**1. Introduction**

**1.1 Purpose of this Document**

* This System Design Document (SDD) specifies the architecture, Components, interfaces, and other design considerations for **FarmBotika**, an **end-to-end smart-agriculture** platform. It bridges the gap between requirements and implementation, ensuring that developers, testers, and stakeholders share a clear, common blueprint.

**1.2 Scope of the System**

FarmBotika will enable small-scale farmers to:

* Monitor field conditions, example- Soil moisture, temperature, pest detection- via IoT sensors.
* Access AI-Powered recommendations and chat-based agronomic support.
* Manage farm data, plan planting schedules, and track yields.
* Receive alerts and notifications on mobile and web interfaces.

**1.3 Intended Audience**

1. **Developers:** Implement modules per design.
2. **Testers:** Develop and execute test cases.
3. **DevOps/ Infrastructure Engineers:** Deploy and operate the system.
4. **Project Managers/ Stakeholders:** Review high-level architecture and risk mitigations.

**1.4 Glossary**

* **API Gateway:** Single entry point for client requests, routing to microservices.
* **LLM:** Large Language Model, used for agronomy Q&A.
* **Celery:** Distributed task queue.
* **JWT:** JSON Web Token, for stateless authentication.

**2. System Overview and Design Philosophy**

**2.1 High Level Description**

FarmBotika is implemented as a microservices-based web application:

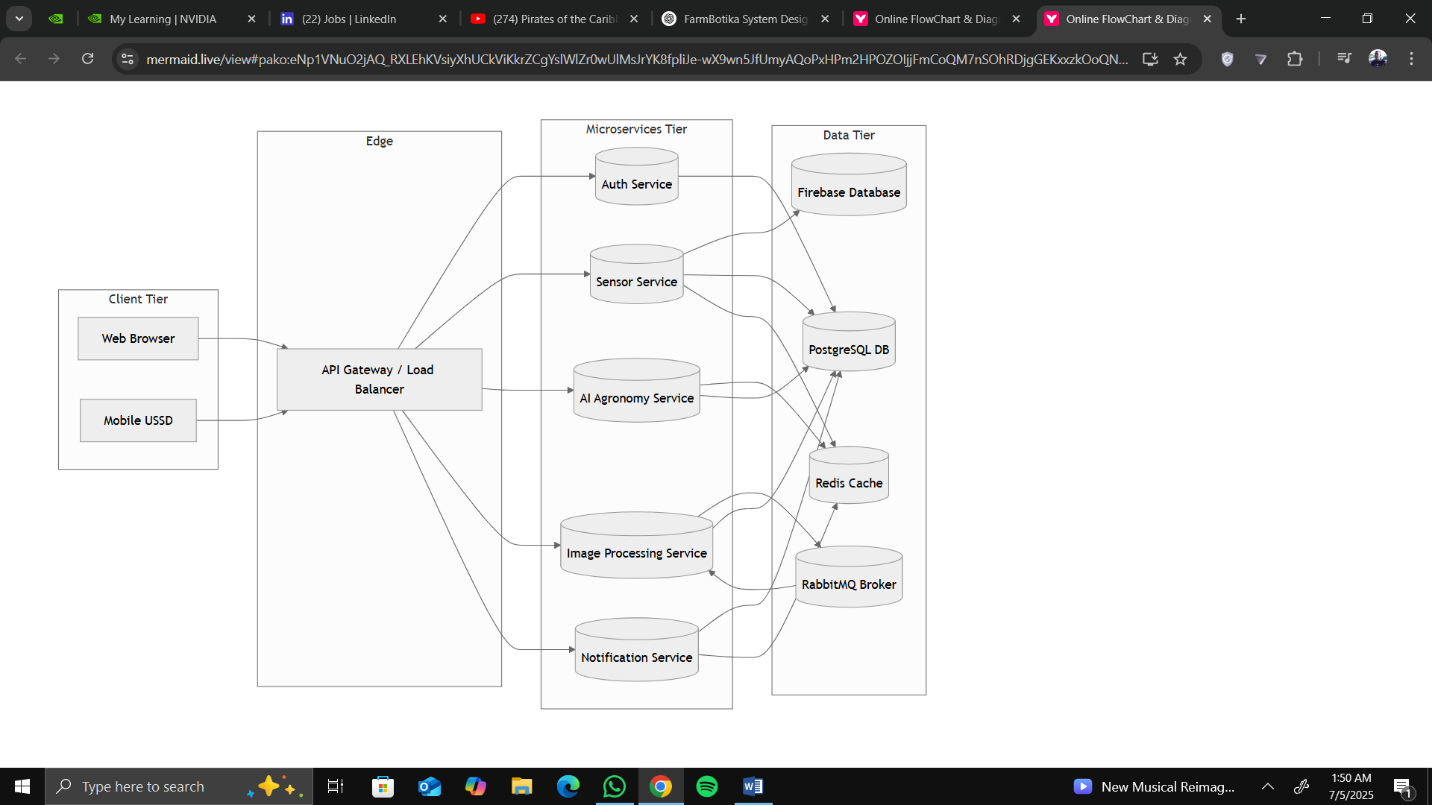
* **Frontend**: React, Tailwind CSS, JavaScript, Mobile-responsive.
* **Backend:** Python, Django REST Framework services, Node JS.
* **AI Service:** Flask microservice hosting the LLM for agronomic advice.
* **Data Pipeline:** Celery + RabbitMQ for asynchronous tasks.
* **Database:** PostgreSQL for relational data; Firebase for IoT data; Redis for caching and Celery broker.

**2.2 Design Principles**

* **Modularity:** Each service encapsulates a distinct domain (User, Sensor, AI).
* **Scalability:** Stateless services behind load balancers, horizontal scaling.
* **Reusability:** Shared libraries for authentication, logging, error handling.
* **Performance:** Caching hot data in Redis; lazy loading in Frontend.
* **Security:** HTTP everywhere; JWT for API access; encrypted storage for sensitive data.

**3. Architectural Design**

**3.1 Architecture Diagram**



**3.2 Architecture Style**

* **Microservices:** communicating via REST and AMPQ.
* **Client-Server:** React frontend calls backend via API Gateway.
* **Event-Driven** for long-running tasks.

**3.3 Component Description**

|  |  |
| --- | --- |
| **Component** | **Description** |
| 1. **API Gateway** | Routes requests to appropriate backend services; handles rate limiting and authentication. |
| 1. **Auth Service** | Manages user signup/login, role-based access (Admin, Staff, Farmer). |
| 1. **Sensor Service** | Ingests IoT sensor data; validation and storage. |
| 1. **AI Service** | Hosts LLM inference; processes user queries for agronomic advice. |
| 1. **Image Service** | Receives field images; runs pest detection computer-vision pipeline; stores results. |
| 1. **Web Frontend** | React SPA; dashboards, data entry forms, real-time charts. |
| 1. **Mobile Frontend** | Progressive Web App optimized for low-bandwidth mobile devices. |
| 1. **Database** | PostgreSQL with tables for users, farms, sensors, readings, alerts. |
| 1. **Cache** | Redis for session data, recent sensor readings, AI response caching. |
| 1. **Message Broker** | RabbitMQ for Celery tasks (e.g., image processing, bulk notifications). |
| 1. **CI/CD Pipeline** | GitHub Actions building, testing, and deploying containers to Docker. |

**4. Detailed Design**

**4.1 Module / Feature Descriptions**

**4.1.1 User Management**

* **Functionality:** Register/login, role, role assignment, password reset.
* **Inputs:** username, password, email, role.
* **Outputs:** JWT token, user profile
* **Business Rules:**
  + Passwords hashed with Bcrypt.
  + Email verification required.
* **Data Elements**: userID, username, email, password\_hash, role

**4.1.2 Sensor Data Ingestion**

* **Functionality:** Accepts sensor payloads (JSON), validates, stores readings.
* **Inputs:** sensor\_id, timestamps, metrics (moisture, temperature).
* **Outputs:** HTTP 201 on success; queued alerts if thresholds crossed.
* **Business Rule:** 
  + Reject data older than 24 hours.
* **Data Elements:** readingID, sensor\_id, metric\_type, value, timestamp.

**4.1.3 AI-Powered Agronomy Chat**

* **Functionality:** Processes farmer questions; returns LLM response.
* **Inputs:** user\_query, context (farm\_id, recent\_readings).
* **Outputs:** text\_response, confidence\_score.
* **Business Rule:**
  + Rate-limit queries per user, example: 20/day.
* **Data Elements**: queryID, userID, timestamp, prompt, response, cost\_metrics.

**4.1.4 Notification Service**

* **Functionality:** Sends alerts and informational messages to users, and handles message automation.
* **Inputs:** userID, message\_body, notification\_type (alert info, warning), trigger\_source (Sensor, AI, Admin)
* **Outputs:** Push/Email/SMS alert sent to user.
* **Business Rules:**
  + High-Priority alerts, example: critical soil dryness, are marked as “urgent”.
  + Alerts are rate-limited per user, example: max 10/hour.
  + Users can mark alerts as read or delete them.
  + Unread alerts are highlighted in the dashboard.
* **Data Elements:** notification\_id, userID, message, type, status, triggered\_by, created\_at, read\_at

**4.1.5 Farm Management**

* **Functionality:** Allows users to create, update, and manage farm details (name, location, crop type).
* **Inputs:** name, location (GPS), size, crop\_type, userID
* **Outputs:** new or updated farm profile, farm overview on dashboard, associated sensors and reports.
* **Business Rules:**
  + One user can own multiple farms.
  + Each farm must have a unique name per user.
  + Location is validated using geolocation services.
  + Users can deactivate a farm.
* **Data Elements:** farm\_id, userID, name, location, size\_in\_hectares, crop\_type, status, created\_at, upd\_at.

**NB:**<< Image Recognition service to be added >>

**4.2 Interface Design**

**4.2.1 UI Mockups**

* **Dashboard**: real-time charts of soil moisture, alerts panel.
* **Chart Screen:** text input, message history, quick-tips, sidebar.

**4.2.2 Navigation Flow:**

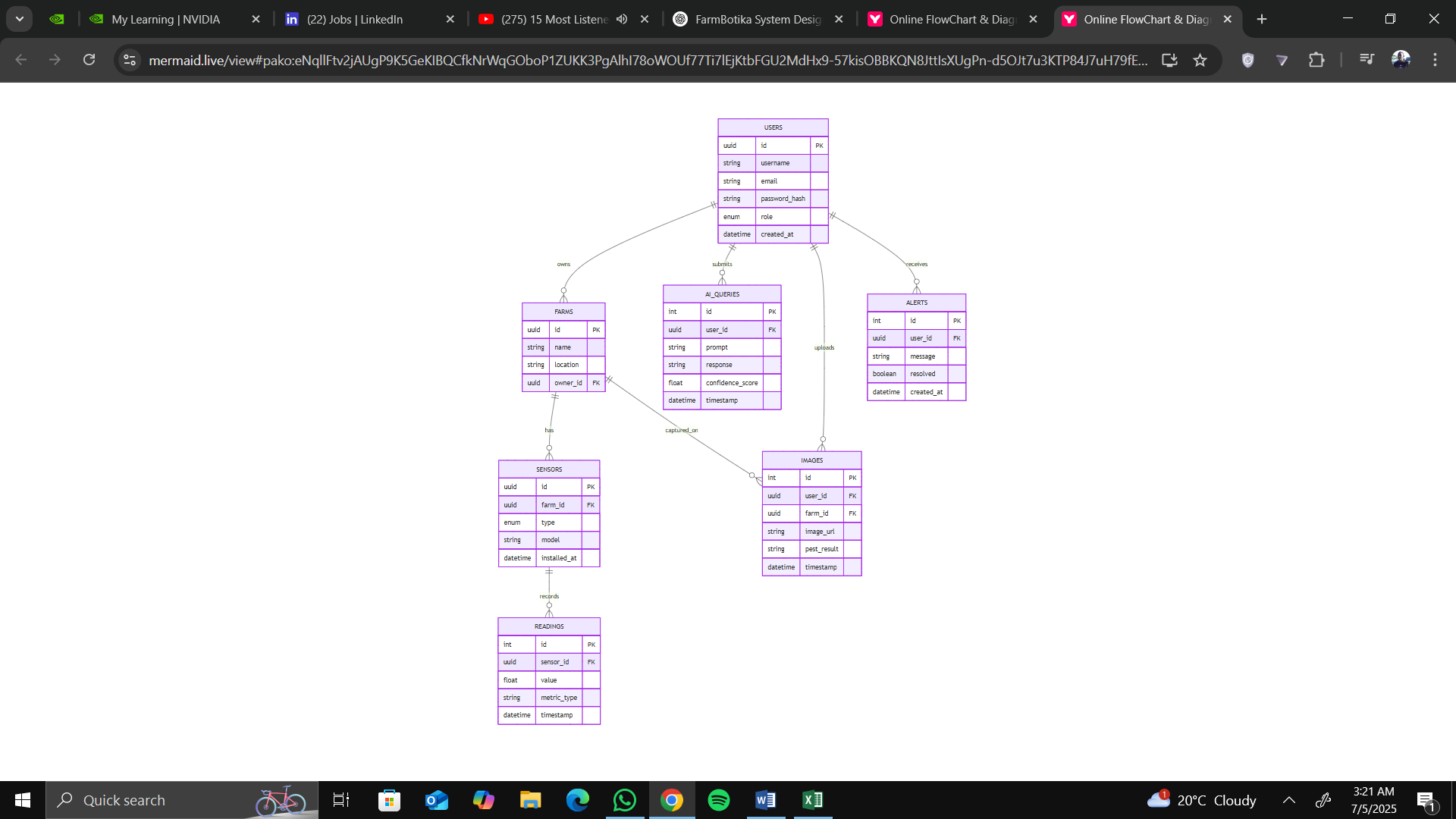
* Login --> Dashboard --> Farm Overview.
* Dashboard --> Chat --> View Recommendation
* Alerts --> Detail Page --> Acknowledge / Dismiss

**4.2.3 API Interface Specifications**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Endpoint | Method | Request Body | Response | Status Codes |
| /api/auth/login | POST | {email, password} | {token, user} | 200, 401 |
| /api/sensors/{id}/readings | POST | {timestamp, metrics} | {reading\_id} | 201, 400 |
| /api/ai/query | POST | {prompt, context} | {response, confidence} | 200, 429, 500 |

**5. Database Design**

**5.1 ER Diagram**



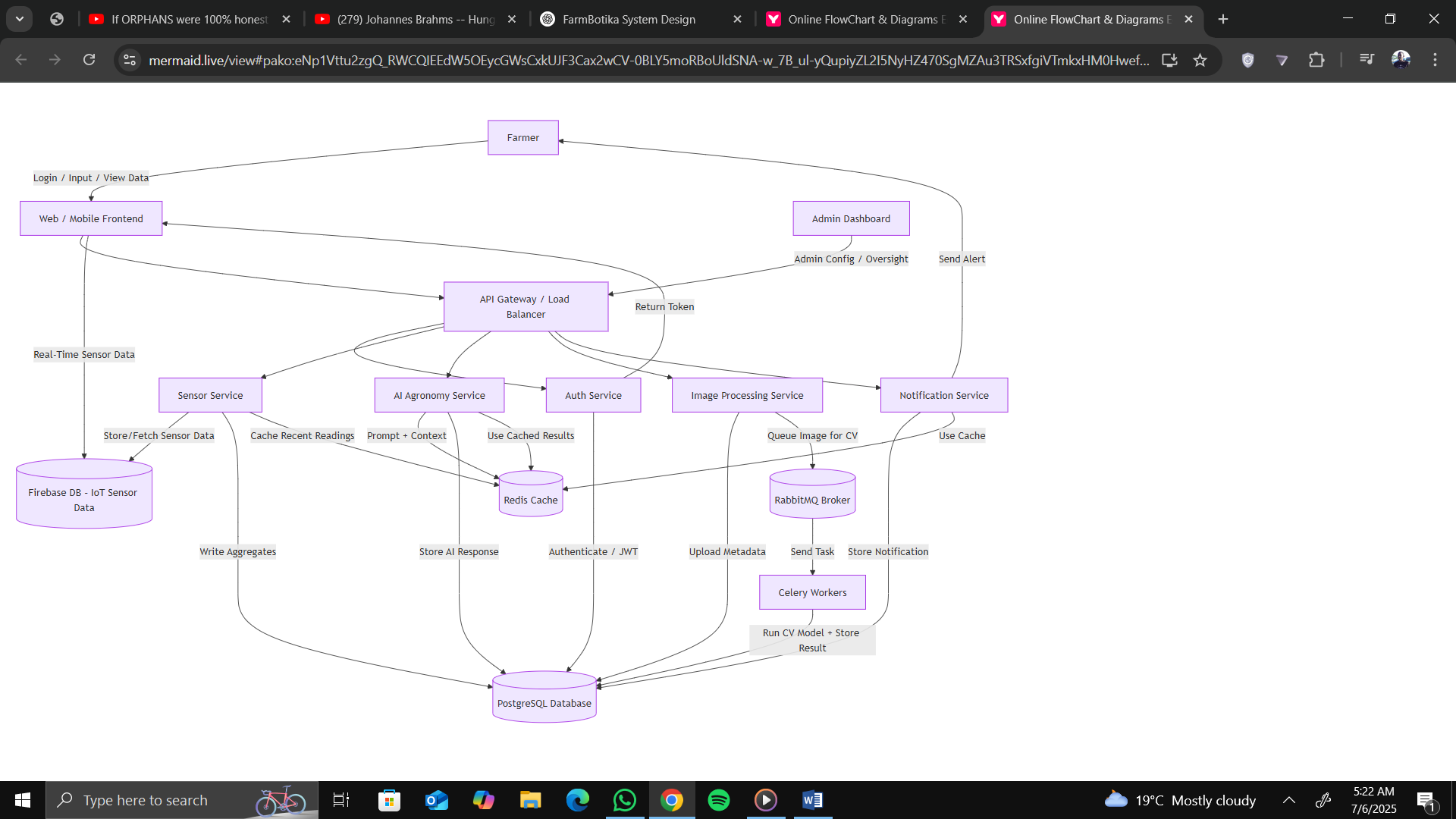
**5.2 Normalization**

This design follows **Third Normal Form (3NF)** to eliminate redundancy

**5.3 Data Dictionary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table** | **Field** | **Type** | **Null** | **Description** |
| **users** | id | UUID PK | No | Unique user identifier |
|  | email | VARCHAR(255) | No | User email |
|  | password\_hash | VARCHAR(255) | No | BCrypt hash |
|  | role | ENUM | No | admin, staff, farmer |
| **sensors** | id | UUID PK | No | Unique sensor identifier |
|  | farm\_id | UUID FK | No | Associated farm |
|  | type | VARCHAR(50) | No | e.g., moisture, temperature |
| **readings** | id | SERIAL PK | No |  |
|  | sensor\_id | UUID FK | No |  |
|  | value | FLOAT | No |  |
|  | timestamp | TIMESTAMP | No |  |

**6. Data Flow Diagram**

****

**7. Non-Functional Design Considerations**

**(a.) Performance Optimization:**

* Redis Caching for recent reading and AI Responses.
* Database indexing on sensor\_id, timestamp.

**(b.) Security Design:**

* Authentication with JWT; token expiry in 1 hour; refresh tokens via /auth/refresh.
* HTTP enforced.
* Encryption of sensitive fields at rest.

**(c.) Scalability Plans:**

* Dockerized services deployed in Docker.

**(d.) Availability and Fault Tolerance:**

* Retry logic in Celery tasks.

**(e.) Usability and Accessibility:**

* Responsive design for mobile network constraints.

**NB: More sections coming soon!!**