

# **Assignment 1: Ruby on Rails**

CS390P: Web Application Architecture

John Samson

September 23<sup>rd</sup>, 2018

## Abstract

For most of us, it is hard to remember a world without the internet, or the world wide web. While mobile phones have grown in popularity, the world wide web is still an important means of accessing information in 2018. Web development has changed significantly since its rise to popularity in the 1990s. In the 7<sup>th</sup> grade I remember handwriting HTML to create a simple website; slow and laborious work. Today we use powerful frameworks to launch a new web server in minutes, and avoid writing as much HTML as possible.

We'll be using the popular Ruby on Rails framework to create a simple web application that keeps track of courses and sections for a university. Ruby on Rails incorporates the object-oriented Ruby language within the powerful Rails web application framework. We'll be using SQLite3 as a database management tool to create new database entries for different courses, edit and destroy them, all without writing a single SQL server query. We'll guide the reader through basic Ruby on Rails setup on Windows, learn how to create a *scaffold* to create a simple pre-configured website using rails, and learn how to setup a *one-to-many* relationship between courses and sections, and refer to one database table from another. In addition, we'll configure Git from the windows command line, and link our local and remote repositories to easily backup our work and see differences in our code between commits.

## Introduction

Today we have two simple goals: Setup Ruby on Rails and create a simple website for a university. While Rails provides an extremely streamlined framework for creating web servers, setting up Rails can be difficult for newcomers. For this reason, we have chosen to use the pre-packaged ‘RailsInstaller’, available here:

<http://railsinstaller.org/en>

RailsInstaller installs all of the basic components we will need, in addition to setting up our *environment path* variables in Windows. Advanced readers hoping for more control over can download and install individual components at their discretion.

Once we have Ruby on Rails setup properly, we’ll create a basic website *scaffold* to see the awesome power of Rails firsthand. We’ll then set out to create a more advanced scaffold, one that initializes our ‘sections’ page with *references* to our ‘courses’ database. This will allow us to create a simple dropdown menu of possible courses when creating a new section, and demonstrate how different pages of the same website can share information with each other.

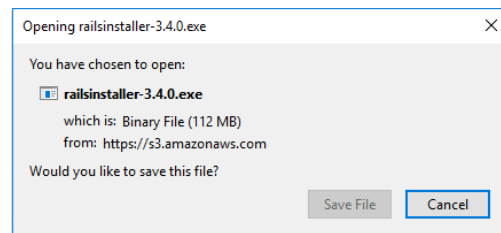
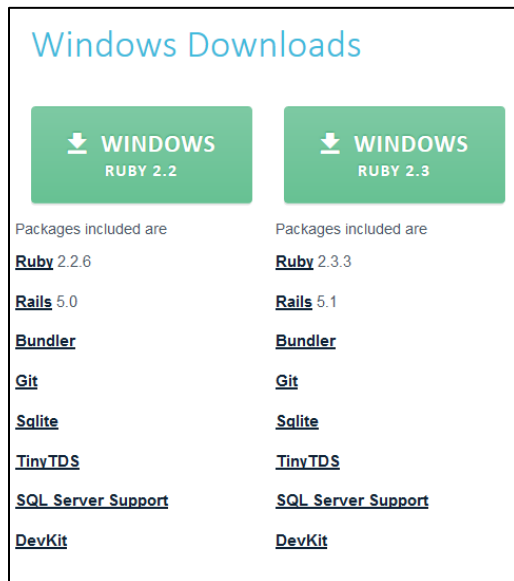
## Method/measurement

### Step 1: Download and install RailsInstaller.

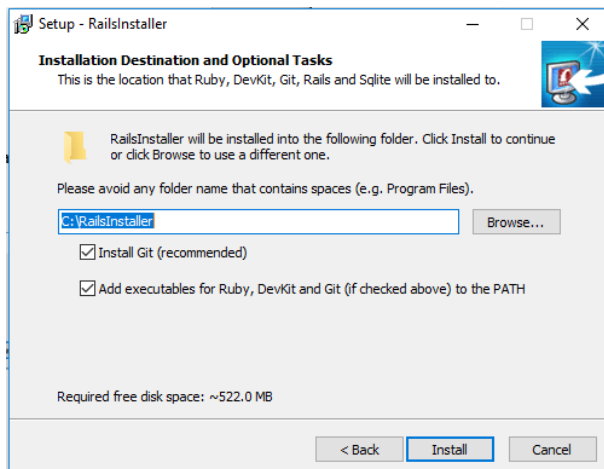
- Navigate to <http://railsinstaller.org/en>



- Click the download button for “Windows Ruby 2.3”, as shown below. Save the file ‘railsinstaller-3.4.0.exe’ somewhere on your computer.



- Run ‘railsinstaller-3.4.0.exe’ to begin installation. Accept the license and proceed to the next screen:



- Here we want to check both checkboxes, the first to install Git (a simple form of command-line revision control), the second to add relevant executable files to your system’s *environment path* variables.

- Once the installer has completed, open the windows command prompt (Windows button + R, then type in 'cmd' and press Enter) and check the versions of Ruby, Rails, SQLite3, and Git to ensure they have installed properly:

```
C:\Users\Work_Time>ruby -v
ruby 2.3.3p222 (2016-11-21 revision 56859) [i386-mingw32]

C:\Users\Work_Time>rails -v
Rails 5.1.6

C:\Users\Work_Time>sqlite3 --version
3.8.7.2 2014-11-18 20:57:56 2ab564bf9655b7c7b97ab85cafc8a48329b27f93

C:\Users\Work_Time>git --version
git version 2.18.0.windows.1
```

Advanced users wanting a higher level of control can install the following components individually:

- Ruby language + Devkit: <https://rubyinstaller.org/downloads/>
- Rails: <https://rubyonrails.org/>
- Bundler: <https://bundler.io/>
- SQLite3: <https://www.sqlite.org/download.html>

## Step 2: Create a new Rails project

Now let us begin. Here we will use the command line interface to create a new Rails project and create a Git repository so we can use revision control to backup and track changes in our application. Here are some basic Command Prompt commands for readers new to the command line interface:

Command	Result
cd filepath	Navigates to a different file path or folder.
cd ..	Navigates one folder up in the file system.
dir	Displays the contents of the current folder.

Command	Result
<code>mkdir foldername</code>	Creates a new folder in the current directory.
<code>rm filename.txt</code>	Deletes a file.
<code>rm -rf foldername /s</code>	Deletes a folder and all of its contents.

- Create a new folder in a convenient place:

```
C:\>mkdir CS390P-Web-Application-Architecture
C:\>cd CS390P-Web-Application-Architecture
C:\CS390P-Web-Application-Architecture>
```

- Now let's create a Rails project! Enter 'rails new [name]' and give your project a name. You should see quite a bit of output to the command line.

```
C:\CS390P-Web-Application-Architecture>rails new Assignment01
create
create  README.md
create  Rakefile
create  config.ru
create  .gitignore
create  Gemfile
```

### Step 3: Setup Git for Revision Control

Since we decided to add Git as part of our RailsInstaller installation, Rails will automatically create a new empty local Git repository within our project folder. Since we'll be continuing on in our learning about web applications, we'll be having one larger repository with multiple Rails projects. Where and how you decide to setup your projects and repository is entirely up to you.

- Once you've settled on a location on your hard drive, enter 'git init' to initialize a new local Git repository (Rascia, T., 2015).

```
C:\CS390P-Web-Application-Architecture>git init
Initialized empty Git repository in C:/CS390P-Web-Application-Architecture/.git/
```

- Once you have a local repository for your project, either login to or create a free account on GitHub so we can have an online backup for our work. Once you have an account, create a new repository:

### Create a new repository

A repository contains all the files for your project, including the revision history.

---

Owner

Repository name

0P-Web-Application-Architecture ✓

Great repository names are short and memorable. Need inspiration? How about [urban-journey](#).

Description (optional)

CS390P Web Application Architecture - Fall 2018, MSU Denver

---

☒ Public  
 Anyone can see this repository. You choose who can commit.

☐ Private  
 You choose who can see and commit to this repository.

---

☐ Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾ ⓘ

---

Create repository

- Now link your local Git repository to your GitHub account:

```
git config --global user.name "Username"
git config --global user.email "myemail@email.com"
git remote set-url origin https://github.com/Username/MyProject.git
```

- Now let's add the entire folder, and every file in it, to revision control.

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

- Before we can upload our project files to GitHub, we need to *commit* to the *master* branch of our repository.

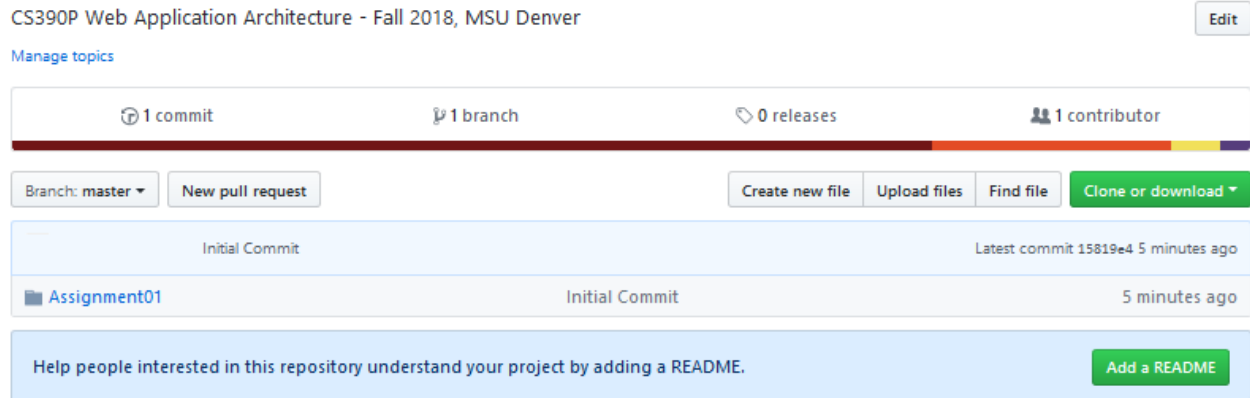
Here we use the parameter '-am' to commit all files, and include a message:

```
C:\CS390P-Web-Application-Architecture>git commit -am "Initial Commit"
[master (root-commit) 15819e4] Initial Commit
74 files changed, 1150 insertions(+)
```

- Now let's *push* our new files to the master branch of our remote repository:

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

- Looking back to GitHub, we can see we've successfully uploaded our project folder and files:



- Similarly, when we need to pull the most recent changes from a project with multiple contributors we can enter:

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

#### Step 4: Rails Scaffolds

Part of the power of Rails is its ability to, with the use of simple command-line parameters, pre-configure a web application for us, and pre-populate various page elements and fields. Any Rails scaffold will setup the basic Model-View-Controller architecture of our web application, as well as create a *database migration* (think revision control, but for databases) to create tables in our database. First, we will setup a simple scaffold for our university application, then we will use a more powerful scaffold to create a one-to-many relationship between courses and sections.

We want our course page to look something like this:

Courses			
Name	Department	Number	Credit Hours
Intro to Astronomy	AST	1200	4



Our section page should look something like this:

Sections			
Semester	Number	Course	Room Number
Fall 2018	1	Intro to Astronomy	200

Now that we have some idea of what we want our website to look like, we can generate a scaffold to quickly create a functional web application.

- First navigate to your Rails project folder:

```
C:\CS390P-Web-Application-Architecture>cd Assignment01
C:\CS390P-Web-Application-Architecture\Assignment01>
```

- Now create a new scaffold. Enter 'rails generate scaffold Course name:string department:string number:integer credit\_hours:integer' to create a new scaffold for Course information (Ruby, S., Copeland, D.B. & Thomas, D., 2017):

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails generate scaffold Course name:string de
partment:string number:integer credit_hours:integer
```

- Now enter 'rails generate scaffold Section course:string semester:string number:integer room\_number:integer' to create a scaffold for Section information:

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails generate scaffold Section course:string
semester:string number:integer room_number:integer
```

- Enter 'rails db:migrate' to create a new database migration for the two database tables we just created:

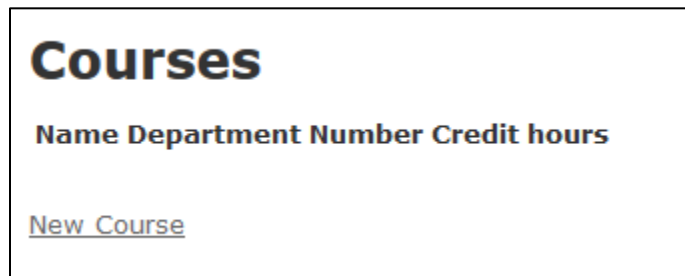
```
C:\CS390P-Web-Application-Architecture\Assignment01>rails db:migrate
== 20180923203627 CreateCourses: migrating =====
-- create_table(:courses)
   -> 0.0012s
== 20180923203627 CreateCourses: migrated (0.0014s) =====

== 20180923203955 CreateSections: migrating =====
-- create_table(:sections)
   -> 0.0007s
== 20180923203955 CreateSections: migrated (0.0009s) =====
```

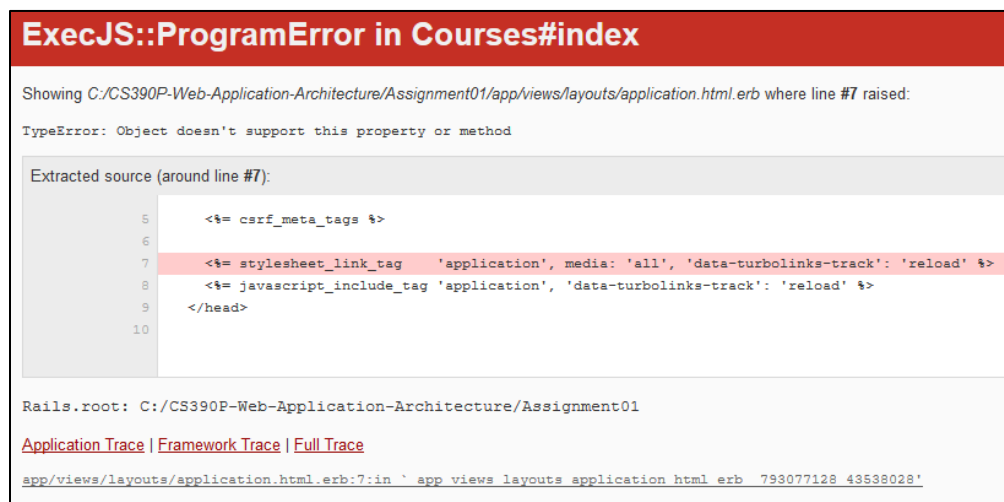
- Now we can finally run our rails server, and see what sort of web page our scaffolds created. Enter 'rails server' or 'rails s' to launch your new Rails web server:

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails server
=> Booting Puma
=> Rails 5.1.6 application starting in development
=> Run `rails server -h` for more startup options
*** SIGUSR2 not implemented, signal based restart unavailable!
*** SIGUSR1 not implemented, signal based restart unavailable!
*** SIGHUP not implemented, signal based logs reopening unavailable!
Puma starting in single mode...
* Version 3.12.0 (ruby 2.3.3-p222), codename: Llamas in Pajamas
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
Use Ctrl-C to stop
```

- Navigate your browser to 'localhost:3000/courses'. If you're lucky, you should see the following:



- In our experience, however, the first time we ran 'rails s', we ended up with the following error message upon navigating to our courses page:



- Fortunately, a simple change seemed to fix the problem completely. Navigate to 'Project\app\assets\javascripts' within your project folder. Find the 'application.js' JavaScript file and edit the following lines (we recommend using Notepad++ for readability, available here:

<https://notepad-plus-plus.org/download/v7.5.8.html>):

Replace this:

```
//= require rails-ujs  
//= require turbolinks  
//= require_tree .
```

With this:

```
//= require rails-ujs  
//= require turbolinks  
// require_tree .
```

- Refreshing 'localhost:3000/courses' displays the page properly now:

# Courses

Name	Department	Number	Credit hours
------	------------	--------	--------------

[New Course](#)

- Click 'New Course' to add a new course:

## New Course

Name

Department

Number

Credit hours

[Back](#)

- Click 'Create Course' to create a new course.
- You should see the following confirmation screen:

Course was successfully created.

**Name:** Necromancy

**Department:** NEC

**Number:** 2600

**Credit hours:** 7

[Edit](#) | [Back](#)

- Now we can see that we've added a new course to our database:

Courses				
Name	Department	Number	Credit hours	
Necromancy	NEC	2600	7	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
<a href="#">New Course</a>				

- We can see that our Courses page seems to be functioning properly. Now let's examine our Sections page.

Navigate your browser to 'localhost:3000/sections'. You should see the following:

Sections			
Course	Semester	Number	Room number
<a href="#">New Section</a>			

- Enter 'New Section' to create a new Section:

## New Section

Course

Semester

Number

Room number

[Back](#)

- Enter 'Create Section' to create the section.

You should see the following confirmation screen:

Section was successfully created.

**Course:** Necromancy

**Semester:** Fall 2018

**Number:** 1

**Room number:** 250

[Edit](#) | [Back](#)

- We can see that we have indeed created a new Section:

Sections				
Course	Semester	Number	Room	number
Necromancy	Fall 2018	1	250	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
<a href="#">New Section</a>				

Congratulations! You've created your first simple web application. Impressive as it is to see Rails work its magic, our Courses and Sections pages no know thing about one another and aren't able to share information in database tables between each other. Let's conclude our study by fixing this small problem.

### Step 5: Rails Relationships

Our goal now is to create a *one-to-many* relationship between courses and sections and display that information as a dropdown menu of possible courses when we create a new section.

- It is a bit easier to start over than worry about first destroying our scaffold then the corresponding database table. Enter `'rmdir Assignment01 /s'` delete our previous project:

```
C:\CS390P-Web-Application-Architecture>rmdir Assignment01 /s
Assignment01, Are you sure (Y/N)? y
```

- Now let's remake our previous project:

```
C:\CS390P-Web-Application-Architecture>rails new Assignment01
```

- Then remake same Course scaffold we made previously (don't forget to navigate to your project directory first):

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails generate scaffold Course name:string department:string number:integer credit_hours:integer
```

- This time, rather than setting Section's 'course' information to a string, let's have our scaffold set it to a *reference* to our Course database table.

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails generate scaffold Section course:references semester:string number:integer room_number:integer
```

- We've made changes to the database, so we need to migrate them:

```
C:\CS390P-Web-Application-Architecture\Assignment01>rails db:migrate
```

- We'll need to remove the '=' from 'application.js' in the 'Project\app\assets\javascripts' folder once again as well.
- This would be a great time to commit:

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

and push:

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

- Now let's create a few more courses to choose from to fill our dropdown menu:

Courses						
Name	Department	Number	Credit hours			
Necromancy	NEC	2600	7	<a href="#">Show</a>	<a href="#">Edit</a>	<a href="#">Destroy</a>
Intro to Archaeology	ARC	1050	4	<a href="#">Show</a>	<a href="#">Edit</a>	<a href="#">Destroy</a>
Advanced Hypnotherapy	PSY	3400	4	<a href="#">Show</a>	<a href="#">Edit</a>	<a href="#">Destroy</a>
America's Rail System	HIS	2300	4	<a href="#">Show</a>	<a href="#">Edit</a>	<a href="#">Destroy</a>
<a href="#">New Course</a>						

- If we navigate to 'localhost:3000/sections', we can see that nothing has changed, and that we can still enter 'unauthorized' Course names as we wish:

### New Section

Course

Semester

Number

Room number

[Back](#)

First, we need to change the *model* of our web application to reflect the relationship between Courses and Sections. Ruby uses the keywords `belongs_to` and `has_many` to denote a one-to-many relationship (Ruby on Rails Guides, 2018). We'll need to edit some model and view files to do so (Beaty, S., 2018).

- Find 'course.rb' in the 'Project\app\models' folder and edit the file as follows:

```
class Course < ApplicationRecord
  has_many :sections
end
```

- Similarly, make sure the Section scaffold created the relationship in the 'section.rb' model file:

```
class Section < ApplicationRecord
  belongs_to :course
end
```

Now we need to change some information in some *view* files for both Courses and Students.

- Navigate to 'app\views\sections\' , then edit the '\_form.html.erb' file as follows.

Change this:

```
<div class="field">
  <%= form.label :course_id %>
  <%= form.text_field :course_id, id: :section_course_id %>
</div>
```

to this:

```
<div class="field">
  <%= form.label :course_id %>
  <%= form.collection_select :course_id, Course.all, :id, :name %>
</div>
```

Here we invoke the 'collection\_select' method to create a simple dropdown menu.

- Now edit the 'index.html.erb' file as follows.

Edit this:

```
<tbody>
  <%= @sections.each do |section| %>
    <tr>
      <td><%= section.course %></td>
```

To look like this:

```
<tbody>
  <%= @sections.each do |section| %>
    <tr>
      <td><%= section.course.name %></td>
```

- Now edit the 'show.html.erb' file as follows.

Edit this:

```
<p>
  <strong>Course:</strong>
  <%= @section.course %>
</p>
```

To look like this:

```
<p>
  <strong>Course:</strong>
  <%= @section.course.name %>
</p>
```

We're almost done! One last view file to edit.

- Navigate to 'app\views\courses\' , then add the following to the 'show.html.erb' file:

```
<ol>
  <%= @course.sections.each do |section| %>
    <li> <%= section.name %>
  <%= end %>
</ol>
```

Now let's run a 'git diff' to review the changes we've made, line by line, in every file in our project:

```
C:\CS390P-Web-Application-Architecture>git diff
```

First, we changed our Course model:

```
--- a/Assignment01/app/models/course.rb
+++ b/Assignment01/app/models/course.rb
@@ -1,4 +1,5 @@
 class Course < ApplicationRecord
+  has_many :sections
 end
```

Then we added an ordered list in our Course view, accessible by Sections view.

```
--- a/Assignment01/app/views/courses/show.html.erb
+++ b/Assignment01/app/views/courses/show.html.erb
@@ -5,6 +5,12 @@
 <%= @course.name %>
 </p>
+<ol>
+  <%= @course.sections.each do |section| %>
+    <li> <%= section.name %>
+  <%= end %>
+</ol>
+
```



We changed the new Section form, adding a dropdown menu for Courses:

```
-- a/Assignment01/app/views/sections/_form.html.erb
+++ b/Assignment01/app/views/sections/_form.html.erb
@@ -13,7 +13,7 @@

<div class="field">
  <%= form.label :course_id %>
-   <%= form.text_field :course_id, Course.all, :id, :name %>
+   <%= form.collection_select :course_id, Course.all, :id, :name %>
</div>
```

And the Sections index page:

```

--- a/Assignment01/app/views/sections/index.html.erb
+++ b/Assignment01/app/views/sections/index.html.erb
@@ -16,7 +16,7 @@
 <tbody>
   <% @sections.each do |section| %>
     <tr>
-      <td><%= section.course %></td>
+      <td><%= section.course.name %></td>
       <td><%= section.semester %></td>
       <td><%= section.number %></td>

```

And show Section pages:

```
--- a/Assignment01/app/views/sections/show.html.erb
+++ b/Assignment01/app/views/sections/show.html.erb
@@ -2,7 +2,7 @@

<p>
  <strong>Course:</strong>
-  <%= @section.course %>
+  <%= @section.course.name %>
</p>
```

If we navigate one last time to `localhost:3000/sections`, we should be able to test our working web application, complete with dropdown menu showing each course in our new Section page.

## Results

Once again, we have our list of courses. Let's make sure we can access this list from our new Section page:

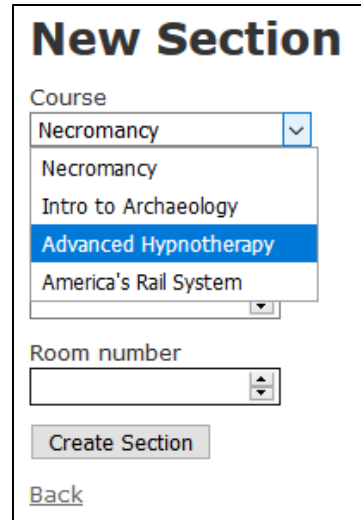
## Courses

Name	Department	Number	Credit	hours	
Necromancy	NEC	2600	7		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
Intro to Archaeology	ARC	1050	4		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
Advanced Hypnotherapy	PSY	3400	4		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
America's Rail System	HIS	2300	4		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[New Course](#)

We can see we have successfully created a running web application using Rails scaffolds to quickly create references between two databases and edit some Ruby files to create the dropdown shown here.

Here's the updated new Section page:



**New Section**

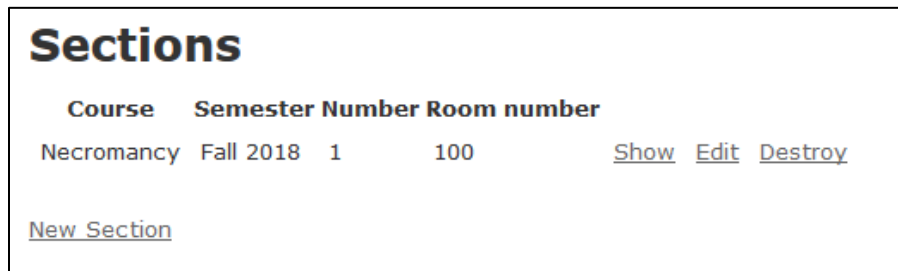
Course  
Necromancy ▼  
Necromancy  
Intro to Archaeology  
Advanced Hypnotherapy  
America's Rail System

Room number

Create Section

[Back](#)

Our Sections page, with a new section added:



Course	Semester	Number	Room number	
Necromancy	Fall 2018	1	100	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[New Section](#)

## Conclusion

We can clearly see how powerful the Ruby on Rails framework is for quickly creating simple web applications. While not without its share of error messages and hiccups, setup using Rails Installer was a breeze. We can see how Rails sets up quite a lot of different files and all of the application architecture for us using a simple `scaffold` command. We learned enough Ruby syntax to call a simple `collection select` function to create a dropdown menu and share data between two SQLite3 databases. While this simple example might not be beautiful to look at just yet, we have a much greater understanding of the architecture and components of web applications, and how we can use a framework like Ruby on Rails to quickly create a website to call our very own.

## Bibliography

Ruby, S., Copeland, D.B. & Thomas, D. (2017). *Agile Web Development with Rails 5.1*. The Pragmatic Programmers, LLC. Raleigh, NC.

Rascia, T. (2015). *Getting Started with Git* [Website]. Retrieved from <https://www.taniascascia.com/getting-started-with-git/>.

Beaty, S. (2018). *ActiveRecord* [PowerPoint Slides]. Retrieved from <https://gouda.msudenver.edu/moodle/mod/assign/view.php?id=7736>.

Ruby on Rails Guides (2018). *Active Record Associations* [Website]. Retrieved from [https://guides.rubyonrails.org/association\\_basics.html](https://guides.rubyonrails.org/association_basics.html)

## Instructor Feedback

What was too difficult, too easy?

Finding exactly the right files to install for the initial setup, I seemed to have a hard time installing individual components before I found RailsInstaller. After that initial snag, I had some trouble with error messages before editing the one line in application.js. All things considered, this assignment was of a very appropriate difficulty level.

What would have made the learning experience better?

As it stands, it's a great, relatively simple introduction to web development with Ruby on Rails.

What did you learn?

Quite a lot about web servers, building on my previous experience in penetration testing and security. I was a bit shocked when I realized how easy Rails makes it for you, and pleasantly surprised.

How did you learn it?

Using every resource I had, from instructor slides, the 'official' Ruby on Rails guides, web tutorials and YouTube videos to find that RailsInstaller was the way to go for this first assignment. Once we had the server running, it was a lot of typical programmer trial and error, fiddling with different view files and parameters until I found the element that needed fixing.