American International University-Bangladesh (AIUB)

Final – Term Group Project (Summer-2025)

Project Title: Farm Management System (Agroculture)

Course: Software Engineering Process and Configuration Management

Sec:A

Submitted By-

NAME	ID
FARDIN HOQUE	24-93450-2
SHAH FAHIM CHOWDHURY	24-93245-1
RASHIDA BEGUM LIMA	24-93566-3
RIYA BASAK RISHA	24-93464-2

Course Instructor-

DR. MOHAMMAD RABIUL ISLAM
Assistant Professor
Department of Computer Science
American International University-Bangladesh

Contents

1. Problemt Statement:

Farm Management System (Agriculture) is a critical sector for developing economies like Bangladesh, yet the connection between farmers and consumers often suffers from inefficiencies, limited transparency, and dependence on intermediaries. Farmers typically have limited access to broader markets, which restricts their ability to secure fair prices. Middlemen often control distribution channels, leading to reduced profits for producers and inflated prices for consumers.

From the buyer's perspective, obtaining fresh and reliable agricultural products can be challenging due to the lack of a trusted platform that offers transparent product details, pricing, and origin information. Additionally, there is minimal opportunity for buyers to provide feedback, rate quality, or engage directly with producers.

While some online marketplaces exist, they are often designed for general retail purposes and do not address the specific needs of agricultural trade. They may be costly, complex for rural users, and lacking in features tailored to farm-to-consumer interactions. There is a clear need for an affordable, user-friendly, and secure web-based solution that facilitates direct transactions between farmers and buyers, supports agricultural community engagement, and operates efficiently even in low-bandwidth rural environments.

2. Objective:

The **AgroCulture – Farm Management System** is developed to address these challenges with the following objectives:

1. **Enable Direct Farmer-to-Consumer Transactions**
    - Provide an intuitive platform for farmers to upload, update, and manage product listings, including images, descriptions, pricing, and category details.
    - Empower farmers to manage their own profiles and establish a recognizable digital presence.
2. **Enhance the Buyer's Purchasing Experience**
    - Implement browsing and search functionality for quick product discovery based on keywords, categories, or price range.
    - It allow buyers to add products to the shopping-cart and complete purchases using **Cash-on-Delivery (COD)** to accommodate varying levels of digital payment adoption.
3. **Integrate Feedback and Quality Assurance Mechanisms**
    - Enable buyers to leave product reviews and ratings after purchase, increasing transparency and building trust.
    - Display average ratings and feedback summaries to guide purchasing decisions.
4. **Promote Agricultural Knowledge Sharing**
    - Include a blog module for posting agricultural tips, seasonal farming advice, and market updates.

- Support interactive engagement with features such as likes and comments to foster community connections.

5. **Ensure Secure and Reliable System Operations**
   - Implement authentication and role-based access control to protect user data and platform integrity.
   - Validate and sanitize all inputs to mitigate security risks.
6. **Support Scalability and Cost-Effectiveness**
   - Use open-source technologies (PHP, MySQL) to keep deployment and maintenance costs low.
   - Design the platform to handle future expansions in features, users, and transaction volume without significant re-engineering.

3. Scope:

The **AgroCulture-Farm Management System** is designed as a web-based application that enables farmers and buyers to interact directly through a secure and user-friendly platform. The system provides the following scope:

1. **User Management**
   - Registration and login for farmers and buyers with role-based access.
   - Profile management including updating personal details and profile pictures.
2. **Item Management**
   - Farmers will upload, update, and delete product listings with attributes such as item name, category, description, price, and image.
   - Products are stored in a searchable database and displayed in the marketplace.
3. **Marketplace and Buyer Interaction**
   - Buyers will browse and search for products using name, product category or price.
   - Buyers will add products to the shopping cart, view items, and proceed with order placement.
   - Orders are confirmed using **Cash-on-Delivery (COD)** as the primary payment method.
4. **Reviews and Feedback**
   - Buyers can post reviews and rate products they purchase.
   - Reviews are displayed alongside products, contributing to transparency and buyer decision making.
5. **Blog and Community Engagement**
   - A blogging feature allows farmers and buyers to share agricultural tips, advice, and news.
   - Users can like and comment on blog posts to promote community interaction.
6. **Order and Transaction Management**
   - Buyers can view their order history and status.
   - Farmers can access transaction details related to their products.
7. **System Architecture**
   - Built using PHP and MySQL for lightweight, cost-effective deployment.
   - Designed to run in standard web environments such as XAMPP or hosting servers.
   - Configured to operate in low-bandwidth conditions, making it suitable for rural use.

4. Limitations:

Despite its usefulness, the system also has certain limitations:

1. **Payment Options**
    - Only **Cash-on-Delivery (COD)** is supported; digital payment gateways (e.g., mobile banking, credit cards) are not yet integrated.
2. **Scalability**
    - The system is suitable for small to medium-scale deployments but may require optimization to handle very high user or transaction volumes in a nationwide rollout.
3. **Security Constraints**
    - While basic authentication and validation are provided, advanced security features such as multi factor authentication, encryption at rest, and intrusion detection are not implemented in the initial version.
4. **Logistics and Delivery Tracking**
    - The platform records transactions but does not include a fully integrated delivery tracking or logistics management system. Delivery arrangements remain manual.
5. **Offline Accessibility**
    - The system requires internet connectivity and does not offer offline support. This may limit usage in areas with poor or intermittent internet service.
6. **Limited Analytics and Reporting**
    - The current system does not provide advanced analytics dashboards (e.g., sales trends, demand forecasting). Reporting features are minimal.
7. **Language and Localization**
    - The interface is primarily in English and does not yet support multilingual or local language options, which may affect accessibility for some farmers.

5. Stakeholder Identification:

Stakeholder identification is the process of recognizing all individuals, groups, and organizations that influence, benefit from, or are affected by the *AgroCulture* system. Since Agile/Scrum thrives on collaboration and continuous feedback, understanding the stakeholders and their roles is crucial to delivering a system that meets both business and user needs.

5.1 Primary Stakeholders

1. End Users (Buyers & Sellers):

- **Buyers (Customers):** Users who browse, search, and purchase agricultural products. Their feedback ensures that the system provides easy navigation, secure transactions, and effective product filtering.

- **Sellers (Farmers, Local Producers, Vendors):** Users who upload, manage, and sell products on the platform. Their needs influence features like product uploading, profile management, and sales tracking.
- **Contribution:** End users provide continuous feedback during Sprint Reviews and influence the Product Backlog prioritization.

2. Product Owner:

- Mainly represents the voice of the customer and business stakeholders.
- Defines and manages the product backlog, sets priorities, and ensures features deliver maximum value.
- In *AgroCulture*, the Product Owner ensures that critical features like product search, cart, and profile management align with customer expectations.

3. Development Team:

- A cross-functional group of developers, designers, and testers.
- Responsible for turning backlog items into functional increments (e.g., implementing '*productMenu.php'* or '*myCart.php'*).
- Self-organizing and collaborative, they ensure quality, scalability, and iterative delivery.

5.2 Secondary Stakeholders

1. Scrum Master:

- Facilitates Agile processes and ensures that the Scrum framework is followed.
- Removes obstacles (technical or organizational) and shields the team from distractions.
- Helps maintain team motivation and ensures productivity.

2. System Administrators / IT Support:

- Manage deployment, hosting servers, database reliability, and uptime.
- Ensure security patches, backups, and smooth technical operations.

3. Business Sponsors / Investors:

- Provide financial resources and strategic direction for the project.
- Expect timely delivery of features that support revenue generation and business growth.

4. Regulatory Authorities:

- Ensure that the marketplace complies with digital commerce, consumer rights, and agricultural trade regulations.
- May influence security standards (e.g., data privacy laws) and licensing requirements.

5.3 Tertiary Stakeholders

1. Third-Party Service Providers:

- Includes payment gateways, logistics partners, and external APIs.
- Their integration directly impacts user experience (e.g., secure payments, product delivery).

2. Community Organizations / Cooperatives:

- Local agricultural cooperatives and associations who may adopt or promote the platform.
- Provide credibility and encourage wider adoption among farming communities.

3. Academic / Research Partners (Optional):

- Universities or research groups may use the system for analyzing agricultural trends, digital adoption, or consumer behavior.

5.4 Importance of Stakeholder Identification

- **Feedback Loop:** In Agile, stakeholders provide feedback after every sprint, ensuring continuous alignment with real-world needs.
- **Risk Reduction:** By engaging stakeholders early, potential issues (e.g., compliance or usability problems) are identified before they become critical.
- **Value Delivery:** Helps the Product Owner prioritize backlog items that deliver the highest business and user value.

6. Software Process Model: Agile/Scrum:

The "AgroCulture" project was developed using an **Agile methodology**, with its structure and features strongly suggesting the application of the **Scrum framework**. While formal process documents are not available, an analysis of the source code artifacts provides clear evidence of an iterative and incremental development lifecycle focused on delivering value to the end-user.

6.1 Agile/Scrum Methodology:

Agile is a contemporary method for software development that emphasizes adaptability, teamwork, and input from customers. It involves building software in small, functional increments. Scrum is a popular framework for implementing Agile, organizing work into short cycles called "sprints." This mainly allows the development team to adapt changes and deliver a working product more efficiently.

For the project like "AgroCulture," an iterative model like Agile/Scrum is significantly more effective than a traditional sequential model (e.g., Waterfall). A Waterfall approach would require all requirements to be defined upfront, which is impractical for a digital marketplace where user feedback can significantly alter the direction of features.

- **Flexibility over Rigidity:** Agile allowed the development team to build the application feature by feature, such as delivering a functional user profile system (‘profileView.php’, ‘profileEdit.php’) before finalizing the complexities of the shopping cart (‘myCart.php’). This ensures that a usable product is available at every stage.
- **Customer Collaboration:** The model facilitates continuous feedback. A demo of the ‘productMenu.php’ page could lead to stakeholder requests for new filter categories, which can be easily incorporated into the next development cycle, a task difficult in a rigid, pre-planned model.

6.2 Incremental Delivery: The project is a collection of distinct features. Delivering these features in increments ensures that the most critical functions (like user registration and product viewing) are developed and released first, providing immediate value and allowing for early user testing

6.3 Evidence from the "AgroCulture" Project:

The choice of an Agile/Scrum model is inferred from the following characteristics of the project's codebase:

A. Component-Based Architecture:

The application is broken down into distinct, feature-oriented components, a hallmark of Agile development. This modularity allows for features to be developed, tested, and integrated incrementally.

- **User’s Profile Management**: Functionality for users to view and manage their profiles is handled by dedicated files ‘profileView.php’and ‘profileEdit.php’.
- **Digital Marketplace**: The core e-commerce features are separated into logical parts:
    - A main market page directs users to different functions.
    - A product menu displays items, which can be filtered by category.
    - A separate shopping cart page (myCart.php) manages items the user wishes to purchase.
- **Product Discovery**: A dedicated productSearch.php file exists to help users find specific items.

B. Iterative and Incremental Development:

The component-based structure strongly suggests an iterative development plan, where each component represents a "user story" or feature completed in a separate sprint.

- A potential development cycle could have focused on delivering the user profile first (profileView.php, profileEdit.php).
- A subsequent sprint could have delivered the ability to view products (productMenu.php).

C. Focus on User Value and Reusability:

The project emphasizes delivering functional software and efficient development, which are key principles of the Agile manifesto.

- **Working Software**: Every provided PHP file is a functional part of the application that delivers direct value to an end-user, from logging in and editing a profile to searching for and adding products to a cart.
- **Reusable Components**: The project uses a central menu.php file for navigation, which is included in multiple other pages such as myCart.php, market.php, and profileView.php. This practice avoids code duplication and aligns with the Agile principle of efficient and sustainable development.

6.4 The Scrum Framework in Detail:

6.4.1 Scrum Team Roles and Responsibilities:

- **Product Owner:** This individual acted as the voice of the customer. Their primary responsibility was to manage and prioritize the **Product Backlog**. They would have defined the acceptance criteria for each feature. For instance, for the shopping cart feature, the criteria would be: "A user must be logged in. A user can add a product. The cart page (myCart.php) must display the correct product image, name, and price."
- **Scrum Master:** This role focused on process facilitation and removing impediments. The Scrum Master would ensure that Scrum events (like the Daily Scrum) were productive and would shield the Development Team from external distractions, allowing them to focus on their tasks, such as implementing the database logic (db.php) without interruption.
- **Development Team:** The developers responsible for building the software. This team was self-organizing, collectively deciding how to turn Product Backlog items into a functional increment of the application. For example, during a sprint, one developer could work on the front-end styling in login.css while another could implement the back-end PHP logic in profileEdit.php and its corresponding database update script.

6.4.2 Scrum Events: The Rhythm of the Project:

The development of "AgroCulture" was structured around a series of recurring events within fixed-length **Sprints** (e.g., two-week cycles).

- **Sprint Planning:** At the start of each sprint, the team would select a set of high-priority items from the Product Backlog to form the **Sprint Backlog**.
- **Example Sprint: "Product Discovery"**
    - **Sprint Goal:** "The user will be able to easily find any product they are looking for."
    - **Selected Items:** User Story 1: "As a buyer, I want to filter products by category." User Story 2: "As a buyer, I want to search for a product by its name."

- **Technical Tasks Created:**
    1. Modify productMenu.php to include an HTML filter dropdown.
    2. Write PHP logic in productMenu.php to read the $_GET parameter for the filter and modify the SQL query accordingly.
    3. Create the search form UI in productSearch.php.
    4. Implement the back-end logic to process the search query.

- **Daily Scrum:** A short daily around fifteen minutes meeting for the Development Team to coordinate their activities. It is not a status report but a planning session. A developer might say last day I completed the UI designs for profileView.php and today I will work on connecting it to the session variables to display user data. I have no roadblocks.

- **Sprint Review:** This spring meeting held at the end of the sprint for demonstrating the completed work to stakeholders. The Development Team would present the now-functional product filtering on the productMenu.php page. This is not a formal presentation but a hands-on session where stakeholders can try the new features and provide feedback that will be added to the Product Backlog for future sprints.

- **Sprint Retrospective:** An internal meeting for the Scrum Team to reflect on the sprint and identify opportunities for process improvement. The team might conclude, "We spent a lot of time debugging database connection issues. For the next sprint, let's create a more robust db.php connection function to be reused everywhere."

6.4.3 Scrum Artifacts: Creating Transparency:

- **Product Backlog:** The master list of all desired features for "AgroCulture," prioritized by the Product Owner. Each item would be a user story with clear acceptance criteria.

User Story	Priority
User should register and log in to the system.	High
User should visit profile page to view and edit their info (profileView.php, profileEdit.php).	High
Use should view a list of all products (productMenu.php).	High
User will add items to the shopping cart (myCart.php).	Medium
User will filter products by category (productMenu.php).	Medium
Farmer should upload a new product (uploadProduct.php linked from profileView.php).	High

The Agile/Scrum model is not merely a theoretical fit but is demonstrably the blueprint used to construct the "AgroCulture" application. The project's architecture, characterized by its distinct, feature-driven components (myCart.php, profileView.php), and the clear separation of concerns (e.g., menu.php for navigation), are direct outcomes of an iterative and incremental process. This methodology empowered the team to build a complex, user-focused application in a flexible, efficient, and transparent manner.

7. Configuration Management Plan:

7.1 Purpose: For establishing a structured process it is necessary to properly manage, control and maintain all project artifacts. For our Farm Management System, these artifacts include- source codes, database schema, different types of documentation, release packages etc. Throughout the software development lifecycle, consistency, traceability and integrity are required for ensuring the quality of the software product.

7.2 Baseline: A baseline in software configuration management refers to a version of a product or component that has been formally reviewed and approved, acting as a stable reference point. After it is set, a baseline can only be modified through a structured change control process.

Key Characteristics of a Baseline:

1. It mainly act like point of reference in future development phases for comparison.
2. It can be duplicated or restored at any moment using version control.
3. Maintains uniformity across development, testing, and deployment environments.
4. Modifications necessitate approval to ensure stability and traceability.

Importance of Baselines:

1. Establish a clear milestone in the development workflow.
2. Aid in monitoring progress and pinpointing issues.
3. Allow reverting to a stable version in case of errors.
4. Facilitate audits and compliance by documenting precise system states.

Baseline Types:

1. Development baseline: A record of the application during its active development phase. This baseline changes regularly as developers add new features, resolve bugs, and modify the database schema. Also, it defines the branches or commits that are currently being worked on.

For the Farm Management System, it includes-

1. The most recent PHP code from development or feature branches
2. Ongoing database schema modifications
3. Updated documentation drafts

Purpose in the Farm Management System –

1. Enables collaborative coding for modules like crop management, sales tracking, and inventory.
2. Supports integration testing prior to moving features to a release candidate.

2. Release baseline: A reliable, validated version of the application that is ready for release to a staging or production environment. This version is a potential candidate for the upcoming production rollout. A verified tag such as **(v1.0-release)** that includes the application code and a snapshot of the SQL **(agroculture.sql export)** for our Farm Management System software.

In Farm Management System, contents include:

1. A tagged Git release **(v1.0-release)**.
2. A corresponding MySQL database export **(agroculture.sql)** that has been verified with the code.
3. Release notes and instructions for installation.

Purpose in the Farm Management System:

1. This ensures that the tested code and database schema work seamlessly together without any issues.
2. It also offers a consistent version for both deployment and rollback.

3. Production baseline: The specific version of the Farm Management System that is actively operating in the production environment and utilized by actual users is the most reliable and regulated baseline. The code that is presently deployed, modifications are made solely through pull requests and release tags.

In Farm Management System, components include:

1. Deployed PHP code.
2. The associated live database structure and configurations.
3. Authorized hotfixes implemented via controlled change requests.

Role in the Farm Management System:

1. Ensures system reliability for farmers, staff, and stakeholders who use it on a daily basis.
2. Acts as the primary rollback option in the event of deployment problems.

7.3 Versioning: Versioning involves assigning distinct identifiers to different versions of the Farm Management System's source code, database structure, and associated resources. This process enables the development team to effectively track, manage, and retrieve any version of the project whenever necessary.

Purpose Farm Management System Project:

1. **Traceability**: Guarantees that every modification-whether it's implementing a new crop tracking feature or correcting an issue with a sales report is recorded and can be traced back.
2. **Rollback Support**: Ensures the system can revert to a stable version if recent changes cause issues.
3. **Parallel Development**: Facilitates the simultaneous development of multiple features without conflicts.

Implementation in This Project:

Git-based Version Control: The files such as PHP, HTML templates, CSS & JS resources, and the **(agroculture.sql)** database dump, are managed with the help of Git repository.

Branching Strategy:

- Development branch for ongoing development work.
- Release branch for versions that have been tested and are stable.
- Main or Production branch used for live deployment.

Tagging: Each release is marked with a semantic versioning format such as v1.0.0, v1.2.1.

Database Versioning: Changes to the MySQL schema are archived as updated SQL dumps with corresponding version tags to avoid discrepancies between the code and the database.

Change Log: A (CHANGELOG.md) file is maintained to record every change made.

7.4 Roles and Responsibilities: In the Farm Management System project, well-defined roles and responsibilities facilitate effective collaboration, consistent version management, and dependable deployments. Each team member is assigned specific tasks that align with configuration management practices, covering all aspects from development and testing to release and ongoing maintenance. This setup helps ensure system stability, avoids version conflicts, and guarantees that both the application code and the database stay synchronized throughout the project's lifecycle.

Below table is attached to mention the roles and responsibilities for each member of our group to successfully complete the Farm Management System Project.

Roles & Responsibilities Table

Role	Team Member	Responsibilities
Leader	*FARDIN HOQUE*	Approves release schedules and deployment plans, Oversees adherence to versioning policies, Coordinates between developers, testers, and deployment teams.
Process Analyst	*FARDIN HOQUE & RIYA BASAK RISHA*	Maintains Git repository structure and access control, Creates and manages baselines (development, release, production), Ensures database versions match code for each release, Approves and merges pull requests into release and production branches.
Developer	*SHAH FAHIM CHOWDHURY*	Develops assigned modules such as crop inventory, expense tracker, weather integration), Follows version control workflow (branch naming, commit messages), Updates and shares database schema changes.
Documentation Lead	*RASHIDA BEGUM LIMA*	Tests the system in staging using release baselines, validates database integrity and functional requirements, Approves versions before production deployment.
Deployment Engineer	*RIYA BASAK RISHA*	Deploys approved versions to the live server, maintains backups of code and database for production baselines, Monitors stability after deployment and applies hotfixes as needed.

8. Use Case Diagram: The Use Case diagram for the Farm Management System is attached below-

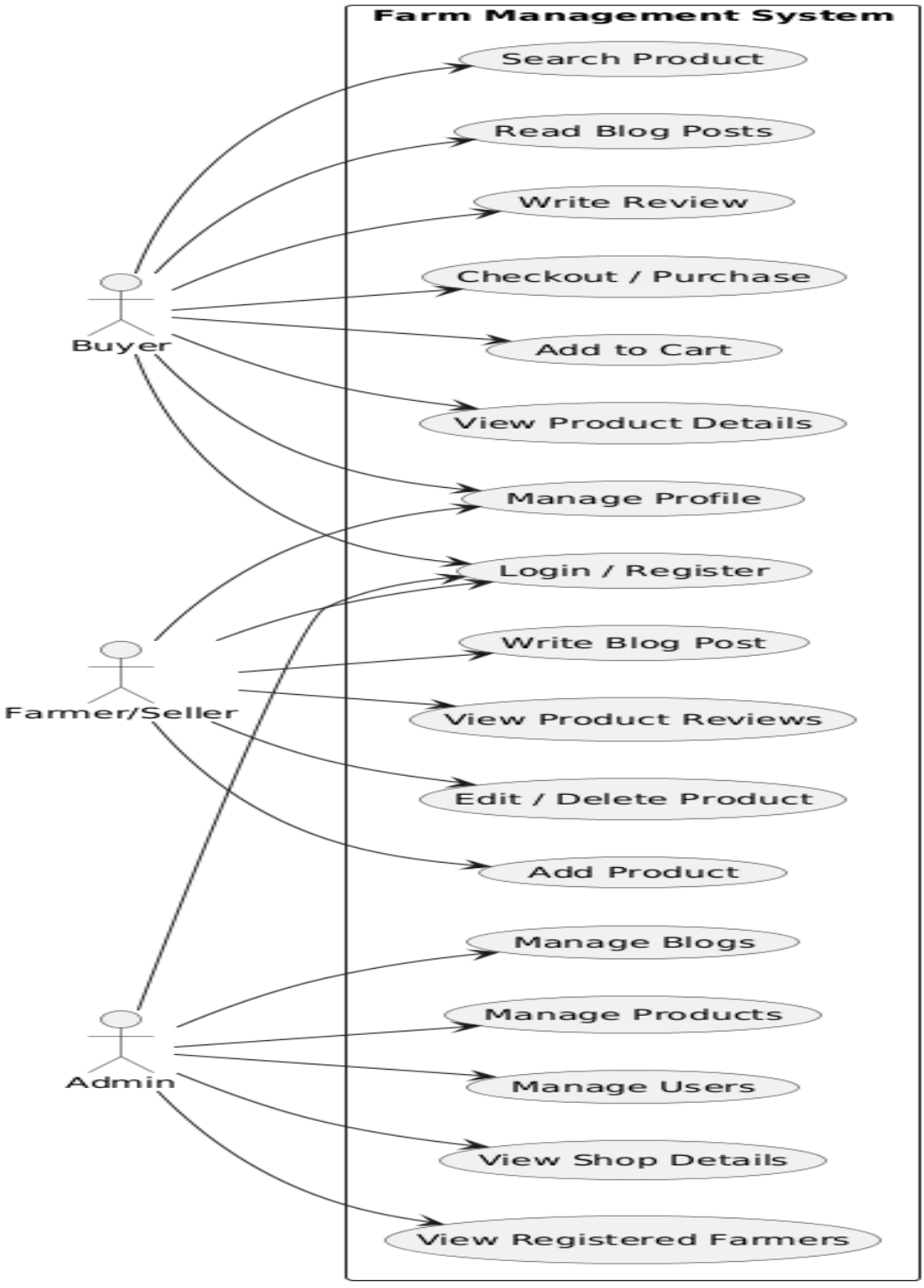


Figure: Use Case Diagram of FMS

9. Functional Specifications:

The Farm Management System is an online platform designed to connect farmers, buyers, and administrators. It enables the sale of products online, sharing blogs, and provides administrative oversight of the marketplace. The functional requirements are organized based on the three primary users of the system.

9.1 Buyer Functional Requirements:

Buyer Registration & Login:

- The system allows buyers to create a new account by fulfilling the registration form.
- The system allows the buyer's email or username and password before permit access.
- The system offers buyers to recover forgotten passwords via email.

Manage Buyer Profile:

- The system allows buyers to update personal information such as name, email, phone, and address.
- The system validates input fields to ensure data integrity.

Search and View Products:

- The system allows buyers to search products by using name, price range, or category.
- The system offers display product details, including name, price, description, seller information, images.

Add to Cart:

- The system allows buyers to add products to shopping cart for later checkout.
- The system offers display the updated cart with quantities and prices.

Checkout / Purchase:

- The system allows buyers to proceed to checkout from the cart.
- The system supports payment through available payment gateways or COD (Cash on Delivery).
- The system provides generate an order confirmation.

Write Product Review:

- The system allows buyers to write reviews for purchased products.
- The system display reviews alongside the product details.

Read Blog Posts:

- The system allows buyers to view and read blogs posted by farmers.
- The system allows searching blogs by title or category.

9.2 Farmer/Seller Functional Requirements:

Farmer Registration & Login:

- The system allows farmers to create an account with personal details such as farm name, location, and contact info.
- The system need verify credentials before allowing access.

Manage Farmer Profile:

- The system allows farmers to update their profile information and farm details.
- The system offers changing passwords securely.

Add Product:

- The system allows farmers to list recent products by entering product name, description, category, price, and uploading images.
- The system shall validate required fields before submission.

Edit / Delete Product:

- The system allows farmers to update product details or remove them from the store.
- The system prevents deletion if an active order is pending.

View Product Reviews:

- The system allows farmers to view reviews and ratings for their products.
- The system displays reviewer details and date.

Write Blog Post:

- The system offers farmers to create and publish blog posts with text and images.
- The system shall allow editing and deleting blog posts.

9.3 Admin Functional Requirements

Admin Login:

- The system allows the admin to log in with a secure username and password.

- The system should restrict access to admin features without proper authentication.

Manage Users:

- The system allows the admin to view all registered buyers and farmers.
- The system allows the admin to deactivate, activate, or delete user accounts.

Manage Products:

- The system offers the admin to view all listed products.
- The system offers allow removing inappropriate or prohibited products.

Manage Blogs:

- The system allows the admin to view, approve, or remove blog posts that violate policies.

View Registered Farmers:

- The system shall allow the admin to show the details of all farmers, including contact info and farm name.

View Shop Details:

- The system allows the admin to view the shop/product listing details for each farmer.

10. Gantt Chart: The Gantt chart for FMS is attached below-

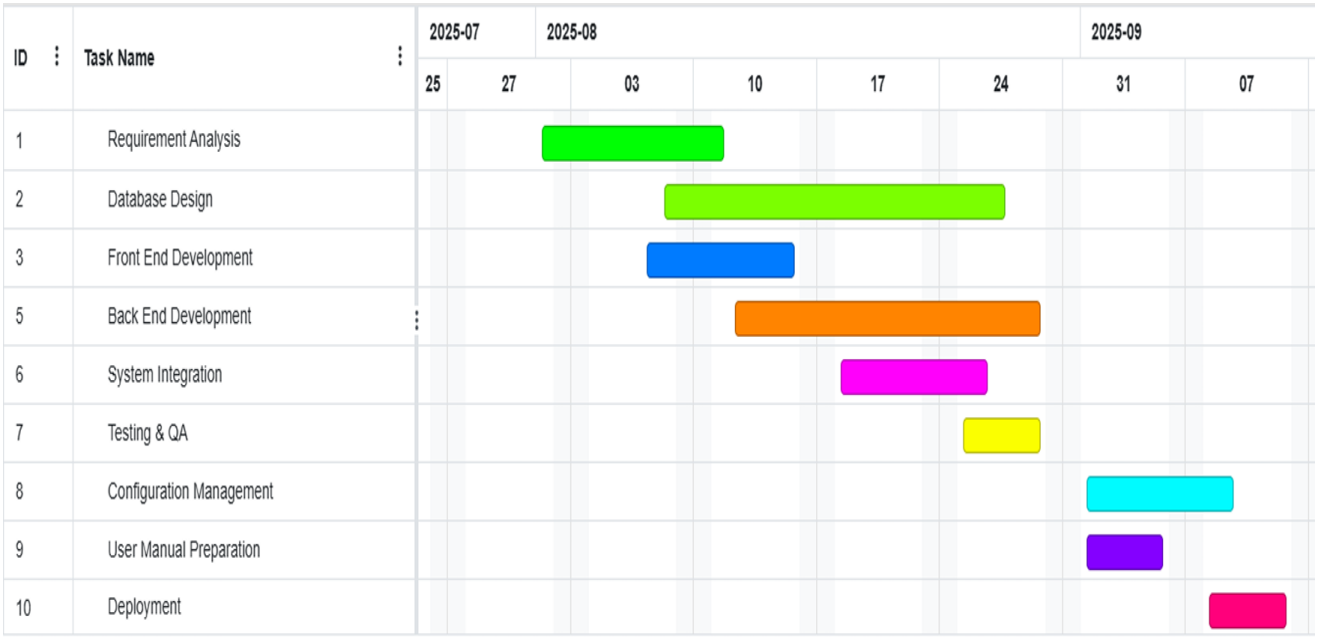


Figure: Gantt Chart of FMS

11. Budget:

The budget forecast details the anticipated expenses for development, testing, deployment and ongoing maintenance of our "Farm Management System". These costs are determined by evaluating resource allocation, the necessary hardware and software, employee salaries, and contingency funds.

Table: Budget Analysis

Category	Description	Estimated Cost (BDT)
Personnel Costs	Salaries for project team members including Project Manager, Configuration Manager, Developer, QA, Deployment Manager.	BDT 80,000
Hardware & Infrastructure	Servers, backup storage, networking equipment, and local development machines.	BDT 10,000
Software & Tools	Licenses for IDEs, MySQL database, project management tools (e.g., Jira, Trello), and design software (Figma, Photoshop).	BDT 30,000
Cloud Hosting	AWS/Google Cloud hosting for production and staging environments	BDT 10,000
Testing Resources	Test devices, simulation tools, and automated testing frameworks.	BDT 20,000
Training & Documentation	Staff training sessions and preparation of user/developer manuals.	BDT 20,000
Contingency (10%)	Risk buffer for unexpected costs during the project lifecycle.	BDT 10,000

12. Tools Selection Justification:

The choice of tools for this project is contingent on supporting version control, collaboration, automation, and deployment. Every chosen tool supports Agile/Scrum principles and will support development and maintenance.

Here are justifications for each tools:

1. Git – Git is provided by virtually all the version control systems Cherin encountered. GitHub also allows for pull requests, code review, issue tracking, and branch protection, resulting in optimal team collaboration capabilities. Git is a distributed version control system, meaning each developer has an entire local copy of the codebase while online and offline.

2. GitHub – GitHub is the best CI/CD pipeline configuration tool because it has built-in CI/CD automation (not much setup required from the team). There is no external server that needs to be paid for. Set up a build, test, and deploy process directly in GitHub, substantially saving money and time for small teams.

3. XAMPP – Without requiring much configuration, XAMPP offered a local development setup that brought together Apache, MySQL, and PHP, allowing users to run and test applications locally, making it suitable for offline use.

4. MySQL – Since the existing source code was built off this stack, it provided continuity. PHP is a server-side language that is fast for developing web applications. 'MySQL' is a popular, Lightweight and it's an open-source relational database system.

5. phpMyAdmin – phpMyAdmin is a web-based database administration tool that allowed an easy way to administer data to MySQL database using a web GUI without having to really know any SQL commands.

6. Visual Studio Code – the motive for choosing this IDE was similar lightweightness of battery consumption, fast is my development environment and help. A simple debugger and wide array of tools and extensions are available too.

7. Draw.io / Lucidchart – Rather than make write-up documents on use case diagrams, ERDs, architecture models, I used these tools to prepare visual diagrams for better documentation.

8. Teams – Offers chat, video calls, and file sharing on one platform for real time collaboration, ensuring tighter integration with organizational workflows.

This combination of tools helped the development team manage tasks efficiently, quickly troubleshoot issues, and ultimately deliver a performant and reliable farm management system.

Table: Tool Selection Summary

Tool/Technology	**Purpose**	**Justification**
Git	Version control	Tracks changes, supports branching, enables rollback
GitHub	Remote repository	Centralized code hosting, collaboration, CI/CD integration
XAMPP	Local server	Easy local setup of Apache, MySQL, PHP for testing
MySQL	Database	Reliable relational DB for structured data storage

phpMyAdmin	DB management	GUI for easier database administration
Visual Studio Code	IDE	Lightweight, extensible, integrated debugging
Draw.io / Lucidchart	Diagramming	Easy creation of system diagrams for documentation
Microsoft Teams	Team Communication	Ease of access, sharing contents, real-time collaboration and operating daily stand-ups

13. Work Distribution in Scrum Framework:

In this project, the work distribution was accomplished using the Scrum Agile methodology which focuses on iterative and incremental development. Our work is divided into sprints, with each sprint delivering a product increment that can be potentially shippable. The development team divided into roles such as product owner', scrum master', development Team and others supports accountability and collaboration.

Key scrum activities include:

- Sprint Planning – Picking items from the product backlog level and refining them for the Sprint
- Daily Stand-up – Short daily touch point with team for updates and blockers
- Sprint Review mainly demonstrates product increment to stakeholders
- Sprint Retrospective reflects back on process to improve team performance
- Product Backlog Refinement – Reviewing and updating the product backlog items in the project
- Sprint Goal Definition – clear objectives of what the team will focus on during the sprint
- Release planning – Note what items will be planned for the next product releases

- Increment Delivery & Validation – Validate completed product increment based on the process defined in the sprint

The methodology allows for development that can adapt and leverage feedback while aligning with any changes to requirements. The following Scrum-based Work Distribution Sheet provides directions on our sprints, backlog items, responsible roles and project deliverables:

Table: Work Distribution Table

Sprint	**Backlog Item / Task**	**Description**	**Role Responsible**	**Deliverable**	**Duration**
Sprint 1	Requirement Analysis	Gather farm management system needs and prepare Product Backlog.	***Fardin Hoque***	Initial Product Backlog Document	2 weeks
Sprint 1	Database Design	Create ER diagram and MySQL schema.	***Riya Basak Risha***	ER Diagram, Schema SQL File	2 weeks
Sprint 2	Frontend Development	Design UI in PHP/HTML/CSS/JS.	***Rashida Begum Lima***	UI Mockups, Working Pages	2 weeks
Sprint 2	Backend Development	Implement core features: login, product CRUD.	***Fardin Hoque***	PHP Scripts, API Endpoints	2 weeks
Sprint 3	Integration	Connect frontend and backend, sync with DB.	***Shah Fahim Chowdhury***	Fully Functional Module	1 week
Sprint 3	Testing & Bug Fixes	Unit, integration, and acceptance testing.	***Riya Basak Risha***	Test Reports, Fixed Bugs	1 week
Sprint 4	Deployment & Documentation	Host final system, prepare manuals.	***Rashida Begum Lima & Shah Fahim Chowdhury***	Live System, Documentation	1 week

14. System Design: Farm Management System

14.1. UML Class Diagram of Farm Management System:

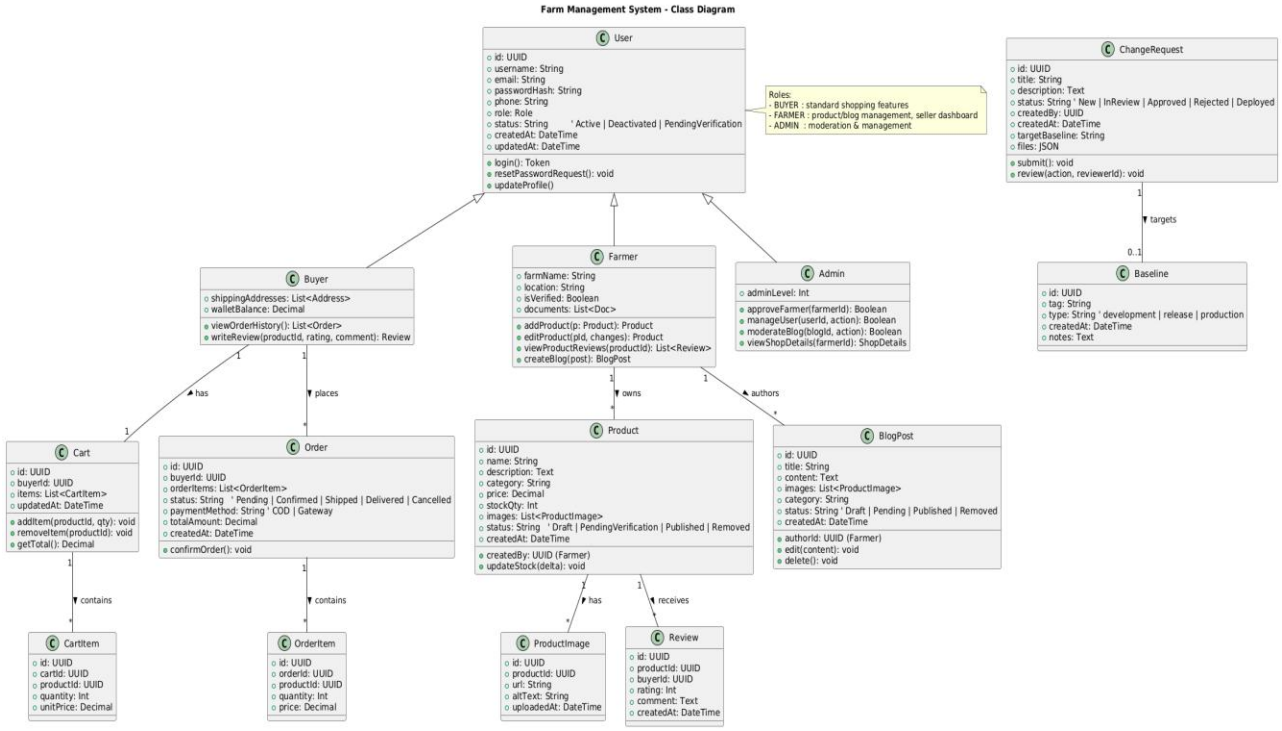
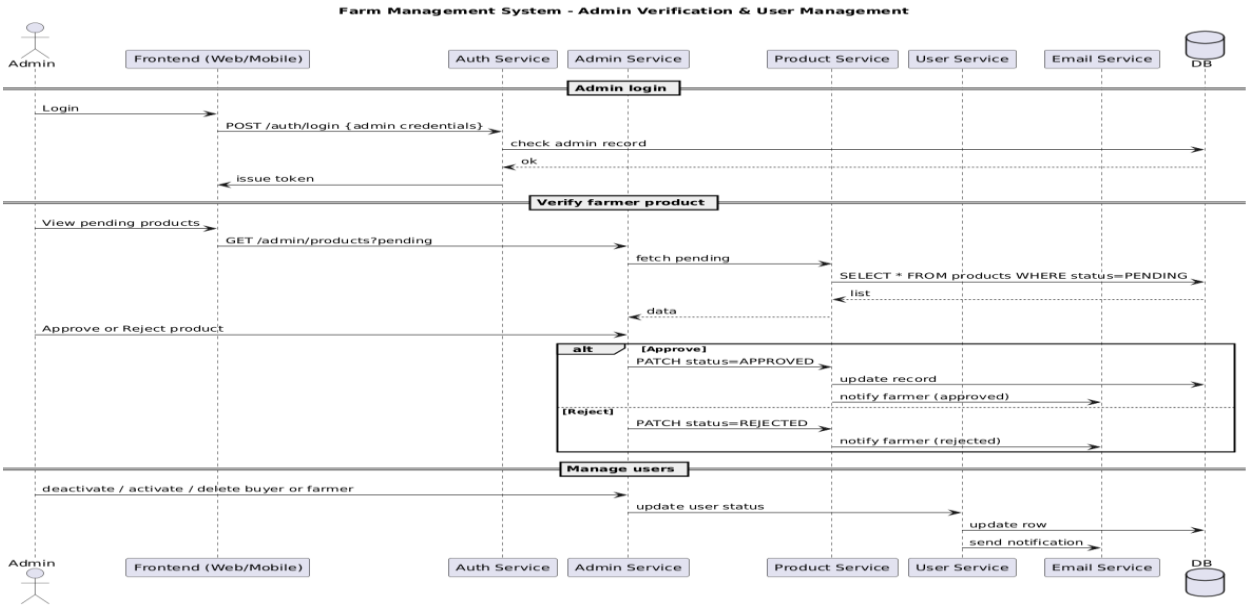


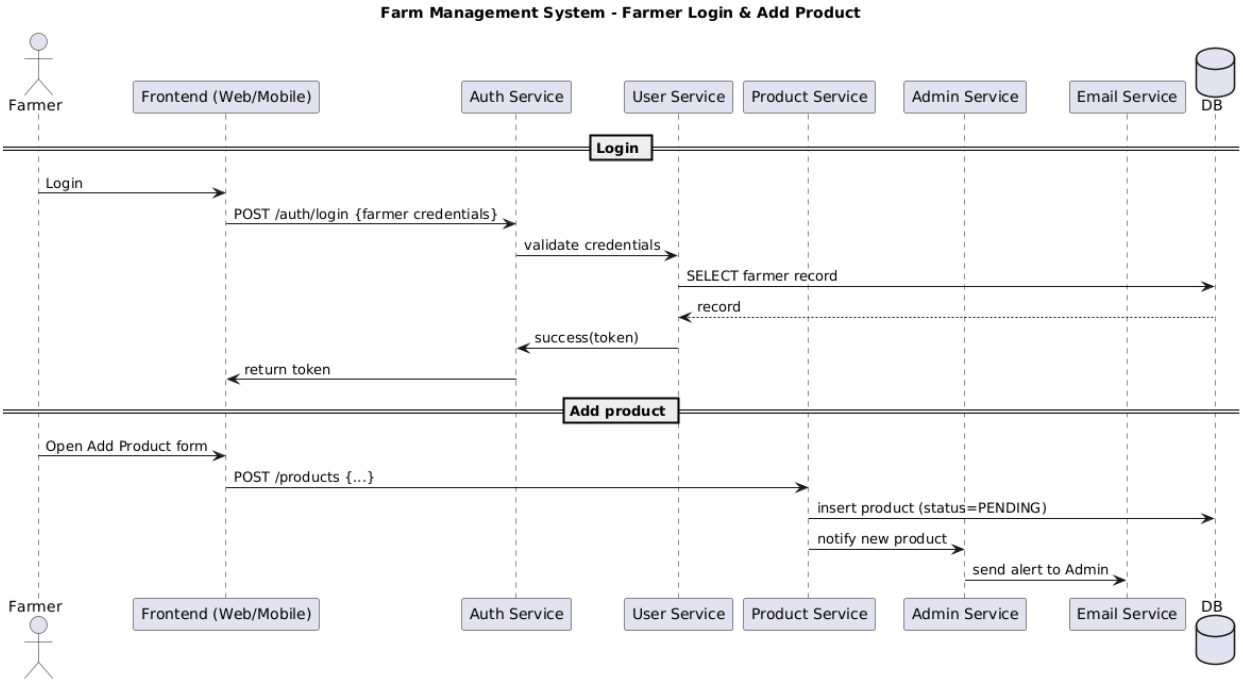
Figure: Class Diagram of FMS

14.2. UML Sequence Diagram: The sequence diagrams are divided into four sections for our Farm Management System Project. There are three types of users in our system: Admin, Farmer and Buyers. Their roles and functions are shown in the sequence diagrams respectively.

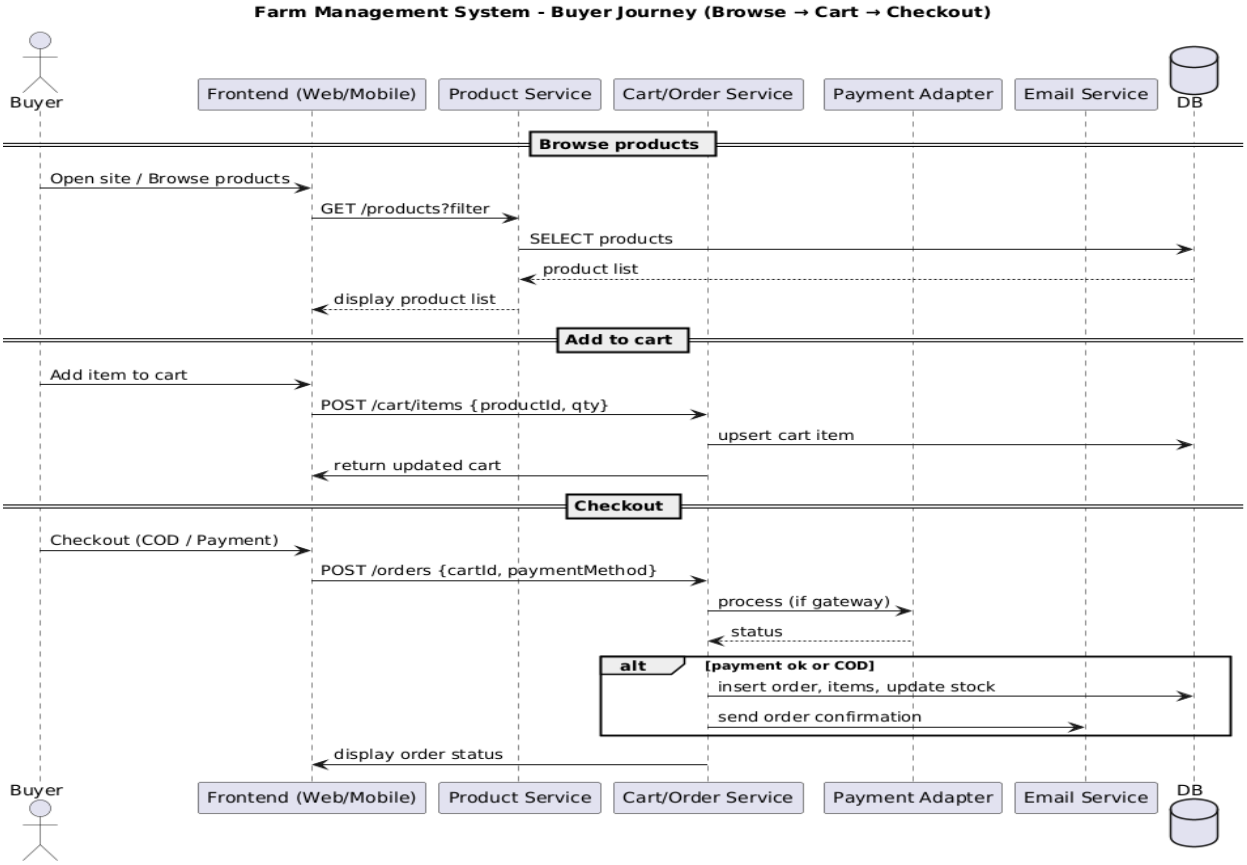
14.2.1. Admin's Sequence Diagram – Verify Farmer Product & Manage Users



14.2.2. Farmer's Sequence Diagram – Farmer Login & Add Product



14.2.3. Buyers Sequence Diagram –



14.3. Version Control Policy Design:

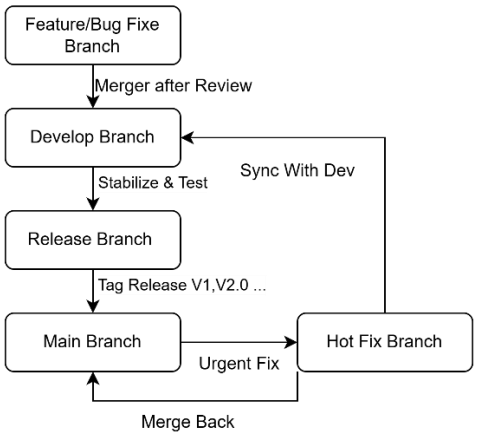


Figure: Version Control Flow

1. Branching Strategy
   - Main Branches
     - main → Always stable, production ready code.
     - develop → Integration branch for features before release.
   - Supporting Branches
     - feature/<feature-name> → For new feature development.
     - bugfix/<issue-id> → For fixing defects.
     - release/<version> → Used to prepare a new release (testing, documentation).
     - hotfix/<issue-id> → Quick fixes for urgent issues on production.
2. Commit Policy
   - Commit messages must be clear and follow a convention:

type>(<scope>): <short description>
Types: feat, fix, docs, style, refactor, test, chore
Example:
   - feat(auth): add user login and registration
   - fix(db): corrected schema for farm products table
   - Commits should be small and frequent, not large dumps of code.
3. Merge & Pull Request Rules
   - All merges to develop and main must be done via Pull Requests (PRs).
   - Each PR should be reviewed by at least one other team member.
   - Conflicts must be resolved locally before merging.
   - Use squash merge for small features to keep history clean.
4. Versioning Scheme
   - Follow Semantic Versioning (SemVer):
   - MAJOR.MINOR.PATCH
     - MAJOR → Breaking changes (v2.0.0)
     - MINOR → New features, backward-compatible (v1.1.0)
     - PATCH → Bug fixes, no new features (v1.0.1)
   - Releases tagged in GitHub with version numbers.

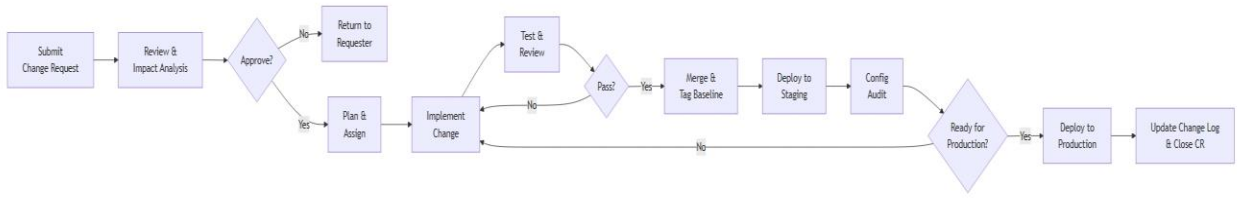
5. Access Control & Roles

- Product Owner (Fardin Hoque) → Approves releases, manages main branch.
- Scrum Master (Riya Basak) → Ensures process is followed, reviews merges.
- Developers (Shah Fahim) → Works on feature/bugfix branches.
- Tester/QA (Rashida Lima) → Validates changes before merging into develop

The version control policy for the Farm Management System project is based on Git and GitHub, ensuring collaboration, traceability, and quality control. The project adopts a branching model with a stable main branch, a develop branch for integration, and supporting branches for features, bug fixes, releases, and hotfixes. Commit messages follow a structured format (type, scope, description) to improve clarity. All changes to develop and main require pull requests with peer review, ensuring code quality and preventing accidental overwrites. The project follows semantic versioning (MAJOR.MINOR.PATCH) for tagging releases, making it easy to track progress and manage updates. Role-based access ensures that each team member contributes effectively: the Product Owner manages releases, the Scrum Master enforces process, developers work on features, and the QA engineer validates changes.

14.4. Change Request Flowchart:

The change request flowchart details for Farm Management System are described below-

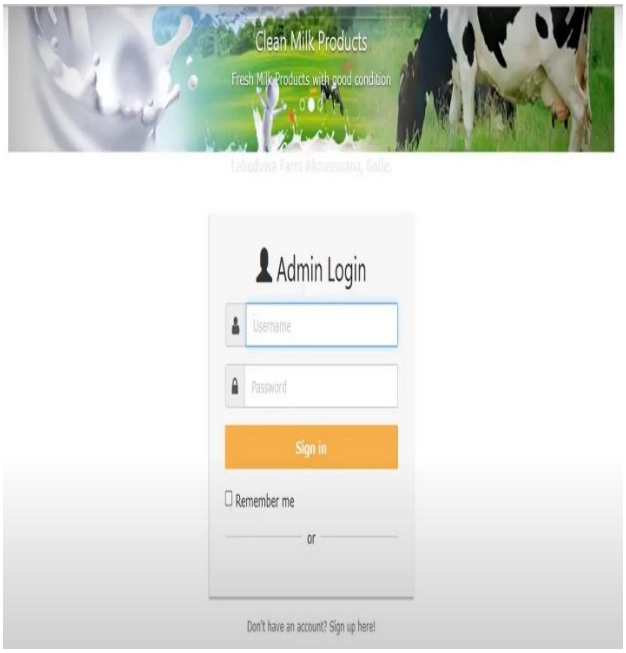


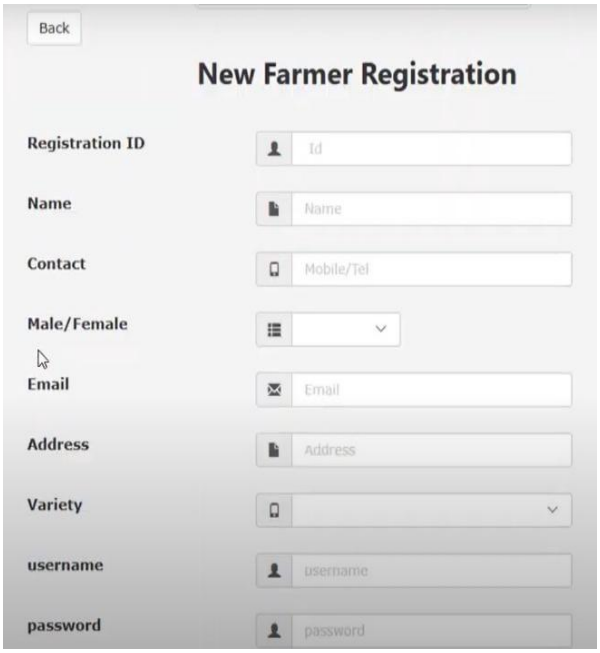
1. Submit Change Request– The individual requesting the change submits a Change Request (CR) with relevant information.
2. Review & Impact Analysis– The team assesses the potential risks, costs, and feasibility associated with the change.
3. Approval Decision–
    - If No → Return the request to the requester.
    - If Yes → Proceed to the next phase.
4. Plan & Assign– Formulate a plan and designate responsibilities for implementing the change.
5. Implement Change– Execute the planned change.
6. Test & Review– Confirm that the functionality operates as intended.
    - If it fails → Reimplement the change.
    - If it passes → Continue to the next step.
7. Merge & Tag Baseline– Update the code repository to reflect the stable version.
8. Deploy to Staging– Conduct tests in the staging environment.
9. Configuration Audit– Verify compliance and readiness for production deployment.
10. Ready for Production-
    - If No → Perform a recheck.
    - If Yes → Move forward with deployment.
11. Deploy to Production– Launch the change into the live environment.

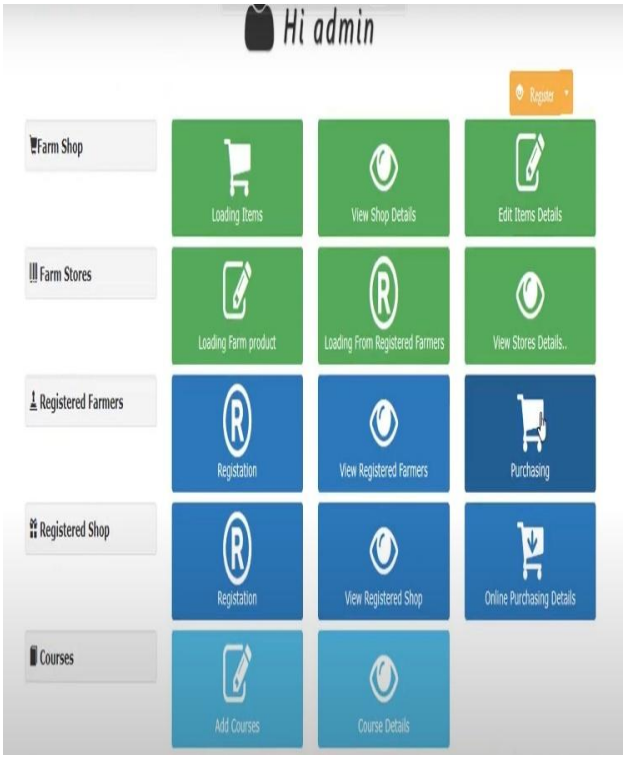
12. Update Change Log & Close CR– Record the change in the log and formally close the request.

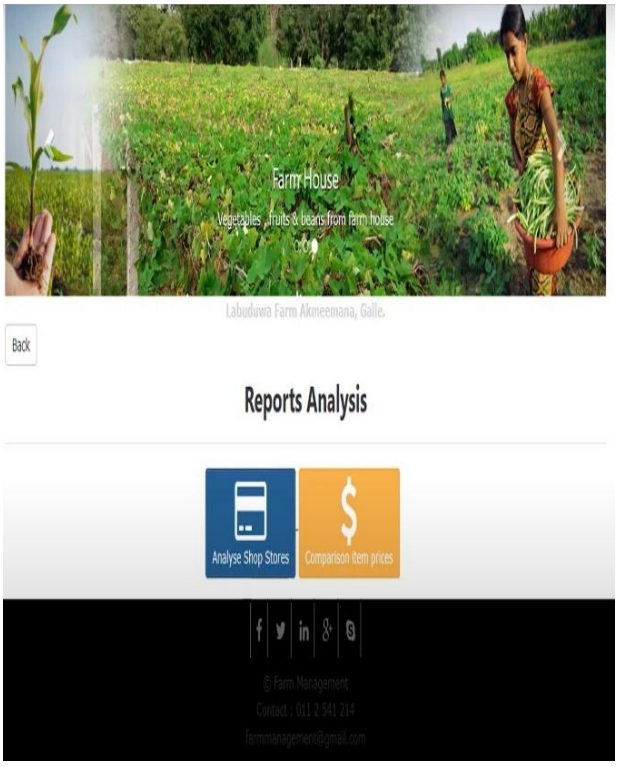
15. Implementation:

15.1. User Interfaces of Farm Management System









16. Features:

16.1. Change Request Submission

The Change Request Submission process is the formal starting point for any proposed modification to the "Farm Management System" web application. Its purpose is to provide a structured, documented, and transparent method for stakeholders to request changes, whether it's fixing a bug, adding a new feature, or altering existing functionality. This process ensures that no unauthorized work is undertaken and that every proposed change is formally evaluated for its technical impact, business value, and resource requirements before being approved. For the "Farm Management System" project, this applies to any modification of its components, from the profileView.php page to the underlying database structure.

To understand how this process works for the "Farm Management System" project, let's consider a realistic scenario that might occur after the initial launch of the application.

Step 1: Identifying the Need for a Change
After the initial launch, the project team observes that while sellers can upload new items via the uploadProduct.php link on their profile, they have no way to manage these products afterward. They cannot edit the price or description, nor can they remove a product that is sold out. This is a significant functional gap.

Step 2: Initiating the Formal Request
Instead of informally asking a developer to add the feature, the Project Manager or Product Owner initiates the formal process by completing a **Change Request (CR) Form**. This ensures the requirement is captured in detail and can be tracked.

Step 3: Completing the "Farm Management System**" Change Request Form**
The form is filled out with specific details pertaining to the project:

- **Change Request ID:** CR-047
- **Title:** "Feature: Implement Seller Product Management Dashboard"
- **Requestor:** Product Owner
- **Submission Date:** September 12, 2025
- **Description of Change:**
  Currently, registered users have a link on their profileView.php page to uploadProduct.php. However, once a product is submitted, there is no functionality for the seller to view, edit, or delete their own listings. This request is to create a new 'My Products' dashboard where a logged-in seller can see a list of all products they have submitted and have options to edit or delete each one.
- **Justification / Business Need:**
  This is a critical feature for seller retention and marketplace accuracy. Without it, sellers cannot manage their inventory, update pricing, or remove items that are no longer available. This leads to a poor user experience for sellers and results in buyers seeing outdated or inaccurate listings on the productMenu.php page.
- **Affected Configuration Items (CIs):**
  - **Existing Files to be Modified:**

        - profileView.php: A new link, "My Products," needs to be added to the user's action buttons.
        - productMenu.php: The logic may need to be updated to ensure it doesn't display products that a seller has deleted or marked as inactive.
    - **New Files to be Created:**
        - myProducts.php: A new page to display a table of the seller's products, fetched from the fproduct table.
        - editProduct.php: A new page with a form, pre-filled with a product's details, allowing the seller to update it.
        - deleteProduct.php: A back-end script to handle the logic of removing a product from the database.
    - **Database Schema:** The fproduct table may require a new column, such as seller_id, to link each product to the user who uploaded it.
- **Priority:** High

- **Change Request ID:** CR-048
- **Title:** Feature Enhancement: Add "Out of Stock" Status for Products
- **Requestor:** Fatima Chowdhury, Seller Relations Lead
- **Submission Date:** September 12, 2025
- **Change Category:** Feature Enhancement
- **Priority:** High
- **Description:**
  Currently, there is no mechanism for a seller to mark a product as "Out of Stock." Products remain listed on the productMenu.php page even when they are no longer available. This request is to add a feature for sellers to toggle a product's availability status and to visually indicate this status to buyers.

- **Justification / Business Case:**
  This feature is critical for marketplace integrity and user trust. Buyers are becoming frustrated when they attempt to purchase items that are no longer available. This leads to a poor user experience and lost sales. Providing an "Out of Stock" status will ensure the product catalog is accurate and will improve the experience for both buyers and sellers.

Affected Components:

- **Database Schema:** The fproduct table will require a new column, such as stock_status (e.g., 'In Stock', 'Out of Stock').
- **PHP Source Code:**
    - productMenu.php: Must be modified to visually display the "Out of Stock" status (e.g., gray out the item and disable the "Add to Cart" button).
    - A new page/feature (myProducts.php dashboard) will be required for sellers to manage the stock status of their listings.
    - myCart.php: Logic will be needed to prevent out-of-stock items from being purchased.

Step 4: The Review and Approval Workflow

Once submitted, the Change Request CR-047 enters the formal review workflow:

1. **Technical Impact Analysis:** A lead developer reviews the request. They confirm that the

fproduct table indeed needs a seller_id foreign key. They estimate the total work will take approximately 25-30 hours of development time.
2. **Change Control Board (CCB) Review:** The CCB (comprising the Product Owner, lead developer, and a QA tester) convenes to discuss the request.
3. **Decision:** Given the high priority and the critical need for this functionality, the CCB approves CR-047. It is then added to the project's Product Backlog and prioritized for an upcoming development sprint.

16.2. Change Log Tracking

Types of Change Logs at Farm Management System
To meet different needs, the Farm Management project maintains two distinct types of change logs: a formal log for project documents and a user-facing log for the software itself.

16.2.1. Document Change Log

For formal project documents, such as the System Requirements Specification (SRS) or this very policy document, a change log is maintained within the document itself, typically in an appendix.
Structure and Example: The following table illustrates the change history for a requirements document related to the user profile feature.

16.2.2. Software CHANGELOG.md

For the software itself, a single CHANGELOG.md file is kept in the root of the project's Git repository. This file is intended for a broader audience, including developers and stakeholders, and provides a high-level summary of changes between different versions of the application.

Structure and Example: The log is organized by version, with changes categorized for clarity.

Version	Date	Author/Editor	Description of Change	Reason for Change
1	20-07-25	*RIYA BASAK RISHA*	Initial draft of the User Profile requirements document created.	Initial requirements gathering.
1.1	01-08-25	*FARDIN HOQUE*	Added requirement for "Address" field in user profile.	As per stakeholder feedback (CR-012).
1.2	15-08-25	*SHAH FAHIM CHOWDHURY & RASHIDA BEGUM LIMA*	Added requirement for an "Upload Product" button for sellers.	To enable core seller functionality (CR-021).

Changelog - Farm Management System Application
All notable changes to this project will be documented in this file.

[1.1.0] - 2025-09-12
Added

- Feature: Sellers can now mark products as "In Stock" or "Out of Stock" from a new "My Products" dashboard. (Implements: CR-048)
- Feature: The productMenu.php page now visually indicates when a product is out of stock and disables the "Add to Cart" button. (Implements: CR-048)

Changed

- The database schema for the fproduct table was updated to include a stock_status column.

[1.0.1] - 2025-09-08
Fixed

- Bug: Corrected an image caching issue in profileView.php where a user's new profile picture would not display immediately after being uploaded. (Fixes: CR-051)

[1.0.0] - 2025-09-01
Added

- Initial release of the Farm Management System Digital Marketplace.
- User registration, login, and profile management (profileView.php, profileEdit.php).
- Product catalog with filtering (productMenu.php).
- Shopping cart functionality (myCart.php).

16.2.3 Tools and Automation

The Farm Management System project leverages an integrated tool chain to automate and enhance the change log tracking process.

- **Git and GitHub:** This is the core of our tracking system. It provides the immutable commit history, branching capabilities, and the Pull Request workflow for peer review. Commands like git log --graph are used to visualize the history directly in the terminal.

- **Project Management Tool Integration:** Our Farm Management System is configured to recognize CR IDs in commit messages. When a commit with Implements: CR-048 is pushed to GitHub, the system automatically links that commit back to the original Change Request ticket, creating a powerful, bi-directional traceability link between the business requirement and the code that implements it.

- **Continuous Integration (CI) System:** Our automated CI server monitors the main branch. When a new change (like the merger of CR-048) is detected, it automatically builds and tests the application. This ensures that the change log is not just a record of code, but a record of code that has been verified and is in a stable state.

16.2.4 Auditing and Review

The change logs are formal project artifacts and will be reviewed at key milestones. Before a new version is released, the CHANGELOG.md will be audited by the QA Lead and Project Manager to ensure all changes are documented and correspond to approved Change Requests. This review is a mandatory step in our release process and serves as a final quality gate.

16.3 Configuration Baseline Management

In our Agile/Scrum environment, baselining is not a one-time event but a recurring activity that aligns with our development sprints. The lifecycle of a baseline is deeply integrated into our workflow.

Step 1: The Sprint Commitment - The Developmental Baseline At the beginning of a new sprint, the development team commits to a set of features from the product backlog. The current state of the main branch at this point serves as an informal **Developmental Baseline**. All new feature branches (e.g., feature/product-rates) are created from this specific point, ensuring that all developers start their work from the same stable foundation.

Step 2: The Pre-Release Candidate as the sprint progresses, features are completed and merged back into the main branch via Pull Requests. At the end of the sprint, when all committed features are integrated and have passed automated tests, the main branch represents a new, feature-complete version of the application. This state, representing the fulfillment of the sprint's "Definition of Done," is the candidate for a new Product Baseline.

Step 3: The Technical Lead initiates the formal baseline process:

1. **Baseline Audit:** A final review is conducted to ensure all Configuration Items (CIs) are present and correct. This includes verifying that all source code (.php, .css files), the database migration scripts, and any new user documentation are included.
2. **Tagging:** A semantic version tag (e.g., v1.1.0) is applied to the final commit of the sprint in Git. This tag is the official, permanent marker for the baseline.
3. **CI/CD Pipeline Execution:** The Continuous Integration server is triggered to run a build from this specific tag. The process automatically archives all the CIs, including a snapshot of the database schema, creating a complete and self-contained package for the baseline.
4. **CCB Approval:** The Change Control Board (CCB) formally reviews and approves the tagged version. Once approved, v1.1.0 is declared the official Product Baseline.

Step 4: The Handover - The Baseline as a Contract The newly established Product Baseline serves as a formal "contract" between the development and Quality Assurance (QA) teams. The development team hands over v1.1.0 to QA, who will conduct their formal testing exclusively against this immutable version. This prevents ambiguity and ensures that what is tested is exactly what will be deployed.

Auditing and Verifying Baseline:
To ensure the integrity of the Farm Management System application, regular audits are performed. A baseline audit is a formal check to verify that the deployed version of the application matches the contents of its corresponding Product Baseline.

Example Scenario: Verifying the Production Environment Suppose the live website is running version v1.1.0. An audit would involve:

1. Retrieving the official v1.1.0 baseline package from the artifact repository.
2. Comparing the checksums of the deployed files (e.g., myCart.php, productMenu.php) on the web server against the checksums of the files in the baseline package.
3. Comparing the live database schema against the schema snapshot stored with the baseline. Any discrepancies would indicate an unauthorized or unrecorded change and would immediately trigger an incident response.

The Role of Baselines in Maintenance and Support:
Baselines are critical for long-term maintenance. If a user reports a bug in the production version (v1.1.0), the development team can instantly recreate the exact environment of that version.
Workflow for a Hotfix:

1. A developer checks out the v1.1.0 tag in Git to create a new branch: hotfix/login-bug-for-v1.1.0.
2. They develop and test the fix in an environment that precisely mirrors the v1.1.0 baseline.
3. Once the fix is complete and verified, it is merged back into the main branch.
4. A new patch baseline, v1.1.1, is created, formally tagged, and deployed to production. This disciplined process ensures that fixes are applied in a controlled manner and prevents the introduction of unintended side effects.

16.4 Version Tree Visualization for configuration

The visualization feature will cover all Configuration Items (CIs) managed within the project's official Git repository. This includes the complete history of all PHP source code files (e.g., profileView.php, myCart.php), front-end assets (login.css), and project documentation. The tool is designed to provide a comprehensive visual history of every version of every tracked component.

16.4.1 Technical Implementation and Features

The Version Tree Visualization tool is implemented as a dedicated front-end view within the project's internal management dashboard. It is designed to be a real-time, interactive interface for exploring the project's development history.

- **Data Integration:** The view is powered by a direct integration with our Git repository's API. This allows it to fetch the latest data on all branches, tags, and commits, ensuring the visualization is always current and accurately reflects the state of the codebase.
- **Interactive Graph:** The development history is rendered as an interactive graph. This graphical interface allows team members to perform several key actions:
    - **Zoom:** Users can zoom in and out of the graph to view specific timeframes, from the entire project history down to a single day's commits.
    - **Filter:** The view can be filtered to display the history of a single file or a specific project module. For example, a developer can filter the graph to see only the commits and branches that have affected myCart.php.

- **Configuration Item Mapping:** The system can map specific files (like profileView.php) to the versions (tags) they belong to. This allows a user to click on a baseline tag (e.g., v1.1.0) within the visualization and see the exact list of files and their states that constitute that specific release.

16.4.2 Practical Application and Workflow

The Version Tree Visualization tool is integral to the daily workflow of the Farm Management System team, especially when managing parallel tasks.

Scenario: Consider a situation where Baseline v1.0.0 has been established. Two new tasks are approved:

1. A critical bug in the user login process needs an immediate fix. A developer creates a hotfix/login-security branch.
2. A new feature for product ratings (CR-045) is scheduled for the next release. Another developer creates a feature/product-ratings branch.

How the Visualization Tool is Used:

- **Progress Tracking:** The Project Manager can view the interactive graph and see both branches diverging from the main branch. They can track the number of commits on each branch to monitor progress without needing to read raw Git logs.
- **Code Review and Integration:** When the hotfix is complete, the visualization clearly shows its history. The lead developer can review the small, isolated set of changes before approving the merger back into the main branch.
- **Dependency Management:** After the hotfix is merged, the graph updates. The developer working on the feature/product-ratings branch can visually confirm that their branch is now behind the main branch and that they need to merge the latest changes from main into their feature branch to incorporate the security fix before continuing their work.
- **Debugging:** If a bug is later discovered in profileView.php, a developer can use the file filter to see a simplified tree showing only the history of that specific file, making it much easier to identify which commit introduced the issue.

17. Testing & Evaluation:

The testing and evaluation phase of the **Farm Management System (AgroCulture)** was designed to ensure that the system met its functional and non-functional requirements, maintained configuration integrity, and achieved a high level of reliability before final submission. Testing was conducted using a mixture of black-box functional tests, configuration audits, and metrics analysis. Below is a detailed account of the testing activities performed.

17.1 Functional Testing:

- User Authentication: Both valid and invalid login scenarios were tested to confirm proper validation.

- Product Management: Farmers were able to upload, edit, and delete products with correct validation of mandatory fields such as price and image. Invalid entries (e.g., negative prices, missing images) were rejected by the system.

- Cart and Checkout: Buyers tested adding single and multiple items to the cart, proceeding to checkout, and placing orders. The system correctly validated stock availability and generated order confirmations.

- Review and Ratings: Buyers could leave reviews linked to products they purchased. Reviews displayed accurately on product pages.

- Blog Features: Farmers tested posting blogs, and Buyers interacted through likes and comments, confirming engagement functionalities worked as expected.

17.2 Configuration Audit:

- Baselines were reviewed to ensure that commits and tagged releases match documented release notes.
- All code changes were traceable to GitHub Issues, ensuring alignment with Scrum backlog items.
- Database migrations were checked by applying them on a fresh MySQL instance in CI/CD pipelines to prevent schema drift.
- CHANGELOG.md was reviewed to confirm it reflected the actual changes merged in each release.

17.3 Metrics and Evaluation:

- Defect Density: Approximately 0.6 defects per module were identified and resolved during the test cycle, indicating relatively stable performance.
- Effort Variance: Planned effort was 100 hours, while actual logged effort reached 110 hours, showing a +10% variance. This variance was within acceptable academic project limits and mainly due to additional configuration audits.
- Change Requests: A total of 6 change requests were logged. Out of these, 5 were approved and implemented, while 1 was rejected due to being out of the current project scope.

Metric	Formula	Planned / Actual Data	Result	Interpretation
Defect Density	Defects per Module= Total Defects / Modules	3 defects / 5 modules	0.6 defects/module	Low defect rate indicates stable and reliable system.
Effort Variance	(Actual Effort – Planned Effort) / Planned Effort × 100	Planned: 100 hrs | Actual: 110 hrs	+10% variance	Slight overrun due to extra audits; still within acceptable project limits.
Change Requests	Count and classification	Logged: 6 | Implemented: 5 | Rejected: 1	5 implemented, 1 rejected	High adaptability, minimal scope creep, effective change management.

Table: Process Metrics

Project Repository Link: