



**Module Code & Module Title**  
**CS5002NP Software Engineering**

**Assessment Weightage & Type**  
**Individual Coursework**

**Year and Semester**  
**2<sup>nd</sup> Year – 2<sup>nd</sup> Semester**

**Student Name: Pranish Gurung**  
**Group: C5**

**London Met ID: 22069046**

**College ID: NP04CP4A220092**

**Assignment Due Date: 7<sup>th</sup> May, 2024**

**Assignment Submission Date: 7<sup>th</sup> May, 2024**

*I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.*

## Table of Contents

1.Introduction .....	1
2. Work Breakdown Structure (WBS) .....	4
3. Gantt Chart.....	5
4. Use Case Diagram .....	6
5. High Level Use Case Description.....	8
6. Expanded Use Case Description .....	11
7. Communication/Collaboration Diagram.....	13
7.1 Steps included in Communication/collaboration diagram .....	14
7.2 Final Collaboration diagram .....	20
8. Sequence Diagram.....	21
8.1 Steps involved in Sequence Diagram .....	21
8.2 Final sequence diagram .....	26
9. Class Diagram.....	27
9.1 Steps involved in creating class diagram .....	27
9.2 Final Class Diagram .....	33
10. Further Development.....	34
11. Prototype .....	37
12. Conclusion .....	63
13.References .....	65

## List of Figures

Figure 1 Waterfall methodology .....	2
Figure 2 Work Breakdown Structure .....	4
Figure 3 Gantt chart .....	5
Figure 4 Use case diagram .....	7
Figure 5 Domain classes of collaboration diagram.....	14
Figure 6 Adding control object for collaboration diagram .....	15
Figure 7 Adding Boundary object .....	16
Figure 8 Adding actor for Collaboration diagram .....	17
Figure 9 Association.....	18
Figure 10 Adding messages.....	19
Figure 11 Final collaboration diagram .....	20
Figure 12 Domain classes.....	21
Figure 13 Adding control object lifeline.....	22
Figure 14 Adding boundary object lifeline .....	22
Figure 15 Adding actor Sequence diagram .....	23
Figure 16 Determining activation period.....	24
Figure 17 Adding messages.....	25
Figure 18 Final Sequence diagram .....	26
Figure 19 Drawing domain classes .....	28
Figure 20 Association in class diagram .....	29
Figure 21 Aggregation in Class diagram .....	30
Figure 22 Composition in class diagram .....	31
Figure 23 Inheritance in class diagram.....	32
Figure 24 Class diagram .....	33
Figure 25 Welcome page .....	37
Figure 26 Splash Screen.....	38
Figure 27 Home page.....	39
Figure 28 Sign-In page.....	40
Figure 29 Sign-up page.....	41
Figure 30 Login page .....	42
Figure 31 Product page.....	43
Figure 32 Plant description page.....	44
Figure 33 Payment method Product page.....	45
Figure 34 Order page .....	46
Figure 35 Product Scan page.....	47
Figure 36 After Scanning.....	48
Figure 37 My Cart.....	49
Figure 38 Course page.....	50
Figure 39 Course details page .....	51
Figure 40 Course Payment page .....	52
Figure 41 After Payment page .....	53
Figure 42 Completed Course page .....	54
Figure 43 Ongoing Course page .....	55

Figure 44 Certificate of Course completion page .....	56
Figure 45 Drawer page.....	57
Figure 46 Forum Page .....	58
Figure 47 Recommendation of Chat page.....	59
Figure 48 Recommendation message page.....	60
Figure 49 Mock Test page.....	61
Figure 50 Dashboard Admin .....	62

## Table of Tables

Table 1 High level use case of Register Customer .....	8
Table 2 High level use case of Join Course .....	8
Table 3 High level use case of Purchase plant .....	8
Table 4 High level use case of ask for recommendation .....	8
Table 5 High level use case of take exam.....	9
Table 6 High level use case of Post Forum.....	9
Table 7 High level use case of Make Payment .....	9
Table 8 High level use case of Generate report .....	9
Table 9 High level use case of View Report.....	10
Table 10 High level use case of Login.....	10
Table 11 High level use case of Send Notification .....	10
Table 12 Expanded use case of "Register Customer".....	11
Table 13 Expanded use case of Join Course.....	12

## 1.Introduction

Software engineering is the process of developing, testing and deploying computer applications to solve real-world problems by adhering to a set of engineering principles and best practices. The field of software engineering applies a disciplined and organized approach to software development with the stated goal of improving quality, time and budget efficiency, along with the assurance of structured testing and engineer certification. **(Yasar, 2023)** It is a systematic approach to the development of software that ensures that the software is reliable, efficient, and meets the needs of the users. It is a complex and challenging field, but it is also a rewarding one. Software engineering can work on a wide variety of projects, from small personal project to large enterprise applications. They also can work with a variety of technologies, from traditional programming languages to trending artificial intelligence. **(Nicholas, 2023)**

This coursework is a significant part of the module, contributing 35% towards the final grade. McGregor Institute of Botanical Training, based in Ireland's Godavari, Lalitpur, has been operating in Nepal for nearly 7 years. Offering a range of undergraduate and postgraduate courses in agriculture and horticulture, it's affiliated with Dublin City University. Responding to the growing interest in agriculture, McGregor plans to introduce short-term certification courses in horticulture. Additionally, they aim to sell various plant varieties, some at minimal fees or for free, and establish a community forum for plant enthusiasts to share ideas, organize programs, and seek expert advice. In terms of detailed specifications, the proposed system will facilitate user registration, course enrolment, plant purchase, payments, expert recommendations, report preparation, certification exams, and forum participation. The planning, requirements modelling and analysis phase will involve preparing a Work Breakdown Structure and Gantt Chart based on a chosen methodology. As Use Case Model will be developed, along with expanded description for selected use cases. Communication diagrams, including Collaboration and Sequence diagrams, will be created to illustrate system interactions. A domain class table and Analysis Class diagram will outline domain classes.

Moving forward, The McGregor Institute of Botanical Training is using the Waterfall methodology along with Unified Modelling Language (UML) to develop their computerized system. This method follows a step-by-step approach, where each stage has to be finished before moving on to the next. It begins with analysing requirements, then designing the system, implementing it, testing thoroughly, deploying it, and finally maintaining it. If they catch small errors early on, they can fix them within the same stage. But if there are serious issues, they might need to restart that stage from the beginning. Below is a diagram illustrating the different phases of the Waterfall model.

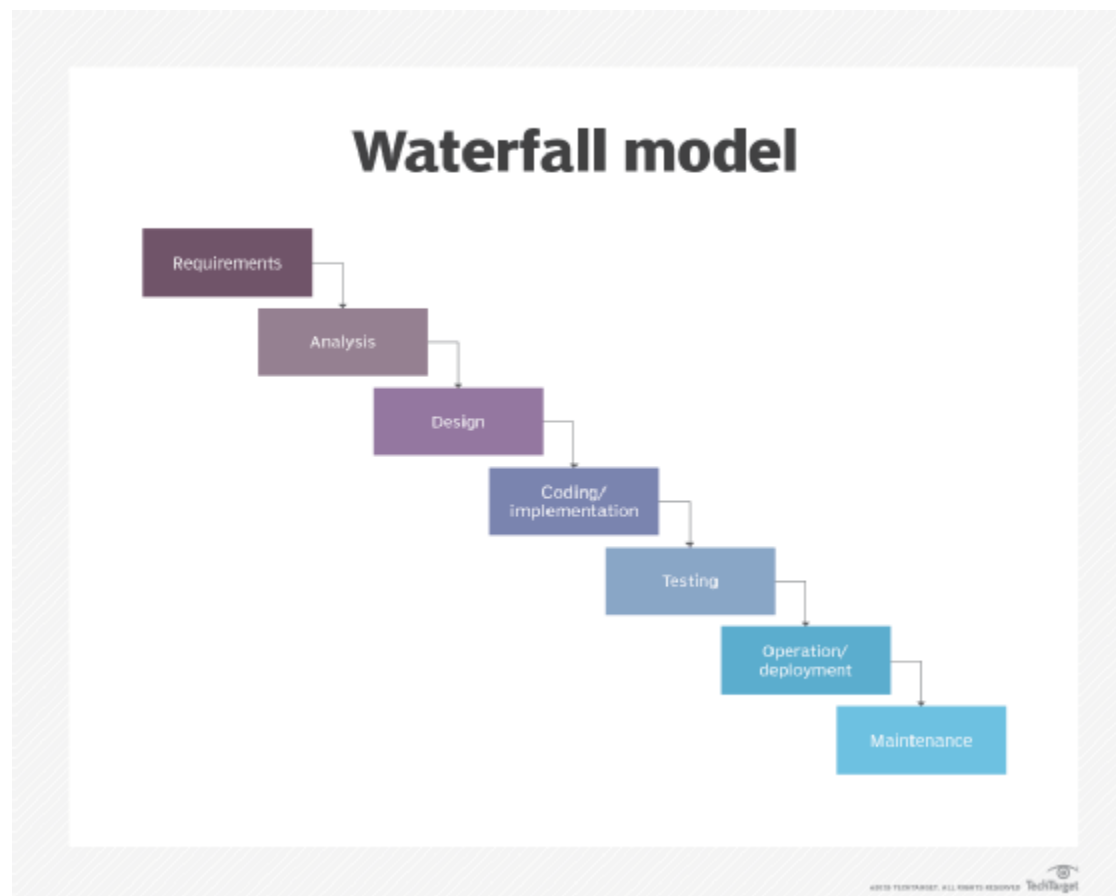


Figure 1 Waterfall methodology

The sequential phases in waterfall model:

- **Requirements:** First, we talk to users and stakeholders to figure out what they need from the software. Then we write down all these requirements in a document called the Software Requirement Specification.

- **Analysis:** Once we've collected all the wishes, we look at them closely. We try to understand what's really important and what might be less important. It's like sorting through a bunch of ideas to figure out which ones are most crucial.
- **Design:** Based on the requirements we collected, we make a plan for how the software will be built. This plan includes things like how different parts of the software will work together, what each part will do, and how they will communicate.
- **Coding/Implementation:** Now it's time to actually write the code for the software. We break the project into smaller pieces and work on each piece separately. Once each piece is done, we test it to make sure it works properly.
- **Testing:** After all the pieces of the software are done, we put everything together and test the whole thing to make sure it works as expected and meets all the requirements we collected at the beginning.
- **Operation/Deployment:** Once the testing is complete and we're sure the software works well, we install it in the real environment where it will be used. We make sure everything is set up correctly and test it again just to be sure.
- **Maintenance:** Finally, once the software is being used, we keep an eye on it to make sure it keeps working well. We fix any problems that come up, add new features if needed, and make improvements to keep it running smoothly.

## 2. Work Breakdown Structure (WBS)

A Work Breakdown Structure (WBS) is like a roadmap that breaks down a big project into smaller, manageable tasks. It's like dividing a big pizza into slices so everyone knows what they need to do to make the whole thing happen.

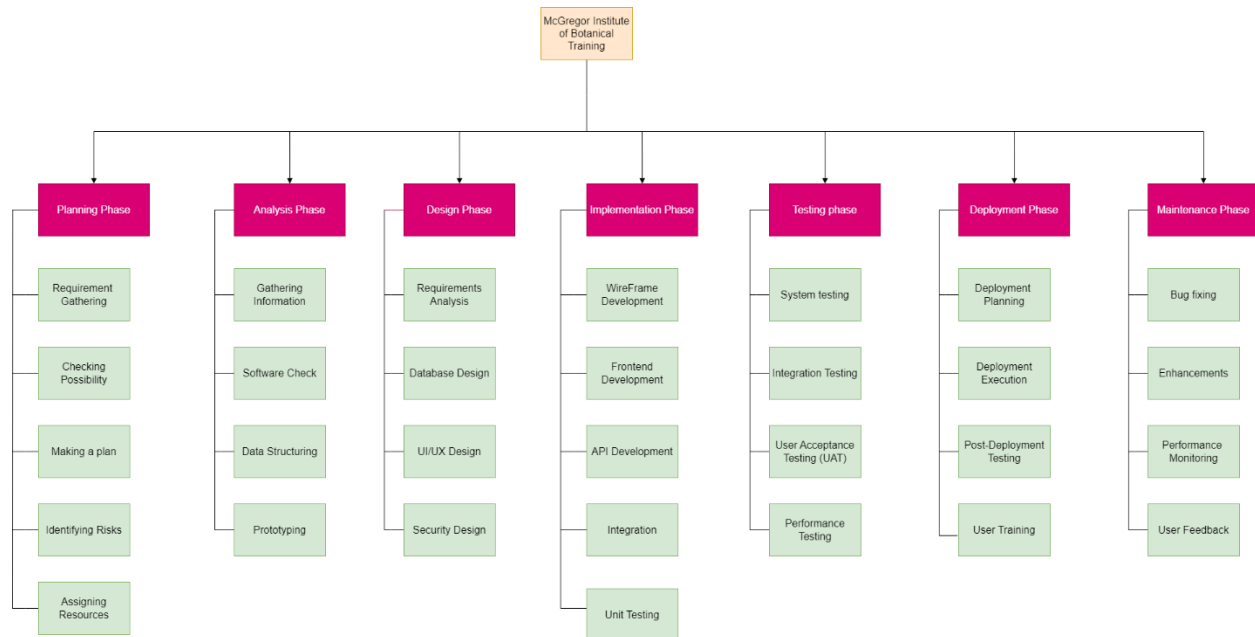


Figure 2 Work Breakdown Structure



### 3. Gantt Chart

A Gantt chart is like a visual schedule for a project. On one side, it lists all the tasks, while on the other, it lays out a timeline. Each task is represented by a bar, indicating its start, end, and duration. The McGregor Institute of Botanical Training employed this chart to map out their work, following the step-by-step waterfall method. This tool allowed them to quickly spot tasks progressing smoothly and those falling behind schedule. It's like having a map for your project that helps you on course.

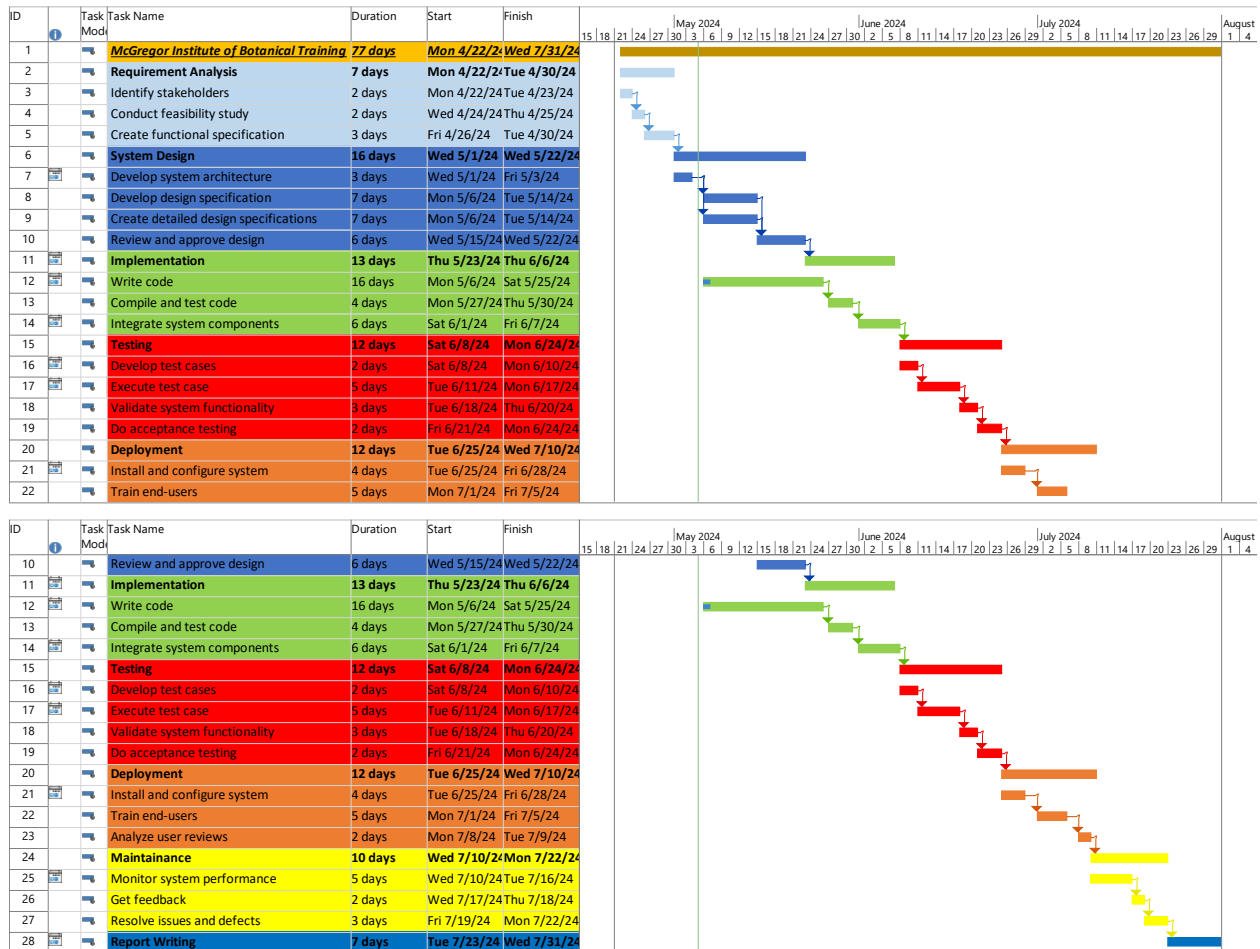
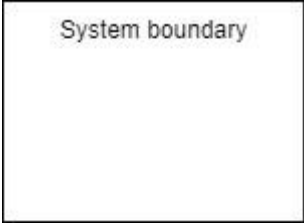
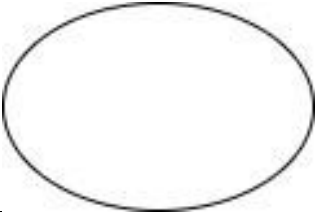


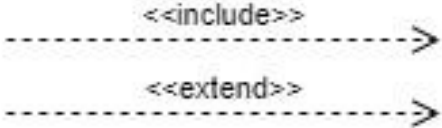


Figure 3 Gantt chart

## 4. Use Case Diagram

A use case diagram is a visual representation that shows how different users (called actors) interact with a system to achieve specific tasks or goals (called use cases). It helps to understand the functionality of a system by illustrating the relationships between users and the actions they can perform within the system. (Walker, 2024)

Name	Symbol	Description
System boundary		It represents the scope of the system. All the use cases of system are placed inside it where as actors who interact with the system are outside.
Use case		A use case represents a goal a user wants to achieve using the system.
Association		It is used to associate a user case and actor in order to indicate that the actor participates in that use case.
Actor		Actors are anything that interacts with the system and needs to share information with it.
Relationship		In a use case diagram, lines between users and tasks show they're connected. These lines also indicate if a task is included in another task or if it extends beyond the usual interactions.

Use case diagram of the “Mcgregor Institute of botanical training’s system” is presented below:

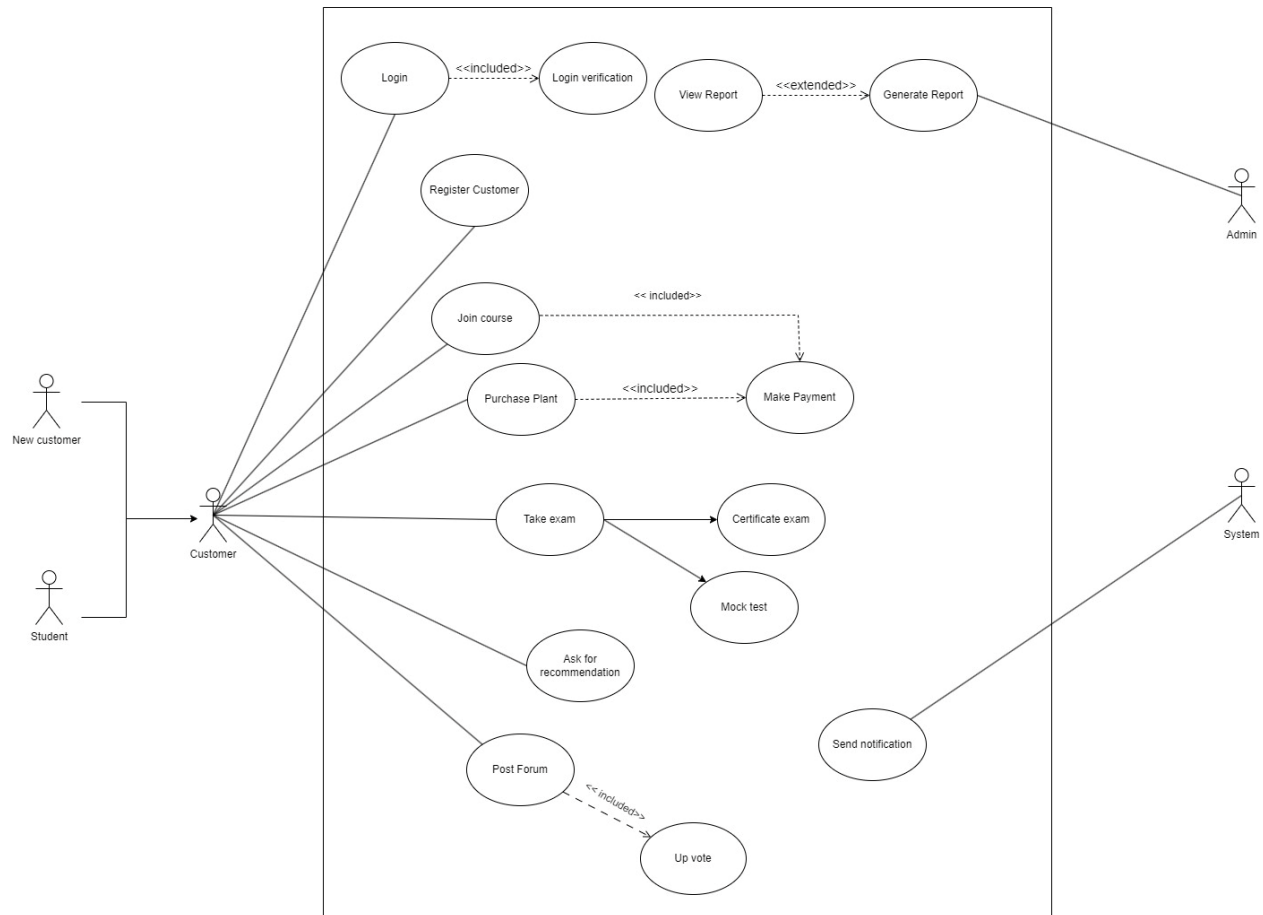


Figure 4 Use case diagram

## 5. High Level Use Case Description

A high-level use case is like a quick summary that explains what a task is about in simple words. It's not super detailed, just enough to understand the task easily. The main goal is to show how complex the task might be to develop.

Use Case	Register Customer
Actor	Customer
Description	A new customer creates an account by providing their personal details to access features.

*Table 1 High level use case of Register Customer*

Use Case	Join Course
Actor	Customer
Description	If a customer is interested in learning and graduate, they can sign up for the Institute of Botanical Training course. These courses offer both paid and unpaid options.

*Table 2 High level use case of Join Course*

Use Case	Purchase Plant
Actor	Customer
Description	A customer discovers a plant they like, choose it for purchase and then pays for it.

*Table 3 High level use case of Purchase plant*

Use Case	Ask for recommendation
Actor	Customer
Description	A customer is asking the experts for advice on which plants or crops would grow well in their soil.

*Table 4 High level use case of ask for recommendation*

Use Case	Take exam
Actor	Student
Description	Students take exam on a particular topic or subject.

*Table 5 High level use case of take exam*

Use Case	Post forum
Actor	Customer
Description	A customer joins an online forum to share information, ask questions, connect with others, comment on post, and gain knowledge.

*Table 6 High level use case of Post Forum*

Use Case	Make Payment
Actor	Customer
Description	The customer begins paying by choosing how they want to pay and giving the needed information, then the system checks everything and send a message confirming the payment.

*Table 7 High level use case of Make Payment*

Use Case	Generate Report
Actor	Customer
Description	The admin creates reports about customer reports about customers, business revenue and details of plants bought using data to analyse and make decisions and setting criteria for the reports.

*Table 8 High level use case of Generate report*

Use Case	View Report
Actor	Customer
Description	The customer looks at a report that was made to analyse and make decisions.

*Table 9 High level use case of View Report*

Use Case	Login
Actor	Customer, Student
Description	The customer provides his/her email or username and password and system verify the provided credentials that grants access to the customer.

*Table 10 High level use case of Login*

Use Case	Send Notification
Actor	System
Description	The system sends messages and alerts to the customers.

*Table 11 High level use case of Send Notification*

## 6. Expanded Use Case Description

### a. Register Customer

**Use Case:** Register Customer

**Actor:** Customer

**Description:** A new customer creates an account by providing their personal details to access features.

Typical course of event:

Customer	System
1. Customer visits the registration page.	
	2. A form is presented for the customer to fill in their necessary information.
3. The customer provide their personal details.	
	4. The information is checked for validity and a customer account is created.
	5. A confirmation message is sent to the customer.
6. The customer receives a confirmation of successful registration.	
	7. The customer is redirected to the welcome page.

*Table 12 Expanded use case of "Register Customer"*

Alternative course:

- In step 2, if the customer's details are not valid when they fill out the forum, the use case stops there.
- In step 3, if the customer decides not to sign up for membership after reaching, the process skips directly to step 6.

**b. Join Course**

**Use Case:** Join course

**Actor:** Student

**Description:** If a customer is interested in learning and graduate, they can sign up for the Institute of Botanical Training Course. This course provides both paid and unpaid options.

Typical Course Events

Student	System
	1. The system display available course.
2. Student selects the desired course.	
	3.System checks calculate cost and check paid and unpaid.
	4.The system displays payment method.
5. Student makes payment.	
	6. The system enrol student in course.

*Table 13 Expanded use case of Join Course*

Alternative Course of event:

- In step 2, If customer decides not to proceed with join course after browsing the course. They can return to main menu.



## 7. Communication/Collaboration Diagram

A collaboration diagram is a visual representation that illustrates how different objects or actors in a system interact with each other to achieve a specific task or goal. It shows the flow of communication between components, including the messages exchanged and their sequence, helping to analyse and refine the system's design during the software development lifecycle. There are 4 major components in collaboration diagram and they are:

- **Objects:**

An object represents a specific entity or component within the system, such as user, a database, or software module. It's usually depicted as a rectangle with the object's name written inside.

- **Actor:**

An actor is an external or relationship between objects or actors in the collaboration diagram. It shows how they interact or communicate with each other to achieve a particular task.

- **Link:**

A link is a connection or relationship between objects or actors in the collaboration diagram. It shows how they interact or communicate with each other to achieve a particular task.

- **Message:**

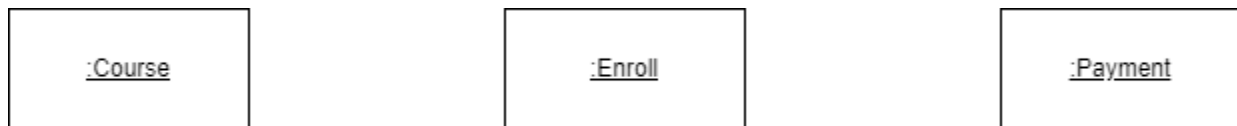
A message represents communication between objects or actors in the system. It indicates the flow of information, requests, or responses between components. Messages are depicted as arrows or lines between objects or actors, often labelled with the content of the communication.

Here, I have decided to make the collaboration diagram of the use case “Register Customer”. There are sequence of steps that are required to be carried out while registering the new customer. The collaboration diagram of registering a customer in the system of “Mcgregor Institute of Botanical Training” can be produced by following steps:

### 7.1 Steps included in Communication/collaboration diagram

#### 1. Determining the domain class and creating its objects

At first, we go through the expanded description of the use case “Register Customer” and select the domain classes. The domain classes for registering customer are customer, registration, Login and the objects of domain classes as follow:



*Figure 5 Domain classes of collaboration diagram*

## 2. Determining the control object

The control object is like the manager of a team made up of boundary and entity objects. It's responsible for overseeing how information moves between them. The control object for this use case is RegisterCustomer which is shown below:

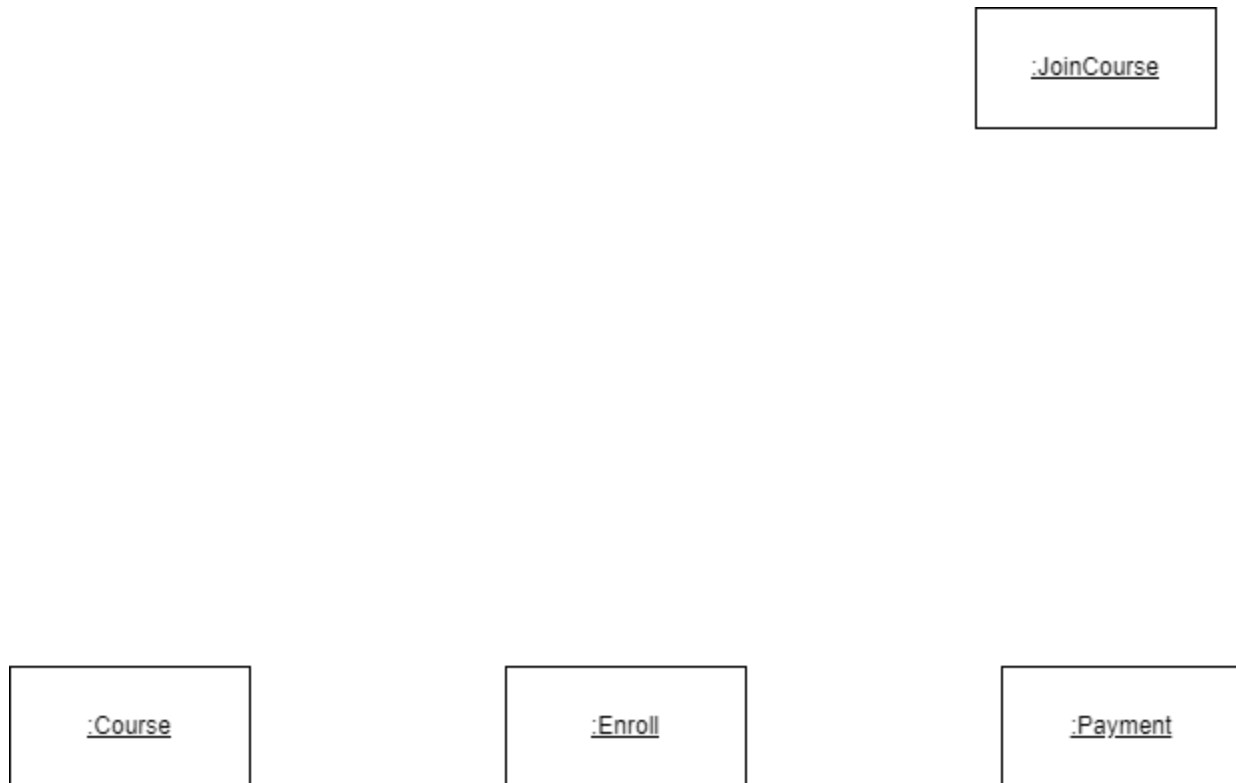


Figure 6 Adding control object for collaboration diagram

### 3. Determining the boundary object

Boundary objects control how users interact with the system on screens. In this case, 'RegisterCustomerUI' is the boundary object.

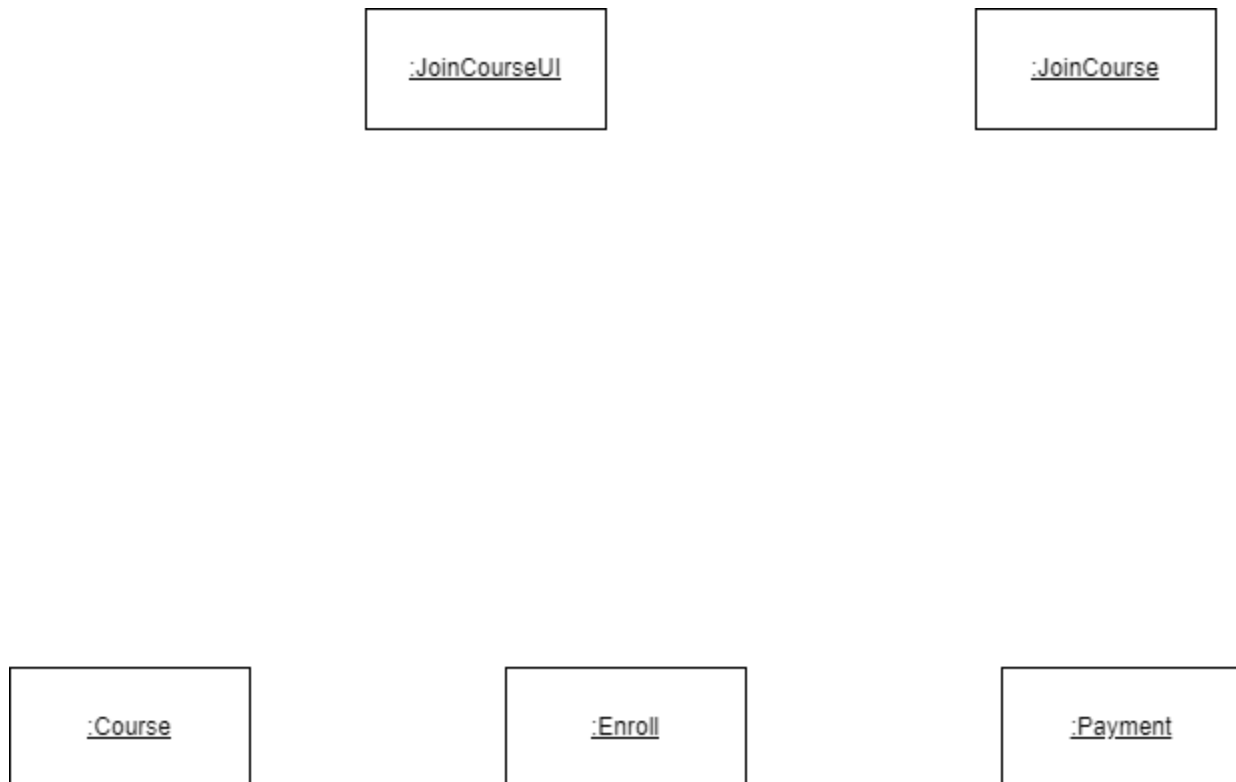


Figure 7 Adding Boundary object

#### 4. Determining the actor of use case

Actors are external entities like users, customer, or suppliers that interact with the system. They send receive message to and from the system, representing what goes into the system and what comes out. In this case, the actor is 'Customer'.

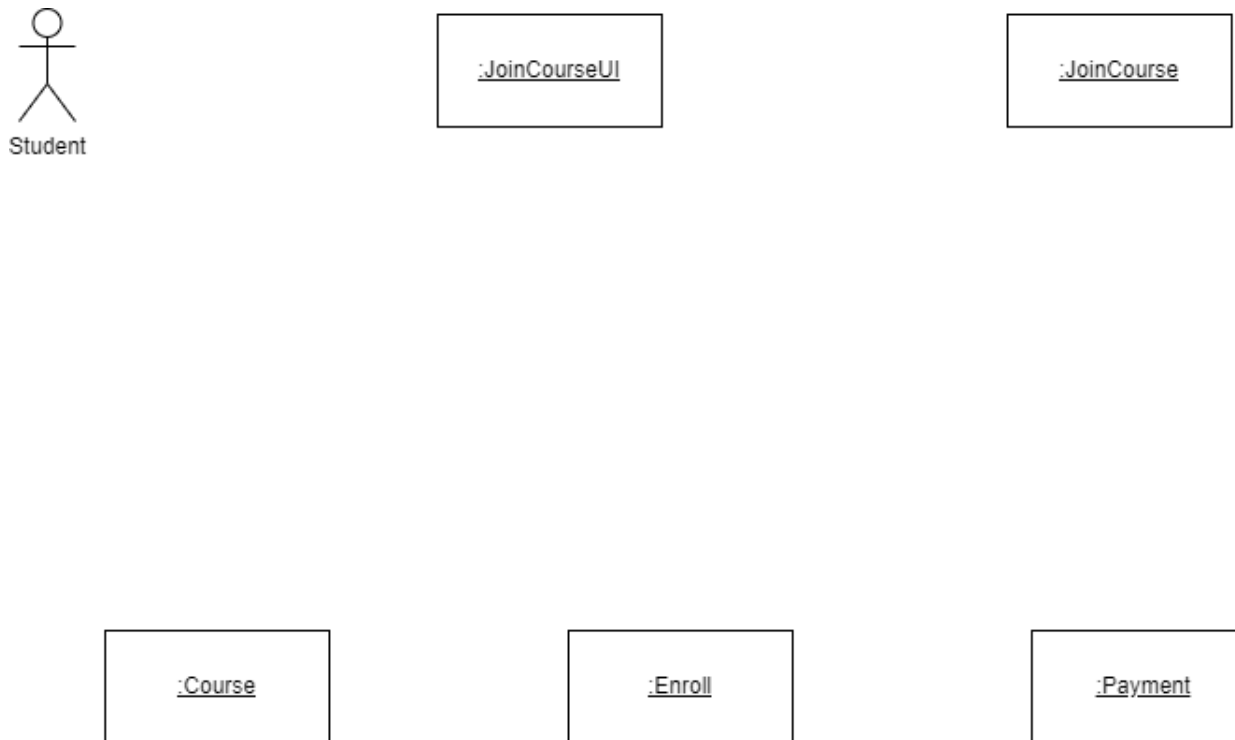


Figure 8 Adding actor for Collaboration diagram

## 5. Association

Now, we match up the actor with the boundary object, control object, and domain class objects, that are involved.

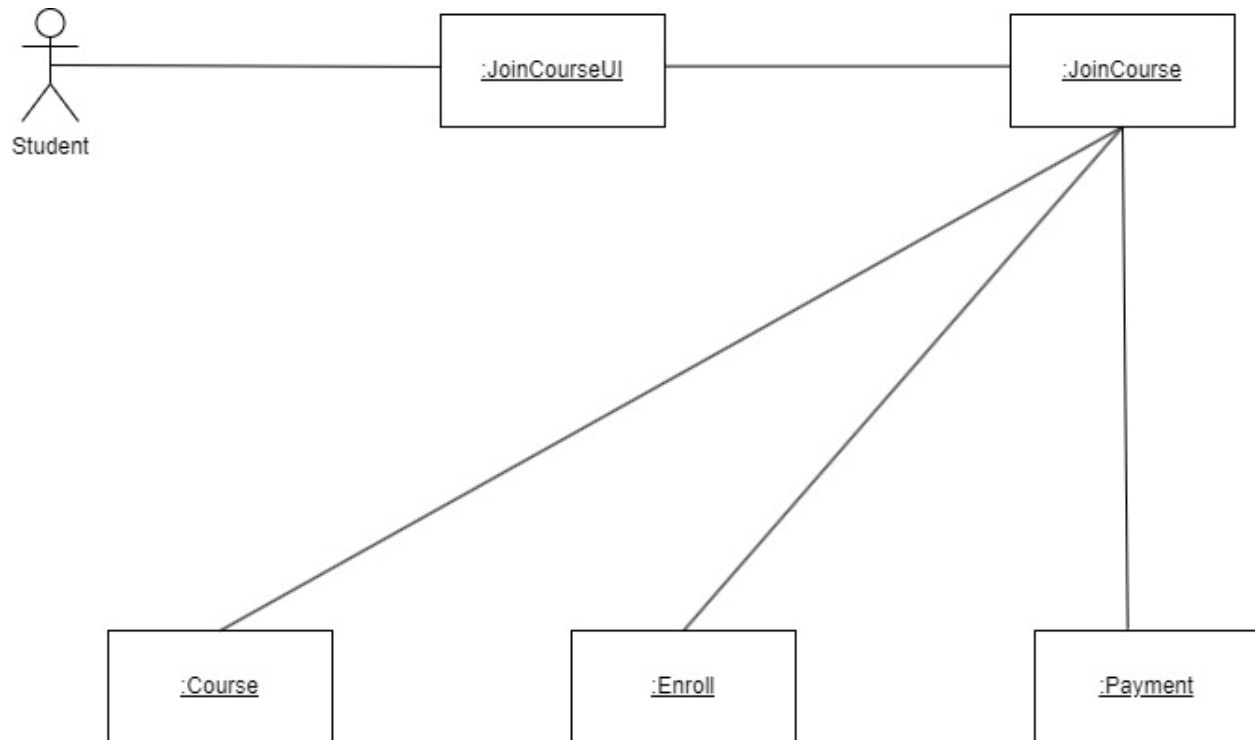


Figure 9 Association

## 6. Adding messages

After connecting all the factors in the diagram, the next step is to illustrate how they communicate by adding messages.

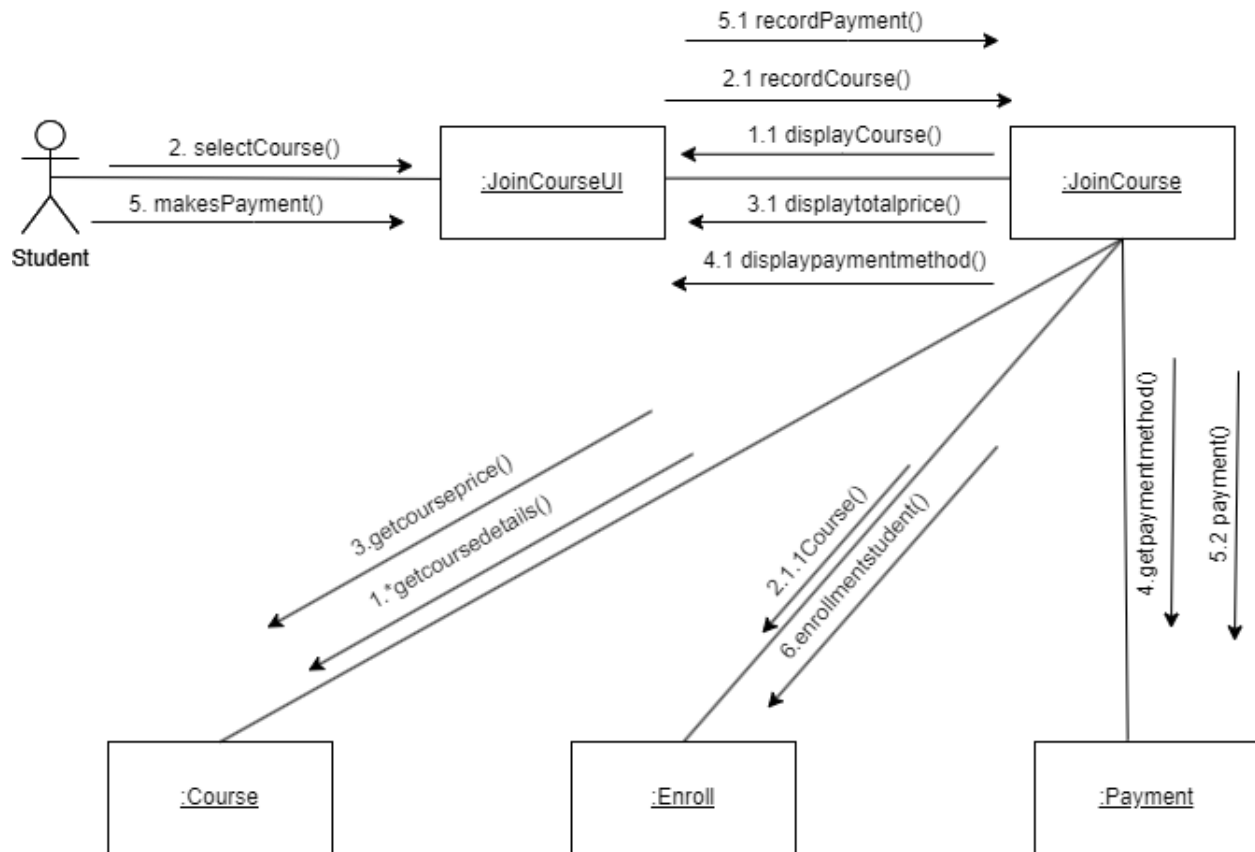


Figure 10 Adding messages

## 7.2 Final Collaboration diagram

After completing all the steps, the final collaboration diagram of the use case “Register Customer” is as follow:

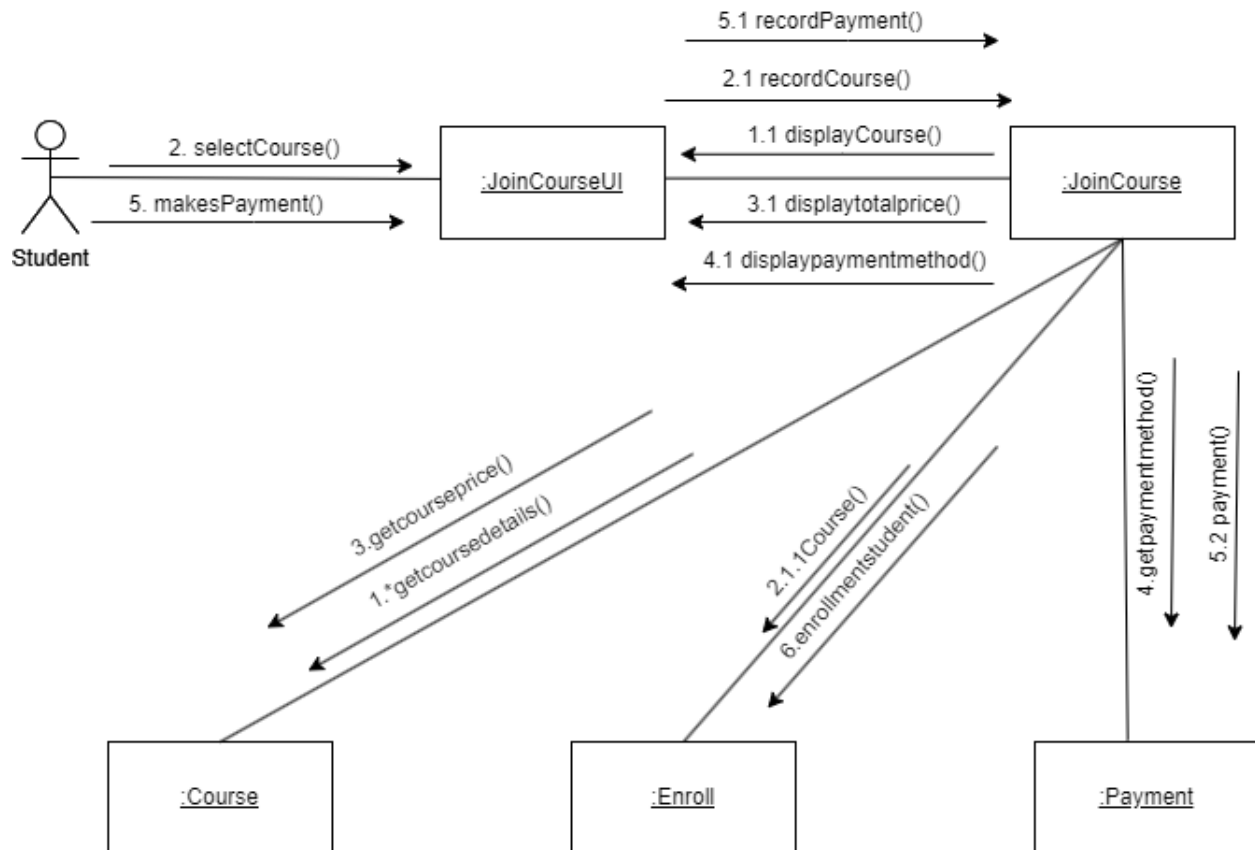


Figure 11 Final collaboration diagram



## 8. Sequence Diagram

A sequence diagram is a type of diagram that displays the interactions between different parts of a system in a step-by-step order to accomplish a task. It uses vertical lines called lifelines to represent each part. Horizontal arrows, known as message lines, show the messages or actions passed between the parts. These messages are labelled with their names and any additional information they carry. A part from illustrating how information moves between parts, sequence diagrams can also indicate when messages are sent and how long they take. The major components of sequence diagram are:

- Objects
- Lifelines
- Messages
- Activation

The sequence diagram for McGregor Institute of Botanical Training is created on the collaboration diagram created earlier. Sequence diagram for the use case (McGregor Institute of Botanical Training) is created with following steps:

### 8.1 Steps involved in Sequence Diagram

#### 1. Finding the domain classes and determining its object

The domain classes for sequence diagram are similar to that of in collaboration diagram i.e. Course, Enroll, Payment. So, the required objects for this use case are:

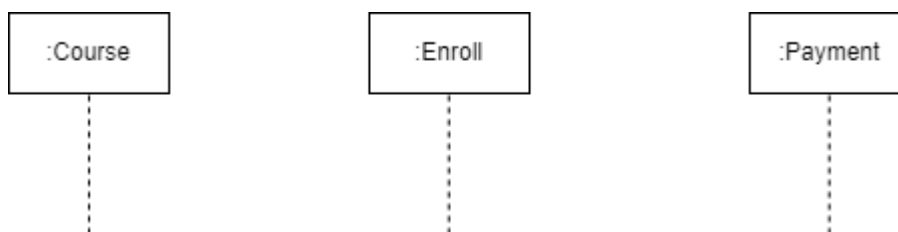


Figure 12 Domain classes

## 2. Adding Control Object lifeline

The lifeline with a control element indicates a controlling entity that organize the interactions between boundaries and entities. For this use case, the control object lifeline is JoinCourse which is shown below:

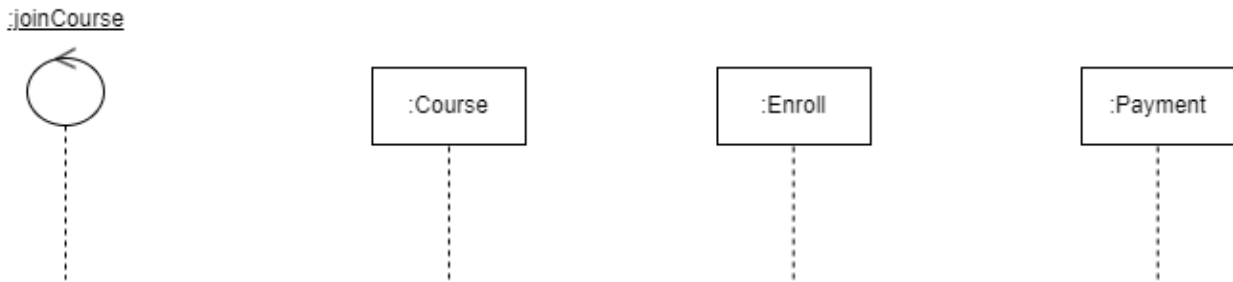


Figure 13 Adding control object lifeline

## 3. Adding boundary object lifeline

The lifeline of a boundary element indicates a system boundary. For this use case, the boundary object life line is JoinCourseUI which is shown below:

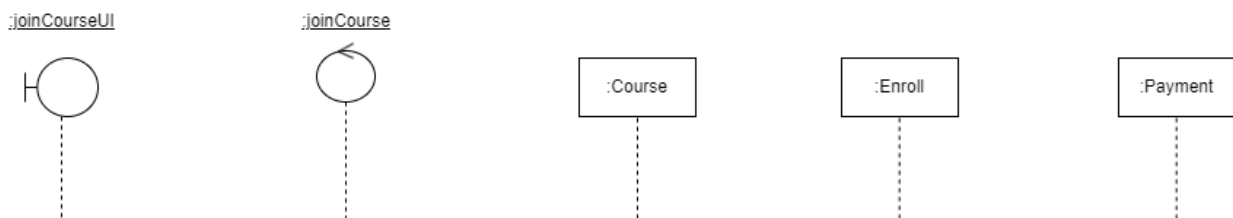


Figure 14 Adding boundary object lifeline

#### 4. Draw Actor lifeline

The external factor that interacts with the system is called actor. If the sequence diagram is owned by a use case, a lifeline notation with an actor element symbol is used. The actor life line for this use case's sequence diagram is given below:

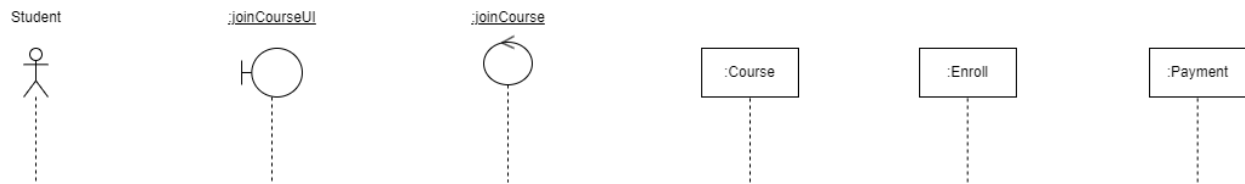


Figure 15 Adding actor Sequence diagram

#### 5. Determining the activation period

Some of the component in the diagram are active for the whole process, where as some are active for only certain period of time. The back shaded rectangle indicate that an object is active during an interaction between two object is in passive state. The length of the rectangle indicated the duration of the object staying active.

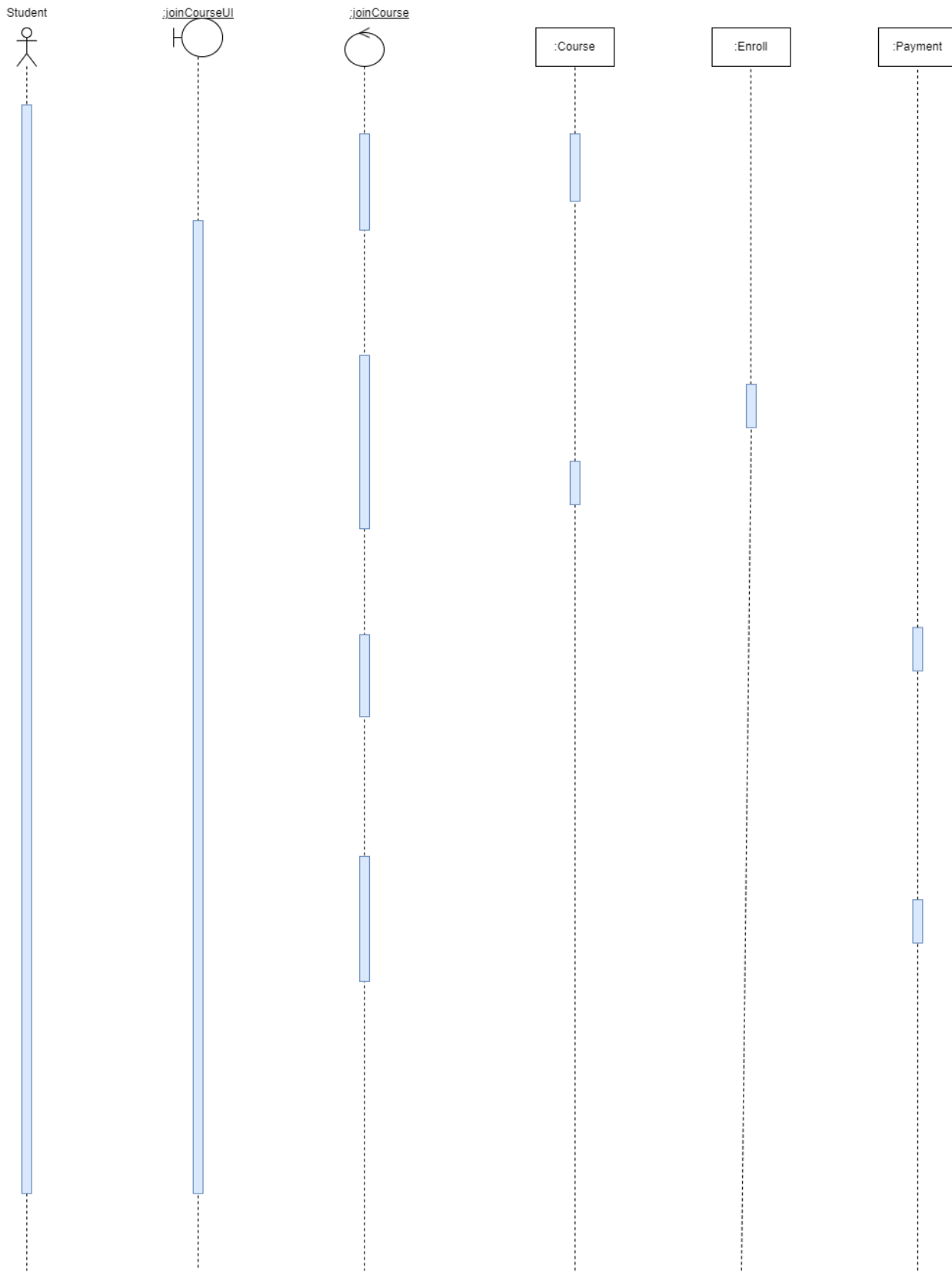


Figure 16 Determining activation period

## 6. Adding message for final diagram

Messages are the most important elements of a sequence diagram. They indicate when one object calls on operation on another object. Now we associate the components with each other by adding messages between them and also, we add the frame which is shown below:

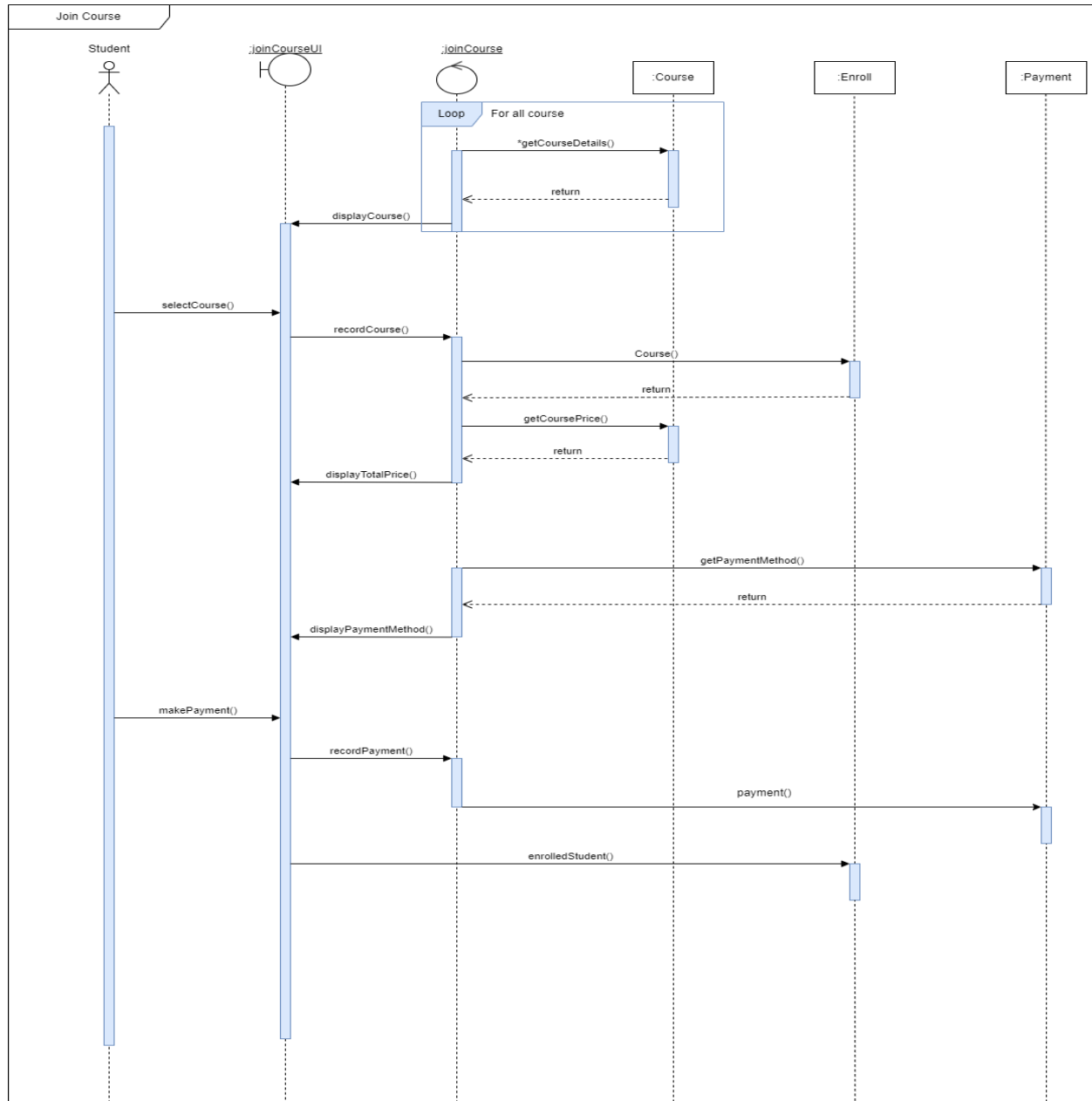


Figure 17 Adding messages

## 8.2 Final sequence diagram

After completing all steps, the final sequence diagram of the use case “Join Course” is shown below:

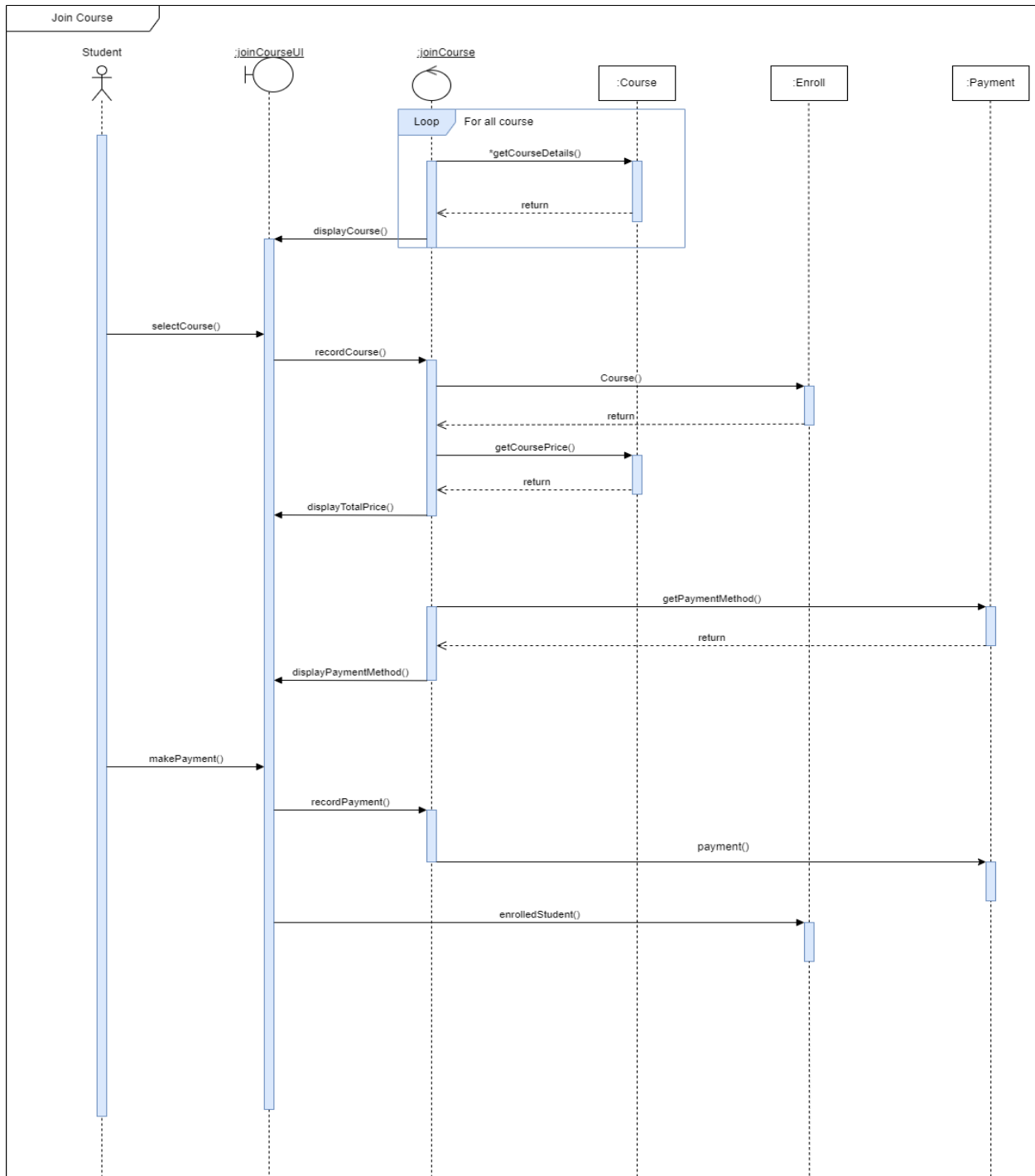


Figure 18 Final Sequence diagram

## 9. Class Diagram

A class diagram in the Unified Modelling Language (UML) is a visual representation that outlines the structure of a system. It depicts the various classes within the system, along with their attributes and operations (or methods), and illustrates how these classes are related to one another. Essentially, it provides a clear picture of the building blocks of the system and how they interact.

### 9.1 Steps involved in creating class diagram

- **Finding the domain classes**

While creating a class diagram, first of all we should list out the domain classes. The classes to support the use cases of “Mcgregor Institute of Botanical Training” are:

Use Case	Domain classes
Register Customer	New customer, Registration
Join Course	Student, Course, Payment
Purchase plant	Customer, Plant, Payment, Purchase
Ask for recommendation	Customer, Plant, expert
Take exam	Student, Exam, Course
Post forum	Customer, Forum
Generate report	Admin, Report
Send notification	Customer, Notification

So, the domain classes needed to support use cases are:

- Customer
- Registration
- New customer
- Student
- Course
- Payment
- Plant
- Expert
- Exam
- Forum
- Admin
- Notification
- Report

- Drawing Domain classes

All the domain classes of the system are drawn below:

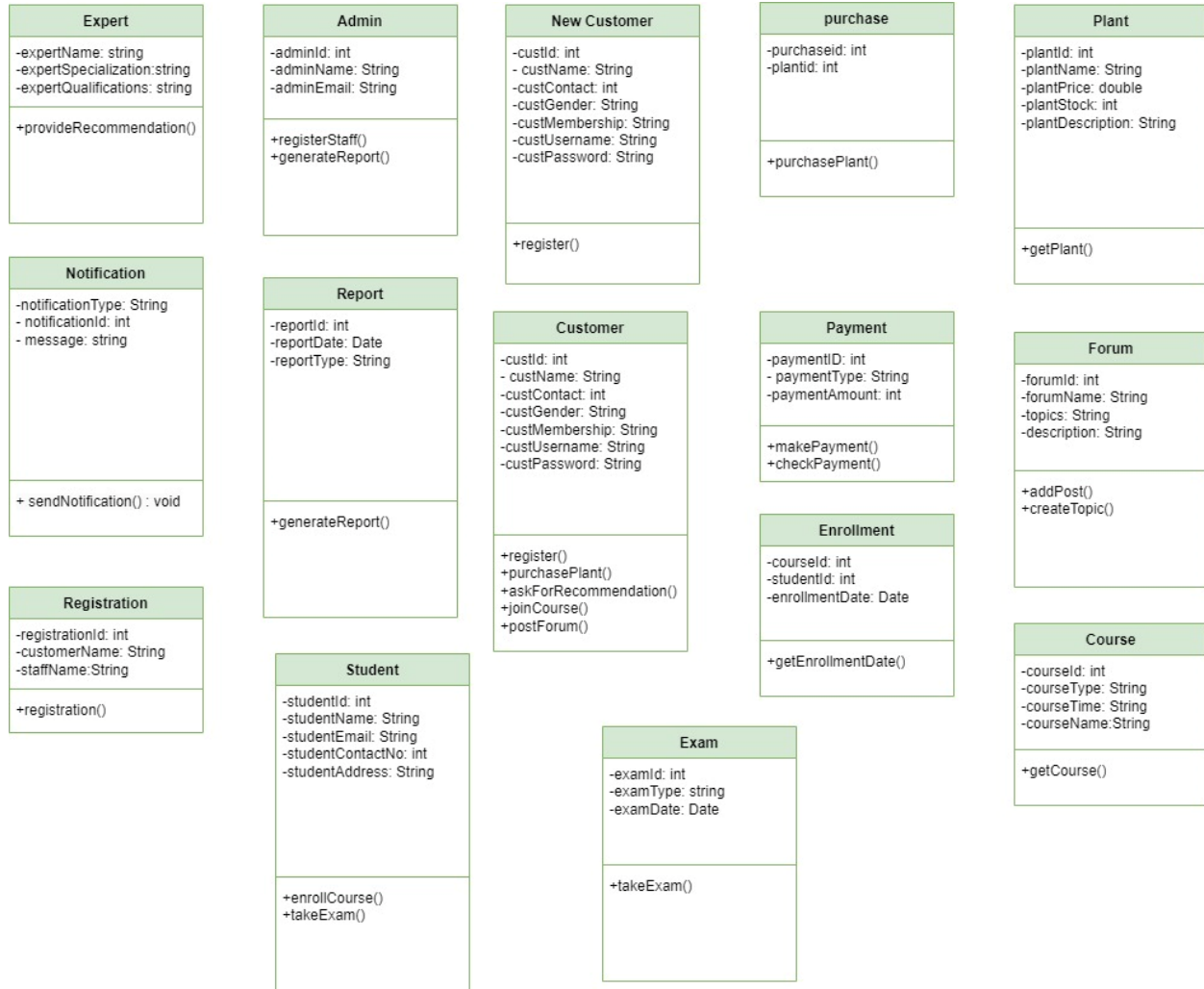


Figure 19 Drawing domain classes



## 1. Relationship

When making a class diagram, there are various types of relationships between classes. These relationships describe how classes are connected to each other and interact.

### a. Association

Association is represented by a solid line connecting two classes. It signifies a connection between instances of one class and instances of other classes.

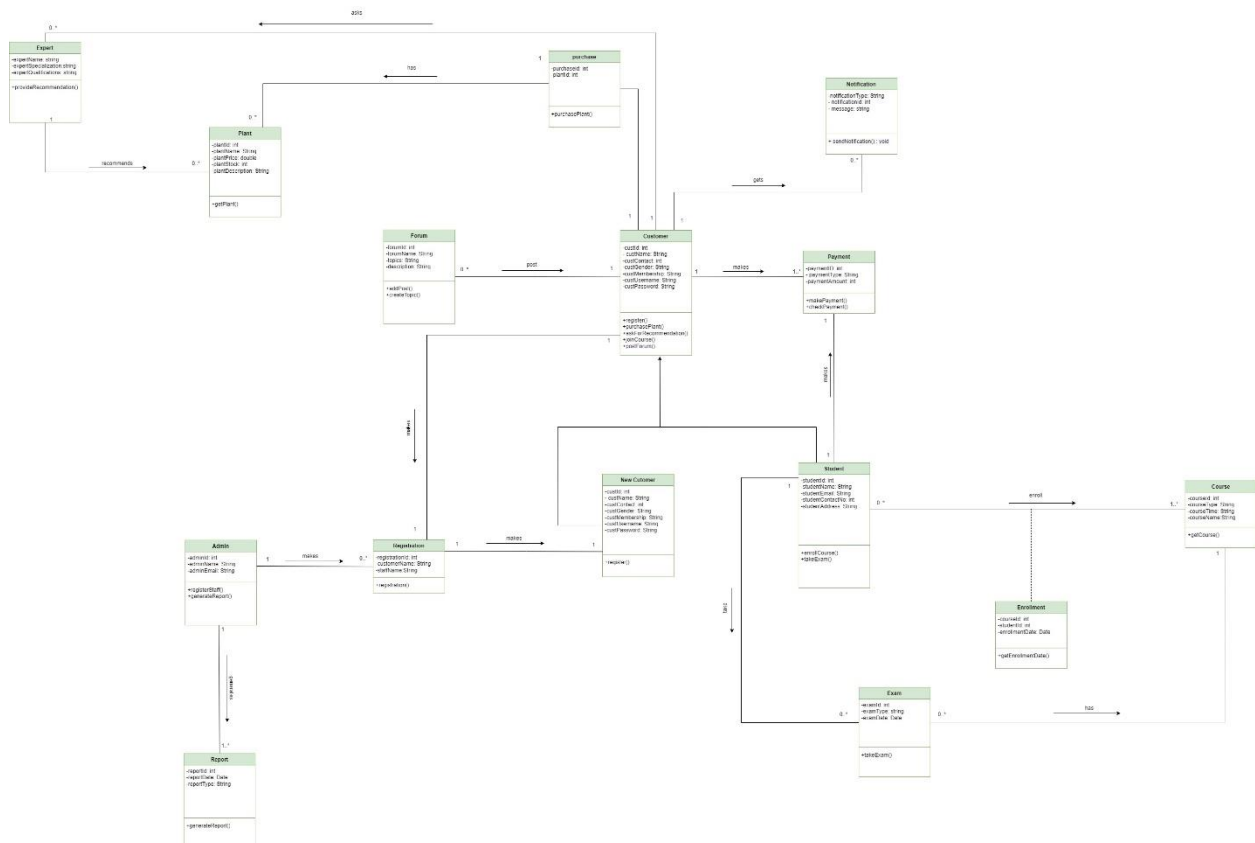


Figure 20 Association in class diagram

There are two sub-types of association, they are aggregation and composition.

### i. Aggregation

Aggregation is a specific kind of association that shows the connection between two classes, where one class is considered a part of the other class. Changing or removing one class doesn't impact the other one. This relationship is illustrated by using an empty diamond shape at the end of the line linking the two classes. In our system, an aggregation relationship appears as follows:

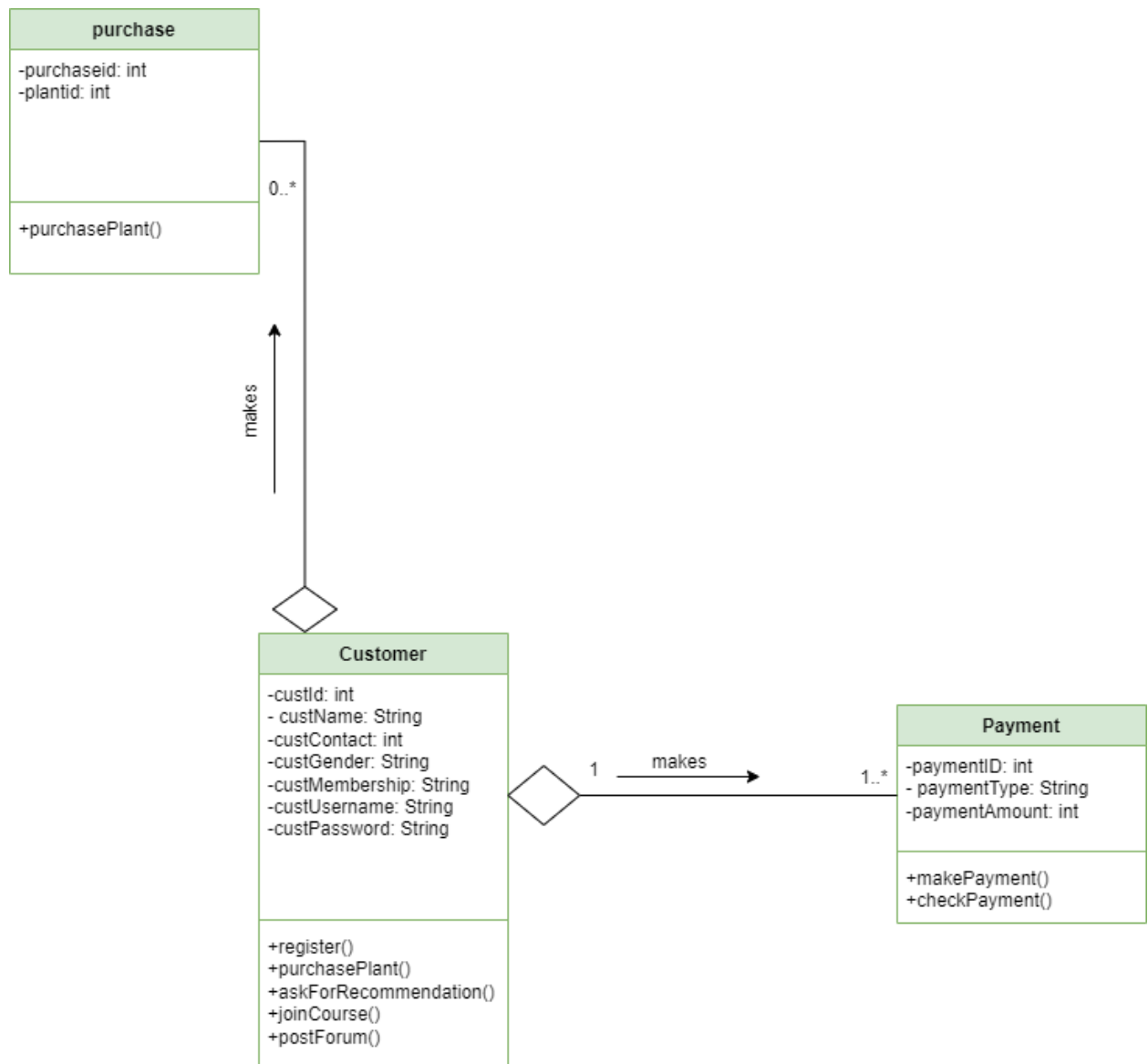


Figure 21 Aggregation in Class diagram

## ii. Composition

Composition is a stronger type of relationship than aggregation, where the smaller parts are completely dependent on the larger whole. In composition, the smaller parts cannot exist independently without being part of the whole. This relationship is represented visually by drawing a line with a filled diamond shape at the end pointing towards the whole class. In our system, we show the composition relation between classes using this same diagram notation.

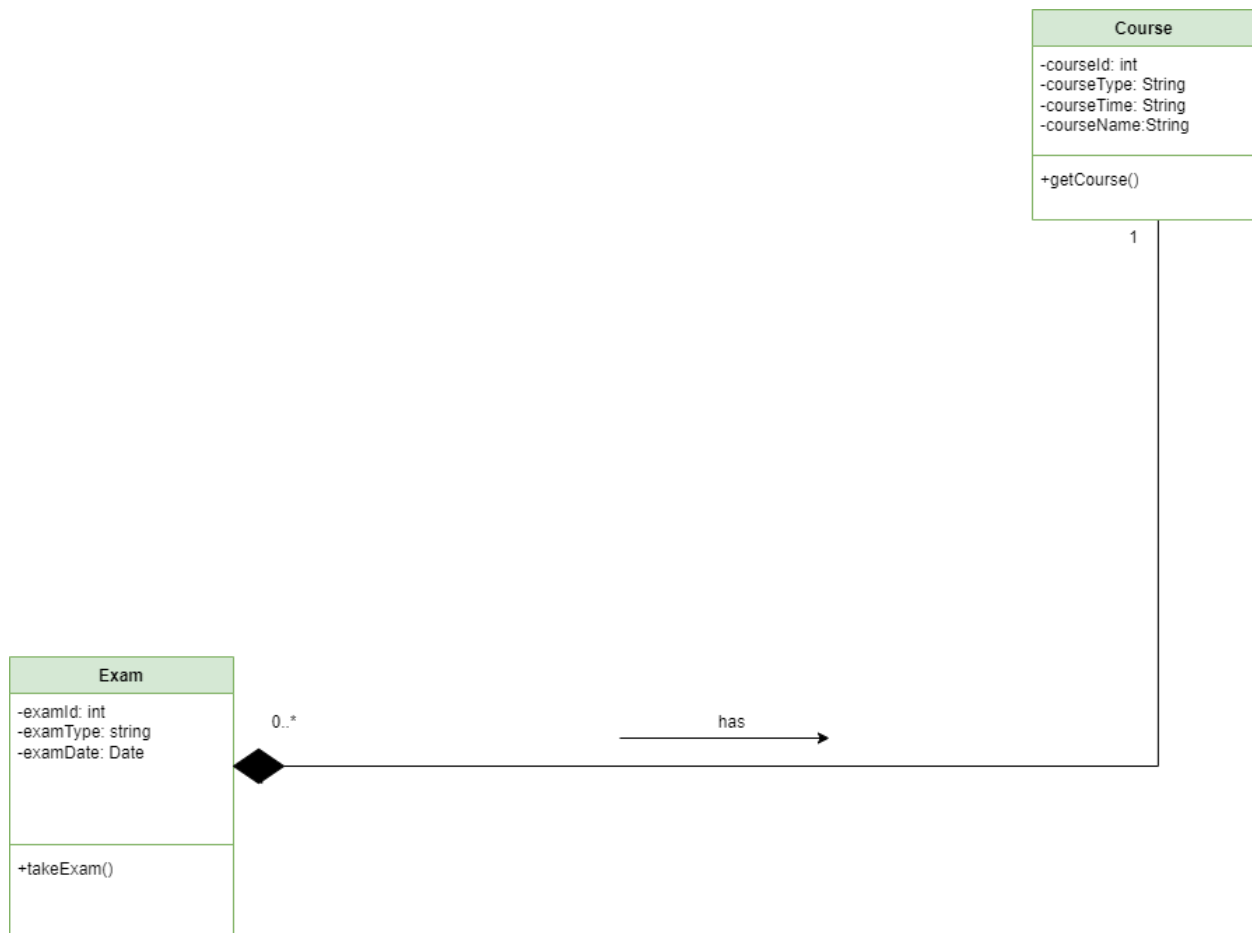


Figure 22 Composition in class diagram

### iii. Inheritance

Inheritance is a way for one class (the subclass) to receive characteristics from another class (the superclass). It's like the subclass is a child, and the superclass is the parent. The child class automatically gets all the traits and abilities of the parent class. We show this inheritance relationship by drawing a solid line with a triangle arrow pointing from the child class to the parent class. The arrow indicates that the child is inheriting from the parent.

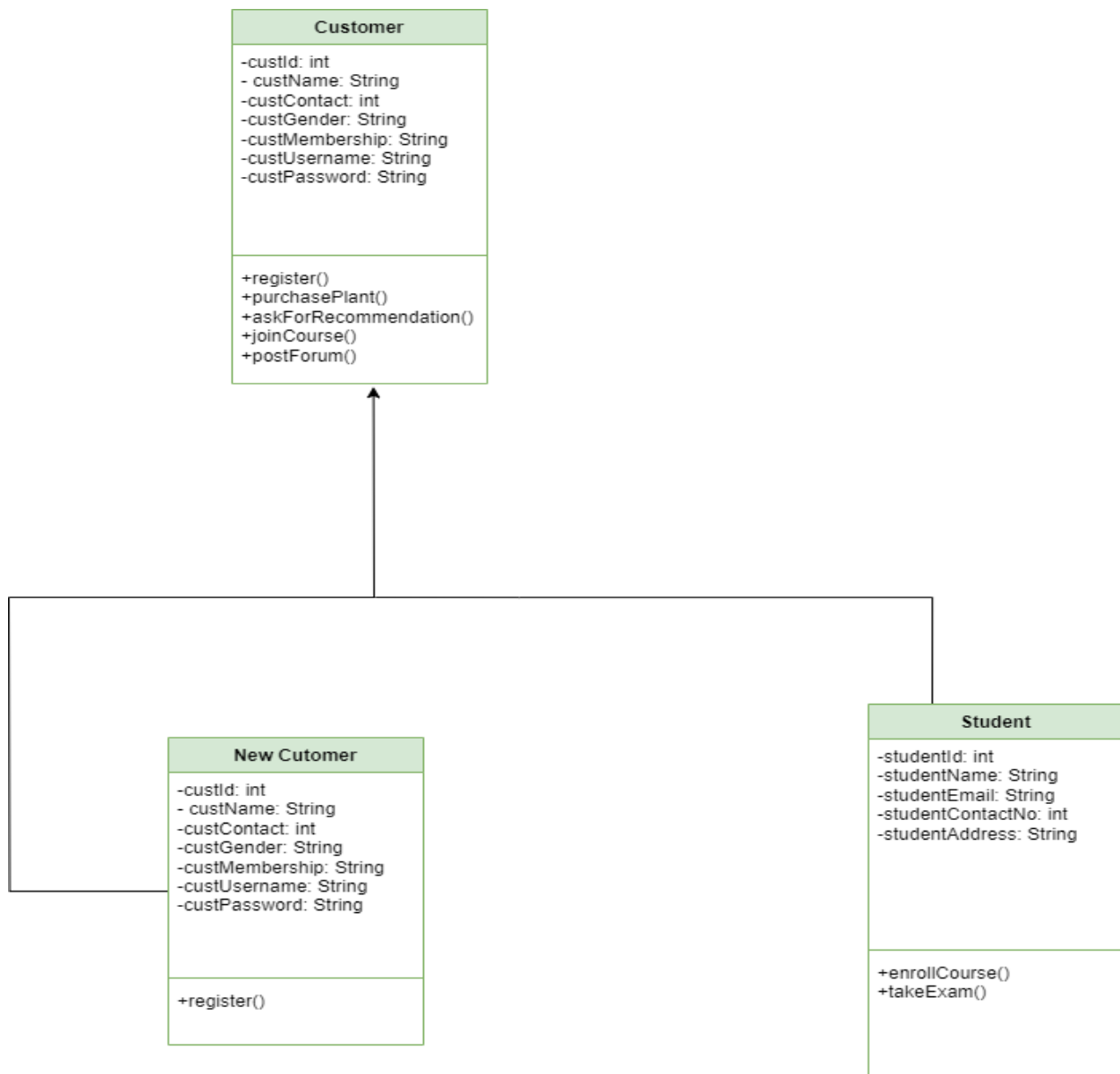


Figure 23 Inheritance in class diagram

## 9.2 Final Class Diagram

After completing all the steps and connecting all classes accordingly, the final class diagram is obtained which is shown below:

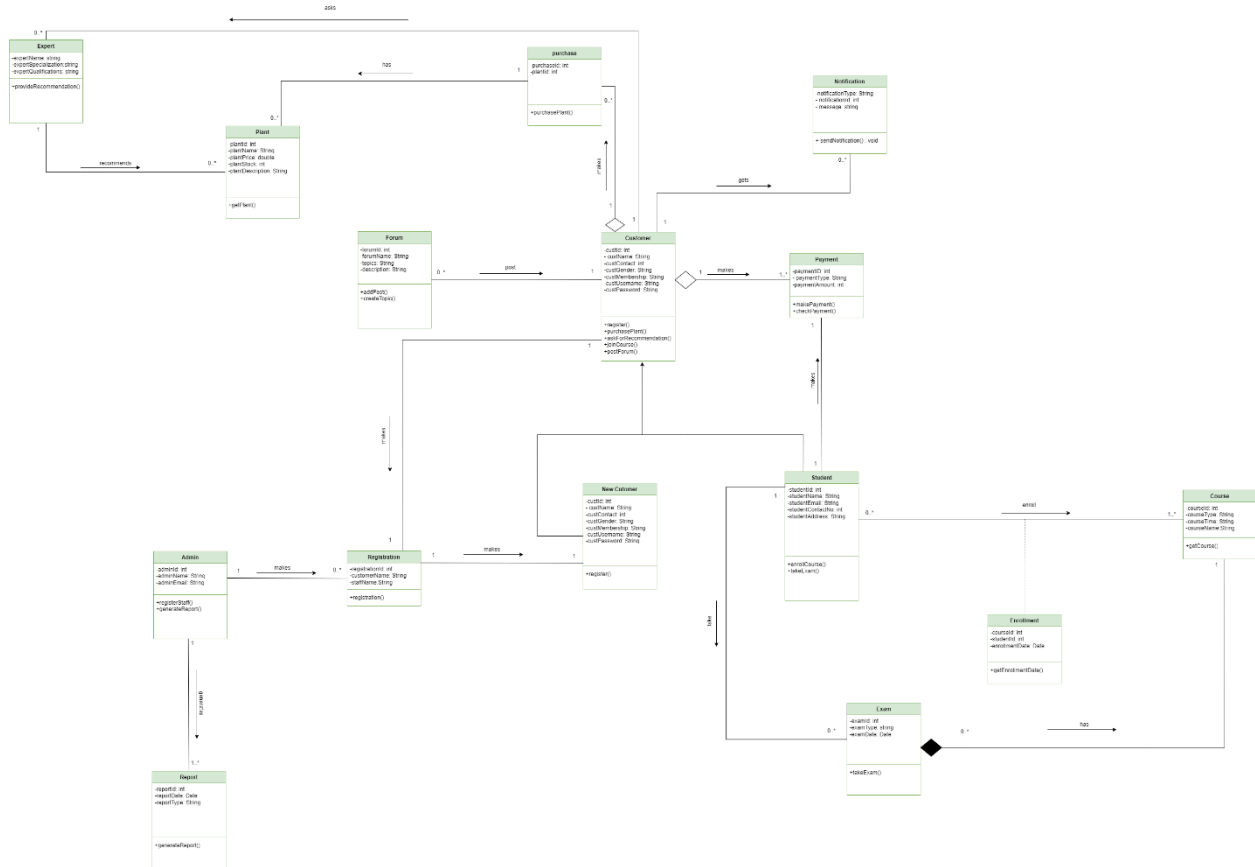


Figure 24 Class diagram

## 10. Further Development

The McGregor Institute of Botanical Training is embarking on a significant expansion of its business operations. In pursuit of this endeavor, the institute has opted to utilize the waterfall methodology for the management and execution of their project. The waterfall methodology is a structured and sequential approach to software development, whereby each phase must be meticulously completed before proceeding to the next. This methodology is particularly suitable for projects with well-defined requirements and minimal anticipated changes throughout the development process.

The project at McGregor Institute has progressed through the initial phases of requirement analysis and system design. During the requirement analysis phase, the project team diligently gathered information regarding the institute's new venture and the specific functionalities required from the software. Through interviews with stakeholders and in-depth discussions with domain experts, the team was able to document detailed requirements, ensuring a comprehensive understanding of the project scope. Key functionalities identified during this phase include the need for a user-friendly system that facilitates various tasks such as purchasing plants, making payments, enrolling in training courses, taking exams, and participating in forums.

Following the requirement analysis phase, the project transitioned to the system design phase. Here, the team focused on creating detailed visual representations to illustrate the structure and behaviour of the software system. Use case diagrams, class diagrams, collaboration diagrams, and sequence diagrams were developed to provide a holistic view of how the software would function and how different components would interact with one another. Additionally, a user-friendly interface was designed to enhance usability and ensure a seamless experience for both users and administrators.

As the project advances, the next phase on the horizon is implementation. During this phase, the project team will translate the design specifications into actual code, bringing the envisioned software system to life. To facilitate this process, the team has selected a

monolithic architecture, which aligns well with the structured approach of the waterfall methodology. In a monolithic architecture, all components of the system are tightly integrated, making it easier to manage and maintain. This architectural choice will ensure consistency and cohesion across the entire system.

In addition to the system architecture, the team has also identified key design patterns that will be utilized during implementation. The Singleton pattern, for example, will be employed to ensure that critical components of the system are instantiated only once, enhancing efficiency and resource management. Similarly, the Factory pattern will be utilized for object creation, promoting code reusability and maintainability. Additionally, the Decorator pattern will be used to dynamically add functionality to existing objects, further enhancing the flexibility and extensibility of the system.

For the development phase, the team has selected appropriate tools and technologies to support their efforts. IntelliJ IDEA, an integrated development environment (IDE) for Java and other programming languages, has been chosen as the primary development tool. Java will serve as the primary programming language for implementing the software's core functionalities. Additionally, relational database management systems (RDBMS) such as MySQL or PostgreSQL will be implemented for data storage and management, ensuring scalability and reliability.

Throughout the development phase, the project team will adhere to a structured approach, breaking down the system into smaller, manageable parts. This modular approach will enable the team to focus on developing core functionalities first, ensuring that the software meets the institute's requirements and objectives. Unit testing will be conducted for each component to verify its functionality and identify any potential issues early in the development process.

Once the implementation phase is complete, the project will transition to the testing phase. This phase is crucial for ensuring that the software meets all specified requirements and functions as intended. Various testing techniques, including functional

testing, integration testing, performance testing, and security testing, will be employed to validate the software's functionality and reliability. Automated testing frameworks such as JUnit and Selenium will be utilized to streamline the testing process and improve efficiency.

Following successful testing, the project will enter the deployment phase. During this phase, the software will be deployed to production servers and made available to McGregor Institute's users. The deployment process will be carefully managed to minimize disruptions and ensure a smooth transition to the new system. Additionally, training sessions will be conducted for the institute's staff to familiarize them with the software and its features, enabling them to effectively utilize the system in their daily operations.

With the software successfully deployed, the project will enter the maintenance phase. During this phase, the project team will be responsible for addressing any issues or bugs that may arise, as well as implementing updates and enhancements based on user feedback. Regular maintenance activities will ensure that the software remains reliable, secure, and up-to-date, meeting the evolving needs of McGregor Institute and its users.

In conclusion, the McGregor Institute of Botanical Training's project expansion is guided by the waterfall methodology, a structured and sequential approach to software development. By meticulously following each phase of the waterfall model, from requirement analysis and system design to implementation, testing, deployment, and maintenance, the project team aims to deliver a high-quality software solution that meets the institute's requirements and objectives. With careful planning, effective collaboration, and diligent execution, McGregor Institute is well-positioned to achieve success in their ambitious expansion efforts.



## 11. Prototype

A prototype is like a rough draft of something new, like a product or invention. Engineers and developers create it to test out their ideas before making the final version. It's not perfect, but it helps them see what works and what needs fixing. They try it out in different ways to make sure it's good enough for people to use. If it works well in these tests, it's more likely to make customers happy. But if there are problems, they might have to change or even stop making it. (kirvan, 2023). I have also created the prototype for the 'Mc Gregor Institute' using 'Figma'.

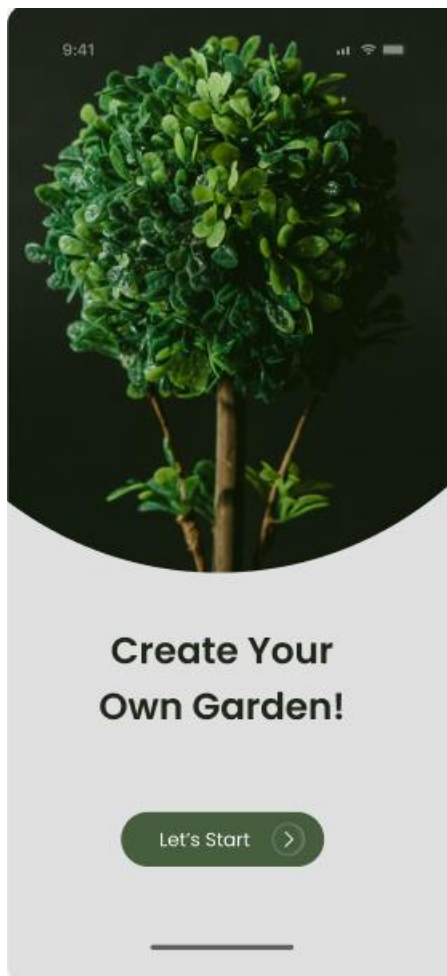


Figure 25 Welcome page



*Figure 26 Splash Screen*

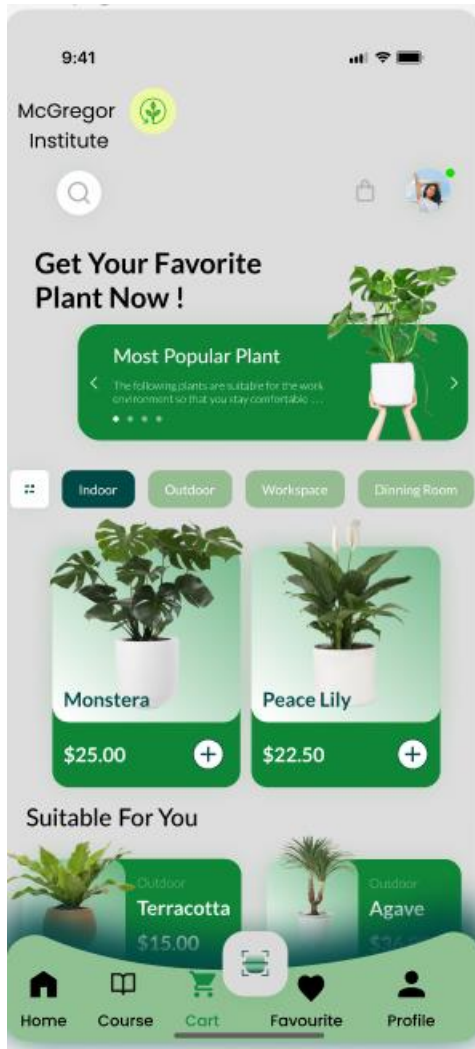


Figure 27 Home page

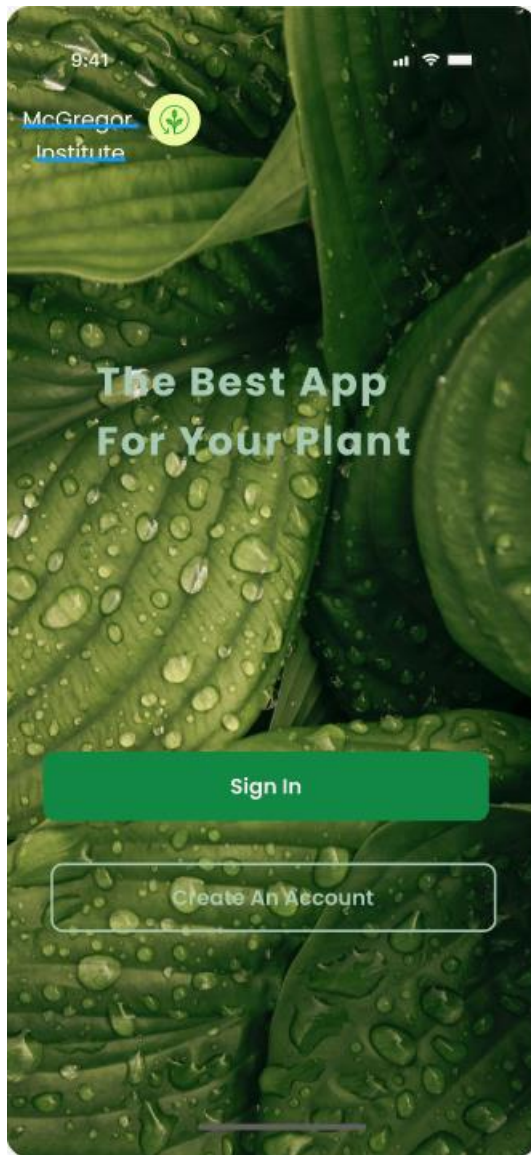


Figure 28 Sign-In page

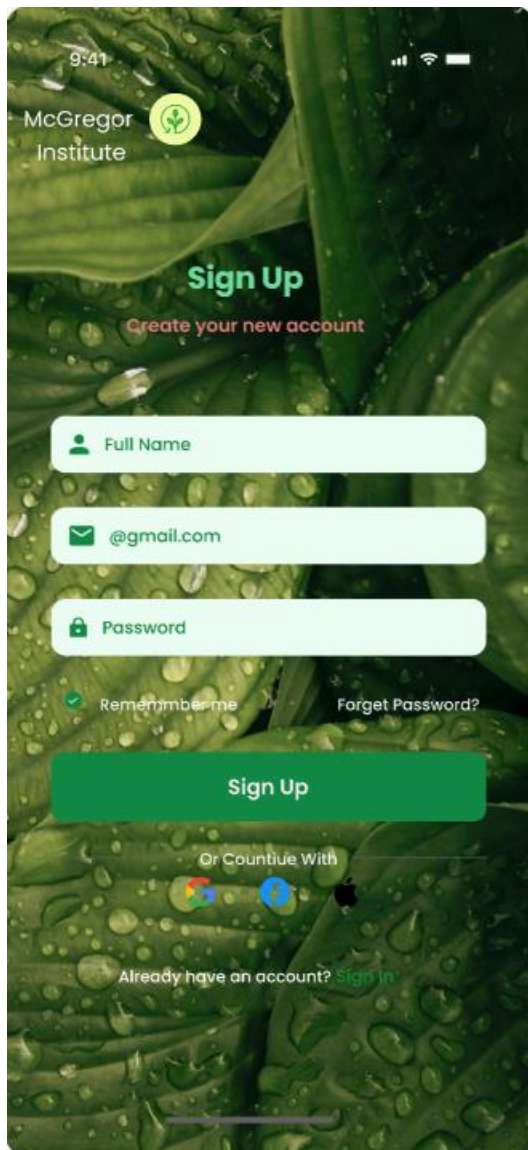


Figure 29 Sign-up page

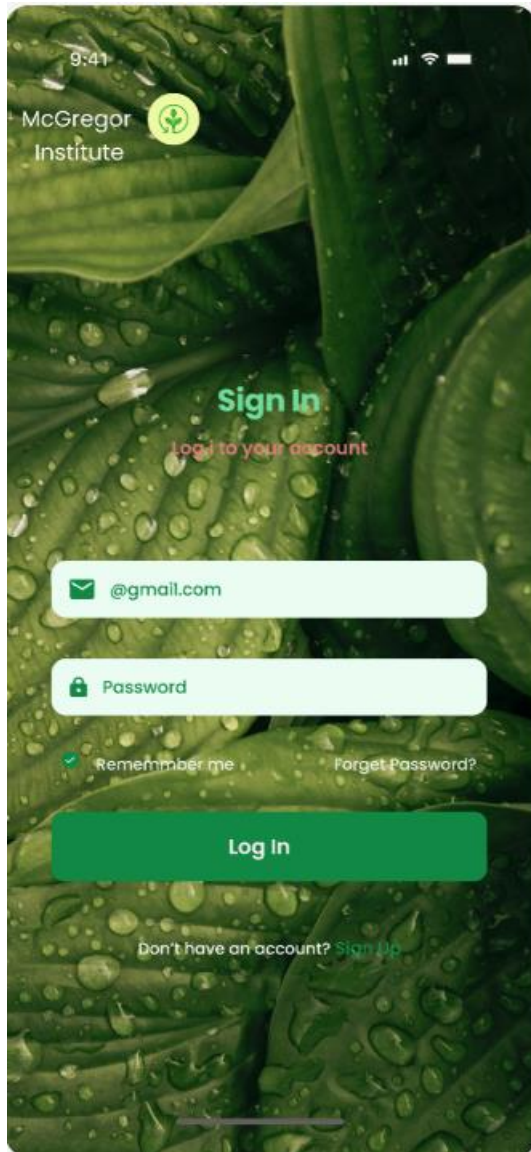


Figure 30 Login page



Figure 31 Product page





Figure 32 Plant description page



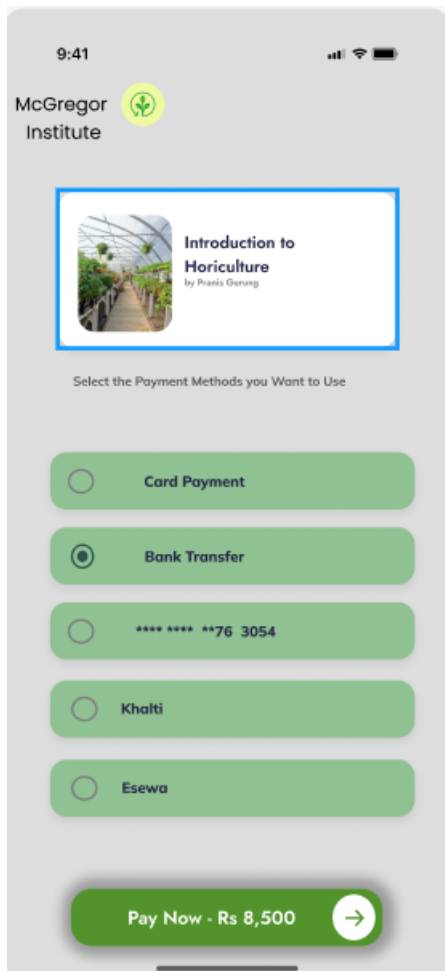


Figure 33 Payment method Product page

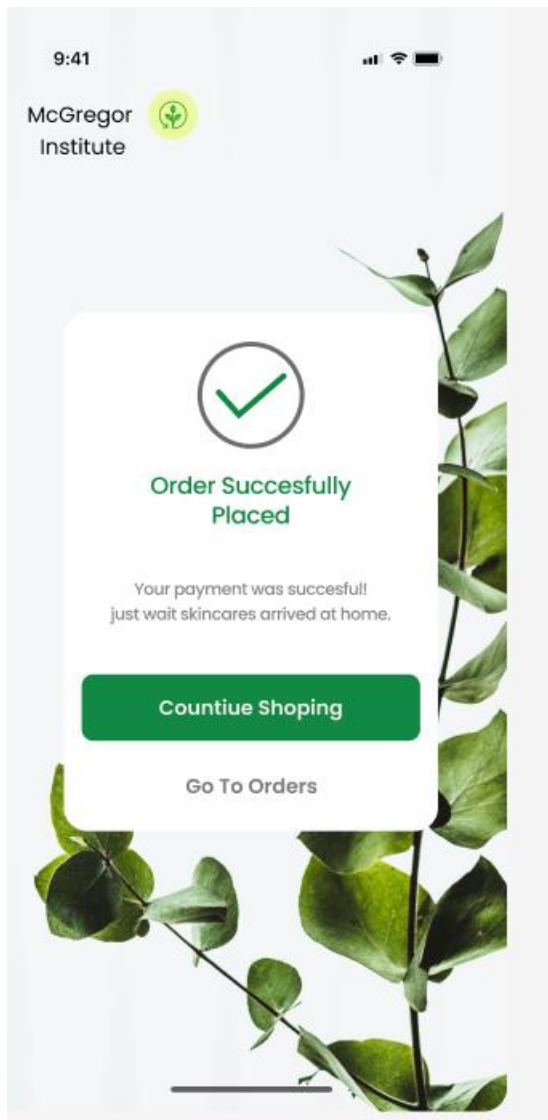


Figure 34 Order page

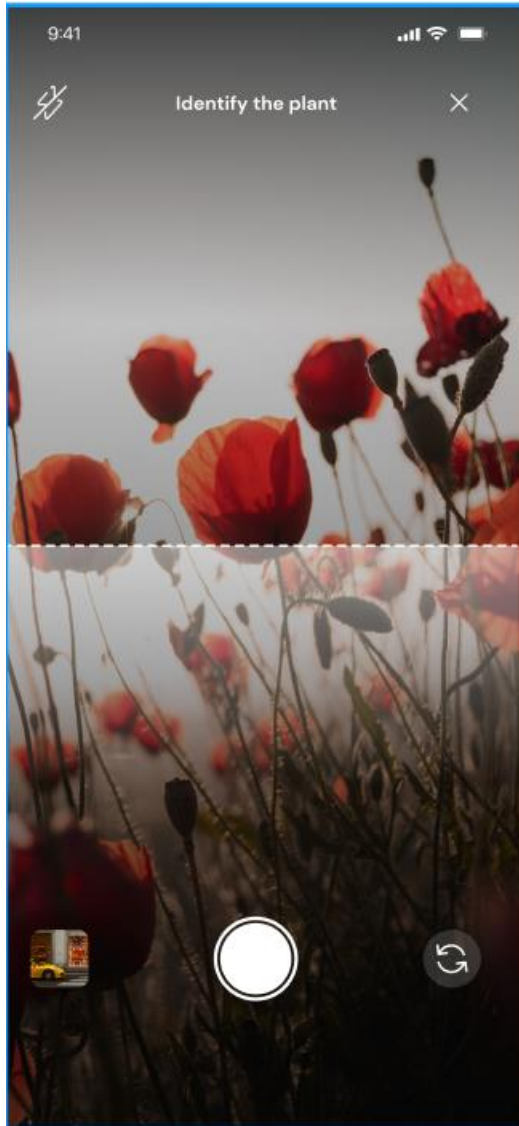


Figure 35 Product Scan page

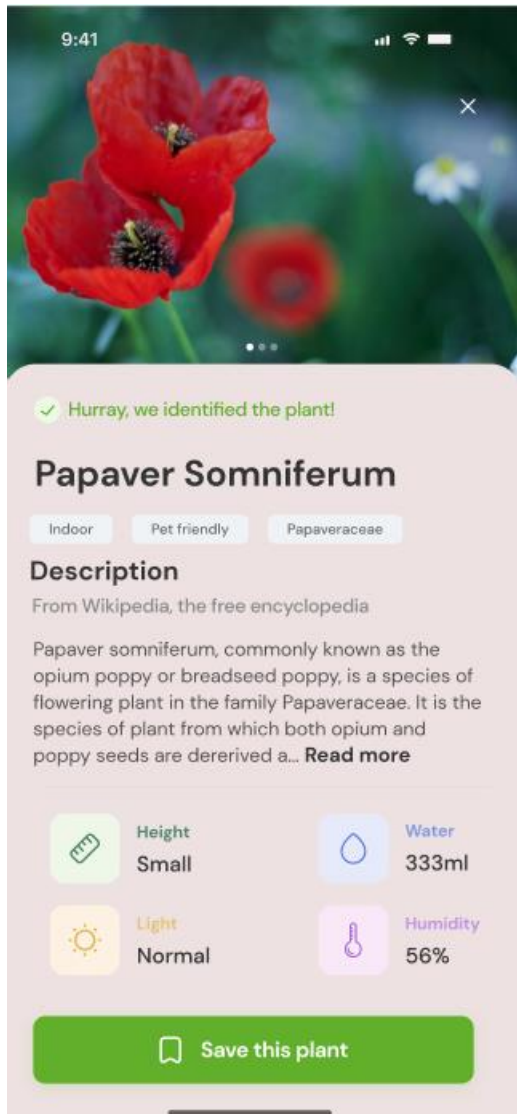


Figure 36 After Scanning

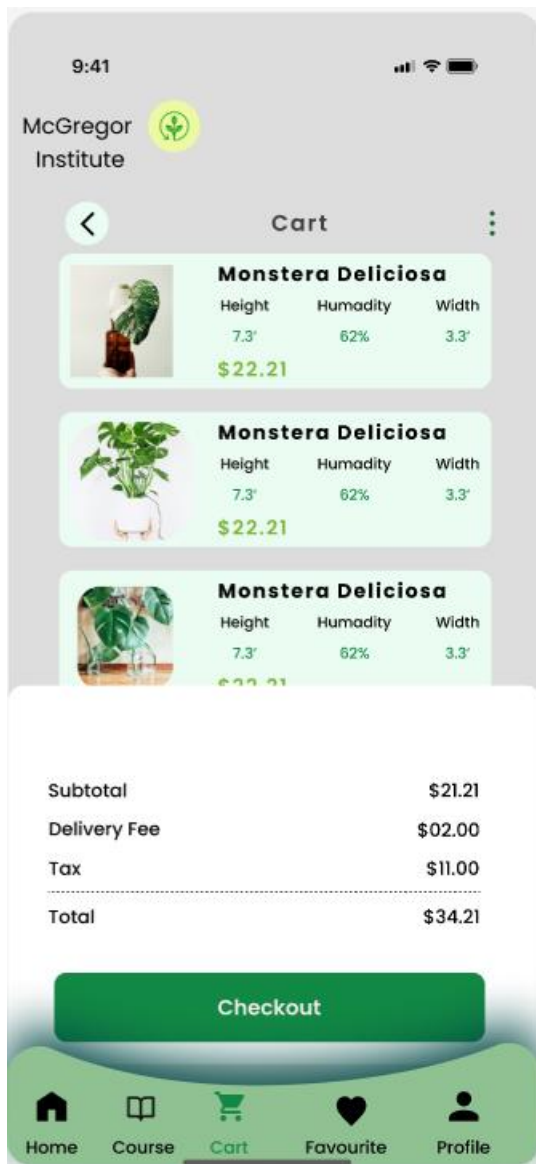


Figure 37 My Cart

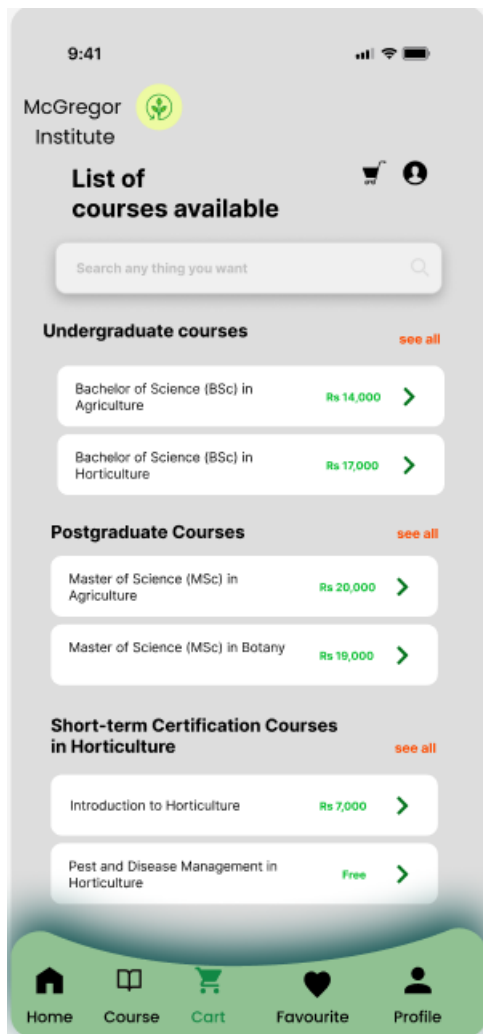


Figure 38 Course page



Figure 39 Course details page

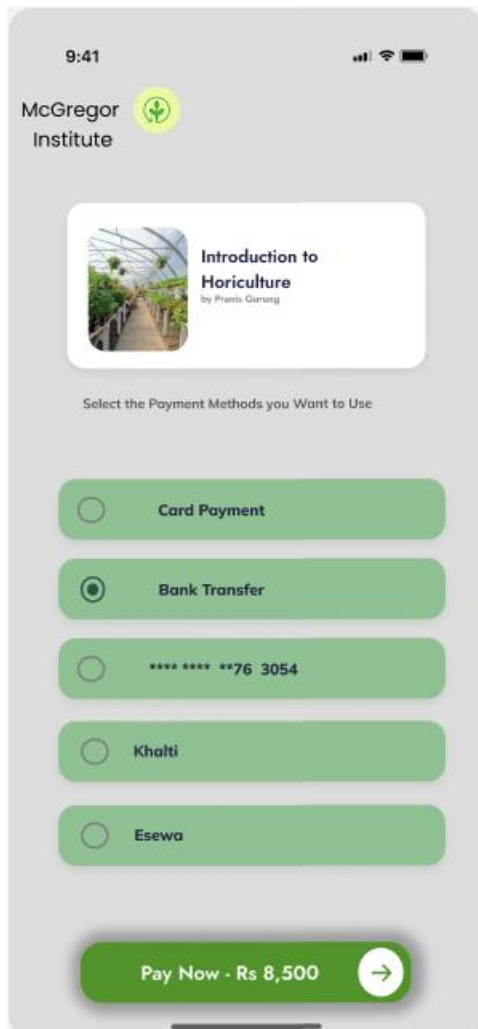


Figure 40 Course Payment page



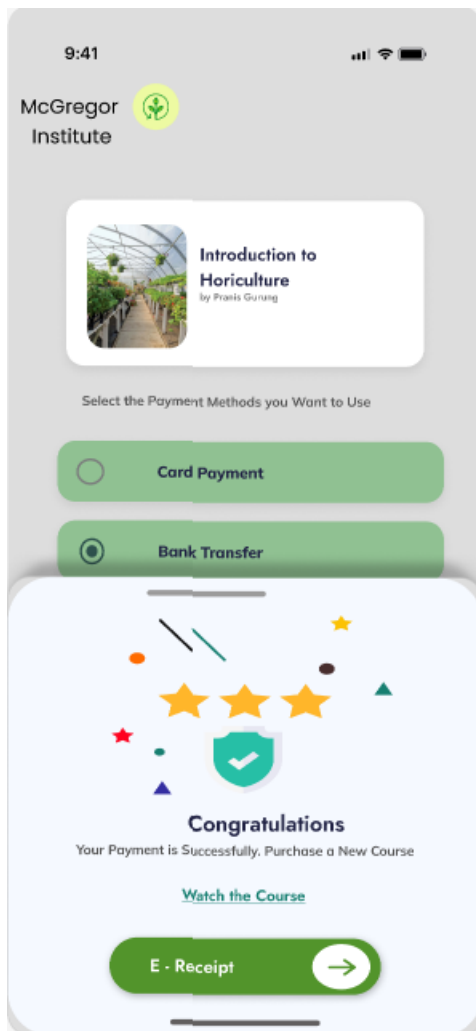


Figure 41 After Payment page



Figure 42 Completed Course page

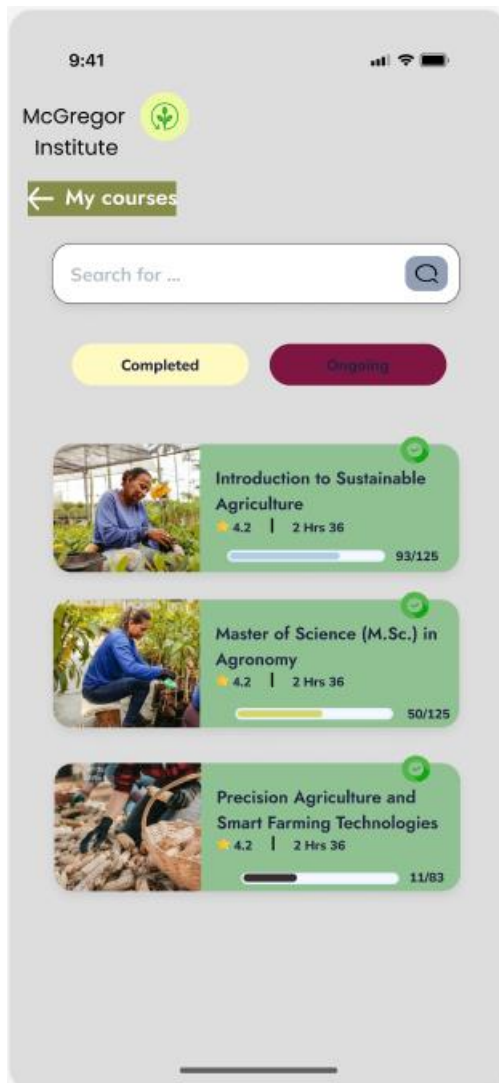


Figure 43 Ongoing Course page



Figure 44 Certificate of Course completion page

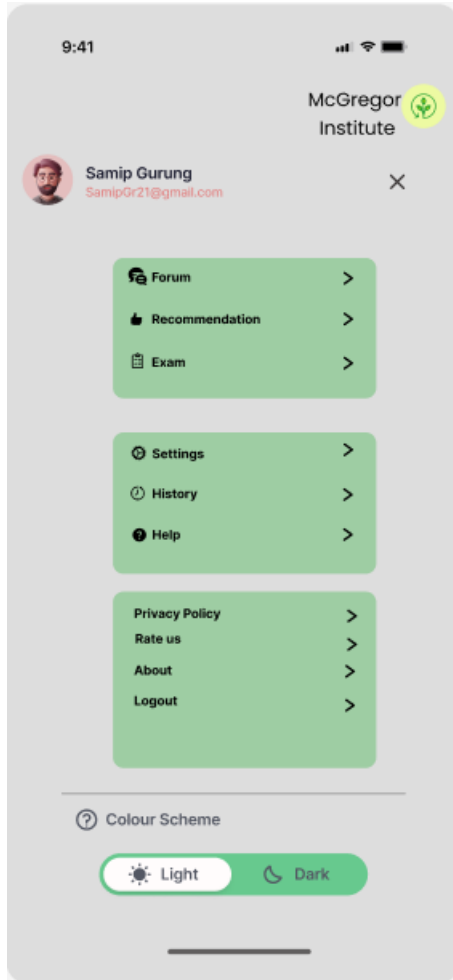


Figure 45 Drawer page

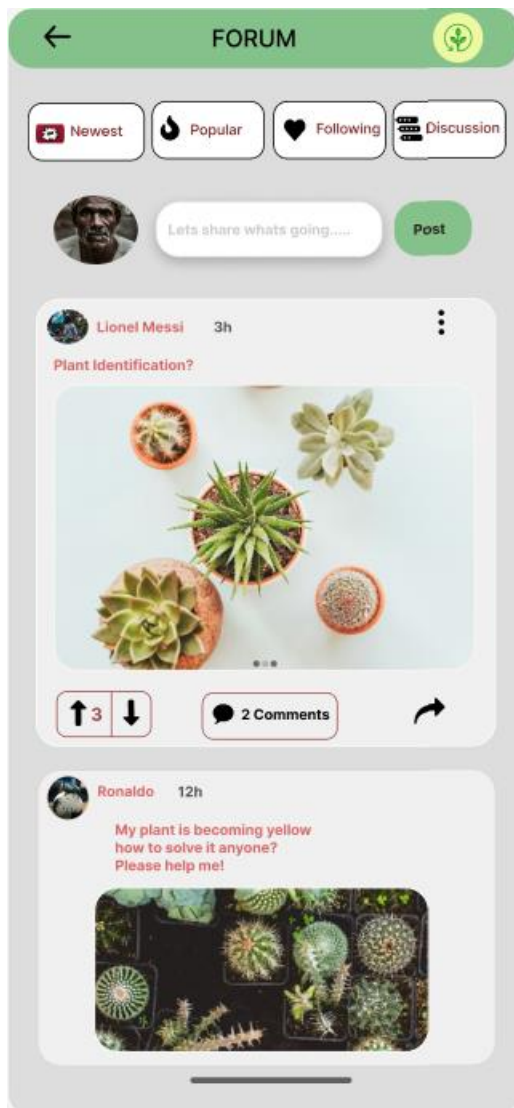


Figure 46 Forum Page

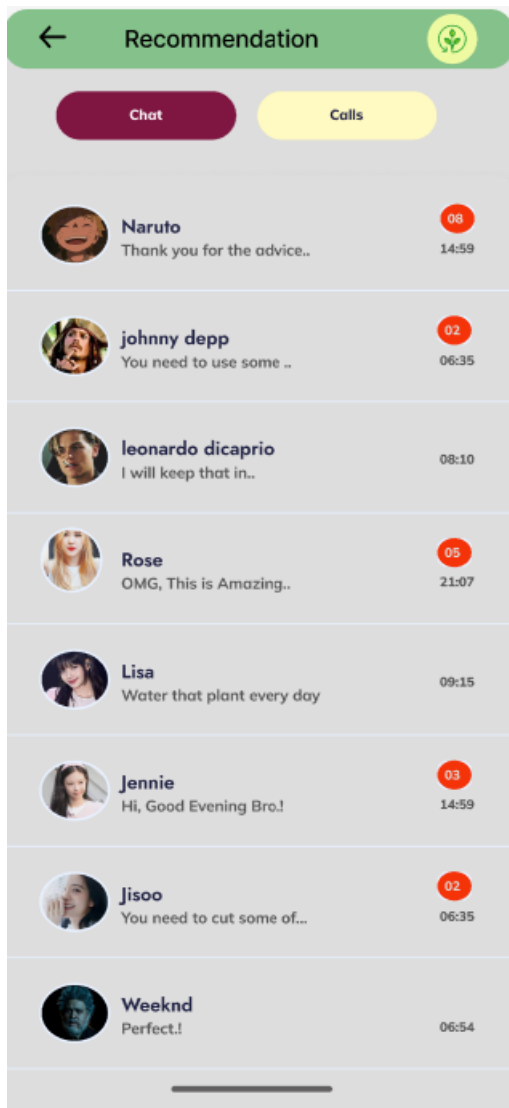


Figure 47 Recommendation of Chat page



Figure 48 Recommendation message page



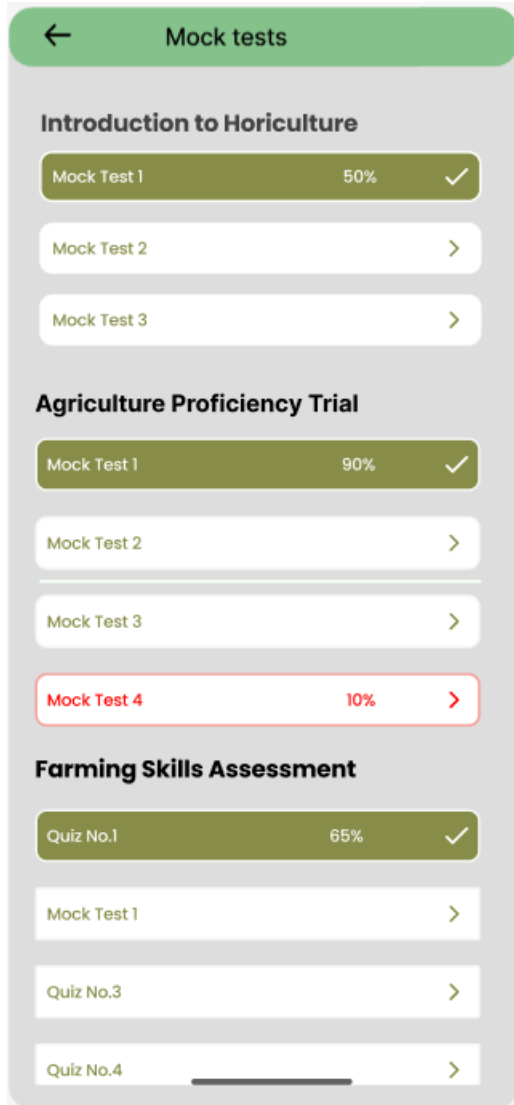


Figure 49 Mock Test page

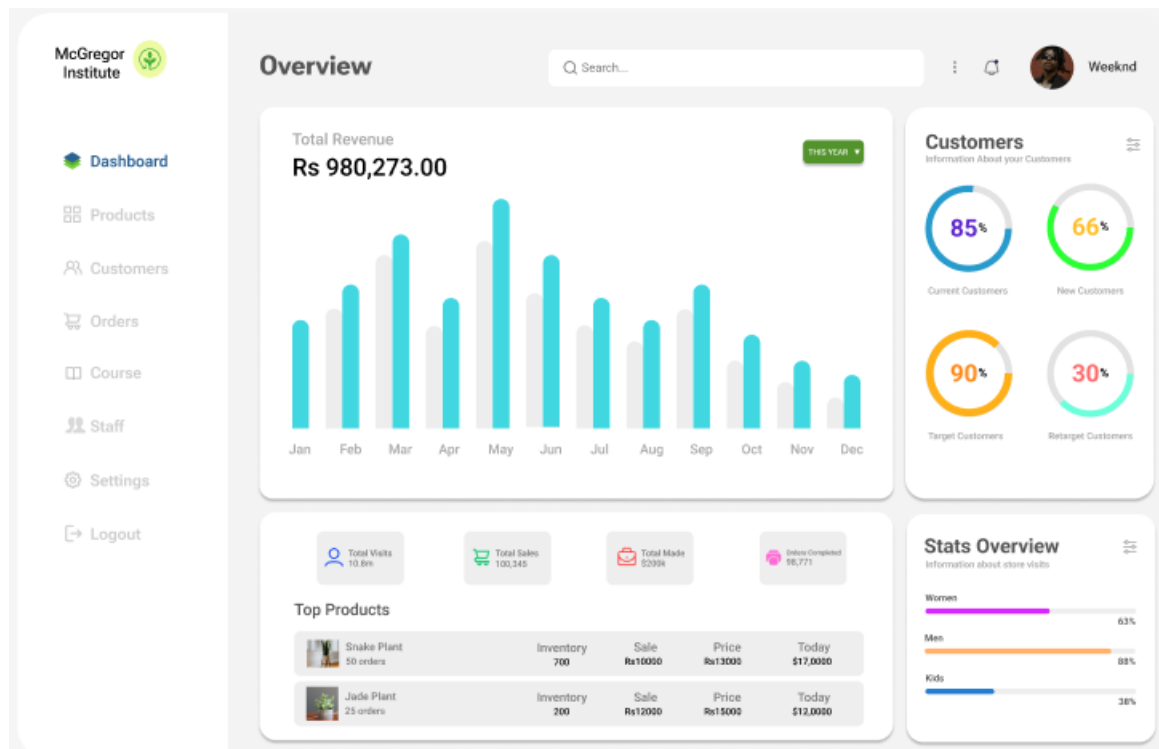


Figure 50 Admin Dashboard

## 12. Conclusion

This coursework focused on developing software for the McGregor Institute of Botanical Training, which recently expanded into the transportation sector. The software's purpose is to offer the short-term certification course related to horticulture for anyone interested. Apart from the certification courses, they are looking forward to selling different varieties of plants charging a minimal fee and free in some cases. Following the instructions, I opted for the waterfall methodology and utilized Unified Modelling Language (UML) for the project.

To organize the project tasks, I created a Gantt chart based on the waterfall methodology. This chart helped schedule the development work for the system effectively. Then, I proceeded to develop a use case model, which included a comprehensive use case diagram illustrating the entire system's functionalities. Additionally, I provided high-level descriptions for all the identified use cases and detailed descriptions for two specific ones. From these detailed descriptions, I crafted collaboration and sequence diagrams for one of the use cases. Creating these diagrams was challenging as I aimed to ensure they accurately depicted the system requirements while remaining easy to understand. To address this challenge, I conducted thorough research and sought advice from experts in the field to create comprehensive and understandable diagrams.

Next, I tackled the creation of a class diagram, which proved to be the most challenging aspect of the project. Analysing the use cases to identify domain classes and their properties, attributes, and methods required careful consideration. Additionally, determining the appropriate relationships between classes, such as inheritance, aggregation, and association, was somewhat confusing. To clarify my understanding, I extensively consulted various resources, including websites, lecture materials, videos, books, and journals. Moreover, I sought assistance from my module leader, seniors, and classmates, which greatly helped me overcome obstacles and make informed decisions.

Finally, I developed a prototype of the system using the 'Figma' tool, which was a completely new experience for me. This prototype served as a preliminary version of the software, allowing for visual exploration and feedback before proceeding with full-scale development.

Overall, completing this coursework presented both challenges and rewards. It provided valuable opportunities to develop a wide range of skills, including problem-solving, communication, research, and technical abilities. Each aspect of the project contributed to my personal growth and equipped me with knowledge and skills that will undoubtedly benefit me in future endeavours.

### 13. References

kirvan, p., 2023. *SearchERP*. [Online]

Available at: <https://www.techtarget.com/searcherp/definition/prototype>

[Accessed 2 May 2024].

Nicholas, A., 2023. *xaltius.tech*. [Online]

Available at: <https://xaltius.tech/an-introduction-to-software-engineering/>

[Accessed 23 April 2024].

Walker, A., 2024. *guru99.com*. [Online]

Available at: <https://www.guru99.com/use-case-diagrams-example.html>

[Accessed 26 April 2024].

Yasar, K., 2023. *www.techtarget.com*. [Online]

Available at: <https://www.techtarget.com/whatis/definition/software-engineering>

[Accessed 23 April 2024].