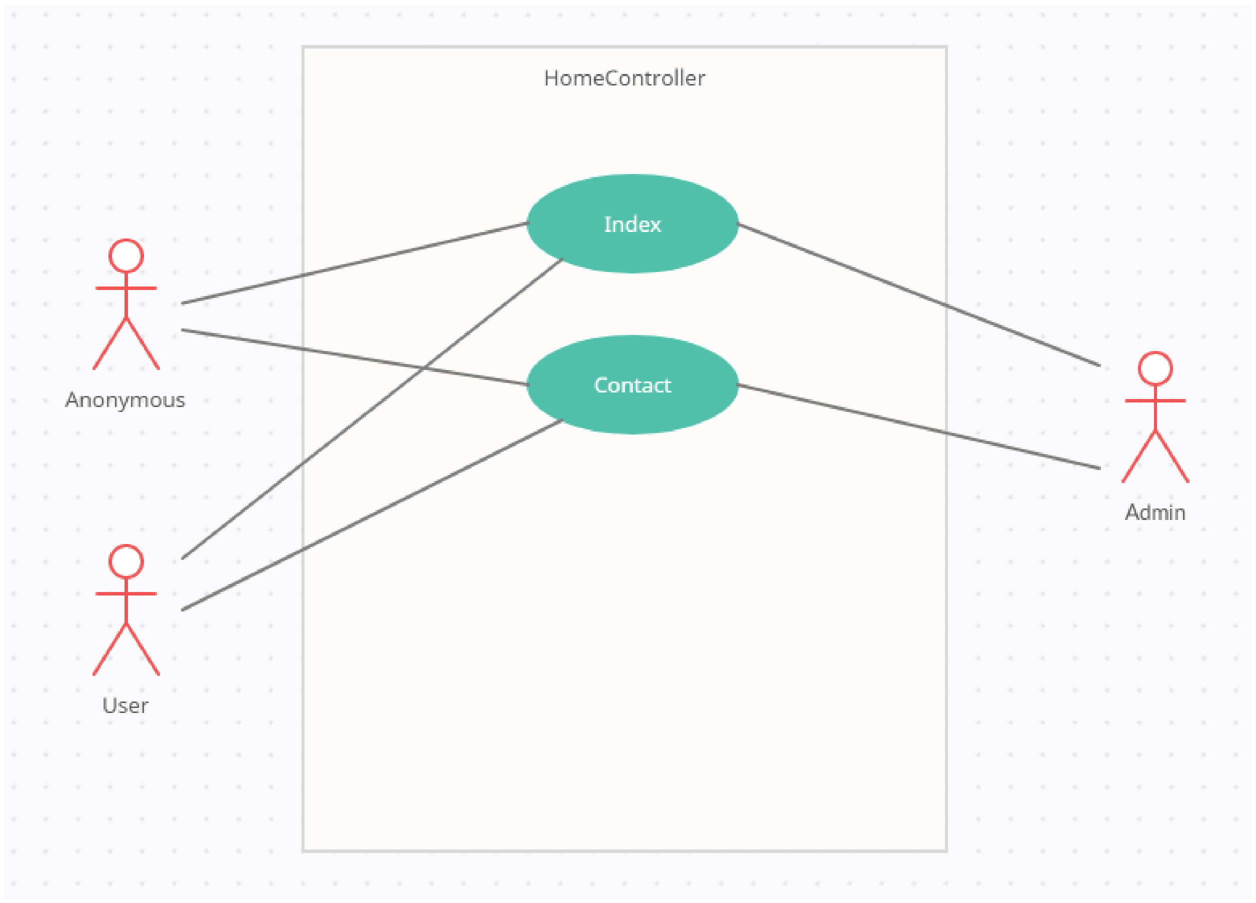
Enrollment Manager App

Ryan Palmer and Jae Taylor

This app is designed to allow staff members of a university scheduling team to manage and view courses and students within the University.

Enrollment Manager is an MVC web application with built in Role Security, User Registration, and a framework to add, edit, and delete entries on a remote repository.

The following pages will provide a brief description of the main controllers, followed by how we progressed through the building of the application.



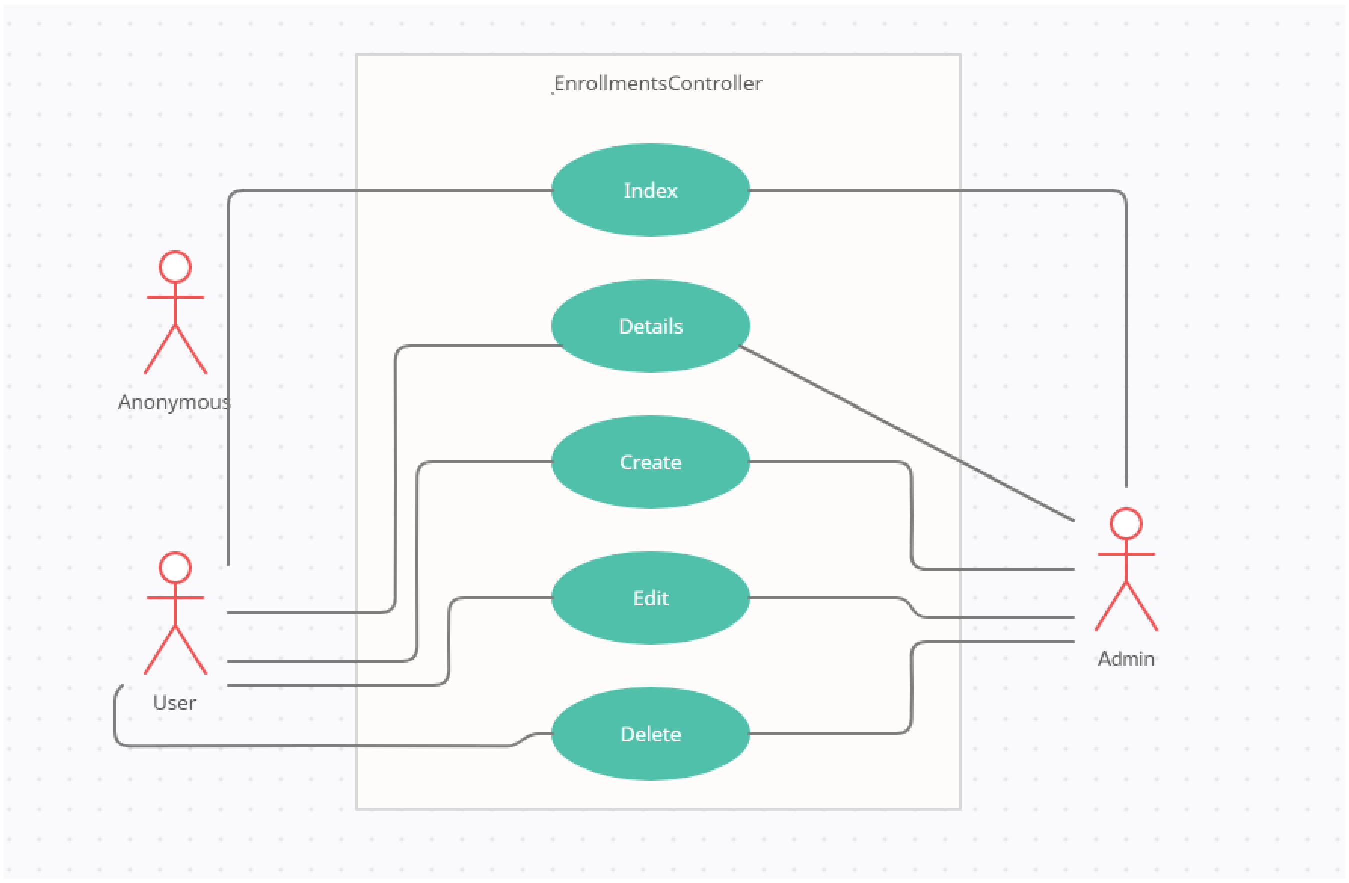
Home Controller

The home controller is the landing page for the site, and the only accessible controller to Anonymous Users who aren't logged in. The Index will either display a login form if the user isn't logged in, or it will greet them if they are logged in.

In addition, the nav bar will dynamically change to only show users what they have access to.

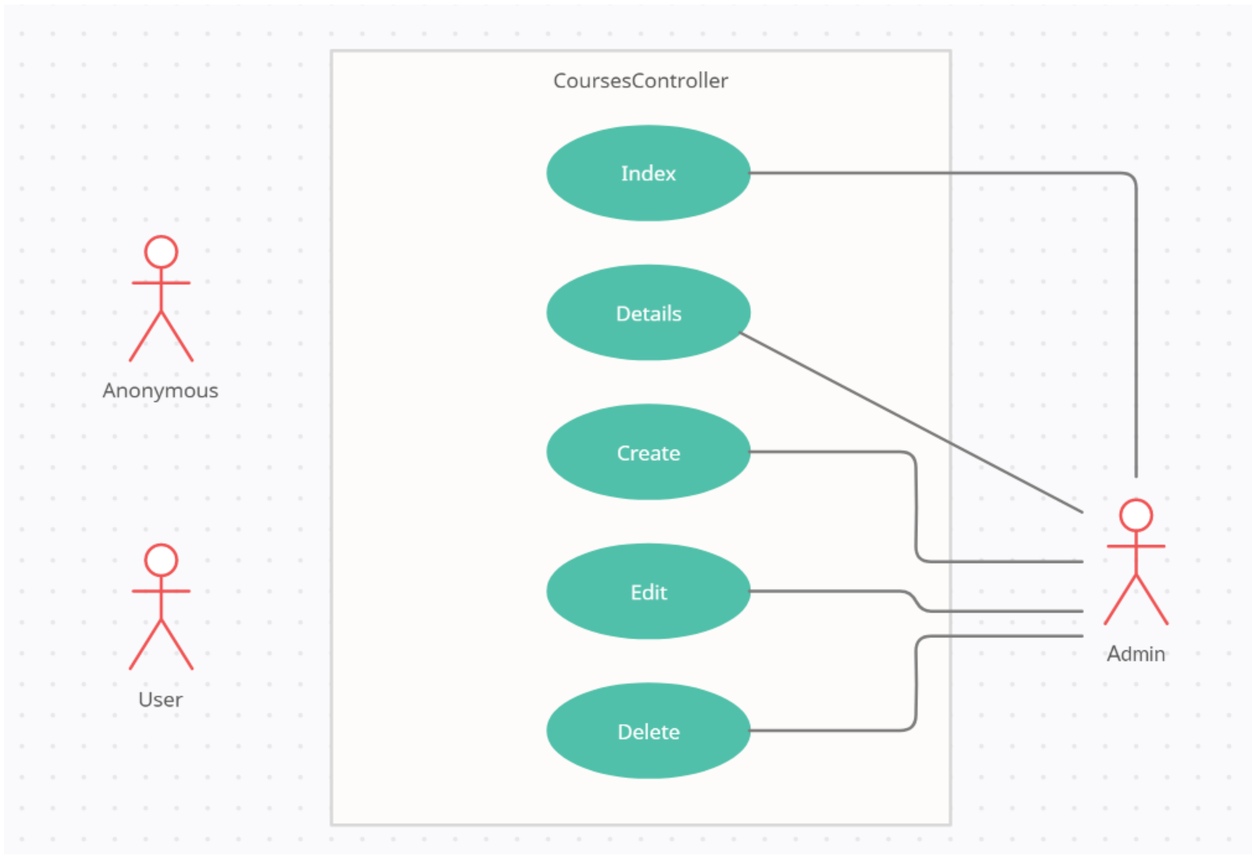
The index also includes a live count of several database sizes, including students, courses, classes, and enrollments.

The contact form would allow unsigned users to contact a technician to help them register their role, but for the purposes of this project, users can instead register on the site.



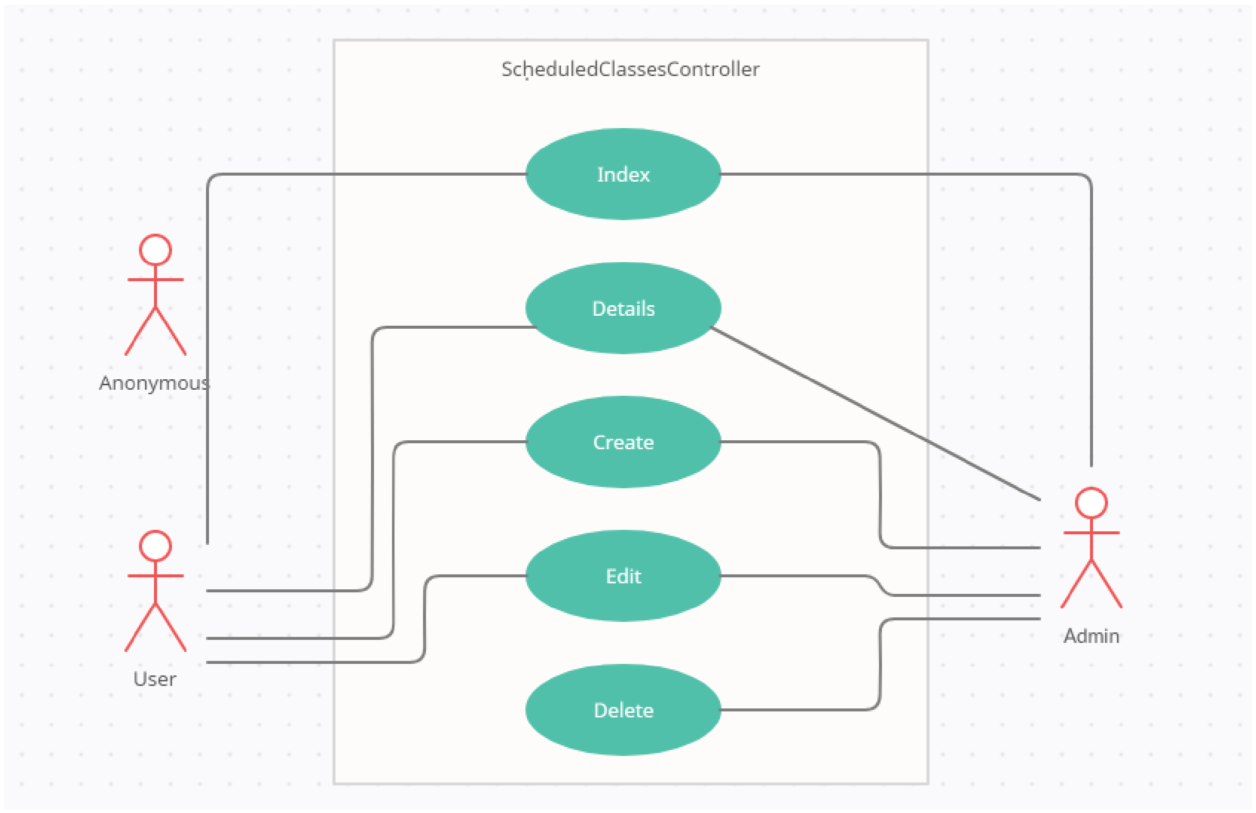
Enrollments Controller

The enrollments Controller is only accessible to those who are signed in, but those that can are provided full access to add, edit, or delete entries connecting students to their scheduled classes.



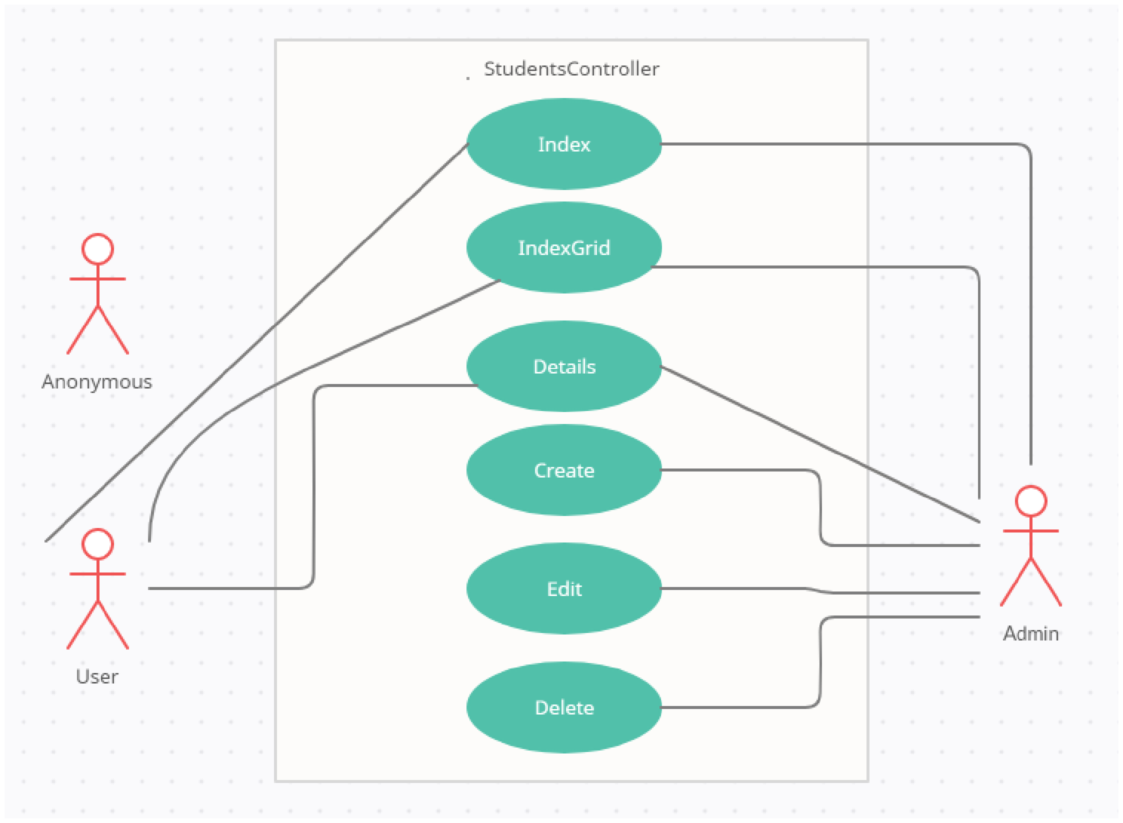
Courses Controller

The courses controller is only accessible by admins, and so if a change does need to be made, staff can reach out via the contact form to let an admin know.



Scheduled Classes Controller

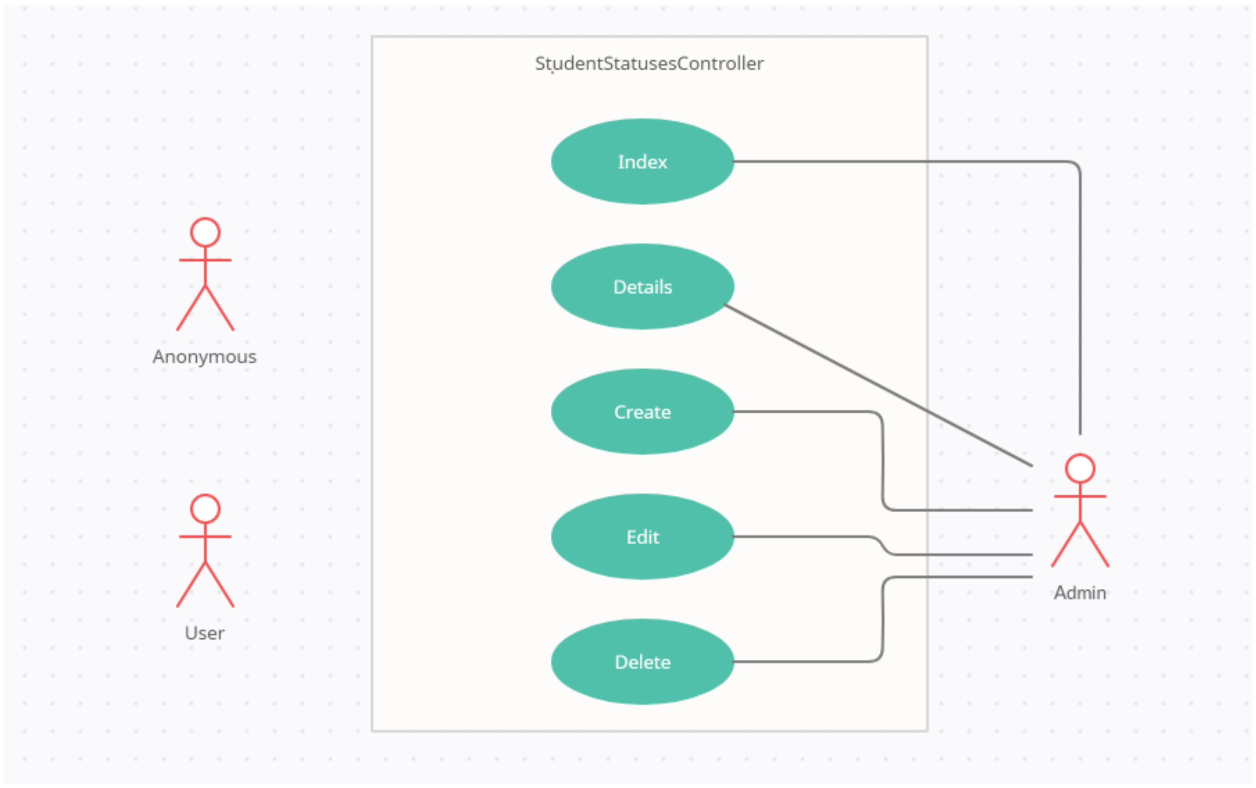
Similar to Enrollments, staff can access the Scheduled Class controller to establish new course classes to place students into. They cannot delete an entry once created, however, and must contact an admin to do so.



Students Controller

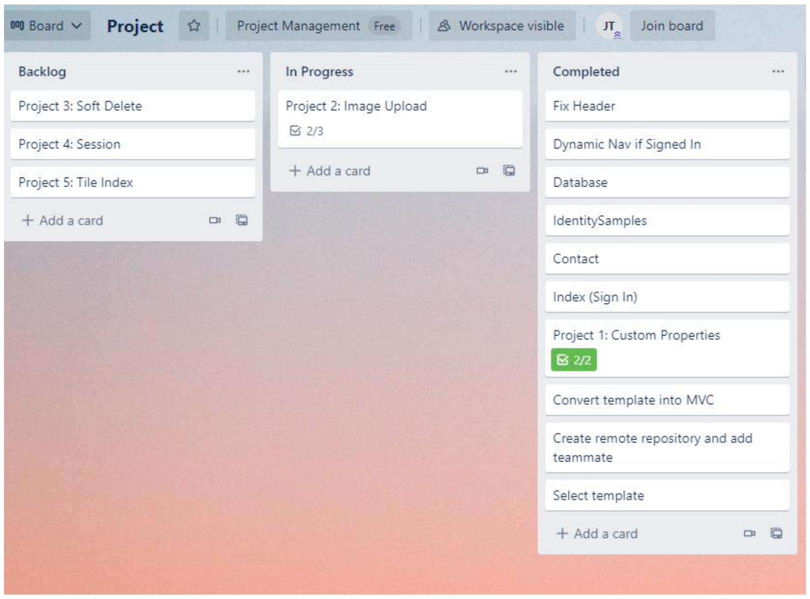
This is where the students at the University can be viewed and, if an Admin, created and edited.

This page's index can be viewed in either a list or a grid, and utilizes a Session Variable to remember which one you were on when you go into a student's details.



Student Statuses Controller

Like Courses, the student statuses controller is only accessible by admins, and so if a change does need to be made, which is unlikely, staff can reach out via the contact form to let an admin know.



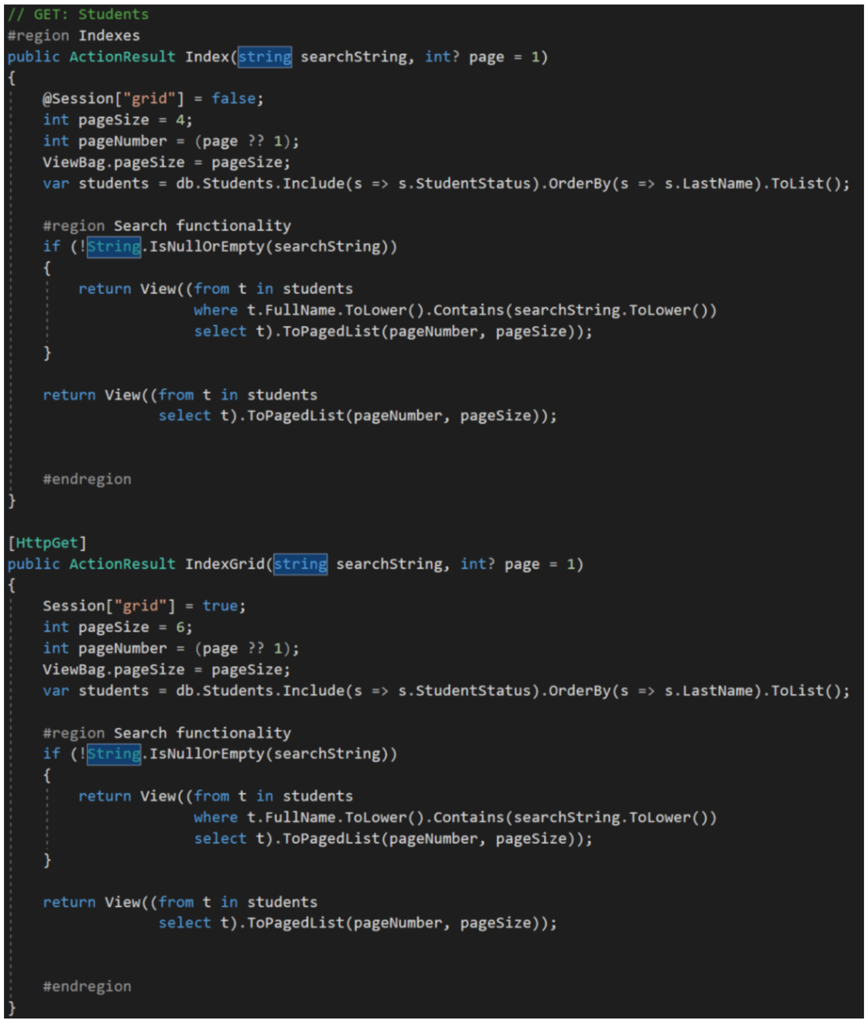
Backlog
- Project 3: Soft Delete
- Project 4: Session
- Project 5: Tile Index
- + Add a card

In Progress
- Project 2: Image Upload 2/3
- + Add a card

Completed
- Fix Header
- Dynamic Nav if Signed In
- Database
- IdentitySamples
- Contact
- Index (Sign In)
- Project 1: Custom Properties 2/2
- Convert template into MVC
- Create remote repository and add teammate
- Select template
- + Add a card

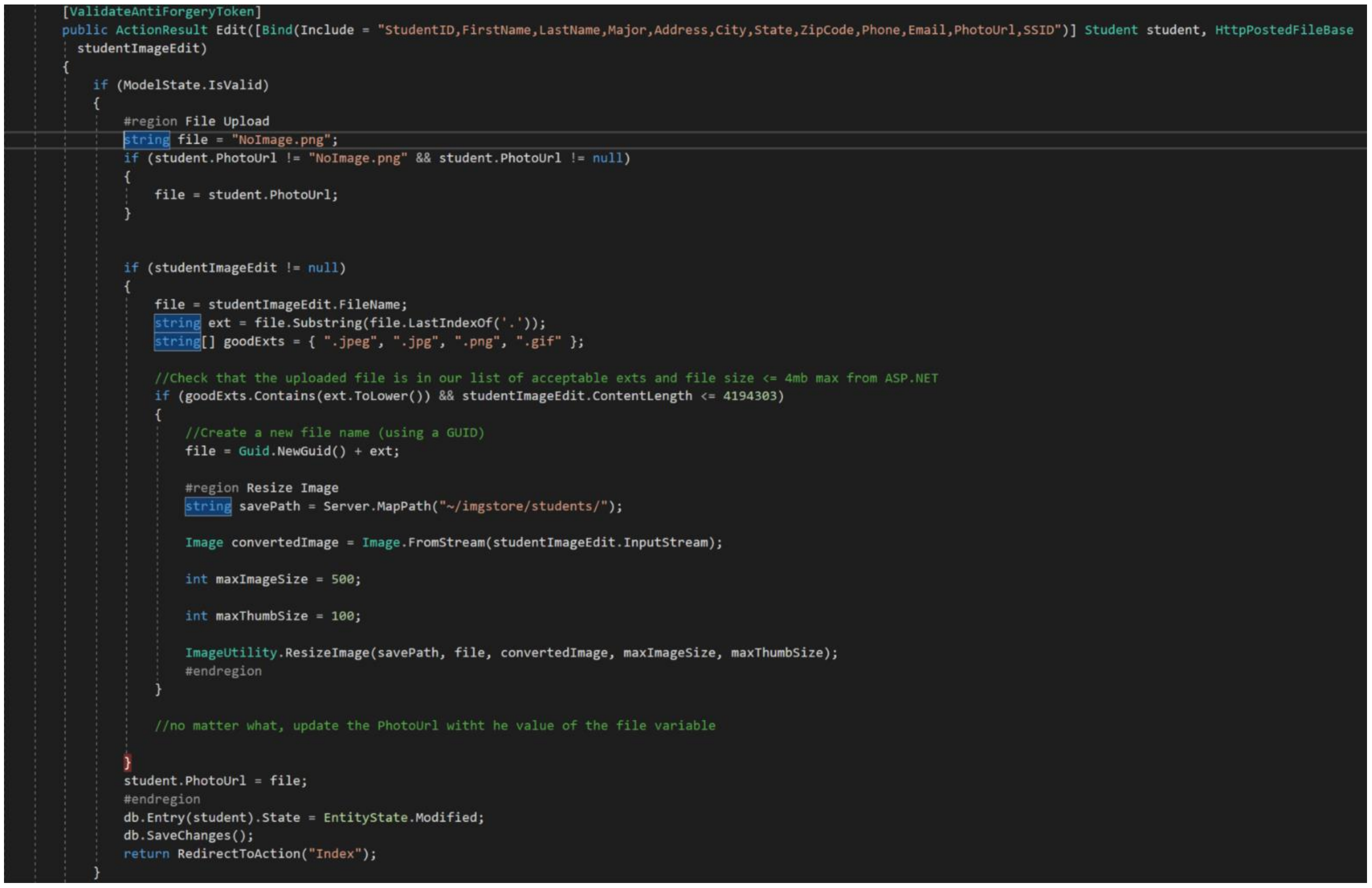
Progress and Challenges

We managed our goals and tasks through a Trello board, prioritizing more base level features to allow us to build on more advanced components.



```
// GET: Students
#region Indexes
public ActionResult Index(string searchString, int? page = 1)
{
    @Session["grid"] = false;
    int pageSize = 4;
    int pageNumber = (page ?? 1);
    ViewBag.pageSize = pageSize;
    var students = db.Students.Include(s => s.StudentStatus).OrderBy(s => s.LastName).ToList();

    #region Search functionality
    if (!String.IsNullOrEmpty(searchString))
    {
        return View((from t in students
                     where t.FullName.ToLower().Contains(searchString.ToLower())
                     select t).ToPagedList(pageNumber, pageSize));
    }

    return View((from t in students
                 select t).ToPagedList(pageNumber, pageSize));


    #endregion
}

[HttpGet]
public ActionResult IndexGrid(string searchString, int? page = 1)
{
    Session["grid"] = true;
    int pageSize = 6;
    int pageNumber = (page ?? 1);
    ViewBag.pageSize = pageSize;
    var students = db.Students.Include(s => s.StudentStatus).OrderBy(s => s.LastName).ToList();

    #region Search functionality
    if (!String.IsNullOrEmpty(searchString))
    {
        return View((from t in students
                     where t.FullName.ToLower().Contains(searchString.ToLower())
                     select t).ToPagedList(pageNumber, pageSize));
    }

    return View((from t in students
                 select t).ToPagedList(pageNumber, pageSize));


    #endregion
}
```

Code to check a session variable determining if the Index will be a Grid



```
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "StudentID,FirstName,LastName,Major,Address,City,State,ZipCode,Phone,Email,PhotoUrl,SSID")] Student student, HttpPostedFileBase
  studentImageEdit)
{
    if (ModelState.IsValid)
    {
        #region File Upload
        string file = "NoImage.png";
        if (student.PhotoUrl != "NoImage.png" && student.PhotoUrl != null)
        {
            file = student.PhotoUrl;
        }


        if (studentImageEdit != null)
        {
            file = studentImageEdit.FileName;
            string ext = file.Substring(file.LastIndexOf('.'));
            string[] goodExts = { ".jpeg", ".jpg", ".png", ".gif" };

            //Check that the uploaded file is in our list of acceptable exts and file size <= 4mb max from ASP.NET
            if (goodExts.Contains(ext.ToLower()) && studentImageEdit.ContentLength <= 4194303)
            {
                //Create a new file name (using a GUID)
                file = Guid.NewGuid() + ext;

                #region Resize Image
                string savePath = Server.MapPath("~/imgstore/students/");

                Image convertedImage = Image.FromStream(studentImageEdit.InputStream);

                int maxImageSize = 500;

                int maxThumbSize = 100;

                ImageUtility.ResizeImage(savePath, file, convertedImage, maxImageSize, maxThumbSize);
                #endregion
            }

            //no matter what, update the PhotoUrl witht he value of the file variable

        }
        student.PhotoUrl = file;
        #endregion
        db.Entry(student).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
```

Lab 2: Image Upload

Above is the code used to edit a students Image. When a new student is created without a supplied image, the image will default to a placeholder. In case an entry is somehow created without this placeholder, the placeholder will be assigned anyways at the top of the edit. This led to a problem of images being lost every time someone made an edit to something unrelated (like their name). To solve this, we check to make sure the current image isn't the placeholder or just empty, and if so, it'll set the designated edit to put their original picture back where it was.

After this, the app checks if the user supplied a new image to upload, and if so, runs the photo through some optimization. After that, the image is reassigned to the PhotoUrl of the Student, saves the changes, and takes the user back to the list.