

ESOF-3050 Software Engineering Design
Fall 2021

Software Project Document

Leader: Nicholas Imperius 0645031

Sukhraj Deol 1171944

Jimmy Tsang 1098204

Kristopher Poulin 0883832

11/27/21

Approval

This document has been read and approved by the following team members responsible for its implementation:

PRINT NAME: Nicholas Imperius

Signature:



PRINT NAME: Kristopher Poulin

Signature:



PRINT NAME: Jimmy Tsang

Signature:



PRINT NAME: Sukhraj Deol

Signature:



I. Table Of Contents

I. Table Of Contents	3
II. Introduction	6
A. Glossary	6
III. Domain Analysis	8
IV. Business Model	9
V. Functional Requirements	10
VI. Non-Functional Requirements	11
VII. CASE Tools	12
VIII. Product Specification	13
A. Use Case Diagram	13
B. Use Case Descriptions	14
IX. Software Process Model	21
X. Risks Involved	23
XI. Design Overview	24
XII. Software Architecture	25
A. Java Implementation	25
A. Database Model And Diagram	25
B. Interaction Diagram	26
C. Communication Protocols and Messaging Format	27
D. User Interfaces (GUI), Software Interfaces	28
XIII. Object-Oriented Design	42
A. Complete Class Diagrams	42
B. Algorithms Used and Complexity	45
C. Detailed Design	46
XIV. Activity Diagrams	48
XV. Design Impacts	51
XVI. Test Approach	52
XVII. Test Plan	53
A. Features To Be Tested	53
B. Features Not To Be Tested	54

C. Testing Tools and Environment	54
XVIII. Test Cases	55
A. Logging In	55
B. Searching For Courses	55
C. Registering For a Course	56
D. Viewing Enrolled Courses	56
E. Dropping a Course	57
F. Viewing Course Grades	57
G. Viewing Active Courses Offered	57
H. Modifying Grades	58
I. Viewing the Employee List	58
J. Viewing the Student List	59
K. Adding a Member	59
L. Adding a Course	60
M. Removing an Employee	60
N. Removing a Course	61
O. Class Capacity Error Handling	61
P. Testing for Incorrect Input	61
XIX. Cost and Duration Estimate	63
XX. Roles and Responsibilities	68
A. Ownership of Subsystem	68
B. Deliverables and Milestones	68
C. Project Leader	69
XXI. Conclusion	70
Appendix A	71
A. Test Log 1	71
B. Test Log 2	71
C. Test Log 3	71
D. Test Log 4	71
E. Test Log 5	72
F. Test Log 6	72
G. Test Log 7	72
H. Test Log 8	72
I. Test Log 9	72
J. Test Log 10	72
K. Test Log 11	73
L. Test Log 12	73
M. Test Log 13	73
N. Test Log 14	73

O. Test Log 15

73

P. Test Log 16

74

XXII. References

75

II. Introduction

Our product is a University Registration System (URS) that is connected to a database. The URS will be responsible for managing student enrollment, as well as the course history for the university. In addition, it will allow employees to schedule classes, students to search and enroll in courses, instructors to manage their classes, and administrators to access information about grades, and student lists for each course, and add or remove university employees. Students will be able to enroll in any course they wish to, and will only be able to see the grades of courses they are currently enrolled in and have previously taken. Students cannot register if the class is full. If not enough students are registered in a class, then that class will be cancelled. Instructors will be assigned to at least one section for a course, where they will be able to view the students enrolled and modify their grades. Instructors can also see the list of employees at the university. Administrators will have the ability to view the student and employee list, add and remove courses offered, and add and remove university members. The URS database will be created and managed using MySQL, while Java will be used to create the Graphical User Interface and functionality. The database will include information consisting of student names, ID, employee names, employee ID, and SIN. In addition, it will be responsible for completing queries and sending the information back to the user interface. The goal of how we designed our system was to reduce the amount of time spent doing tasks and provide a robust interface to gather as much necessary information as possible. In our design process, we focused on achieving a simplistic interface that can provide all the functionality required from a university registration system. With that in mind, we had to carefully plan our user interfaces as well as the functions of the system to create something that would satisfy both of our constraints.

A. Glossary

Administrator	Department chairs, deans, and other top executives.
Faculty Member	Either an administrator or instructor
Instructor	Someone teaching a course.
Student	Someone who is enrolled in a course can be an undergraduate or

	graduate student and full-time or part-time.
GUI	Is a graphical user interface, this is the interface that the user will be interacting with when using our system.
Java	An object-oriented programming language
MySQL	An open-source language used to create relational databases.
Database	Will hold all personal information such as names, passwords, SIN and other information.
Server	A system that will communicate with many other systems and provide information to them.
Client	A system that interacts with a server, requires information stored on the server to show to the user of the system.
Class	Code that will create objects, member variables, and implementations of member functions.
Method	A block of statements to be executed by a method call.
Method Call	A way to execute a method in a programming language.
Course	A learning source that covers one or more subjects.

III. Domain Analysis

A university registration system is an integral part of any student's career. For many students, this can be a stressful and difficult process if the system is hard to use. Online registration allows students to do this task much easier than before. Older registration systems have very low efficiency because everything was made of paper, students must have paid their student fees in full before they could register. In addition, records of the registration process were only shown on the student ID which made it difficult to get information at any given moment [1]. The registration process usually consists of 3 phases: the registration phase, the adjustment phase, and the dropping phase [2]. Based on our research on the 3 phases, we tried to design elegant solutions to help students during the registration process in each phase. In our domain, we also found that current systems in place have central databases that store information about the system [3]. We have decided to go along with an application-based system that uses the commonly found Client-Server architecture to connect users to our database. We will be able to verify account credentials through look-up tables as well as store any pertinent information required.

IV. Business Model

In the University Registration and Database System that we are deploying to universities, certain processes and tasks are required. The basic functionality required would be the ability to browse, register, schedule and enter marks for courses. Our system will store certain information on the members of the university that including member type, first name, last name, social insurance number, date of birth, home address, and status. Students will be able to browse and register for courses, check course eligibility, drop a course, and view grades for their currently enrolled courses. Administrators will be able to hire/recruit and fire/retire members from the system, add or remove courses, and view any course information. Part-time or full-time instructors will be able to add grades to the courses that they are teaching, view courses, view enrolled students on their own and other courses being offered. Importantly, our system will be able to provide concurrent access, so that multiple users can perform the same tasks at the same time without any system error. For example, if two were to register for the same course with only one opening available, the first person to register will be successful but the second person would get an error message when they try to register.

An economic impact can be the cost of performing maintenance and upgrades when needed to ensure that the system never shuts down unexpectedly. A societal impact can be the ease of access to utilize the functionalities of the system for both students and staff members. A more streamlined system can reduce bad experiences that can occur when using a registration system. An example of this can be when registration for classes opens, the server can handle all of the traffic without crashing or slowing down, leading to a more pleasant experience.

V. Functional Requirements

- A student must be classified as an Undergraduate Student or Graduate student, which includes Ph.D. or Master's Degree students.
- All university members must have a first name, surname, SIN, Date of Birth, address, ID number, and status.
- All students will be considered either part-time or full-time, if the student is enrolled in 3 or more courses, they will be classified as full-time otherwise part-time. An employee would manually choose their status.
- A student must be able to enroll in a course as long as they meet the year level requirement and the course is not equal to or exceeding max capacity.
- A student must be able to drop a course based on eligibility conditions that must be met. These conditions include the course being active or not yet started and the course's last day to withdraw date has not been reached.
- Each class has one and only one Instructor per section that can modify the grades for each student.
- Class sections must have specific naming conventions as well, including a subject, number, title, section, and description.
- Administrators must have the ability to add or remove employees and students from the system.
- Each student must not have any conflicts with their course timetable.

VI. Non-Functional Requirements

One non-functional requirement will be the speed of the system. The goal is that the system will be able to complete normal requests in at most 2 seconds. That being said, when a student registers for a course, the entire process should take no more than 5 seconds. This would fall under the performance non-functional requirement. Another non-functional requirement will be the type of operating system that the software will run on. Windows will be the operating system that this software will be built to run on, however, the software can run on other operating systems as well, meaning that it will not be limited to Windows only. This would fall under the system's non-functional requirement. Lastly, will be the capacity of the registration system. The registration system should be able to hold at least 4 active users at the same time without any problems or exceptions occurring. This would fall under the capacity non-functional requirement.

VII. CASE Tools

During the development of our system, we will be utilizing CASE tools to help automate tasks in our project. For front-end CASE tools, we used Google Drawings and Creately to draw the diagrams and help us visualize the database and its connections. We used Google Docs for our documentation, as it allowed us to edit the same document in real-time. For sharing code between our machines, we are using a Github repository to push changes and merge our updates together. For the back-end CASE tools, we used MySQL for database design and queries, Eclipse for the Java integration and implementation, SceneBuilder to develop a GUI, and XAMPP to connect our database and java application together in a Client-Server way. Our GitHub repository can be found here: https://github.com/nickimps/ESOF3050_Project.

VIII. Product Specification

A. Use Case Diagram

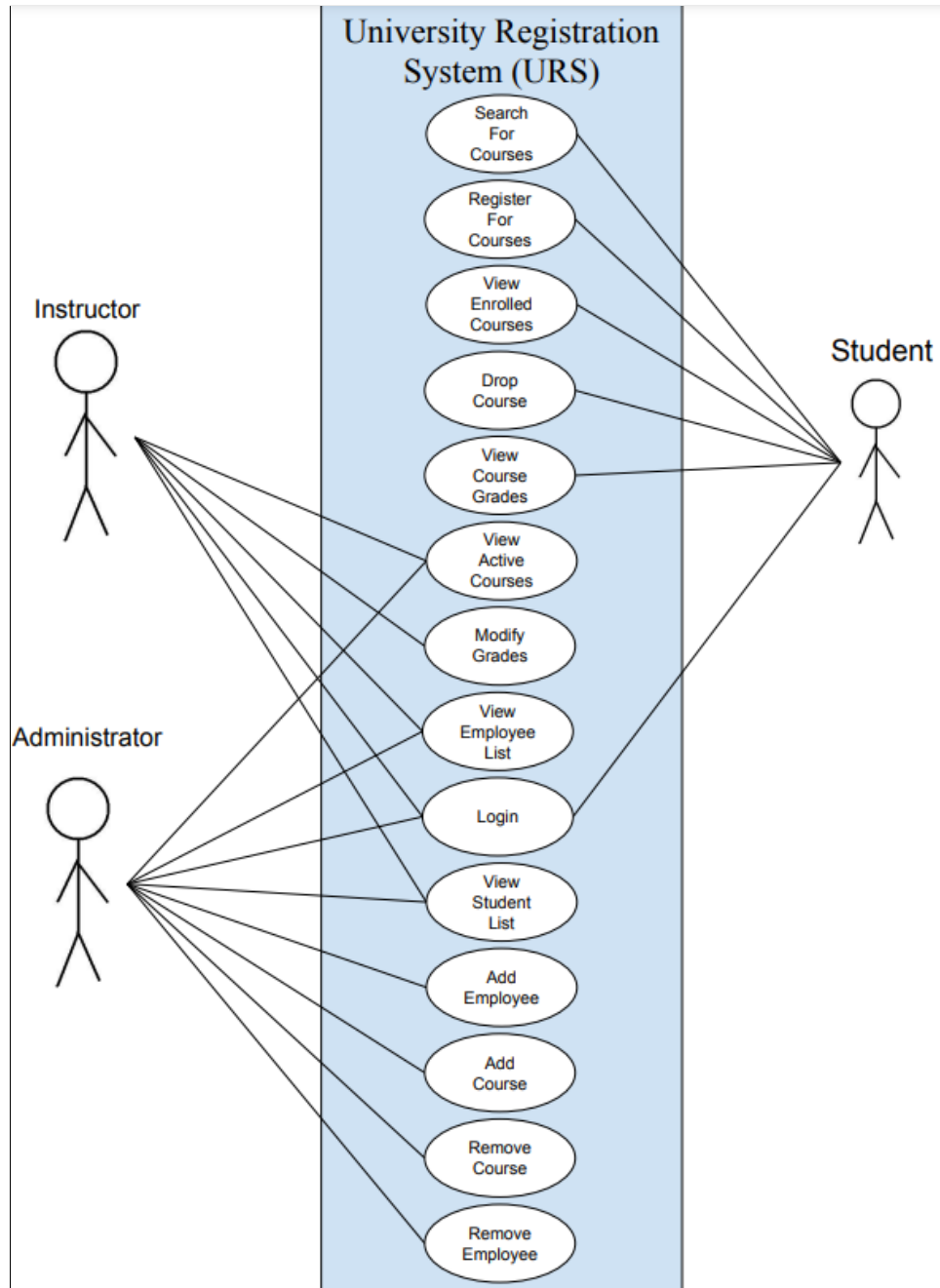


Figure 1: Use Case Diagram of our University Registration System

B. Use Case Descriptions

Brief Description:

The `Login` use case enables an existing user to log in to the system and access any resources available to them.

Step-by-step Description:

1. The user will enter their username and password and click the login button.
2. If the user's credentials are in the system's database, then the system will display the corresponding welcome screen to the user.

Brief Description:

The `Search for Courses` use case allows for users to search for a course by subject, number, title, program of study, as well as by keyword.

Step-by-step Description:

1. The user enters in the piece of course information that they want to filter by and clicks the search button
2. The system displays all available courses with those specific parameters

Brief Description:

The Register For Course use case allows students to enroll in a course if they meet the required eligibility.

Step-by-step Description:

1. User selects a course

The Search for Courses use case is used.

2. The system determines if the student is eligible
 - a. If Eligible
 - i. The system allows the student to click the enroll button and the system updates the enrollment limit of the course.
 - b. If Not Eligible
 - i. The system does not allow the student to proceed further and provides a reason why.

Brief Description:

The View Enrolled Courses use case allows student users to view all their currently enrolled classes.

Step-by-step Description:

1. The student selects the view enrolled courses button.
2. The system displays all the courses the student is enrolled in.

Brief Description:

The `Drop Course` use case allows students to drop out of courses, given that the conditions are met by the student.

Step-by-step Description:

1. The student selects a course

The `View Enrolled Course` use case is used

2. The Student clicks on the drop course button.
3. The system verifies if the conditions are met
 - a. Conditions met
 - i. Updates the change and removes it from their enrolled courses
 - b. Conditions not met
 - i. The system provides the conditions that have not been met.

Brief Description:

The `View Course Grades` use case allows for students to view their grades in their registered courses.

Step-by-step Description:

1. The Student clicks on the view course grades button.
2. The system displays all their enrolled courses with the corresponding grade.

Brief Description:

The `View Active Courses` use case allows for administrators to view all active courses being offered.

Step-by-step Description:

1. The employee clicks on the view active courses button.
2. The system displays all active courses with the following information:
 - Subject
 - 4-digit number
 - Title
 - Section Number
 - Instructor
 - Description
 - Lecture Time

Brief Description:

The `Modify Grades` use case allows instructors to enter students' grades into the system for their class section.

Step-by-step Description:

1. The instructor selects the course they are teaching
 The `Search for Courses` use case is used.
2. The instructor clicks on the student they wish to add the grade to.
3. Instructor inputs the grade into the system.
4. System updates with the newly inputted grades.

Brief Description:

The `View Employee List` use case allows for administrators and instructors to view all staff currently employed.

Step-by-step Description:

1. The employee clicks on the view current staff button.
2. The system displays the following information for all employees:
 - Employee Name
 - Job Title
 - Part-time or Full-Time status, if applicable

Brief Description:

The `View Student List` use case allows administrators to view all students currently enrolled in the university.

Step-by-step Description:

1. The Student clicks on the view student list button.
2. The system displays the following information for all students:
 - Student Name
 - Student ID
 - Year Level

Brief Description:

The Add Employee use case allows for administrators to add a member to the system.

Step-by-step Description:

1. The administrator clicks on the Add Employee button.
2. Administrator enters the following information:
 - First Name
 - Surname
 - SIN
 - Date Of Birth
 - Home/Work Address
 - Employee ID
 - Part-time/Full-time status.
3. The administrator then clicks the Add button.
4. The system reflects on these changes and updates the system

Brief Description:

The Add Course use case allows for administrators to add a course that will be offered to the students.

Step-by-step Description:

1. The administrator clicks on the add course button.
2. Administrator enters the following information:
 - Subject
 - 4-digit number
 - Title
 - Section Number
 - Instructor
 - Description
 - Lecture Time
3. System updates with the new course information.

Brief Description:

The `Remove Course` use case allows for administrators to remove a course that is offered to the students.

Step-by-step Description:

1. Administrator searches for the course.
`Use View Active Courses` use case.
2. The administrator clicks the remove button.
3. System updates with the removal of the course. System updates the enrolled students' timetable to reflect the new change.

Brief Description:

The `Remove Employee` use case allows for administrators to remove an employee because of termination or any other reason.

Step-by-step Description:

1. The Administrator selects the remove employee button.
2. The system shows a list of all active employees.
3. User selects the employee to be removed.
4. System updates the employee list with the new changes.

IX. Software Process Model

For our project, we will be following the waterfall model. Although the waterfall model may not be as preferred compared to the agile or prototyping model methodologies because of the time constraints, we felt that it gave us flexibility. The waterfall model is nice as it allows us to complete a stage of development, realize there is an issue, and then move back to reassess and fix the issue before continuing on. The waterfall model is a widely used life cycle model used in software engineering. The model consists of 4 general stages: requirements, analysis, design, and implementation [4]. With this model, we cannot move on to the next phase until we have completed the current phase, so nothing overlaps. In the requirements phase, we explore and refine what kind of requirements are needed in the program, the deadline, as well as any restrictions to meet the client's needs. In the analysis phase, we analyze the specifications to generate models and documents to determine "what the product is supposed to do". At the end of this phase, we develop a blueprint called a Software Project Management Plan, which will describe the whole development process in detail. In the design phase, we break down the product as a whole into parts called modules, which we will then independently design. Lastly, in the implementation phase, we take all these modules that are then rigorously tested and coded separately. Afterwards, they are then integrated into the system. Finally, more testing is done on the project as a whole until the client is satisfied and the program is delivered in full.

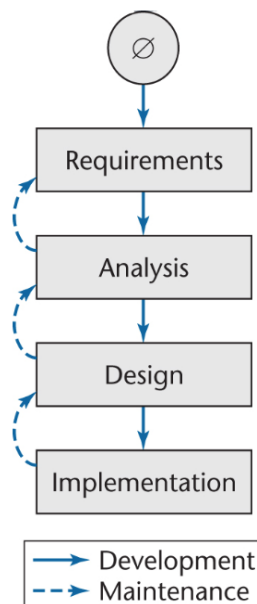


Figure 2: Waterfall Model Diagram

For our project, in the requirements phase, we determined what features are needed. In our program, we need to design and develop a University Registration System (URS) that will manage student and staff information. In addition, the program must allow users to browse for courses, register for courses, schedule courses, and enter marks. In the analysis phase, we developed a Software Project Management Plan, which displays crucial information about the project including the roles and responsibilities of each member. The cost and duration estimation, the description of risks involved as well as the solutions to these risks, and the style of documentation were also used in developing the Software Project Management Plan. In the design phase, we will be dividing the workload into 4 to allow each group member to equally design their respective modules. In the implementation phase, we will be getting together, either virtually or in-person, and combining all the modules to create our final product, which will then be tested until we deem it satisfactory.

X. Risks Involved

One risk that can occur is time constraint. Time constraints can occur if the work is not done in a timely fashion. Leaving the work to the last minute can be considered crunching, in which intense pressure to complete the project will occur in a short amount of time. This kind of pressure can lead to mistakes that could go unnoticed, causing the system to not be functional. This can be mitigated by completing the work on schedule and leaving time for proper revision and extensive testing. Another possible risk could be SQL injections. This can occur if the SQL statement inputs are not validated; therefore, leading to a potential security breach from malicious code. This can be mitigated by sanitizing the input statements. The use of stored procedures can also help mitigate SQL injection attacks. One more risk that can occur is associated with the cost of the system. It is possible that the system development doesn't follow the cost estimation guidelines. This can lead to a cost deficit, resulting in more money being spent than anticipated. This can be mitigated by following the cost estimation guidelines initially outlined in the planning stage while also updating the cost estimation throughout the life cycle.

XI. Design Overview

The goal of how we designed our system was to reduce the amount of time spent doing tasks and provide a robust interface to gather as much necessary information as possible. In our design process, we focused on achieving a simplistic interface that can provide all the functionality required from a university registration system. With that in mind, we had to carefully plan our user interfaces as well as the functions of the system to create something that would satisfy both of our constraints. For the design of our project, we plan on using The General Hierarchy design pattern, as described in [5], since our system has a natural hierarchy between our classes. All actors in our system are university members and those can be classified into students and employees. In addition, employees can be further classified into an instructor or an administrator. Our database will have tables for our information that relate to each other. We have the university member table which shares the member ID with the course, course section, course grades, course list and login table. The course table is related to the course section, course grade, and course list table which uses the course name and member ID from the course table. In short, all our classes and tables in the database have a natural hierarchy to them too which shows that The General Hierarchy design pattern best suits our system. When designing our system, we also wanted to keep in mind the idea of reusability so we can sell to more than one university. If we only have to change minor appearances, like having a university colour scheme and logo, and the backend code and algorithms would be the same, then this will help save cost when deploying to other universities.

XII. Software Architecture

A. Java Implementation

Our system was designed using a front-end CASE tool called SceneBuilder. This allowed us to create different UI interfaces called scenes to which we can add text fields, scroll boxes, buttons, and labels to. Scenebuilder uses JavaFX in the controller classes to allow us to add functionality to the scenes. With that, we created a controller class for each scene in our system. Each controller class can pass information to another controller class. For instance, when a user logs in and they are a student, the login controller will send the member ID of the student to all the scenes that require that information to process inputs. Lastly, we created the main class that is our driver code to initiate the login screen for the users of the system.

A. Database Model And Diagram

The best model to use would be the relational database model. This is because the relationships will be many-to-many, one-to-one, or many-to-one. As shown in the diagram in Figure 3, the UniversityMember-to-Section relationship is many-to-one. This means that there can be many students that register for a section of a course but only one section of a course per student. The Login-to-UniversityMember relationship is one-to-one. This means that there will only be one login for each university member. The Course-to-Section relationship is one-to-many. This means that a course can have many sections but needs at least one. The Section-to-CourseGrades relationship is one-to-many as well. This means that in each section, there can be many course grades but each course grade relates to only one section. The Section-to-CourseList relationship is one-to-many because each section will have many university members, specifically student types, and each student will only be in one section of a course.

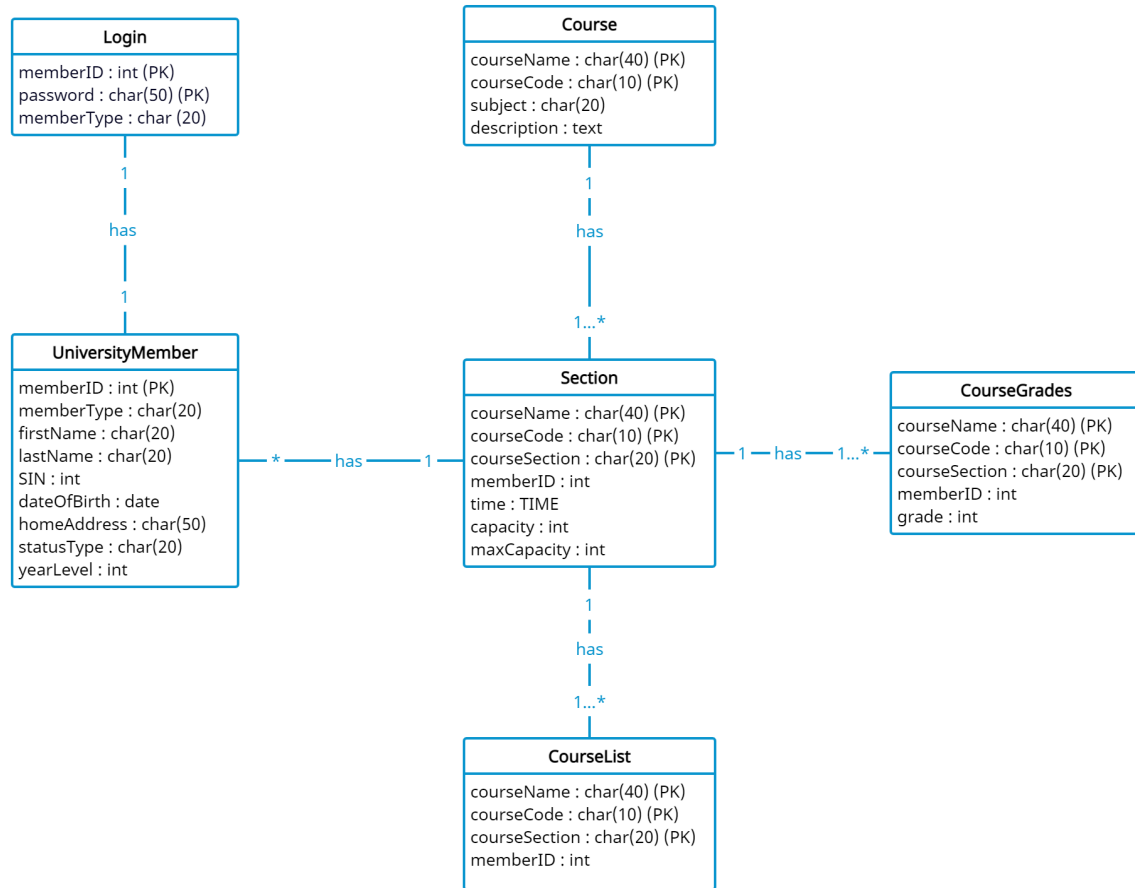


Figure 3: Entity-Relationship Diagram

B. Interaction Diagram

The interaction diagram in Figure 4 shows the sequence of interactions for how the system's classes will interact with each other when the student is attempting to log into the system.

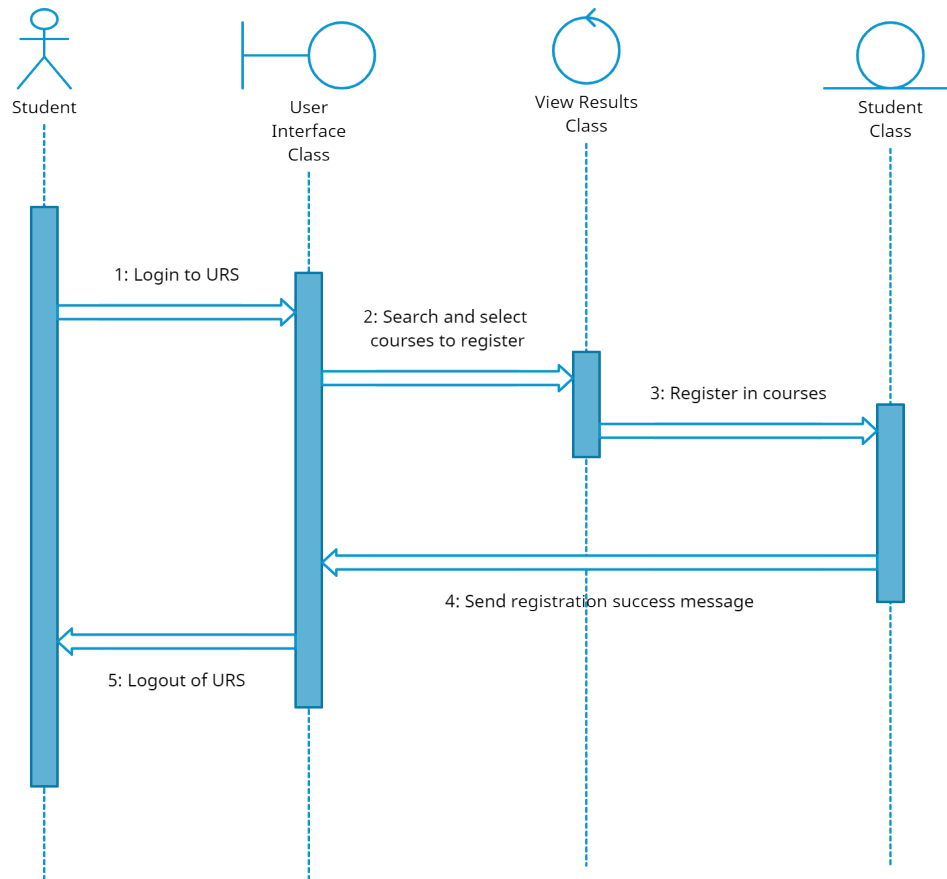


Figure 4: Sequence Diagram for a student searching and enrolling for a course

C. Communication Protocols and Messaging Format

We used the client-server architecture to implement the communication protocols and messaging format in our system. Since our system is going to be application-based, users will be using the internet which requires remote access to gain access to the system. In addition, by using this architecture, we can distribute the work across different machines and even design the client-side and server-side separately. All the data will be stored in a centralized database and can be distributed to whichever user requires it. Lastly, the client-server architecture allows for simultaneous access from multiple users that are accessing information for our database.

The server's main activities start with initialization and then it will begin listening for client connections. The server will send a message to a client when the connection is successful between them and it also handles the disconnection of them too. The client's main activities start with initialization as well and then it begins the connection process with the server. From here, it

will communicate with the user and send messages/actions to the server based on the user's requests. When the user is done and wants to log out, the client will send the disconnection message to the server and the server will terminate the connection cleanly.

The system will be a Thin-Client System because the client side will not be responsible for any heavy computations, this will be done on the server-side in the database where queries will be made and specific information will be sent back to the client. The client and server will use Internet and Transmission Control Protocols, IP and TCP, to communicate with each other. The Internet Protocol will allow us to send messages between the client and server, which includes breaking apart longer messages to make sure they are sent successfully. Meanwhile, the Transmission Control Protocol will handle the connection between the client and the server which includes the sending and receiving of messages as well as making sure they are sent or received successfully. Each client will have a unique IP address and could share a hostname while the server will have a port number assigned to it. For a client to initiate this connection, the client will need the hostname and port number to be able to connect to the server.

D. User Interfaces (GUI), Software Interfaces

In this section, we show our final GUI design as well as some outputs from using the system. First will be the initial login screen as shown in Figure 5, where the user will log in with their credentials and be brought to the appropriate screen depending on if they are a student, instructor or administrator.

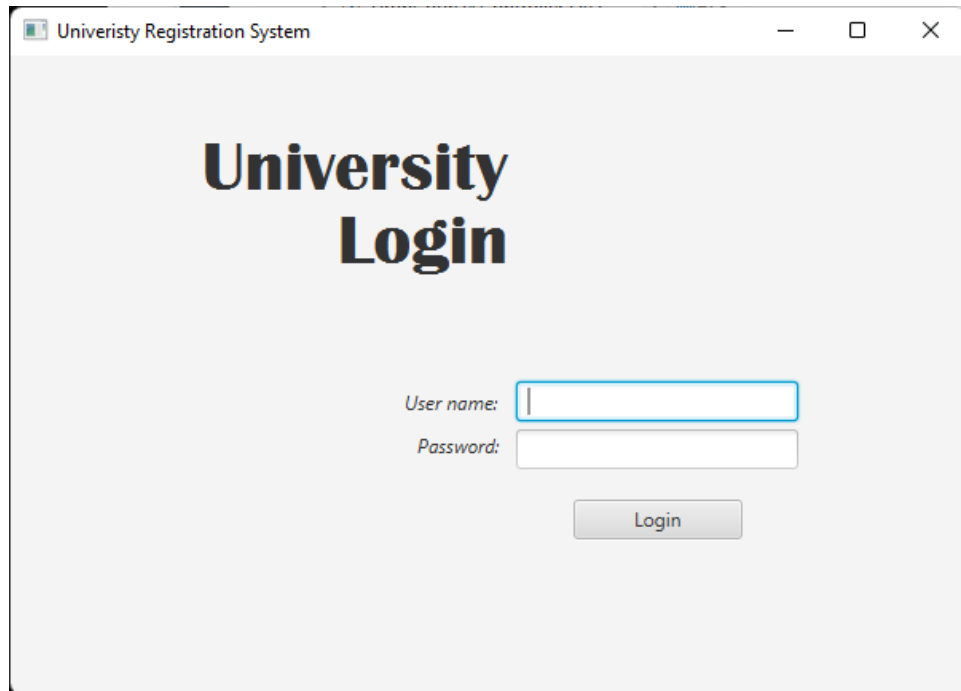


Figure 5: Login Interface

If a student logs into the system, they will be presented with the screen shown in Figure 6 where they can search and register for courses, view enrolled courses, view their grades and drop a course. In addition, we added a “Hello, ...” label that will dynamically change for whoever is logged in to add some personality to the system.

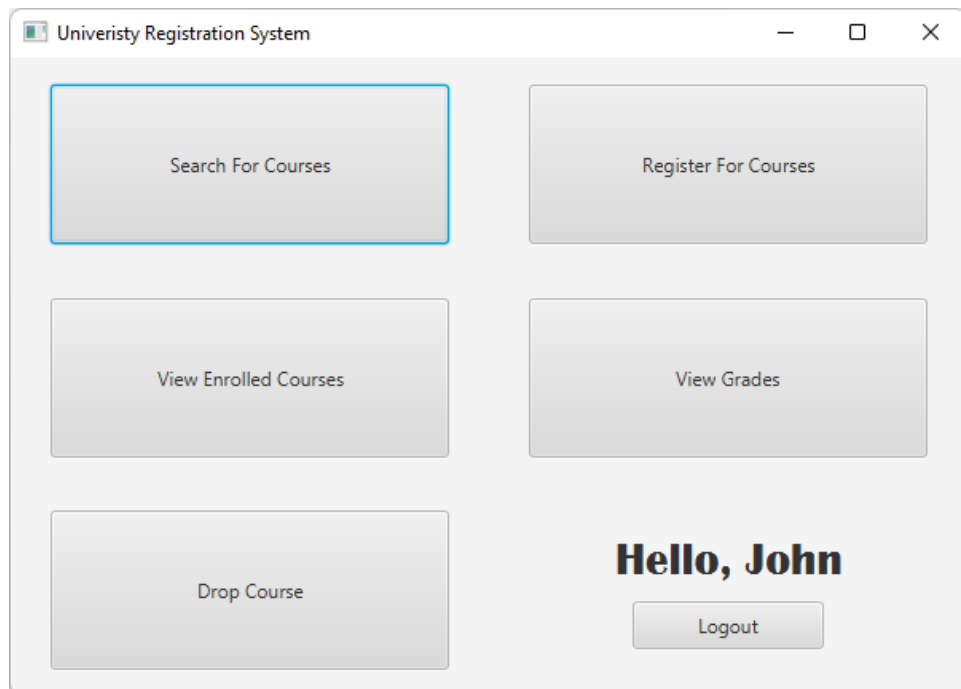


Figure 6: Welcome screen for when a student logs in

When a student chooses the “Search For Courses” button, they will be presented with the screen shown in Figure 7, where they can enter and search by a subject, course number, section or keyword. The server will then relay the search results and put all the information about the courses that fit that criteria in the scroll box. Results will be displayed as shown in Figure 8.

Univeristy Registration System

Search for Courses

4-Digit Name

Code

Section

Keyword

Search

Back

Figure 7: Search For Course Interface

Univeristy Registration System

Search for Courses

4-Digit Name

Code

Section

Keyword

Search

Back

ESOF-3050-FA
Software Engineering
 Instructor: Ken Ironside
 A project oriented course in which students apply software engineering principles of requirements elicitation, specifications, design, implementation and testing to solve engineering problems. The course content focuses on object oriented methodology and the use of Unified Modeling Language (UML) to specify, visualize, construct and document the artifacts of the software system. Topics include: concepts of object orientation; UML modeling and class diagrams; developing software requirements; client-server architecture; software design patterns; software implementation and testing; basic architectural patterns.

ESOF-3050-FB
Software Engineering
 Instructor: Ken Ironside

Figure 8: Search For Course Interface with results shown

If a student instead chooses the “Register For Courses” button, they get a similar screen as shown in Figure 9, but now the search will return all information minus the course description and there will be checkboxes beside each course. When results have been displayed on the screen, if the Class has zero remaining seats or the class year level is greater than the student year level, then they will be unable to enroll in it and it will be grayed out, as seen in Figure 10 with the class EDUC-6411-FA. When you select the checkbox beside the course, you will see the course display in the darker portion on the right side of the screen. From here, when you click on the enroll button you will see a message display as shown in Figure 11.

The screenshot shows a web application window titled "Univeristy Registration System". Inside the window, the main heading is "Register For Courses". On the left side, there is a vertical form with the following elements: a label "4-Digit Name" above a text input field, a label "Code" above a text input field, a label "Section" above a text input field, a label "Keyword" above a text input field, a "Search" button, an "Enroll" button, and a "Back" button. The main content area is divided into two vertical sections. The left section is currently empty, while the right section is a solid gray rectangle, indicating that search results are not yet displayed.

Figure 9: Register For Course Interface

The interface is titled "Univeristy Registration System" and "Register For Courses". On the left, there are input fields for "4-Digit Name" (containing 'e'), "Code", "Section", and "Keyword", followed by "Search", "Enroll", and "Back" buttons. The main area displays a list of courses with a vertical scrollbar on the right. The visible courses are:

- CHEM-2211-FA
Organic Chemistry I
☒ Lecture Time: 14:30:00
Instructor: Bill Gates
Seats Remaining: 16/35
- EDUC-3910-FA
Global Citizenship Education
☐ Lecture Time: 16:00:00
Instructor: Thomas Andre
Seats Remaining: 45/45
- EDUC-6411-FA
Cognition and Learning
☐ Lecture Time: 18:45:00
Instructor: Chadwick Boucher
Seats Remaining: 20/20
- EMEC-1112-FA

The right side of the interface is a large grey rectangular area.

Figure 10: Register For Course Interface with results shown.

The interface is titled "Univeristy Registration System" and "Register For Courses". The input fields and buttons on the left are the same as in Figure 10. The main area is now empty. At the bottom of the interface, the text "Successfully Enrolled!" is displayed.

Figure 11: Register For Course Interface when the enroll button is clicked.

The view enrolled courses screen will display the course name and title along with the lecture time and the instructor for the course as shown in Figure 12.

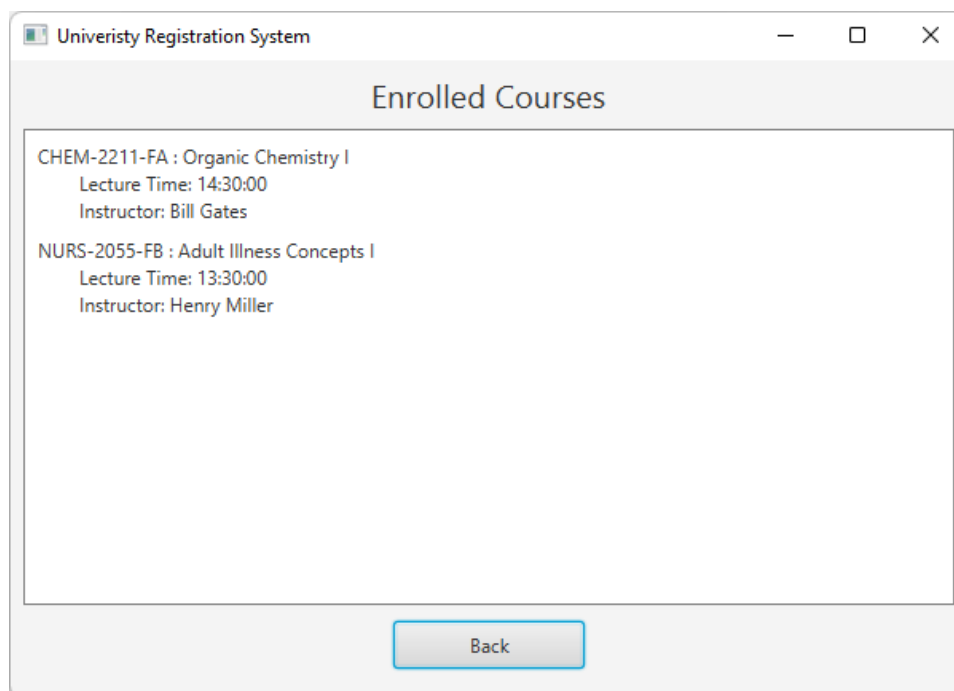


Figure 12: Student viewing their enrolled courses.

The view course grades screen will display the course name along with the student's grade currently in the course as shown in Figure 13.

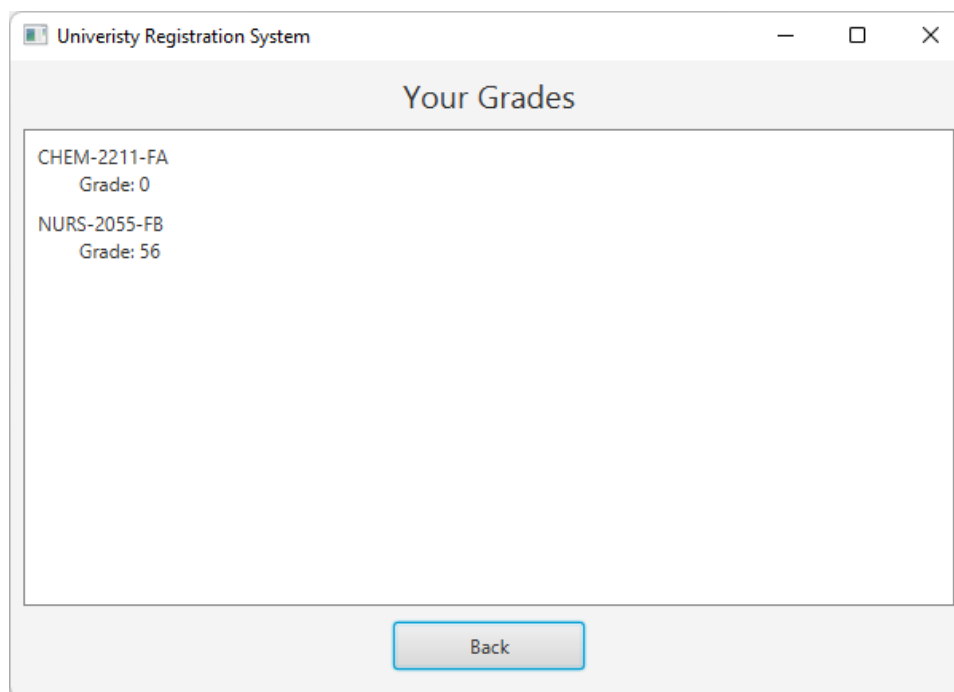


Figure 13: Student viewing their current grades.

The last interface for the student perspective is the drop course screen. Here, the student will be presented with their current courses that are eligible to be dropped along with a checkbox

beside them, the student will check off all the courses that they would like to drop and click the drop button as shown in Figure 14.

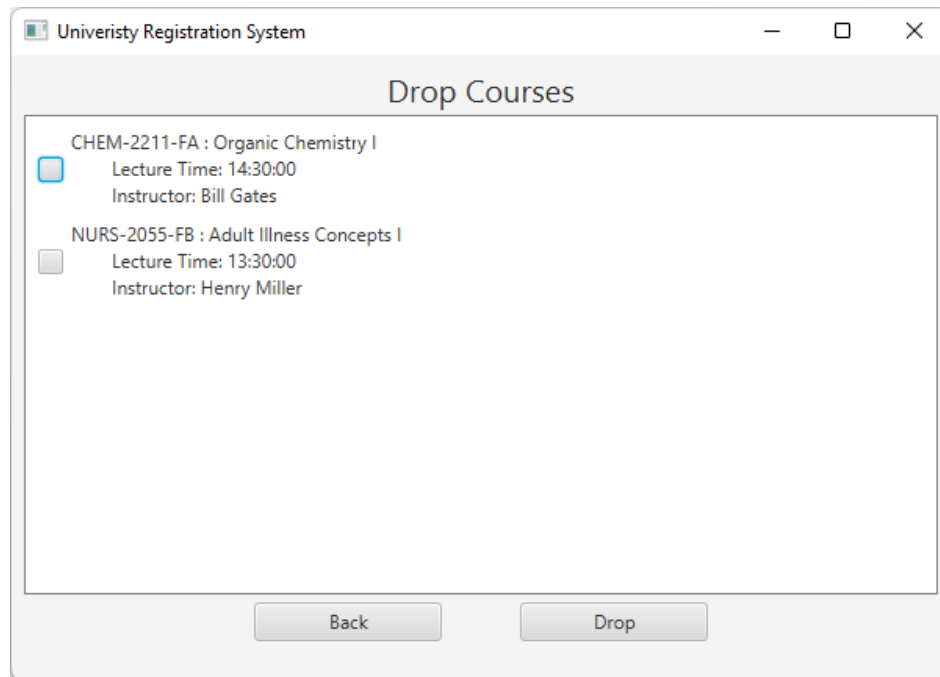


Figure 14: Drop Course Interface

If an instructor were to log in to the system now, they would get a screen as shown in Figure 15 where they can view their courses, modify the grades of the courses they teach, and view the list of employees and students.

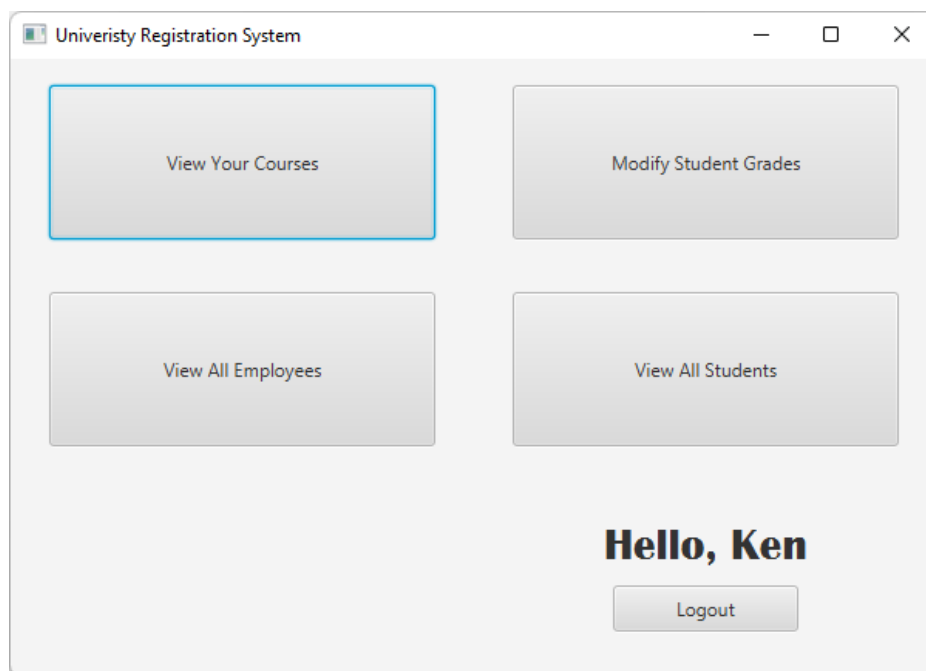


Figure 15: Welcome screen for when an instructor logs in

The view your courses screen will display a list of courses that the instructor is currently teaching at the university as shown in Figure 16. It will display the course name and title along with the lecture time and the capacity.

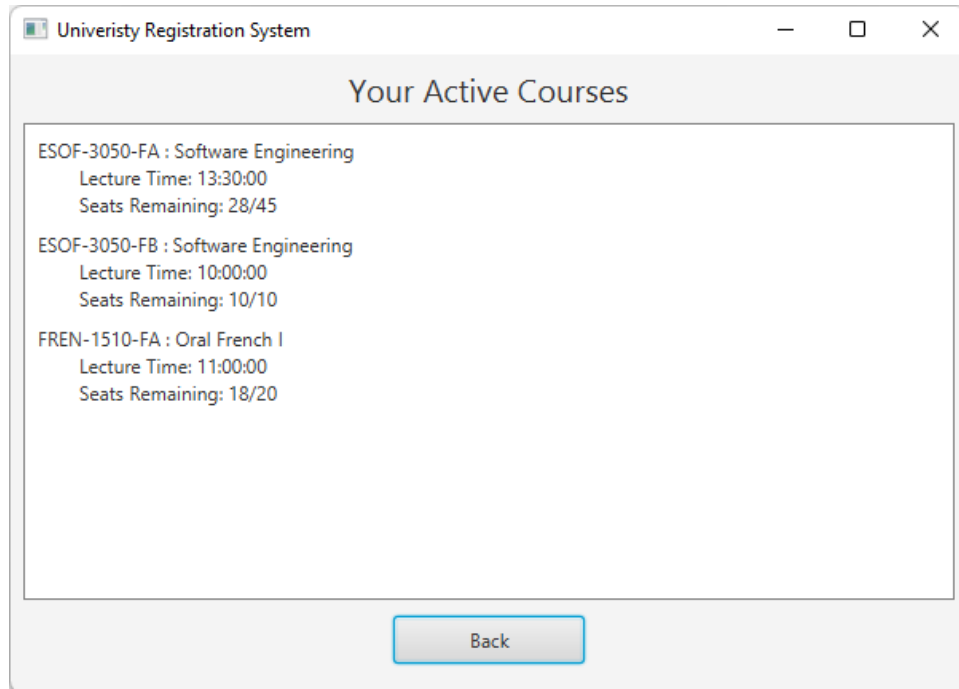


Figure 16: View Active Courses Interface

The modify grades screen will show all the grades that are taught by the current instructor as shown in Figure 17. In the column in the middle, it will show the students that are enrolled in that selected course as shown in Figure 18. Choosing a student they will be shown their current grade and then the new grade that you would like to input as shown in Figure 19.

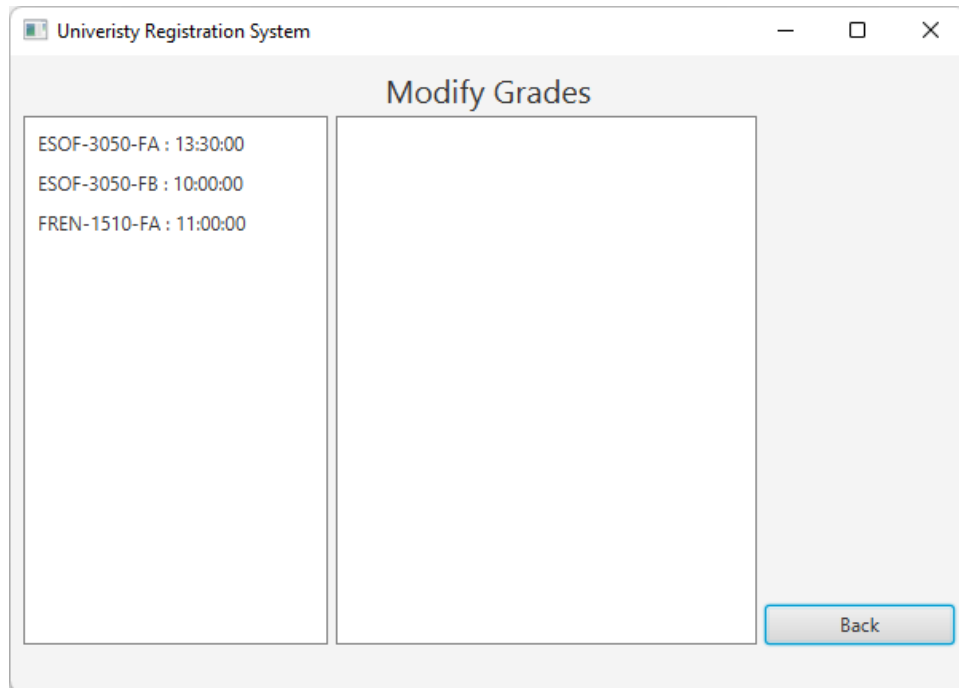


Figure 17: Initial Modify Grades Interface

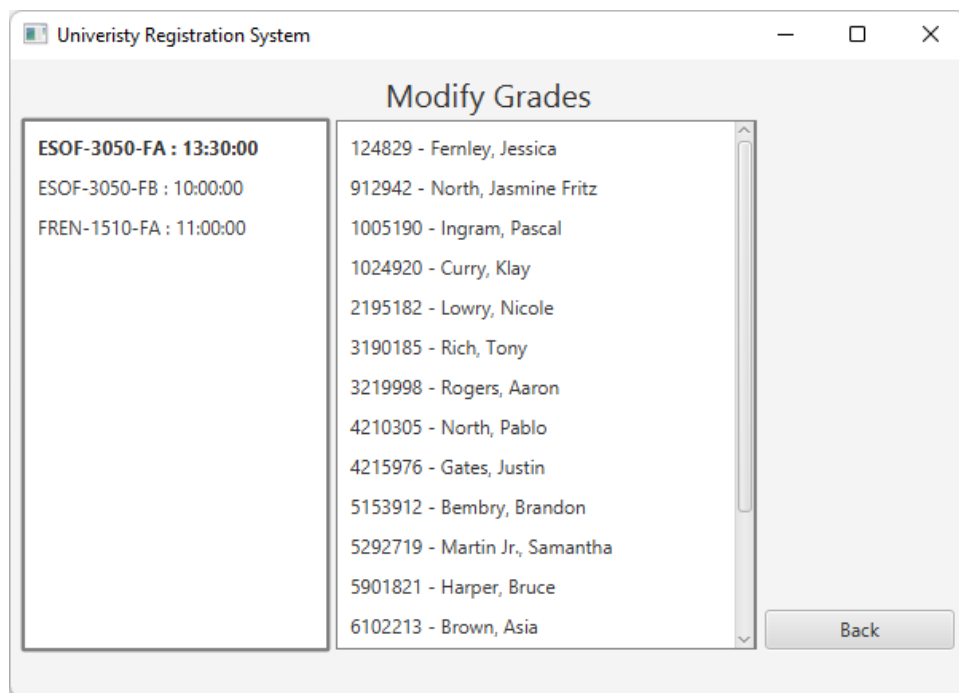


Figure 18: Modify Grades Interface with a course selected

Univeristy Registration System

Modify Grades

ESOF-3050-FA : 13:30:00
ESOF-3050-FB : 10:00:00
FREN-1510-FA : 11:00:00

124829 - Fernley, Jessica
912942 - North, Jasmine Fritz
1005190 - Ingram, Pascal
1024920 - Curry, Klay
2195182 - Lowry, Nicole
3190185 - Rich, Tony
3219998 - Rogers, Aaron
4210305 - North, Pablo
4215976 - Gates, Justin
5153912 - Bembry, Brandon
5292719 - Martin Jr., Samantha
5901821 - Harper, Bruce
6102213 - Brown, Asia

Current Grade: 88%
New Grade:

Change
Back

Figure 19: Modify Grades Interface with a student selected

The view all employees screen and view all students screen will be the same from both an Instructor and an Administrator viewpoint. On both screens, the user will be displayed all the information of either all of the employees as shown in Figure 20 or all the students as shown in Figure 21.

Univeristy Registration System

Employee List

Andre, Thomas
Instructor
ID: 2419522
SIN: 298960408
Date of Birth: 1966-02-28
Address: 829 Third Street, Thunder Bay, ON, P0B 6G2, Canada
Status: Full-Time

Boucher, Chadwick
Instructor
ID: 2419215
SIN: 638799434
Date of Birth: 1961-09-27
Address: 941 Bankers Street, Thunder Bay, ON, P5X 7J3, Canada
Status: Part-Time

Back

Figure 20: View All Employees Interface

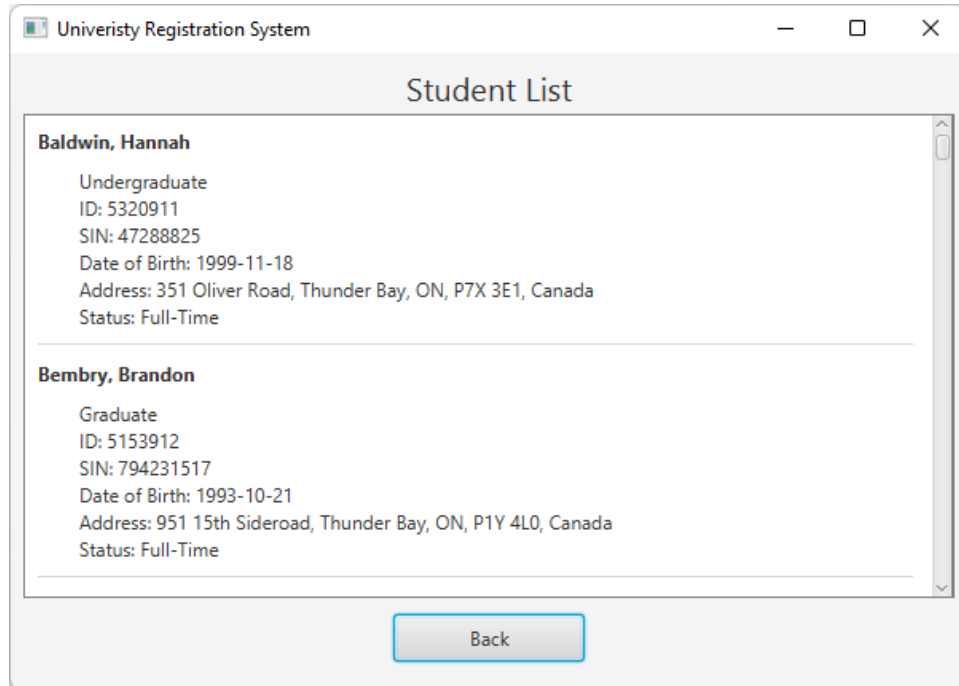


Figure 21: View All Students Interface

If an administrator were to log in to the system now, they would get a screen as shown in Figure 22. They can view the employee and student list, add and remove employees as well as add and remove courses.

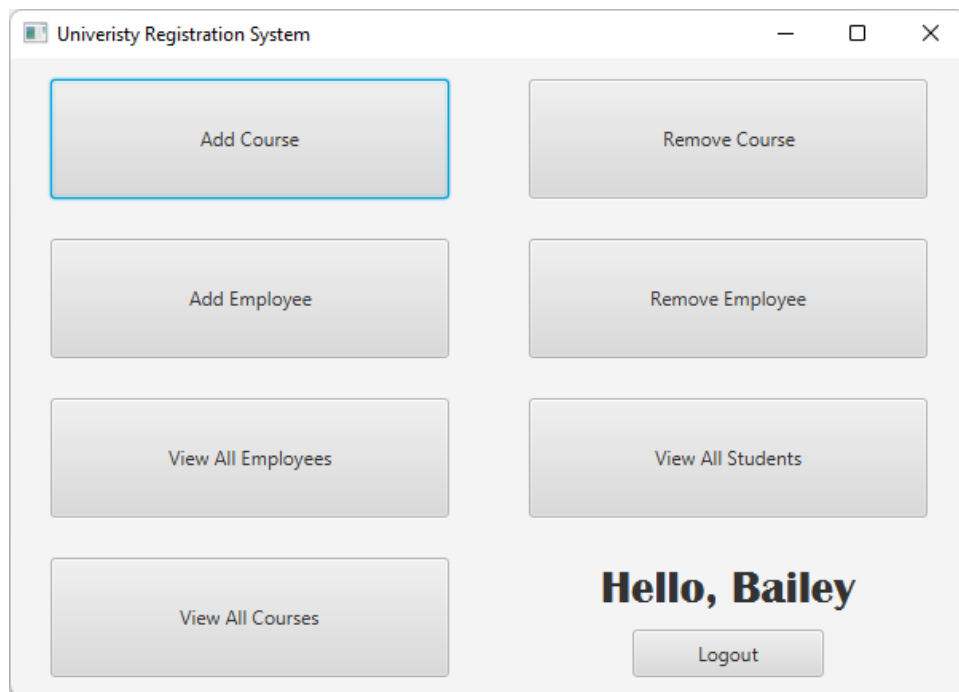
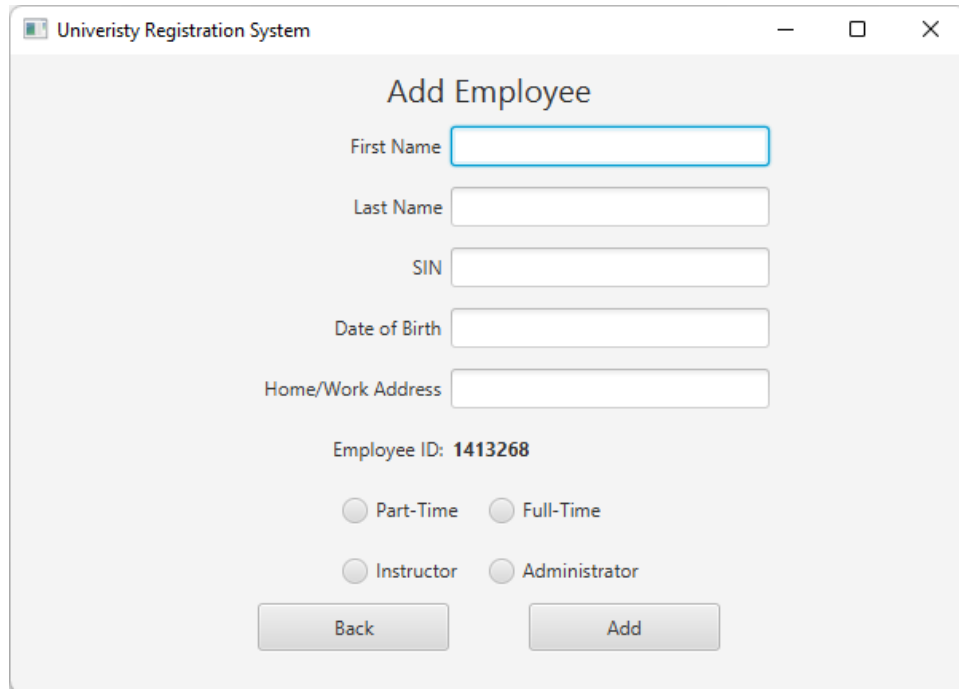


Figure 22: The welcome screen for when an administrator logs in

The add employee interface will provide input fields to the administrator for all the required information about the employee as seen in Figure 23. In addition, an employee ID is automatically generated every time this screen is opened



The screenshot shows a web application window titled "Univeristy Registration System". Inside the window is a form titled "Add Employee". The form contains the following fields and controls:

- First Name:
- Last Name:
- SIN:
- Date of Birth:
- Home/Work Address:
- Employee ID: 1413268 (displayed in bold)
- Part-Time: ☐
- Full-Time: ☐
- Instructor: ☐
- Administrator: ☐
- Back:
- Add:

Figure 23: Add Employee Interface

The remove employee screen will show a list of all employees with checkboxes beside their names, the administrator would select all the employees they wish to remove and click remove as shown in Figure 24.

Univeristy Registration System

Remove Employee

- ☒ Andre, Thomas - 2419522 - Instructor
- ☐ Boucher, Chadwick - 2419215 - Instructor
- ☐ Davis, Precious - 5129851 - Administrator
- ☐ Emperius, Nicholas - 2192100 - Instructor
- ☐ Ford, Harrison - 3450968 - Instructor
- ☐ Game, Squid - 7544327 - Instructor
- ☐ Gates, Bill - 7821942 - Instructor
- ☐ Hemsworth, Christina - 7892152 - Instructor
- ☐ Hull, Bailey - 9000546 - Administrator
- ☐ Ironside, Ken - 646314 - Instructor
- ☐ James, Shaquille - 2199921 - Administrator
- ☐ Jobbs, Steve - 5678762 - Instructor

Back Remove

Figure 24: Remove Employee Interface

The add course screen will provide input fields to gather all the necessary information about the course that is being added to the system as shown in Figure 25. A drop-down will populate with all instructors that are able to be taught for this course as shown in Figure 26.

Univeristy Registration System

Add Course

Course Code

4-Digit Number

Title

Section

Instructor

Description

Lecture Time

Capacity

Back Add

Figure 25: Add Course Interface

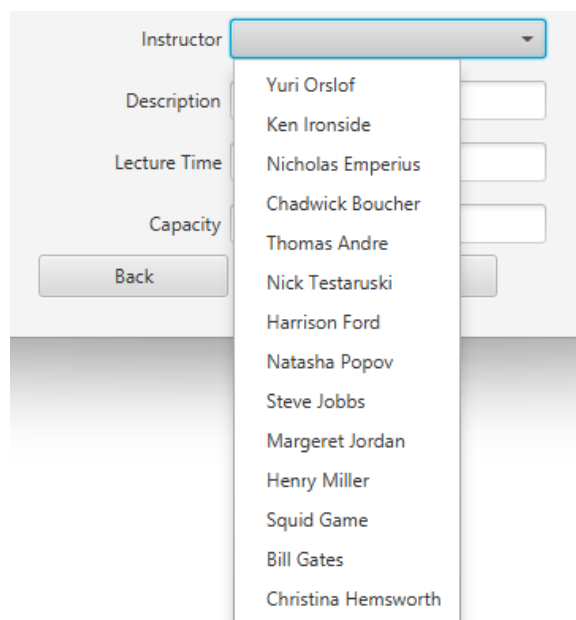


Figure 26: Drop down menu in the Add Course Interface

Lastly, the remove course screen allows administrators to remove courses from the system. Similar to the remove employee screen, there are checkboxes located beside the employee, select all the employees that you wish to delete and they will be gone.

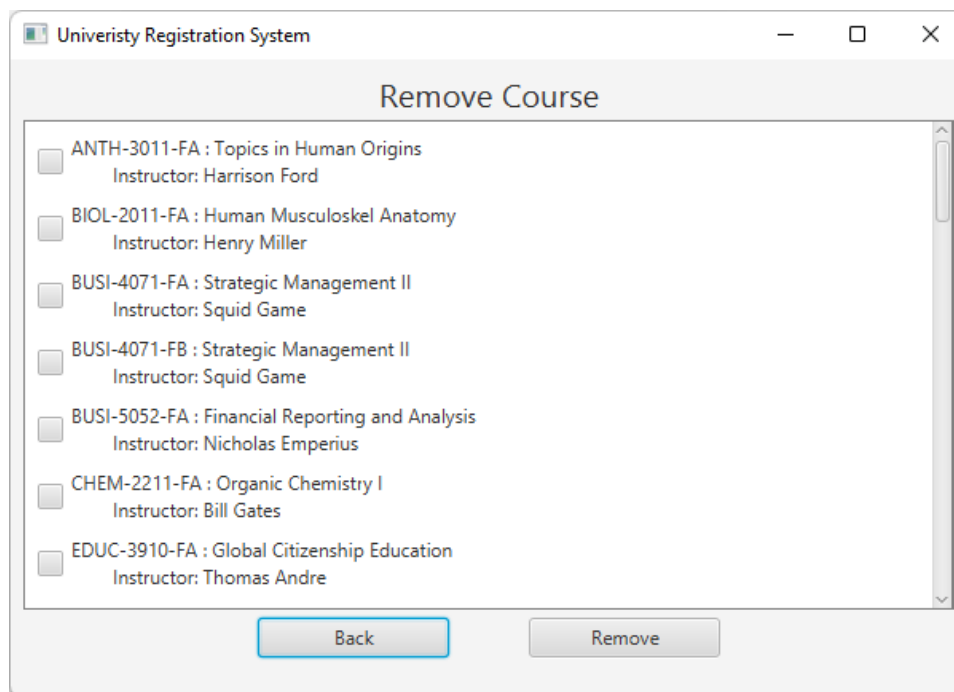


Figure 27: Remove Course Interface

XIII. Object-Oriented Design

A. Complete Class Diagrams

In Figure 28 below, we have a complete class diagram of our system. As you can see, we have 5 entity classes, 2 control classes and 2 boundary classes. The entity classes represent the main objects in our system, namely university members, students, employees, instructors, and administrators. According to [6], entity classes are usually nouns and represent data, control classes are verbs and connect the entity and boundary classes, and lastly, boundary classes are also nouns but mainly interfaces that the actor uses.

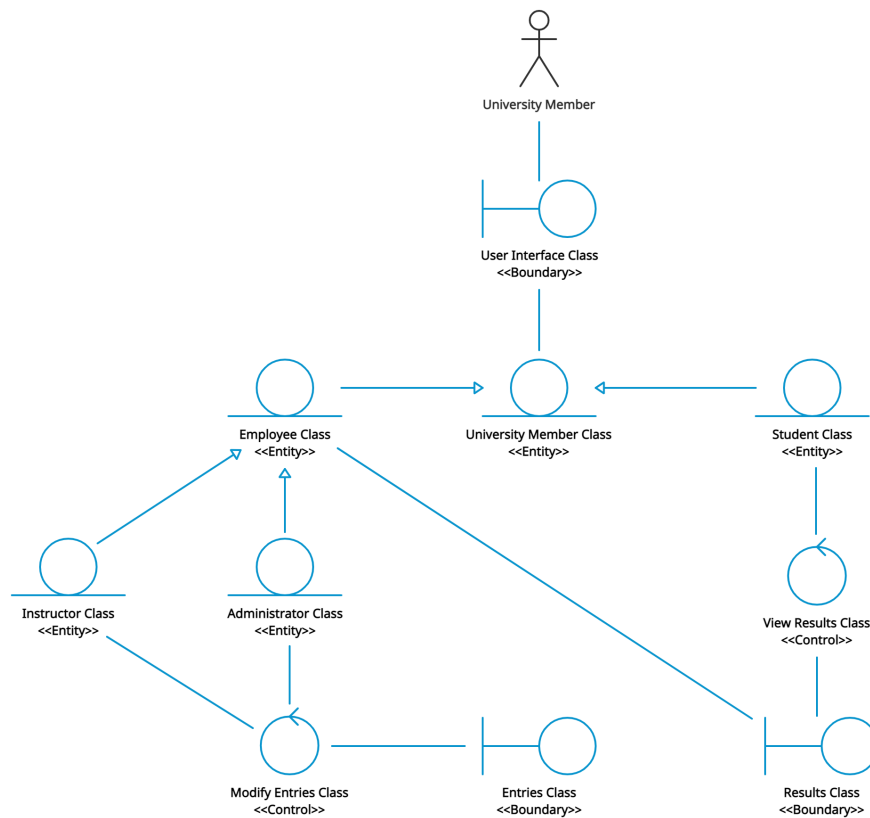


Figure 28: The overall class diagram of our system

In Figure 29, we have a class diagram showing the specific attributes of each method that contains variables within it. The prefixes in the following are - for *private* and + for *public* variables or methods.

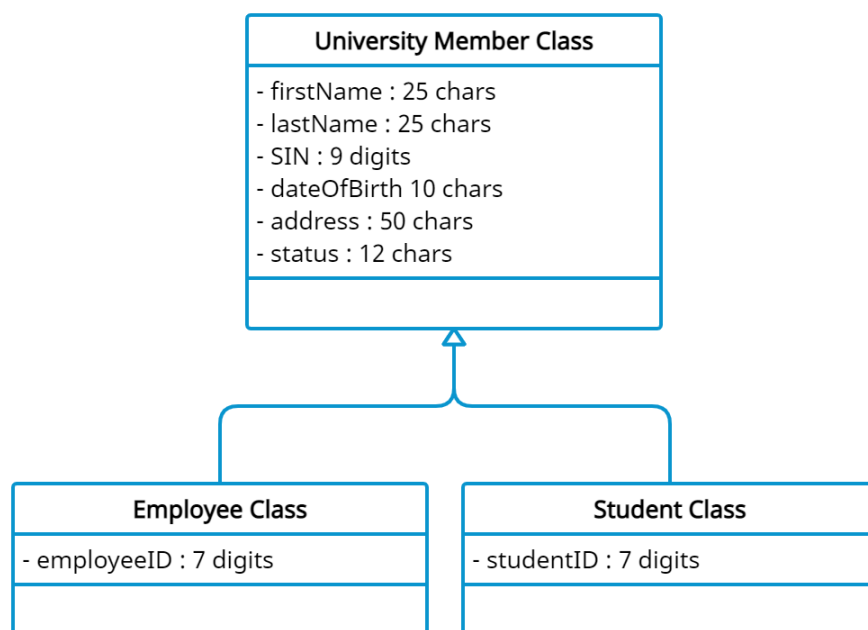


Figure 29: The overall class diagram with the format of the attributes specified

The following classes are the methods and attributes shown in a programming design language (PDL) for Java in alphabetical order. We have also included indicators of which classes are an entity, control, or boundary classes too.

<<entity class>> Administrator Class
+ addEmployee (employee : universityMember, employeeID : int) : void + removeEmployee (employeeID : int) : void + addCourse (subject : string, courseCode : string, courseName : string, courseSection : string, instructor : universityMember, courseDesc : string, courseTime : string) : void + removeCourse (subject : string, courseCode : string, courseSection : string) : void

<<entity class>> Employee Class
- employeeID : int
+ getEmployeeID () : int + setEmployeeID (employeeID : int) : void + viewEmployees () : void

+ viewStudents () : void + viewCourses () : void

<<boundary class>> Entries Class
+ showEntriesInterface () : void

<<entity class>> Instructor Class
+ modifyGrades (studentID : int, newGrade : int) : void

<<control class>> Modify Entries Class
+ updateEmployeesTable () : void + updateCourseTable () : void + updateCourseGradesTable () : void

<<boundary class>> Results Class
+ showResultsInterface () : void

<<entity class>> Student Class
- studentID : int
+ getStudentID () : int + setStudentID (studentID : int) : void + searchForCourse (subject : string, courseCode : string, courseSection : string) : void + registerForCourse (subject : string, courseCode : string, courseSection : string) : void + dropCourse (subject : string, courseCode : string, courseSection : string) : void + viewEnrolledCourses () : void

+ viewGrades () : void

<<entity class>> University Member Class	
- firstName : string - lastName : string - SIN : int - dateOfBirth : string - address : string - status : string	
+ getFirstName () : string + setFirstName (firstName : string) : void + getLastName () : string + setLastName (lastName : string) : void + getSIN () : int + setSIN (SIN : int) : void + getDateOfBirth () : string + setDateOfBirth (dateOfBirth : string) : void + getAddress () : string + setAddress (address : string) : void + getStatus () : string + setStatus (status : string) : void + login () : void	

<<control class>> View Results Class	
+ queryCourseTable () : void + queryCourseGradesTable () : void + queryEnrolledCoursesTable () : void	

B. Algorithms Used and Complexity

Most of the methods found in our system will have a time complexity of $O(1)$ and other methods that contain loops will have a complexity of $O(n)$. There is one occurrence of a nested join that is used, which is a complexity of $O(n^2)$. When researching the complexity of SQL queries, we found that on average they could range from $O(1)$ to $O(n^2)$ in complexity [7]. For instance, a simple SELECT statement to get the entries in a table is $O(1)$ if we have a number to count to, compared to using a primary key in the search, it would give complexity of $O(\log n)$.

Our java code will not have any algorithms to do sorting of information. This will be done through SQL queries and the information received from the database will be outputted directly to the user interface. In our Java code, we have classes that implement functions with nested loops which at worst case would be $O(n^2)$. In conclusion, our system has a worst-case complexity of $O(n^2)$.

C. Detailed Design

Figures 30, 31, and 32, are pseudocode that show a more detailed analysis of some of the methods found in our system code.

```

public void searchCourses () throws SQLException
{
    store the values from the text fields in local variables
    create 'SELECT' query string from input values
    try (create a connection with database)
    {
        execute query

        foreach (result : the set of results from the query)
        {
            store the individual string values in a variable
            output the result to the text field in the GUI
        }
    }
    catch (SQLException e)
    {
        handle the error message
    }
}

```

Figure 30: Detailed Pseudocode of the searchCourses method found in the student class

```

public void addEmployee() throws SQLException
{
    store the values from the text fields in local variables

    create 'INSERT' query string from input values

    try (create a connection with database)
    {
        execute 'INSERT' query
    }

    catch (SQLException e)
    {
        handle the error message
    }
}

```

Figure 31: Detailed Pseudocode of the addEmployee method found in the administrator class

```

public void modifyGrades() throws SQLException
{
    store the new grade from the text field in a variable

    create 'UPDATE' query string from input values

    try (create a connection with database)
    {
        execute 'UPDATE' query
    }

    catch (SQLException e)
    {
        handle the error message
    }
}

```

Figure 32: Detailed Pseudocode of the modifyGrades method found in the instructor class

XIV. Activity Diagrams

Concurrent activities are activities that occur at the same time. For example, if we have multiple users trying to register for the same course at the same time when there is only one available spot left for that section. When this happens, it is defined as a race condition. A race condition is a situation in which two or more threads try to access and modify the same data at the same time. Within our database, there are multiple situations in which a race condition can occur. These conditions include multiple students registering for a course with one open spot, multiple administrators trying to remove a course at the same time, multiple administrators trying to remove an employee at the same time, and logging in with the same account on multiple devices. With the other use cases, there would not be any race conditions, since information is only being accessed and not modified. In Figure 33, there is an activity diagram of the register for course activity. It will first check if there is someone already accessing the information to prevent a race condition. If it is not being accessed, then we check if there are available spots in the class. If there is room, then the system will concurrently increment the course size and update the database with the registration information.

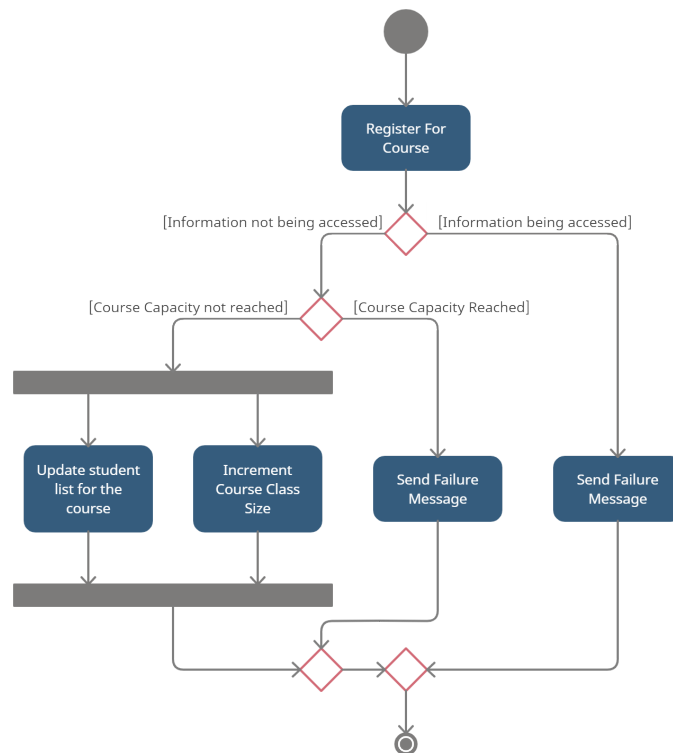


Figure 33: Activity Diagram for registering for a course

In Figure 34 below, there is an activity diagram to show the removed employee activity. Like the register for course diagram, we want to make sure that no other administrator is modifying the database at the same time. If no one else is, then concurrently the database will update all records containing that employee and then remove the employee from the list of all employees.

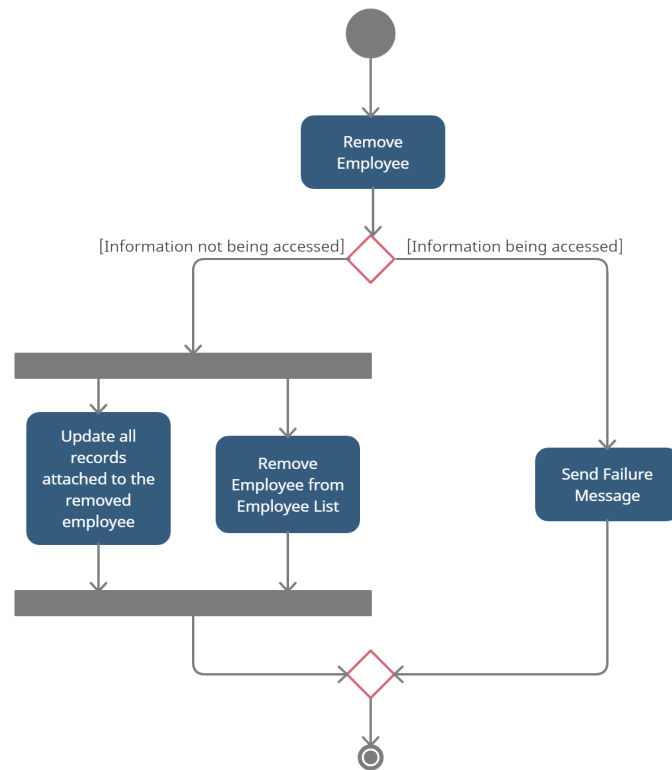


Figure 34: Activity Diagram for removing an employee

In Figure 35, the search for courses activity is shown. The concurrent activities that take place in this activity are displaying results to the user and querying for the information which happens at the same time.

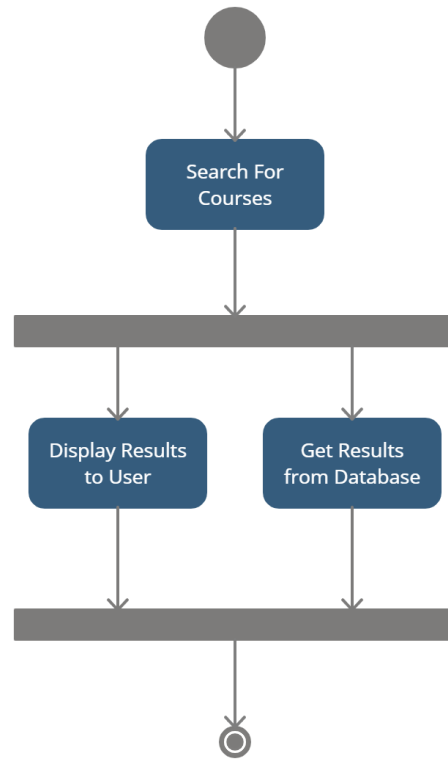


Figure 35: Activity Diagram for searching for a course

XV. Design Impacts

As we are developing a University Registration System (URS), the market for this type of software can be very profitable. Universities are large businesses, with each handling millions of dollars every year. In Canada alone, there are 280 post-secondary institutions. Within these institutions, there is a need for a reliable and responsive registration system to allow students, professors, and other staff to access information relating to courses, classes, and staff. As previously mentioned in our Software Project Management Plan, the rough estimated cost of developing our system is ~\$90,000. After developing the URS, it would be much quicker and more efficient to reuse our code for other systems, with some slight variation based on each school. If we were to market our URS to even half of these schools, we would be looking at a revenue of \$12.6 million, not to mention post-development maintenance and updates. From the perspective of the universities, we would be providing a reliable and simplistic registration system, potentially much better than their previous software, that would allow for efficient use for both students and staff. A societal impact can be the ease of access to utilize the functionalities of the system for both students and staff members. A more streamlined system can reduce bad experiences that can occur when using a registration system. An example of this can be when registration for classes opens, the server can handle all of the traffic without crashing or slowing down, leading to a more pleasant experience.

XVI. Test Approach

Initially, we tested our system using a black-box approach with different group members to try and observe the system without the user knowing any internal implementation of what is being tested. As we progressed near the completion of the system, we started basing our test cases off a glass-box approach. To do this, we must test key areas of our system, the utility, reliability, robustness, and performance will be the four main areas that we focus on during our testing. The utility should be tested first because without utility there is no purpose to what is being created. Reliability is an important aspect to test since we want to ensure that failures are minimal and if there are errors, we want to ensure that the amount of downtime is reduced as much as possible. Robustness ensures that the system can handle a wide range of allowable inputs and still provide a correct output such as a name having a special character in it and how that affects searching for that student. Lastly, performance is key to meeting time and space constraints. If the program exceeds the amount of storage space available, then it cannot be used. Testing is a critical process and must be carefully done not only in the development phase but also throughout the lifespan of the system to allow for updates.

XVII. Test Plan

A. Features To Be Tested

Client-Connection:

1. Connection to the server
 - a. This will be tested by attempting to log in to the system. If the login is successful, then that means that the user has a proper connection to the server.
2. Students
 - a. Register for courses
 - b. Drop registered courses
 - c. View enrolled courses
 - d. Search for courses
 - e. View registered course grades
3. Administrators
 - a. View active courses
 - b. View employee list
 - c. View student list
 - d. Add employee
 - e. Add course
 - f. Remove Employee
 - g. Remove Course
4. Instructors
 - a. View active courses
 - b. Modify grades
 - c. View employee list
 - d. View student list
5. Error Handling
 - a. Make sure that no errors occur that can crash the program. All errors must be handled without disruption.

B. Features Not To Be Tested

We will not be testing the GUI as it already works with local data and can show lists of output from the tables in the database. We believe that the GUI is simple and stable since it is mainly responsible for gathering input from the user while the back-end code is what we should be focusing our testing efforts on.

C. Testing Tools and Environment

To develop our GUI, we used SceneBuilder, which is a drag and drop program that allows us to create GUI's as a .fxml file. We can assign variable names to features like text fields and scroll panes as well as functions to be called when buttons are clicked. We can easily import these .fxml files into an IDE to add functionality. The IDE we are using is Eclipse, which will allow us to create our java code and implement the functionality required from our system. Our database, including our tables and queries, will be done in MySQL. We are also able to connect the MySQL database to our java code which allows us to simulate a client-server connection.

XVIII. Test Cases

Appendix A shows all of our test logs to go with their associated test case as labelled in this section. Some test logs indicate a failure of the test. This was during our testing phase and for our final system we have confirmed that each and every test case is a success.

A. Logging In

Purpose: Allow the user to log in to the system using their assigned login credentials consisting of their username and password.

Inputs: The user will enter their username and password into the text fields.

Expected Outputs: Upon clicking the login button, the user should be granted access to the system if the correct credentials are provided.

Pass/Fail Criteria:

Pass: The user can log in to the system using their credentials successfully.

Fail: The user is unable to log in to the system regardless of their credentials being entered correctly.

Test Procedure: Using a pre-existing account, enter the correct credentials of the user into the username and password text field. Click the login button and record whether the user can log into the system.

B. Searching For Courses

Purpose: Allows student users to search for all currently available courses.

Input: The user clicks on the ‘Search For Courses’ button and then inputs information about a course. Information consists of a course code, name, and section.

Expected Outputs: After hitting the search button, the program will display all available courses matching the criteria provided in the search.

Pass/Fail Criteria:

Pass: All available courses that match the search criteria provided will be displayed in our table.

Fail: Either all or some available courses that match the search criteria are not displayed.

Test Procedure: The student navigates to the ‘Search For Courses’ interface and will begin entering search criteria. Once the ‘Search’ button is clicked, the program will then display all available courses that match the criteria provided.

C. Registering For a Course

Purpose: Allow the user to register for any applicable course.

Inputs: The user will click on the register button when selecting a course.

Expected Outputs: Upon clicking the register button, the user should be registered to the course that they selected.

Pass/Fail Criteria:

Pass: The user successfully registers for the course. A status label will be used to show the outcome.

Fail: The user is unable to register for the course. A status label will be used to show the outcome.

Test Procedure: The user will select a course they wish to register for and then click on the register button. As long as the requirements are met by the student, the student will be able to enroll in the course. The course should now show up when the user views their enrolled courses.

D. Viewing Enrolled Courses

Purpose: To allow students to view all of the courses that they are currently enrolled in.

Inputs: Load with enrolled courses displayed.

Expected Outputs: A list of currently enrolled courses for that student is displayed.

Pass/Fail Criteria:

Pass: All currently enrolled courses and only courses that the student is enrolled in are displayed.

Fail: Not at all enrolled courses appear or courses the student is not enrolled in appearing.

Test Procedure: The student will select the ‘View Enrolled Courses’ button. The interface should change and then display a list of the courses that the student has enrolled in.

E. Dropping a Course

Purpose: Allow the user to drop any course they are currently registered in.

Inputs: The user will click on the drop button when viewing their course list.

Expected Outputs: Upon clicking the drop button, the user should be able to drop the course.

Pass/Fail Criteria:

Pass: The user successfully drops the course. A status label will be used to show the outcome.

Fail: The user is unable to drop the course. A status label will be used to show the outcome.

Test Procedure: The user will click on the 'Drop Courses' button which will take them to a separate page. The user will then click on the course they wish to drop and then click on the 'Drop' button to remove the course. Ensure the course is dropped by having the user view their enrolled courses to see if the course has been removed.

F. Viewing Course Grades

Purpose: Allow the student user to view their current grades in all of their courses

Inputs: The user will click on the 'View Grades' button.

Expected Outputs: Upon clicking the 'View Grades' button, the program will display the grades for all of the classes that a particular user enrolled in.

Pass/Fail Criteria:

Pass: The user is successfully able to view all of their grades.

Fail: The user is unable to view all of their grades, or they view the grades of another student.

Test Procedure: The user will select the 'View Grades' button. Once clicked, the program will then display grades for all courses the user is enrolled in.

G. Viewing Active Courses Offered

Purpose: Allow the employees to view all active courses being offered by the institution.

Inputs: The user will click on the 'View Active Courses' button.

Expected Outputs: Upon clicking the ‘View Active Courses’ button, the user should be able to see a list of courses currently being offered at the time.

Pass/Fail Criteria:

Pass: The user is successfully able to see the list of courses currently being taught.

Fail: The user is unable to see some, or all, of the courses currently being taught.

Test Procedure: The user will press the ‘View Active Courses’ button. A complete list of all active courses should be displayed. Check each course to ensure that none are missing.

H. Modifying Grades

Purpose: Allow instructors to enter or change the grades of the students in the classes that they are teaching.

Inputs: The instructor chooses a student and enters a grade for the student.

Expected Outputs: The grade for the student in that particular class will be changed.

Pass/Fail Criteria:

Pass: The student's grade is successfully updated in the database.

Fail: The student's grade remains unchanged.

Test Procedure: A list of students in the classes that are taught by the instructor is displayed in a list. The instructor chooses a student then proceeds to enter a new grade. The instructor will save the changes. Can repeat as many times as necessary.

I. Viewing the Employee List

Purpose: Allows the employee used to view all currently employed staff members.

Inputs: User clicks the ‘View Employee List’ button.

Expected Outputs: The program will display the names of all the currently active employees as well as their position and their type of employment, whether they are full-time or part-time employees, if applicable.

Pass/Fail Criteria:

Pass: The program displays all the currently active employees as well as information about them.

Fail: The program does not display the active employees or does not display all active employees.

Test Procedure: The user selects the ‘View Employee List’ button and the program then displays the names of all staff currently employed as well as their position and employment type.

J. Viewing the Student List

Purpose: Employees should be able to see the complete list of students that are currently enrolled at the university.

Inputs: The user will click on the ‘View Student List’ button.

Expected Outputs: The user should be taken to the View Student List page where the entire list of students should be visible.

Pass/Fail Criteria:

Pass: The user should be able to see the entire list of currently enrolled students at the university.

Fail: The list is missing some, or all, of the currently enrolled students at the university.

Test Procedure: Click on the ‘View Student List’ button to view the list of enrolled students. Ensure that all of the listed students are being displayed with no errors.

K. Adding a Member

Purpose: Allow administrators to add new members to the University Registration System.

Inputs: The new member’s first name, last name, SIN, date of birth, home/work address, part-time/full-time status. An ID will be automatically generated for the new member.

Expected Outputs: The system will display a status message indicating the member was added. The new member will be added to the system.

Pass/Fail Criteria:

Pass: The member is successfully added to the database.

Fail: The member is not added to the database.

Test Procedure: The administrator enters all the information about this new member. The system auto-generates a member ID and when the ‘Add’ button is clicked, the database is updated.

L. Adding a Course

Purpose: Allows administrators to erect a new course offering to the University Registration System.

Inputs: The administrator would input the subject, 4-digit number, title, section number, instructor, description, and lecture time.

Expected Outputs: The system would display a status message indicating the course offering was successfully added.

Pass/Fail Criteria:

Pass: The course is successfully added to the database.

Fail: The course is not successfully added to the database.

Test Procedure: The administrative user would navigate to the 'Add Course' button and then proceed to insert the relevant information needed to create a course. This information includes the subject, 4-digit number, title, section number, instructor, description, and lecture time. Once the information is entered, they click the 'Add' button and the system would then update with that course now being active.

M. Removing an Employee

Purpose: Allows administrative users to remove an employee from the system because of termination or for any other reason.

Inputs: The user would select the employee to be removed from the system.

Expected Outputs: The system would display a message that the employee selected is removed from the system.

Pass/Fail Criteria:

Pass: The employee is successfully removed from the system.

Fail: The employee is not removed from the system.

Test Procedure: The administrator would navigate to the 'Remove Employee' button and then find the employee that is to be removed from the database. Once the employee is selected, the user selects 'Remove' and then the system would update the database.

N. Removing a Course

Purpose: Allows administrators to remove courses from the database.

Inputs: The selection of the active course.

Expected Outputs: The system would display a message indicating that the course(s) selected are removed.

Pass/Fail Criteria:

Pass: The course is successfully removed from the database.

Fail: The course is not removed from the database.

Test Procedure: The administrator would navigate to the 'Remove Course' button and then find the course(s) that they would like to remove from the database. Once the course(s) is selected, the administrator will click 'Remove' and then the system will update the database.

O. Class Capacity Error Handling

Purpose: Prevent students from enrolling in classes that have reached their maximum capacity.

Inputs: Write code that will prevent students from enrolling in full classes.

Expected Outputs: Students should not be able to click on classes that are already full.

Pass/Fail Criteria:

Pass: The full classes are unable to be clicked on.

Fail: The full classes are able to be clicked on and the student can enroll.

Test Procedure: Try to click on a full course during enrollment. If you are not able to add the course to the enroll list then we are successful.

P. Testing for Incorrect Input

Purpose: Prevent errors when querying from the database.

Inputs: In a text field, the input required being only characters not integers or symbols

Expected Outputs: System should display an error message prompting the user to redo the input

Pass/Fail Criteria:

Pass: User is aware of the incorrect input

Fail: An error is thrown and the system crashes

Test Procedure: Go to a text field, let's say search courses. In the course name text field enter a series of characters with an integer or symbol and click search.

XIX. Cost and Duration Estimate

To estimate the overall cost and duration of the project, we used intermediate COCOMO, a series of methods that will help us estimate these values. The intermediate Constructive Cost Model, otherwise known as COCOMO, is a method developed by Barry Boehm [5]. The first step is to compute development time, which is split into two calculations. Firstly, we estimate two values, the length of the product in KDSI, essentially, how many thousands of lines of source statements will be needed, as well as the product's development mode, a measure of the intrinsic level of difficulty. In our case, we will be using the constants of 3.2 and 1.05 that Boehm believes best-fit projects of organic model, which are smaller and straightforward. Other development modes include semi-detached and embedded mode, which are far more complex and are bigger projects. We considered the development mode to be organic because of our team size, the task at hand, and our experience. The team size is fairly small, with only 4 members. In addition, the task has been clearly defined and fully understood, which was provided in the problem presentation. Lastly, although we are students, we have low to nominal experience programming databases in previous courses, which can be applied to this project. Using those standard values from the organic development we can calculate the nominal effort after estimating the KDSI using the calculation:

$$Nominal\ Effort = 3.2 * (KDSI)^{1.05} person-months$$

To calculate KDSI, we first estimate the number of lines of code in thousands, otherwise known as KLOC (Kilo-Lines Of Code) and determine a suitable relationship from converting KLOC to KDSI, as there are possibly some differences. In an article published in 2010 by the *Global Journal of Pure and Applied Life Sciences* [8], the authors have used KLOC and KDSI interchangeably, which we will replicate. In addition, the textbook, *Object-Oriented Software Engineering* by Lethbridge [6] does not mention KDSI at all, instead, they use KLOC for all COCOMO calculations performed. In our cost and duration estimation, we will be using the conversion of 1 KLOC = 1 KDSI, based on the conversion that the authors of the research article used.

Initially, we estimated our lines of code to be 2.277 KLOC in our software project management plan document. Some of the values for the lines of code were estimated based on a smaller database we did for a flight registration system in a previous course; however, now that

the program has been completed, we can recalculate the KLOC and thus, re-examine the cost and duration calculations. In our final design, we have 656 lines of code designated for the database initialization, which is 0.656 KLOC. This 0.656 KLOC includes creating the database, creating tables, and creating queries for inserting information into the database to properly test the functionality. Within our GUI, we have 15 fxml files, which are used in order to create the layout of the screens. The total number of lines of code for the GUI is 937, for an average of 62 lines of code per fxml file, which totals 0.937 KLOC. Lastly, our Java implementation that was responsible for all the functionality of our URS had 16 classes. In those 16 classes, there are a total of 3465 lines of code, which can be restated as 3.465 KLOC, for an average of 216 lines of code per class. The total number of lines of code is then calculated as

$$KDSI = KLOC = 0.656 + 0.937 + 3.465$$

$$KDSI = 5.058$$

This means that our final calculation for the number of lines of code is over twice the estimation provided in our management plan. From this, we can then calculate the base nominal effort to develop this program, which will be calculated using:

$$Nominal\ Effort = 3.2 * 5.058^{1.05} person-months = 17.55 person-months$$

Afterwards, we multiply the nominal effort by fifteen software development multipliers, which were values provided by Boehm and each has a different value depending on the difficulty we believe the process will take, which is given by this table:

Cost Drivers	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Database size		0.94	1.00	1.08	1.16	
Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Execution time constraint			1.00	1.11	1.30	1.66
Main storage constraint			1.00	1.06	1.21	1.56
Virtual machine volatility*		0.87	1.00	1.15	1.30	
Computer turnaround time		0.87	1.00	1.07	1.15	
Personnel Attributes						
Analyst capabilities	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Programmer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience*	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project Attributes						
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

*For a given software product, the underlying virtual machine is the complex of hardware and software (operating system, database management system) it calls on to accomplish its task.

Figure 36: Intermediate COCOMO Effort Multiplier

Cost Drivers	Reason	Rating	Effort Multiplier
Required Software Reliability	Can result in scheduling conflicts and system crashes from overload. It is the main interface for gathering information about your present, past and future courses. As well, it's the main interface for employees to submit grades and adjust the courses offered by the university	High	1.15
Database Size	The end goal of this database is to be a university registration system for thousands of students and instructors over many years. That is a large amount of information to be stored.	High	1.08
Product Complexity	The complexity of this system is nominal, the design of the database and GUI is simple; however, the client-server architecture is slightly challenging, so these balance out.	Nominal	1.00
Execution Time Constraint	Very minimal in today's technology compared to the technology in 1984 when Intermediate COCOMO was created. Thus, we have chosen the smallest effort multiplier for all 4.	Nominal	1.00
Main Storage Constraint		Nominal	1.00
Virtual Machine Volatility		Low	0.87
Computer Turnaround Time		Low	0.87
Analyst Capabilities	We have limited experience in this space aside from a few projects in courses. We have learned all the required information before, but have yet to create an entire system like this before.	Low	1.19
Applications Experience	Application experience is nominal as we have created some databases before as well as we know how a university registration system works. However, we are still students and still lack	Nominal	1.00

	knowledge on how to efficiently design our database.		
Programmer Capability	Since we have our current university career as programming experience, we are confident in our capabilities	Nominal	1.00
Virtual Machine Experience	With only about 1 year of experience in the required fields for this project, we feel as though we lack experience here as well.	Low	1.10
Programming Language Experience	We feel as though we have some experience in Java Programming (4 years) and Database Design (4 months) from courses taken in school.	Nominal	1.00
Use of Modern Programming Practices	Again, due to lack of experience since most of what we know has been from school learning instead of on-site practices.	Low	1.10
Use of Software Tools	A very basic level that we have learned in school courses.	Low	1.10
Required Development Schedule	The development schedule takes place during this term which gives us around 3 months to complete it.	Nominal	1.00

Multiplying all the effort multipliers together, we get

$$Multiplier = 1.49$$

Now that we know the effort multiplier, we can now calculate the new nominal effort

$$Nominal\ Effort = 1.49 * 17.55\ person - months$$

$$Nominal\ Effort = 26.15\ person - months$$

According to the University of Waterloo [9] and University of Victoria [10], software engineer co-op students in Canada make approximately \$3500 a month. If you assume that the average workweek is 40 hours/week, then the average hourly salary is approximately \$22/hour. Now that we have these values, we can calculate the total cost of the project, which ends up being

$$\begin{aligned} \text{Total Cost} &= 26.15 \text{ person} - \text{months} * \$3500 / \text{month} \\ \text{Total Cost} &= \$91,525 \end{aligned}$$

Therefore, based on the intermediate COCOMO method, we can calculate that the cost of the project will be \$106,890 and the duration of the project is calculated to be 26.15 person-months. As there are four members in the group, we can divide the total person-months by 4, which gives us 6.54 person-months for every group member. Continuing on, we can break down the duration even more, if we assume each month has 30 days, then the number of person-days is 196 per person for every group member.

These values are much higher than what was originally estimated. Originally, we estimated that the project would cost \$46,305 and the duration was 13.23 total person-months. Our estimations were developed based on previous smaller database projects as well as rough estimations of how many lines of code we thought classes would be based on the example given in class. In addition, there were many additions to the number of classes and the number of queries being performed in our database throughout the development. In terms of the duration, this is absurdly long, the reason for this could be that we incorporated comments in the calculation of our number of lines of code. This was included as we felt that comments were a critical part of our program, and detailed explanations of how everything functions in the program are a must. If comments were excluded from the estimation from the lines of code, then the cost and duration estimate would be lower, however, it would still be higher than the original estimations.

XX. Roles and Responsibilities

A. Ownership of Subsystem

As this is a group project, all group members will be supporting each other. However, we will be designating each member a subsystem that they will be responsible for. In our case, we divided responsibilities into four subsystems, which will be:

- Database Design: Nicholas
- GUI Design: Jimmy
- Java Implementation: Sukhraj
- Documentation: All of us

Although one member will be assigned the lead of their subsystem, the other group members will also be responsible for assisting in the completion of every subsystem.

B. Deliverables and Milestones

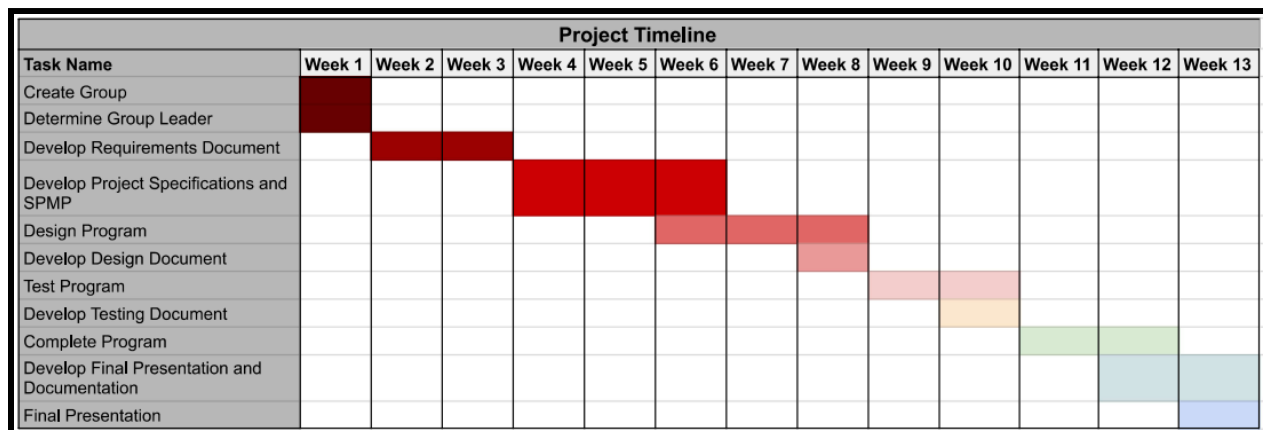


Figure 37: Gantt Chart of our Project Timeline

From the Gantt chart of our project timeline, we adhered to our planned timeline as much as possible and we were able to complete and develop a successful system by the project deadline and completed all deliverables on time.

For this project, we had multiple documents that had to be delivered on time. For those due dates, see below:

- Project Requirements Document: Sept 22nd, 2021

- Project Specifications and Software Project Management Plan Document: October 19th, 2021
- Project Design Document: Nov 2nd, 2021
- Project Testing Document: Nov 17th, 2021
- Final Demonstration, Presentation, Documentation, and Code Submission: Nov 27th, 2021

C. Project Leader

As our group has 4 members, we divided the role of group leaders amongst each other. The project leader is responsible for delegating tasks, motivating team members, and making people's roles clear, in addition to recording the meeting minutes/log files. However, this does not mean that other group members were not allowed to have input, as all group members were equally responsible for the completion of this project. Listed below are the group members as well as the schedule for when they were considered the group leader.

- Nicholas Imperius: Sept 8th - Sept 29th
- Sukhraj Deol: Sept 30th - Oct 22nd
- Jimmy Tsang: Oct 23rd - Nov 11th
- From Nov 12th - Nov 27th we collectively took small rotations being a leader in certain meetings since Kristopher was not attending any meetings and we needed someone to lead.

XXI. Conclusion

In summary, we developed a fully working university registration system that is fully operational. The functions of the system vary for each type of member, whether they are a student, instructor, or administrator, but they all have the ability to login into the URS. Students have the ability to search for courses, register for courses, view enrolled courses, drop courses, and view course grades. Instructors can view their active courses, modify grades in those courses, and view the employee and student list. Administrators can view active courses, view employee and student lists, add and remove employees, and add and remove courses. Extensive documentation was prepared, detailing all aspects of the project, including a design overview, testing overview, and management plans. Overall, when testing the functionality of our final program, there were no errors found and the system successfully passed our test cases. In conclusion, we found this project to be a fun and challenging way to learn about database design, GUI design, and Client-Server architecture. We learned valuable design lessons and important time management skills that are needed in software development as well as gained a greater understanding of what the Software Development Life Cycle process requires.

Appendix A

A. Test Log 1

Actual Results: Logging in with the correct credentials brought the user to the correct screen.

For example, a student logging in brought the student to the student welcome screen.

Incident Reports: When logging in with incorrect information, no prompt signifying that it was an incorrect username/password.

Outcome: Failure

B. Test Log 2

Actual Results: Searching for courses based on specific keywords, consisting of a course code, name, or section would display all available courses matching the search criteria. If there are no courses found that match the criteria, then it would display that there are no matching courses.

Incident Reports: No incidents were found with this test case.

Outcome: Success

C. Test Log 3

Actual Results: A student was able to register in courses successfully.

Incident Reports: If the student was already registered for a section, it would allow them to choose the section again which would result in an error.

Outcome: Failure

D. Test Log 4

Actual Results: All courses that the student is enrolled in were successfully displayed.

Incident Reports: No incidents were found with this test case.

Outcome: Success

E. Test Log 5

Actual Results: A student user was successfully able to select from their enrolled courses to drop a course, or multiple courses at once if they choose to do so.

Incident Reports: No incidents were found with this test case.

Outcome: Success

F. Test Log 6

Actual Results: All course grades were successfully displayed properly.

Incident Reports: No incidents were found with this test case.

Outcome: Success

G. Test Log 7

Actual Results: All available courses were successfully displayed properly.

Incident Reports: No incidents were found with this test case.

Outcome: Success

H. Test Log 8

Actual Results: The grade was successfully modified in the database.

Incident Reports: When trying grades that were less than 0 or greater than 100, the system allowed that modification when it should not have.

Outcome: Failure

I. Test Log 9

Actual Results: All university employees have successfully displayed properly

Incident Reports: No incidents were found with this test case.

Outcome: Success

J. Test Log 10

Actual Results: All university students were successfully displayed in proper fashion

Incident Reports: No incidents were found with this test case.

Outcome: Success

K. Test Log 11

Actual Results: The member was successfully added to the database

Incident Reports: When input fields were left blank or with invalid characters, the system still accepted the input when it should not have.

Outcome: Failure

L. Test Log 12

Actual Results: The course was successfully added to the database.

Incident Reports: When input fields were left blank or with invalid characters, the system still accepted the input when it should not have.

Outcome: Failure

M. Test Log 13

Actual Results: The employee was successfully removed from the database.

Incident Reports: No incidents were found with this test case.

Outcome: Success

N. Test Log 14

Actual Results: The course was successfully removed from the database.

Incident Reports: When a course was removed, students were still enrolled in the course which does not exist in the system anymore.

Outcome: Failure

O. Test Log 15

Actual Results: The student cannot enroll in classes that have reached max capacity.

Incident Reports: No incidents were found with this test case.

Outcome: Success

P. Test Log 16

Actual Results: The system displayed that there were no classes found for those search results

Incident Reports: No incidents were found with this test case.

Outcome: Success

XXII. References

- [1] Peng, Yu, et al. "Design and Implementation of the Online Course Registration System at Tsinghua University." *2012 International Conference on Systems and Informatics (ICSAI2012)*, 2012, <https://doi.org/10.1109/icsai.2012.6223244>.
- [2] Liu, Ying, et al. "Design and Implementation of Student Registration System for Universities." *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, 2012, <https://doi.org/10.1109/cecnet.2012.6202263>.
- [3] Adamov, Abzetskhan, et al. "Good Practice of Data Modeling and Database Design for UMIS. Course Registration System Implementation." *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*, 2014, <https://doi.org/10.1109/icaict.2014.7035949>.
- [4] *SDLC - Waterfall Model*. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. [Accessed: 23-Oct-2021].
- [5] Schach, Stephen R. *Object-Oriented and Classical Software Engineering*. McGraw-Hill, 2011.
- [6] Lethbridge, Timothy Christian, and Laganière R. *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. The McGraw-Hill Companies, 2005.
- [7] Jegorov, Aleksei. "Estimate Time Complexity of Java and SQL Query." *Medium*, Dev Genius, 16 July 2021, <https://blog.devgenius.io/estimate-time-complexity-of-java-and-sql-query-afa13a88a981>.
- [8] Duke, S. Okor, Obidinnu, and Nwafili, "AN IMPROVED COCOMO SOFTWARE COST ESTIMATION MODEL," *Global Journal of Pure and Applied Life Science*, vol. 16, no. 4, p. 479, Apr. 2010.
- [9] "Co-op earnings," *Co-operative Education*, 22-Apr-2021. [Online]. Available: <https://uwaterloo.ca/co-operative-education/about-co-op/co-op-earnings>. [Accessed:23-Oct-2021].
- [10] "Co-op salaries - University of Victoria," *UVic.ca*. [Online]. Available: <https://www.uvic.ca/coopandcareer/co-op/about-coop/salaries/index.php>. [Accessed: 23-Oct-2021].