LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CS5002NI Software Engineering

Coursework Weightage and Type

35% Individual Coursework

AY 2023-2024

Student Name: Dikshya Sharma

London Met ID: 22067520

Assignment Submission Date: 7th March 2024

Submitted To: Mr. Rubin Thapa

I confirm that I understand my coursework needs to be submitted online via Google Classroom 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 mark of zero will be awarded.

Acknowledgement

I would like to take this opportunity to express my sincere gratitude and appreciation to everyone who helped in completion of this coursework. First and foremost, I am so grateful for my module leader Mr. Rubin Thapa, my tutor Mr. Dipesh Raj Adhikari, and all my respected teachers, for their continuous guidance and all helpful advice and assistance they gave while working on the coursework. Their expertise and feedback and constant encouragement were immensely beneficial.

In order for me to complete this coursework on time, my lecturers and tutors were of the greatest possible help. Whether it was in person or online through google hangouts, they were always willing to assist. Furthermore, I am thankful to the university and all the members of Islington College for assigning this coursework.

I can undoubtedly say that the abilities and knowledge that I learned from this coursework will be beneficial to me in the days to come. Finally, I extend my gratitude to all my friends for helping me overcome the obstacles with their skills. I greatly appreciate it. Thank you all so much.

Abstract

In this segment of the CS5002NI Software Engineering module, which constitutes 35% of the final grade, we delve into various diagrams, design tools, and methodologies crucial for building a project prototype. Unlike previous coursework, which emphasized procedural and functional programming, this segment shifts focus on object-oriented programming (OOP). We explore OOP concepts alongside different phases of development, including the Software Development Life Cycle (SDLC) and Yourdon methodologies.

Our approach entails a detailed roadmap for developing the proposed system. It commences with detailed planning using tools such as the Work Breakdown Structure (WBS) and Gantt Chart to organize tasks efficiently. Subsequently, we move to the modeling phase, creating Use Case and Collaboration diagrams as blueprints for the final system. The design phase follows, where we refine architecture, select methodologies, and implement design patterns to ensure a robust system. The objective is to craft a system that effectively meets our goals within the context of SDLC and Yourdon methodologies.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This coursework is the second part of the Software Engineering (CS5002NI) module, carrying a weightage of 35% towards the final grade. It focuses on learning different diagrams, design tools and methodologies ultimately leading to the development of a project prototype. Unlike the previous coursework, which focused on the procedural and functional programming, this coursework shifts its attention to the object-oriented programming (OOP) aspect. The components of OOP programming and different phases of development are discussed.

In this document, roadmap for developing a proposed system is outlined which involves careful planning and design to ensure success. It starts with precise planning which involves WBS, Gantt Chart and other essential tools to organize the development task effectively which is based on chosen methodology. Then, there comes a modelling phase where Use case diagrams, Collaboration diagrams and Collaboration Diagrams are created which serves as blueprint for the final system. And the design phase focuses on refining architecture and features by also selecting right methodology and implementing design patterns which contributes to developing a robust system.

In our project journey, Figma, Draw.io, Team Gantt and Microsoft Word emerged as invaluable allies. Figma facilitated seamless prototyping, Draw.io aided in visualizing system architecture and diagrams, while MS Word streamlined report authoring and formatting. Similarly, Team Gantt to create detailed Gantt chart, allowing us to visualize project timelines, allocate resources efficiently, and track progress effectively. These tools played pivotal roles in enhancing comprehension and efficiency throughout the development process.

1.1. Introduction to Business

The McGregor Institute of Botanical Training which is based in Ireland with a branch in Godawari, Lalitpur, Nepal has been in operation for about seven years. It is a training-based institute which is associated with Dublin City University that offers courses in horticulture and agriculture. In response to the increased interest in agriculture, it recently announced intentions to launch short-term certifications programs in horticulture. Additionally, they aim to offer a variety of plants for sale and foster a community of plant enthusiasts through a dedicated online platform. This platform will serve as a forum for discussions, program organization, and expert recommendations.

1.2. Aims and Objectives

The aim of this coursework is to utilize Object-Oriented Programming (OOP) concepts to establish a robust foundation through planning, analysis, and design ensuring efficient development and implementation of the proposed system.

The objectives of the system are as follows:

- To utilize the OOP concepts and explore various software development methodologies.
- To create a Gantt chart to illustrate project workflow and timeline.
- To construct Use Case Models with diagrams and detailed descriptions that acts as a blueprint for the final system.
- To create class diagrams for better visualization of system components, classes, objects, and entities.
- To develop a prototype that incorporates all the required features.

1.3. Business Cases

The Business Cases for the “McGregor Institute of Botanical training System” are listed below:

- To use any system services, a user must first register and become a member of the system.
- Students are free to choose from a wide range of courses, including undergraduate, postgraduate, graduate, and short-term certification programs.
- Students can choose between paid and unpaid courses depending on their preference.
- Customers can freely browse plants, make selections, and can purchase if they wish and complete transactions according to their preferences.
- The admin should get access to all institute details, including students, customers, staff, experts, and plants for generating a report accordingly.
- Any users can actively engage in forums for discussions and conversations on diverse topics.
- Users can express their opinions, research findings and discoveries through forum posts and can interact on forums by liking and commenting on each other’s posts.
- Customers have the facility to provide feedback, and comments on plants along with their prices and other system services.
- Any available botanical experts can provide recommendations to users on crops, plants, and soil types based on user location.

2. Methodology

The software development methodology refers to the development SDLC (software development life cycle) model which are structured approaches to a software development project and combine well-defined steps with a design or process philosophy. It generally takes the form of defined phases, created to describe the “how” and the life cycle of a piece of software. Software development involves a variety of methodologies designed to support the software at every stage of its life cycle. These models are chosen depending on the project’s, its developers’ needs and preferences. For example, Agile, Scrum, Waterfall , FDD and more (Lombardi, 2022).

In this coursework, the project employs Feature Driven Development. Feature Driven Development (FDD) is an agile framework that focuses on features which are small pieces of code. It organizes its development tasks around the primary features of the software which includes members such as Project Manager, Chief Architect, Development Manager, Chief Programmer, Class owner and domain experts who are responsible for overseeing the entire project, designing the software system watching development team activities, assisting the system design, coding and testing and checking the requirements of the project respectively (Lombardi, 2022).

For the purpose of creating feature sets quickly, FDD follows these five steps:

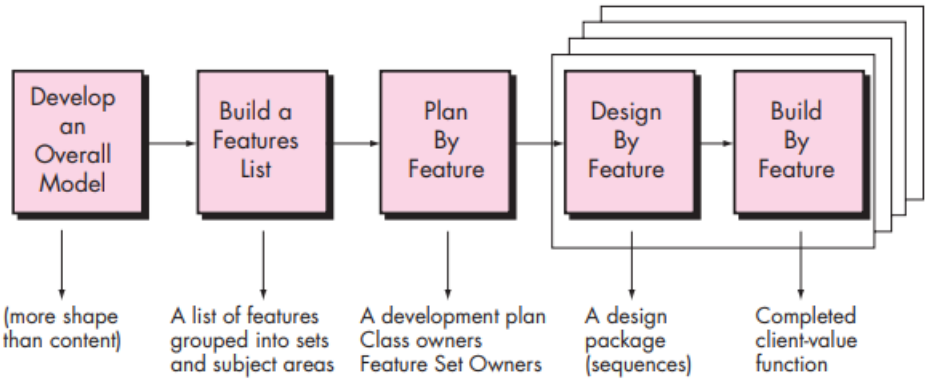


Figure 1: FDD (Pressman, 2010)

The steps are explained below:

a. **Develop an overall model.** The problem or need are defined by developers that they want the application to resolve.
b. **Build a features list.** Then, based on the needs of the customer, the developers determine which features are required for the application to function properly.
c. **Plan by feature.** The sequence in which features are developed is planned by the developers. Additionally, they attempt to predict risks and obstacles to development.
d. Design by feature. The chief programmer assigns roles and sets the features' priorities.
e. Build by feature. After a feature is built and tested, authorized versions are added to the finished product by the developers (Lombardi, 2022).

3. Work Breakdown Structure(WBS)

A work breakdown structure (WBS) is a project management tool that uses a step-by-step process to accomplish huge projects with numerous moving parts. A work breakdown structure (WBS) can combine scope, cost, and deliverables into a single tool by dividing the project into smaller components (Christine Organ, 2022).

Work breakdown structures (WBS) are essential in project management because they provide a systematic approach to breaking down project work and attaining specific objectives. To begin, they identify significant project tasks and define the project's critical path, which is essential for scheduling and prioritization. Second, the work breakdown structure (WBS) supports in the scheduling of activities to generate the performance measurement baseline (PMB), which guides project monitoring and assessment. Furthermore, the WBS outlines project deliverables, ensuring that they are in line with the project's aims (Coursera, 2024).

For our project, we have implemented a feature-driven methodology for breaking down works. This approach entails identifying project features and breaking them down into smaller tasks. Each feature is analysed to determine its scope, cost, and deliverables, which are then integrated into the overall project WBS.

Below, Work Breakdown Structure is outlined:

3.1 WBS of the System

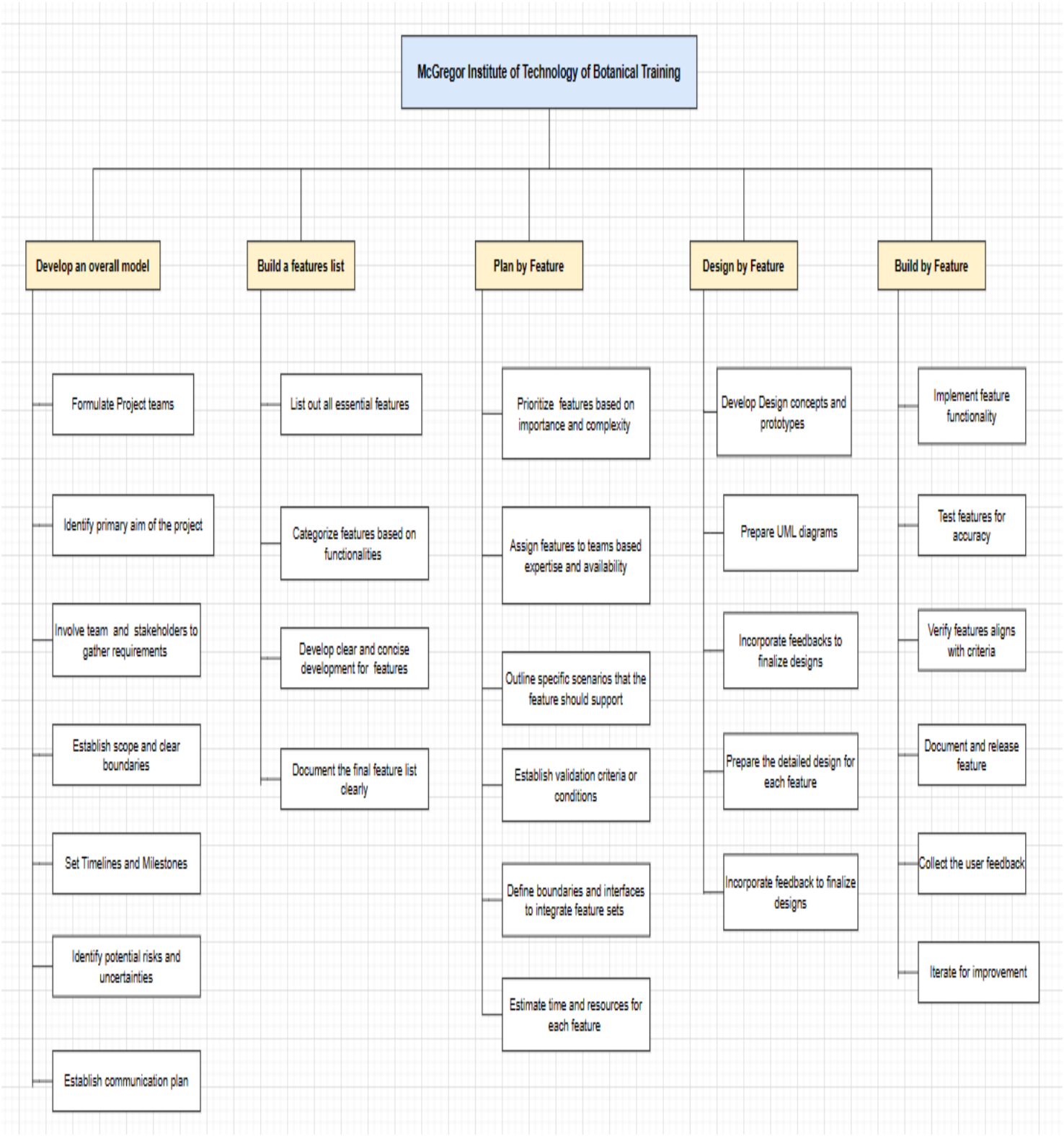


Figure 2: WBS of the system.

The main deliverable of the project is to develop an overall model of the website. This involves several sub-deliverables: Formulating project teams, identifying the project's primary aim and scope, establishing timelines and milestones, gathering requirements from team members and stakeholders, and creating an overall model including a sitemap and wireframes. Additionally, the project includes building a features list, planning, designing, and building features by priority, and finally, documenting and releasing the final website. This Work Breakdown Structure (WBS) provides a systematic approach to ensure the project progresses smoothly, with all aspects considered and completed within the specified timeframe and budget.

4. Gantt Chart

A Gantt chart is a horizontal bar chart created in 1917 as a production control tool by American engineer and social scientist Henry L. Gantt, which is widely used in project management to plan, coordinate, and monitor tasks in a project. They offer a graphical representation of a timetable that can be used. There are three essential elements that comprise a Gantt chart: tasks or activities that must be finished, displayed along the y-axis; progress stages or milestones, displayed along the x-axis (either at the top or bottom of the chart); and progress bars, which display the current progress of each task as horizontal bars (Lutkevich, 2021).

As it's a bar chart format, it's possible to check progress with a quick glance and can easily display a visual demonstration of the whole project, timelines and deadlines of all tasks, relationships, and dependencies between the various activities project phases. The assignment of tasks, resource allocation, remote work collaboration, and planning and scheduling can all be enhanced by using a Gantt chart. The Gantt chart has following components:

Task List: A vertical list of all tasks required to complete the project. Each task should be clearly defined and have a specific deadline.

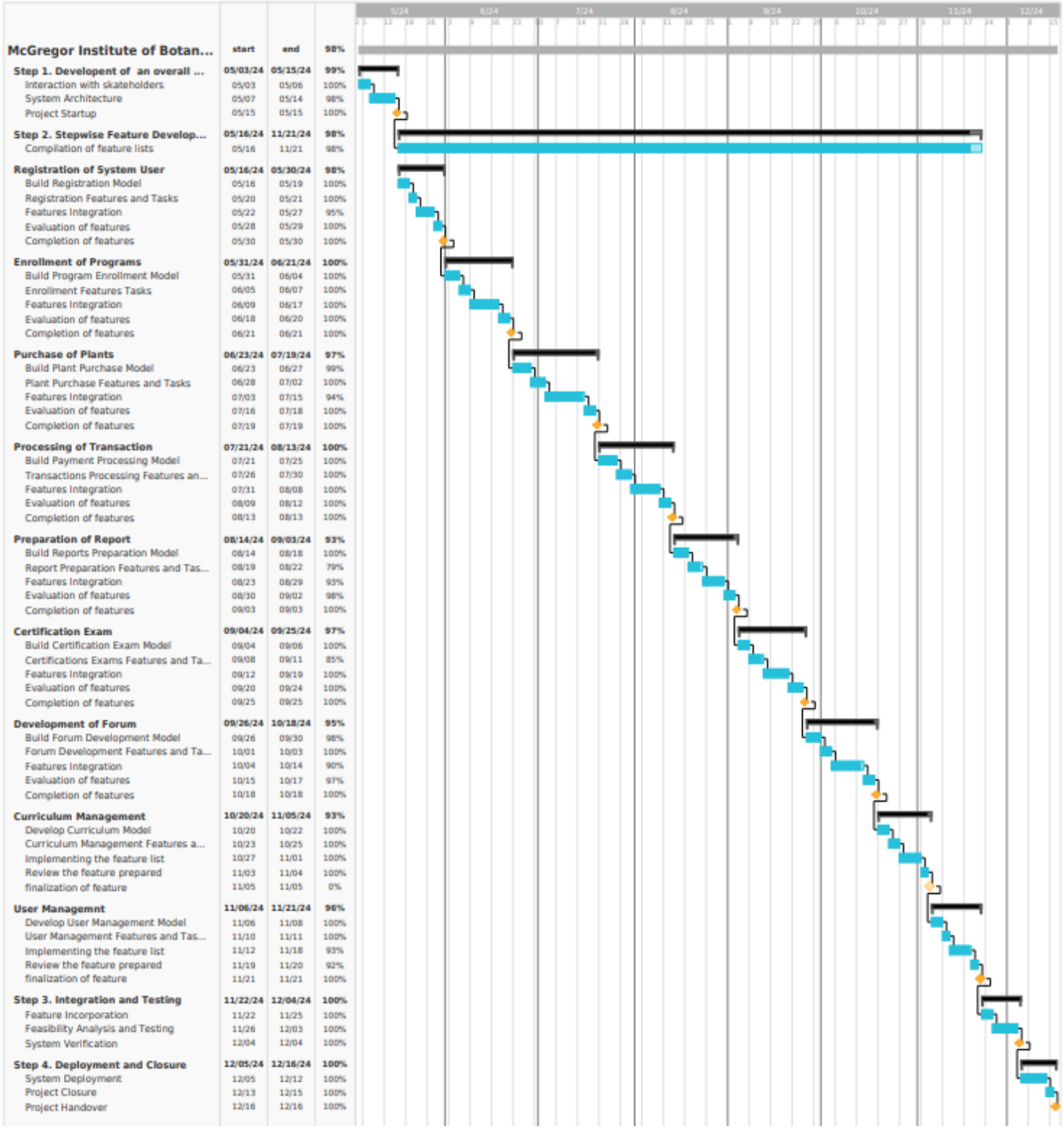
Timeline: A horizontal axis that represents the duration of the project. Depending on the project's duration, it is typically divided into days, weeks, or months.

Bars: A horizontal line that represents the duration of each task where task's start date is indicated by the left end of the bar, while the end date is indicated by the right end of the bar. The length of the bar represents the duration of the task.

Dependencies: Dependencies are shown using arrows that link the bars representing the dependent tasks. Arrows indicate the relationship between tasks.

Milestones: Milestones are significant events in the project, such as completing a major task or delivering a critical component. Diamonds represent them in the Gantt chart and are usually labelled with a specific date (Abdullahi, 2023).

4.1. Gant Chart of the System



McGregor Institute of Botan...	start	end	98%
Step 1. Developent of an overall ...	05/03/24	05/15/24	99%
Interaction with skateholders	05/03	05/06	100%
System Architecture	05/07	05/14	98%
Project Startup	05/15	05/15	100%
Step 2. Stepwise Feature Develop...	05/16/24	11/21/24	98%
Compilation of feature lists	05/16	11/21	98%
Registration of System User	05/16/24	05/30/24	98%
Build Registration Model	05/16	05/19	100%
Registration Features and Tasks	05/20	05/21	100%
Features Integration	05/22	05/27	95%
Evaluation of features	05/28	05/29	100%
Completion of features	05/30	05/30	100%
Enrollment of Programs	05/31/24	06/21/24	100%
Build Program Enrollment Model	05/31	06/04	100%
Enrollment Features Tasks	06/05	06/07	100%
Features Integration	06/09	06/17	100%
Evaluation of features	06/18	06/20	100%
Completion of features	06/21	06/21	100%
Purchase of Plants	06/23/24	07/19/24	97%
Build Plant Purchase Model	06/23	06/27	99%
Plant Purchase Features and Tasks	06/28	07/02	100%
Features Integration	07/03	07/15	94%
Evaluation of features	07/16	07/18	100%
Completion of features	07/19	07/19	100%
Processing of Transaction	07/21/24	08/13/24	100%
Build Payment Processing Model	07/21	07/25	100%
Transactions Processing Features an...	07/26	07/30	100%
Features Integration	07/31	08/08	100%
Evaluation of features	08/09	08/12	100%
Completion of features	08/13	08/13	100%
Preparation of Report	08/14/24	09/03/24	93%
Build Reports Preparation Model	08/14	08/18	100%
Report Preparation Features and Tas...	08/19	08/22	79%
Features Integration	08/23	08/29	93%
Evaluation of features	08/30	09/02	98%
Completion of features	09/03	09/03	100%
Certification Exam	09/04/24	09/25/24	97%
Build Certification Exam Model	09/04	09/06	100%
Certifications Exams Features and Ta...	09/08	09/11	85%
Features Integration	09/12	09/19	100%
Evaluation of features	09/20	09/24	100%
Completion of features	09/25	09/25	100%
Development of Forum	09/26/24	10/18/24	95%
Build Forum Development Model	09/26	09/30	98%
Forum Development Features and Ta...	10/01	10/03	100%
Features Integration	10/04	10/14	90%
Evaluation of features	10/15	10/17	97%
Completion of features	10/18	10/18	100%
Curriculum Management	10/20/24	11/05/24	93%
Develop Curriculum Model	10/20	10/22	100%
Curriculum Management Features a...	10/23	10/25	100%
Implementing the feature list	10/27	11/01	100%
Review the feature prepared	11/03	11/04	100%
finalization of feature	11/05	11/05	0%
User Managemnt	11/06/24	11/21/24	96%
Develop User Management Model	11/06	11/08	100%
User Management Features and Tas...	11/10	11/11	100%
Implementing the feature list	11/12	11/18	93%
Review the feature prepared	11/19	11/20	92%
finalization of feature	11/21	11/21	100%
Step 3. Integration and Testing	11/22/24	12/04/24	100%
Feature Incorporation	11/22	11/25	100%
Feasibility Analysis and Testing	11/26	12/03	100%
System Verification	12/04	12/04	100%
Step 4. Deployment and Closure	12/05/24	12/16/24	100%
System Deployment	12/05	12/12	100%
Project Closure	12/13	12/15	100%
Project Handover	12/16	12/16	100%

Figure 3: Gantt Chart of the System.

The above Gantt chart outline a detailed roadmap for the McGregor Institute of Botany's project development, covering across various phases and timeframes. Initiating from May 3 to December 16, each stage of the project is carefully scheduled to ensure timely completion. The initial phase focuses on crafting an overall project plan, setting the foundation for ensuing steps. From May 16 to November 21, stepwise feature development unfolds, with tasks like user registration, program enrolment, and plant purchase being thoroughly developed and integrated.

Following this, integration and testing occur from November 22 to December 4, ensuring the seamless incorporation and functionality of all features. Finally, deployment and closure take place from December 5 to December 16, marking the project's culmination and handover. Despite minor delays in certain feature developments, the project remains on track, poised to deliver a robust solution for the McGregor Institute of Botany within the specified timeframe.

5. Unified Modelling Language (UML)

A Unified Modelling Language (UML) is a visual modelling language that combines visualization with standardization to result in higher quality, better compliance, and enhanced productivity. The UML diagrams are made from elements also known as object-oriented concepts which plays a crucial role in defining the structure, behaviour, and relationships within a software system. UML is further divided into structural and behavioural diagram. Structural diagram includes Class Diagram and behavioural diagrams include use case, sequence, and collaboration diagram (Indeed Editorial Team, 2023).

5.1. Use-Case Model

A use case model is a visual representation of the interactions between an actor and a system which is expressed using UML. It is used to visualize the flow and behaviour of the system, illustrate the functionality of the system and represent key system-user interactions (Daly, 2022).

5.2. Use-Case Diagram

A use case is a detailed description of a user's interaction with a product or system which can outline the failure scenarios, and any critical variations or exceptions. A use case is designed to reveal system demands early on in the process. It’s a powerful modelling technique offering numerous advantage which includes capturing requirements, planning and validation, Flexible development, and clear communication. As it helps in finding minor requirements early on in the project saves a ton of time by identifying exceptions to a successful scenario. Each use case is designed specially to cover only one application of the system (Lad, 2022).

Use-case diagrams are made up of various components, which are as follows:

- **System**

The system is used to define the scope of the use case and drawn as a rectangle (Nishadha, 2022).

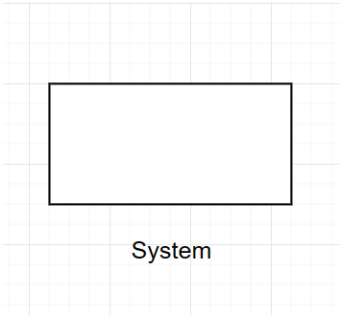


Figure 4: System(Use Case Diagram).

- **Actor**

  Any entity that plays a part in a certain system is an actor in a use case diagram. This can be an individual, a company, or an outside system, and it's typically depicted like the skeleton below (Nishadha, 2022).

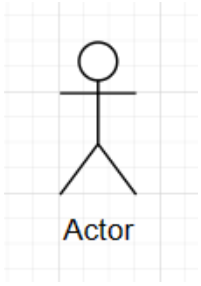


Figure 5: Actor(Use Case Diagram).

- **Use Case**

  A use case represents a function or an action within the system. It's drawn as an oval and named with the function (Nishadha, 2022).

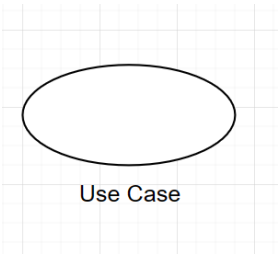


Figure 6: Use Case(Use Case Diagram),

- **Relationships**

It is the component of the use-case diagram that shows the connection between the actors and the use cases (Visual Paradigm, 2022). It is further divided into 4 types which are as:

- **Association**

It is the relationship that shows the connection between the use case and the actors. It is represented by a simple line or 2 double-headed lines (javatpoint, 2021).

Figure 7: Association(Use Case Diagram).

- **Generalization**

Generalizations mean the association between the actors in order to help re-use of common properties (javatpoint, 2021). It follows parent-child concept and depicts how a child class inherits from its parent class.

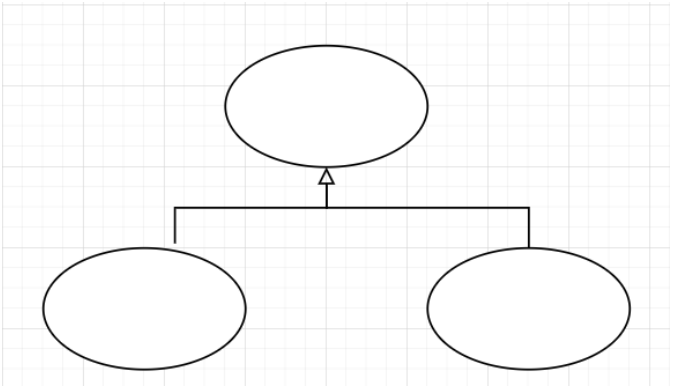


Figure 8:Generalization(Use Case Diagram).

- **Include**

The "include" relationship in a use case diagram indicates that one use case is always included in another use case (javatpoint, 2021).

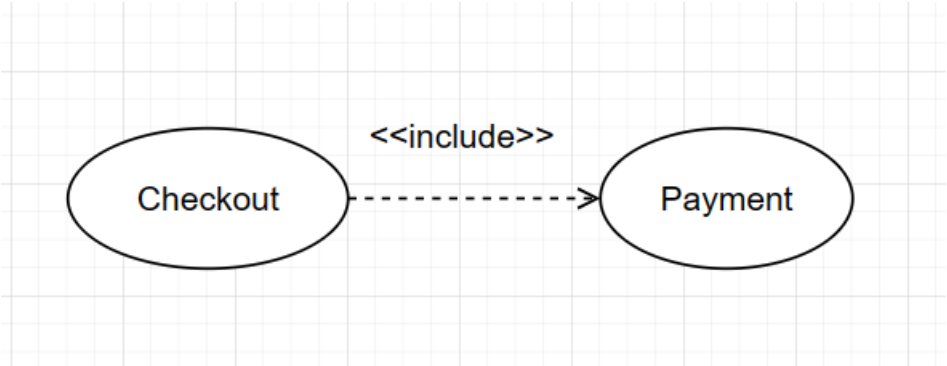


Figure 9: Include(Use Case Diagram).

- **Extend**

The "extend" relationship indicates that one use case may optionally extend the behaviour of another use case (javatpoint, 2021).

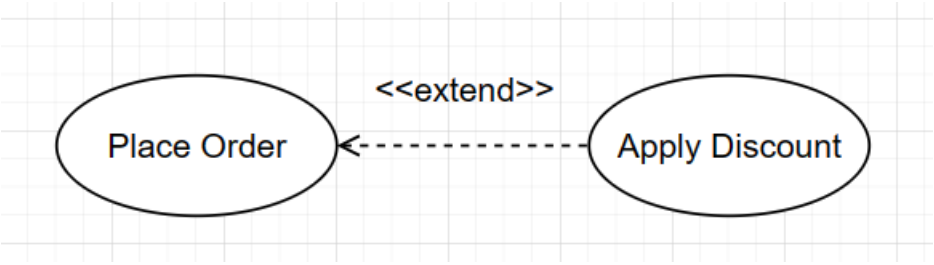


Figure 10: Extend(Use Case Diagram).

5.2.1. Use Case Diagram of the System

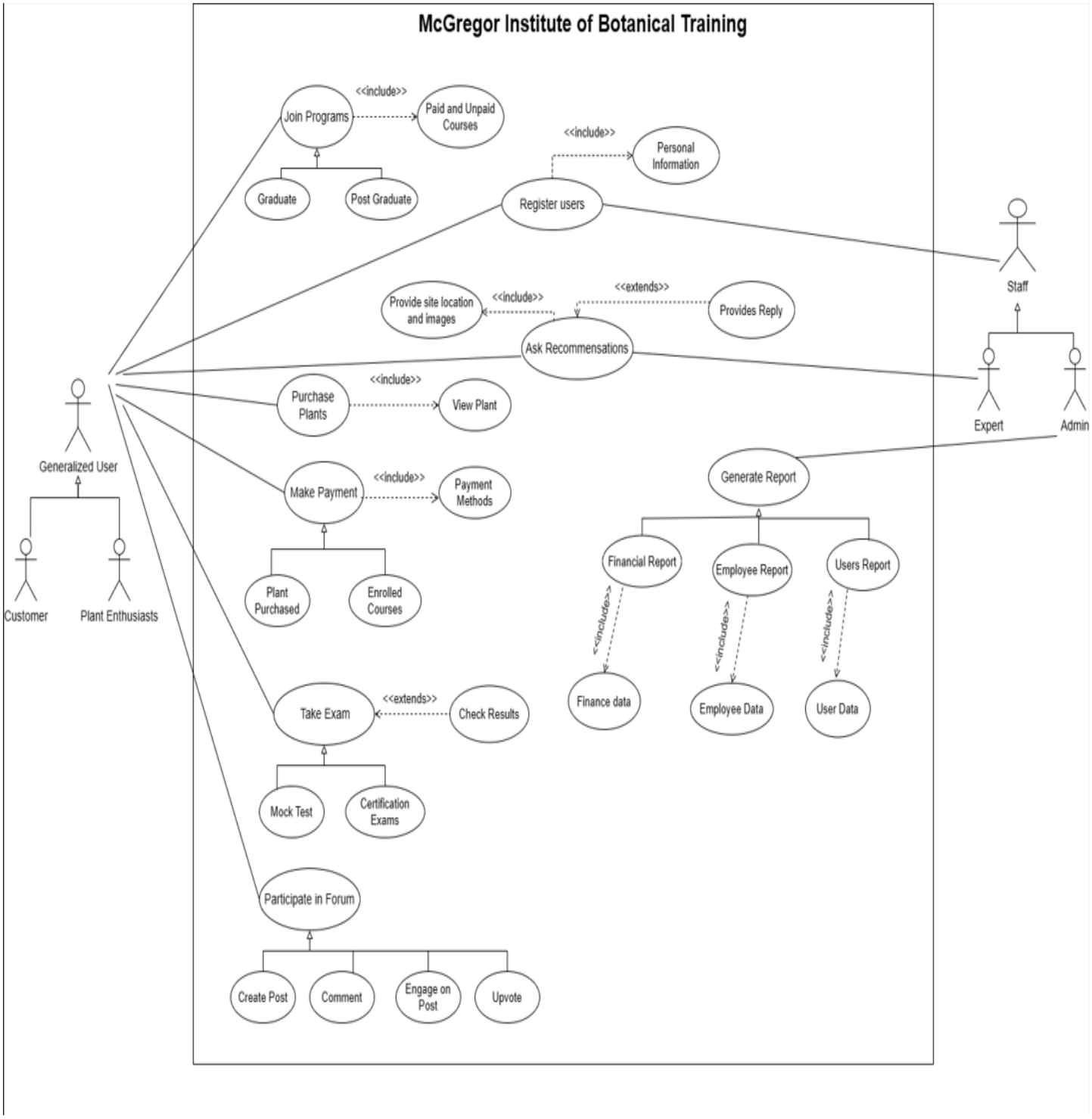


Figure 11: Use Case Diagram of the System.

5.3. High Level Description

The high-level Use Case description is simply a summary description of the task, which includes purpose of a use case and the actor involved. They are handy in early development stages when specific design decision hasn't been made yet (Carol Britton, 2005).

5.3.1. High Level Description of Use Cases

The high-level description of the use cases that are shown above in fig --: Use case diagram is briefly discussed below:

a. Register Users

Use Case:	**Register Users**
Actor:	Staff, Customers
Description:	Each New User enters their credentials to register to the system which is further verified by staff.

b. Join Programs

Use Case:	**Join Programs**
Actor:	Customers
Description:	Customers browses available program and selects the program of their preferences and then gets enrolled in the program.

c. Purchase Plants

Use Case:	**Purchase Plants**
Actor:	Customers
Description:	Customers navigate through a list of available plants, choose the desired plants and finalize the transaction to proceed for the payment.

d. Make Payment

Use Case:	**Make Payment**
Actor:	Customers
Description:	Customers provide payment details and complete transactions for various services offered by institute.

e. Take Exam

Use Case:	**Take Exam**
Actor:	Customers
Description:	Customers participate in mock tests or certifications exams upon meeting the necessary requirements.

f. Ask Recommendations

Use Case:	**Ask Recommendations**
Actor:	Customers, Expert
Description:	Customers seek advice from experts regarding suitable plants or crops based on location and soil conditions and expert provide recommendation in response.

g. Participate in Forum

Use Case:	**Participate in Forum**
Actor:	Customers
Description:	Customers engage in discussions, share opinions, and seek advice within the plant enthusiast community forum.

h. Generate Report

Use Case:	**Generate Report**
Actor:	Admin
Description:	Admins create detailed reports regarding finances, employee activities and user-related data.

5.4. Expanded Use Case Description

The expanded use case description are detailed steps involved in specific use case involved in the system. Expanded Use Case Description of two use cases from figure – are below:

5.4.1. Expanded Use Case Description of Register User

Use Case: Register User

Actor: Staff, Customers

Purpose: To register new member in the system.

Overview: The customer or new member interested in the company's service fills out the form with their name, phone number, address, gender, and other required details which is further verified by staff of the company.

Type: Essential

Action Steps

Actor Action	System Response
1. A new user or customer accesses the registration page of the company's application.	2. System presents the registration form to the new user.
3. A new user provides their personal information in the systems registration form**.**	4. System verifies accuracy the provided information.
5. A new user submits the filled-out registration form to the system.	6. Staff validates the provided credentials and securely stores the details.
	7. System asks for confirmation of registration to the new user.
8. A new user confirms registration.	9. System redirects new user to the login page.

Table 1:Expanded Use Case Description of Register User.

Alternative Course of Action

Line 3: A new user didn’t fill details according to the requirements of input fields. Use Case Ends.

Line 5: Invalid Information provided. Use Case Ends.

Line 8: A new user doesn’t confirm registration. Use Case Ends.

5.4.2. Expanded Use Case Description of Purchase Plants

Use Case: Purchase Plants

Actor: Customers

Purpose: To purchase plants in the system.

Overview: Customers who want to buy plants visit the company's application or online store. They browse a list of the plants that are available, which can contain different types, sizes, and cost ranges. Customers find the plants they like and proceed to make their final selection after choosing which ones to buy. This involves confirming the chosen plants and providing any extra information that may be required, including the quantity needed or preferred method of delivery.

Type: Essential

Action Steps

Actor Action	System Response
1. A user interested in purchasing plants accesses the plant purchasing section in the application.	2. System presents the list of available plants to the customer.
3. Customer browses and finds their desired plants.	
4. Customer adds their selected plants to the cart.	
5. Customer adds the quantity required.	6. System checks for the stock quantity and updates cart.
7. The customer confirms their selection and proceeds to checkout.	
8. The customer selects the delivery method and additional information needed to proceed for the transaction.	9. System sends confirmation message to the customer.
10. Customer confirms the order.	11. System redirects customers to the payment page.

Table 2:Expanded Use Case Description of Purchase Plant.

Alternative Course of Action

Line 3: Customer do not find their desired plants. Use Case Ends.

Line 6: Store lacks the quantity user want. Use Case Ends.

Line 10: Customer denies confirmation. Use Case Ends.

5.5. Communication Diagram

UML communication diagrams, which were originally called collaboration diagrams, illustrate the connections between objects and the messages that move across different applications. It provides a visual representation of the flow of communication between different elements, making it easier to identify any potential issues or areas of improvement.It visually demonstrates the interaction between different objects in a software system (Nalimov, 2021).

The collaboration diagram has four primary notations, which are as follows:

Objects: Objects are represented as rectangle shapes with labels inside, following along with the "object name: class name" convention. An object should be noted if it has a property or state that has significant impacts on the collaboration (Lewis, 2023).

Figure 12: Objects (Communication Diagram).

Actors: Actors are instances that state the interaction in the diagram. Each actor has a name and a role, with one actor initiating the entire use case (Lewis, 2023).

Figure 13: Actors (Communication Diagram).

Links: Links connect objects with actors and are represented using a solid line between two elements. Each link is an instance where messages can be sent (Lewis, 2023).

Figure 14: Links(Communication Diagram).

Messages: Messages are shown as a labelled arrow placed near a link. These messages are communications between objects that convey information about the activity and can include the sequence number (Lewis, 2023).

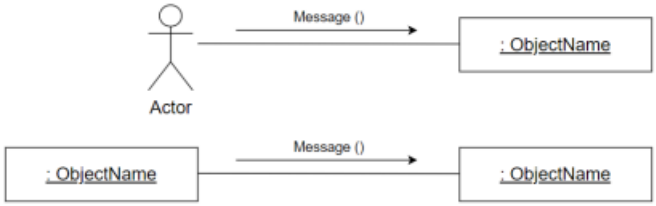


Figure 15:Messages(Communication Diagram).

5.5.1. Communication Diagram of System

To create the Collaboration/Communication Diagram for the **"Purchase Plants"** use case, a structured approach outlined is followed which is explained below:

a. Selecting Domain Classes:

Initially, we identify key nouns from the expanded use case description. For **"Purchase Plants**," potential domain classes include Participants, Plants Availability, Plants Cart, and Plant Purchase Transaction.

b. Creating Object List:

Next, a list of objects derived from the domain classes, as well as boundary and control objects are created. For example, objects like **"Plants Registry"** to store available plants, **"Plants Cart"** to contain selected plants, and **"Plant Purchase Transaction"** to record transactions. In addition, we offer a boundary object called **"Purchase Plants Interface"** for managing screen interactions and a control object called **"Plant Registry"** for overseeing plant data flow.

c. Adding Actors:

Actors like **"System Participant"** are created alongside boundary objects, indicating their involvement in system interactions.

d. Establishing Associations:

Lines are drawn between objects and actors to depict their associations and interactions, illustrating communication pathways within the system.

e. Incorporating Messages:

Messages are inserted to signify interaction between objects and actors, with each message numbered to depict the communication flow. Once all messages are incorporated, the collaboration diagram is finalized.

By following these steps, the Collaboration Diagram for the "**Purchase Plants**" use case is developed, facilitating a comprehensive representation of system interactions and communication channels. The Collaboration Diagram of the purchase plants representing the system is shown below:

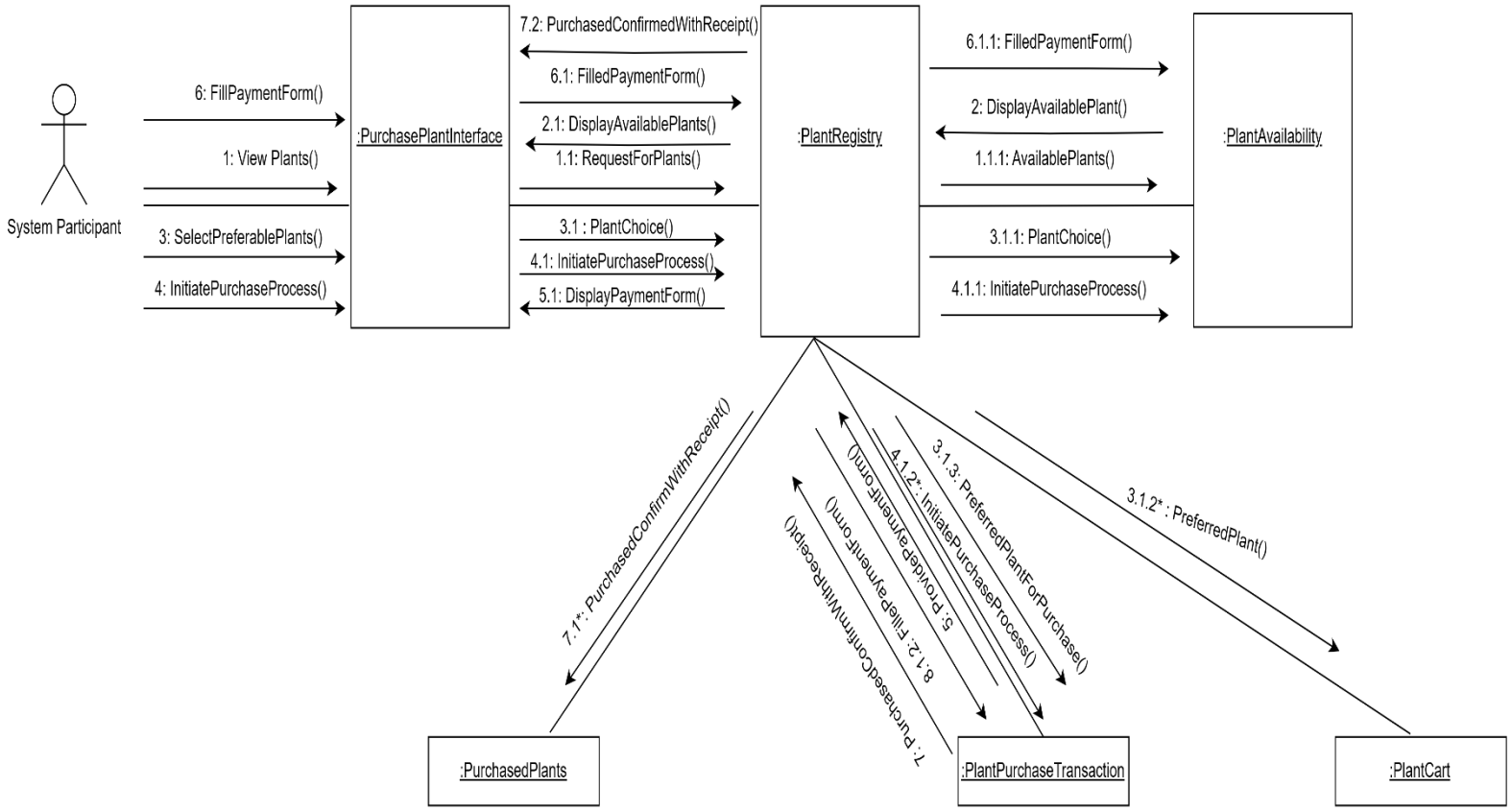


Figure 16: Communication Diagram of the system.

The collaboration diagram outlines interactions between objects within the system for the "**Purchase Plants**" use case. The System Participant, representing external entities like customers or authorized employees, initiates the process by viewing available plants and selecting preferred options. The system responds by displaying available plants and presenting a payment form. Once the System Participant fills out the payment form, the system validates the purchase process and may prompt for confirmation. After confirmation, the system files the completed payment form and sends a purchase confirmation to the System Participant. This concise depiction provides a clear understanding of the communication flow between the System Participant and the system during the plant purchase process.

5.6. Sequence Diagram

A sequence diagram is a type of UML diagram that shows the interactions between objects or components of a system. It can model simple sequential flow, branching, iteration, recursion, and concurrency where x-axis represents with objects, while y-axis represents the time (U, 2021).

a. **Objects:** Objects are instance of a class which is represented by a rectangular shape (U, 2021).

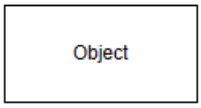


Figure 17: Objects(Sequence Diagram).

b. **Lifeline:** Lifeline is dotted line that extends down the vertical axis from the base of each object which indicates the life span of an object over a period of time (McNeish, 2022).

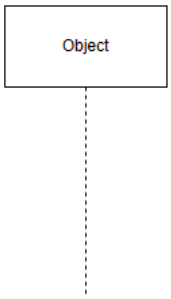


Figure 18: Lifeline(Sequence Diagram).

c. **Actor:** Actors in a sequential diagram is used to specify the role played by a user or system that interacts with the system's objects (Pandey, 2024).

Figure 19: Actor(Use Case Diagram).

d. **Activation:** The time that an item or component is actively processing a message is shown by these horizontal bars, which are frequently displayed on top of a lifeline. They display an object's state of activity and inactivity (VanZandt, 2023).

Figure 20: Activation(Sequence Diagram).

e. **Messages:** Messages are the horizontal arrows or lines that connect lifelines, indicating the order of messages passed between objects. They are represented as arrow from caller to receiver which can flow in any direction (VanZandt, 2023). It comes with a description called message signature: message_name(argument): return_type.

Messages are of different types:

- **Synchronous Message:** Synchronous Message is like sending a text and waiting for a reply before sending another (Pandey, 2024).

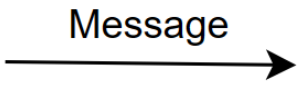


Figure 21: Synchronous Message(Sequence Diagram).

- **Asynchronous Message:** Asynchronous Message does not require a reply from the receiver before continuing (Pandey, 2024).

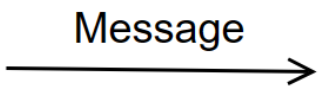


Figure 22: Asynchronous Message(Sequence Diagram).

- **Return Message:** Return Message represents the response returned by the receiving object (VanZandt, 2023).

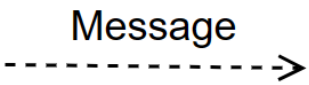


Figure 23: Return Message(Sequence Diagram).

- **Creation Message:** Creation Message are used to create a new object in the sequence diagram (Pandey, 2024).

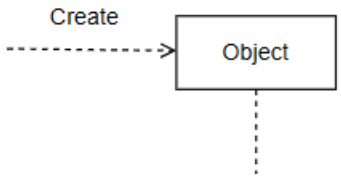


Figure 24: Creation Message(Sequence Diagram).

- **Destructive Message:** A destructive message is used to represent the termination of object (VanZandt, 2023).

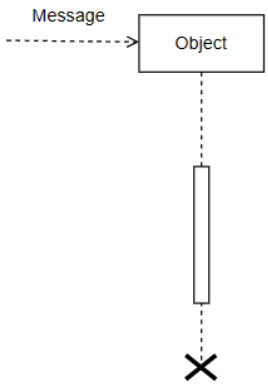


Figure 25: Destructive Message(Sequence Diagram).

- **Reflexive Message:** Reflexive Message or self-message are the messages sent from an object to itself (VanZandt, 2023).

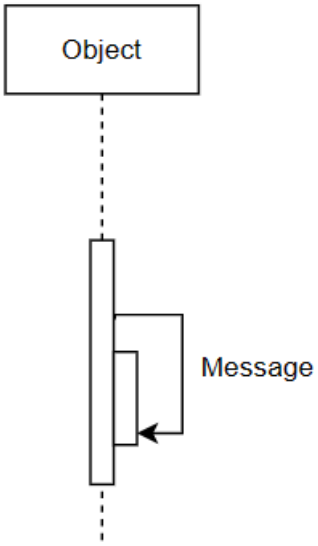


Figure 26: Reflexive Message(Sequence Diagram).

- **Note:** Note or a comment is represented as a rectangle with a folded-over corner which can be linked to the related object with a dashed line.

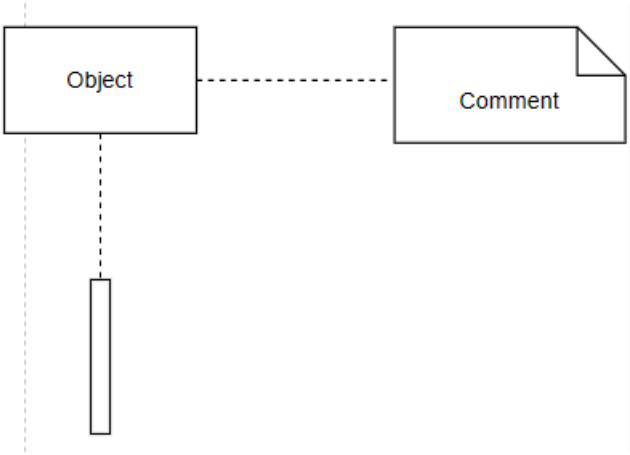


Figure 27: Note(Sequence Diagram).

- **Sequence Fragment:** A fragment in a sequence diagram is a rectangular frame drawn over part of the diagram that represents conditional structures that affect the flow of a message (Nalimov, 2021). It can be of different types; Alternative, options and loop.

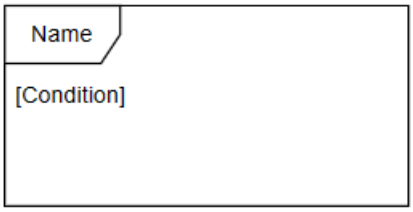


Figure 28: Sequence Fragment(Sequence Diagram).

5.6.1. Sequence Diagram of the System

Creating a thorough sequence diagram for the **"Purchase Plants"** use case is a methodical procedure designed to illustrate the orderly progression of system interactions. To start, the main actors are identified in order to establish their roles and responsibilities. Then, domain objects that are relevant to the use case are taken out of the detailed description, making sure that all the important parts are covered. Then, these items are placed within the system according to the order in which they are executed, with the main actor in charge of carrying out operations. Every object has a lifeline that shows how it has existed over time, and there are activation bars that indicate when it is active.

Arrows with different heads are used to depict the flow of messages between actors and the system. These arrows indicate different message types, including destructive, synchronous, return, creation, and reflexive messages. Where necessary, informative remarks might be added to the diagram to give context and clarification. Furthermore, pieces like alternatives, loops, and optional paths are used to effectively illustrate complex circumstances. The sequence diagram, which provides a visual representation of the "Purchase Plants" use case and its interactions inside the system, is completed once all components have been integrated.

Following these steps, the Sequence Diagram for the "Purchase Plants" use case is developed, providing a detailed depiction of the chronological sequence of interactions and message exchanges within the system. The Sequence Diagram for purchasing plants is presented below, offering a clear visualization of the system's operation.

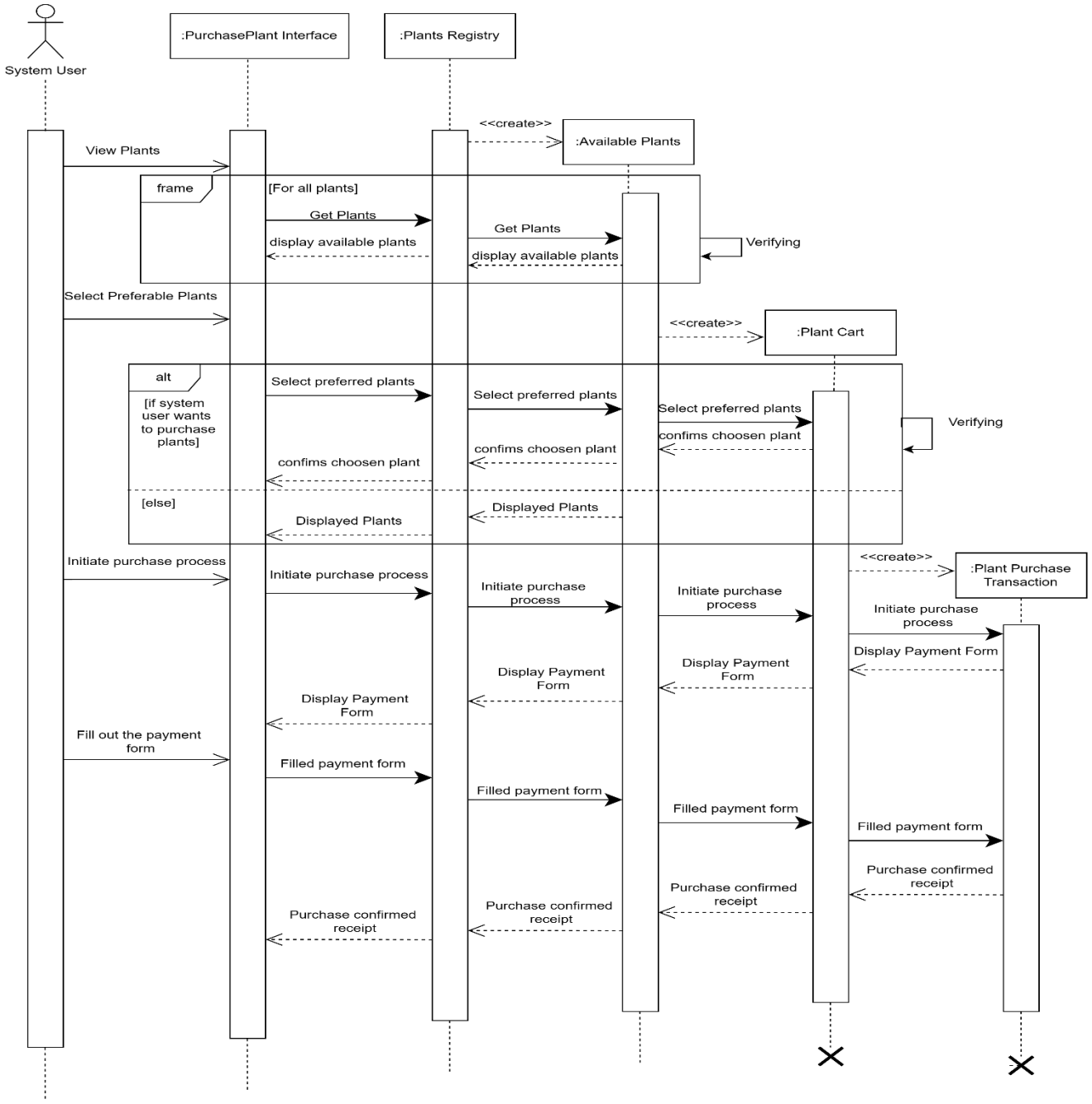


Figure 29: Sequence Diagram of the system.

The sequence diagram above offers a detailed insight into the sequential interactions within a plant purchasing system. At its core, the diagram outlines the interplay between different system components and the user. The process begins with the user's action of viewing available plants, facilitated by the Purchase Plant Interface, which retrieves relevant data from the Plants Registry. Upon selecting preferred plants, a virtual shopping cart is instantiated to hold the selections.

Then, the user initiates the purchase process, triggering the creation of a new plant purchase transaction. The system then prompts the user to fill out a payment form, concluding in the submission of completed payment details. Following validation by the system, a confirmation receipt is generated, signifying the successful completion of the purchase. This sequential flow of interactions ensures a streamlined and user-friendly experience for individuals navigating the plant purchasing system.

5.7. Class Diagram

Class Diagrams are the structure diagram of the UML which consists of attributes and functions. It is used when developing an object-oriented system model to show the classes in a system and the associations between these classes. The class diagram is divided into three parts where the first part has class name, second has attributes and third has methods (Sommerville, 2011).

The class diagram has various relationships between the classes. They are as follow:

- **Generalization**: Generalization, which stands for inheritance, is the most common relationship in class diagrams. This connection links specialized subclasses to their generalized parent classes. It's depicted as a line with a solid arrowhead pointing from the subclass to the superclass (McNeish, 2022).

Figure 30: Generalization(Class Diagram).

- **Association:** In a class diagram, an association stands for a structural link and the means of communication between classes. It represents the relationship between the classes as a solid line. Each end of the line, where it connects to a class, is called an association end (McNeish, 2022).

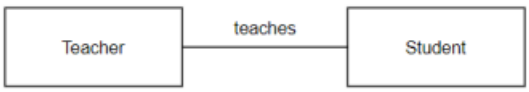


Figure 31: Association(Class Diagram).

- **Aggregation:** An aggregation association is used to represent a whole/part relationship, where one class represents a larger whole composed of smaller parts. Visually, it appears similar to a regular association, but with an open diamond shape added at the "whole" end. This diamond signifies the aggregation, indicating that one class contains or is composed of instances of the other class (McNeish, 2022).

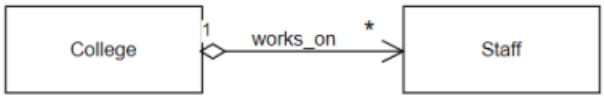


Figure 32: Aggregation(Class Diagram).

- **Composition**: A more powerful kind of aggregation is composition. It demonstrates how one class completely owns and regulates another. In diagrams, it's represented by a filled diamond shape at the "whole" end of the association line (McNeish, 2022).

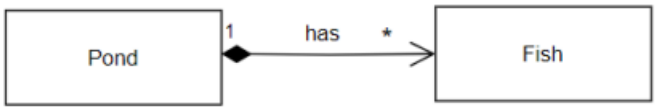


Figure 33: Composition(Class Diagram).

5.7.1. Class Diagram of the System

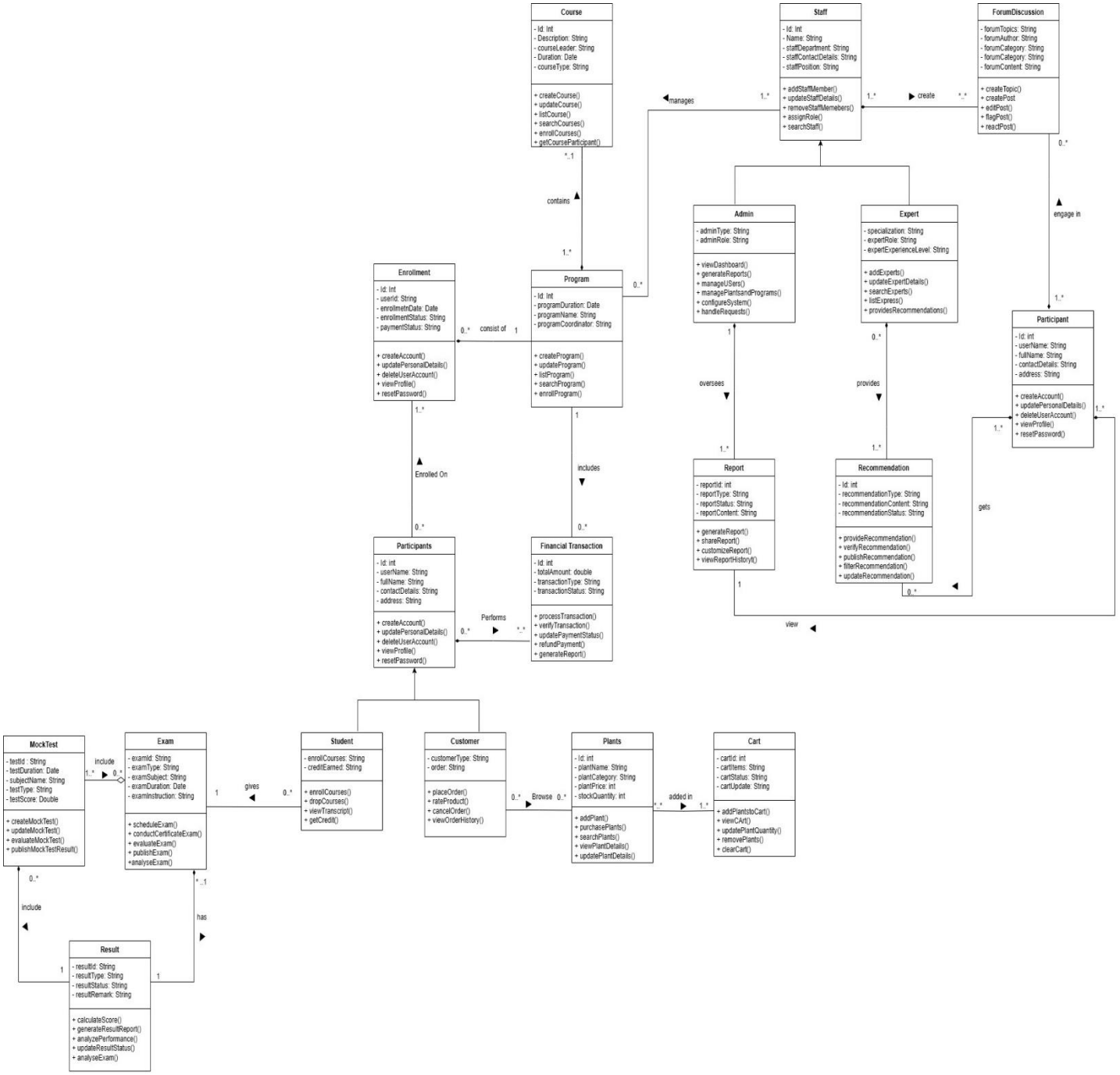


Figure 34: Class Diagram of the system.

The class diagram provided outlines the structure of a plant purchasing system, comprising essential classes and their relationships. The "Customer" class represents system users, encompassing attributes such as customer ID, name, and contact details. Correspondingly, the "Plant" class encapsulates plant information like ID, name, and price. "Order" and "Payment" classes manage customer orders and payments, respectively, with attributes such as order ID, status, and payment method.

A "Cart" class facilitates virtual shopping cart functionality for customers, linked to their respective orders. Relationships between classes, indicated by associations, elucidate interactions within the system. For instance, a customer may have multiple orders, while an order can comprise multiple plants. The diagram visualizes the system's core functionalities, guiding the development of a robust plant purchasing platform.

6. Further Development

McGregor Institute of Botanical Training aim in launching its online service for offering multiple services like short-term certifications programs in horticulture, offer a variety of plants for sale and foster a community of plant enthusiasts through a dedicated online platform to serve as a forum for discussions, program organization, and expert recommendations. The overall planning and analysis of the project “McGregor Institute of Botanical Training” is completed so far. All the necessary features have been included and the project is all set for further processes like development, testing followed by deployment and maintenance.

6.1. Development

In the development phase, McGregor Institute of Botanical Training focuses on bringing its online platform vision to life. This stage involves transforming the planned features and requirements into a practical and intuitive digital environment that serves the goals of the company. In order to achieve scalability, maintainability, and efficiency, key components of development include designing software architecture, implementing design patterns, and choosing appropriate programming paradigms.

6.1.1. Programming Paradigm

A programming paradigm is an essential method or approach to programming that offers a collection of ideas, concepts, and methods for creating and executing computer programs (Liyan, 11). Object-Oriented Programming, Functional Programming, Declarative Programming and Event-Driven Programming are some of the examples of popular programming paradigm. The choice of programming paradigm plays a significant role in shaping the structure and behaviour of the online platform. To promote modularity and reusability, the development of McGregor Institute of training is based on **Object-Oriented Programming Paradigm**.

The core idea of object-oriented programming is that objects are just instances of classes where Code is arranged into objects that encapsulates both data and behavior (Liyan, 11).Since this program is a long-lasting product, numerous new

developers will likely upgrade the system's capabilities. Therefore, it is important to use the right paradigms to bring about changes speedily.

6.1.2. Software Architecture

Software architecture is the high-level planning and structuring of software, where key choices are made about the overall layout of the program, including component relationships, data flow patterns, and the means of communication between various system components (Manchanda, 2023). The chosen software architecture for McGregor Institute's online platform is the **Model-View-Controller (MVC)**. By adopting the MVC architecture, McGregor Institute achieves separation of concerns, modularity, and maintainability in its online platform development. This architectural style facilitates parallel development, enhances code reusability, and simplifies maintenance, enabling efficient collaboration among development teams and seamless evolution of the platform over time.

Among all the different layers, the **4-layer architecture** is ideal for McGregor Institute's online platform due to its **complexity and long-term scalability** needs. Layered architecture patterns are n-tiered patterns with components organized horizontally where all of the components are interrelated but not dependent on one another. The presentation layer, which manages user interfaces and interactions, is located at the top of the hierarchy. The business layer beneath it contains the critical logic and rules that govern program enrollment, plant sales, and community participation. The persistence layer sits beneath, allowing for efficient data storage and retrieval using object-relational mapping techniques. Finally, at the foundation, the database layer securely stores all platform data, assuring its integrity and availability. These layers work together to produce a unified design that enables the platform's scalability, maintainability, and efficient operation (Walpita, 2019).

6.1.3. Design Pattern

Software design patterns are communicative objects and classes that are customized to tackle a broad design problem in a specific environment. Software design patterns are reusable solutions to typical challenges encountered during software design and development. There are three types of Design Patterns: Creational Design Pattern, Structural Design Pattern and Behavioral Design Pattern. The instantiation process is abstracted using the Creational Design Pattern which contribute to make a system independent of the manner in which its items are formed, constructed, and displayed (geeksforgeeks, 2024). **Creational design patterns** are further divided into several categories from which the **Factory Method pattern** is used by McGregor Institute.

The Factory Method Design Pattern allows subclasses to alter the type of objects created by providing an interface in a superclass. It encapsulates object creation logic in a separate method, fostering loose coupling between the creator and the created objects (geeksforgeeks, 2024).

The Factory Method pattern is a valuable choice for McGregor Institute's online platform as it aims to launch a comprehensive service offering multiple functionalities. By centralizing the creation of objects, McGregor Institute can utilize the Factory Method pattern to create various types of objects required for its platform, including user accounts, course offerings, and plant listings. This approach abstracts the actual object creation process, allowing for flexibility and scalability in managing different types of objects.

In addition to the Factory Method pattern, McGregor Institute's online platform also incorporates the **Model-View-Controller (MVC)** design pattern for handling the **view component**.

6.2. Testing

Testing is an important step in the development of McGregor Institute's online platform, ensuring that the software satisfies quality requirements and performs as planned. This phase involves systematically checking and validating the platform's numerous components and features to detect and resolve any problems or issues before deployment. McGregor Institute uses a variety of testing approaches, including black box and white box testing, to thoroughly evaluate the platform's functionality, dependability, and performance.

6.2.1. Black Box Testing

Black box testing is a software testing methodology in which the testing team examines the workings of an application without first gaining a thorough understanding of its underlying structure and design. In testing, the input value is simply compared to the output value. Because of its nature, black box testing is sometimes termed specification-based testing, closed box testing, or opaque box testing (Ashtari, 2022).

McGregor Institute of Botanical Training employs Black Box Testing to evaluate the functionality of their online platform from an end-user perspective without the need to fully understand its internal structure. By simulating real-world scenarios and testing various functionalities, such as user registration, course enrolment, and plant purchasing, Black Box Testing ensures comprehensive coverage and ease of implementation. This approach contributes to the platform's reliability, usability, and overall quality.

Below, the Black Box Testing is outlined:

Black Box Testing

Project Name: McGregor Institute of Botanical Training System

Module Name: Login

Created By: Dikshya Sharma

Created Date: April 20, 2024

Test Date: April 25, 2024

Test Number	**Test_Case_Login_01**
Test Scenario	Verify the login of the user in the system
Test Objective	To ensure users successfully login, to the system
Test Case	Enter valid credentials to login in the system.
Pre-Conditions	Need a valid user account to log in to the system.
Test Steps	a. Navigate to the Login Page. b. Enter valid credentials. c. Submit login form.
Expected Outcome	The user will navigate to the home page of the system.
Actual Outcome	The user navigated to the home page of the system.
Result	Successful Login to the system.
Priority	High Priority.
Remark	Test was successful

Table 3: Black Box Testing of Login Feature(Successful).

After conducting Test_Case_Login_01, it was observed that the login functionality of the system performed as expected. The test was successful, ensuring that users can securely access the system with their credentials.

Test Number	Test_Case_Login_02
Test Scenario	Verify the login functionality with invalid credentials.
Test Objective	To prevent users from logging into the system with incorrect credentials.
Test Case	Invalid Credentials Login Attempt.
Pre-Conditions	A valid user account is required to access the system.
Test Steps	a. Navigate to the Login Page. b. Enter invalid credentials (e.g., incorrect username or password). c. Submit the login form.
Expected Outcome	The user should receive an error message indicating that the submitted credentials are incorrect.
Actual Outcome	The user did not receive an error message indicating incorrect credentials despite navigated to the home page of the system.
Result	Unsuccessful Login to the system.
Priority	High Priority.
Remark	Test executed as expected, validating the system's handling of invalid login attempts.

Table 4: Black Box Testing of Login(Unsuccessful).

Test_Case_Login_02 was conducted to assess the system's response to invalid login attempts. Despite the expectation for the system to display an error message upon submission of incorrect credentials, it instead allowed access to the home page without indicating any error. This outcome indicates a failure in preventing unauthorized access. This security measure ensures protection against unauthorized login attempts. Later, the issue was corrected to maintain the system's integrity and security.

6.2.2. White Box Testing

White Box Testing, also known as structural, code-based, or glass box testing, is a software testing method that focuses on the software's internal logic, structure, and coding. It gives testers full application knowledge, including access to source code and design papers, allowing them to analyze and validate the software's internal workings, infrastructure, and interconnections (Akhtar, 2023).

McGregor Institute of Botanical Training employs White Box Testing to assess the functionality of their online platform by looking at its underlying structure, code routes, and logic. This method entails evaluating the software's internal workings, such as code statements, branching, and loops, to assure comprehensive testing coverage. White Box Testing, which analyzes the underlying code, identifies potential vulnerabilities, optimization opportunities, and places for development in the platform. McGregor Institute uses White Box Testing to improve platform robustness, performance, and security, with the goal of providing a high-quality user experience.

Below, the White Box Testing is outlined:

White Box Testing

Project Name: McGregor Institute of Botanical Training System

Module Name: Join Program

Created By: Dikshya Sharma

Created Date: April 20, 2024

Test Date: April 29, 2024

Test Number	**Test_Case_Join_Program_01**
Test Scenario	Verify the enrollment of users in the program.
Test Objective	To ensure the enrollment of users in the system.
Test Case	Select the preferred program to get enrolled in.
Pre-Conditions	- A valid user account is required for login and access to the programs. - The Join Program page must be accessible.
Test Steps	a. Log in with valid credentials to access the system. b. Navigate to the Join Program page. c. Select the desired program from the available options. d. Complete any necessary fields or forms related to program enrollment. e. Submit the enrollment request.
Expected Outcome	- The system should successfully process the enrollment request. - Upon successful enrollment, the user should receive confirmation.
Actual Outcome	The enrollment request is processed successfully, and the user receives confirmation.
Result	Successful Enrollment in the program.
Priority	High Priority.
Remark	Test executed successfully without any issues.

Table 5: White Box Testing of Join Program.

Test_Case_Join_Program_01 was conducted to validate the system's capability to enroll users in programs successfully. Following the predefined steps, the user logged in with valid credentials and navigated to the Join Program page. Upon selecting the desired program and submitting the enrollment request, the system processed it without encountering any issues. Consequently, the user received confirmation of successful enrollment. This outcome indicates that the system effectively facilitates program enrollment, thereby meeting the test objective.

6.3. Deployment

Deployment serves as a crucial stage within the Systems Development Life Cycle (SDLC), in which a software program or system is released and made available to end users. It entails installing, configuring, and activating the software in the target environment to ensure smooth integration and peak performance acting as the crucial bridge between the development and operation phases (Chisel, 2023).

In the process of deploying the McGregor Institute system, we anticipate encountering various challenges during testing. However, we are committed to resolving any issues to ensure that the system meets its requirements. To facilitate this, we have opted for a cloud-based solution, **utilizing Google Cloud Platform (GCP)** with automated and continuous deployment practices.

Google Cloud Platform offers a diverse range of services and operates on a pay-as-you-go model, making it a suitable choice for our deployment needs. During the deployment process, we will meticulously configure the system's functional components within the GCP environment. Leveraging various GCP services, we will optimize the system's performance, scalability, and database management capabilities.

6.4. Maintenance

Software maintenance is a crucial component of programming that guarantees the long-term efficiency, security, and dependability of software. Software maintenance is the process of upgrading and adjusting already-existing software to make sure it keeps working as intended and can adapt to the user's changing needs. Feature enhancements, speed optimization, and bug fixes are all part of the process (Davidson, 2023).

In order to maintain the effectiveness of the McGregor Institute of Botanical Training System, a thorough Maintenance Plan has been developed. To ensure peak performance, this plan includes routine system updates and health checks. To guard against data loss and cyber threats, scheduled data backups and security setups are put into place. Because of its auto-scaling architecture, the system can easily handle growing user needs. User input is gathered for ongoing development through timely surveys. To keep up with changing needs, hardware specifications are updated, and a thorough user manual is included to improve the user experience. Regular security updates also provide defense against attacks and weaknesses, maintaining the integrity of the system and shielding private information. The McGregor Institute system is nonetheless reliable, safe, and sensitive to user demands thanks to these safeguards.

6.5. Prototype Development

Prototype development is like building rough drafts of ideas before making the final product. They help teams see what works and what needs improvement early on, saving time and money later. In this section, the created prototypes are displayed.

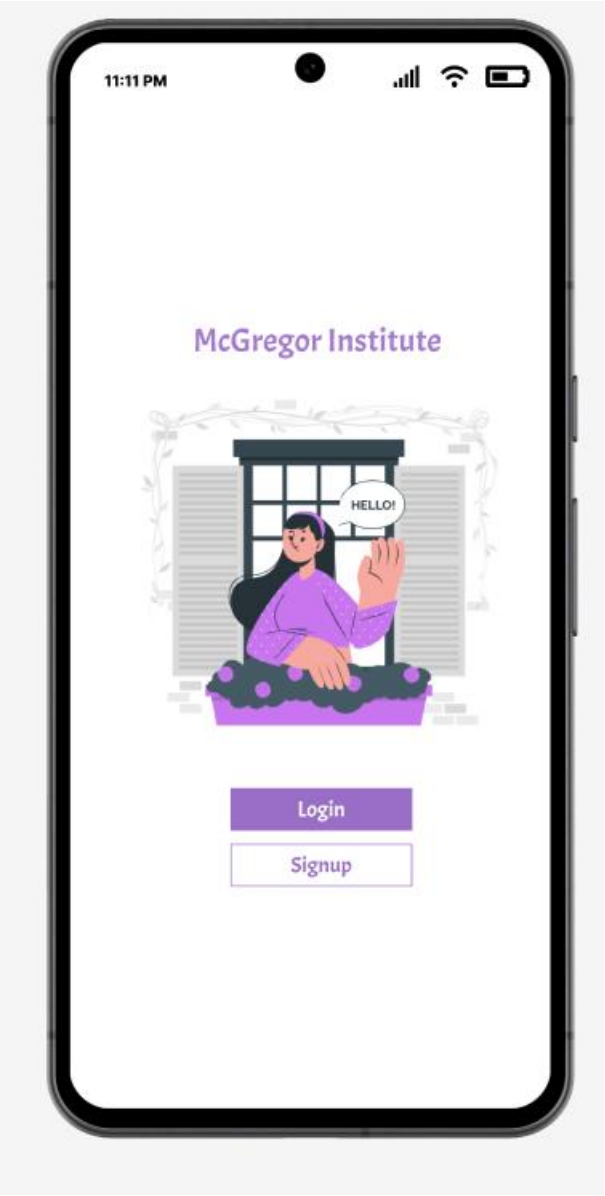


Figure 35: Prototype of Login/Register.

Presented here is the prototype for the login/registration page.

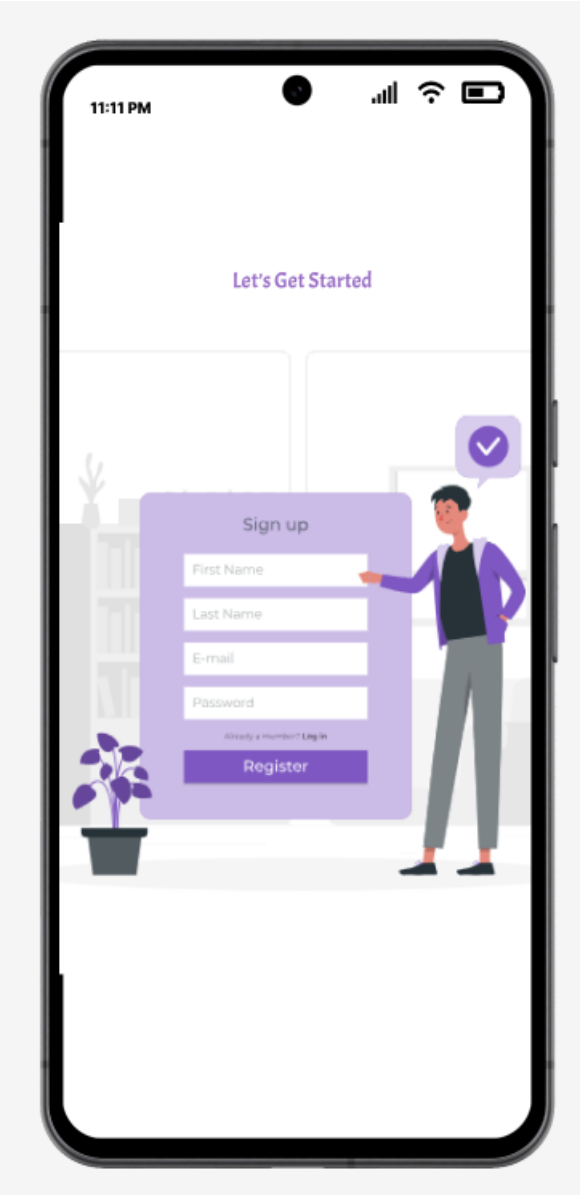


Figure 36: Prototype of Register Page.

Presented here is the prototype for the registration page.

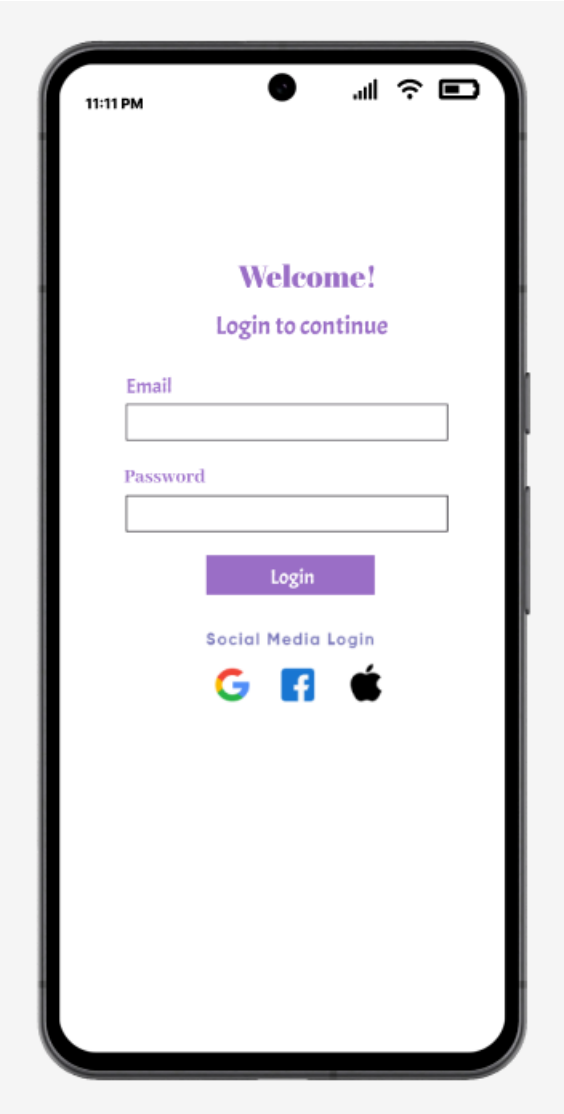


Figure 37: Prototype of Login Page.

Presented here is the prototype for the login page.

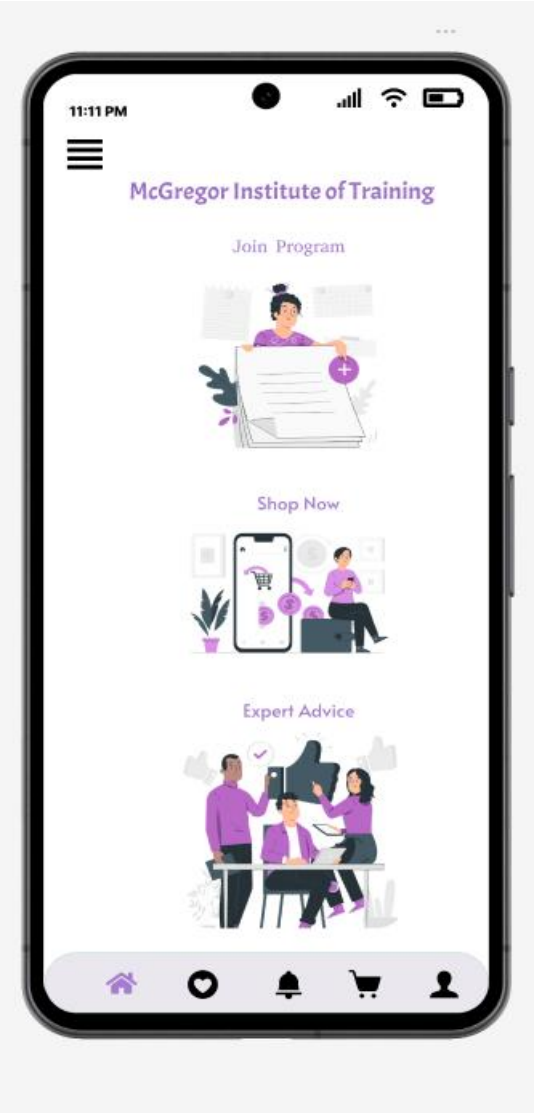


Figure 38: Prototype of Home Page.

Presented here is the prototype for the Home page.

Figure 39: Prototype of Products Page.

Presented here is the prototype for the Products page.

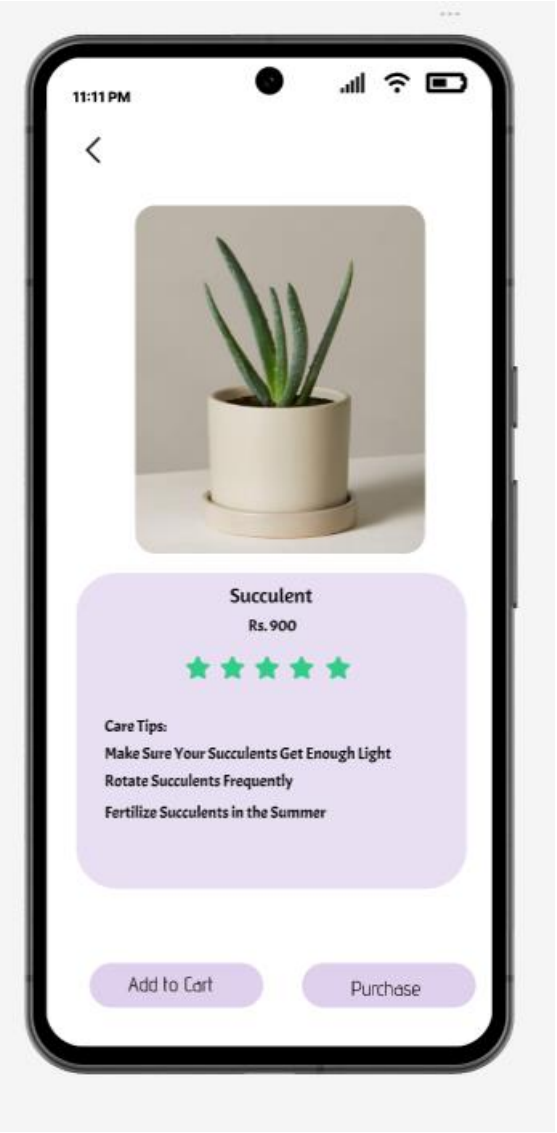


Figure 40: Prototype of Product Description.

Presented here is the prototype for the Product Description page.

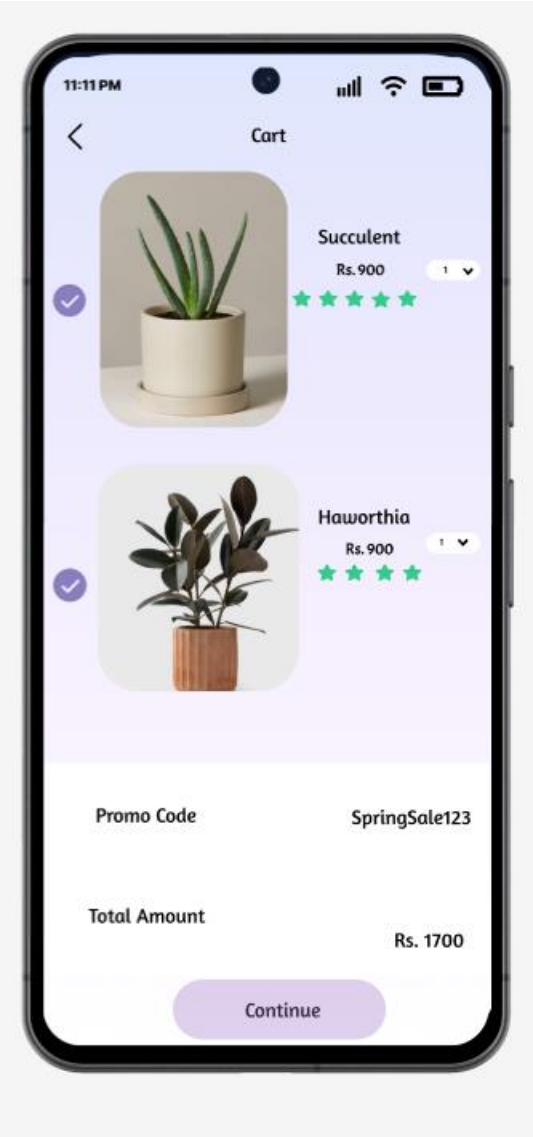


Figure 41: Prototype of Added Cart.

Presented here is the prototype for the Cart page.

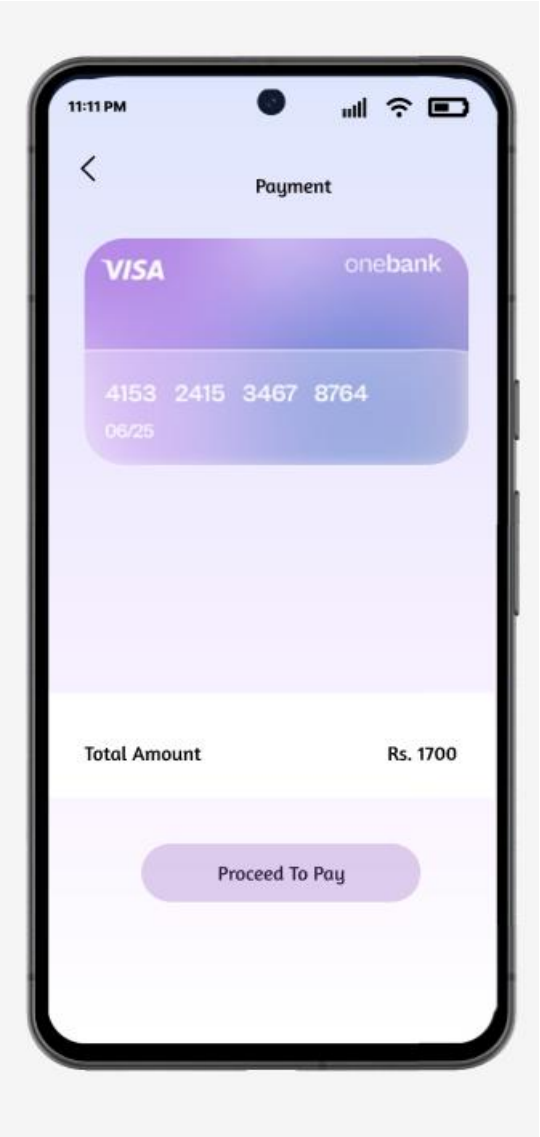


Figure 42: Protype of Payment Page.

Presented here is the prototype for the Payment page.

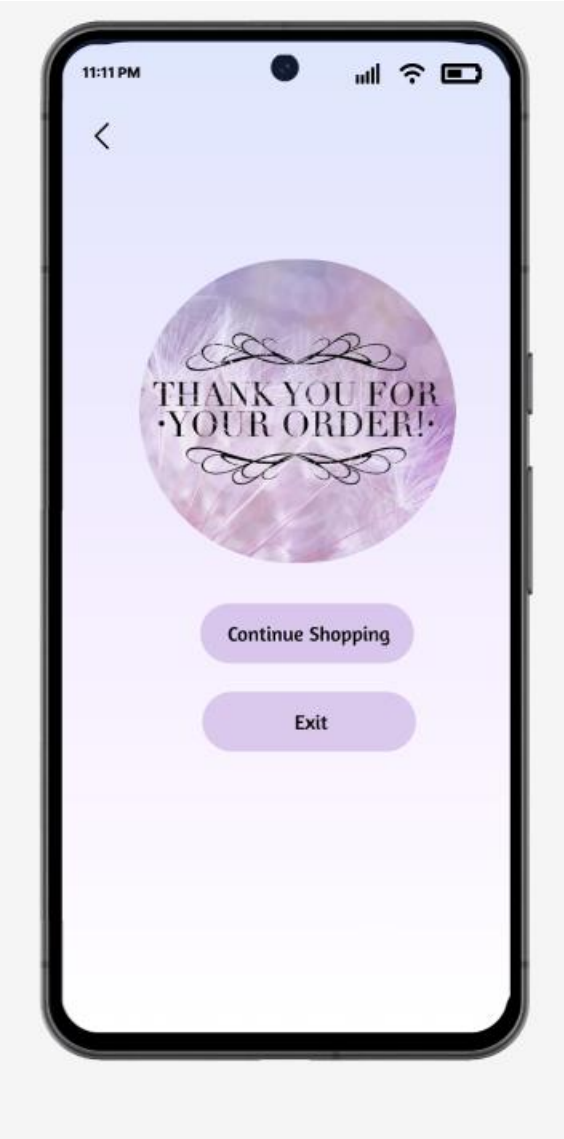


Figure 43: Prototype of successful purchase page.

Presented here is the prototype for the order success page.

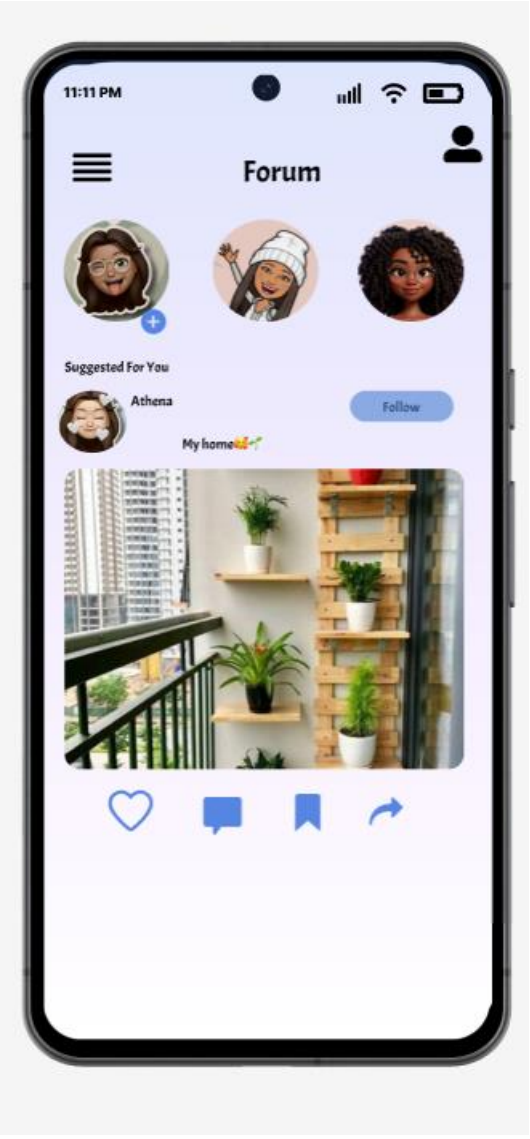


Figure 44: Prototype of Forum.

Presented here is the prototype for the forum page.

Figure 45: Prototype of Expert Recommendations Page.

Presented here is the prototype for the Recommendations page.

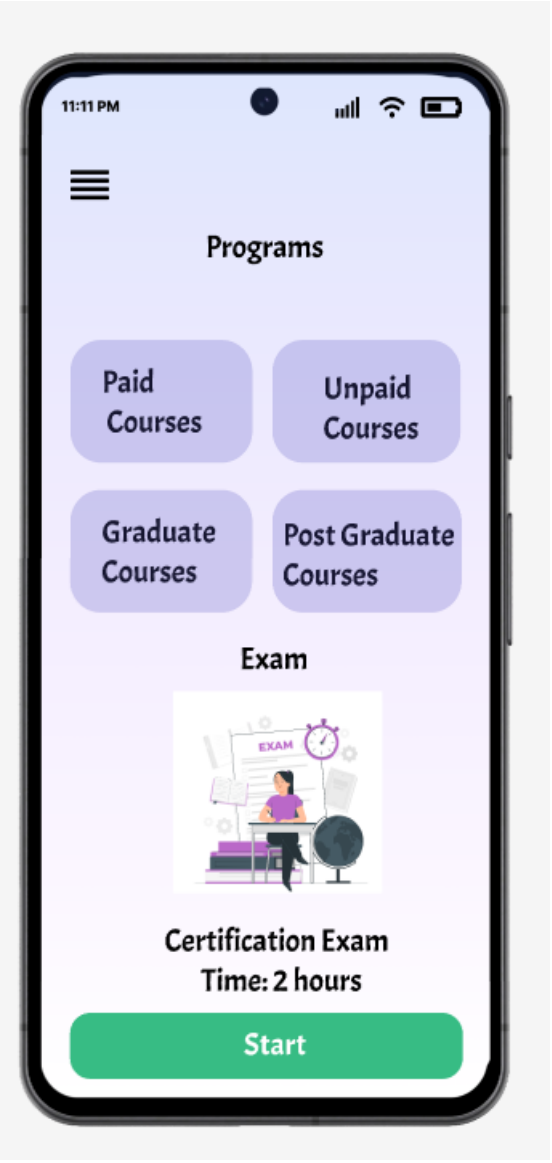


Figure 46: Prototype of Programs and Exams Details

Presented here is the prototype for the Programs page.

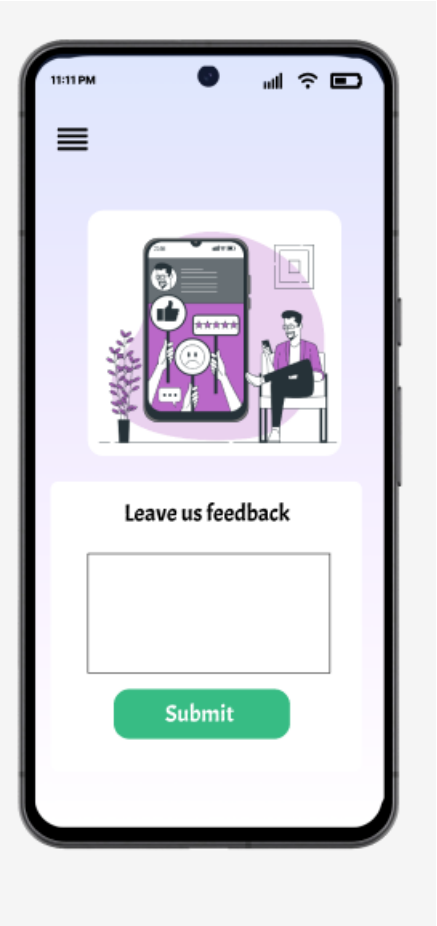


Figure 47: Prototype of Feedback Page.

Presented here is the prototype for the Feedback page.

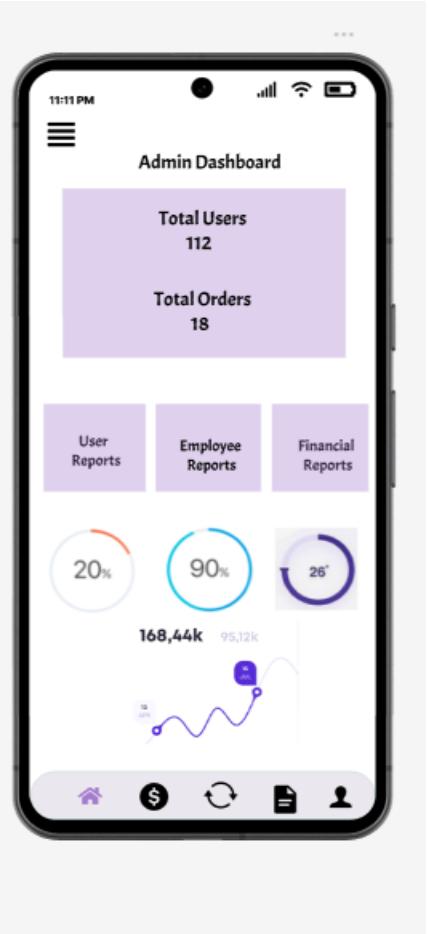


Figure 48: Prototype of Admin Dashboard.

Presented here is the prototype for the Admin Dashboard page.

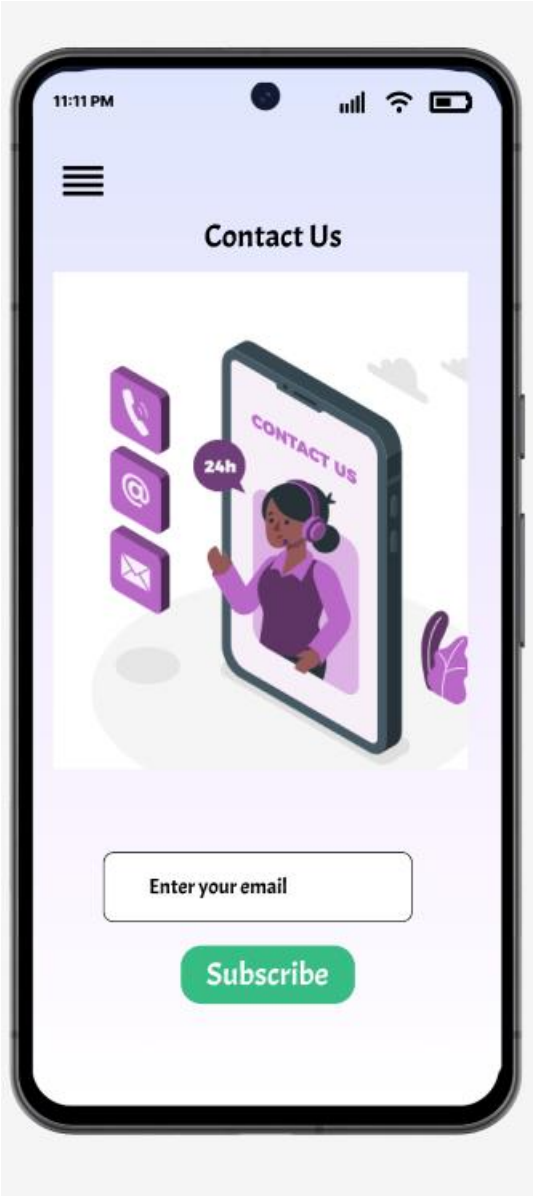


Figure 49: Prototype of Contact Us Page.

Presented here is the prototype for the Contact Us page.

7. Conclusion

In this project for the McGregor Institute of Botanical Training, we embarked on developing a new software system to enhance their system. Although it was challenging to complete, the results were really satisfying. Our system was carefully planned and constructed using a variety of techniques, such as Gantt charts and UML diagrams. Our planning process was defined by an intense focus on detail, and the Gantt chart, which outlined each stage and its related timetable, acted as a guide. Likewise, to see the complex relationships between various components, we made use of diagrams.

After doing a thorough analysis, we carefully chose to implement the Feature-Driven Development (FDD) technique, which is renowned for its flexibility and effectiveness in meeting changing project needs. Before moving on to the finished product, we developed prototypes to test and improve our ideas.Furthermore, we ensured proper testing procedures, maintenance processes, as according to the chosen programming paradigm, and implementation of architectural design patterns.

The project offered us great insights into the complexities of real-world software development. Despite facing obstacles along the way, our collaborative attitude and determination pushed us to victory.
With our enhanced understanding and refined skills, we are prepared to confidently tackle similar challenges in the future.

8. References

Abdullahi, A., 2023. *What is a Gantt chart and how does it work?.* [Online] Available at: https://www.techrepublic.com/article/gantt-chart/ [Accessed 6 April 2024].

Akhtar, H., 2023. *What is White Box Testing? (Example, Types, & Techniques) | BrowserStack.* [Online] Available at: https://www.browserstack.com/guide/white-box-testing [Accessed 28 April 2024].

Ashtari, H., 2022. *Black Box Testing vs. White Box Testing.* [Online] Available at: https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/ [Accessed 24 April 2024].

Carol Britton, J. D., 2005. Use Cases. In: *A Student Guide to Object-Oriented Development.* Oxford: Elsevier Butterworth-Heinemann, p. 405.

Chisel, 2023. *What is Deployment? Meaning and Process | Glossary.* [Online] Available at: https://chisellabs.com/glossary/what-is-deployment/ [Accessed 29 April 2024].

Christine Organ, C. B., 2022. *Work Breakdown Structure (WBS) In Project Management – Forbes Advisor.* [Online] Available at: https://www.forbes.com/advisor/business/what-is-work-breakdown-structure/ [Accessed 24 April 2024].

Coursera, 2024. *Work Breakdown Structure (WBS): Overview, Uses, Software | Coursera.* [Online] Available at: https://www.coursera.org/articles/work-breakdown-structure [Accessed 20 April 2024].

Daly, N., 2022. *What Is a Use Case & How To Write One | Wrike.* [Online] Available at: https://www.wrike.com/blog/what-is-a-use-case/ [Accessed 7 April 2024].

Davidson, T., 2023. *Importance of Software Maintenance in Software Engineering | Clean Commit.* [Online]

Available at: https://cleancommit.io/blog/importance-of-software-maintenance-in-software-engineering/
[Accessed 29 April 2024].

geeksforgeeks, 2024. *Software Design Patterns Tutorial - GeeksforGeeks.* [Online] Available at: https://www.geeksforgeeks.org/software-design-patterns/#types-of-software-design-patterns
[Accessed 20 April 2024].

Indeed Editorial Team, 2023. *What Is Unified Modeling Language (UML)? Types and Uses | Indeed.com.* [Online] Available at: https://www.indeed.com/career-advice/career-development/what-is-uml
[Accessed 7 April 2024].

javatpoint, 2021. *Use-Case Model - Javatpoint.* [Online] Available at: https://www.javatpoint.com/use-case-model
[Accessed 6 April 2024].

Lad, A., 2022. *What is a use case? Definition, template, and how to write one - LogRocket Blog.* [Online] Available at: https://blog.logrocket.com/product-management/what-is-a-use-case-template-how-to-write/
[Accessed 6 April 2024].

Lewis, S., 2023. *What is a Collaboration Diagram?.* [Online] Available at: https://www.techtarget.com/searchsoftwarequality/definition/collaboration-diagram
[Accessed 13 April 2024].

Liyan, A., 11. *What is a programming paradigm?. A programming paradigm is a fundamental… | by Ario Liyan | Medium.* [Online] Available at: https://medium.com/@Ariobarxan/what-is-a-programming-paradigm-ec6c5879952b
[Accessed 20 April 2024].

Lombardi, P., 2022. *indeed.* [Online] Available at: 11 Software Development Methodologies (Plus How To Pick One) | Indeed.com
[Accessed 12 April 2024].

Lutkevich, B., 2021. *What is a Gantt chart? - Definition from SearchSoftwareQuality.* [Online]
Available at: https://www.techtarget.com/searchsoftwarequality/definition/Gantt-chart
[Accessed 6 April 2024].

Manchanda, A., 2023. *What is Software Architecture? | Net Solutions.* [Online]
Available at: https://www.netsolutions.com/insights/why-software-architecture-matters-to-build-scalable-solutions/
[Accessed 20 April 2024].

McNeish, K., 2022. *UML Class Diagrams.* [Online]
Available at: https://www.codemag.com/article/0201061/UML-Class-Diagrams
[Accessed 13 April 2024].

McNeish, K., 2022. *UML Sequence Diagrams.* [Online]
Available at: https://www.codemag.com/article/0203081/UML-Sequence-Diagrams
[Accessed 13 April 2024].

Nalimov, C., 2021. *Sequence diagram with fragments | Gleek | Gleek.* [Online]
Available at: https://www.gleek.io/blog/sequence-diagram-fragments
[Accessed 13 April 2024].

Nalimov, C., 2021. *UML communication diagrams: how are they used? | Gleek | Gleek.* [Online]
Available at: https://www.gleek.io/blog/uml-communication-diagram
[Accessed 13 April 2024].

Nishadha, 2022. *Use Case Diagram Tutorial (Guide with Examples) | Creately.* [Online]
Available at: https://creately.com/guides/use-case-diagram-tutorial/
[Accessed 6 April 2024].

Pandey, S., 2024. *What is Sequence Diagram? - Naukri Code 360.* [Online]
Available at: https://www.naukri.com/code360/library/what-is-sequence-diagram
[Accessed 13 April 2024].

Pressman, R. S., 2010. Feature Driven Development (FDD). In: F. M. Schilling, ed. *SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH.* New York: McGraw-Hill, pp. 86-87.

Sommerville, I., 2011. Class Diagram. In: M. H. M. G. C. B. J. H. Marcia Horton, ed. *Software Engineering.* Boston : Pearson Education, Inc., , pp. 129-130.

ugkijhjkj, 567. *kjgfghgjkhjhk.* [Online]
Available at: kkhj
[Accessed 78 88 99].

U, J., 2021. *UML: Sequence Diagram. Sequence diagram helps us to understand… | by Joshua U | Medium.* [Online]
Available at: https://medium.com/@joshuaudayagiri/uml-sequence-diagram-5c8d1f0b41d6
[Accessed 13 April 2024].

VanZandt, P., 2023. *What is Sequence Diagram? Definition and Sequence Diagrams in UML.* [Online]
Available at: https://ideascale.com/blog/what-is-sequence-diagram/
[Accessed 13 April 2024].

Visual Paradigm, 2022. *The Four Types of Relationship in Use Case Diagram - Visual Paradigm Blog.* [Online]
Available at: https://blog.visual-paradigm.com/the-four-types-of-relationship-in-use-case-diagram/
[Accessed 6 April 2024].

Walpita, P., 2019. *Software Architecture Patterns — Layered Architecture | by Priyal Walpita | Medium.* [Online]
Available at: https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057
[Accessed 4 April 2024].