Module Code & Module Title

CS5002NI SOFTWARE ENGINEERING

Assessment Weightage & Type

35% Individual Coursework

Year and Semester

2022-23 Spring

Student Name:Arbit Bhandari

London Met ID:22068111

College ID: np01cp4a220446@islingtoncollege.edu.np

Assignment Due Date: Monday, 6th May 2024

Assignment Submission Date: Monday, 6th May 2024

Table of Contents

Table of Table

Table of Figures

1. Introduction

The McGregor Institute of Botanical Training, based in Ireland but operating in Nepal, is working on creating a new software system. This system aims to modernize various aspects of institute management, such as student registration, course enrollment, plant procurement, fee payments, expert guidance based on local conditions, report generation, certification exams, and an online forum for collaborative discussions. By integrating technology into botanical education, the institute seeks to enhance accessibility, streamline administrative processes, and foster a vibrant learning community. This report provides a detailed overview of the software's objectives, functionality, implementation strategy, and anticipated benefits, underscoring the institute's commitment to innovation and educational excellence in the field of agriculture and horticulture.

Aim and Objective

The aim of this coursework is to comprehensively analyze, design, and plan the development of a software system for the McGregor Institute of Botanical Training that supports features such as user registration, course enrollment, plant sales, expert recommendations, report generation, certification exams, and an online community forum. The key objectives include preparing a Work Breakdown Structure and Gantt chart to outline the scope and schedule, modeling requirements through UML diagrams like Use Case diagrams, Collaboration diagrams, and Sequence diagrams, identifying domain classes and creating an Analysis Class diagram, defining the architectural approach, design patterns, development tools/platforms, feature priorities, testing strategy, and maintenance plan, as well as designing user interface prototypes showcasing the major functionalities of the proposed system.

2. WBS(Work Breakdown Structure)

A Work Breakdown Structure (WBS) is a hierarchical and incremental decomposition of a project into phases, deliverables, and work packages. It is a tree structure that organizes and defines the total work scope of the project. The WBS starts with the project as the top-level deliverable and successively derives smaller and more manageable components or work packages. It provides a visual representation of the work required to complete the project objectives, making it easier to assign responsibilities, allocate resources, and track progress throughout the project lifecycle (workbreakdownstructure, 2024).

The Work Breakdown Structure presents a hierarchical diagram illustrating the various phases and processes involved in the McGregor Institute Platform. It outlines five main stages: Initiation, Planning, Execution, Control, and Close Out. Each phase and sub-process is Represented as a point, with connections showing their relationships and flow within the overall project management framework.

Objective of Work Breakdown Structure

The objective of creating a Work Breakdown Structure (WBS) is to systematically decompose the project scope into smaller, more manageable tasks and deliverables.

1. Scope Definition: It helps in clearly defining the scope of the project by breaking down the work into smaller, more tangible components.

2. Task Organization: It organizes the project work into logical groupings, making it easier to allocate resources, assign responsibilities, and track progress.

3. Estimation: It facilitates accurate estimation of time, cost, and resources required for each task or deliverable.

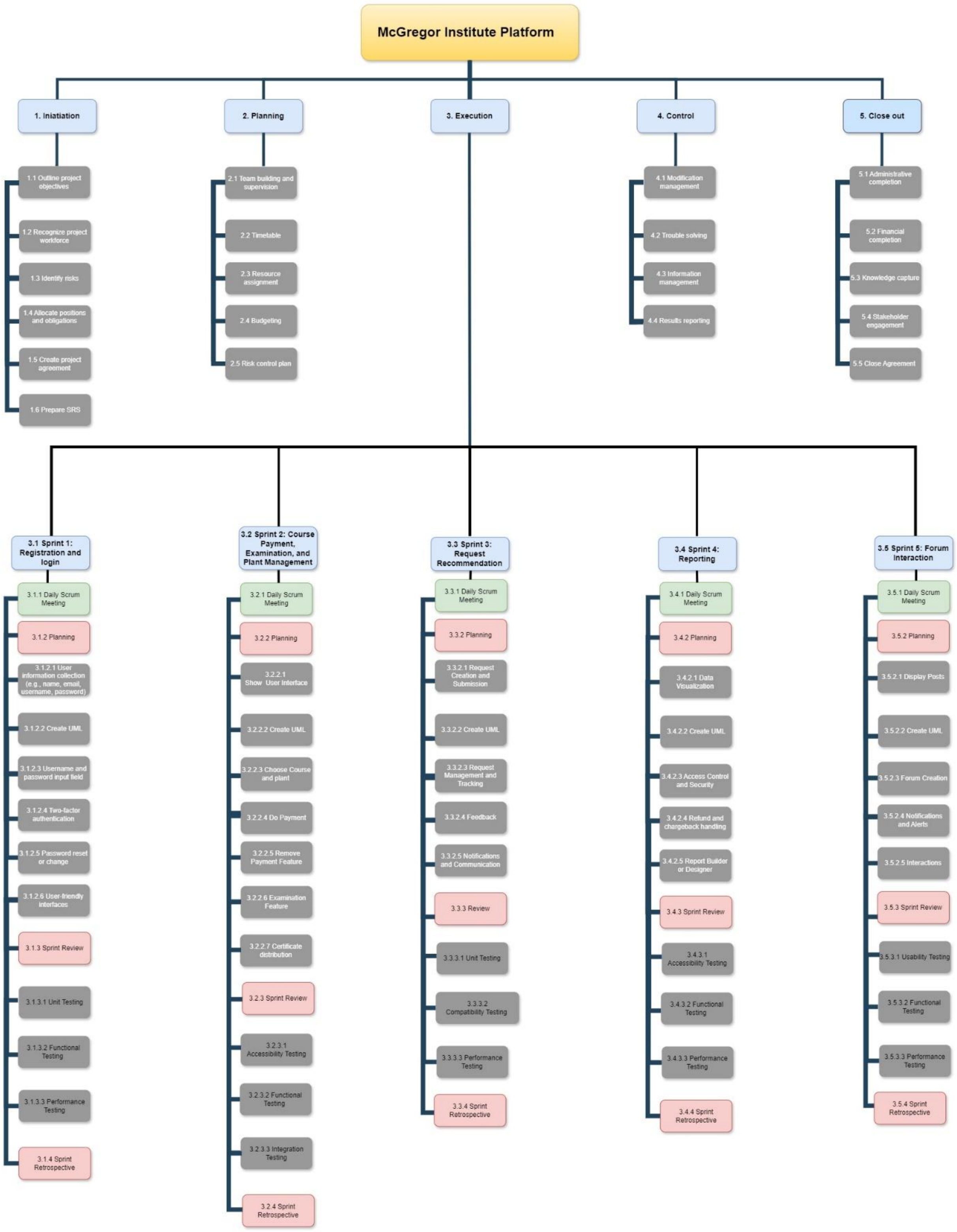

McGregor Institute Platform
1. Iniatiation
1.1 Outline project objectives
1.2 Recognize project workforce
1.3 Identify risks
1.4 Allocate positions and obligations
1.5 Create project agreement
1.6 Prepare SRS
2. Planning
2.1 Team building and supervision
2.2 Timetable
2.3 Resource assignment
2.4 Budgeting
2.5 Risk control plan
3. Execution
4. Control
4.1 Modification management
4.2 Trouble solving
4.3 Information management
4.4 Results reporting
5. Close out
5.1 Administrative completion
5.2 Financial completion
5.3 Knowledge capture
5.4 Stakeholder engagement
5.5 Close Agreement
3.1 Sprint 1: Registration and login
3.1.1 Daily Scrum Meeting
3.1.2 Planning
3.1.2.1 User information collection (e.g., name, email, username, password)
3.1.2.2 Create UML
3.1.2.3 Username and password input field
3.1.2.4 Two-factor authentication
3.1.2.5 Password reset or change
3.1.2.6 User-friendly interfaces
3.1.3 Sprint Review
3.1.3.1 Unit Testing
3.1.3.2 Functional Testing
3.1.3.3 Performance Testing
3.1.4 Sprint Retrospective
3.2 Sprint 2: Course Payment, Examination, and Plant Management
3.2.1 Daily Scrum Meeting
3.2.2 Planning
3.2.2.1 Show User Interface
3.2.2.2 Create UML
3.2.2.3 Choose Course and plant
3.2.2.4 Do Payment
3.2.2.5 Remove Payment Feature
3.2.2.6 Examination Feature
3.2.2.7 Certificate distribution
3.2.3 Sprint Review
3.2.3.1 Accessibility Testing
3.2.3.2 Functional Testing
3.2.3.3 Integration Testing
3.2.4 Sprint Retrospective
3.3 Sprint 3: Request Recommendation
3.3.1 Daily Scrum Meeting
3.3.2 Planning
3.3.2.1 Request Creation and Submission
3.3.2.2 Create UML
3.3.2.3 Request Management and Tracking
3.3.2.4 Feedback
3.3.2.5 Notifications and Communication
3.3.3 Review
3.3.3.1 Unit Testing
3.3.3.2 Compatibility Testing
3.3.3.3 Performance Testing
3.3.4 Sprint Retrospective
3.4 Sprint 4: Reporting
3.4.1 Daily Scrum Meeting
3.4.2 Planning
3.4.2.1 Data Visualization
3.4.2.2 Create UML
3.4.2.3 Access Control and Security
3.4.2.4 Refund and chargeback handling
3.4.2.5 Report Builder or Designer
3.4.3 Sprint Review
3.4.3.1 Accessibility Testing
3.4.3.2 Functional Testing
3.4.3.3 Performance Testing
3.4.4 Sprint Retrospective
3.5 Sprint 5: Forum Interaction
3.5.1 Daily Scrum Meeting
3.5.2 Planning
3.5.2.1 Display Posts
3.5.2.2 Create UML
3.5.2.3 Forum Creation
3.5.2.4 Notifications and Alerts
3.5.2.5 Interactions
3.5.3 Sprint Review
3.5.3.1 Usability Testing
3.5.3.2 Functional Testing
3.5.3.3 Performance Testing
3.5.4 Sprint Retrospective

Figure 1 WBS (Work Breakdown Structure)

2.1 Gantt chart

A Gantt chart is a visual tool used for project planning and scheduling. It displays a list of tasks or activities along the vertical axis, and their scheduled start and end dates are represented by horizontal bars along the horizontal time axis. The length of each bar corresponds to the duration of the task, allowing you to see the sequence of tasks, their overlap, and dependencies at a glance. Gantt charts provide a simple yet effective way to communicate the project timeline, track progress, and identify potential delays or bottlenecks (GRANT, 2023).

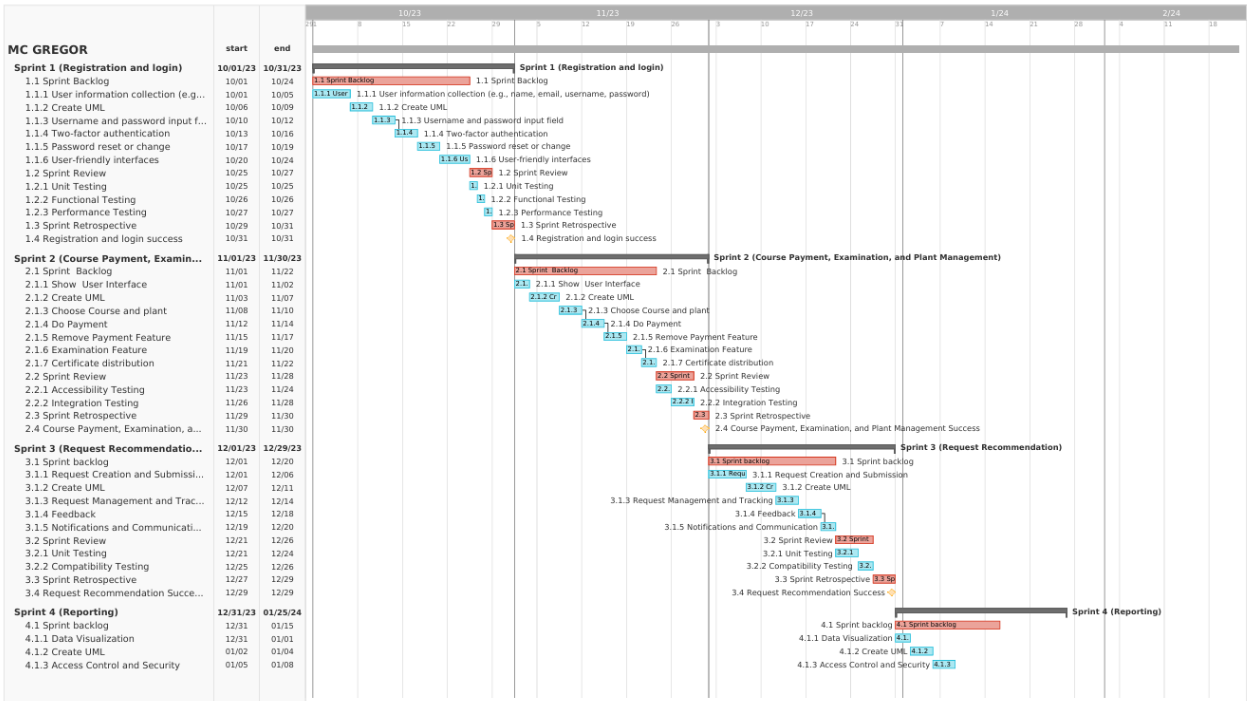
Objective of Gantt Chart

1. Schedule Visualization: A Gantt chart provides a clear visual representation of the project schedule, allowing project managers and team members to easily understand the sequence of tasks and their timing.

2. Task Dependencies: It illustrates the dependencies between tasks, showing which tasks are dependent on others and identifying critical paths in the project timeline.

3. Resource Allocation: By indicating when each task is scheduled to occur, the Gantt chart helps in allocating resources effectively, ensuring that team members are available when needed.

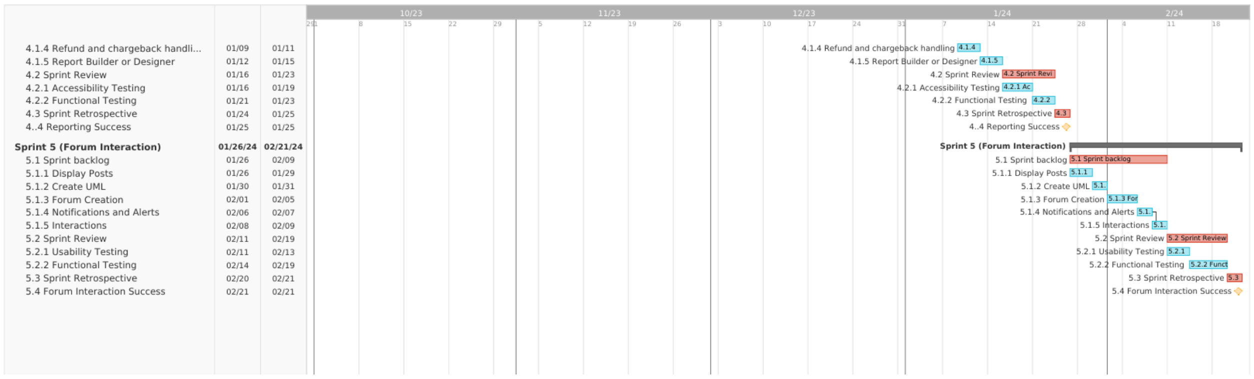
4. Progress Tracking: Project progress can be tracked against the planned schedule by updating the Gantt chart with actual start and end dates for tasks. This allows for early identification of delays or deviations from the planned timeline.



MC GREGOR	start	end
Sprint 1 (Registration and login)	**10/01/23**	**10/31/23**
1.1 Sprint Backlog	10/01	10/24
1.1.1 User information collection (e.g…	10/01	10/05
1.1.2 Create UML	10/06	10/09
1.1.3 Username and password input f…	10/10	10/12
1.1.4 Two-factor authentication	10/13	10/16
1.1.5 Password reset or change	10/17	10/19
1.1.6 User-friendly interfaces	10/20	10/24
1.2 Sprint Review	10/25	10/27
1.2.1 Unit Testing	10/25	10/25
1.2.2 Functional Testing	10/26	10/26
1.2.3 Performance Testing	10/27	10/27
1.3 Sprint Retrospective	10/29	10/31
1.4 Registration and login success	10/31	10/31
Sprint 2 (Course Payment, Examin…	**11/01/23**	**11/30/23**
2.1 Sprint Backlog	11/01	11/22
2.1.1 Show User Interface	11/01	11/02
2.1.2 Create UML	11/03	11/07
2.1.3 Choose Course and plant	11/08	11/10
2.1.4 Do Payment	11/12	11/14
2.1.5 Remove Payment Feature	11/15	11/17
2.1.6 Examination Feature	11/19	11/20
2.1.7 Certificate distribution	11/21	11/22
2.2 Sprint Review	11/23	11/28
2.2.1 Accessibility Testing	11/23	11/24
2.2.2 Integration Testing	11/26	11/28
2.3 Sprint Retrospective	11/29	11/30
2.4 Course Payment, Examination, a…	11/30	11/30
Sprint 3 (Request Recommendatio…	**12/01/23**	**12/29/23**
3.1 Sprint backlog	12/01	12/20
3.1.1 Request Creation and Submissi…	12/01	12/06
3.1.2 Create UML	12/07	12/11
3.1.3 Request Management and Trac…	12/12	12/14
3.1.4 Feedback	12/15	12/18
3.1.5 Notifications and Communicati…	12/19	12/20
3.2 Sprint Review	12/21	12/26
3.2.1 Unit Testing	12/21	12/24
3.2.2 Compatibility Testing	12/25	12/26
3.3 Sprint Retrospective	12/27	12/29
3.4 Request Recommendation Succe…	12/29	12/29
Sprint 4 (Reporting)	**12/31/23**	**01/25/24**
4.1 Sprint backlog	12/31	01/15
4.1.1 Data Visualization	12/31	01/01
4.1.2 Create UML	01/02	01/04
4.1.3 Access Control and Security	01/05	01/08

10/23
11/23
12/23
1/24
2/24
Sprint 1 (Registration and login)
Sprint 2 (Course Payment, Examination, and Plant Management)
Sprint 3 (Request Recommendation)
Sprint 4 (Reporting)

Figure 2 Gantt Chart



	start	end
4.1.4 Refund and chargeback handli…	01/09	01/11
4.1.5 Report Builder or Designer	01/12	01/15
4.2 Sprint Review	01/16	01/23
4.2.1 Accessibility Testing	01/16	01/19
4.2.2 Functional Testing	01/21	01/23
4.3 Sprint Retrospective	01/24	01/25
4..4 Reporting Success	01/25	01/25
Sprint 5 (Forum Interaction)	**01/26/24**	**02/21/24**
5.1 Sprint backlog	01/26	02/09
5.1.1 Display Posts	01/26	01/29
5.1.2 Create UML	01/30	01/31
5.1.3 Forum Creation	02/01	02/05
5.1.4 Notifications and Alerts	02/06	02/07
5.1.5 Interactions	02/08	02/09
5.2 Sprint Review	02/11	02/19
5.2.1 Usability Testing	02/11	02/13
5.2.2 Functional Testing	02/14	02/19
5.3 Sprint Retrospective	02/20	02/21
5.4 Forum Interaction Success	02/21	02/21

The Gantt chart represents a project following the Scrum methodology. The above Gantt chart describe all the work done and duration of time from start to end. Scrum is an agile project management method focused on iterative development and adaptability, dividing projects into short sprints for incremental progress.

3. Use Case Model

A use-case model illustrates how users interact with a system to achieve objectives, detailing user actions and system responses. It comprises elements like actors, use cases, and their associations. Use-case diagrams provide a simplified graphical representation of the model, focusing on specific purposes. Multiple diagrams may depict different subsets of the model consistently. Tools ensure automatic consistency across diagrams when model components are modified. Packages can organize the use-case model to streamline analysis, planning, navigation, communication, development, and maintenance (javatpoint, 2024).

Objective of Use Case:

1. Understanding User Requirements: Use-case models help in eliciting, analyzing, and documenting user requirements by capturing the interactions between users and the system.

2. Communication: They serve as a communication tool between stakeholders, developers, and designers, facilitating a common understanding of the system's functionality and behavior.

3. Scope Definition: Use-case models help in defining the scope of the system by identifying the primary goals or tasks that users need to accomplish.

4. Guidance for Design and Development: They provide guidance for designing and implementing the system, informing decisions about user interface design, system architecture, and feature prioritization.

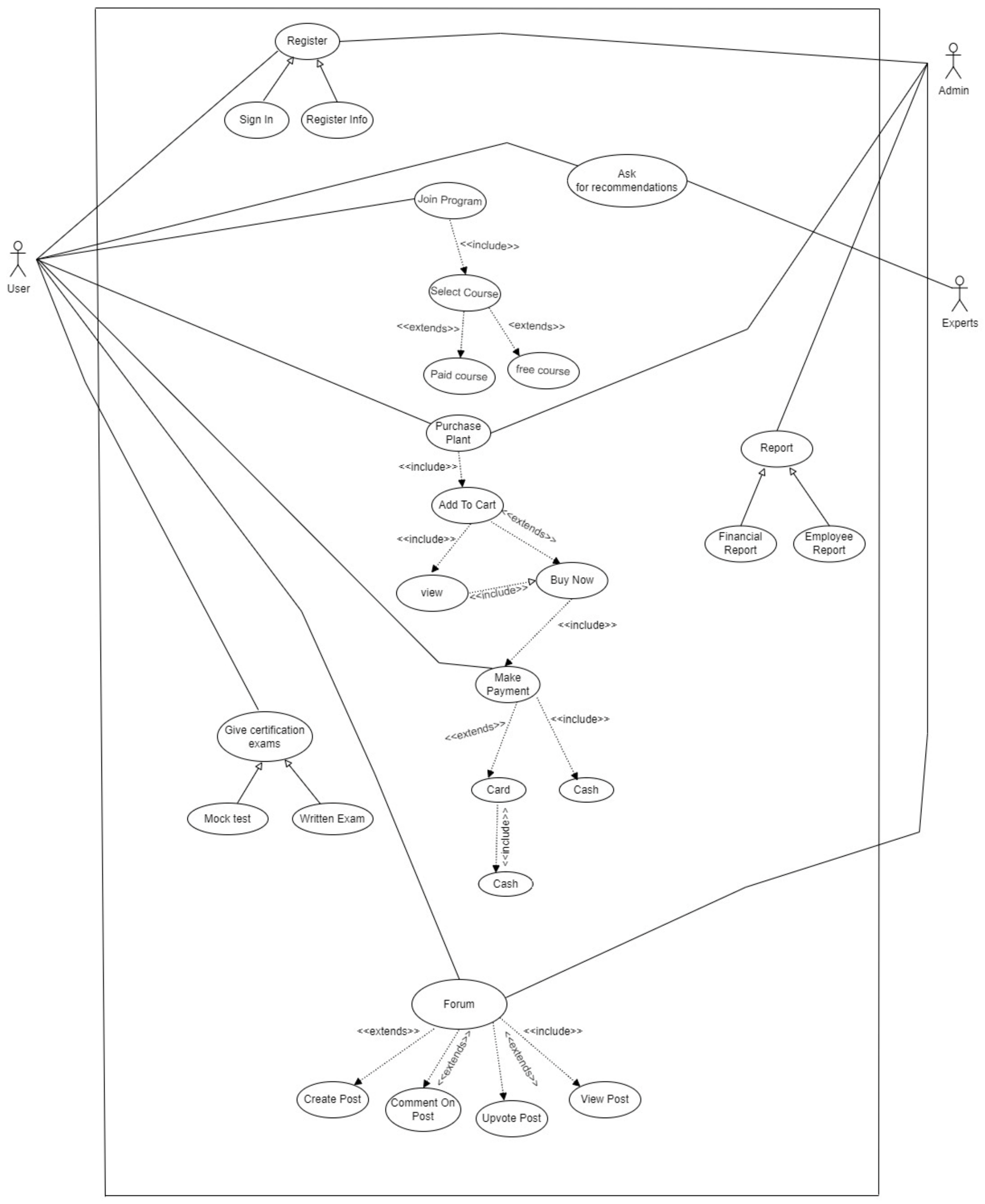


Figure 3 Use Case Model

3.1 High-level use case

Use case: Register

Actor: User

Description: The user can create a new account on the platform.

Use case: Join Program.

Actor: User

Description: The user can plan their learning by selecting courses.

Use case: Purchase Plant.

Actor: User, Admin

Description: The user can purchase plant on the platform.

Use case: Payment.

Actor: User

Description: The user can make payment for purchased courses and plant.

Use case: Ask for recommendations.

Actor: User, Admin

Description: The user asked for recommendations and the admin replay as per request

Use case: Report preparation.

Actor: Admin

Description: It generates reports related to the Financial Report, Employee Report

Use case: Take certification exams.

Actor: User

Description: The user can take mock tests and written exams to obtain certifications

Use case: Forum

Actor: User, Actor

Description: The user can participate in a forum by creating posts, commenting, and viewing posts.

3.2 Expanded Use Case

Use Case	Purchase Plant
Actors	User (Initiator)
Description	Users browse plants, add them to cart, select a secure payment method like mobile banking or e-wallets, and receive a payment receipt after completing the transaction.
Action Steps:	
Actor Action	**System Response**
A. The User Chooses plants from the available options.	B. The System Shows the selected plants to the User.
C. The User start the purchase of the selected plants.	D. The System Requests user information for the purchase (e.g., delivery address, contact details).
E. The User Provides necessary information for the purchase.	F. The System Displays available payment methods to the User.
G. The User Selects a preferred payment method from the available options (e.g., mobile banking, e-wallet).	H. The System Guides the User through the payment process.
I. The User Makes the payment by providing accurate payment details and confirming the transaction.	J. The System Records the payment information and processes the transaction.
An alternative course of action:	
Line C: The User cancels the purchase.	(Use Case Ends)
Line G: The User doesn't have a supported payment method. (Use Case Ends)	
Line I: The user doesn't have sufficient funds to proceed with the payment process. (Use Case Ends)	

Table 1 Expanded use case diagram of Purchase Plant

Use Case	Register Customer
Actors	User (Initiator)
Description	A user registers as a customer on the system to access features such as browsing plants, making purchases, and receiving personalized recommendations
Action Steps:	
Actor Action	**System Response**
A. The User Accesses the registration page on the system.	B. The System Presents the registration form to the User.
C. The User Enters personal information into the registration form (e.g., name, email address, password).	D. The System Validates the entered information and prompts the User to correct any errors if necessary.
E. The User Submits the registration form with accurate information.	F. The System Registers the User as a customer and generates a unique customer identifier.
An alternative course of action:	
Line C: The User abandons the registration process. (Use Case Ends)	
Line D: The entered information is invalid or incomplete. (User corrects errors or abandons the process)	

Table 2 Expanded use case diagram of Register Customer

4. Collaboration Diagram

A collaboration diagram, or communication diagram, depicts the connections and exchanges between software objects within the Unified Modeling Language (UML), providing insight into the dynamic behavior of a specific use case and the responsibilities of each object involved. To craft one, developers pinpoint the essential structural components necessary for executing an interaction and construct a model showcasing the connections between these components. Numerous software vendors provide tools for crafting and modifying collaboration diagrams (Lewis, 2024).

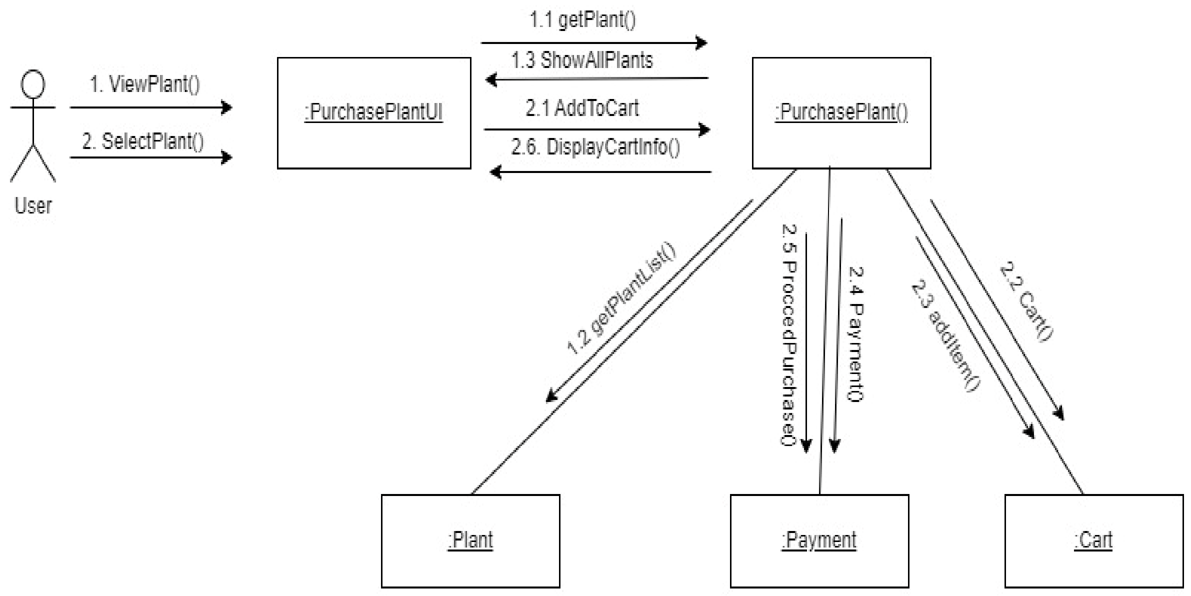


Figure 4 Collaboration Diagram of Purchase Plant

Creating collaboration diagrams typically involves several key steps. First, identify the key objects or components involved in the system or process being modeled. Then, determine the interactions between these objects, including messages sent and received. Next, organize these interactions into a sequence that represents the flow of control or data. Once the sequence is established, you can draft the collaboration diagram using standardized symbols and notation, illustrating how objects collaborate to achieve a specific task or process. Finally, review and refine the diagram to ensure clarity, accuracy, and completeness.

4.1 Sequence Diagram

A sequence diagram in the Unified Modeling Language (UML) visually represents interactions and message order between system objects or components over time. It's crucial for understanding system dynamics, aiding in design, analysis, and communication of task sequences or use cases (visual-paradigm, 2024).

Objective of Sequence Diagram

The objective of a sequence diagram is to depict the interactions between objects or components in a system over time, illustrating the sequence of messages exchanged among them to achieve a specific functionality or behavior. It helps in visualizing the dynamic behavior of the system, including the order of message exchanges, timing constraints, and the flow of control between objects.

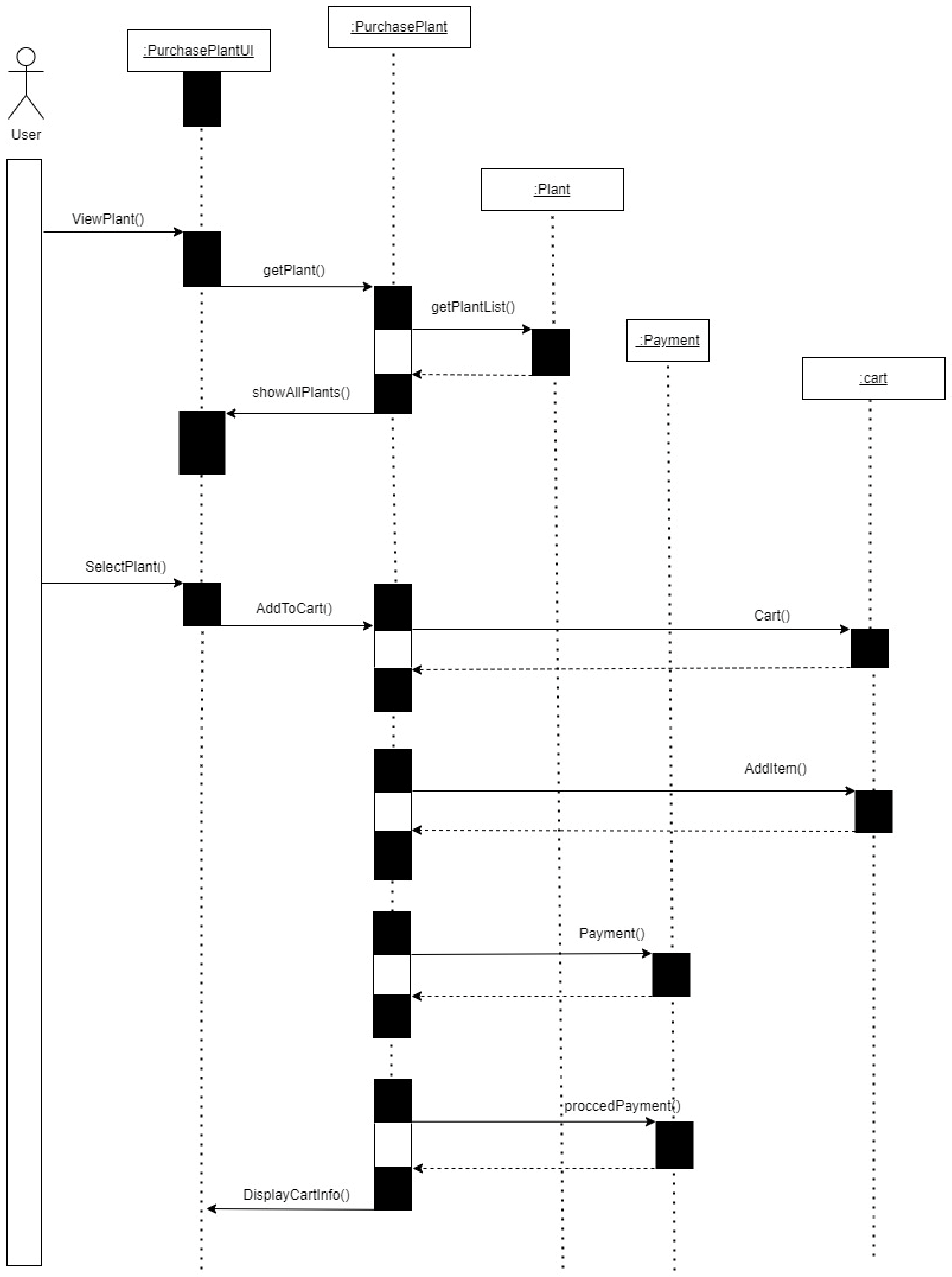


Figure 5 Sequence Diagram of Purchase Plant

5. Class diagram

A class diagram is a type of UML (Unified Modeling Language) diagram that illustrates the structure of a system by showing the classes of the system, their attributes, methods, and relationships between the classes. It provides visual representation of static structure of the system, highlighting the entities involved and their interactions (paradigm, 2024).

The class diagram was created through several steps:

- Identification of domain classes relevant to each use case.
- Determination of attributes and methods for each class.
- Establishment of relationships between elements.
- Utilization of appropriate symbols, notations, and conventions.

The domain classes needed to support the use cases are:

User,

Plant,

Payment,

Report,

Admin,

Exam,

Course

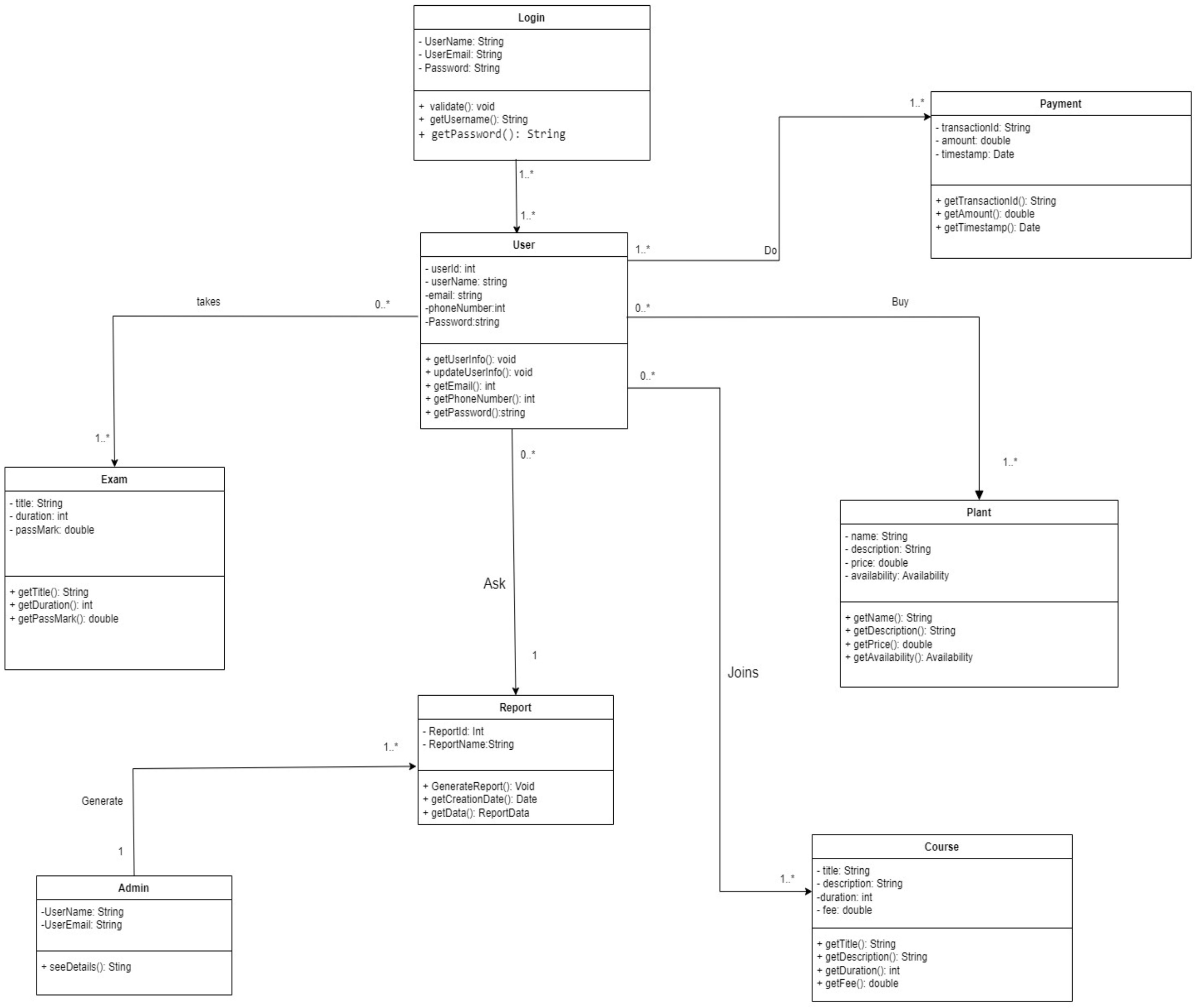


Figure 6 Class Diagram

6. Futher Development

6.1 Architectural choice.

I have done this project on agile scrum methodology

For this project, our team used the Agile Scrum methodology to develop the software in repeating cycles called Sprints. We started by making a list of all the work that needed to be done. At the beginning of each Sprint, we selected some of the highest priority tasks to complete during that Sprint. Every day, the team met quickly to discuss progress and plans. After each Sprint ended, we showed what we had completed to the project's key stakeholders and got their feedback. We also had a retrospective meeting where the team discussed what went well and what could be improved for the next Sprint. Following this cycle of Sprints with frequent feedback allowed us to consistently make progress on the software and adapt to any changes needed along the way.

Define clearly about scrum methodology

Scrum is a way of managing projects by breaking it down into short cycles called Sprints. Each Sprint is usually 2-4 weeks long. At the start of a Sprint, the team decides which tasks they can realistically complete during that time period. They meet daily to discuss progress and any blockers. At the end of the Sprint, they review the completed work and get feedback. They also have a retrospective meeting to discuss what went well and what needs improvement. This cycle repeats in the next Sprint. Scrum emphasizes frequent inspection, adaptation, and continuous improvement of the product being developed. It allows for flexibility if requirements change during the project.

Scrum Methodology: A Clear Definition

We chose to use the Scrum methodology over other approaches like Waterfall or Agile for a few key reasons:

1. Flexibility - Scrum lets us change things as we go. This was important because what the customer wants can change as we make the software.

2. Faster Delivery - Scrum splits our work into short 2-4 week chunks called Sprints. This means we can show the customer something sooner and get feedback quicker.

3. Seeing Progress - Scrum has lots of meetings where we talk about what's going on. This helps everyone know how things are going and if there are any problems.

4. Making Things Better - At the end of each Sprint, we have a meeting to talk about what went well and what could be better. This helps us work smarter.

5. Teamwork - Scrum is all about working together and taking responsibility. This can make the team feel good and make the software better.

Using Scrum helps us deal with changes and make sure we're making the best software we can.

Using Scrum for McGregor Project: A smooth Approach

1. Step-by-Step Building: Scrum breaks down the project into small parts and focuses on finishing one part at a time. This helps in building the different parts gradually, like signing up, taking courses, buying plants, and so on.

2. Changing Plans: Scrum is good for projects where things might change. If the institute wants to add new courses or make changes, Scrum can handle it easily.

3. Teamwork and Talking: Scrum encourages everyone in the team to work closely together and talk every day. This helps everyone understand what needs to be done and makes it easier to make decisions.

4. Listening to Users: With Scrum, the team shows what they've done to the people. This helps them get feedback and make sure they're making the way users want it.

5. Getting Better: Scrum includes meetings where the team talks about what went well and what they can improve. This helps them make better over time.

6. Choosing What's Important: Scrum helps the team decide what to work on first based on what's most important and what's hardest. This way, they can tackle the tricky stuff early and make sure they're building the most important things first.

By using Scrum, the McGregor Institute of Botanical Training can build step by step, handle changes easily, work together well, listen to users, improve over time, and focus on what matters most.

The project is chosen to done in layered architecture

The McGregor Institute project will be developed using a layered architecture approach. A layered architecture is a software design pattern that structures the application into separate layers or tiers, each with a specific role and responsibility. This architectural style promotes separation of concerns, modularity, and code reusability.

Layered architecture: A Clear Definition

A layered architecture separates the different parts of a software system into distinct horizontal layers. The bottom layer directly manages the data storage, like the database. The layer above that handles all data access operations between the database and the rest of the application. Another layer contains the core business logic - the rules and processing for things like course registrations, payments, recommendations etc. The top layer is the user interface where users interact with the system through screens, pages, menus etc. This layered approach keeps each part of the system focused on a specific role, making the overall design more organized and easier to develop, understand, and maintain over time.

Its advantage over McGregor Institute project

1. Keeping Things Organized: Layers help divide the work into different parts, like how the website looks, how it works behind the scenes, and where it stores information. This makes it easier to manage and reuse code.

2. Growing and Changing: Each layer can be adjusted or replaced without messing up the whole system. So, if the institute wants to add new stuff or connect with other systems, it's easier to do.

3. Staying Safe: Layers help add security at different levels, like making sure only the right people can access certain parts. This makes the website safer from attacks.

4. Working Together: Different teams can work on different parts of the website at the same time. This speeds up the work and fits well with the way the project is being managed.

5. Testing Made Easier: Each layer can be tested separately, making it simpler to find and fix any problems. This helps make sure the website works well and is reliable.

Layered Architecture for the McGregor Institute Project: Making the Connection

For the McGregor Institute project, we are separating the different parts of the software into distinct layers. At the bottom is the database layer for storing and fetching data like user details, courses, plants, and payments. Above that is the data access layer which handles all interactions between the database and the rest of the application. The next layer contains the core business logic - the rules and processing for things like course registrations, purchases, recommendations etc. At the top is the user interface layer with the screens, menus, and pages that students, plant enthusiasts, and admins will use. Having these different layers makes the whole system more organized and manageable. Each layer focuses on just its specific role. It also allows our team to work on separate layers simultaneously during our sprints.

This project is done in Object Oriented Paradigm

In this project, the Object-Oriented Paradigm is used to organize the system's design and implementation around objects. Each object represents a specific entity or concept within the system, such as User, Course, Plant, Payment, Recommendation, Report, Exam, and ForumPost.

Here are some key advantages of following the Object-Oriented paradigm for the McGregor Institute project,:

1. Code Organization: By breaking down the system into objects representing real-world entities like courses, students, and plants, the code becomes more organized and easier to understand and maintain.

2. Reusability: Objects can inherit properties and behaviors from other similar objects, allowing code reuse and avoiding duplication. This saves time and effort.

3. Extensibility: New features like additional course types or plant varieties can be easily added by creating new objects that inherit from existing ones, making the system more adaptable and scalable.

4. Encapsulation: Each object encapsulates its own data and functions, preventing external code from directly accessing or modifying its internal workings. This promotes code stability and maintainability.

5. Collaboration: Objects can interact with each other by calling each other's methods, enabling different parts of the system to collaborate and work together seamlessly.

6.2 Design Pattern.

Design patterns are reusable solutions to commonly occurring problems in software design. The Gang of Four (GoF) patterns are a widely recognized and influential collection of 23 software design patterns introduced in the book "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

Behavioral patterns are like recipes for how objects should interact and behave in a software system. They help us organize how different parts of our code communicate with each other. For example, think of the Observer Pattern as a way for objects to stay updated about changes happening in another object. It's like setting up a notification system so that when something changes in one part of the code, other parts get automatically updated.

Creational patterns are all about making it easier to create objects in our code. They give us different ways to create objects depending on what we need. For instance, the Factory Method Pattern is like having a customizable factory that can make different types of products based on what we ask for. And the Singleton Pattern is like ensuring there's only one boss in the office, no matter how many times you ask for a new one.

Structural patterns are like blueprints for organizing the structure of our code. They help us build our software in a way that's easy to understand and modify. Take the Adapter Pattern, for example. It's like having a translator between two people who speak different languages. It helps them understand each other without having to change their ways of communicating.

The project is done on structural pattern using mvc pattern

For this project, I have chosen the MVC pattern, which is a Structural design pattern that separates the application logic into three interconnected components: Model, View, and Controller. This separation of concerns allows for better code organization, maintainability, and scalability.

By using the MVC pattern, the project's codebase is structured as follows:

1. Model: This component encapsulates the data and business logic of the application. It includes classes or modules for user management, course management, plant management, payment management, recommendation engine, report generation, certification management, and forum management.

2. View: The View component is responsible for rendering the user interface and presenting data to the user. It includes various user interfaces, such as registration and login pages, course enrollment pages, plant catalog and purchase pages, payment gateway integration, recommendation request and display pages, report generation interfaces, certification exam and mock test interfaces, and community forum pages.

3. Controller: The Controller acts as an intermediary between the Model and View components. It receives user input, processes it by invoking the appropriate methods in the Model, and updates the View accordingly. The project includes controllers for user management, course management, plant management, payment processing, recommendation handling, report generation, certification management, and forum operations.

By adopting the MVC pattern, a Structural design pattern, the project achieves a clear separation of concerns, improved code organization, and better maintainability. The Model handles the application's data and business logic, the View manages the user interface, and the Controller facilitates communication between the Model and View components, providing a well-structured and extensible architecture.

MVC pattern make the project easy and smoother

Separation of Concerns:

1. The separation of concerns provided by MVC allows for better code organization and maintainability.
2. Each component (Model, View, and Controller) has a specific responsibility, making it easier to understand, modify, and test individual components without affecting the entire system.
3. Changes in the user interface (View) can be made without impacting the underlying business logic (Model), and vice versa, reducing the risk of introducing bugs and making the system more maintainable.

Code Reusability:

1. By separating the application logic from the presentation layer, MVC promotes code reusability.
2. The Model and Controller components can be reused across different Views or user interfaces, reducing duplication of code and promoting consistency.
3. For example, the same Plant Model and Plant Controller can be used for different Views, such as the plant catalog, purchase page, and admin management interface.

Parallel Development:

1. The separation of concerns facilitated by MVC allows multiple developers to work on different components of the application simultaneously.
2. One team can focus on developing and testing the Models (business logic), while another team works on the Views (user interfaces) and Controllers, increasing development efficiency and reducing bottlenecks.

Testability:

1. MVC architecture makes it easier to write and execute unit tests for each component independently.
2. Models can be tested without involving the user interface, and Views can be tested without relying on the application logic, leading to more robust and maintainable code.
3. This modular testing approach helps catch and fix issues early in the development process, ensuring a smoother overall development experience.

Flexibility and Extensibility:

1. The MVC pattern provides a flexible and extensible architecture.

2. As the application grows or requirements change, new Views or Controllers can be added without modifying the existing Model or other components, reducing the risk of introducing bugs and making the system more scalable.

3. For example, if the institute decides to introduce a new feature for managing greenhouse operations, a new Controller and View can be added without impacting the existing components.

Clear Roles and Responsibilities:

1. MVC defines clear roles and responsibilities for each component, making it easier for developers to understand and work on different parts of the application.

2. The Model handles the business logic and data management, the View is responsible for rendering the user interface, and the Controller acts as an intermediary between the two, facilitating communication and control flow.

6.3 Development Plan

The development plan outlines the tools, resources, and programming platforms that will be used for building the proposed system, as well as the tentative priority order of features and artifacts to be implemented.

Tools and Resources:

Programming Language: Java or Python (depending on the team's expertise)

Web Framework: Spring (Java) or Django (Python) for building the web application
Database: MySQL or PostgreSQL for data storage
Front-end Technologies: HTML, CSS, JavaScript, and a modern JavaScript framework like React or Angular
Version Control: Git and a hosting service like GitHub or GitLab
Project Management: Jira or Trello for task tracking and project management
Testing Frameworks: JUnit (for Java) or pytest (for Python) for unit testing, and Selenium for end-to-end testing

Priority Order of Artifacts/Features:

User Registration and Authentication: Implement user registration, login, and authentication mechanisms.
Course Management: Develop functionality for managing courses (graduate, postgraduate, and certification), including course creation, enrollment, and payment processing.
Plant Catalog and Purchase: Create a catalog of plant varieties, allowing users to view and purchase plants, with integration of the payment system.
Discussion Forum: Implement the discussion forum feature, enabling users to create posts, comment, and upvote/downvote.
Certification Exams and Mock Tests: Implement the functionality for conducting certification exams and mock tests, including result tracking and evaluation.

Reporting and Analytics: Build reporting and analytics features for generating financial reports, employee reports, and user data reports.

Admin Panel: Create an admin panel for managing courses, users, plants, and other administrative tasks.

Performance Optimization and Security: Optimize the application for performance and implement necessary security measures, such as input validation and secure authentication mechanisms.

Responsive Design and Accessibility: Ensure the application is responsive and accessible on various devices and for users with disabilities.

6.4 Testing Plan

The McGregor Institute of Botanical Training project, it's essential to have a detailed testing plan in place. Here's an outline of a testing plan that can be followed:

1. Unit Testing:

Test individual parts (like user registration, course enrollment) to make sure they work correctly.

Use pretend data or a special setup for testing.

2. Integration Testing:

Check that different parts work together well.

Make sure that when you do something (like enroll in a course), it shows up correctly on the screen.

3. End-to-End Testing:

Pretend to be a real user and try out different things.

Make sure everything works from start to finish.

4. Performance Testing:

Check if the app is fast enough and doesn't break when lots of people use it.

See how it handles different situations, like lots of users at once.

5. Security Testing:

Look for ways no one could break in and steal information.

Make sure only the right people can do certain things.

6. Continuous Monitoring and Feedback:

Keep an eye on how the app is doing in the real world.

Listen to what people say and fix any problems we find.

By doing all of this, we make sure the McGregor Institute of Botanical Training project stays strong and works well for everyone who uses it.

6.5 Maintan Plan

1. Keeping the Code in Good Shape:

Make sure everyone follows the same rules when writing code.

Regularly tidy up the code to make it easier to understand and work with.

Use tools like Git to keep track of changes and work together smoothly.

2. Writing Things Down:

Write down how everything works: the code, how it's set up, and how to put it all together.

Keep these documents updated whenever things change.

Ask developers to explain what they're doing in simple terms.

3. Keeping Software Up-to-Date:

Regularly check and update any outside tools or programs the project relies on.

Make sure everything still works well together.

Plan carefully when making changes to avoid causing problems.

4. Keeping Things Safe:

Stay informed about any new ways no one try to mess with the project.

Fix any security problems as soon as they're found.

Check regularly for weak spots by testing how secure everything is.

5. Making Sure Everything Runs Smoothly:

Use tools to watch how well the project runs when people are using it.

Look for things that could be slowing it down or causing problems.

Fix anything that's causing trouble to keep things running smoothly.

6. Preparing for the Worst:

Have a plan for what to do if everything suddenly stops working.

Test this plan regularly to make sure it actually works.

Make sure you can get everything back up and running quickly if something goes wrong.

7 Prototypes

A prototype is a preliminary version of a product or idea that is used to test and evaluate its design, functionality, and feasibility before full-scale production or implementation. It allows designers and developers to gather feedback, make improvements, and refine the final product.

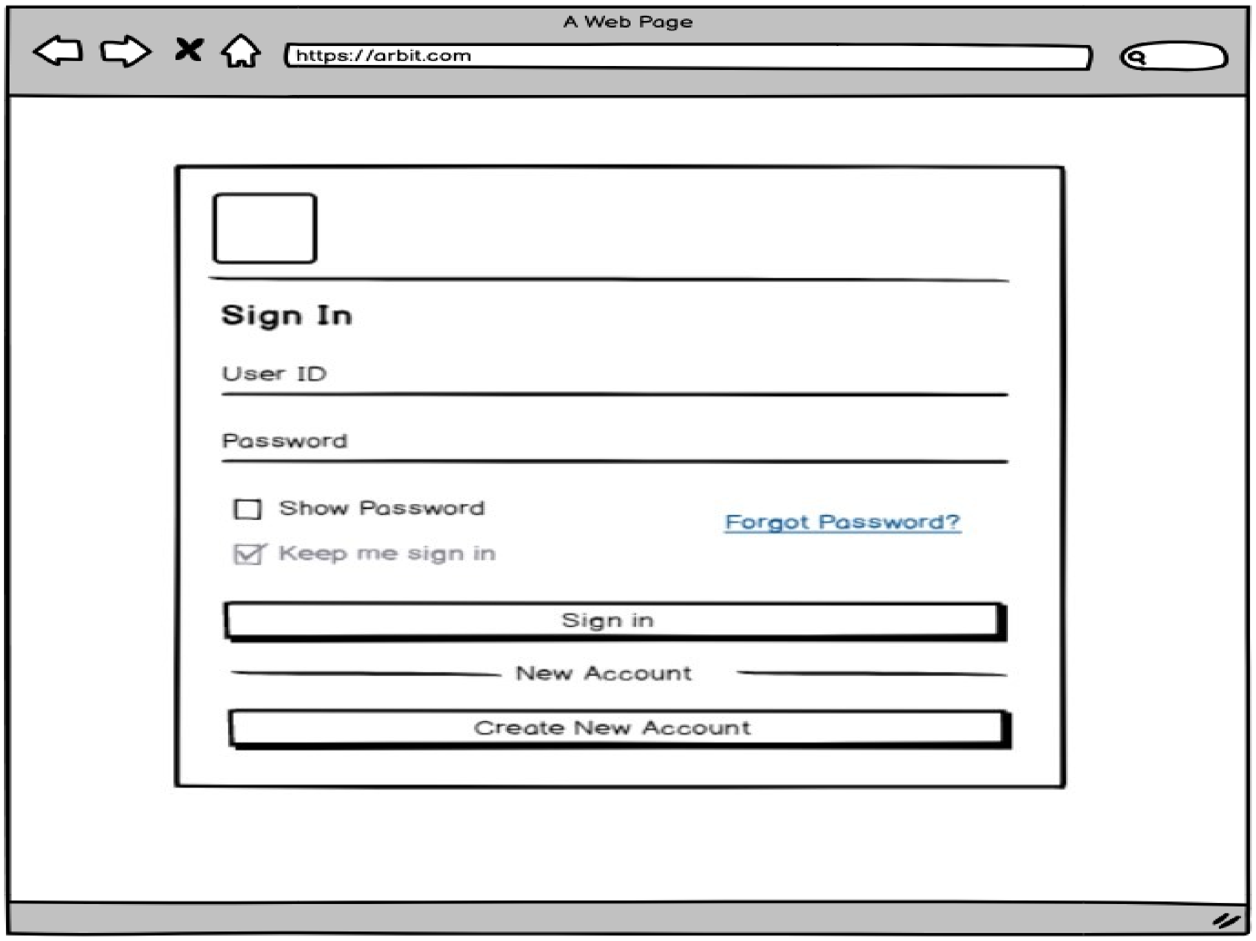
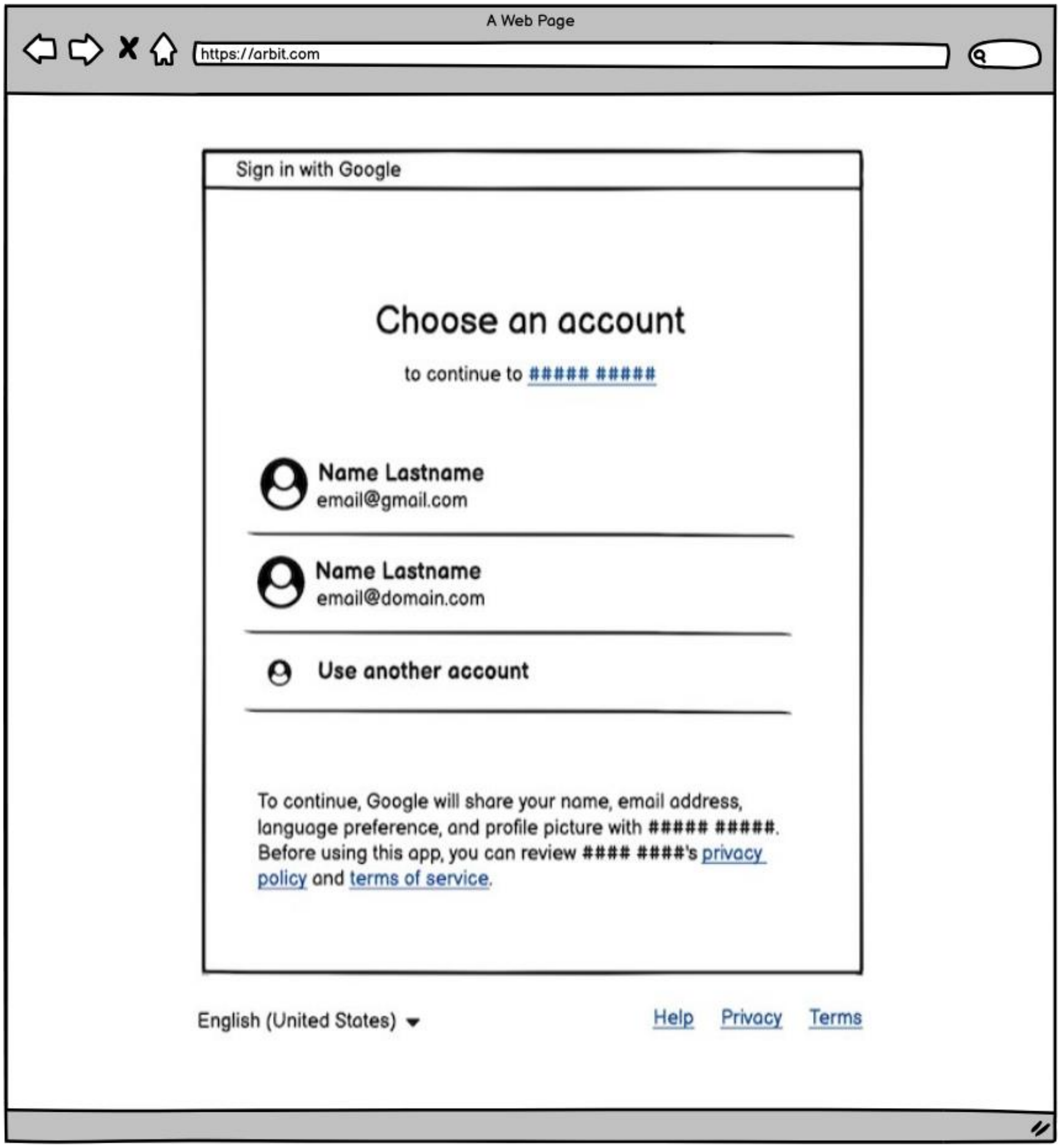


Figure 7 Prorotype of Sign In



Choose an account

to continue to ##### #####

Name Lastname
email@gmail.com

Name Lastname
email@domain.com

Use another account

To continue, Google will share your name, email address, language preference, and profile picture with ##### #####. Before using this app, you can review #### ####'s privacy policy and terms of service.

Figure 8 Prorotype Sign In with choosing accounts

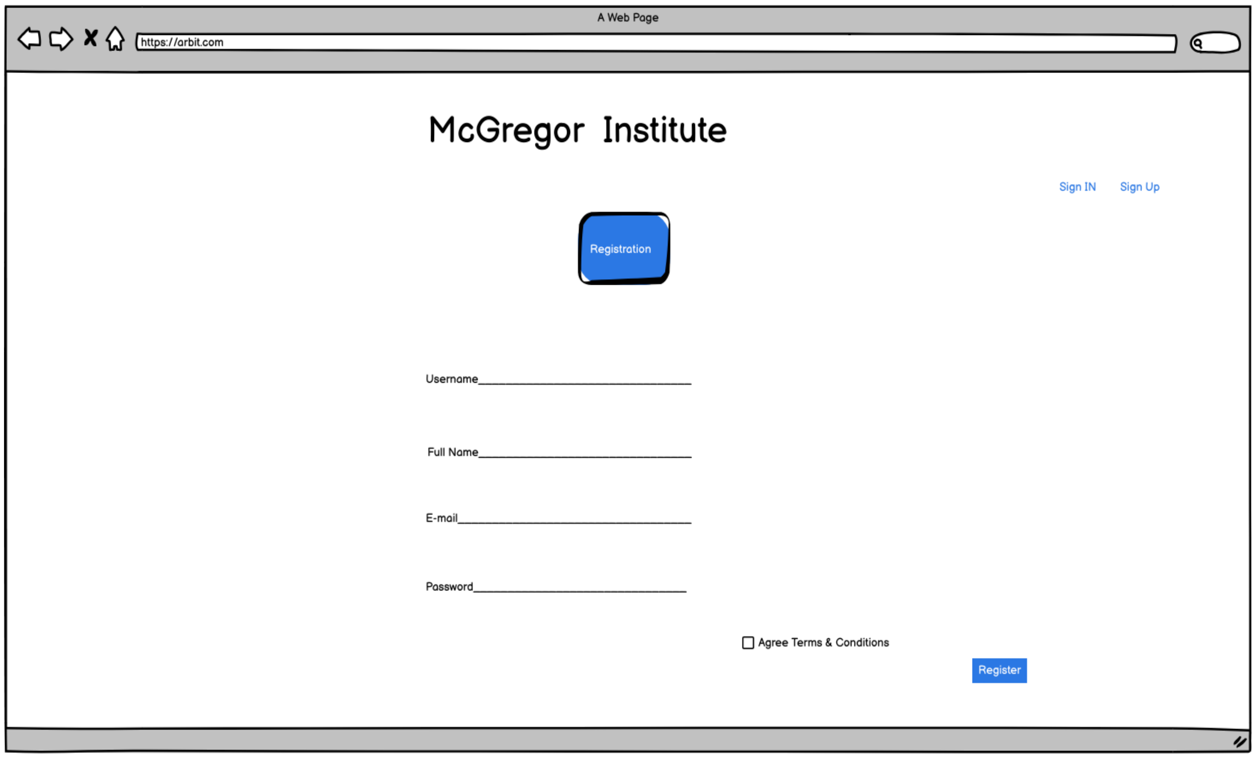


Figure 9 Prorotype of Registration

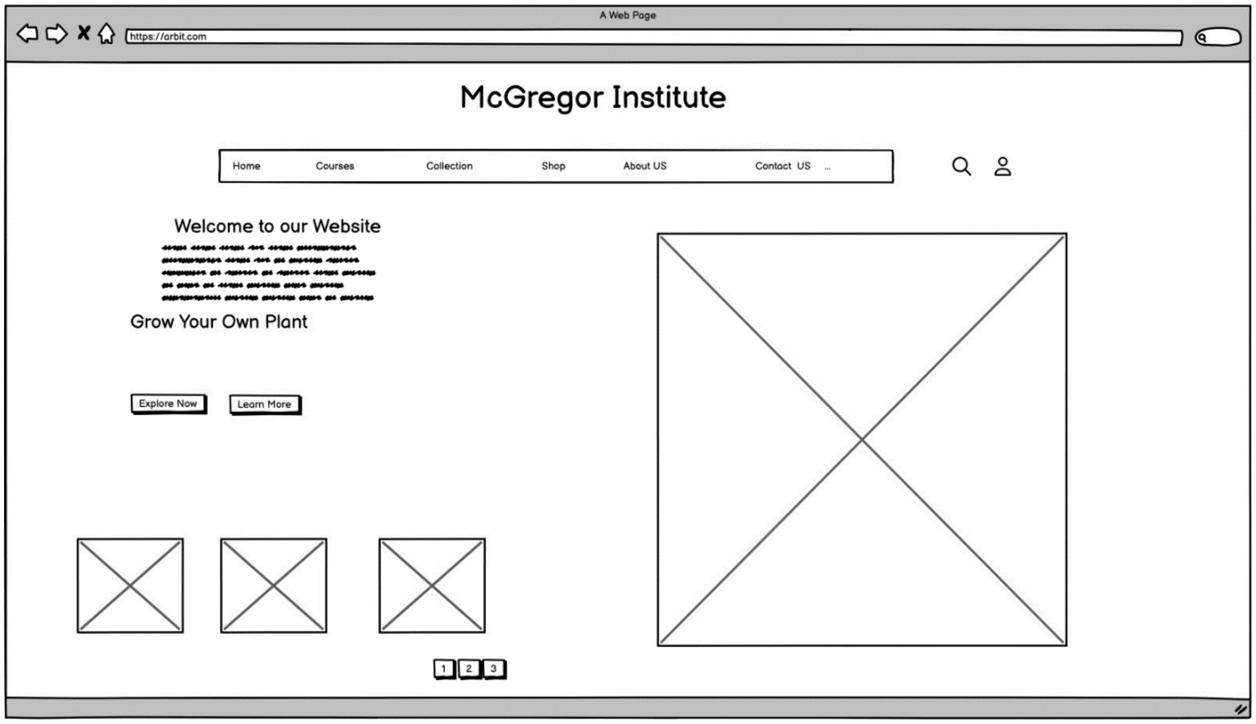


Figure 10 Prorotype of Home Page

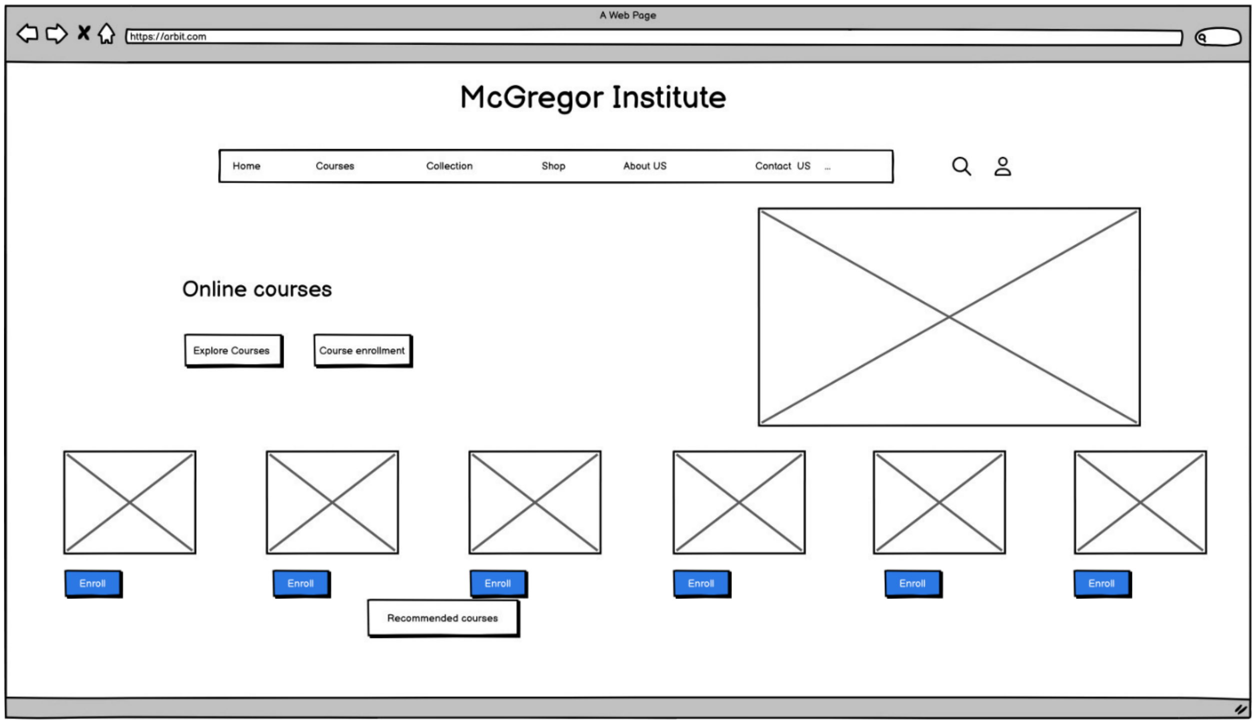


Figure 11 Prorotype of enroll course

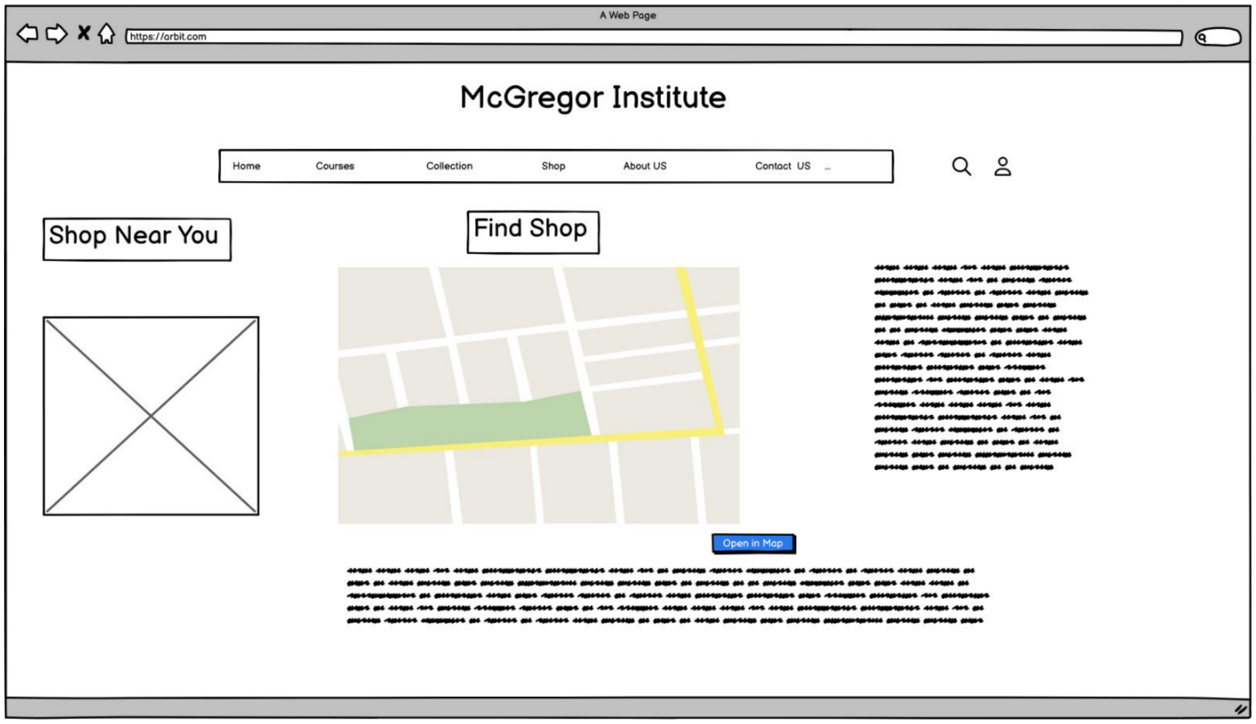


Figure 12 Prorotype of finding virtual shop

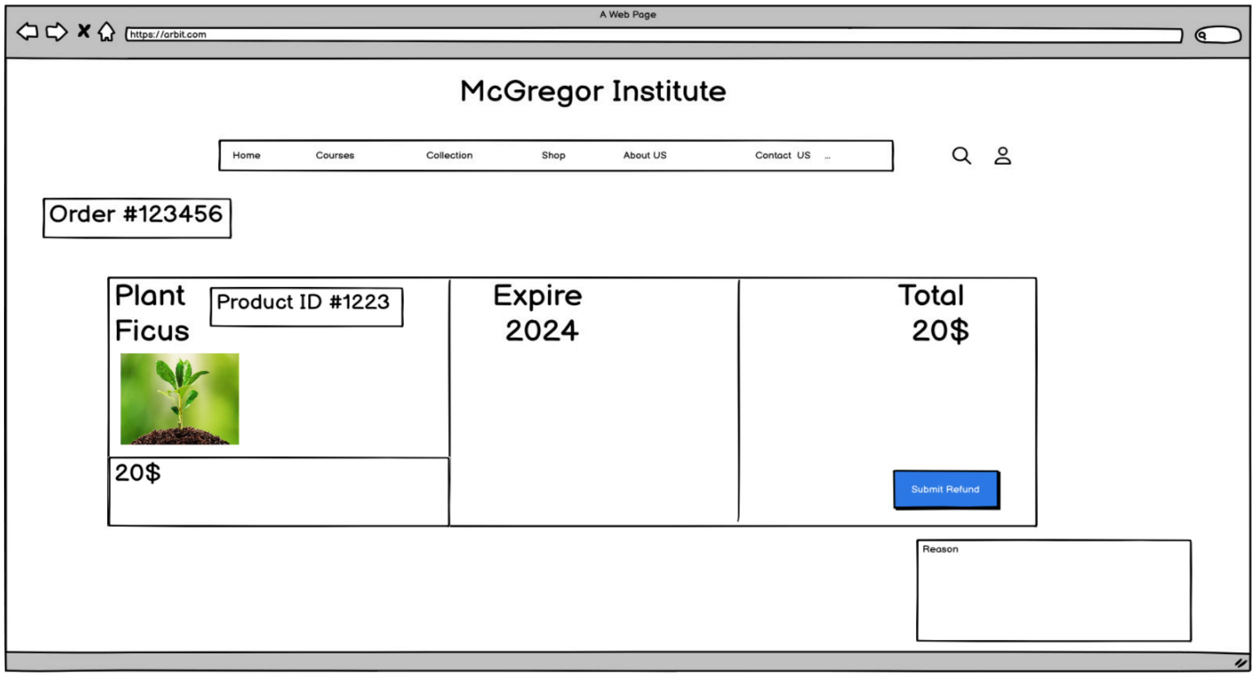


Figure 13 Prorotype of refund

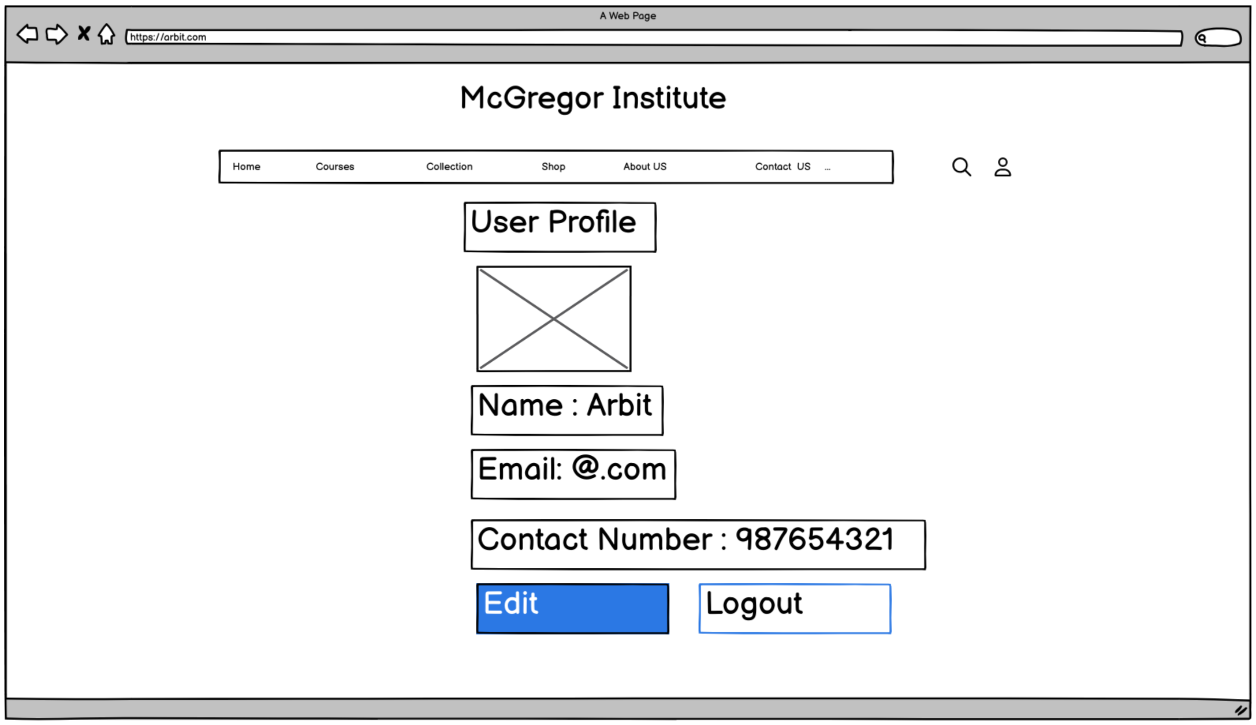


Figure 14 Prorotype of User Profile

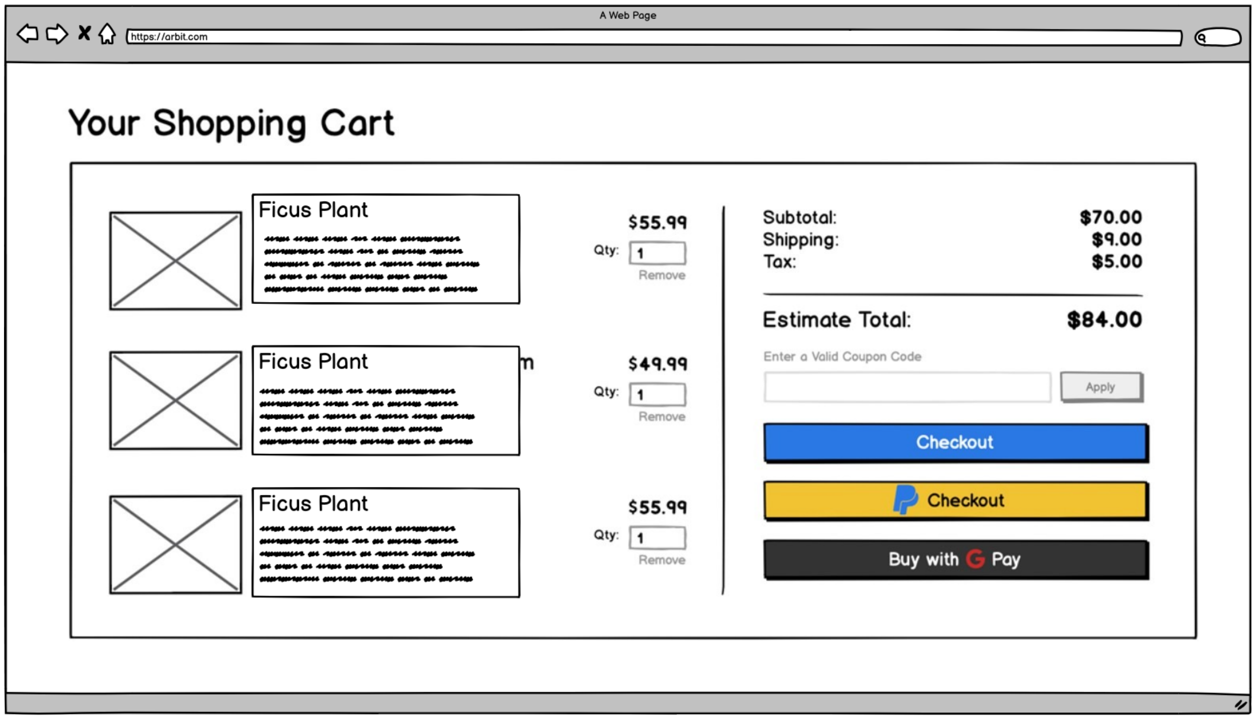


Figure 15 Prorotype of Cart

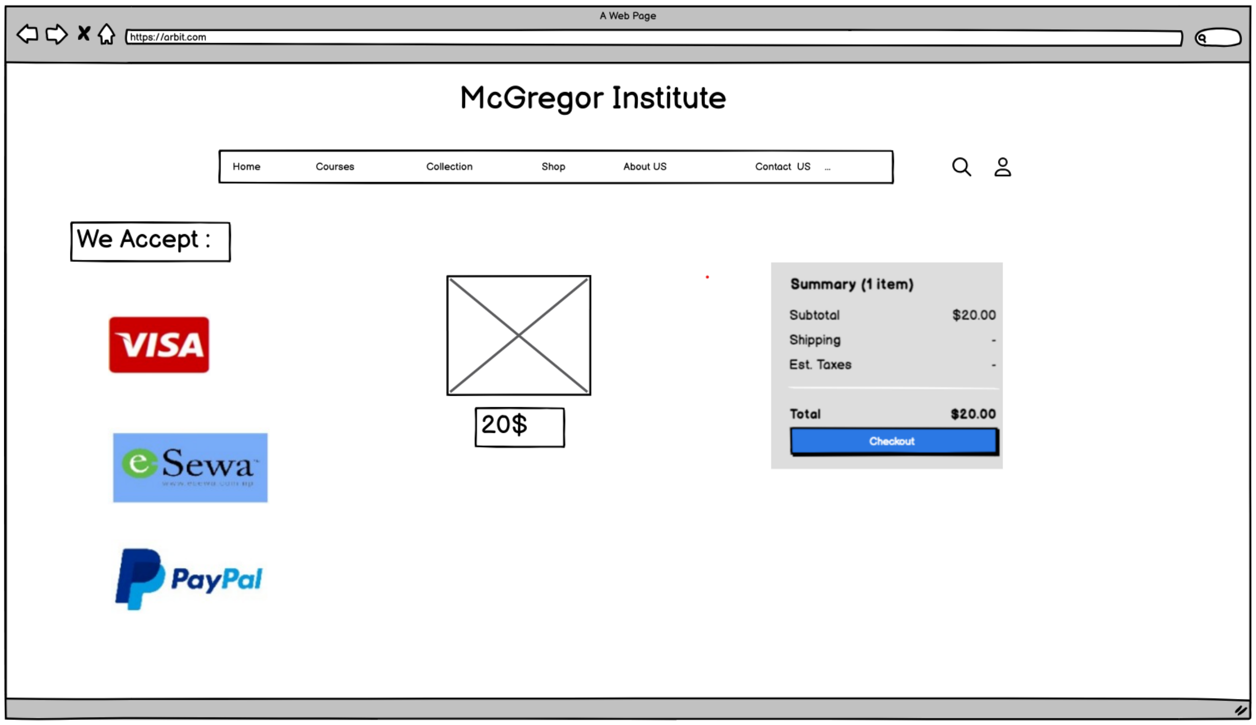


Figure 16 Prorotype of Payment

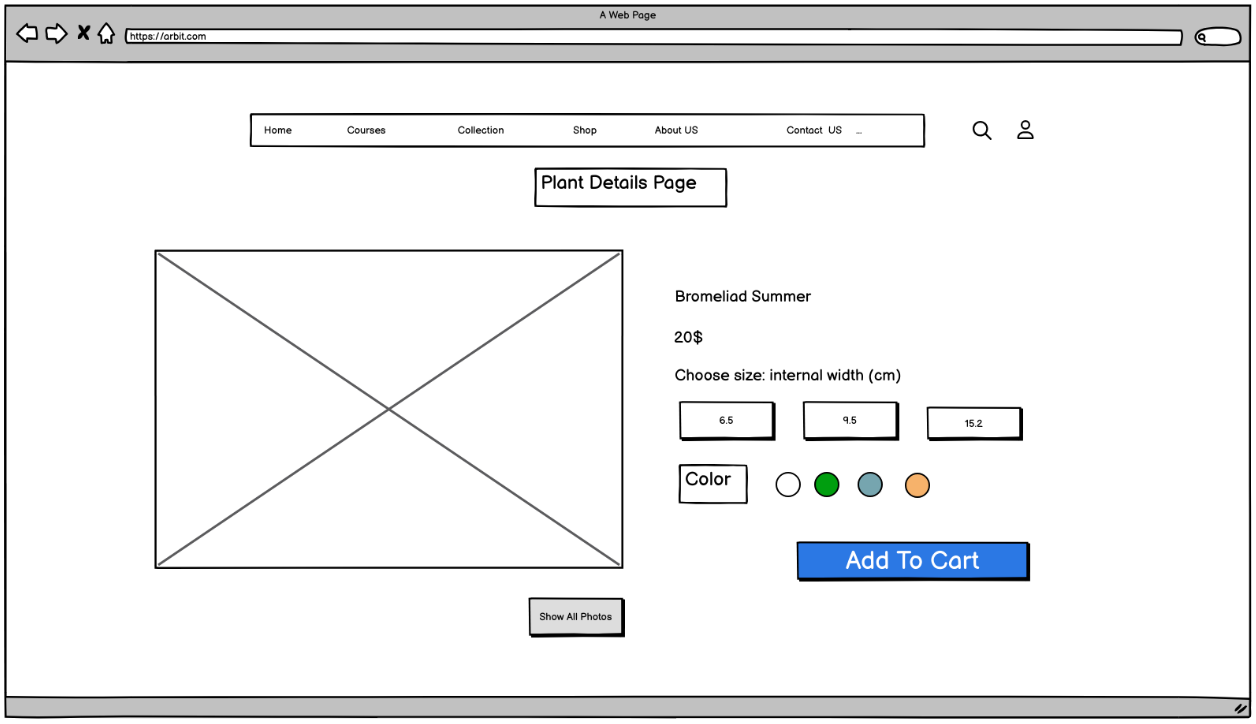


Figure 17 Prorotype of Product Details

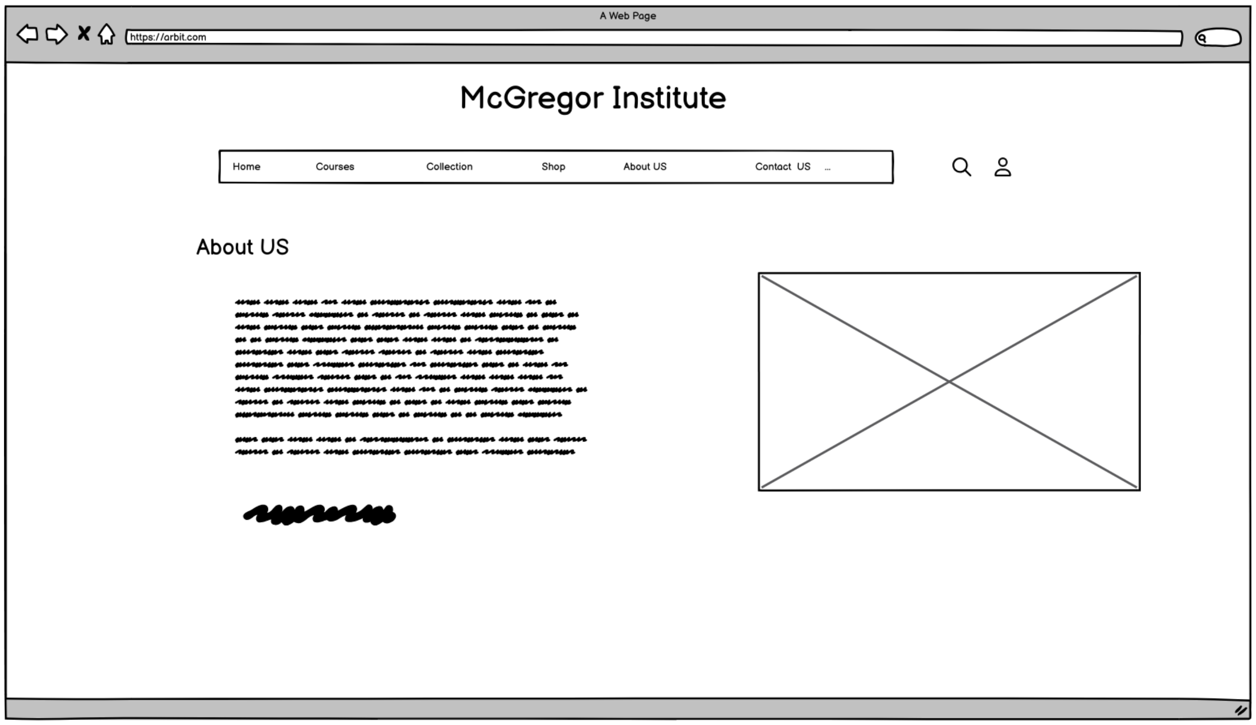


Figure 18 Prorotype of About Us

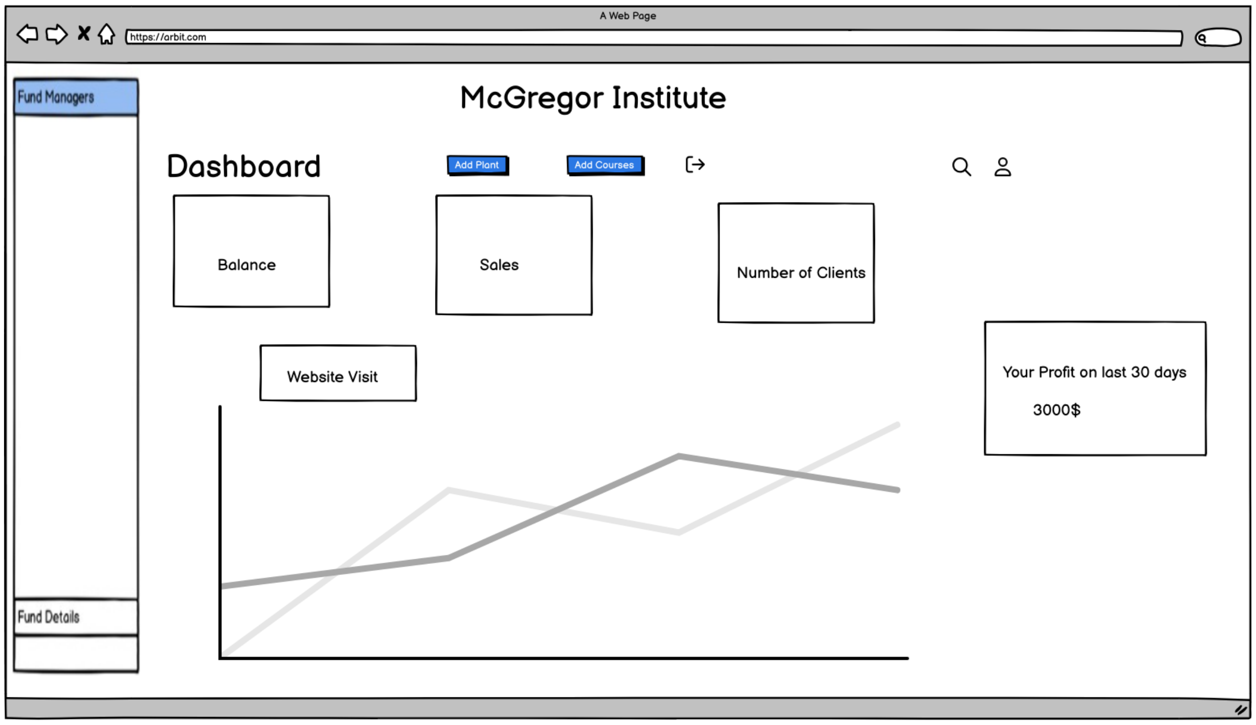


Figure 19 Prorotype of Admin Dashboard

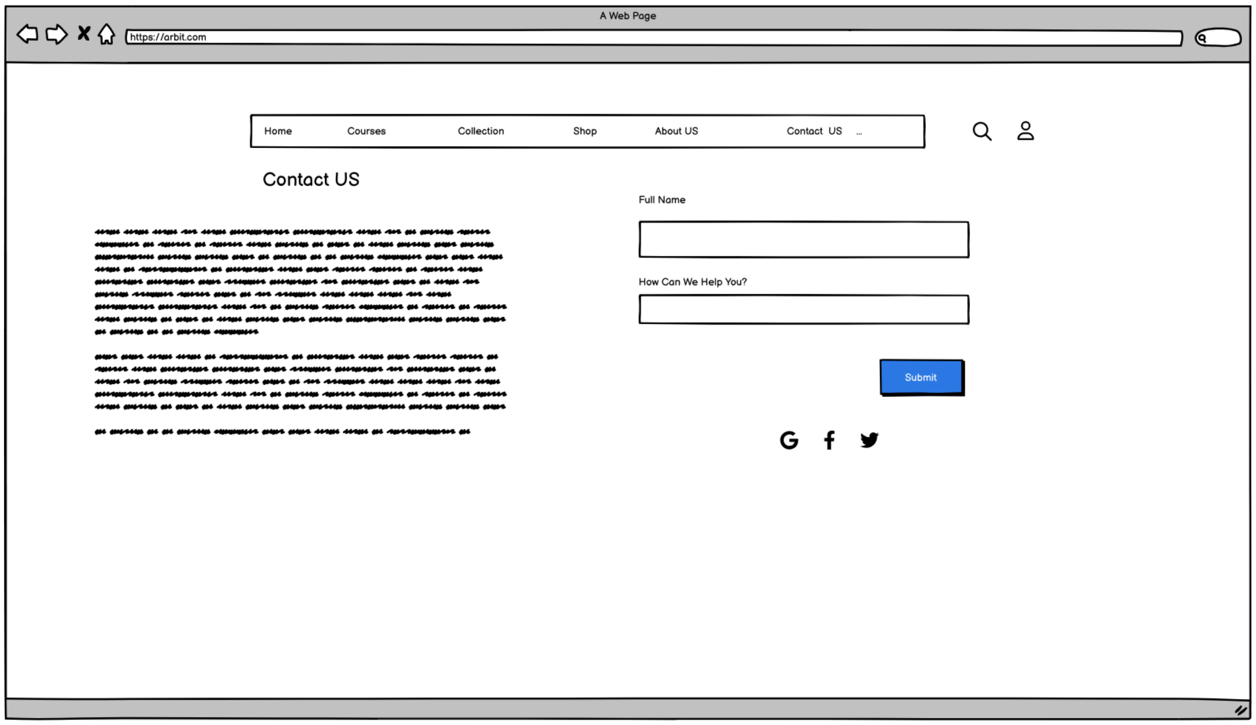


Figure 20 Prorotype of Contact Us

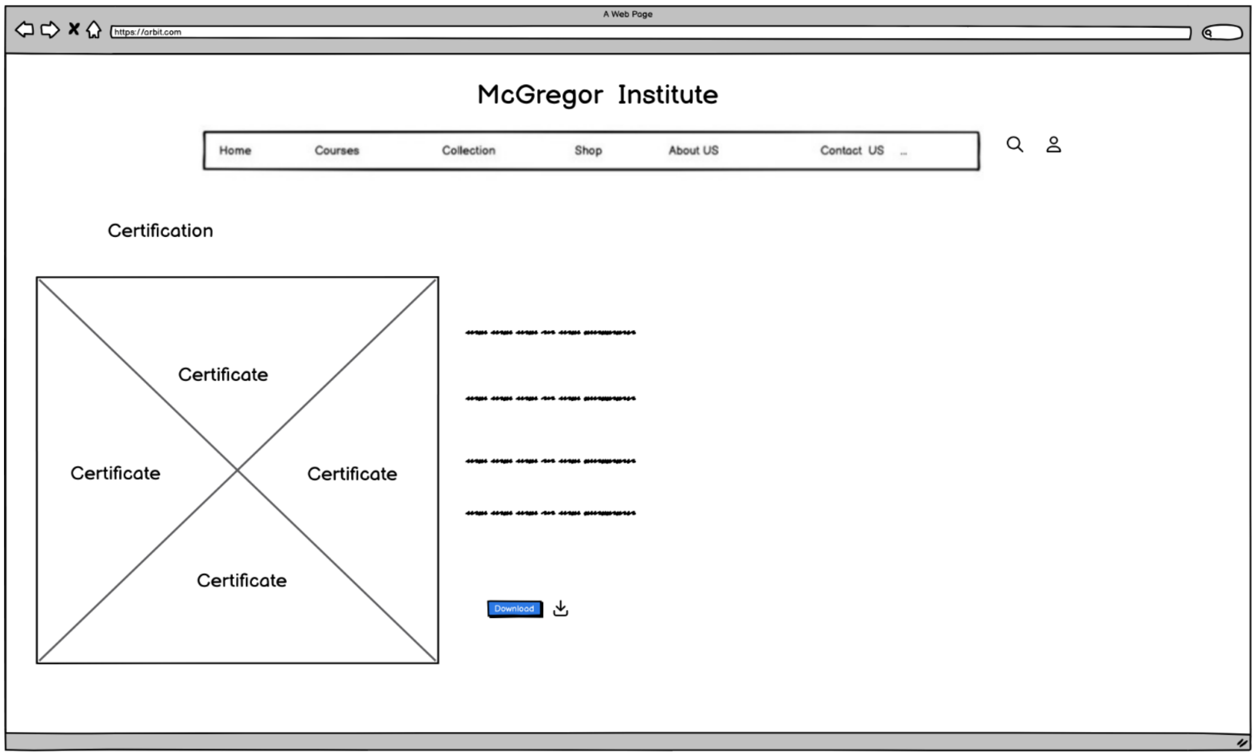


Figure 21 Prorotype of Certificate

8 Conclusion

In short, this coursework assignment aims to create a plan for building a software system specifically for the McGregor Institute of Botanical Training. It includes making different documents like schedules, diagrams, and models to understand what the system needs to do and how it will work. By doing this, it sets a strong foundation for knowing what the system needs and how to make it. It also decides how the system will be built, tested, and maintained. Creating design prototypes helps to visualize what the system will look like and how people will use it. Overall, this coursework shows how to develop software step by step, making sure it meets the institute's requirements and is easy to use for everyone involved.

9 References

GRANT, M. (2023, 04 26). *Investopedia*. Retrieved from Investopedia: https://www.investopedia.com/terms/g/gantt-chart.asp#:~:text=A%20Gantt%20chart%20is%20a,resources%2C%20planning%2C%20and%20dependencies.

javatpoint. (2024, 5 1). *javatpoint*. Retrieved from javatpoint: https://www.javatpoint.com/use-case-model

Lewis, S. (2024, 5 1). *techtarget*. Retrieved from techtarget: https://www.techtarget.com/searchsoftwarequality/definition/collaboration-diagram#:~:text=A%20collaboration%20diagram%2C%20also%20known,Unified%20Modeling%20Language%20(UML).

paradigm, v. (2024, 5 2). *visual-paradigm*. Retrieved from visual-paradigm: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

visual-paradigm. (2024, 5 1). *visual-paradigm*. Retrieved from visual-paradigm: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

workbreakdownstructure. (2024, 4 26). *workbreakdownstructure*. Retrieved from workbreakdownstructure: https://www.workbreakdownstructure.com/