Kevin Ngovanduc
10/2/2018
CS390P: Assignment #2

Abstract:

Continuing from the first assignment, Assignment 2's requirements were to add a many-to-many relationship with Students and Sections, add validation, implement a search method, and write unit tests for the app. This paper will discuss the process that was taken to update this web application using Ruby on Rails with RubyMine.

Motivation:

Similarly to Assignment 1, my goal is to provide a step-by-step introduction on how to update these particular features. Almost all web applications have tables with relationships, searches, etc, so it is very important to understand how they work. My hopes is for the reader to avoid any issues I encountered by providing a straightforward guide.

Introduction:

My strategy was to work from the first assignment, utilize version control in case of emergency, and follow Dr. Beaty's slides as they provide information that is crucial to completing the assignment. Reading from the Powerpoint, I wrote down which files to edit and what commands to execute so I was prepared to tackle the requirements that were asked of me.

Method/Measurement:

After I made a copy of the functioning Assignment 1, I switched local branches and started working on a copy in case if I ran into any issues. Continuing from where I left off in the last assignment, I executed these commands in the command prompt:

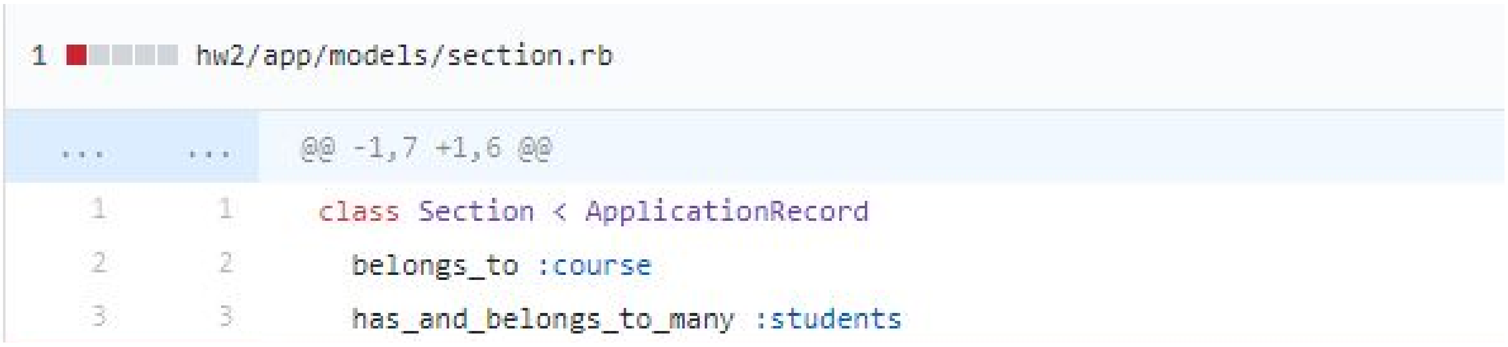
To generate a scaffold for students:
Rails generate scaffold Student name:string

To create a table that links students and sections:
Rails generate model SectionsStudents section:references student:references --force-plural

Finally, to migrate the database:
Rails db:migrate

Then, I edited the corresponding files to add a many-to-many relationship with students and sections by editing the model files, and adding parameters to the controller files.

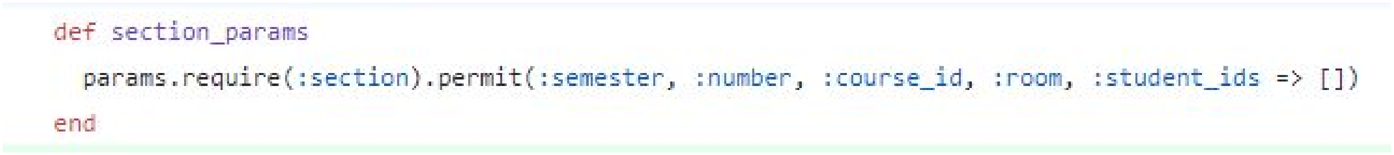
Sections.rb model:



```
1 ■■■■■ hw2/app/models/section.rb

@@ -1,7 +1,6 @@
class Section < ApplicationRecord
  belongs_to :course
  has_and_belongs_to_many :students
```

(models/Students.rb similarly has has_and_belongs_to_many :sections)

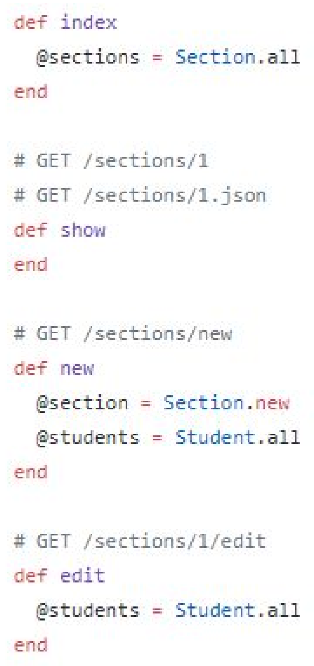
Sections.rb controller parameters:



```
def section_params
  params.require(:section).permit(:semester, :number, :course_id, :room, :student_ids => [])
end
```

(controllers/Students.rb also has the parameters added … :section_ids => [])

Sections.rb controller:

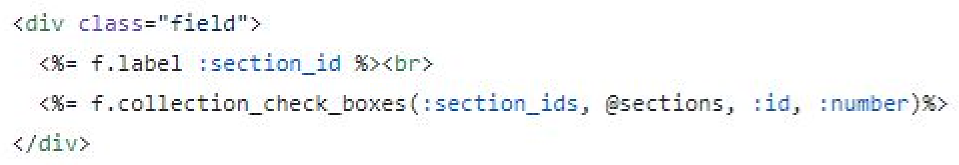


```
def index
  @sections = Section.all
end

# GET /sections/1
# GET /sections/1.json
def show
end

# GET /sections/new
def new
  @section = Section.new
  @students = Student.all
end

# GET /sections/1/edit
def edit
  @students = Student.all
end
```

(Students.rb controller was also edited similarly)

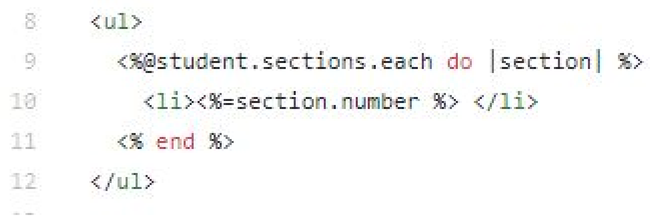
I proceeded to edit the view files for Section and Student so the Create, Edit, and Show methods work in junction with the new features.

view/Students _form.html.erb for checkboxes:



```
<div class="field">
  <%= f.label :section_id %><br>
  <%= f.collection_check_boxes(:section_ids, @sections, :id, :number)%>
</div>
```

view/Students show.html.erb:



```
<ul>
  <%@student.sections.each do |section| %>
    <li><%=section.number %> </li>
  <% end %>
</ul>
```

*(/view/**Sections was also edited identically**)*

For adding validations to all my classes, here is what I added for Courses, Sections, & Students:

Courses

- Validates presence of numbers, credits, department, name (these can't be null)
- Validates numericality of credits

Sections

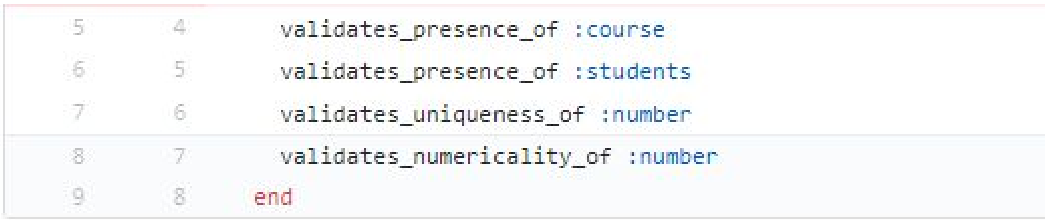
- Validates presence of course
- Validates numericality and uniqueness of number
- Implementing similarly to a real-life application, the semester and room can be left empty

Students

- Validates presence and uniqueness of name

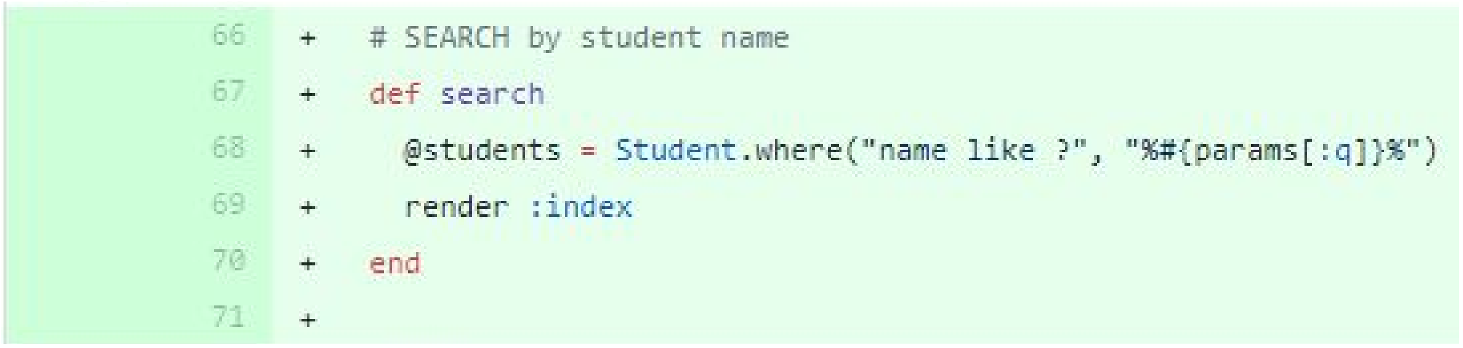
These validations were stored in each model, where they all had different validations depending on their variables.

Example code for validating Course.rb model:


```
  validates_presence_of :course
  validates_presence_of :students
  validates_uniqueness_of :number
  validates_numericality_of :number
end
```

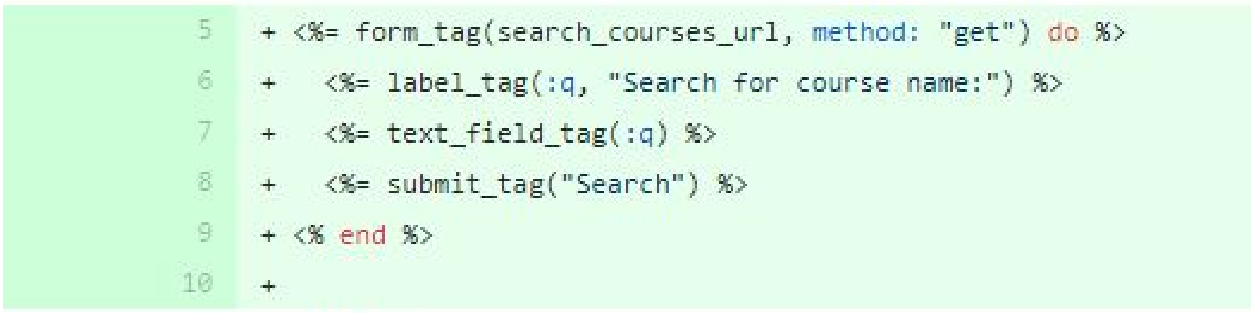
Another requirement of the assignment was to add a search function to Course, Sections, and Students. I added a search method in each respective controller, where they used an SQL query to check for the item being searched, with :q being the parameter passed in.

Student.rb controller:

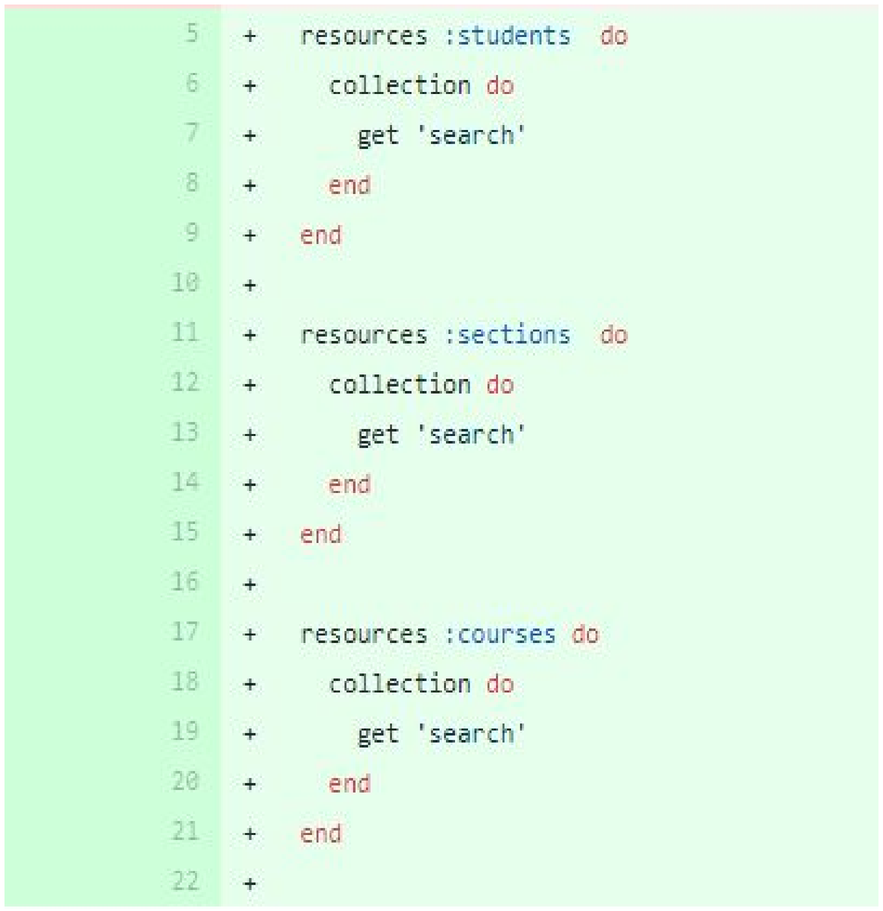

```
+ # SEARCH by student name
+ def search
+   @students = Student.where("name like ?", "%#{params[:q]}%")
+   render :index
+ end
+
```

Similarly to this example code, my Course and Section controllers have an identical search method, but I used other parameters such as semester for Section and name for Student. After this call is made, the index is rendered for the matching parameters if any.

For the index.html.erb files, I put in the following code for Courses, while Sections and Students were identical but the form tag urls were changed:


```
+ <%= form_tag(search_courses_url, method: "get") do %>
+   <%= label_tag(:q, "Search for course name:") %>
+   <%= text_field_tag(:q) %>
+   <%= submit_tag("Search") %>
+ <% end %>
+
```

Last but not least, I edited my routes.rb file so that the search function is routed properly:

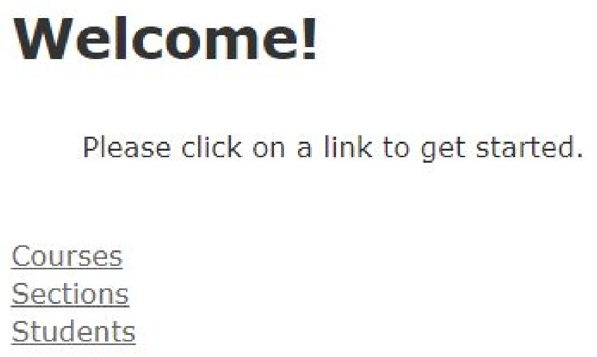


```
+   resources :students  do
+     collection do
+       get 'search'
+     end
+   end
+
+   resources :sections  do
+     collection do
+       get 'search'
+     end
+   end
+
+   resources :courses do
+     collection do
+       get 'search'
+     end
+   end
+
```

Results:

Screenshots of the updated features:

<u>Homepage:</u>

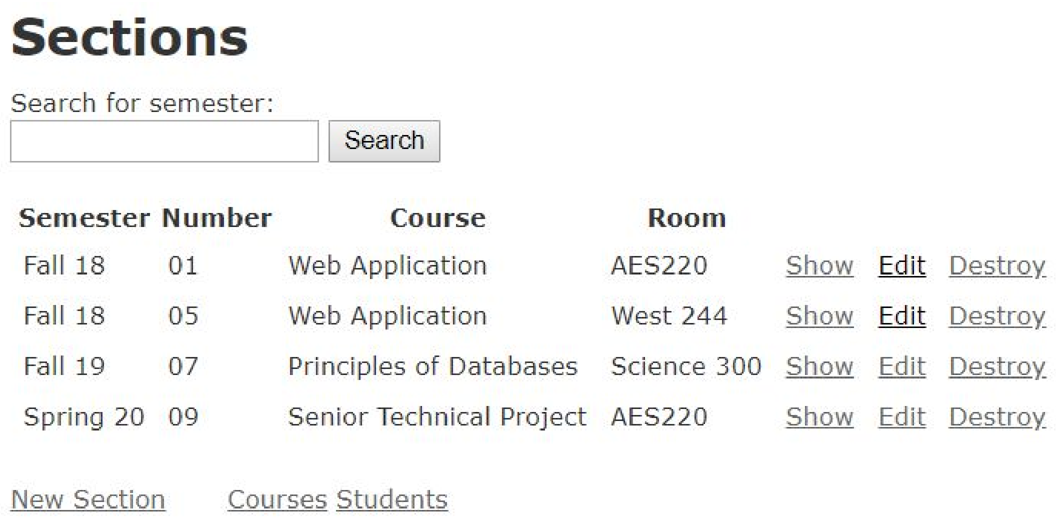


Welcome!

Please click on a link to get started.

Courses
Sections
Students

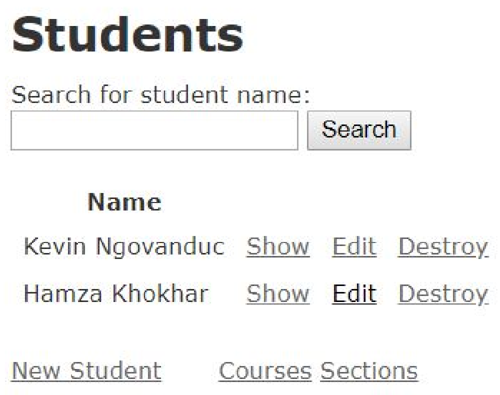
Sections and Students page:


Sections

Search for semester:

Semester	Number	Course	Room			
Fall 18	01	Web Application	AES220	Show	Edit	Destroy
Fall 18	05	Web Application	West 244	Show	Edit	Destroy
Fall 19	07	Principles of Databases	Science 300	Show	Edit	Destroy
Spring 20	09	Senior Technical Project	AES220	Show	Edit	Destroy

New Section   Courses Students

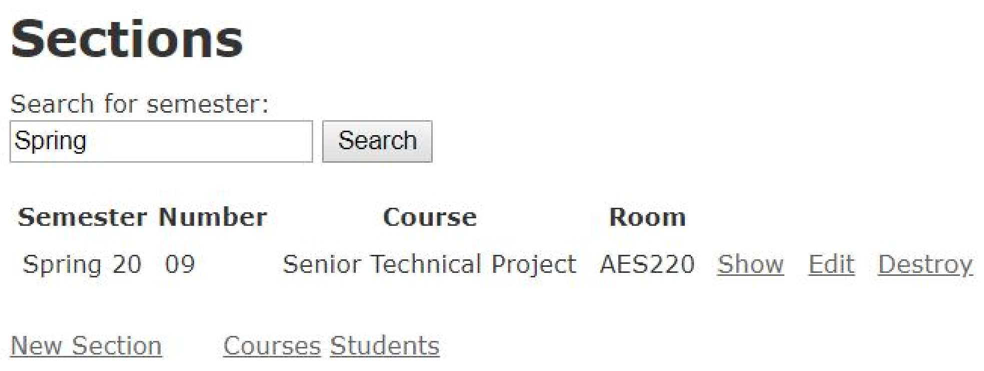

Students

Search for student name:

Name			
Kevin Ngovanduc	Show	Edit	Destroy
Hamza Khokhar	Show	Edit	Destroy

New Student   Courses Sections

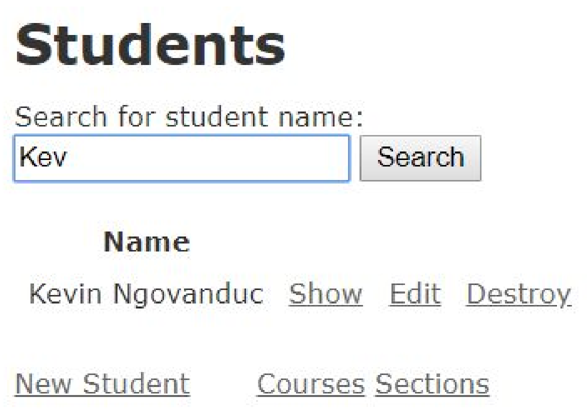
Searching results 'Spring' for Sections and 'Kev' for Students:


Sections

Search for semester:

Semester	Number	Course	Room			
Spring 20	09	Senior Technical Project	AES220	Show	Edit	Destroy

New Section   Courses Students



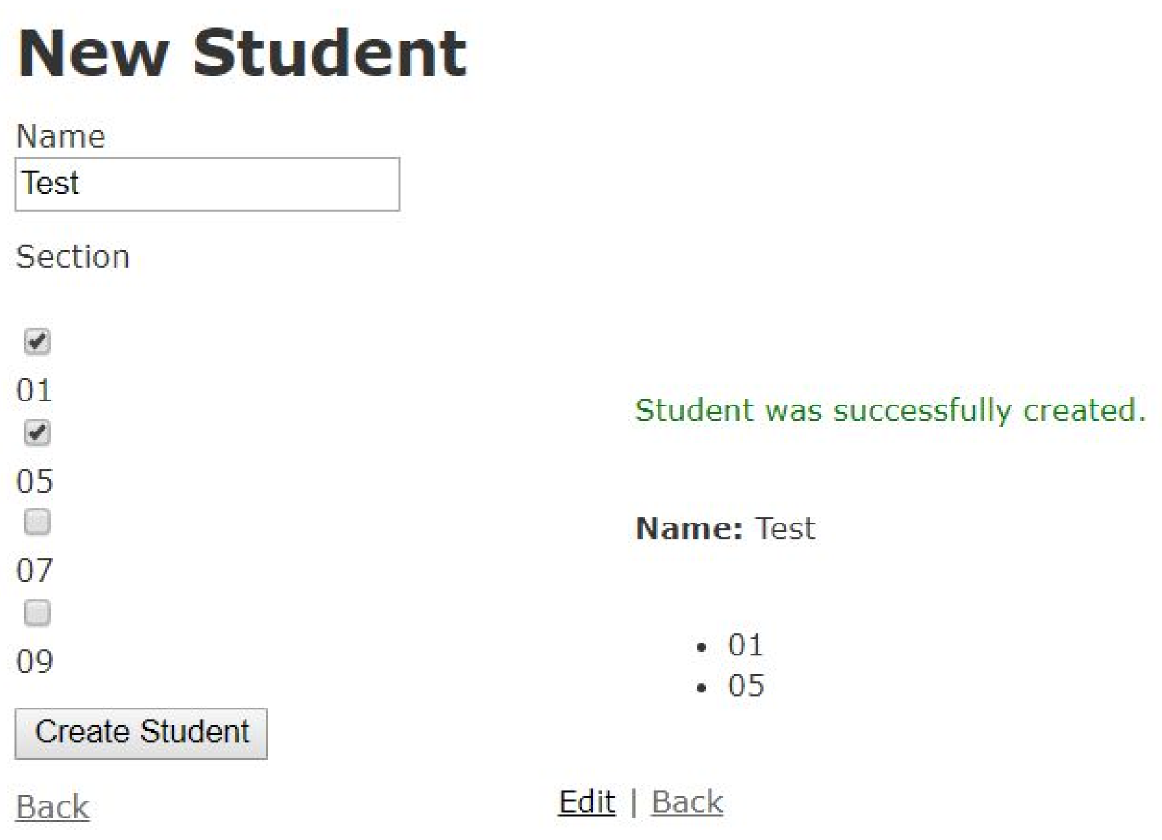
Students

Search for student name:
Kev
Search

Name			
Kevin Ngovanduc	Show	Edit	Destroy

New Student   Courses Sections

Showing a many-to-many relationship with Sections and Students:



New Student

Name
Test

Section

☑ 01
☑ 05
☐ 07
☐ 09

Create Student

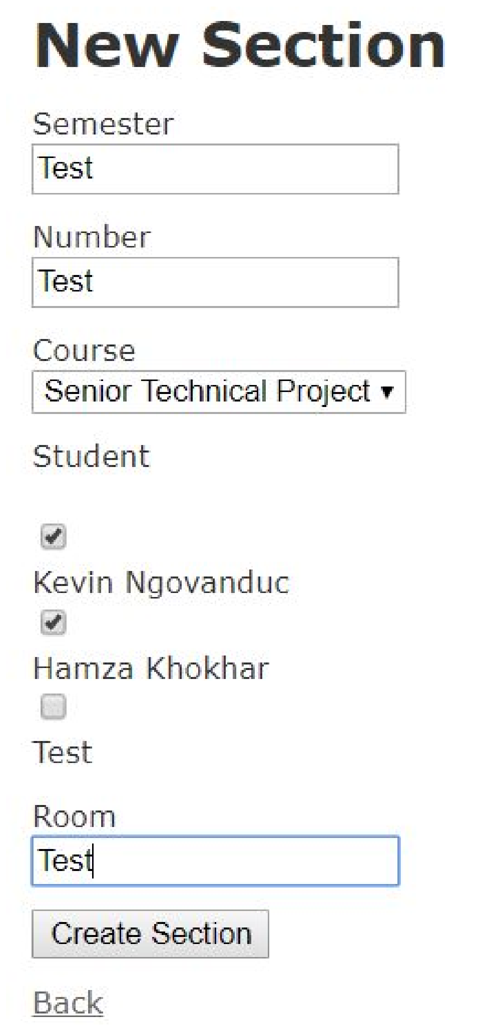
Back

Student was successfully created.

Name: Test

- 01
- 05

Edit | Back



New Section

Semester

Test

Number

Test

Course

Senior Technical Project ▾

Student

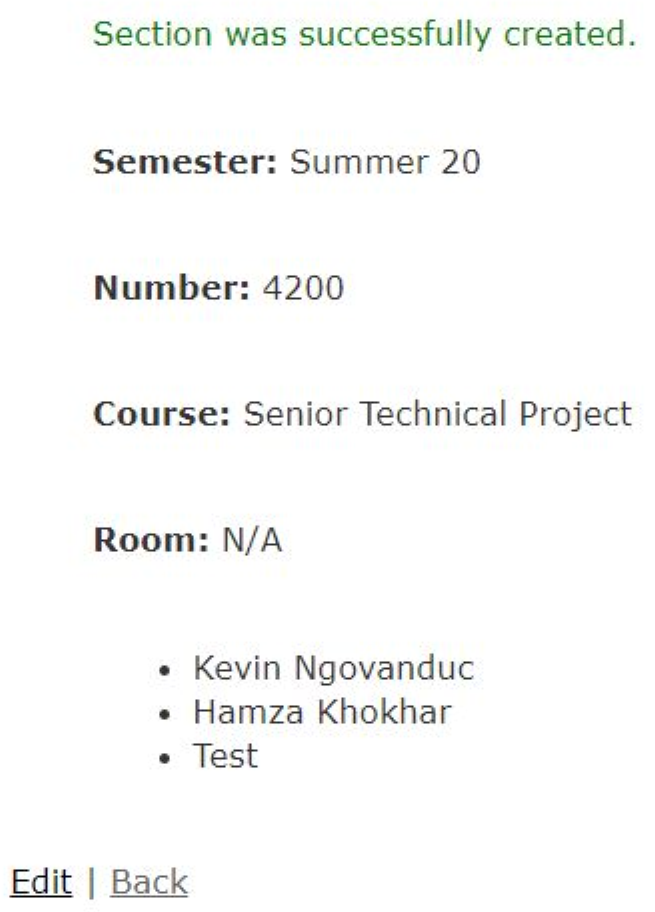
✓ Kevin Ngovanduc

✓ Hamza Khokhar

☐ Test

Room

Test



Section was successfully created.

Semester: Summer 20

Number: 4200

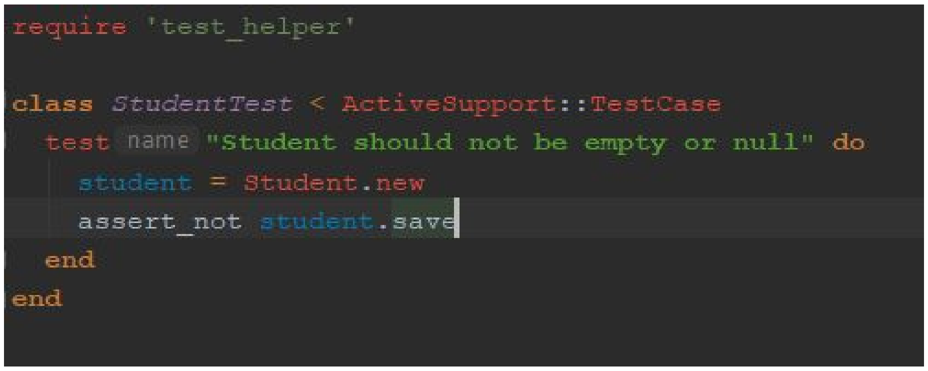
Course: Senior Technical Project

Room: N/A

- Kevin Ngovanduc
- Hamza Khokhar
- Test

Test cases:

I unfortunately did not have time to go into testing into detail, but I looked at basic test cases and wrote a simple test whether the Student name was null or empty:



```ruby
require 'test_helper'

class StudentTest < ActiveSupport::TestCase
  test name "Student should not be empty or null" do
    student = Student.new
    assert_not student.save
  end
end
```

When I was implementing the views for Student and Section, I was able to refer to the student name foreign key quite effortlessly, but I had a difficult time finding the appropriate foreign key to use in the **_form.html.erb** file for the Students view. I ended up using a checkbox selection for which section to pick, while using the section number as a foreign key.

Another issue I ran into was when I was generating the model for StudentsSections; since Ruby is awfully particular about naming convention, I spent hours trying to figure out why the table created wasn't being linked together. I later found out that I had to use the command --force-plural so that the table is referred to properly. I believe that all the requirements in this assignment were met and working, I also went ahead and changed the application by adding a homepage and links from all pages to improve user experience.

The working application of Assignment 2 can be found on: ***github/kngovand/webApp/tree/hw2***

Bibliography:

Formal:

1. Dr. Beaty's PowerPoint slides, 03ActiveRecord
2. Guides.rubyonrails.org. (2018). *Active Record Associations — Ruby on Rails Guides*. [online] Available at: https://guides.rubyonrails.org/association_basics.html [Accessed 3 Oct. 2018].
3. Guides.rubyonrails.org. (2018). *Testing Rails Applications — Ruby on Rails Guides*. [online] Available at: https://guides.rubyonrails.org/testing.html [Accessed 6 Oct. 2018].

Informal:

1. Stack Overflow. (2018). *Ruby on Rails plural (controller) and singular (model) convention - explanation*. [online] Available at: https://stackoverflow.com/questions/10078139/ruby-on-rails-plural-controller-and-singular-model-convention-explanation [Accessed 3 Oct. 2018].
2. Cohen, K. (2018). *Creating a Simple Search in Rails 4*. [online] Koren Leslie Cohen. Available at: http://www.korenlc.com/creating-a-simple-search-in-rails-4/ [Accessed 5 Oct. 2018].

Reflection:

To be completely honest, I didn't know where to start on the first assignment and all the packages, files, etc, started to overwhelmed me. Now that I'm starting to get a hang of Ruby on Rails, it seems fun and exciting to develop a web application! I'm now starting to understand what Dr. Beaty meant by how RoR will hold your hand if you let it, but you will make it difficult if you do not follow the conventions and rules. I can say I am looking forward to adding new features to my application.