Software Requirements Specification

For

AgriSmart

Version 1.0

Prepared by Victor Wanyungu

Masinde Muliro University of Science and Technology

September 30 2024

Revision History

Name	**Date**	**Reason for changes**	**Version**
		Initial version	1.0
Software processing model	10/11/2024	Adding software processing model appendix	1.1

1. Introduction

1.1 Purpose

This SRS outlines the software requirements for the development of a mobile application that facilitates direct transactions between farmers and customers, with delivery personnel acting as intermediaries. The app will enable farmers to list their produce, customers to place orders, and delivery personnel to manage deliveries. This document details all functional and nonfunctional requirements to guide the development of the whole system..

1.2 Document Conventions

- **REQ-X** is used to indicate unique functional requirements.
- High priority requirements are in bold.

1.3 Intended Audience and Reading Suggestions

The intended audience includes:

- **Developers**

  Focus on sections detailing functional and system requirements. The ideal reading sequence for a developer is chapter 3, 4, 5, 6, 1 and 2.

- **Project Managers**

  Focus on scope, performance, and schedule. The ideal reading sequence for a project manager is chapter 1, 2, 4, 5, 6, and 3.

- **Testers**

  Review all requirements for clarity and completeness. The ideal reading sequence for a tester is chapter 1, 2, 4, 5, 6, and 3.

1.4 Product Scope

AgriSmart is an application built to connect farmers directly to their customers to combat exploitation by middlemen therefore more profits for the farmer.

1.4.1 Benefits

1. Offering a digital marketplace for farmers
2. Access to fresh produce by the customers
3. Flexible job opportunities for delivery personnel

1.4.2 Objectives

1. Provide wider market for farmers
2. Contribute to providing more job opportunities for the youth

1.4.2 Goals

1. Promote healthy eating habits through access to fresh produce
2. Promoting local business and agriculture

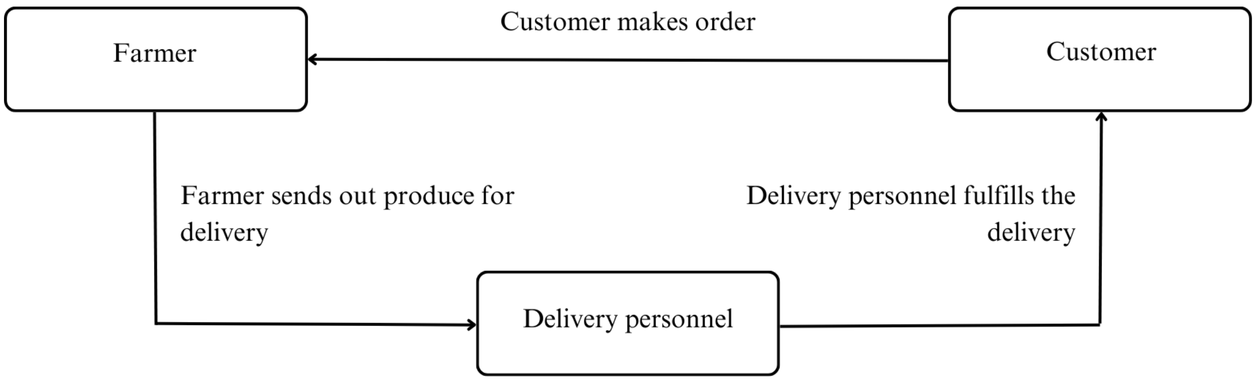
1.5 References

- IEEE Standard for Software Requirements Specifications
- [Insert relevant articles or references related to agricultural platforms]

2. Overall Description

2.1 Product Perspective

This app is a new product that integrates e-commerce with logistics, catering specifically to the agricultural market. It provides a bridge between farmers and end customers, with delivery personnel completing the link.


Farmer
Customer makes order
Customer
Farmer sends out produce for delivery
Delivery personnel fulfills the delivery
Delivery personnel

2.2 Product Functions

The app will allow users to:

- **Farmers**
    - List products
    - track sales
    - manage inventory.
- **Customers**
    - Browse products
    - order products
    - track delivery
- **Delivery Personnel**
    - Accept delivery job
    - manage delivery jobs.

2.3 User Classes and Characteristics

- **Farmers**

  May not be tech-savvy. Require simple product management tools.

- **Customers**

  E-commerce savvy. Require an intuitive and easy to use interface for browsing and purchasing products.

- **Delivery Personnel**

  Require real-time navigation and job tracking. They are the most frequent users of the application.

2.4 Operating Environment

The app will run on Android and iOS smartphones, requiring internet connectivity for real-time updates and order functionality.

2.5 Design and Implementation Constraints

- **Payment Gateway Integration**

  The app must comply with regional financial regulations.

- **Mobile Platform Compatibility**

  The app must work on Android (version 7.0 and above) and iOS (version 12.0 and above).

- **Language limitations**

  The app must support both English and Kiswahili

- **Storage**

  The app must not exceed 100 MB

2.6 User Documentation

The app will include:

- **User support**

  Users will be provided with social media links, an email and phone number to contact user support staff for help and inquiries.

- **Tutorials**

The app will come with tutorials in the Help section.

2.7 Assumptions and Dependencies

2.7.1 Assumptions

- The app assumes that farmers have access to smartphones and internet.

2.7.2 Dependencies

- It relies on third-party APIs for payment processing and GPS navigation.

3. External Interface Requirements

3.1 User Interfaces

The app will feature distinct user interfaces for:

- **Farmers**

  Dashboard to manage products and view orders.

- **Customers**

  Product browsing, cart, and order history.

- **Delivery Personnel**

  Job listings and real-time navigation.

3.2 Hardware Interfaces

3.2.1. Supported Device Types

The app will be designed to run on smartphones and tablets with the following specifications:

- **Android Devices**:
  - Version: Android 7.0 (Nougat) and above.
  - Devices: Mobile phones, tablets, and phablets.
- **iOS Devices**:
  - Version: iOS 12.0 and above.
  - Devices: iPhones and iPads.

3.2.2 Nature of data and control interactions

The app interacts with several hardware components on mobile devices to provide the following functionalities:

- **GPS Module**:
  The app communicates with the device's GPS hardware to provide real-time location tracking. This allows the delivery personnel to navigate routes, and customers to track their orders. The GPS data is collected continuously and sent to the server via HTTP/HTTPS protocols.
- **Camera**:
  The camera will be used by delivery personnel to capture proof of delivery by taking photographs of the delivered goods or obtaining customer signatures. The image data is uploaded to the server and associated with the order for verification purposes.
- **Touchscreen**:
  Users interact with the app through the mobile device's touchscreen. All inputs, such as tapping buttons, scrolling through lists, and entering text, are processed and sent to the app's backend for further actions (e.g. placing an order, accepting a delivery).

3.2.3 Communication protocols

- **HTTP/HTTPS**:
  All data communication between the app and the server will occur over HTTPS to ensure encrypted and secure transmission of sensitive information, such as payment details and personal information.

3.2.4 Control Interactions

- **Real-Time Notifications**:
  The app will use push notifications via Firebase Cloud Messaging (for Android) and Apple Push Notification Service (for iOS) to alert users about new orders, order updates, and delivery statuses.
- **Battery Usage**:
  The app will be optimized to minimize battery drain, even with constant GPS tracking and push notifications. However, the GPS and data usage will require efficient battery management strategies to avoid excessive power consumption on user devices.

3.3 Software Interfaces

3.3.1 Connections to Software Components

The app connects to several software components to facilitate its functionality. These include:

- **Operating Systems**:
  - **Android**: Version 7.0 (Nougat) and above.
  - **iOS**: Version 12.0 and above.

- **Database**:
  - **PostgreSQL** (version 13.0 and above): This is the primary relational database used for storing user data, product listings, orders, and delivery information. PostgreSQL offers support for large-scale applications and complex queries required for the app's operation.
- **Backend Framework**:
  - **Node.js** (version 16.0 and above): The backend of the system is built using Node.js, which processes incoming requests from the mobile app and performs the required operations on the database. It also communicates with external services such as payment gateways and GPS APIs.
- **Cloud Infrastructure**:
  - **AWS** (Amazon Web Services): The app will be deployed on AWS for reliable, scalable cloud services. Services like **S3** will be used for storing images (e.g., proof of delivery), and **RDS** (Relational Database Service) will host the PostgreSQL database.
- **Payment Gateway**:
  - The app will integrate with Mpesa, enabling customers to pay for their orders, and ensuring farmers receive their funds promptly.
- **GPS Services**:
  - **Google Maps API**: The app integrates with Google Maps API for real-time tracking of deliveries and location-based services for delivery personnel and customers.
- **Push Notifications**:
  - **Firebase Cloud Messaging**: To send push notifications to users on both Android and iOS regarding order updates, delivery statuses, and other relevant alerts.

3.3.2 Data items and messages

- **Incoming Data**:
  - **User Data**: When a user registers or logs in, the system receives their personal details, such as name, email, and phone number, and stores it in the PostgreSQL database.
  - **Order Data**: Customers submit product orders, including product details, quantity, delivery address, and payment information.
  - **Delivery Data**: Delivery personnel update the order status during different stages of the delivery process (e.g., picked up, in transit, delivered), which is sent back to the backend.
- **Outgoing Data**:
  - **Order Confirmation**: After successful order placement, confirmation details are sent to both the customer and the farmer.
  - **Payment Confirmation**: Upon successful payment processing, the system sends a receipt to the customer and a notification to the farmer regarding the confirmed sale.
  - **Delivery Status Updates**: Real-time delivery tracking updates are sent to the customer via push notifications, showing the current status of their order.

3.3.3 Services and communication protocols

- **Authentication Service**:
  OAuth 2.0 is used for user authentication, ensuring secure access and permission control for users logging in through social accounts (Google, Facebook) or standard email/password methods. This integrates directly with the Node.js backend and database.
- **RESTful API**:
  The system uses a REST API to handle communication between the mobile app (client-side) and the backend (server-side). The API handles requests such as order placement, product listing, and delivery status updates. The API communicates via **HTTPS** to ensure data security.
- **Payment Service**:
  Communication with the payment gateway happens through their respective APIs using secure HTTPS communication. The system sends payment details to the payment processor and receives transaction confirmations, which are then stored in the database.
- **GPS Service**:
  Google Maps API is used for location services. The app sends the delivery address and real-time GPS coordinates to the API and receives route suggestions, estimated delivery times, and other map-related data.

3.3.4 Shared data across components

- **User Data**:
  User credentials and profiles (stored in PostgreSQL) are shared across the authentication service (OAuth 2.0) and the user management features in the mobile app.
- **Order Data**:
  Order details are shared between the customer interface, farmer interface, and delivery personnel interface. The system ensures real-time updates are synced across all parties involved, maintaining consistency between the app's front end and the backend.
- **Delivery Tracking Data**:
  GPS data is continuously shared between the mobile app and the Google Maps API to ensure customers and farmers can track the progress of their deliveries in real time.

3.3.5 Implementation constraints

- **Database Synchronization**:
  The system uses PostgreSQL as a central repository for all data. A database connection pool is implemented in Node.js to ensure efficient and scalable access to the database, especially during peak usage.
- **API Rate Limits**:
  Integration with external APIs (Google Maps, Stripe/PayPal) will adhere to the API rate limits, ensuring that the system handles requests without hitting usage caps, especially for critical operations like payment processing and GPS tracking.

3.4 Communications Interfaces

3.4.1 Communication Functions

- **Push Notifications**:
  Push notifications are critical for sending real-time updates to users about order status, delivery tracking, and promotions. These notifications will be sent via Firebase Cloud Messaging (FCM)
- **Email Notifications**:
  The app will also send important updates via email, including:
  - Order confirmations.
  - Payment receipts.
  - Promotional offers or notifications.

  Emails will be formatted in HTML and sent via an external email service provider for scalability and reliability.

3.4.2 Network server communication protocols

- **HTTPS (Hypertext Transfer Protocol Secure)**:
  All communication between the app and the backend server will use HTTPS to ensure encryption and security of transmitted data, especially sensitive information like personal details, payment data, and delivery addresses.
- **WebSocket Protocol**:
  Real-time features such as live delivery tracking and order updates may require the use of WebSocket connections to establish persistent communication channels for instant notifications and real-time data sync.

3.4.3 Message formatting

- **JSON (JavaScript Object Notation)**:
  All data exchanged between the app and backend will be formatted in **JSON** for its lightweight structure and ease of use with web services. This includes data such as:
  - User details
  - Product information
  - Order information
  - Delivery tracking details.
- **HTML**:
  HTML will be used for rendering the content of email notifications sent to users.

3.4.4 Communication security and encryption

- **Data Encryption**:
  All sensitive data transmitted between the client (mobile/web) and the server will be encrypted using **TLS (Transport Layer Security)**, specifically TLS 1.2 or higher, to protect data in transit. This includes:
  - User authentication details.
  - Payment information.
  - Delivery addresses.

- **Two-Factor Authentication (2FA)**:
  The app will implement 2FA for added security, especially for delivery personnel and farmers accessing their accounts, to prevent unauthorized access.
- **OAuth 2.0**:
  For user authentication, the app will use OAuth 2.0, which provides a secure way to authenticate users and manage session tokens.

3.4.5 Data transfer rates and synchronization mechanisms

- **Data Transfer Rates**:
  The app will handle typical e-commerce data, such as user profile information, product listings, and order updates. As such, data transfer rates will be optimized to ensure quick performance without overwhelming user devices or the backend infrastructure. Expected data transfer rates:
    - Typical REST API request/response: 100–200 KB per interaction.
    - File upload/download (for images, proof of delivery): Up to 5 MB per image, with compression applied as needed.
- **Synchronization Mechanism**:
  The app will use WebSockets to ensure that all parties (farmers, customers, and delivery personnel) receive real-time updates on order statuses, delivery tracking, and product availability. Data will be updated automatically without requiring user intervention.
- **Offline Mode**:
  To handle poor network conditions, the app will include an offline data synchronization feature. When a connection is re-established, all offline actions will be synced to the backend.

4. System Features

4.1 Product Listing and Management for Farmers

4.1.1 Description and Priority

This feature allows farmers to list, update, and manage their products. Farmers can add details such as price, quantity, product type, and availability. This is a high priority feature as it forms the core of the application’s functionality. Without this feature, farmers would not be able to list their products for sale.

Priority component ratings (1-9):

- Benefit: 9
- Penalty: 8
- Cost: 5
- Risk: 3

4.1.2 Stimulus/Response Sequences

- **Stimulus**: The farmer logs in and navigates to the “Manage Products” section.
  Response: The system displays all previously listed products and an option to add new products.

- **Stimulus**: The farmer clicks "Add Product" and enters the product name, price, quantity, and other details.
  Response: The system validates the input data and saves the product to the database, making it available for customers to view.
- **Stimulus**: The farmer updates or deletes an existing product.
  Response: The system updates or removes the product from the listings. If the farmer deletes a product, it is no longer available for purchase.

4.1.3 Functional Requirements

- **REQ-1**: The system must allow farmers to add, update, or delete products with fields for name, description, price, quantity, and availability.
- **REQ-2**: The system must validate product inputs before allowing a product to be listed. Validation includes checking for missing fields or invalid data types.
- **REQ-3**: Farmers must be able to view a list of their active products and manage (edit or delete) them from the same interface.
- **REQ-4**: The system must notify farmers when inventory levels of a product fall below a predefined threshold (e.g., 10 units remaining).
- **REQ-5**: The system must allow farmers to add images to product listings, ensuring image size does not exceed 5 MB and the file format is JPG or PNG.
- **REQ-6**: When a product is added or updated, the system should immediately reflect the changes in the customer-facing product list.
- **REQ-7**: The system must prevent farmers from adding duplicate products (same name and attributes).
- **REQ-8**: Error conditions, such as missing required fields or invalid price formats, must trigger an error message that explains the issue and suggests corrective action.

4.2 Order Placement for Customers

4.2.1 Description and Priority

This feature allows customers to browse products, add items to their cart, and place orders. It includes filtering options and order summary features. This is a high priority feature as it facilitates the purchase process, the main goal of the application.

Priority component ratings (1-9):

- Benefit: 9
- Penalty: 9
- Cost: 6
- Risk: 4

4.2.2 Stimulus/Response Sequences

- **Stimulus**: The customer browses the list of available products.
  Response: The system displays all products, filtered by location or category as specified by the customer.
- **Stimulus**: The customer selects products and clicks "Add to Cart."
  Response: The system adds the selected products to the cart and calculates the total cost.
- **Stimulus**: The customer clicks "Checkout" and enters payment and delivery details.
  Response: The system processes the payment, confirms the order, and provides the customer with an estimated delivery time.

4.2.3 Functional Requirements

- **REQ-9**: The system must allow customers to browse products by category, location, and availability.
- **REQ-10**: Customers must be able to add products to a shopping cart, with the option to review the cart before placing an order.
- **REQ-11**: The system must automatically calculate the total cost, including taxes and delivery fees, and display it to the customer during checkout.
- **REQ-12**: The system must offer multiple payment options (e.g. credit card, mobile money).
- **REQ-13**: Customers must receive an order confirmation upon successful payment, and the order must be saved in the database for tracking.
- **REQ-14**: If there are issues with payment processing, the system must provide an error message and allow the customer to try again.
- **REQ-15**: The system must allow customers to cancel an order before it is dispatched for delivery.
- **REQ-16**: Orders must be assigned a unique order ID for tracking purposes, and both the customer and farmer should be able to view the status of the order.

4.3 Delivery Management for Delivery Personnel

4.3.1 Description and Priority

This feature allows delivery personnel to accept, manage, and track deliveries. It includes route navigation, delivery confirmation, and proof of delivery functionalities. This is a high priority feature as it ensures orders are fulfilled and delivered to customers.

Priority component ratings (1-9):

- Benefit: 8
- Penalty: 7
- Cost: 5
- Risk: 3

4.3.2 Stimulus/Response Sequences

- **Stimulus**: The delivery person logs in and sees a list of available deliveries.
  Response: The system displays the delivery orders available for acceptance, including pickup and drop-off locations.

- **Stimulus**: The delivery person accepts a delivery job.
  Response: The system assigns the delivery job and provides navigation instructions to the pickup location.
- **Stimulus**: The delivery person completes the delivery and uploads a photo or signature for proof of delivery.
  Response: The system marks the delivery as complete and notifies the customer and farmer.

4.3.3 Functional Requirements

- **REQ-17**: The system must display available delivery jobs to delivery personnel, showing relevant details like pickup and drop-off locations.
- **REQ-18**: Delivery personnel must be able to accept or decline a delivery job within the app.
- **REQ-19**: The system must provide real-time navigation and route information using the GPS module on the delivery personnel's device.
- **REQ-20**: Delivery personnel must upload proof of delivery (photo) through the app, which is stored on the server for future reference.
- **REQ-21**: The system must notify the customer and farmer when the order has been successfully delivered.
- **REQ-22**: If a delivery is delayed beyond the estimated time, the system should notify the customer and provide updated delivery details.
- **REQ-23**: Delivery personnel should have access to contact information for both the farmer (for pickup) and the customer (for drop-off) during the delivery process.
- **REQ-24**: In case of a failed delivery, the system must record the reason for failure and allow the delivery personnel to reschedule or return the order to the farmer.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Response Time

The app should respond to user inputs (e.g. button clicks, page loads) within 2 seconds for 95% of actions.

Rationale: Fast response times are crucial for maintaining a smooth user experience, especially for customers browsing products and placing orders.

5.1.2 Order Processing Time

Orders, including payment processing and confirmation, must be completed within 30 seconds after submission by the customer.

Rationale: Quick order processing is necessary to ensure customer satisfaction and to prevent users from abandoning the process due to delays. This also applies to real-time updates being sent to farmers and delivery personnel.

5.1.3 Scalability

The system must support at least 5,000 concurrent users without a noticeable drop in performance, ensuring that new requests are handled promptly.

Rationale: The app is expected to grow in popularity, and it must be capable of scaling to handle large volumes of traffic during peak periods without affecting response times or system stability.

5.1.4 Data Sync and Update Times

All changes to product listings (e.g. updates to inventory or pricing) must be reflected in the customer-facing view within 5 seconds.

Rationale: Accurate and timely updates to product data ensure that customers are always interacting with the most current information, avoiding potential stock issues or discrepancies in pricing.

5.1.5 Real-Time GPS Tracking

GPS tracking updates for delivery personnel must occur every 5–10 seconds, and this information must be available to both the customer and the farmer in real-time.

Rationale: Real-time GPS tracking is crucial for transparency and customer trust, as it allows both customers and farmers to follow the delivery progress closely.

5.1.6 Push Notification Latency

Push notifications (e.g. order confirmations, delivery status updates) must be delivered within 10 seconds of the event occurring.

Rationale: Timely notifications are essential to keep users informed about important events like order updates and delivery progress, ensuring a seamless experience.

5.1.7 Database Query Performance

Database queries, such as retrieving product listings or searching orders, must be executed within 1 second for 90% of queries.

Rationale: Efficient database performance is critical for real-time access to information, especially as users frequently browse products, check order statuses, and manage deliveries.

5.1.8 System Uptime and Availability

The system should maintain 99.9% uptime, with no more than 9 hours of downtime per year (excluding planned maintenance).

Rationale: High availability is critical for an app that manages real-time transactions between farmers, customers, and delivery personnel. Any downtime can lead to lost sales and a poor user experience.

5.1.9 Image Upload and Download Times

Images (e.g. product photos or proof of delivery) must upload within 5 seconds for standard network speeds (4G or higher).

Rationale: Quick image uploads ensure that farmers can list their products promptly and that delivery personnel can upload proof of delivery efficiently, reducing operational delays.

5.1.10 Error Recovery Time

In case of a system error (e.g. payment failure, connectivity issues), the system must provide recovery options or error messages within 3 seconds and allow the user to retry the action without losing progress.

Rationale: Fast error handling prevents user frustration and ensures that critical actions, such as payment processing, can be retried smoothly.

5.2 Safety Requirements

5.2.1 Data Loss Prevention

The app must implement automated backups of all critical data, including user profiles, product listings, and order information, to prevent data loss in case of server failure or system crashes.

Safeguards:

- The system must perform daily backups of the database, stored in secure, geographically redundant locations (e.g., cloud storage).
- Backup processes must be tested periodically to ensure data can be restored effectively in case of a failure.

Actions to Prevent:

- The system must prevent data deletion without proper authorization. For instance, critical data (e.g. orders, payment records) must only be deleted by authorized personnel after a thorough validation process.

5.2.2 Financial Security

All financial transactions must comply with industry standards (e.g., PCI DSS) to protect user payment information and prevent fraud.

Safeguards:

- **Payment information** (e.g. credit card details, mobile money) must be encrypted during transmission using **TLS 1.2 or higher**.
- The app must integrate **multi-factor authentication (MFA)** for financial transactions to ensure that only authorized users can initiate payments.

- **Transaction monitoring** must be implemented to detect unusual or fraudulent activities, with alerts generated for investigation.

Actions to Prevent:

- Unauthorized access to sensitive financial data must be prevented through stringent access control mechanisms, including user roles and permissions.
- Any attempt to manipulate payment records must trigger an immediate alert for administrative review.

5.2.3 Physical Safety of Delivery Personnel

The app must include features that prioritize the safety of delivery personnel during deliveries.

Safeguards:

- Delivery personnel must have access to real-time **GPS tracking** and a **panic button** within the app that sends an alert in case of emergency (e.g., safety concerns during delivery).
- Customers' addresses must be verified to reduce the risk of sending delivery personnel to unsafe or unverified locations.
- Delivery routes should be optimized using trusted GPS services (e.g., Google Maps) to avoid high-risk areas when possible.

Actions to Prevent:

- The app must prevent delivery personnel from accessing customers' full contact details except when necessary for delivery purposes (e.g., calling to confirm delivery).
- The app must prohibit delivery personnel from deviating too far from the planned route without notifying the system.

5.2.4 Compliance with External Regulations

The app must comply with all relevant external policies, regulations, and safety standards, including those related to data protection, online transactions, and delivery services.

Safeguards:

- The app must adhere to local transportation and delivery laws that may dictate the operation of delivery services, such as restrictions on delivery vehicles or service areas.

Actions to Prevent:

- The app must ensure that users are informed of their rights regarding their personal data, including how it is collected, stored, and used.
- The app must prevent non-compliant behavior (e.g., unverified users listing products for sale, or incomplete financial transactions) by enforcing stringent compliance checks.

5.2.5 Liability Protection

The app must include legal disclaimers to protect the developers from liability in case of loss, damage, or harm during product use.

Safeguards:

- Terms and conditions and liability waivers must be included and agreed upon during user registration, clarifying that the app serves as an intermediary between farmers, customers, and delivery personnel, and is not responsible for any physical harm or property damage resulting from the delivery process.
- The app must clearly define user responsibilities, including guidelines for the proper handling of products by farmers and customers.

Actions to Prevent:

- Misuse of the app (e.g. fraudulent listings, false claims) must trigger penalties such as user suspension or permanent account deletion.

5.2.6 Proof of Delivery and Dispute Resolution

The app must include a mechanism for verifying that deliveries are completed safely and correctly to minimize disputes and ensure proper resolution.

Safeguards:

- Delivery personnel must capture proof of delivery via photo or customer signature upon order completion. This proof is uploaded to the system and stored securely for reference in case of disputes.
- The app must include a dispute resolution system for cases where customers claim incomplete or damaged deliveries. Both parties (customers and delivery personnel) must have the opportunity to provide evidence, and the system must assist in reaching a resolution.

Actions to Prevent:

- The app must prevent delivery personnel from marking deliveries as complete without proper evidence (e.g., proof of delivery).
- Any discrepancies in the delivery process must be flagged for review before marking the order as finalized.

5.2.7 Safety Certifications

The app must comply with any necessary certifications related to online platforms, payment systems, and delivery services.

Certifications:

- **SSL/TLS Certification**: To ensure secure communication between the app, server, and third-party services.

5.3 Security Requirements

5.3.1 Data Encryption

All sensitive user data, including personal information, payment details, and communications, must be encrypted during transmission and at rest.

Safeguards:

- Data must be transmitted using TLS 1.2 or higher to ensure encrypted communication between the client and server.
- Data stored in the database (e.g., user profiles, orders, payment records) must be encrypted.

Purpose: Prevent unauthorized access to sensitive data, protecting users from data breaches and cyberattacks.

5.3.2 User Identity Authentication

All users (farmers, customers, delivery personnel) must authenticate using secure credentials to access the app.

Safeguards:

- The app must support OAuth 2.0 for user authentication, allowing secure login using email/password or third-party services like Google and Facebook.
- 2 factor authentication must be implemented for delivery personnel and farmers, particularly for accessing financial data or accepting deliveries, to add an extra layer of security.

Purpose: Prevent unauthorized access to user accounts and ensure only legitimate users can interact with the app's sensitive functions (e.g. placing orders, managing deliveries).

5.3.3 Role-Based Access Control (RBAC)

The system must implement role-based access control to ensure that users only have access to the features they need based on their role (farmer, customer, or delivery personnel).

Safeguards:

- Customers should only have access to product browsing, ordering, and account management.
- Farmers should only be able to manage their product listings and view their sales data.
- Delivery personnel should only access delivery-related information (e.g., pickup locations, delivery addresses).

Purpose: Minimize the risk of unauthorized users gaining access to sensitive or restricted parts of the system.

5.3.4 Privacy of User Data

The app must comply with data protection regulations to ensure that users' personal data is handled with the utmost privacy.

Safeguards:

- Users must be informed about the app’s data collection policies, including what data is collected, how it is used, and how long it is retained.
- Users must have the ability to **delete their accounts** and associated data upon request
- Personal information (e.g., customer addresses, phone numbers) must only be accessible to delivery personnel when absolutely necessary (i.e., during the delivery process).

Purpose: Protect user privacy by ensuring data is collected and processed transparently and securely, and by complying with legal privacy obligations.

5.3.5 Secure Payment Processing

All payment transactions must be processed securely to prevent fraud and protect users’ financial information.

Safeguards:

- All payment data must be tokenized and encrypted to prevent storage of sensitive credit card information on the app's servers.

Purpose: Ensure that all financial transactions are conducted securely, reducing the risk of fraud or data theft.

5.3.6 Audit Logging

The app must maintain detailed audit logs of all user interactions, particularly those related to account management, financial transactions, and delivery operations.

Safeguards:

- Logs must record user actions, such as logging in/out, placing orders, and updating product listings, with timestamps.
- Logs must be protected from unauthorized tampering and stored securely, with access restricted to system administrators.

Purpose: Ensure accountability by providing a clear record of all actions taken within the system, useful for investigating any security incidents or disputes.

5.3.7 Session Management

Secure session management practices must be implemented to protect user sessions and prevent session hijacking.

Safeguards:

- User sessions must expire after a specified period of inactivity to reduce the risk of unauthorized access.
- Session tokens must be securely stored and invalidated on logout to prevent reuse by attackers.

Purpose: Prevent unauthorized access to user accounts if a device is left unattended or if a session is compromised.

5.3.8 Secure API Communication

All communication between the mobile app and the backend server, including API calls, must be encrypted and secured.

Safeguards:

- **RESTful APIs** must be secured using **HTTPS** to ensure all data exchanges are encrypted.
- API access must be protected with API keys and tokens to ensure only authorized users and services can interact with the backend.

Purpose: Protect the integrity of the data exchanged between the mobile app and backend services, preventing man-in-the-middle (MITM) attacks or data tampering.

5.3.9 User Consent and Data Sharing

The app must seek explicit user consent before collecting or sharing any personal data with third-party services (e.g., payment processors, marketing platforms).

Safeguards:

- The app must present a clear and detailed privacy policy to users during registration, explaining what data is collected and shared.
- Users must have the ability to opt out of data sharing for non-essential services (e.g. marketing or analytics).

Purpose: Ensure that users have full control over their personal data

5.4 Software Quality Attributes

5.4.1 Adaptability

The app must be adaptable to different markets, allowing customization of language, currency, and legal requirements.

Measure: The system should support at least two languages at launch.

Purpose: Ensures the app can be expanded into different regions without major redesigns.

5.4.2 Availability

The app must maintain a 99.9% uptime, ensuring that users have access to the system without interruptions.

Measure: The app should have no more than 9 hours of downtime annually, including planned maintenance.

Purpose: Ensure continuous access for customers placing orders and for farmers managing their products.

5.4.3 Correctness

The app must accurately perform its functions, including product listing, order placement, and delivery tracking.

Measure: The error rate in transactions must not exceed 0.5% of total operations.

Purpose: Correctness ensures users receive the expected outputs from the system, fostering trust in the platform's reliability.

5.4.4 Flexibility

The system must be flexible enough to support future feature updates, such as adding new payment gateways or delivery options, without significant changes to the core structure.

Measure: New features should be implementable with less than 10% modification to the existing codebase.

Purpose: Flexibility ensures the system can evolve and adapt to new business needs and user demands.

5.4.5 Interoperability

The app must seamlessly integrate with third-party services such as payment gateways and GPS APIs.

Measure: Integration time for new services should not exceed 2 weeks, with minimal system downtime.

Purpose: Ensures the system can work with various external services, allowing for scalability and enhanced user experience.

5.4.6 Maintainability

The app's codebase should be easy to maintain, with well-documented code and modular design for easier updates and debugging.

Measure: Developers should be able to fix bugs or make updates within 1–2 hours on average, depending on the issue's complexity.

Purpose: A maintainable system ensures that the app can quickly adapt to changes, fix issues, and introduce new features.

5.4.7 Portability

The app must be portable across different operating systems, primarily Android and iOS, without significant differences in user experience or performance.

Measure: The app should support at least 2 operating systems (i.e., Android 7.0+ and iOS 12.0+), with no more than 5% feature variation between platforms.

Purpose: Ensures that the app can reach a broader audience by being available on different platforms.

5.4.8 Reliability

The system must reliably process all transactions and manage data integrity during order placements, updates, and deliveries.

Measure: The app should exhibit a 99.5% **success rate f**or all order-related processes (placing orders, updating delivery statuses).

Purpose: High reliability minimizes the risk of data loss or incorrect order processing, maintaining user trust in the app.

5.4.9 Reusability

Key components of the system, such as user authentication, payment handling, and delivery tracking, should be reusable across different modules or future applications.

Measure: At least 30% of the code base (especially core functions) should be reusable for future projects or modules within the same app.

Purpose: Reusability ensures efficient development, reducing time and effort required for future expansions.

5.4.10 Robustness

The app must gracefully handle unexpected conditions without crashing or losing data.

Measure: The system must recover from 95% of network-related failures within 5 seconds, and all transactions must be logged to avoid data loss.

Purpose: Robustness ensures that the app can withstand various conditions and continue functioning without major disruptions.

5.4.11 Testability

The system must be easily testable.

Measure: Unit tests, integration tests, and end-to-end tests must be implemented to ensure comprehensive testing across all system modules.

Purpose: Testability ensures that issues can be detected and resolved early in the development process, leading to higher quality software.

5.4.12 Usability

The app must be easy to use for all user groups (farmers, customers, delivery personnel) with a focus on intuitive design and minimal learning curve.

Measure: Users should be able to complete key tasks (e.g., placing an order, listing a product) within 3 steps on average.

Purpose: High usability ensures that the app provides a positive user experience, encouraging user retention and minimizing support requests.

5.5 Business Rules

5.5.1 Role-Based Access Control (RBAC)

Different roles (farmers, customers, delivery personnel, administrators) have distinct access levels and permissions within the app. The system must implement role-based access control to ensure that each role can only access functions and data specific to their responsibilities.

Functional Requirements:

- The app must authenticate users and assign roles (e.g., farmer, customer, delivery personnel) upon registration.
- Role-based dashboards must be provided, showing relevant features (e.g., product listing for farmers, order history for customers).
- Admin users should have full access to manage all user roles and monitor system activity.

5.5.2 Farmers' Permissions

Farmers can only list, update, and manage their own products. They cannot access or modify other farmers’ listings. The system must enforce data isolation so that each farmer can only interact with their own product data.

Functional Requirements:

- Farmers must be able to create and edit product listings, view order history, and manage inventory only for their products.
- Farmers should be notified when an order is placed or when stock levels are low.
- Farmers cannot view or edit other farmers' products or sales data.

5.5.3 Customers' Permissions

Customers can browse products, place orders, view order history, and track deliveries. They cannot access the management functions available to farmers or delivery personnel. Customer access should be restricted to product browsing and ordering functions.

Functional Requirements:

- Customers must be able to browse products, add items to the cart, and place orders.
- Customers should only be able to view their own order history and track deliveries in real time.
- Customers should be able to leave feedback or rate the products/delivery service but cannot modify product listings or delivery details.

5.5.4 Delivery Personnel Permissions

Delivery personnel can only access delivery jobs assigned to them. They can view customer delivery addresses and contact information only for assigned orders. The system must limit delivery personnel access to job-related information to protect customer privacy.

Functional Requirements:

- Delivery personnel must be able to view available delivery jobs and accept them through the app.
- Once a job is accepted, delivery personnel can view pickup and drop-off locations, customer names, and contact details.
- Delivery personnel should be able to update the delivery status (e.g., in transit, delivered) and provide proof of delivery.
- Delivery personnel cannot access customer orders or product listings unrelated to their deliveries.

5.5.5 Administrative Permissions

Admins have full access to the app, including user management, system monitoring, and reports. They can intervene in disputes, monitor transactions, and enforce security policies. Administrators should have access to all areas of the app to ensure proper system functioning, dispute resolution, and security enforcement.

Functional Requirements:

- Admins must be able to create, suspend, or delete user accounts (farmers, customers, and delivery personnel).
- Admins can view and modify any order, product listing, or delivery data in case of disputes or fraud investigations.

- Admins must have access to system logs, error reports, and performance metrics for monitoring.
- Admins should be able to enforce compliance with security policies (e.g., password resets, enforcing two-factor authentication).

5.5.6 Order Management

Once an order is placed, customers cannot modify it, but they can cancel it within a certain time frame before the order is picked up by delivery personnel. The system must include restrictions that prevent order modifications after submission, but allow for cancellations within defined rules.

Functional Requirements:

- Customers must be able to cancel orders before the order is dispatched for delivery.
- Once delivery personnel accept the order, the cancellation option should be disabled, and the customer can only track the delivery.
- The system must notify farmers and delivery personnel immediately upon order cancellation.

5.5.7 Payment Processing

Payments are processed once the order is confirmed, and funds are held until the order is delivered and accepted by the customer. Refunds are issued in cases of non-delivery or disputes. The payment system must handle transactions securely and allow for refunds based on the delivery outcome.

Functional Requirements:

- Payments must be securely processed via third-party gateways once the customer places an order.
- The system must hold payments until delivery is completed and confirmed by the customer.
- In case of non-delivery or disputes, the system must provide an option to issue refunds to the customer based on the resolution of the dispute.

5.5.8 Dispute Resolution

Disputes between customers, farmers, or delivery personnel must be managed within the app, with admins having the final say in resolving disputes. The system must provide a mechanism for reporting and resolving disputes, with admins having oversight.

Functional Requirements:

- The app must allow customers to raise disputes related to product quality, incorrect deliveries, or delayed orders.
- Farmers and delivery personnel must be able to respond to disputes within a defined timeframe.
- Admins must have the ability to review the evidence (e.g., proof of delivery, customer complaints) and make a final decision on the outcome.

5.5.9 Data Privacy and Security

Users must not have access to sensitive data beyond what is necessary for completing their tasks. Customer data (e.g., delivery addresses) is only shared with delivery personnel for the duration of the delivery. The system must enforce strict data access controls to protect user privacy.

Functional Requirements:

- The app must restrict access to personal data based on role and only when necessary for task completion.
- Once a delivery is completed, the delivery personnel should no longer have access to the customer's personal data.
- Customer data should be encrypted and protected from unauthorized access at all times.

6. Other Requirements

6.1 Reuse Objectives

The app's architecture must support reuse of components for future projects or extensions, minimizing the need for rewriting code.

- **Modular Design**: The system architecture should be modular, with clear interfaces and APIs for components such as user authentication, payment processing, and product management.
- **Documentation**: Comprehensive documentation must be provided for reusable components, including setup instructions and usage guidelines to facilitate integration into future applications.

6.2 Accessibility Requirements

The app must be accessible to all users, including those with disabilities.

- **User Testing**: The app should undergo user testing with individuals who have disabilities to identify and address any accessibility issues prior to launch.

6.3 Deployment Requirements

The app must be deployable in a cloud environment, ensuring scalability and performance.

- **Cloud Provider**: The app should be hosted on a cloud platform (e.g., AWS, Azure) to leverage features such as load balancing, auto-scaling, and high availability.
- **CI/CD Pipeline**: A continuous integration and continuous deployment (CI/CD) pipeline must be established to facilitate regular updates and rollbacks while minimizing downtime.

6.4 Monitoring and Maintenance Requirements

The system must have monitoring tools in place to track performance, user activity, and error rates in real time.

- **Monitoring Tools**: Utilize tools such as **Google Analytics** for user tracking

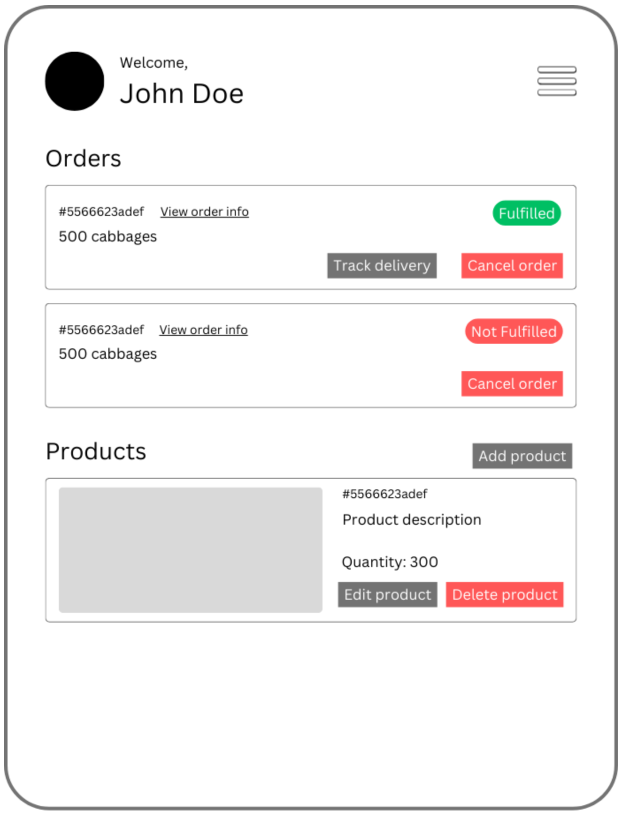
- **Maintenance Schedule**: Regular maintenance must be conducted at least once a month to ensure system performance, apply security patches, and optimize database performance.

Appendix A: Glossary

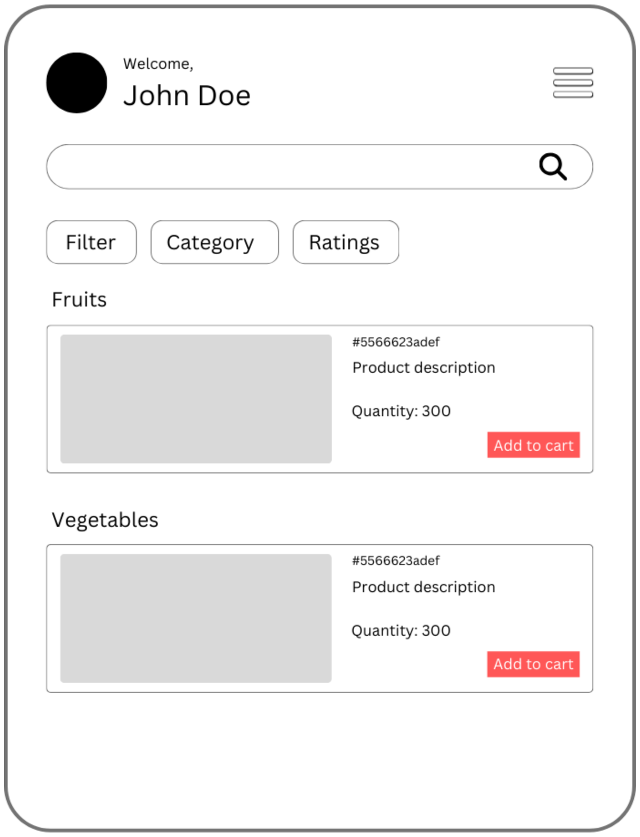
- **GPS**: Global Positioning System used for real-time tracking.
- **HTTPS**: Secure communication protocol.

Appendix B: Analysis Models

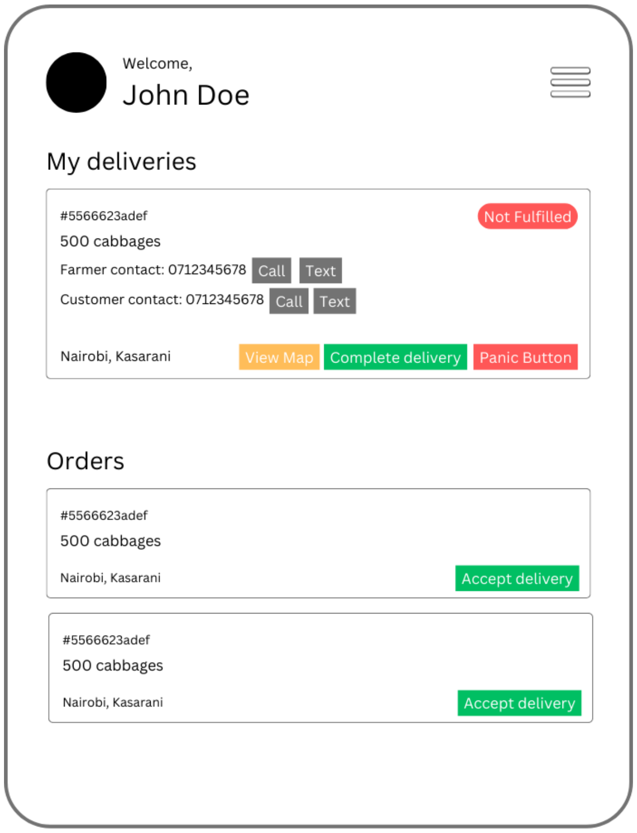
Farmer dashboard



Customer interface



Delivery personnel interface



Welcome,
John Doe

My deliveries

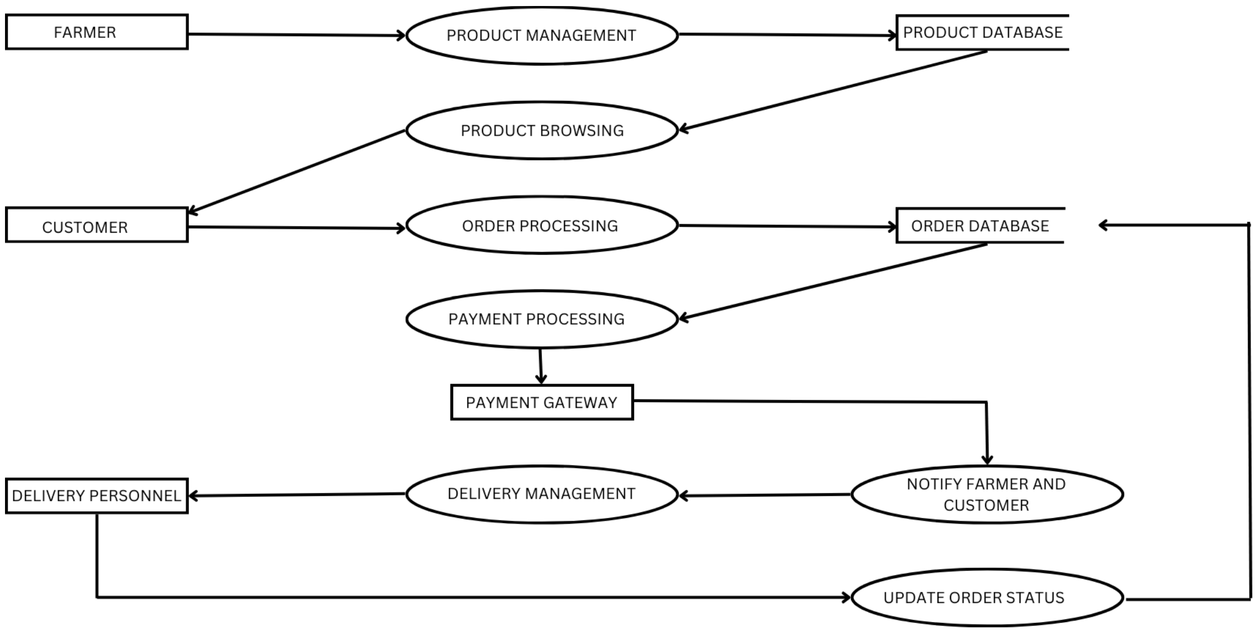
#5566623adef
500 cabbages
Farmer contact: 0712345678
Customer contact: 0712345678
Nairobi, Kasarani

Orders

#5566623adef
500 cabbages
Nairobi, Kasarani

#5566623adef
500 cabbages
Nairobi, Kasarani

Data Flow Diagram



Appendix C: To Be Determined List

- Final decision on payment gateways.
- List of third-party APIs for navigation and location services.

Appendix D: Software Processing Model

Agile (Scrum) is the most suitable software processing model for the development of AgriSmart app due to the following reasons:

1. **Sprints**

   Scrum divides development into short iterations called sprints which is ideal for developing specific features of the app incrementally, quickly and effectively.

2. **Regular Feedback and Adaptability**

   Each sprint includes a sprint review to gather feedback from stakeholders therefore ensuring that the development team can adapt to changing requirements and priorities.

3. **Product Backlog and Prioritization**

   Scrum uses a product backlog to manage and prioritize features based on customer needs and business value. This helps focus development efforts on the most important aspects of the app first and ensures continuous delivery of value.

4. **Visibility and Transparency**

   Scrum's daily stand-up meetings and sprint planning sessions ensure transparency, allowing the team to track progress, identify potential blockers, and make real-time adjustments.

Model	Reason for elimination
Waterfall model	Lacks flexibility for evolving requirements. Modifying stages after completion is challenging
V model	Best suited for projects with fixed requirements and extensive testing phases unlike AgriSmart which has evolving requirements.
Incremental model	Allows partial delivery but lacks the user-centered focus and continuous feedback loops of Scrum. It’s less suitable for evolving requirements and frequent testing cycles needed for this project.
Rapid Appllication Development (RAD) model	It heavily depends on availability of skilled developers and continuous user engagement, which may be difficult.
Iterative model	Provides some flexibility but does not prioritize frequent user feedback as much as Scrum does. Early design flaws may also become difficult to correct later on, making it less suitable for evolving requirements.
Prototype model	Focuses on building a temporary system for refining requirements but not for delivering a complete product incrementally. This approach may lead to delays in developing a final, fully functional app.
Spiral model	Emphasizes risk analysis and is often used for large, complex systems. Its complexity and cost are unnecessary for this project, while Scrum offers a simpler approach that efficiently manages risks and development cycles.

Kanban	Suitable for continuous delivery but lacks the structured iterations (sprints) and defined roles that Scrum provides. Kanban may not be as effective for managing the evolving requirements and specific feature priorities of the app.
XP (Extreme Programming)	XP focuses heavily on coding practices like pair programming and test-driven development, which may not be feasible given the varied development needs of this project. Scrum provides a more balanced approach with flexibility and collaboration beyond coding.