

# **University Registration Management System**

**CS2043 SJ01A – Dr. Kim**

**December 14<sup>th</sup>, 2020**

## **Authors**

Emilie Ouellette - 3664958

Nathan Young - 3627443

Joshua Hickman - 3451937

Ronan Boyd – 3673610

Syed Saeed - 3173298

# INDEX

<b>3.....</b>	<b>Team Profile</b>
<b>4.....</b>	<b>Project Overview</b>
<b>5-6.....</b>	<b>Risk Management</b>
<b>7.....</b>	<b>Project Planning</b>
<b>8-11.....</b>	<b>Design – Architecture Draft</b>
(10-11).....	User Interface Draft
<b>12-14.....</b>	<b>Planning and Execution</b>
(12).....	Checklist (Verification and Validation)
<b>15-16.....</b>	<b>Front-End Examples</b>
<b>17.....</b>	<b>Lessons Learned</b>
<b>18.....</b>	<b>Conclusion</b>

# TEAM PROFILE

## RONAN BOYD

Qualifications: Java

Strength: Programming

Role: Backend Developer

## SYED SAEED

Qualifications: System Design

Strength: System Analyst

Role: Developer Team Member

## JOSH HICKMAN

Qualifications: Java, MATLAB

Strength: Programming

Role: Front-End Developer

## NATHAN YOUNG

Qualifications: Java

Strength: Programming, Attention to detail

Role: Front-End and Back-End Developer

## EMILIE OUELLETTE

Qualifications: Java

Strength: Planning, Documentation

Role: Project Manager

**GITHUB REPOSITORY:** <https://github.com/emilie-cs2043/CS2043-Project>

**GOOGLE DRIVE:** <https://drive.google.com/drive/folders/1zUtC4iskWr5G6xXtwVPKA7u-Zo6moBVe?usp=sharing>

## PROJECT OVERVIEW

The goal of this project is to create a university registration system which can be used by both professors and students. The system revolves around course registration.

Courses and new students will be added to the system by the program administrators. This allows for minimal discrepancies and a complete listing to be available.

Students will be able to see the course offerings available to them in the current and upcoming semesters to allow them to complete their registration. Five courses can be selected, along with waiting lists if some of their initial choices become filled (10 students maximum). Other courses may be cancelled if registration numbers fall below the minimum amount allowed (less than 3 students). A grace period is set each semester to allow students to access the system and change their schedules by adding or dropping courses after the semester has started.

Once a student completes their registration, their tuition gets updated within their profile. Students can request to see the billing total for their semester. This can change depending on if students decide to change their schedule. Changes made during the above-mentioned grace period will be billed accordingly, including full refunds.

Professors are also able to access the system in order to see which classes they will be teaching, along with the current class list of these courses. The administrator of the program will be tasked to assign the professors to the appropriate courses at a maximum of 2 classes per professor.

This program facilitates preparations for the semester and makes sure students and professors alike feel ready to jump into the next session.

# RISK MANAGEMENT

Due to the size of the project, the nature of group work, and time limits, a risk management chart had to be created in order to identify potential risks (*Figure 1*). By doing this, it allowed us to prepare alternative trajectories for the project in order to mitigate the outcomes of such risks and remain on schedule.

Event Likelihood ↓	C3 Personal emergencies (Illness, family emergency...)	C3 Outside commitments (Family events, appointments...)	C3 Clash in team member schedules	C3 Miscalculating how long it will take to complete a certain task	C2 Program functionalities not working properly
Low	✖				
Medium		✖	✖		✖
High				✖	

Figure 1 - Risk Assessment Chart

\*C2 = Risks concerning the functionality of the program

\*C3 = Risks concerning scheduling and team members.

## PLANS AND CHALLENGES - MITIGATING RISKS

### 1) PERSONAL EMERGENCIES

- **Plan:** In the case that a team member has an emergency outside the project and is unable to participate for an indefinite amount of time, the group will cooperate to distribute that team members previously assigned task. If new tasks are to be assigned at a later date and the team member is still unavailable, the group will proceed to split extra tasks amongst themselves.
- **Challenge:** It may be impossible to predict when the team member will be available to join the project again.

### 2) OUTSIDE COMMITMENTS

- **Plan:** In the case of outside commitments, the team will continue to work on the project and redistribute that team members task if there is an approaching deadline.

- **Challenge:** Events are planned and most times not subject to rescheduling which mean they may sometimes clash with deadlines.

### 3) CLASH IN TEAM MEMBER SCHEDULES

- **Plan:** In the case of scheduling clashes, the team members can discuss and find a time that is available for everyone. This may include smaller slots in greater numbers if needed. Sometimes, some team members may not be able to attend certain team sessions and thus must be updated on new developments.
- **Challenge:** Making sure to keep a clear and consistent record of new developments.

### 4) MISCALCULATING HOW LONG IT WILL TAKE TO COMPLETE A CERTAIN TASK

- **Plan:** If certain aspects of the project take more time than initially planned, a rework of the schedule must be done. Tasks may have to be pushed back or assigned to certain team members while others finish the previous task.
- **Challenge:** Since it is many of the members first project of this nature, it is hard to create a realistic schedule from the start. Team members may need to dedicate more time to the project last minute, leading to clashes with schedules outside of class.

### 5) PROGRAM FUNCTIONALITIES NOT WORKING PROPERLY

- **Plan:** To effectively utilize the time allotted to the project, the team must prioritize which functionalities are the most important to implement first. This way, if some functionality does not work as intended, at least the extra time to fix it is being spent on important features needed for completion. This way, crucial functionalities are more likely to be present for deadlines.
- **Challenge:** Gaps in knowledge can push back progress if time must be put in to learn it. This can delay future tasks. Or a team member possessing that knowledge may be unavailable during a short or indefinite amount of time.

By sticking to these plans and staying realistic with what challenges we face, we can quickly decide on new schedules and tasks distributions that do not further delay the project.

## PROJECT PLANNING

Before starting on the project, it was important to know exactly what work we had ahead of us. By visualizing a Work Breakdown Structure (*Figure 2*), we were able to have a better idea of how to proceed.

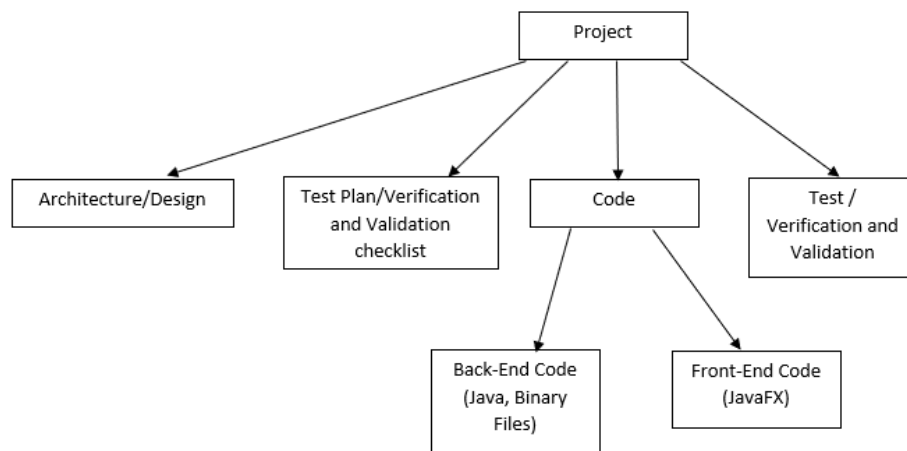


Figure 2 - Work Breakdown Structure

Continuing on from the structure, a PERT chart was created (*Figure 3*) to better estimate the proportion of work that would be needed for each stage. The difficulty in measuring this lies in the fact that this type of project was new to us, thus hard to estimate the time needed. The fact that we would be meeting on a few days notice for undisclosed amounts of time also made it hard to gauge how many meetings could be considered as a “day” in the chart.

Although there was uncertainty in scheduling in the beginning, this made us learn the importance of agreeing on set times and planning in advance what we would like to accomplish during the meeting.

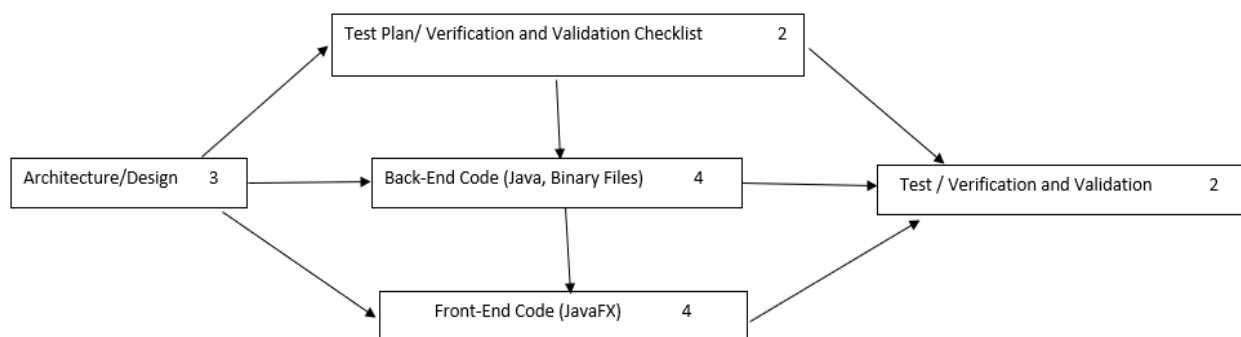


Figure 3 – PERT Chart

## DESIGN – ARCHITECTURE DRAFT

Next, we needed to decide on what the program must be able to do and further clarify the ideas expressed in the project overview. We identified the main actors and assigned the main things these actors should be able to accomplish.

### MAIN GOALS

#### ❖ Student

- Can register to courses
- Can see their tuition balance
- Can drop classes
- Can see their list of classes

#### ❖ Professor

- Can see their class list
- Can see the list of classes they are teaching

#### ❖ Administrator

- Can add students to the system
- Can add courses to the system

Based on this list, we were able to have a straightforward idea of the priority features of our system. This ensured that all team members were on the same page. From here, we could start the design process and also build on these ideas and add features of secondary importance as we go along, such as a login credentials, functionality to differentiate the current semester based on system date/time, prerequisite listings and waitlists.

After a few brainstorming sessions, we were able to come up with a baseline of features we wanted to implement and were ready to create an architecture draft of our system. This included an initial draft of a Use Case diagram (*Figure 4*) and a Class diagram (*Figure 5*).

*\*Later on in the report, a reworking of these diagrams will be presented to better reflect the true nature of our system.*



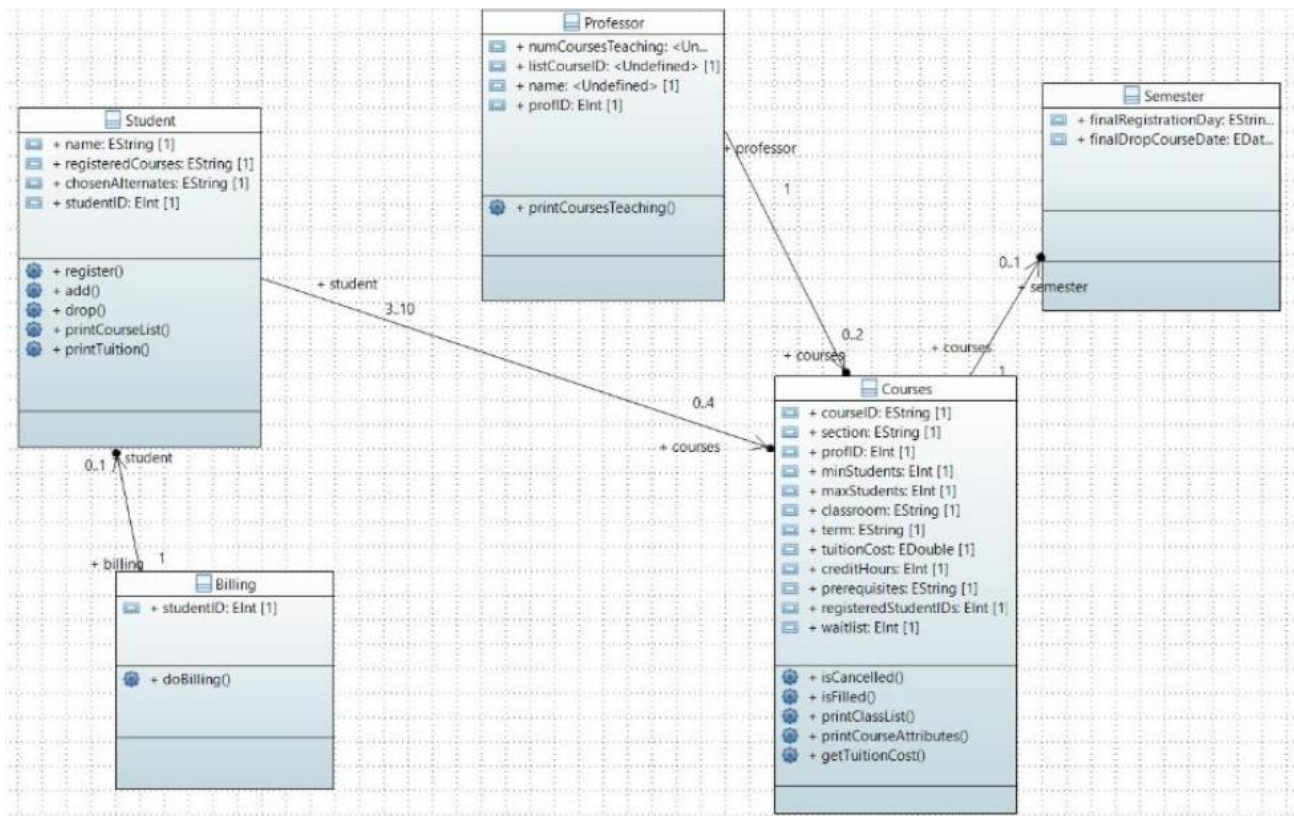


Figure 4 – Class Diagram Draft

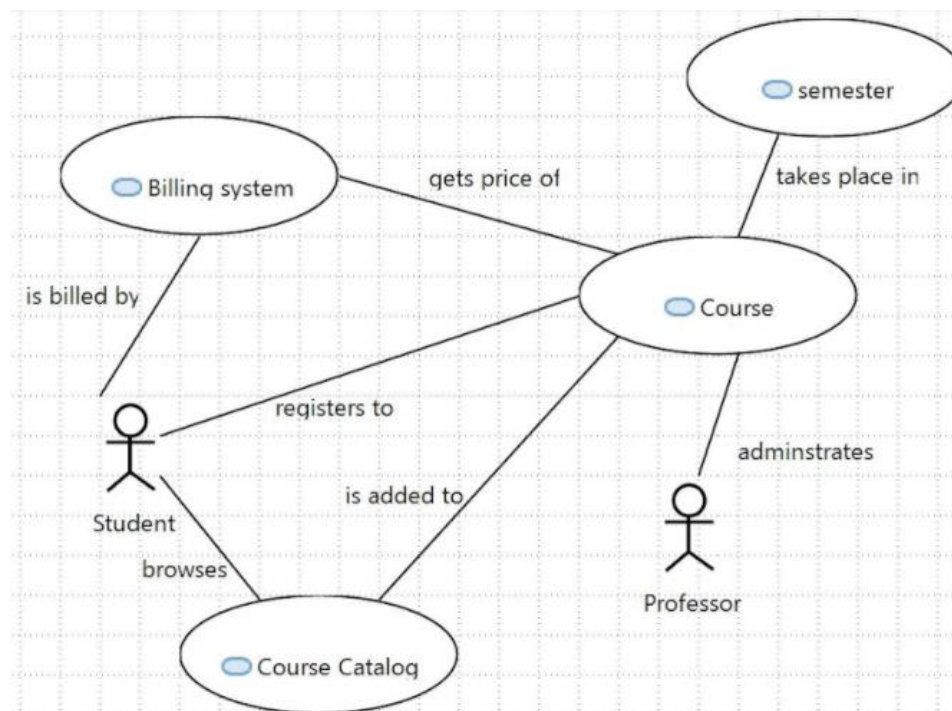


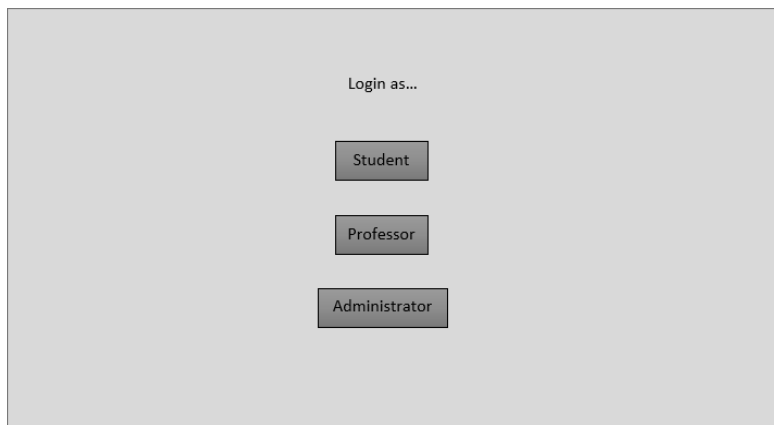
Figure 5 – Use Case Diagram Draft

## USER INTERFACE DRAFT

To give us a greater sense of direction, we created a User Interface Draft which gave us a better idea of what exactly we were building before starting the coding process. By visualizing what the user may see, it made it easier for us to notice how different classes interact with each other and enabled us to discover new things we had not thought about such as creating a database of sorts to store students, courses and professors, as well as any course or student listings associated with these entities.

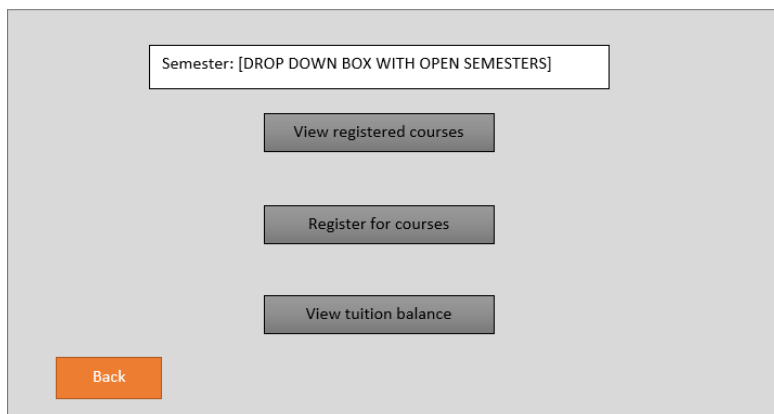
The draft includes some of the major interface pages that a user would run into. Although the final interface was different than the drafts below, the ideas within them served as a template to follow, some of these ideas will be listed beside their respective image.

### Opening Screen



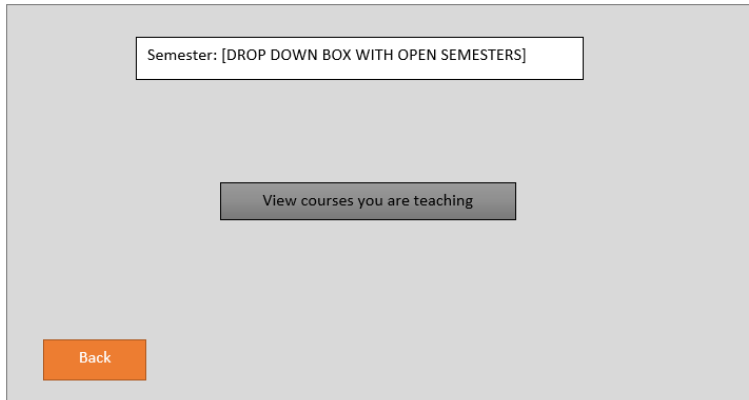
From here, we were able to decide that a login page with login credentials would be a good idea to implement.

### Student Page



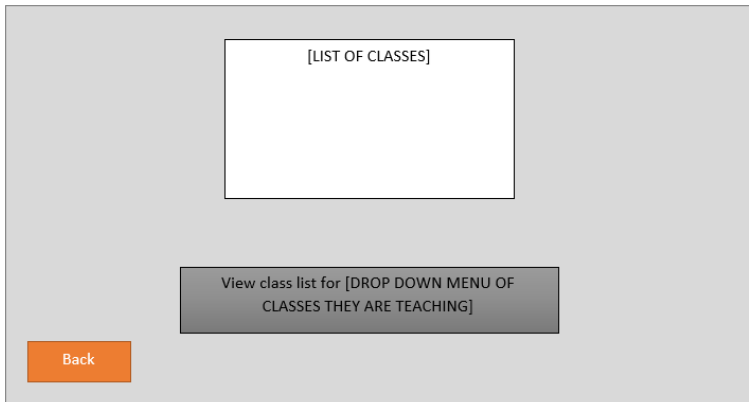
This would lead to discussions on how we would handle the idea of a semester

## Professor Page



A wireframe for a Professor Page. It features a light gray background. At the top, there is a white rectangular box containing the text "Semester: [DROP DOWN BOX WITH OPEN SEMESTERS]". Below this, centered, is a dark gray button with the text "View courses you are teaching". In the bottom left corner, there is an orange button with the text "Back".

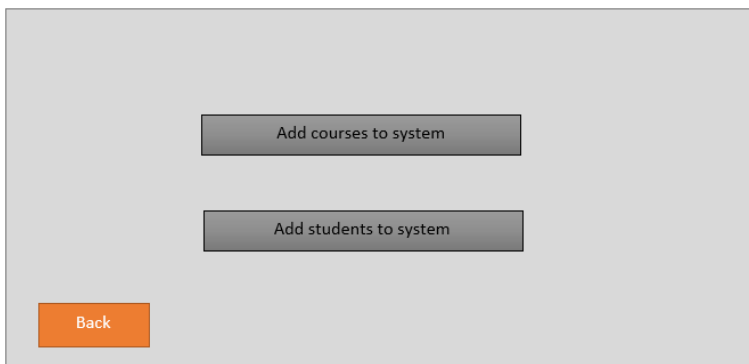
## Professor Course Listing/Class Listing Page



A wireframe for a Professor Course Listing/Class Listing Page. It features a light gray background. In the center, there is a white rectangular box containing the text "[LIST OF CLASSES]". Below this, centered, is a dark gray button with the text "View class list for [DROP DOWN MENU OF CLASSES THEY ARE TEACHING]". In the bottom left corner, there is an orange button with the text "Back".

This led us to think a bit ahead in the front-end side of things on how we could display the class list to the professor.

## Administrators Page



A wireframe for an Administrators Page. It features a light gray background. In the center, there are two dark gray buttons stacked vertically. The top button contains the text "Add courses to system" and the bottom button contains the text "Add students to system". In the bottom left corner, there is an orange button with the text "Back".

This helped us realize the scope of the administrator class and what it's main functions should be.

# PLANNING AND EXECUTION

Since we decided on doing a lot of the testing while in the process of coding, we developed a checklist to keep us on track and to make sure we do not lose sight of our goals.

## CHECKLIST

### VERIFICATION – ARE WE BUILDING THE SYSTEM RIGHT?

- Data such as students, professors, courses, class lists, and student lists must be saved to binary files for subsequent program usage.
- Binary files should be updated after changes have been confirmed by either a student, administrator or student.
- Binary files must be initially populated before official execution of the program.
- Students must be added to waiting lists (queues) if courses are full.
- A login page must manage access for administrators, students and professors and bring them to their respective home pages.
- Back-end functionality will be designed with Java and must be tested before implementing too many front-end features.
- Front-end features will be designed with JavaFX.

### VALIDATION – ARE WE BUILDING THE RIGHT SYSTEM?

- Students cannot register to more than five courses.
- Professors cannot teach more than 2 courses.
- Courses must have between 3-10 students.
  - Over 10 registrations: Additional students are added to a waiting list
  - Under 3 registrations: The course is automatically cancelled after a date specified for the semester.
- Tuition must be adjusted each time a student changes their course selections before the stated semester deadline.
- Professors must be able to access class lists for each of their courses at any given time.
- Students must be able to access a list of their registered courses for the semester.
- If a student drops a course and there is a waiting list for this offering, the next student in the waiting list queue gets registered to the course.
- Course catalog must be available during registration periods for a given semester

For the execution of our system, we first focused on building 3 main classes:

- 1) Student
- 2) Course
- 3) Professor

Along with this, we had a test driver class so we could test new functions before moving on with new features. Along the way, we added a Semester class to help keep our course offerings in order.

Once we were confident with our design, we moved on to finding a way to store our data in binary files so that we did not lose data such as new students registering to classes or new course offerings being added by the administrator.

After testing our features and making sure we were satisfied that our code followed the checklist requirements stated above, we got started on developing the front-end with JavaFX. A few team members that felt more confident in this aspect took the lead and got it started.

Once we could see that our system had all the important features working, we created a new Class Diagram (Figure 6) and Use Case Diagram (Figure 7) that better reflected our system.

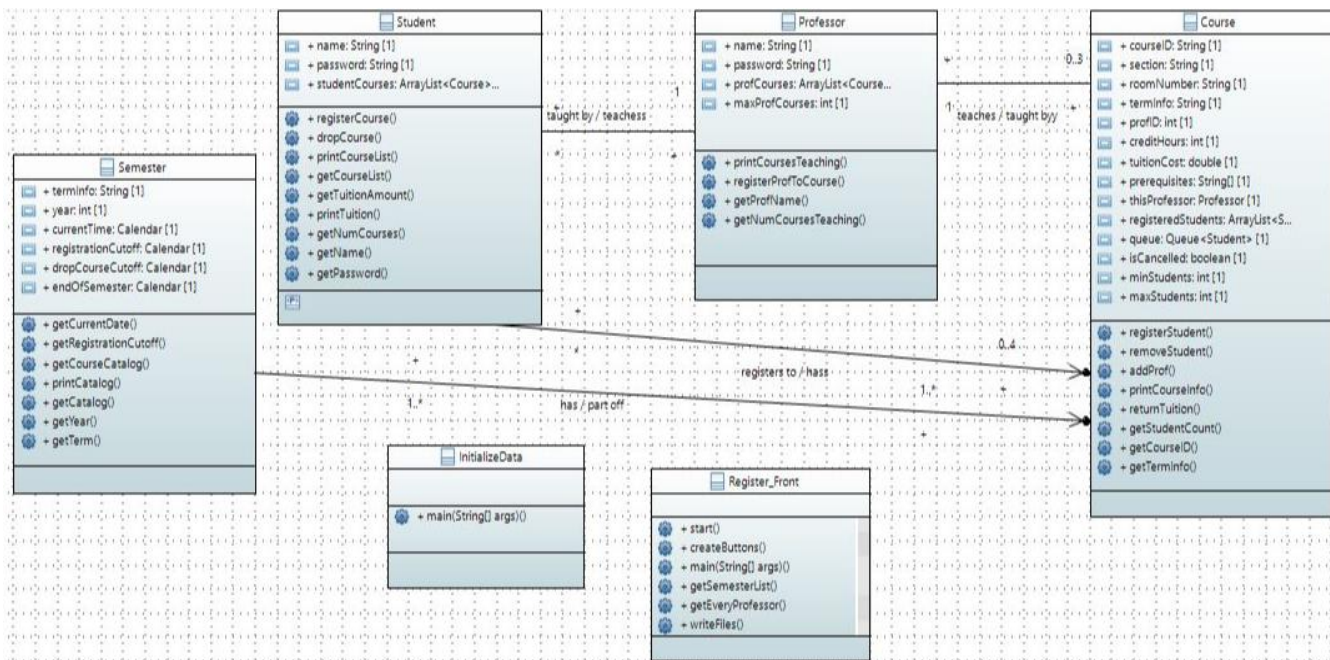


Figure 6 – Updated Class Diagram

\*\*InitializeData and Register\_Front classes were added to the diagram so that we could visualize the entirety of the program. The InitializeData class is where data is saved to binary files, and the Register\_Front is where our JavaFX code was implemented and where new data could be written to files.

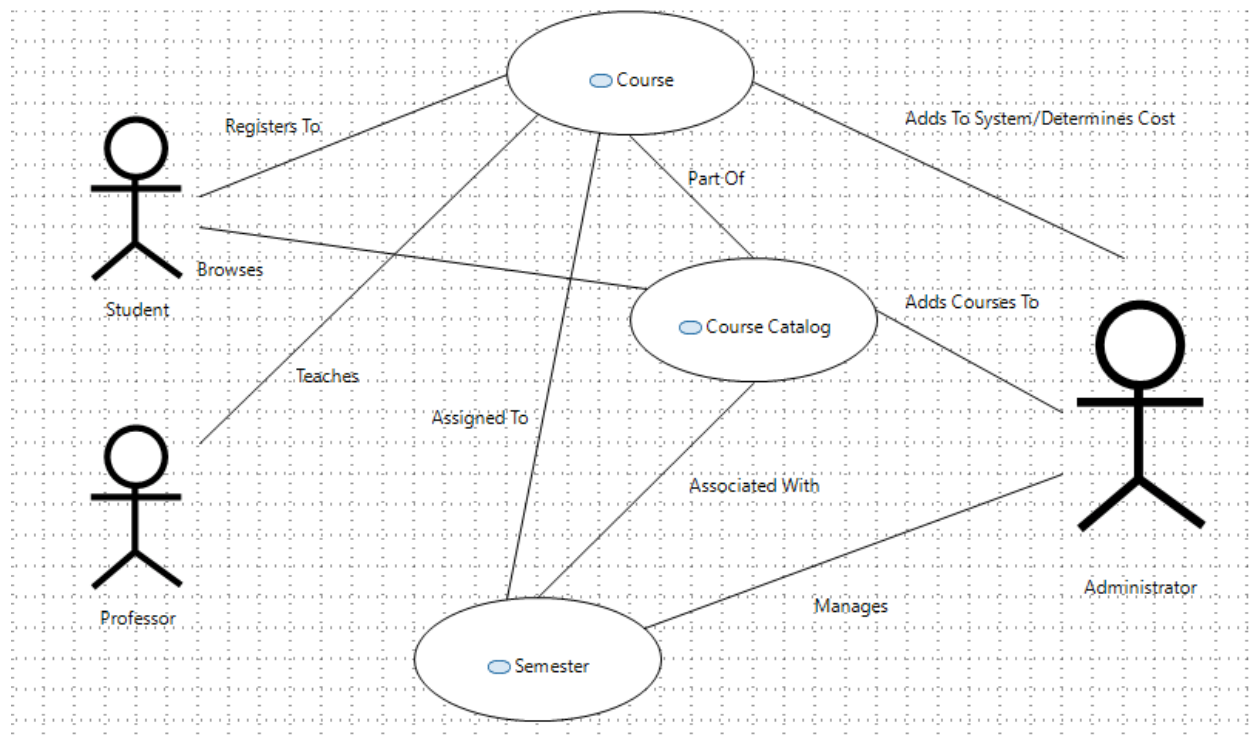


Figure 7 – Updated Use Case Diagram

In the updated class diagram, the Billing System class is no longer reflected as a part of our system since reworking our design, although many of the original variables and methods that were in the draft are still included. In the Use Case Diagram, some changes have been made as well, such as adding the administrator actor and getting rid of the Billing System case.

## FRONT END EXAMPLES

Below are some of the front-end user interface windows from the program.

### Login Page

The screenshot shows a window titled "Registration Management System" with a title bar containing standard Windows window controls. The main content area has a light gray background. At the top, there are three buttons: "Login", "Register", and "Back". To the right of these buttons, there are two input fields. The first is labeled "Your Student ID number:" and the second is labeled "Your password:". Both fields are empty.

### Student Landing Page

The screenshot shows a window titled "Registration Management System" with a title bar containing standard Windows window controls. The main content area has a light gray background. At the top, there are four buttons: "Register for Courses", "View registered courses", "Display tuition balance", and "Sign out".

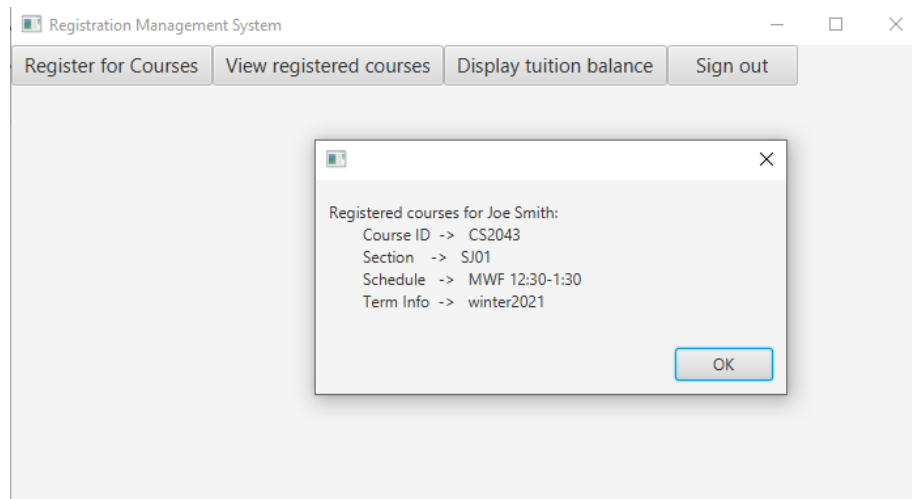
### Course Catalog for Student Registration

The screenshot shows a window titled "Registration Management System" with a title bar containing standard Windows window controls. The main content area has a light gray background. It displays three course entries, each with a radio button to its left. The first entry is selected, indicated by a blue circle. The entries are:

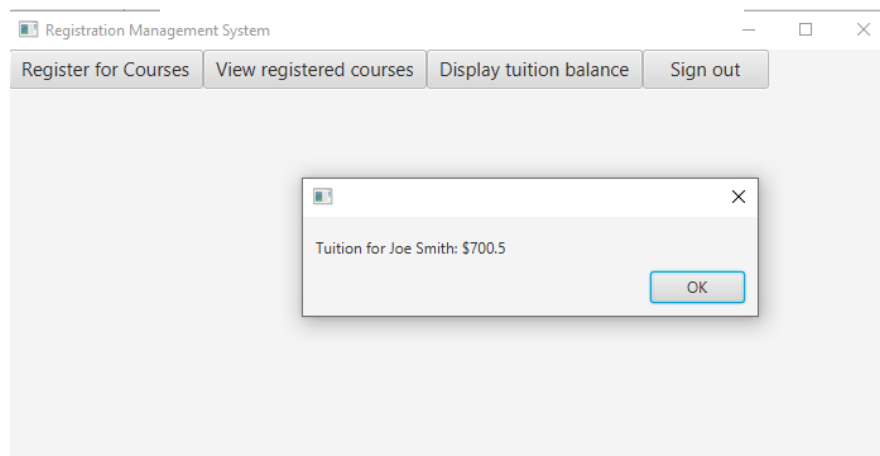
Course ID	Section	Schedule	Term Info
CS2043	SJ01	MWF 12:30-1:30	winter2021
AB2000	SJ02A	MTF 4:30-8:20	winter2021
CC1111	SJ111A	MTT 3:30-9:30	winter2021

Below the course entries, there are two buttons: "Register for selected" and "Back".

### Clicking "View registered courses"



### Clicking "Display tuition balance"



Although we were unable to implement front end functionality for all aspects of the project, we were able to have important checkpoints implemented.



## LESSONS LEARNED

As previously mentioned, this was the first time some of the team members had worked on a project of this nature. Because of this, a lot of things were learned along the way. Notably:

- The importance of time management and allowing extra time at the end in case we ran into any of the aforementioned risks.
- Managing the project in a way that would allow us to proceed linearly but also with the flexibility to alter previous decisions if needed.
- Writing meaningful comments in our code since it can grow very fast.
- Keeping records of documentation in order for all team members to stay on the same page and to not lose sight of the goals of the project.
- The importance of utilizing design strategies (UML) and keeping clear records of code progression, accessible to all team members. Either through GitHub or a Google Drive.
- Pair Programming allows the team to brainstorm solutions, stay consistent, and speeds up the coding process.
- Utilizing the different strengths that team members possess to achieve milestones in a more efficient and knowledgeable way.

## CONCLUSION

In conclusion, the project was a great way for us to experience firsthand the many inner workings of building a project from scratch. This knowledge will serve us in the future, both academically and personally. The emphasis on planning both the design and test cases made us appreciate the fact that much more goes into a program than just coding. As we grow as students, and later on in the work environment, these skills will be refined with practice and projects will become much easier to manage, especially in relation to time.