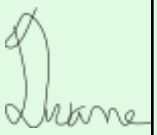
Re:forest

SIGNATURE BLOCK				
Statement: I did my share of the work, and I have a general understanding of the contents of the assignment.				
Team Member	**Contribution**	**% of Total**	**Signature**	**Date**
Amilcar Vasquez	*Report 1: Enumerated Functional Requirements / Report 1: Problem Statement* *Report 1.1: Use Cases (Traceability Matrix, Fully-Dressed Description, User Effort Estimation)* *Report 1.2: User Interface Specification* *Report 1.3: Identifying Subsystems* *Report 2.0: Data Model and Persistent Data Storage* *Report 2.1 Data Types and Operations* *Report 2.1 Traceability Matrix* *Report 2.2: User Interface Design and Implementation* *Report 2.2: Algorithms and Data Structures* *Report 3: Group effort*	14.29%		*5/19/2025*
Bryan Hernandez	*(Assist): Report 1: Glossary of Terms / (Assist): Report 1: User Stories / (Assist): Report 1: On-Screen Appearance Requirements* *(Assist): Report 1.1: User Interface Specification* *Report 1.2:System Sequence Diagrams* *Report 1.3: Global Control Flow* *Report 1.3: Hardware Requirements* *Report 2.0: Conceptual Model, Concept definitions, Association definitions, Attribute definitions, Traceability matrix* *Report 2.1: Interaction diagrams* *Report 2.2: User Interface Design and Implementation* *Report 3: Group effort*	14.29%		*1/27/2025*

Christian Hope	*(Assist): Report 1: Problem Statement* *Report 1.1: Functional Requirements Specification (Stakeholders, Actors & Goals)* *Report 1.2: User Interface Specification* *Report 1.3: Plan of Work* *Report 2.0: Conceptual Model, Data Model, and Persistent Data Storage* *Report 2.2: Project Management & Plan of Work* *Report 3: Group effort*	14.29%		*1/27/2025*
Daryn Forman	*Report 1: Enumerated Nonfunctional Requirements* *Report 1.2: Use Cases (Casual Description)* *Report 1.2: User Interface Specification* *Report 1.3: Architecture Styles* *Report 2.0: System Operation Contracts* *Report 2.1: Class Diagram* *Report 2.2: Project Management & Plan of Work* *Report 3: Group effort*	14.29%		*1/27/2025*
Duane Arzu	*Report 1.1: Glossary of Terms* *Report 1.2: System Sequence Diagrams* *Report 1.2: User Interface Specification* *Report 2.0: System Operation Contracts* *Report 3: Group effort*	14.29%		*1/27/2025*
Immanuel Garcia	*Report 1.1: Problem Statement / Report 1: User Stories* *Report 1.2: System Sequence Diagrams* *Report 1.2: User Interface Specification* *Report 1.3: Revised Report 1.3 / (Update): UML Package Diagram to spec. / (Update): System Sequence Diagram to spec. / (Assist): Identifying Architecture / (Assist): Architectural Styles* *(Assist) Report 2.0: Data Model and Persistent Data Storage, System Operation Contracts* *Report 2.0: Data Model and Persistent Data Storage* *Report 2.2: User Interface Design and Implementation* *Report 2.2: Algorithms and Data Structures* *Report 3: Group effort*	14.29%		*1/27/2025*

Miguel Vairez	*Report 1: On-Screen Appearance Requirements* *Report 1.2: User Interface Specification* *Report 1.3: Global Control Flow* *Report 2.0: Conceptual Model, Data Model and Persistent Data Storage* *Report 2.2: Design of Tests* *Report 3: Group effort*	14.29%		*1/27/2025*

Table of Contents

Table of Contents **4**

1. Customer Statement of Requirements **6**

a. Statement Description 6

2. Glossary 9

3. System Requirements **10**

Enumerated Functional Requirements 10

Enumerated Nonfunctional Requirements (FURPS+) 12

On-Screen Appearance Requirements 13

4. Functional Requirements Specification **14**

a. Stakeholders 14

b. Actors & Goals 15

c. Use Cases 16

i. Casual Description 16

ii. Use Case Diagram 18

iii. Traceability Matrix 19

iv. Fully Dressed Description 20

d. System Sequence Diagrams 23

5. Effort Estimation **28**

6. Domain Analysis and Modeling **32**

a. Domain Model 32

i. Concept definitions 32

ii. Association definitions 34

iii. Attribute definitions 35

iv. Domain Model Diagram 37

v. Traceability Matrix 38

b. System Operation Contracts 39

7. Interaction Diagrams **46**

a. Diagrams 46

b. Description 49

c. Design Patterns 50

8. Class Diagram & Interface Specification **52**

a. Class Diagram 52

b. Data Types and Operation Signatures 53

c. Traceability Matrix 57

d. OCL Contract Specifications 58

9. System Architecture **60**
a. Architecture Styles 60
b. UML Package Diagram 61
c. Subsystem-to-Hardware Mapping 63
d. Persistent Data Storage 64
e. Network Protocol 65
f. Global Control Flow 67
g. Hardware Requirements 68
10. Algorithms and Data Structures **69**
a. Algorithms 69
b. Data Structures 71
c. Concurrency 72
11. User Interface Design and Implementation **73**
Description 74
Ease-of-Use Implementation 75
Usability principles applied 75
User Effort Estimation 75
12. Design of Tests **76**
List of Test Cases (Use Cases 3 & 4) 76
Test Coverage 77
Integration Testing Strategy 78
Plan for Testing 78
13. History of Work **79**
b. Current Status - Accomplishments 80
c. Future Work & Opportunities 80
14. References **82**
15. Summary of Changes 82

1. Customer Statement of Requirements

a. Statement Description

Problem Statement. Belize undertakes significant reforestation initiatives to combat deforestation, promote biodiversity, and mitigate climate change. However, despite planting thousands of trees annually, the country lacks an effective system to track the survival rates and locations of these trees. Without proper monitoring, it becomes challenging to assess the effectiveness of these initiatives, identify areas requiring additional intervention, and make data-driven decisions for future planting efforts. Current methods, which may involve manual surveys or inconsistent reporting, result in inefficiencies and data gaps that hinder long-term sustainability goals.

Environmental Organization's Perspective. As an environmental organization focused on reforestation, we aim to plant trees and ensure their survival and long-term impact. We often receive funding and resources to support tree-planting initiatives. Still, without a robust tracking system, it is difficult to provide accurate reports on the success rates of these projects. Our current approach involves manually tagging and monitoring selected trees, which is time-consuming and not scalable. Furthermore, lacking a centralized database means different stakeholders, including government agencies, NGOs, and private landowners, struggle to share and analyze data effectively.

A digital tracking system would revolutionize how we manage our reforestation efforts. Ideally, we need a solution that allows us to:

- Record the exact location of newly planted trees.

- Monitor the health and survival rates of trees over time.
- Generate reports that demonstrate the impact of our reforestation projects.
- Enable community members and volunteers to contribute data using a simple mobile application.

Such a system would help us optimize resource allocation, improve accountability to donors, and ensure that our reforestation efforts lead to tangible, long-lasting environmental benefits.

Government Agency's Perspective As a government agency responsible for environmental sustainability, we need reliable data to develop policies and regulations that promote effective reforestation. Currently, our department receives reports from various organizations, but it is difficult to establish clear trends and make informed decisions due to inconsistencies in data collection methods. We often struggle with:

- Lack of standardized procedures for tree monitoring across different organizations.
- Inability to quickly assess the impact of reforestation projects at a national scale.
- Limited public engagement in reporting issues like illegal logging or tree die-offs.

A centralized digital platform that consolidates reforestation data would significantly enhance our ability to oversee national initiatives. This platform should provide:

- A user-friendly dashboard displaying real-time data on tree survival rates.
- Geographic heat maps showing areas with successful or struggling trees.
- Predictive analytics to determine where future planting efforts should be focused.
- The ability for citizens to report illegal logging activities or tree damage.

Such a system would help us enforce environmental policies more effectively and foster greater public participation in conservation efforts.

Community Volunteer's Perspective. As a volunteer involved in tree-planting efforts, I am passionate about making a difference in my local environment. However, one of the most frustrating aspects of volunteering is not knowing whether the trees I planted survive beyond the initial planting phase. Many of us return to the same sites months later, only to find that trees have been lost to drought, pests, or human activity.

A mobile application that allows volunteers to log tree plantings and follow up on their growth would be incredibly valuable. Such an app should include:

- An easy-to-use interface for entering tree data, including species, location, and planting date.
- Photo upload capabilities to document growth progress.
- Notifications reminding volunteers to check on trees they planted.
- The ability to collaborate with others and share updates on community efforts.

With such a tool, volunteers like me would feel more engaged and motivated to continue participating in reforestation projects, knowing that our efforts are making a measurable impact.

Conclusion. A robust tree-tracking system is crucial to ensuring the success of Belize's reforestation initiatives. By leveraging technology to collect, analyze, and visualize data, we can improve the survival rates of newly planted trees, optimize resource allocation, and foster greater community engagement. This system would provide environmental organizations, government agencies, and volunteers with the necessary tools to work together toward a more sustainable future for Belize's forests.

2. Glossary

1. **Reforestation**
    - *Definition*: The process of planting trees where the forest has been depleted due to activities like logging, agriculture, or natural disasters.
2. **Afforestation**
    - *Definition*: Establish a forest or tree stand in an area with no previous tree cover.
3. **Deforestation**
    - *Definition*: Humans clear or thin forests to make the land available for other uses.
4. **Agroforestry**
    - *Definition*: A land-use management system where trees or shrubs are grown around or among crops or pastureland, integrating agricultural and forestry practices for sustainable benefits.
5. **Carbon Sequestration**
    - *Definition*: The process by which trees and plants absorb carbon dioxide from the atmosphere, helping to mitigate climate change.
6. **Biodiversity**
    - *Definition*: The variety of plant and animal life in a particular habitat, indicating the health of an ecosystem.
7. **Ecosystem Services**
    - *Definition*: The benefits humans receive from natural environments, such as clean water, air, and pollination of crops.
8. **Silviculture**
   Definition: The practice of controlling the establishment, growth, composition, and quality of forests to meet diverse needs and values.
9. **Forest Degradation**
    - *Definition*: The reduction in the capacity of a forest to provide goods and services, often due to human activities like overharvesting or pollution.
10. **Remote Sensing**
    - *Definition*: Using satellite or aerial imagery to collect information about the Earth's surface, commonly used in monitoring forest cover and health.

11. **Dendrology**
    - The scientific study of trees and woody plants, often used to determine the best species for reforestation efforts
12. **Tree Mortality Rate**
    - The percentage of trees that do not survive over a given period due to environmental conditions, disease, or human activity.

3. System Requirements

Enumerated Functional Requirements

REQ ID	Priority	Requirement Description
REQ-1	High	The system shall allow users to register and create an account.
REQ-2	High	Users shall be able to log in securely using modern authentication mechanisms.
REQ-3	High	The system shall provide an interface to add new reforestation projects, including location, tree species, and planting dates.
REQ-4	High	The system shall display reforestation areas on an **interactive map**, showing locations where trees have already been planted to prevent redundancy.
REQ-5	Medium	The system shall provide **search and filtering options** for specific reforestation projects based on date, location, or status.

REQ-6	Medium	The system shall allow users with the `Environmentalist` role to **upload images and reports** about tree growth and site conditions.
REQ-7	High	The system shall allow administrators to **verify and approve** submitted data before publication.
REQ-8	Medium	The system shall generate **automated reports** on reforestation progress, including statistics on trees planted, survival rates, and environmental impact.
REQ-9	Medium	The system shall send **notifications and alerts** for upcoming maintenance activities, such as watering or replanting.
REQ-10	Medium	The system shall integrate with **mapping APIs** (such as Google Maps) to display reforestation sites with geolocation support.

Enumerated Nonfunctional Requirements (FURPS+)

Identifier	PW	Requirement
NONREQ-1	3	The system shall provide all users with a **non-cluttered, user-friendly,** and **intuitive interface**.
NONREQ-2	5	The system shall display an **interactive map** for locating and tracking trees efficiently.
NONREQ-3	4	The system shall provide a **mobile-friendly interface optimized** for various screen sizes.
NONREQ-4	3	The system shall **support user preferences** for notifications, such as growth milestones.
NONREQ-5	4	The system shall allow administrators **to monitor and manage user** contributions effectively.
NONREQ-6	2	The mobile application **shall provide a visual history** of conservation efforts in specific areas.
NONREQ-7	4	The system **shall ensure high security for user data**, including encryption and secure logins.
NONREQ-8	3	The system shall **log and provide detailed analytics on reforestation progress**.
NONREQ-9	2	The system shall **comply with environmental data standa**rds and best practices.

NONREQ-10	4	The system shall **provide notifications** to users.
NONREQ-11	2	The system shall allow integration with third-party apps for extended functionality.
NONREQ-12	5	The system **shall offer automated backup and recovery of user data.**

On-Screen Appearance Requirements

ID	PW	Requirement
ONSREQ-1	3	The app interface should feature a clean and organized layout with clearly defined sections for different safety features and resources, ensuring ease of navigation and user understanding.
ONSREQ-2	3	Text displayed on the app interface should be legible and accessible to users of all ages and visual abilities, with adjustable font sizes to accommodate individual preferences.
ONSREQ-3	3	The app should utilize a color scheme that enhances readability and visual clarity, with appropriate contrast between text and background elements to aid users with visual impairments
ONSREQ-4	3	The use of intuitive icons and graphical elements should be incorporated throughout the app interface to visually communicate key safety features, actions, and alerts, enhancing user comprehension and engagement.
ONSREQ-5	2	The app interface should be designed to adapt seamlessly to different screen sizes and resolutions, ensuring optimal viewing and interaction experiences across a wide range of mobile devices and tablets.

4. Functional Requirements Specification

a. Stakeholders

Stakeholder	Role	Interest	Involvement
Environmentalists	Conservation advocates and sustainability experts	Tracking reforestation, analyzing environmental data, and recommending best practices	Monitoring progress, providing insights, and ensuring ecological sustainability
Belize Forest Department	Regulatory authority overseeing reforestation efforts	Ensuring compliance with forestry policies, verifying data accuracy, and managing reforestation resources	Approving reports, enforcing guidelines, and using system analytics for decision-making
General Users	Individuals or organizations interested in reforestation	Exploring ongoing projects, contributing data, and supporting conservation efforts	Browsing projects, submitting environmental data, and engaging in tree-planting initiatives

b. Actors & Goals

Actor	Role	Goal
Environmentalists	Initiating reforestation efforts.	An environmentalist is responsible for adding new tree projects. In the context of Re: Forest, this may be any person who advocates for reforestation initiatives.
Users	General participants engaging with the system	Users access relevant data for personal or organizational initiatives, mainly viewing and tracking projects.
System Administrators	Manage user roles, verify data, and maintain system functionality	An administrator ensures the system runs efficiently, verifies data integrity, and oversees user access and permissions

c. Use Cases

i. Casual Description

UC1: RegisterUser

- **Description:** A new user registers for an account by entering their name, email, and password. The system validates the input, checks for duplicate accounts, and securely stores the user's credentials.
- **Related Requirement(s):** REQ-1

UC2: Login User

- **Description:** A registered user logs into the system using their email and password. The system authenticates the user and grants access to the dashboard. If login fails, an error message is displayed.
- **Related Requirement(s):** REQ-2

UC3: AddProject

- **Description:** An authenticated Environmentalist user adds a new reforestation project by entering the project's location (via map), tree species, and planting date. The system stores this data and updates the interactive map to reflect the new project
- **Related Requirement(s):** REQ-3, REQ-4

UC4: ViewProject

- **Description:** Any user can view reforestation projects on an interactive map. Each project displays metadata like species planted and planting date. Already-planted areas are visually marked to avoid redundancy. Reports can be viewed by Admins and environmentalists
- **Related Requirement(s):** REQ-4, REQ-10, REQ-8

UC5: SearchProjects

- **Description:** Users apply filters such as location, date, or status to find specific reforestation projects. The system dynamically displays matching results
- **Related Requirement(s):** REQ-5

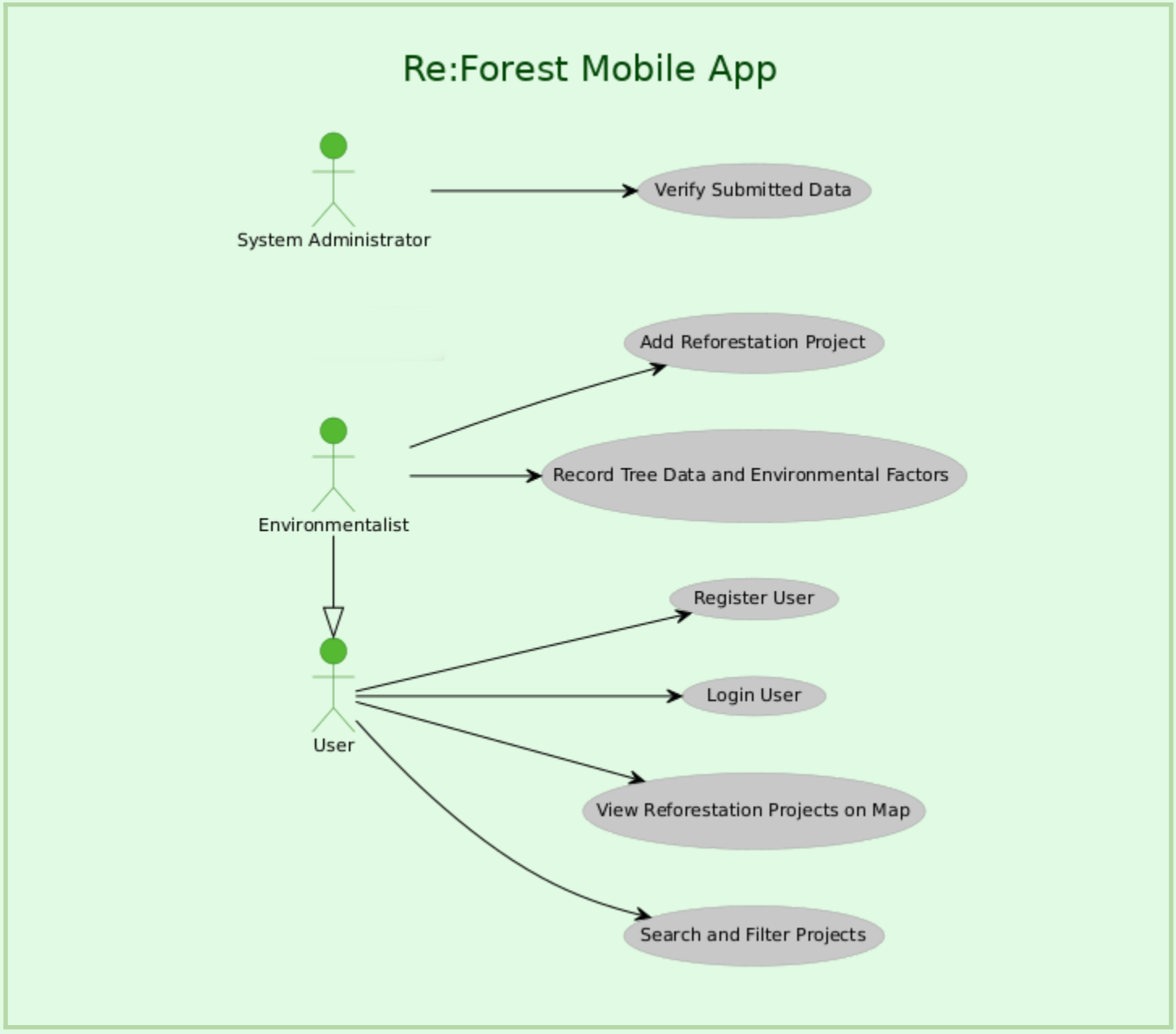
UC6: RecordTreeData (Environmentalist)

- **Description:** Users with the Environmentalist role upload images and progress reports for a project, documenting tree growth, soil condition, and site status. The system attaches this data to the relevant project for administrative review.
- **Related Requirement(s):** REQ-6

UC7: VerifyUserData

- **Description:** Admins review new project entries, reports, and image uploads submitted by Environmentalists. They verify the content and approve for publication or reject it with feedback, notifying of the action taken.
- **Related Requirement(s):** REQ-7, REQ-9

ii. Use Case Diagram



iii. Traceability Matrix

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7
REQ-1	3	X						
REQ-2	3		X					
REQ-3	3			X				
REQ-4	3			X	X			
REQ-5	2					X		
REQ-6	3						X	
REQ-7	3						X	X
REQ-8	2							
REQ-9	3							
REQ-10	2				X			
Max PW		**3**	**3**	**3**	**3**	**2**	**3**	**3**
Total PW	**28**	**3**	**3**	**6**	**6**	**2**	**3**	**6**

iv. Fully Dressed Description

UC3: AddProject

Initiating Actor	Environmental Researcher / Organization
Actor's Goal	Submit a new reforestation project proposal
Participating Actors	System
Preconditions	The user must be logged into the system and have the necessary permissions to submit a project
Postconditions	The project is successfully created and stored in the database for further approval

Flow of Events for Main Success Scenario

Step	Actor Action
1.	**User** selects "Create New Project" from the dashboard.
2.	**User** fills out the project form, including the project name, location, and tree species.
3.	**User** submits the project for review.
4.	**System** validates the input data and ensures all required fields are completed.
5.	**System** saves the project to the database and assigns it a unique ID.
6.	**System** notifies of new project submission.
7.	**User** receives a confirmation message stating the project has been submitted for review.

Flow of Events for Main Unsuccessful Scenario

Step	Actor Action
5a)	If the input data is incomplete, the **System** prompts the user to fill in missing fields.
6a)	If system validation fails, the **System** displays an error message, and the project is not submitted.

UC4: ViewProject

Initiating Actor	General User / Researcher
Actor's Goal	View reforestation projects on an interactive map
Initiating Actors	System
Preconditions	The user is logged in and has access to the interactive map feature
Postconditions	The user successfully views project locations and related details

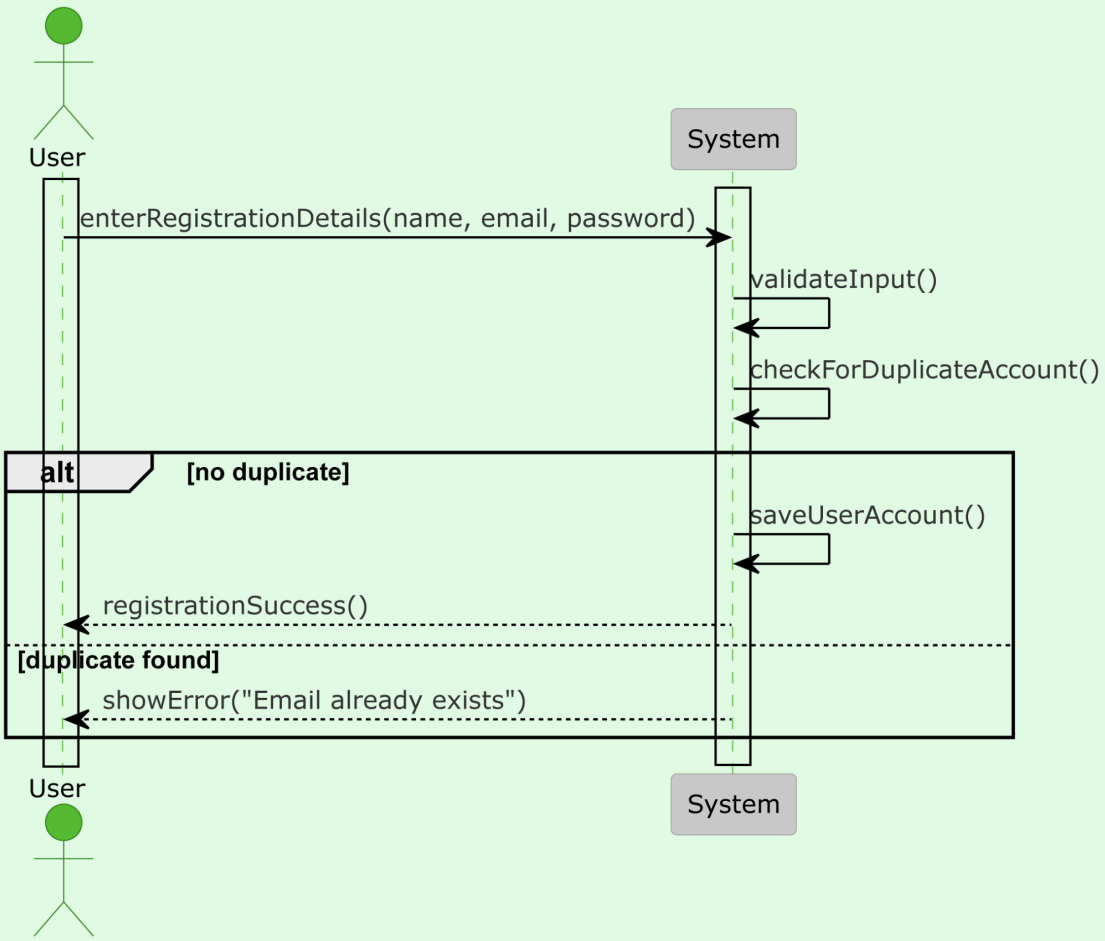
Flow of Events for Main Success Scenario

Step	Actor Action
1.	**User** navigates to the "Projects Map" section.
2.	**System** loads an interactive map with geotagged project locations.
3.	**User** zooms and pans the map to explore different regions.
4.	**User** clicks on a project marker.
5.	**System** displays a dialogue with project details.
6.	**User** clicks "View More" to see detailed information.
7.	**System** redirects the user to the full project page.

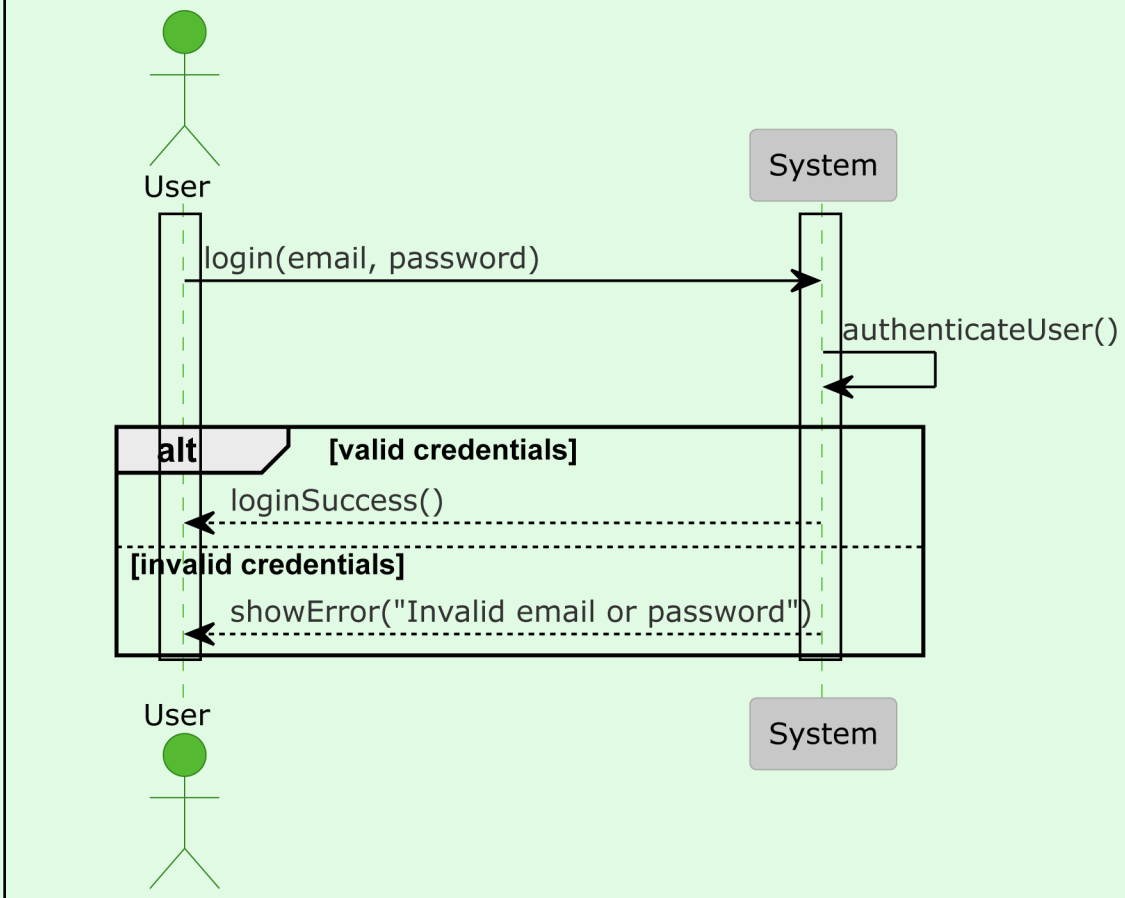
Flow of Events for Main Unsuccessful Scenario

Step	Actor Action
2a)	If the map fails to load due to network issues, the **System** displays an error message and suggests retrying.
5a)	The **System** displays a “Data Not Available” message if project data is missing.

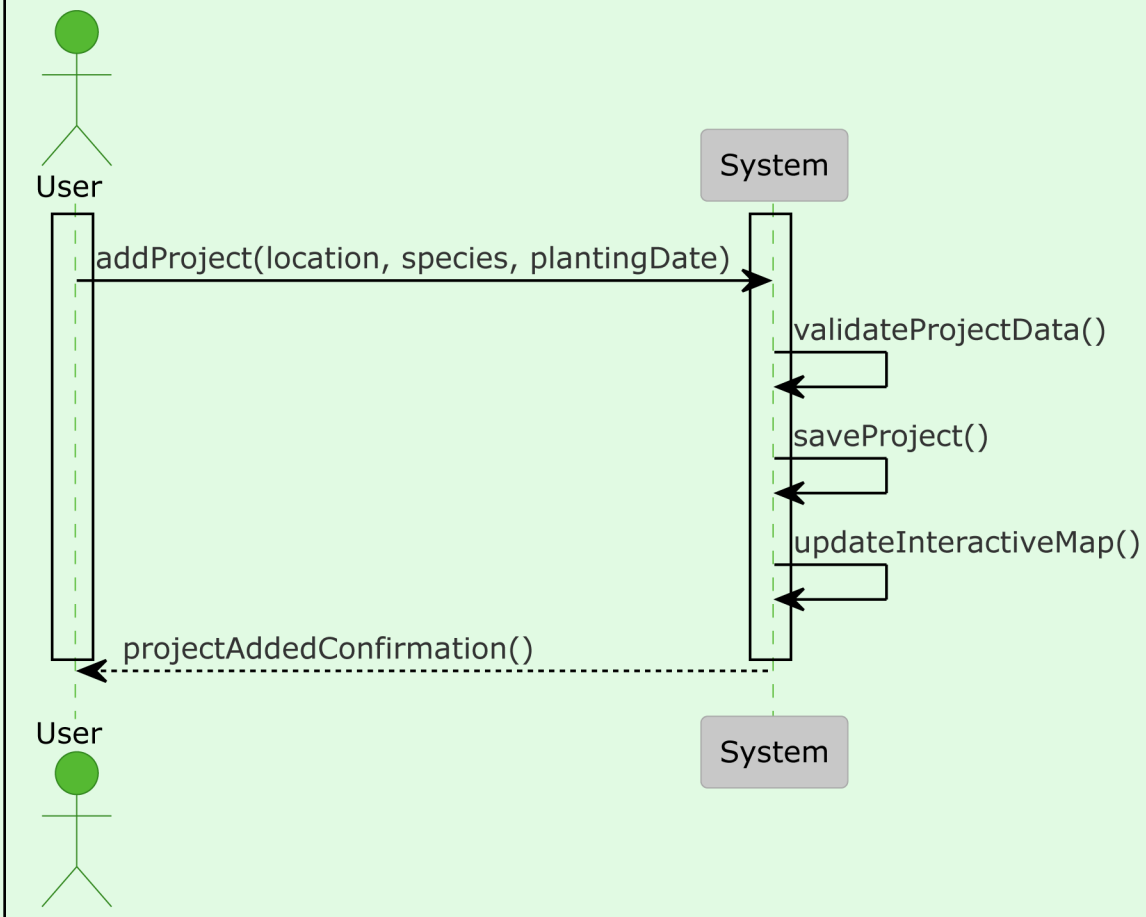
d. System Sequence Diagrams

Use Case	**System Sequence Diagram**
UC1: RegisterUser	

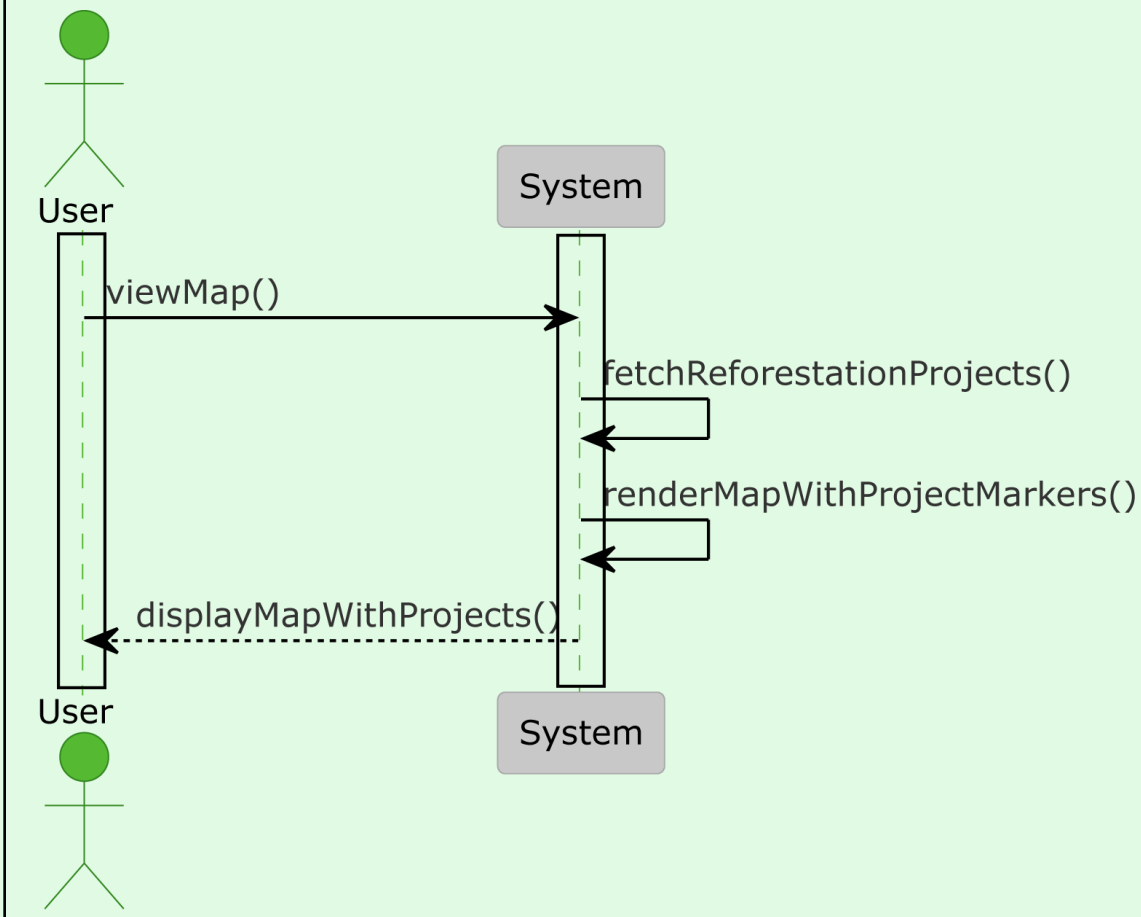
UC2: LoginUser



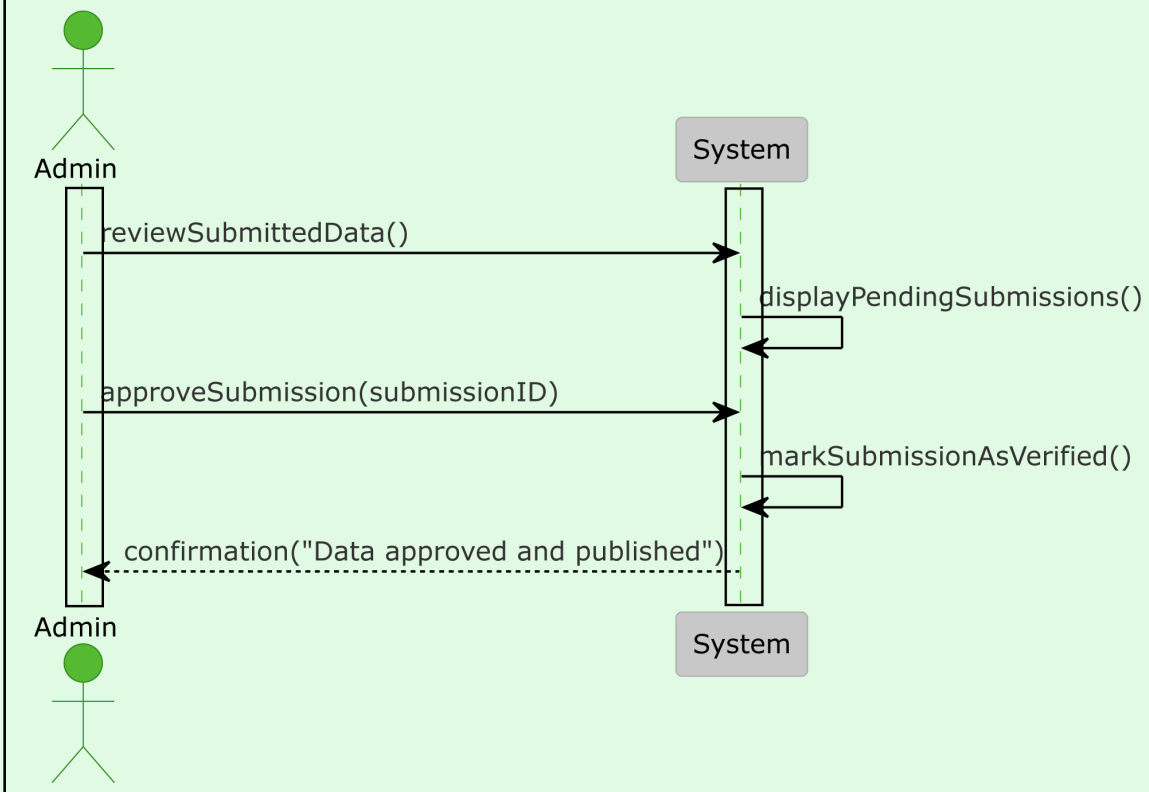
UC3: AddProject



UC4: ViewProject



UC7: VerifyUserData



5. Effort Estimation

Complexity

Complexity	Transactions	Weight
Simple	≤ 3	5
Average	4 – 7	10
Complex	≥ 7	15

Use Case	Est. Transactions	Complexity	Weight
UC1: RegisterUser	3	Simple	5
UC2: LoginUser	3	Simple	5
UC3: AddProject	6	Average	10
UC4: ViewProject	4	Average	10
UC5: SearchProjects	5	Average	10
UC6: RecordTreeData	6	Average	10
UC7: VerifyUserData	4	Average	10

Calculate Use Case Weight (UUCW)

➔ Add the weights of all use cases:

➔ UUCW = 5 + 5 + 10 + 10 + 10 + 10 + 10 = 60

Assign Weight:

Actor Type	Description	Weight
Simple	API or another system	1
Average	Interactive or protocol-based	2
Complex	GUI with human interaction	3

- 1 Admin (Complex – 3)
- 1 Environmentalist (Complex – 3)
- 1 General User (Complex – 3)

UAW = 3 + 3 + 3 = 9

Calculate Unadjusted Use Case Points (UUCP)

UUCP = UUCW + UAW

UUCP = 60 + 9 = 69

Compute Technical Complexity Factor (TCF)

Factor	Weight	Est. Rating	Product
T1: Distributed System	2	3	6
T2: Performance	1	3	3
T3: End-user efficiency	1	3	3
T4: Complex processing	1	3	3
T5: Reusability	1	2	2
T6: Easy to install	0.5	2	1
T7: Easy to use	0.5	3	1.5
T8: Portable	2	2	4
T9: Easy to change	1	3	3
T10: Concurrent	1	2	2
T11: Security	1	3	3
T12: Access third-party API	1	2	2
T13: User training required	1	2	2

➔ TF = Sum of (Weight × Rating) = 36.5

➔ TCF = 0.6 + (0.01 × TF) = 0.6 + 0.365 = 0.965

Compute Environmental Complexity Factor (ECF)

Factor	Weight	Est. Rating	Product
E1: Familiarity with dev.	1.5	3	4.5
E2: Application experience	0.5	3	1.5
E3: OO experience	1	4	4
E4: Lead analyst capability	0.5	4	2
E5: Motivation	1	4	4
E6: Stable requirements	2	3	6
E7: Part-time workers	-1	2	-2
E8: Difficult programming	-1	2	-2

➔ EF = Sum = 18

➔ ECF = 1.4 + (-0.03 × EF) = 1.4 - 0.54 = 0.86

Calculate Use Case Points (UCP)

➔ UCP = UUCP × TCF × ECF

➔ UCP = 69 × 0.965 × 0.86 ≈ 57.16

Estimate Effort

- Given:

  Productivity Factor (PF) = 28 hours per UCP

- Effort = UCP × PF = 57.16 × 28 ≈ 1600.48 hours

Estimated Total Effort: ~1600 hours

6. Domain Analysis and Modeling

a. Domain Model

i. Concept definitions

UC1: RegisterUser

Type	Concept Name	Concept Definition
K	User	Represents an individual registering to use the system.
D	Authentication Controller	Handles user registration, input validation, and secure storage of credentials.

UC2: LoginUser

Type	Concept Name	Concept Definition
K	User	A registered individual is attempting to access the system.
D	Authentication Controller	Validates credentials and grants access to system functionality.

UC3: AddProject

Type	Concept Name	Concept Definition
K	User (Environmentalist)	An authorized user role responsible for initiating reforestation projects.
K	Tree (Project)	Represents a tree entry in a reforestation project. Contains metadata like location, planting date.
K	Tree Species	Indicates the type of tree planted.
D	Data Controller	Stores project data including location, species, and planting info.

UC4: ViewProject

Type	Concept Name	Concept Definition
K	User	System users who view available reforestation projects.
D	Data Controller	Retrieves and displays project details and tree locations.

UC5: SearchProjects

Type	Concept Name	Concept Definition
K	User	Performs filtered searches on tree projects.
D	Data Controller	Handles query logic to return matching tree data based on criteria.

UC6: RecordTreeData

Type	Concept Name	Concept Definition
K	User (Environmentalist)	Inputs environmental and soil-related data per reforestation site.
K	Tree	Updated with environmental data.
D	Data Controller	Stores and associates environment data with relevant tree records.

UC7: VerifyUserData

Type	Concept Name	Concept Definition
K	Submission	Record submitted by users needing validation.
D	Verification Controller	Reviews and approves or rejects submissions.
D	Notification System	Notifies users of the verification outcome.

ii. Association definitions

UC1: RegisterUser

Concept Pair	Association Description	Association Name (Action)
User ↔ Authentication Controller	User data is validated and stored via the controller.	registers user

UC2: LoginUser

Concept Pair	Association Description	Association Name (Action)
User ↔ Authentication Controller	User credentials authenticated by the controller.	logs in

UC3: AddProject

Concept Pair	Association Description	Association Name (Action)
User ↔ Tree	Environmentalist adds tree data to the system.	adds project
Tree ↔ Tree Species	Each tree is assigned a species.	is of the species
User ↔ Data Controller	User submits project data via controller.	submits project data

UC4: ViewProject

Concept Pair	Association Description	Association Name (Action)
User ↔ Data Controller	Data Controller fetches data for display.	views project

UC5: SearchProjects

Concept Pair	Association Description	Association Name (Action)
User ↔ Data Controller	Data Controller returns filtered data.	searches projects

UC6: RecordTreeData

Concept Pair	Association Description	Association Name (Action)
User ↔ Tree	Environmentalist adds environmental data.	records environmental data
User ↔ Data Controller	Sends environmental data to controller.	submits environment data

UC7: VerifyUserData

Concept Pair	Association Description	Association Name (Action)
Submission ↔ Verification Controller	Verification Controller processes submissions.	verifies submission
Notification System ↔ User	Notifies user of verification status.	notifies user

iii. Attribute definitions

UC1 & UC2: User / Authentication Controller

Concept Name	Attribute	Description
User	id	Primary Key
	first_name	First Name of the user
	last_name	Last Name of the user
	email	Email address
	password	Password (stored securely)
Role	id	Primary Key
	name	Role title (e.g., admin, environmentalist)

UC3 – UC6: Tree / Tree Species / Data Controller

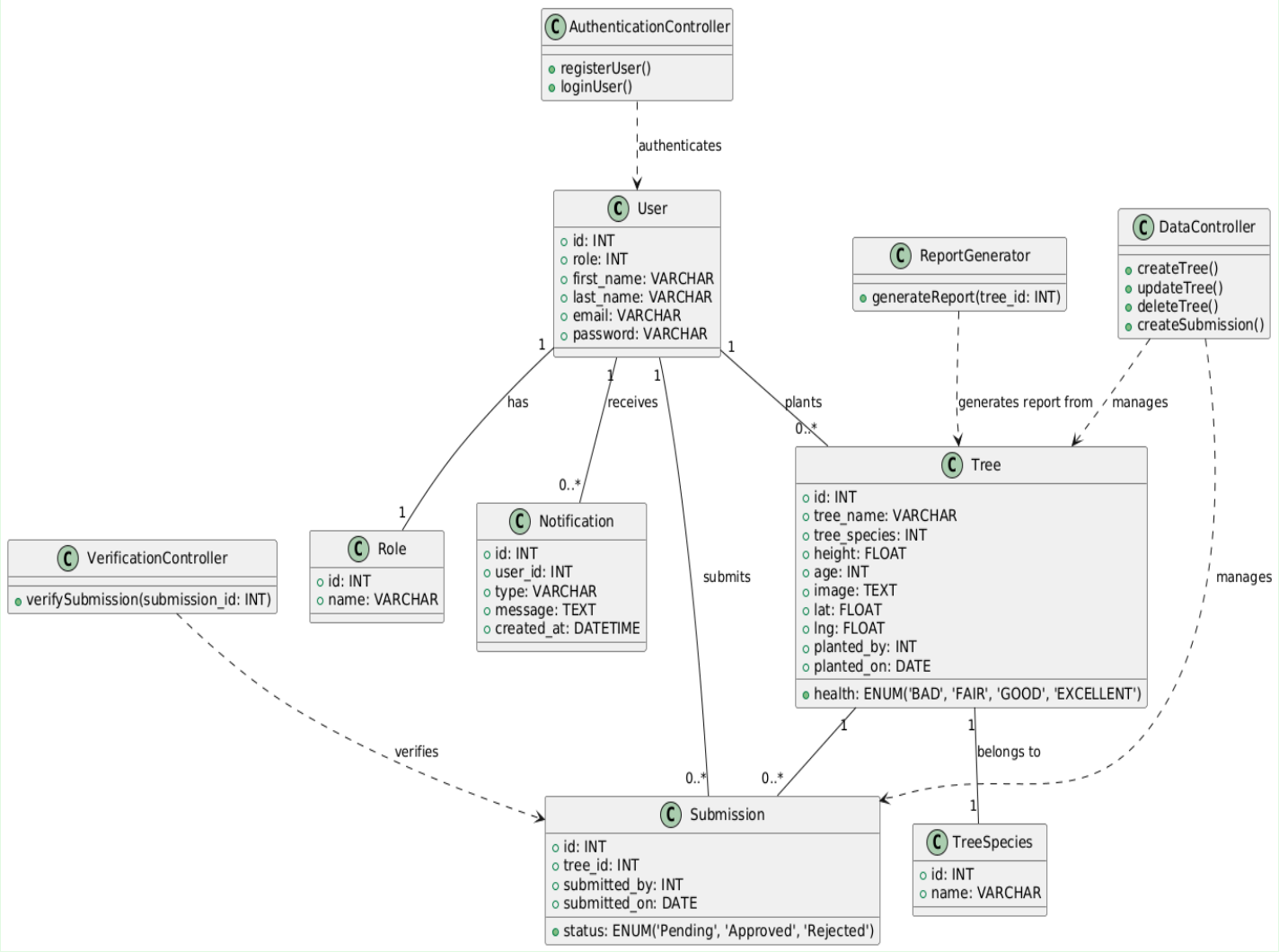
Concept Name	Attribute	Description
Tree	id	Primary Key

	tree_name	Label or nickname for the tree
	tree_species	Foreign Key referencing Tree Species
	height	Measured height
	health	Enum: BAD, FAIR, GOOD, EXCELLENT
	age	Age estimate
	image	Associated image file
	lat	Latitude
	lng	Longitude
	planted_by	Foreign Key referencing User
	planted_on	Date of planting
Tree Species	id	Primary Key
	name	Common or scientific name

UC7: Submission / Verification Controller / Notification System

Concept Name	Attribute	Description
Submission	id	Primary Key
	tree_id	Foreign Key to Tree
	submitted_by	Foreign Key to User
	submitted_on	Date of submission
	status	Enum: Pending, Approved, Rejected
Verification Controller	-	Processes based on rules (logic defined separately)
Notification System	id	Primary Key
	message	Text message sent to user

iv. Domain Model Diagram



v. Traceability Matrix

Use Case	User	Role	Tree	Tree Species	Authentication Controller	Data Controller	Notification	Submission	Report Generator	Verification Controller
Register User	✔	✔			✔					
Login User	✔				✔					
Add Project	✔		✔	✔		✔	✔			
View Project	✔		✔	✔		✔				
Search Projects	✔		✔	✔		✔				
Record Tree Data	✔		✔	✔		✔	✔	✔	✔	
Verify User Data	✔	✔						✔		✔

b. System Operation Contracts

Register User

Operation	`registerUser(name: String, email: String, password: String)`
Responsibilities	`Register the user into the system`
Cross References	Use Cases: RegisterUser
Preconditions	- The email is not already registered in the system. - The password meets the system's security requirements.
Postconditions	- A new User instance u was created (instance creation). - `u.name` became `name` (attribute modification). - `u.email` became `email` (attribute modification). - `u.password` became `password` (attribute modification). - `u` was associated with the system (association formed).

Login User

Operation	`loginUser(email: String, password: String)`
Responsibilities	`Authenticate the user into the system`
Cross References	Use Case: LoginUser
Preconditions	- The email is registered in the system.

Operation	`loginUser(email: String, password: String)`
Responsibilities	`Authenticate the user into the system`
	- The password matches the stored password for the email.
Postconditions	- The User instance u was authenticated (attribute modification). - `u` was granted access to the system (association formed).

View Project

Operation	`viewProject(projectID: String)`
Responsibilities	`Return project data`
Cross References	Use Case: ViewProject
Preconditions	- The user is logged in. - The `projectID` exists in the system.
Postconditions	- The Project instance p was retrieved from the database (association formed). - `p` details were displayed to the user (attribute modification).

Search Projects

Operation	`searchProjects(filterCriteria: FilterCriteria)`
Responsibilities	`Filter the projects based on a criterion`
Cross References	Use Case: SearchProjects
Preconditions	- The user is logged in. - The filter criteria are valid.
Postconditions	- A list of Project instances matching the filter criteria was retrieved (association formed). - The list was displayed to the user (attribute modification).

Record Tree Data

Operation	`recordTreeData(treeID: String, soilComposition: String, environmentalFactors: EnvironmentalData)`
Responsibilities	`Save inputted data into the database as a 'Tree'`
Cross References	Use Case: RecordTreeData
Preconditions	- The user is an environmentalist with the necessary permissions. - The `treeID` exists in the system.
Postconditions	- The Tree instance `t` was updated with the new soil composition (attribute modification). - `t.environmentalFactors` became `environmentalFactors` (attribute modification).

Operation	`recordTreeData(treeID: String, soilComposition: String, environmentalFactors: EnvironmentalData)`
Responsibilities	`Save inputted data into the database as a 'Tree'`
	- `t` was associated with the Project instance `p` (association formed). - `t` was stored in the database (association formed).

Verify User Data

Operation	`verifyUserData(submissionID: String)`
Responsibilities	`Verify the submitted data before creating an instance of a data structure`
Cross References	Use Case: VerifyUserData
Preconditions	- The user is an administrator. - The `submissionID` exists in the system.
Postconditions	- The Submission instance `s` was marked as verified (attribute modification). - `s

Database Schema

File Format

As mentioned, the database will be in PostgreSQL so the predominant file format will be SQL. However, for import/export operations, JSON or CSV files may be used. This may include importing tree species from an API such as Species+, which will facilitate using tree species with correct scientific names mapped to localized names. Below is the **database schema** with descriptions:

Database Schema & Descriptions

Table: `Role`
This table stores the different roles a user can have.

Column Name	Data Type	Constraints	Description
`id`	INT (Primary Key)	AUTO_INCREMENT	Unique identifier for each role
`name`	VARCHAR(255)	NOT NULL, UNIQUE	Name of the role (e.g., Admin, User, Environmentalist)

Table: `User`
This table stores user information.

Column Name	Data Type	Constraints	Description
`id`	INT (Primary Key)	AUTO_INCREMENT	Unique identifier for each user
`role`	INT (Foreign Key)	REFERENCES `Role(id)`, NOT NULL	User's role
`first_name`	VARCHAR(255)	NOT NULL	First name of the user
`last_name`	VARCHAR(255)	NOT NULL	Last name of the user
`email`	VARCHAR(255)	NOT NULL, UNIQUE	User's email address
`password`	VARCHAR(255)	NOT NULL	Hashed password

Table: `Tree_Species`
This table stores different tree species' names.

Column Name	Data Type	Constraints	Description
`id`	INT (Primary Key)	AUTO_INCREMENT	Unique identifier for each species
`name`	VARCHAR(255)	NOT NULL, UNIQUE	Name of the tree species

Table: `Tree`
This table stores information about trees planted in the system.

Column Name	Data Type	Constraints	Description
`id`	INT (Primary Key)	AUTO_INCREMENT	Unique identifier for each tree
`tree_name`	VARCHAR(255)	NOT NULL	A custom name given to the tree

Column Name	Data Type	Constraints	Description
`tree_species`	INT (Foreign Key)	REFERENCES `Tree_Species(id)`, NOT NULL	The species of the tree
`height`	FLOAT	NULLABLE	Height of the tree in meters
`health`	ENUM('BAD', 'FAIR', 'GOOD', 'EXCELLENT')	NOT NULL	Health condition of the tree
`age`	INT	NULLABLE	Age of the tree in years
`image`	TEXT	NULLABLE	URL or file path of the tree image
`lat`	FLOAT	NOT NULL	Latitude coordinate of the tree location
`lng`	FLOAT	NOT NULL	Longitude coordinate of the tree location
`planted_by`	INT (Foreign Key)	REFERENCES `User(id)`, NOT NULL	User who planted the tree
`planted_on`	DATE	NOT NULL	Date the tree was planted

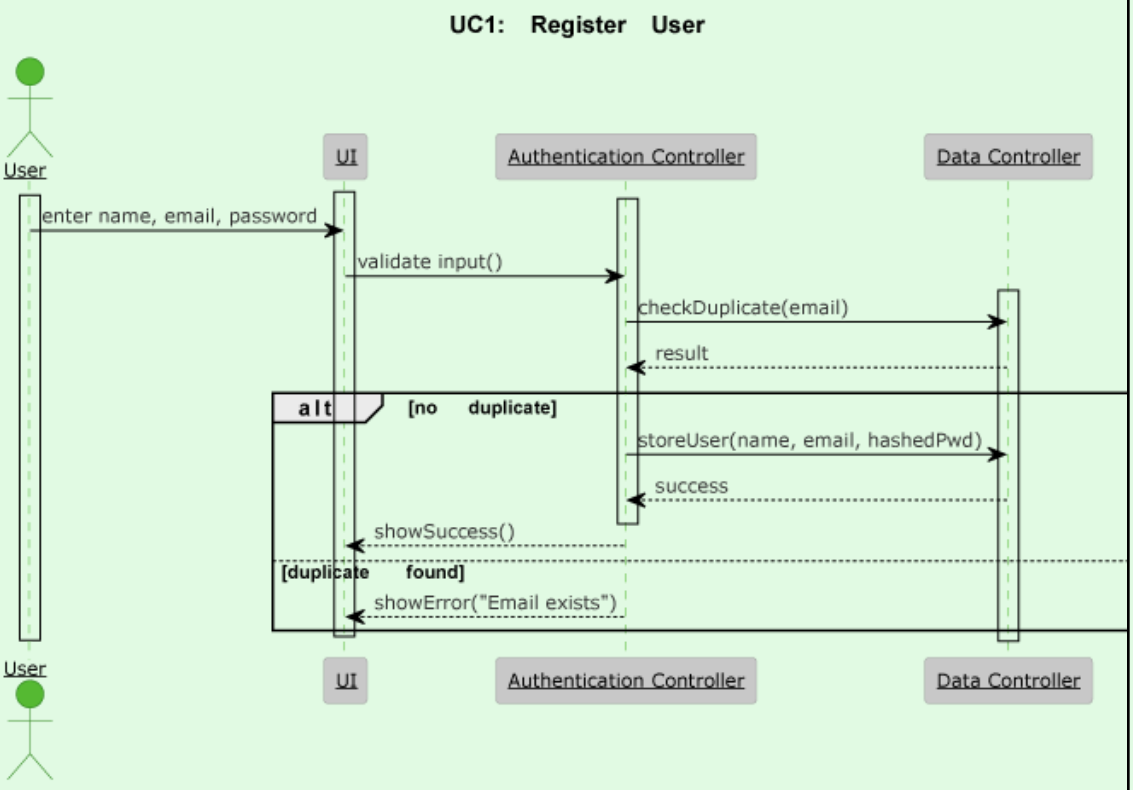
Relationships

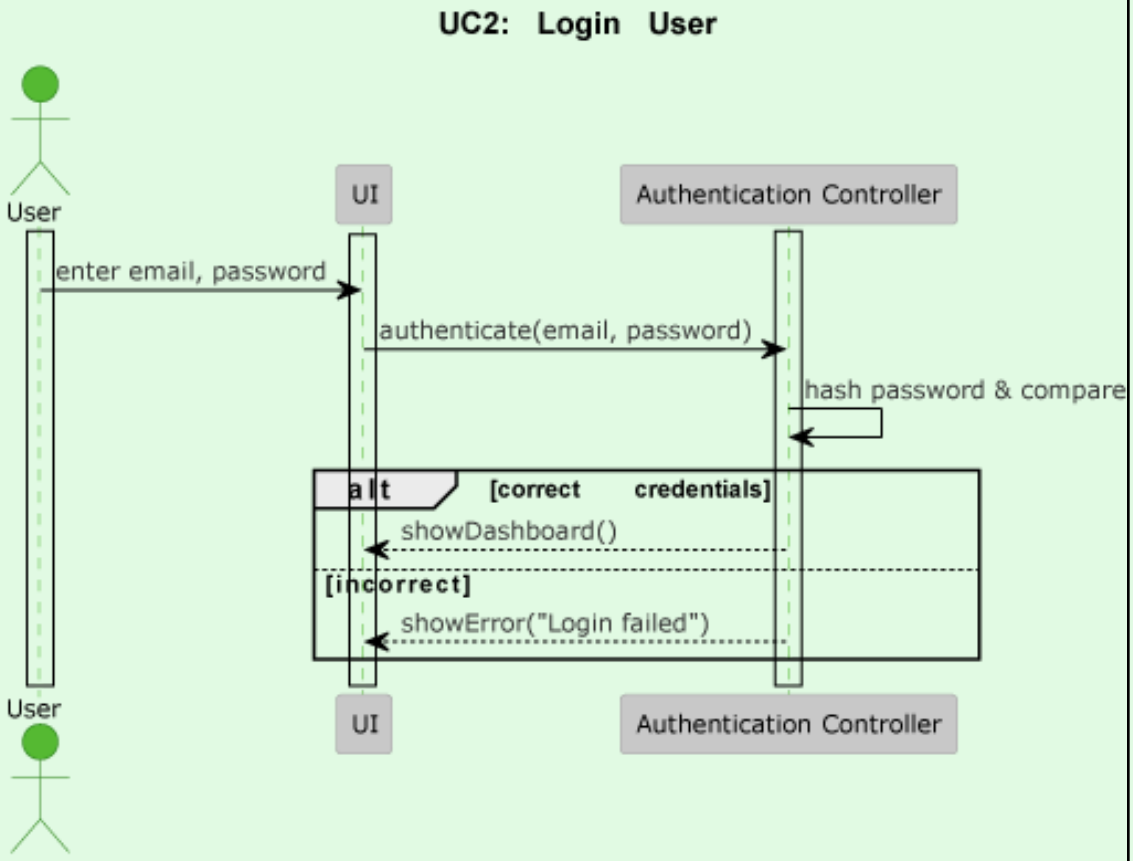
- **Users** have a **Role** (`User.role` → `Role.id`).
- **Trees** are linked to a **Tree Species** (`Tree.tree_species` → `Tree_Species.id`).
- **Trees** are planted by a **User** (`Tree.planted_by` → `User.id`).

This schema ensures that data integrity is maintained and follows **referential integrity** using foreign key constraints.

7. Interaction Diagrams

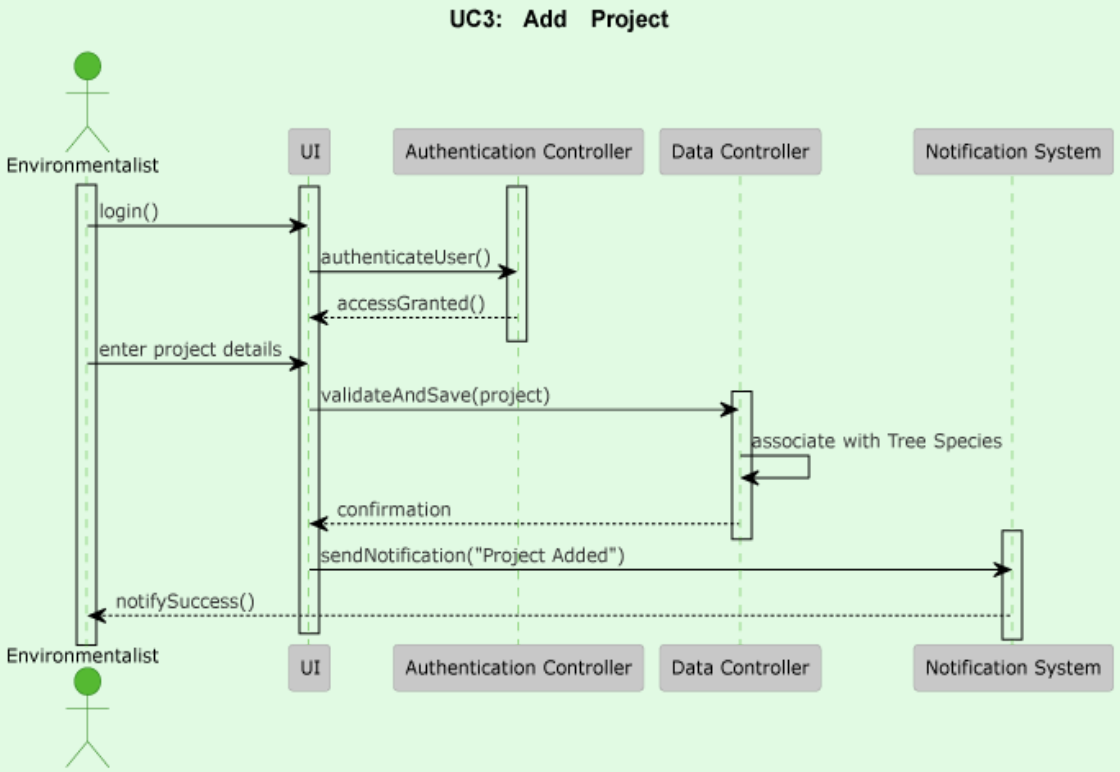
a. Diagrams

Use Case	Interaction Diagrams
RegisterUser	

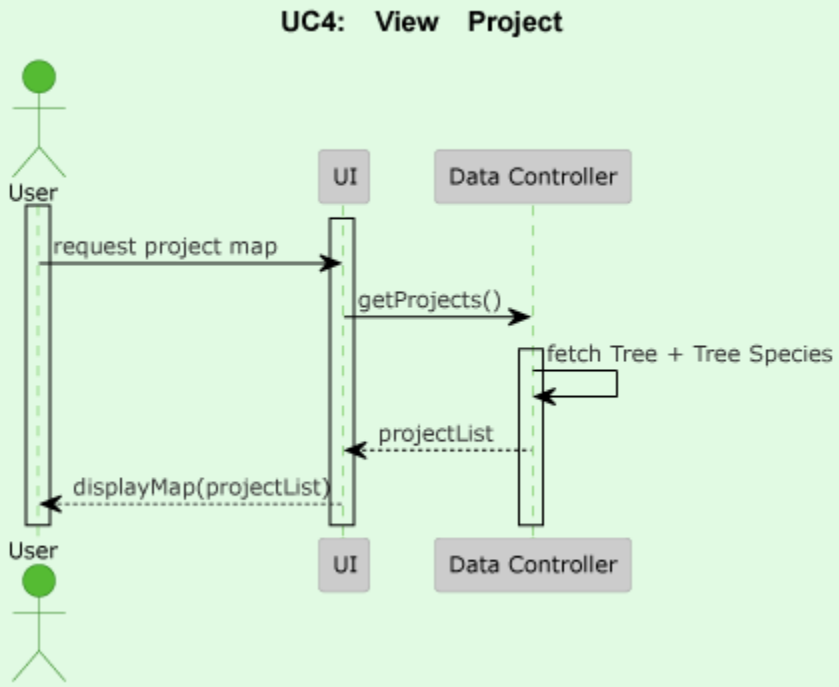
Login User	

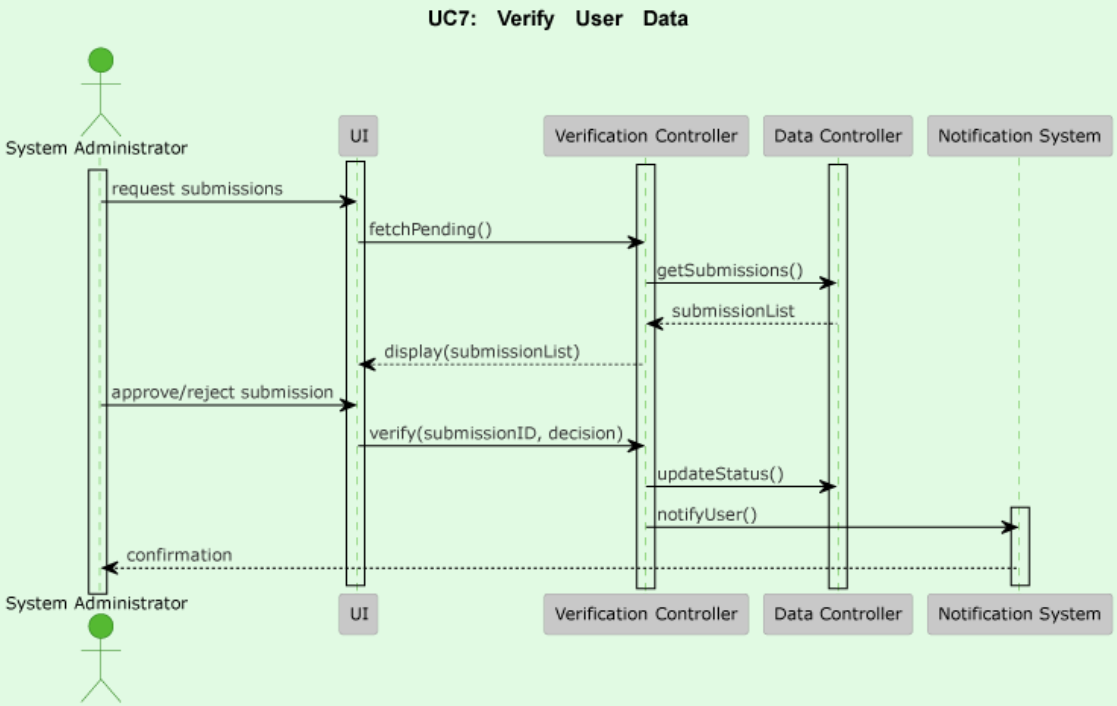
UC2: Login User
User
UI
Authentication Controller
enter email, password
authenticate(email, password)
hash password & compare
alt
[correct credentials]
showDashboard()
[incorrect]
showError("Login failed")
User
UI
Authentication Controller

AddProject



ViewProject



VerifyUserData	

b. Description

The system sequence diagrams (SSDs) illustrate the interactions between external actors (such as users, environmentalists, and administrators) and internal system components during the execution of key use cases in the Reforestation Management System.

Each diagram captures the **sequence of messages** exchanged between actors and system elements, focusing on:

- **Actor actions** (e.g., registration, login, data entry)
- **System responses** (e.g., validation, data storage, notification)
- **Controller responsibilities** (authentication, data management, verification)
- **Domain concept manipulation** (trees, species, submissions, roles)

These diagrams highlight the responsibility of each controller and show how key concepts like Tree, Tree Species, and Submission are created, accessed, or updated. Diagrams also represent internal system mechanisms such as:

- Notification dispatch via the Notification System
- Verification routing to the Verification Controller
- Secure access control via the Authentication Controller
- Data persistence is handled by the Data Controller

The diagrams emphasize **role-based behavior**, ensuring that only environmentalists can add projects or record environmental data, while administrators handle data verification. This reflects the system's **access control architecture**, reinforcing integrity and separation of concerns.

c. Design Patterns

Controller Pattern

Used for: Orchestrating the logic behind each use case.

Evidence:

- The presence of a `Controller` or `*Controller` class (e.g., `Authentication Controller`, `Data Controller`, `Verification Controller`) clearly demonstrates this pattern.
- Controllers handle message routing and coordinate between the UI and domain objects.

Purpose:

- **Decouples** user interface logic from business logic.
- Assigns **responsibilities** to an intermediary object rather than directly to UI or domain objects.

Information Expert Pattern

Used for: Assigning responsibility to the object that has the information necessary to fulfill it.

Evidence:

- The `Data Controller` retrieves, creates, or updates tree data because it "knows" about `Tree`, `Tree Species`, and `Submission`.
- `The Authentication Controller` knows user credentials and validates them.
- `Verification Controller` handles approval/rejection because it has access to the `Submission` data and criteria.

Low Coupling / High Cohesion (Design Principles)

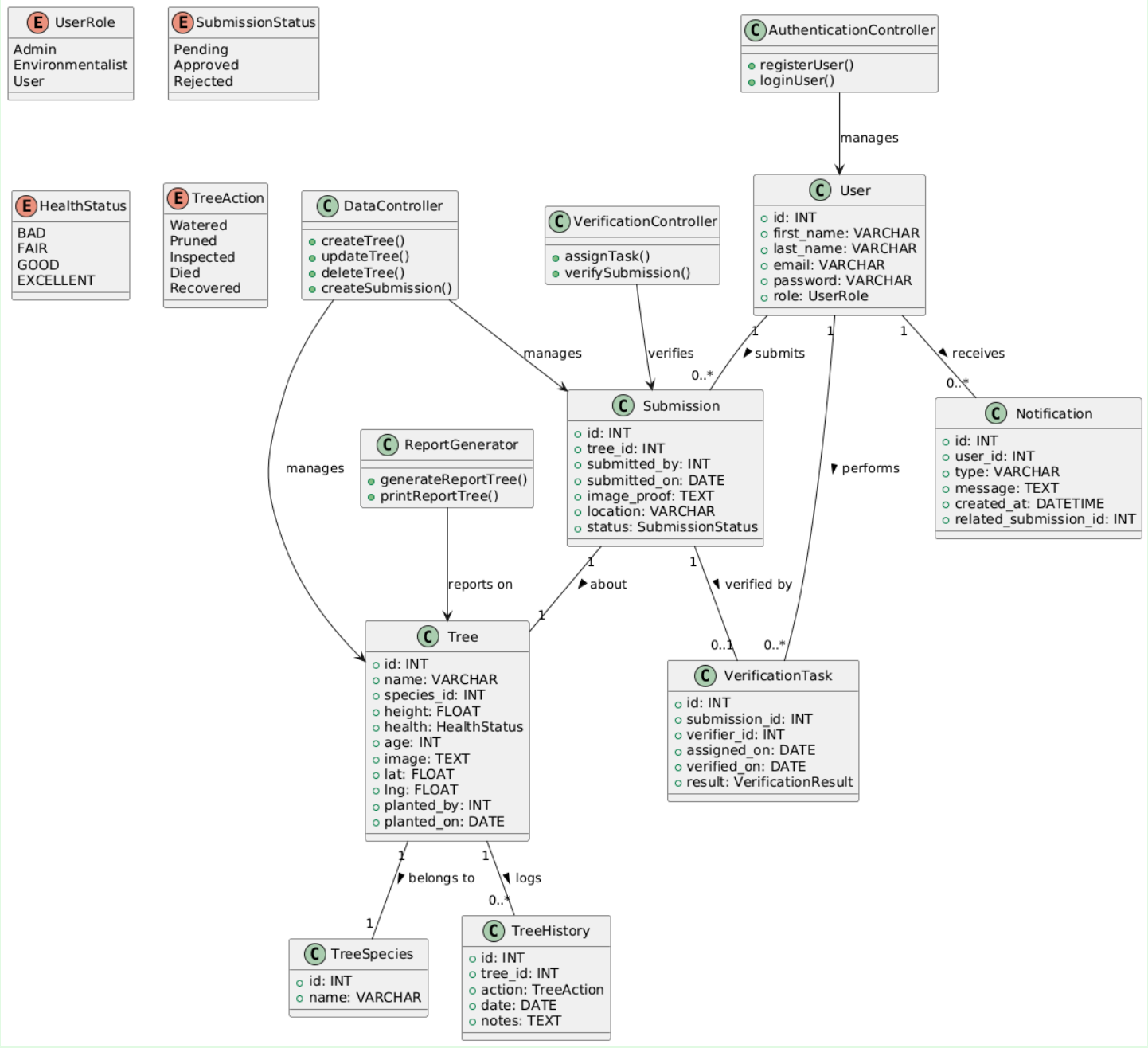
Used for: Avoiding overloading any one object and keeping responsibilities focused.

Evidence:

- Separation into multiple controllers, each focused on a domain-specific concern.
- `The Notification System`, `Report Generator`, and `Data Controller` are all separate entities with single, well-defined purposes.

8. Class Diagram & Interface Specification

a. Class Diagram



b. Data Types and Operation Signatures

Class: User

Represents any individual interacting with the system (admin, environmentalist, general user).

Attribute	Type	Description
id	Integer	Unique identifier for each user
role	Role	Role assigned to the user (e.g. admin)
first_name	String	User's first name
last_name	String	User's last name
email	String	User's email address (used for login)
password	String	Hashed password for authentication

Operations:

+ register(email: String, password: String): Boolean
+ login(email: String, password: String): Boolean
+ viewProfile(): User
+ updateProfile(details: User): Boolean

Class: Role

Defines user permissions and access levels in the system.

Attribute	Type	Description
id	Integer	Unique ID for the role
name	String	Role name (Admin, Environmentalist)

Class: Tree

Represents a reforestation project or an individual planted tree.

Attribute	Type	Description
id	Integer	Unique ID for each tree
tree_name	String	Assigned name or label for the tree
tree_species	TreeSpecies	Species of the tree
height	Float	Height of the tree in meters
health	Enum	Health condition: BAD, FAIR, GOOD, EXCELLENT
age	Integer	Age of the tree in years
image	String	Path to tree image (URL)
lat	Float	Latitude coordinate
lng	Float	Longitude coordinate
planted_by	User	User who planted the tree
planted_on	Date	Date of planting

Operations:

+ addTree(treeData: Tree): Boolean
+ updateTree(id: Integer, treeData: Tree): Boolean
+ getTreeById(id: Integer): Tree
+ deleteTree(id: Integer): Boolean

Class: TreeSpecies

Defines species that can be planted in reforestation efforts.

Attribute	Type	Description
id	Integer	Unique identifier for each species
name	String	Common name of the tree species

Class: Notification

Represents alerts and system messages sent to users.

Attribute	Type	Description
id	Integer	Unique identifier for the notification
type	String	Notification type (e.g., approval, warning)
message	String	Message content

Operations:

+ sendNotification(userId: Integer, message: String): Boolean
+ getNotificationsByUser(userId: Integer): List<Notification>

Class: VerificationController

Handles verification of user-submitted data.

Operations:

+ approveSubmission(subId: Integer): Boolean
+ rejectSubmission(subId: Integer): Boolean
+ validateSubmission(sub: Submission): Boolean

Class: AuthenticationController

Manages login and registration.

Operations:

+ registerUser(user: User): Boolean
+ authenticateUser(email: String, password: String): Boolean

Class: DataController

Handles CRUD operations related to trees and submissions.

Operations:

+ createTree(tree: Tree): Boolean
+ updateTree(tree: Tree): Boolean
+ createSubmission(submission: Submission): Boolean
+ getAllTrees(): List<Tree>
+ getAllSubmissions(): List<Submission>

Class: ReportGenerator

Generates analytical or printable reports from tree data.

Operations:

+ generateTreeHealthReport(): Report
+ generateUserActivityReport(): Report
+ exportReport(report: Report, format: String): Boolean

c. Traceability Matrix

Domain Concept	Derived Class(es)	Explanation
User	`User`, `AuthenticationController`, `Notification`	`User` became its class; Authentication duties were offloaded to `AuthenticationController`; notifications to users were abstracted separately.
Role	`Role`	Directly became its class to support Role-Based Access Control (RBAC).
Tree (Project)	`Tree`, `TreeSpecies`, `Submission`, `ReportGenerator`	`Tree` became the core class. The species part was moved to `TreeSpecies`, data logging to `Submission`, and analytics to `ReportGenerator`.
Tree Species	`TreeSpecies`	Mapped directly to a class with a single attribute (`name`) to normalize the database and reduce repetition.
Submission	`Submission`, `VerificationController`	`Submission` represents data entry; `VerificationController` was separated for verification logic.
Verification Controller	`VerificationController`	Directly modeled as a control class to handle approval/rejection operations.
Notification System	`Notification`	Modeled as a stand-alone class to handle message type, content, and user linkage.
Report	`ReportGenerator`	A utility-style controller handles report logic, hence `ReportGenerator`.
Authentication Controller	`AuthenticationController`	Handles login, registration, and token management logic.
Data Controller	`DataController`	General data operations (CRUD) are centralized in this utility controller.

d. OCL Contract Specifications

User

```
context User
inv UniqueEmail: User.allInstances()->isUnique(u | u.email)
```

- Ensures no two users have the same email.

```
context User
inv MustHaveRole: self.role <> null
```

- A user must always have a defined role.

Tree

```
context Tree
inv ValidHealthState: self.health = 'BAD' or self.health = 'FAIR' or self.health
= 'GOOD' or self.health = 'EXCELLENT'
```

- Enforces the enum constraint on health.

```
context Tree
inv CoordinatesValid: self.lat >= -90 and self.lat <= 90 and self.lng >= -180 and
self.lng <= 180
```

- Valid latitude/longitude bounds.

```
context Tree
inv SpeciesMustBeSet: self.tree_species <> null
```

- Tree must be linked to a valid species.

Submission

```
context Submission
inv MustReferenceTree: self.tree_id <> null
```

- Submissions must be linked to a tree.

```
context Submission
inv ValidStatus: self.status = 'Pending' or self.status = 'Approved' or
self.status = 'Rejected'
```

- Enforces correct submission states.

TreeSpecies

```
context TreeSpecies
inv UniqueSpeciesName: TreeSpecies.allInstances()->isUnique(s | s.name)
```

- Each species must have a unique name.

AuthenticationController

Since this class is behavioral (not data-driven), its constraints would likely exist in pre/post conditions rather than invariants.

```
context AuthenticationController::login(email: String, password: String): Boolean
pre: User.allInstances()->exists(u | u.email = email)
post: result = true implies User.allInstances()->any(u | u.email = email and
u.password = password)
```

- Only allow login if credentials match.

9. System Architecture

a. Architecture Styles

The reforestation system follows a **Monolithic Architecture,** integrating all components into a single codebase for simplicity and ease of deployment. The system consists of the following layers:

- **Presentation Layer:** Provides the UI for web and mobile users, following the Model-View-Controller (MVC) design pattern to separate UI from backend logic.
- **Application Layer:** Manages core business logic, including authentication, project management, and data validation. It exposes RESTful API endpoints (e.g., GET /trees, POST /reports) with JWT authentication for secure access.
- **Data Layer:** Handles all database interactions through an ORM, ensuring efficient data retrieval and manipulation. This layer stores user accounts, project details, and environmental data, using indexing and geospatial queries for optimized searches.

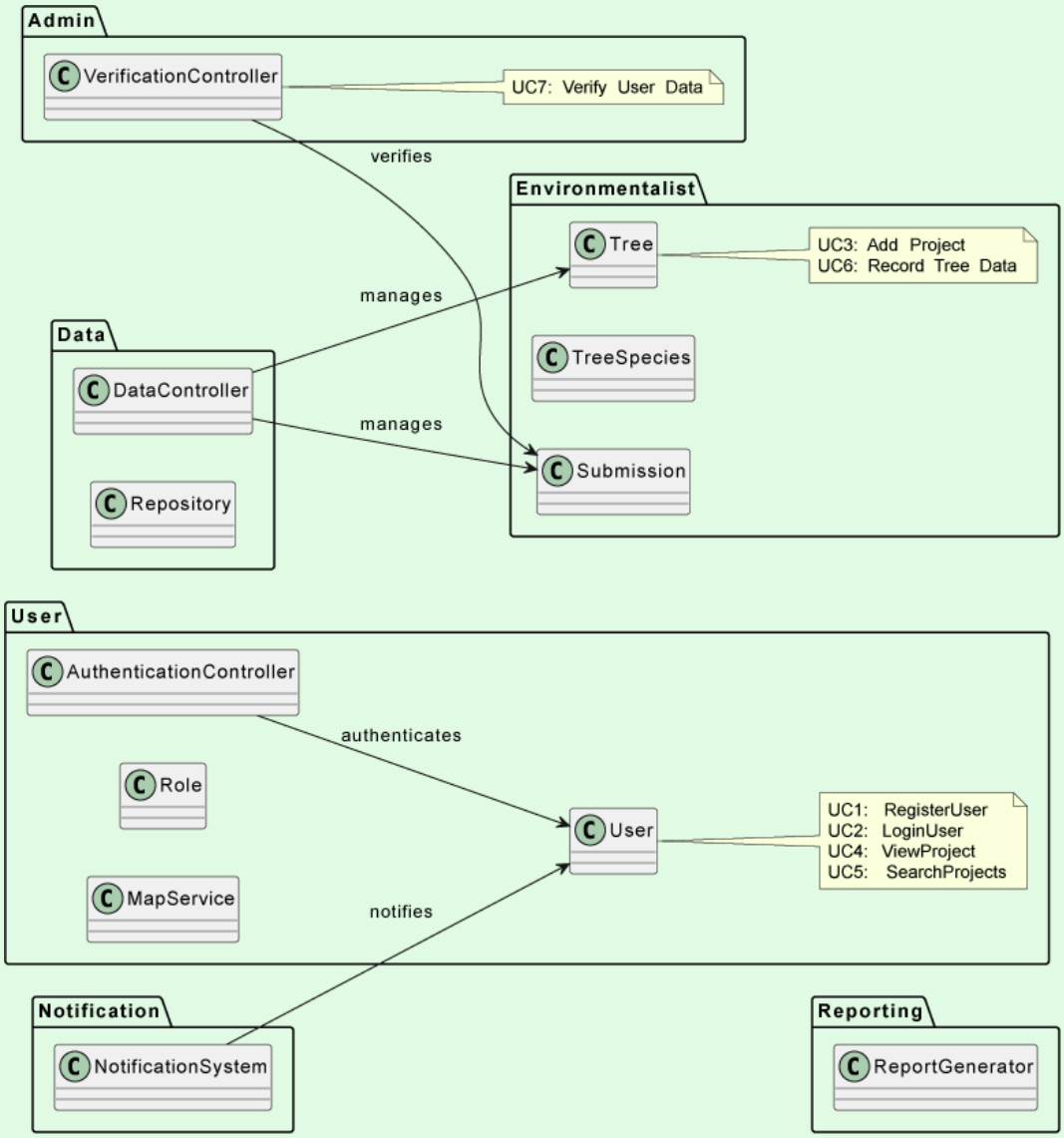
Event-Driven Components

- **Tree Maintenance Notifications**: Sends reminders for maintenance tasks and alerts users when a tree needs attention.
- **Administrator Approval Alerts**: Triggers admin review requests on project submissions, automatically updating the interactive map when approved.

Architectural Patterns

- **MVC**: Organizes the Presentation Layer.
- **RESTful API**: Ensures structured communication between the frontend and backend.
- **Event-Driven Notifications**: Automates tasks and updates for improved user experience.

b. UML Package Diagram



Package Structure

Package: `User`

- UC1: RegisterUser
- UC2: LoginUser
- UC4: ViewProject
- UC5: SearchProjects
- Classes/Controllers
    - AuthenticationController
    - User
    - Role
    - MapService

Package: `Environmentalist`

- UC3: Add Project
- UC6: Record Tree Data
- Classes:
    - Tree
    - TreeSpecies
    - Submission

Package: `Admin`

- UC7: Verify User Data
- Classes:
    - VerificationController
    - Submission
    - VerificationLog

Package: `Data`

- Handles database I/O for all roles
- Classes:
    - DataController
    - Repository

Package: `Notification`

- Handles Push Notifications
- Classes:
    - NotificationSystem

Package: `Reporting`

- Generate general reports
- Classes:
    - ReportGenerator

c. Subsystem-to-Hardware Mapping

Given that Re:Forest is primarily a **web-based application optimized for smartphones**, subsystem-to-hardware mapping focuses on ensuring optimal performance, scalability, and availability across a typical client-server architecture.

1. Client Devices (Smartphones, Tablets, Desktops)

- **Subsystems Used**:
    - *Presentation Layer* (UI/UX built with MVC design → Svelte Kit)
    - *MapService* (interactive geolocation views)
    - *NotificationSystem* (receives push notifications Firebase Messaging)
- **Hardware Requirements**:
  Smartphones: Minimum 1.5GHz processor, 2GB RAM, screen resolution ≥ 320x575px.
    - **Tablets/Desktops**: Any modern device capable of running a JavaScript-enabled browser.
    - **Network**: Minimum 3 Mbps connection recommended to load maps and send/receive data smoothly.

2. Web Server / Application Server

- **Subsystems Deployed**:
    - *Application Layer*: Hosts the business logic, authentication, API endpoints
    - *Notification System*: Generates and sends event-driven alerts
    - *ReportGenerator*: Generates downloadable data insights
- **Deployment Environment**:
    - Hosted on a VPS provider (Vercel).
    - Requires 4+ CPU cores, 4GB+ RAM for concurrent user support
    - Supports RESTful API endpoints secured via HTTPS and JWT

3. Database Server

- **Subsystems Used**:
    - *Data Layer*: Manages persistent storage via ORM
    - *Repository, Submission, VerificationLog Classes*
- **Database**: PostgreSQL via NeonDB (relational DB with geospatial extensions)
- **Hardware Requirements**:

- Dedicated or cloud-based instance
- Minimum 100GB storage with SSD-backed volumes for I/O efficiency
- 2+ vCPUs, 4GB+ RAM, auto-scaling enabled for high traffic periods
- Regular backups and role-based access enforced

4. Map and Geolocation Services (Third-Party APIs)

- **Subsystems Used**:
  - *MapService*, *Tree*, *TreeSpecies* components
- **Integration**:
  - Mapbox API accessed via frontend and backend
  - Requires client-side JavaScript and API key management on the backend
  - Latency must be minimized to support near real-time map updates

5. Push Notification Service

- **Subsystems Used**:
  - *NotificationSystem*
- **Deployment**:
  - Integrated with Firebase Cloud Messaging (FCM) or a similar service
  - Supports mobile and browser push notifications asynchronously
  - Event triggers are processed on the server and dispatched via external APIs

d. Persistent Data Storage

Our Need for Persistent Data Storage

Our system **requires persistent data storage** to retain user accounts, project details, environmental data, and administrative records beyond a single execution. Therefore, a **relational database** is the chosen storage strategy. In essence, the entirety of the data will be stored. Data models are addressed below:

Persistent Objects

The system must store the following objects:

- **Users**: Stores user information, including their role (e.g., administrator, environmentalist, general user).
- **Roles**: Defines user roles within the system.
- **Trees (Projects)**: Stores details of individual planted trees, including species, health, and GPS location.
- **Tree Species**: Lists species of trees that can be planted.

Storage Management Strategy

We are using a **Relational Database Management System (RDBMS)** (**PostgreSQL**) due to:

- **Data Integrity**: Ensures referential integrity using foreign keys.
- **Scalability**: Supports large datasets efficiently.
- **Indexing**: Optimizes search queries.

Security: Enables role-based access and user authentication.

e. Network Protocol

ReForest uses a secure, event-responsive network protocol stack, tailored to a SvelteKit frontend, deployed on Vercel, backed by Neon DB, and utilizing Firebase Cloud Messaging (FCM) for notifications. Communication between subsystems is managed via modern web protocols that emphasize speed, scalability, and responsiveness across mobile and web platforms.

1. Transport Protocol

- **Primary Protocol**: HTTPS (via Vercel's edge network)
- **Port**: TCP 443 (secure by default)
- **Purpose**: All client-server interactions are encrypted and routed through Vercel's edge functions, improving latency and security.

2. Application Layer Protocols

- RESTful API over HTTPS (via SvelteKit Endpoints)
    - SvelteKit routes serve both frontend and backend logic.

- RESTful endpoints like POST /api/reports or GET /api/trees are mapped to SvelteKit server functions.

3. Push Notification Protocol

- **Service**: Firebase Cloud Messaging (FCM)
- **Protocol**: HTTP v1 API (used from server-side SvelteKit endpoints)
- **Use Cases**:
  - Admin project approval alerts
  - General user system messages
- **Format**: JSON payload with FCM device tokens
- **Delivery**:
  - Mobile: via Firebase SDK (Android/iOS)
  - Web: via Firebase's Web Push with Service Workers

4. Authentication Protocol

- JWT-based Auth (Stateless)
  - Used for API protection and role-based access
  - Tokens are issued at login and stored in HttpOnly cookies or local storage (with care)
  - Authentication middleware on SvelteKit endpoints checks token validity

5. Database Communication

- **Database**: Neon DB (PostgreSQL) with serverless architecture
- **Protocol**: SQL over TLS (PostgreSQL)
- **Strategy**:
  - All data interaction is done via secure server-side API routes.
  - Neon automatically scales with demand and supports connection pooling via PgBouncer-like mechanisms.

6. Error Handling and Data Validation

- **Client-Side Validation**: Handled via SvelteKit's form actions and bindings
- **Server-Side Validation**: Enforced using schema validators (e.g., Zod or custom logic)

7. Performance and Optimization

- **Edge Caching (Vercel)**:
  - Static assets and GET requests are cached at the edge for fast delivery
  - ISR (Incremental Static Regeneration) used where applicable
- **Compression**: All responses use GZIP or Brotli via Vercel's automatic compression
- **Rate Limiting**:
  - Applied at the API gateway layer (Vercel functions + custom logic)
  - Based on IP and user role to avoid abuse

f. Global Control Flow

Execution Orders

Our system is event-driven, meaning users interact with it differently depending on their needs rather than following a strict step-by-step process. Users can register, log in, add reforestation projects, update tree health data, and view project details anytime. The system waits for user actions and responds accordingly rather than enforcing a fixed sequence of operations.
However, some processes within the system are linear when necessary. For example:

- A user must register before they can add a project.
- Submitted data must go through administrator approval before being made public.

This combination of event-driven and linear workflows ensures flexibility while maintaining data integrity.

Time Dependency

Our system includes time-sensitive functionalities such as:

- Tree Maintenance Notifications: The system sends alerts for watering, pruning, or other tree care tasks.
- Project Approval Reminders: Administrators receive notifications about pending data approvals.
- Data Updates: Users can periodically update tree health conditions.

These features are event-driven but work on periodic schedules, ensuring timely critical updates.

Event-Response vs. Real-Time System

Our system is event-response based, reacting to user inputs but not having strict real-time constraints. Users enter data, and the system processes it asynchronously. However, some real-time elements exist, such as:

- Interactive Map Updates: Newly planted trees appear on the map immediately after approval.
- Live Data Entries: Volunteers can upload tree status updates in real time via mobile devices.

This balance ensures efficiency without requiring high-performance real-time computing.

g. Hardware Requirements

System Resources

Our system is a web-based platform, so it mainly depends on standard computing resources:

- **Screen Display:** The system is optimized for desktops, tablets, and mobile devices.
  - **Minimum resolution:** 320x575 pixels.
- **Disk Storage:** The system stores images, reports, and project details.
  - **Minimum required:** 1MB of available storage per user instance.
- **Communication Network:** The system relies on internet access for data synchronization and map updates.
  - **Minimum required:** 3 Mbps connection for smooth performance as if it's under there will be some issues loading the map.

Special Requirements

- **Geolocation Support:** The system integrates with mapping APIs (MapBox) to track tree locations.
- **Mobile Device Compatibility:** Volunteers can use smartphones to submit data.
- **JavaScript:** The frontend requires JavaScript to maintain interactivity

10. Algorithms and Data Structures

a. Algorithms

This system's main logic centers around data validation and processing rather than complex mathematical models or numerical simulations. However, some lightweight algorithms are used:

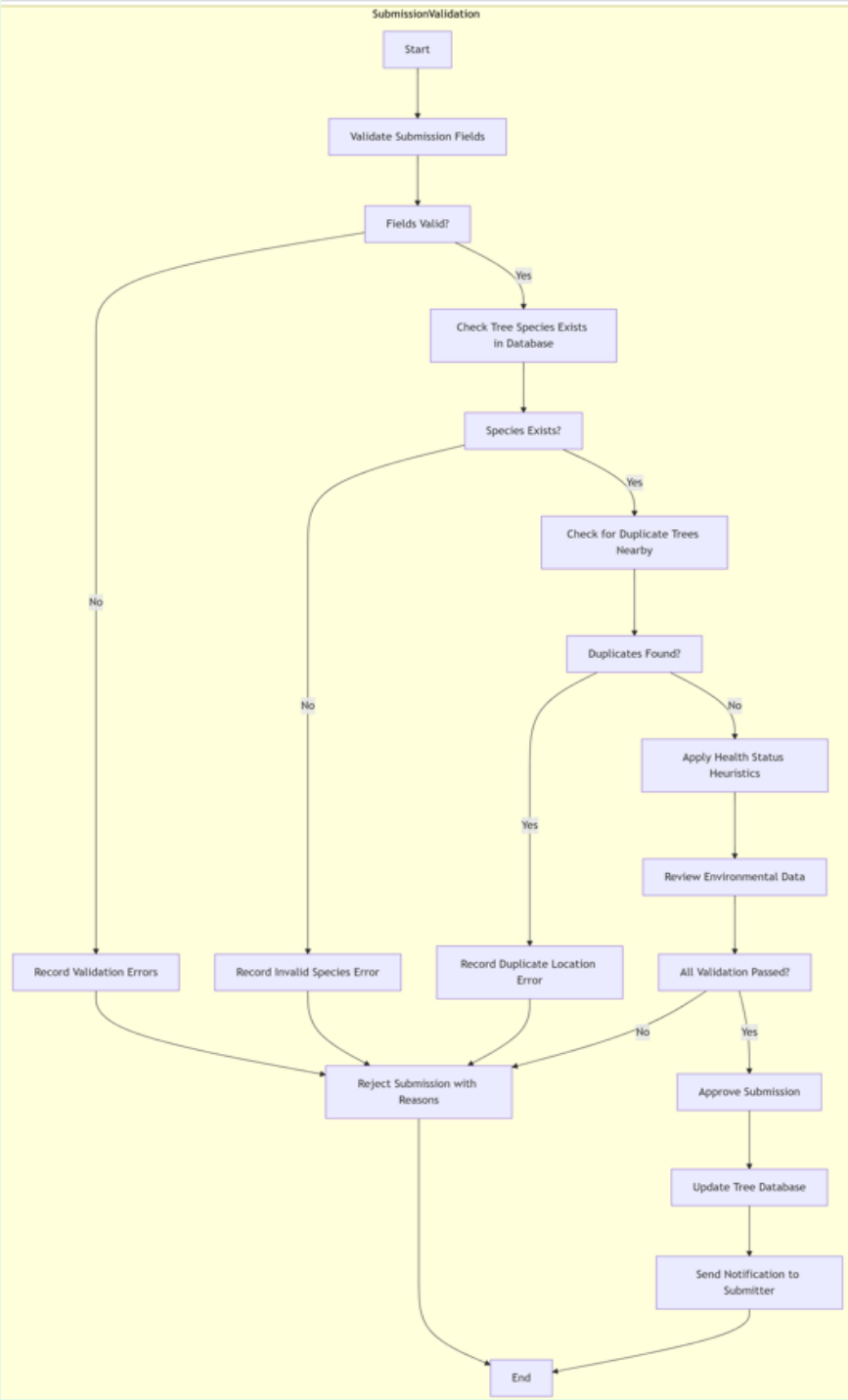
Verification Algorithm

Purpose: To verify user-submitted tree data in `Submissions`.

Approach:

- Input validation (e.g., image size/type, location validity, date constraints)
- Cross-check tree species against the `TreeSpecies` table
- Automatically flag duplicates using spatial proximity (based on GPS `lat/lng`)
- Suggest health status using heuristics (optional future extension)

Activity Diagram – Verification Algorithm (simplified):



Tree Submission Verification Flow

1. **Start:** User submits tree data.
2. **Field Validation:**
    - Check required fields (e.g., location, species, planting date).
    - If anything is missing or incorrect → reject with errors.
3. **Species Validation:**
    - Confirm species exists in the database.
    - If not → reject with error message.
4. **Duplicate Check:**
    - Look for nearby trees using GPS data.
    - If another tree is too close → flag as possible duplicate.
5. **Environmental Review:**
    - Compare reported tree health with local environmental data.
    - Flag unusual or unlikely health reports.
6. **Final Decision:**
    - If all checks pass → approve and add to database.
    - Notify user of approval.
7. **If Rejected:**
    - Send user detailed reasons so they can fix and resubmit.

b. Data Structures

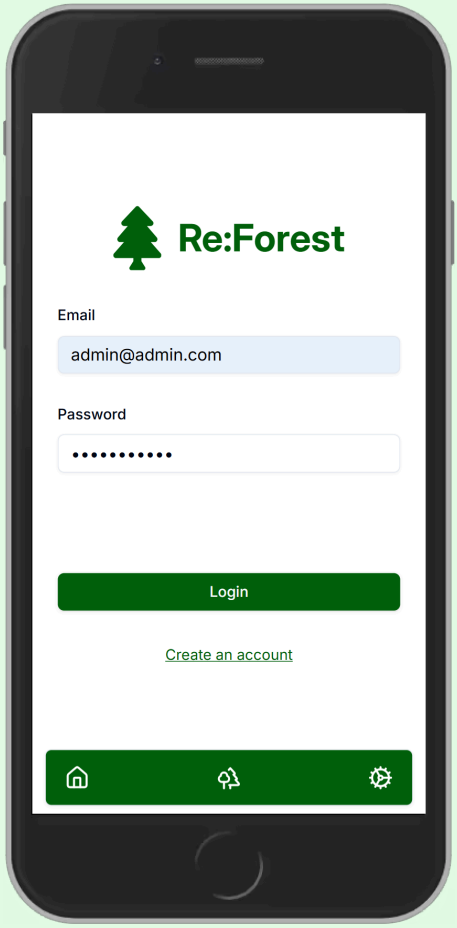
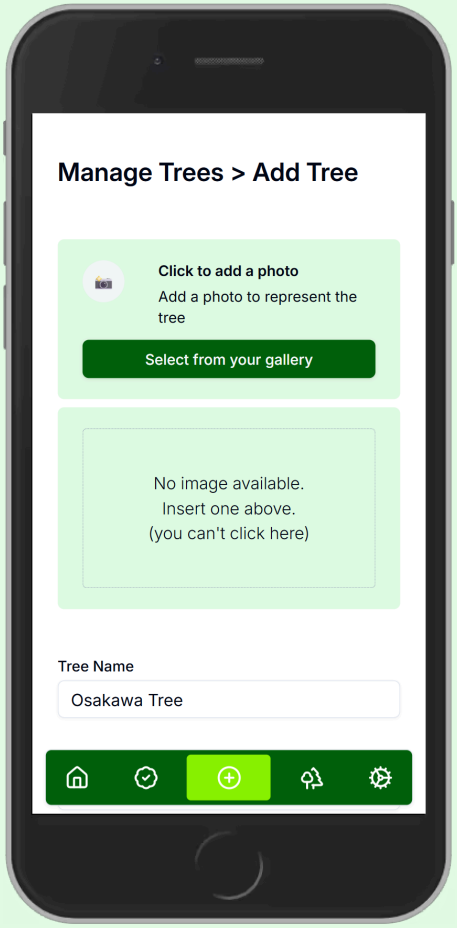
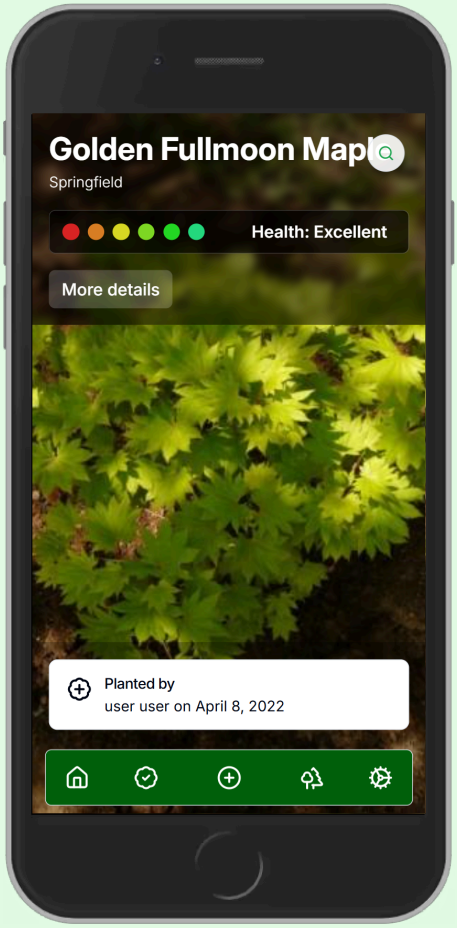
While no highly complex data structures are used, the following are employed:

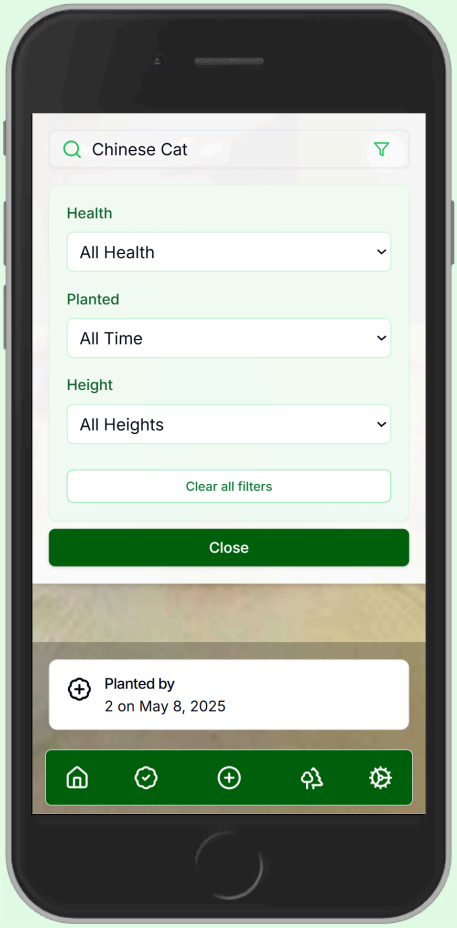
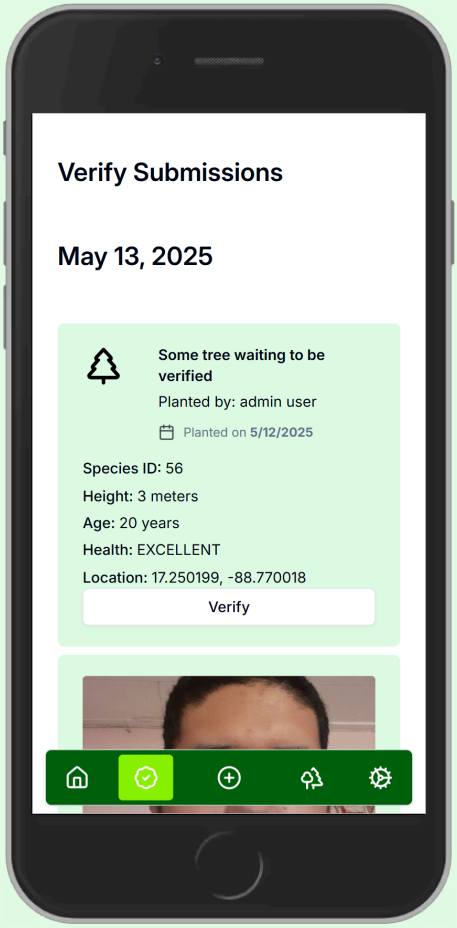
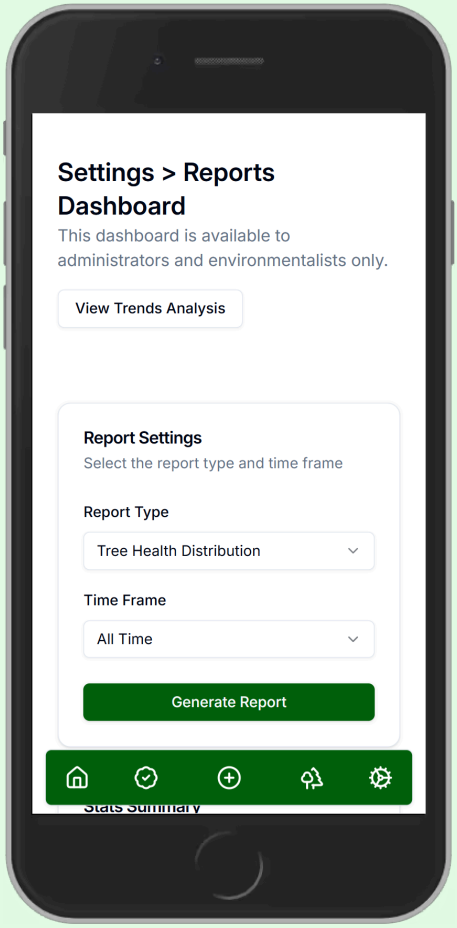
- **Arrays/Lists**: Used for storing collections like submissions, notifications, or trees (especially in front-end or APIs).
- **Hash Maps** (or dictionaries): Used internally to quickly lookup tree species, users, roles, and submission statuses.
- **Criteria**:
    - **Performance**: For frequent lookups (e.g., species ID → species name), hash maps ensure constant time access.
    - **Simplicity**: Arrays are simple and ideal for ordered iteration (like listing trees or reports).

c. Concurrency

- **Not implemented** in the current design.

11. User Interface Design and Implementation

UC2 LoginUser	UC3 AddProject	UC4: ViewProject
		

UC5: SearchProjects	UC7: VerifyUserData	Reporting
		

Description

- **Significant Change #1:** The tree submission form was restructured into a simplified two-step process:
- **Step 1 –** Upload Image & Capture Location
- **Step 2 –** Enter Tree Details
- **Why:** User feedback indicated that the original long form was overwhelming on mobile. This new step-based flow reduces cognitive load, improves readability, and helps users stay focused by grouping related tasks.

Ease-of-Use Implementation

- **Consistent Layout:** Every screen uses a predictable two-part layout (Navigation + Content) for familiarity and flow.
- **Clear Visual Cues:** Icons and labels follow the ShadCN design language for clarity and visual consistency.
- **Guidance Built-In:** Tooltips and placeholder text provide inline instructions and examples.
- **Efficient Interaction:** Tree submissions can be completed in 3 taps or fewer, minimizing interaction effort on smaller screens.

Usability principles applied

- **Visibility of System Status:** Users receive immediate feedback on form submission (e.g., “Tree saved successfully” or “Image upload failed”).
- **Match Between System and Real World:** Labels use familiar terms like **“Tree Height”**, **“Location”**, and **“Species”** to align with user expectations.
- **Error Prevention:** The **Submit** button remains disabled until **all required fields** are valid, preventing incomplete or accidental submissions.

User Effort Estimation

Use Case	Navigation Effort (Taps)	Data Entry Effort (Keystrokes/Taps)	Total Effort
RegisterUser	2 taps (Menu → Registration)	25 keystrokes (Name, Email, Password) + 2 taps (Submit, Confirm via link/app)	**4 taps + 25 keystrokes**
UserLogin	1 tap (Open login screen)	15 keystrokes (Email + Password) + 1 tap (Login) *Autofill may reduce keystrokes*	**2 taps + 15 keystrokes** *(less w/ autofill)*
AddProject	2 taps (Menu → "Create Project")	12 keystrokes (Name) + 1 tap (Map) + 4 keystrokes (Area) + 1 dropdown tap + 2 taps (Upload) + 1 tap (Submit)	**7 taps + 16 keystrokes**

ViewProject	1 tap (Open map view)	1 tap (Select project marker/card)	**2 taps**
SearchProjects	1 tap (Activate search)	10 keystrokes (filter criteria) + 1 tap (Search/Filter)	**2 taps + 10 keystrokes**
VerifyUserData	2 taps (Admin menu → Pending Approvals)	1 tap (Open submission) + 1 tap (Approve)	**4 taps**

12. Design of Tests

List of Test Cases (Use Cases 3 & 4)

Use Case 3: Add Tree Data

Test Case ID	Test Description	Expected Result
TC-2	Insert tree with non-existent Tree Species ID	Error or exception thrown; insertion blocked
TC-3	Insert tree with non-existent User ID	Error or exception thrown; insertion blocked
TC-3.1	Submit valid tree data	Submission is created with the status “Pending.”
TC-3.2	Submit without image	Error message displayed, submission blocked
TC-3.3	Submit with an invalid species ID	Error: “Invalid species”
TC-3.4	Submit duplicate location	Warning shown or duplicate rejected

Test Case ID	Test Description	Expected Result
TC3.5	Submit with missing location (lat/lng)	Submission not accepted
TC-4	Insert tree with invalid health enum	Error or validation failure
TC-5	Insert tree with edge-case height (zero)	Insertion blocked or validation failed
TC-6	Insert tree with extreme lat/lng coordinates	Insert fails due to location validation
TC-7	Insert tree with a future planted date	Date validation failure or rejection
TC-9	Insert tree with missing optional image field	Insert fails due to missing image

Test Coverage

We aim to achieve thorough test coverage across both our front-end components and back-end logic. Testing is implemented using **Vitest**, a modern and fast unit testing framework designed for Vite-powered projects. Our current test suite includes:

- **Unit tests** for core functions and business logic (e.g., form validation, role-based access).
- **Component tests** for key UI components, especially those related to submission forms and project viewing.
- **Mocked API interaction tests** to simulate responses from the backend without requiring a live server.

Code coverage is tracked automatically using Vitest's built-in coverage reporting. Coverage statistics are uploaded and visualized through GitHub Actions using the `vitest --coverage` flag and are reviewed during every pull request. Our targets include:

- **>= 90% coverage** on critical modules (e.g., `AuthenticationController`, `DataController`, `VerificationController`)

- **>= 80% overall coverage** across the codebase

These metrics ensure a high level of confidence in our system behavior and help catch regressions early.

Integration Testing Strategy

Our integration testing focuses on verifying that components and services function correctly when combined, particularly between the front-end UI, controller logic, and persistent data layer.

Plan for Testing

- **Vitest with Supertest**: Integration tests simulate full workflows (e.g., user registration, login, tree data submission) by calling actual service methods and asserting system state changes.
- **Mocked Database**: We use an in-memory mock of PostgreSQL schema or mock service layers to avoid polluting the actual database.
- **Workflow Scenarios Tested**:
  - Full tree submission flow (from form to database)
  - Authentication flow (register → login → access)
  - Submission verification flow (submit → verify → notify)
- **CI/CD Integration**: All integration tests are run automatically using **GitHub Actions** on every push and pull request to the `main` and `dev` branches. Failures in integration tests block merges

13. History of Work

Throughout the course of this project, we adopted a collaborative, agile-style workflow that emphasized continuous integration, transparent communication, and iterative development. Unlike the original milestones laid out in Reports #1 and #2, which envisioned a more linear progression, our actual development process evolved dynamically as we adapted to new insights, challenges, and team member availability.

Initially, our plan scheduled the implementation of core use cases by Week 4. However, integration and debugging took slightly longer due to real-time issues with asynchronous GitHub updates, causing a two-week delay in deploying a stable build. Despite this, we compensated by streamlining other parts of the workflow and accelerating testing and report generation in later weeks.

We used GitHub as our primary version control and collaboration platform. Team members contributed frequently, pushing code to a shared repository and uploading key assets like reports and final documentation. This ensured everyone had access to the latest versions and fostered accountability.

To coordinate tasks and milestones, we relied heavily on Asana, which served as our project management dashboard. Asana enabled us to assign responsibilities, track deadlines, and maintain visibility into individual and collective progress. The platform helped us stay on track even when some tasks required reprioritization or redistribution.

Frequent team meetings were held to align on technical issues, design decisions, and project direction. These sessions were invaluable for debugging, reviewing deliverables, and adjusting the course when needed. Outside of scheduled meetings, we stayed in touch via a dedicated WhatsApp group, which allowed for both quick updates and asynchronous problem-solving when real-time communication wasn't possible.

Despite smooth collaboration overall, we encountered occasional technical issues, particularly with GitHub not reflecting some changes immediately after uploads. This was mitigated by improving our commit/push discipline and clarifying our merge protocols.

b. Current Status - Accomplishments

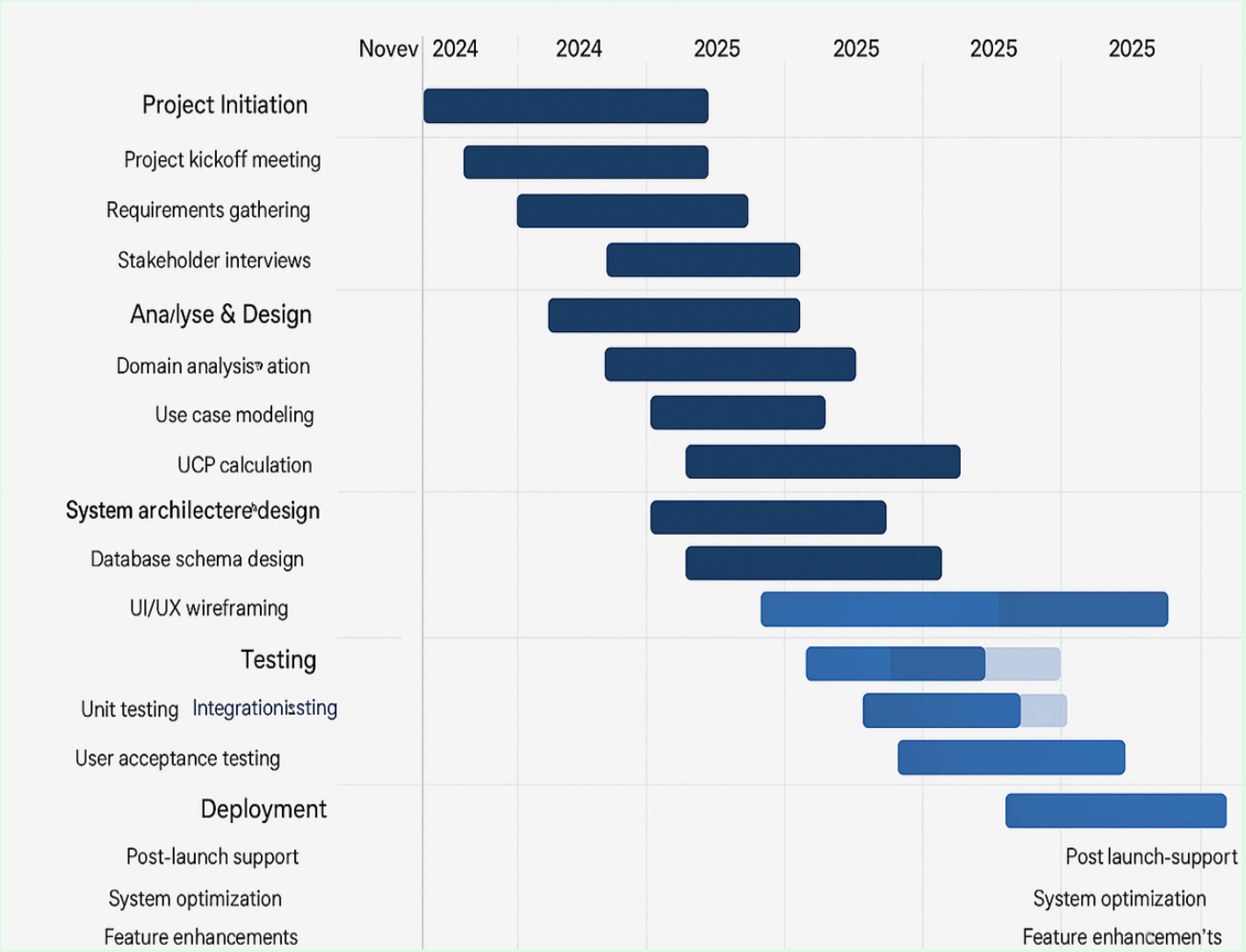
- ★ All use cases from the requirements traceability matrix were fully implemented.
- ★ A working push notification system was integrated and tested.
- ★ Dynamic report generation was implemented, displaying key project metrics and environmental data.
- ★ The full-stack web application was successfully deployed using Vercel, providing a live demo environment.
- ★ Backend classes were designed to modularize and streamline database transactions, improving maintainability.
- ★ Database backend implemented using NeonDB.
- ★ Application fully deployed via Vercel.

c. Future Work & Opportunities

Looking forward, the project offers several promising directions for enhancement:

- Maintenance notifications can be more deeply integrated with user calendars and external APIs.
- Enhanced Analytics: Add visual dashboards for survival rates, CO_2 offset projections, and user engagement trends.
- AI-Based Image Analysis: Incorporate ML tools to auto-analyze uploaded images for growth validation or disease detection.
- Multilingual Support: To serve a broader demographic, implement interface localization.
- Public Participation Module: Allow citizens to adopt tree sites or sponsor reforestation efforts via the platform.
- Field Testing: Deploy the application in a real-world pilot and gather usability feedback from environmentalists and admin users.

Project Timeline:



14. References

- *"Neon Serverless Postgres — Ship faster," Neon. https://neon.tech/*
- *"A free database designer for developers and analysts." https://dbdiagram.io/*
- *"Svelte." https://svelte.dev/*
- *"UML Class Diagram Tutorial" https://www.lucidchart.com/pages/uml-class-diagram*
- *"All about UML interaction diagrams" https://www.lucidchart.com/pages/uml-interaction-diagram*
- *https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/*

15. Summary of Changes

- ➔ Updated all requirements to match actual implementation decisions employed.
- ➔ Updated use case diagram to properly represent the Use Cases identified.
- ➔ Updated traceability matrix to reflect Use Cases and functional requirements.
- ➔ Concept Details are now listed by use case.
- ➔ Updated Interaction diagrams to reflect concepts more accurately.
- ➔ Updated system sequence diagrams
- ➔ Updated Class/domain diagrams.
- ➔ User Interface design updated to match final designs.
- ➔ Recreated package diagram.
- ➔ Package structures now reflect use cases.
- ➔ Valid References are now included.
- ➔ Interaction diagrams now include the design patterns used.
- ➔ Object Constraint Language contracts are now included.

- ➔ History of Work updated with accomplishments
- ➔ Network Protocols are now included.
- ➔ Mapping Subsystems to hardware section now included.
- ➔ Effort Estimation calculated.