Analysis and Design Document: Mushroom Guardians *ICS 2309 – Commercial Programming & BCT 2315 – Computer Systems Project JHUB Africa | April 2025*

1. Introduction

Mushroom Guardians is a context-aware, IoT-driven farm management platform built to empower mushroom farmers through intelligent data collection and actionable insights. Having identified the challenges faced by smallholder mushroom growers, our project has moved from conceptualization and requirements gathering to the stage of detailed analysis and design. This document outlines the functional and non-functional expectations, the architecture, design patterns, UI/UX principles, and the overall system blueprint. The insights presented here will guide the upcoming implementation phases.

This phase bridges the visionary goals and technical execution. It transforms farmer-centric needs and commercial aspirations into concrete system models, interfaces, databases, and workflows, in line with modern software engineering principles.

2. Functional Requirements

Our system is centered around key functions designed to meet both operational and commercial goals:

- User Authentication: A secure login and registration system with roles such as Admin, Farm Manager, and Viewer.
- Real-Time Monitoring Dashboard: Display of current temperature, humidity, and CO_2 values from connected sensors.
- Threshold Alerts: Immediate notifications when critical environmental parameters exceed safe ranges.
- Farm and Device Management: Admin panel to onboard new farms, link sensors, and define alert thresholds.

- Historical Analytics: Charts and reports showing environmental trends over time, powered by TimescaleDB.
- Data Export: Capability to download reports in CSV or PDF formats.
- Mobile-Friendly Views: Responsive interfaces that adapt to phones and tablets.

3. Non-Functional Requirements

- Performance: The system should support up to 100 concurrent users and stream sensor data with a latency below 500ms.
- Scalability: Designed with microservices and FIWARE context brokers to scale horizontally.
- Security: JWT authentication, role-based access, HTTPS-only communication, encrypted data at rest.
- Maintainability: Modular codebase, standardized APIs, well-commented and peer-reviewed.
- Availability: 99.5% uptime target with automatic failover via cloud hosting.
- Compliance: GDPR and Kenyan data protection law considerations, especially with sensor data linked to identifiable farms.

4. Software and Hardware Requirements

Software Stack:

- Frontend: React.js, TailwindCSS
- Backend: Node.js + Express
- Database: PostgreSQL with TimescaleDB extension
- IoT Middleware: FIWARE Orion Context Broker, IoT Agent, QuantumLeap
- CI/CD: GitHub Actions, Docker
- Deployment: Heroku (development), AWS (production)
- Monitoring: Grafana + Prometheus

Hardware:

- Raspberry Pi with Wi-Fi
- DHT11 (temperature and humidity sensor)
- MQ135 or equivalent CO_2 sensor
- Power supply, SD cards, cabling

5. System Architecture

We adopt a three-tier architecture to ensure separation of concerns:

- Presentation Layer: Web frontend built in React.js. Fetches data via secured REST APIs. Renders real-time dashboards and reports.
- Application Layer: Node.js/Express backend handling business logic, API endpoints, and interfacing with the FIWARE broker.
- Data Layer: PostgreSQL with TimescaleDB and FIWARE QuantumLeap for efficient time-series storage.

Data flows from sensors to FIWARE IoT Agent, gets updated into the Orion Context Broker, then passed to QuantumLeap for persistence and long-term querying.

6. Design Patterns

- MVC Pattern: Applied to frontend (React components) and backend (routes/controllers/services).
- Observer Pattern: Used to track sensor data updates and trigger alert notifications.
- Factory Pattern: For dynamic creation of sensor configurations and user roles.
- Singleton Pattern: Managing context broker connections and configurations.

7. User Interface and UX Design

UI/UX design has been driven by principles of clarity, accessibility, and responsiveness. High-fidelity wireframes were created in Figma and iterated with user feedback from prospective farmers. Major UI components include:

- Login Page: Clean form with validation and password reset.
- Dashboard: Real-time gauges and trend charts, color-coded for thresholds.
- Farm View: Displays linked sensors and current readings.
- Admin Panel: Farm registration, threshold configuration, user role management.

Each interface supports both English and Kiswahili, ensuring usability for local users. The layout follows a mobile-first design to support users with limited hardware access.

8. Database Design

Database modeling followed a normalization-first approach, ensuring minimal redundancy and maximum query efficiency. Tables include:

- `users`: Stores user credentials, role, and linked farm IDs.
- `farms`: Contains metadata on registered farms and configurations.
- `sensors`: Device IDs, types, and operational parameters.
- `readings`: Timestamped sensor data ingested from FIWARE.
- `alerts`: Captured violations of environmental thresholds.
- `logs`: System and user activity for auditing.

An Entity-Relationship Diagram (ERD) has been designed to reflect relationships such as one-to-many between farms and sensors, and one-to-one between users and roles.

9. Security Architecture

Security is integral to Mushroom Guardians. All communications between frontend, backend, and FIWARE services use HTTPS. JWT tokens ensure authenticated API access. Passwords

are hashed using bcrypt, and input validation is enforced to mitigate XSS and SQL injection attacks. Users are granted permissions based on roles, and an audit trail is maintained for all critical actions.

10. Scalability and Deployment Strategy

The system is cloud-native, built with scaling in mind:

- Docker containers package the application for portability.
- GitHub Actions automate build, test, and deploy phases.
- Heroku is used for rapid deployment during development.
- AWS EC2 + RDS will host production services, with load balancing.
- Grafana dashboards provide performance and uptime monitoring.

11. Agile Development Plan

Work is divided into bi-weekly sprints:

- Sprint 1: Repo setup, wireframes, sensor simulation.
- Sprint 2: Auth system and database.
- Sprint 3: FIWARE integration and live dashboards.
- Sprint 4: Alerts, reports, and admin features.
- Sprint 5: CI/CD setup, security, and documentation.
- Sprint 6: Final testing, deployment, and presentation.

Stand-ups are held daily and retrospectives bi-weekly. JIRA and GitHub Projects are used for sprint management.

12. Risk Analysis

Risk	Mitigation Plan
Hardware failure	Pre-test sensors; maintain spare kits.
Integration with FIWARE	Mock services and use of sandbox environments.
Performance bottlenecks	Load testing and caching of heavy analytics queries.
User adoption challenges	Multilingual support and training materials.

13. Collaboration Tools and Workflow

The team uses the following collaboration suite:

- GitHub: Code versioning, issue tracking, and documentation.
- Trello: Task management.
- Slack: Daily coordination and status updates.
- Google Drive: Documentation and report storage.
- Figma: UI/UX collaboration.

14. Conclusion

The design outlined in this document is practical, scalable, and grounded in real needs. With a solid architectural foundation and user-centered approach, Mushroom Guardians is well-positioned to deliver a transformative solution in the agri-tech landscape. This blueprint will guide the implementation phase and ensure the successful deployment of a meaningful and commercially viable product.