Table of Contents

Graduation Date Filtering

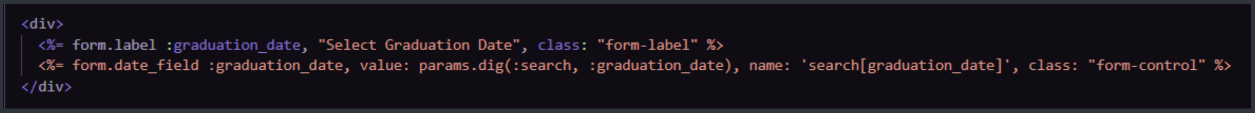
User Goal: Being able to search for all students who graduate on, before, or after a specific date.
Part 1: Getting and searching by a specific date.
Part 2: Sorting by before or after a specific date.

Sorting By Graduation Date

In The View:

The following div and form elements should be added to app\views\students_search_form_html.erb under the Search By Major field. They implement a field where a date can be typed as mm / dd / yyyy, or manually selected using a built-in calendar option. The input value is then passed as :graduation_date.

```
    <div>

      <%= form.label :graduation_date, "Select Graduation Date", class: "form-label" %>

      <%= form.date_field :graduation_date, value: params.dig(:search, :graduation_date), name: 
'search[graduation_date]', class: "form-control" %>

    </div>
```



```
<div>
  <%= form.label :graduation_date, "Select Graduation Date", class: "form-label" %>
  <%= form.date_field :graduation_date, value: params.dig(:search, :graduation_date), name: 'search[graduation_date]', class: "form-control" %>
</div>
```

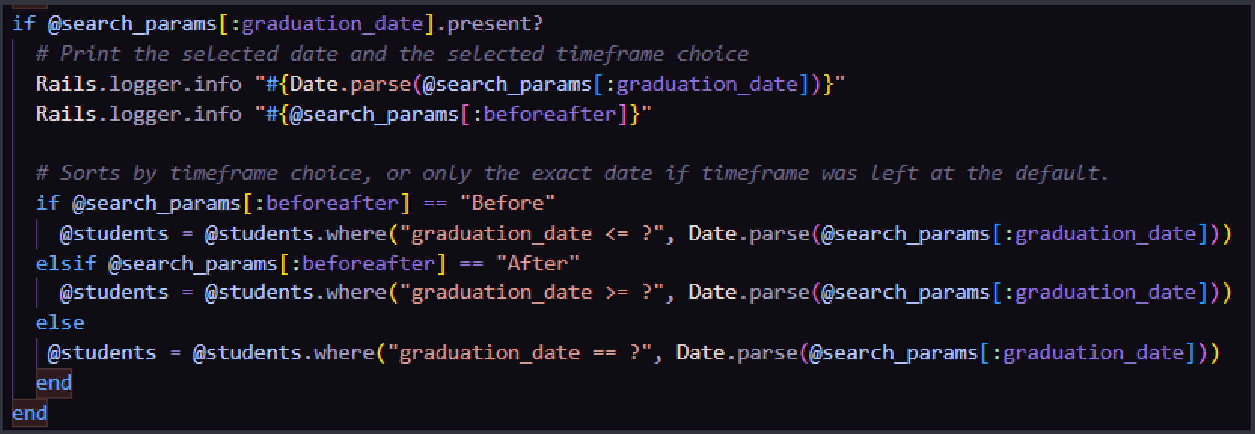
In the Controller:

Implement the following if statement to check if the search query contains a graduation date or if the field was left blank.

```
if @search_params[:graduation_date].present?
end
```

Inside the if statement but before the end statement, insert the if/elsif/else sequence below. The statements will read the value of :beforeafter from the form (explained in the next section) to determine what comparison to make. Inside of each statement, @students is searched for students whose graduation_date attribute compares to the search form's :graduation_date value. The use of the Date.parse method and "?" instead of direct comparison are used as additional insurance of data integrity.

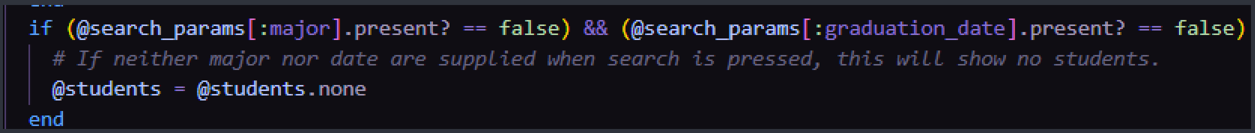
```
    if @search_params[:beforeafter] == "Before"

      @students = @students.where("graduation_date <= ?", 
Date.parse(@search_params[:graduation_date]))

    elsif @search_params[:beforeafter] == "After"

      @students = @students.where("graduation_date >= ?", 
Date.parse(@search_params[:graduation_date]))

    else

     @students = @students.where("graduation_date == ?", 
Date.parse(@search_params[:graduation_date]))

    end
```



```
if @search_params[:graduation_date].present?
  # Print the selected date and the selected timeframe choice
  Rails.logger.info "#{Date.parse(@search_params[:graduation_date])}"
  Rails.logger.info "#{@search_params[:beforeafter]}"

  # Sorts by timeframe choice, or only the exact date if timeframe was left at the default.
  if @search_params[:beforeafter] == "Before"
    @students = @students.where("graduation_date <= ?", Date.parse(@search_params[:graduation_date]))
  elsif @search_params[:beforeafter] == "After"
    @students = @students.where("graduation_date >= ?", Date.parse(@search_params[:graduation_date]))
  else
   @students = @students.where("graduation_date == ?", Date.parse(@search_params[:graduation_date]))
  end
end
```

However, there is not yet anything to specifically handle a search where neither a major nor a graduation date was input (even if the timeframe is specified). Insert the following statement, which will show zero students as there are zero valid criteria.

```
  if (@search_params[:major].present? == false) && (@search_params[:graduation_date].present? == false)

    @students = @students.none

  end
```



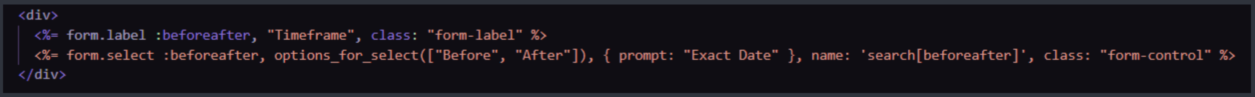
```
  if (@search_params[:major].present? == false) && (@search_params[:graduation_date].present? == false)
    # If neither major nor date are supplied when search is pressed, this will show no students.
    @students = @students.none
  end
```

Sorting By Timeframe

In the View:

The following div and form elements should be added to app\views\students_search_form_html.erb right above or below the graduation_date form elements. They implement a dropdown where the default option is “Exact Date”, but other selectable options are “Before” and “After”. The chosen value is then passed as :beforeafter.

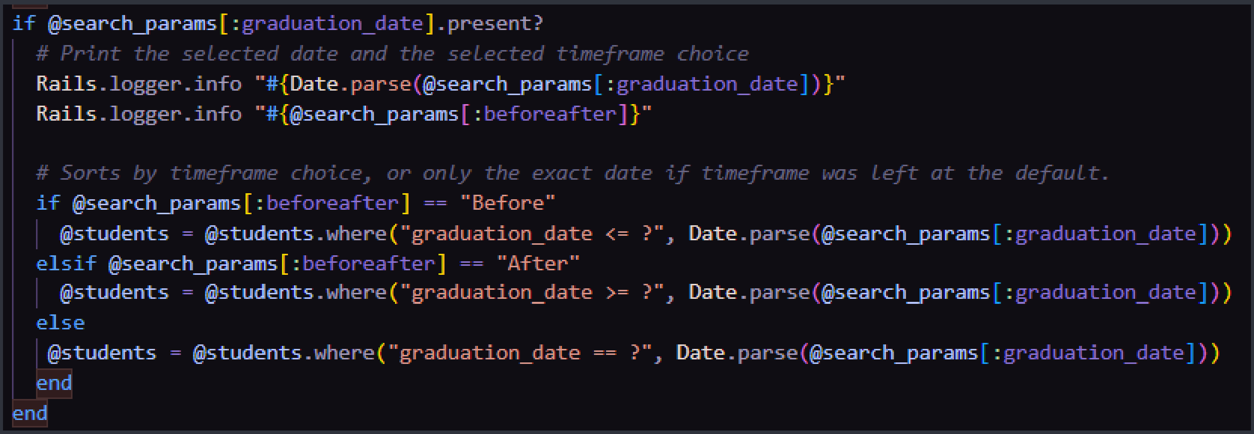
```
    <div>

      <%= form.label :beforeafter, "Timeframe", class: "form-label" %>

      <%= form.select :beforeafter, options_for_select(["Before", "After"]), { prompt: "Exact Date" }, name: 'search[beforeafter]', class: "form-control" %>

    </div>
```



```
<div>
  <%= form.label :beforeafter, "Timeframe", class: "form-label" %>
  <%= form.select :beforeafter, options_for_select(["Before", "After"]), { prompt: "Exact Date" }, name: 'search[beforeafter]', class: "form-control" %>
</div>
```

In the Controller:

If instructions from the prior section were followed, the if/elsif/else statements evaluating :beforeafter are already implemented and nothing else needs to be added inside the outer if statement. For clarity, the sequence checks if the user selected “Before”, then “After”, or does an “Exact” comparison if neither was selected (because Exact is the default option). For a transcribed excerpt, see **Sorting By Graduation Date** under **In the Controller**.



```ruby
if @search_params[:graduation_date].present?
  # Print the selected date and the selected timeframe choice
  Rails.logger.info "#{Date.parse(@search_params[:graduation_date])}"
  Rails.logger.info "#{@search_params[:beforeafter]}"

  # Sorts by timeframe choice, or only the exact date if timeframe was left at the default.
  if @search_params[:beforeafter] == "Before"
    @students = @students.where("graduation_date <= ?", Date.parse(@search_params[:graduation_date]))
  elsif @search_params[:beforeafter] == "After"
    @students = @students.where("graduation_date >= ?", Date.parse(@search_params[:graduation_date]))
  else
   @students = @students.where("graduation_date == ?", Date.parse(@search_params[:graduation_date]))
  end
end
```

Search Result Documentation

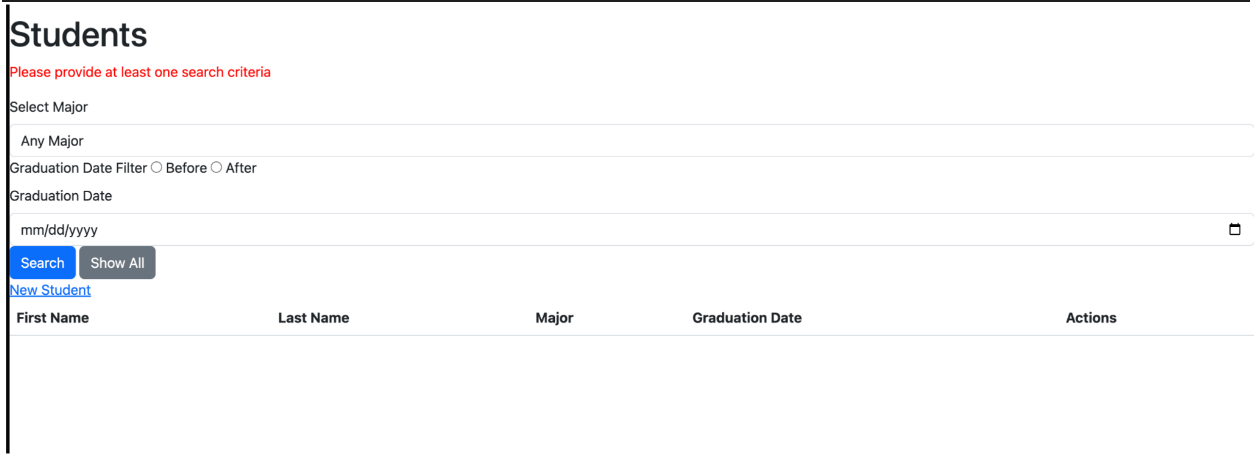
Overall(ie what should it do and look at in the end)
User Story(provided by Deb): As an employer, I want to be able to search for students to filter based on major and graduation date.

In the end: In the end you should be able to search by graduation date or major or both and only output first name, last name, major, graduation date, and the actions for that student(show, edit, and delete).

Steps

1.No student will be displayed until a search is completed
Code(what files?): under the students_controller.rb

```
else
   #If no search feilds are filled, show an alert or show no students
   flash.now[:alert] = "Please provide at least one search criteria"
   @students = Student.none
 end
```



Students

Please provide at least one search criteria

First Name	Last Name	Major	Graduation Date	Actions

How it works: There will be a message, in red, at the top of the page instructing the user to fill at least one search criteria out, i.e. the graduation date field or the major field. Once either field or both fields are filled out and searched the message goes away.

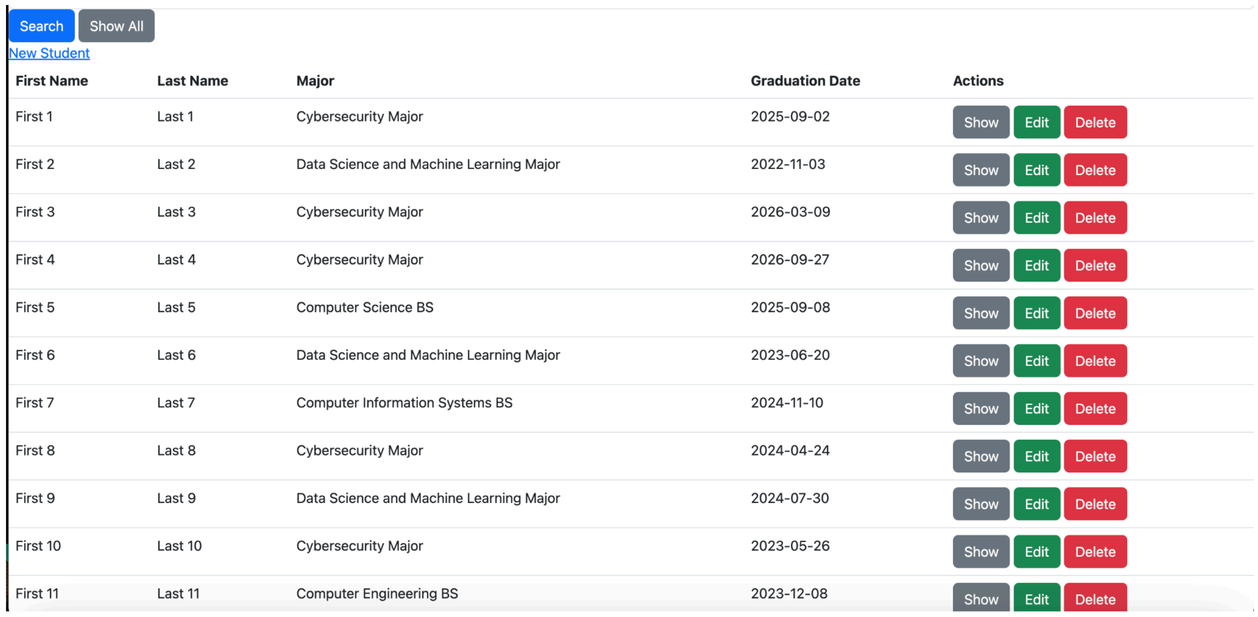
You will also need to add the button to make the search
Code(what files?): _Search_form.erb

```
<div>
   <!--submit button and show all button-->
     <%= form.submit "Search", class: "btn btn-primary" %>
</div>
```

2.An option to show all students if chosen

Code(what files?): students_controller.rb

```
if params[:show_all]
     @students = Student.all
```



Search Show All

New Student

First Name	Last Name	Major	Graduation Date	Actions
First 1	Last 1	Cybersecurity Major	2025-09-02	Show Edit Delete
First 2	Last 2	Data Science and Machine Learning Major	2022-11-03	Show Edit Delete
First 3	Last 3	Cybersecurity Major	2026-03-09	Show Edit Delete
First 4	Last 4	Cybersecurity Major	2026-09-27	Show Edit Delete
First 5	Last 5	Computer Science BS	2025-09-08	Show Edit Delete
First 6	Last 6	Data Science and Machine Learning Major	2023-06-20	Show Edit Delete
First 7	Last 7	Computer Information Systems BS	2024-11-10	Show Edit Delete
First 8	Last 8	Cybersecurity Major	2024-04-24	Show Edit Delete
First 9	Last 9	Data Science and Machine Learning Major	2024-07-30	Show Edit Delete
First 10	Last 10	Cybersecurity Major	2023-05-26	Show Edit Delete
First 11	Last 11	Computer Engineering BS	2023-12-08	Show Edit Delete

How it works: This code will show all students in your database.

You will also need to add the button to make the search

Code(what files?): _Search_form.erb

```
<div>
   <!--submit button and show all button-->
     <%= form.submit "Search", class: "btn btn-primary" %>
     <%= link_to "Show All", students_path(show_all: true), class: "btn btn-secondary"
%>
   </div>
```

3.When searching for a student an employer should be able to pick a filter one or both of the following to display students

-Pick a major

-Pick before or after graduation date

Code(what files?):

```
elsif @search_params[:major].present? || @search_params[:graduation_date].present?
     #Filter by major is provided
     if @search_params[:major].present? && @search_params[:major] != "all"
       @students = @students.where(major: @search_params[:major])
     end
     #Filter by graduation date if the date and before/after is provided
     if @search_params[:grad_date_filter].present? &&
@search_params[:graduation_date].present?
     begin
       selected_date = Date.parse(@search_params[:graduation_date])
       Rails.logger.info "Parsed Date: #{selected_date}"

       if @search_params[:grad_date_filter] == 'before'
         @students = @students.where("graduation_date < ?", selected_date)
       elsif @search_params[:grad_date_filter] == 'after'
         @students = @students.where("graduation_date > ?", selected_date)
       end

     rescue ArgumentError
       #Return no results if date is invalid
       flash.now[:alert] = "Invalid date format"
       @students = Student.none
     end
   end
```

How it works: This code checks to see which field you entered information in and if you didnt fill the form/field out correctly there will be an alert that will show up telling you to go back and fix the wrong form/field. It then sees which field is imputed, if its only the major it will only show the students that are in that major that you selected, if its only the graduation date then it will show the students that are either before or after that date, and if its both fields it takes that major and graduation date and only shows the following students that meet that criteria.

Code(what files?): _Search_form.erb

```
<div>
   <!--Choose the Major-->
     <%= form.label :major, "Select Major", class: "form-label" %>
     <%= form.select :major, options_for_select(Student::VALID_MAJORS), { prompt: "Any
Major" }, name: 'search[major]', class: "form-control" %>
   </div>

   <div>
   <!--Graduation date filter(Before or After)-->
       <%= form.label :grad_date_filter, "Graduation Date Filter", class: "form-label"
%>
       <%= form.radio_button :grad_date_filter, 'before', checked: params.dig(:search,
:grad_date_filter) == 'before',  name: 'search[grad_date_filter]' %> Before
```

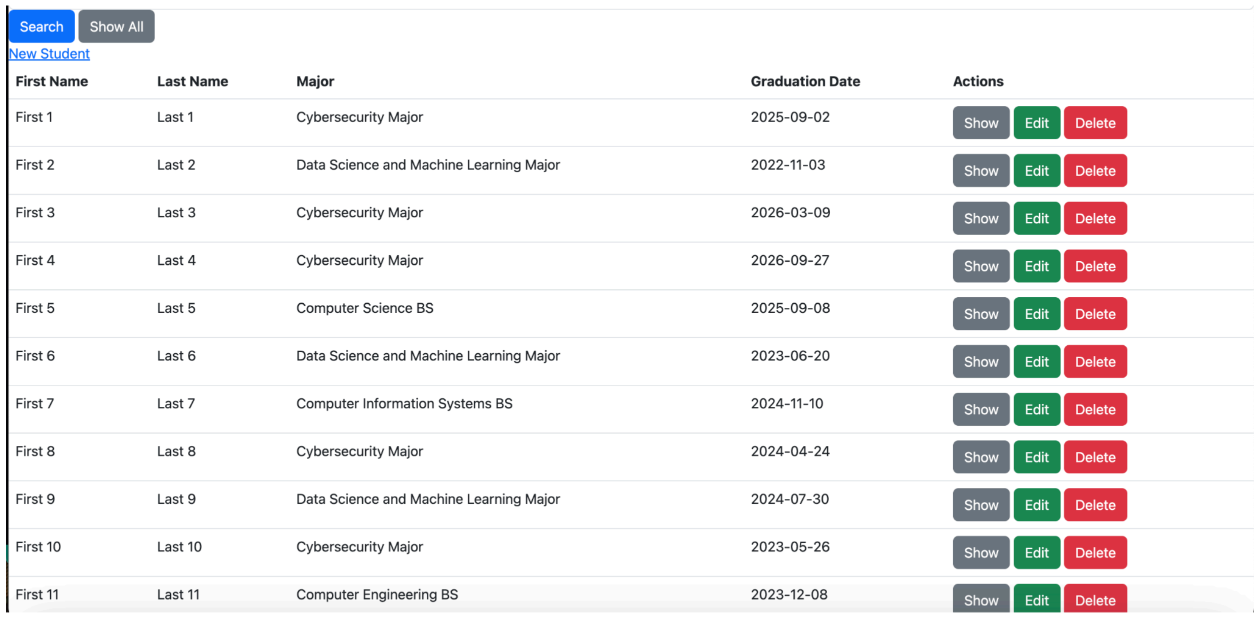
```
       <%= form.radio_button :grad_date_filter, 'after', checked: params.dig(:search,
:grad_date_filter) == 'after',  name: 'search[grad_date_filter]' %> After
   </div>

   <div>
   <!--Date Picker-->
       <%= form.label :graduation_date, "Graduation Date", class: "form-label" %>
       <%= form.date_field :graduation_date, value: params.dig(:search,
:graduation_date), name: 'search[graduation_date]', class: "form-control" %>
   </div>
```

4.Search results should only display First and Last name, major, graduation date, and actions(Show, Edit, Delete)
How it works: How it works: This code creates a table only showing the first name, last name, graduation date, and the actions(show, edit, and delete).

Code(what files?): index.html.erb

```
<table class="table">
 <thead>
   <tr>
     <th>First Name</th>
     <th>Last Name</th>
     <th>Major</th>
     <th>Graduation Date</th>
     <th>Actions</th>
   </tr>
 </thead>
 <tbody>
   <% @students.each do |student| %>
     <tr>
       <td><%= student.first_name %></td>
       <td><%= student.last_name %></td>
       <td><%= student.major %></td>
       <td><%= student.graduation_date %></td>
       <td>
         <%= link_to 'Show', student_path(student), class: "btn btn-secondary
hover-grow" %>
         <%= link_to 'Edit', edit_student_path(student), class: "btn btn-success
hover-grow" %>
         <%= link_to 'Delete', student_path(student), method: :delete, data: {
confirm: "Are you sure?" }, class: "btn btn-danger hover-grow" %>
```

```
       </td>
     </tr>
   <% end %>
 </tbody>
</table>
```



First Name	Last Name	Major	Graduation Date	Actions
First 1	Last 1	Cybersecurity Major	2025-09-02	Show Edit Delete
First 2	Last 2	Data Science and Machine Learning Major	2022-11-03	Show Edit Delete
First 3	Last 3	Cybersecurity Major	2026-03-09	Show Edit Delete
First 4	Last 4	Cybersecurity Major	2026-09-27	Show Edit Delete
First 5	Last 5	Computer Science BS	2025-09-08	Show Edit Delete
First 6	Last 6	Data Science and Machine Learning Major	2023-06-20	Show Edit Delete
First 7	Last 7	Computer Information Systems BS	2024-11-10	Show Edit Delete
First 8	Last 8	Cybersecurity Major	2024-04-24	Show Edit Delete
First 9	Last 9	Data Science and Machine Learning Major	2024-07-30	Show Edit Delete
First 10	Last 10	Cybersecurity Major	2023-05-26	Show Edit Delete
First 11	Last 11	Computer Engineering BS	2023-12-08	Show Edit Delete

Final result code

student_controller.rb

```
class StudentsController < ApplicationController
 before_action :set_student, only: %i[ show edit update destroy ]

 # GET /students or /students.json
 def index
   @search_params = params[:search] || {}
   @students = Student.all

   #"Show All" button shows all students
   if params[:show_all]
     @students = Student.all

   #Searches only if major or graduation date is filled out correctly
   elsif @search_params[:major].present? || @search_params[:graduation_date].present?
     #Filter by major is provided
     if @search_params[:major].present? && @search_params[:major] != "all"
```

```
      @students = @students.where(major: @search_params[:major])
    end
    #Filter by graduation date if the date and before/after is provided
    if @search_params[:grad_date_filter].present? &&
@search_params[:graduation_date].present?
    begin
      selected_date = Date.parse(@search_params[:graduation_date])
      Rails.logger.info "Parsed Date: #{selected_date}"

      if @search_params[:grad_date_filter] == 'before'
        @students = @students.where("graduation_date < ?", selected_date)
      elsif @search_params[:grad_date_filter] == 'after'
        @students = @students.where("graduation_date > ?", selected_date)
      end

    rescue ArgumentError
      #Return no results if date is invalid
      flash.now[:alert] = "Invalid date format"
      @students = Student.none
    end
  end

else
  #If no search feilds are filled, show an alert or show no students
  flash.now[:alert] = "Please provide at least one search criteria"
  @students = Student.none
end

  Rails.logger.info "Search Params: #{@search_params.inspect}"
  Rails.logger.info "Resulting Students: #{@search.inspect}"
end

# GET /students/1 or /students/1.json
def show
end

# GET /students/new
def new
  @student = Student.new
end

# GET /students/1/edit
```

```
 def edit
 end

 # POST /students or /students.json
 def create
   @student = Student.new(student_params)

   respond_to do |format|
     if @student.save
       format.html { redirect_to student_url(@student), notice: "Student was 
successfully created." }
       format.json { render :show, status: :created, location: @student }
     else
       format.html { render :new, status: :unprocessable_entity }
       format.json { render json: @student.errors, status: :unprocessable_entity }
     end
   end
 end

 # PATCH/PUT /students/1 or /students/1.json
 def update
   respond_to do |format|
     if @student.update(student_params)
       format.html { redirect_to student_url(@student), notice: "Student was 
successfully updated." }
       format.json { render :show, status: :ok, location: @student }
     else
       format.html { render :edit, status: :unprocessable_entity }
       format.json { render json: @student.errors, status: :unprocessable_entity }
     end
   end
 end

 # DELETE /students/1 or /students/1.json
 def destroy
   @student.destroy!

   respond_to do |format|
     format.html { redirect_to students_url, notice: "Student was successfully 
destroyed." }
     format.json { head :no_content }
   end
 end

 private
   # Use callbacks to share common setup or constraints between actions.
   def set_student
     @student = Student.find(params[:id])
   end

   # Only allow a list of trusted parameters through.
   def student_params
     params.require(:student).permit(:first_name, :last_name, :school_email, :major,
:minor, :graduation_date, :profile_picture)
   end
end
```

_search_form.html.erb

```
<%= form_with(url: students_path, method: :get, local: true, class: "form-inline") do
|form| %>
   <div>
   <!--Choose the Major-->
     <%= form.label :major, "Select Major", class: "form-label" %>
     <%= form.select :major, options_for_select(Student::VALID_MAJORS), { prompt: "Any
Major" }, name: 'search[major]', class: "form-control" %>
   </div>

   <div>
   <!--Graduation date filter(Before or After)-->
       <%= form.label :grad_date_filter, "Graduation Date Filter", class: "form-label"
%>
       <%= form.radio_button :grad_date_filter, 'before', checked: params.dig(:search,
:grad_date_filter) == 'before',  name: 'search[grad_date_filter]' %> Before
       <%= form.radio_button :grad_date_filter, 'after', checked: params.dig(:search,
:grad_date_filter) == 'after',  name: 'search[grad_date_filter]' %> After
   </div>

   <div>
   <!--Date Picker-->
       <%= form.label :graduation_date, "Graduation Date", class: "form-label" %>
       <%= form.date_field :graduation_date, value: params.dig(:search,
:graduation_date), name: 'search[graduation_date]', class: "form-control" %>
```

```
   </div>

   <div>
   <!--submit button and show all button-->
     <%= form.submit "Search", class: "btn btn-primary" %>
     <%= link_to "Show All", students_path(show_all: true), class: "btn btn-secondary"
%>
   </div>
<% end %>
```

index.html.erb

```
<p style="color: green"><%= notice %></p>

<h1>Students</h1>
<% if flash[:alert] %>
   <p style="color: red"><%= flash[:alert] %></p>
<% end %>

<%= render 'search_form', search_params: @search_params %>



<%= link_to "New Student", new_student_path %>

<table class="table">
 <thead>
   <tr>
     <th>First Name</th>
     <th>Last Name</th>
     <th>Major</th>
     <th>Graduation Date</th>
     <th>Actions</th>
   </tr>
 </thead>
 <tbody>
   <% @students.each do |student| %>
     <tr>
       <td><%= student.first_name %></td>
       <td><%= student.last_name %></td>
       <td><%= student.major %></td>
       <td><%= student.graduation_date %></td>
       <td>
         <%= link_to 'Show', student_path(student), class: "btn btn-secondary
hover-grow" %>
         <%= link_to 'Edit', edit_student_path(student), class: "btn btn-success
hover-grow" %>
         <%= link_to 'Delete', student_path(student), method: :delete, data: {
confirm: "Are you sure?" }, class: "btn btn-danger hover-grow" %>
       </td>
     </tr>
   <% end %>
 </tbody>
</table>
```

Rspec Testing Assignment: William Maddock

1. Overview of Rspec and Its Importance in Testing

Introduction to RSpec:

RSpec is a popular testing framework for Ruby and Ruby on Rails applications, providing developers with a clear and expressive syntax to write tests. It is designed to support **Behavior-Driven Development (BDD)**, a methodology that focuses on defining how an application should behave from the user's perspective. With RSpec, you can write tests for your application's models, controllers, views, and APIs, ensuring that each part of the application works as expected.

2. Getting Started with Rspec in Rails

Setting Up Rspec: Provide a step-by-step guide on how to add Rspec to a Rails project.

- Add the following to your Gemfile:

```
group :development, :test do
  gem 'rspec-rails'
end
```

- Then, run the following commands:

```
bundle install
rails generate rspec:install
```

File Structure:

Request Specs: Located in spec/requests/, these specs are used to test HTTP requests to your application's endpoints. They simulate requests to the controllers and ensure that the responses are what you expect. In the context of the Students feature, the request specs are testing the various routes (index, show, create, update, delete) for the Students resource.

Example:
spec/requests/students_spec.rb

Model Specs: Stored in spec/models/, these specs test the models in your application. They ensure that your ActiveRecord models behave as expected, including validations, associations, and custom methods.

Example:
spec/models/student_spec.rb

Controller Specs (Deprecated): While Rails controller specs were commonly used in older Rails applications, they are now less popular, with request specs being preferred. Controller specs test the internal workings of the controller actions but are typically less comprehensive than request specs.

Example:
spec/controllers/students_controller_spec.rb

Feature Specs: These live in spec/features/ and test the application's functionality from the user's perspective. Feature specs involve simulating user actions (like visiting a page, filling out forms, etc.) to ensure the app behaves correctly in real-world scenarios.

Factories/Fixtures: Many tests will also rely on factories (defined using FactoryBot or similar libraries) or fixtures. These files can be found in spec/factories/ and contain blueprints for creating instances of models during the tests.

Helper Specs and Support Files: Additional support files, helpers, and shared examples may be included in spec/support/ or spec/helpers/.

3. Assigned Tests and Tasks

Pagination of Search Results (Optional):

This test involves ensuring that when users search for data (e.g., students, classes), the results are properly paginated.

Key points for testing pagination:

- Ensure that a set number of results are displayed on each page (commonly 10 or 20 results per page).
- Verify that there are navigation links/buttons to access the next or previous pages.
- Check edge cases where the search might return very few results or no results at all.
- Confirm that the results displayed on a specific page are accurate and do not overlap with results from other pages.

RSpec Tests for CRUD (Create, Read, Update, Delete) Functionality:

CRUD operations form the backbone of any application dealing with data management, and testing these functions ensures that the core business logic behaves as expected.

Create:

- Test that new data can be created successfully (e.g., creating a new student or record).
- Ensure that all necessary validations are checked before creating the record, such as mandatory fields.
- Handle invalid input cases (e.g., missing required fields) to ensure proper error messages are displayed.

Read:

- Verify that existing data can be read and displayed correctly.
- Ensure that the right data is fetched based on the user's query or filter (e.g., fetching student details by ID).

Update:

- Confirm that data can be updated successfully (e.g., modifying a student's details).
- Ensure proper validations when updating data (e.g., invalid input or duplicate values should not be allowed).
- Test the behavior when the user attempts to update non-existent records.

Delete:

- Test that records can be deleted and that the deletion is reflected in the database.
- Handle cases where the user attempts to delete a non-existent or invalid record.
- Verify that the correct confirmation or success messages are shown upon successful deletion.

4. Writing Rspec Tests for CRUD Functionality

Create a New Student (POST /students):

Ensure that a new student is created when valid parameters are provided. Test cases should verify error handling:

```
it "creates a new student and redirects" do
  expect {
    post students_path, params: { student: { first_name: "John", last_name: "Doe",
school_email: "john.doe@school.com", major: "Computer Science", graduation_date: "2025-
05-15" } }
  }.to change(Student, :count).by(1)

  expect(response).to have_http_status(:found) # Expect redirect after creation
end
```

Show a Student (GET /students/:id):

Test the display of a student's information when the student exists. Write a test for handling a non-existent student:

```
it "returns a 200 OK status and shows the student's details" do
   get student_path(student)
   expect(response).to have_http_status(:ok)
   expect(response.body).to include(student.first_name)
end
```

Edit/Update Student (PATCH /students/:id)

Tests for Updating a Student:

```
it 'updates a student and redirects to the student profile page' do

  student = Student.create(name: 'John Doe', email: 'john@example.com')


  patch "/students/#{student.id}", params: { student: { name: 'Jane Doe' } }


  student.reload

  expect(student.name).to eq('Jane Doe')

  expect(response).to redirect_to(student_path(student))

end
```

Test for Invalid Update (e.g., Missing Required Fields):

```
it 'does not update the student with invalid data and re-renders the edit page' do

  student = Student.create(name: 'John Doe', email: 'john@example.com')
  patch "/students/#{student.id}", params: { student: { name: '' } } # Invalid data

  student.reload
  expect(student.name).to eq('John Doe') # Name should remain unchanged
  expect(response).to render_template(:edit)
end
```

Delete a Student (DELETE /students/:id):

```
it 'deletes a student and redirects to the students list' do
  student = Student.create(name: 'John Doe', email: 'john@example.com')

  delete "/students/#{student.id}"

  expect(Student.exists?(student.id)).to be_falsey # Student should be deleted
  expect(response).to redirect_to(students_path)
end
```

Test for Deleting a Non-existent Student:

```
it 'handles deletion of a non-existent student' do
  delete "/students/9999" # Assuming this ID does not exist
  expect(response).to redirect_to(students_path)

  expect(flash[:alert]).to be_present # Optional: Ensure that a flash message is displayed

ends
```

5. Optional Pagination Testing

Test the pagination of search results, ensuring the correct number of results are shown on each page:

```
it "paginates results and returns the correct number of students per page" do
    get students_path, params: { page: 1, per_page: 10 }
    expect(response.body).to include("Showing 10 students")
end
```

Here is the same test only more in depth. I used this test in my portfolio:

```
it "paginates results and returns the correct number of students per page" do
  # Create 25 students for pagination test
  24.times do |i|
    Student.create!(first_name: "Student #{i + 1}", last_name: "Test", school_email: "test#{i +
1}@msudenver.edu", major: "Computer Science BS", graduation_date: "2025-05-15")
  end

  # Verify that the students were created
  expect(Student.count).to eq(25) # Ensure 25 students exist
end
```

7. Helpful Resources

Rspec Documentation: https://rspec.info/documentation/

Rails Guides on Testing: https://guides.rubyonrails.org/testing.html

Arc Toolkit: https://www.tpgi.com/arc-platform/arc-toolkit/