System Implementation

# 1. Introduction

This chapter documents the actual implementation phase of the smart agriculture management system. The system is designed to serve farmers, buyers, drivers, and administrators in a unified platform to manage farms, crops, weather conditions, soil health, orders, and transport logistics.  
  
Following a requirements analysis and architectural design phase, this stage focused on translating design specifications into functional modules through iterative development. The project adhered to Agile methodology, with development divided into structured sprints that facilitated rapid prototyping, validation, and feedback integration.  
  
Technologies employed included Django REST Framework for the backend, PostgreSQL + PostGIS for relational and geospatial data storage, Redis + Celery for background tasks, and Djoser + JWT for secure user authentication.  
  
Tools such as Postman, GitHub, Docker, and Pytest/unittest were used for development, testing, and version control. All APIs were tested iteratively, and continuous feedback loops enabled quick resolution of bugs and alignment with evolving project requirements.

# 2. Development Environment

|  |  |
| --- | --- |
| Component | Technology / Tool |
| Backend | Python 3.13, Django REST Framework |
| Geospatial DB | PostgreSQL 15 + PostGIS Extension |
| Authentication | Djoser + JWT with Secure Cookies |
| Asynchronous Tasks | Celery + Redis |
| Testing Tools | Django unittest, mock.patch |
| API Testing | Postman |
| Version Control | Git + GitHub |
| IDE | VS Code |
| Task Automation | Custom management commands + crontab |
| Deployment | (Planned) Docker + Azure |

# 3. Implementation Strategy (Agile & Sprint Breakdown)

The system was implemented in five Agile sprints, each approximately one week long. Daily commits and regular testing ensured the product evolved incrementally and remained stable throughout development.

|  |  |  |
| --- | --- | --- |
| Sprint | Activities | Deliverables |
| 1 | Set up Django project, configure PostgreSQL + PostGIS, initialize GitHub repo | Working repo, initial models (User, Farm) |
| 2 | Implement user auth with JWT & cookie storage, define role-based access | User login/logout + secure routes |
| 3 | Create modules for Farm, Crop, Soil, and Weather | API endpoints for farm management + soil records |
| 4 | Implement orders, transport, and signal-based evaluations | Dynamic order flow + Celery weather task |
| 5 | Testing, debugging, validation, and documentation generation | Unit tests, ER diagram, final integration |

# 4. Module-by-Module Implementation

## a) User Authentication Module

Implemented using Djoser with JWT tokens stored securely in HttpOnly cookies. A CustomJWTAuthentication class was defined to support cookie-based login verification. Role-based access (FARMER, BUYER, DRIVER, ADMIN) was enforced across endpoints using custom permission classes (IsFarmerOrAdmin, IsOwner).

## b) Core Business Logic Modules

|  |  |
| --- | --- |
| Module | Summary |
| Farm Module | Allows authenticated farmers to register farms with geolocation (point and polygon geometry). |
| Crop Module | Enables farmers to track planted crops, planting seasons, and expected harvests. |
| Soil Module | Stores soil samples per farm and auto-triggers evaluation for crop suitability using a Hugging Face model. |
| Weather | Fetches and records weather conditions per farm using OpenWeatherMap API via scheduled Celery tasks. |
| Order System | Buyers can place crop orders, tracked through multiple statuses (PND, DLVD, etc.). |
| Transport | Admins can assign drivers to deliver orders; routes and duration are logged using GIS fields. |

## c) API Implementation

All APIs were implemented as RESTful endpoints using Django REST Framework’s APIView, ListCreateAPIView, RetrieveAPIView, and ModelViewSet where applicable.  
  
Postman was used extensively to simulate CRUD operations and validate authentication flows. Serializer validation ensured only correct data entered the database, e.g., validate\_owner() to restrict farm ownership to FARMER role only.  
  
Signal-based automation (e.g., post\_save on Soil and Crop) ensured evaluations and dependencies were handled transparently.