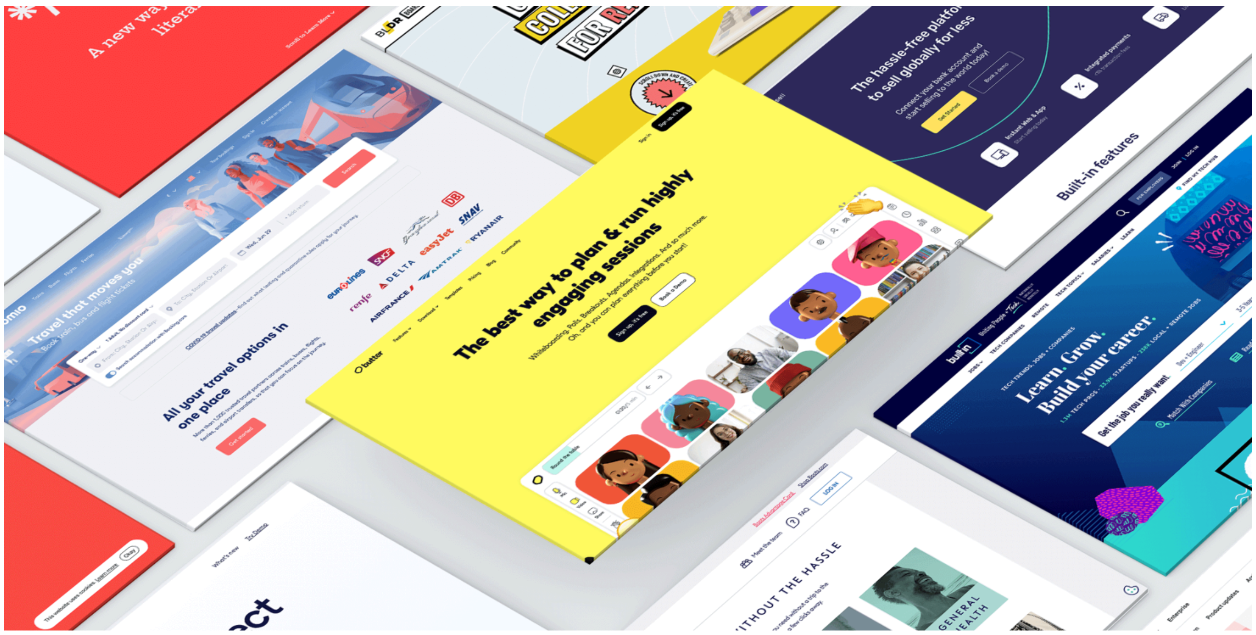
User Interface and Bootstrap Design

Table of Contents:



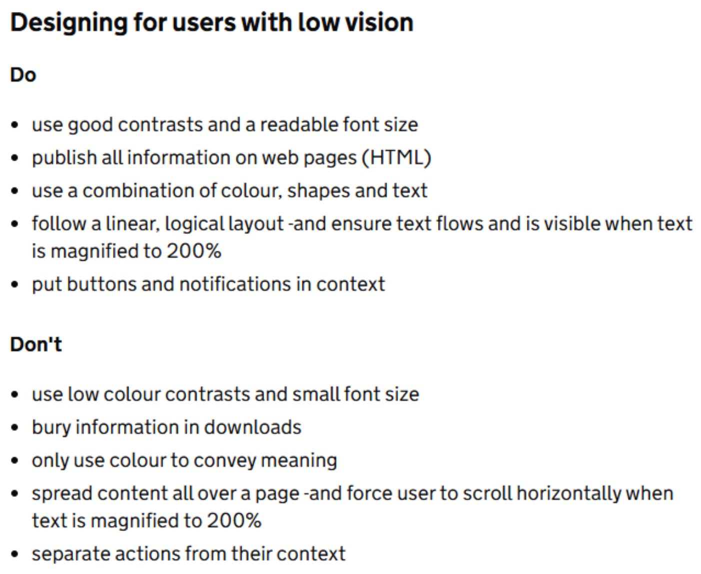
(Credit to https://www.uxdesigninstitute.com/blog/good-ui-design-9-examples/ for image)

Why it's important to have intuitive and good UI design:

When creating a web application, it's important to have a User Interface that is helpful and is user-friendly to guide your users to whatever you are designing without issues. However, what does it mean to be helpful and user-friendly? While there are many ways to design something helpful and user-friendly, a great start is to follow the design standards from these:

- https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/
- https://www.w3schools.com/whatis/whatis_responsive.asp

The following resources provide a baseline of what should be intuitive for developers to implement for their users. For example, this below poster from Accessibility from Gov.uk deals with users with low vision. Users with low vision might need the “do’s” listed below to properly navigate through your application, while the “don’ts” need to avoid as much as possible to benefit your users. This would mean that, understanding that developers need to take account for users that might or need the additional help to fulfill the goal of your application



Designing for users with low vision

Do

- use good contrasts and a readable font size
- publish all information on web pages (HTML)
- use a combination of colour, shapes and text
- follow a linear, logical layout -and ensure text flows and is visible when text is magnified to 200%
- put buttons and notifications in context

Don't

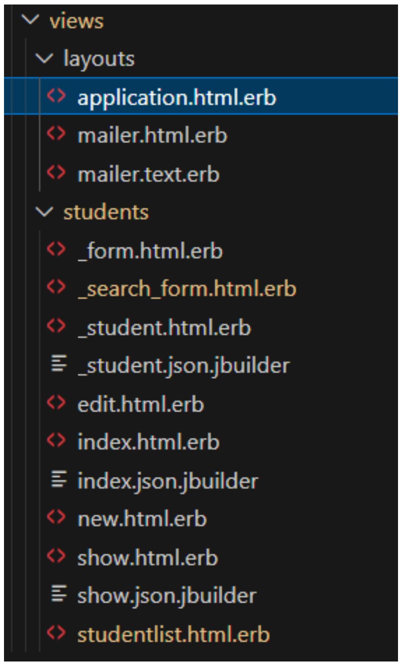
- use low colour contrasts and small font size
- bury information in downloads
- only use colour to convey meaning
- spread content all over a page -and force user to scroll horizontally when text is magnified to 200%
- separate actions from their context

(image is from https://accessibility.blog.gov.uk/2016/09/02/dos-and-donts-on-designing-for-accessibility/)

Keep in mind that these resources are not the only way to improve your UI. There are plenty of different ways to improve your UI.

Where do I create and edit my UI?

Typically when creating your application, the HTML files will be the center of focus for all User Interface interactions. For example, make sure to keep on eye on your files as show below:



```
views
  layouts
    application.html.erb
    mailer.html.erb
    mailer.text.erb
  students
    _form.html.erb
    _search_form.html.erb
    _student.html.erb
    _student.json.jbuilder
    edit.html.erb
    index.html.erb
    index.json.jbuilder
    new.html.erb
    show.html.erb
    show.json.jbuilder
    studentlist.html.erb
```

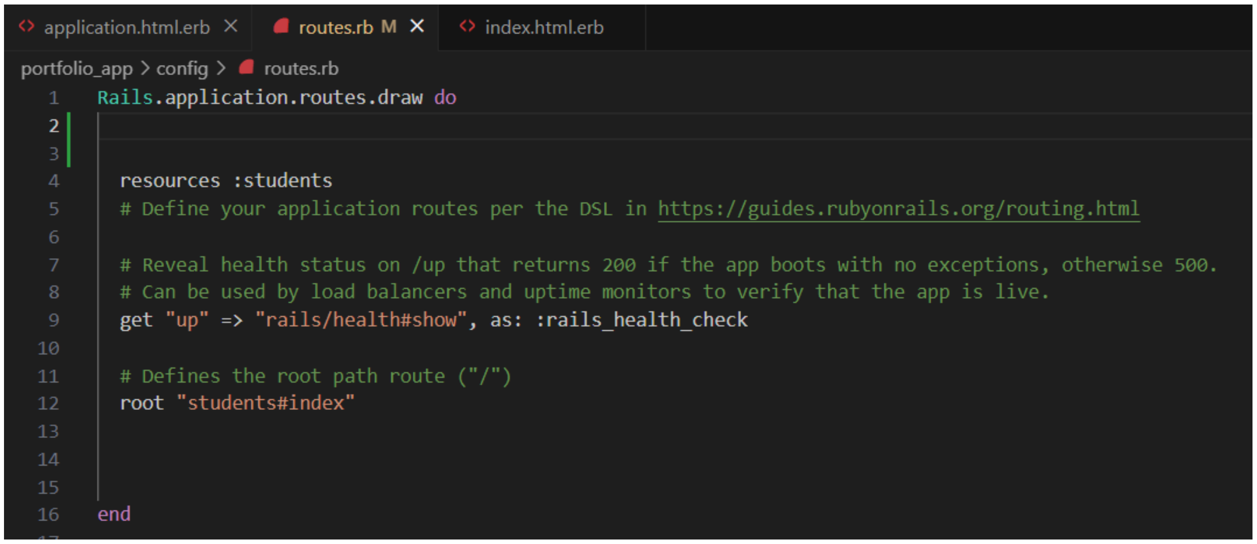
(Within the "Views" folder, we can see multiple html files to see what the users can possible access in our application)

In addition, make sure to take a look at what page or pages when starting up your program, this is important to understand where a user will start when the application or website runs as a developer will form their User Interface around controlling where Users start.

For example:

- Users start on a "Login" page to indicate to log in to there account
- Users start on a message to inform their next step for the application
- You don't want every User to start on a list of private information

You can find the location of where your application starts in a “routes.rb” or “routes”-type file as shown below:



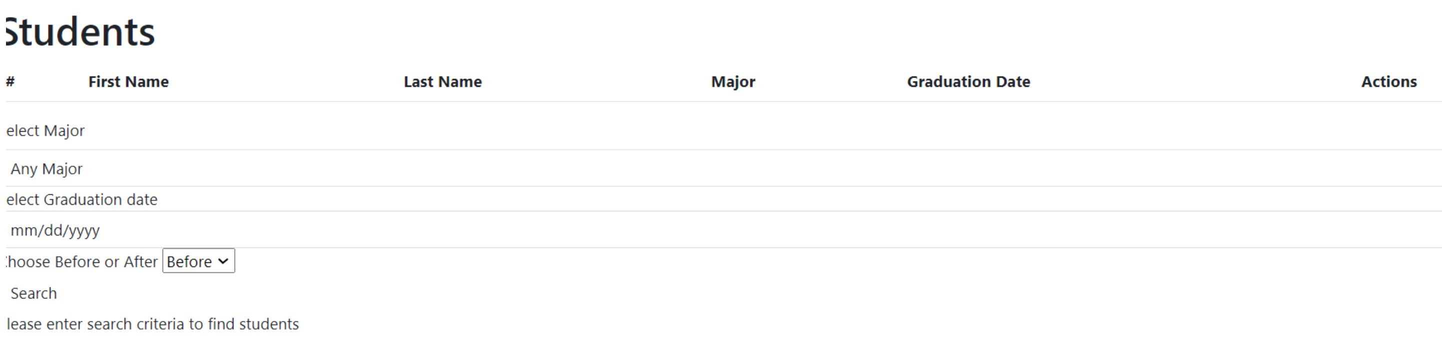
```
Rails.application.routes.draw do


  resources :students
  # Define your application routes per the DSL in https://guides.rubyonrails.org/routing.html

  # Reveal health status on /up that returns 200 if the app boots with no exceptions, otherwise 500.
  # Can be used by load balancers and uptime monitors to verify that the app is live.
  get "up" => "rails/health#show", as: :rails_health_check

  # Defines the root path route ("/")
  root "students#index"



end
```

(In the image when users start the application, the first location they are sent to showing:: “**root “studentsindex”**” which is the students index)

Understanding User Interface vs. Bootstrap User Interface Design

When working with formatting for the User Interface and the Bootstrap User Interface, it’s important to know that the Bootstrap’s UI is the layer placed on top of the normal user interface. This means that your Bootstrap’s UI is **always** viewable no matter where your applications go or routes to. **It’s highly recommended to avoid being information concerning a database on the Bootstrap layer to avoid errors and for privacy reasons.** Instead, think about the Bootstrap layer to benefit universally all users. You want this to help with usability, utility, and convenience.

An example of Index.html without any spring boot application



Students

#	First Name	Last Name	Major	Graduation Date	Actions

elect Major

Any Major

elect Graduation date

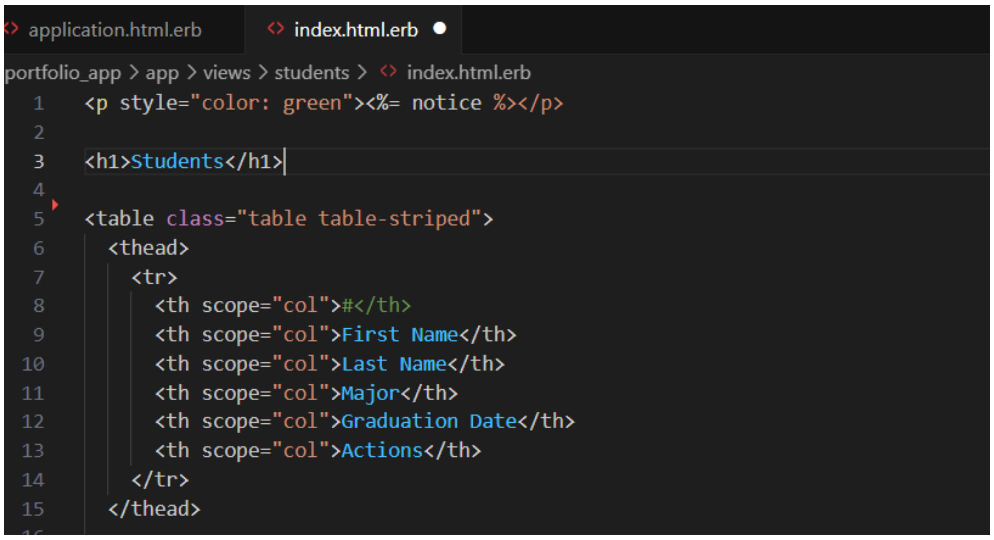
mm/dd/yyyy

:hoose Before or After Before

Search

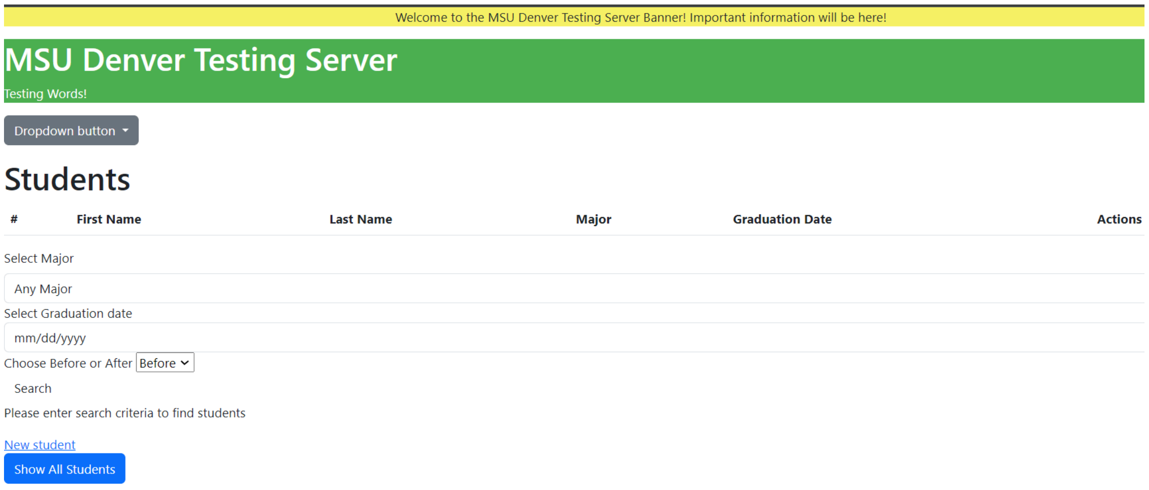
lease enter search criteria to find students

Example of the code above:



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

<h1>Students</h1>

<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First Name</th>
      <th scope="col">Last Name</th>
      <th scope="col">Major</th>
      <th scope="col">Graduation Date</th>
      <th scope="col">Actions</th>
    </tr>
  </thead>
```

Example of a possible Bootstrap layer on index:


Welcome to the MSU Denver Testing Server Banner! Important information will be here!

MSU Denver Testing Server

Testing Words!

Students

#	First Name	Last Name	Major	Graduation Date	Actions

Select Major

Any Major

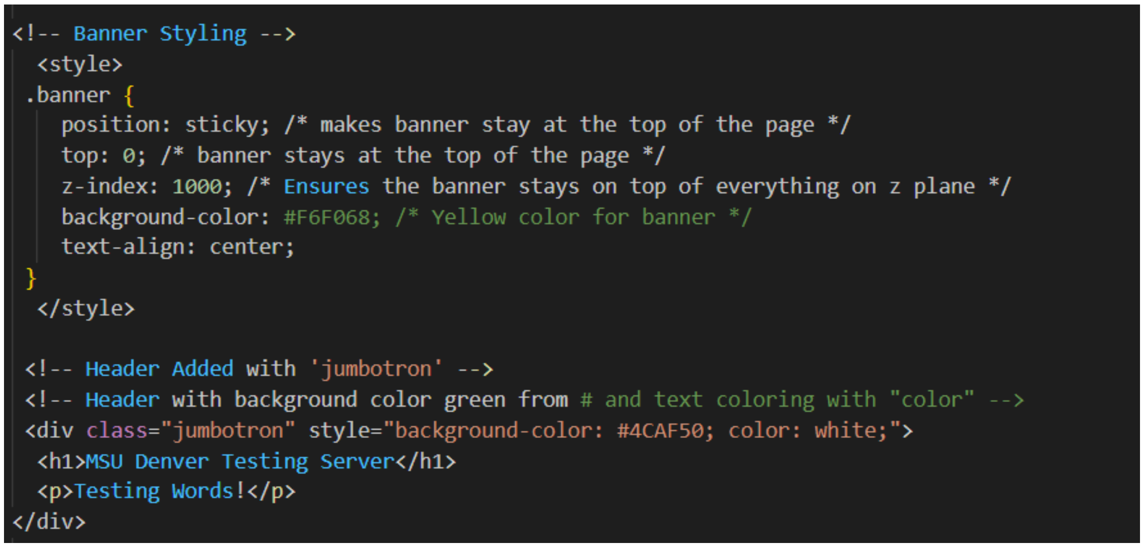
Select Graduation date

mm/dd/yyyy

Choose Before or After Before

Please enter search criteria to find students

Code for Bootstrap layer called “application.html”



```
<!-- Banner Styling -->
  <style>
 .banner {
    position: sticky; /* makes banner stay at the top of the page */
    top: 0; /* banner stays at the top of the page */
    z-index: 1000; /* Ensures the banner stays on top of everything on z plane */
    background-color: #F6F068; /* Yellow color for banner */
    text-align: center;
 }
  </style>

 <!-- Header Added with 'jumbotron' -->
 <!-- Header with background color green from # and text coloring with "color" -->
 <div class="jumbotron" style="background-color: #4CAF50; color: white;">
  <h1>MSU Denver Testing Server</h1>
  <p>Testing Words!</p>
</div>
```

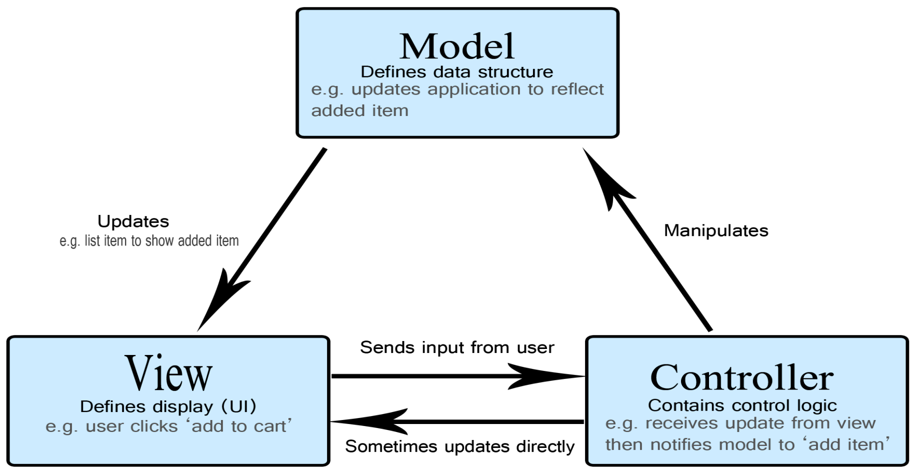
<u>Possibilities with Bootstrap:</u>

As you can see with the example images above, you can see that in image one without the Bootstrap layer the design is still usable, but very simple. While in images three, there is varied information to welcome the user and alert them with information through the banner at the top. **However, this is not the only way to design a user interface that is helpful and user-friendly. There are many ways to design your UI with Bootstrap to help the user such as:**

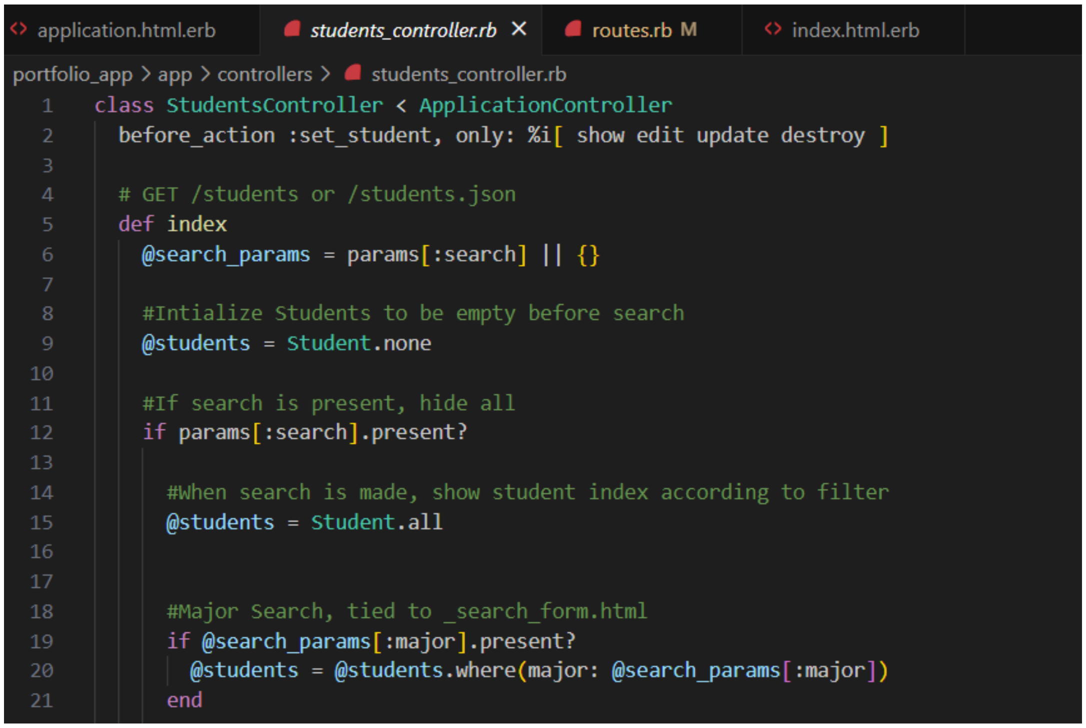
- A button to help navigate to a certain page
- Images that would link to a website
- A sidebar that when hover, displays a menu to prevent clutter on the screen for users
- Banners to notify users about important information
- Color-coded messages or buttons to help with specific users
  - For example: A red button labeled "**Delete"** would help users to be cautious.
  - A blue button labeled "**Submit"** would help users identify where to confirm a data entry

Understanding the "View" for the Model, View, Controller(MVC)

Another aspect to keep in mind is the thought of the MVC model for this project and other future projects. The Model, View, Controller(MVC) model is important to keep in mind when you are designing your application. The importance of the "View" is what the Users will see when the Controller sends data from the Model/database to the Views for users, meaning that the "View" needs to be designed to display requested data.

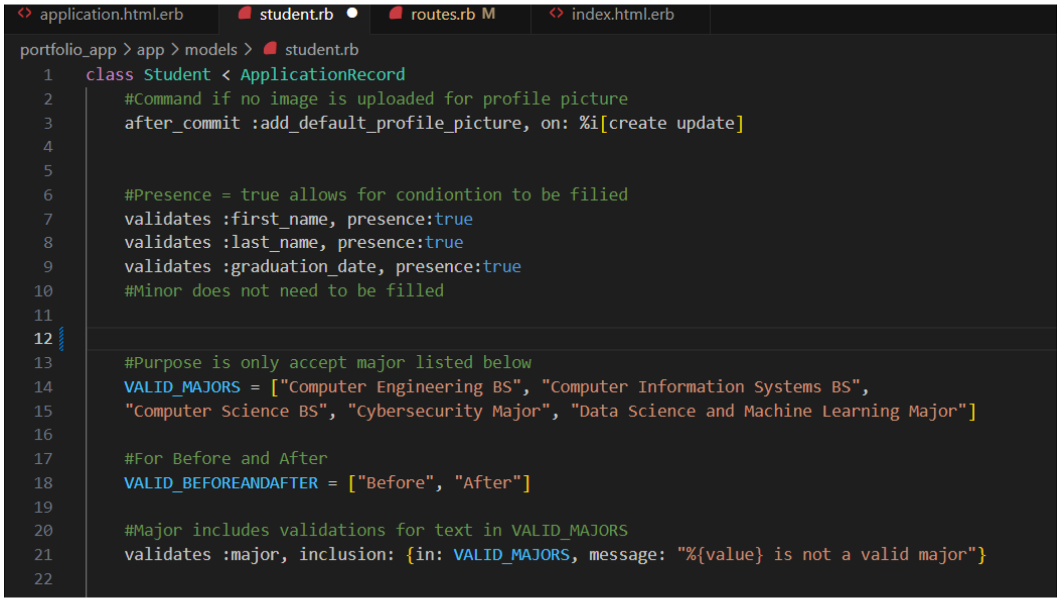

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

In our application, it would look like this::



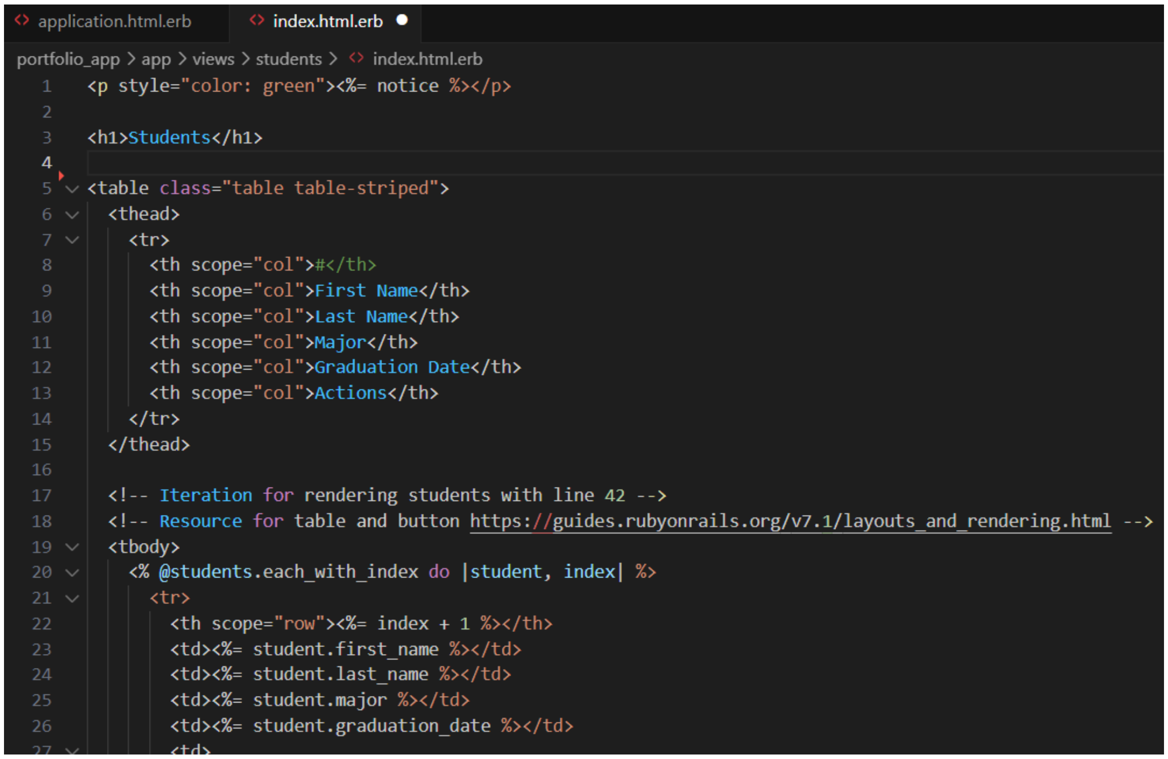
```ruby
class StudentsController < ApplicationController
  before_action :set_student, only: %i[ show edit update destroy ]

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

    #Intialize Students to be empty before search
    @students = Student.none

    #If search is present, hide all
    if params[:search].present?

      #When search is made, show student index according to filter
      @students = Student.all


      #Major Search, tied to _search_form.html
      if @search_params[:major].present?
        @students = @students.where(major: @search_params[:major])
      end
```

1. In the image above, the user would search/request a major for the controller to filter out students.



```ruby
class Student < ApplicationRecord
    #Command if no image is uploaded for profile picture
    after_commit :add_default_profile_picture, on: %i[create update]


    #Presence = true allows for condiontion to be filied
    validates :first_name, presence:true
    validates :last_name, presence:true
    validates :graduation_date, presence:true
    #Minor does not need to be filled


    #Purpose is only accept major listed below
    VALID_MAJORS = ["Computer Engineering BS", "Computer Information Systems BS",
    "Computer Science BS", "Cybersecurity Major", "Data Science and Machine Learning Major"]

    #For Before and After
    VALID_BEFOREANDAFTER = ["Before", "After"]

    #Major includes validations for text in VALID_MAJORS
    validates :major, inclusion: {in: VALID_MAJORS, message: "%{value} is not a valid major"}

```

2. The controller will search for students in the major based off the model(image above)



```
1   <p style="color: green"><%= notice %></p>
2
3   <h1>Students</h1>
4
5   <table class="table table-striped">
6     <thead>
7       <tr>
8         <th scope="col">#</th>
9         <th scope="col">First Name</th>
10        <th scope="col">Last Name</th>
11        <th scope="col">Major</th>
12        <th scope="col">Graduation Date</th>
13        <th scope="col">Actions</th>
14      </tr>
15    </thead>
16
17    <!-- Iteration for rendering students with line 42 -->
18    <!-- Resource for table and button https://guides.rubyonrails.org/v7.1/layouts_and_rendering.html -->
19    <tbody>
20      <% @students.each_with_index do |student, index| %>
21        <tr>
22          <th scope="row"><%= index + 1 %></th>
23          <td><%= student.first_name %></td>
24          <td><%= student.last_name %></td>
25          <td><%= student.major %></td>
26          <td><%= student.graduation_date %></td>
27          <td>
```

3. **The Controller will take the requested data from the database and display it according to the View(image above)**
    a. A table is created from Line 5 to Line 15
    b. From Line 20 to Line 26 is where the controller is displaying the students' information from the database into the view for users