Documentation

This application was designed for Sith University administrators and staff control over:

- Student scheduling
- Course list
- Course information
- The students taking the courses
- Student statuses (Prospect Current Student, Former Student Withdrawn, Former Student Dismissed, Alumni, or Booted)
- Class statuses (Active, Pending, Completed, or Cancelled)

Admin Functionality:

The roles that can use this web application are Admin, Scheduler, and User(Anonymous or logged in, no privileges.) The Admin gets the following functionality:

- Home: Can view Index and Contact
- Courses: Can view active and retired classes and details of the classes. They can also edit, delete, and create the classes. This will reflect in the Database it is connected to and update the current data models.
- Enrollments: Can view Index and Details of each enrollment. They can also edit, delete, and create enrollments.
- Scheduled classes: Can view Index and Details of each class status. They can also edit, delete, and create class statuses.
- Students: Can view the Index of students and their details. They can also edit, delete, and create students.

Documentation

- Student Statuses: can view Index and Details of the student statuses.

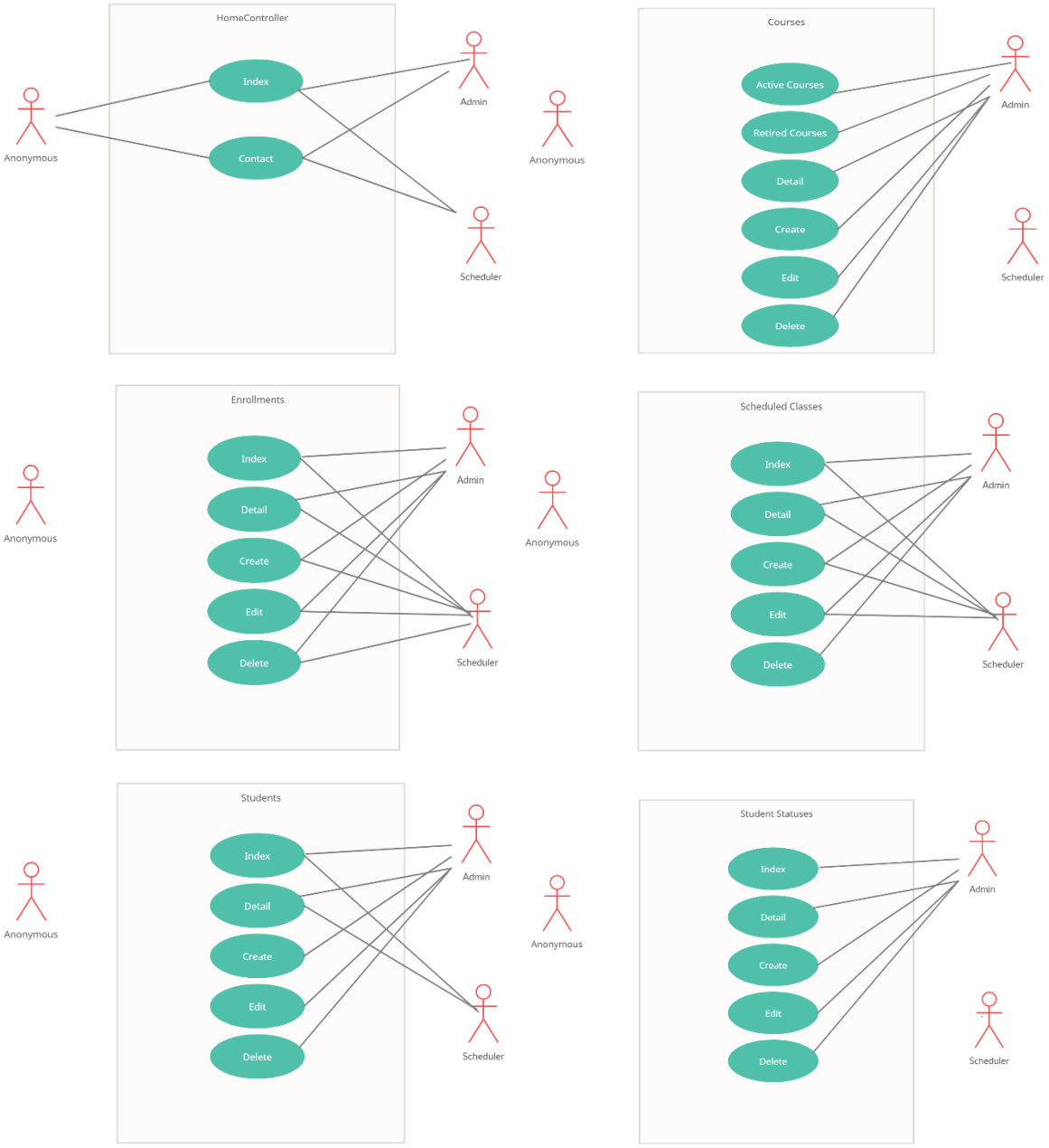
Scheduler Functionality:

- Home: can view Index and Contact.
- Courses: Can only access the tile view of courses. (Index) Has no ability to edit, delete, or create courses.
- Enrollments: Can view Index and Details of each enrollment. They can also edit, create, and delete each enrollment.
- Scheduled Classes: Can view Index and Details of each scheduled class. They can also edit and create scheduled class. They cannot delete a scheduled class.
- Students: They can only view Index and Details of each student. Has no ability to edit, create, or delete student records.
- Student Statuses: Does not have access to Index or Details. Has no ability to edit, create, or delete a status for a student record.
-

Functionality of users and anonymous:

- Home: Can view Index and Contact.
- Courses: Can only view tile view of Index. Does not have any ability to create, edit, or delete records.
- Enrollments: No access.
- Scheduled Classes: No access.
- Students: No access.
- Student Statuses: No access.

Documentation



Other Web App Functionality and Technicalities:

- Tile / card view for students
- Image Upload for students
- Custom Properties for students

Documentation

- Details of the application, standards it has to meet, what technologies were used, how to test the application.

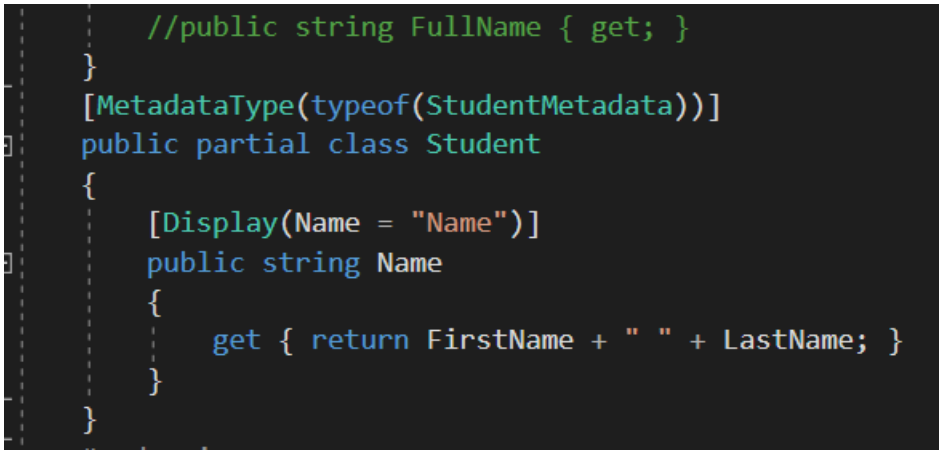
Technologies used:

- ASP.NET MVC
- ADO.NET Entity Framework
- Identity Samples
- MySQL Server
- HTML, CSS, JavaScript, jQuery
- Trello
- Creatively

Custom Properties for Student

Through the normalization process of building our database, we decided to store a student's first name and last name as two separate fields. This has caused the process of enrolling a student in a scheduled class slightly more difficult as when we enroll a student, we choose from a dropdown of first names. Since we don't store calculated fields in a database, we can create a custom property in our partial (buddy) class to represent the full name of a student instead of just their first name and apply metadata to the property.

1. First we built a custom property that combines the student's first name and last name. We did this is the partial buddy class.



```
    //public string FullName { get; }
}
[MetadataType(typeof(StudentMetadata))]
public partial class Student
{
    [Display(Name = "Name")]
    public string Name
    {
        get { return FirstName + " " + LastName; }
    }
}
```

Documentation

2. We then utilized this in the student index, details, and delete views.

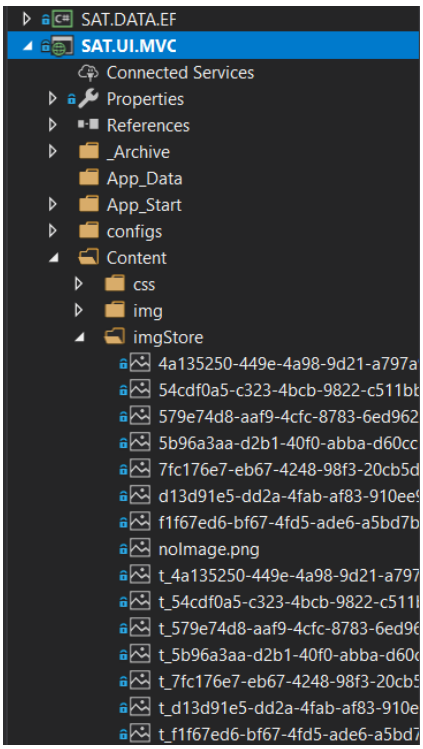
```
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<br/>
<p>
    <a href="@Url.Action("StudentGridView","Students")">Tile View</a>
</p>
<table class="table">
    <tr>
        <th>
            @*@Html.DisplayNameFor(model => model.PhotoUrl)*@
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        @*<th>
            @Html.DisplayNameFor(model => model.LastName)
        </th>*@
        <th>
```

Documentation

Image Upload

For the SAT application, we need a way to update existing student photos. We will use our PhotoUrl field to track the name of the image as we do not store images in a database (size). For file uploads, it is important to think of the steps as a user – update the view to allow for an image to be uploaded and then process the image in the Controller. When we process the image, we need to check that a file was uploaded, get the name and extension of the file, validate the file extension to prevent "bad" extensions, save the file to the server and associate the file to the object being updated.

1. We had to add a StudentImages folder to contain student folders.



2. We found a default picture display for when photo is null, that simply shows "No Image". This indicates no record is found and is a way to freshen the feel of the page. This is found in the imgStore folder of the project.

Documentation

3. We then needed to implement the functionality in the controller of the student. We updated the functions on the POST: create, edit, delete.

   i.e POST CREATE (Student Controller):

```
// POST: Students/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include =
 "StudentId,FirstName,LastName,Major,Address,City,State,ZipCode,Phone,Email,PhotoUrl,SSID")] Student student,
 HttpPostedFileBase StudentPhoto)
{
    string file = "noImage.png";
    if (StudentPhoto != null)
    {
        file = StudentPhoto.FileName;

        string ext = file.Substring(file.LastIndexOf("."));
        string[] goodExts = { ".jpg", ".jpeg", ".png", ".gif" };

        if (goodExts.Contains(ext.ToLower()) && StudentPhoto.ContentLength <= 4194304)
        {
            //Create a new file name (use a GUID)
            file = Guid.NewGuid() + ext;



            #region Resize Image
            //this informs the program to save the image to this location in our file structure.
            string savePath = Server.MapPath("~/Content/imgStore/");
            Image convertedImage = Image.FromStream(StudentPhoto.InputStream);
            int maxImageSize = 500;
            int maxThumbSize = 100;
```

4. Create input on the create and edit view so that the user can search for a file on their computer.

   i.e CREATE VIEW (Student):

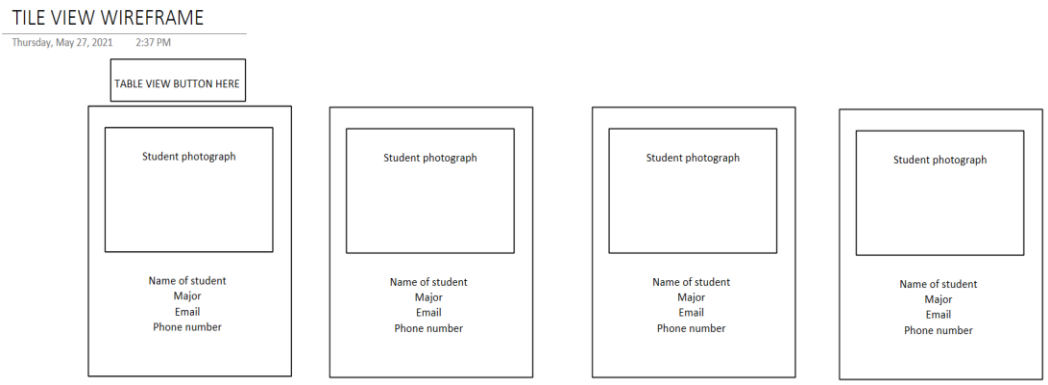
```
<div class="form-group">
    @Html.LabelFor(model => model.PhotoUrl, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        <input type="file" name="StudentPhoto" style="display: inline" />
        @Html.ValidationMessageFor(model => model.PhotoUrl, "", new { @class = "text-danger" })
    </div>
</div>
```

Documentation

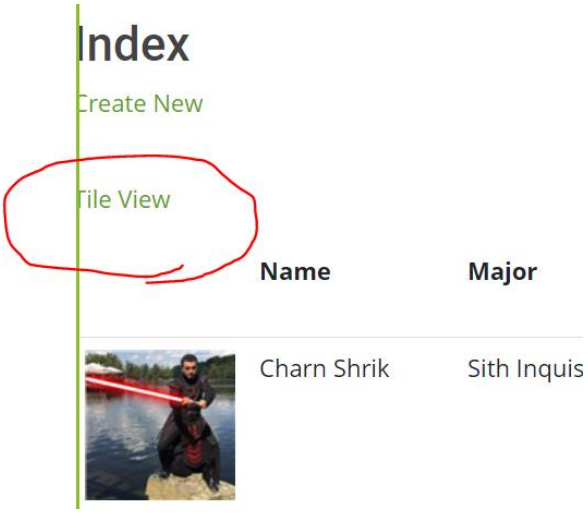
Tile View / Card View for Students

When building an application, a user may request to have the option to view data in multiple ways. We can accommodate the user's request by adding custom styling. We are going to build a "tiled" layout for Student Statuses and allow the user to toggle between a table layout and the tiled layout. First, we need to determine what the tiled view will look like, so let's create a wireframe.

1. We created a wire frame for our student cards:



2. We updated the index view in Students to have a link to Index Tile view. Only Admins can see this.



3. We then made an HTML structure that would house the card for each student. We edited the automatically scaffolded MVC list for the student objects to display what we wanted on each card.

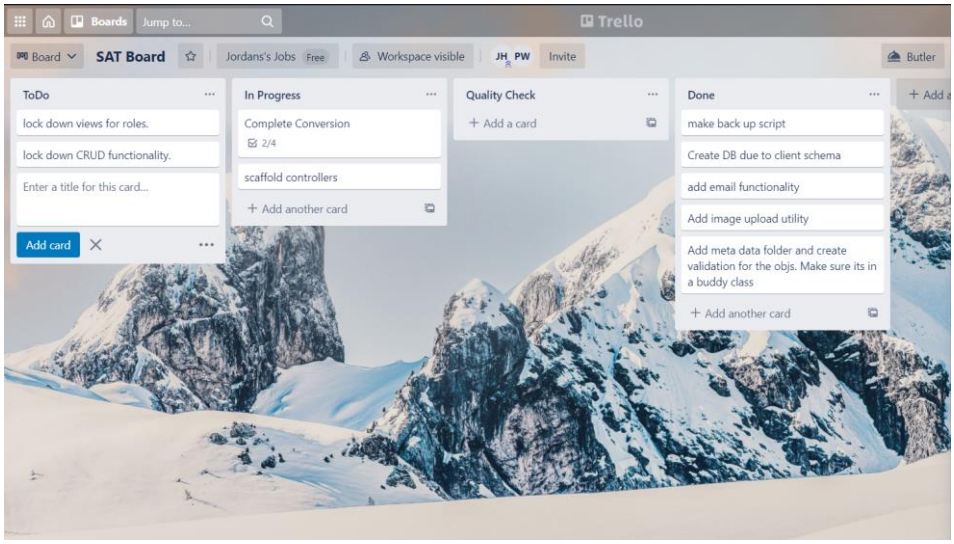
Documentation

```
@foreach (var item in Model)
{

    <div class="card col-md-3" style="width: 18rem;">
        <div class="other-border">
            <a href="@Url.Action("Details","Students", new { id=item.StudentId})"> <img src="~/Content/imgStore/@item.PhotoUrl" alt="Student" title="@item.FirstName"
                style="height:300px; width:100%" /></a>
            <div class="card-body">
                <h5 class="card-title">@item.FirstName @item.LastName</h5>
                <hr />
                <p class="card-text">@item.Major</p>
                <p class="card-info">@item.Email </p>
                <p class="card-info">@item.Phone</p>


            </div>
        </div>
    </div>

}
div>
```

4. We then linked an action to this view in the Student Controller.

```
public ActionResult StudentGridView()
{
    List<Student> students = db.Students.ToList();
    return View(students);
}
```

Agile / Scrum Documentation (TRELLO Board)



Trello is a useful tool to organize while in paired programming. This was utilized throughout the build of this first sprint of the web app. Team meetings will continue to plan for the second sprint of this application. Stay tuned for version 2!