**Learning Management System**

1. **Group Members:**

Nicolas Gamez - Chief Programmer

Hoa Nguyen - Testing/Debugging

Sang Tran - GUI Programmer

Caitlin Boake - Data storage/Backup Programmer

Christian Tolentino - Programming Secretary

1. **Life-cycle model: Rapid Prototyping**

Programming Language: ASP.NET with Entity Framework - C#, HTML, and CSS. Razor

IDE: Visual Studios 2017

Data storage and retrieval: Microsoft Azure, Azure SQL Database, t-SQL

1. **GitHub Repo**

GitHub: <https://github.com/JustCode1995/LMSProject>

**Introduction:**

This document serves as the project plan for the Pyramid SchemA+ Learning Management System (PSLMS) software development effort. The content of the document is divided into 6 sections: Objective, Requirements, Analysis - User Characteristics, Team Model, Design, Implementation, and Testing.

**Objective:**

Pyramid SchemA+ Learning Management System is a software program that helps users create, manage, and deliver courses. This software product consists of two separate parts: a server component and a user interface. The server component performs the core functionality, such as creating, managing, and delivering courses, authenticating users, serving data and notifications, etc. The user interface runs inside a browser as a web, which is used by administrators, professors, and students. The system allows users to create content (student information, class section details, etc.), organize it into courses, enroll students to said courses, and finally, monitor and assess the students’ performances (assignments, grades, etc.). Administrators are setting up and configuring the LMS, such as adding, editing, deleting, and monitoring student accounts, enrollment, courses, and staff members. Professors are preparing the lessons and accessing the students’ progress. Students are viewing the classes their enrolled in and their final GPA.

**Requirements:**

During this phase, we communicated to determine the goals and expectations for this project as well as present and possible problems that would need to be acknowledged during the build. The requirements are the goals that Pyramid Schema+ is to accomplish. The functional requirements for the PSLMS web application are as follows:

* The system should provide a GUI into a DBMS for PSLMS users. The database engine will provide the standard features expected of a DBMS, including data transfer, storage, modifications, relationship management, and access control. End users will connect to the DBMS via a web browser, as appropriate to the project.
* The system should restrict access to Pyramid SchemA+ data to access levels of authorized users
* The system should provide the ability to enter and update users (student, admin, and professor) information, including names, login credentials, access levels, etc.
* The system should provide the ability to enter, update, and submit assignments and grades

The nonfunctional requirements are constraints on the web app or functions offered by the web app. These are the requirements that are concerned with the functions delivered by the applications:

* The system should work on any operating system.
* The system should work on any web browser.
* The system’s UI should be attractive and user-friendly
* The system should be capable for queries and display the queries
* The system should be easy to use.

**Analysis - User Characteristics:**

This system will perform different functions depending on the type of user: student or staff. The Staff is divided to Professor and Admin, identified by their access level. The access level for Admin is 1, and the access level for Professor is 2. The administrators will have the control of the system, thus they will have administrator rights.

The features available to Admin:

* Add New Student
* Create Login
* Manage Existing Students
* Manage Existing Logins
* Enroll Into Class
* Manage Existing Enrollments
* New Staff
* Create Login
* View All Staff
* Manage Login

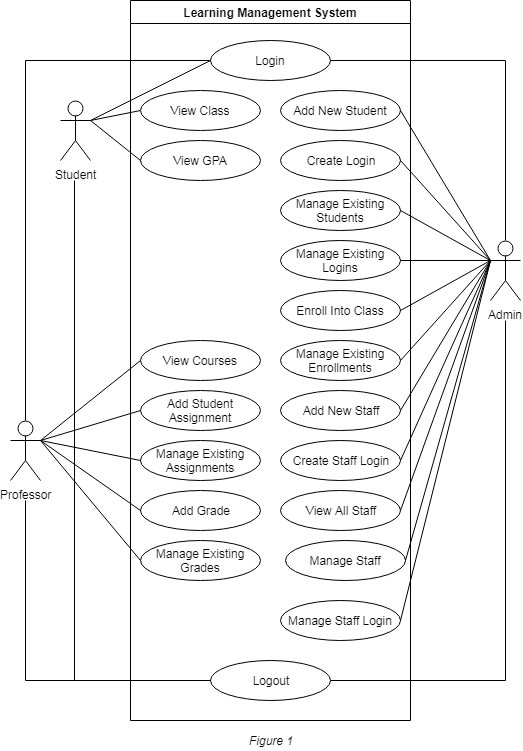
The features available to Professor:

* View Courses
* Add Student Assignment
* Manage Existing Assignments
* Add Grade
* Manage Existing Grades

The features available to Student:

* View Class
* View GPA

These available features are visually represented in the following UML Use Case diagram shown in the following photo.



All users can log in and log out. After each user logs in, the next page displays their list of available features as buttons.

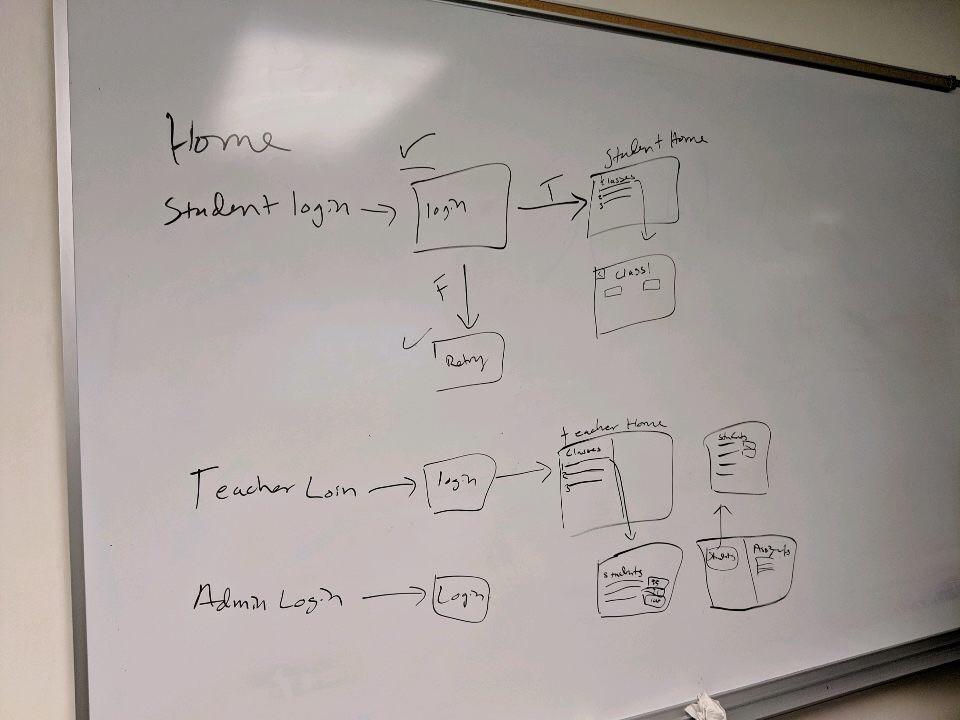
**Team Model:**

Pyramid SchemA+ was developed by using a classical chief programmer team model. This team has one chief programmer who designs the architecture of the software, distributes and assigns the coding workload to the team members, and handles any interfacing issues that may arise. The team is led by Nick, who does the majority of the programming. He is in charge of the architectural design and the coding of the critical sections of Pyramid SchemA+. When Pyramid SchemA+ has a bug or is down, it is Nick’s responsibility to handle that interfacing issue. Nick is also in charge of reviewing other team members’ work and make sure that everything is up-to-date. The backup programmer knows as much as the Chief Programmer. Caitlin is the Backup Programmer, so she assists the regular programmers by answering their questions or helping with their code. She is also in charge of designing the database. The programmers are Sang and Hoa, and their main focus was coding what had been assigned by Nick. For example, Nick assigned Sang to change the links to buttons and assigned Hoa to test and debug the program. Cristhian is the Programming Secretary, who is responsible for documenting and creating the UML diagrams.

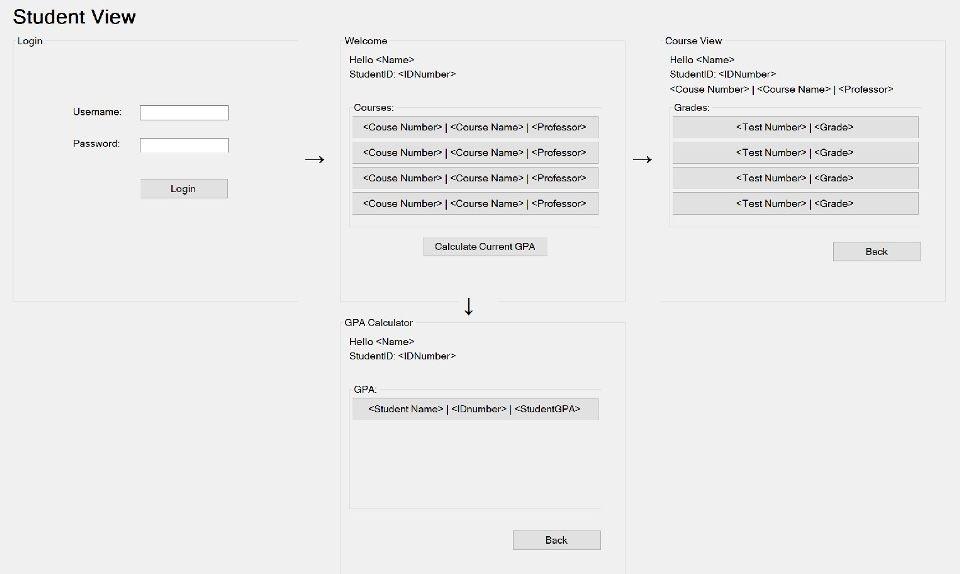
**Design**:

The software life cycle for Pyramid SchemA+ is Rapid Prototyping since we changed our platforms from Visual Basic to ASP.NET MVC because we wanted to create a web interface versus a software that would have to be downloaded on each computer. The MVC pattern separates the elements of the app (the business logic, UI logic, and input logic), while loosely coupling between these three different aspects. The business logic is located in the model; the UI logic is located in the views; and the input logic is located in the controllers.

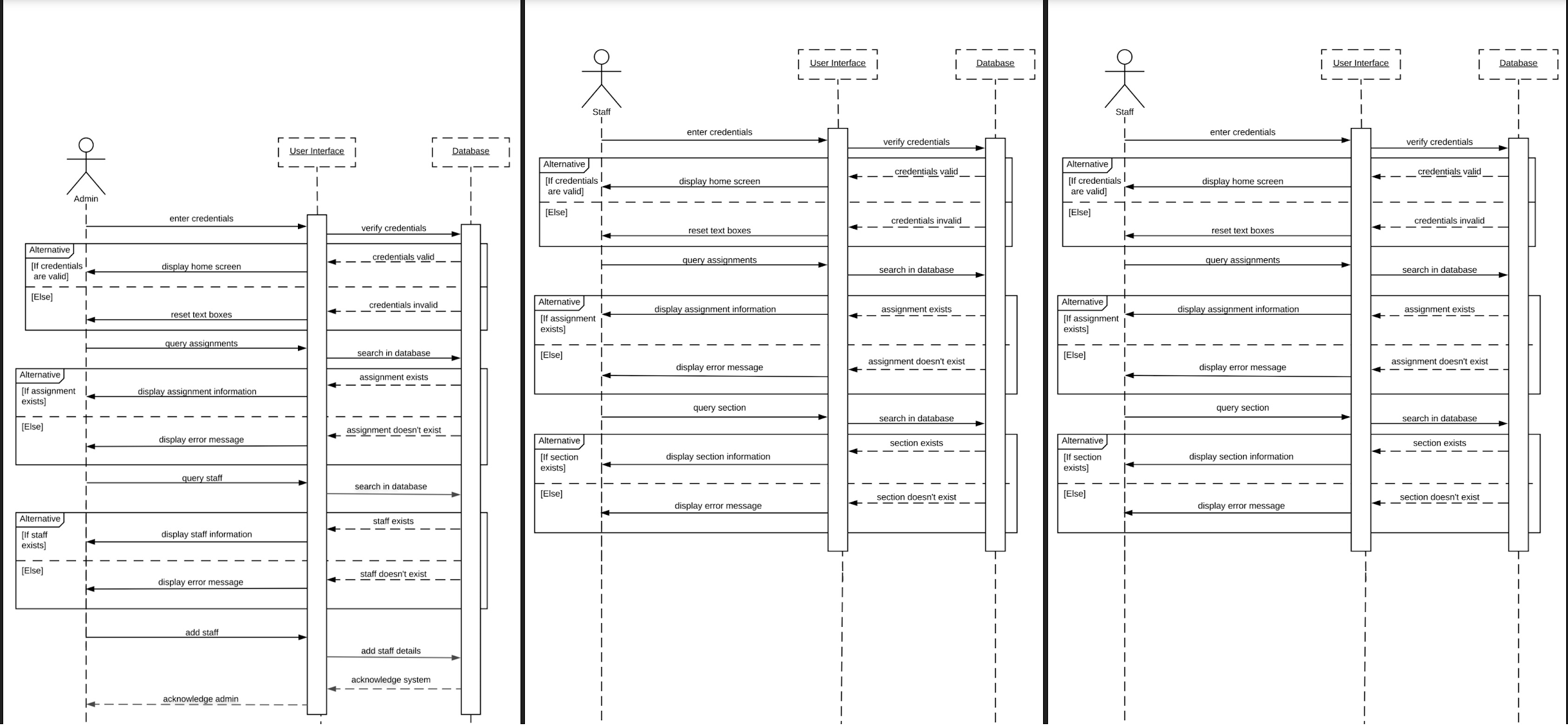
As a team, we agreed to have weekly team meetings in the library. We all committed to be present in these meetings, whether it was in-person or video chat via Zoom. In our first meeting, we listed out the software requirements and drew up how the Home page/Login screen is supposed to look like, as shown in the following photo.



As a prototype of the webpages, Nick used Visual Basic just for us to get a visual idea, as shown in the photo below.



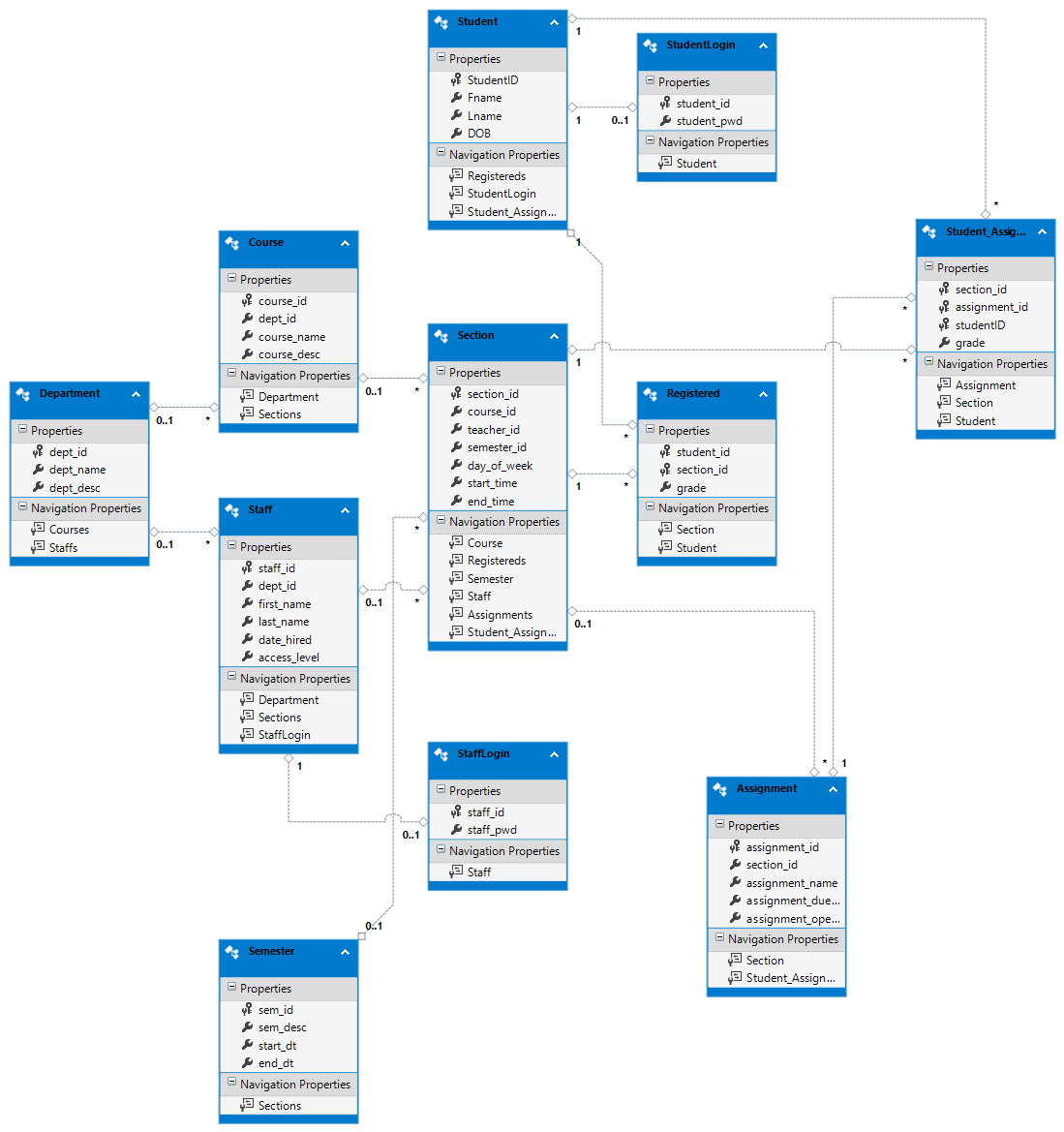
In the above image, the arrows indicated the transitions from one page to the next page.



The above sequence diagrams for each user are visual summaries of their individual use cases. These diagrams show the sequences of the login use case for all users. The sequence is as follows: User enters credentials in the user interface; the UI verifies the credentials in the database. The database sends back a message whether the credentials are valid or not. If they are, then the UI displays the next view per user. If not, then the UI displays an error message and resets the textboxes. Similar sequences for query assignments for Student and Professor. The user queries assignments on the UI, which then searches in the database, then the database invokes a message whether or not the assignment exists to the UI, which in turn sends that to the student or professor.

**Implementation**

Pyramid SchemA+ was built on an ASP.NET MVC framework with model, controllers, and views, which is a web development model that utilizes elements from C#, HTML, and CSS. The model for our program is an Entity-Relationship Diagram, as shown in the figure below.



In this project, our web app retrieves data from a database on Azure, provides it to the view or updates it. Updated data is written to the database. Views are the components that display the web’s user interface. Generally, this UI displays the model data. The controllers handle the browser requests. They retrieve data and call view templates that return a response. In our MVC web app, the view only displays information; the controller handles and responds to user input and interaction. The controller handles route data and query-string values, and it passes these values to the view. The controller uses these values to query the database. For example, [http://localhost:####/Students/Edit/9](about:blank) is a request to edit the student with ID = 9, from the Students table in our database, using the Students controller.

**Testing:**

During this phase, the programmers coded and tested each code artifact separately, linked together all the code artifacts, and tested the product as a whole. Every programmer is personally responsible for making sure that his/her work is correct before committing it to GitHub. The available features outlined in the Analysis – User Characteristics section are used to create test cases:

* Add a student
* Add a staff
* Modify a student
* Modify a staff
* Delete a student
* Delete a staff
* Compute GPA

In addition to these direct tests, it is necessary to perform the following additional tests:

* Attempt to add a student that is already on file.
* Attempt to add a staff that is already on file.
* Attempt to delete a student that is not on file.
* Attempt to delete a staff that is not on file.
* Attempt to modify a student that is not on file.
* Attempt to modify an assignment that is not on file.
* Attempt to update each field of a staff twice and check that the second version is stored.