



Department of Mathematics and Information Sciences
University of North Texas at Dallas
Dallas - Texas

College Class Scheduler

IT Capstone I

Carol Every
Cristian Chavez
Dylan Park
Viviana Hernandez

Supervisor:

Dr. Saif Al-Sultan

April 27, 2021

Abstract

Our proposed college class scheduler is designed to assist faculty in helping students schedule their future college courses. Course, degree plan and student information can be entered into the application. From information input the system is able to generate a report. This report will show users what classes have been completed and what classes are still required to complete a particular degree. The application will be able to provide this report by utilizing a database that is capable of holding a repository of all degree plans- combined with previous student information in order to deduce what courses the student still has to complete. The system will be developed in Java and use a MySQL database management system. The team will follow Scrum method of development.

Contents

1	Introduction	8
1.1	Introduction	9
1.2	Problem Definition	9
1.3	Motivation	9
1.4	Project Outline	10
1.4.1	Chapters Organization	10
2	Literature Review	11
2.1	Introduction	12
2.2	Related Work	12
2.2.1	Ellucian Degree Works	12
2.2.2	Conclusive Systems Advisor	13
2.2.3	Texas Common Course Numbering System	14
2.2.4	eConnect through Dallas College (formerly Dallas County Community College)	14
2.2.5	CAESked: A Class Scheduler for WMU Students	15
2.2.6	Software Requirements Specification of A University Class Scheduler	16
2.2.7	CyberMatrix	16
2.2.8	UniTime	17
2.3	Software Development	18
2.4	The Process Model Used In This Project	19
2.5	Software Implementation Tools	20
2.5.1	Programming Languages	20
2.5.2	Database Management Systems	25

2.5.3	UML Language	27
2.6	Summary	35
3	Requirement Specification and System Modeling	36
3.1	Introduction	37
3.2	Requirement Engineering Process	37
3.2.1	Elicitation	37
3.2.2	Analysis	38
3.2.3	Defining and Documenting Requirements	38
3.2.4	Requirement Specification and Agreement	38
3.3	Requirements Documentation	39
3.3.1	User Requirements	39
3.3.2	System Requirements	39
3.4	Requirement Models	41
3.5	Summary	44
4	System Design	45
4.1	Introduction	46
4.2	Database Design	46
4.3	Graphical User Interface Design	48
4.4	System Navigation	58
4.5	Summary	60
5	Implementation and Testing	61
5.1	Introduction	62
5.2	GUI and Explanations	62
5.3	Testing Scenarios	69
5.4	Summary	81
6	Conclusion and Future Work	82
6.1	Conclusion	83
6.2	Future Work	83
	Appendices	85

A Paper Degree Plan Example	85
B Implementation Code	88
B.1 Login Page	88
B.2 Main Page	93
B.3 Student	98
B.4 Plans	104
B.5 Courses	108
B.6 Plan Courses	115
B.7 Student Courses	120
B.8 Reports	126
B.9 UI Class	129
B.10 Report Data Class	146
B.11 Connection Pool Class	152
B.12 MySQL Database Creation Code	153

List of Figures

2.1	Composite Structure Diagram	28
2.2	Class Diagram.	29
2.3	Object Diagram.	30
2.4	Component Diagram.	30
2.5	State Machine Diagram	31
2.6	Use Case Diagram.	32
2.7	Sequence Diagram.	33
2.8	Timing Diagram	34
2.9	Interaction Overview Diagram	34
3.1	User Log-In Options.	41
3.2	Check Course Availability	42
3.3	Generate Report	43
3.4	Texas Common Courses Report	43
3.5	Course Pass Rate Report	43
4.1	Conceptual Model	46
4.2	Logical Model	47
4.3	Physical Model	48
4.4	Log-in Page	49
4.5	Main Page	50
4.6	Students Page	52
4.7	Plans Page	53
4.8	Course Page	55

4.9	Plan/Courses Page	56
4.10	Student Courses Page	57
4.11	Generate Report	58
4.12	System Diagram	59
5.1	Login Final GUI	62
5.2	Main Final GUI	63
5.3	Student Final GUI	64
5.4	Plans Final GUI	65
5.5	Courses Final GUI	66
5.6	Plan Courses Final GUI	67
5.7	Student Courses Final GUI	68
5.8	Reports Final GUI	69
5.9	Login test Incorrect password	70
5.10	Main Page Functionally Test	71
5.11	Students Page Functionally Test	72
5.12	Plans Page Functionally Test	72
5.13	Courses Page Functionally Test	73
5.14	Courses Page Drop Down List Test	73
5.15	Plan Courses Functionally Test	74
5.16	Student Courses Functionally Test	74
5.17	Reports Page Functionally Test	75
5.18	Reports Page Functionally Test	75
5.19	Signout Functionally Test	76
5.20	Data saved to MySQL Database Functionally Test	76
5.21	Grading system course required.	77
5.22	Major course remains on list until grade of C or higher.. . . .	77
5.23	Major course removed from list after passing grade.	78
5.24	Screen Resize Presentation Test	79
5.25	Screen Resize Presentation Test	79
5.26	Field Selection Color Change Test	80
5.27	Grayscale Presentation Test	80

Acronyms

ACGM Academic Course Guide Manual

CGI Common Gateway Interface

CSS Cascading Style Sheets

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IT Information Technology

JS JavaScript

JVM Java Virtual Machine

MSSQL Microsoft Structured Query Language

MVC Model-View-Controller

MySQL My Structured Query Language

NCSA National Center for Supercomputing Applications

ODBC Open Database Connection standard

OMG Object Management Group

PHP PHP: Hypertext Preprocessor

RDBMS Relational Database Management System

REST Representational State Transfer

TCCNS Texas Common Course Numbering System

UML Unified Modeling Language

UNTD University of North Texas at Dallas

W3C World Wide Web Consortium

WMU Western Michigan University

WWW World Wide Web

Chapter 1

Introduction

Objectives:

- Provides an introduction to the background for our proposed system.
 - Outlines and defines the need for our proposed system.
 - Explains the time frame for each part of our development cycle.
-

1.1 Introduction

This chapter will explain our problem statement and the motivation behind the development of our software design. It will illustrate project outline, which will clearly guide and state what our system will need to succeed.

1.2 Problem Definition

Our goal in developing this degree planning application is to create a practical and relevant tool that can be used to provide convenient reference notes for student advising. Our target audience for the software includes faculty members. Currently, degree plans are produced and dispersed as paper documents and images (Appendix ref4YearPlan). The purpose of our proposed system is to produce informative degree progression status for each student in a digital format. This application is not meant to be a substitute for advising. Instead, the intent is to provide a quickly accessible overview of how many courses have been completed in a degree plan and how many courses are still required. Data generated from this digital information provides factual talking points for faculty, students, and advisers as they select the best courses each semester until graduation.

1.3 Motivation

Our motivation for developing this system is to be able to provide a solution for the issues that students are having with scheduling classes. The proposed system will help automate the process of scheduling classes and help students generate their upcoming schedule and for faculty to maintain up-to-date degree audits. This system is intended for the use of students and faculty at the University of North Texas at Dallas. We also intend to make it as simple as possible for ease of use. The system would give the users suggestions for how to build their schedule as well as distinguishing classes that would best suit the user.

1.4 Project Outline

The following describes the proposed outline of the project.

1.4.1 Chapters Organization

- **Chapter 1: (Introduction)** This chapter introduces the original context to the system and provides an outline for the project.
- **Chapter 2: (Literature Review)**
This chapter provides reference and reading for background information on related systems and technologies.
- **Chapter 3: (Requirement Specification and System Modeling)**
This chapter defines the Requirement Engineering Process for our system and lists its system and user requirements.
- **Chapter 4: (Design)** This chapter shows the general Graphical User Interface (GUI) and database design for our system, and explains the system navigation.
- **Chapter 5: (Implimentation and Testing)** This chapter will show the code for our system, and will show tests for our system, as well as their results.
- **Chapter 6: (Conclusion and Future Work)** A conclusion of the proposed system will be given in this chapter. In addition, this chapter will present the possible directions of the future work.

Chapter 2

Literature Review

Objectives:

- Provide examples of related systems that provide similar functionality to our own system.
 - Define different types of software development and explain which one we will use to develop our system.
 - Define different possible programming languages and database systems that were possible to use in this system.
 - Define Unified Modeling Language (UML) and provide examples of different types of models we could use in our system.
-

2.1 Introduction

Proper selection of tools to be utilized in software design solutions can greatly enhance the quality of the product. Choosing the correct tools for tasks also affects the ease with which that product is produced. In this chapter, first we present a general overview of similar software applications and useful features available to solve some issues we are addressing. From this overview, we will focus on features relevant to our proposed application. This overview will also help determine the uniqueness of our proposed system and its features. Next we cover some common and useful tools available for planning, designing and implementing software systems. The goal of this comparative overview and available tools research is to create a framework that guides our software development process. Among the tools covered are various software development models, programming languages, database management systems and unified modeling languages.

2.2 Related Work

This section introduces the work that has been done in this area, It will show example software similar to the system we are proposing. The following paragraphs will explain these systems in detail.

2.2.1 Ellucian Degree Works

Ellucian Degree Works is a computerized set of academic planning tools designed to provide college advisors and students with near real time data for degree completion planning [1]. The software allows students to quickly view degree goals that are “ontrack” and “offtrack”. Degree Works client campuses range between 3,000 to 15,000 students. These can be 2-year Public Institutions, 4-year Private or Public Institutions. Degree Works is website portal based software accessible by Android, iOS devices, tablet, PC or Mac with an internet connection. Users need skills in Scribe, Surecode, Transit, Shepentry and SQL.

Features

- Determine what requirements you need to fulfill to complete your degree.
- View individual course grades, cumulative grade-point average (GPA), and major average.
- Determine which courses you have taken or transferred, and which ones count as electives
- View transfer credits, waivers, and exemptions applied toward your degree.

- See how your coursework can be applied toward another major, minor, certificate or major concentration using the ‘What If’ option.
- Estimate how many semesters it will take you to graduate.
- Learn the prerequisites for courses by clicking on the course number
- Look Ahead - a planning tool that allows you to see a degree audit showing courses for which you plan to register in future semesters, a list of “Courses you are considering”
- Easily document advising notes
- Provide an Athletics Eligibility auxiliary audit
- Provide a Financial Aid auxiliary audit.

2.2.2 Conclusive Systems Advisor

Conclusive Systems Advisor is managed by the vendor with upfront pricing. This online degree audit system cost 4 to 10 times less than Degree Works [2]. The vendor handles updates, patches and backups. Advisor runs on a proprietary programming language and has a point-and-click interface for users. Its online web portal can be accessed by Android, iOS devices, tablet, PC or Mac with an internet connection.

Features

- Entirely Web-Based...
- Wholly-Managed Systems
- Academic Planning/Scheduling
- What-If Scenarios
- Company ‘Features’ Matter Too
- Financial-Aid Report
- Grad-Checks
- Document Management Integration

2.2.3 Texas Common Course Numbering System

Texas Common Course Numbering System (TCCNS) is an online system that was designed to facilitate the transfer of general academic courses among Texas public institutions [3]. It allows the user to verify lower-division course compatibility with over 130 higher learning institutions in the state of Texas. With this, the user can cross-reference any Texas Common course with any higher education institution to confirm the compatible courses. The home page offers three different selections to traverse through the common course matrix. The users can cross-reference compatibility by selecting the following: Search by School, Compare Schools, or Search by Course.

When selecting the option to Compare Schools, the user can select two institutions at a time which allows the user to verify against the entire common course matrix. The first column reflects the number type and the title of the course. The TCCNS courses are identified by a four-character course prefix which represent the course type or academic discipline, followed by a four-digit course number. All prefix/number combinations correspond to the course descriptions listed in the Academic Course Guide Manual known as Academic Course Guide Manual (ACGM) which is published by the Texas Higher Education Coordinating Board. All TCCCN courses are listed in alphabetical order and listed side by side, which is a well laid out visual hierarchy. On the Homepage, under the Download Matrix, TCCNS provides the ability to download the matrix for the current and prior years as a single Excel document if they so desire.

The functionality of the system works well. The system does provide the user with the course description, as the system was designed to do. The only drawback is that the user must scroll down a list that contains over three hundred and fifty courses to find the desired one. One way its efficiency could be improved is by having tabs that expand or collapse which would allow the user to click on a course based on alphabetical or course number.

2.2.4 eConnect through Dallas College (formerly Dallas County Community College)

eConnect through Dallas College is a web interface that provides a variety of online services which include My Program of Study [4]. Program of Study is all course of study, requirements that a student must successfully complete to confer a degree, diploma, or certificate from the University. With this, the user has the option to explore, select/change, or view/print their program of study. Once logged into eConnect, under My Program of Study, the user explores which degree or certificate best fits them.

After the user gains access to My Program of Study, they are prompted to select an active program of study that they wish to view. Within a few minutes, the system automatically builds a personalized program of study based on the selection. The user also has the option to choose/change their Program of Study. The user is allowed to review as many programs as they see fit. The system requires the user to select a catalog year before proceeding. After a few minutes, the report generated is tailored to the user specific requirements.

eConnect is how we would desire our proposed application to function. This system will have some similarities to our proposed software, primarily with regards to the functionality of identifying and generating the required courses. The primary differentiating factor of our system is that it will strive to show relevant class options to meet requirements. With the option to allow/inform the students of other college equivalent TCCNS compatible courses regardless of where they are offered.

2.2.5 CAESked: A Class Scheduler for WMU Students

This project was developed by Chris Fruin and Jerry Grochowski at Western Michigan University in 2010 for their senior project. While it is similar in many ways to our project, some differences are this software is more focused on building class schedules for a singular semester, whereas our project focuses more on tracking degree requirements throughout the courses an Information Technology (IT) undergrad takes to become a graduate. Another difference is that this scheduler is made for Western Michigan University (WMU) Students specifically where our program would be specified towards University of North Texas at Dallas (UNTD) students. All that being said, there are some important similarities between the two projects that can be useful to our project.

The software (referred to as CAESked from here on out) is designed to help a student build a semester's class schedule by choosing the most optimal available times for each class the student wants to take [5]. A student can easily add the classes they need to take, and the classes are added to a graphical display of the week. Using this graph, the student can then make educated choices on the most optimal class times/days to choose. It is not able to see a student's previous classes or judge requirements (something our project is aiming to handle instead). CAESked also includes some extra features like the ability to see detailed info for a selected class and even has a map of campus that highlights the location of any one class. The documentation papers state CAESked uses PHP: Hypertext Preprocessor (PHP) and JavaScript and is a web application. It directly integrates data from the school to keep itself up to date, and it is accessible through its own website.

The similarities between CAESked and ours gives me a good idea of the type of data we will need to process, as well as the data we will need to store. In order to keep track of any student's completed classes, we will need a database of all current classes, as well as their requirements. We can do this by hand, but the most optimal way would technically be through some API or similar web scraping that keeps up with changing course requirements and other changes. Despite this being a scheduler and handling a different part of the class registration process, a lot of the work that was done at the very least stands as a good example and inspiration for ways to handle similar issues in our project

2.2.6 Software Requirements Specification of A University Class Scheduler

This paper was written by Deanna Needell, Jeff Stuart, Tamara Thiel, Sergiu Dascalu, and Frederick Harris Jr. at the University of Nevada [6]. While it is not a paper on the fully realized project, it has class diagrams and explanations on the design, and the project is similar enough that it can also be related back to our idea. It is different in scope to our project, as the paper specifies a program that is more so meant to be used by college administrators to schedule many course classes.

Despite its differences, many of the design choices made can still apply to our own course calculator. For example, it was written in C++, a class based language. All of its data files were also stored in XML format, which is something good to consider. A more modern option vs XML would be JSON format, but both are good ways to store data and possibly hand it between APIs. It uses Qt for its user interface, something slightly similar to a more advanced implementation of JavaFX. The methods in which they store class data is probably one of the more important takeaways from this paper, having classes for different professors, rooms, classes, etc. Using a system like that to organize classes within the program will be in our best interest.

2.2.7 CyberMatrix

CyberMatrix is a software program which permits single or multiuser software for scholar class scheduling. The software program's major focus is for colleges and different instructional establishments which could want to have fast scheduled classes. The software has many features that are made available in order to provide a more efficient way to schedule classes that are needed for students. It has automatic scheduling which it would take the input of what classes are needed for a student and choose from a list of possible schedules. If they choose to do so they can also have bulk input in which generates classes that students require to finish.

Another feature that can be beneficial is that it can also let you adjust the schedule manually. If the schedules that are suggested do not seem to fit the schedule of the student they can be edited to better suit the students schedule. The software contains a feature which will check if the student has finished the prerequisites in order to take the following class. The search option can also narrow down the classes which the student may be looking for. It also allows for different views which can be either viewed by class schedule, student schedule, instructor schedule, and as well as classroom schedule.

A feature that seemed interesting is that it also has multi language support which can change the text into the language equivalent which non-English speakers can understand. It has system requirements of at least 2GB RAM and 30 MB disk space.[7].

2.2.8 UniTime

Unitime is a complete academic scheduling software that helps develop course timetables amongst different things. Which consists of scheduling college students to individual lessons and sharing rooms with different events. It permits more than one college and departmental agenda manager to coordinate efforts to construct and alter a timetable that meets their numerous organizational desires whilst taking into account the minimization of pupil course conflicts. The software has four different components compromised of course time tabling and management, examination timetabling, event management, and student scheduling. Starting with course timetabling its main purpose would be to place every course at a time which could now no longer interfere with other classes that the pupil is taking. Course management coordinates efforts to build and modify a schedule that meets their diverse organizational needs while allowing for minimization of student course conflicts. As it does with courses it also builds complete exam schedule each term while minimizing conflicting exam placement for students. The student scheduling procedure is the matching of the units of classes required by every pupil to the appropriate available class spaces in order to fulfill the various pupil academic requirements[8].

Our design differs from similar projects for multiple reasons. Primarily, the biggest difference is the fact that this tool is being designed specifically for University of North Texas at Dallas students to be used in conjunction with the MyUNTD system already in place. The design and focus of this application will be degree completion. It is meant to be a tool that provides a convenient overview of courses students still require in order to graduate.

2.3 Software Development

- Incremental Development Model

The Incremental Development Model also follows the waterfall model [9]. However, an incremental model is developed in multiple cycles, but each cycle is still a subsequent release of the previous cycle. It splits the Requirements up into system increments. Requirements are developed based on priorities. Although the development is performed in multiple incremental cycles, the release is still a single release like the waterfall model. When the development of an increment is started, it freezes the requirements. We must complete every increment. After completion, they can hand it to the client for feedback. After receiving feedback then we would start working on the following increments. The requirements for later increments can continue to evolve.

- Scrum

Scrum is one of the modern agile models used to produce and deliver high-quality complex products [10]. A key part of the Scrum framework is to help small teams of people work together on projects with clearly defined components. Once roles, relationships, events, artifacts and rules are defined, the team functions whether there is one small group or multiple small groups working together. Scrum expands on the iterative and incremental models by leaving room for the knowledge gained as teams work on and make decisions about the project or problems based on what is known. The idea is to use transparency, inspection, and adaptation to continuously improve the product, the team, and the working environment. Scrum teams have a Product owner, Development team and Scrum Master. They also have guiding values that help teams build trust. Teams use a Sprint to plan out the work to be completed in increments that do not exceed one month at a time. When one Sprint is completed the next one starts. All team members must understand and mutually define what “Done” means.

- Waterfall

The Waterfall Model is a process model that utilizes a linear life cycle model, where each task leads to the next task until you need to perform maintenance, where you can reset at the appropriate level of the waterfall [11]. In this model, the entire team focuses on a step in the process and does not continue until the step is done. The waterfall approach was one of the first SDLC models to be widely used in software engineering. The steps you have to take in the waterfall model are as follows: Requirement Gathering, System Design, Implementation, Integration and Testing, Deployment, and

finally Maintenance. In order to utilize the Waterfall model, your project requirements need to be well documented, and your project should be stable. The other important factor is that the project needs to be short, as in longer projects it is hard to judge progress within steps, and changes in requirements cannot be made.

- Extreme Programming

Extreme programming follows the agile process that aims to produce higher quality software[12]. Extreme programming is the most specific of the agile in terms of engineering practices of software development. It was also one of the first agile methods and it was primarily dominant in the 90s and early 00s. It is thought that XP was the most significant approach to changing software development. XP focuses on the application of programming techniques, clear communication, and team work. In XP requirements are expressed as scenarios which are directly implemented as a series of tasks. In which programmers work in pairs where they make tests for each task before going forward with the code[13]. At first extreme programming was considered to be controversial due to the fact that it introduced multiple agile practices that were different. The principals that make up extreme programming are collective ownership, continuous integration, incremental planning, on-site customer, pair programming, re-factoring, simple design, small releases, sustainable pace, and test first development. Although it is an agile programming framework companies pick and choose which XP practices work the best with there way of work. It is used often with scrum which is a management focused agile method[14].

2.4 The Process Model Used In This Project

The process model we will use in this project will be Scrum. We chose this model for a couple reasons. Firstly, it is an agile model, meaning it is more modernized compared to other non agile models. It is also geared towards smaller teams, allowing more focus on specific portions of the project through development [10]. The development team organizes themselves, and all work together on all portions of the project. This is imperative in a smaller group, allowing for more close relation and contact between members. The Scrum cycle consists of different ‘Sprints’ where the team works to produce a small releasable product each month, of which they increment into the final product. Each day of the sprint there is a ‘Daily Scrum’ where the team meets to discuss what is and is not working, and what goals to achieve for the day. After that, the team works for the rest of the day on their own goals. We think the Scrum model facilitates a lot of good practices that benefit our smaller group size.

2.5 Software Implementation Tools

There are many different tools for developing a software. This section will go over different types of programming languages, database management systems, and UML language.

2.5.1 Programming Languages

- HTML:

When one logs into the World Wide Web, with no regards to what type of device or browser one is using, your device must be able to communicate. There has to be a universal language that all these different devices using different platforms can all understand. This is where Hypertext Markup Language (HTML) comes in to play. It is part of the foundational coding languages on the internet. It creates the structure of a web page. It uses “tags” to markup a language, hence the name. It adds bits of code to markup the text which communicates to the browser how to display the page. It is used to define all the content, the text, the images, and the links. Web browsers like Firefox and Chrome translate HTML language into visual web pages. Without a browser, this language is just words on a page. Depending on what web browser one uses, each web browser will translate HTML slightly different. HTML, Cascading Style Sheets (CSS) and Javascript need one another to make a modern website. These code languages make up front-end web development.

HTML is constantly evolving, HTML was developed by Tim Berners-Lee [15]. It became more popular after the Mosaic browser adapted it, which was developed at the National Center for Supercomputing Applications (NCSA). Since then HTML has grown in numerous ways, but for it to work on the Web, everyone must share the same unified protocols.

- CSS:

CSS is a part of the standards established and adopted by the World Wide Web Consortium (W3C) [16]. When the styling language CSS is paired with Hypertext Markup Language (HTML) designers are able to enhance the presentation style and create consistent layouts for web pages. HTML provides the content (text) and CSS allows style options like font size, type and color, margins sizes and background color to be easily changed. Tables can also be positioned and styled. Options like these allow web designers to create unique and more personalized looks for websites. If a font size needs to change, update the style sheet and all associated pages for the website will reflect the update. Changes can

be page or element specific. Once CSS code is completed the resulting style can be reused on multiple pages for a current website project or slightly altered for a different look on another website.

Before CSS web pages defaulted to browser setting for font size and color. Information could be presented online but not styled efficiently. This meant web pages were presented as text while lacking the formatting necessary to make it easy for users to read. Because World Wide Web (WWW) users were utilizing the internet more and more for electronic publishing, there was a need to style web page layouts. One of the earliest goals was the ability to present web pages in the style of newspapers—with columns. Several style sheet options were being explored in the early 1990s to solve this problem however, CSS distinguished itself with the unique ability to cascade (“allows several style sheets to influence the presentation of a document”). With this cascading ability change request between designers, browsers and readers could now be accommodated. Web page readers can customize the look of the page to suit them as they view information with a larger font or different page color. These reader changes can be reset in the browser to the default web author settings and do not affect how the page is displayed to other readers. Collaboration between Håkon Wium Lie [17], Bert Bos [18] and Tim Berners-Lee [19] helped keep CSS- browser and operating system independent. The style sheet can be created in any text editor and this file saves with a .css extension.

Today CSS is used on most websites. CSS visual design modules have evolved through the years and added features that accommodate the rapidly changing delivery of information on the internet. With CSS a web page can be displayed seamlessly in various browsers like Firefox, Chrome, Edge, Safari or Opera. Websites with CSS can be displayed on Android, Microsoft Windows, Apple Mac or iOS and Linux systems. Websites can be responsive so that the web page displays properly regardless of device screen size – desktop, tablet or phone. CSS Flexbox Layout Module allows web authors to create one dimensional (row or column) of elements. While the CSS Grid Layout gives the ability to create two-dimensional (rows and columns) of elements. New modules and integrations continue to expand the abilities of CSS.

- JavaScript:

JavaScript (JS) is a lightweight, interpreted, prototype-based language focused on front-end web development [20]. It runs on the client side of web technology and is used to alter the look and behavior of web technology. In addition to websites, JavaScript can also be used in other environments like Node.js, which is a server side environment that runs JavaScript programs in its own engine. It is

not to be confused with Java; while both are owned by Oracle, Java and JavaScript handle massively different functions. JavaScript's main functions in things like websites or web apps are usually to:

- Add or modify HTML on the page
- Add or modify CSS elements
- Handle a user's clicks
- Perform basic operations on user inputted data
- Handle local cookie storage

When run server side, JavaScript can be useful in the generation of web pages, and handling of API requests. For example, Node.js can be used to create a simple web server that can handle Hypertext Transfer Protocol (HTTP) requests. These servers are known as Representational State Transfer (REST) servers. This in turn can be used to create RESTful Web Services that can send, retrieve, and modify data on the server. These types of services allow programmers to utilize large data sets that other established services have already, and also can be used internally to modify data in a database easily. By utilizing web-based technology, JavaScript remains highly compatible across many devices and has good backwards comparability between different versions.

- Java:

Java is considered to be a High-Level language, which in essence means that it is similar to English and easy to use. As such Java is platform-independent, which means that you can write a program and run it on different types of machines. Although the program is written in Java it needs to be translated for the computer to understand. Java uses a compiler often referred to as Java virtual machine(Java Virtual Machine (JVM)). The job of JVM is to take the high-level language, which is Java, and translate it into machine code for it to be executed[21].

Java was developed by a team led by James Gosling at Sun Microsystems in 1991. Its original name was Oak but then was later renamed as we know it now Java in 1995. It has become extremely popular mainly since you can write a program once and run it anywhere. "As stated by its designer, Java is simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multithreaded,and dynamic." Today java is also used for building stand-alone applications across multiple platforms.[21].

- PHP:

Hypertext Preprocessor (PHP) is an open source, server side, general purpose scripting language. PHP can be used on operating systems like Linux, Unix variants (HP-UX, Solaris and OpenBSD), Microsoft Windows, macOS and Android. When installed along with web server software, like Apache or Microsoft IIS, developers from novice to expert have a wide selection of today's most popular back-end tools available.

One of the most popular features of PHP is the ability to create dynamic web pages. Once installed on a web-server PHP can be used with HTML to expand the functionality of web pages. PHP syntax directly embedded into HTML source code easily adds features to web pages that HTML alone does not have. These PHP-enabled web pages are created and edited like regular HTML pages. Added features include things like collecting values from user input and converting these values to other formats (e.g. yards to feet). Other user data inputs like address books entries, forum history and survey responses can also be collected. PHP can then take this user's submitted data and store it in a database like My Structured Query Language (MySQL) [22]. Using PHP, HTML and MySQL is a popular and inexpensive database solution for businesses. However PHP can connect to any database that supports the Open Database Connection standard via the Open Database Connection standard (ODBC) extension including Microsoft Structured Query Language (MSSQL), Oracle and Sybase [23]. In addition to the above features PHP comes bundled with the GDLibrary [24]. This library allows developers to create simple graphics or to edit existing graphics. With the PHP parser (Common Gateway Interface (CGI) or server module), a web server and a web browser, developers can build their web pages, view the output and make adjustments as needed.

PHP has many other features besides these. It allows developers to use procedural or object-oriented programming or a mixture of both. Also, PHP itself can be extended to add even more features. For example, PHP-GTK can be used by advanced developers to create client side desktop applications [22]. The abundance of features and wide compatibility with various hardware and software result in more than 20 million websites around the globe using PHP [25]. Yet the simplicity of PHP allows it to be installed for learning purposes on a home computer in a budget friendly setup. PHP, Apache and MySQL are all open source free software.

- Python:

Python is an Object-Oriented programming language developed by Guido van Rossum [26]. It is a free and open source language, and has recently become one of the most popular data science languages. On top of that, it is also widely used in web development. Python runs as the back-end to many web servers and APIs. Python is very easy to develop with due to its massive and easily accessible library of ready-made software and libraries you can implement and contribute to. This makes starting projects very easy. The syntax for Python is very English-based and uses white space indentation. While being similar to languages like Java and C++, it stands to be easier to both read and write. Python is mainly an interpreted language, meaning it is not always (but sometimes is) compiled before running. Generally, the Python interpreter you install will convert the code line by line to byte code instead of compiling it all to machine code. This can effect run times, but it allows Python to be platform independent, and lets you collaborate on projects across platforms.

Good Python code adheres to the Zen of Python [27]. These are suggested practices that Python programs should follow. Some of these include:

- Beautiful is better than ugly.
- Sparse is better than dense.
- Errors should never pass silently.
- In the face of ambiguity, refuse the temptation to guess.

This list can be read from any Python compiler by calling the command: `import this`. While not absolute requirements, following these rules is a good way to ensure you are thinking Pythonically, that is thinking and writing code that follows the same general style.

- C++:

C++ is a programming language that "is close to machine" and as well as "close to the problem to be solved". What it means is that it makes it simple for the machine to handle the important aspects that are obvious to the programmer and so the solution can be expressed concisely. C++ comes from the programming language known as C it just has added function argument checking, consts, classes, constructors and destructors, exceptions, and templates. With these features, C++ is based on the idea of providing both direct mappings of built-in operations and types to hardware to provide efficient memory use and low-level operations, and affordable and flexible abstraction mechanisms to

provide user-defined types with the same notational support, range of uses, and performances as built-in types. C++ deals with fundamentals such as memory, mutability, abstraction, resource management, algorithms, error handling, and modularity. C++ is biased toward system programming meaning it uses hardware resources, has serious resource constraints, or closely interacts with code that does. C++ like other modern programming languages usage has a vast range. C++ has four different programming styles procedural, data abstraction, object-oriented, and generic. The creator of C++ is Bjarne Stroustrup who invented it in 1979 which started as an extension of the C language.[28].

- Ruby on Rails:

Ruby on Rails also known as Rails is a software framework that is designed to support the development of web applications. It is written in the language Ruby [29]. It follows the Model-View-Controller (MVC) architectural pattern. The main advantage that Ruby on Rails offers over other frameworks is that it can release cycles in a fast and simple way. The framework advocates “Conventions over Configurations” which just means that one can get applications running without an enormous amount of code required. Another great advantage is the meta-programming technique, which allows the use of writing programs using programs. Another asset is that Ruby on Rails offers scaffolding. With scaffolding, one can create temporary codes to get the application up and running, which allows one to see how it will all come together.

The founding “fathers” of Ruby and Ruby on Rails are Yukihiro Matsumoto and David Heinemeier Hansson [30]. Yukihiro Matsumoto is the creator of the high-level programming language from Japan, which was first launched in 1995. David Heinemeier Hansson developed Ruby on Rails in Ruby in 2003. It was not until 2004 when Hansson released it as open-source [29].

Rails continue to develop and implement new projects. The following list contains the names of projects that use Ruby on Rails: Airbnb, Basecamp, Couchsurfing, GitHub, Dribbble, and Hulu. Ruby on Rails’ latest version is 6.0 and was released on August 16, 2019 [29].

2.5.2 Database Management Systems

- Oracle Database

Oracle Database in currently on release 19 and release 20 is currently being evaluated [31]. The database runs on operating systems like Oracle Solaris, Microsoft Windows, Linux, IBM AIX and HP-UX. Oracle sees the databases of the future possible in their Autonomous Database [32]. These

databases use cloud based technology and machine learning to automate routine database management tasks like updates, security, and repairs. They are called self-driving databases. Some other editions of Oracle database include Standard, Enterprise, Developer and Personal [33]. There is also a free Oracle Database XE (Express Edition) that is available for download with an account [34].

- MySQL:

MySQL (also known as My Structured Query Language) was one of the first open-sourced database that the world had seen. It is a Relational Database Management System (RDBMS) which ensures that the data being put into the database is very structured and organized [35]. Not to be confused with SQL, which runs on top of MySQL to query the databases. It uses SQL to perform operations such as storing, retrieving and modification of a database. It is a database tool that when used alongside other tools such as PHP and Apache Web Server creates a huge dynamic package making it very powerful. It runs on several operating systems including: Windows, Linux and Mac OS X. MySQL provides a prominent feature that is multi-user access that can access it at any point in time. My SQL database system comprises a wide range of database technology and so many technologies supports that cater to users with different requirements. MySQL is one of the most popular database softwares that exist today. The latest version available today is MySQL 8.0.

- SQLite:

SQLite is a self-contained version of an SQL database engine [36]. It does not require a server and takes very little to set up. It is a transactional database engine, much like Microsoft SQL. It is hosted in the public domain, meaning it is free to use for any project. The entire database is able to run off of a single file and is able to run off of the same disk that is requesting the data. SQLite does not have a server counterpart, making it able to be useful for storage of local files. The library files are cross platform, and can be kept very small. Reading data from a SQLite database can be quicker than from straight file reading due to the way SQL transactions are organized. SQLite is very useful for smaller projects that are unable to host a fully fledged server, and also only need to store and read files locally. It is built to be very future proof and has long-term support in mind for future development.

- Microsoft SQL:

Microsoft SQL server is a relational database management system that was developed by Microsoft. It was first started in the year 1989 and has been updated since then. Microsoft SQL server comes in different varieties and styles depending on what it is going to be used for. It has standard versions

and specialized versions of it. It does have a free version which is called developer that is available for the public. Microsoft SQL server is described as fast and agile. It also contains an AI in order to help with faster predictions and better security. It can be used with with the language of your choice with open source support. It can run on windows, Linux, and containers on premises, in the cloud, or in hybrid environments.[37].

2.5.3 UML Language

In this section, we will describe some of the 14 different Unified Modeling Language diagrams. The UML is a language of blueprint for software [38]. It categorizes them into two different diagrams, one being structural and the other behavioral. The Unified Modeling Language models the static structure of the system. It shows the relationships between Classes, Objects, Attributes, and Operations. The choice of what models to create has a profound influence upon how one attacks a problem, and they shape a solution. No single model is sufficient: complex systems are best approached through small sets of nearly independent models. It may express every model at different levels of details and functionality built into each. They connect the best models to reality. The UML movement has grown since it was first created.

The Object Management Group (OMG) is a non-profit corporation founded in 1989 [39]. It has over 800 members and works to establish industry guidelines and specifications to provide an accepted framework for application development.

Rational corporation hired Grady Booch, Ivar Jacobson and James Rumbaugh developers of UML, and they worked together to unify and bring their methods of compliance with each other [40]. When it was first developed there were over 50 different notations out there to do Object-Oriented Software Modeling. They took the best of all of them. With this, they created a big committee inside OMG to work on putting them together. Then eventually came out with the first approved UML 1.1 version in 1997.

UML is a standardized modeling language that can be used across various types of programming languages and development processes. This would make it easier to understand it and be able to apply it to our project College Class Scheduler. They're useful in an Agile development environment and keep development productive and focused. UML can make it simpler to navigate source code and plan out new features before any other work is done. It can also make it easier to communicate with either technical and non-technical people and have it evolve as the project continues [41].

- Composite Structure Diagrams:

The Composite Structure Diagram was introduced in UML 2.0 [42]. It is designed to showcase the relationships between classes, interfaces, and packages. More specifically, it is made to show the internal connections in classes. These types of diagrams are useful when you want to show a large amount of classes and the internal workings of each class. Figure 2.1 shows a Class Diagram [42].

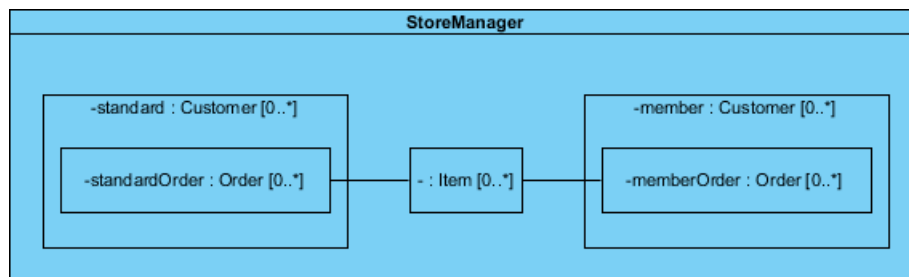


Figure 2.1: Composite Structure Diagram

- Deployment Diagrams:

Deployment Diagrams are designed to show the run-time configuration of all the components that run within your system [43]. It shows the hardware, software, and middle-ware used throughout the project. These types of UML documents are best used to describe systems where you are designing both the software to be used, as well as custom hardware, and how everything interacts with each other. It can also describe the makeup of a server, with connections to the web-server, app server, and any databases and devices that utilize it.

- Package Diagrams:

Package diagrams are structural diagrams made to display the organization of the different high-level elements in a project [44]. Package diagrams show the overall structure and dependencies between different modules by separating them into packages. These packages are used for organizing larger systems, and can contain diagrams, documents and other types of UML documentation. One use of this type of diagram is to use it to organize a large number of other UML diagrams and other supporting documents used throughout your project. In this way you are able to keep track of each portion of your project and can refer down into each package in order to learn more about a specific part.

- Profile Diagrams:

Profile diagrams provide a way to describe extensions in UML classes [45]. These extensions can be

used for different effects. For example, surrounding the name of your element with << and >> symbols allows you to make that element appear as primitive. You can also add tags to elements, which can be useful for marking who wrote a class or for version control. Finally, there are constraints you can add onto a class, which limits the value of some variable in the class to specific values, avoiding errors. These used in conjunction can be added to other UML documents in order to give more detail to classes.

- Class Diagrams:

Class diagrams are structural (static) type diagrams [46]. Class diagrams are used to describe a set of objects, their attributes and the relationships between those objects. Each class in the class diagram is represented by a rectangle divided into three sections. The top section gives the class name, the middle section is for class attributes, and the third section is for class operations/methods. Figure 2.2 shows a Class Diagram [47].

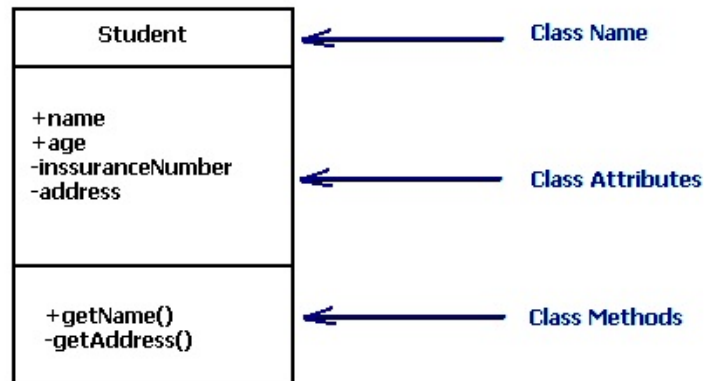


Figure 2.2: Class Diagram.

Attributes (fields, instance variables) and operations/methods are entered left justified, with the first letter of the names in lowercase. Attributes are noted in the format *visibility name:type*. Operation/methods are noted with *visibility name(parameters) : return_type*. Symbols are used to designate visibility as public (+), private (-), protected (#), package (~) or derived (\). Static methods are underlined. Arrows connect and define related classes.

- Object Diagram:

Object diagram is another structural diagram type [48]. It is used to show an instance or a “snapshot” of class objects at a particular instance. This gives a more detailed view of objects and their

values. Individual object slots are defined by $-objectName : type-attribute = value$ and links show associations between objects [49]. Figure 2.3 shows a Object Diagram.

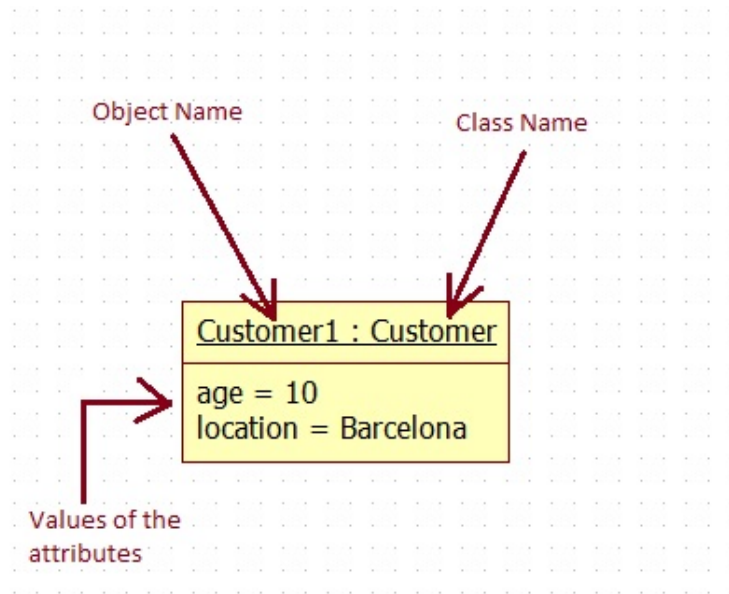


Figure 2.3: Object Diagram.

- Component diagram:

Component diagram is a structural (static) type diagram [50]. These diagrams show system components along with provided or required interfaces, ports, connectors and the relationships between these components. Components can be logical (business or process components) or they can be physical (software). Component diagrams show system dependencies at a very detailed level or from an overview level [51]. Figure 2.4 shows a Component Diagram.

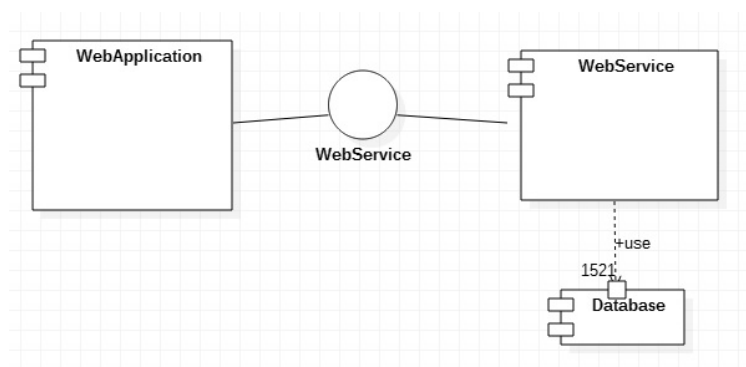


Figure 2.4: Component Diagram.

- State Machine Diagrams:

State machine diagrams are behavioral type diagrams [52]. These diagrams describe the behavior of objects according to their state at the moment. Using certain triggers and an event pool what happens in the system is diagrammed. There are two kinds of state machines - behavioral state machine (finite state transitions) and protocol state machine (usage protocol or lifecycle). Figure 2.5 shows a State machine diagram [53].

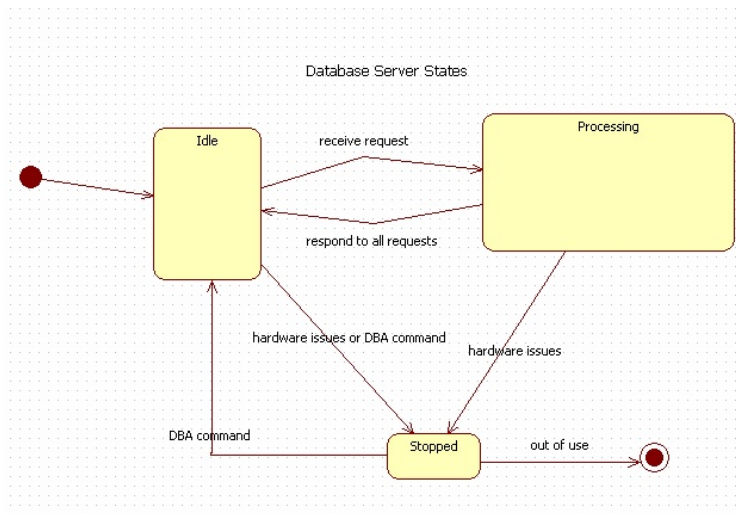


Figure 2.5: State Machine Diagram

- Use-Case Diagrams:

A Use-case diagram is a visual representation of what a system must do [38]. This model creates a visual representation so that anyone who is not a software engineer can understand. This diagram falls under the behavioral UML diagram type. This model models the process of a system using actors and use-cases and how they influence each other. Actors are referred to as the users, and the use-cases are referred to as the set of actions, services, and functions that the system needs to do. This model of this diagram is very handy because it provides a great visual representation of what the system would look like and how the system would function. By developing this diagram, one can see if there are any internal or external factors that can affect the outcome of the system. Figure 2.6 shows a Use Case Diagram [38].

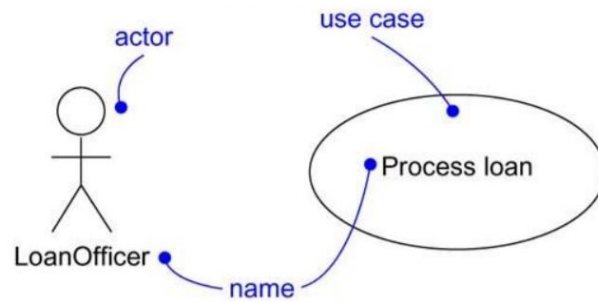


Figure 2.6: Use Case Diagram.

- Communication Diagrams:

We can describe the communication diagram as a model that reflects who is communicating with whom [38]. This model shows us how the actors and objects are communicating. With this model, you have undirected lines that show that communication is occurring but with no return method. The reason for this is that we are to assume that their communication is a two-way process. The communication is noted with every undirected line we assign the name of the message that is occurring.

- Activity Diagram:

The activity diagram works much like the flowchart [38]. The basic nature of this diagram is scenario specific. They represent a set of activities performed in a particular scenario. The primary purpose of this diagram is to layout all the positive and negative scenarios that can happen. However, the activity diagram is more detailed than the flowchart. This model reflects both computational and organizational workflows. The activity diagram falls under the behavioral UML diagram type. The cases modeled are sequential and concurrent. In this diagram, we identify major activities associated with conditions. One important thing to point out is that an activity diagram does not match exactly with code. The activity diagram is made to understand the flow of activities and is primarily used by business users.

- Sequence Diagram:

The sequence diagram or as sometimes referred to as an event diagram and is a structured representation of behavior as a series of sequential events over time[41]. It can be used to depict workflow, captures the exchange of information and responsibility through the system. It can also be used to make explanatory models for use case scenarios. The sequence elements are arranged in a horizontal sequence where messages are sent back and forth between elements. An actor is used to represent a user, a stereotyped element includes boundary, control, and entity can be used to represent screens, controllers, or database items. An example of sequence diagram is shown in figure 2.7[54].

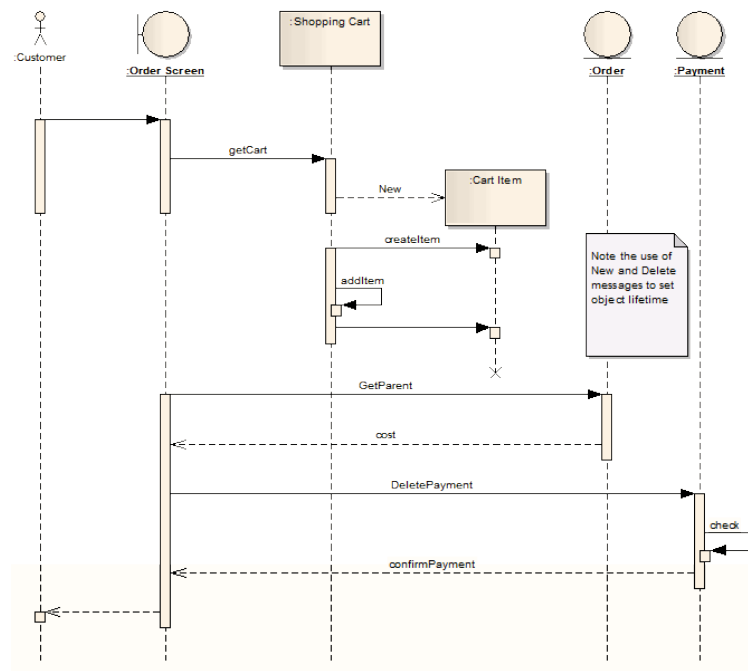


Figure 2.7: Sequence Diagram.

- Timing Diagram: A timing diagram explain the behavior of different objects in a time scale. It can be used to see how much time each step of a process takes and be used to evaluate to make it more efficient[41]. Another use of the timing diagram can define hardware driven or embedded software components and specify time-driven business processes. An example of a timing diagram is shown 2.8[55].

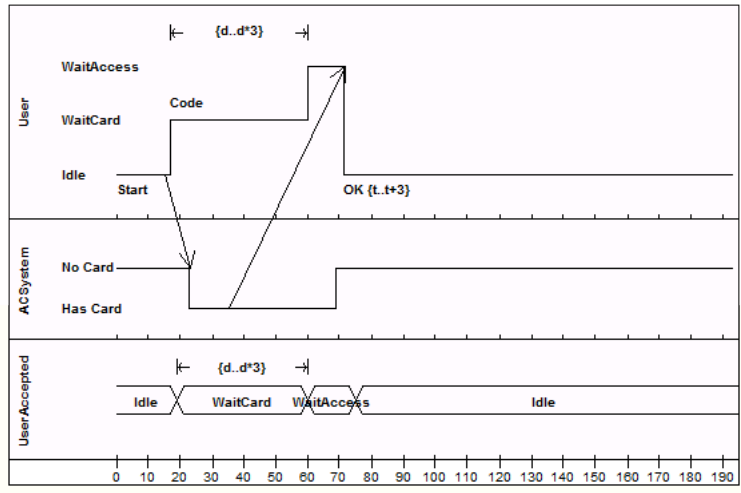


Figure 2.8: Timing Diagram

- Interaction Overview Diagram: Interaction overview diagrams show how the cooperation between other interaction diagrams in order to illustrate a control flow. It includes initial nodes, flow final nodes, activity final nodes, decision nodes merge nodes, fork nodes, and join nodes[41]. There are two types of elements one of which is interaction elements which display an inline interaction diagram. They can be any one of the four types Sequence, Timing, Communication or Interaction Overview. Another element is an interaction occurrence elements are references to an existing interaction diagram. An example of an Interaction Overview Diagram is shown 2.9[56].

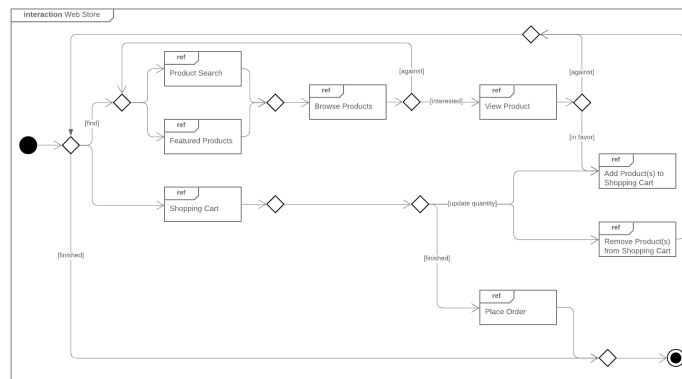


Figure 2.9: Interaction Overview Diagram

2.6 Summary

By evaluating the different applications, tools, and models that are available today, we concluded the following. The common and distinctive attributes of each application when compared to our system. The Scrum process model we elected because of all its features and agility. The particular programming language and tools that we chose and why we opted to use them. The next chapter will go over the requirement engineering process and illustrate the requirements of our system.

Chapter 3

Requirement Specification and System Modeling

Objectives:

- Define the Requirement Engineering Process and how it is used in our system.
 - Explain the user and system requirements of our system.
 - Illustrate the requirements of our system using requirement models.
 - Illustrate the system flow with an activity diagram.
-

3.1 Introduction

This chapter will introduce the concept of the requirements of the engineering process which encompass Elicitation, Analysis, Defining requirements and Specification. This chapter will also provide context about the user and system requirements for the application. Which will demonstrate an overview of the entire system by looking at it from a broad perspective.

3.2 Requirement Engineering Process

The Requirement Engineering Process is the process in which a project's requirements are either decided or changed. These requirements are used by developers, and users alike in order to understand how a system works and the rules it operates on. The steps of the Requirement Engineering Process include Elicitation, Analysis, Defining requirements, and Specification.

3.2.1 Elicitation

The first and by far the most important step when starting your engineering process is eliciting the requirements. Eliciting can be defined as capturing, discovering, or gathering the requirements needed to begin the process [57]. Normally this job or task is assigned to a business analyst, application architect, or something similar. Not only does this person obtain or gather all the information, but they also must manage any of the requirements that a stakeholder may have. This step is crucial, and it requires input from other people that would potentially be affected by the future business solution. This process ensures that the requirements will meet their business needs. In order to achieve this, one must identify potential stakeholders very early in the project. By ensuring that whoever is responsible manages the requirements of the elicitation process. The business analyst must learn to identify, track, and report the progress towards the requirement completion. This person would also oversee defining, documenting, and analyzing any business problems to discover any hidden requirements. This individual must facilitate effective requirement brainstorming sessions. They must also use critical questions to initiate the requirement elicitation process. They must also capture and communicate assumptions that the stakeholder requirements need. A major cause of project overruns or failures is usually identified by missing and/or misunderstood requirements.

3.2.2 Analysis

One of the steps that is important in the requirement engineering process would be requirement analysis. In the requirement analysis you have to specify which software operational characteristics. It will also indicate software interface with other system elements and establishes constraints that software must meet. The analysis step allows a software engineer to elaborate on basic requirements. Requirement analysis also allows the engineer to build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed. [14].

3.2.3 Defining and Documenting Requirements

Another part of the quality framework for this project is Documentation. From the proposal to completion of the project, each phase should be thoroughly documented. In this step, user and system requirements are defined to give clear and concise descriptions that set guidelines for the scope of the project. User requirements provide non-technical descriptions of tasks to be completed. While system requirements provide detailed technical specifications. Each task is an individual unit of work that must be completed [58]. These requirements are then translated into graphical representations like use-case, sequence or activity diagrams. Diagrams are used to make the process easier to understand. Relationships between tasks, who is responsible for completing tasks, what deliverables are expected and how end-users will interact with the application can all be translated into illustrations. After a period of reviewing, collaboration, feedback and updating the definition of the application prototype should be established. Each task must go through a verification process to demonstrate that the required task has been completed properly. These documented requirements and diagrams become part of the reference points that map the direction of the project. During and after a project the documentation serves as a communication medium, information repository, provides information and tells users how to use and administer the software [59].

3.2.4 Requirement Specification and Agreement

This is the final main step of the Requirements Engineering Process. This step involves the formal finalizing and review of all requirements, and creation of the requirement models used to describe them. These diagrams can then be shared internally as guidelines for future development of the software, or externally to clients. Diagrams you could use to display these requirements include Entity-relationship model, and Function Decomposition diagrams. Moving forward, this document is referred to throughout the design of

the product, and can continue to be edited and changed as requirements are added or changed.

3.3 Requirements Documentation

This section will describe the User and System requirements for the application. The User requirements talk about our system in a higher level sense, giving an overview of the entire system. The system requirements will give specifics on exactly how different parts of the application function, and how it relates back to the system as a whole.

3.3.1 User Requirements

1. The system shall allow the user to create an account and login to the system.
2. The system shall allow the user to input courses.
3. The system shall allow the user to input degree plans.
4. The system shall allow the user to input student information.
5. The system shall store related data in a database management system.
6. The system shall allow storage of schedule rotation for upcoming semesters.
7. The system shall allow selection of the appropriate degree plan for each student.
8. The system shall be able to generate a report of courses still required for graduation
9. The system shall not remove major courses until a grade of C or higher is achieved.
10. The system shall generate a report that updates to reflect successfully completed courses

3.3.2 System Requirements

1. When the user logs in the system shall take the user to the main page which contains the following buttons: Students, Plans, Courses, Plan Courses, Student Courses, Reports, Browse, and Sign Out. Each respective button takes the user to the following specific areas.
2. The student page must accept specific input about each student so that each individual has a distinct profile that can later be used for database queries. The input fields will accept the unique identifier

Student ID and the student's designated degree plan can be selected. First and last name, year, and semester student enrolled in this plan. Each page will have a save button that submits input data to the database and a back button that returns the user to the main page. Information for a list of students can be entered as the screen will clear each time the save button is clicked.

3. The plans page must contain text fields to input the plan name, year, and hours. Each plan name will be the official catalog degree title. The user shall be allowed to input degree plan names such as Information Technology or Mathematics. Year will associate the plan entered with the catalog year and hours are the number of hours required to complete the degree plan. This page should also have a save button to save the information and a back button to return to the main page.
4. The courses page must contain text fields to input information like course code, course number, course name, and credit hours. Drop-down lists will be used to select when the course is offered (Fall, Spring or Both) and any prerequisites or corequisites for a particular course. This page will also have a save button to save the information and a back button to return to the main page.
5. The plan courses page must bring the data entered for plans and the data entered for courses together in order to complete the structure of each degree plan. This page will use drop down lists to associate a plan with a particular course. Another drop down list will be used to designate the plan course as either a major, core, or elective course. This page will also have a save button to save the information and a back button to return to the main page.
6. Next a student courses page must bring the data entered for students and the data entered for courses together to reflect the students completed courses. On the page text fields will be used to enter a specific student's ID number and year course was taken. Drop down lists will be use to select course, year, and grade. This page will also have a save button to save the information and a back button to return to the main page.
7. The reports page must allow selection of a specific student. This will done by text field and a student's ID number. When a student's ID number is entered clicking a generate report button will produce a report that list all courses student must complete to meet degree requirements.
8. A major Couse must remain on student's list of required classes until a grade of C' or higher is achieved.
9. Database queries will be use to filter report data so that only courses required to complete a degree plan will be displayed on the list.

10. Text tips, hover effects, field selection highlights will be used to guide users through system usage.

3.4 Requirement Models

This section contains Use-Case diagrams to show some of the use cases in our system. The diagrams will be generated with UML. As mentioned in section 2.5.3 there are 14 types of UML diagrams. We will be using use case diagrams in order to describe scenarios.

Figure 3.1 shows a use case for how a user can use the log-in page. It contains a graphical layout with three options available. First, existing users will be able to log-in with their user-name and password to access options on the main page. Second, new users are directed to the new user setup page. On the setup page a user-name, password, and email address will be required. Third, for existing users that have forgotten their password – an option to have the password emailed to them.

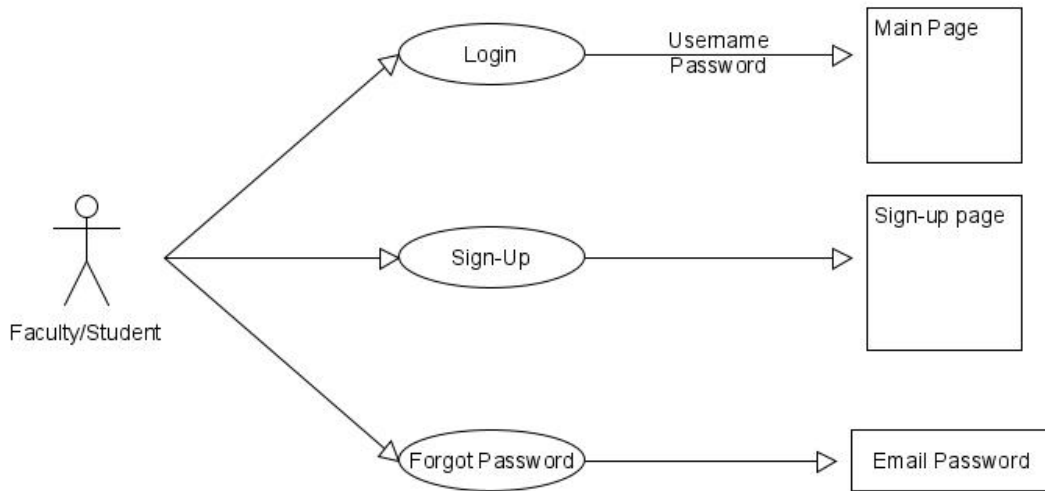


Figure 3.1: User Log-In Options.

Figure 3.2 represents when generating a report, the user will be able to input a student course schedule for a semester and compare that schedule to the course’s availability, confirming that it is available.

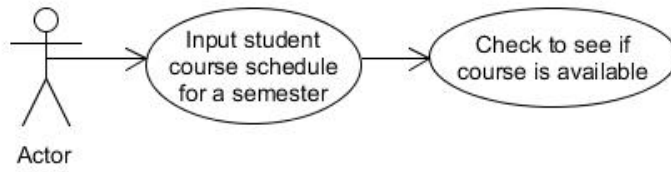


Figure 3.2: Check Course Availability

Figure 3.3 represents when the generate report button is pressed, after inputting a course schedule, the program will generate a report containing the classes still needed in order to graduate on time. It also shows the save button which will allow you to save the report and back button which will take you back to the main page.

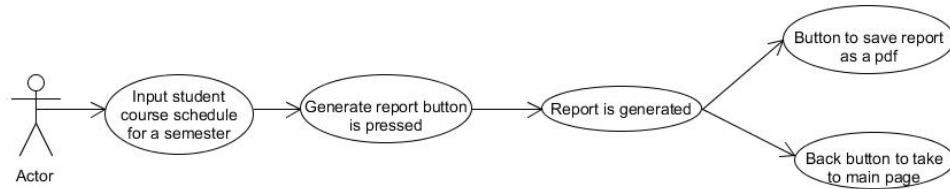


Figure 3.3: Generate Report

Figure 3.4 represents when the user generates report they will be shown that the courses are Texas Common Courses.



Figure 3.4: Texas Common Courses Report

Figure 3.5 represents when the user generates the report classes with higher failure rates are distinguished from the ones with high passing rates. As well as recommending the user courses with the higher pass rate if they are available for the semester.

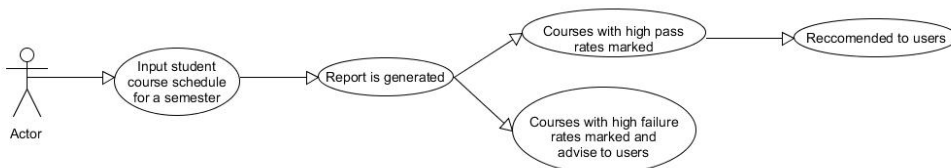


Figure 3.5: Course Pass Rate Report

3.5 Summary

In this chapter we defined what system requirements specifications are and all the steps involved in it. This chapter also provided the user and system requirements for our application by providing illustrations for each of those requirements. In the following chapter, we will introduce the design for our system.

Chapter 4

System Design

Objectives:

- Illustrate our system's database design.
 - Introduce the graphical user interfaces of our system.
 - Explain the navigation mechanism of our system.
 - Illustrate the class diagram for our system.
-

4.1 Introduction

In this chapter we will describe the design of the system. We will showcase the database design, as well as the GUI. We will also explain our system navigation, showing how one would use the software.

4.2 Database Design

In this section graphical layouts of the proposed database are presented as models that define the data to be stored and the relationships between that data. The database design process involves designing a Conceptual Model, Logical Model, and Physical Model to represent the database. The Conceptual Model organizes the basic entities of a system based on their relationships. The Logical Model is used to determine basic data types, as well as remove many to many relationships. And the Physical Model is used to assign data types that are specific to the database management system used, as well as to determine keys. These models provide quick reference diagrams for how the underlying information is organized. Each model progressively adds depth to the formation of the database structure.

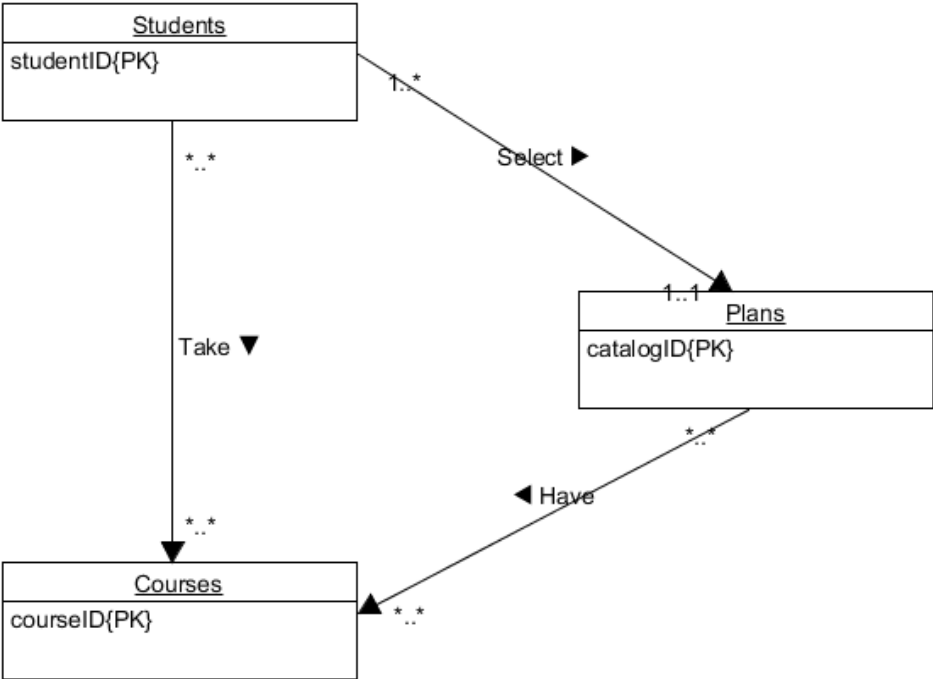


Figure 4.1: Conceptual Model

The conceptual model is a graphical demonstration of a database [60], the external main entities are named and connected to provide a user view of the database and is the simplest in terms of comprehension. The highest level is typically used to interact with business people or someone who is not technically versed in the database. The conceptual model is composed of concepts and ideas such as entities and relationships. We used the previously identified requirements to build the Conceptual Model shown in Figure 4.1 with the tables *Students*, *Plans*, and *Courses*. The real life concept of students taking courses became a relationship in the model. One *Student* will take many *Courses*. Many students select one type of degree plan. These *Plans* have *Courses* that must be completed. Selected entities later become tables in the database.

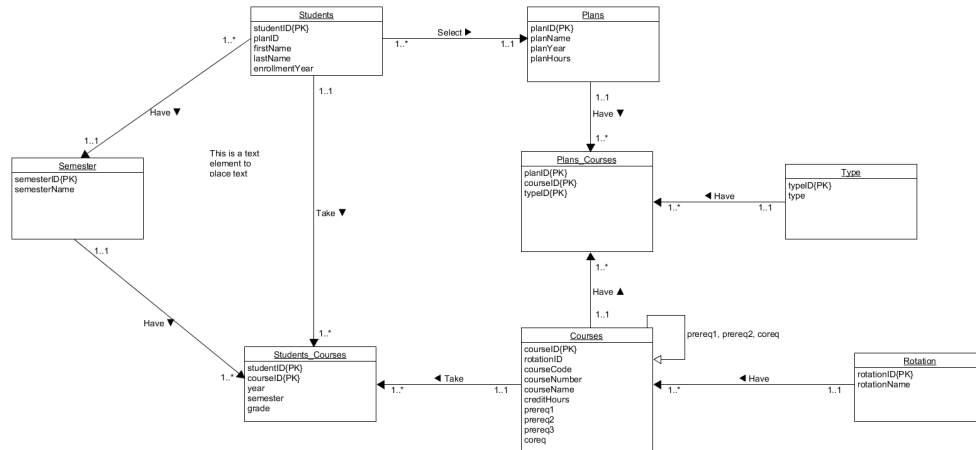


Figure 4.2: Logical Model

Next the Logical model, shown in figure 4.2, defines what internal data is required within each entity. For example each *Student* requires attributes like a unique ID, name and enrollment year. These details add columns to the tables and establish a clear accurate definition of each entity. The unique IDs are designated as primary keys to provide an exclusive reference points to each entity and foreign keys bind related entities together. The relationship between *Course* and *Plan* is established by adding *planID* as a foreign key to the course table. Recursive relationships are labeled with an arrow that originates and ends at the same table. A recursive relationship is used here when courses have prerequisites which are already on the *Course* table, so there was no need to create a separate table. As our database is a relational database we must remove any many-to-many relationships. That is why there were tables added in the logical when compared to the conceptual model. For example a table was created between *Students* and *Courses* called *Students_Courses* which stands to combine the two tables and remove the many to many relationship between them. In other cases in order to avoid redundant data entry or the need to update in multiple tables some entities

were divided into separate tables improving efficiency and normalizing the database structure. Instead of repeating fall, spring or both for each course in multiple location a separate *Rotation* table was created.

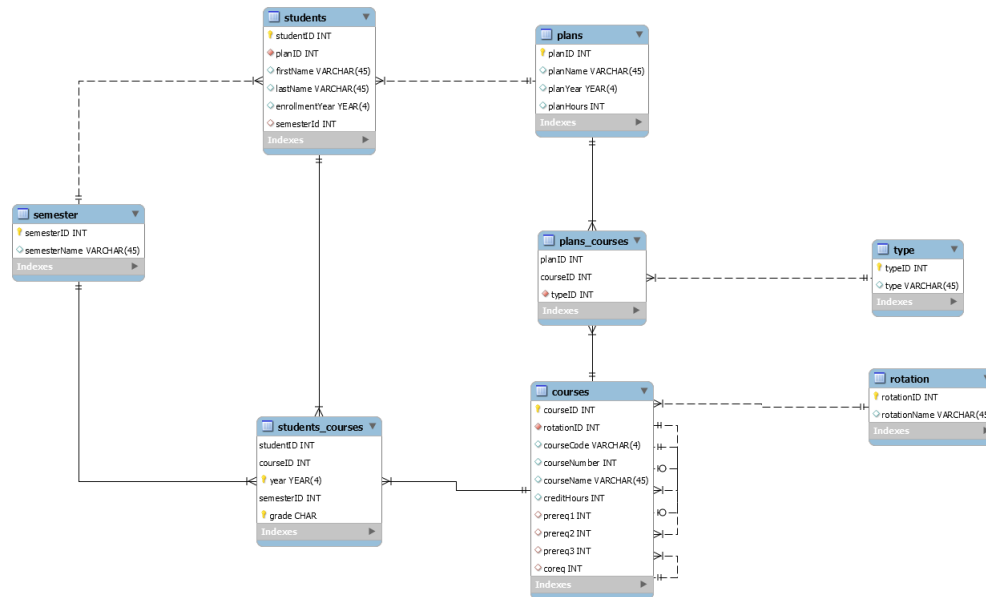


Figure 4.3: Physical Model

Finally the physical model, shown in figure 4.3, is a graphical representation of how the database will be implemented. Data types are added for each entity attribute. For example ID values are designated as INT and student names as VARCHAR(45). These data type selections affect the storage requirements for the database. In addition to data type, icons distinguish primary and foreign keys for each entity. Solid and dashed lines visually display cardinality and relationships between entities. The physical model reflects how all entities in the database are structured in order to work together, so that student degree plan information is available. These data models serves as reference points in the upcoming stage of database development, which is database implementation.

4.3 Graphical User Interface Design

This section details the graphical user interface of each page of our system.

Figure 4.4 shows the log-in page. The user is able to enter their Username and Password, then click login to enter the system. There is also a Sign Up button that can be pressed to take the user to the Sign Up page and create an account, as well as a Forgot Password link that can help the user retrieve their password.

Login to my College Class Scheduler

Username:

Password:

[Forgot Password?](#)

Figure 4.4: Log-in Page

Figure 4.5 shows the Main page, or the main menu of the system. It contains navigational buttons connecting to various parts of the system. There are buttons for: Student, Plans, Courses, Plan/Courses, Student/Courses, Reports and Browse. Each of these buttons will take you to their respective pages shown below. There is also a Sign Out button that is used to securely sign the user out and end the session.

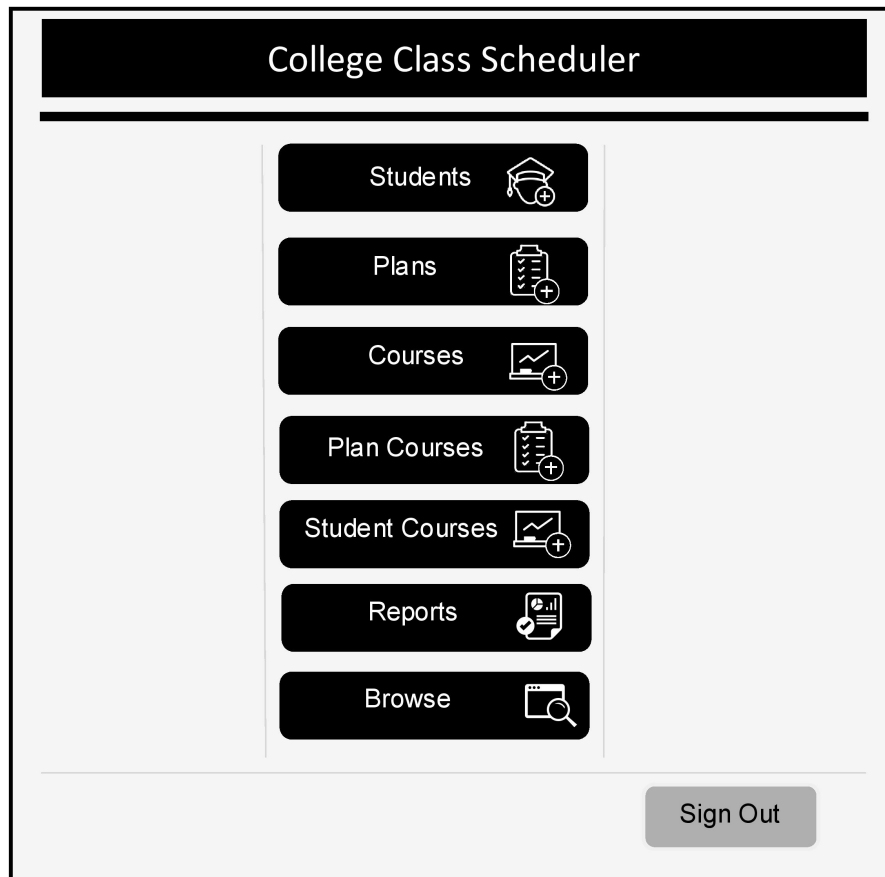


Figure 4.5: Main Page

Figure 4.6 shows the page the user will see after pressing the 'Student' button on the main page. This page allows the user to enter in student information into the database. The page will have a text box to input the student ID, a drop-down menu to select the plan they are on, and two text boxes to input the student's first and last name. The page will also have a text box to input what year the student is, and a drop-down menu selecting the student's semester. All the information on this page is required. After it is all entered the user can click save to add the student to the database and return to the main page, or the user can click back to cancel entering student information and return to the main page.

Students

Please enter and select the following then click save.

Students (Required)	
Student ID:	<input type="text"/>
Plans:	<input type="text" value="▼"/>
First Name:	<input type="text"/>
Last Name:	<input type="text"/>
Year:	<input type="text"/>
Semester:	<input type="text" value="▼"/>

Back

Save

Figure 4.6: Students Page

Figure 4.7 shows what the user will see on the plan page after selecting the button 'Plans' on the main page. The purpose of the page is to allow the user to select a plan that suits them best. What is shown will be a form which will contain text-fields that are required to fill out. The top text-field is labeled "Plan" which the user will input the plan which they require to use. The second text-field is labeled "Year" which will allow the user to input the year of the plan they have selected. The final text field is labeled "Hours"

which will contain the course hours of the chosen plan. It also contains a 'Save' button that will give the user the option to save the plan they have selected and a 'Back' button so the user can return the previous page they had accessed.

Plans

Please enter the following then click save.

Plans (Required)	
Plans:	<input type="text"/>
Year:	<input type="text"/>
Hours:	<input type="text"/>

Figure 4.7: Plans Page

Figure 4.8 shows the page the user will see after pressing the 'Courses' button on the main page. This page is for entering course information into the database. There are four required fields, as well as three optional ones. Firstly there are three text box for entering the course name, number, and credit hours. There is also a drop-down menu to select what term the course takes place. Finally there are three optional drop-down menus that allows the user to select up to two prerequisite courses, as well as one co-requisite one. After all of the course information has been entered, the user can either choose save to input the data into the database and return to the main page, or back to cancel inputting the information and return to the main page.

Courses

Please enter and select the following then click save.

Courses (Required *)	
Course Code:	<input type="text"/>
Course Number:	<input type="text"/>
Course Name:	<input type="text"/>
Credit Hours:	<input type="text"/>
Rotation:	<input type="text"/> ▼
Prerequisite 1:	<input type="text"/> ▼
Prerequisite 2:	<input type="text"/> ▼
Co-requisite:	<input type="text"/> ▼

Back

Save

Figure 4.8: Course Page

Figure 4.9 shows what the user will see if they click the “Plan/Courses” button on the Main page. The purpose of this interface is to combine data from Plans and Courses, thereby building complete degree plan structures. First on this GUI is a drop down list that allows the user to select which Plan ID they want to add information to. After selecting a Plan ID the next option is Course ID. The Course ID drop down list connects a course to the selected plan. With the last drop down list, Type ID, one category (major,

core or elective) can be designated for each course selected. When the Save button is clicked- selected plan information will be saved into the database. Duplicate entries will not be saved and a message will alert users if a particular entry has already been created. To exit this page, the user can click the “Back” button which returns them to the Main page.

Plan Courses
(Required)

Plan:

Course:

Type:

Back **Save**

Figure 4.9: Plan/Courses Page

Figure 4.10 shows what the user will see if they click the 'Student Courses' option in the Main page. The user will see a form that contains text-fields and drop down lists for user input. The first text-field will be "Student ID" which the user will be prompted to input there ID. Following that it will be a drop down list labeled "Course ID" which will allow the user to select a course. The second text field is labeled "Year" which the user will input the year of the chosen course. The following two drop down lists will be "Semester" and "Grade" which the user will have the option to select from the lists given. The page also contains a 'Back' and 'Save' button. The 'Save' button will save the progress the user has done on the form and the 'Back' button will take the user to the previous page.

Student Courses

Please enter and select the following then click save.

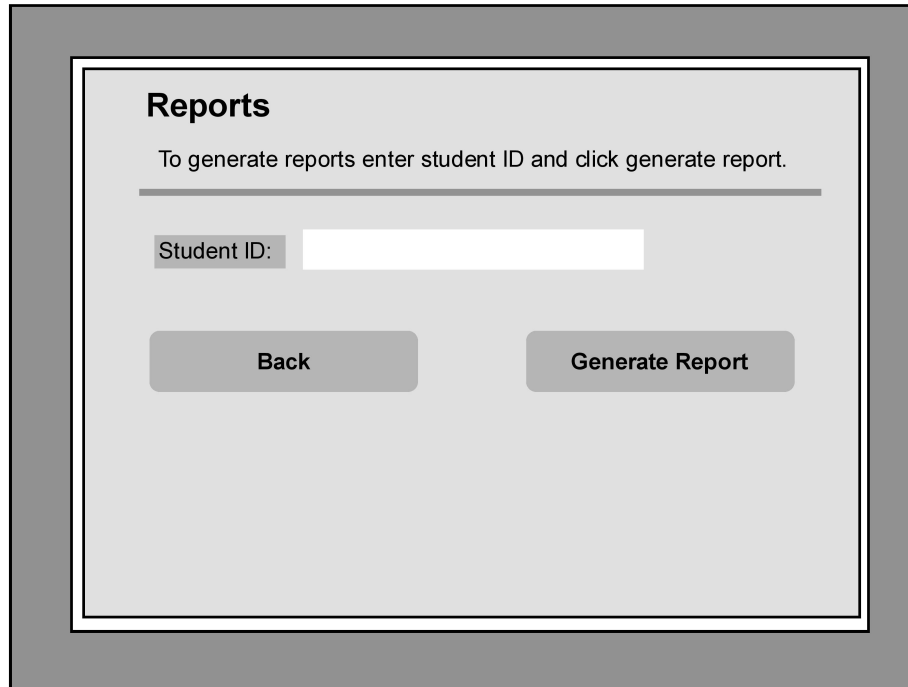
Student Courses (Required)	
Student ID:	<input type="text"/>
Course:	<input type="text" value="▼"/>
Year:	<input type="text"/>
Semester:	<input type="text" value="▼"/>
Grade:	<input type="text" value="▼"/>

Back

Save

Figure 4.10: Student Courses Page

Figure 4.11 displays what the user will view when the user clicks on "Reports" button. It takes the user to the reports page where the user will input the student ID. After inputting the student's ID, the user can generate the degree summary report by clicking "Generate Report" button. It also gives the user the option to click on "Back" button which will take the user to the Main page.



The screenshot shows a web page titled "Reports". Below the title is a subtitle: "To generate reports enter student ID and click generate report." There is a horizontal line below the subtitle. Underneath the line is a label "Student ID:" followed by a white text input field. Below the input field are two buttons: "Back" on the left and "Generate Report" on the right. The entire content is enclosed in a light gray border.

Figure 4.11: Generate Report

4.4 System Navigation

This section shows a model of the relationships between each system component and how those components work together to take information input, process it and output reports. Double-sided arrows (\leftrightarrow) represent the ability to move back and forth between application pages. For example, clicking the "Login" button on the Log-in page will bring up the Main page. Clicking the "Back" button on the Main page will go back to the Log-in page. In addition to navigating between pages, buttons are used to perform functions like save information and generate reports. Some information will be manually entered into text fields. Drop down list contain common options that need to be readily available in the system. Some data will automatically

calculate to avoid manual entry of information where possible. Each page of the application handles specific portions of student degree plan information. Degree information can then be summarized and presented in various manners by selecting an option on the Reports page. These reports can be printed or email to students. The system diagram below is a graphical layout of expected system behavior. Figure 4.12 shows College Class Scheduler System Diagram.

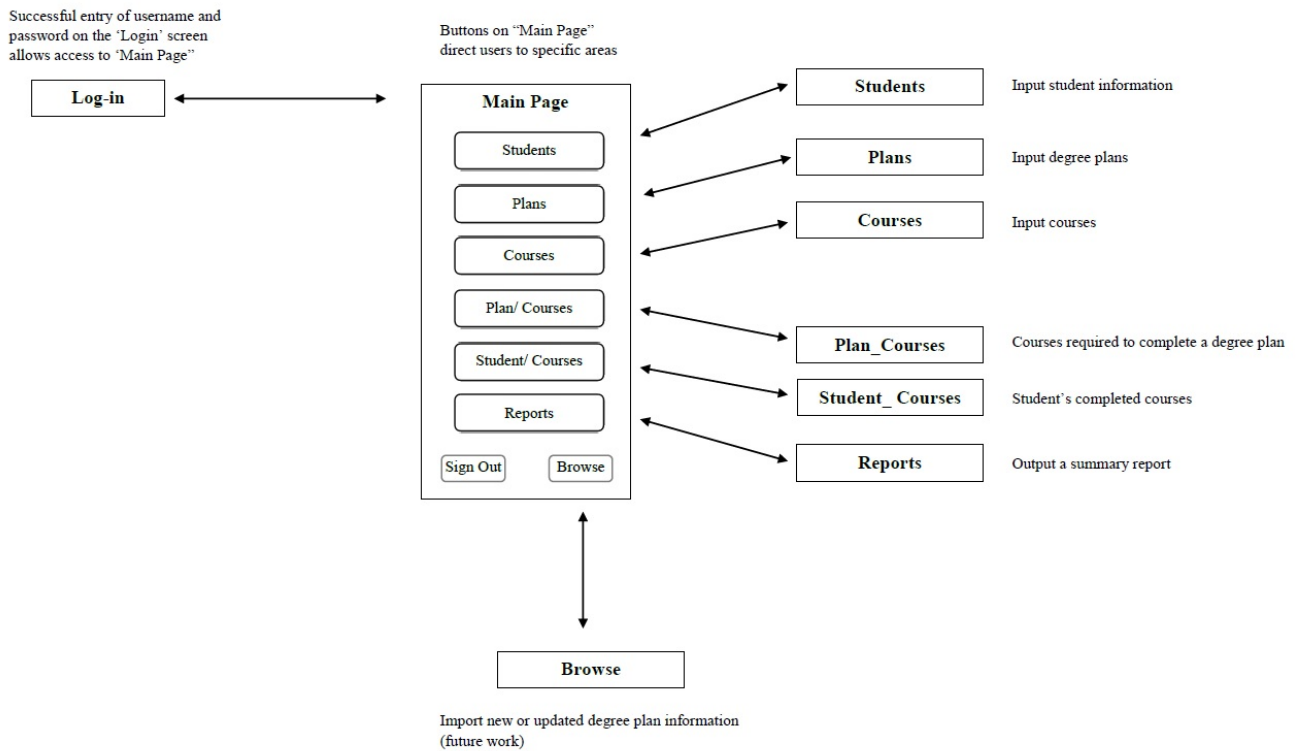


Figure 4.12: System Diagram

4.5 Summary

This chapter represented the steps of our database design and explained the data models of our database. It explained the GUI of each part of our software. We also walked through how to navigate the system through the System Navigation diagram.

The next chapter will go over the code used to develop our system.

Chapter 5

Implementation and Testing

Objectives:

- Present a working version of the new application.
 - Demonstrate how development tools were utilize to implemented each GUI.
 - Provide the code used to implement each GUI.
 - Present testing scenarios that demonstrate some capabilities of the new system.
-

5.1 Introduction

This chapter will present screenshots from the newly developed application *College Class Scheduler*. Each screenshot represents a graphical user interface (GUI) that was developed to provide functionality in the system. Along with images the code required to create each GUI will also be presented. Which development tools and how they were used will be included. In the last section of this chapter some testing scenarios are presented. Description are provided for each test condition.

5.2 GUI and Explanations

This section will cover each part of our GUI. and our database, as well as the code needed to create it.

Login Page:

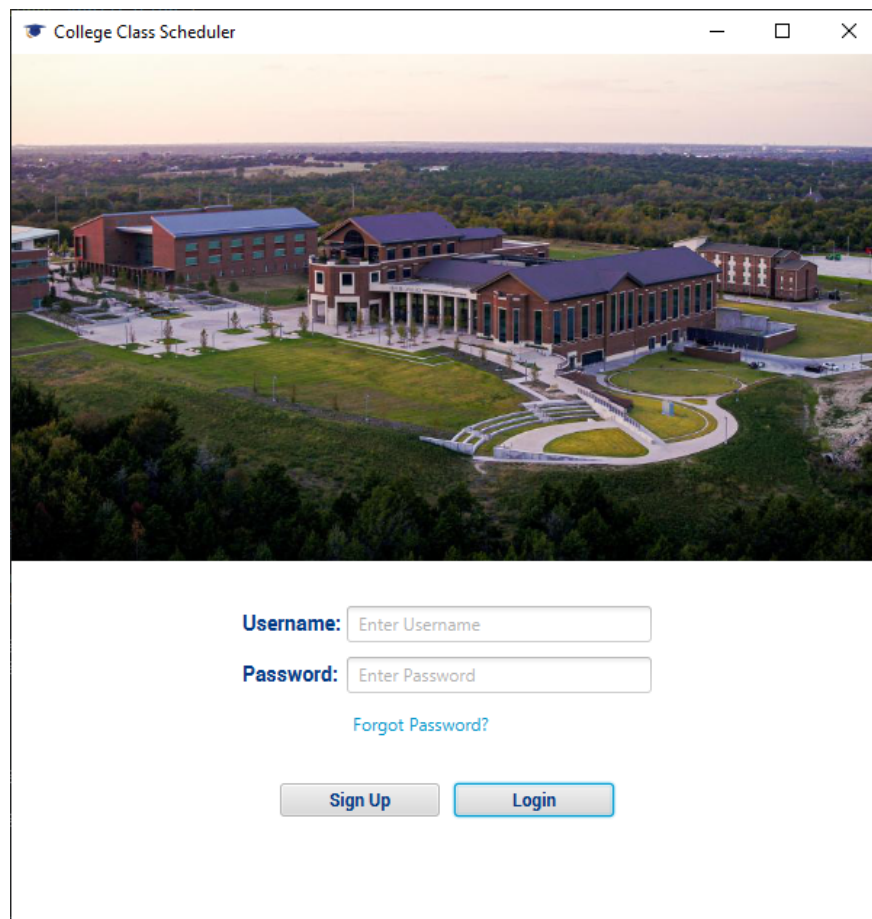


Figure 5.1: Login Final GUI

Figure 5.1 shows the Login Page. This page allows the user to enter a username and password to log in to the system. There are also buttons to sign up, and for forgotten password (currently unimplemented). The code for this page's implementation can be found in Appendix B.1. Image credited to the UNTD Website[61]

Main Page:

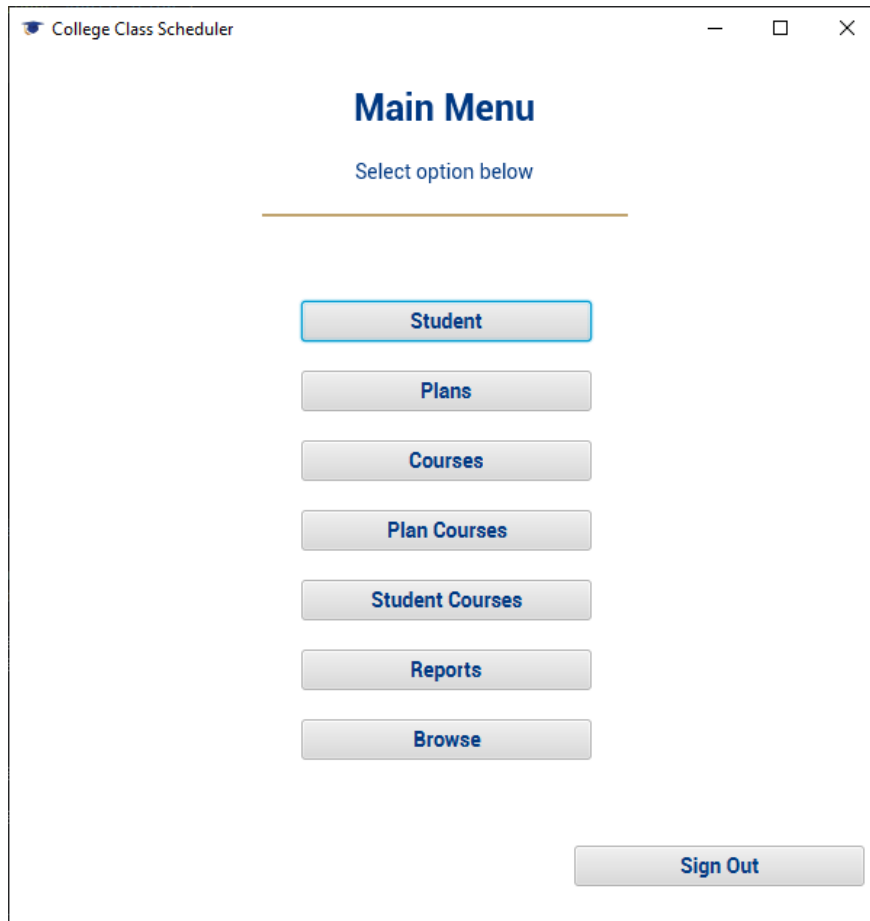


Figure 5.2: Main Final GUI

Figure 5.2 shows The Main Menu for the system. There are buttons for each part of the system, and also the log out button. The code for this page's implementation can be found in Appendix B.2.

Student Page:

College Class Scheduler

Students

Please enter the following student information then click save.

Student ID:

Plans:

First Name:

Last Name:

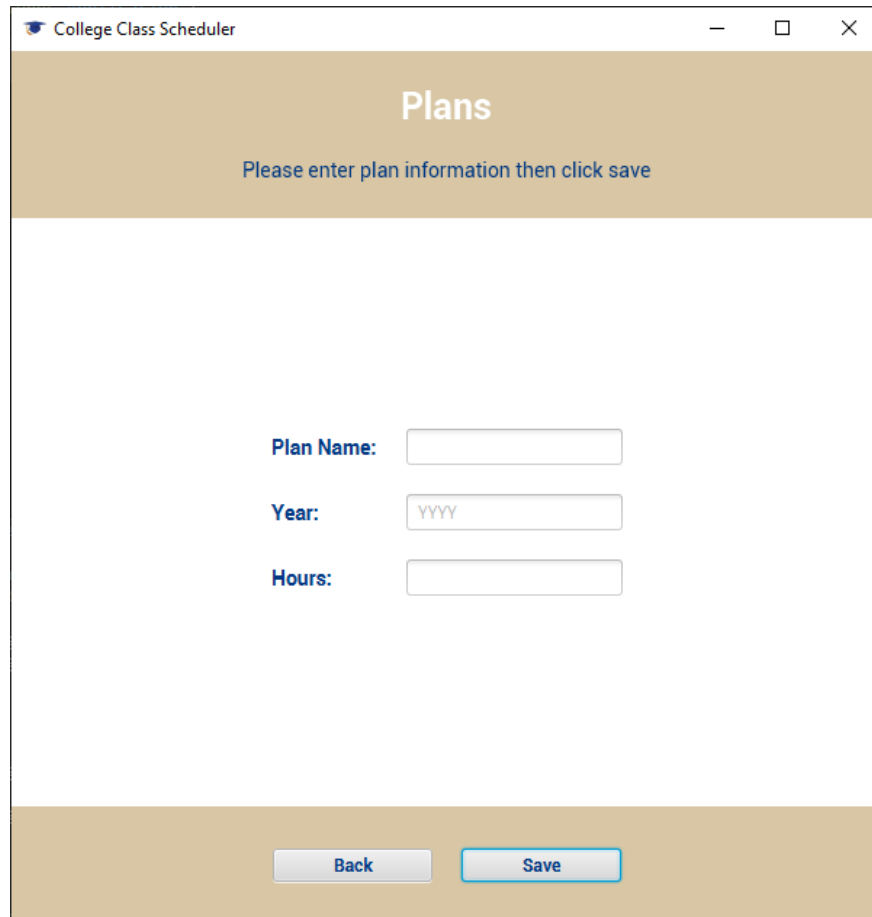
Year:

Semester:

Figure 5.3: Student Final GUI

Figure 5.3 shows the Student Page. This page allows the user to input student information into the database. The code for this page's implementation can be found in Appendix B.3.

Plans Page:



College Class Scheduler

Plans

Please enter plan information then click save

Plan Name:

Year:

Hours:

Figure 5.4: Plans Final GUI

Figure 5.4 shows the Plans Page. This page allows the user to input plan information into the database. The code for this page's implementation can be found in Appendix B.4.

Courses Page:

College Class Scheduler

Courses

Please enter course information then click save.

Course Code:

Course Number:

Course Name:

Credit Hours:

Rotation:

Prerequisite 1:

Prerequisite 2:

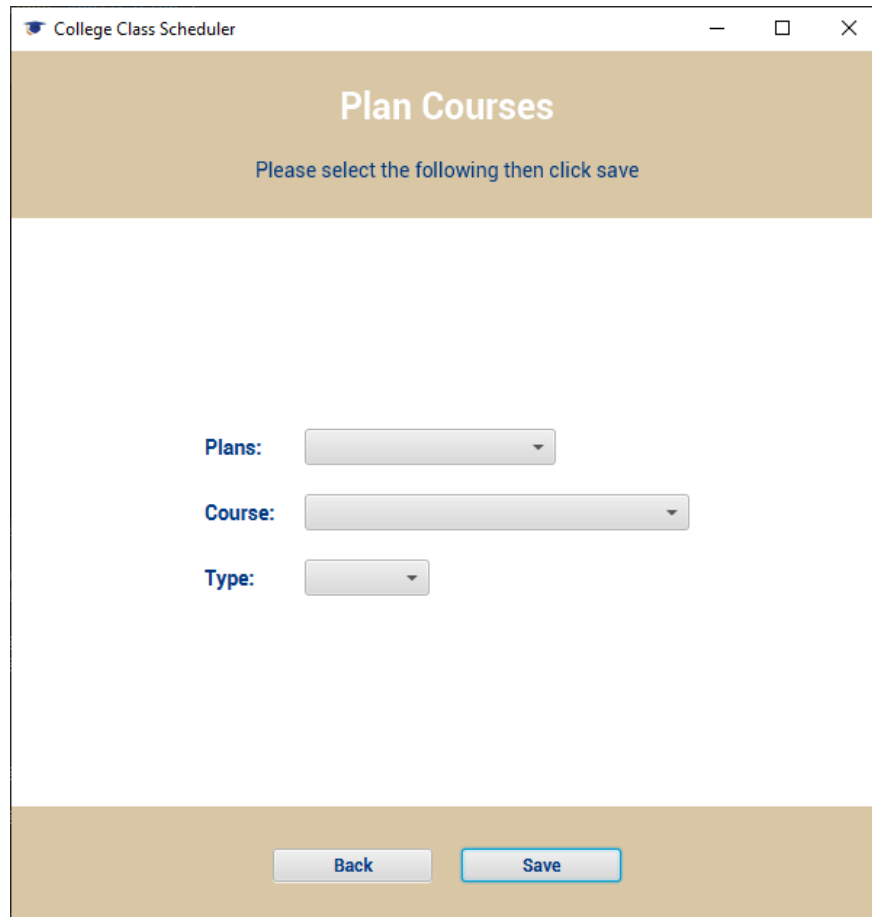
Prerequisite 3:

Co-requisite:

Figure 5.5: Courses Final GUI

Figure 5.5 shows the Courses Page. This page allows the user to input course information into the database. The code for this page's implementation can be found in Appendix B.5.

Plan Courses Page:

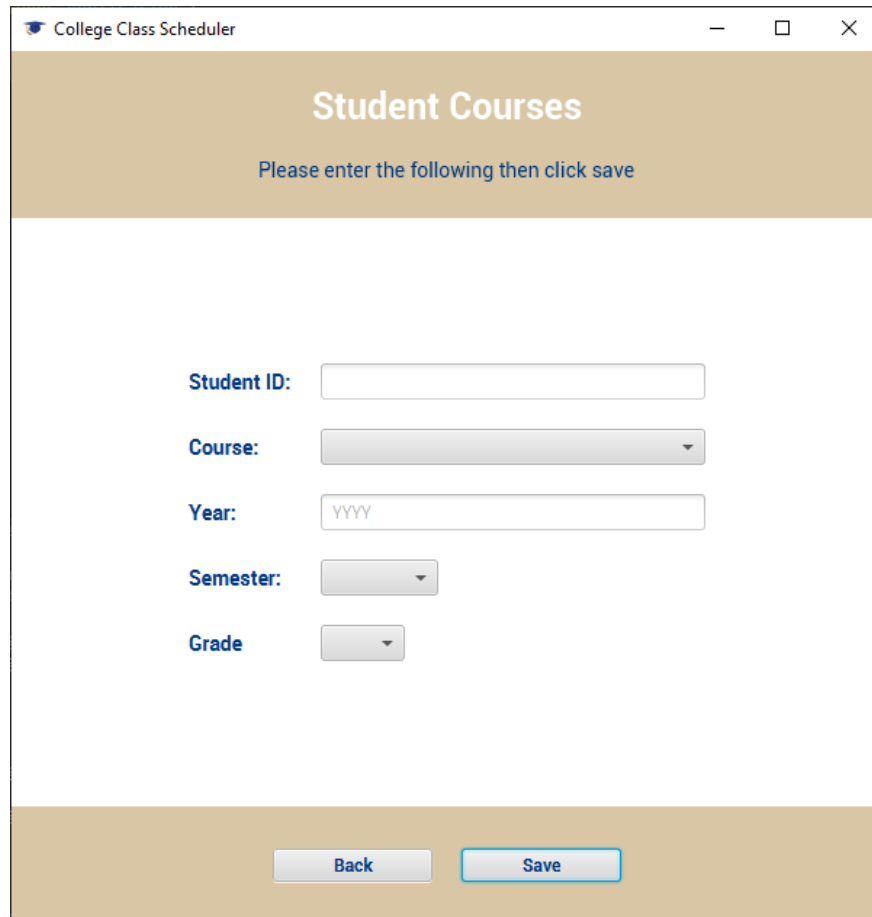


The screenshot shows a web browser window titled "College Class Scheduler". The page has a light brown header with the text "Plan Courses" and a sub-header "Please select the following then click save". Below this, there are three dropdown menus labeled "Plans:", "Course:", and "Type:". At the bottom of the page, there are two buttons: "Back" and "Save".

Figure 5.6: Plan Courses Final GUI

Figure 5.6 shows the Plan Courses Page. This page allows the user to link courses to specific plans. The code for this page's implementation can be found in Appendix B.6.

Student Courses Page:



The screenshot shows a web application window titled "College Class Scheduler". The main heading is "Student Courses" in a large, bold font. Below the heading, a blue instruction reads "Please enter the following then click save". The form contains five input fields: "Student ID" (a text box), "Course" (a dropdown menu), "Year" (a text box with "YYYY" as a placeholder), "Semester" (a dropdown menu), and "Grade" (a dropdown menu). At the bottom of the form, there are two buttons: "Back" and "Save".

Figure 5.7: Student Courses Final GUI

Figure 5.7 shows the Student Courses Page. This page allows the user to mark which courses any student has taken. The code for this page's implementation can be found in Appendix B.7.

Reports Page:

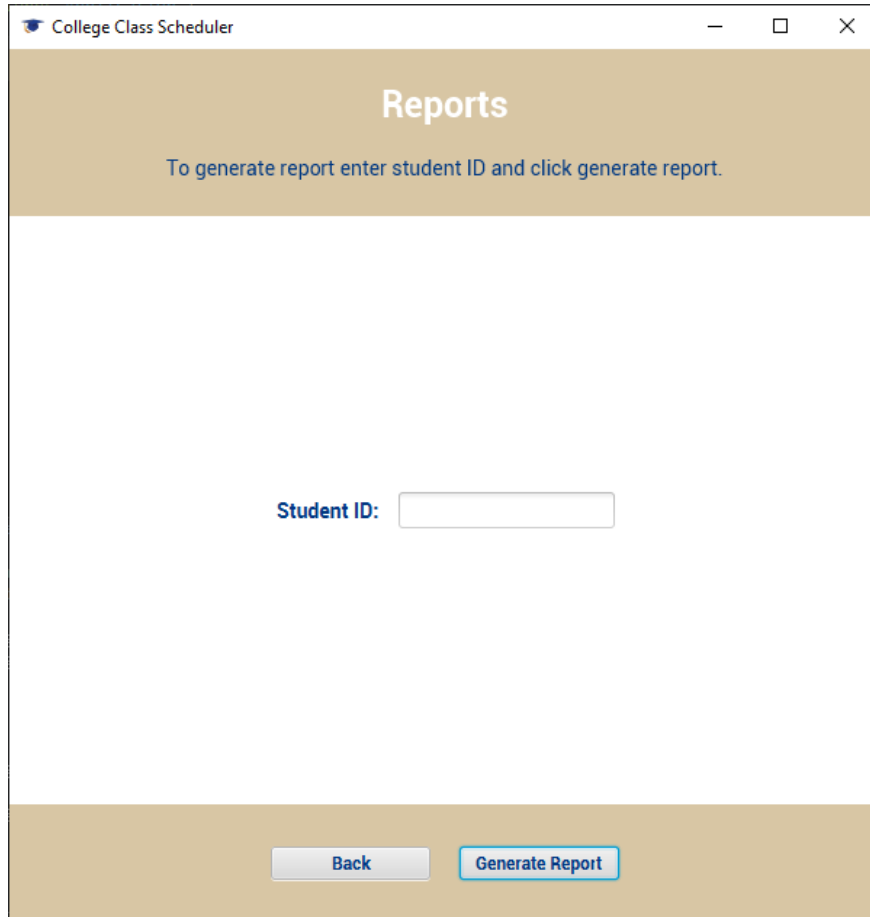


Figure 5.8: Reports Final GUI

Figure 5.8 shows the Reports page. This page allows the user to see what courses remain for any student's plan based on what courses they completed. The code for this page's implementation can be found in Appendix B.8.

5.3 Testing Scenarios

Test Scenario 1: Functionally

In this scenario user logins were tested to verify login ability. An unsuccessful login results in an Incorrect username or password alert.

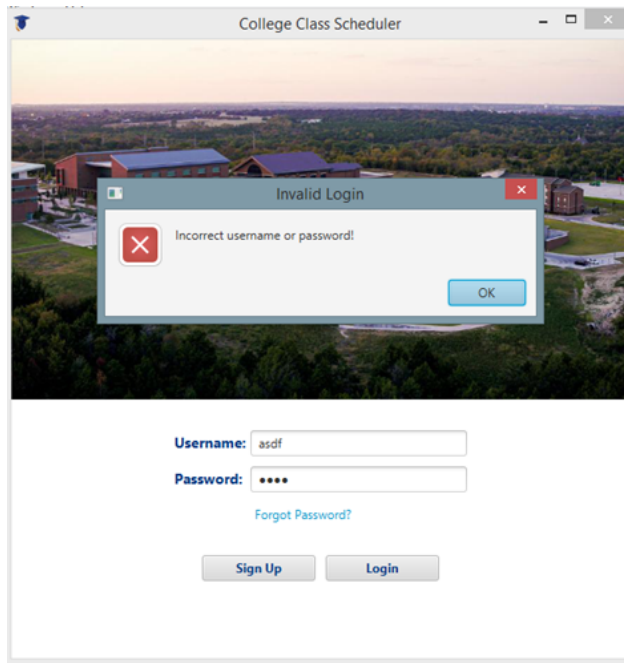


Figure 5.9: Login test Incorrect password

A successful login presents the user with the **Main** page where they can select specific areas to enter data as needed.

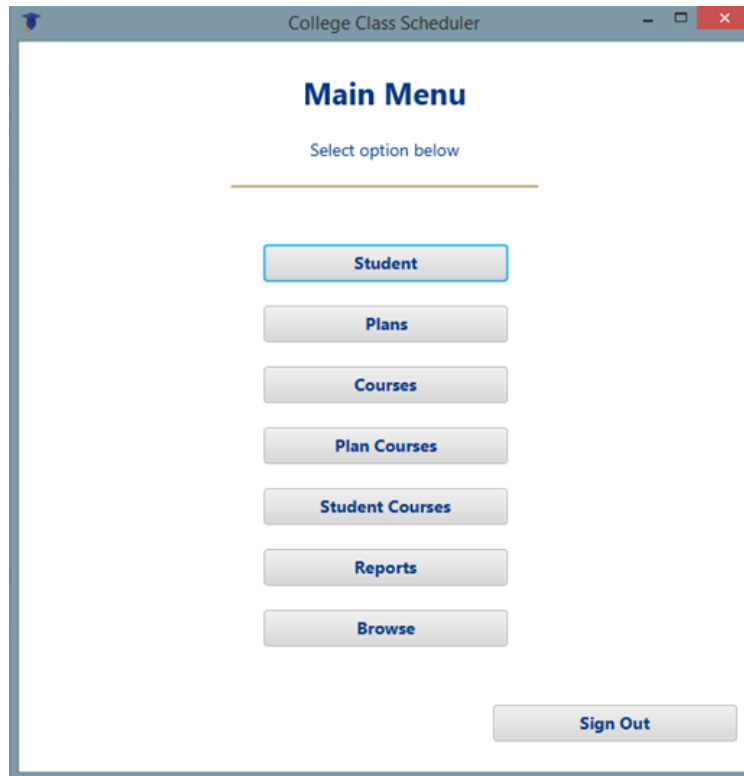
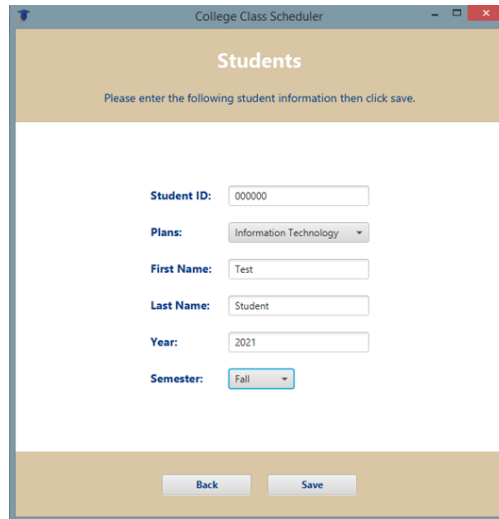


Figure 5.10: Main Page Functionally Test

Navigating down the list from the top is the first option **Students**. Clicking this button presents the Student page where basic student information can be entered into the system.



College Class Scheduler

Students

Please enter the following student information then click save.

Student ID:

Plans:

First Name:

Last Name:

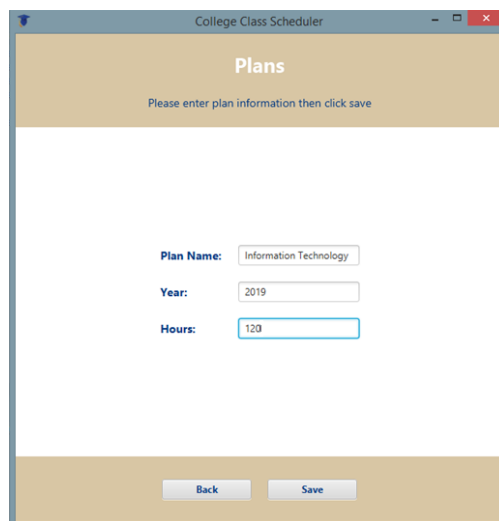
Year:

Semester:

Figure 5.11: Students Page Functionally Test

Clicking the **Save** button will enter the data into the database and clear the fields in preparation for the next entry. On each page clicking the **Back** button will return the user to the Main page where they can make another selection.

The next option is **Plans**. Here various degree plans can be entered - the name of the degree plan, the catalog year, and how many hours it will take to attain this degree.



College Class Scheduler

Plans

Please enter plan information then click save

Plan Name:

Year:

Hours:

Figure 5.12: Plans Page Functionally Test

The Courses button presents the **Courses** screen. Here detailed information about courses offered can be entered into the text field. The drop down lists provide convenient access to reusable lists of data that is stored in the database. Courses that are already in the database will not be saved as duplication is not allowed.

College Class Scheduler

Courses

Please enter course information then click save.

Course Code:

Course Number:

Course Name:

Credit Hours:

Rotation:

Prerequisite 1:

Prerequisite 2:

Prerequisite 3:

Co-requisite:

Figure 5.13: Courses Page Functionally Test

College Class Scheduler

Courses

Please enter course information then click save.

Course Code:

Course Number:

Course Name:

Credit Hours:

Rotation:

Prerequisite 1:

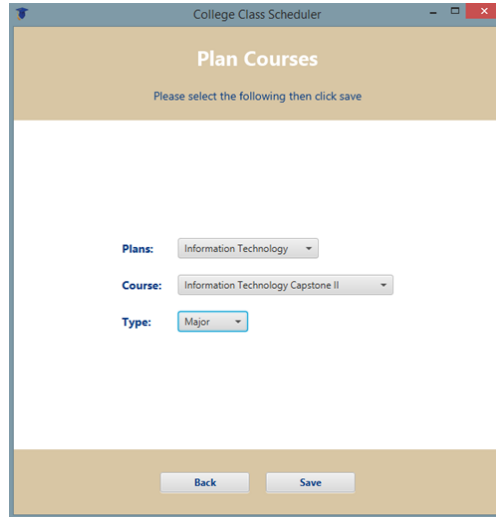
Prerequisite 2:

Prerequisite 3:

Co-requisite:

Figure 5.14: Courses Page Drop Down List Test

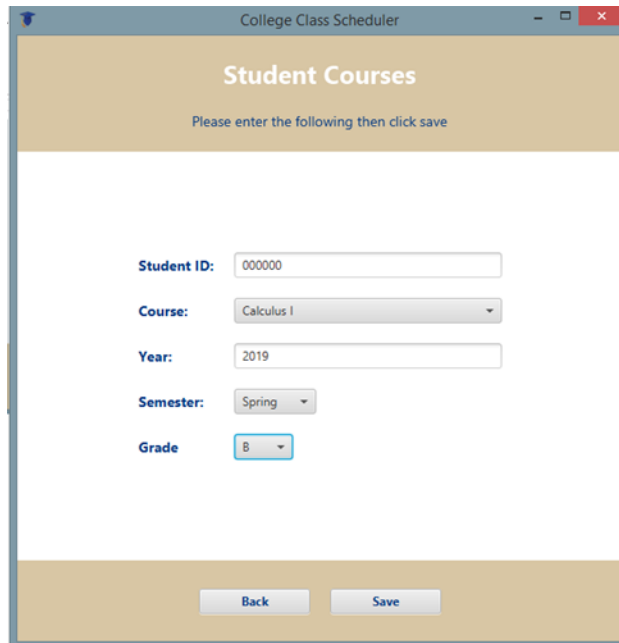
The **Plan Courses** button presents a screen where previously entered Plans and Courses are combined to build a complete degree plan in the database.



The screenshot shows a web application window titled "College Class Scheduler". The main heading is "Plan Courses" with a sub-instruction: "Please select the following then click save". Below this, there are three dropdown menus: "Plans:" with the value "Information Technology", "Course:" with the value "Information Technology Capstone II", and "Type:" with the value "Major". At the bottom of the form, there are two buttons: "Back" and "Save".

Figure 5.15: Plan Courses Functionally Test

Likewise the **Student Courses** button presents a screen that links students with their completed courses and the grade they received.



The screenshot shows a web application window titled "College Class Scheduler". The main heading is "Student Courses" with a sub-instruction: "Please enter the following then click save". Below this, there are five input fields: "Student ID:" with the value "000000", "Course:" with the value "Calculus I", "Year:" with the value "2019", "Semester:" with the value "Spring", and "Grade" with the value "B". At the bottom of the form, there are two buttons: "Back" and "Save".

Figure 5.16: Student Courses Functionally Test

Next the **Reports** button where all the data entered previously is brought together. After typing the Student's ID number in the text field, click the Generate Report button. This will present a list of classes that are still required in order for this particular student to complete their degree plan requirements.

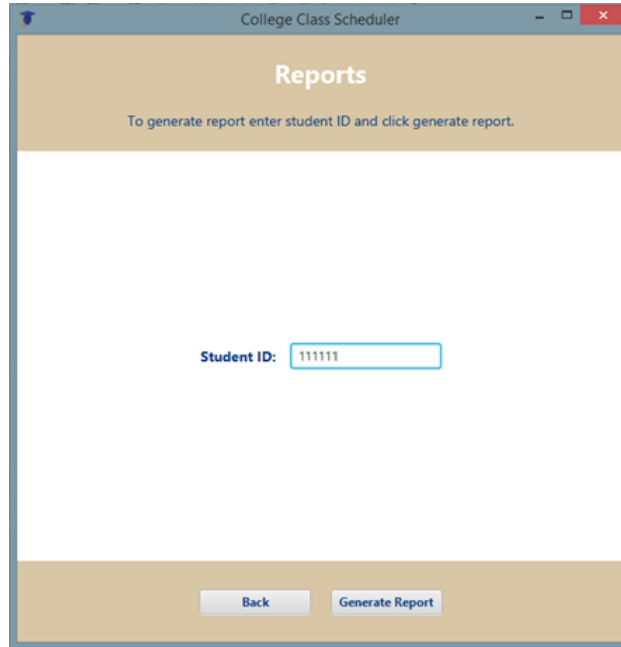


Figure 5.17: Reports Page Functionally Test

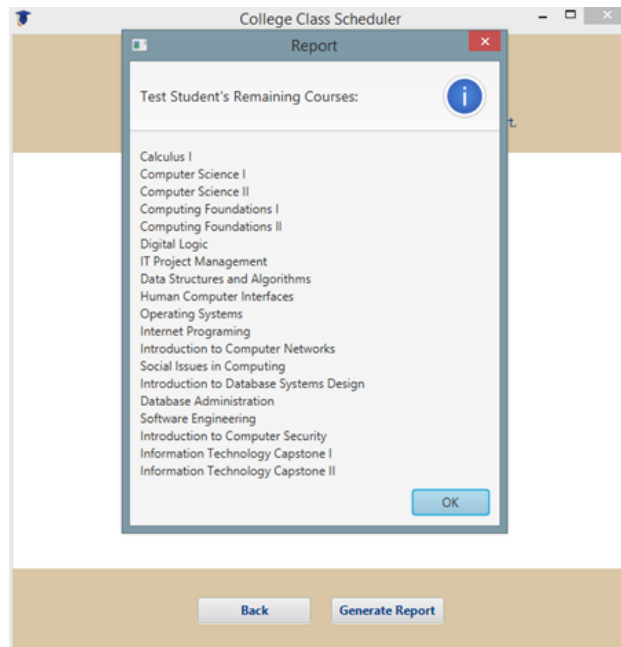


Figure 5.18: Reports Page Functionally Test

When the user is ready to close the College Class Scheduler from the Main page click the **Signout** button which exits the application and returns the user to the login screen.

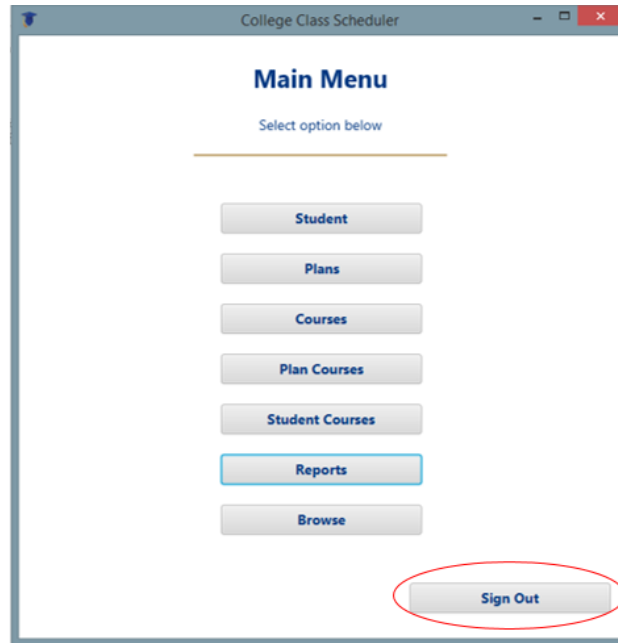


Figure 5.19: Signout Functionally Test

For each of the selection options on the Main page- each text field and drop down list was tested and verified to save the submitted data to the database.

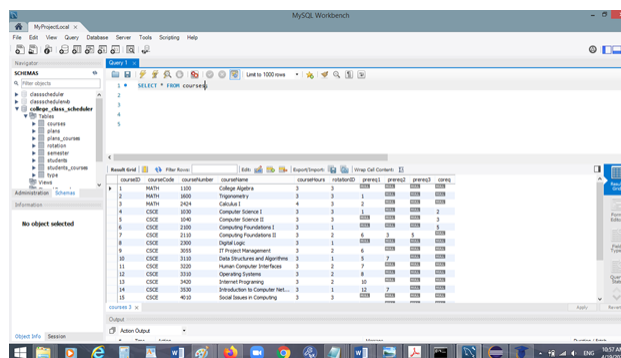


Figure 5.20: Data saved to MySQL Database Functionally Test

Test Scenario 2: Reports

All data entered works together to provide a summary of what classes are still required for a student. All major courses require a grade of C or higher otherwise these courses will remain on the list of courses to be completed.

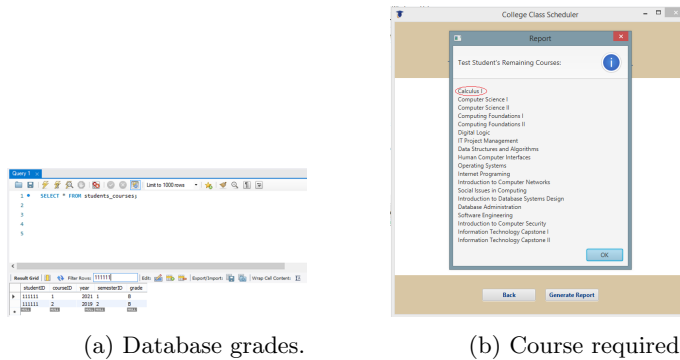


Figure 5.21: Grading system course required.

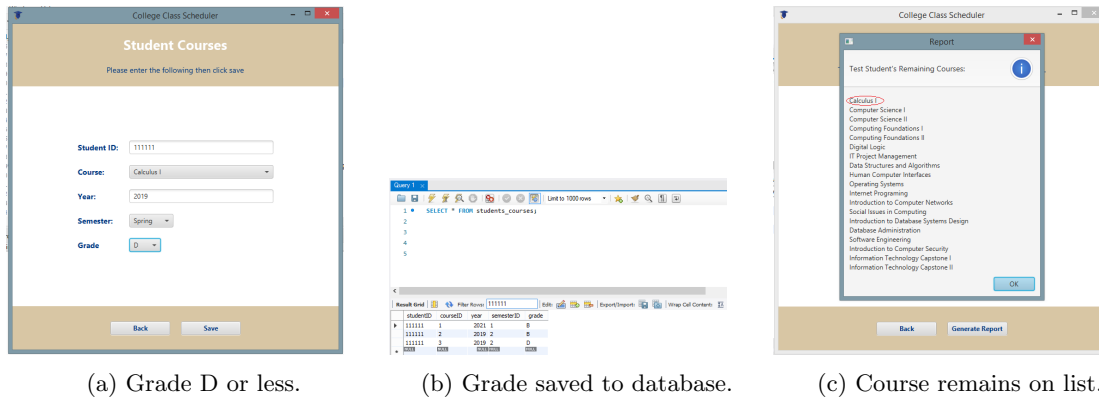
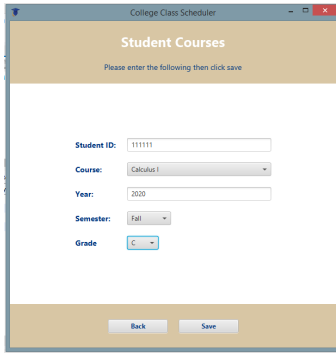
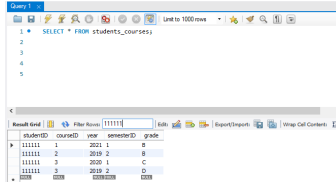


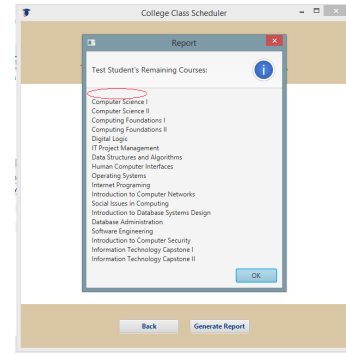
Figure 5.22: Major course remains on list until grade of C or higher..



(a) Grade "C" or higher.



(b) Grade saved to database.



(c) Course removed from list.

Figure 5.23: Major course removed from list after passing grade.

Test Scenario 3: Presentation

Buttons, images, and text fields are centered on the application so they remain in place as the screen size changes.

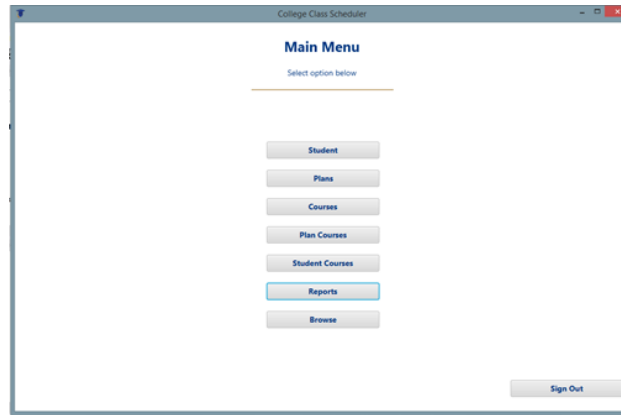


Figure 5.24: Screen Resize Presentation Test

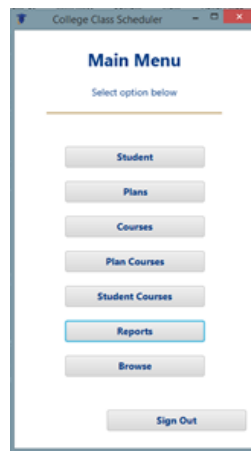


Figure 5.25: Screen Resize Presentation Test

Pages are formatted to have consistent font types and colors. Font sizes and bold text are used to enhance readability. Buttons, text fields, and drop down lists are controlled by size properties to maintain similar proportions on each page.



Figure 5.26: Field Selection Color Change Test

Each text field changes color when clicked to indicate the field is selected and ready for user input.

Drop down list have a hover effect when the mouse passes over them to indicate they are clickable for more options.

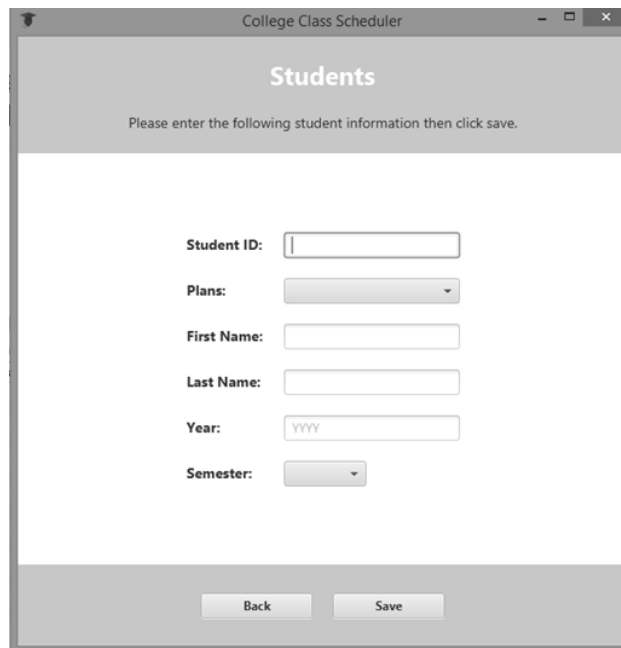


Figure 5.27: Grayscale Presentation Test

Each screen was viewed in grayscale to verify readability of text prompts from a different perspective.

5.4 Summary

This chapter presented the new application *College Class Scheduler*. The component graphical user interfaces were presented in order to demonstrate the variety of functionally linked together to create this new application. Code used to create each GUI was presented. Testing scenarios provided further detailed insight into how the built-in features of the *College Class Scheduler* perform.

Chapter 6

Conclusion and Future Work

Objectives:

- Describe the chapters of the system's proposal report.
 - List future work for the system.
-

6.1 Conclusion

This concludes the report presenting the college class scheduler. This degree planning application was designed to be used as a desktop platform. This system is intended for faculty use to automate the process of scheduling classes. The following is a summarization of all the previous chapters:

- Chapter 1 provided an introduction to the background of the proposed system. It outlined and defined the proposed system by each section.
- Chapter 2 provided examples of related systems that were similar to the application. It outlined and highlighted the selected tools that were used to implement the application.
- Chapter 3 provided a brief definition of the Requirement Engineering Process and its implementation. It defined the user and system requirements. This chapter showcased the requirements of the system using the models needed and explained the system flow with diagrams.
- Chapter 4 introduced the graphical user interface of the system. This chapter showcased the system's database design, class diagram, and explained the navigation mechanism.
- Chapter 5 presented the implementation of the system. It provided the code needed for each graphical user interface and demonstrated the utilization of the tools on each GUI. The systems functionality was proved by providing testing scenarios.

6.2 Future Work

Without the constraint of time the team would have implemented the following features to help improve the application.

- Sign Up – The team wanted to implement a sign up page that would allow the user to create an account. By creating an account, the user's course work information would be saved for future use.
- Browse – The team wanted to implement a Browse button that would allow the user to automatically import a student's report to automate the data entry.
- Report – The team wanted to add an additional feature that once the report was generated it would divide the courses into different semesters based on the courses remaining for each user.

- Texas Common Course Numbering (TCCN) – The team wanted to add Texas Common Course Numbering system (TCCN) to the courses. The system would cross-reference and mark all transferable credits from other institutions to UNTD.

Appendix A

Paper Degree Plan Example

The documents below are the current IT degree requirements as of 2019.



2019-2020 UNT Dallas Catalog Degree Requirements:

Information Technology (Bachelor of Arts)

Use this checklist as a guide for selecting classes and refer to your online Academic Advisement Report to review and monitor degree and graduation requirements.

NAME: _____

UNTD ID Number: _____

Core Curriculum Recommendations – 42 Hours

Core Foundation Area	TCCNS #	UNTD #	Course	Hrs	Grade	Course/Term
010	Communication	ENGL 1301	ENGL 1313	College Writing I *(C or better required)	3	
010	Communication	ENGL 2311	TECM 2700	Technical Writing *(C or better required)	3	
020	Mathematics	MATH 1314	MATH 1100	College Algebra	3	
030	Life & Physical Sciences	CHEM 1411	CHEM 1410	General Chemistry I	3	
030	Life & Physical Sciences	CHEM 1412	CHEM 1420	General Chemistry II	3	
040	Language, Philosophy, & Culture	Varies	Varies	Any Course in Foundation Area	3	
050	Creative Arts	Varies	Varies	Any Course in Foundation Area	3	
060	American History	HIST 1301	HIST 2610	US History to 1865	3	
060	American History	HIST 1302	HIST 2620	US History 1865 to Present	3	
070	Government & Political Science	GOVT 2305	PSCI 1040	American Government: Laws & Institutions	3	
070	Government & Political Science	GOVT 2306	PSCI 1050	American Government: Process & Policy	3	
080	Social & Behavioral Science	Varies	Varies	Any Course in Foundation Area	3	
090	Component Area Option	Varies	CHEM 1430/40	CHEM labs	2	
090	Component Area Option	Varies	Varies	Any Course in Foundation Area	4	

Major Curriculum Requirements – 59 Total Hours

**Must maintain advanced CSCE course GPA of 2.75 or better*

IT Major (58 Hours)

TCCNS #	UNTD #	Course Name	Hrs	Grade	Course/Term
COSC 1436	CSCE 1030	Computer Science I	*Prerequisite MATH 1100	3	
COSC 1437	CSCE 1040	Computer Science II	*Prerequisite CSCE 1030	3	
UNTD only	CSCE 2100	Computing Foundations I	*Co-requisite CSCE 1040	3	
UNTD only	CSCE 2110	Computing Foundations II	*Prerequisite CSCE 2100	3	
UNTD only	CSCE 2300	Digital Logic		3	
UNTD only	CSCE 3055	IT Project Management	*Prerequisite CSCE 2100	3	
UNTD only	CSCE 3310	Operating Systems	*Prerequisite CSCE 2300	3	
UNTD only	CSCE 3110	Data Structures	*Prerequisite CSCE 2110	3	
UNTD only	CSCE 3220	Human Computer Interfaces	*Prerequisite CSCE 2110	3	
UNTD only	CSCE 3420	Internet Programming	*Prerequisite CSCE 3110	3	
UNTD only	CSCE 3530	Intro to Computer Networks	*Prerequisite CSCE 3310	3	
UNTD only	CSCE 4010	Social Issues in Computing	*Prerequisite Junior Standing	3	
UNTD only	CSCE 4350	Intro to Database Systems Design	*Prerequisite CSCE 2110	3	
UNTD only	CSCE 4360	Database Administration	*Prerequisite CSCE 4350	3	
UNTD only	CSCE 4444	Software Engineering	*Prerequisite CSCE 2110	3	
UNTD only	CSCE 4550	Intro to Computer Security	*Co-requisite CSCE 4905	3	
UNTD only	CSCE 4905	Capstone I	*Prerequisite CSCE 3055	3	
UNTD only	CSCE 4925	Capstone II	*Co-requisite CSCE 4444	3	
UNTD only	CSCE 4925	Capstone II	*Prerequisite CSCE 4905	3	
MATH 2413	MATH 1710	Calculus I	*Prerequisite MATH 1600	4	

Electives/Minor –19+ Hours (Additional hours may be required based on coursework completed above. 120+ total hours needed for graduation)

Course	Hrs	Grade	Course/Term
MATH 1680 Elementary Probability & Statistics	3		
MATH 1600 Trigonometry	3		
Advanced Level Elective (3XXX or 4XXX)	3		
Minor, Certificate or General Elective	3		
Minor, Certificate or General Elective	3		
Minor, Certificate or General Elective	3		
Minor, Certificate or General Elective	3		
Minor, Certificate or General Elective	3		
Minor, Certificate or General Elective (if needed)	3		
Minor, Certificate or General Elective (if needed)	3		

Revised 4/22/19



2019-2020 UNT Dallas Catalog Degree Requirements:

Information Technology (Bachelor of Arts)

Use this checksheet as a guide for selecting classes and refer to your online Academic Advisement Report to review and monitor degree and graduation requirements.

UNT Dallas Degree Requirements:

Requirement	Complete
120 hours (minimum) required for degree completion	
Residency Requirement – At least 30 hours must be completed at UNT Dallas	
42 Advanced Hours (3XXX or 4XXX) – 24 advanced hours must be completed at UNT Dallas	
UNT Dallas GPA 2.0 or higher required	
Overall GPA 2.0 or higher required with “C” or better required in all major requirements	
Advanced CSCE Course GPA 2.75 or higher required	

BA in Information Technology Suggested 4-Year Plan

*Please note this sample list of courses meets all the requirements of the degree and illustrates a typical course sequence. Other courses may be used, and may be completed in a different order. Please refer back to the degree requirements listed above.

Semester 1	Semester 2	Semester 3	Semester 4
ENGL 1313	TECM 2700	HIST 2610	HIST 2620
MATH 1100	MATH 1600	CHEM 1410/40	CHEM 1420/40
PSCI 1040	PSCI 1050	CSCE 2300	CSCE 2110
LING 2050	CSCE 1030	CSCE 1040	MATH 1680
ART 1300	SOCI 1510	CSCE 2100	CSCE 3310

Semester 5	Semester 6	Semester 7	Semester 8
CSCE 3110	CSCE 3220	CSCE 4905	CSCE 4925
CSCE 3530	CSCE 3055	CSCE 4010	CSCE 4550
CSCE 4350	CSCE 3420	CSCE 4444	3/4XXX Elective
MATH 1710	CSCE 4360	Elective Any Level	Elective Any Level
Elective Any Level	Elective Any Level	Elective Any Level	Elective Any Level (If necessary)

Appendix B

Implementation Code

This section provides the Java and MySQL code needed for the implementation of each class of the system.

B.1 Login Page

```
1 import javafx.geometry.Insets;
2 import javafx.geometry.Pos;
3 import javafx.scene.Scene;
4 import javafx.scene.control.Button;
5 import javafx.scene.control.Hyperlink;
6 import javafx.scene.control.Label;
7 import javafx.scene.control.PasswordField;
8 import javafx.scene.control.TextField;
9 import javafx.scene.image.ImageView;
10 import javafx.scene.layout.Background;
11 import javafx.scene.layout.BackgroundFill;
12 import javafx.scene.layout.BorderPane;
13 import javafx.scene.layout.GridPane;
14 import javafx.scene.layout.HBox;
15 import javafx.scene.layout.VBox;
16 import javafx.scene.paint.Color;
```

```

17 import javafx.scene.text.Font;
18 import javafx.scene.text.FontWeight;
19
20 public class Login {
21
22     //private static Text titleL;
23     private static Label userL, passL;
24     private static TextField uName;
25     private static PasswordField pass;
26     private static Button login, signUp;
27     private static Hyperlink link;
28
29     public static Scene getPage() {
30
31
32         BorderPane pane = new BorderPane();
33         pane.setBackground(new Background(new BackgroundFill(Color.
34             WHITE, null, null)));
35
36         VBox vboxT = new VBox(10);
37         vboxT.setAlignment(Pos.CENTER);
38         vboxT.setPrefHeight(50);
39
40         HBox hboxB = new HBox(10);
41         hboxB.setAlignment(Pos.CENTER);
42
43         GridPane centerL = new GridPane(); // create and set pane
44         centerL.setAlignment(Pos.CENTER);
45
46         // create the components and add them to panes

```

```

46         //titleL = new Text(20, 20, "Login to College Class Scheduler
           ");
47         //titleL.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
48         //titleL.setFill(Color.rgb(0, 56, 130));
49
50
51         ImageView image = new ImageView("campus_skyview.jpg");
52         image.setFitHeight(600);
53         image.setFitWidth(600);
54         image.setPreserveRatio(true);
55         vboxT.getChildren().addAll(image);
56
57
58
59         double buttonWidth = 110;
60         login = new Button("Login");
61         login.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
62         login.setTextFill(Color.rgb(0, 56, 130));
63         login.setPrefWidth(buttonWidth);
64
65         signUp = new Button("Sign Up");
66         signUp.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
67         signUp.setTextFill(Color.rgb(0, 56, 130));
68         signUp.setPrefWidth(buttonWidth);
69
70         Login.setLink(new Hyperlink("Forgot Password?"));
71
72         hboxB.getChildren().addAll(signUp, login);
73         hboxB.setSpacing(10);
74
75

```

```

76     userL = new Label("Username:_");
77     userL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
78     userL.setTextFill(Color.rgb(0, 56, 130));
79
80     passL = new Label("Password:");
81     passL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
82     passL.setTextFill(Color.rgb(0, 56, 130));
83
84     double TextFieldWidth = 210;
85     uName = new TextField();
86     uName.setPromptText("Enter_Username");
87     //uName.setBorderProperty(Color.rgb(0, 56, 130));
88     //uName.setBorder(Color.rgb(0, 56, 130));
89     uName.setPrefWidth(TextFieldWidth);
90     //uName.setBorder(new Border(new BorderStroke(Color.rgb(0, 56,
91         130), BorderStrokeStyle.SOLID, CornerRadii.EMPTY,
92         BorderWidths.DEFAULT)));
93
94     pass = new PasswordField();
95     pass.setPromptText("Enter_Password");
96     pass.setPrefWidth(TextFieldWidth);
97
98     // add pane to border pane
99     centerL.add(userL, 0, 0);
100    centerL.add(uName, 1, 0);
101    centerL.add(passL, 0, 1);
102    centerL.add(pass, 1, 1);
103    centerL.add(link, 1, 2);
104    //centerL.setMargin(link, new Insets(10, 10, 10, 10)); //
105        Insets(top, right, bottom, left)
106    // add gap

```

```

104         centerL.setVgap(10);
105
106         pane.setCenter(centerL);
107         pane.setTop(vboxT);
108         pane.setBottom(hboxB);
109
110         Insets x = new Insets(0, 10, 75, 0); //Insets(top, right,
111         bottom, left)
112         pane.setPadding(x);
113
114
115         Scene loginPage = new Scene(pane, 600, 600); // create and set
116
117         return loginPage;
118     }
119
120     public Hyperlink getLink() {
121         return link;
122     }
123
124     public static void setLink(Hyperlink link) {
125         Login.link = link;
126     }
127
128     public static Button getLoginButton() {
129         return login;
130     }
131
132     public static TextField getUser() {
133         return uName;

```

```

134     }
135
136     public static PasswordField getPass() {
137         return pass;
138     }
139 }

```

B.2 Main Page

```

1 import javafx.scene.Scene;
2 import javafx.scene.layout.*;
3 import javafx.scene.paint.Color;
4 import javafx.scene.shape.Line;
5 import javafx.scene.text.Font;
6 import javafx.scene.text.FontWeight;
7 import javafx.scene.text.Text;
8 import javafx.scene.control.*;
9 import javafx.geometry.*;
10
11 public class MainPage {
12
13     private static Button addSt, addPls, addCor, planCor, stCor, rpts,
14         brow, signOut;
15
16     private static Text titleL, titleL2;
17
18     public static Scene getPage() {
19         BorderPane pane = new BorderPane();
20         pane.setBackground(new Background(new BackgroundFill(Color.
21             WHITE, null, null)));
22
23         VBox vboxT = new VBox(20);

```

```

21     vboxT.setAlignment(Pos.CENTER);
22     HBox hboxB = new HBox(20);
23     hboxB.setAlignment(Pos.CENTER_RIGHT);
24     VBox vbox = new VBox(20); // create and set pane
25     vbox.setAlignment(Pos.CENTER);
26     Insets listX = new Insets(11, 12, 13, 14);
27     vbox.setPadding(listX);
28
29     // create the components and add them to panes
30     titleL = new Text(20, 20, "Main_Menu");
31     titleL.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
32     titleL.setFill(Color.rgb(0, 56, 130));
33
34
35
36     titleL2 = new Text(20, 20, "Select_option_below");
37     titleL2.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
38     titleL2.setFill(Color.rgb(0, 56, 130));
39
40     vboxT.getChildren().addAll(titleL, titleL2);
41     Line line = new Line();
42     line.setStartX(250);
43     line.setStrokeWidth(2);
44     line.setStroke(Color.rgb(193, 165, 113));
45
46
47
48     vboxT.getChildren().addAll(line);
49
50
51     // create button

```



```

52     double buttonWidth = 200;
53     addSt = new Button("Student");
54     addSt.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
55     addSt.setTextFill(Color.rgb(0, 56, 130));
56
57     addPls = new Button("Plans");
58     addPls.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
59     addPls.setTextFill(Color.rgb(0, 56, 130));
60
61     addCor = new Button("Courses");
62     addCor.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
63     addCor.setTextFill(Color.rgb(0, 56, 130));
64
65     planCor = new Button("Plan_Courses");
66     planCor.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
67     planCor.setTextFill(Color.rgb(0, 56, 130));
68
69     stCor = new Button("Student_Courses");
70     stCor.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
71     stCor.setTextFill(Color.rgb(0, 56, 130));
72
73     rpts = new Button("Reports");
74     rpts.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
75     rpts.setTextFill(Color.rgb(0, 56, 130));
76
77     brow = new Button("Browse");
78     brow.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
79     brow.setTextFill(Color.rgb(0, 56, 130));
80
81     signOut = new Button("Sign_Out");
82     signOut.setFont(Font.font("Roboto", FontWeight.BOLD, 14));

```

```

83         signOut.setTextFill(Color.rgb(0, 56, 130));
84
85
86         addSt.setPrefWidth(buttonWidth);
87         addPls.setPrefWidth(buttonWidth);
88         addCor.setPrefWidth(buttonWidth);
89         planCor.setPrefWidth(buttonWidth);
90         stCor.setPrefWidth(buttonWidth);
91         rpts.setPrefWidth(buttonWidth);
92         brow.setPrefWidth(buttonWidth);
93         signOut.setPrefWidth(buttonWidth);
94
95         hboxB.getChildren().addAll(signOut);
96
97         vbox.getChildren().addAll(addSt, addPls, addCor, planCor,
98             stCor, rpts, brow);
99
100        pane.setCenter(vbox);
101        pane.setTop(vboxT);
102        pane.setBottom(hboxB);
103
104        Insets x = new Insets(25, 11, 25, 11);
105        pane.setPadding(x);
106
107        Scene scene = new Scene(pane, 600, 600); // create and set
108        return scene;
109    }
110
111    public static Button getStudentsButton() {
112        return addSt;
113    }

```

```
113
114     public static Button getPlansButton() {
115         return addPls;
116     }
117
118     public static Button getCoursesButton() {
119         return addCor;
120     }
121
122     public static Button getPlanCoursesButton() {
123         return planCor;
124     }
125
126     public static Button getStudentCoursesButton() {
127         return stCor;
128     }
129
130     public static Button getReportsButton() {
131         return rpts;
132     }
133
134     public static Button getBrowseButton() {
135         return brow;
136     }
137
138     public static Button getSignoutButton() {
139         return signOut;
140     }
141 }
```

B.3 Student

```
1 import javafx.scene.Scene;
2 import javafx.scene.layout.*;
3 import javafx.scene.paint.Color;
4 import javafx.scene.text.Font;
5 import javafx.scene.text.FontWeight;
6 import javafx.scene.text.Text;
7 import javafx.scene.control.*;
8
9 import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.Collections;
12
13 import javafx.geometry.*;
14
15 //import javafx.event.*;
16 public class Students {
17     // step 1: define your components (e.g. button, text, label....)
18     private static TextField sidTF, fnameTF, lnameTF, yearTF;
19     private static ComboBox<String> plansCB, semesterCB;
20     private static Button back, save;
21     private static Label sidL, plansL, fnameL, lnameL, yearL, semesterL;
22     private static Text text1, text2;
23
24     public static Scene getPage() throws SQLException {
25
26         // step 2: Create panes
27
28         BorderPane pane = new BorderPane();
29         pane.setBackground(new Background(new BackgroundFill(Color.
```

```

        WHITE, null, null)));
30
31     VBox vboxT = new VBox(20);
32     vboxT.setAlignment(Pos.CENTER);
33     HBox hboxB = new HBox(20);
34     hboxB.setAlignment(Pos.CENTER);
35     GridPane centerL = new GridPane();// create and set pane
36     centerL.setAlignment(Pos.CENTER);
37
38     // create the components and add them to panes
39     text1 = new Text(20, 20, "Students");
40     text1.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
41     text1.setFill(Color.WHITE);
42
43     text2 = new Text(20, 20, "Please_center_the_following_student_
44         information_then_click_save.");
45     text2.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
46     text2.setFill(Color.rgb(0, 56, 130));
47
48     vboxT.getChildren().addAll(text1, text2);
49     vboxT.setBackground(new Background(new BackgroundFill(Color.
50         rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
51         0, 0, 0))));
52     vboxT.setPrefHeight(115);
53
54     double buttonWidth = 110;
55     back = new Button("Back");
56     back.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
57     back.setTextFill(Color.rgb(0, 56, 130));
58     back.setPrefWidth(buttonWidth);

```

```

57
58         save = new Button("Save");
59         save.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
60         save.setTextFill(Color.rgb(0, 56, 130));
61         save.setPrefWidth(buttonWidth);
62
63         hboxB.getChildren().addAll(back, save);
64         hboxB.setBackground(new Background(new BackgroundFill(Color.
65             rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
66             0, 0, 0))));
67
68         sidL = new Label("Student_ID:");
69         sidL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
70         sidL.setTextFill(Color.rgb(0, 56, 130));
71
72         plansL = new Label("Plans:");
73         plansL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
74         plansL.setTextFill(Color.rgb(0, 56, 130));
75
76         fnameL = new Label("First_Name:");
77         fnameL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
78         fnameL.setTextFill(Color.rgb(0, 56, 130));
79
80         lnameL = new Label("Last_Name:");
81         lnameL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
82         lnameL.setTextFill(Color.rgb(0, 56, 130));
83
84
85         yearL = new Label("Year:");

```

```

86     yearL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
87     yearL.setTextFill(Color.rgb(0, 56, 130));
88
89     semesterL = new Label("Semester:");
90     semesterL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
91     semesterL.setTextFill(Color.rgb(0, 56, 130));
92
93     sidTF = new TextField();
94     fnameTF = new TextField();
95     lnameTF = new TextField();
96     yearTF = new TextField();
97     yearTF.setPromptText("YYYY");
98     plansCB = new ComboBox<String>();
99     semesterCB = new ComboBox<String>();
100
101     // add pane to border pane
102     centerL.add(sidL, 0, 0);
103     centerL.add(sidTF, 1, 0);
104     centerL.add(plansL, 0, 1);
105     centerL.add(plansCB, 1, 1);
106     centerL.add(fnameL, 0, 2);
107     centerL.add(fnameTF, 1, 2);
108     centerL.add(lnameL, 0, 3);
109     centerL.add(lnameTF, 1, 3);
110     centerL.add(yearL, 0, 4);
111     centerL.add(yearTF, 1, 4);
112     centerL.add(semesterL, 0, 5);
113     centerL.add(semesterCB, 1, 5);
114
115     // add gap
116     centerL.setVgap(20);

```

```

117         centerL.setHgap(20);
118
119         pane.setCenter(centerL);
120         pane.setTop(vboxT);
121         pane.setBottom(hboxB);
122
123         Insets x = new Insets(0, 0, 0, 0); //Insets(top, right,
124         bottom, left)
125
126         Scene plans = new Scene(pane, 600, 600); // create and set
127         return plans;
128     }
129
130
131     public static Button getBackButton() {
132         return back;
133     }
134
135     public static Button getSaveButton() {
136         return save;
137     }
138
139     public static TextField getStudentID() {
140         return sidTF;
141     }
142
143     public static ComboBox<String> getPlans() {
144         return plansCB;
145     }
146

```



```

147     public static TextField getFirstName() {
148         return fnameTF;
149     }
150
151     public static TextField getLastName() {
152         return lnameTF;
153     }
154
155     public static TextField getYear() {
156         return yearTF;
157     }
158
159     public static ComboBox<String> getSemester() {
160         return semesterCB;
161     }
162
163     public static void updateComboBoxes() throws SQLException {
164         plansCB.getItems().clear();
165         ArrayList<String> planNames = UI.populateComboBox("SELECT_
166             planName_FROM_plans;");
167         Collections.sort(planNames);
168         plansCB.getItems().addAll(planNames);
169         semesterCB.getItems().clear();
170         ArrayList<String> semesterNames = UI.populateComboBox("SELECT_
171             semesterName_FROM_semester;");
172         semesterCB.getItems().addAll(semesterNames);
173     }
174 }
175 }

```

B.4 Plans

```
1  import javafx.scene.Scene;
2  import javafx.scene.layout.*;
3  import javafx.scene.paint.Color;
4  import javafx.scene.text.Font;
5  import javafx.scene.text.FontWeight;
6  import javafx.scene.text.Text;
7  import javafx.scene.control.*;
8  import javafx.geometry.*;
9
10 public class Plans {
11     // step 1: define your components (e.g. button, text, label....)
12     private static TextField tfplans, tfyear, tfhours;
13     private static Button back, save;
14     private static Label plansL, yearL, hoursL;
15     private static Text text1, text2;
16
17     public static Scene getPage() {
18
19         // step 2: Create panes
20
21         BorderPane pane = new BorderPane();
22         pane.setBackground(new Background(new BackgroundFill(Color.
23             WHITE, null, null)));
24
25         VBox vboxT = new VBox(20);
26         vboxT.setAlignment(Pos.CENTER);
27         HBox hboxB = new HBox(20);
28         hboxB.setAlignment(Pos.CENTER);
```

```

29         GridPane centerL = new GridPane(); // create and set pane
30         centerL.setAlignment(Pos.CENTER);
31
32         // create the components and add them to panes
33         text1 = new Text(20, 20, "Plans");
34         text1.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
35         text1.setFill(Color.WHITE);
36
37         text2 = new Text(20, 20, "Please_enter_plan_information_then_
           click_save");
38         text2.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
39         text2.setFill(Color.rgb(0, 56, 130));
40
41
42         vboxT.getChildren().addAll(text1, text2);
43         vboxT.setBackground(new Background(new BackgroundFill(Color.
           rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
           0, 0, 0))));
44         vboxT.setPrefHeight(115);
45
46         double buttonWidth = 110;
47         back = new Button("Back");
48         back.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
49         back.setTextFill(Color.rgb(0, 56, 130));
50         back.setPrefWidth(buttonWidth);
51
52         save = new Button("Save");
53         save.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
54         save.setTextFill(Color.rgb(0, 56, 130));
55         save.setPrefWidth(buttonWidth);
56

```

```

57     hboxB.getChildren().addAll(back, save);
58     hboxB.setBackground(new Background(new BackgroundFill(Color.
        rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
        0, 0, 0))));
59     hboxB.setPrefHeight(80);
60
61
62     plansL = new Label("Plan Name:");
63     plansL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
64     plansL.setTextFill(Color.rgb(0, 56, 130));
65
66     yearL = new Label("Year:");
67     yearL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
68     yearL.setTextFill(Color.rgb(0, 56, 130));
69
70
71     hoursL = new Label("Hours:");
72     hoursL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
73     hoursL.setTextFill(Color.rgb(0, 56, 130));
74
75     tfplans = new TextField();
76     tfhours = new TextField();
77     tfyear = new TextField();
78     tfyear.setPromptText("YYYY");
79
80     // add pane to border pane
81     centerL.add(plansL, 0, 0);
82     centerL.add(tfplans, 1, 0);
83     centerL.add(yearL, 0, 1);
84     centerL.add(tfyear, 1, 1);
85     centerL.add(hoursL, 0, 2);

```

```

86         centerL.add(tfhours , 1, 2);
87
88         // add gap
89         centerL.setVgap(20);
90         centerL.setHgap(20);
91
92         pane.setCenter(centerL);
93         pane.setTop(vboxT);
94         pane.setBottom(hboxB);
95
96         Insets x = new Insets(0, 0, 0, 0);
97         pane.setPadding(x);
98
99         Scene plans = new Scene(pane, 600, 600); // create and set
100        return plans;
101    }
102
103    public static Button getBackButton() {
104        return back;
105    }
106
107    public static Button getSaveButton() {
108        return save;
109    }
110
111    public static TextField getPlanName() {
112        return tfplans;
113    }
114
115    public static TextField getYear() {
116        return tfyear;

```

```

117     }
118
119     public static TextField getHours() {
120         return tfhours;
121     }
122 }

```

B.5 Courses

```

1 import javafx.scene.Scene;
2 import javafx.scene.layout.*;
3 import javafx.scene.paint.Color;
4 import javafx.scene.text.Font;
5 import javafx.scene.text.FontWeight;
6 import javafx.scene.text.Text;
7 import javafx.scene.control.*;
8
9 import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.Collections;
12
13 import javafx.geometry.*;
14
15 //import javafx.event.*;
16 public class Courses {
17     // step 1: define your components (e.g. button, text, label....)
18     private static TextField coursecodeTF, coursenumTF, coursenameTF,
19         credithoursTF;
20     private static ComboBox<String> rotationCB, prereq1CB, prereq2CB,
21         prereq3CB, coreqCB;
22     private static Button back, save;

```

```

21     private static Label coursecodeL , coursenumber , coursenameL ,
           credithoursL , rotationL , prereq1L , prereq2L , prereq3L , coreqL ;
22     private static Text text1 , text2 ;
23
24     public static Scene getPage() throws SQLException {
25
26         // step 2: Create panes
27
28         BorderPane pane = new BorderPane() ;
29         pane.setBackground(new Background(new BackgroundFill(Color .
           WHITE, null , null))) ;
30
31         VBox vboxT = new VBox(20) ;
32         vboxT.setAlignment(Pos.CENTER) ;
33         HBox hboxB = new HBox(20) ;
34         hboxB.setAlignment(Pos.CENTER) ;
35         GridPane centerL = new GridPane() ; // create and set pane
36         centerL.setAlignment(Pos.CENTER) ;
37
38         // create the components and add them to panes
39         text1 = new Text(20, 20, "Courses") ;
40         text1.setFont(Font.font("Roboto" , FontWeight.BOLD, 25)) ;
41         text1.setFill(Color.WHITE) ;
42
43
44         text2 = new Text(20, 20, "Please_enter_course_information_then
           _click_save.") ;
45         text2.setFont(Font.font("Roboto" , FontWeight.NORMAL, 14)) ;
46         text2.setFill(Color.rgb(0, 56, 130)) ;
47
48         vboxT.getChildren().addAll(text1 , text2) ;

```

```

49         vboxT.setBackground(new Background(new BackgroundFill(Color.
           rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
           0, 0, 0))));
50         vboxT.setPrefHeight(100);
51
52         double buttonWidth = 110;
53         back = new Button("Back");
54         back.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
55         back.setTextFill(Color.rgb(0, 56, 130));
56         back.setPrefWidth(buttonWidth);
57
58
59
60         save = new Button("Save");
61         save.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
62         save.setTextFill(Color.rgb(0, 56, 130));
63         save.setPrefWidth(buttonWidth);
64
65
66         hboxB.getChildren().addAll(back, save);
67         hboxB.setBackground(new Background(new BackgroundFill(Color.
           rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
           0, 0, 0))));
68         hboxB.setPrefHeight(60);
69
70         coursecodeL = new Label("Course_Code:");
71         coursecodeL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
72         coursecodeL.setTextFill(Color.rgb(0, 56, 130));
73
74         coursenumber = new Label("Course_Number:");
75         coursenumber.setFont(Font.font("Roboto", FontWeight.BOLD, 14))

```



```

76         ;
77         coursenumber.setTextFill(Color.rgb(0, 56, 130));
78
79         coursenamel = new Label("Course_Name:");
80         coursenamel.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
81         coursenamel.setTextFill(Color.rgb(0, 56, 130));
82
83         credithoursL = new Label("Credit_Hours:");
84         credithoursL.setFont(Font.font("Roboto", FontWeight.BOLD, 14))
85         ;
86         credithoursL.setTextFill(Color.rgb(0, 56, 130));
87
88         rotationL = new Label("Rotation:");
89         rotationL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
90         rotationL.setTextFill(Color.rgb(0, 56, 130));
91
92         prereq1L = new Label("Prerequisite_1:");
93         prereq1L.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
94         prereq1L.setTextFill(Color.rgb(0, 56, 130));
95
96         prereq2L = new Label("Prerequisite_2:");
97         prereq2L.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
98         prereq2L.setTextFill(Color.rgb(0, 56, 130));
99
100        prereq3L = new Label("Prerequisite_3:");
101        prereq3L.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
102        prereq3L.setTextFill(Color.rgb(0, 56, 130));
103
104        coreqL = new Label("Co-requisite:");
105        coreqL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));

```

```

105         coreqL.setTextFill(Color.rgb(0, 56, 130));
106
107
108         coursecodeTF = new TextField();
109         coursenumbertf = new TextField();
110         coursenameTF = new TextField();
111         credithoursTF = new TextField();
112         rotationCB = new ComboBox<String>();
113         prereq1CB = new ComboBox<String>();
114         prereq2CB = new ComboBox<String>();
115         prereq3CB = new ComboBox<String>();
116         coreqCB = new ComboBox<String>();
117
118         // add pane to border pane
119         centerL.add(coursecodeL, 0, 0);
120         centerL.add(coursecodeTF, 1, 0);
121         centerL.add(coursenumbertf, 0, 1);
122         centerL.add(coursenumbertf, 1, 1);
123         centerL.add(coursenameL, 0, 2);
124         centerL.add(coursenameTF, 1, 2);
125         centerL.add(credithoursL, 0, 3);
126         centerL.add(credithoursTF, 1, 3);
127         centerL.add(rotationL, 0, 4);
128         centerL.add(rotationCB, 1, 4);
129         centerL.add(prereq1L, 0, 5);
130         centerL.add(prereq1CB, 1, 5);
131         centerL.add(prereq2L, 0, 6);
132         centerL.add(prereq2CB, 1, 6);
133         centerL.add(prereq3L, 0, 7);
134         centerL.add(prereq3CB, 1, 7);
135         centerL.add(coreqL, 0, 8);

```

```

136         centerL.add(coreqCB, 1, 8);
137
138         // add gap
139         centerL.setVgap(20);
140         centerL.setHgap(20);
141
142         pane.setCenter(centerL);
143         pane.setTop(vboxT);
144         pane.setBottom(hboxB);
145
146         Insets x = new Insets(0, 0, 0, 0);
147         pane.setPadding(x);
148
149         Scene plans = new Scene(pane, 600, 600); // create and set
150         return plans;
151     }
152
153     public static Button getBackButton() {
154         return back;
155     }
156
157     public static Button getSaveButton() {
158         return save;
159     }
160
161     public static TextField getCourseCode() {
162         return coursecodeTF;
163     }
164
165     public static TextField getCourseNumber() {
166         return coursenumbertf;

```

```

167     }
168
169     public static TextField getCourseName() {
170         return coursenamеTF;
171     }
172
173     public static TextField getCreditHours() {
174         return credithoursTF;
175     }
176
177     public static ComboBox<String> getRotation() {
178         return rotationCB;
179     }
180
181     public static ComboBox<String> getPrereq1() {
182         return prereq1CB;
183     }
184
185     public static ComboBox<String> getPrereq2() {
186         return prereq2CB;
187     }
188
189     public static ComboBox<String> getPrereq3() {
190         return prereq3CB;
191     }
192
193     public static ComboBox<String> getCoreq() {
194         return coreqCB;
195     }
196
197     public static void updateComboBoxes() throws SQLException {

```

```

198         rotationCB.getItems().clear();
199         ArrayList<String> rotationNames = UI.populateComboBox("SELECT_
           rotationName_FROM_rotation;");
200         rotationCB.getItems().addAll(rotationNames);
201         prereq1CB.getItems().clear();
202         prereq2CB.getItems().clear();
203         prereq3CB.getItems().clear();
204         coreqCB.getItems().clear();
205         ArrayList<String> courseNames = UI.populateComboBox("SELECT_
           courseName_FROM_courses;");
206         Collections.sort(courseNames);
207         prereq1CB.getItems().addAll(courseNames);
208         prereq2CB.getItems().addAll(courseNames);
209         prereq3CB.getItems().addAll(courseNames);
210         coreqCB.getItems().addAll(courseNames);
211     }
212 }

```

B.6 Plan Courses

```

1  import javafx.scene.Group;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.*;
4  import javafx.scene.paint.Color;
5  import javafx.scene.text.Font;
6  import javafx.scene.text.FontWeight;
7  import javafx.scene.text.Text;
8  import javafx.scene.control.*;
9
10 import java.sql.SQLException;
11 import java.util.ArrayList;

```

```

12 import java.util.Collections;
13
14 import javafx.geometry.*;
15
16 public class PlanCourses {
17     // step 1: define your components (e.g. button, text, label....)
18     private static ComboBox<String> tfPlanCor, tfCourse, tfType;
19     private static Button back, save;
20     private static Label planL, courseL, majorL;
21     private static Text text1, text2;
22
23     public static Scene getPage() throws SQLException {
24
25         // step 2: Create panes
26
27         BorderPane pane = new BorderPane();
28         pane.setBackground(new Background(new BackgroundFill(Color.
29             WHITE, null, null)));
30
31         VBox vboxT = new VBox(20);
32         vboxT.setAlignment(Pos.CENTER);
33         HBox hboxB = new HBox(20);
34         hboxB.setAlignment(Pos.CENTER);
35         GridPane centerL = new GridPane(); // create and set pane
36         centerL.setAlignment(Pos.CENTER);
37
38         // create the components and add them to panes
39         text1 = new Text(20, 20, "Plan_Courses");
40         text1.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
41         text1.setFill(Color.WHITE);

```

```

42     text2 = new Text(20, 20, "Please select the following then
        click save");
43     text2.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
44     text2.setFill(Color.rgb(0, 56, 130));
45
46     vboxT.getChildren().addAll(text1, text2);
47     vboxT.setBackground(new Background(new BackgroundFill(Color.
        rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
        0, 0, 0))));
48     vboxT.setPrefHeight(115);
49
50     double buttonWidth = 110;
51     back = new Button("Back");
52     back.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
53     back.setTextFill(Color.rgb(0, 56, 130));
54     back.setPrefWidth(buttonWidth);
55
56     save = new Button("Save");
57     save.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
58     save.setTextFill(Color.rgb(0, 56, 130));
59     save.setPrefWidth(buttonWidth);
60
61     hboxB.getChildren().addAll(back, save);
62     hboxB.setBackground(new Background(new BackgroundFill(Color.
        rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
        0, 0, 0))));
63     hboxB.setPrefHeight(80);
64
65     Scene scene2 = new Scene(new Group(), 450, 250);
66
67     tfPlanCor = new ComboBox<String>();

```

```

68         tfCourse = new ComboBox<String>();
69         tfType = new ComboBox<String>();
70
71         Group root = (Group) scene2.getRoot();
72         root.getChildren().add(tfPlanCor);
73
74         planL = new Label("Plans:");
75         planL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
76         planL.setTextFill(Color.rgb(0, 56, 130));
77
78         courseL = new Label("Course:");
79         courseL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
80         courseL.setTextFill(Color.rgb(0, 56, 130));
81
82         majorL = new Label("Type:");
83         majorL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
84         majorL.setTextFill(Color.rgb(0, 56, 130));
85
86
87         // add pane to border pane
88         centerL.add(planL, 0, 0);
89         centerL.add(tfPlanCor, 1, 0);
90         centerL.add(courseL, 0, 1);
91         centerL.add(tfCourse, 1, 1);
92         centerL.add(majorL, 0, 2);
93         centerL.add(tfType, 1, 2);
94
95         // add gap
96         centerL.setVgap(20);
97         centerL.setHgap(20);
98

```



```

99         pane.setCenter(centerL);
100        pane.setTop(vboxT);
101        pane.setBottom(hboxB);
102
103        Insets x = new Insets(0, 0, 0, 0);
104        pane.setPadding(x);
105
106        Scene plans = new Scene(pane, 600, 600); // create and set
107        return plans;
108    }
109
110    public static Button getBackButton() {
111        return back;
112    }
113
114    public static Button getSaveButton() {
115        return save;
116    }
117
118    public static ComboBox<String> getPlan() {
119        return tfPlanCor;
120    }
121
122    public static ComboBox<String> getCourse() {
123        return tfCourse;
124    }
125
126    public static ComboBox<String> getType() {
127        return tfType;
128    }
129

```

```

130     public static void updateComboBoxes() throws SQLException {
131         tfPlanCor.getItems().clear();
132         ArrayList<String> planNames = UI.populateComboBox("SELECT_
           planName FROM plans;");
133         Collections.sort(planNames);
134         tfPlanCor.getItems().addAll(planNames);
135         tfCourse.getItems().clear();
136         ArrayList<String> courseNames = UI.populateComboBox("SELECT_
           courseName FROM courses;");
137         Collections.sort(courseNames);
138         tfCourse.getItems().addAll(courseNames);
139         tfType.getItems().clear();
140         ArrayList<String> types = UI.populateComboBox("SELECT_type_
           FROM type;");
141         tfType.getItems().addAll(types);
142     }
143 }

```

B.7 Student Courses

```

1  import javafx.scene.Scene;
2  import javafx.scene.layout.*;
3  import javafx.scene.paint.Color;
4  import javafx.scene.text.Font;
5  import javafx.scene.text.FontWeight;
6  import javafx.scene.text.Text;
7  import javafx.scene.control.*;
8
9  import java.sql.SQLException;
10 import java.util.ArrayList;
11 import java.util.Collections;

```

```

12
13 import javafx.geometry.*;
14
15 public class StudentCourses {
16     // step 1: define your components (e.g. button, text, label....)
17     private static TextField tfstudID, tfyear;
18     private static Button back, save;
19     private static Label studentL, courseL, yearL, semesterL, gradeL;
20     private static Text text1, text2;
21     private static ComboBox<String> courseB, semesterB, gradeB;
22
23     public static Scene getPage() throws SQLException {
24
25         // step 2: Create panes
26
27         BorderPane pane = new BorderPane();
28         pane.setBackground(new Background(new BackgroundFill(Color.
29             WHITE, null, null)));
30
31         VBox vboxT = new VBox(20);
32         vboxT.setAlignment(Pos.CENTER);
33         HBox hboxB = new HBox(20);
34         hboxB.setAlignment(Pos.CENTER);
35         GridPane centerL = new GridPane(); // create and set pane
36         centerL.setAlignment(Pos.CENTER);
37
38         // create the components and add them to panes
39         text1 = new Text(20, 20, "Student_Courses");
40         text1.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
41         text1.setFill(Color.WHITE);

```

```

42
43     text2 = new Text(20, 20, "Please_enter_the_following_then_
         click_save");
44     text2.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
45     text2.setFill(Color.rgb(0, 56, 130));
46
47
48     vboxT.getChildren().addAll(text1, text2);
49     vboxT.setBackground(new Background(new BackgroundFill(Color.
         rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
         0, 0, 0))));
50     vboxT.setPrefHeight(115);
51
52     double buttonWidth = 110;
53     back = new Button("Back");
54     back.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
55     back.setTextFill(Color.rgb(0, 56, 130));
56     back.setPrefWidth(buttonWidth);
57
58     save = new Button("Save");
59     save.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
60     save.setTextFill(Color.rgb(0, 56, 130));
61     save.setPrefWidth(buttonWidth);
62
63     hboxB.getChildren().addAll(back, save);
64     hboxB.setBackground(new Background(new BackgroundFill(Color.
         rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
         0, 0, 0))));
65     hboxB.setPrefHeight(80);
66
67     studentL = new Label("Student_ID:");

```

```

68         studentL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
69         studentL.setTextFill(Color.rgb(0, 56, 130));
70
71         courseL = new Label("Course:");
72         courseL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
73         courseL.setTextFill(Color.rgb(0, 56, 130));
74
75         yearL = new Label("Year:");
76         yearL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
77         yearL.setTextFill(Color.rgb(0, 56, 130));
78
79         semesterL = new Label("Semester:");
80         semesterL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
81         semesterL.setTextFill(Color.rgb(0, 56, 130));
82
83         gradeL = new Label("Grade");
84         gradeL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
85         gradeL.setTextFill(Color.rgb(0, 56, 130));
86
87
88         tfstudID = new TextField();
89         tfyear = new TextField();
90         tfyear.setPromptText("YYYY");
91         courseB = new ComboBox<String>();
92         semesterB = new ComboBox<String>();
93         gradeB = new ComboBox<String>();
94         gradeB.getItems().addAll("A", "B", "C", "D", "F", "I", "W");
95
96         // add pane to border pane
97         centerL.add(studentL, 0, 0);
98         centerL.add(tfstudID, 1, 0);

```

```

99         centerL.add(courseL, 0, 1);
100        centerL.add(courseB, 1, 1);
101        centerL.add(yearL, 0, 2);
102        centerL.add(tfyear, 1, 2);
103        centerL.add(semesterL, 0, 3);
104        centerL.add(semesterB, 1, 3);
105        centerL.add(gradeL, 0, 4);
106        centerL.add(gradeB, 1, 4);
107
108        // add gap
109        centerL.setVgap(20);
110        centerL.setHgap(20);
111
112        pane.setCenter(centerL);
113        pane.setTop(vboxT);
114        pane.setBottom(hboxB);
115
116        Insets x = new Insets(0, 0, 0, 0);
117        pane.setPadding(x);
118
119        Scene studentCourses = new Scene(pane, 600, 600); // create and
120        set
121        return studentCourses;
122    }
123
124    public static Button getBackButton() {
125        return back;
126    }
127
128    public static Button getSaveButton() {
129        return save;

```

```

129     }
130
131     public static TextField getStudentID () {
132         return tfstudID;
133     }
134
135     public static ComboBox<String> getCourse () {
136         return courseB;
137     }
138
139     public static TextField getYear () {
140         return tfyear;
141     }
142
143     public static ComboBox<String> getSemester () {
144         return semesterB;
145     }
146
147     public static ComboBox<String> getGrade () {
148         return gradeB;
149     }
150
151     public static void updateComboBoxes () throws SQLException {
152         courseB.getItems().clear();
153         ArrayList<String> courseNames = UI.populateComboBox("SELECT_
154             courseName _FROM courses ;");
155         Collections.sort(courseNames);
156         courseB.getItems().addAll(courseNames);
157         semesterB.getItems().clear();
158         ArrayList<String> semesterNames = UI.populateComboBox("SELECT_
159             semesterName _FROM semester ;");

```

```

158         semesterB.getItems().addAll(semesterNames);
159     }
160 }

```

B.8 Reports

```

1  import javafx.scene.Scene;
2  import javafx.scene.layout.*;
3  import javafx.scene.paint.Color;
4  import javafx.scene.text.*;
5  import javafx.scene.control.*;
6  import javafx.geometry.*;
7
8  public class Reports {
9      // step 1: define your components (e.g. button, text, label....)
10     private static TextField studentId;
11     private static Button backbtn, reportbtn;
12     private static Label studentIdL;
13     private static Text titleL, titleAdd;
14
15     public static Scene getPage() {
16
17         // step 2: Create panes
18
19         BorderPane pane = new BorderPane();
20         pane.setBackground(new Background(new BackgroundFill(Color.
21             WHITE, null, null)));
22
23         VBox vboxT = new VBox(20);
24         vboxT.setAlignment(Pos.CENTER);
25         HBox hboxB = new HBox(20);

```



```

25     hboxB.setAlignment(Pos.CENTER);
26     GridPane centerL = new GridPane();// create and set pane
27     centerL.setAlignment(Pos.CENTER);
28
29     // create the components and add them to panes
30     titleL = new Text(20, 20, "Reports");
31     titleL.setFont(Font.font("Roboto", FontWeight.BOLD, 25));
32     titleL.setFill(Color.WHITE);
33
34     titleAdd = new Text("To_generate_report_center_student_ID_and_
35         click_generate_report.");
36     titleAdd.setFont(Font.font("Roboto", FontWeight.NORMAL, 14));
37     titleAdd.setFill(Color.rgb(0, 56, 130));
38
39     vboxT.getChildren().addAll(titleL, titleAdd);
40     vboxT.setBackground(new Background(new BackgroundFill(Color.
41         rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
42         0, 0, 0))));
43     vboxT.setPrefHeight(115);
44
45     double buttonWidth = 110;
46     backbtn = new Button("Back");
47     backbtn.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
48     backbtn.setTextFill(Color.rgb(0, 56, 130));
49     backbtn.setPrefWidth(buttonWidth);
50
51     reportbtn = new Button("Generate_Report");
52     reportbtn.setFont(Font.font("Roboto", FontWeight.BOLD, 12));
53     reportbtn.setTextFill(Color.rgb(0, 56, 130));
54     reportbtn.setPrefWidth(buttonWidth);

```

```

53
54     hboxB.getChildren().addAll(backbtn, reportbtn);
55     hboxB.setBackground(new Background(new BackgroundFill(Color.
        rgb(194, 166, 114, 0.64), CornerRadii.EMPTY, new Insets(0,
        0, 0, 0))));
56     hboxB.setPrefHeight(80);
57
58     studentIdL = new Label("Student ID:");
59     studentIdL.setFont(Font.font("Roboto", FontWeight.BOLD, 14));
60     studentIdL.setTextFill(Color.rgb(0, 56, 130));
61
62     studentId = new TextField();
63
64     // add pane to border pane
65     centerL.add(studentIdL, 0, 0);
66     centerL.add(studentId, 1, 0);
67
68     // add gap
69     centerL.setVgap(10);
70     centerL.setHgap(10);
71
72     pane.setCenter(centerL);
73     pane.setTop(vboxT);
74     pane.setBottom(hboxB);
75
76     Insets x = new Insets(0, 0, 0, 0);
77     pane.setPadding(x);
78
79     Scene scene = new Scene(pane, 600, 600); // create and set
80     return scene;
81 }

```

```
82
83     public static Button getBackButton() {
84         return backbtn;
85     }
86
87     public static Button getGenerateButton() {
88         return reportbtn;
89     }
90
91     public static TextField getStudentID() {
92         return studentId;
93     }
94
95     public static void updateComboBoxes() {
96
97     }
98 }
```

B.9 UI Class

```
1 import java.sql.Connection;
2 import java.sql.ResultSet;
3 import java.sql.SQLException;
4 import java.util.ArrayList;
5
6 import org.apache.log4j.Logger;
7 import org.apache.log4j.varia.NullAppender;
8
9 import com.zaxxer.hikari.HikariDataSource;
10
11
```

```

12
13 import javafx.application.Application;
14 import javafx.scene.Scene;
15 import javafx.scene.control.Alert;
16 import javafx.scene.control.Alert.AlertType;
17 import javafx.scene.image.Image;
18 import javafx.stage.Stage;
19
20 public class UI extends Application {
21
22     private Scene loginPage, mainPage, studentsPage, plansPage,
23         coursesPage, planCoursesPage, studentCoursesPage,
24         reportsPage;
25
26     public static String sqlUser = "", sqlPass = "";
27
28     // application launch method
29     public static void main(String[] args) {
30         // initialize hikari logger and set to silent
31         Logger.getRootLogger().addAppender(new NullAppender());
32         // launch application
33         Application.launch(args);
34     }
35
36     @Override
37     public void start(Stage primaryStage) throws Exception {
38         // initiate all page scenes in memory
39         loginPage = Login.getPage();
40         mainPage = MainPage.getPage();
41         studentsPage = Students.getPage();
42         plansPage = Plans.getPage();
43         coursesPage = Courses.getPage();

```

```

42     planCoursesPage = PlanCourses.getPage();
43     studentCoursesPage = StudentCourses.getPage();
44     reportsPage = Reports.getPage();
45
46     // set primary stage info and show login page
47     //primaryStage();
48
49     primaryStage.setTitle("College_Class_Scheduler");
50     primaryStage.setScene(loginPage);
51
52
53
54     Image icon = new Image("cap_1.png");
55     primaryStage.getIcons().add(icon);
56
57     primaryStage.show();
58
59
60
61
62     // event: username enter key
63     Login.getUser().setOnAction(e -> {
64         login(primaryStage);
65     });
66
67     // event: password enter key
68     Login.getPass().setOnAction(e -> {
69         login(primaryStage);
70     });
71
72     // event: login button

```

```

73     Login.getLoginButton().setOnAction(e -> {
74         login(primaryStage);
75     });
76
77     // event: sign out from main page
78     MainPage.getSignoutButton().setOnAction(e -> {
79         primaryStage.setScene(loginPage);
80     });
81
82     // event: students button from main page
83     MainPage.getStudentsButton().setOnAction(e -> {
84         try {
85             Students.updateComboBoxes();
86         } catch (SQLException e1) {
87             e1.printStackTrace();
88         }
89         primaryStage.setScene(studentsPage);
90     });
91
92     // event: save button from students page
93     Students.getSaveButton().setOnAction(e -> {
94         try {
95             sqlSave("INSERT INTO students (studentID ,
96                 firstName , lastName , enrollmentYear ,
97                 planID , semesterID) VALUES (
98                 + Students.getStudentID().
99                 getText() + ", " +
100                 Students.getFirstName().
101                 getText() + ", " +
102                 + Students.getLastName().
103                 getText() + ", " +

```

```

100         Students.getYear().getText
101         () + ",_"
102     + getID("SELECT planID FROM
103         plans WHERE planName=" +
104         Students.getPlans().
105         getValue() + " ");")
106     + ",_" + getID("SELECT
107         semesterID FROM semester
108         WHERE semesterName="
109         + Students.
110         getSemester
111         ().
112         getValue()
113         + " ");")
114     + ");");
115 } catch (SQLException eStudents) {
116     eStudents.printStackTrace();
117 }
118
119 Students.getStudentID().clear();
120 Students.getPlans().setValue(null);
121 Students.getFirstName().clear();
122 Students.getLastName().clear();
123 Students.getYear().clear();
124 Students.getSemester().setValue(null);
125
126 try {
127     Students.updateComboBoxes();
128 } catch (SQLException e1) {
129     e1.printStackTrace();
130 }

```

```

118         });
119
120         // event: back button from students page
121         Students.getBackButton().setOnAction(e -> {
122             primaryStage.setScene(mainPage);
123             Students.getStudentID().clear();
124             Students.getPlans().setValue(null);
125             Students.getFirstName().clear();
126             Students.getLastName().clear();
127             Students.getYear().clear();
128             Students.getSemester().setValue(null);
129         });
130
131         // event: plans button from main page
132         MainPage.getPlansButton().setOnAction(e -> {
133             primaryStage.setScene(plansPage);
134         });
135
136         // event: save button from plans page
137         Plans.getSaveButton().setOnAction(e -> {
138             try {
139                 sqlSave("INSERT INTO plans (planName, planYear
140                     , planHours) VALUES ('" + Plans.
141                         getPlanName().getText()
142                             + ", " + Plans.getYear().
143                                 getText() + ", " + Plans.
144                                     getHours().getText() + "');
145             } catch (SQLException eCourses) {
146                 eCourses.printStackTrace();
147             }

```



```

144
145         Plans.getPlanName().clear();
146         Plans.getYear().clear();
147         Plans.getHours().clear();
148     });
149
150     // event: back button from plans page
151     Plans.getBackButton().setOnAction(e -> {
152         primaryStage.setScene(mainPage);
153         Plans.getPlanName().clear();
154         Plans.getYear().clear();
155         Plans.getHours().clear();
156     });
157
158     // event: courses button from main page
159     MainPage.getCoursesButton().setOnAction(e -> {
160         try {
161             Courses.updateComboBoxes();
162         } catch (SQLException e1) {
163             e1.printStackTrace();
164         }
165         primaryStage.setScene(coursesPage);
166     });
167
168     // event: save button from courses page
169     Courses.getSaveButton().setOnAction(e -> {
170         try {
171             // optional values check
172             String prereq1, prereq2, prereq3, coreq;
173             if (Courses.getPrereq1().getValue() == null) {
174                 prereq1 = "NULL";

```

```

175         } else {
176             prereq1 = String.valueOf(getID("SELECT
                _courseID _FROM_ courses _WHERE_
                _courseName='"
177                                     + Courses.getPrereq1()
                .getValue() + "'");
                ));
178         }
179         if (Courses.getPrereq2().getValue() == null) {
180             prereq2 = "NULL";
181         } else {
182             prereq2 = String.valueOf(getID("SELECT
                _courseID _FROM_ courses _WHERE_
                _courseName='"
183                                     + Courses.getPrereq2()
                .getValue() + "'");
                ));
184         }
185         if (Courses.getPrereq3().getValue() == null) {
186             prereq3 = "NULL";
187         } else {
188             prereq3 = String.valueOf(getID("SELECT
                _courseID _FROM_ courses _WHERE_
                _courseName='"
189                                     + Courses.getPrereq3()
                .getValue() + "'");
                ));
190         }
191         if (Courses.getCoreq().getValue() == null) {
192             coreq = "NULL";
193         } else {

```

```

194         coreq = String.valueOf(getID(
195             "SELECT courseID FROM
                courses WHERE
                courseName=" +
                Courses.getCoreq()
                .getValue() + " ");
196     });
197
198     sqlSave("INSERT INTO courses (courseCode ,
                courseNumber , courseName , courseHours ,
                rotationID , prereq1 , prereq2 , prereq3 ,
                coreq) VALUES (
199         + Courses.getCourseCode().
                getText() + ", " +
                Courses.getCourseNumber().
                getText() + ", "
200         + Courses.getCourseName().
                getText() + ", " +
                Courses.getCreditHours().
                getText() + ", "
201         + getID("SELECT rotationID
                FROM rotation WHERE
                rotationName="
202             + Courses.
                getRotation
                ().
                getValue()
                + " ");
203         + ", " + prereq1 + ", " +
                prereq2 + ", " + prereq3 +

```

```

203         ", " + coreq + ");");
204     } catch (SQLException eCourses) {
205         eCourses.printStackTrace();
206     }
207
208     Courses.getCourseCode().clear();
209     Courses.getCourseNumber().clear();
210     Courses.getCourseName().clear();
211     Courses.getCreditHours().clear();
212     Courses.getRotation().setValue(null);
213     Courses.getPrereq1().setValue(null);
214     Courses.getPrereq2().setValue(null);
215     Courses.getPrereq3().setValue(null);
216     Courses.getCoreq().setValue(null);
217
218     try {
219         Courses.updateComboBoxes();
220     } catch (SQLException e1) {
221         e1.printStackTrace();
222     }
223 });
224
225 // event: back button from courses page
226 Courses.getBackButton().setOnAction(e -> {
227     primaryStage.setScene(mainPage);
228     Courses.getCourseCode().clear();
229     Courses.getCourseNumber().clear();
230     Courses.getCourseName().clear();
231     Courses.getCreditHours().clear();
232     Courses.getRotation().setValue(null);
233     Courses.getPrereq1().setValue(null);

```

```

234         Courses.getPrereq2().setValue(null);
235         Courses.getPrereq3().setValue(null);
236         Courses.getCoreq().setValue(null);
237     });
238
239     // event: plan courses button from main page
240     MainPage.getPlanCoursesButton().setOnAction(e -> {
241         try {
242             PlanCourses.updateComboBoxes();
243         } catch (SQLException e1) {
244             e1.printStackTrace();
245         }
246         primaryStage.setScene(planCoursesPage);
247     });
248
249     // event: save button from plan courses page
250     PlanCourses.getSaveButton().setOnAction(e -> {
251         try {
252             sqlSave("INSERT INTO plans_courses (courseID ,
253                 planID , typeID) VALUES (
254                     + getID("SELECT courseID FROM
255                         courses WHERE courseName='
256                         " + PlanCourses.getCourse
257                             ().getValue()
258                             + " ';" )
259                     + ",_"
260                     + getID("SELECT planID FROM
261                         plans WHERE planName=" +
262                             PlanCourses.getPlan().
263                             getValue() + " ';" )
264                     + ",_" + getID("SELECT typeID _

```

```

FROM_type_WHERE_type=" +
    PlanCourses.getType().
        getValue() + " '";")
258         + ");");
259     } catch (SQLException eCourses) {
260         eCourses.printStackTrace();
261     }
262
263     PlanCourses.getPlan().setValue(null);
264     PlanCourses.getCourse().setValue(null);
265     PlanCourses.getType().setValue(null);
266
267     try {
268         PlanCourses.updateComboBoxes();
269     } catch (SQLException e1) {
270         e1.printStackTrace();
271     }
272 });
273
274 // event: back button from plan courses page
275 PlanCourses.getBackButton().setOnAction(e -> {
276     primaryStage.setScene(mainPage);
277     PlanCourses.getPlan().setValue(null);
278     PlanCourses.getCourse().setValue(null);
279     PlanCourses.getType().setValue(null);
280 });
281
282 // event: student courses button from main page
283 MainPage.getStudentCoursesButton().setOnAction(e -> {
284     try {
285         StudentCourses.updateComboBoxes();

```

```

286         } catch (SQLException e1) {
287             e1.printStackTrace();
288         }
289         primaryStage.setScene(studentCoursesPage);
290     });
291
292     // event: save button from student courses page
293     StudentCourses.getSaveButton().setOnAction(e -> {
294         try {
295             sqlSave("INSERT INTO students_courses (
296                 studentID , courseID , year , semesterID ,
297                 grade) VALUES (
298
299                     + StudentCourses.getStudentID
300                       ().getText() + ", "
301                     + getID("SELECT courseID FROM
302                       courses WHERE courseName='
303
304
305                       +
306                       StudentCourses
307                       .getCourse
308                       ().
309                       getValue()
310                       + " '");
311                     + ", " + StudentCourses.
312                       getYear().getText() + ", "
313                     + getID("SELECT semesterID
314                       FROM semester WHERE
315                       semesterName='")
316                       +
317                       StudentCourses
318                       .

```

```

302         + ",_" + StudentCourses.
           getSemester
           ().
           getValue()
           + "';")
           getGrade().getValue() + "
           ');");
303     } catch (SQLException eCourses) {
304         eCourses.printStackTrace();
305     }
306
307     StudentCourses.getStudentID().clear();
308     StudentCourses.getCourse().setValue(null);
309     StudentCourses.getYear().clear();
310     StudentCourses.getSemester().setValue(null);
311     StudentCourses.getGrade().setValue(null);
312
313     try {
314         StudentCourses.updateComboBoxes();
315     } catch (SQLException e1) {
316         e1.printStackTrace();
317     }
318     });
319
320     // event: back button from student courses page
321     StudentCourses.getBackButton().setOnAction(e -> {
322         primaryStage.setScene(mainPage);
323         StudentCourses.getStudentID().clear();
324         StudentCourses.getCourse().setValue(null);
325         StudentCourses.getYear().clear();
326         StudentCourses.getSemester().setValue(null);

```



```

327         StudentCourses.getGrade().setValue(null);
328     });
329
330     // event: reports button from main page
331     MainPage.getReportsButton().setOnAction(e -> {
332         primaryStage.setScene(reportsPage);
333     });
334
335     // event: generate report button from reports page
336     Reports.getGenerateButton().setOnAction(e -> {
337         String coursesString = "";
338         try {
339             ReportData data = new ReportData(Integer.
340                 parseInt(Reports.getStudentID().getText())
341                 );
342             for (int i = 0; i < data.getRemainingCourses()
343                 .size(); i++) {
344                 coursesString = coursesString + data.
345                     getRemainingCourses().get(i).
346                     getCourseCode() + " "
347                     + data.
348                         getRemainingCourses()
349                         .get(i).
350                         getCourseNumber()
351                         + " "
352                     + data.
353                         getRemainingCourses()
354                         .get(i).
355                         getCourseName() +
356                         "\n";
357             }

```

```

345         Alert alert = new Alert(AlertType.INFORMATION)
346             ;
347         alert.setTitle("Report");
348         alert.setHeaderText(data.getFName() + "_" +
349             data.getLName() + "'s Remaining Courses:")
350             ;
351         alert.setContentText(coursesString);
352         alert.showAndWait();
353     } catch (SQLException e1) {
354         e1.printStackTrace();
355     }
356     Reports.getStudentID().clear();
357 });
358
359 // event: back button from reports page
360 Reports.getBackButton().setOnAction(e -> {
361     primaryStage.setScene(mainPage);
362     Reports.getStudentID().clear();
363 });
364 // catch server offline
365 }
366
367 // handles user/pass for login, throws error if incorrect
368 public static void setLoginInfo(String user, String pass) {
369     sqlUser = user;
370     sqlPass = pass;
371 }
372
373 public void login(Stage primaryStage) {
374     setLoginInfo(Login.getUser().getText(), Login.getPass().
375         getText());

```

```

372     try {
373         HikariDataSource loginTestDs = new HikariDataSource();
374         loginTestDs.setJdbcUrl("jdbc:mysql://localhost:3306/
           college_class_scheduler");
375         loginTestDs.setUsername(sqlUser);
376         loginTestDs.setPassword(sqlPass);
377         Connection loginTest = loginTestDs.getConnection();
378         primaryStage.setScene(mainPage);
379         Login.getUser().clear();
380         Login.getPass().clear();
381         loginTest.close();
382         loginTestDs.close();
383     } catch (SQLException e) {
384         Alert alert = new Alert(AlertType.ERROR);
385         alert.setTitle("Invalid Login");
386         alert.setHeaderText(null);
387         alert.setContentText("Incorrect username or password!"
           );
388         alert.showAndWait();
389     }
390 }
391
392 public static ArrayList<String> populateComboBox(String query) throws
           SQLException {
393     Connection con = ConnectionPool.getConnection();
394     ResultSet result = con.createStatement().executeQuery(query);
395     ArrayList<String> list = new ArrayList<String>();
396     while (result.next()) {
397         list.add(result.getString(1));
398     }
399     con.close();

```

```

400         return list;
401     }
402
403     public static int getID(String query) throws SQLException {
404         Connection con = ConnectionPool.getConnection();
405         ResultSet result = con.createStatement().executeQuery(query);
406         result.next();
407         int id = result.getInt(1);
408         con.close();
409         return id;
410     }
411
412     public static void sqlSave(String query) throws SQLException {
413         Connection con = ConnectionPool.getConnection();
414         con.createStatement().executeUpdate(query);
415         con.close();
416     }
417
418 }

```

B.10 Report Data Class

```

1  import java.sql.Connection;
2  import java.sql.ResultSet;
3  import java.sql.SQLException;
4  import java.util.ArrayList;
5
6  public class ReportData {
7      int studentID;
8      String fName, lName;
9      int enrollmentYear;

```

```

10     int planID;
11     int semesterID;
12     ArrayList<courseData> remainingCourses = new ArrayList<courseData>();
13
14     public ReportData(int studentID) throws SQLException {
15         // fill student data
16         this.studentID = studentID;
17         Connection con = ConnectionPool.getConnection();
18         ResultSet result = con.createStatement().executeQuery("SELECT
19             *_FROM_students_WHERE_studentID_=__" + this.studentID);
20         while (result.next()) {
21             fName = result.getString(2);
22             lName = result.getString(3);
23             enrollmentYear = result.getInt(4);
24             planID = result.getInt(5);
25             semesterID = result.getInt(6);
26         }
27         con.close();
28
29         // fill remainingCourses
30         Connection con2 = ConnectionPool.getConnection();
31         ResultSet result2 = con2.createStatement().executeQuery(
32             "SELECT_*_FROM_plans_courses ,_courses_WHERE_
33             plans_courses.courseID_NOT_IN_(SELECT_
34             courseID_FROM_students_courses_WHERE_
35             studentID_=__"
36             + this.studentID
37             + " _AND_(grade_=_\"A\" _OR_
38             grade_=_\"B\" _OR_grade_=_
39             \"C\")) _AND_PlanID_=__(
40             SELECT_planID_from_

```

```

students_WHERE_studentID_ =
    _”
34         + this.studentID + ”)_AND_
           plans_courses.courseID_ =_
           courses.courseID;”);
35     while (result2.next()) {
36         courseData newCourse = new courseData();
37         newCourse.setCourseID(result2.getInt(4));
38         newCourse.setCourseCode(result2.getString(5));
39         newCourse.setCourseNumber(result2.getInt(6));
40         newCourse.setCourseName(result2.getString(7));
41         newCourse.setCreditHours(result2.getInt(8));
42         newCourse.setRotationID(result2.getInt(9));
43         newCourse.setPrereq1(result2.getInt(10));
44         newCourse.setPrereq2(result2.getInt(11));
45         newCourse.setPrereq3(result2.getInt(12));
46         newCourse.setCoreq(result2.getInt(13));
47         remainingCourses.add(newCourse);
48     }
49     con2.close();
50 }
51
52 public int getStudentID() {
53     return studentID;
54 }
55
56 public String getFName() {
57     return fName;
58 }
59
60 public String getLName() {

```

```

61         return lName;
62     }
63
64     public int getEnrollmentYear () {
65         return enrollmentYear;
66     }
67
68     public int getPlanID () {
69         return planID;
70     }
71
72     public int getSemesterID () {
73         return semesterID;
74     }
75
76     public ArrayList<courseData> getRemainingCourses () {
77         return remainingCourses;
78     }
79
80     public class courseData {
81         private int courseID;
82         private String courseCode;
83         private int courseNumber;
84         private String courseName;
85         private int creditHours;
86         private int rotationID;
87         private int prereq1 , prereq2 , prereq3 , coreq;
88
89         private courseData () {
90
91     }

```

```
92
93     public int getCourseID() {
94         return courseID;
95     }
96
97     public void setCourseID(int courseID) {
98         this.courseID = courseID;
99     }
100
101     public String getCourseCode() {
102         return courseCode;
103     }
104
105     public void setCourseCode(String courseCode) {
106         this.courseCode = courseCode;
107     }
108
109     public int getCourseNumber() {
110         return courseNumber;
111     }
112
113     public void setCourseNumber(int courseNumber) {
114         this.courseNumber = courseNumber;
115     }
116
117     public String getCourseName() {
118         return courseName;
119     }
120
121     public void setCourseName(String courseName) {
122         this.courseName = courseName;
```



```
123     }
124
125     public int getCreditHours() {
126         return creditHours;
127     }
128
129     public void setCreditHours(int creditHours) {
130         this.creditHours = creditHours;
131     }
132
133     public int getRotationID() {
134         return rotationID;
135     }
136
137     public void setRotationID(int rotationID) {
138         this.rotationID = rotationID;
139     }
140
141     public int getPrereq1() {
142         return prereq1;
143     }
144
145     public void setPrereq1(int prereq1) {
146         this.prereq1 = prereq1;
147     }
148
149     public int getPrereq2() {
150         return prereq2;
151     }
152
153     public void setPrereq2(int prereq2) {
```

```

154         this.prereq2 = prereq2;
155     }
156
157     public int getPrereq3() {
158         return prereq3;
159     }
160
161     public void setPrereq3(int prereq3) {
162         this.prereq3 = prereq3;
163     }
164
165     public int getCoreq() {
166         return coreq;
167     }
168
169     public void setCoreq(int coreq) {
170         this.coreq = coreq;
171     }
172
173     }
174 }

```

B.11 Connection Pool Class

```

1 import java.sql.Connection;
2 import java.sql.SQLException;
3
4 import com.zaxxer.hikari.HikariConfig;
5 import com.zaxxer.hikari.HikariDataSource;
6
7 public class ConnectionPool {

```

```

8
9     private static HikariConfig config = new HikariConfig();
10    private static HikariDataSource ds;
11
12    static {
13        config.setJdbcUrl("jdbc:mysql://localhost:3306/
14            college_class_scheduler");
15        config.setUsername(UI.sqlUser);
16        config.setPassword(UI.sqlPass);
17        config.addDataSourceProperty("cachePrepStmts", "true");
18        config.addDataSourceProperty("prepStmtCacheSize", "250");
19        config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");
20        ds = new HikariDataSource(config);
21    }
22
23    public static Connection getConnection() throws SQLException {
24        return ds.getConnection();
25    }
26
27    private ConnectionPool() {
28    }

```

B.12 MySQL Database Creation Code

```

1 CREATE DATABASE IF NOT EXISTS college_class_scheduler;
2 USE college_class_scheduler;
3 CREATE TABLE plans (planID int unsigned NOT NULL AUTOINCREMENT, planName
4     varchar(45), planYear year(4), planHours int, PRIMARY KEY(planID));
5 CREATE TABLE semester (semesterID int unsigned NOT NULL, semesterName varchar
6     (45), PRIMARY KEY(semesterID));

```

```

5 CREATE TABLE students (studentID int unsigned NOT NULL, firstName varchar(45),
  lastName varchar(45), enrollmentYear year(4), planID int unsigned,
  semesterID int unsigned, PRIMARY KEY(studentID),FOREIGN KEY(planID)
REFERENCES plans(planID) ON DELETE SET NULL ON UPDATE CASCADE, FOREIGN KEY
  (semesterID) REFERENCES semester (semesterID)ON DELETE CASCADE ON UPDATE
  CASCADE);
6 CREATE TABLE TYPE (typeID int unsigned NOT NULL, TYPE varchar(45), PRIMARY KEY
  (typeID));
7 CREATE TABLE rotation (rotationID int unsigned NOT NULL, rotationName varchar
  (45), PRIMARY KEY(rotationID));
8 CREATE TABLE courses (courseID int unsigned NOT NULL AUTO_INCREMENT,
  courseCode varchar(45), courseNumber int, courseName varchar(45),
  courseHours int, rotationID int unsigned, prereq1 int unsigned, prereq2
  int unsigned, prereq3 int unsigned, coreq int unsigned, PRIMARY KEY(
  courseID), FOREIGN KEY(rotationID) REFERENCES rotation (rotationID) ON
  DELETE SET NULL ON UPDATE CASCADE, FOREIGN KEY(prereq1) REFERENCES courses
  (courseID) ON DELETE SET NULL ON UPDATE CASCADE, FOREIGN KEY(prereq2)
  REFERENCES courses(courseID) ON DELETE SET NULL ON UPDATE CASCADE, FOREIGN
  KEY(prereq3) REFERENCES courses(courseID) ON DELETE SET NULL ON UPDATE
  CASCADE, FOREIGN KEY(coreq) REFERENCES courses(courseID) ON DELETE SET
  NULL ON UPDATE CASCADE);
9 CREATE TABLE students_courses (studentID int unsigned NOT NULL, courseID int
  unsigned NOT NULL, year Year, semesterID int unsigned NOT NULL, grade char
  (1), PRIMARY KEY (studentID, courseID, semesterID), FOREIGN KEY (courseID)
  REFERENCES courses (courseID)ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN
  KEY(studentID) REFERENCES students (studentID) ON DELETE CASCADE ON
  UPDATE CASCADE, FOREIGN KEY (semesterID) REFERENCES semester (semesterID)
  ON DELETE CASCADE ON UPDATE CASCADE);
10 CREATE TABLE plans_courses (planID int unsigned NOT NULL, courseID int
  unsigned NOT NULL, typeID int unsigned NOT NULL, PRIMARY KEY (planID,
  courseID, typeID), FOREIGN KEY (planID) REFERENCES plans (planID)ON DELETE

```

```
        CASCADE ON UPDATE CASCADE, FOREIGN KEY (courseID) REFERENCES courses (
courseID)ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY(typeID)
REFERENCES TYPE (typeID) ON DELETE CASCADE ON UPDATE CASCADE);
11 INSERT INTO rotation (rotationID , rotationName) VALUES (1, 'Fall'), (2, '
Spring'), (3, 'Both');
12 INSERT INTO semester (semesterID , semesterName) VALUES (1, 'Fall'), (2, '
Spring');
13 INSERT INTO TYPE (typeID , TYPE) VALUES (1, 'Major'), (2, 'Core'), (3, '
Elective');
```

Bibliography

- [1] E. C. L.P, “Aligns students, advisors, and institutions to a common goal—on-time graduation.” <https://www.ellucian.com/solutions/ellucian-degree-works>, 2020. accessed September 7, 2020.
- [2] C. Systems, “The degree audit system.” <https://www.conclusivesystems.com>. accessed September 7, 2020.
- [3] T. D. S. M. John D Spencer, “Texas common course numbering system.” "<https://www.tccns.org/>. accessed September 7, 2020.
- [4] A. Offices, “econnect through dallas college.” <https://econnect.dcccd.edu/>. accessed September 6, 2020.
- [5] C. Fruin and J. Grochowski, “Caesked: A class scheduler for wmu students,” *Computer Science Senior Projects*, 2010.
- [6] D. M. N. J. A. Stuart, T. C. T. S. M. Dascalu, and F. C. Harris Jr, “Software requirements specification of a university class scheduler.”
- [7] “Cybermatrix corporation, inc.,” ”2020 (accessed September 6, 2020)”. accessed September 6, 2020.
- [8] T. Müller, “University timetabling.” <https://www.unitime.org/>. accessed September 6, 2020.
- [9] A. B. Adel Alshamrani, “A comparison between three sdlc models waterfall model, spiral model, and incremental/iterative model,” *International Journal of computer science issues.*, 2015. accessed October 2, 2020.
- [10] “The scrum guide.” <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>. accessed October 2, 2020.

- [11] "Sdlc - waterfall model." https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. accessed October 4, 2020.
- [12] "What is extreme programming (xp)?." <https://www.agilealliance.org/glossary/xp/>. accessed October 5, 2020.
- [13] C. A. Kent Beck, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2 ed., 2004.
- [14] I. Sommerville, *Software Engineering*. Pearson, 10 ed., 2015.
- [15] D. Raggett, A. Le Hors, and I. Jacobs, "Html 4.01 specification," *HTML 4.0 Specification*, vol. W3C Recommendation, revised on April 24, 1998, no. 4.0, pp. 19–22, revised on 1998.
- [16] W. W. W. Consortium, "The degree audit system." <https://www.w3.org/Consortium/>. accessed September 7, 2020.
- [17] H. W. Lie, "Cascading html style sheets – a proposal 1994." <https://www.w3.org/People/howcome/p/cascade.html>. accessed September 10, 2020.
- [18] c.-c. o. C. Bert Bos, "The css saga, chapter 20, cascading style sheets, designing for the web, by h akon wium lie and bert bos (2nd edition, 1999, addison wesley, isbn 0-201-59625-3)." <https://www.w3.org/Style/LieBos2e/history/>. accessed September 18, 2020.
- [19] W. W. W. Consortium, "Biography sir tim berners-lee." <https://www.w3.org/People/Berners-Lee/>. accessed September 10, 2020.
- [20] "Javascript." https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. accessed September 20, 2020.
- [21] Y. D. Liang, *Introduction to Java programming : brief version*. Pearson, eleventh edition ed., 2018.
- [22] "What can php do?." <https://www.php.net/manual/en/intro-whatcando.php>. accessed September 21, 2020.
- [23] A. Giurca, "Accessing databases with php," *Accessing Databases with PHP*, 2005.
- [24] "Bradley, angela. "what is php used for?." [thoughtco.com/what-is-php-used-for-2694011](https://www.thoughtco.com/what-is-php-used-for-2694011). accessed September 21, 2020.

- [25] “Advantages and disadvantages of php.” <https://www.phpbabu.com/advantages-and-disadvantages-of-php/>. accessed September 21, 2020.
- [26] P. J. Deitel and H. Dietal, *Intro to Python for the computer and data sciences: learning to program with AI, big data and the cloud*. Pearson Education, Inc., first edition ed., 2019.
- [27] “Pep 20 – the zen of python.” <https://www.python.org/dev/peps/pep-0020/>. accessed September 22, 2020.
- [28] B. Stroustrup, *The C++ programming language*. Pearson Education, 2013.
- [29] “Ruby on rails.” https://worddisk.com/wiki/Ruby_on_Rails/. accessed September 21, 2020.
- [30] “Ruby (programming language).” [https://worddisk.com/wiki/Ruby_\(programming_language\)](https://worddisk.com/wiki/Ruby_(programming_language))/. accessed September 21, 2020.
- [31] “Oracle database software downloads.” <https://www.oracle.com/database/technologies/oracle-database-software-downloads.html>. accessed October 2, 2020.
- [32] “Oracle autonomous database.” <https://www.oracle.com/a/ocom/docs/kuppingercole-autonomous-database-4368706.pdf>. accessed October 2, 2020.
- [33] “Oracle technology global price list.” <https://docs.oracle.com/en/database/oracle/oracle-database/19/dblic/database-licensing-information-user-manual.pdf>. accessed October 2, 2020.
- [34] “Free oracle database for everyone.” <https://www.oracle.com/database/technologies/appdev/xe.html>. accessed October 2, 2020.
- [35] M. Kofler, *The Definitive Guide to MySQL 5*. Apress, 2006.
- [36] “About sqlite.” <https://sqlite.org/about.html>. accessed October 4, 2020.
- [37] “Microsoft data platform.” <https://www.microsoft.com/en-us/sql-server>. accessed October 5, 2020.
- [38] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005. accessed September 26, 2020.

- [39] H.-E. Eriksson, M. Penker, B. Lyons, and D. Fado, *UML 2 Toolkit*. Wiley Publishing, 2003.
- [40] J. A. S. Michael Jesse Chonoles, *UML 2 for Dummies*. For Dummies, 2003.
- [41] “Introducing types of uml diagrams.” <https://www.lucidchart.com/blog/types-of-UML-diagrams>. accessed September 25, 2020.
- [42] “What is composite structure diagram?.” <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-composite-structure-diagram/>. accessed September 27, 2020.
- [43] “Uml 2 deployment diagrams: An agile introduction.” <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-composite-structure-diagram/>. accessed September 27, 2020.
- [44] “What is package diagram?.” <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/>. accessed September 27, 2020.
- [45] “What is profile diagram?.” <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-profile-diagram/>. accessed September 27, 2020.
- [46] “Uml class and object diagrams overview.” <https://www.uml-diagrams.org/class-diagrams-overview.html>. accessed September 25, 2020.
- [47] “Class diagram in uml.” <https://learn-it-with-examples.com/it-architecture-management/it-architecture/uml/class-diagram-uml.html>. accessed September 25, 2020.
- [48] “Uml class and object diagrams overview.” <https://www.uml-diagrams.org/class-diagrams-overview.html#object-diagram>. accessed September 25, 2020.
- [49] “Object diagram in uml.” <https://learn-it-with-examples.com/it-architecture-management/it-architecture/uml/object-diagram-uml.html>. accessed September 25, 2020.
- [50] “Uml component diagrams.” <https://www.uml-diagrams.org/component-diagrams.html>. accessed September 25, 2020.
- [51] “Component diagram in uml.” <https://learn-it-with-examples.com/it-architecture-management/it-architecture/uml/component-diagram-uml.html>. accessed September 25, 2020.

- [52] “State machine diagrams.” <https://www.uml-diagrams.org/state-machine-diagrams.html#behavioral-state-machine>. accessed September 25, 2020.
- [53] “State machine (statechart) diagram in uml.” <https://learn-it-with-examples.com/it-architecture-management/it-architecture/uml/state-machine-diagram-uml.html>. accessed September 25, 2020.
- [54] “Sequence diagram.” https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/sequencediagram.html. accessed September 25, 2020.
- [55] “Timing diagram.” https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/timingdiagram.html. accessed September 25, 2020.
- [56] “Interaction overview diagram.” https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/interactionoverviewdiagram.html. accessed September 25, 2020.
- [57] I. Sommerville, *Software Engineering 10th Edition*. Pearson, 2016. accessed October 17, 2020.
- [58] “How to create useful software process documentation.” http://westfallteam.com/Papers/Useful_Software_Process_Documentation.pdf. accessed October 17, 2020.
- [59] “Software engineering 9 -chapter 30 documentation.” https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters/PDF/Ch_30%20Documentation.pdf. accessed October 17, 2020.
- [60] C. B. Thomas Connolly, *Database Systems: A Practical Approach to Design, Implementation, and Management (6th Edition)*. Pearson, 2015.
- [61] “Untd skyview.” <https://www.untDallas.edu/news/caruth-police-institute-approved-provide-able-training> accessed April 17, 2021.