Ruby on Rails  
Text with Directions, References and Code

**Lecture 16: Text with Directions References and Code**

Ruby resource:

[www.ruby-lang.org](http://www.ruby-lang.org/)

To view current directory:

Pwd

To create a directory:

mkdir directoryname

To move into a directory:

cd directoryname

\*\*This will only work if the directoryname is listed as one of the folders within the current directory

To list all directories and files within current directory:

Ls

To list all directories and files within current directory including hidden files:

ls –a

To move out of a directory (1 level up the tree):

cd ..

helloworld.rb contents:

puts "Hello World!"

To run helloworld.rb from the command line:

ruby helloworld.rb

To create a variable called hello and print the value of hello:

hello = "Hello World!"

puts hello

The following is a method:

def methodname

#contents of method

End

Content of helloworldmethod.rb

def hello

puts "Hello World!"

end

Then call the method by simply naming it:

hello

Updated hello method:

def hello(anything)

puts anything

end

To call this method:

hello "My name is mashrur"

hello "Welcome to the complete ruby on rails developer course"

hello "Check it out this prints anything I want"

**Lecture 18 Working with Strings – Text with Directions, References and Code**

To start irb console:

irb

String concat structure:

String1 + String2

String1 + " " + String2 + String3

To find out what class the object belongs to:

Variable\_name.class

1.class

To find out methods available to an object:

objectname.methods

Some methods that can be used on objects:

objectname.nil?

objectname.empty?

objectname.length

objectname.reverse

String interpolation (remember has to be within double quotes):

name = "Mashrur"

"My name is #{name}"

\*\*This will print out the value of name in the line "My name is...."

To escape the evaluation of #{variablename} within a String, prepent with a \:

\#{variablename}

To get input from the command line use the following method:

gets.chomp

To assign the input to a variable so it can be referenced later on:

variablename = gets.chomp

**Lecture 21 Working with Numbers – Text with Directions, References and Code**

To add and display the value of 1 + 2:

puts 1 + 2

Different operations:

1 + 2

1 \* 2

1 / 2

1 - 2

1 % 2

To indicate a number is a float instead of an integer include a . in the number:

20 is an integer, 20.0 is a float

or

20.to\_f

Methods you can use:

object.odd?

22.odd?

object.even?

22.even?

Comparisons:

a == b

1 == 2

3 == 3

5 < 2

2 <= 5

5 > 2

5 && 6

5 || 6

Generate a random number between 0 and less than 10:

rand(10)

To convert an string object to integer:

objectname.to\_i

"5".to\_i

To convert an object to string:

objectname.to\_s

5.to\_s

**Lecture 24 Methods and Branching – Text with Directions, References and Code**

Structure of an if condition:

if #condition

#execute logic

End

#variant

if #condition

#execute logic

else

#execute different logic

End

#variant

if #condition

#execute logic

elsif #different condition

#execute logic 2

else

#execute logic 3

End

Code worked on:

def multiply(first\_number, second\_number)

first\_number.to\_f \* second\_number.to\_f

end

def divide(first\_number, second\_number)

first\_number.to\_f / second\_number.to\_f

end

def subtract(first\_number, second\_number)

second\_number.to\_f - first\_number.to\_f

end

def mod(first\_number, second\_number)

first\_number.to\_f % second\_number.to\_f

end

puts "What do you want to do? 1) multiply 2) divide 3) subtract 4) find remainder"

prompt = gets.chomp

puts "Enter in your first number"

first\_number = gets.chomp

puts "Enter in your second number"

second\_number = gets.chomp

if prompt == '1'

puts "You have chosen to multiply #{first\_number} with #{second\_number}"

result = multiply(first\_number, second\_number)

elsif prompt == '2'

puts "You have chosen to divide"

result = divide(first\_number, second\_number)

elsif prompt == '3'

puts "You have chosen to subtract"

result = subtract(first\_number, second\_number)

elsif prompt == '4'

puts "You have chosen to find the remainder"

result = mod(first\_number, second\_number)

else

puts "You have made an invalid choice"

end

puts "The result is #{result}"

**Lecture 26 Arrays and Iterators – Text with Directions, References and Code**

Array, created by including elements within square brackets:

a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

The index for an array starts with 0, so in the array above a[0] is 1

Some methods you can use on arrays:

arrayname.empty?

arrayname.include?(itemname)

arrayname.reverse

arrayname.reverse! # use ! at the end to change the original array

arrayname.shuffle

arrayname.push(30) # will append new element 30 to the end array

arrayname << 25 # << known as shovel operator will also append new element to the end of the array

arrayname.unshift("someelement") # will add element "some element" to the beginning of the array

arrayname.pop # will remove the last element of the array and return 1

arrayname.uniq # will remove all the duplicates and display (will not change the original array)

arrayname.uniq! # will remove all the duplicates in the original array

A range:

(0..25).to\_a

will create an array with elements from value 0 to 25

(0..99).to\_a.shuffle!

will create an array with elements from value 0 to 99 in random order

To loop through an array named y using the .each method and print out the value of each element:

y.each { |i| puts i }

In plain terms: For each element i in array y print the value of i

To execute iteration through an array called names using a block:

names.each do |randomvariablename| # starts the do block

puts "Hello #{randomvariablename}" # executes code for each element

end # ends the do block

To capitalize (or use another method) on each element of array called names:

names.each { |randomvariablename| puts "Hello #{randomvariablename.capitalize}" }

Using the select method to pickup all the odd numbers from an array y:

y.select { |number| number.odd? } # selects the value and returns it only if the condition is met

To join the elements of an array named p:

p.join

To join the elements of an array with space in between each element:

p.join(" ")

To join the elements of an array with dash in between each element:

p.join("-")

**Lecture 28 Hashes – Text with Directions, References and Code**

To create a hash called my\_details include the elements within { }:

my\_details = {'name' => 'mashrur', 'favcolor' => 'red'}

To access the value and notify me what favcolor is:

my\_details["favcolor"]

Alternate syntax to create key, value pairs in hash:

myhash = {a: 1, b: 2, c: 3, d: 4}

To access the value for key c above:

myhash[:c]

To add a key,value pair to the hash above:

myhash[:d] = 7

myhash[:name] = "Mashrur"

To delete a key,value pair simply delete the key:

myhash.delete(:d)

To iterate through a hash using .each method and print out value:

myhash.each { |somekey, somevalue| puts somevalue }

To iterate through a hash using .each method and print out both key and value in friendly format:

myhash.each { |somekey, somevalue| puts "The key is #{somekey} and the value is #{somevalue}" }

To iterate through and delete a items from a hash based on a condition (in the condition below if the value is greater than 3:

myhash.each { |k, v| myhash.delete(k) if v > 3 }

Use select method to display items only if value of the item is odd

myhash.select { |k, v| v.odd? }

**Lecture 30 Ruby Style Guide – Text with Directions, References and Code**

Link for style guide:

<https://github.com/bbatsov/ruby-style-guide>

Soft tabs: 2 spaces, convert tabs to spaces

class names: CamelCase

ThisIsCamelCase

file names: snake\_case

this\_is\_snake\_case.rb

**Lecture 34 Solution: Area Code Homework**

dial\_book = {

"newyork" => "212",

"newbrunswick" => "732",

"edison" => "908",

"plainsboro" => "609",

"sanfrancisco" => "301"

}

def get\_city\_names(somehash)

somehash.each { |k, v| puts k }

end

def get\_area\_code(somehash, key)

somehash[key]

end

loop do

puts "Do you want to lookup an area code based on a city name?(Y/N)"

answer = gets.chomp

if answer != "Y"

break

end

puts "Which city do you want the area code for?"

get\_city\_names(dial\_book)

puts "Enter your selection"

prompt = gets.chomp

if dial\_book.include?(prompt)

puts "The area code for #{prompt} is #{get\_area\_code(dial\_book, prompt)}"

else

puts "You entered a city name not in the dictionary"

end

end

**Lecture 37 Introduction to Object Oriented Programming – Text with Directions, References and Code**

Object-oriented programming (OOP) is a programming paradigm that uses objects and their interactions to design and program applications

- Allows the program to block off areas of code that perform certain tasks independently of other areas in the application.

*Encapsulation* - concept of blocking off areas of code and not making it available to the rest of the program

*Abstraction* - is simplifying a complex process of a program, an enterprise software solution for example by modeling classes appropriate for it

*Inheritance* - is used where a class inherits the behavior of another class, referred to as the superclass

*Polymorphism* - is when a class inherits the behaviors of another class, but has the ability to not inherit everything and change some of it’s inherited behaviors. For example to write a method that does something differently from the inherited method

*Classes* - It is a blueprint that describes the state and behavior that the objects of the class all share. A class can be used to create many objects. Objects created at runtime from a class are called instances of that particular class.

Example of a user class

class User

attr\_accessor :name, :email

def initialize(name, email)

@name = name

@email = email

end

def run

puts "Hey I'm running"

end

def self.identify\_yourself

puts "Hey I am a class method"

end

end

user = User.new("mashrur", "mashrur@example.com")

user.run

User.identify\_youself # to run this class method you don't need an instance of user you can directly call the

# class User

**Lecture 39 Inheritance and Modules – Text with Directions, References and Code**

user class with destructable module:

module Destructable

def destroy(anyobject)

puts "I will destroy the object"

end

end

class User

include Destructable

attr\_accessor :name, :email

def initialize(name, email)

@name = name

@email = email

end

def run

puts "Hey I'm running"

end

def self.identify\_yourself

puts "Hey I am a class method"

end

end

class Buyer < User

def run

puts "Hey I'm not running and I'm in buyer class"

end

end

class Seller < User

end

class Admin < User

end

**Lecture 41 Ruby Project with JSON – Text with Directions, References and Code**

user\_permissions\_template.json:

{"accounts": "read",

"policies": "write",

"users": "write"}

user.rb:

require 'json'

class User

attr\_accessor :email, :name, :permissions

def initialize(\*args)

@email = args[0]

@name = args[1]

@permissions = User.permissions\_from\_template

end

def self.permissions\_from\_template

file = File.read 'user\_permissions\_template.json'

JSON.load(file, nil, symbolize\_names: true)

end

def save

self\_json = {email: @email, name: @name, permissions: @permissions}.to\_json

open('users.json', 'a') do |file|

file.puts self\_json

end

end

end

runner.rb:

require 'pp'

require\_relative 'user'

user = User.new 'joe@example.com', 'joe'

pp user

user.save

**Lecture 43 Ruby on Rails Kickoff– Text with Directions, References and Code**

Rails overview:

guides.rubyonrails.org

rubyonrails.org

weblog.rubyonrails.org

Creator of Rails: David Heinemeier Hansson

david.heinemeierhansson.com

Ruby meetups:

ruby.meetup.com

Code repository:

github.com

Devise gem code repository:

github.com/plataformatec/devise

Gems for ruby and ruby on rails:

rubygems.org

Bootstrap gem code repository:

github.com/twbs/bootstrap-sass

**Lecture 45 Model, View, Controller and Rails App Structure – Text with Directions, References and Code**

To create a new directory called rails\_projects:

mkdir rails\_projects

To start a new rails application called test\_app:

rails new test\_app

MVC - Model, View, Controller

General flow of Rails application:

-> Request made at browser

-> Request received at router of rails application

-> Request routed to appropriate controller

-> Controller either renders a view template or communicates with model

-> Model communicates with database

-> Model sends back information to controller

-> Controller renders view

**Lecture 48 Solution to Homework Assignment**

Step 1: Ensure you are in the code/rails\_projects directory and not in an existing rails application directory like code/rails\_projects/test\_app

Step 2: Create a new rails application using the command ->

rails new alpha-blog

Step 3: Change directory into the new application using ->

cd alpha-blog

Step 4: Run the rails server using the command ->

rails s -b 0.0.0.0

Step 5: Preview the application and ensure the Rails welcome page shows up by clicking on preview and then Port 3000 (Default) on the top menu of your nitrous IDE

Step 6: Append a /pages/home to the URL and you'll receive a routing error

Step 7: Go to config/routes.rb file and add the following two routes ->

get 'pages/home', to: 'pages#home'

get 'pages/about', to: 'pages#about'

Step 8: Now if you reload the page after saving the routes file you'll receive uninitialized constant pages controller error

Step 9: Under app/controllers create a pages\_controller.rb file by right clicking on controllers folder and selecting the 'New File' option

Step 10: Fill in the pages\_controller.rb file with the following code:

class PagesController < ApplicationController

def home

end

def about

end

end

Step 11: Now if you reload any of the pages you'll get a missing template error, so create a folder under views called pages (right click on the views folder and select 'New Folder')

Step 12: Under pages folder, create two files named home.html.erb and about.html.erb

Step 13: Fill in these two files with HTML code

Step 14: To upload an image to your app that you want to reference in your code, go to app/assets/images and right click on images, then select the upload files to images option. Select the image you want to upload. Once uploaded check the images folder and note the name of your file (for example profileimage.png). Then to show this image in your HTML page and to link it to another page, put in the following code (this is if the file you're referencing is profileimage.png) ->

<%= link\_to image\_tag('profileimage.png'), 'www.mashrurhossain.com' %>

The image tag is for the image, and the link\_to turns the image to a link and then you provide the link you want it to go to after the comma

If you simply want to display an image without a link, put in the following code ->

<%= image\_tag 'profileimage.png' %>

Step 15: Now if you go to either the pages/home or pages/about link (appended to your nitrous preview page URL) you will get the home and/or about pages

Step 16: Enter in the path to the about page in the home.html.erb and the homepage in the about.html.erb pages as below ->

In the home.html.erb page

<%= link\_to 'About', pages\_about\_path %>

In the about.html.erb page

<%= link\_to 'Home', pages\_home\_path %>

Ensure you have the appropriate links by checking with rake routes

Step 17: Click the links in the refreshed browser pages to ensure they work

Good luck!

**Lecture 50 Root Route, Git and Version Control – Text with Directions, References and Code**

To set root route to pages controller home:

Navigate to config/routes.rb file and enter in the following code ->

root 'pages#home'

The reference to the root path within the application code would be root\_path

Git references:

git-scm.com

git-scm.com/book/en/v2

gitref.org

First time (required only once to configure git for your environment):

git config --global user.name "Your name"

git config --global user.email "Your email"

replace Your name and Your email above with your actual name and email address which you want shown on repo

To display git config settings:

git config --list

Some useful git commands:

To initialize a git repository for your application (do this from within the application directory) ->

git init

To add/track all files ->

git add -A

To commit changes/updates/additions to repository ->

git commit -m "A useful message to help remember details of commit"

To check current state of file updates with already committed code in repo ->

git status

To reject latest changes ->

git checkout -f

Site to create free online code repositories:

github.com

bitbucket.org

**Lecture 52 Setup GitHub Repository – Text with Directions, References and Code**

To display your public SSH key:

cat ~/.ssh/id\_rsa.pub

When creating github repo for application ensure you click the SSH button then push existing repo:

git remote add origin git@github.com:yourgithubaccountname/alpha-blog.git

git push -u origin master # Remember you only need to use this command the first time

To view remotes setup in your environment (from your app directory):

git remote -v

For future pushes to repository:

git push

To deploy app to production, we'll use heroku in this course, link to heroku:

[www.heroku.com](http://www.heroku.com/)

**Lecture 54 Production Deploy – Text with Directions, References and Code**

Prepartion for heroku deploy:

- Remove sqlite3 gem from top of application to within group :development, :test do block

- Create a group production ->

group :production do

gem 'pg'

gem 'rails\_12factor'

end

- Save Gemfile

- Run bundle install --without production to update Gemfile.lock file

- Commit your changes to git repo ->

git add -A

git commit -m "Make app production ready"

Command to install heroku toolbelt to your nitrous environment:

wget -qO- <https://toolbelt.heroku.com/install-ubuntu.sh> | sh

Check heroku:

heroku –v ???

heroku version

heroku # for list of common heroku commands

From your app directory:

To login to your heroku account from your nitrous env ->

heroku login

To add your SSH key to your heroku account so you don't have to use username and password everytime ->

heroku keys:add

To create a new production version of your app hosted in heroku ->

heroku create

To push your application code to heroku (deploy your app) ->

git push heroku master

Ensure you have committed all your local changes to your git repo prior to pushing to heroku by checking git status

To change the name of your application ->

heroku rename newnameofyourapp

replace newnameofyourapp above with the name you'd like to give your app

Your app will then be accessible from the following browser URL ->

newnameofyourapp.herokuapp.com

**Lecture 56 CRUD and Scaffold – Text with Directions, References and Code**

Query language to communicate with database: SQL (Structured Query Language)

CRUD actions:

C - Create

R - Read

U - Update

D – Delete

Scaffold generator command to create Article model, articles controller, views for articles and migration file to create articles table:

rails generate scaffold Article title:string description:text

Command to see routes present for your app:

rake routes

The line resources :articles in the config/routes.rb file provides the following routes:

- index of articles (listing)

- new article

- create article

- edit article

- update article (put and patch)

- show article

- delete article

From UI perspective ->

- index lists all the articles in the db of the app

- new article deals with the form to enter in new article details

- create handles the submission of the items in the new article form

- edit article deals with the form to enter in edit details for an existing article

- update article deals with the submission of the edited details

- show article displays an individual article based on selection

- delete article deletes an article from the db

**Lecture 59 Introduction to Models, Migrations and Rails Console – Text with Directions, and References**

Model name: Article, class: Article -> Capitalized A and singular

File name: article.rb -> singular and all lowercase

Controller file name: articles\_controller.rb, class: ArticlesController -> camel case class name, snake case file name both plural

Views folder: articles

Table name: articles -> plural of model

Model name: User, class: User -> Capitalized U and singular

File name: user.rb -> singular and all lowercase

Controller file name: users\_controller.rb, class: UsersController -> camel case class name, snake case file name both plural

Views folder: users

Table name: users -> plural of model

To generate a migration to create a table (in this example articles):

rails generate migration create\_articles

To add attributes for the table in the migration file, add the following inside create\_table block:

t.string :title

t.text :description

t.timestamps

To run the migration file and create the articles table:

rake db:migrate (I like this one better)

OR

bundle exec rake db:migrate

To rollback a migration (undo the last migration):

rake db:rollback

To add a column (example: created\_at column) to the articles table:

rails generate migration add\_created\_at\_to\_articles

Then within the def change method in the migration file:

add\_column :articles, :created\_at, :datetime

To add a different column (example: name) to a users table:

rails generate migration add\_name\_to\_users

Then within the def change method in the migration file:

add\_column :users, :name, :string

In the above two adding column methods, the first argument is the name of the table, second is the attribute name and third is the type

To create a model file for Article:

- In the app/models folder create a file called article.rb

- Fill it in with the following ->

class Article < ActiveRecord::Base

end

To start the rails console:

rails console

To test connection to the articles table:

Article.all # classname.all will list all the articles in the articles table

Then simply type in Article (classname) to view the attributes

To create a new article with attributes title and description:

article = Article.new(title: "This is a test title", description: "This is a test description")

article.save

OR

article = Article.new

article.title = "This is a test title"

article.description = "This is a test description"

article.save

Another method to do the same:

article = Article.create(title: "This is a test title", description: "This is a test description") # This will hit the table right away without needing the article.save line

**Lecture 61 Edit, Delete, and Validations – Text with Directions, References and Code**

To find an article with id 2 and edit it's title:

article = Article.find(2) # Here assumption is article with id of 2 was being looked for

article.title = "This is an edited title"

article.save

To delete an article, example with id 5:

article = Article.find(5)

article.destroy

To add validations presence and length validations to article model for title and description:

class Article < ActiveRecord::Base

validates :title, presence: true, length: {minimum: 3, maximum: 50}

validates :description, presence: true, length: {minimum: 10, maximum: 300}

end

To find errors in article object while saving (if it's rolled back):

article.errors.any?

article.errors.full\_messages

**Lecture 63 Create New Articles from UI – Text with Directions and Code**

In the config/routes.rb file add the following line to add all the routes for articles:

resources :articles

This will add the following routes:

routes path HTTP verb link controller#action

articles index articles GET /articles articles#index

new article new\_article GET /articles/new articles#new

create article POST /articles articles#create

edit article edit\_article GET /articles/:id articles#edit

update article PATCH /articles/:id articles#update

show article article GET /articles/:id articles#show

delete article DELETE /articles/:id articles#destroy

To create articles controller with a new action, under app/controllers create a file named articles\_controller.rb (snake case):

class ArticlesController < ApplicationController

def new

@article = Article.new

end

end

To create a view, under app/views create a folder named articles and within it create a file named new.html.erb then fill in the following:

<h1>Create an article</h1>

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

<p>

<%= f.label :title %><br/>

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

</p>

<p>

<%= f.label :description %><br/>

<%= f.text\_area :description %>

</p>

<p>

<%= f.submit %>

</p>

<% end %>

Create action and private article\_params method for string parameters in the articles controller (Note: This is not complete):

def create

@article = Article.new(article\_params)

@article.save

redirect\_to article\_path(@article)

end

private

def article\_params

params.require(:article).permit(:title, :description)

end

**Lecture 65 Complete New and Show Actions – Text with Directions and Code**

Completed create action in articles controller:

def create

@article = Article.new(article\_params)

if @article.save

flash[:notice] = "Article was successfully created"

redirect\_to article\_path(@article)

else

render 'new'

end

end

Flash message code added to application.html.erb under app/views/layouts folder (right under <body> and above <%= yield %>:

<% flash.each do |name, msg| %>

<ul>

<li><%= msg %></li>

</ul>

<% end %>

Code added to display errors in the new.html.erb template under app/views/articles folder:

<% if @article.errors.any? %>

<h2>The following errors prevented the article from getting created</h2>

<ul>

<% @article.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

<% end %>

To create the show action, add the following show method to articles\_controller.rb file:

def show

@article = Article.find(params[:id])

end

To create the show view, add a show.html.erb file under the app/views/articles folder and fill in the code:

<h1>Showing selected article</h1>

<p>

Title: <%= @article.title %>

</p>

<p>

Description: <%= @article.description %>

</p>

**Lecture 67 Edit Articles – Text with Directions and Code**

Route for editing articles takes the form ->

/articles/:id/edit

Edit action in the articles controller:

def edit

@article = Article.find(params[:id])

end

Update action in the articles controller:

def update

@article = Article.find(params[:id])

if @article.update(article\_params)

flash[:notice] = "Article was successfully updated"

redirect\_to article\_path(@article)

else

render 'edit'

end

end

To create edit template, create a file named edit.html.erb under the app/views/articles folder and fill in the following code:

<h1>Edit existing article</h1>

<% if @article.errors.any? %>

<h2>The following errors prevented the article from getting created</h2>

<ul>

<% @article.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

<% end %>

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

<p>

<%= f.label :title %><br/>

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

</p>

<p>

<%= f.label :description %><br/>

<%= f.text\_area :description %>

</p>

<p>

<%= f.submit %>

</p>

<% end %>

**Lecture 69 List Articles and Layout Links – Text with Directions and Code**

To create listings index first add the index action to acticles controller:

def index

@articles = Article.all

end

Create the view file index.html.erb under app/views/articles folder:

<h1>Listing all articles</h1>

<p>

<%= link\_to "Create new article", new\_article\_path %>

</p>

<table>

<tr>

<th>Title</th>

<th>Description</th>

</tr>

<% @articles.each do |article| %>

<tr>

<td><%= article.title %></td>

<td><%= article.description %></td>

<td><%= link\_to 'Edit', edit\_article\_path(article) %></td>

<td><%= link\_to 'Show', article\_path(article) %></td>

</tr>

<% end %>

</table>

Then update the views with links ->

show.html.erb:

<h1>Showing selected article</h1>

<p>

Title: <%= @article.title %>

</p>

<p>

Description: <%= @article.description %>

</p>

<%= link\_to "Edit this article", edit\_article\_path(@article) %> |

<%= link\_to "Back to articles listing", articles\_path %>

Add the back to articles listing path to the bottom of both new.html.erb and edit.html.erb pages:

<%= link\_to "Back to articles listing", articles\_path %>

**Lecture 71 Destroy and Partials – Text with Directions and Code**

Add this link to the homepage (root route) so you can access the blog from the homepage:

<%= link\_to "Alpha Blog", articles\_path %>

Under app/views/layouts folder create a \_messages.html.erb file (messages partial) and remove the following code from application.html.erb to this file:

<% flash.each do |name, msg| %>

<ul>

<li><%= msg %></li>

</ul>

<% end %>

In place of this code in the application.html.erb add the following code:

<%= render 'layouts/messages' %>

Create a file under app/views/articles folder called \_form.html.erb and fill it in with the following code (copied from the new or edit.html.erb page):

<% if @article.errors.any? %>

<h2>The following errors prevented the article from getting created</h2>

<ul>

<% @article.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

<% end %>

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

<p>

<%= f.label :title %><br/>

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

</p>

<p>

<%= f.label :description %><br/>

<%= f.text\_area :description %>

</p>

<p>

<%= f.submit %>

</p>

<% end %>

<%= link\_to "Back to articles listing", articles\_path %>

Then remove the code above from both new.html.erb and edit.html.erb files and in it's place add the following code:

<%= render 'form' %>

To add the destroy method, first add the following to the articles controller:

def destroy

@article = Article.find(params[:id])

@article.destroy

flash[:notice] = "Article was successfully deleted"

redirect\_to articles\_path

end

Then in the index.html.erb (listings page) add the following link as one of the <td> items under the show article link:

<td><%= link\_to 'Delete', article\_path(article), method: :delete, data: {confirm: "Are you sure?"} %></td>

**Lecture 73 Deploy and Wrap Section – Text with Directions and Code**

Completed articles controller after adding before\_action:

class ArticlesController < ApplicationController

before\_action :set\_article, only: [:edit, :update, :show, :destroy]

def index

@articles = Article.all

end

def new

@article = Article.new

end

def edit

end

def create

@article = Article.new(article\_params)

if @article.save

flash[:notice] = "Article was successfully created"

redirect\_to article\_path(@article)

else

render 'new'

end

end

def update

if @article.update(article\_params)

flash[:notice] = "Article was successfully updated"

redirect\_to article\_path(@article)

else

render 'edit'

end

end

def show

end

def destroy

@article.destroy

flash[:notice] = "Article was successfully deleted"

redirect\_to articles\_path

end

private

def set\_article

@article = Article.find(params[:id])

end

def article\_params

params.require(:article).permit(:title, :description)

end

end

To deploy app to production ->

- Ensure you committed your code to your git repo

- Ensure your gemfile has sqlite3 in group dev and pg and rails\_12factor in group production

- Command to deploy to heroku: git push heroku master

- Once deployed run the following to run your migration files in production: heroku run rake db:migrate

Go to your-app-name.herokuapp.com and verify deploy

To learn CSS go to:

w3schools.com/css/default.asp

To learn about bootstrap, go to:

getbootstrap.com

**Lecture 76 Install Bootstrap – Text with Directions and Code**

Bootstrap sass gem page:

<https://github.com/twbs/bootstrap-sass>

Add the following gem to your gemfile above the gem 'sass-rails':

gem 'bootstrap-sass', '~> 3.3.5'

To install the gem to your app run:

bundle install --without production

Create a file called custom.css.scss under app/assets/stylesheets folder

Add the following lines to the file:

@import "bootstrap-sprockets";

@import "bootstrap";

Add the following line to your application.js file in the app/assets/javascripts folder under the line that says //= require jquery\_ujs:

//= require bootstrap-sprockets

Make a commit to your git repo

**Lecture 79 Complete Homepage– Text with Directions and Code**

After completion of the prior video, below is what the application.html.erb looks like:

<!DOCTYPE html>

<html>

<head>

<title>AlphaBlog</title>

<%= stylesheet\_link\_tag 'application', media: 'all', 'data-turbolinks-track' => true %>

<%= javascript\_include\_tag 'application', 'data-turbolinks-track' => true %>

<%= csrf\_meta\_tags %>

</head>

<body>

<%= render 'layouts/navigation' %>

<%= render 'layouts/messages' %>

<div class="container">

<%= yield %>

</div>

<%= render 'layouts/footer' %>

</body>

</html>

The image to be added as background for the jumbotron should be added in the app/assets/images folder

Below is what the custom.css.scss file looks like:

$navbar-default-bg: black;

@import "bootstrap-sprockets";

@import "bootstrap";

#logo {

float: left;

font-size: 1.7em;

color: #fff;

text-transform: uppercase;

letter-spacing: -1px;

font-weight: bold;

}

#logo:hover {

color: #fff;

text-decoration: none;

}

.center {

text-align: center;

}

.jumbotron {

background-image: asset-url('new\_cover\_page.png');

background-size: cover;

height: 550px;

}

.jumbotron h1 {

color: #fff;

text-align: center;

margin-bottom: 30px;

letter-spacing: -1px;

font-weight: bold;

}

.btn-xlarge {

font-size: 1.7em;

background-color: black;

}

footer {

margin-top: 45px;

padding-top: 5px;

border-top: 1px solid #eaeaea;

color: #777;

}

footer a:hover {

color: #222;

}

footer small {

float: left;

}

footer ul {

float: right;

list-style: none;

}

footer ul li {

float: left;

margin-left: 15px;

}

Below is what the \_navigation.html.erb file looks like:

<nav class="navbar navbar-default">

<div class="container-fluid">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<%= link\_to "Alpha blog", root\_path, class: "navbar-brand", id: "logo" %>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav">

<li><%= link\_to "Articles", articles\_path %></li>

<li><a href="#">Link</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Actions <span class="caret"></span></a>

<ul class="dropdown-menu">

<li><%= link\_to "New Article", new\_article\_path %></li>

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">One more separated link</a></li>

</ul>

</li>

</ul>

<form class="navbar-form navbar-left" role="search">

<div class="form-group">

<input type="text" class="form-control" placeholder="Search">

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>

<ul class="nav navbar-nav navbar-right">

<li><a href="#">Link</a></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Dropdown <span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a href="#">Action</a></li>

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

</ul>

</li>

</ul>

</div><!-- /.navbar-collapse -->

</div><!-- /.container-fluid -->

</nav>

Below is what the \_footer.html.erb file looks like:

<div class="container">

<footer class="footer">

<small>

Copyright © <a href="enter in a link here">The Complete Ruby on Rails Developer</a>

by <a href="enter in a link here">Mashrur Hossain</a>

</small>

<nav>

<ul>

<li><%= link\_to "About", about\_path %></li>

</ul>

</nav>

</footer>

</div>

**Lecture 81 Style Form Template– Text with Directions and Code**

In the new and edit.html.erb files in the app/views/articles folder add the align center to h1 tags:

<h1 align="center">Edit existing article</h1>

<h1 align="center">Create new article</h1>

Below is the current state of the \_form.html.erb partial after styling the form portion:

<% if @article.errors.any? %>

<h2>The following errors prevented the article from getting created</h2>

<ul>

<% @article.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

<% end %>

<div class='row'>

<div class='col-xs-12'>

<%= form\_for(@article, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :title %>

</div>

<div class="col-sm-8">

<%= f.text\_field :title, class: "form-control", placeholder: "Title of article", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :description %>

</div>

<div class="col-sm-8">

<%= f.text\_area :description, rows: 10, class: "form-control", placeholder: "Body of article" %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

</div>

</div>

**Lecture 83 Style Messages – Text with Directions and Code**

In your articles\_controller.rb file change the flash[:notice] to be flash[:success], there should be 3 such occurances in that file

Update your \_messages.html.erb file with the following code:

<div class="row">

<div class="col-xs-10 col-xs-offset-1">

<% flash.each do |name, msg| %>

<div class='alert alert-<%="#{name}" %>'>

<a href="#" class="close" data-dismiss="alert">×</a>

<%= content\_tag :div, msg, :id => "flash\_#{name}" if msg.is\_a?(String) %>

</div>

<% end %>

</div>

</div>

Update your \_form.html.erb at the top to remove the error portion and replace it with the following:

<%= render 'shared/errors', obj: @article %>

Create a folder called shared under app/views/ folder. Then within the shared folder file named \_errors.html.erb to create the errors partial, then fill it in with the following code:

<% if obj.errors.any? %>

<div class="row">

<div class="col-xs-8 col-xs-offset-2">

<div class="panel panel-danger">

<div class="panel-heading">

<h2 class="panel-title">

<%= pluralize(obj.errors.count, "error") %>

prohibited this article from being saved:

</h2>

<div class="panel-body">

<ul>

<% obj.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

</div>

</div>

</div>

</div>

<% end %>

**Lecture 85 Style Show View – Text with Directions and Code**

Articles show page, show.html.erb:

<h2 align="center">Title: <%= @article.title %></h2>

<div class="well col-xs-8 col-xs-offset-2">

<h4 class="center description"><strong>Description:</strong></h4>

<hr>

<%= simple\_format(@article.description) %>

<div class="article-actions">

<%= link\_to "Edit this article", edit\_article\_path(@article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(@article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

<%= link\_to "View all articles", articles\_path, class: "btn btn-xs btn-success" %>

</div>

</div>

Add the following classes to custom.css.scss page:

.article-actions {

border-top: 1px solid #eaeaea;

padding-top: 5px;

}

.description {

margin-top: 0;

}

**Lecture 87 Style List View – Text with Directions and Code**

Add styling to the index.html.erb page:

<h1 align="center">Listing all articles</h1>

<% @articles.each do |article| %>

<div class="row">

<div class="col-xs-8 col-xs-offset-2">

<div class="well well-lg">

<div class="article-title">

<%= link\_to article.title, article\_path(article) %>

</div>

<div class="article-body">

<%= truncate(article.description, length: 100) %>

</div>

<div class="article-actions">

<%= link\_to "Edit this article", edit\_article\_path(article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

</div>

</div>

</div>

</div>

<% end %>

Add the following code to the custom.css.scss page:

.article-title {

font-weight: bold;

font-size: 1.5em;

}

.article-body {

border-top: 1px solid #eaeaea;

padding-top: 15px;

padding-bottom: 15px; }

**Lecture 89 Deploy to Production – Text with Directions and Code**

To deploy to heroku:

git status # To ensure you have no uncommitted changes, if you do commit the code

git push heroku master

To run any un-run migration files in heroku:

heroku run rake db:migrate

Don't forget to post the link to your herokuapp link in the discussion area!

Homework assignment to start section 6:

Go to -> [http://guides.rubyonrails.org/association\_basics.h...](http://guides.rubyonrails.org/association_basics.html)

and read about one-to-many (has\_many) associations

**Lecture 92 Create Users – Text with Directions and Code**

To create a feature branch:  
git checkout -b nameofbranch

To create a feature branch named create-users:  
git checkout -b create-users

To view list of branches:  
git branch

To move to master branch:  
git checkout master

To move to already created feature branch:  
git checkout nameofbranch

Important to remember is to always have master branch deployable to production and work on all new features and additions in feature branches

To create a migration to create users table:

rails generate migration create\_users

Then within the migration file add in the following within the create\_table block to add the username, email and timestamps columns (created\_at and updated\_at):

t.string :username  
t.string :email  
t.timestamps

To run the migration file and create the users table:

rake db:migrate  
  
Add a user.rb model file under app/models folder and fill in the following:  
 class User < ActiveRecord::Base  
 end

Then start rails console and test out connections:

rails console  
User.all  
User  
user = User.create(username: "test", email: "test@example.com")  
user = User.create(username: "test2", email: "test2@example.com")

To grab first user and update their email address:  
 user = User.find(1)  
 OR  
 user = User.first  
user.email = [test3@example.com](mailto:test3@example.com)  
user.save

To destroy user with id = 2:  
user = User.find(2)  
user.destroy

To make a commit of the changes made in the feature branch:

git add –A  
git commit -m "create users table and user model"  
To merge the changes in the feature branch to the master branch first switch to master branch:  
git checkout master  
git merge nameofbranch  
git push to push code to repository

To delete a feature branch that is no longer needed and has been merged to master already:  
git branch -d nameofbranch

To delete a feature branch that is no longer needed but has NOT been merged to master:  
git branch -D nameofbranch

**Lecture 94 Add User Validations – Text with Directions and Code**

Validations for User class:

- username must be present and unique

- email must be present and unique

- validate email format using regex

user.rb model file after validations added:

class User < ActiveRecord::Base

validates :username, presence: true,

uniqueness: { case\_sensitive: false },

length: { minimum: 3, maximum: 25 }

VALID\_EMAIL\_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i

validates :email, presence: true, length: { maximum: 105 },

uniqueness: { case\_sensitive: false },

format: { with: VALID\_EMAIL\_REGEX }

end

**Lecture 96 One to Many Associations – Text with Directions and Code**

To generate a migration to add user\_id column to articles table:

rails generate migration add\_user\_id\_to\_articles

Then within the change method:

add\_column :articles, :user\_id, :integer

Then run the migration file to effect the change:

rake db:migrate

Add the following line to article.rb model file:

belongs\_to :user

Add the following line to user.rb model file:

has\_many :articles

Also add the following line to user.rb model file(this has nothing to do with the association):

before\_save { self.email = email.downcase }

Ensure you have a couple of users created by using the rails console. Then add in 1 line to grab a user to the create action to temporarily hardcode a user to articles:

def create

@article = Article.new(article\_params)

@article.user = User.first

if @article.save

flash[:success] = "Article was successfully created"

redirect\_to article\_path(@article)

else

render 'new'

end

end

Ensure you get rid of the debugger line if you no londer need it within the create action, you can add that line as you need to your actions if you want to pause execution of a request

You can add in the following line to display the debug(params) to your development environment UI output:

(this will be in the app/views/layouts/application.html.erb file under render footer and above </body>)

<%= debug(params) if Rails.env.development? %>

**Lecture 98 Show User Info in Articles – Text with Directions and Code**

Add the following code to the index.html.erb file under app/views/articles folder within the div for article-body and under the article.description:

<div class="article-meta-details">

<small>Created by: <%= article.user.username if article.user%>,

<%= time\_ago\_in\_words(article.created\_at) %> ago,

last updated: <%= time\_ago\_in\_words(article.updated\_at) %> ago</small>

</div>

And then add styling to a new article-meta-details class in the custom.css.scss page under the app/assets/stylesheets folder:

.article-meta-details {

border-top: 1px solid #eaeaea;

margin-top: 15px;

}

**Lecture 100 Add Secure Password – Text with Directions and Code**

To create a migration to add password\_digest column to users table:

rails generate migration add\_password\_digest\_to\_users

Then pull up the migration file and fill in the column details within the def change block:

add\_column :users, :password\_digest, :string

Then save the file and run rake db:migrate to make the change to the table

In the model file (user.rb) add the following method:

has\_secure\_password

In the gemfile add the gem:

gem 'bcrypt', '~> 3.1.7'

Run the following from the command line:

bundle install --without production

Test it out from the rails console by creating a couple of test users and updating password for an existing user

To authenticate and test password for a user, first grab the user:

user = User.last # (or User.find(enter id of user here))

user.authenticate("providecorrectpassword") # This will return the user object

user.authenticate("enterincorrectpassword") # This will return false

**Lecture 102 New User Signup – Text with Directions and Code**

To add the route for the new form add the following two lines to the routes.rb file:

get 'signup', to: 'users#new'

resources :users, except: [:new]

Create a file under app/controllers folder called users\_controller.rb and fill in the following:

class UsersController < ApplicationController

def new

@user = User.new

end

def create

@user = User.new(user\_params)

if @user.save

flash[:success] = "Welcome to the alpha blog #{@user.username}"

redirect\_to articles\_path

else

render 'new'

end

end

private

def user\_params

params.require(:user).permit(:username, :email, :password)

end

end

To create a view, first create a folder called users under the app/views directory, then within it create a new file called new.html.erb and fill in the following code:

<h1 align="center">Signup for Alpha Blog</h1>

<%= render 'shared/errors', obj: @user %>

<div class='row'>

<div class='col-xs-12'>

<%= form\_for(@user, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :username %>

</div>

<div class="col-sm-8">

<%= f.text\_field :username, class: "form-control", placeholder: "Enter username", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :email %>

</div>

<div class="col-sm-8">

<%= f.email\_field :email, class: "form-control", placeholder: "Enter email" %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :password %>

</div>

<div class="col-sm-8">

<%= f.password\_field :password, class: "form-control", placeholder: "Enter password" %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit "Sign up", class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

</div>

</div>

Now test out signing up a couple of users from the UI (sign up a couple of invalid ones too to ensure the validations are working and displaying correctly)

Then you can test them out from the rails console

**Lecture 104 Edit User – Text with Directions and Code**

First add the following two actions in the users\_controller.rb file:

def edit

@user = User.find(params[:id])

end

def update

@user = User.find(params[:id])

if @user.update(user\_params)

flash[:success] = "Your account was updated successfully"

redirect\_to articles\_path

else

render 'edit'

end

end

Then create a \_form.html.erb form partial file under the app/views/users folder and fill it in with the following code:

<%= render 'shared/errors', obj: @user %>

<div class='row'>

<div class='col-xs-12'>

<%= form\_for(@user, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :username %>

</div>

<div class="col-sm-8">

<%= f.text\_field :username, class: "form-control", placeholder: "Enter username", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :email %>

</div>

<div class="col-sm-8">

<%= f.email\_field :email, class: "form-control", placeholder: "Enter email" %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :password %>

</div>

<div class="col-sm-8">

<%= f.password\_field :password, class: "form-control", placeholder: "Enter password" %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit(@user.new\_record? ? "Sign up" : "Update account", class: 'btn btn-primary btn-lg') %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

</div>

</div>

The updated new.html.erb file under app/views/users folder should look like below:

<h1 align="center">Sign up for Alpha Blog</h1>

<%= render 'form' %>

Create a new file under app/views/users called edit.html.erb and fill it in with the following code:

<h1 align="center">Edit your account</h1>

<%= render 'form' %>

Test out creating and editing a couple of users from the browser

**Lecture 106 Show User and Profile Image – Text with Directions and Code**

To have a profile image associated with an email account you control, add one to the site en.gravatar.com. This will be the globally recognized avatar (gravatar for short) associated with that email address.

In the users\_controller.rb file add in the following method:

def show

@user = User.find(params[:id])

end

In the application\_helper.rb file under the app/helpers folder, add the following method:

def gravatar\_for(user, options = { size: 80})

gravatar\_id = Digest::MD5::hexdigest(user.email.downcase)

size = options[:size]

gravatar\_url = "https://secure.gravatar.com/avatar/#{gravatar\_id}?s=#{size}"

image\_tag(gravatar\_url, alt: user.username, class: "img-circle")

end

Under the app/views/articles folder, create a file called \_article.html.erb and fill it in with the following code:

<% obj.each do |article| %>

<div class="row">

<div class="col-xs-8 col-xs-offset-2">

<div class="well well-lg">

<div class="article-title">

<%= link\_to article.title, article\_path(article) %>

</div>

<div class="article-body">

<%= truncate(article.description, length: 100) %>

<div class="article-meta-details">

<small>Created by: <%= article.user.username if article.user%>,

<%= time\_ago\_in\_words(article.created\_at) %> ago,

last updated: <%= time\_ago\_in\_words(article.updated\_at) %> ago</small>

</div>

</div>

<div class="article-actions">

<%= link\_to "Edit this article", edit\_article\_path(article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

</div>

</div>

</div>

</div>

<% end %>

Change the index.html.erb file under the app/views/articles folder to look like below:

<h1 align="center">Listing all articles</h1>

<%= render 'article', obj: @articles %>

Create a new file named show.html.erb under the app/views/users folder and fill in the following code:

<h1 align="center">Welcome to <%= @user.username %>'s page</h1>

<div class="row">

<div class="col-md-4 col-md-offset-4 center">

<%= gravatar\_for @user, size: 150 %>

</div>

</div>

<h4 align="center"><%= @user.username %>'s articles</h4>

<%= render 'articles/article', obj: @user.articles %>

**Lecture 108 Add User Index – Text with Directions and Code**

First in the users\_controller.rb file add the following index action:

def index

@users = User.all

end

Create a file named index.html.erb under the app/views/users folder and fill it in with the following:

<h1 align="center">All Alpha Bloggers</h1>

<div align="center">

<% @users.each do |user| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li><%= link\_to gravatar\_for(user), user\_path(user) %></li>

<li class="article-title">

<%= link\_to user.username, user\_path(user) %>

</li>

<li><small><%= pluralize(user.articles.count, "article") if user.articles %></small></li>

</div>

</div>

</ul>

<% end %>

</div>

Add the following class to the custom.css.scss file:

.listing {

list-style: none;

padding-left: 0;

}

Update the show.html.erb file under the app/views/articles folder and add the following bit of code:

<% if @article.user %>

<ul class="listing">

<div class="row center">

<div class="col-md-4 col-md-offset-4">

<li>Created by:</li>

<li><%= link\_to gravatar\_for(@article.user), user\_path(@article.user) %></li>

<li class="article-title">

<%= link\_to @article.user.username, user\_path(@article.user) %>

</li>

<li><small>

<%= pluralize(@article.user.articles.count, "article") if @article.user.articles %>

</small></li>

</div>

</div>

</ul>

<% end %>

After addition of the code above, the show.html.erb file in the app/views/articles folder should look like below:

<h2 align="center">Title: <%= @article.title %></h2>

<div class="well col-xs-8 col-xs-offset-2">

<% if @article.user %>

<ul class="listing">

<div class="row center">

<div class="col-md-4 col-md-offset-4">

<li>Created by:</li>

<li><%= link\_to gravatar\_for(@article.user), user\_path(@article.user) %></li>

<li class="article-title">

<%= link\_to @article.user.username, user\_path(@article.user) %>

</li>

<li><small>

<%= pluralize(@article.user.articles.count, "article") if @article.user.articles %>

</small></li>

</div>

</div>

</ul>

<% end %>

<h4 class="center description"><strong>Description:</strong></h4>

<hr>

<%= simple\_format(@article.description) %>

<div class="article-actions">

<%= link\_to "Edit this article", edit\_article\_path(@article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(@article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

<%= link\_to "View all articles", articles\_path, class: "btn btn-xs btn-success" %>

</div>

</div>

**Lecture 110 Add Pagination – Text with Directions and Code**

Add the two gems required for pagination to the gemfile:

gem 'will\_paginate', '3.0.7'

gem 'bootstrap-will\_paginate', '0.0.10'

Then from the command line run:

bundle install --without production

This will install the gems

Change the index action in the articles\_controller.rb file to look like below:

def index

@articles = Article.paginate(page: params[:page], per\_page: 5)

end

Change the index.html.erb page under the app/views/articles folder to look like below:

<h1 align="center">Listing all articles</h1>

<div align="center">

<%= will\_paginate %>

</div>

<%= render 'article', obj: @articles %>

<div align="center">

<%= will\_paginate %>

</div>

Change the index action in the users\_controller.rb file to look like below:

def index

@users = User.paginate(page: params[:page], per\_page: 5)

end

Change the show action in the users\_controller.rb file to look like below:

def show

@user = User.find(params[:id])

@user\_articles = @user.articles.paginate(page: params[:page], per\_page: 5)

end

Add the <%= will\_paginate %> line to the index.html.erb view under the app/views/users folder and make it look like below:

<h1 align="center">All Alpha Bloggers</h1>

<div align="center">

<%= will\_paginate %>

<% @users.each do |user| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li><%= link\_to gravatar\_for(user), user\_path(user) %></li>

<li class="article-title">

<%= link\_to user.username, user\_path(user) %>

</li>

<li><small><%= pluralize(user.articles.count, "article") if user.articles %></small></li>

</div>

</div>

</ul>

<% end %>

<%= will\_paginate %>

</div>

Change the bottom of the show.html.erb view under the app/views/users folder by adding the pagination and @user\_articles code and make it look like below:

<h1 align="center">Welcome to <%= @user.username %>'s page</h1>

<div class="row">

<div class="col-md-4 col-md-offset-4 center">

<%= gravatar\_for @user, size: 150 %>

</div>

</div>

<h4 align="center"><%= @user.username %>'s articles</h4>

<div class="center">

<%= will\_paginate @user\_articles %>

</div>

<%= render 'articles/article', obj: @user\_articles %>

<div class="center">

<%= will\_paginate @user\_articles %>

</div>

**Lecture 112 Add Login Form – Text with Directions and Code**

Add the login/logout routes in the config/routes.rb file:

get 'login', to: 'sessions#new'

post 'login', to: 'sessions#create'

delete 'logout', to: 'sessions#destroy'

Create a sessions\_controller.rb file under app/controllers folder and fill it in with the following:

class SessionsController < ApplicationController

def new

end

def create

end

def destroy

end

end

Create a new folder under app/views called sessions, then within the sessions folder create a new file called new.html.erb and fill it in with the following, this will be your log in form:

<h1 align="center">Log in</h1>

<%= form\_for(:session, :html => {class: "form-horizontal", role: "form"}, url: login\_path) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :email %>

</div>

<div class="col-sm-8">

<%= f.email\_field :email, class: "form-control", placeholder: "Enter email", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :password %>

</div>

<div class="col-sm-8">

<%= f.password\_field :password, class: "form-control", autocomplete: "off" %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit "Log in", class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

**Lecture 114 Create and Destroy Sessions – Text with Directions and Code**

Fill in the create and destroy actions in the sessions\_controller.rb file with the following:

def create

user = User.find\_by(email: params[:session][:email].downcase)

if user && user.authenticate(params[:session][:password])

session[:user\_id] = user.id

flash[:success] = "You have successfully logged in"

redirect\_to user\_path(user)

else

flash.now[:danger] = "There was something wrong with your login information"

render 'new'

end

end

def destroy

session[:user\_id] = nil

flash[:success] = "You have logged out"

redirect\_to root\_path

end

Add a logout link to your navigation partial (\_navigation.html.erb) in the app/views/layouts folder by changing one of the default links which are listed as <li> item:

<li><%= link\_to 'Log out', logout\_path, method: :delete %></li>

**Lecture 116 Authentication Methods – Text with Directions and Code**

Add the methods and code to make the application\_controller.rb file look like below:

class ApplicationController < ActionController::Base

# Prevent CSRF attacks by raising an exception.

# For APIs, you may want to use :null\_session instead.

protect\_from\_forgery with: :exception

helper\_method :current\_user, :logged\_in?

def current\_user

@current\_user ||= User.find(session[:user\_id]) if session[:user\_id]

end

def logged\_in?

!!current\_user

end

def require\_user

if !logged\_in?

flash[:danger] = "You must be logged in to perform that action"

redirect\_to root\_path

end

end

end

Update the navigation partial where you added the logout link in the last lecture (last ul class in the view) and make it look like below:

<ul class="nav navbar-nav navbar-right">

<% if logged\_in? %>

<li><%= link\_to 'Log out', logout\_path, method: :delete %></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Your Profile<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><%= link\_to "Edit your profile", edit\_user\_path(current\_user) %></li>

<li><%= link\_to "View your profile", user\_path(current\_user) %></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

</ul>

</li>

<% else %>

<li><%= link\_to 'Log in', login\_path %></li>

<li><%= link\_to 'Sign up', signup\_path %></li>

<% end %>

</ul>

Extract the redundancy from the users\_controller.rb file by using a before\_action :set\_user on top to make the file look like below:

class UsersController < ApplicationController

before\_action :set\_user, only: [:edit, :update, :show]

def index

@users = User.paginate(page: params[:page], per\_page: 5)

end

def new

@user = User.new

end

def create

@user = User.new(user\_params)

if @user.save

flash[:success] = "Welcome to the alpha blog #{@user.username}"

redirect\_to articles\_path

else

render 'new'

end

end

def edit

end

def update

if @user.update(user\_params)

flash[:success] = "Your account was updated successfully"

redirect\_to articles\_path

else

render 'edit'

end

end

def show

@user\_articles = @user.articles.paginate(page: params[:page], per\_page: 5)

end

private

def user\_params

params.require(:user).permit(:username, :email, :password)

end

def set\_user

@user = User.find(params[:id])

end

end

**Lecture 118 Restrict Actions from UI – Text with Directions and Code**

Updated and completed navigation partial:

<nav class="navbar navbar-default">

<div class="container-fluid">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<%= link\_to "Alpha blog", root\_path, class: "navbar-brand", id: "logo" %>

</div>

<!-- Collect the nav links, forms, and other content for toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav">

<li><%= link\_to "Articles", articles\_path %></li>

<li><%= link\_to "Users", users\_path %></li>

<% if logged\_in? %>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Actions <span class="caret"></span></a>

<ul class="dropdown-menu">

<li><%= link\_to "New Article", new\_article\_path %></li>

<li><a href="#">Another action</a></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">One more separated link</a></li>

</ul>

</li>

<% end %>

</ul>

<form class="navbar-form navbar-left" role="search">

<div class="form-group">

<input type="text" class="form-control" placeholder="Search">

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>

<ul class="nav navbar-nav navbar-right">

<% if logged\_in? %>

<li><%= link\_to 'Log out', logout\_path, method: :delete %></li>

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Your Profile<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><%= link\_to "Edit your profile", edit\_user\_path(current\_user) %></li>

<li><%= link\_to "View your profile", user\_path(current\_user) %></li>

<li><a href="#">Something else here</a></li>

<li role="separator" class="divider"></li>

<li><a href="#">Separated link</a></li>

</ul>

</li>

<% else %>

<li><%= link\_to 'Log in', login\_path %></li>

<li><%= link\_to 'Sign up', signup\_path %></li>

<% end %>

</ul>

</div><!-- /.navbar-collapse -->

</div><!-- /.container-fluid -->

</nav>

Updated home.html.erb in the app/views/pages folder:

<div class="center jumbotron">

<h1>ALPHA BLOG</h1>

<%= link\_to "Sign up now", signup\_path, class: "btn btn-danger btn-lg btn-xlarge" %>

</div>

Updated \_article.html.erb partial in the app/views/articles folder with the logged in restrictions:

<% obj.each do |article| %>

<div class="row">

<div class="col-xs-8 col-xs-offset-2">

<div class="well well-lg">

<div class="article-title">

<%= link\_to article.title, article\_path(article) %>

</div>

<div class="article-body">

<%= truncate(article.description, length: 100) %>

<div class="article-meta-details">

<small>Created by: <%= article.user.username if article.user%>,

<%= time\_ago\_in\_words(article.created\_at) %> ago,

last updated: <%= time\_ago\_in\_words(article.updated\_at) %> ago</small>

</div>

</div>

<% if logged\_in? && current\_user == article.user %>

<div class="article-actions">

<%= link\_to "Edit this article", edit\_article\_path(article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

</div>

<% end %>

</div>

</div>

</div>

<% end %>

Updated show.html.erb (show articles) page in the app/views/articles folder:

<h2 align="center">Title: <%= @article.title %></h2>

<div class="well col-xs-8 col-xs-offset-2">

<% if @article.user %>

<ul class="listing">

<div class="row center">

<div class="col-md-4 col-md-offset-4">

<li>Created by:</li>

<li><%= link\_to gravatar\_for(@article.user), user\_path(@article.user) %></li>

<li class="article-title">

<%= link\_to @article.user.username, user\_path(@article.user) %>

</li>

<li><small>

<%= pluralize(@article.user.articles.count, "article") if @article.user.articles %>

</small></li>

</div>

</div>

</ul>

<% end %>

<h4 class="center description"><strong>Description:</strong></h4>

<hr>

<%= simple\_format(@article.description) %>

<div class="article-actions">

<% if logged\_in? && current\_user == @article.user %>

<%= link\_to "Edit this article", edit\_article\_path(@article), class: "btn btn-xs btn-primary" %>

<%= link\_to "Delete this article", article\_path(@article), method: :delete,

data: { confirm: "Are you sure you want to delete the article?"},

class: "btn btn-xs btn-danger" %>

<% end %>

<%= link\_to "View all articles", articles\_path, class: "btn btn-xs btn-success" %>

</div>

</div>

Update the home action in the pages\_controller.rb file to look like below:

def home

redirect\_to articles\_path if logged\_in?

end

**Lecture 120 Restrict Actions in Controller – Text with Directions and Code**

Completed articles\_controller.rb file:

class ArticlesController < ApplicationController

before\_action :set\_article, only: [:edit, :update, :show, :destroy]

before\_action :require\_user, except: [:index, :show]

before\_action :require\_same\_user, only: [:edit, :update, :destroy]

def index

@articles = Article.paginate(page: params[:page], per\_page: 5)

end

def new

@article = Article.new

end

def edit

end

def create

@article = Article.new(article\_params)

@article.user = current\_user

if @article.save

flash[:success] = "Article was successfully created"

redirect\_to article\_path(@article)

else

render 'new'

end

end

def update

if @article.update(article\_params)

flash[:success] = "Article was successfully updated"

redirect\_to article\_path(@article)

else

render 'edit'

end

end

def show

end

def destroy

@article.destroy

flash[:danger] = "Article was successfully deleted"

redirect\_to articles\_path

end

private

def set\_article

@article = Article.find(params[:id])

end

def article\_params

params.require(:article).permit(:title, :description)

end

def require\_same\_user

if current\_user != @article.user

flash[:danger] = "You can only edit or delete your own articles"

redirect\_to root\_path

end

end

end

Completed users\_controller.rb file

class UsersController < ApplicationController

before\_action :set\_user, only: [:edit, :update, :show]

before\_action :require\_same\_user, only: [:edit, :update]

def index

@users = User.paginate(page: params[:page], per\_page: 5)

end

def new

@user = User.new

end

def create

@user = User.new(user\_params)

if @user.save

flash[:success] = "Welcome to the alpha blog #{@user.username}"

redirect\_to articles\_path

else

render 'new'

end

end

def edit

end

def update

if @user.update(user\_params)

flash[:success] = "Your account was updated successfully"

redirect\_to articles\_path

else

render 'edit'

end

end

def show

@user\_articles = @user.articles.paginate(page: params[:page], per\_page: 5)

end

private

def user\_params

params.require(:user).permit(:username, :email, :password)

end

def set\_user

@user = User.find(params[:id])

end

def require\_same\_user

if current\_user != @user

flash[:danger] = "You can only edit your own account"

redirect\_to root\_path

end

end

end

To deploy to heroku, first ensure you have committed all your changes to your git repo, then:

git push heroku master

Then run any pending migrations in heroku:

heroku run rake db:migrate

Don't forget to post a link to your heroku app to the discussions area!

**Lecture 122 Sign in Upon Sign Up – Text with Directions and Code**

Update the users controller create action to look like the following:

def create

@user = User.new(user\_params)

if @user.save

session[:user\_id] = @user.id

flash[:success] = "Welcome to the alpha blog #{@user.username}"

redirect\_to user\_path(@user)

else

render 'new'

end

end

**Lecture 124 Add Admin User Functionality – Text with Directions and Code**

To generate a migration and add the admin column to users table:

rails generate migration add\_admin\_to\_users

Within the migration file, in the def change method, add in the following code:

add\_column :users, :admin, :boolean, default: false

Then run rake db:migrate to run the migration file and add the column to the users table:

rake db:migrate

Jump on rails console and you can grab a user object and set the user to admin by setting their admin column to true:

user = User.find(idofuser)

or

user = User.last or User.first or User.find\_by(email: "emailofuser")

user.admin?

user.toggle!(:admin)

Then user.admin? # should be true at this point

Update require\_same\_user method in the articles\_controller.rb file:

def require\_same\_user

if current\_user != @article.user and !current\_user.admin?

flash[:danger] = "You can only edit or delete your own articles"

redirect\_to root\_path

end

end

Then proceed to articles views, in the \_article.html.erb partial under the app/views/articles folder, update the line where you're checking for <% if logged\_in? &&....%> to:

<% if logged\_in? && (current\_user == article.user || current\_user.admin?) %>

Make the same update to the show.html.erb file under app/views/articles folder:

<% if logged\_in? && (current\_user == article.user || current\_user.admin?) %>

In \_navigation.html.erb partial under app/views/layouts folder add admin next to where you display Your Profile:

Your Profile <%= "Admin" if current\_user.admin? %>

**Lecture 126 Complete Admin User – Text with Directions and Code**

Updated users\_controller.rb file (several updates):

class UsersController < ApplicationController

before\_action :set\_user, only: [:edit, :update, :show]

before\_action :require\_same\_user, only: [:edit, :update, :destroy]

before\_action :require\_admin, only: [:destroy]

def index

@users = User.paginate(page: params[:page], per\_page: 5)

end

def new

@user = User.new

end

def create

@user = User.new(user\_params)

if @user.save

session[:user\_id] = @user.id

flash[:success] = "Welcome to the alpha blog #{@user.username}"

redirect\_to user\_path(@user)

else

render 'new'

end

end

def edit

end

def update

if @user.update(user\_params)

flash[:success] = "Your account was updated successfully"

redirect\_to articles\_path

else

render 'edit'

end

end

def show

@user\_articles = @user.articles.paginate(page: params[:page], per\_page: 5)

end

def destroy

@user = User.find(params[:id])

@user.destroy

flash[:danger] = "User and all articles created by user have been deleted"

redirect\_to users\_path

end

private

def user\_params

params.require(:user).permit(:username, :email, :password)

end

def set\_user

@user = User.find(params[:id])

end

def require\_same\_user

if current\_user != @user and !current\_user.admin?

flash[:danger] = "You can only edit your own account"

redirect\_to root\_path

end

end

def require\_admin

if logged\_in? and !current\_user.admin?

flash[:danger] = "Only admin users can perform that action"

redirect\_to root\_path

end

end

end

Updated user.rb model file under app/models folder:

class User < ActiveRecord::Base

has\_many :articles, dependent: :destroy

before\_save { self.email = email.downcase }

validates :username, presence: true,

uniqueness: { case\_sensitive: false },

length: { minimum: 3, maximum: 25 }

VALID\_EMAIL\_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i

validates :email, presence: true, length: { maximum: 105 },

uniqueness: { case\_sensitive: false },

format: { with: VALID\_EMAIL\_REGEX }

has\_secure\_password

end

Updated index.html.erb file under app/views/users folder:

<h1 align="center">All Alpha Bloggers</h1>

<div align="center">

<%= will\_paginate %>

<% @users.each do |user| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li><%= link\_to gravatar\_for(user), user\_path(user) %></li>

<li class="article-title">

<%= link\_to user.username, user\_path(user) %>

</li>

<li><small><%= pluralize(user.articles.count, "article") if user.articles %></small></li>

<% if logged\_in? and current\_user.admin? %>

<li><%= link\_to "Delete this user", user\_path(user), method: :delete,

data: { confirm: "Are you sure you want to delete the user and all their articles?" } %></li>

<% end %>

</div>

</div>

</ul>

<% end %>

<%= will\_paginate %>

</div>

To deploy to heroku, ensure you have committed your code to your git repo, then:

git push heroku master

Then run any pending migrations:

heroku run rake db:migrate

To set admin user from heroku app run:

heroku run rails console

Then grab a user (example, user = User.last):

user.toggle!(:admin)

That will set the admin column to true (if it was false)

Homework assignment for section 7:

Read [http://guides.rubyonrails.org/association\_basics.h...](http://guides.rubyonrails.org/association_basics.html#the-has-many-through-association) and try to come up with 2 many-to-many association examples in 1 to 2 lines (let's say between patients and doctors as an example) and post them to the discussions

**Lecture 129 Category Model and Unit Tests – Text with Directions and Code**

Create a category\_test.rb file under test/models folder and fill it in with the following:

require 'test\_helper'

class CategoryTest < ActiveSupport::TestCase

def setup

@category = Category.new(name: "sports")

end

test "category should be valid" do

assert @category.valid?

end

test "name should be present" do

@category.name = " "

assert\_not @category.valid?

end

test "name should be unique" do

@category.save

category2 = Category.new(name: "sports")

assert\_not category2.valid?

end

test "name should not be too long" do

@category.name = "a" \* 26

assert\_not @category.valid?

end

test "name should not be too short" do

@category.name = "aa"

assert\_not @category.valid?

end

end

Create a model file named category.rb under the app/models folder and fill it in:

class Category < ActiveRecord::Base

validates :name, presence: true, length: { minimum: 3, maximum: 25 }

validates\_uniqueness\_of :name

end

To create a migration file to create the categories table:

rails generate migration create\_categories

Then fill in the migration file within the create\_table block:

t.string :name

t.timestamps

Run the migration to create the categories table:

rake db:migrate

**Lecture 131 Categories Controller and Tests – Text with Directions and Code**

Create a file named categories\_controller\_test.rb under the test/controllers folder and fill it in with the following:

require 'test\_helper'

class CategoriesControllerTest < ActionController::TestCase

def setup

@category = Category.create(name: "sports")

end

test "should get categories index" do

get :index

assert\_response :success

end

test "should get new" do

get :new

assert\_response :success

end

test "should get show" do

get(:show, {'id' => @category.id})

assert\_response :success

end

end

Under app/controllers folder create a file named categories\_controller.rb and fill it in:

class CategoriesController < ApplicationController

def index

end

def new

end

def show

end

end

In your config/routes.rb file add in the following route for categories:

resources :categories, except: [:destroy]

Under app/views folder create a folder called categories and create 3 files, new.html.erb, index.html.erb and show.html.erb

**Lecture 133 Integration Test and Feature – Text with Directions and Code**

Under test/integration folder create a file named create\_categories\_test.rb which will be the integration test for creating categories and fill it in with the following:

require 'test\_helper'

class CreateCategoriesTest < ActionDispatch::IntegrationTest

test "get new category form and create category" do

get new\_category\_path

assert\_template 'categories/new'

assert\_difference 'Category.count', 1 do

post\_via\_redirect categories\_path, category: {name: "sports"}

end

assert\_template 'categories/index'

assert\_match "sports", response.body

end

end

Fill in the new.html.erb template under the app/views/categories folder:

<h1 align="center">Create new category</h1>

<%= render 'shared/errors', obj: @category %>

<%= form\_for(@category, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :name %>

</div>

<div class="col-sm-8">

<%= f.text\_field :name, class: "form-control", placeholder: "Category name", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

Fill in the code in the categories\_controller.rb file under the app/controllers folder:

class CategoriesController < ApplicationController

def index

@categories = Category.all

end

def new

@category = Category.new

end

def create

@category = Category.new(category\_params)

if @category.save

flash[:success] = "Category was created successfully"

redirect\_to categories\_path

else

render 'new'

end

end

def show

end

private

def category\_params

params.require(:category).permit(:name)

end

end

Fill in the categories index.html.erb view under the app/views/categories page:

<h1 align="center">Listing All Categories</h1>

<div align="center">

<% @categories.each do |category| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li class="article-title">

<%= link\_to "#{category.name}", category\_path(category) %>

</li>

</div>

</div>

</ul>

<% end %>

</div>

**Lecture 135 Integration Test for Invalid Category – Text with Directions and Code**

Add the following test to your create\_categories\_test.rb file under test/integration folder:

test "invalid category submission results in failure" do

get new\_category\_path

assert\_template 'categories/new'

assert\_no\_difference 'Category.count' do

post categories\_path, category: {name: " "}

end

assert\_template 'categories/new'

assert\_select 'h2.panel-title'

assert\_select 'div.panel-body'

end

**Lecture 137 Integration Test and Feature: Listing Category – Text with Directions and Code**

Under test/integration folder create a file named list\_categories\_test.rb and fill it in (Note this is the corrected version with the correct class name, in the video an error is made which is fixed in the next video):

require 'test\_helper'

class ListCategoriesTest < ActionDispatch::IntegrationTest

def setup

@category = Category.create(name: "sports")

@category2 = Category.create(name: "programming")

end

test "should show categories listing" do

get categories\_path

assert\_template 'categories/index'

assert\_select "a[href=?]", category\_path(@category), text: @category.name

assert\_select "a[href=?]", category\_path(@category2), text: @category2.name

end

end

Add pagination to the app/views/categories/index.html.erb file and make it look like below:

<h1 align="center">Listing All Categories</h1>

<div align="center">

<%= will\_paginate %>

<% @categories.each do |category| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li class="article-title">

<%= link\_to "#{category.name}", category\_path(category) %>

</li>

<li><small><%= pluralize(category.articles.count, "article") %></small></li>

</div>

</div>

</ul>

<% end %>

<%= will\_paginate %>

</div>

Update the index action in the categories\_controller.rb file to accomodate pagination:

def index

@categories = Category.paginate(page: params[:page], per\_page: 5)

end

**Lecture 139 Fix List Categories Test – Text with Directions and Code**

Due to the same class name of CreateCategoriesTest, an instance variable (object) initiated in 1 test lasted the lifecycle of the object which spanned multiple files (since same class), thus causing the conflict. Correcting the class name removed the list\_categories\_test.rb file from being part of the CreateCategoriesTest class and the @category intance variable was no longer spanning that file (and it's tests) and promptly ended when the create\_categories\_test.rb file completed.

**Lecture 141 Admin User Requirement and Test – Text with Directions and Code**

Update categories\_controller\_test.rb file to create admin user and then simulate log in:

require 'test\_helper'

class CategoriesControllerTest < ActionController::TestCase

def setup

@category = Category.create(name: "sports")

@user = User.create(username: "john", email: "john@example.com", password: "password", admin: true)

end

test "should get categories index" do

get :index

assert\_response :success

end

test "should get new" do

session[:user\_id] = @user.id

get :new

assert\_response :success

end

test "should get show" do

get(:show, {'id' => @category.id})

assert\_response :success

end

test "should redirect create when admin not logged in" do

assert\_no\_difference 'Category.count' do

post :create, category: { name: "sports" }

end

assert\_redirected\_to categories\_path

end

end

Add code to require logged in admin users to categories\_controller.rb file:

class CategoriesController < ApplicationController

before\_action :require\_admin, except: [:index, :show]

def index

@categories = Category.paginate(page: params[:page], per\_page: 5)

end

def new

@category = Category.new

end

def create

@category = Category.new(category\_params)

if @category.save

flash[:success] = "Category was created successfully"

redirect\_to categories\_path

else

render 'new'

end

end

def show

end

private

def category\_params

params.require(:category).permit(:name)

end

def require\_admin

if !logged\_in? || (logged\_in? and !current\_user.admin?)

flash[:danger] = "Only admins can perform that action"

redirect\_to categories\_path

end

end

end

**Lecture 143 Update Test and Navbar – Text with Directions and Code**

Update the create\_categories\_test.rb integration test file under test/integration folder to sign in an admin user:

require 'test\_helper'

class CreateCategoriesTest < ActionDispatch::IntegrationTest

def setup

@user = User.create(username: "john", email: "john@example.com", password: "password", admin: true)

end

test "get new category form and create category" do

sign\_in\_as(@user, "password")

get new\_category\_path

assert\_template 'categories/new'

assert\_difference 'Category.count', 1 do

post\_via\_redirect categories\_path, category: {name: "sports"}

end

assert\_template 'categories/index'

assert\_match "sports", response.body

end

test "invalid category submission results in failure" do

sign\_in\_as(@user, "password")

get new\_category\_path

assert\_template 'categories/new'

assert\_no\_difference 'Category.count' do

post categories\_path, category: {name: " "}

end

assert\_template 'categories/new'

assert\_select 'h2.panel-title'

assert\_select 'div.panel-body'

end

end

Add sign\_in\_user method to test\_helper.rb file under test folder:

def sign\_in\_as(user, password)

post login\_path, session: {email: user.email, password: password}

end

Update the navigation partial to display categories including restrictions based on admin user for new categories path by adding the following code right under the <% end %> block for Actions and above the </ul> tag:

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Categories <span class="caret"></span></a>

<ul class="dropdown-menu">

<li><%= link\_to "All Categories", categories\_path %></li>

<% Category.all.each do |category| %>

<li><%= link\_to "#{category.name}", category\_path(category) %></li>

<% end %>

<% if logged\_in? and current\_user.admin? %>

<li role="separator" class="divider"></li>

<li><%= link\_to "Create New Category", new\_category\_path %></li>

<% end %>

</ul>

</li>

**Lecture 145 Build Article and Category Association – Text with Directions and Code**

To generate a migration file to create the article\_categories table:

rails generate migration create\_article\_categories

Within the migration file add in the following to add article\_id and category\_id to the table:

t.integer :article\_id

t.integer :category\_id

Run rake db:migrate to create the table:

rake db:migrate

Create a model file named article\_category.rb under the app/models folder and fill it in:

class ArticleCategory < ActiveRecord::Base

belongs\_to :article

belongs\_to :category

end

Update the article.rb and category.rb model files to include the following lines:

article.rb ->

has\_many :article\_categories

has\_many :categories, through: :article\_categories

category.rb ->

has\_many :article\_categories

has\_many :articles, through: :article\_categories

Then hop on the rails console and test out the associations:

ArticleCategory.all

ArticleCategory

article = Article.last

category = Category.last

article.categories

article.categories << category

category.articles

category.articles << article

**Lecture 147 Add Associations from UI – Text with Directions and Code**

Update the \_form.html.erb partial under the app/views/articles folder and add the part to associate categories:

<%= render 'shared/errors', obj: @article %>

<div class='row'>

<div class='col-xs-12'>

<%= form\_for(@article, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :title %>

</div>

<div class="col-sm-8">

<%= f.text\_field :title, class: "form-control", placeholder: "Title of article", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :description %>

</div>

<div class="col-sm-8">

<%= f.text\_area :description, rows: 10, class: "form-control", placeholder: "Body of article" %>

</div>

</div>

<div class="form-group">

<div class="row">

<div class="col-sm-offset-2 col-sm-8">

<%= f.collection\_check\_boxes :category\_ids, Category.all, :id, :name do |cb| %>

<% cb.label(class: "checkbox-inline input\_checkbox") {cb.check\_box(class: "checkbox") + cb.text} %>

<% end %>

</div>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

</div>

</div>

Update the article\_params method under private in the articles\_controller.rb file to allow category\_ids:

def article\_params

params.require(:article).permit(:title, :description, category\_ids: [])

end

**Lecture 149 Update Views – Text with Directions and Code**

Add in the following code under description and above article-actions in the show.html.erb file under app/views/articles folder:

<% if @article.categories.any? %>

<p>Categories: <%= render @article.categories %></p>

<% end %>

Under app/views/categories folder add a \_category.html.erb partial and fill it in with the following:

<span class="badge"><%= link\_to category.name, category\_path(category) %>  </span>

Update styling in your custom.css.scss file under app/assets directory:

.badge {

background-color: #CEF6F5;

text-transform: uppercase;

letter-spacing: -1px;

font-weight: bold;

}

.input\_checkbox input {

width: auto !important;

}

Next update the \_article.html.erb partial under the app/views/articles folder above the if logged in block for article-actions:

<% if article.categories.any? %>

<p>Categories: <%= render article.categories %></p>

<% end %>

Update the show.html.erb template under the app/views/categories folder:

<h1 align="center"><%= "Category: " + @category.name %></h1>

<div align="center">

<%= will\_paginate @category\_articles %>

</div>

<%= render 'articles/article', obj: @category\_articles %>

<div align="center">

<%= will\_paginate @category\_articles %>

</div>

Update the categories\_controller.rb file show action:

def show

@category = Category.find(params[:id])

@category\_articles = @category.articles.paginate(page: params[:page], per\_page: 5)

end

Update the index.html.erb page under app/views/categories folder to add number of articles in each category:

<h1 align="center">Listing All Categories</h1>

<div align="center">

<%= will\_paginate %>

<% @categories.each do |category| %>

<ul class="listing">

<div class="row">

<div class="well col-md-4 col-md-offset-4">

<li class="article-title">

<%= link\_to "#{category.name}", category\_path(category) %>

</li>

<li><small><%= pluralize(category.articles.count, "article") %></small></li>

</div>

</div>

</ul>

<% end %>

<%= will\_paginate %>

</div>

**Lecture 151 Edit Categories – Text with Directions and Code**

In the categories\_controller.rb file create edit and update actions:

def edit

@category = Category.find(params[:id])

end

def update

@category = Category.find(params[:id])

if @category.update(category\_params)

flash[:success] = "Category name was successfully updated"

redirect\_to category\_path(@category)

else

render 'edit'

end

end

Create a new partial \_form.html.erb under app/views/categories folder and fill in the the code below(except the title, you can copy the rest of the code from the new.html.erb file):

<%= render 'shared/errors', obj: @category %>

<%= form\_for(@category, :html => {class: "form-horizontal", role: "form"}) do |f| %>

<div class="form-group">

<div class="control-label col-sm-2">

<%= f.label :name %>

</div>

<div class="col-sm-8">

<%= f.text\_field :name, class: "form-control", placeholder: "Category name", autofocus: true %>

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<%= f.submit class: 'btn btn-primary btn-lg' %>

</div>

</div>

<% end %>

<div class="col-xs-4 col-xs-offset-4">

[ <%= link\_to "Cancel request and return to articles listing", articles\_path %> ]

</div>

Then update the new.html.erb in the same app/views/categories folder to look like below:

<h1 align="center">Create new category</h1>

<%= render 'form' %>

Create an edit.html.erb file under the same app/views/categories folder and fill it in:

<h1 align="center">Edit Category Name</h1>

<%= render 'form' %>

Update the show.html.erb template under the app/views/categories folder to display an edit category path:

<h1 align="center"><%= "Category: " + @category.name %></h1>

<div align="center">

<% if logged\_in? and current\_user.admin? %>

<div class="row">

<span class="badge"><%= link\_to "Edit Category Name", edit\_category\_path(@category) %></span>

</div>

<% end %>

<%= will\_paginate @category\_articles %>

</div>

<%= render 'articles/article', obj: @category\_articles %>

<div align="center">

<%= will\_paginate @category\_articles %>

</div>

**Lecture 153 Deploy to Development – Text with Directions and Code**

To deploy your app to heroku, first check git status to ensure you don't have any uncommitted code, if you do, make a commit of it

Then deploy to heroku:

git push heroku master

Then run any pending migrations in heroku:

heroku run rake db:migrate

Dont forget to post a link for your heroku app to the discussion area!!! This step is very important

Homework assignment:

1. Create integration test for signing up a user. Create a feature branch and do this in the feature branch

2. Create integration test for creating new article, use a feature branch for this as well

You can reference the integration tests for categories as a guide, post your tests to the discussions area or post a link to your github repo

Congratulations on completing section 7!

**Lecture 155 Finance Tracker Requirements**

1. Authentication system, users can sign-up, edit their profile, log in/logout

2. Users can track stocks, up to 10 per user. The profile page will display all the current stock prices

3. User can search for a stock symbol using a search bar

4. User can choose to add a stock from results or search to their portfolio

5. User can look for friends, or other users of the app, by name (first or last or email)

6. User can view portfolio of stocks their friends are tracking for investing ideas.

7. Must be mobile friendly, so the styling has to be responsive

Need fast development, but needs to be presentable

No test framework required at this time since you’re building a prototype fast to display at an investors gathering

Most important – core functionality, time and presentation

**Lecture 156 Finance Tracker Requirements**

Assignment 1:

1. Create a new rails application from your rails projects directory called finance-tracker  
(hint: rails new name-of-app to create a new rails app)

2. Change directory into your finance-tracker folder and ensure that the rails server runs and you are able to preview the app.  
3. Initialize a git repository for your app files, the message should be "Initial commit of app" (ensure you do this from the finance-tracker directory)  
4. Update the readme file to say something relevant about the app. Example: "This is the finance tracker app from the Complete Ruby on Rails Developer course"  
5. Make a commit of your change

**Lecture 157 Finance Tracker Requirements**

Assignment 2:

1. Create a github repository for your finance-tracker app in your github account (now you should have two github repositories, 1 for alpha-blog and 1 for finance-tracker in your github profile)  
2. Push your code to your github repository so it's saved  
3. Re-organize your gemfile to put sqlite3 gem in group development, test and create a group for production  
4. In your production group add in gem pg (postgres) and rails\_12factor to make your app heroku ready  
5. Run bundle install --without production to update your .gemfile.lock file  
6. Create a welcome controller, a root route for welcome#index, an index action in your welcome\_controller.rb file (should be empty at this time)  
7. Create a welcome folder under your app/views/ folder and within the folder create an index.html.erb file and fill it in with <h1>Welcome to the finance tracker app</h1>  
8. Preview your app and ensure it goes to this page in local development  
9. Commit your code to your git repo  
10. Push your code to your github repo  
11. Create a heroku app for your application (you can reference steps taken during the alpha-blog, ensure you do this from your finance-tracker app folder)  
12. Rename the heroku app to something more friendly and reasonable  
13. Deploy your code to your heroku app and ensure you are able to preview the app and it takes you to the "Welcome to the finance tracker app" page  
14. Post a link to your heroku app page in the discussions  
15. IMPORTANT: Don't proceed to the next video unless you're able to get the above working. It is necessary that you are able to perform the tasks above and the videos going forward will make this assumption (You can reference the alpha-blog videos for help at any time). If you want to skip production deploy with heroku, then you can skip those steps, but you have to atleast get a github repository and working root route/view working.

**Lecture 158 Devise and Bootstrap**

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

*Steps to customize devise (given in the command line)*

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

**Lecture 159 Devise and Bootstrap – Text Directions and Code**

In your gemfile, add the devise gem:

gem 'devise'

Then run:

bundle install --without production

Then install devise in your application:

rails generate devise:install

rails generate devise User

rake db:migrate to add users table

Add the following line to the application\_controller.rb file under app/controllers:

before\_action :authenticate\_user!

Add a logout link to the homepage which is the index.html.erb view under app/views/welcome folder:

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

Add gem 'twitter-bootstrap-rails' in your gemfile and bundle install --without production

Then run the following commands:

rails generate bootstrap:install static

rails g bootstrap:layout application

override using Y

Then add gem 'devise-bootstrap-views' in your gemfile and bundle install --without production

Under your app/assets/stylesheets folder, add the following line to your application.css file above the \*= require\_tree . line:

\*= require devise\_bootstrap\_views

Then run the following two commands from the terminal:

rails g devise:views:locale en

rails g devise:views:bootstrap\_templates

In your config/routes.rb file add the following line:

devise\_for :users

Deploy to heroku and test out authentication by signing up some users and then logging in/out

A problem faced here by students is after adding the before\_action :authenticate\_user! in the application controller, the check doesn't seem to be working. Also, the logout link may generate an error "No route matches [GET] "/users/sign\_out" even with the method: :delete. If you face this error, the issue may be the following: The welcome controller was inheriting from ActionController::Base instead of ApplicationController, once you correct this it should work

**Lecture 161 Stock Model – Text Directions and Code**

Remove the sidebar div from the application.html.erb file under app/views/layouts folder

Remove(or code out) the following code for forgot your password link from the \_links.html.erb partial under the app/views/devise/shared folder:

<%- if devise\_mapping.recoverable? && controller\_name != 'passwords' %>

<%= link\_to t(".forgot\_your\_password", :default => "Forgot your password?"), new\_password\_path(resource\_name)

%><br />

<% end -%>

Generate Stock model:

rails g model Stock ticker:string name:string last\_price:decimal

Create the table by running the migration:

rake db:migrate

Add gem 'stock\_quote' to your gemfile and run bundle install --without production

Hop on rails console to get stock price for a stock:

StockQuote::Stock.quote("symbol")

StockQuote::Stock.quote("symbol").open

StockQuote::Stock.quote("symbol").previous\_close

Now need to add a view page for my portfolio, we'll call it my portfolio, so go to config/routes.rb file and add in the following route:

get 'my\_portfolio', to: 'users#my\_portfolio'

Create a users\_controller.rb file under app/controllers and add in my\_portfolio action:

class UsersController < ApplicationController

def my\_portfolio

end

end

under views add users folder and under it create a my\_portfolio.html.erb file and fill it in:

<h1>My Portfolio</h1>

You may add this as your root route in your config/routes.rb file by changing root to users#my\_portfolio

**Lecture 163 Stock Price Methods – Text Directions and Code**

Now we will add the two class level methods to the stock.rb model file under app/models folder and an third price method:

def self.find\_by\_ticker(ticker\_symbol)

where(ticker: ticker\_symbol).first

end

def self.new\_from\_lookup(ticker\_symbol)

looked\_up\_stock = StockQuote::Stock.quote(ticker\_symbol)

return nil unless looked\_up\_stock.name

new\_stock = new(ticker: looked\_up\_stock.symbol, name: looked\_up\_stock.name)

new\_stock.last\_price = new\_stock.price

new\_stock

end

def price

closing\_price = StockQuote::Stock.quote(ticker).close

return "#{closing\_price} (Closing)" if closing\_price

opening\_price = StockQuote::Stock.quote(ticker).open

return "#{opening\_price} (Opening)" if opening\_price

'Unavailable'

end

We are adding the self. prior to the method name, because these methods are not tied to any objects or object lifecycle, we need to be able to use them without having any instances of a stock.

**Lecture 165 UI for Stocks – Text Directions and Code**

First add a route to your config/routes.rb file to search stocks:

get 'search\_stocks', to: 'stocks#search'

Now we add a stocks\_controller.rb file under app/controllers and within it we define a search method. We are putting the search method in the stocks\_controller because we want to have an easy way to search for stocks from our view, so we'll call it from there:

class StocksController < ApplicationController

def search

if params[:stock]

@stock = Stock.find\_by\_ticker(params[:stock])

@stock ||= Stock.new\_from\_lookup(params[:stock])

end

if @stock

render partial: 'lookup'

else

render status: :not\_found, nothing: true

end

end

end

To temporarily view output from the browser, code out the render partial: 'lookup' line and add in the following line under it:

render json: @stock

Then append the following to your root URL to view the results:

search\_stocks?stock=AAPL

search\_stocks?stock=GOOG

Now we will create the lookup partial, create a stocks folder under app/views folder and within it create a file named \_lookup.html.erb

a. build the form\_tag

<div id="stock-lookup">

<h3>Search for Stocks</h3>

<%= form\_tag search\_stocks\_path, remote: true, method: :get, id: 'stock-lookup-form' do %>

<div class="form-group row no-padding text-center col-md-12">

<div class="col-md-10">

<%= text\_field\_tag :stock,

params[:stock],

placeholder: 'Stock Ticker Symbol',

autofocus: true,

class: 'form-control search-box input-lg' %>

</div>

<div class="col-md-2">

<%= button\_tag(type: :submit, class: 'btn btn-lg btn-success') do %>

<i class='fa fa-search'></i> Look up a stock

<% end %>

</div>

</div>

<% end %>

</div>

Go to my\_portfolio.html.erb under app/views/users and add the following line just to show the form:

<%= render 'stocks/lookup' %>

Now lets add the rest of the code to the lookup partial under app/views/stocks folder->

So if we have @stock instance variable then we want to do something with it, add the code below under the <% end %> and above the last </div> tag:

<% if @stock %>

<div id="stock-lookup-results" class="well results-block">

<strong>Symbol:</strong> <%= @stock.ticker %>

<strong>Name:</strong> <%= @stock.name %>

<strong>Price:</strong> <%= @stock.price %>

</div>

<% end %>

**Lecture 167 Form Submission and Ajax – Text Directions and Code**

Now that we have the id's and the form that submits via ajax, we want to handle the return of that ajax action so we create a stocks.js file in app/assets/javascripts folder

$(document).ready(function() {

init\_stock\_lookup();

})

then above it we type in:

var init\_stock\_lookup;

init\_stock\_lookup = function(){

$('#stock-lookup-form').on('ajax:success', function(event, data, status){

$('#stock-lookup').replaceWith(data);

init\_stock\_lookup();

});

}

You need to add the init\_stock\_lookup(); again since the listeners are gone once you replace with the data that's returned so you have to re-initialize it.

Now go to my\_portfolio URL from the browser and test out a few stock symbols

Next create a custom.css.scss file under app/assets/stylesheets folder and add some styling:

.results-block {

display: inline-block;

}

Now next thing we want to handle are errors, add the following to stocks.js file above the closing } for the init\_stock\_lookup = function()

$('#stock-lookup-form').on('ajax:error', function(event, xhr, status, error){

$('#stock-lookup-results').replaceWith('');

$('#stock-lookup-errors').replaceWith('Stock was not found.');

});

Add the following to lookup partial after the <% end %> and above the last </div>:

<div id="stock-lookup-errors"></div>

**Lecture 169 Spinner and more Ajax – Text Directions and Code**

Test out the stock lookup functionality from your browser, if you're having difficulty pulling it up, you can try to disable turbolinks from the application.js file by simply removing the = from after the // and make it look like below:

//= require jquery

//= require jquery\_ujs

//= require twitter/bootstrap

// require turbolinks

//= require\_tree .

Now lets add a spinner to make it look good while waiting to get search results

Add the following to the form inside the lookup partial right under the <% end %> and above the <% if @stock %> lines:

<%= render 'common/spinner' %>

Then under views folder create a folder named common and there create a \_spinner.html.erb file with the following code:

<div id="spinner" class="row col-md-12 text-center spinner" style="display: none;">

<i class='fa fa-spinner fa-spin fa-2x'></i> Processing...

</div>

Now you need to show the spinner using some javascript, so add the following code to application.js file under app/assets folder at the bottom

var hide\_spinner = function(){

$('#spinner').hide();

}

var show\_spinner = function(){

$('#spinner').show();

}

Now we want to add code for show\_spinner and hide\_spinner in the stock.js file

$('#stock-lookup-form').on('ajax:before', function(event, data, status){

show\_spinner();

});

$('#stock-lookup-form').on('ajax:after', function(event, data, status){

hide\_spinner();

});

You'll see that the spinner is spinning after you enter an invalid request, so add hide\_spinner to your stocks.js file under the ajax:error event handler

$('#stock-lookup-form').on('ajax:error', function(event, xhr, status, error){

hide\_spinner();

$('#stock-lookup-results').replaceWith(' ');

$('#stock-lookup-errors').replaceWith('Stock was not found.');

});

Now do a commit of your code, push to github and deploy to heroku and test out the stock lookup functionality

**Lecture 171 Many to Many Associations – Text Directions and Code**

So now we will create a many to many association between users and stocks because a stock can have many users tracking it and and a user can track many stocks, run the following command to create this:

rails generate scaffold UserStock user\_id:integer stock\_id:integer

Verify the migration file then run rake db:migrate to create the user\_stocks table:

rake db:migrate

in the user.rb model file add:

has\_many :user\_stocks

has\_many :stocks, through: :user\_stocks

in the stock.rb model file add:

has\_many :user\_stocks

has\_many :users, through: :user\_stocks

in the user\_stocks.rb model file add:

belongs\_to :user

belongs\_to :stock

In your config/routes.rb file ensure you have the resources :user\_stocks line, move it below devise\_for :users and add in the following:

resources :user\_stocks, except: [:show, :edit, :update]

Now go to lookup partial and add the ability to add a stock to the user\_stocks so they can be saved, add the following under the display of @stock.price (Price) within the <% if @stock %> branch:

<%= link\_to "Add to my Stocks", user\_stocks\_path(user: current\_user, stock\_ticker: @stock.ticker,

stock\_id: @stock.id ? @stock.id : ''), class: 'btn btn-xs btn-success', method: :post %>

Add the ability to create a new UserStock for an existing stock or a new one, add this to the create action in user\_stocks controller

def create

if params[:stock\_id].present?

@user\_stock = UserStock.new(stock\_id: params[:stock\_id], user: current\_user)

else

stock = Stock.find\_by\_ticker(params[:stock\_ticker])

if stock

@user\_stock = UserStock.new(user: current\_user, stock: stock)

else

stock = Stock.new\_from\_lookup(params[:stock\_ticker])

if stock.save

@user\_stock = UserStock.new(user: current\_user, stock: stock)

else

@user\_stock = nil

flash[:error] = "Stock is not available"

end

end

end

respond\_to do |format|

if @user\_stock.save

format.html { redirect\_to my\_portfolio\_path, notice: "Stock #{@user\_stock.stock.ticker} stock was successfully added" }

format.json { render :show, status: :created, location: @user\_stock }

else

format.html { render :new }

format.json { render json: @user\_stock.errors, status: :unprocessable\_entity }

end

end

end

**Lecture 173 Stock Restrictions – Text Directions and Code**

Add rules to not allow adding after 10 stocks have been added or if a stock has already been added, add the 3 methods below to user.rb file

def can\_add\_stock?(ticker\_symbol)

under\_stock\_limit? && !stock\_already\_added?(ticker\_symbol)

end

def under\_stock\_limit?

(user\_stocks.count < 10)

end

def stock\_already\_added?(ticker\_symbol)

stock = Stock.find\_by\_ticker(ticker\_symbol)

return false unless stock

user\_stocks.where(stock\_id: stock.id).exists?

end

So now in the lookup file wrap the link\_to "Add to my stocks" option in an if clause:

<% if current\_user.can\_add\_stock?(@stock.ticker) %>

<%= link\_to "Add to my Stocks", user\_stocks\_path(user: current\_user, stock\_ticker: @stock.ticker,

stock\_id: @stock.id ? @stock.id : ''), class: 'btn btn-xs btn-success', method: :post %>

<% end %>

now we need to display an else clause to show why the stock cannot be added: add the following before the end

<% else %>

<span class='label label-default'>

Stock cannot be added because you have already added

<% if !current\_user.under\_stock\_limit? %>

10 stocks

<% end %>

<% if current\_user.stock\_already\_added?(@stock.ticker) %>

this stock

<% end %>

</span>

<% end %>

In your user\_stocks\_controller.rb file, remove the second stock from the line below within the create action:

notice: "Stock #{@user\_stock.stock.ticker} was successfully added" }

**Lecture 175 Listing Stocks – Text Directions and Code**

Now we're going to add a display of the stocks the user is already tracking in the my\_portfolio page in the users\_controller.rb file within the my\_portfolio action:

@user\_stocks = current\_user.stocks

@user = current\_user

Then add the line below to the my\_portfolio view under app/views/users folder:

<%= render 'stocks/list' %>

Now create a \_list.html.erb partial in the app/views/stocks folder and fill in the code below:

<table class="table table-striped">

<thead>

<tr>

<th>Name</th>

<th>Symbol</th>

<th>Current Price</th>

<% if @user.id == current\_user.id %>

<th>Actions</th>

<% end %>

</tr>

</thead>

<tbody>

<% @user\_stocks.each do |user\_stock| %>

<tr>

<td><%= user\_stock.name %></td>

<td><%= user\_stock.ticker %></td>

<td><%= user\_stock.price %></td>

<% if @user.id == current\_user.id %>

<td>

<%= link\_to 'Delete',

user\_stock\_path(user\_stock),

:method => :delete,

:data => { :confirm => 'Are you sure?' },

:class => 'btn btn-xs btn-danger' %>

</td>

<% end %>

</tr>

<% end %>

</tbody>

</table>

To make the destroy action work, make the following change in the user\_stocks\_controller.rb file within the destroy action:

redirect\_to my\_portfolio\_path, notice: 'Stock was successfully removed from portfolio.'

You can now verify in server, do a commit and push to github/heroku and verify in production.

**Lecture 177 User Details – Text Directions and Code**

Add first\_name and last\_name to users table via migration and add a full\_name method to User

Step 1: rails generate migration add\_first\_and\_last\_name\_to\_users

Step 2: within migration file add the following within the def change method:

add\_column :users, :first\_name, :string

add\_column :users, :last\_name, :string

Run rake db:migrate to create the table

Now add a full\_name method to user.rb file

def full\_name

return "#{first\_name} #{last\_name}".strip if (first\_name || last\_name)

"Anonymous"

end

Add the following to new.html.erb and edit.html.erb under app/views/devise/registrations folder under the email field:

<div class="form-group">

<div class="col-md-6 no-padding">

<%= f.label :first\_name %>

<%= f.text\_field :first\_name, class: "form-control" %>

</div>

<div class="col-md-6 no-padding left-padding">

<%= f.label :last\_name %>

<%= f.text\_field :last\_name, class: "form-control" %>

</div>

</div>

Add the two classes to custom.css.scss:

.no-padding {

padding: 0 !important;

}

.left-padding {

padding-left: 15px !important;

}

Now create a new folder called user under controllers and within the user folder create a file called registrations\_controller.rb and fill it in:

class User::RegistrationsController < Devise::RegistrationsController

before\_filter :configure\_permitted\_parameters

protected

def configure\_permitted\_parameters

devise\_parameter\_sanitizer.for(:sign\_up).push(:first\_name, :last\_name)

devise\_parameter\_sanitizer.for(:account\_update).push(:first\_name, :last\_name)

end

end

Now we have to update the routes to allow for this registrations controller update, so update the route for devise in the routes.rb file:

devise\_for :users, :controllers => { :registrations => "user/registrations" }

**Lecture 179 UI Views for User Updates – Text Directions and Code**

Update the navigation portion of application.html.erb as below:

<% if current\_user %>

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-responsive-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">FinanceTracker</a>

<div class="navbar-collapse collapse navbar-responsive-collapse">

<ul class="nav navbar-nav col-md-6">

<li><%= link\_to "My Portfolio", my\_portfolio\_path %></li>

<li><%= link\_to "My Profile", edit\_registrations\_path(current\_user) %></li>

<li><%= link\_to "Link3", "/path3" %></li>

</ul>

<ul class="nav navbar-right col-md-4">

<li class="col-md-8 user-name text-right">

<i class="fa fa-user"></i>

<%= current\_user.full\_name.present? ? current\_user.full\_name : current\_user.email %>

</li>

<li class="col-md-4 logout"><%= link\_to("Logout", destroy\_user\_session\_path,

class: 'btn btn-xs btn-danger', method: :delete) %></li>

</ul>

</div>

<% end %>

Add the following classes to custom.css.scss file for styling:

.user-name {

padding-top: 15px;

}

.logout {

padding-left: 0;

}

.nav.navbar-right li {

.btn {

color: #fff !important;

margin-top: 5%;

}

.btn-danger:hover {

background-color: darken(#d9534f, 20%) !important;

}

}

Move over the navigation code (navbar) to a \_navigation.html.erb partial under the app/views/layouts folder from the application.html.erb file and in it's place put the following line in the application.html.erb:

<%= render 'layouts/navigation' %>

move the link to finance tracker outside the if block within the \_navigation.html.erb file:

<%= link\_to 'Finance Tracker', root\_path, class: "navbar-brand" %>

If you have a logout path in your index page you may remove it at this time

Make a commit, push to github and deploy to heroku

**Lecture 181 Adding Friends – Text Directions and Code**

Now we're going to add friends to the application, ability to view others users using the app and follow them so we can track their stock picks

First thing is add routes, add the following route to config/routes.rb file:

get 'my\_friends', to: 'users#my\_friends'

Now you have to create a my\_friends action in the users\_controller.rb file, then a my\_friends.html.erb file under the app/views/users folder, fill it in with some temporary text

Verify that the route is loading in the browser and you are able to view the text

Create a friendship model which will create a friendships table

rails generate model Friendship

then go to friendship.rb model file and add in

belongs\_to :user

belongs\_to :friend, :class\_name => 'User'

in the user.rb file

has\_many :friendships

has\_many :friends, through: :friendships

Add these two lines to the migration file within the create table:

t.belongs\_to :user

t.belongs\_to :friend, class: 'User'

Then run rake db:migrate to create the table

Do a commit and push to github

**Lecture 183 Friendship Views – Text Directions and Code**

In users\_controller.rb file within the my\_friends action, create an @friendships instance variable:

@friendships = current\_user.friends

In the navbar (\_navigation.html.erb) add a link to my\_friends\_path in place of Link3

In the views for my\_friends.html.erb under the app/views/users folder, add the following:

<%= render 'friends/lookup' %>

<% if @friendships.size > 0 %>

<% else %>

<div class='row col-lg-12'>

<p class="lead">You don't have any friends yet. Please add some!</p>

</div>

<% end %>

Copy over the code from stocks/list to this after <% if @friendships.size > 0 %> above:

<table class="table table-striped">

<thead>

<tr>

<th colspan="3" class="page-header">

<h2>My Friends</h2>

</th>

</tr>

<tr>

<th>Name</th>

<th>Email</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<% @friendships.each do |friend| %>

<tr>

<td><%= friend.full\_name %></td>

<td><%= friend.email %></td>

<td>

<%= link\_to "View Profile", user\_path(friend), class: "btn btn-primary btn-xs" %>

<%= link\_to 'Remove Friend',

friendship\_path(friend),

:method => :delete,

:data => { :confirm => 'Are you sure?' },

:class => 'btn btn-xs btn-danger' %>

</td>

</tr>

<% end %>

</tbody>

</table>

Now add the following routes to routes.rb file:

resources :users, only: [:show]

resources :friendships

**Lecture 185 Search Friends from UI – Text Directions and Code**

Create a folder called friends under app/views folder and within it create a partial called \_lookup.html.erb and fill it in (You can copy over code from stocks/lookup partial and make updates as below):

<div id="friend-lookup">

<h3>Search for Friends</h3>

<%= form\_tag search\_friends\_path, remote: true, method: :get, id: 'friend-lookup-form' do %>

<div class="form-group row no-padding text-center col-md-12">

<div class="col-md-10">

<%= text\_field\_tag :search\_param,

params[:search\_param],

placeholder: 'First Name, Last Name or Email',

autofocus: true,

class: 'form-control search-box input-lg' %>

</div>

<div class="col-md-2">

<%= button\_tag(type: :submit, class: 'btn btn-lg btn-success') do %>

<i class='fa fa-search'></i> Look up a friend

<% end %>

</div>

</div>

<% end %>

<%= render 'common/spinner' %>

<% if @users %>

<% if @users.size > 0 %>

<h3>Search Results</h3>

<div id="friend-lookup-results" class="well results-block col-md-10">

<table class="search-results-table col-md-12">

<tbody>

<% @users.each do |user| %>

<tr>

<td><strong>Name:</strong> <%= user.full\_name %></td>

<td><strong>Email:</strong> <%= user.email %></td>

<td><strong>Profile:</strong> <%= link\_to "View Profile", user\_path(user), class: "btn btn-xs btn-success" %>

<% if current\_user.not\_friends\_with?(user.id) %>

<%= link\_to "Add as my friend", add\_friend\_path(user: current\_user, friend: user),

class: 'btn btn-xs btn-success', method: :post %>

<% else %>

<span class='label label-primary'>

You are friends

</span>

<% end %>

</td>

</tr>

<% end %>

</tbody>

</table>

</div>

<% else %>

<p class="lead col-md-12">

No people match this search criteria

</p>

<% end %>

<% end %>

<div id="friend-lookup-errors"></div>

</div>

Add routes for the views for search and add friend in the routes.rb file:

get 'search\_friends', to: 'users#search'

post 'add\_friend', to: 'users#add\_friend'

**Lecture 187 Ajax for Search – Text Directions and Code**

Create a friends.js file under app/assets/javascripts folder and fill it in by copying over the stocks.js file and changing all the stock to friend:

var init\_friend\_lookup;

init\_friend\_lookup = function(){

$('#friend-lookup-form').on('ajax:before', function(event, data, status){

show\_spinner();

});

$('#friend-lookup-form').on('ajax:after', function(event, data, status){

hide\_spinner();

});

$('#friend-lookup-form').on('ajax:success', function(event, data, status){

$('#friend-lookup').replaceWith(data);

init\_friend\_lookup();

});

$('#friend-lookup-form').on('ajax:error', function(event, xhr, status, error){

hide\_spinner();

$('#friend-lookup-results').replaceWith(' ');

$('#friend-lookup-errors').replaceWith('Person was not found.');

});

}

$(document).ready(function() {

init\_friend\_lookup();

})

Now add search and add\_friend methods to users\_controller.rb file:

def search

@users = User.search(params[:search\_param])

if @users

@users = current\_user.except\_current\_user(@users)

render partial: 'friends/lookup'

else

render status: :not\_found, nothing: true

end

end

def add\_friend

@friend = User.find(params[:friend])

current\_user.friendships.build(friend\_id: @friend.id)

if current\_user.save

redirect\_to my\_friends\_path, notice: "Friend was successfully added."

else

redirect\_to my\_friends\_path, flash[:error] = "There was an error with adding user as friend."

end

end

**Lecture 189 Search Methods – Text Directions and Code**

Now we have to add a few methods in the user.rb model file

def not\_friends\_with?(friend\_id)

friendships.where(friend\_id: friend\_id).count < 1

end

def except\_current\_user(users)

users.reject {|user| user.id == self.id}

end

def self.search(param)

return User.none if param.blank?

param.strip!

param.downcase!

(first\_name\_matches(param) + last\_name\_matches(param) + email\_matches(param)).uniq

end

def self.first\_name\_matches(param)

matches('first\_name', param)

end

def self.last\_name\_matches(param)

matches('last\_name', param)

end

def self.email\_matches(param)

matches('email', param)

end

def self.matches(field\_name, param)

where("lower(#{field\_name}) like ?", "%#{param}%")

end

Remove turbolinks from application.js file if having issues with search

**Lecture 191 Show and Remove Friend and Wrap Finance Tracker – Text Directions and Code**

<h2>User details</h2>

<dl class="dl-horizontal">

<dt><strong>Full Name</strong></dt>

<dd><%= @user.full\_name %></dd>

<dt><strong>Email</strong></dt>

<dd><%= @user.email %></dd>

</dl>

<% if @user.stocks.size > 0 %>

<%= render 'stocks/list' %>

<% else %>

<p class="lead">This user is not tracking any stocks</p>

<% end %>

<%= link\_to "Back", my\_friends\_path, class: 'btn btn-primary' %>

**Section 9**

**Lecture 194 Start Photo App – Text Directions and Code**

Generate a new rails application:

rails new photo-app

Cd into the app:

cd photo-app

To setup homepage, generate a controller with an action:

rails generate controller welcome index

Then set the root route to this in the config/routes.rb file by uncommenting the root 'welcome#index' line

Test it out in the preview

You can update the page as you need, you can find this index view in the app/views/welcome folder in a file called index.html.erb

Initialize a git repo for the app and make a commit

Prepare the app for deployment to heroku by moving the sqlite3 gem to group development and then creating a group production and adding the gems necessary below it:

group :production do

gem 'pg'

gem 'rails\_12factor'

end

Then run:

bundle install --without production

Create a github repository in your github account for this app, follow steps to setup remote for this repo from your app

Make a commit of your code and push to your github repo

Create a heroku app by using heroku create

Rename the app to something you like by using heroku rename nameofyourchoice

Ensure your latest changes are committed using git status, if not, make a commit

To deploy your app to production:

git push heroku master

Then test out the URL to view your app in production

**Lecture 196 Setup Authentication System – Text Directions and Code**

First add the following gems to the gemfile:

gem 'devise'

gem 'twitter-bootstrap-rails'

gem 'devise-bootstrap-views'

Then run bundle install --without production

Then install devise:

rails generate devise:install

rails generate devise User

Pull up the migration file that just got created and uncomment the 4 lines under confirmable:

t.string :confirmation\_token

t.datetime :confirmed\_at

t.datetime :confirmation\_sent\_at

t.string :unconfirmed\_email

Pull up the user.rb model file under app/models and in the line for devise, add in a:

:confirmable,

after :registerable, entry

Run your migration now to create the users table:

rake db:migrate

In your application\_controller.rb file under app/controllers add in:

before\_action :authenticate\_user!

In your welcome\_controller.rb file under app/controllers add in:

skip\_before\_action :authenticate\_user!, only: [:index]

Run the following generators to install bootstrap themed styling:

rails generate bootstrap:install static

rails generate bootstrap:layout application # select Y to force override after hitting enter

rails generate devise:views:locale en

rails generate devise:views:bootstrap\_templates

In the application.css file under app/assets/stylesheets folder, right above the line that says \*= require\_tree add in the following line:

\*= require devise\_bootstrap\_views

**Lecture 198 Sending Email in Production – Text Directions and Code**

First add in your credit card details to your heroku account

Then enter in:

heroku addons:create sendgrid:starter

Set the sendgrid credentials you created for heroku:

heroku config:set SENDGRID\_USERNAME=enterintheusername

heroku config:set SENDGRID\_PASSWORD=enterinthepassword

To display your settings you can type in:

heroku config:get SENDGRID\_USERNAME

*[\*\*\*See Q&A to answer questions for C9\*\*\*]*

Open your .zshrc file and enter in the following as well (if using cloud9, you might not have the .zshrc file and will instead need to use the .profile or .bashrc file (additional details are in the discussions):

export SENDGRID\_USERNAME=enterintheusername

export SENDGRID\_PASSWORD=enterinthepassword

Then open a new terminal window for these to take effect

Under config/environment.rb file add in the following code at the bottom:

ActionMailer::Base.smtp\_settings = {

:address => 'smtp.sendgrid.net',

:port => '587',

:authentication => :plain,

:user\_name => ENV['SENDGRID\_USERNAME'],

:password => ENV['SENDGRID\_PASSWORD'],

:domain => 'heroku.com',

:enable\_starttls\_auto => true

}

Now update the development.rb file under config/environments folder and add the following two lines:

config.action\_mailer.delivery\_method = :test

config.action\_mailer.default\_url\_options = { :host => 'http://previewurlforyourapp'}

My preview url looks like this: [http://ruby-on-rails-123170.nitrousapp.com:3000](http://ruby-on-rails-123170.nitrousapp.com:3000/)

Now update the production.rb file under config/environments folder and add the following two lines:

config.action\_mailer.delivery\_method = :smtp

config.action\_mailer.default\_url\_options = { :host => 'yourherokuappname.herokuapp.com', :protocol => 'https'}

Test it out in development by signing up a user and then grabbing the confirmation link from the web output in your terminal and copying/pasting the link in your browser

**Lecture 200 Update Layout and Test Production Email – Text Directions and Code**

Go to application.html.erb file under app/views/layouts folder and remove the sidebar code, then add in the following right after the </ul> tag after Link3:

<ul class="nav navbar-right col-md-4">

<% if current\_user %>

<li class="col-md-8 user-name">

<%= link\_to ('<i class="fa fa-user"></i> ' + truncate(current\_user.email, length: 25)).html\_safe,

edit\_user\_registration\_path, title: 'Edit Profile' %>

</li>

<li class="col-md-1"> </li>

<li class="col-md-3 logout"><%= link\_to('Logout', destroy\_user\_session\_path,

class: 'btn btn-xs btn-danger', title: 'Logout', :method => :delete) %></li>

<% else %>

<li class="col-md-4 pull-right">

<%= link\_to('Sign In', new\_user\_session\_path, class: 'btn btn-primary', title: 'Sign In') %>

</li>

<% end %>

</ul>

Create a file under app/assets/stylesheets folder called custom.css.scss and fill it in with the following(edit as you need):

.user-name {

padding: 0 !important ;

padding-left: 5px;

padding-top: 15px !important;

text-align: center;

a {

color: black !important;

margin: 0 !important;

padding: 5px!important;

}

a:hover, a:focus {

color: #000 !important;

}

}

.logout {

padding-left: 0;

}

.nav.navbar-right {

padding-bottom: 10px;

padding-top: 5px;

}

.nav.navbar-nav {

.navbar-link {

border-radius: 5px;

color: #fff;

margin-top: 15px;

padding: 8px;

}

.navbar-link:focus, .navbar-link:hover {

background: #3071a9;

color: #fff;

}

li a {

margin-right: 5px;

}

}

.nav.navbar-right li {

.btn {

color: #fff !important;

margin-top: 5%;

}

.btn-danger:hover, .btn-danger:focus {

background-color: darken(#d9534f,20%) !important;

}

.btn-primary:hover, .btn-primary:focus {

background-color: darken(#428bca,20%) !important;

}

}

.btn-primary:visited, .btn-danger:visited {

color: #fff;

}

Open your devise.rb file under config/initializers folder and change the from address in the line:

config.mailer\_sender = 'changethis@example.com'

Deploy to heroku (after doing local commit) and test out app, remember to run:

heroku run rake db:migrate

# to run all your pending migrations in production

**Lecture 202 Build Homepage – Text Directions and Code**

Open your index.html.erb page under app/views/welcome folder and fill in the code (or add styling as you like):

<% if current\_user %>

<h3>Welcome: <%= current\_user.email %></h3>

<% else %>

<div class="jumbotron">

<h2>Welcome to the Photo Management App!</h2>

<p class="lead">

You'll love managing your Photos with our application. Sign up!

</p>

</div>

<div class="row">

<div class="plans clearfix">

<div class="col-lg-3 col-lg-offset-3 plan">

<h2>Premium Plan</h2>

<div class="price pull-right">

Price: $10

</div>

<div class="features">

<ul>

<li>Unlimited Image Uploads</li>

<li>Responsive design</li>

<li>Access anywhere</li>

</ul>

</div>

<p>

<%= link\_to 'Sign Up', new\_user\_registration\_path, class: 'btn btn-primary sign-up' %>

</p>

</div>

<div class="col-lg-3 plan">

<h2>Amaze Plan</h2>

<div class="price pull-right">

Price: $20

</div>

<div class="features">

<ul>

<li>Unlimited Image Uploads</li>

<li>Responsive design</li>

<li>Access anywhere</li>

<li class='extra'>Unlimited projects</li>

</ul>

</div>

<p>

<%= link\_to 'Sign Up', new\_user\_registration\_path, class: 'btn btn-primary sign-up' %>

</p>

</div>

</div>

</div>

<% end %>

Then open your custom.css.scss file under app/assets/stylesheets folder and add the following to the bottom of the file, note: the file name in the asset-url for background-image below needs to be the name of the image file you upload to your app/assets/images folder:

.features {

ul {

margin-left: 0;

padding-left: 15px;

}

}

.jumbotron {

text-align: center;

background-image: asset-url('meeting.png');

background-size: cover;

color: #fff;

font-family: 'Helvetica Neue' !important;

font-stretch: expanded;

padding-top: 25px;

padding-bottom: 25px;

margin-left: 15px;

margin-right: 15px;

text-shadow: -1px 0 #555, 0 1px #555, 1px 0 #555, 0 -1px #555;

/\*

h2 {

font-size: 35px !important;

.lead {

font-family: 'Helvetica Neue' !important;

font-size: 30px !important;

}

} \*/

}

.no-left-padding {

padding-left: 0 !important;

}

.listing {

list-style: none;

padding-left: 0;

}

In the application.html.erb page under app/views/layouts folder, change the following to col-lg-12 from col-lg-9:

<div class="container">

<div class="row">

<div class="col-lg-12">

<%= bootstrap\_flash %>

<%= yield %>

Go to stripe.com and sign-up for an account

**Lecture 204 Stripe and Payment Introduction – Text Directions and Code**

Guide for basic rails implementation of Stripe checkout:

<https://stripe.com/docs/checkout/guides/rails>

Note down your test secrety key and your test publishable key

In your gemfile add the stripe gem:

gem 'stripe'

Then bundle install --without production

Create a file named stripe.rb within your config/initializers folder and fill it in:

Rails.configuration.stripe = {

:publishable\_key => ENV['STRIPE\_TEST\_PUBLISHABLE\_KEY'],

:secret\_key => ENV['STRIPE\_TEST\_SECRET\_KEY']

}

Stripe.api\_key = Rails.configuration.stripe[:secret\_key]

Open your .zshrc file and fill in

export STRIPE\_TEST\_SECRET\_KEY=yoursecrettestkeyfromstripe

export STRIPE\_TEST\_PUBLISHABLE\_KEY=yourpublishabletestkeyfromstripe

Open a new terminal window for these to take effect

Now set these for heroku from your terminal window type in:

heroku config:set STRIPE\_TEST\_SECRET\_KEY=yoursecrettestkey

heroku config:set STRIPE\_TEST\_PUBLISHABLE\_KEY=yourpublishabletestkey

**Lecture 206 Payment Model – Text Directions and Code**

Create a payment model:

rails generate model Payment email:string token:string user\_id:integer

Run the migration file to create the payments table:

rake db:migrate

In your user.rb model file under app/models folder enter in the following code:

has\_one :payment

accepts\_nested\_attributes\_for :payment

In your payment.rb model file under app/models folder enter in the following code for attr\_accessors and methods:

class Payment < ActiveRecord::Base

attr\_accessor :card\_number, :card\_cvv, :card\_expires\_month, :card\_expires\_year

belongs\_to :user

def self.month\_options

Date::MONTHNAMES.compact.each\_with\_index.map { |name, i| ["#{i+1} - #{name}", i+1]}

end

def self.year\_options

(Date.today.year..(Date.today.year+10)).to\_a

end

def process\_payment

customer = Stripe::Customer.create email: email, card: token

Stripe::Charge.create customer: customer.id,

amount: 1000,

description: 'Premium',

currency: 'usd'

end

end

**Lecture 208 Update Form for Credit Card Payment – Text Directions and Code**

In the app/views/devise/registrations folder, open the new.html.erb file and add this code below to the section right under the closing </div> below the password confirmation:

<%= fields\_for( :payment ) do |p| %>

<div class="row col-md-12">

<div class="form-group col-md-4 no-left-padding">

<%= p.label :card\_number, "Card Number", data: { stripe: 'label'} %>

<%= p.text\_field :card\_number, class: "form-control", required: true, data: { stripe: 'number'} %>

</div>

<div class="form-group col-md-2">

<%= p.label :card\_cvv, "Card CVV", data: { stripe: 'label'} %>

<%= p.text\_field :card\_cvv, class: "form-control", required: true, data: { stripe: 'cvv'} %>

</div>

<div class="form-group col-md-6">

<div class="col-md-12">

<%= p.label :card\_expires, "Card Expires", data: { stripe: 'label'} %>

</div>

<div class="col-md-3">

<%= p.select :card\_expires\_month, options\_for\_select(Payment.month\_options),

{ include\_blank: 'Month' },

"data-stripe" => "exp-month",

class: "form-control", required: true %>

</div>

<div class="col-md-3">

<%= p.select :card\_expires\_year, options\_for\_select(Payment.year\_options.push),

{ include\_blank: 'Year' },

class: "form-control",

data: { stripe: "exp-year" }, required: true %>

</div>

</div>

</div>

<% end %>

**Lecture 210 Javascript Events – Text Directions and Code**

Go to your application.html.erb file under app/views/layouts folder and right above the line for <%= javascript\_include\_tag "application" %> enter in the following:

<%= javascript\_include\_tag "https://js.stripe.com/v2/" %>

Go to new.html.erb file under app/views/devise/registrations folder and on top of the file add in the following code:

<script language="Javascript">

Stripe.setPublishableKey("<%= ENV['STRIPE\_TEST\_PUBLISHABLE\_KEY'] %>");

</script>

In the <%= form\_for line add a class of 'cc\_form' and make it look like below:

<%= form\_for(resource, :as => resource\_name, :url => registration\_path(resource\_name), html: { role: "form", class: 'cc\_form' }) do |f| %>

Under app/assets/javascripts folder, create a file called credit\_card\_form.js and fill it in with the following code:

$(document).ready(function() {

var show\_error, stripeResponseHandler, submitHandler;

submitHandler = function (event) {

var $form = $(event.target);

$form.find("input[type=submit]").prop("disabled", true);

//If Stripe was initialized correctly this will create a token using the credit card info

if(Stripe){

Stripe.card.createToken($form, stripeResponseHandler);

} else {

show\_error("Failed to load credit card processing functionality. Please reload this page in your browser.")

}

return false;

};

$(".cc\_form").on('submit', submitHandler);

stripeResponseHandler = function (status, response) {

var token, $form;

$form = $('.cc\_form');

if (response.error) {

console.log(response.error.message);

show\_error(response.error.message);

$form.find("input[type=submit]").prop("disabled", false);

} else {

token = response.id;

$form.append($("<input type=\"hidden\" name=\"payment[token]\" />").val(token));

$("[data-stripe=number]").remove();

$("[data-stripe=cvv]").remove();

$("[data-stripe=exp-year]").remove();

$("[data-stripe=exp-month]").remove();

$("[data-stripe=label]").remove();

$form.get(0).submit();

}

return false;

};

show\_error = function (message) {

if($("#flash-messages").size() < 1){

$('div.container.main div:first').prepend("<div id='flash-messages'></div>")

}

$("#flash-messages").html('<div class="alert alert-warning"><a class="close" data-dismiss="alert">×</a><div id="flash\_alert">' + message + '</div></div>');

$('.alert').delay(5000).fadeOut(3000);

return false;

};

});

**Lecture 212 Extend Devise Registration Controller – Text Directions and Code**

List of test credit cards for stripe link:

<https://stripe.com/docs/testing>

Create a registrations\_controller.rb file under app/controllers folder and fill it in:

class RegistrationsController < Devise::RegistrationsController

def create

build\_resource(sign\_up\_params)

resource.class.transaction do

resource.save

yield resource if block\_given?

if resource.persisted?

@payment = Payment.new({ email: params["user"]["email"],

token: params[:payment]["token"], user\_id: resource.id })

flash[:error] = "Please check registration errors" unless @payment.valid?

begin

@payment.process\_payment

@payment.save

rescue Exception => e

flash[:error] = e.message

resource.destroy

puts 'Payment failed'

render :new and return

end

if resource.active\_for\_authentication?

set\_flash\_message :notice, :signed\_up if is\_flashing\_format?

sign\_up(resource\_name, resource)

respond\_with resource, location: after\_sign\_up\_path\_for(resource)

else

set\_flash\_message :notice, :"signed\_up\_but\_#{resource.inactive\_message}" if is\_flashing\_format?

expire\_data\_after\_sign\_in!

respond\_with resource, location: after\_inactive\_sign\_up\_path\_for(resource)

end

else

clean\_up\_passwords resource

set\_minimum\_password\_length

respond\_with resource

end

end

end

protected

def configure\_permitted\_parameters

devise\_parameter\_sanitizer.for(:sign\_up).push(:payment)

end

end

Update the registrations route for devise users in the config/routes.rb file:

devise\_for :users, :controllers => { :registrations => 'registrations' }

In the next video the payment bug will be resolved where the first attempt at sign-up fails but the second attempt succeeds. This issue can be resolved by removing turbolinks. To remove turbolinks, simply remove the line that references turbolinks starting with //= in the application.js file under the app/assets/javascripts folder. In addition you can also remove the turbolinks gem from the gemfile and do a bundle install --without production.

Now it's time to deploy to heroku and test out the app! Don't forget to post a link to your heroku app in the discussions area! Congratulations on having completed customizing payments with Stripe!

**Lecture 214 Fix Conflict Bug – Text Directions and Code**

To remove turbolinks:

remove the line //= require turbolinks from the application.js file under app/assets/javascripts folder

Additionally you can also remove the turbolinks gem from the gemfile and do a bundle install --without production after it

**Lecture 216 Image Upload – Text Directions and Code**

Add the following gems to your gemfile:

gem 'carrierwave'

gem 'mini\_magick'

gem 'fog'

Then run bundle install --without production

Generate the Image resource:

rails generate scaffold Image name:string picture:string user:references

Run the migration to create the images table:

rake db:migrate

Add styling to all image views:

rails g bootstrap:themed Images

Enter Y through all the questions to override the existing image views

Open your user.rb model file under app/models and enter in the following line:

has\_many :images

Ensure your image.rb model file has the line belongs\_to :user

To generate an uploader:

rails generate uploader Picture

Add the following line to your image.rb model file to associate images with picture:

mount\_uploader :picture, PictureUploader

Pull up the \_form.html.erb partial under app/views/images folder and update it to make it look like below:

<%= form\_for @image, :html => { multipart: true, :class => "form-horizontal image" } do |f| %>

<% if @image.errors.any? %>

<div id="error\_expl" class="panel panel-danger">

<div class="panel-heading">

<h3 class="panel-title"><%= pluralize(@image.errors.count, "error") %> prohibited this image from being saved:</h3>

</div>

<div class="panel-body">

<ul>

<% @image.errors.full\_messages.each do |msg| %>

<li><%= msg %></li>

<% end %>

</ul>

</div>

</div>

<% end %>

<div class="control-group">

<%= f.label :name, :class => 'control-label' %>

<div class="controls">

<%= f.text\_field :name, :class => 'form-control' %>

</div>

<%= error\_span(@image[:name]) %>

</div>

<div class="control-group">

<%= f.label :picture, :class => 'control-label' %>

<div class="controls">

<%= f.file\_field :picture, accept: 'image/jpeg,image/gif,image/png' %>

</div>

<%= error\_span(@image[:picture]) %>

</div>

<%= f.submit nil, :class => 'btn btn-primary' %>

<%= link\_to t('.cancel', :default => t("helpers.links.cancel")),

images\_path, :class => 'btn btn-default' %>

<% end %>

Update your create action in the images\_controller.rb file under app/controllers folder by adding the line below under @image = Image.new...:

@image.user = current\_user

To display the image in the show page, open the show.html.erb file under app/views/images folder and update it as follows:

<%- model\_class = Image -%>

<div class="page-header">

<h1><%=t '.title', :default => model\_class.model\_name.human.titleize %></h1>

</div>

<dl class="dl-horizontal">

<dt><strong><%= model\_class.human\_attribute\_name(:name) %>:</strong></dt>

<dd><%= @image.name %></dd>

<dt></dt>

<dd><%= image\_tag(@image.picture.url, size: "300x300") if @image.picture? %></dd>

</dl>

<%= link\_to t('.back', :default => t("helpers.links.back")),

images\_path, :class => 'btn btn-default' %>

<%= link\_to t('.edit', :default => t("helpers.links.edit")),

edit\_image\_path(@image), :class => 'btn btn-default' %>

<%= link\_to t('.destroy', :default => t("helpers.links.destroy")),

image\_path(@image),

:method => 'delete',

:data => { :confirm => t('.confirm', :default => t("helpers.links.confirm", :default => 'Are you sure?')) },

:class => 'btn btn-danger' %>

**Lecture 218 Image Size Validations – Text Directions and Code**

In app/uploaders/picture\_uploader.rb file uncomment the following lines:

def extension\_white\_list

%w(jpg jpeg gif png)

end

Add the following to app/models/image.rb file:

validate :picture\_size

private

def picture\_size

if picture.size > 5.megabytes

errors.add(:picture, "should be less than 5MB")

end

end

In the app/views/images/\_form.html.erb partial add the following validation at the bottom:

<script type="text/javascript">

$('#image\_picture').bind('change', function() {

var size\_in\_megabytes = this.files[0].size/1024/1024;

if (size\_in\_megabytes > 5) {

alert('Maximum file size is 5MB.');

}

});

</script>

To install latest imagemagick:

sudo apt-get install imagemagick --fix-missing

Open your picture\_uploader.rb file within the app/uploaders folder and add the following two lines right below the class definition:

include CarrierWave::MiniMagick

process resize\_to\_limit: [300, 300]

Update the app/views/images/index.html.erb file and make it look like below:

<%- model\_class = Image -%>

<div class="page-header">

<h1><%=t '.title', :default => model\_class.model\_name.human.pluralize.titleize %></h1>

</div>

<table class="table table-striped">

<thead>

<tr>

<th><%= model\_class.human\_attribute\_name(:name) %></th>

<th><%= model\_class.human\_attribute\_name(:picture) %></th>

<th><%=t '.actions', :default => t("helpers.actions") %></th>

</tr>

</thead>

<tbody>

<% @images.each do |image| %>

<tr>

<td><%= link\_to image.name, image\_path(image) %></td>

<td><%= image\_tag image.picture.url, size: "100x100" %></td>

<td>

<%= link\_to t('.edit', :default => t("helpers.links.edit")),

edit\_image\_path(image), :class => 'btn btn-default btn-xs' %>

<%= link\_to t('.destroy', :default => t("helpers.links.destroy")),

image\_path(image),

:method => :delete,

:data => { :confirm => t('.confirm', :default => t("helpers.links.confirm", :default => 'Are you sure?')) },

:class => 'btn btn-xs btn-danger' %>

</td>

</tr>

<% end %>

</tbody>

</table>

<%= link\_to t('.new', :default => t("helpers.links.new")),

new\_image\_path,

:class => 'btn btn-primary' %>

**Lecture 220 Image Upload in Production – Text Directions and Code**

*DID NOT DO THIS SECTION (didn’t want to pay for amazon web service)*

In your picture\_uploader.rb file under app/uploaders folder, replace the line that says storage :file with the following:

if Rails.env.production?

storage :fog

else

storage :file

end

Sign up for Amazon web services at aws.amazon.com

Follow the video exactly as shown:

1) Create IAM user

2) Create S3 bucket

3) Create policy with s3 bucket details

4) Attach policy to IAM user created

Here is a sample policy, replace the code with your s3 bucket name as needed below:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:\*",

"Resource": [

"arn:aws:s3:::yours3bucketname",

"arn:aws:s3:::yours3bucketname/\*"

]

},

{

"Effect": "Allow",

"Action": "s3:ListAllMyBuckets",

"Resource": "arn:aws:s3:::\*"

}

]

}

**Lecture 222 Complete Prod Image Upload – Text Directions and Code**

*DID NOT DO THIS SECTION (didn’t want to pay for amazon web service)*

Set your credentials for amazon IAM user and S3 bucket with heroku:

heroku config:set S3\_ACCESS\_KEY=youraccesskeyforIAMuser

heroku config:set S3\_SECRET\_KEY=yoursecretykeyforIAMuser

heroku config:set S3\_BUCKET=yours3bucketname

Under config/initializers folder create a file called carrier\_wave.rb and fill it in:

if Rails.env.production?

CarrierWave.configure do |config|

config.fog\_credentials = {

:provider => 'AWS',

:aws\_access\_key\_id => ENV['S3\_ACCESS\_KEY'],

:aws\_secret\_access\_key => ENV['S3\_SECRET\_KEY']

}

config.fog\_directory = ENV['S3\_BUCKET']

end

end

Do a commit of your code, push to github, deploy to heroku, run any pending migrations and test it out!