Assignment 3: Mining Gems

CS390P: Web Application Architecture

John Samson

October 28th, 2018

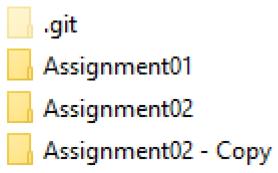
Abstract

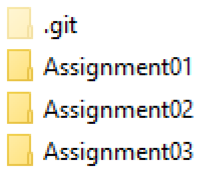
In our work so far, we've covered many different functions of the built-in Ruby on Rails framework. One of the more attractive features of Rails is its handling of additional packages and libraries; a wealth of features are available to us by installing different *gems*. Here we'll continue expanding the functionality and security of our website using gem files. First, we'll use the Bootstrap gem to easily create an attractive *navigation bar*, as currently we have no means of navigating between our different pages. We'll briefly adjust some Cascading Style Sheet (*CSS)* options within our application layout file, then use the Devise gem to implement basic *user authentication*.

Introduction

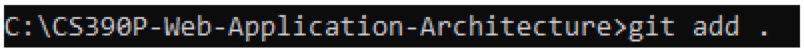
We'll continue our work on our University website, focusing on improved navigation and security with a *navigation bar* and *user authentication*. Almost every website today has some sort of compact navigation system that allows users to a wide variety of features and pages within a single compact interface. First, we'll simply add a table of links within our application *layout* file. Then we'll begin using *Ruby gems* to quickly add functionality to our website. We'll create a far more visually pleasing and dynamic navigation bar using the Bootstrap gem. We'll supplement this visual upgrade with a brief exploration of Cascading Style Sheets and how we can embed style information within HTML tags to format the margins of our website. We'll finish our work here by using the Devise gem to quickly add a simple user authentication system, leaving us with a far more complete and secure website than we created previously.

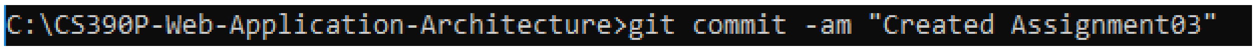
- Let's setup a new Project, building on our previous work. Make a copy of the previous assignment:

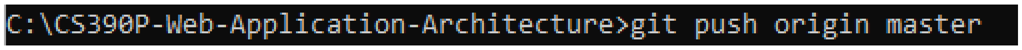

.git
Assignment01
Assignment02
Assignment02 - Copy


.git
Assignment01
Assignment02
Assignment03

- Now add the new folder and files to the Git repository we setup previously:


```
C:\CS390P-Web-Application-Architecture>git add .
```

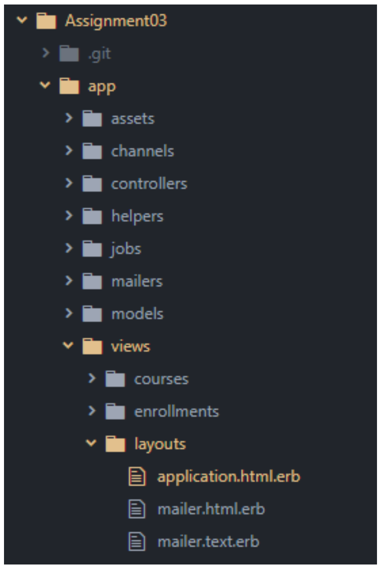

```
C:\CS390P-Web-Application-Architecture>git commit -am "Created Assignment03"
```


```
C:\CS390P-Web-Application-Architecture>git push origin master
```

Simple Navigation bar using Layout file

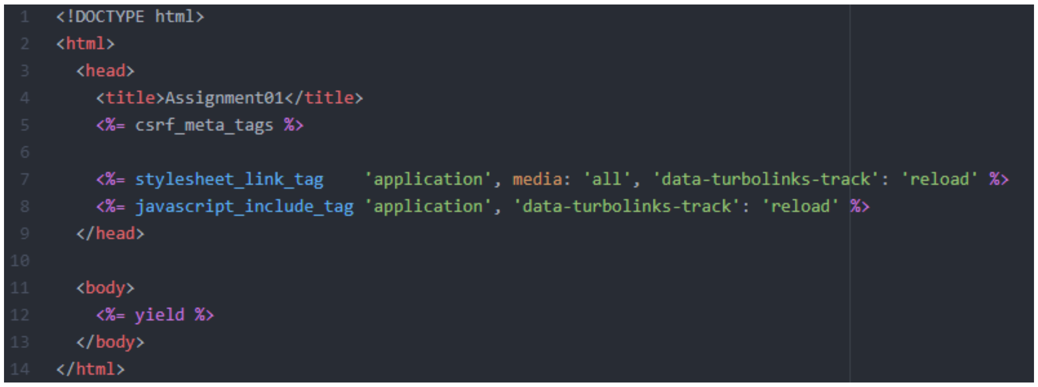
To begin we'll create a simple HTML navigation bar, perhaps if only to contrast it to the one we will create using the Bootstrap gem in the next step. We can edit the '`application.html.erb`' *layout* file to create a table that will be displayed on every subpage of our website. While this option certainly works, we'll find the Bootstrap option far more pleasing to the eye.

- A useful tool for creating Rails applications is the Atom Text Editor, available here: https://atom.io/. Atom comes with a built-in file browser, and many other useful features for programmers. We'll be using Atom moving forward for its easy navigation through the many different folders of our Rails application.
- Navigate to '`app/views/layouts`' and open the '`application.html.erb`' layout file:



```
Assignment03
  .git
  app
    assets
    channels
    controllers
    helpers
    jobs
    mailers
    models
    views
      courses
      enrollments
      layouts
        application.html.erb
        mailer.html.erb
        mailer.text.erb
```

- Here we can see the layout file that describes the basic HTML structure and style information for our entire application:

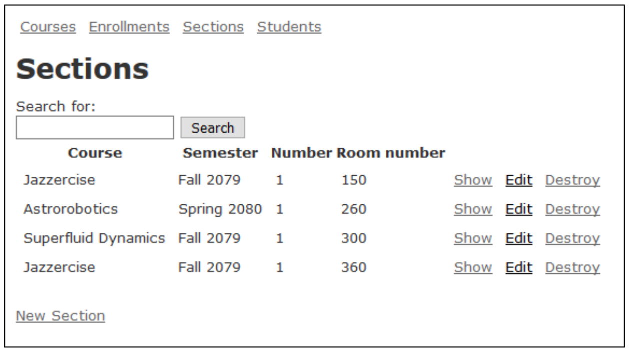


```
<!DOCTYPE html>
<html>
  <head>
    <title>Assignment01</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

- Add the following code block to create a simple table of links to our different pages using *URL helpers*. URL helpers work with routes.rb file to allow us to conveniently call the *link_to* function from our views and controllers (Learn.co, 2018):

```
<table>
  <td><%= link_to "Courses", courses_path %></td>
  <td><%= link_to "Enrollments", enrollments_path %></td>
  <td><%= link_to "Sections", sections_path %></td>
  <td><%= link_to "Students", students_path %></td>
</table>
```

- We can see the result is simple yet effective enough to allow us to navigate between pages:



Courses Enrollments Sections Students

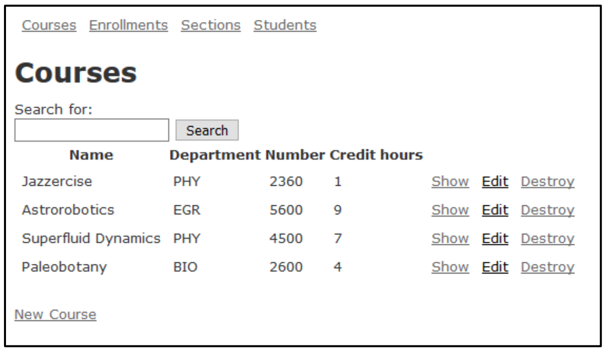
Sections

Search for:

Search

Course	Semester	Number	Room number			
Jazzercise	Fall 2079	1	150	Show	Edit	Destroy
Astrorobotics	Spring 2080	1	260	Show	Edit	Destroy
Superfluid Dynamics	Fall 2079	1	300	Show	Edit	Destroy
Jazzercise	Fall 2079	1	360	Show	Edit	Destroy

New Section



Courses Enrollments Sections Students

Courses

Search for:

Search

Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy
Astrorobotics	EGR	5600	9	Show	Edit	Destroy
Superfluid Dynamics	PHY	4500	7	Show	Edit	Destroy
Paleobotany	BIO	2600	4	Show	Edit	Destroy

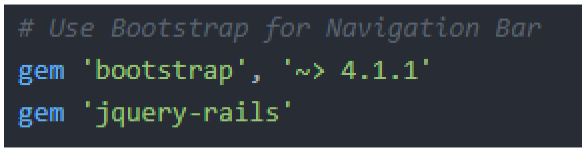
New Course

We see that we can create a simple table within our application layout file to add a navigation bar to every subpage of our website. Let us explore a better-looking option using our very first Ruby *gem* file.

Navigation bar using Bootstrap Gem

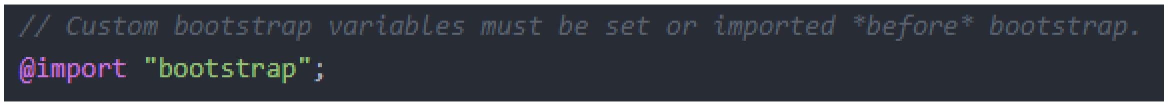
Rails provides a wealth of additional features via packages and libraries called *gems*. Gems are managed in the *Gemfile* in the root directory of your Rails project. We'll use a gem called Bundler to manage and install our gems, and another popular frontend framework gem, Bootstrap to create a better looking navigation bar.

- First enter ‘`gem install bundle`’ to install Bundler. Now let’s add the Bootstrap gem and the necessary JQuery dependency gem for it. In your Gemfile, add the following lines:



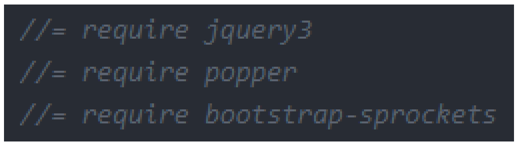
```
# Use Bootstrap for Navigation Bar
gem 'bootstrap', '~> 4.1.1'
gem 'jquery-rails'
```

- Bootstrap works with Cascading Style Sheets (CSS) within HTML, so we need to allow our application to use it. We’ll need to update the file type from ‘`.css`’ to ‘`.scss`’ (“Sassy” CSS) format. SCSS format offers certain expanded functions from normal CSS stylesheets. First change the file format, rename ‘`app/assets/stylesheets/application.css`’ to ‘`app/assets/stylesheets/application.scss`’ and edit the file to include the following (Blevins, 2018):



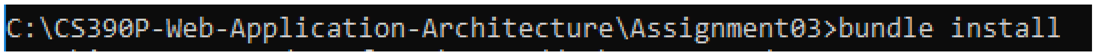
```
// Custom bootstrap variables must be set or imported *before* bootstrap.
@import "bootstrap";
```

- We need to include the following statements in the ‘`app/javascripts/application.js`’ file. We also need to remove the line ‘`//= require_tree .`’:



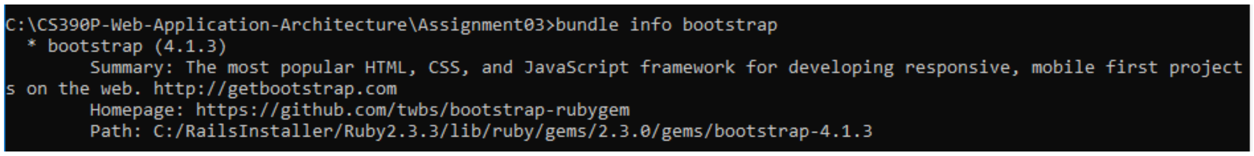
```
//= require jquery3
//= require popper
//= require bootstrap-sprockets
```

- Now that we have our Gemfile, stylesheets, and ‘application.js’ file setup, let’s run bundle install to install our new gems:



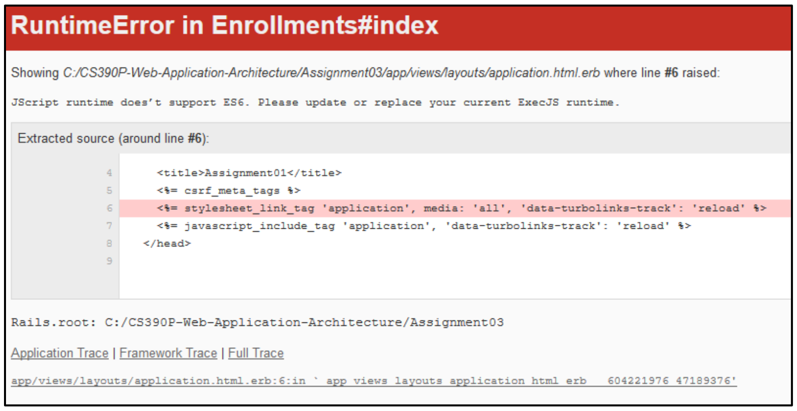
```
C:\CS390P-Web-Application-Architecture\Assignment03>bundle install
```

- We can use the ‘`bundle info`’ command to verify our installation of the Bootstrap gem:



```
C:\CS390P-Web-Application-Architecture\Assignment03>bundle info bootstrap
  * bootstrap (4.1.3)
        Summary: The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first project
s on the web. http://getbootstrap.com
        Homepage: https://github.com/twbs/bootstrap-rubygem
        Path: C:/RailsInstaller/Ruby2.3.3/lib/ruby/gems/2.3.0/gems/bootstrap-4.1.3
```

- It is here that we encountered a very precarious runtime error message:



RuntimeError in Enrollments#index

Showing *C:/CS390P-Web-Application-Architecture/Assignment03/app/views/layouts/application.html.erb* where line **#6** raised:

```
JScript runtime does't support ES6. Please update or replace your current ExecJS runtime.
```

Extracted source (around line **#6**):

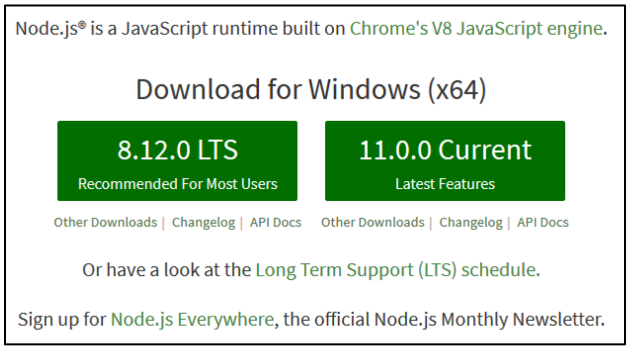
```
    <title>Assignment01</title>
    <%= csrf_meta_tags %>
    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
  </head>
```

```
Rails.root: C:/CS390P-Web-Application-Architecture/Assignment03
```

Application Trace | Framework Trace | Full Trace

```
app/views/layouts/application.html.erb:6:in ` app views layouts application html erb   604221976 47189376'
```

- After some amount of time troubleshooting, we realized we're missing the Node.js runtime environment, available here: https://nodejs.org/en/. In order for Bootstrap to run JavaScripts on our website, ExecJS needs the Node.js runtime dependency. Download and install the Node.js runtime environment, choosing between stable and current builds, as shown below:



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

8.12.0 LTS
Recommended For Most Users

11.0.0 Current
Latest Features

Other Downloads | Changelog | API Docs   Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

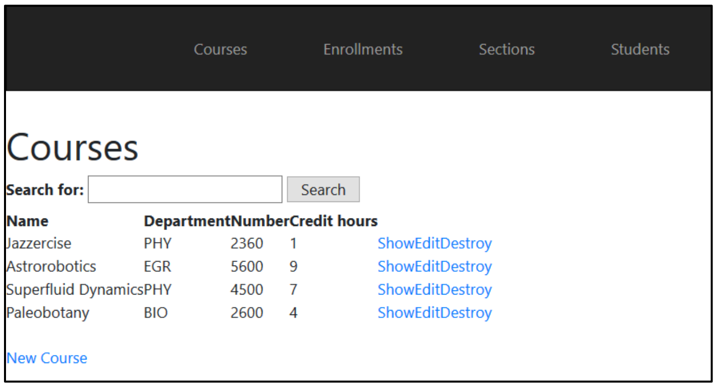
- Add the following lines in the '`app/views/layouts/application.html.erb`' layout file:

```
<meta charset="utf-8">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
```

- Now add the following block to create a *navbar* class using the Bootstrap gem. Here we will create an unordered list using the URL helpers to quickly reference the links we need (Bootstrap, 2018):

```
<nav class="navbar navbar-inverse">
  <ul class="nav">
    <li class="nav-link"><%= link_to 'Courses', courses_path, :class => 'navbar-link' %></li>
    <li class="nav-link"><%= link_to 'Enrollments', enrollments_path, :class => 'navbar-link' %></li>
    <li class="nav-link"><%= link_to 'Sections', sections_path, :class => 'navbar-link' %></li>
    <li class="nav-link"><%= link_to 'Students', students_path, :class => 'navbar-link' %></li>
  </ul>
</nav>
```

- Here we have our working navbar using Bootstrap, a striking improvement over our simple table of links.



Courses

Search for: [ ] Search

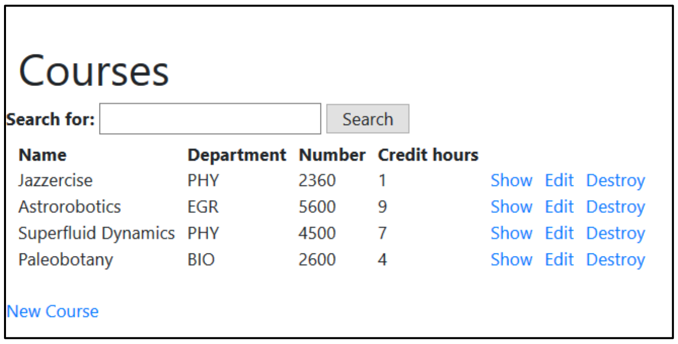
Name	Department	Number	Credit hours	
Jazzercise	PHY	2360	1	ShowEditDestroy
Astrorobotics	EGR	5600	9	ShowEditDestroy
Superfluid Dynamics	PHY	4500	7	ShowEditDestroy
Paleobotany	BIO	2600	4	ShowEditDestroy

New Course

- Before we can enjoy our upgraded navbar, we can't help but notice how crowded the text of our index table is. Let's fix that by designating the CSS stylesheets for the '`application.html.erb`' application layout file. Here we can add the following block within the *header* to *pad* any element within the table header (td) and table data (td) tag:

```
<style>
th, td {
    padding-left: 7px;
}
</style>
```

- Now each element in our Courses index table has more 'breathing room':



Courses

Search for: [ ] Search

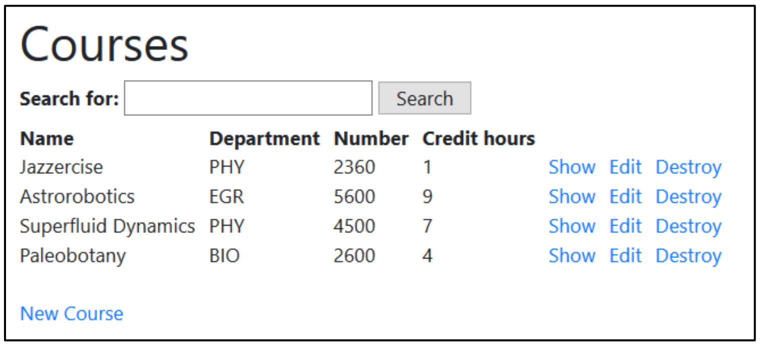
Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy
Astrorobotics	EGR	5600	9	Show	Edit	Destroy
Superfluid Dynamics	PHY	4500	7	Show	Edit	Destroy
Paleobotany	BIO	2600	4	Show	Edit	Destroy

New Course

- We can do the same for the 'Header 1' (h1) and division (div) HTML tags:

```
<style>
th, td, h1, div {
    padding-left: 7px;
}
</style>
```

- We can see that CSS allows us to quickly change the stylesheets for our website, using the `<style>` tag within the application layout file.



Courses

Search for: [ ] Search

Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy
Astrorobotics	EGR	5600	9	Show	Edit	Destroy
Superfluid Dynamics	PHY	4500	7	Show	Edit	Destroy
Paleobotany	BIO	2600	4	Show	Edit	Destroy

New Course

While Bootstrap allows us to quickly create a visually pleasing navigation bar for our website, it is not our only option to enhance our application. Rails developers frequently use gems to expand functionality in a variety of ways; as the saying goes, for nearly any website function, "there's a gem for that".

Authentication with the Devise Gem

While our University website has never looked better, all of the information in our database is available to anyone that can access our application. We need to implement a *user authentication* system to allow only verified users access to our website and our database. For this we'll use the Devise gem to quickly create a user authentication system, complete with the appropriate database model and sign in page.

- In the Gemfile, add Devise:

```
gem 'devise'
```

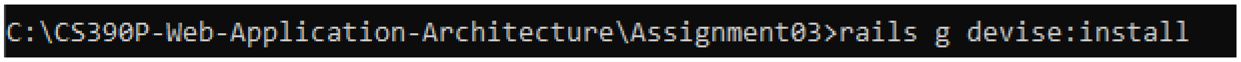
- Install Devise via Bundler:

```
C:\CS390P-Web-Application-Architecture\Assignment03>bundle install
```

- Verify the Devise installation:

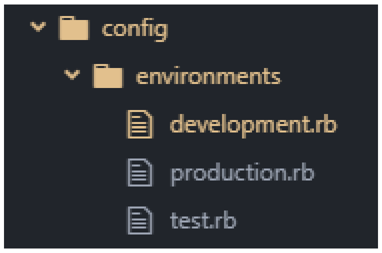
```
C:\CS390P-Web-Application-Architecture\Assignment03>bundle info devise
  * devise (4.5.0)
        Summary: Flexible authentication solution for Rails with Warden
        Homepage: https://github.com/plataformatec/devise
        Path: C:/RailsInstaller/Ruby2.3.3/lib/ruby/gems/2.3.0/gems/devise-4.5.0
```

- Enter the following command to generate the necessary files for Devise.

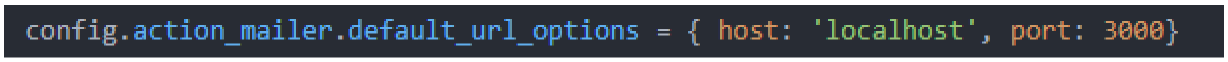


```
C:\CS390P-Web-Application-Architecture\Assignment03>rails g devise:install
```

- Edit the '`development.rb`' configuration file:

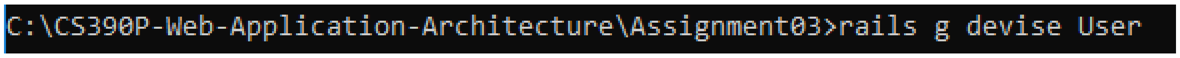


- Add the following line to specify the default URL for our website. For now, we'll stick with '`localhost:3000`'.



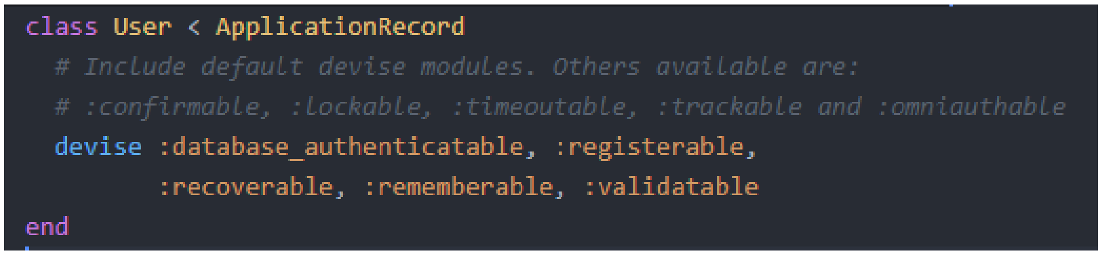
```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000}
```

- Now let's generate the appropriate User model for our authentication system. Note that we could implement a user *authorization* system here, in a similar manner. We would simply need to generate another model for a different user authorization level ('Admin' or 'Teacher', for example). Enter the following command to create the User model (Devise, 2018):



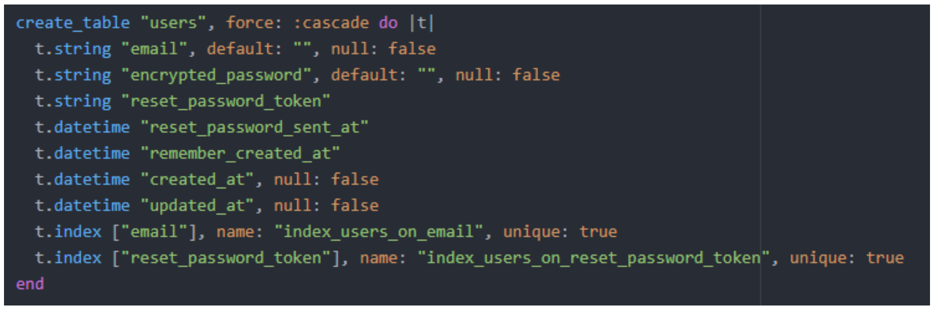
```
C:\CS390P-Web-Application-Architecture\Assignment03>rails g devise User
```

- We can see that Devise has generated the appropriate User model in the 'app/models/user.rb' file:



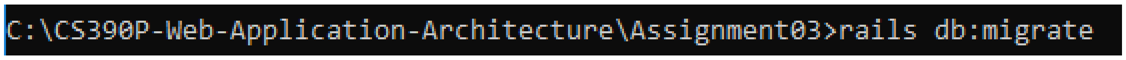
```
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable
end
```

- We can also see that the database schema '`db/schema.rb`' has also been updated.



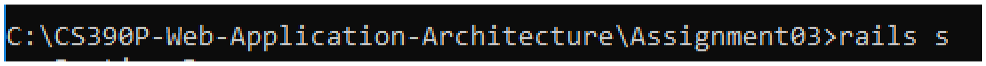
```
create_table "users", force: :cascade do |t|
  t.string "email", default: "", null: false
  t.string "encrypted_password", default: "", null: false
  t.string "reset_password_token"
  t.datetime "reset_password_sent_at"
  t.datetime "remember_created_at"
  t.datetime "created_at", null: false
  t.datetime "updated_at", null: false
  t.index ["email"], name: "index_users_on_email", unique: true
  t.index ["reset_password_token"], name: "index_users_on_reset_password_token", unique: true
end
```

- We’ve made major changes to the database, so let’s perform a database migration:

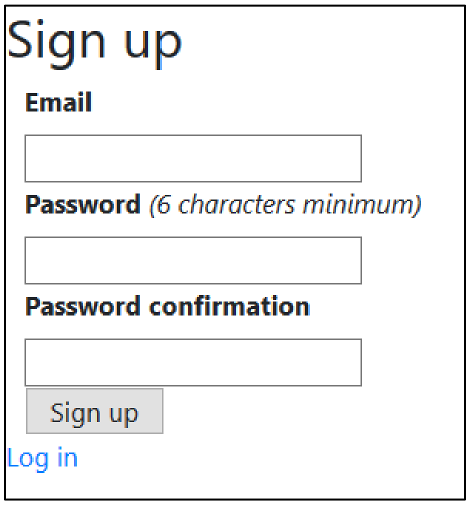


```
C:\CS390P-Web-Application-Architecture\Assignment03>rails db:migrate
```

- When we run the server again, we’re greeted with the Sign-up page Devise created for us. Let’s create our first User by entering an email address and password.



```
C:\CS390P-Web-Application-Architecture\Assignment03>rails s
```

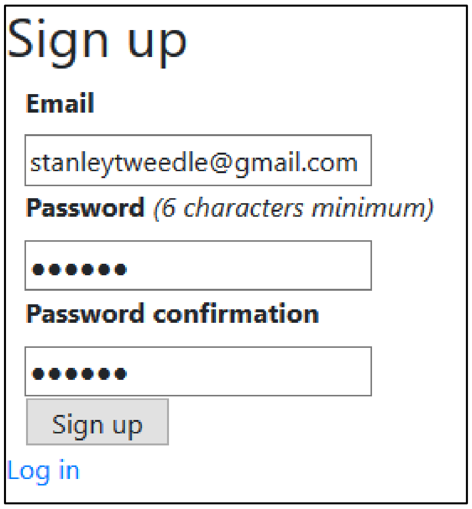


Sign up

Email

Password *(6 characters minimum)*

Password confirmation



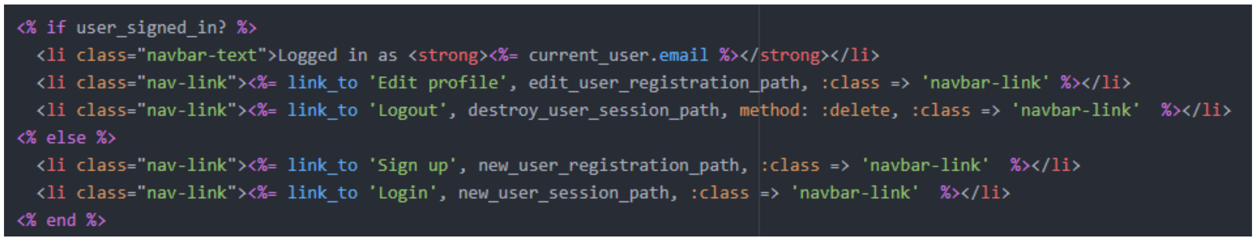
Sign up

Email stanleytweedle@gmail.com

Password *(6 characters minimum)* ●●●●●●

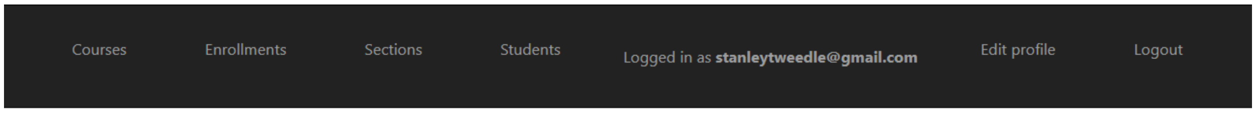
Password confirmation ●●●●●●

- We want to let the User know when they are logged into our site. Add the following block to your unordered list (ul) within the navbar we created previously in the ‘`application.html.erb`’ application layout file: Here we refer to the ‘`user_signed_in?`’ helper to display the User’s email if they’re logged in:

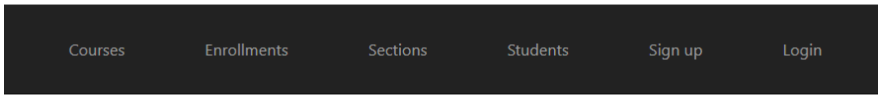


```
<% if user_signed_in? %>
  <li class="navbar-text">Logged in as <strong><%= current_user.email %></strong></li>
  <li class="nav-link"><%= link_to 'Edit profile', edit_user_registration_path, :class => 'navbar-link' %></li>
  <li class="nav-link"><%= link_to 'Logout', destroy_user_session_path, method: :delete, :class => 'navbar-link'  %></li>
<% else %>
  <li class="nav-link"><%= link_to 'Sign up', new_user_registration_path, :class => 'navbar-link'  %></li>
  <li class="nav-link"><%= link_to 'Login', new_user_session_path, :class => 'navbar-link'  %></li>
<% end %>
```

- Here we have Stanley Tweedle logged in:



- and logged out:



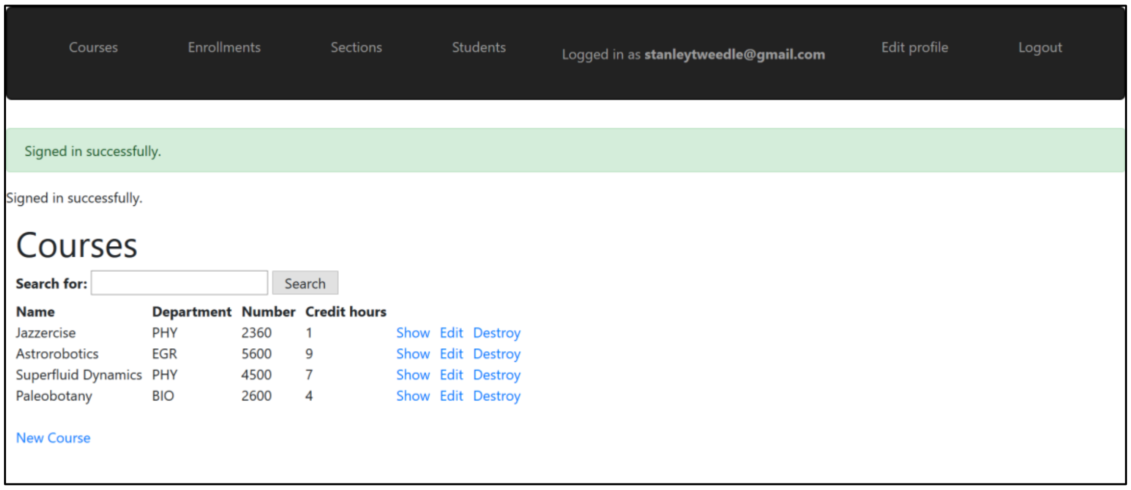
- Now let's show the user directly whenever they successfully login or logout. Here we'll access the *notice* and *alert* parameters within the layout file '`app/views/layouts/application.html.erb`'. Add the following block before the `<%= yield %>` line within the *body* of the layout file (Steininger, 2018):

```
<% if notice %>
  <p class="alert alert-success"><%= notice %></p>
<% end %>
<% if alert %>
  <p class="alert alert-danger"><%= alert %></p>
<% end %>
<%= yield %>
```

- Because we access the *notice* parameter in the layout file, we no longer neet it in each individual Show view file. Remove the following line from the '`app/views/courses/show.html.erb`' Show Course view file:

```
<p id="notice"><%= notice %></p>
```

- Likewise remove the preceding line from the '`app/views/enrollments/show.html.erb`', '`app/views/sections/show.html.erb`' and the '`app/views/students/show.html.erb`' Show view files.



Signed in successfully.

Signed in successfully.

Courses

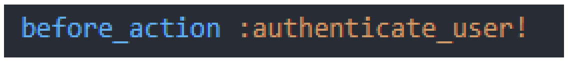
Search for: Search

Name	Department	Number	Credit hours			
Jazzercise	PHY	2360	1	Show	Edit	Destroy
Astrorobotics	EGR	5600	9	Show	Edit	Destroy
Superfluid Dynamics	PHY	4500	7	Show	Edit	Destroy
Paleobotany	BIO	2600	4	Show	Edit	Destroy

New Course

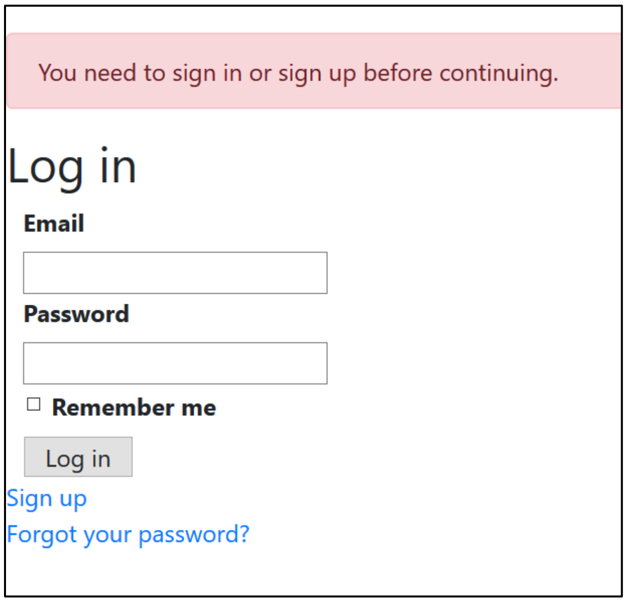
Our application is nearly fully functional, albeit a simple instructional case. To add the most basic layer of security for our website and our database, we must specify that we don't want to display *any* information to an unauthenticated user. Let us define a *before_action* within the application controller to authenticate the user before any information is displayed.

- Edit the ‘`app/controllers/application_controller.rb`’ application controller file and add the following *before_action* to allow only authenticated users into our website:



```
before_action :authenticate_user!
```

- Accessing any page of our site now while *not* logged in prompts us to Log in or Sign up:



You need to sign in or sign up before continuing.

If we want to continue making our site look nicer, we can update the stylesheets for each page, and perform lots of small fixes and tinker with our Navigation bar. A logical next step might be to move our Search bar into the Navigation bar. Once again, we can see that gems allow us to quickly add new features and functions (along with their appropriate models, views, and database tables) to our web applications.

Conclusion

Much like Rails itself, gems allow us to use pre-written frameworks and libraries to quickly add the features we need. We find that given a simple problem; create a navbar, we have dozens of different gems available, each with many different formatting options we can use. Gems will do much of the hard work for us; when we added our authentication with Devise, all of the models and views were automatically generated for us. While creating web applications we need to keep in mind that it is just as important to create *functional* and *secure* applications, as it is to create *easy to navigate* applications that are pleasing to the eye.

Bibliography

Learn.co (2018). *Rails Url Helpers Readme* [Website]. Retrieved from

https://learn.co/lessons/rails-url-helpers-readme

Bootstrap (2018). *Navbar* [Website]. Retrieved from

https://getbootstrap.com/docs/4.0/components/navbar/.

w3schools (2018). *Bootstrap Navigation Bar* [Website]. Retrieved from

https://www.w3schools.com/bootstrap/bootstrap_navbar.asp

Blevins, M. (2018). *Adding a navbar to your Rails app with Bootstrap* [Blog]. Retrieved from

https://medium.com/@mblevdev/adding-a-navbar-to-your-rails-app-with-bootstrap-a16cbd887f14

Devise (2018). *Devise readme* [Website]. Retrieved from

https://github.com/plataformatec/devise

Steininger, P. (2018). *Adding Authentication with Devise* [Website]. Retrieved from

https://guides.railsgirls.com/devise

Instructor Feedback

What was too difficult, too easy?

By far the most difficult part of this assignment was troubleshooting the bug I had. Once I found out that I needed Node.js, the assignment itself was quite straightforward.

What would have made the learning experience better?

Perhaps more time spent learning CSS, something I didn’t get too deep into in this assignment.

What did you learn?

I learned how, much like Rails itself, gems are a powerful way to quickly add functions and features to a website. While it can be a bit hard to setup the various configuration files and dependencies needed for a given gem, its satisfying to see how quickly a gem can give us the features that we want.

How did you learn it?

For this assignment I found a number of helpful online tutorials, including some official documentation for Devise to help. As for the navbar, once I had Bootstrap installed I spent a great deal of time simply fiddling with the various features of the *navbar* class. I’m finding that much like designing an effective Word or PowerPoint document, designing web applications has a similar level of fine-tuning and design options available. One can spend hours working until a site looks “just right”.