**Agriculture Store and Inventory Management System**

Software Requirements Specification

Version 1.0

20 June, 2024.

GROUP-1

Technical Lead

Prepared for

<PROJECT 2 COURSE>

**Table of Contents**

Revision History Error! Bookmark not defined.

Document Approval Error! Bookmark not defined.

1. Introduction Error! Bookmark not defined.

1.1 Purpose **Error! Bookmark not defined.**

1.2 Scope **Error! Bookmark not defined.**

1.3 Definitions, Acronyms, and Abbreviations **Error! Bookmark not defined.**

1.4 References **Error! Bookmark not defined.**

1.5 Overview **Error! Bookmark not defined.**

2. General Description Error! Bookmark not defined.

2.1 Product Perspective **Error! Bookmark not defined.**

2.2 Product Functions **Error! Bookmark not defined.**

2.3 User Characteristics **Error! Bookmark not defined.**

2.4 General Constraints **Error! Bookmark not defined.**

2.5 Assumptions and Dependencies **Error! Bookmark not defined.**

3. Specific Requirements Error! Bookmark not defined.

3.1 External Interface Requirements **Error! Bookmark not defined.**

3.1.1 User Interfaces **Error! Bookmark not defined.**

3.1.2 Hardware Interfaces **Error! Bookmark not defined.**

3.1.3 Software Interfaces **Error! Bookmark not defined.**

3.1.4 Communications Interfaces **Error! Bookmark not defined.**

3.2 Functional Requirements **Error! Bookmark not defined.**

3.2.1 <Functional Requirement or Feature #1> **Error! Bookmark not defined.**

3.2.2 <Functional Requirement or Feature #2> **Error! Bookmark not defined.**

3.3 Use Cases **Error! Bookmark not defined.**

3.3.1 Use Case #1 **Error! Bookmark not defined.**

3.3.2 Use Case #2 **Error! Bookmark not defined.**

3.4 Classes / Objects **Error! Bookmark not defined.**

3.4.1 <Class / Object #1> **Error! Bookmark not defined.**

3.4.2 <Class / Object #2> **Error! Bookmark not defined.**

3.5 Non-Functional Requirements **Error! Bookmark not defined.**

3.5.1 Performance **Error! Bookmark not defined.**

3.5.2 Reliability **Error! Bookmark not defined.**

3.5.3 Availability **Error! Bookmark not defined.**

3.5.4 Security **Error! Bookmark not defined.**

3.5.5 Maintainability **Error! Bookmark not defined.**

3.5.6 Portability **Error! Bookmark not defined.**

3.6 Inverse Requirements **Error! Bookmark not defined.**

3.7 Design Constraints **Error! Bookmark not defined.**

3.8 Logical Database Requirements **Error! Bookmark not defined.**

3.9 Other Requirements **Error! Bookmark not defined.**

4. Analysis Models Error! Bookmark not defined.

4.1 Sequence Diagrams **Error! Bookmark not defined.**

4.3 Data Flow Diagrams (DFD) **Error! Bookmark not defined.**

4.2 State-Transition Diagrams (STD) **Error! Bookmark not defined.**

5. Change Management Process Error! Bookmark not defined.

A. Appendices Error! Bookmark not defined.

A.1 Appendix 1 **Error! Bookmark not defined.**

A.2 Appendix 2 **Error! Bookmark not defined.**

**REVISION HISTORY**

| **Version** | **Date** | **Description of Change** | **Author** |
| --- | --- | --- | --- |
| 1.0 | 2024-07-18 | Initial creation | GROUP-1 |

**DOCUMENT APPROVAL**

| **Name** | **Role** | **Signature** | **Date** |
| --- | --- | --- | --- |
| GROUP-1 | Project Lead |  | 2024-07-18 |

TEAM MEMBERS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NO** | **NAME** | **ID** | **SET** | **POSITION** |
| **1** | **MOHAMED KAMARA** | **9020** | **2** | **Project Manager/Backend/Documentation** |
| **2.** | **AUGUSTINE TAMBA**  **ALLIEU** | **8322** | **5** | **Documentation** |
| **3** | **ELSIE DORIS BANGURA** | **7815** | **2** | **QA/Frontend** |
| **4** | **ISHA SESAY** | **7193** | **3** | **QA/Frontend** |
| **5** | **ABU JAMAL KAMARA** | **8463** | **4** | **Back-**  **End(server-side)** |
| **6** | **ABU SANKOH** | **8001** | **3** | **Back-End(Database)** |
| **7** | **MODIE JOHN GBLA** | **7965** | **1** | **Front-End** |
| **8** | **MOHAMED AD GBLA** | **7723** | **1** | **Front-End** |
| **9** | **SOLOMON AB KAMANDA** | **7925** | **4** | **UI/UX DESIGNER** |
| **10** | **TATEKULOR ALOYSIUS**  **KAMARA** | **7499** | **5** | **UI/UX Designer** |

**1. INTRODUCTION**

**1.1 PURPOSE**

The purpose of this document is to provide a comprehensive and detailed description of the Agriculture Store and Inventory Management System for Bombali District. This document outlines the system's objectives, scope, functionalities, user roles, constraints, and assumptions. It serves as a foundational reference for all stakeholders involved in the development, implementation, and maintenance of the system.

**1.2 SCOPE**

The Agriculture Store and Inventory Management System is a web-based application designed to streamline and manage the distribution of fertilizers to farmers in Bombali District. The system ensures an efficient, transparent, and secure process for managing fertilizer distribution and caters to multiple user roles including Admin, Director of Agriculture, Farmers, Farmer Registration Officer (FRO), and Store Keeper.

**1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS**

* **Admin**: System Administrator responsible for managing the system and user accounts.
* **Director of Agriculture**: Responsible for reviewing and approving fertilizer applications.
* **Farmer**: End-user applying for and collecting fertilizers.
* **FRO**: Farmer Registration Officer responsible for registering and updating farmer details.
* **Store Keeper**: Responsible for managing the inventory and issuing fertilizers.
* **MySQL**: Relational database management system used for storing system data.
* **React**: JavaScript library used for building the user interface of the system.
* **Django**: Python web framework used for developing the backend of the system.

## 1.4 REFERENCES

* **Agile Methodology Documentation**: Agile Alliance.
* **Django Official Documentation**: Django Software Foundation. (n.d.). Django Project Documentation. Retrieved from <https://docs.djangoproject.com/en/stable/>.
* **React Official Documentation**: Facebook Inc. (n.d.). ReactJS Documentation. Retrieved from https://reactjs.org/docs/getting-started.html.
* **MySQL Official Documentation**: Oracle Corporation. (n.d.). MySQL Documentation. Retrieved from <https://dev.mysql.com/doc/>.

### Similar Projects and Case Studies

* **Open Source Agriculture Management System**: AgriTech. (2021). AgriManager. Retrieved from <https://github.com/agritech/agrimanager>.
* **Inventory Management System in Django and React**: Mohamed Kamara, A. (2023). Inventory Management System. Retrieved from <https://github.com/programmerkamara/inventory-management-system>.
* **Farm Management System**: Smith, J. (2020). Build a Farm Management System. CodeProject. Retrieved from https://www.codeproject.com/Articles/1205470/Build-a-Farm-Management-System.
* **Django + React Example Projects**: Vinta Software. (2022). Django React Boilerplate. Retrieved from <https://github.com/vintasoftware/django-react-boilerplate>.
* **Agriculture Management System Case Study**: Johnson, R., & Brown, K. (2020). Analysis of Agricultural Management Systems. Journal of Agricultural Informatics, 12(3), 45-67. doi:10.1016/j.agrinf.2020.03.012. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0308521X20302153>

**1.5 OVERVIEW**

This document is organized into the following sections:

1. **Introduction**: Provides an overview of the document's purpose, scope, and structure.
2. **General Description**: Describes the system's perspective, functions, user characteristics, constraints, assumptions, and dependencies.
3. **Specific Requirements**: Details the requirements including external interfaces, functional requirements, use cases, classes/objects, non-functional requirements, inverse requirements, design constraints, logical database requirements, and other requirements.
4. **Analysis Models**: Includes sequence diagrams, data flow diagrams, and state-transition diagrams.
5. **Change Management Process**: Outlines procedures for managing changes to the system.
6. **Appendices**: Contains additional information and resources.

### **METHODOLOGY**

**Agile Methodology** was adopted for the development of the Agriculture Store and Inventory Management System. This approach was chosen for the following reasons:

1. **Iterative Development**: Agile supports iterative development, allowing for incremental improvements and refinements based on user feedback and evolving requirements. This ensures that the system can adapt to changes and new requirements as they arise.
2. **Continuous Feedback**: Regular feedback from stakeholders, including farmers, storekeepers, and administrators, is integrated throughout the development process. This helps in aligning the system with user needs and expectations.
3. **Flexibility and Adaptability**: The Agile methodology facilitates flexibility and adaptability, enabling the development team to respond quickly to changes in requirements or unforeseen challenges. This is particularly important for incorporating new features or modifications.
4. **Collaboration**: Agile emphasizes collaboration between cross-functional teams, including developers, designers, and stakeholders. This collaborative approach enhances communication and ensures that all aspects of the system are considered.
5. **Delivering Value**: Agile focuses on delivering functional and valuable increments of the system regularly. This ensures that essential features are available early and can be used by stakeholders while additional features are developed.
6. **Risk Management**: By working in short iterations and regularly reviewing progress, Agile helps in identifying and addressing risks early in the development process. This reduces the likelihood of major issues arising later in the project.

This methodology supports the dynamic nature of the Agriculture Store and Inventory Management System, enabling it to meet the needs of its users effectively while accommodating future enhancements.

# 2. GENERAL DESCRIPTION

## 2.1 PRODUCT PERSPECTIVE

The Agriculture Store and Inventory Management System is a stand-alone, web-based application designed to facilitate the distribution of fertilizers to farmers in Bombali District. The system operates independently, interfacing with users through a web browser, and is supported by a backend server running Django and a MySQL database.

## 2.2 PRODUCT FUNCTIONS

The system provides distinct functionalities tailored to different user roles:

* **Admin**: Manages user accounts, performs system maintenance, manages access control, monitors audit logs, and configures system settings.
* **Director of Agriculture**: Reviews and approves/rejects fertilizer applications, assigns fertilizer quantities, generates unique codes, and notifies farmers.
* **Farmers**: Registers at the office, applies for fertilizers, checks application status, and presents unique codes to collect fertilizers.
* **Farmer Registration Officer (FRO)**: Registers farmers, collects necessary information, and verifies/upgrades farmer details.
* **Store Keeper**: Validates unique codes, manages fertilizer inventory, and issues fertilizers based on approved applications.

## 2.3 USER CHARACTERISTICS

* **Admin**: Technical users with responsibilities in system management and maintenance.
* **Director of Agriculture**: Users with authority to approve fertilizer applications.
* **Farmers**: End-users applying for and collecting fertilizers.
* **Farmer Registration Officer (FRO)**: Users responsible for farmer registration and updates.
* **Store Keeper**: Users managing the inventory and issuing fertilizers.

## 2.4 GENERAL CONSTRAINTS

* System accessibility is limited to web browsers.
* The system does not support offline functionality.
* Limited to the Bombali District.

## 2.5 ASSUMPTIONS AND DEPENDENCIES

* Users have internet access and web browsers to interact with the system.
* The system relies on stable backend server hosting and database management.

# 3. SPECIFIC REQUIREMENTS

## 3.1 EXTERNAL INTERFACE REQUIREMENTS

**3.1.1 User Interfaces**



* **Web-Based Interface**: The system will feature a web-based user interface accessible via modern web browsers, including Chrome, Firefox, Safari, and Edge.
* **Responsive Design**: The user interface will be designed with a responsive layout to ensure optimal usability across a variety of devices and screen sizes, including desktops, laptops, tablets, and smartphones.
* **Accessibility**: The interface will adhere to accessibility standards (e.g., WCAG 2.1) to ensure that it is usable by individuals with disabilities. This includes support for screen readers and keyboard navigation.

**3.1.2 Hardware Interfaces**

* **General Requirements**: There are no specific hardware interface requirements beyond standard computing devices capable of running a modern web browser and having internet access.
* **Minimum Specifications**: For optimal performance, the system is recommended to be accessed on devices meeting the following minimum specifications:
  + **Processor**: Intel Core i3 or equivalent
  + **Memory**: 4 GB RAM
  + **Storage**: 1 GB of available storage
  + **Network**: Stable internet connection with a minimum speed of 1 Mbps

**3.1.3 Software Interfaces**

* **Database Integration**: The system will integrate with MySQL for database management, leveraging its capabilities for storing and querying data efficiently.
* **Backend API**: The backend operations will be managed through API endpoints provided by the Django framework. This includes RESTful API endpoints for CRUD (Create, Read, Update, Delete) operations and other business logic.
* **Authentication/Authorization**: Integration with Django’s built-in authentication and authorization features to manage user access and permissions.

**3.1.4 Communications Interfaces**

* **Protocols**: The system will use HTTP/HTTPS protocols for communication between the client and server. HTTPS will be enforced to ensure secure data transmission and protect user information.
* **Data Formats**: Data exchanged between the client and server will be formatted in JSON for API requests and responses to facilitate easy data handling and interoperability.
* **Error Handling**: Proper error handling mechanisms will be implemented to provide meaningful error messages and status codes, improving the user experience and debugging process.

## 3.2 FUNCTIONAL REQUIREMENTS

### 3.2.1 Admin

* Create, edit, or delete user accounts for various roles.
* Perform system maintenance tasks such as backups, updates, and troubleshooting.
* Manage role-based access control to ensure appropriate permissions.
* Monitor and manage audit logs for accountability and security.
* Configure system settings and preferences as required.

### 3.2.2 Director of Agriculture

* Review and approve/reject fertilizer applications submitted by farmers.
* Assign the quantity of fertilizer bags for approved applications.
* Generate unique codes for each approved application.
* Notify farmers of their application status and unique code via SMS or email.

### 3.2.3 Farmers

* Register at the office to obtain a valid login for applying for fertilizers.
* Apply for fertilizers specifying the type and quantity required.
* Check the status of their fertilizer application to know if it is approved or rejected.
* Present the unique code received upon approval at the store to collect fertilizers.

### 3.2.4 Farmer Registration Officer (FRO)

* Register farmers at the office, collecting necessary information such as name, contact details, and farm details.
* Verify and update farmer details as needed to ensure accuracy in the system.

### 3.2.5 Store Keeper

* Validate the unique code presented by the farmer to ensure authenticity.
* View available inventory of fertilizers and update stock levels after issuing fertilizers.
* Issue fertilizers to farmers based on approved applications and validated codes.

## 3.3 USE CASES

### 3.3.1 Use Case #1: Farmer Registration

**Actors**: Farmer Registration Officer (FRO)  
**Description**: The FRO registers farmers by collecting their information and updating the system.

### 3.3.2 Use Case #2: Fertilizer Application Approval

**Actors**: Director of Agriculture  
**Description**: The Director reviews and approves or rejects fertilizer applications and assigns quantities.

## 3.4 CLASSES / OBJECTS

### 3.4.1 Class: User

**Attributes**: username, password, role  
**Methods**: create(), edit(), delete()

### 3.4.2 Class: FertilizerApplication

**Attributes**: farmer, type, quantity, status, unique\_code  
**Methods**: submit(), approve(), reject()

### 3.5 NON-FUNCTIONAL REQUIREMENTS

**3.5.1 Performance**

* **Response Time**: The system must deliver responses to user actions (e.g., form submissions, data retrieval) within 2 seconds under normal load conditions.
* **Scalability**: The system should be scalable to handle up to 10,000 concurrent users without significant degradation in performance. This includes efficient handling of database queries and API requests.
* **Throughput**: The system should support high-throughput operations, allowing for up to 1000 transactions per minute during peak usage.

**3.5.2 Reliability**

* **Uptime**: The system must achieve 99.9% uptime to ensure continuous availability and reliability.
* **Error Handling**: It should gracefully handle and log errors without crashing. Users should be presented with user-friendly error messages and recovery options.
* **Data Integrity**: The system must ensure data integrity through validation checks and transaction management to prevent data corruption or loss.

**3.5.3 Availability**

* **System Availability**: The system must be available 24/7, with minimal scheduled downtime for maintenance, ideally less than 1 hour per month.
* **Disaster Recovery**: The system should include disaster recovery plans to restore functionality within 4 hours in the event of a major failure.
* **Redundancy**: Critical components should be redundant to minimize the impact of hardware or software failures on system availability.

**3.5.4 Security**

* **Authentication**: Implement strong user authentication mechanisms, including multi-factor authentication (MFA) for sensitive operations.
* **Authorization**: Ensure role-based access control (RBAC) to restrict access to sensitive data and functionalities based on user roles.
* **Data Encryption**: All sensitive data, including user information and communication between the client and server, must be encrypted using industry-standard encryption protocols (e.g., TLS/SSL).
* **Vulnerability Management**: Regularly update and patch system components to address known security vulnerabilities and conduct periodic security audits.

**3.5.5 Maintainability**

* **Code Quality**: Adhere to coding standards and best practices to ensure high code quality and ease of maintenance. Implement comprehensive code documentation.
* **Modular Design**: Design the system in a modular fashion to facilitate easy updates and modifications. Each module should have a clear and defined interface.
* **Testing**: Implement automated testing for critical functionalities and maintain a suite of test cases to verify system changes and new features.
* **Logging**: Maintain detailed logs for system activities and errors to assist in troubleshooting and performance monitoring.

**3.5.6 Portability**

* **Cross-Platform Support**: The system should be accessible from various operating systems (e.g., Windows, macOS, Linux) and devices (e.g., desktops, tablets) through a web browser.
* **Compatibility**: Ensure compatibility with major web browsers (e.g., Chrome, Firefox, Safari, Edge) and mobile devices to provide a consistent user experience.
* **Deployment Flexibility**: The system should support deployment on various environments, including on-premises servers, cloud platforms (e.g., AWS, Azure), and hybrid configurations.

## 3.6 INVERSE REQUIREMENTS

* The system will not support multiple languages.
* The system will not include a feedback mechanism or chatting system.
* The system will not generate PDF/Doc reports.
* The system will not have offline functionality.
* The system will be accessible only via web browsers and is not designed as a mobile application.

## 3.7 DESIGN CONSTRAINTS

* **Frontend Technology**: The frontend of the system must be developed using React.js to leverage its component-based architecture and efficient state management capabilities. This ensures a dynamic and responsive user interface.
* **Backend Technology**: The backend must be developed using Django, a high-level Python web framework that encourages rapid development and clean, pragmatic design. Django's robust ORM and built-in admin interface are crucial for managing data and user roles efficiently.
* **Database Management**: MySQL is the mandated database management system for this project. MySQL's reliability, scalability, and support for complex queries are essential for handling the system's data requirements effectively.
* **Integration Constraints**: The system must seamlessly integrate the React frontend with the Django backend via RESTful APIs or GraphQL to ensure smooth data exchange and functionality.

**3.8 LOGICAL DATABASE REQUIREMENTS**

* **Database Management System**: The system will utilize MySQL as the database management system. MySQL provides the reliability, scalability, and efficiency required for managing the relational data associated with the Agriculture Store and Inventory Management System.
* **Database Schema Design**:
  + **Users Table**: Managed by the CustomUser model, which extends AbstractUser to include additional user types. This table includes fields for username, password, and user type (Admin, Director of Agriculture, Farmer, Farmer Registration Officer, Store Keeper).
  + **Admin Table**: Stores additional details specific to Admin users, including phone number, gender, and address.
  + **DirectorOfAgriculture Table**: Contains additional attributes for Director of Agriculture users such as phone number, gender, and address.
  + **Farmer Table**: Holds farmer-specific information including type (Individual, CBO, NGO), phone number, address, contact details, and farm details.
  + **FarmerRegistrationOfficer Table**: Maintains records for Farmer Registration Officers with their phone number, gender, and address.
  + **StoreKeeper Table**: Records details about Store Keepers, including phone number, gender, and address.
  + **Fertilizer Table**: Manages information about different types of fertilizers, including type, quantity, and the store keeper responsible for adding the fertilizer.
  + **FertilizerApplication Table**: Tracks applications made by farmers for fertilizers, including the type of fertilizer requested, quantity needed, and application status.
  + **AssignedFertilizer Table**: Contains records of fertilizers assigned to farmers, including the unique code for verification and the quantity assigned.
  + **FertilizerDistribution Table**: Logs the distribution of fertilizers to farmers, including the type, quantity distributed, and the store keeper who distributed it.
* **Relationships and Constraints**:
  + **Foreign Keys**: Various tables use foreign keys to establish relationships, such as linking the Farmer to FertilizerApplication and AssignedFertilizer to Farmer. These relationships ensure referential integrity and support complex queries.
  + **Unique Constraints**: The unique\_code field in the AssignedFertilizer and FertilizerDistribution tables is enforced to prevent duplication and ensure code validity.
  + **Validation**: Methods such as validate\_code in the AssignedFertilizer model are used to validate codes and ensure that only legitimate codes are processed.

**3.7 DESIGN CONSTRAINTS**

* **Frontend Technology**: The frontend of the Agriculture Store and Inventory Management System will be developed using React.js. React’s component-based architecture allows for the creation of a dynamic and responsive user interface, facilitating efficient state management and user interactions.
* **Backend Technology**: The backend will be developed using Django, a high-level Python web framework. Django’s features, such as its Object-Relational Mapping (ORM), built-in admin interface, and security features, are essential for managing data and handling business logic.
* **Database Technology**: MySQL will be used as the database management system. MySQL’s support for complex queries, transactions, and its ability to handle large datasets make it suitable for managing the relational data of the system.
* **Integration Requirements**: The system must integrate React with Django via RESTful APIs or GraphQL. This integration will ensure smooth data exchange between the frontend and backend, enabling dynamic content updates and user interactions.
* **Security Constraints**: The system must incorporate robust security measures, including secure authentication and authorization mechanisms, to protect sensitive data and ensure compliance with data protection regulations.
* **Scalability and Performance**: The system design should consider scalability to accommodate potential future enhancements and increased user loads. Performance optimization techniques, such as indexing and query optimization, should be employed to ensure efficient data retrieval and processing.

# 4. ANALYSIS MODELS

# 4.1 SEQUENCE DIAGRAMS

# 

# 

## 4.2 DATA FLOW DIAGRAMS (DFD)

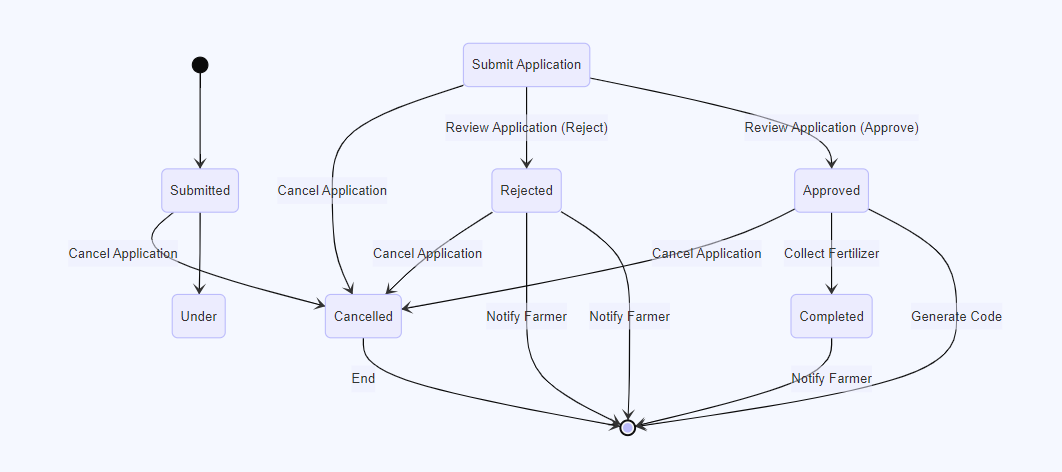
## 

## 

## 4.3 STATE-TRANSITION DIAGRAMS (STD)

## States and Transitions

1. **States**:
   * **Submitted**: The application has been submitted by the farmer but not yet reviewed.
   * **Under Review**: The application is currently being reviewed by the Director of Agriculture.
   * **Approved**: The application has been approved and a unique code is generated.
   * **Rejected**: The application has been rejected by the Director of Agriculture.
   * **Completed**: The farmer has collected the fertilizer from the store.
   * **Cancelled**: The application has been cancelled or invalidated.
2. **Transitions**:
   * **Submit Application**: Moves the application from Submitted to Under Review.
   * **Review Application**: Director of Agriculture reviews the application, transitioning it to either Approved or Rejected.
   * **Generate Code**: If approved, a unique code is generated, and the application moves to Approved.
   * **Notify Farmer**: The farmer is notified of the approval or rejection.
   * **Collect Fertilizer**: If the farmer presents the unique code at the store, the application moves to Completed.
   * **Cancel Application**: The application is moved to Cancelled if the farmer or system decides to cancel it.



**5. CHANGE MANAGEMENT PROCESS**

To ensure effective management of changes within the Agriculture Store and Inventory Management System, the following procedures will be adhered to:

* **Change Proposal**:
  + **Submission**: Changes are proposed by stakeholders or team members through a formal request. Proposals should include a detailed description of the change, the reason for the change, and the expected impact on the system.
  + **Review**: Proposals are reviewed by a change management board or designated reviewers to assess feasibility, risks, and benefits.
* **Change Approval**:
  + **Assessment**: The review board evaluates the proposed change against project goals, resource availability, and potential impacts on system functionality.
  + **Decision**: Based on the assessment, the change is approved, deferred, or rejected. An approval involves documenting the rationale and expected outcomes.
* **Change Implementation**:
  + **Planning**: A detailed plan for implementing the change is developed, including tasks, timelines, and responsible parties.
  + **Execution**: The change is implemented according to the plan. This may involve code updates, configuration changes, or other modifications.
* **Change Monitoring and Evaluation**:
  + **Testing**: Changes are tested to ensure they meet requirements and do not adversely affect existing functionality.
  + **Monitoring**: Post-implementation, the system is monitored to detect any issues or unintended effects resulting from the change.
  + **Documentation**: All changes are documented in a change log, including descriptions, impacts, implementation details, and approval information.
* **Change Log**:
  + **Record Keeping**: A log of all changes is maintained, capturing details such as the change description, date of implementation, impact on the system, and approval status.
  + **Access**: The change log is accessible to relevant stakeholders for review and audit purposes.

# 6. APPENDICES

## 6.1 APPENDIX A: GLOSSARY

* **Admin**: System Administrator.
* **Director of Agriculture**: Responsible for approving applications.
* **Farmer**: End-user applying for fertilizers.
* **FRO**: Responsible for registering farmers.
* **Store Keeper**: Manages inventory and issues fertilizers.