1. **Group Members:**

Alex Luu (solo) – Chief Programmer, Testing/Debugging, GUI programmer, and Data Storage Programmer.

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

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

IDE: Visual Studios 2019

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

1. GitHub Repo

GitHub:

# Introduction

This document serves as the project plan for the Alex’s Best Learning Management System (ABLMS) 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

Alex’s Best 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 (UI). 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, enrolment, 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, I had to determine the expectations and goals for this project since I had to work by myself and I did not have any peers to help me. The functional requirements specifies the goals that Alex’s Best LMS is to perform. These are the functional requirements for the ABLMS web application:

* The software product should provide a GUI into a DBMS for the users. The database should provide the standard features expected of a DBMS, including data transfer, storage, modifications, management of entity relationships, and access control. The users will connect to the DBMS via a web browser, as appropriate to the project.
* The software product should restrict access to ABLMS data to allow only authorized users
* The software product should provide the ability to enter and update users (student, admin, and professor) information, including names, login credentials, access levels, etc.
* The software product should provide the ability to enter, update, and submit assignments and grades
* The non-functional requirements specifies constraints on the web app or functions offered by the web app. These are the non-functional requirements that are concerned with the constraints or the functions delivered by the application:
* The software product should run on any operating system.
* The software product should work on any web browser.
* The software product’s GUI should be attractive and user-friendly.
* The software product should be capable for queries and display the queries.

# Analysis - User Characteristics:

Alex’s Best LMS 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 administrative rights.

The features available to Admin:

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

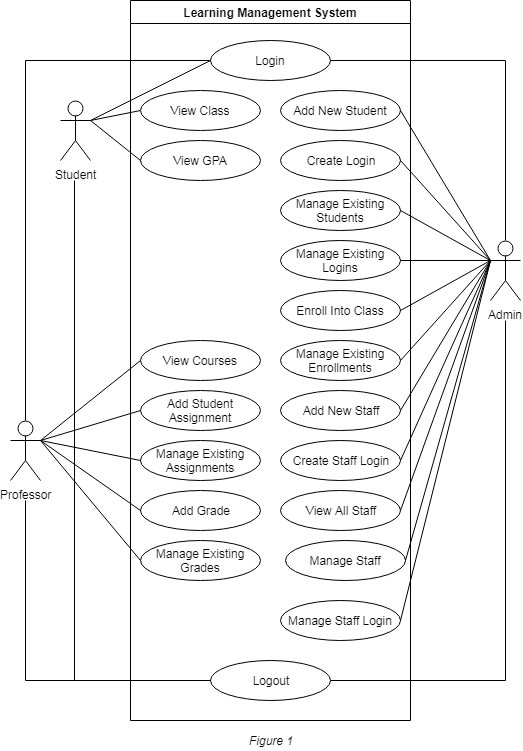
The features available to Professor:

* 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.*

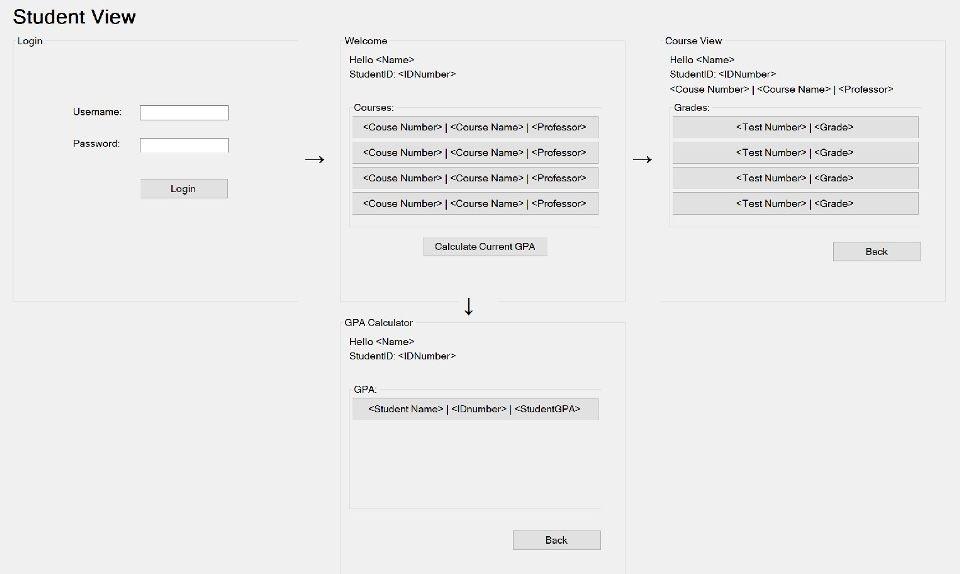
# Team Model

ABLMS was developed by using a solo team model where I had to assume multiple roles due to the extreme circumstances with no chance to meet up with my former group mates. As the chief programmer I had the opportunity to design the architecture of the software and self-manage any interfacing issues that may arise.

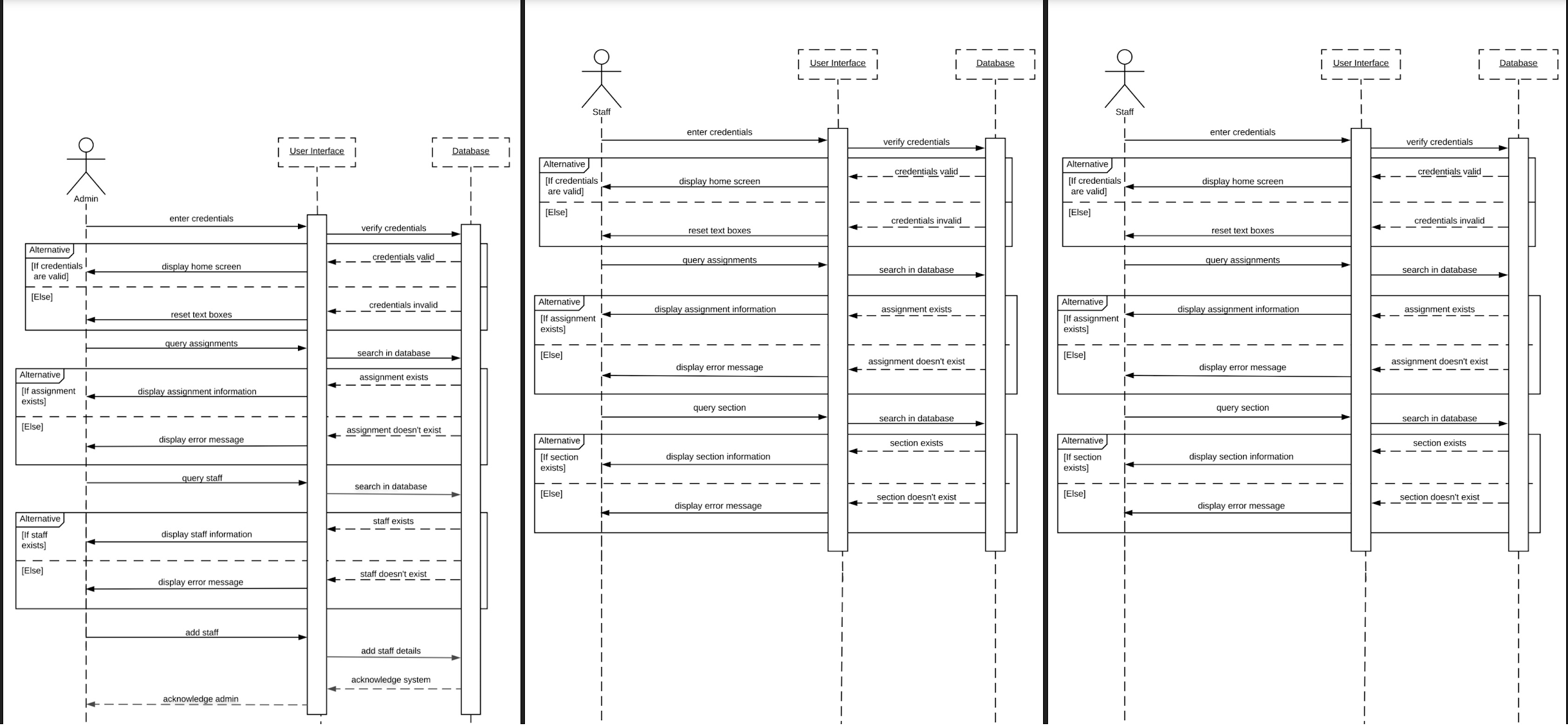
# Design

The software life cycle for ABLMS is Rapid Prototyping since I had changed the platform from Visual Basic to ASP.NET MVC because I 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 prototype of the webpages since I had originally planned on using Visual Basic, I used it to give me a visual idea instead, as shown in the photo below.



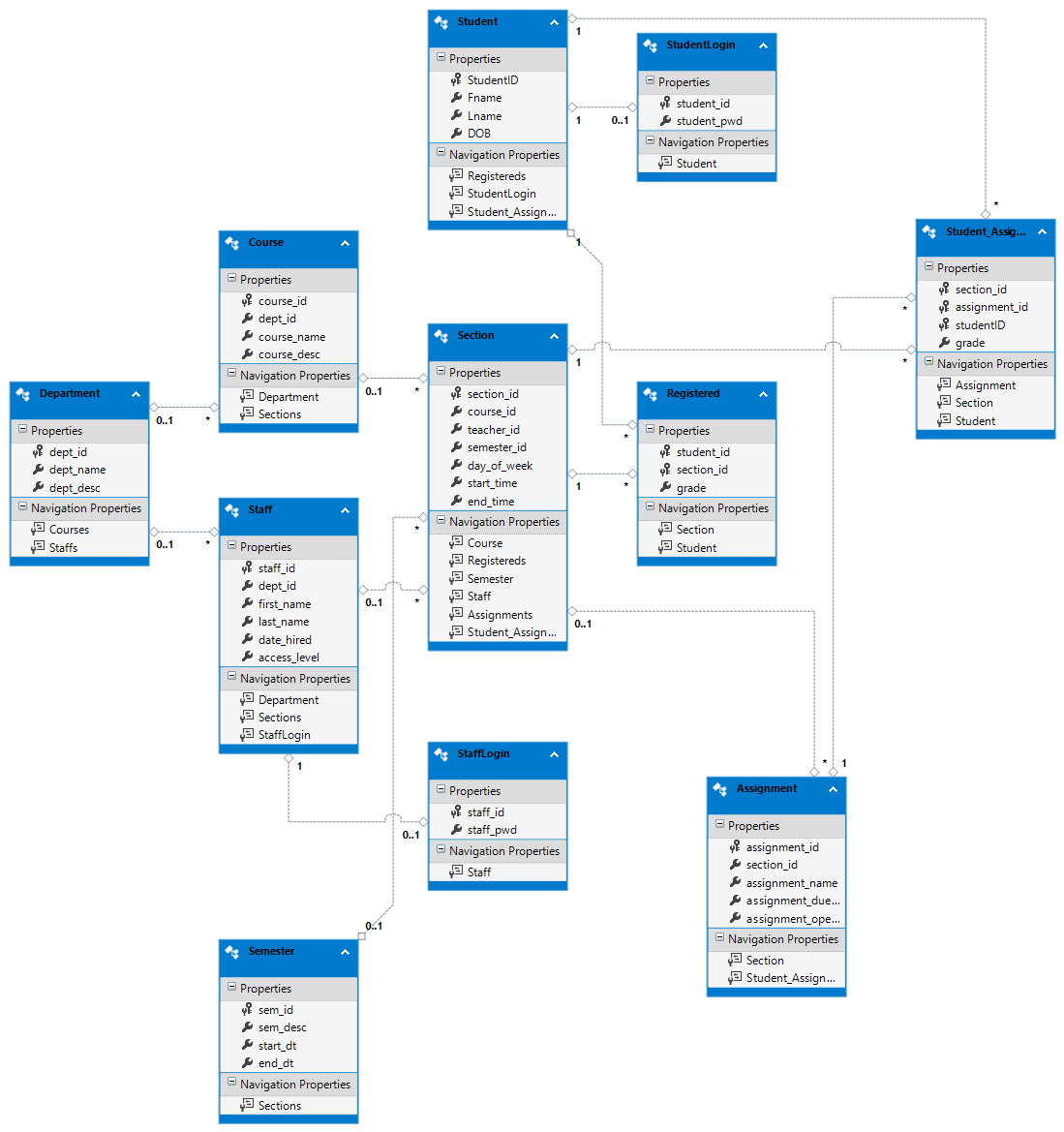
In the above image, the arrows indicated the transitions from one page to the next page.



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. There are 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 the assignment exists to the UI, which in turn sends that to the student or professor.

# Implementation

Alex’s Best LMS 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 my program is an Entity-Relationship Diagram, as shown in the figure below.



In this project, the 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 show the web’s user interface. Generally, this UI displays the model data. The controllers handle the browser requests, they retrieve data, and invoke view templates that return a response. In our MVC web app, the view only displays information; the controller handles and responds to input and interaction of the user. The controller is responsible for query-string and route data values, then it passes these values to the view. The controller uses these values to query the database. For example, http://localhost:####/Students/Edit/9 is a request to edit the student with ID = 9, from the Students table in our database, using the Students controller. The controllers and views work together.

# Testing

During this phase, I had to code and testee each code artifact separately, linked together all the code artifacts, and tested the product. 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.

I ran through each test case, and each time I found something I didn’t approve, or something didn’t output correctly. For example, if the labels looked like name\_id, then I would return to the code and change it to Name ID. I repeated these test cases until the software product met my project expectations.