



 slington college

(इस्लिङ्टन कलेज)

**Module Code & Module Title**

**CS5002NI SOFTWARE ENGINEERING**

**Assessment Weightage & Type**

**35% Individual Coursework**

**Year and Semester**

**2022-23 Spring**

**Student Name: Roshani Rauniyar**

**London Met ID: 22085824**

**College ID: nnp01cp4s230166**

**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 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 marks of zero will be awarded.*

## Table of Contents

2	Introduction .....	1
3	Work Breakdown Structure (WBS).....	4
4	Gantt Chart .....	6
5	Use case Model .....	9
5.1	Use case diagram: .....	12
5.2	High level use case description:.....	13
5.2.1	Register .....	13
5.2.2	Join program.....	13
5.2.3	Purchase plants.....	13
5.2.4	Make payment .....	14
5.2.5	Ask recommendation .....	14
5.2.6	Prepare report .....	14
5.2.7	Take examination .....	15
5.2.8	Engage forum .....	15
5.3	Expanded Use Case Description: .....	16
5.3.1	Register .....	16
5.3.2	Make payment .....	18
6	Sequence Diagram .....	21
7	Communication Diagram.....	24
8	Class diagram .....	27
9	Further development.....	29
9.1	Architectural choice.....	30
9.2	Design pattern.....	32

10	Development Plan .....	39
11	Testing Plan .....	41
12	Maintenance Plan.....	47
13	Prototype deployment.....	52
14	Conclusion.....	64
15	References .....	65

**Table of tables**

Table 1:Use case of register user .....	13
Table 2:Use case of join program .....	13
Table 3:Use case of purchase plant.....	13
Table 4:Use case of make payment.....	14
Table 5:Use case of ask recommendation .....	14
Table 6:Use case of prepare report.....	15
Table 7:Use case of take examination .....	15
Table 8:Use case of engage forum .....	16
Table 9:Expanded use case description of register user.....	17
Table 10:Expanded use case description of make payment .....	20

## Table of Figures

Figure 1: Work breakdown Structure (WBS) .....	5
Figure 2: Gannt chart .....	8
Figure 3: Use case diagram .....	12
Figure 4: Sequence Diagram .....	23
"Figure 5: Communication Diagram .....	26
Figure 6: Class Diagram.....	28

# 1 Introduction

McGregor Institute of Botanical Training, based in Ireland with a branch located in Godawari, Lalitpur, Nepal, has been at the forefront of providing high-quality education in agriculture and horticulture for almost 7 years. Affiliated with Dublin City University, McGregor Institute offers a wide range of undergraduate and postgraduate courses specializing in agriculture and horticulture, catering to individuals passionate about botanical sciences.

With a recent surge in interest in the field of agriculture and horticulture, McGregor Institute is gearing up to introduce a series of short-term certification courses focused on horticulture, catering to anyone eager to enhance their knowledge and skills in plant cultivation. Additionally, recognizing the growing demand for plants and greenery, McGregor Institute aims to offer various plant varieties for sale, charging minimal fees and even providing some plants for free to foster a sense of community among plant enthusiasts.

Moreover, McGregor Institute envisions creating a vibrant platform, a forum, where plant enthusiasts can engage in discussions, share ideas, and organize programs aimed at protecting rare plants and forests. This forum will also serve as a space where individuals can seek expert advice on plant cultivation, with features allowing users to pinpoint their location on a map, share images of soil conditions, and receive tailored recommendations for suitable plants or crops.

To streamline these operations and manage the diverse range of services offered, McGregor Institute requires a comprehensive online system. This system will facilitate user registration, course enrolment, plant purchases, payment processing, expert recommendations, certification exams, financial reporting, and community engagement through the forum. By harnessing technology, McGregor Institute aims

to enhance accessibility, efficiency, and collaboration within the botanical community, furthering its mission of promoting sustainable plant cultivation and conservation.

To bring this project to fruition, we'll address elements of project planning, design, and execution. The following aspects outline the thorough approach to achieving the institute's goals.

- **Work Breakdown Structure and Gantt Chart:** The project will be divided into specific tasks using a Work Breakdown Structure (WBS). Each task will be scheduled and timed using a Gantt Chart, providing a clear roadmap for the project's execution. This planning phase will follow a chosen methodology to ensure a systematic progression and efficient use of resources throughout the project.
- **Use Case Development:** A comprehensive Use Case Model will be created to capture the functional requirements of the proposed system. This will include a Use Case Diagram showing the system's actors and their interactions, along with detailed descriptions of each use case. For key use cases, in-depth descriptions will offer detailed insights into the functionalities, serving as a guide for system design and development.
- **Communication Mapping:** Communication Diagrams, including Collaboration and Sequence diagrams, will be developed from the Use Case Model. These diagrams will illustrate the complex interactions between system components and stakeholders, clarifying the communication flow and data exchange during the execution of important use cases.
- **Class Analysis:** An extensive list of domain classes, derived from the use cases, will be organized into a structured table. This will form the basis for the development of an Analysis Class Diagram, showing the relationships between domain entities and their attributes, providing a comprehensive understanding of the system's structure.
- **Prototype Design:** Prototypes will be designed and developed to provide a tangible visualization of the proposed system's features and functionalities. These prototypes will give stakeholders a glimpse of the system's user interface and

interaction flows, encouraging feedback and refinement before the start of full-scale development.

Following the Agile methodology, the project will carefully navigate through architectural decisions, design patterns, development strategies, testing protocols, and maintenance considerations. This comprehensive approach aims to ensure the smooth realization and deployment of McGregor Institute's botanical training platform, aiming to empower individuals and communities in their pursuit of botanical knowledge and environmental conservation.



## 2 Work Breakdown Structure (WBS)

A Work Breakdown Structure (WBS) is a fundamental instrument in project management. It allows for the systematic division of complex projects into manageable parts. It offers a hierarchical layout of project deliverables, enabling accurate planning, organization, and execution.

For the McGregor Institute of Botanical Training project, creating a WBS is crucial to manage the complex task of building an educational platform and community forum. This project involves a variety of interrelated tasks, from website development to content creation and forum setup. By systematically dividing these broad objectives into smaller, more specific work packages, the WBS provides a structured framework for project management.

By arranging tasks hierarchically within the WBS, from major deliverables to detailed activities, the project team obtains a clear understanding of project scope, resource allocation, and scheduling. Each element of the WBS represents a concrete milestone or aspect of the project, ensuring thorough coverage and responsibility throughout the project lifecycle.

In essence, the WBS serves as a roadmap for project execution, helping stakeholders understand the project's complexities and promoting effective communication and collaboration. By adhering to the principles of formal project management and using the WBS as a foundational tool, the McGregor Institute project can methodically work towards its goals, ultimately delivering a strong and influential botanical training platform.

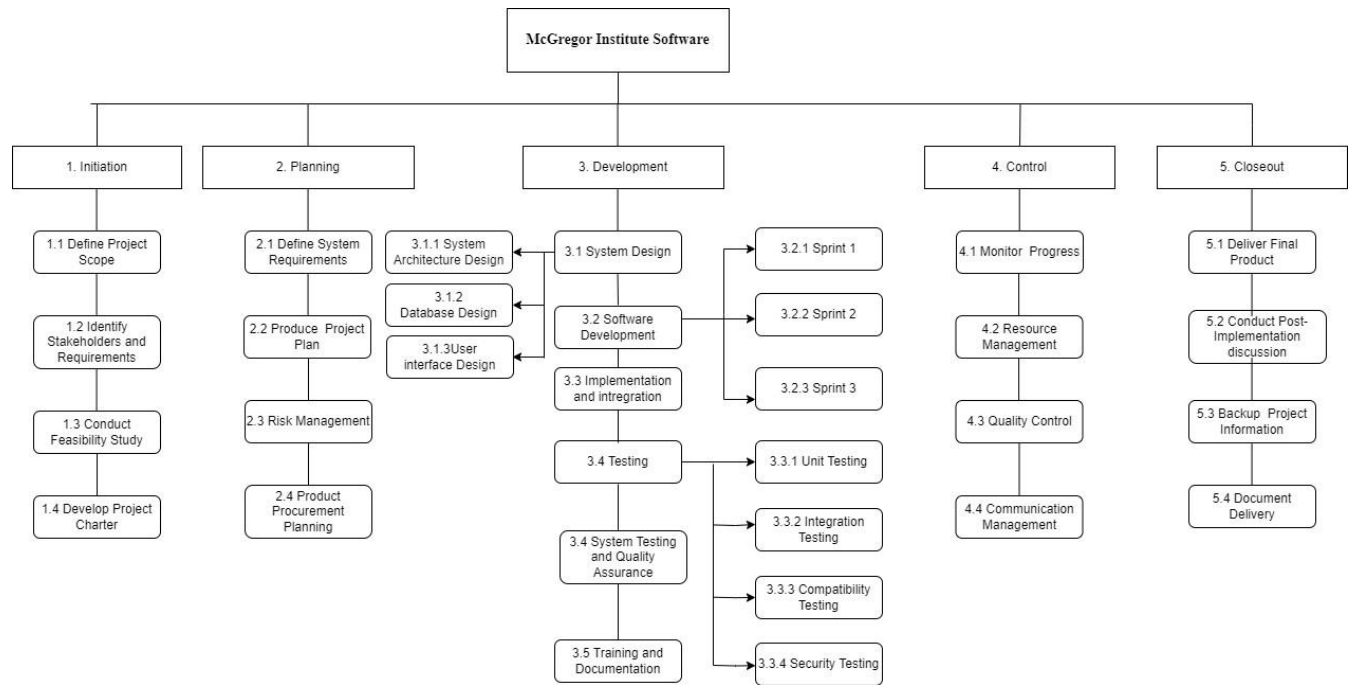


Figure 1: Work breakdown Structure (WBS)

### 3 Gantt Chart

A Gantt Chart is a graphical tool that offers a snapshot of a project's timeline, showcasing tasks, their interconnections, and their respective durations. It's a popular tool in project management, providing a systematic method for planning, overseeing, and tracking project progress from start to finish. (Team Gantt, 2024)

In the context of the McGregor Institute of Botanical Training project, creating a Gantt Chart is crucial for coordinating the multitude of tasks involved in platform development and community engagement. This project involves a variety of interconnected tasks, from initial planning and requirement gathering to website development, content creation, and forum setup. By incorporating these tasks into a Gantt Chart, the project team can gain a clear understanding of task sequencing and resource distribution over time.

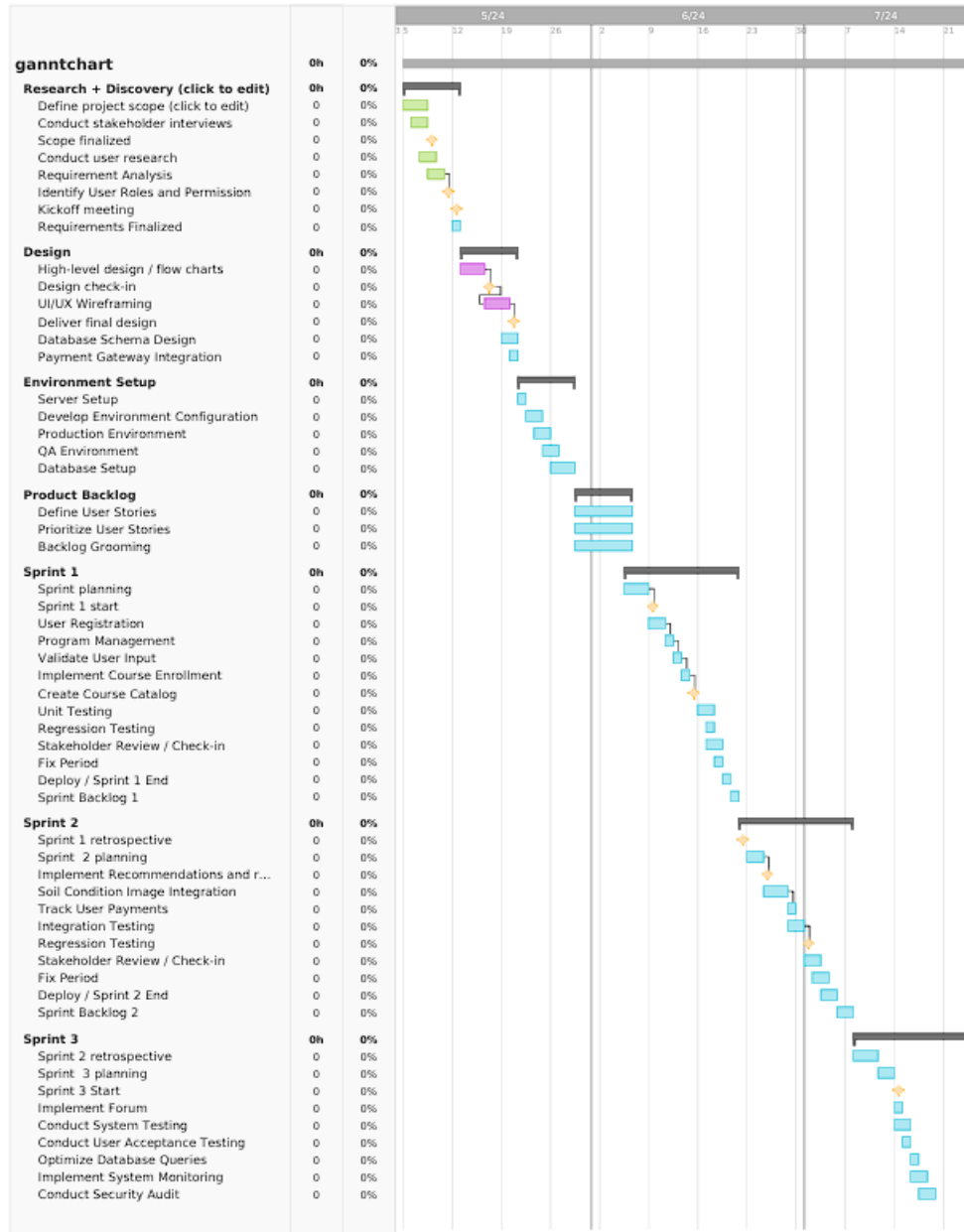
The creation of a Gantt Chart for this project involves several important steps. First, the project manager identifies all project tasks and their dependencies, taking into account factors like task duration, resource availability, and critical path analysis. These tasks are then arranged in chronological order on the Gantt Chart's horizontal axis, marking their start and end dates. Task dependencies are represented using arrows or lines, showing the order in which tasks need to be completed.

Once the Gantt Chart is in place, it becomes a dynamic tool for project planning and execution. Team members can refer to the chart to understand their individual roles, track progress against key milestones, and spot potential bottlenecks or delays. Additionally, stakeholders can gain insight into project timelines and make informed decisions about resource allocation and project prioritization.

In conclusion, the Gantt Chart is instrumental in steering the McGregor Institute project towards successful completion. By offering a structured framework for scheduling and monitoring project activities, the Gantt Chart enables the project team to manage complexities, reduce risks, and ensure the timely and efficient delivery of the botanical training platform and community forum.

Below is a Gantt chart developed to guide and allocate the necessary time for each stage of building an online system for McGregor Institute of Botanical Training.

teamgantt  
Created with Free Edition



teamgantt  
Created with Free Edition

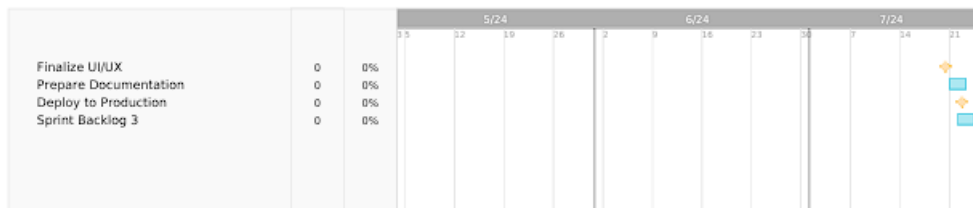


Figure 2: Gannt chart

## 4 Use case Model

In the field of software engineering and system architecture, a Use Case Diagram is an effective instrument that graphically presents a system's functional prerequisites from the standpoint of its users or actors. This diagram delivers a comprehensive view of the interactions between users and the system, demonstrating the diverse tasks, features, and behaviors that the system is required to support to satisfy user requirements and achieve business targets.

For the McGregor Institute of Botanical Training project, constructing a Use Case Diagram is vital in shedding light on the key features and user interactions planned for the botanical training platform and community forum. By systematically outlining the system's use cases, actors, and their interconnections, the Use Case Diagram offers stakeholders a transparent and succinct picture of the system's scope and functionality.

The procedure of creating a Use Case Diagram for this project commences with the project team identifying the main actors or users who will interact with the system. These could encompass learners undertaking horticulture certification courses, website administrators supervising course content, and community members engaging in forum discussions. The objectives or use cases of each actor are then recorded, representing specific actions or tasks they execute within the system.

After pinpointing the actors and use cases, the relationships between them are established, demonstrating how actors interact with the system to achieve their objectives. This includes associations between actors and use cases, as well as relationships among use cases themselves, such as dependencies or extensions.

A Use Case Diagram comprises several key components that collectively illustrate the functionality and interactions of a system from the perspective of its users. These components include:

- I. **Actors:** Actors symbolize the various roles or entities that interact with the system. They can be external entities like users or other systems, or internal entities like system components or subsystems. Actors are represented as stick figures and are positioned outside the system boundary.
- II. **Use Cases:** Use cases symbolize the specific functionalities or tasks that the system needs to perform to meet its users' needs. Each use case describes a series of interactions between the user and the system to achieve a specific goal. Use cases are represented as ovals within the system boundary and are usually labeled with descriptive names.
- III. **Relationships:** Relationships establish connections between actors and use cases, as well as among different use cases. There are three main types of relationships in a Use Case Diagram:
  - **Association:** An association represents a communication or interaction between an actor and a use case. It indicates that the actor participates in the activities described by the use case. Associations are represented as lines connecting actors to use cases.
  - **Generalization:** Generalization represents an "is-a" relationship between two actors or use cases, where one is a specialized version of the other. It indicates that the specialized actor or use case inherits the behaviors and attributes of the generalized actor or use case. Generalization relationships are represented as solid lines with a triangle arrowhead pointing towards the more general actor or use case.
  - **Include:** An include relationship signifies that one use case includes another use case as part of its behavior. It indicates that the included use case is always performed when the base use case is executed.

Include relationships are represented as dashed lines with an arrowhead pointing towards the included use case.

- **Extend:** An extend relationship signifies that one use case extends another use case with additional behavior under certain conditions. It indicates that the extending use case is optional and may be performed based on specific conditions. Extend relationships are represented as dashed lines with an arrowhead pointing towards the extension use case.
- **System Boundary:** The system boundary represents the scope of the system being modeled in the diagram. It defines the boundary between the system and its external actors or entities. Use cases and actors are positioned within the system boundary to indicate their relationship to the system being modeled. Interactions and ensure that system features align with user needs and objectives.



## 4.1 Use case diagram:

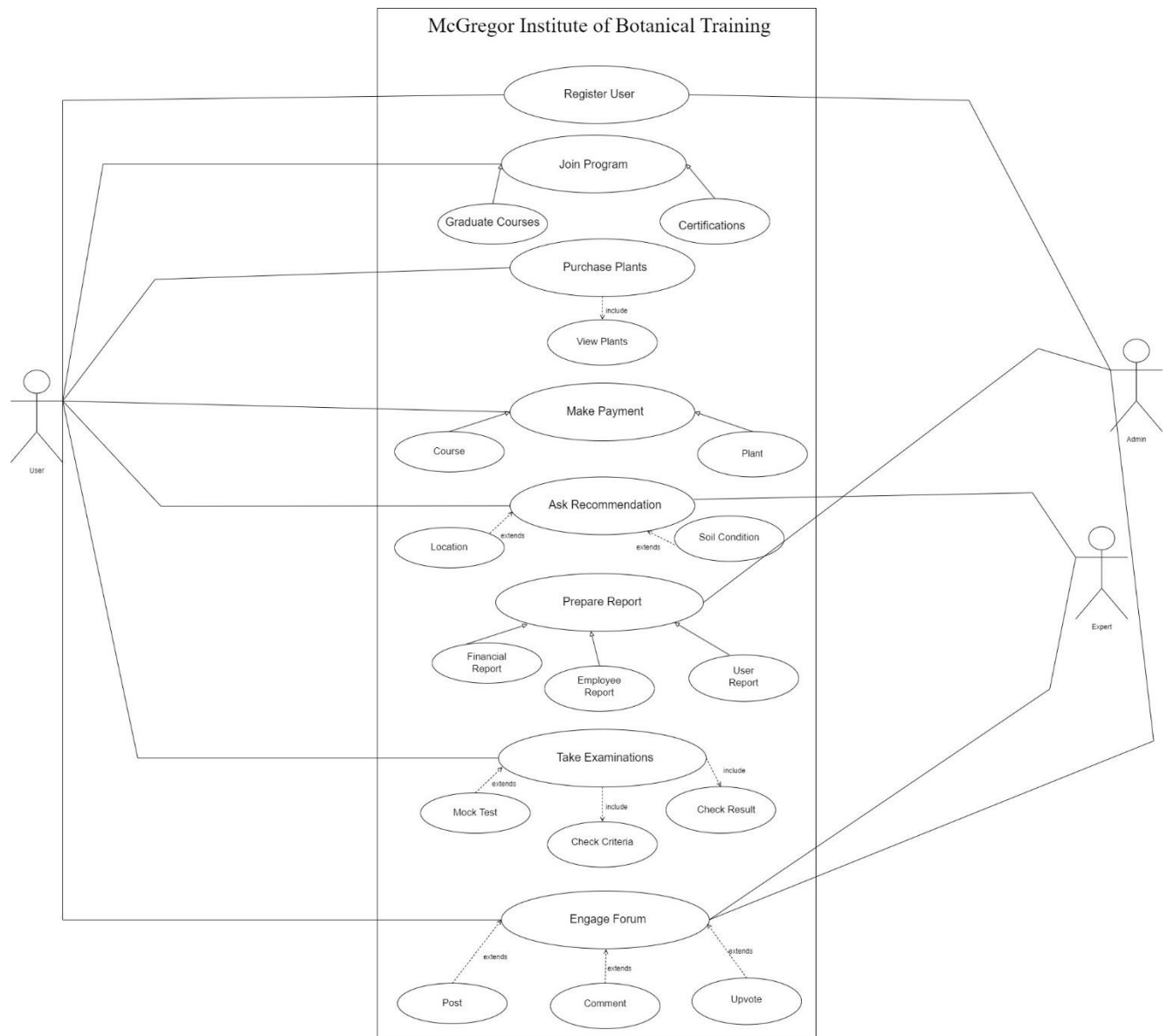


Figure 3: Use case diagram

## 4.2 High level use case description:

### 4.2.1 Register

<b>Use case:</b>	Register User
<b>Actor:</b>	User, Admin
<b>Description:</b>	Prospective users complete a registration form with necessary details. The system validates the information and establishes a new user account.

*Table 1: Use case of register user*

### 4.2.2 Join program

<b>Use case:</b>	Join Program
<b>Actor:</b>	User
<b>Description:</b>	Users navigate through the offered programs, choose their preferred ones, and proceed with enrollment. For paid courses, a payment process may also be initiated.

*Table 2: Use case of join program*

### Purchase plants

<b>Use case:</b>	Purchase Plant
<b>Actor:</b>	User
<b>Description:</b>	Users browse the plant catalog, viewing details such as availability status, species and prices. They then choose the plants they desire to purchase and proceed with the buying process.

*Table 3: Use case of purchase plant*

### 4.2.3 Make payment

<b>Use case:</b>	Make Payment
<b>Actor:</b>	User
<b>Description:</b>	<p>Users choose their payment option (e.g., plant, course). Then the user proceed the payment request request</p> <p>The system securely processes the transaction, verifying payment details and completing the financial exchange. Upon successful completion of payment, the system records transaction data in the user's account history for future reference and display's confirmation message. This ensures transparency and allows users to track their spending.</p>

Table 4: Use case of make payment

### 4.2.4 Ask recommendation

<b>Use case:</b>	Ask Recommendation
<b>Actor:</b>	User, Expert
<b>Description:</b>	<p>Users provide information such as their geographical location (pinning on map), soil condition images (if available), and specific requirements. The system directs the message to experts who analyze the data and suggest suitable plant options.</p>

Table 5: Use case of ask recommendation

### 4.2.5 Prepare report

<b>Use case:</b>	Prepare Report
<b>Actor:</b>	Admin

<b>Description:</b>	The admin accesses a reporting module within the system, selects the type of report needed, and defines the parameters (e.g., date range, user categories). The system generates the report based on the specified criteria and presents it to the admin for review or download.
---------------------	--

Table 6: Use case of prepare report

#### 4.2.6 Take examination

<b>Use case:</b>	Take Examination
<b>Actor:</b>	User
<b>Description:</b>	Users access the examination module from the system, choose the exam they want to take, and start the test. Upon completion, the system automatically grades the exam and provides immediate feedback to the user.

Table 7: Use case of take examination

#### 4.2.7 Engage forum

<b>Use case:</b>	Engage Forum
<b>Actor:</b>	User, Expert, Admin
<b>Description:</b>	Users browse the forum module of the system, accessing through the existing topics or create new ones, post as per the queries, comment and reply to others' posts, and engage in discussions. The system allows interaction and moderation to ensure a positive user experience.

Table 8: Use case of engage forum

### 4.3 Expanded Use Case Description:

#### 4.3.1 Register User

<b>Use case:</b>	Register User
<b>Actor:</b>	User, Admin
<b>Description:</b>	Prospective users complete a registration form with necessary details. The system validates the information and establishes a new user account.
<b>Typical course of events</b>	
<b>Actor Action</b>	<b>System Response</b>
1. The user registers the account with necessary details if the user is first timer and is yet to register	
	2. The system then verifies the details entered by the user and stores in the system.
3. The user is forwarded the login section where the user can login to the system with the registered account	
	4. The system verifies the login details ,upon verification the system pops the main menu and provides the user with courses and plant details.
<b>Alternative course of events:</b>	

<b>Number 1 and number 3:</b> If the details do not meet the conditions or is invalid, an error message is displayed by the system and asks the user to re-enter the details	
User Action	System Action
1. The user confirms the details that are correct or else asks the user to re-enter the details	
	2. The system verifies the registered details, upon verification proceeds to the registration gateway to store the data in the database.
3. The user gets a successful registration alert of the display.	
	4. Upon the registration confirmation, the system pops the login page where the user can login.
5. The user enter the details in the login page to move ahead with the application.	
	6. The system communication with the database to validate the login details as per the registration .upon validation opens the main menu with user's account.

Table 9:Expanded use case description of register user

### 4.3.2 Make payment

<b>Use case:</b>	Make Payment
<b>Actor:</b>	User
<b>Description:</b>	<p>Users choose their payment option (e.g., plant, course). Then the user proceed the payment request request</p> <p>The system securely processes the transaction, verifying payment details and completing the financial exchange. Upon successful completion of payment, the system records transaction data in the user's account history for future reference and display's confirmation message. This ensures transparency and allows users to track their spending.</p>
<b>Typical course of events</b>	
<b>Actor Action</b>	<b>System Response</b>
1. The user selects the plant they desire to buy or the course they desire to get enrolled in and move forwards to the payment section	
	2. The system shows the details of the selected product or the course with the amount they need to make payment.
3. The user selects the payment method upon their desire(wallet, bank) and move ahead to enter details.	

	4. The system process the encrypted payment transaction for safe payment.
5. The user confirms the payment by clicking the confirm button.	
	6. The system then process ahead the payment gateway ,verifies the payment details and makes the transaction and displays a confirmation.
7. The user receives a payment confirmation through email upon success of the payment	
	8. The system safes the payment history and provide the accesss to the product or the course
<b>Alternative course of events:</b> <b>Number 7:</b> If the details of the payment is invalid, an error message is displayed by the system and asks the user to re-enter the details	
1. The user makes a payment which happens to be invalid	
	2. The system shows a error message highlighting the payment is invalid or a error has occurred and forwards to alternative method of payment.
3. The invalid payment gets declined by the payment gateway due to several reasons.	



	4. The system alerts the user about the unsuccessful payment and provide suggestion to make payment again.
5. The user halts the payment and attempts to make make the payment again with correct details.	

*Table 10: Expanded use case description of make payment*

## 5 Sequence Diagram

In the field of software development, a Sequence Diagram is a robust tool that graphically illustrates the interactions and communication between different components or objects within a system over time. It displays the sequence of messages exchanged between objects or actors in a time-ordered manner, providing an in-depth view of the dynamic behavior and sequence of operations within a system.

In the context of the McGregor Institute of Botanical Training project, creating Sequence Diagrams is of great importance in clarifying the complex interactions and workflows that take place within the botanical training platform and community forum. These diagrams are invaluable resources for understanding the sequence of events initiated by user actions, system responses, and external triggers, thereby aiding the design, implementation, and testing of system functionalities.

To create Sequence Diagrams for this project, the project team starts by identifying the main use cases or scenarios that require detailed analysis. These use cases usually represent essential functionalities or user interactions within the system, such as user registration, course enrollment, or forum post submission.

Once the use cases are identified, the team outlines the sequence of steps or interactions involved in each scenario. This includes identifying the objects or actors involved, the messages exchanged between them, and the sequence in which these messages occur. Messages can represent method calls, data exchanges, or other forms of communication between objects.

The resulting Sequence Diagrams provide a step-by-step illustration of the flow of control and data within the system, capturing both the sequential order of interactions and any concurrent activities that occur simultaneously. These diagrams enable stakeholders to gain a comprehensive understanding of the system's behavior and help identify potential issues, such as communication errors or bottlenecks, early in the development process.

Sequence Diagrams play a crucial role in the McGregor Institute project by providing a detailed visualization of system interactions and workflows. By offering insights into the dynamic behavior of the system, these diagrams facilitate effective communication among project stakeholders, guide system design and implementation decisions, and ultimately contribute to the successful development and deployment of the botanical training platform and community forum.

A Sequence Diagram in software engineering represents the interactions between objects or components within a system over time. It consists of several key components that collectively illustrate the flow of messages exchanged between objects or actors. These components include:

- I. **Lifelines:** Lifelines represent the participating objects or components within the system. Each lifeline corresponds to an object or actor involved in the interaction and is depicted as a vertical dashed line extending downwards from the top of the diagram. The name of the object or actor is typically written at the top of the lifeline.
- II. **Messages:** Messages represent the communication or interaction between lifelines. They denote the passing of control or data from one object to another and can take different forms, such as method calls, signals, or data transmissions. Messages are depicted as horizontal arrows connecting lifelines, indicating the flow of communication between objects. They may also include labels indicating the type of message, such as synchronous (denoted by a solid arrow) or asynchronous (denoted by a dashed arrow).
- III. **Activation Boxes:** Activation boxes, also known as activation bars or activation rectangles, represent the period during which an object is active or executing a particular operation. They are depicted as horizontal bars extending from the lifeline and are positioned above the lifeline to indicate the duration of the object's activity. Activation boxes provide a visual representation of the sequence and duration of method calls or operations performed by the object.
- IV. **Return Messages:** Return messages indicate the response or result of a previously sent message. They represent the return of control or data from the

receiving object back to the sending object. Return messages are depicted as arrows extending from the receiving object back to the sending object and are typically labeled with the result of the operation or method call.

- V. **Fragments:** Fragments represent conditional or iterative behavior within a sequence diagram. They allow for the depiction of alternative paths or repeated sequences of interactions based on specific conditions. Fragments are depicted as rectangular boxes enclosing a series of messages and are annotated with conditions or loop indicators to indicate the circumstances under which they are executed.

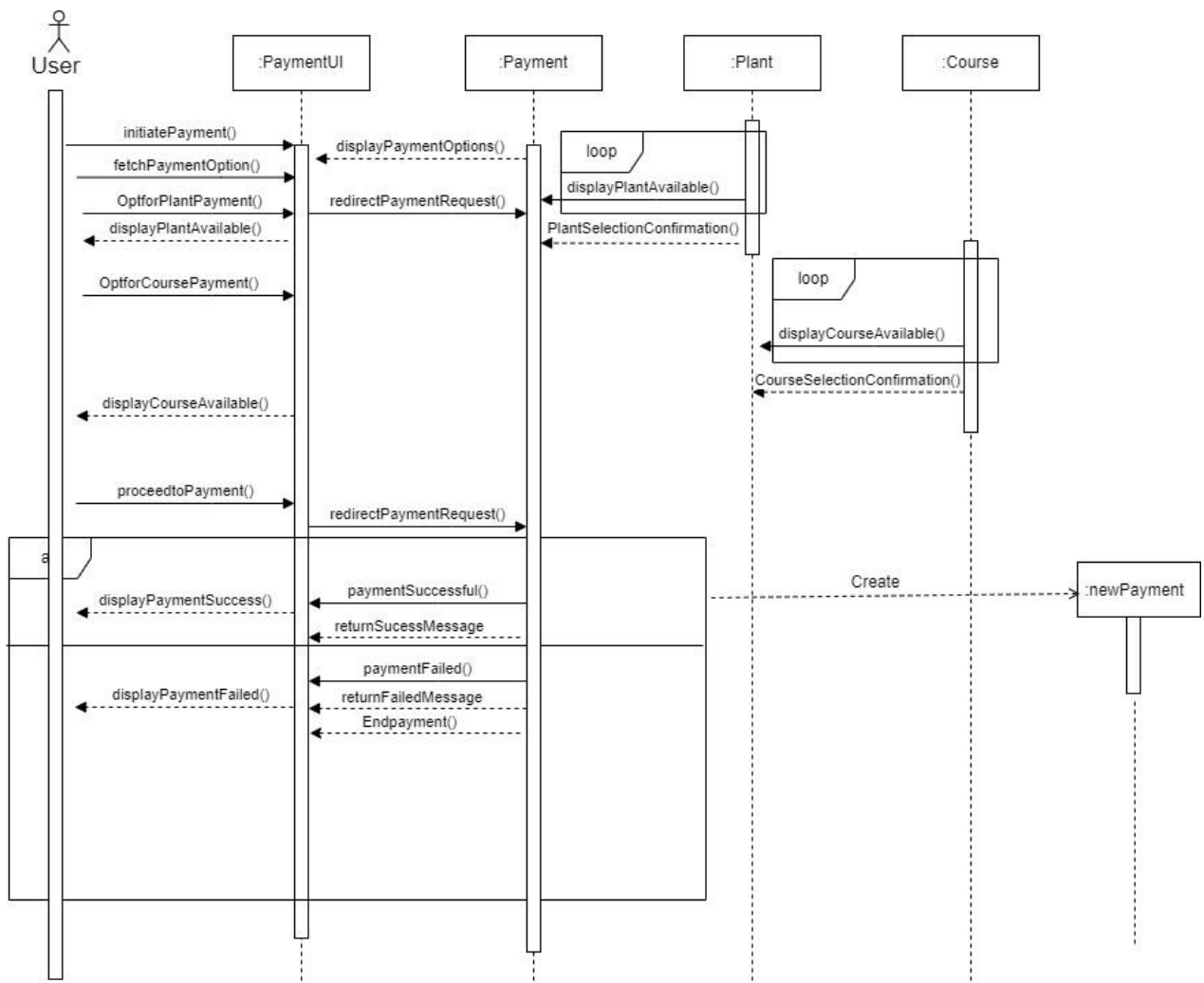


Figure 4: Sequence Diagram

## 6 Communication Diagram

In the domain of software development, a Communication Diagram, also known as a Collaboration Diagram, is an influential tool that graphically portrays the interactions and relationships between various components or objects within a system. Unlike Sequence Diagrams, which concentrate on the time sequence of message exchanges, Communication Diagrams highlight the structural arrangement of the system and the flow of messages between objects in a more abstract and static way.

For the McGregor Institute of Botanical Training project, creating Communication Diagrams is of immense importance in clarifying the structural relationships and communication patterns among the various components of the botanical training platform and community forum. These diagrams are invaluable resources for understanding the system's architecture, component interactions, and message flows, thereby aiding the design, implementation, and maintenance of the system.

To create Communication Diagrams for this project, the project team starts by identifying the main components or objects within the system and their respective relationships. These components might include modules, classes, subsystems, or external interfaces that play a role in the functionality of the system.

Once the components and their relationships are identified, the team outlines the flow of messages or interactions between them. This includes identifying the types of messages exchanged, the direction of message flows, and any dependencies or constraints that govern the communication between components.

The resulting Communication Diagrams provide a comprehensive view of the system's architecture and communication pathways, capturing both the structural organization of the system and the interactions between its components. These diagrams enable stakeholders to gain a thorough understanding of the system's design and behavior, facilitating effective communication, collaboration, and decision-making throughout the development lifecycle.

Communication Diagrams play a crucial role in the McGregor Institute project by providing a static representation of the system's structure and communication patterns. By offering insights into the relationships between system components and the flow of messages between them, these diagrams aid in system design, analysis, and documentation, ultimately contributing to the successful development and deployment of the botanical training platform and community forum.

In an MVC (Model-View-Controller) pattern, a Communication Diagram illustrates the interactions and relationships between the components of the system, namely the Model, View, and Controller. Each component plays a distinct role in the system architecture, and the Communication Diagram depicts how they collaborate to achieve the system's functionality. Here are the components typically found in a Communication Diagram designed using the MVC pattern:

- I. **Model:** The Model represents the application's data and business logic. It encapsulates the data access and manipulation operations, as well as the business rules governing the behavior of the system. In the Communication Diagram, the Model component is depicted as a class or set of classes responsible for managing data and performing operations on it.
- II. **View:** The View represents the presentation layer of the application. It is responsible for rendering the user interface and presenting data to the user in a visually appealing and understandable format. In the Communication Diagram, the View component is depicted as a class or set of classes responsible for displaying data and interacting with the user.
- III. **Controller:** The Controller serves as the intermediary between the Model and View components. It receives user input from the View, processes it by invoking appropriate operations on the Model, and updates the View with the results. In the Communication Diagram, the Controller component is depicted as a class or set of classes responsible for coordinating the interaction between the Model and View.

- IV. **Messages:** Messages represent the communication between the components of the system. In the MVC pattern, messages typically flow between the View, Controller, and Model components to facilitate user interactions and data processing. These messages may include user input events, data queries, data updates, and presentation commands. Messages are depicted as arrows connecting the components in the Communication Diagram, indicating the direction of communication and the flow of data or control.
- V. **Dependencies:** Dependencies represent the relationships between the components of the system. In the MVC pattern, the View depends on the Model for data access and retrieval, while the Controller depends on both the View and Model for coordinating user interactions and data processing. Dependencies are depicted as lines or arrows indicating the directional relationship between components in the Communication Diagram.

Below is a collaboration diagram depicting the communication diagram.

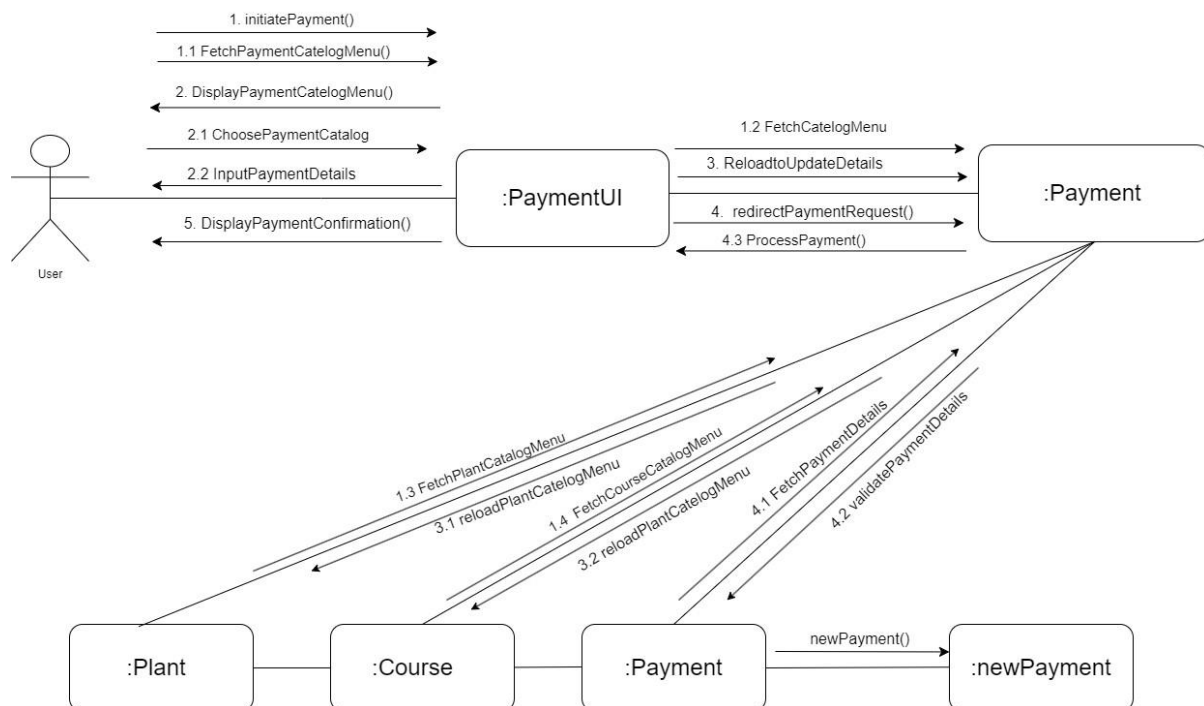


Figure 5: Communication Diagram

## 7 Class diagram

A class diagram is a type of static structure diagram in the Unified Modelling Language (UML) that visually depicts the structure and relationships of classes, interfaces, and other elements within a system. It provides a graphical representation of the system's object-oriented design, showcasing the classes in the system, their attributes, methods, and relationships with other classes.

For the McGregor Institute of Botanical Training project, the class diagram is created based on the use case diagram and other requirements collected during the analysis phase. Here's the process to derive it:

- I. **Class Identification:** Examine the actors and use cases from the use case diagram to pinpoint the primary entities or classes involved in the system. These might include entities like User, Course, Plant, Forum, and so on.
- II. **Attribute and Method Definition:** For each class identified, define its attributes (properties) and methods (behaviours) based on the requirements derived from the use cases. Attributes represent the data held by each class, while methods represent the operations that can be executed on the class.
- III. **Association Establishment:** Analyze the interactions between classes in the use case diagram to establish associations or relationships among them. Associations represent links between classes and can be one-to-one, one-to-many, or many-to-many. For instance, a User class may be associated with multiple Course classes if a user can enroll in multiple courses.
- IV. **Relationship Refinement:** Refine the associations between classes by specifying multiplicity (cardinality) and navigability. Multiplicity determines how many instances of one class are related to instances of another class, while navigability indicates whether one class can directly access another class.
- V. **Inheritance Incorporation:** If there are inheritance relationships between classes, such as subclasses inheriting from a superclass, depict them in the class diagram.



using generalization relationships. This mirrors the hierarchical structure of the system's class hierarchy.

By adhering to these steps, a class diagram is generated that effectively encapsulates the structure and relationships of classes within the McGregor Institute of Botanical Training system, derived from the use case diagram and other requirements. This class diagram acts as a blueprint for the system's implementation, guiding developers in the creation of classes, interfaces, and relationships during the design and development stages of the project.

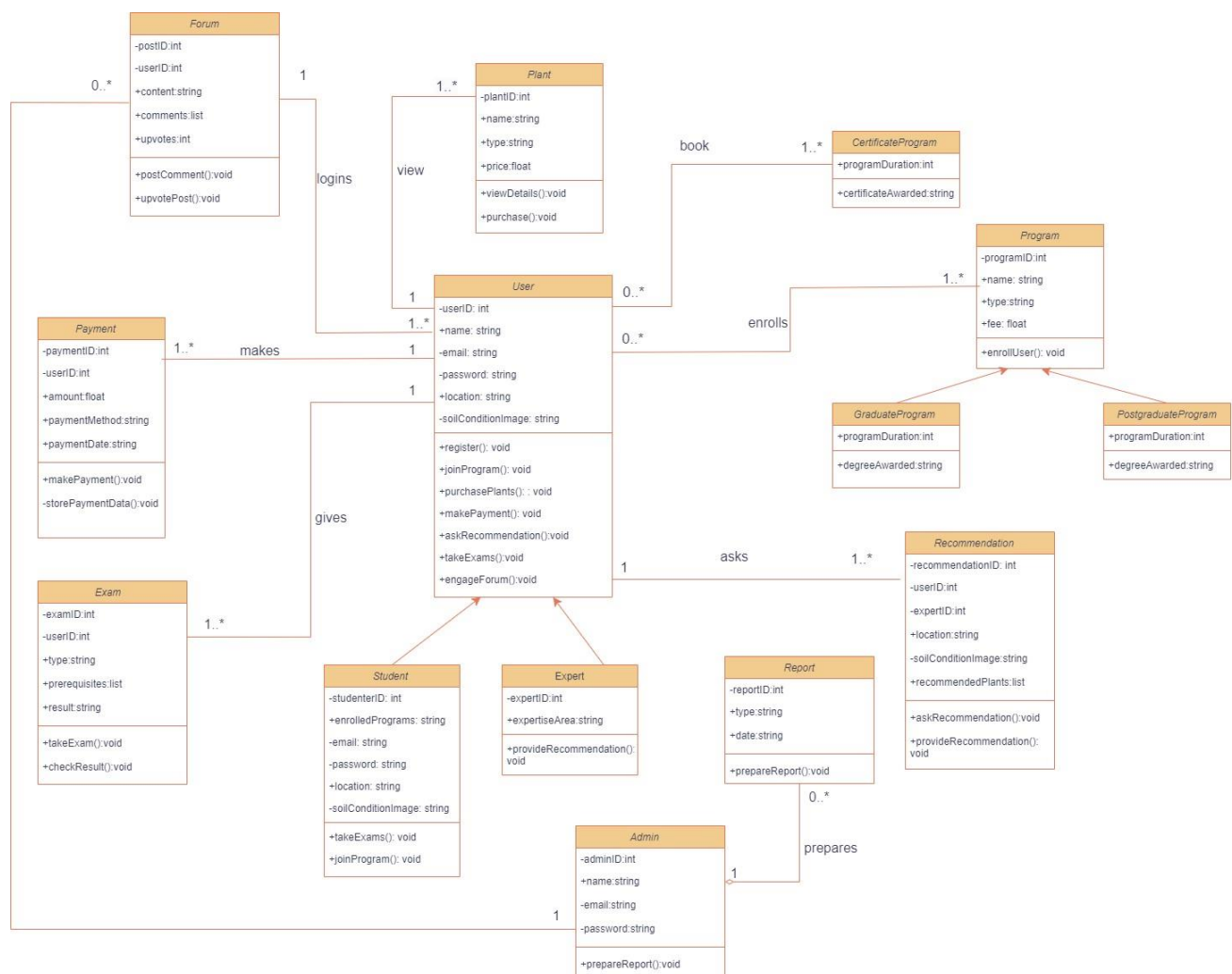


Figure 6: Class Diagram

## 8 Further development

After creating the analysis and design diagrams, we need to adopt a suitable methodology to structure the project into manageable phases, including testing and maintenance. Various methodologies can facilitate this process, such as Agile, Waterfall, and DevOps. These methodologies enable us to divide the project into distinct phases and ensure effective time management throughout development, testing, and maintenance stages. For this project, Agile methodology has been selected as the guiding framework for our development process. Agile is renowned for its iterative and incremental approach, which fosters flexibility, adaptability, and continuous improvement throughout the project lifecycle.

In Agile, the development process is structured into short iterations known as sprints, typically lasting from one to four weeks. Each sprint aims to deliver a potentially shippable product increment, enabling rapid feedback and adaptation to evolving requirements. Agile methodologies prioritize collaboration, communication, and customer satisfaction. Cross-functional teams work closely together, focusing on delivering value to the customer through incremental development and frequent delivery of working software.

## 8.1 Architectural choice

### I. . Layered Architecture Pattern:

Layered architecture, also referred to as n-tier architecture, divides a system into multiple layers, each tasked with specific functionalities. These layers are structured hierarchically, with higher layers dependent on lower ones. Common layers include presentation, business logic, and data access layers.

### II. . Event-Driven Architecture Pattern:

Event-driven architecture (EDA) operates on the principle of events, where components communicate through event generation and consumption. Events signify changes in state or significant actions within the system. Components can respond to events asynchronously, decoupling them from event sources. Example: In a messaging system, the receipt of a new message triggers an event. Subscribers to this event can process the message asynchronously, such as sending notifications or updating dashboards.

### III. Microkernel Architecture Pattern:

Microkernel architecture, also known as plug-in architecture, divides the system into a core (microkernel) providing essential services and optional modules or plug-ins extending its functionality. The microkernel offers minimal services, while plug-ins provide additional features. Example: In a content management system, the core system provides basic features like user authentication and content storage, while plug-ins offer additional functionalities such as SEO optimization or social media integration.

### 4. Microservices Architecture Pattern:

Microservices architecture decomposes a system into small, independently deployable services, each focusing on a specific business capability. Services are developed,

deployed, and scaled independently, communicating via lightweight protocols like HTTP or messaging queues.

Example: In an e-commerce platform, separate services manage user authentication, product catalog management, order processing, and payment processing, each with its database and deployment infrastructure.

These architecture patterns offer diverse approaches to designing software systems, each with its advantages and use cases. By understanding their characteristics and considerations, developers can select the most suitable pattern for specific requirements and constraints.

## 8.2 Design pattern

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It provides a template or guideline for solving design issues, promoting best practices, and facilitating communication among developers.

Design patterns are needed to address recurring problems and challenges in software development, such as managing complexity, promoting scalability, enhancing maintainability, and improving code quality. They provide proven solutions and guidelines that developers can apply to design and implement software systems effectively.

### Importance of Design Patterns:

Design patterns offer several benefits, including:

- **Reusability:** Design patterns encapsulate solutions to common problems, making them reusable across different projects and contexts.
- **Scalability:** Design patterns promote scalable design practices, allowing software systems to accommodate growth and changes over time.
- **Maintainability:** By promoting clear and modular design principles, design patterns enhance the maintainability and readability of code, reducing the likelihood of errors and facilitating future enhancements.
- **Standardization:** Design patterns provide a common vocabulary and set of guidelines for developers, promoting consistency and standardization in software development practices.

### Types of Design Patterns:

Design patterns are typically categorized into three main types:

- **Creational Patterns:** These patterns focus on object creation mechanisms, providing flexible ways to instantiate objects while hiding the creation logic from the client.

- **Structural Patterns:** Structural patterns deal with the composition of classes and objects, defining relationships and interactions between them to form larger structures.
- **Behavioral Patterns:** Behavioral patterns focus on the communication and interaction between objects, defining how objects collaborate to accomplish tasks and responsibilities.

### **Factory Pattern:**

The Factory Pattern is a creational design pattern that provides an interface for creating objects without specifying their concrete classes. It defines an interface or abstract class for creating objects and allows subclasses or implementing classes to alter the type of objects that will be created. The Factory Pattern promotes loose coupling between client code and the created objects, enabling flexibility and extensibility in object creation.

### **Advantages:**

- I. **Flexibility:** The Factory Pattern allows for the creation of objects without specifying their concrete classes, enabling the system to adapt to changing requirements and variations in object creation.
- II. **Encapsulation:** Object creation logic is encapsulated within the factory class or method, abstracting the client code from the complexities of object instantiation.
- III. **Code Reusability:** Factories can be reused across multiple parts of the system, promoting code reuse and minimizing redundancy in object creation logic.
- IV. **Centralized Configuration:** Factories provide a centralized location for configuring and managing object creation, making it easier to maintain and modify the creation process.

### **Disadvantages:**

- I. **Complexity:** Introducing factories can add complexity to the codebase, especially for small-scale projects or simple object creation scenarios.

- II. **Abstraction Overhead:** The use of abstract interfaces or base classes for object creation may introduce additional overhead and complexity in the code.
- III. **Increased Coupling:** While the Factory Pattern promotes loose coupling between client code and created objects, it may increase coupling between client code and the factory itself.

### Singleton Pattern:

The Singleton Pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. It involves defining a class with a method that creates or returns the same instance of the class every time it is invoked. The Singleton Pattern is commonly used in scenarios where a single, shared instance of a class is required across the system, such as logging, database connections, and configuration management.

### Advantages:

- I. **Single Instance:** The Singleton Pattern guarantees that only one instance of the class exists throughout the application, ensuring global access and consistency.
- II. **Lazy Initialization:** Singleton instances are typically lazily initialized, meaning they are created only when they are first requested, optimizing memory usage and startup time.
- III. **Global Access:** Singleton instances provide a global point of access, allowing any part of the system to access and utilize the shared instance without needing to instantiate it explicitly.
- IV. **Thread Safety:** Properly implemented Singleton patterns can ensure thread safety, preventing concurrent creation of multiple instances in multi-threaded environments.

### Disadvantages:

- I. **Global State:** Singleton instances introduce global state into the application, which can make it harder to maintain and test, as changes to the singleton instance can impact the behavior of other parts of the system.
- II. **Dependency Management:** Singleton instances may create tight coupling between classes that depend on them, making it challenging to replace or mock the singleton instance for testing purposes.
- III. **Overuse:** Overuse of the Singleton Pattern can lead to inflexible and tightly coupled designs, inhibiting scalability and maintainability of the codebase.

### **MVC Pattern:**

The Model-View-Controller (MVC) Pattern is a software architectural pattern that separates an application into three interconnected components: Model, View, and Controller. The Model represents the application's data and business logic, the View represents the presentation layer or user interface, and the Controller acts as an intermediary that handles user input, updates the model, and manipulates the view.

### **Advantages:**

- I. **Separation of Concerns:** MVC promotes a clear separation of concerns, with each component responsible for a distinct aspect of the application (data, presentation, and control logic), making the codebase more modular, maintainable, and scalable.
- II. **Code Reusability:** By separating the presentation layer (View) from the business logic (Model) and user input handling (Controller), MVC facilitates code reuse and modifiability, as changes to one component do not necessarily impact the others.
- III. **Testability:** MVC improves the testability of the application by decoupling the components, allowing for easier unit testing, integration testing, and functional testing of each component in isolation.
- IV. **Support for Parallel Development:** MVC enables parallel development of different components by different teams or developers, as long as they adhere to the defined interfaces and communication protocols between components.



**Disadvantages:**

- I. **Complexity:** MVC introduces additional complexity compared to simpler architectural patterns, especially for small-scale or straightforward applications.
- II. **Learning Curve:** Developers may require time and effort to understand and implement MVC effectively, particularly if they are unfamiliar with the pattern or its principles.
- III. **Overhead:** MVC may introduce overhead in terms of communication between components, especially in scenarios where extensive coordination and data passing are required between the Model, View, and Controller.

**Data Access Object (DAO) Pattern:**

The Data Access Object (DAO) Pattern is a structural design pattern that provides an abstract interface for accessing and managing data from a data source, such as a database, file system, or external API. It separates the data access logic from the business logic of the application, encapsulating CRUD (Create, Read, Update, Delete) operations within dedicated DAO classes or interfaces.

**Advantages:**

- I. **Separation of Concerns:** DAO separates data access logic from business logic, promoting a modular and maintainable architecture by isolating database-specific operations.
- II. **Abstraction:** DAO provides a level of abstraction over data access operations, allowing the application to interact with data sources through a consistent interface, regardless of the underlying implementation details.
- III. **Flexibility:** DAO facilitates easy switching or replacement of data sources without affecting the rest of the application, as long as the DAO interface remains unchanged.

- IV. **Testability:** DAO classes can be easily unit tested by mocking or stubbing the data access layer, enabling comprehensive testing of data retrieval, manipulation, and persistence operations.

**Disadvantages:**

- I. **Complexity:** Implementing DAO patterns may introduce additional complexity, especially in applications with complex data access requirements or multiple data sources.
- II. **Performance Overhead:** DAO patterns may incur performance overhead, especially in scenarios where data access operations involve complex queries, transactions, or remote calls.
- III. **Code Duplication:** In some cases, implementing DAO patterns may lead to code duplication, especially if similar data access logic needs to be replicated across multiple DAO classes or interfaces.
- IV. **Maintenance Overhead:** Managing and maintaining a separate layer of DAO classes/interfaces may introduce overhead in terms of codebase maintenance, versioning, and documentation.

**Application of Design Patterns at McGregor Institute of Botanical Training:**

- **Factory Pattern:** The Factory Pattern could be employed in the creation of different types of courses offered by the institute, allowing for flexible instantiation of course objects based on user preferences or enrollment criteria.
- **Singleton Pattern:** The Singleton Pattern could be utilized in managing shared resources such as the database connection pool or configuration settings, ensuring that only one instance of these resources exists throughout the application.
- **MVC Pattern:** The MVC Pattern could be adopted to structure the institute's e-learning platform, with the Model representing course data and business logic, the View handling user interfaces for course enrollment and content viewing, and the Controller managing user interactions and course enrollment processes.

- **DAO Pattern:** The DAO Pattern could be implemented to encaps

For McGregor Institute of Botanical Training's system, which involves managing various functionalities such as user registration, course enrollment, plant purchases, payment processing, recommendation requests, report generation, certification exams, and forum interactions, the best design pattern would be the Model-View-Controller (MVC) pattern.

**Reason for Choosing the MVC Pattern:**

- The MVC pattern offers several advantages that align well with the requirements of McGregor Institute of Botanical Training's system:
  - **Separation of Concerns:** MVC separates the application into three distinct components (Model, View, Controller), promoting a clear division of responsibilities and enhancing maintainability.
  - **Modularity and Reusability:** MVC facilitates modularity, allowing different components to be developed, tested, and modified independently without impacting others, promoting code reusability and flexibility.
  - **Scalability and Flexibility:** MVC supports scalable design practices, enabling the system to accommodate growth and changes over time. New features or changes can be implemented and deployed independently, allowing for flexibility in development and adaptation to changing requirements.
  - **Ease of Testing:** MVC simplifies the testing process by allowing each component to be tested independently, facilitating thorough validation of business logic and UI interactions.

By adopting the MVC pattern, McGregor Institute of Botanical Training can develop a robust, maintainable, and scalable system that meets the needs of its users effectively while facilitating future growth and innovation.

## 9 Development Plan

The development plan serves as a strategic roadmap outlining the tools, resources, and prioritized tasks necessary for the successful implementation of McGregor Institute of Botanical Training's system. It provides a structured approach to project management, guiding the development team through various stages of software development, from initial setup and backend development to frontend design, e-commerce functionality, recommendation systems, certification exams, and reporting mechanisms.

- **Integrated Development Environment (IDE):** JetBrains IntelliJ IDEA or Eclipse for Java development, offering robust features for code editing, debugging, and version control integration.
- **Programming Languages:** Java for backend development, known for its stability, scalability, and extensive ecosystem of libraries and frameworks.
- **Frameworks and Libraries:** Spring Boot for backend microservices, providing rapid application development, dependency injection, and integration with other Spring modules. React.js for frontend development, offering component-based architecture and efficient rendering for dynamic user interfaces. Bootstrap for UI design, ensuring responsive and mobile-friendly layouts. Hibernate for database ORM, simplifying database interactions and reducing boilerplate code.
- **Database:** MySQL or PostgreSQL for data storage and management, offering relational database capabilities and robust transaction support.
- **Version Control:** Git for version control and collaboration, enabling efficient code management, branching, and merging workflows.
- **Project Management:** JIRA or Trello for project tracking and task management, facilitating agile project management methodologies such as Scrum or Kanban.
- **Continuous Integration and Deployment (CI/CD):** Jenkins or Travis CI for automated build and deployment processes, ensuring code quality, consistency, and reliability across development, staging, and production environments.

- **Documentation:** Confluence or GitHub Wiki for documentation and knowledge sharing, providing centralized repositories for project documentation, technical specifications, and user guides.

## 10 Testing Plan

Testing is a crucial process in software development that involves evaluating a system or application to identify defects, errors, or discrepancies between expected and actual behavior. It aims to ensure that the software meets specified requirements, functions correctly, and delivers a positive user experience. Testing plays a critical role in software development for several reasons:

- **Error Detection:** Testing helps identify and rectify defects and errors early in the development process, reducing the cost and effort required for fixing them later.
- **Quality Assurance:** Through rigorous testing, software quality is assured, ensuring that the product meets specified requirements, standards, and user expectations.
- **Risk Mitigation:** Testing mitigates risks associated with software failures, security vulnerabilities, performance issues, and compliance violations, safeguarding stakeholders' interests.
- **Continuous Improvement:** Testing provides valuable feedback for iterative improvement, enabling developers to refine, optimize, and enhance the software over time, thereby ensuring its longevity and competitiveness in the market.

There are various kind of testing some of them are mentioned below:

### Unit Testing:

- **Definition:** Unit testing involves the examination of individual components or units of code in isolation, with the objective of verifying their correctness and functionality.
- **Advantages:**
  - Enables early detection of defects and errors, promoting rapid debugging and resolution.
  - Facilitates code modularity, reusability, and maintainability by isolating and testing specific functionalities or logic units.

- Supports automated testing practices, enhancing efficiency, repeatability, and scalability of testing efforts.
- **Disadvantages:**
  - Limited in scope, may fail to identify integration issues or dependencies between units.
  - Requires comprehensive coverage of codebase, necessitating significant time and effort investment.

#### **b. Integration Testing:**

- **Definition:** Integration testing evaluates the interactions and interfaces between integrated components or modules to ensure seamless interoperability and functionality within the overall system architecture.
- **Advantages:**
  - Validates correct communication, data exchange, and coordination between integrated modules or subsystems.
  - Identifies integration issues, compatibility conflicts, and data inconsistencies, facilitating early resolution and system stabilization.
  - Supports incremental development and testing approaches, ensuring early integration and validation of evolving system components.
- **Disadvantages:**
  - Complexity escalates with the number of integrated components, leading to increased testing overhead and resource requirements.
  - Dependencies on external systems, APIs, or third-party services may introduce challenges in testing and environment setup.

#### **c. System Testing:**

- **Definition:** System testing evaluates the end-to-end functionality, behavior, and performance of the entire software system within its intended environment, encompassing various functional and non-functional aspects.
- **Advantages:**
  - Validates system behavior under real-world scenarios, ensuring all system functionalities work cohesively to meet user requirements.
  - Verifies compliance with functional, performance, security, and regulatory requirements, ensuring software quality and reliability.
  - Identifies system-level defects, performance bottlenecks, and usability issues, enabling timely corrective action and refinement.
- **Disadvantages:**
  - Requires comprehensive test coverage and realistic test scenarios, necessitating thorough planning, execution, and analysis.
  - May not uncover all defects in complex systems with intricate interactions and dependencies, requiring supplementary testing approaches.

### **Black Box Testing:**

- **Definition:** Black box testing is a software testing technique that focuses on assessing the functionality of a system without examining its internal code structure, implementation details, or logic. Testers interact with the system's inputs and observe its outputs to validate its behavior against specified requirements and expected outcomes.
- **Advantages:**
  - Simulates real-world user interactions and scenarios, ensuring that the software behaves as expected from an end-user perspective.
  - Promotes independence between testers and developers, as testers do not require knowledge of the system's internal workings to design and execute tests.



- Facilitates testing from the user's standpoint, identifying usability issues, workflow discrepancies, and functional defects.
- **Disadvantages:**
  - Limited visibility into internal code logic may make it challenging to pinpoint the root cause of defects or failures.
  - Test coverage may be incomplete, as testers may overlook certain paths or conditions within the software.

### **White Box Testing:**

- **Definition:** White box testing, also known as clear box testing or structural testing, involves examining the internal code structure, logic, and implementation details of a software system. Testers have access to the source code and use this knowledge to design test cases that exercise specific paths, conditions, and branches within the code.
- **Advantages:**
  - Provides insight into code coverage, execution paths, and code quality metrics, enabling thorough testing of all code segments and logic branches.
  - Facilitates early detection of coding errors, boundary conditions, and performance bottlenecks, leading to improved code reliability and maintainability.
  - Supports techniques such as code coverage analysis, branch coverage, and path coverage, ensuring comprehensive testing of all code paths.
- **Disadvantages:**
  - Requires in-depth knowledge of programming languages, algorithms, and software design principles, making it dependent on the expertise of testers.
  - May be time-consuming and resource-intensive, especially for large, complex software systems with extensive codebases.

**Grey Box Testing:**

- **Definition:** Grey box testing combines elements of both black box and white box testing approaches. Testers have partial access to the internal structure and design of the software, allowing them to design test cases based on a combination of functional specifications and knowledge of the underlying code.
- **Advantages:**
  - Combines the benefits of black box and white box testing, leveraging both functional and structural testing techniques to ensure thorough coverage and validation.
  - Provides a balanced approach to testing, allowing testers to focus on critical functionality while also assessing code quality, execution paths, and integration points.
  - Enables testers to identify defects and vulnerabilities that may arise due to interactions between software components or modules.
- **Disadvantages:**
  - Requires coordination and collaboration between testers and developers to obtain access to relevant code segments and design specifications.
  - May introduce complexity and overhead, as testers need to strike a balance between functional testing objectives and code-centric validation activities.

McGregor Institute of Botanical Training will adopt a multifaceted testing approach, encompassing black box, white box, and grey box testing methodologies to ensure comprehensive coverage and validation of its software system. Black box testing will be utilized to evaluate the system's functionality from an end-user perspective, simulating real-world scenarios and interactions to validate user workflows, inputs, and outputs. White box testing will be employed to scrutinize the internal code structure and logic, examining individual components and algorithms to identify coding errors, boundary conditions, and performance bottlenecks. Grey box testing will serve as a hybrid approach, leveraging insights from both functional specifications and code knowledge to

design test cases and scenarios that target critical functionality, integration points, and system interactions. By integrating these testing methodologies, McGregor Institute of Botanical Training can ensure the reliability, functionality, and quality of its software system, delivering a seamless user experience and achieving organizational objectives in botanical education and community engagement.

## 11 Maintenance Plan

Maintenance is a vital stage in the software development lifecycle, aimed at preserving the ongoing functionality, dependability, and efficiency of the software product. The McGregor Institute of Botanical Training project, like any other software project, necessitates a thorough maintenance plan to address problems, implement updates, and improve the platform over time. The following is a description of the maintenance plan for the McGregor Institute project, which includes strategies, processes, and resources for continuous maintenance activities.

### I. **Corrective Maintenance:**

- Description: This involves identifying and rectifying defects, errors, or issues in the software after it has been deployed. This includes troubleshooting user-reported problems, resolving bugs, and addressing system failures.
- Reasoning: Corrective maintenance is crucial to ensure the McGregor Institute platform's stability and reliability. By quickly addressing issues and defects, we can minimize disruptions to users and maintain a positive user experience.

### II. **Adaptive Maintenance:**

- Description: This involves making changes to the software to adapt it to changing requirements, technologies, or environments. This includes updating software components, integrating new features, and modifying existing functionalities.
- Reasoning: Adaptive maintenance is critical for the McGregor Institute project to keep up with changing user needs, technological advancements, and market trends. By continuously adapting the platform, we can ensure its long-term relevance and competitiveness.

### III. **Perfective Maintenance:**

- Description: This focuses on enhancing and optimizing the software to improve its performance, usability, and efficiency. This includes optimizing

algorithms, refining user interfaces, and adding new features to enhance functionality.

- Reasoning: Perfective maintenance is essential for the McGregor Institute platform to evolve and grow over time. By continuously improving the platform's capabilities and user experience, we can attract new users, retain existing users, and stay ahead of competitors.

#### **IV. Preventive Maintenance:**

- Description: This involves proactively identifying and addressing potential issues or risks before they impact the software. This includes performing regular audits, conducting security assessments, and implementing preventive measures.
- Reasoning: Preventive maintenance is critical for ensuring the security, stability, and reliability of the McGregor Institute platform. By proactively identifying and mitigating risks, we can minimize the likelihood of security breaches, system failures, and other issues that could disrupt operations.

The maintenance plan for the McGregor Institute of Botanical Training Project includes:

##### **I. Corrective Maintenance:**

- Description: This involves identifying and addressing defects, errors, or issues in the McGregor Institute platform after deployment. This includes troubleshooting reported problems, fixing bugs, and resolving system failures to ensure the platform's stability and reliability.
- Implementation: Implement a structured process for logging and tracking reported issues using a dedicated issue tracking system. Assign responsible individuals or teams to investigate and resolve reported issues promptly. Prioritize issues based on severity, impact on users, and business-criticality.

##### **II. Adaptive Maintenance:**

- **Description:** This focuses on making changes to the McGregor Institute platform to adapt it to evolving requirements, technologies, or environments. This includes updating software components, integrating new features, and modifying existing functionalities to meet changing user needs.
- **Implementation:** Establish a change management process to review, approve, and implement proposed changes to the platform. Conduct impact assessments to evaluate the implications of proposed changes on system functionality, performance, and security. Leverage agile development practices to iteratively introduce enhancements and updates based on user feedback and market trends.

### **III. Perfective Maintenance:**

- **Description:** This aims to enhance and optimize the McGregor Institute platform to improve its performance, usability, and efficiency. This includes optimizing algorithms, refining user interfaces, and adding new features to enhance functionality and user experience.
- **Implementation:** Regularly solicit user feedback and suggestions through feedback forms, surveys, and community forums to identify areas for improvement. Prioritize enhancement initiatives based on user feedback, business objectives, and strategic priorities. Adopt a user-centered design approach to iteratively refine and enhance the platform's capabilities over time.

### **IV. Preventive Maintenance:**

- **Description:** This involves proactively identifying and addressing potential issues or risks before they impact the McGregor Institute platform. This includes conducting regular audits, security assessments, and performance reviews to mitigate risks and ensure system stability and reliability.

- **Implementation:** Implement proactive monitoring and alerting mechanisms to detect anomalies, performance degradation, or security threats in real-time. Conduct regular security audits and vulnerability assessments to identify and remediate potential security vulnerabilities or weaknesses. Establish backup and disaster recovery procedures to minimize downtime and data loss in the event of system failures or emergencies.

Maintenance Processes include:

- I. **Issue Tracking and Resolution:** Utilize a centralized issue tracking system to log, prioritize, and track reported issues and defects. Assign dedicated resources to investigate and resolve reported issues in a timely manner, adhering to predefined service level agreements (SLAs). Conduct thorough testing and validation of fixes before deployment to ensure quality and reliability.
- II. **Change Management:** Establish a formal change management process to review, approve, and implement proposed changes to the platform. Conduct impact assessments to evaluate the potential consequences of proposed changes on system functionality, performance, and security. Implement version control and release management practices to track changes and ensure version consistency across environments.
- III. **User Feedback and Continuous Improvement:** Foster a culture of continuous improvement by actively soliciting user feedback, suggestions, and enhancement requests. Regularly review and prioritize user feedback to identify opportunities for improvement and prioritize enhancement initiatives accordingly. Leverage agile development methodologies to iteratively introduce updates and enhancements based on user feedback and stakeholder input.

**Resource Allocation:** Allocate appropriate resources, including personnel, time, and budget, to support ongoing maintenance activities. This includes dedicated maintenance teams, skilled developers, testing resources, and access to necessary tools and technologies.

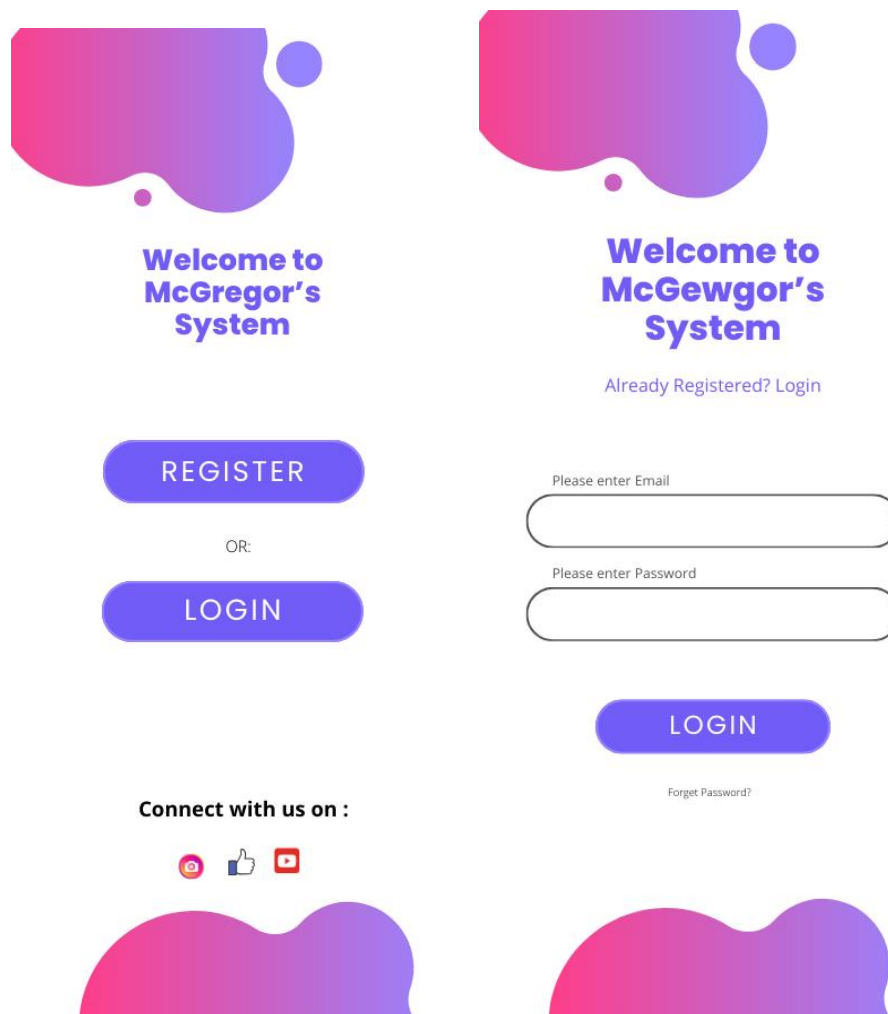
The maintenance plan outlined above provides a structured approach for sustaining the McGregor Institute of Botanical Training platform over the long term. By implementing processes and strategies for corrective, adaptive, perfective, and preventive maintenance, we can ensure the continued functionality, reliability, and effectiveness of the platform, meeting the evolving needs of users and stakeholders. Regular monitoring, proactive risk management, and continuous improvement efforts will contribute to the overall success and sustainability.



## 12 Prototype deployment

Creating prototypes is a pivotal stage in software development, offering a concrete depiction of how a proposed system's user interface and functionalities will operate. These prototypes act as visual guides, allowing stakeholders to perceive and engage with the envisioned features and workflows of the software product. By crafting prototypes, designers and developers can explore design concepts, validate requirements, and solicit early feedback from stakeholders. Prototypes vary in complexity, ranging from basic wireframes to fully interactive mock-ups. To construct design prototypes for the McGregor Institute of Botanical Training project, we will generate visual representations of the proposed system's user interface, focusing on key features outlined in the detailed specification.

### 12.1 Register Prototype



The image displays a user interface prototype for a registration system. It features a light blue background with decorative wavy shapes in shades of blue and green. The main heading is "Welcome to McGregor's System" in a bold, dark blue font. Below this, there is a link "Already Registered? Login" in a smaller, lighter blue font. The registration process is divided into two main sections. The first section has a "REGISTER" button in a rounded rectangle. Below it, the text "OR:" is centered. The second section has a "LOGIN" button in a rounded rectangle. To the right of these buttons, there are two input fields: "Please enter Email" and "Please enter Password", each with a corresponding label above it. Below the "LOGIN" button, there is a link "Forgot Password?". At the bottom, there is a section titled "Connect with us on :" followed by three social media icons (Facebook, Twitter, and YouTube). The entire prototype is framed by a light blue border.

Welcome to McGregor's System

Already Registered? Login

REGISTER

OR:

LOGIN

Please enter Email


Please enter Password

LOGIN

Forgot Password?

Connect with us on :

Facebook Twitter YouTube



## Register


[Already Registered? Login](#)

Please enter your name

Please enter Email

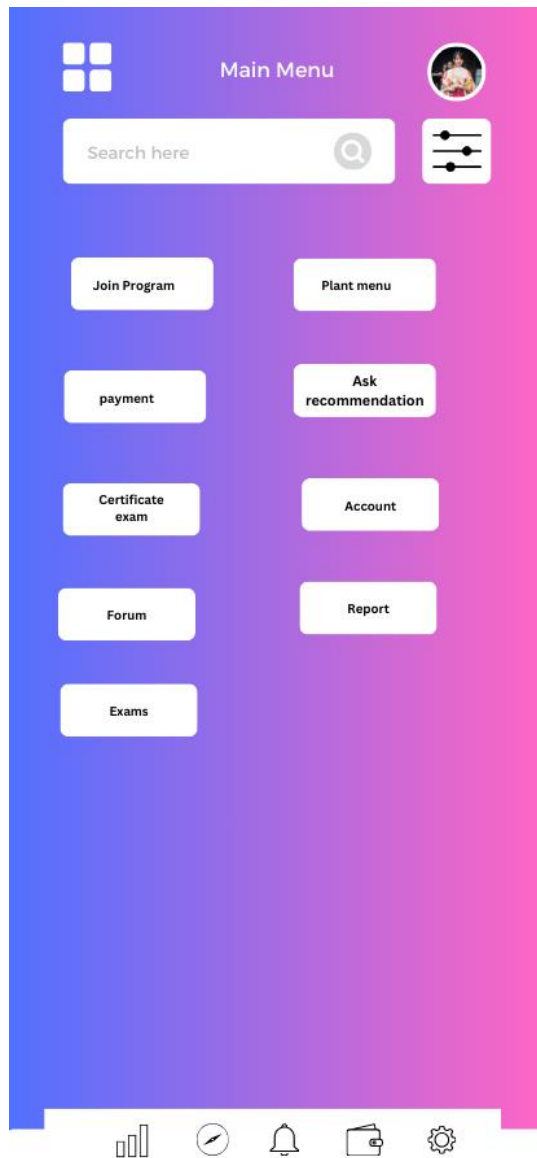
Please enter Password

REGISTER



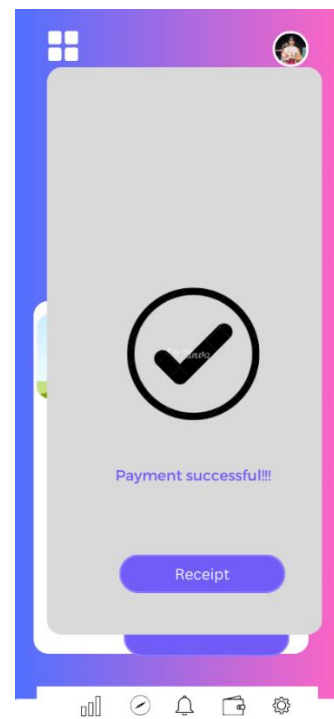
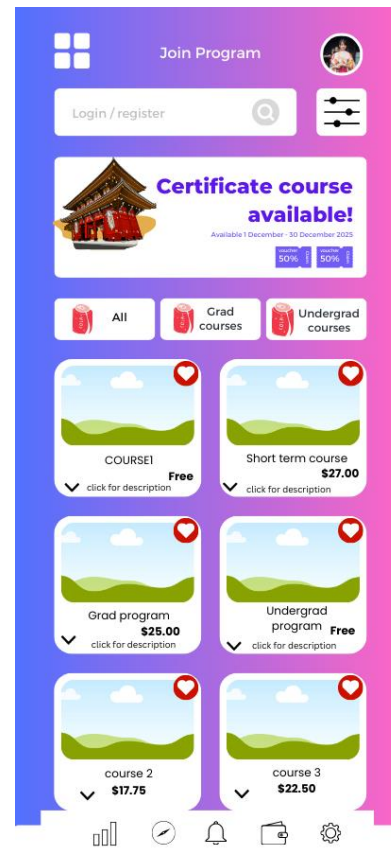
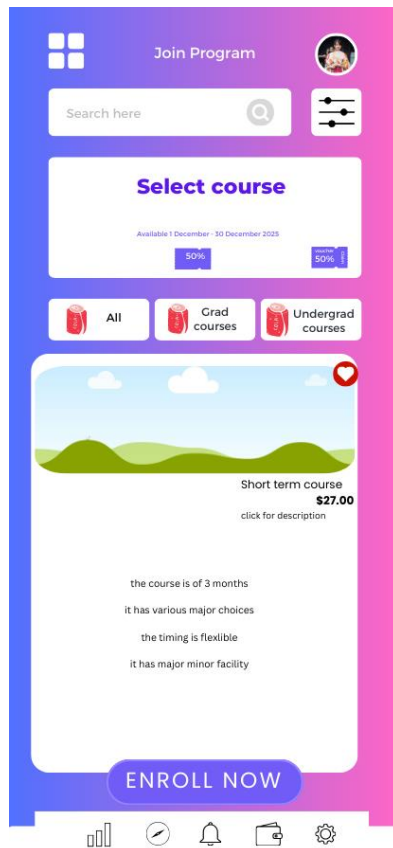
The prototype highlights the register section . Then if the user is already registered the user and directly login and move ahead. The application also features a feature to reset password.

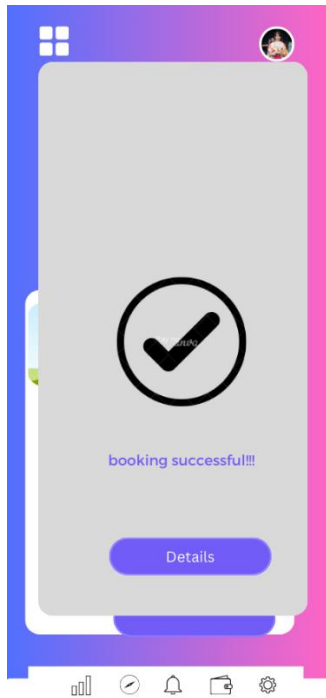
## 12.2 Main menu



The prototype shows the main menu from where the user can access to different option.

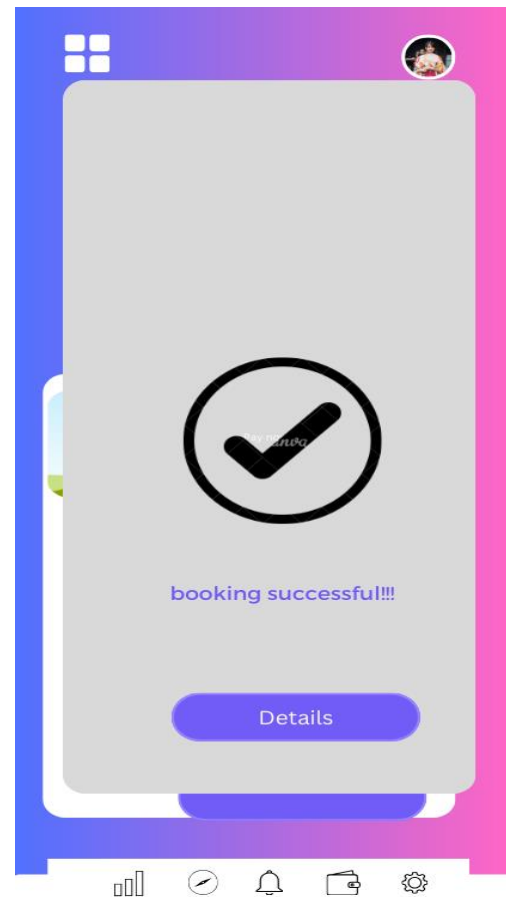
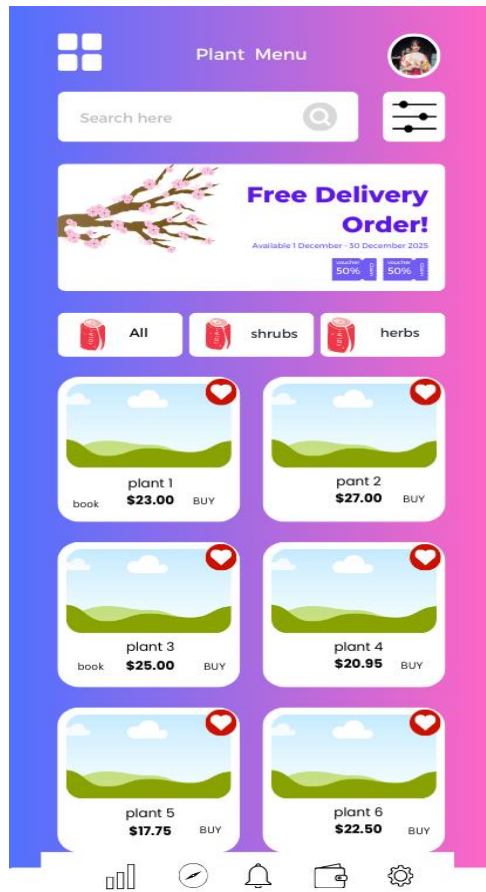
## 12.3 Join program





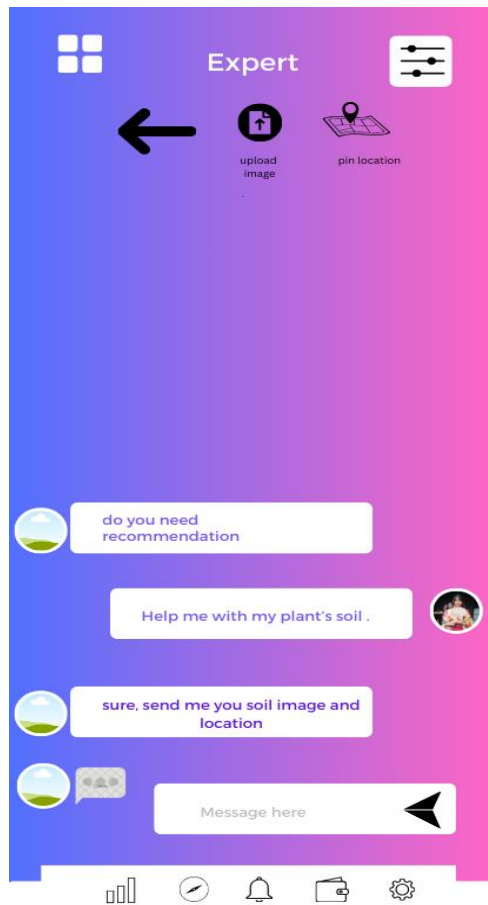
There pops the option of various courses and others which might be paid and unpaid. The user can book the seat for their desired courses and make the payment. upon clicking on the course it shows the description and details of the course.

## 12.4 Purchase plant



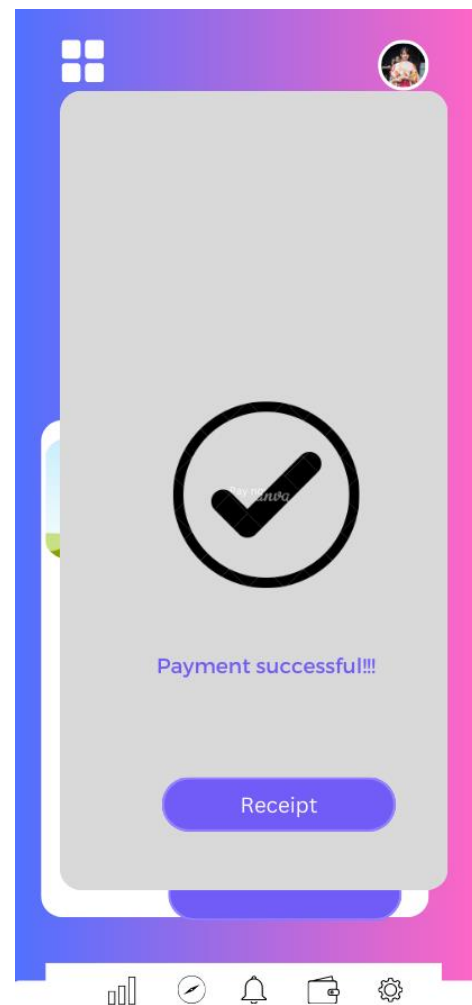
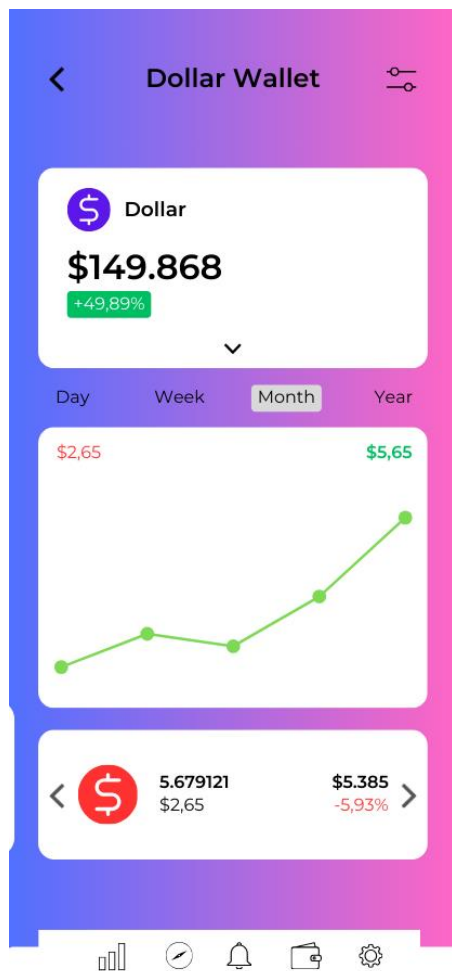
The user can choose the plant they want to purchase. They can see the price, type and various other details of the plant. The user can book the plant they wish to buy and proceed with the payment.

## 12.5 Ask recommendation



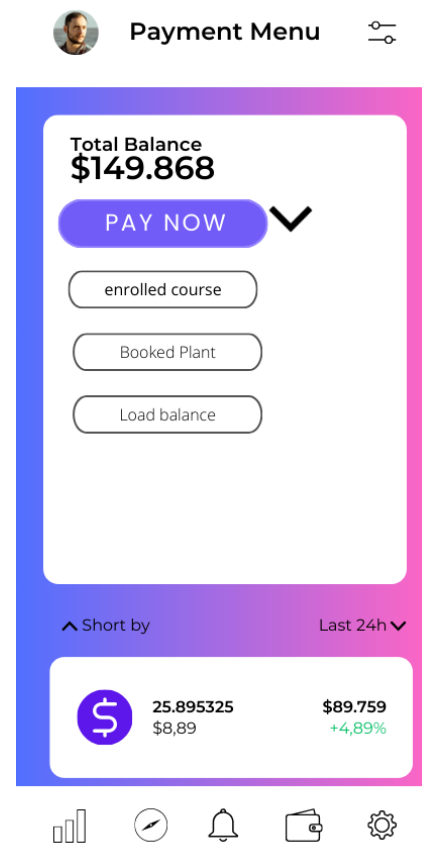
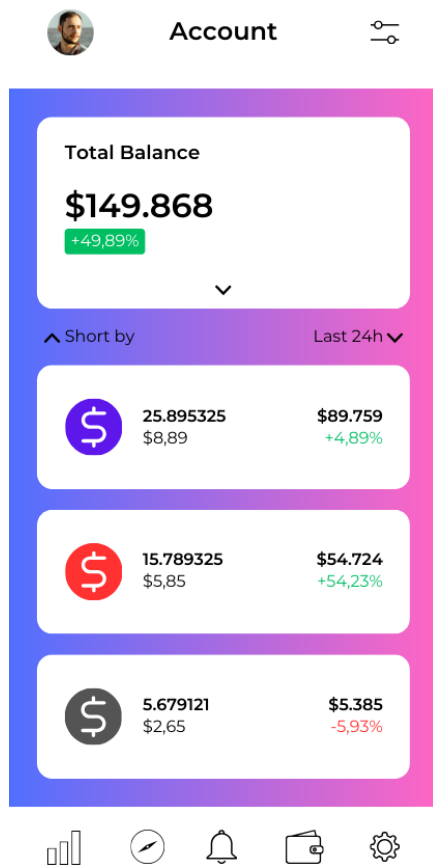
User can ask for recommendation by messaging the experts directly. The prototype provides the option to upload the image of soil condition and also pin point the location in the map option.

## 12.6 Payment



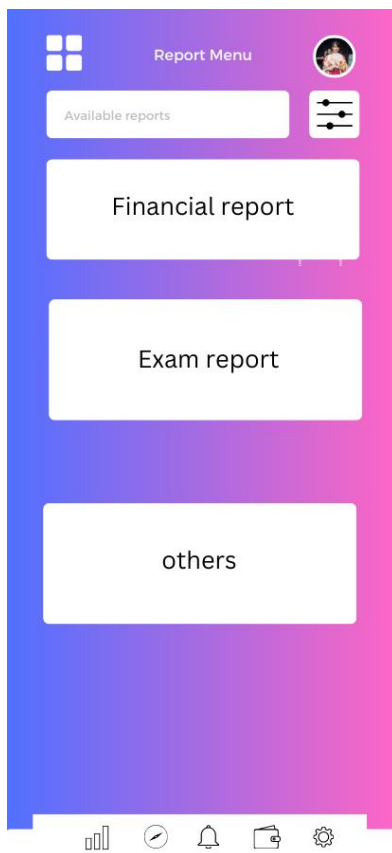
The user can see their money balance in the payment section and see their last payments and other details.





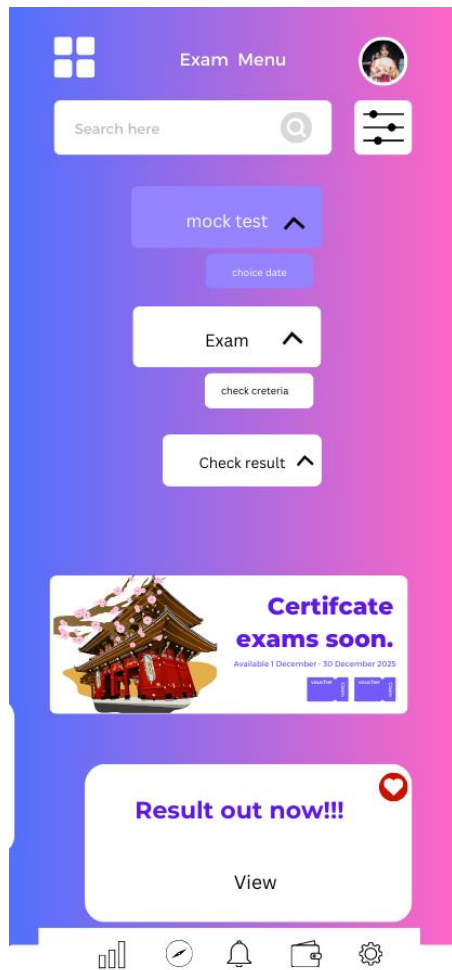
The application can further see more details of their payments.

## 12.7 Report preparation



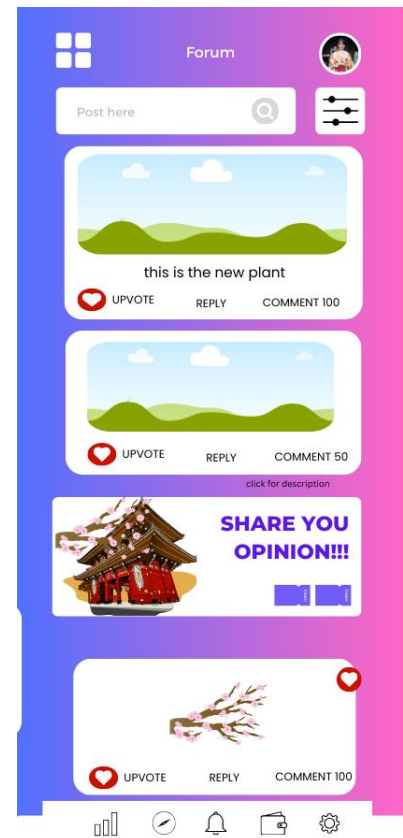
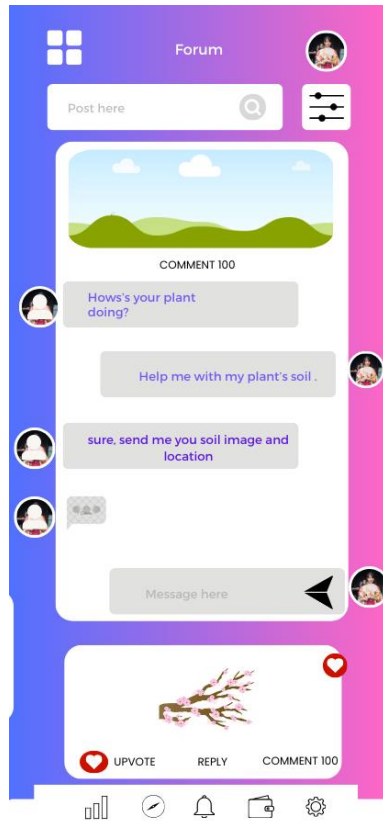
The admin have the access of the report making option. They can prepare various report in the report section.

## 12.8 Take certificate exam



The user can take certification exam after fulfilling the required criteria. They can check result of the exam they appear. Moreover, they can also attempt the mock tests.

## 12.9 Forum



The application also has a forum option where the user can post comments, provide their views, and upvote in others' posts.

## 13 Conclusion

The development of the software system for McGregor Institute of Botanical Trainin provided invaluable insights into the realm of software engineering methodologies and UML diagramming techniques. Through diagrams such as collaboration diagrams, sequence diagrams, use case and class diagrams, I gained a comprehensive understanding of the system's architecture, functionalities, and interactions. Initially, grappling with the intricacies of these diagrams presented a challenge, but with the guidance and support of module leaders and tutors, I navigated through the complexities and successfully created representations that elucidate the system's workings.

Moreover, the adoption of different software development methodologies, coupled with the creation of Gantt charts for project management, enhanced our ability to organize and execute the project's phases efficiently. The iterative nature of the chosen methodology enabled us to adapt to evolving requirements and incorporate feedback seamlessly, ensuring the timely delivery of project milestones.

The creation of prototypes for the software system posed its own set of challenges, particularly in implementing numerous functionalities while maintaining clarity and usability. However, with perseverance and collaborative effort, we overcame these hurdles and developed prototypes that align with the institute's objectives and user requirements.

As I conclude this project, it is evident that the process has not only deepened our understanding of UML diagrams and software development methodologies but also underscored the significance of documentation and reporting in software engineering. The creation of a comprehensive report not only serves as a testament to the system's design and functionality but also highlights the importance of effective communication and presentation skills in conveying the project's essence to stakeholders.

## 14 References

geeksforgeeks. (2024, 02 09). *geeksforgeeks*. Retrieved from geeksforgeeks:  
<https://www.geeksforgeeks.org/use-case-diagram/>

Lucidchart. (2024, 04 24). *Lucidchart*. Retrieved from Lucidchart:  
<https://www.lucidchart.com/pages/uml-use-case-diagram>

Team Gantt. (2024, 04 24). *Gantt Chart*. Retrieved from Gantt Chart:  
<https://www.teamgantt.com/>