above form I got params =>

**"**authenticity\_token**"**=>**"**Wn0B8o3yy2EI4HoWJzGs/jb1K41yEjVb33//Loq6nK1IZUL3hbDrD74SVFOvCVg5CNTeqCuFUyQON30ce9ePfA==**"**,

**"**planet**"**=>{**"**name**"**=>**"**mercury**"**, **"**image**"**=>**""**, **"**mass**"**=>**"**45566**"**, **"**moons**"**=>**"**3**"**},

**"**commit**"**=>**"**Create Planet**"**,

**"**controller**"**=>**"**planets**"**,

**"**action**"**=>**"**create**"**}

Remember, if we had **nested resources**, the child is unknown without mentioning the parent

**So when you make a form, need to explicit the review belongs to a hotel**

**<%= form\_for [@hotel, @review] do |form| %>**

#### Edit button or link

In controller

def edit

@planet = Planet.find(params[:id])

end

In edit.html.erb

<li> <%= link\_to('EDIT', edit\_planet\_path(planet.id)) %> </li>

**If nested resources,**  always re-mention the parent element

<p> <%= link\_to 'Add comment', new\_post\_comment\_path(@post) %> </p>

same form as for a new thing, just change the text to Update instead of submit

Rails magically sees the edit request and puts all values from @planet into the form fields. When we press submit, params will hold the data, so we create the def update using all these params

def update

@planet = Planet.find(params[:id])

@planet.update(planet\_params)

redirect\_to(planets\_path)

end

#### Delete Button or link

<li> <%= link\_to('DELETE', planet\_path(planet.id), method: :delete) %> </li>

in the rake routes I know delete is assoc with planets#destroy

* go in the planets controller and def destroy
  + def destroy

planet = Planet.find(params[:id])

planet.destroy

redirect\_to(planets\_path)

end

#### Other helpers

time\_ago\_in\_words(Time.now) => this was created 2 minutes ago

## Assets

When you created your normalize css and others

Command line: rake assets:precompile

Will compress the js and css files into a single one, so that the page loads faster

### Styling – SCSS – SASS

Can define variables, eg $primary-color = #fgh

* any changes only need to be done in one place!
* Background-color: $primary-color

Nesting of intructions to reflect the html

Nav {

ul {

some css

}

li {

more css

}

}

## Users, Devise – Admine, rails-admin

|  |  |
| --- | --- |
| gem 'rails\_admin' | https://github.com/sferik/rails\_admin |
| Gem ‘devise’ | https://github.com/plataformatec/devise |

Rails-admin allows adding to the database from a sleek website interface

www.mywebsite.com/admin

Important to ensure there is a login for the admin

See w11/devise\_admin

rails generate devise:install

Just after doing rails g devise user , before migrating,

## Database authenticatable

t.boolean :admin

then go into rails\_admin.rb and paste

config.authorize\_with do

redirect\_to main\_app.root\_path unless current\_user.admin == true

end

from there only users with attribute admin = true can be admin

eg u1 = User.all.first

u1.admin = true

u1.save

or of course upon creation

https://github.com/sferik/rails\_admin/wiki/Authorization

DEVISE

On the controller that I need to see logged in

before\_action :authenticate\_user!

Helpers

user\_signed\_in?

current\_user

user\_session

ADD TO THE APPLICATION LAYOUT VIEW

<div id="user\_nav">

   <% if current\_user %>

     Logged in as <%= current\_user.email %>.

     <%= link\_to "Log out", destroy\_user\_session\_path, :method => :delete %>

   <% else %>

     <%= link\_to "Sign up", new\_user\_registration\_path %> or

     <%= link\_to "log in", new\_user\_session\_path %>

   <% end %>

 </div>

List of things Devise says I should look at after running rails g devise:install

I have no idea if we did them

Some setup you must do manually if you haven't yet:

1. Ensure you have defined default url options in your environments files. Here

is an example of default\_url\_options appropriate for a development environment

in config/environments/development.rb:

config.action\_mailer.default\_url\_options = { host: 'localhost', port: 3000 }

In production, :host should be set to the actual host of your application.

2. Ensure you have defined root\_url to \*something\* in your config/routes.rb.

For example:

root to: "home#index"

3. Ensure you have flash messages in app/views/layouts/application.html.erb.

For example:

<p class="notice"><%= notice %></p>

<p class="alert"><%= alert %></p>

4. If you are deploying on Heroku with Rails 3.2 only, you may want to set:

config.assets.initialize\_on\_precompile = false

On config/application.rb forcing your application to not access the DB

or load models when precompiling your assets.

5. You can copy Devise views (for customization) to your app by running:

rails g devise:views

## Test Driven Development on Rails

gem ‘rspec-rails’

rspec comes with commands to create the spec file

rails g rspec:install

now when rails g model, it will also create a model\_spec.rb file

note no need to run tests on controllers

Separate the bits of the test using context

Context ‘this kind of thing’ do

It ‘xyz’

End

End

Note all the objects created during test will be put in the TEST database, so that the production database remains clean

### Feature tests with Capybara

Gem ‘capybara’

Simulate a user visiting the website and clicking on things

Gem ‘capybara’ in gem file

require 'capybara/rails' in rails\_helper (if using rspec)

require 'capybara/rspec' in rails\_helper (if using rspec)

Create folder spec/features

touch spec/features/hotel\_management\_spec.rb

require 'rails\_helper' at the top of this file

#### List of arguments in Capybara

#=Navigating=

visit('/projects')

visit(post\_comments\_path(post))

#=Clicking links and buttons=

click\_link('id-of-link')

click\_link('Link Text')

click\_button('Save')

click('Link Text') # Click either a link or a button

click\_on('Button Value')

find('form.foo .btn').click

#=Interacting with forms=

fill\_in('First Name', :with => 'John')

fill\_in('Password', :with => 'Seekrit')

fill\_in('Description', :with => 'Really Long Text…')

choose('A Radio Button')

choose("radio\_group\_selector"), option: "Option 5"

check('A Checkbox')

uncheck('A Checkbox')

attach\_file('Image', '/path/to/image.jpg')

select('Option', :from => 'Select Box')

unselect('Option', from: select\_box)

find("#select\_id").select("value")

#=scoping=

within(:xpath, "//li[@id='employee']") do

fill\_in 'Name', :with => 'Jimmy'

end

within("li#employee") do

fill\_in 'Name', :with => 'Jimmy'

end

within\_fieldset('Employee') do

fill\_in 'Name', :with => 'Jimmy'

end

within\_table('Employee') do

fill\_in 'Name', :with => 'Jimmy'

end

#=Querying=

page.has\_xpath?('//table/tr')

page.has\_css?('table tr.foo')

page.has\_content?('foo')

page.should have\_xpath('//table/tr')

page.should have\_css('table tr.foo')

page.should have\_content('foo')

page.should have\_no\_content('foo')

expect(page).to have\_selector 'foobar'

find\_field('First Name').value

find\_link('Hello').visible? #false, finds only visible

find\_button('Send').click

find('//table/tr').click

all('a').each { |a| a[:href] }

#=Find actions=

find("input.file").attach\_file

find("input.checkbox").check

find("input.select").choose

find(".button").click\_button

find(".link").click\_link

find(".link").click\_link\_or\_button

find(".link").click\_on

find(".link").click

find("input.text").fill\_in(:with => 'Jimmy')

find("input.select").select

find("input.checkbox").uncheck

find("input.select").unselect

find("input.select").unselect\_option(option: "Option 5")

find("input.checkbox").checked?

find(".button").disabled?

find(".link").hover

find("input.select").selected?

find("input.text").value

find(".text").text

find(".link").visible?

#=Scripting=

result = page.evaluate\_script('4 + 4')

periods = page.evaluate\_script("$('#MainContent\_dd').map(function() { return $(this).text() }).get()")

#=Debugging=

save\_and\_open\_page

screenshot\_and\_open\_image # with the capybara-screenshot gem

#=Asynchronous JavaScript=

click\_link('foo')

click\_link('bar')

page.should have\_content('baz')

page.should\_not have\_xpath('//a')

page.should have\_no\_xpath('//a')

using\_wait\_time 5 do

# assertions

end

Capybara automatically waits for asynchronous operations to complete. When you try to find an element that isn't on the page, it waits and retries until it is there, or a timeout duration elapses. The wait time is defined at**Capybara.default\_wait\_time**

**Here are the methods that waits:**

* find(selector), find\_field, find\_link
* within(selector)(scoping) \*has\_selector?/has\_no\_selector? & assertions
* form & link actions
  + click\_link/button
  + fill\_in
  + check/uncheck, select, choose

**Here are the methods that doesn't wait:**

* visit
* current\_path
* all (selector)
* first(selector)
* execute\_script
* simple accessors: text, value, title, etc.

## Server

Rails server

If have error that a server is already running

ps x | grep rails -> grab ‘rails’ in the list of all processes running (ps x)

spot the id on the left for the rails process

kill *id*, or kill -9 *id*  to force quit

## Questions from homework

W6\_d3\_genres\_songs\_albums

I am confused as to when to create a join table for a many-many relationship. I did it for the song-album relationship, just like what we did in class.

But then I created a many-to-many relationship between albums and genres, not using a join table, but simply saying album has\_many genres, through songs, and vice-versa.

It works. Is it only working because songs has a join-table with both values, making songs an adequate intermediary?

I wanted to list all the genres in a main menu accessible in ALL views. How can I define that? I tried to define an @genre variable in the application controller, hoping it would be picked up in the layout view, but it didn’t work.

Question: when I do Album.first.genres I get an array with the genres of each song in Album.first, and therefore with lots of repetitions.

I can of course do .uniq to remove duplicates, but I wonder why it is so. Bearing in mind that there is no join table between ablum and genres, only a mutual has\_many through songs

Are we going to learn how to navigate multiple controllers?

EG if I want to create a new song, and in the form to do so I declare an album not currently in the database, can I have an if statement to add this album and link it with the song?

Are we going to learn how to use the scss files?