Software Requirements Specification
Inventory Management System for Farmers

Prepared for Final Year Project

June 2, 2025

1 Introduction

1.1 Purpose

This Software Requirements Specification (SRS) outlines the functional and non-functional requirements for the Inventory Management System for Farmers, a web-based application designed to assist farmers in managing crop inventories, accessing a marketplace, detecting crop diseases using AI, receiving weather alerts, and interacting with a chatbot for farming-related queries. Customers can browse and purchase agricultural products directly from farmers with a hybrid payment model (partial online payment and cash on delivery). The document serves as a blueprint for developers, stakeholders, and project evaluators.

1.2 Scope

The Inventory Management System will provide:

- A farmer module for managing crop inventory and selling products in a marketplace.
- AI-based crop disease detection with fertilizer recommendations.
- Real-time weather alerts for cyclone or heavy rain forecasts.
- A customer module for browsing, purchasing, and tracking orders.
- A hybrid payment system requiring 50% online payment and the remainder via cash on delivery (CoD).
- An AI-powered chatbot for farmer support.
- An admin panel for platform management.

The system will be developed using Django (backend), HTML, CSS, Bootstrap, and minimal JavaScript (frontend), with SQLite or PostgreSQL as the database.

1.3 Intended Audience

This SRS is intended for:

- **Developers**: To guide implementation of the system.

- **Farmers**: Primary users managing inventory and sales.
- **Customers**: End-users purchasing agricultural products.
- **Admin**: Platform overseers managing users and content.
- **Project Evaluators**: To assess the project scope and feasibility.

1.4 Definitions and Acronyms

SRS Software Requirements Specification

AI Artificial Intelligence

UI User Interface

CRUD Create, Read, Update, Delete

CoD Cash on Delivery

API Application Programming Interface

2 Overall Description

2.1 Product Perspective

The Inventory Management System is a standalone web application built using Django, with a frontend developed using HTML, CSS, Bootstrap, and minimal JavaScript. It integrates external APIs for weather forecasts and payment processing, and a separately hosted AI model for crop disease detection. The system supports role-based access for farmers, customers, and admins, ensuring a seamless farm-to-consumer ecosystem.

2.2 Product Functions

- User authentication (register, login, logout) with role-based access.
- Farmer module: Manage crop inventory, list products in the marketplace, upload images for disease detection, view weather alerts, and interact with a chatbot.
- Customer module: Browse products, add to cart, make partial online payments, and track orders.
- Admin module: Approve farmer accounts, manage listings, and oversee platform analytics.
- AI-based crop disease detection via image uploads, with fertilizer recommendations.
- Real-time weather alerts for critical conditions (e.g., cyclones, heavy rain).
- Chatbot to answer farming, marketplace, and platform-related queries.

2.3 User Characteristics

- **Farmers**: Basic digital literacy, primarily using mobile devices. Require simple, intuitive UI.
- **Customers**: General internet users familiar with e-commerce platforms.

- **Admin**: Technical users with full access to manage the platform.

2.4 Constraints

- Technology stack is limited to Django (backend), HTML, CSS, Bootstrap, and minimal JavaScript (frontend).
- AI model for disease detection must be hosted separately and integrated via REST API.
- Weather and payment functionalities depend on third-party APIs.
- System must be mobile-responsive due to farmer usage patterns.

2.5 Assumptions and Dependencies

- Users have reliable internet access.
- Images uploaded for disease detection are valid and of sufficient quality.
- Third-party APIs (weather, payment) remain available and functional.
- AI model provides accurate disease detection and recommendations.

3 Specific Requirements

3.1 Functional Requirements

3.1.1 Authentication

- Users (farmers, customers, admin) can register with email, password, and role-specific details.
- Login using email and password with role-based redirection to respective dashboards.
- Logout functionality to end user sessions securely.

3.1.2 Farmer Module

- **Inventory Management**: Farmers can add, update, or delete crops with details (name, quantity, shelf-life, condition).
- **Marketplace Listing**: Farmers can list crops for sale with tags (e.g., organic, seasonal) and price.
- **Disease Detection**: Farmers can upload crop images to detect diseases via an AI model, receiving disease names and fertilizer recommendations.
- **Weather Alerts**: System displays real-time weather forecasts (e.g., cyclone, heavy rain) via a third-party API.
- **Chatbot**: Farmers can interact with an AI chatbot for queries on farming, sales, or platform usage.
- **Reminders**: Auto-notifications for crops nearing expiry.

3.1.3 Customer Module

- **Product Browsing**: Customers can view and search products by category (e.g., vegetables, fruits).
- **Cart Management**: Add/remove items to/from cart.
- **Payment**: Support 50% online payment (via Razorpay or Stripe) and 50% CoD.
- **Order Tracking**: View order status (processing, shipped, delivered).
- **Farmer Profiles**: View farmer details and ratings.

3.1.4 Admin Module

- Approve or reject farmer registrations.
- Manage reported users or listings.
- View platform analytics (e.g., sales, active users).
- Update chatbot FAQ database.

3.1.5 External Integrations

- **Weather API**: Integrate OpenWeatherMap or similar for localized weather alerts.
- **Payment Gateway**: Integrate Razorpay or Stripe for secure online payments.
- **AI Disease Detection**: REST API to accept image uploads and return disease details and fertilizer suggestions.

3.2 Non-Functional Requirements

3.2.1 Performance

- Support up to 100 concurrent users without significant latency.
- Page load time under 2 seconds under normal conditions.

3.2.2 Reliability

- Target 95% uptime.
- Daily database backups to prevent data loss.

3.2.3 Security

- Enable CSRF protection for all forms.
- Validate and sanitize all user inputs.
- Use HTTPS for secure data transmission.
- Secure storage of API keys and user credentials.

3.2.4 Usability

- Mobile-first, responsive UI using Bootstrap.
- Intuitive navigation with clear call-to-action buttons.
- Error messages in simple language for farmers.

3.2.5 Maintainability

- Modular Django app structure for easy updates.
- Use Django admin panel for backend management.
- Well-documented code with comments.

3.3 External Interfaces

3.3.1 User Interface

- Responsive web interface built with HTML, CSS, Bootstrap.
- Separate dashboards for farmers, customers, and admin.
- Minimal JavaScript for dynamic features (e.g., form validation, alerts).

3.3.2 Software Interfaces

- **Django**: Backend framework for handling logic, authentication, and database operations.
- **SQLite/PostgreSQL**: Database for storing user, inventory, and order data.
- **OpenWeatherMap API**: For weather forecasts and alerts.
- **Razorpay/Stripe API**: For online payment processing.
- **AI Model API**: REST API for crop disease detection.

3.4 System Models

- **Use Case Diagram** (not included in text): Represents interactions for farmers (manage inventory, list products, detect diseases), customers (browse, purchase, track), and admin (manage users, analytics).
- **Database Schema** (suggested):
  - `User`: id, email, password, role (farmer/customer/admin).
  - `Farmer`: user_id, name, address, contact.
  - `Crop`: id, farmer_id, name, quantity, shelf_life, condition.
  - `Inventory`: crop_id, quantity, expiry_date.
  - `MarketplaceListing`: crop_id, price, tags, status.
  - `Order`: id, customer_id, farmer_id, items, status, payment_status.

- `Payment`: order_id, amount, method, status.
- `ChatQuery`: id, farmer_id, query, response.

4 Appendices

- API keys for weather and payment services will be stored in environment variables.
- AI model for disease detection will be hosted separately and accessed via REST API.
- System assumes stable internet connectivity for real-time features.