

# Scheduling Administration Tool

An internal scheduling tool for schools and universities.

Worked on 1-19-21 to 1-22-21

Kansas City, MO United States

+1-123-456-7890

info@codeuniversity.edu

ENLIGHT

Home

Courses

Students

Enrollments

Scheduled Classes

Student Status

Account ▾

Students

Create New

Student	Major	Phone	Email	Status	
Pearson, Spencer	IT	816-555-1234	spencer@spencerpearson.net	Current Student	<div></div> <div></div> <div></div>
Swift, John	IT	913-555-3214	john@john.com	Current Student	<div></div> <div></div> <div></div>
Toney, Tessa	IT	913-333-3333	tessa@tony.com	Current Student	<div></div> <div></div> <div></div>
Beck, Isaac	IT	816-698-7654	isaac@beck.com	Current Student	<div></div> <div></div> <div></div>
See, Dave	IT	816-456-2585	david@seesaw.com	Current Student	<div></div> <div></div> <div></div>
Bell, Michael	IT	816-585-8585	michael@bell.com	Current Student	<div></div> <div></div> <div></div>
Arduark, Adam	IT	816-555-5555	ardark@ardark.com	Current Student	<div></div> <div></div> <div></div>

John Swift and Spencer Pearson

[jdavidswift@gmail.com](mailto:jdavidswift@gmail.com)

[spencer.pearson.816@gmail.com](mailto:spencer.pearson.816@gmail.com)

Dear Reader,

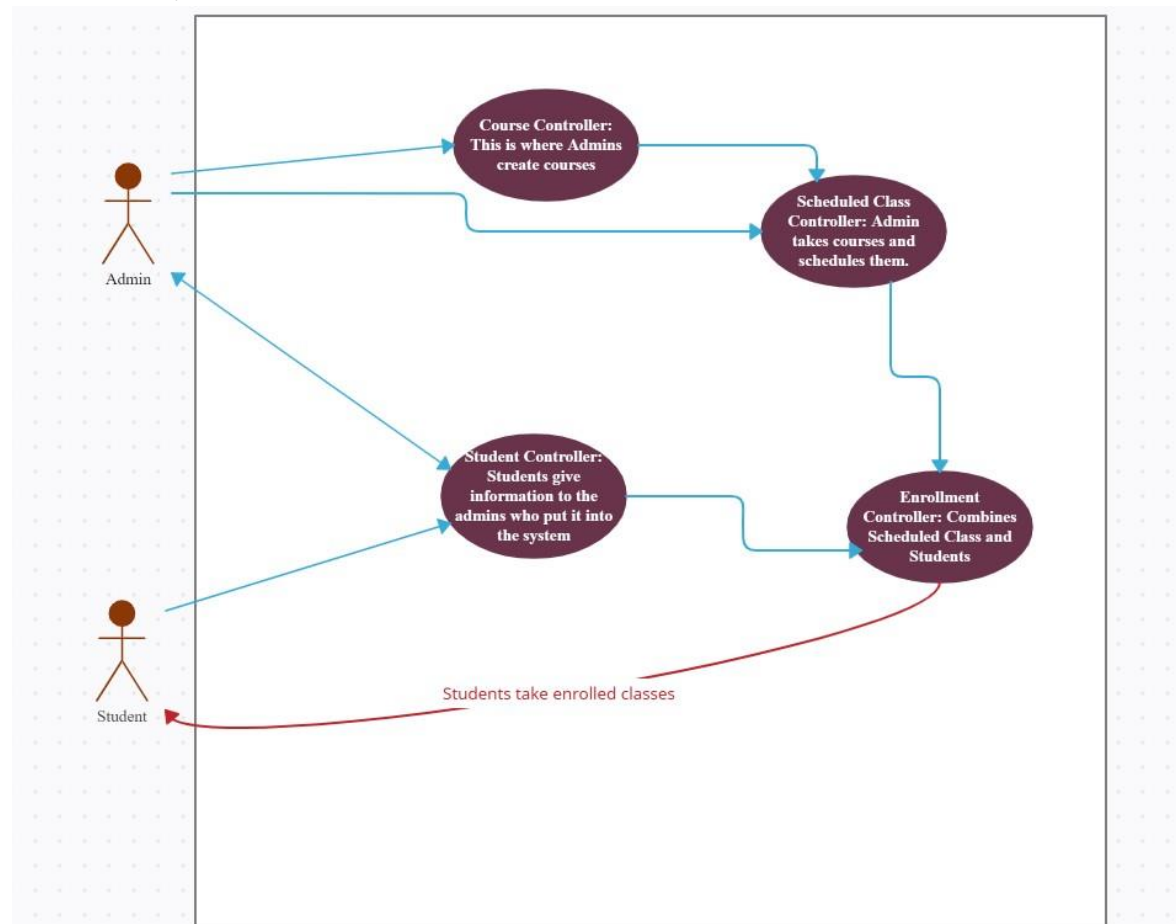
Thank for taking the time to read the documentation for Spencer and John's Scheduling Administration Tool project. This project is part of the Centriq weeklong course Project Management for the Developer. We were given a prompt late Tuesday afternoon and had to work on the project for the rest of Tuesday, most of Wednesday, most of Thursday, and Friday morning. We were not allowed to work on it outside of class, as this was an exercise designed around power programming.

This website is meant to be a tool for admins to use that shows a number of categories – students, courses, scheduled classes, and enrollments – and allows that admin full CRUD functionality. We got all of the functionality working, completed much of the projects challenges, but were then interrupted by the time requirement before we could fully flesh out the UI we envisioned (we got pretty close though).

Thank you for your time,

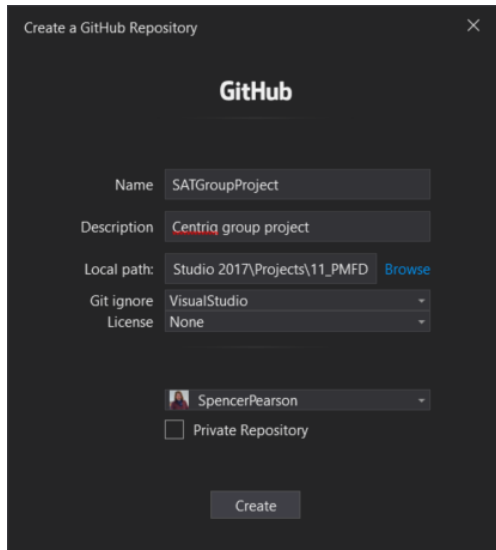
John and Spencer

Below is a visual representation of the pathing of information between the student, the website administrator, and the website.



## Source Control

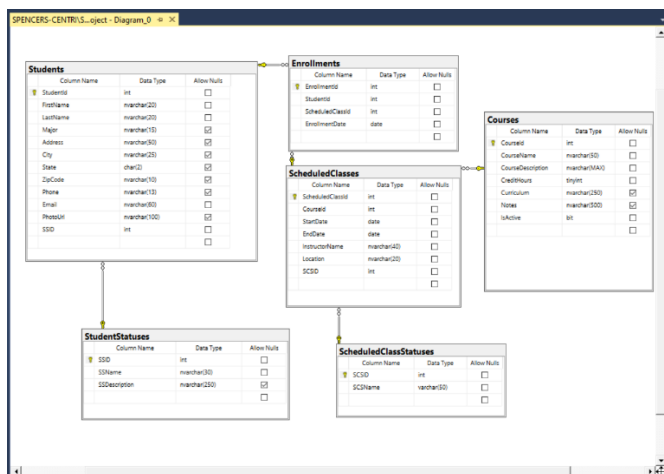
We begin with source control by creating a new public repository in Visual Studio. Once the repository is created, the creator invites all team members to the repository via GitHub. Other team members accept the invite, then clone the remote repository locally.



## Create Database

We built out our database in SQL Server Management Studio. We created a new database, then added tables, setting primary and foreign keys as necessary to build the relationships between all the tables. Once complete, we added data by selecting "edit top 200 rows" for each table and propagating several initial entries in each table. Finally, the driver created a database script for the navigator by right clicking the database in SSMS and choosing Tasks > Generate Script and following the instructions found at [https://centriq.instructure.com/courses/4543/files/255964?module\\_item\\_id=248562](https://centriq.instructure.com/courses/4543/files/255964?module_item_id=248562)

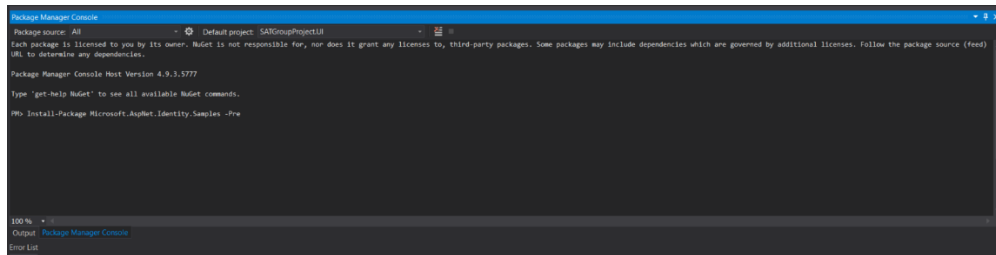
Once the navigator runs the script, an exact copy of the initial database will be created on the navigator's local machine.



Build a simple 2 tier project structure (no picture)

We set up a simple 2 tier project structure in Visual Studio, starting by adding an MVC Project in the .Net Framework with authentication for the UI layer, then adding a C# Class for our data layer.

### Run Identity Samples in the UI layer

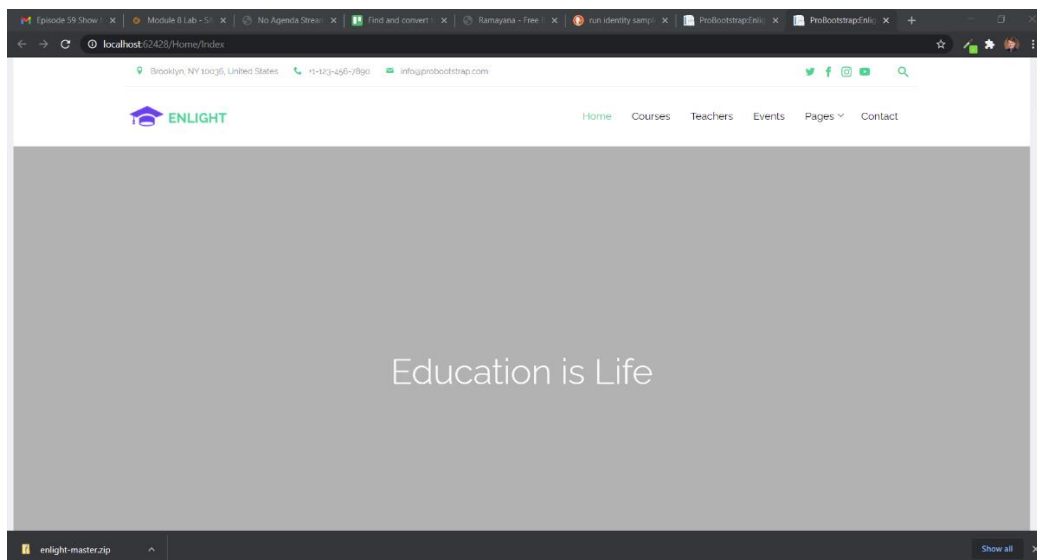


### Find and convert template

To convert a template, we start by creating an \_Archive folder to house all of the original source files that came with our template. We copy the Images, Styles, Scripts and index.html pages into the \_Archive. Then we copy the contents of the Index.html file into our Shared \_Layout view.

After adding a RenderBody() and RenderSection("scripts", required: "false") to the layout, we cut all sections specific to the Index view out of the layout and pasted them into our Home Index.cshtml file.

We dropped all the Images, Styles, Scripts and Fonts into our Contents folder, then we updated all file paths in \_Layout and Index.cshtml for our style sheets, scripts and images to reflect their new paths in the Contents folder.



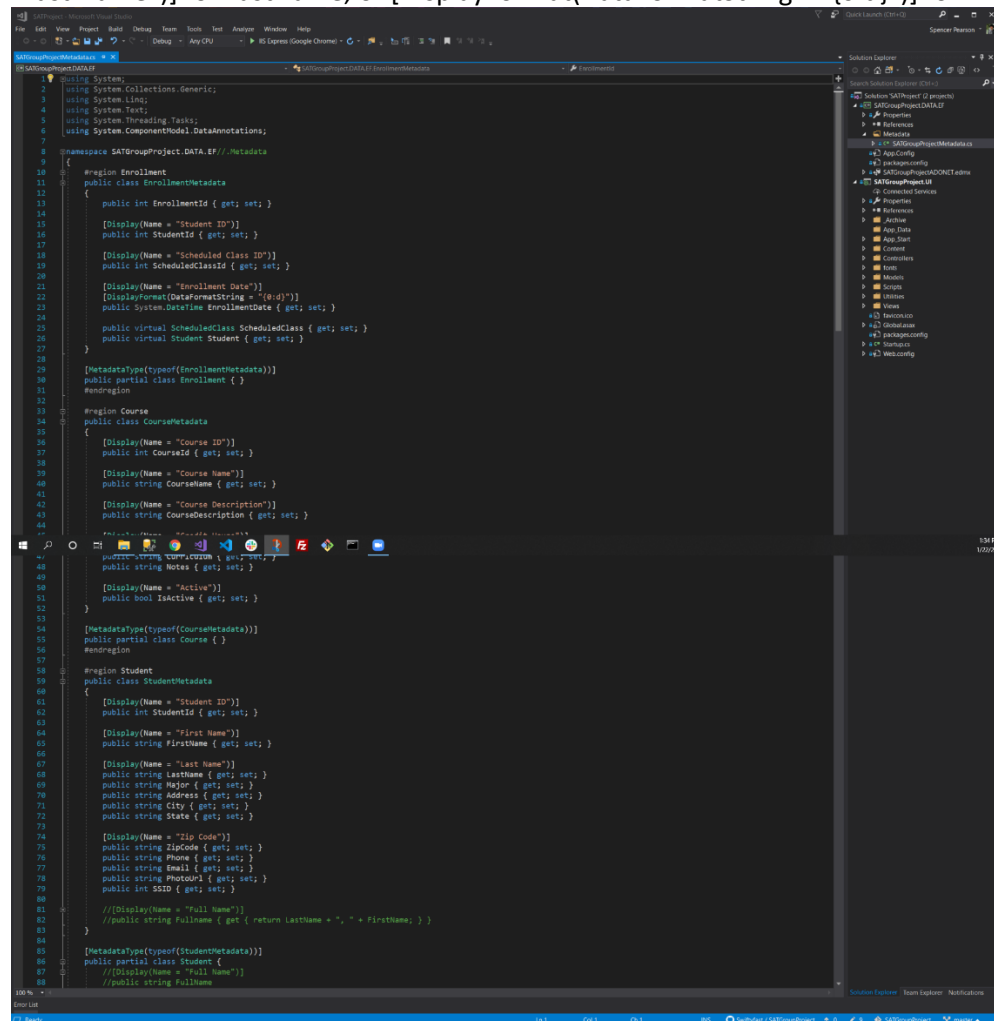
## Use Entity Framework in the Data Layer

In our Data Layer (C# class library) we add a new item and select ADO.NET Entity Data Model under C# Data templates. We choose to make it an EF Designer from Database, then on the next screen we open a new connection to our database by entering ".\sqlexpress" in the ServerName field. Our databases will now populate in the dropdown menu, so we select the SATGroupProject database. Ensuring Entity Framework 6.x is selected, we then select all the relevant tables to include (only include tables that were present in our schema when we built the original database in SSMS). All checkboxes in the lower half should be selected, then we click "Finish."

Build the project, then check the .edmx diagram to ensure all Table Names are correctly spelled. If tables need renaming, be sure to rebuild the project.

Now we can remove the default Class1.cs file and add a Metadata folder to our data layer. In that folder we will add a new class named DatabaseNameMetatata, then we'll copy all of the properties present in the .tt files into this class. We will add a new Class declaration for public partial matching each .tt file as well.

We can now add metadata annotations to properties in these classes as necessary, in order to get the display names and formatting we want the end user to see. Some examples include [Display(Name = "Last Name")] for LastName, or [DisplayFormat(DataFormatString = "{0:d}")] for EnrollmentDate.

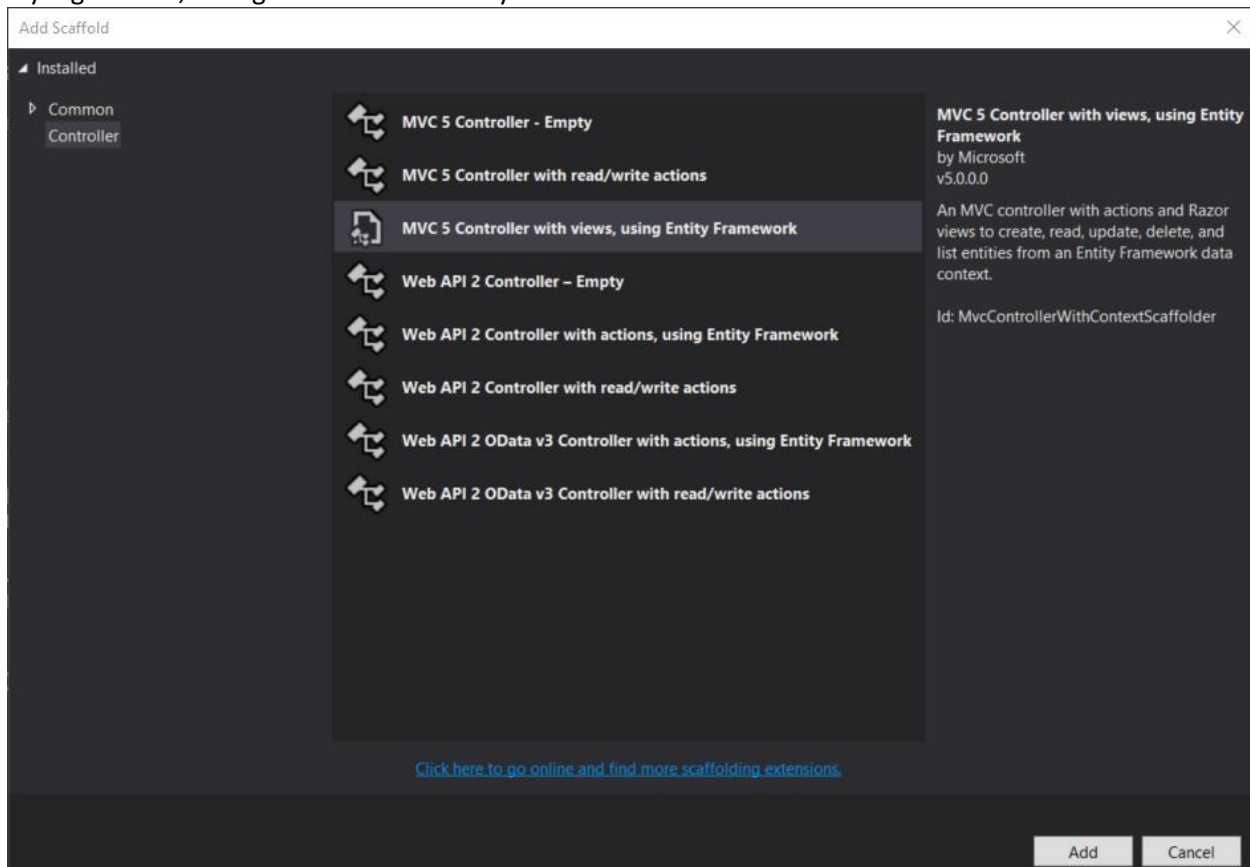


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.ComponentModel.DataAnnotations;
7
8 namespace SATGroupProject.DATA.EP.Metadata
9 {
10     #region Enrollment
11     public class EnrollmentMetadata
12     {
13         public int EnrollmentId { get; set; }
14         [Display(Name = "Student ID")]
15         public int StudentId { get; set; }
16         [Display(Name = "Scheduled Class ID")]
17         public int ScheduledClassId { get; set; }
18         [Display(Name = "Enrollment Date")]
19         [DisplayFormat(DataFormatString = "{0:d}")]
20         public System.DateTime EnrollmentDate { get; set; }
21
22         public virtual ScheduledClass ScheduledClass { get; set; }
23         public virtual Student Student { get; set; }
24     }
25
26     [MetadataType(typeof(EnrollmentMetadata))]
27     public partial class Enrollment { }
28     #endregion
29
30     #region Course
31     public class CourseMetadata
32     {
33         [Display(Name = "Course ID")]
34         public int CourseId { get; set; }
35         [Display(Name = "Course Name")]
36         public string CourseName { get; set; }
37         [Display(Name = "Course Description")]
38         public string CourseDescription { get; set; }
39     }
40
41     [MetadataType(typeof(CourseMetadata))]
42     public partial class Course { }
43     #endregion
44
45     #region Student
46     public class StudentMetadata
47     {
48         [Display(Name = "Student ID")]
49         public int StudentId { get; set; }
50         [Display(Name = "First Name")]
51         public string FirstName { get; set; }
52         [Display(Name = "Last Name")]
53         public string LastName { get; set; }
54         public string Major { get; set; }
55         public string Address { get; set; }
56         public string City { get; set; }
57         public string State { get; set; }
58
59         [Display(Name = "Zip Code")]
60         public string ZipCode { get; set; }
61         public string Phone { get; set; }
62         public string Email { get; set; }
63         public string PhotoUrl { get; set; }
64         public int SSID { get; set; }
65
66         // [Display(Name = "Full Name")]
67         // public string FullName { get { return LastName + ", " + FirstName; } }
68     }
69
70     [MetadataType(typeof(StudentMetadata))]
71     public partial class Student { }
72     // [Display(Name = "Full Name")]
73     // public string FullName
74 }
```

## Scaffold controllers and views

Next we want to build controllers and scaffold out views for all the data classes we want to work with in our application. For each class of data, we start by adding a new controller to our controllers folder and selecting the "MVC5 Controller with views, using Entity Framework" template. Select the relevant Model Class (be sure you aren't selecting the metadata version), and choose the Entities option for the DbContext input. All of the bottom options should be checked, then we can name the controller ClassController and select add.

Be sure to add links in your \_Layout navigation menu to the index view of each controller that you build. Then we go through each view (Index, Details, Create, Edit, Delete) and polish up the look and the styling of each, hiding certain unnecessary data from certain views.



## Lock down the controllers

An important step for securing our application is locking down actions in the controllers that we do not want unauthorized users to gain access to. This simple but important step usually entails adding an `Authorize` annotation to the get and post actions for CRUD functionality, so that only approved roles can create, edit and delete items in the database.

Simply add `[Authorize(Roles = "Admin")]` above any `ActionResult` you want to limit access to. Typically we will leave the Index action and the Details action open, but lock down the Get and Post actions for Create, Edit and Delete.

```
43     return View(student);
44 }
45
46 // GET: Students/Create
47 [Authorize(Roles = "Admin")]
48 public ActionResult Create()
49 {
50     ViewBag.SSID = new SelectList(db.StudentStatuses, "SSID", "SSName");
51     return View();
52 }
53
54 // POST: Students/Create
55 // To protect from overposting attacks, please enable the specific properties you want to bind to, for
56 // more details see https://go.microsoft.com/fwlink/?linkid=317598.
57 [HttpPost]
58 [ValidateAntiForgeryToken]
59 [Authorize(Roles = "Admin")]
60 public ActionResult Create([Bind(Include = "StudentId,FirstName,LastName,Major,Address,City,State,ZipCode,Phone,Email,PhotoUrl,SSID")] Student student,
61     HttpPostedFileBase studentPhoto)
62 {
63     if (ModelState.IsValid)
64     {
65         File Upload
66         db.Students.Add(student);
67         db.SaveChanges();
68         return RedirectToAction("Index");
69     }
70
71     ViewBag.SSID = new SelectList(db.StudentStatuses, "SSID", "SSName", student.SSID);
72     return View(student);
73 }
74
75 // GET: Students/Edit/5
76 [Authorize(Roles = "Admin")]
77 public ActionResult Edit(int? id)
78 {
79     if (id == null)
80     {
81         return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
82     }
83     Student student = db.Students.Find(id);
84     if (student == null)
85     {
86         return HttpNotFound();
87     }
88     return View(student);
89 }
90
91 // GET: Students/Details/5
92 public ActionResult Details(int? id)
93 {
94     if (id == null)
95     {
96         return HttpNotFound();
97     }
98     Student student = db.Students.Find(id);
99     if (student == null)
100    {
101        return HttpNotFound();
102    }
103    return View(student);
104 }
```

## Lock down the views

We created if statements using Razor to prevent users who were not logged in as an admin from seeing functionality and being able to use CRUD.

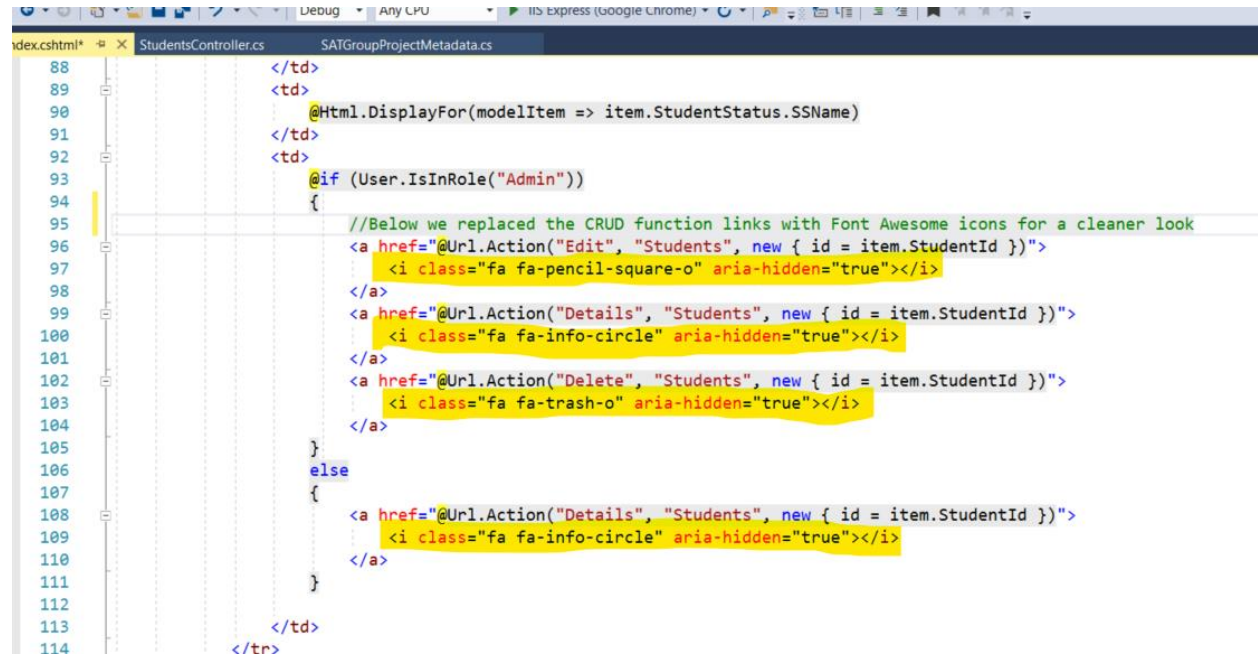
```
1  @model IEnumerable<SATGroupProject.DATA.EF.Student>
2
3  @{
4      ViewBag.Title = "Students";
5  }
6
7  <div class="container">
8      <h2>@ViewBag.Title</h2>
9      @if (User.IsInRole("Admin"))
10     {
11         <div class="row mb-1">
12             <div class="col-md-2 text-left">
13                 @Html.ActionLink("Create New", "Create", null, new { @class = "btn btn-primary" })
14             </div>
15         </div>
16     }
17
18     <table class="table">
19         <tr>
20             <th>
21                 @Html.DisplayNameFor(model => model.FirstName)
22             </th>
23             <th>
```



## Customize views in UI Layer

We customized the views to match the rest of the template. We did this by making use of the Bootstrap btn-primary class for as much as we could (the primary color had been changed to a teal), and using the btn-info class because it matched the teal from the btn-primary.

We also used favicons on several pages to be placeholders instead of the Details, Edit, and Delete keywords.



```
88         </td>
89         <td>
90             @Html.DisplayFor(modelItem => item.StudentStatus.SSName)
91         </td>
92         <td>
93             @if (User.IsInRole("Admin"))
94             {
95                 //Below we replaced the CRUD function links with Font Awesome icons for a cleaner look
96                 <a href="@Url.Action("Edit", "Students", new { id = item.StudentId })">
97                     <i class="fa fa-pencil-square-o" aria-hidden="true"></i>
98                 </a>
99                 <a href="@Url.Action("Details", "Students", new { id = item.StudentId })">
100                     <i class="fa fa-info-circle" aria-hidden="true"></i>
101                 </a>
102                 <a href="@Url.Action("Delete", "Students", new { id = item.StudentId })">
103                     <i class="fa fa-trash-o" aria-hidden="true"></i>
104                 </a>
105             }
106             else
107             {
108                 <a href="@Url.Action("Details", "Students", new { id = item.StudentId })">
109                     <i class="fa fa-info-circle" aria-hidden="true"></i>
110                 </a>
111             }
112         </td>
113     </tr>
114 }
```

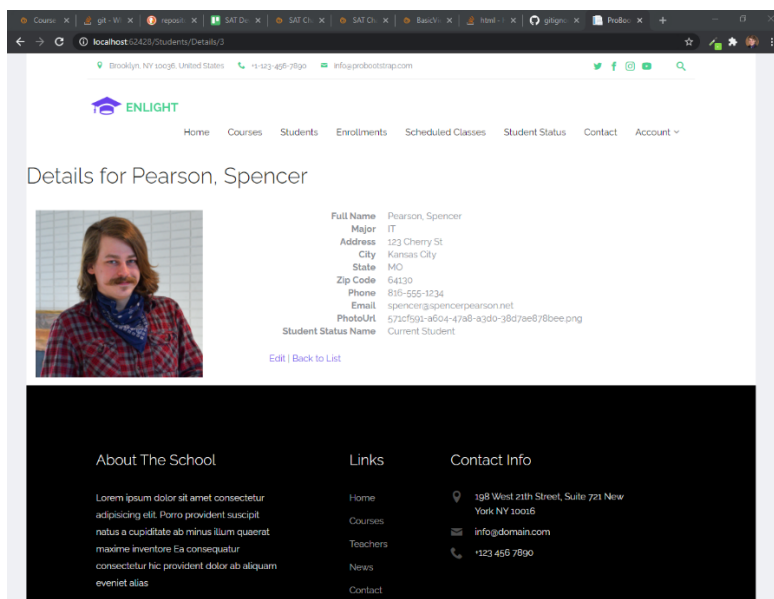
## Custom Properties

We built two custom properties for this project, a Student Fullname and a Scheduled Class Information prop that encompassed location, start date, and course name. We utilized them in any view that made sense (Index, details, and delete views for the student for Fullname and the scheduledclass was utilized in its own views and also the enrollment views.)

```
//Below we have a custom property to return Students' full name
[MetadataType(typeof(StudentMetadata))]
public partial class Student {
    //[Display(Name = "Full Name")]
    //public string FullName
    //{
    //    get
    //    {
    //        return (FirstName + " " + LastName);
    //    }
    //}
    [Display(Name = "Student")]
    public string Fullname { get { return LastName + ", " + FirstName; } }
}
#endregion
```

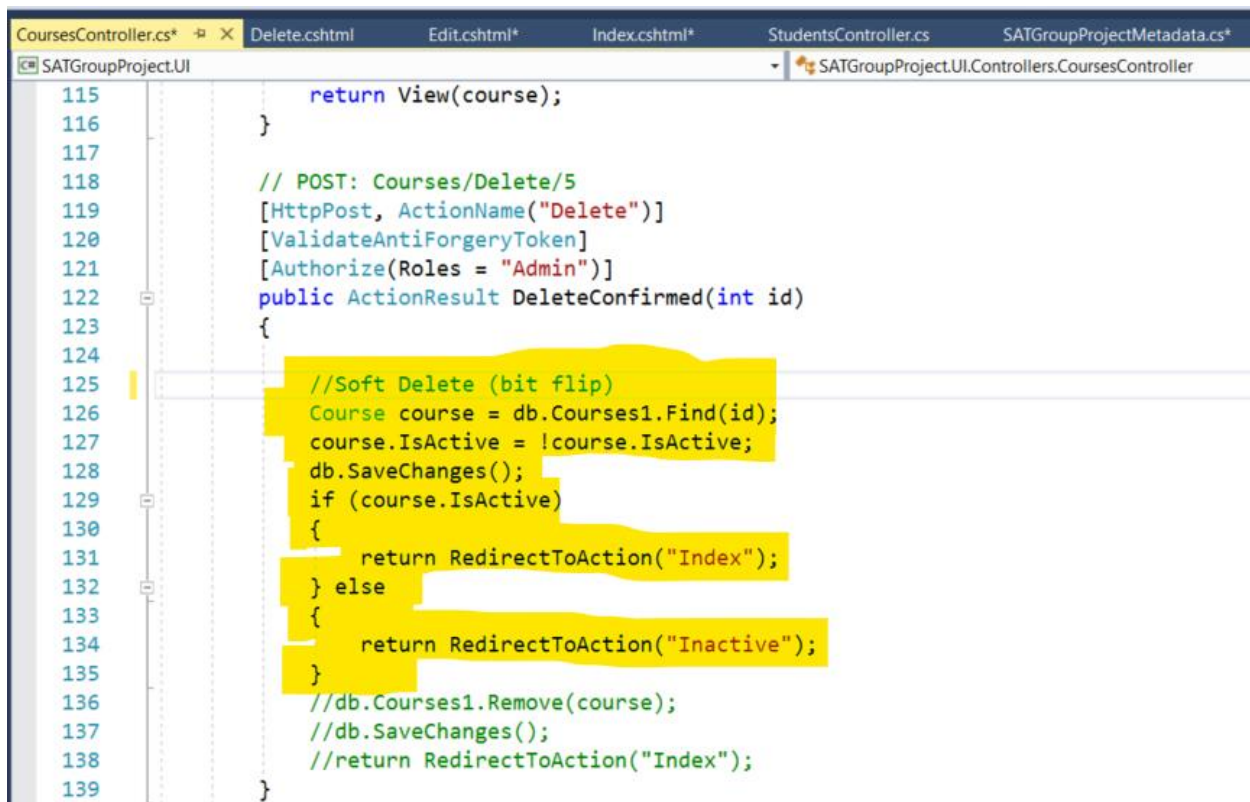
## File Upload

In the edit view of the student controller, an admin can upload a photo of a student to the database. The method will automatically create a new Guid that provides a unique file path. The method will also check different properties of the image (correct datatype, size isn't too big) before saving it.



## Soft Delete functionality

In several of the controllers, changed the delete functionality into a soft delete or 'bit flip'. This entails changing the status of an item (in this case the IsActive part of Courses) and then using the RedirectToAction method to take the user to either a list of all active courses or a list of inactive courses.



```
115         return View(course);
116     }
117
118     // POST: Courses/Delete/5
119     [HttpPost, ActionName("Delete")]
120     [ValidateAntiForgeryToken]
121     [Authorize(Roles = "Admin")]
122     public ActionResult DeleteConfirmed(int id)
123     {
124
125         //Soft Delete (bit flip)
126         Course course = db.Courses1.Find(id);
127         course.IsActive = !course.IsActive;
128         db.SaveChanges();
129         if (course.IsActive)
130         {
131             return RedirectToAction("Index");
132         } else
133         {
134             return RedirectToAction("Inactive");
135         }
136         //db.Courses1.Remove(course);
137         //db.SaveChanges();
138         //return RedirectToAction("Index");
139     }
```