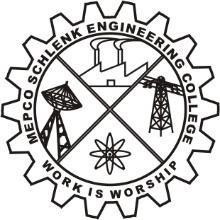
## URBAN GRIEVANCE PORTAL

****

**MINI PROJECT**

***Submitted by***

**K.N.SANJAY(9517202309101)**

**M.S.SUKIL RAJ(9517202309113)**

**K.PRABAKARAN(9517202309085)**

***in***

###### 23AD503 –PRINCIPLES OF SOFTWARE ENGINEERING

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

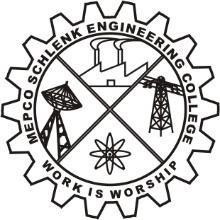
**MEPCO SCHLENK ENGINEERING COLLEGE SIVAKASI**

**NOVEMBER 2025**

## MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

**AUTONOMOUS**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

****

## BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **K N** **SANJAY (9517202309101), M S SUKIL RAJ (9517202309113), K PRABAKARAN (9517202309085)** for the mini project titled **“URBAN GRIEVANCE PORTAL”** in 23AD503 – **PRINCIPLES OF SOFTWARE**

#### **ENGINEERING** during the fifth semester July 2025 – December 2025 under my supervision.

**SIGNATURE SIGNATURE**

Mrs.A.Antony Shyamalin, BE, ME, MS, (PhD) Dr. J. Angela Jennifa Sujana, M.E.,Ph.D

**Assistant Professor, Professor & Head,**

AI&DS Department, AI&DS Department,

Mepco Schlenk Engg. College, Sivakasi Mepco Schlenk Engg. College, Sivakasi.

**ACKNOWLEDGEMENT**

First and foremost we praise and thank **“The Almighty”,** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa SujanaM.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Dr.P.Thendral BE, ME, PhD** Associate Professor, Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Mrs.A.Antony Shyamalin, BE, ME, MS, (PhD)** Assistant Professor of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

**ABSTRACT**

Smart Traffic Automation is a web-based system that enables proactive road usage scheduling and route optimization. Users reserve road usage slots; the system computes optimal paths using Dijkstra’s algorithm while incorporating dynamic traffic conditions, booking-based occupancy and road availability; and an ML component provides congestion prediction and recommended speeds. By smoothing vehicle entry times and suggesting speeds commensurate with predicted congestion, the system seeks to reduce stop-and-go traffic and localized overloads at bottlenecks. This report documents requirements (SRS), high- and low-level architecture, data models, algorithms, testing strategy and results, project metrics and estimations, deployment and operational considerations, ethical and privacy concerns, and an appendix with implementation notes and developer instructions. The prototype is implemented using a React frontend, Node/Express backend with MongoDB, and a Python ML predictor; design choices prioritize modularity, testability and incremental improvements.

# Table of Contents

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE**  **NO** |
|  | **ABSTRACT** | **4** |
| **1** | **INTRODUCTION** | **7** |
|  | **1.1 Agile Methodology for Urban Grievance Project** | **7** |
|  | 1.1.1 Model Chosen | **7** |
|  | 1.1.2 Reason | **7** |
| **2** | **SOFTWARE REQUIREMENT SPECIFICATION** | **9** |
|  | **2.1 Purpose** | **10** |
|  | 2.1.2 Document Conventions | **10** |
|  | 2.1.3 Intended Audience And Reading Suggestions | **10** |
|  | 2.1.4 Product Scope | **11** |
|  | 2.1.5 References | **11** |
|  | **2.2Overall Description** | **11** |
|  | 2.2.1 Product Perspective | **11** |
|  | 2.2.2 Product Functions | **12** |
|  | 2.2.3 User Classes and Characteristics | **12** |
|  | 2.2.4 Operating Environment | **12** |
|  | 2.2.5 Design and Implementation Constraints | **12** |
|  | 2.2.6 User Documentation | **13** |
|  | 2.2.7 Assumptions and Dependencies | **13** |
|  | **2.3 External Interface Requirements** | **13** |
|  | 2.3.1 User Interfaces | **13** |
|  | 2.3.2 Hardware Interfaces | **13** |
|  | 2.3.3 Software Interfaces | **13** |
|  | 2.3.4 Communication Interfaces | **14** |
|  | **2.4 System Features** | **14** |
|  | 2.4.1 Grievance Submission & Tracking | **14** |
|  | 2.4.2 Ticket Assignment & Resolution | **14** |
|  | **2.5 Other Nonfunctional Requirements** | **15** |
|  | 2.5.1 Performance Requirements | **15** |
|  | 2.5.2 Safety Requirements | **15** |
|  | 2.5.3 Security Requirements | **15** |
|  | 2.5.4 Software Quality Attributes | **16** |
|  | 2.5.5 Business Rules | **16** |
|  | **2.6 Other Requirements** | **16** |
| **3** | **SOFTWARE DESIGN DOCUMENT** | **18** |
|  | **3.1 Requirements** | **18** |
|  | **3.2 System Architecture** | **19** |
|  | **3.3 Data Dictionary** | **20** |

|  |  |  |
| --- | --- | --- |
|  | **3.4 Analysis Model** | **21** |
|  | 3.4.1 Usecase Diagram | **21** |
|  | 3.4.2 Activity Diagram | **22** |
|  | 3.4.3 Swimlane Diagram | **25** |
|  | 3.4.4 Sequence Diagram | **27** |
|  | 3.4.5 ER-Diagram | **28** |
|  | 3.4.6 Class Diagram | **29** |
|  | 3.4.7 Data Flow Diagram | **30** |
| **4** | **TESTING AND INTERGRATION** | **33** |
|  | **4.1 Testing Strategy** | **33** |
|  | **4.2 Cyclomatic Complexity** | **34** |
|  | **4.3 Applicability of Blackbox and Whitebox Testing** | **35** |
|  | **4.4 Blackbox Cases** | **36** |
| **5** | **PROJECT METRICS** | **37** |
|  | **5.1 Effort Estimation Using COCOMO Model** | **37** |
|  | **5.2 Functional Point(FP) Estimation** | **38** |
| **6** | **OUTPUT** | **40** |
| **7** | **CONCLUSION** | **165** |
| **8** | **REFERENCES** | **166** |
|  | **APPENDIX-IMPLEMENATATION** | **45** |

**CHAPTER 1**

# Introduction

## Agile Methodology for Urban Grievance Project

### Model Chosen

For the development of the urban grievance project, we adopted the Agile methodology, specifically implementing the Scrum framework. Agile is a flexible, iterative approach to project management and software development that emphasizes collaboration, adaptability, and delivering value to users through incremental progress.

### Reason

We selected Agile for several reasons that align with the needs of this project:

* + - * **Iterative Development**: The project required building a system with multiple features, such as grievance submission, assignment to field agents, resolution tracking, and report generation. Agile’s iterative nature allowed us to develop and test these features in short cycles (sprints), ensuring steady progress and the ability to refine functionality as needed.
      * **User-Centric Design**: The system serves diverse stakeholders, including public users, municipal officers, and field agents, each with unique needs. Agile’s focus on user stories and continuous feedback enabled us to prioritize features that deliver the most value to users, such as a streamlined grievance submission process and real-time status tracking.
      * **Adaptability to Change**: Given the complexity of managing urban grievances, requirements were expected to evolve. Agile provided the flexibility to accommodate changes, such as adding new reporting capabilities or modifying workflows, without derailing the project timeline.
      * **Collaboration and Communication**: The project involved coordination between frontend (React) and backend (Spring Boot) development teams. Agile’s emphasis on daily standups, sprint planning, and retrospectives fostered effective communication, ensuring the team remained aligned on goals and challenges.
      * **Technology Alignment**: Spring Boot and React are well-suited for rapid development and iteration. Agile’s short development cycles complemented this tech stack, allowing us to quickly build RESTful APIs with Spring Boot and create dynamic, user-friendly interfaces with React.

The Agile methodology, through the Scrum framework, aligned seamlessly with the urban grievance project’s goals and development process:

* + - * **Incremental Feature Delivery**: Using Scrum’s sprint structure, we broke the project into manageable features, such as user authentication, grievance submission, assignment workflows, and analytics reporting. Each sprint (typically two weeks) resulted in a working increment of the system, allowing us to test and validate functionality early and often.
      * **Prioritization of User Needs**: By creating user stories for each stakeholder (e.g., “As a public user, I want to file a grievance so that my issue can be addressed”), we ensured that development focused on delivering value. Regular sprint reviews with stakeholders allowed us to gather feedback and adjust priorities, ensuring the system met user expectations.
      * **Continuous Improvement**: Agile’s retrospective meetings at the end of each sprint enabled the team to reflect on what went well and what could be improved. This led to process enhancements, such as optimizing API performance in Spring Boot or improving the React frontend’s responsiveness, resulting in a more robust system.
      * **Enhanced Reporting Capabilities**: The project required generating reports to analyze grievance trends. Agile allowed us to start with basic reporting features and iteratively enhance them based on stakeholder feedback, ensuring the analytics met evolving needs.
      * **Team Coordination**: Scrum’s daily standups kept the team synchronized, especially important given the integration between the React frontend and Spring Boot backend. This ensured smooth data flow between the user interface and the server, creating a cohesive user experience.

In conclusion, adopting the Agile methodology with Scrum enabled us to deliver a functional, user- focused urban grievance system efficiently. It supported iterative development, adaptability, and collaboration, ensuring the project met its objectives while remaining flexible to changing requirements.

**Software Requirements**

**Specification**

**for**

**Urban Grievance Portal**

### Version 1.0 approved

**Prepared by Development Team**

**I.ARCHANA**

**B.MAHALAKSHMI**

**S.AKSHAYAA SIVA SHANKARI**

**CHAPTER 2**

**SOFTWARE REQUIREMENT SPECIFICATION**

* 1. **Purpose**

The **Urban Grievance Portal** is a digital platform that allows citizens to report grievances related to urban infrastructure and public services. The system facilitates efficient tracking, assignment, and resolution of complaints by municipal authorities, improving transparency and accountability. This document outlines the requirements for version 1.0 of the system.

### Document Conventions

This document follows standard **IEEE SRS** formatting guidelines. Key conventions include:

* + - * **Bold text** for section headings
      * *Italic text* for emphasis
      * Monospace text for code or commands
      * Numbered and bulleted lists for clarity
      * Priority levels explicitly mentioned for each requirement

### Intended Audience and Reading Suggestions

The primary audience for this document includes:

* + - * **Developers:** Responsible for system implementation
      * **Project Managers:** Oversee project execution
      * **Municipality Officers:** End-users managing grievances
      * **Testers:** Verify system functionality and compliance
      * **System Administrators:** Maintain and support the system Suggested reading sequence:

1. **Introduction** – Overview of the system
2. **Overall Description** – Understanding functionality
3. **System Features** – Detailed feature breakdown
4. **External Interface Requirements** – Understanding integrations
5. **Nonfunctional Requirements** – Performance and security aspects

### Product Scope

The **Urban Grievance Portal** aims to:

* + - * Enable citizens to **raise complaints** about municipal issues
      * Allow municipality officers to **review, assign, and track grievances**
      * Provide a structured workflow for **field agents** to handle tasks
      * Implement an **escalation mechanism** for unresolved complaints
      * Generate **reports** for municipal analysis and performance evaluation

This system aligns with smart city initiatives by **enhancing urban governance** and **improving citizen satisfaction**.

### References

* + - * **IEEE SRS Standard** (ISO/IEC/IEEE 29148:2018)
      * Municipality Complaint Resolution Guidelines
      * System Design Documents (Use Case, Sequence, and DFD Diagrams)

## Overall Description

### Product Perspective

The **Urban Grievance Portal** is a standalone web-based application designed to digitize and streamline municipal grievance redressal processes. It replaces traditional paper-based and telephone-based complaint mechanisms with an automated, efficient system. The portal interacts with:

* + - * **Users (Citizens)** for complaint registration and tracking
      * **Municipality Officers** for reviewing and assigning complaints
      * **Field Agents** for resolving complaints
      * **External Specialists** when additional expertise is needed

### Product Functions

The core functions of the system include:

* + - * **User Registration & Login**
      * **Complaint Submission & Tracking**
      * **Ticket Assignment & Resolution Updates**
      * **Escalation & External Assistance Handling**
      * **Automated Notifications & Alerts**
      * **Performance Reports & Analytics**

### User Classes and Characteristics

* + - * **Citizens:** End-users who submit grievances; minimal technical expertise required.
      * **Municipality Officers:** Responsible for reviewing and assigning complaints; require administrative access.
      * **Field Service Agents:** Technical personnel who resolve complaints.
      * **External Specialists:** Third-party experts consulted for escalated complaints.

### Operating Environment

* + - * **Platform:** Web-based, accessible via desktop and mobile browsers.
      * **Operating Systems:** Windows, macOS, Linux, Android, iOS.
      * **Technologies Used:** Cloud-based infrastructure, RESTful APIs for integration.

### Design and Implementation Constraints

* + - * Must comply with **government data privacy regulations**.
      * Should handle **high concurrent user traffic**.
      * Should provide **role-based access control**.
      * Integration with **external APIs** for reporting and escalations.

### User Documentation

* + - * **User Guide** for citizens to submit and track complaints.
      * **Administrator Manual** for managing system configurations.
      * **Technical Documentation** for developers maintaining the system.

### Assumptions and Dependencies

* + - * Users have access to **internet-enabled devices**.
      * Municipality officers will actively **monitor and resolve complaints**.
      * External specialists are available for **escalation cases**.
      * The system will be deployed in a **secured cloud environment**.

# External Interface Requirements

### User Interfaces

The system provides a **web-based** user interface accessible via desktop and mobile devices. It features:

* + - * **Dashboard for Users:** Submit and track grievances.
      * **Dashboard for Municipality Officers:** Review, assign, and escalate complaints.
      * **Dashboard for Field Agents:** Update complaint status and resolution progress.
      * **Notifications:** Real-time updates via email/SMS.
      * **Help & Support:** In-app FAQs and contact options.

### Hardware Interfaces

* + - * **Supported Devices:** Desktop computers, smartphones, tablets.
      * **Connectivity:** Works over the internet via browsers.
      * **Server Infrastructure:** Cloud-hosted with backup storage.

### Software Interfaces

* + - * **Operating Systems:** Compatible with Windows, macOS, Linux, Android, iOS.
      * **Databases:** MongoDB for structured complaint storage.
      * **APIs:** RESTful APIs for third-party integrations.
      * **Authentication:** OAuth-based user authentication.

### Communications Interfaces

* + - * **Web Browsing:** HTTPS for secure access.
      * **Email/SMS:** Notifications for complaint updates.
      * **Data Exchange:** JSON-based communication between client and server.

# System Features

### Grievance Submission & Tracking

* + - 1. ***Description and Priority***
         * **High Priority**: Users can submit complaints with images, location, and description.
      2. ***Stimulus/Response Sequences***

1. User submits a complaint.
2. System generates a ticket ID.
3. Municipality officer assigns it to a field agent.
4. User receives updates until resolved.
   * + 1. ***Functional Requirements***
          - REQ-1: Users can submit complaints with necessary details.
          - REQ-2: The system generates unique ticket IDs.

### Ticket Assignment & Resolution

* + - 1. ***Description and Priority***
         * **High Priority**: Officers assign complaints to field agents for resolution.
      2. ***Stimulus/Response Sequences***

1. Officer assigns a complaint.
2. Field agent updates status in real-time.
3. If unresolved in 48 hours, the system escalates it.
   * + 1. ***Functional Requirements***
          - REQ-3: Officers can assign complaints.
          - REQ-4: Escalation occurs if complaints remain unresolved.

# Other Nonfunctional Requirements

### Performance Requirements

* The system must support at least **5000 concurrent users**.
* The response time for complaint submission should not exceed **3 seconds**.
* Ticket status updates should be reflected in real-time with a **delay of no more than 5 seconds**.
* The portal must be available **24/7 with 99.9% uptime**.

### Safety Requirements

* The system must ensure **secure access control** to prevent unauthorized modifications.
* Regular **data backups** should be conducted **daily** to prevent loss of critical complaint records.
* The system should provide **role-based access restrictions** to prevent unauthorized data exposure.
* Disaster recovery mechanisms must be in place to restore operations within **30 minutes** of a system failure.

### Security Requirements

* All **user authentication** must be handled via **OAuth 2.0** or multi-factor authentication (MFA).
* Data transmission should be **encrypted using TLS 1.3** to prevent data interception.
* User passwords must be stored using **bcrypt hashing** with a high iteration count.
* Complaint data should only be accessible to authorized personnel.
* The system must log all activities for **audit tracking and compliance verification**.

### Software Quality Attributes

* **Availability:** The system should maintain an uptime of **99.9%**.
* **Scalability:** The platform must support an increasing number of complaints without performance degradation.
* **Usability:** The interface must be designed with **accessibility standards (WCAG 2.1)** for ease of use.
* **Maintainability:** The system should be modular and allow **future enhancements without major rewrites**.
* **Portability:** The portal should be accessible on **desktop, tablets, and mobile devices**.
* **Interoperability:** The system should allow **API integration with third-party applications**.

### Business Rules

* Only **registered users** can submit complaints.
* Municipality officers **must review new complaints within 24 hours**.
* If a complaint is unresolved for **48 hours**, it must be **automatically escalated**.
* Citizens can provide **feedback on resolved complaints**, and unresolved cases can be reopened.
* Only **authorized personnel** can modify complaint status and assign tasks.
* **Data privacy laws and local regulations** must be adhered to in data handling.

# Other Requirements

* Compliance with **local government regulations**.
* Multi-language support for better accessibility.

# Appendix A: Glossary

* **UGP (Urban Grievance Portal):** The web-based application for grievance management.
* **User:** A citizen who submits a complaint.
* **Municipality Officer:** A government official responsible for assigning and monitoring grievances.
* **Field Service Agent:** A personnel assigned to resolve complaints.
* **External Assistance:** Third-party specialists handling escalated grievances.
* **Ticket:** A unique identifier assigned to each grievance submitted by users.
* **API (Application Programming Interface):** A set of functions allowing the system to communicate with other applications.
* **OAuth:** A secure authentication protocol used for user login and verification.
* **HTTPS (Hypertext Transfer Protocol Secure):** A protocol for secure communication over a computer network.
* **JSON (JavaScript Object Notation):** A lightweight data-interchange format used for communication between system components.

## CHAPTER 3 SOFTWARE DESIGN DOCUMENT

* 1. **Requirements**

Your mileage may vary -- we typically break down the requirements to provide a ballpark estimate.

* + 1. **Estimates**

|  |  |  |
| --- | --- | --- |
| **#** | **Description** | **Hrs. Est.** |
| **1** | User Registration and Authentication | **10** |
| **2** | Grievance Submission Module | **20** |
| 3 | Ticket Review & Assignment Workflow | 25 |
| 4 | Status Tracking & Notifications | 15 |
| 5 | Admin Dashboard for Monitoring | 10 |
| 6 | Reporting & Analytics | 10 |
|  |  |  |
| **7** | External Assistance Integration | **10** |
| **8** | Database Setup & Optimization | **10** |
| **9** | Deployment & Testing | **20** |
|  | **TOTAL** **130** | |

* + 1. **Traceability Matrix**

The table below cross-references the SRS requirements with the corresponding SDD modules to ensure complete coverage

|  |  |
| --- | --- |
| **SRS Requirement** | **SDD Module** |
| Req 1- User Registration | 5.1.1 (Authentication Module) |
| Req 2 - Grievance Submission | 5.1.2 (Grievance Management) |
| Req 3 - Ticket Assignment | 5.1.3 (Workflow Engine) |
| Req 4 - Status Tracking | 5.1.4 (Notification System) |
| Req 5 - Escalation Handling | 5.1.5 (Escalation Module) |
| Req 6 - Reporting & Analytics | 5.1.6 (Reporting System) |
| Req 7 - External Assistance | 5.1.7 (Third-Party Integration) |
| Req 8 - Admin Dashboard | 5.1.8 (Admin Panel) |
| Req 9 - Database Design | 5.2.1 (Database Schema) |
| Req 10 - Deployment & Testing | 5.3.1 (Deployment Plan) |

### System Architecture

The system architecture of the Urban Grievance Portal follows a modular and layered design. It consists of the following components:

* **Frontend Layer:** User Interface built using React.js for accessibility and responsiveness.
* **Backend Layer:** RESTful API using Django/Flask for handling business logic.
* **Database Layer:** PostgreSQL/MySQL for data persistence.
* **Authentication:** OAuth 2.0-based user authentication using JWT.
* **Workflow Engine:** Manages grievance processing, status updates, and escalation rules.

### Data Dictionary

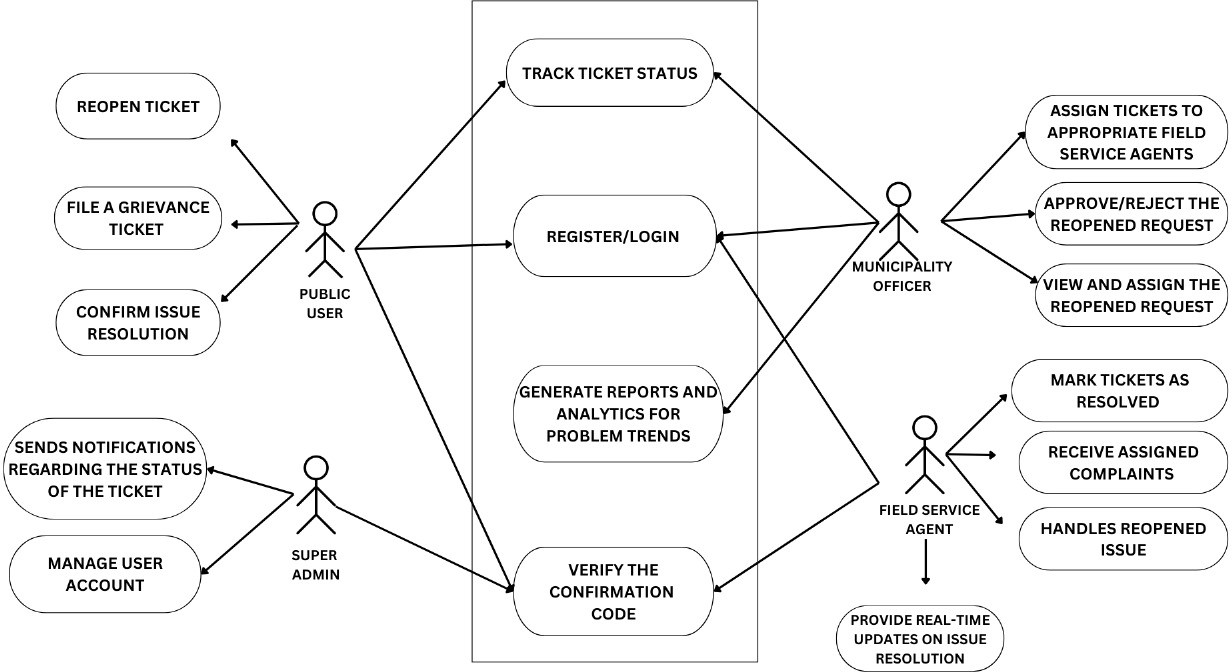
The data dictionary includes a brief description of each element used in this module.

**Table**

|  |  |  |
| --- | --- | --- |
| **Field** | **Notes** | **Type** |
| ID | Unique Identifier from TABLE\_SEQ | DECIMAL |
| NAME | The Name in Object.Name() | VARCHAR |
| VALUE | The Value output from somewhere | VARCHAR |
| EMAIL | User email for contact | VARCHAR |
| DESCRIPTION | Description of grievance | TEXT |
| STATUS | Current status of the problem | VARCHAR |
| CREATED\_AT | Timestamp of grievance created | TIMESTAMP |

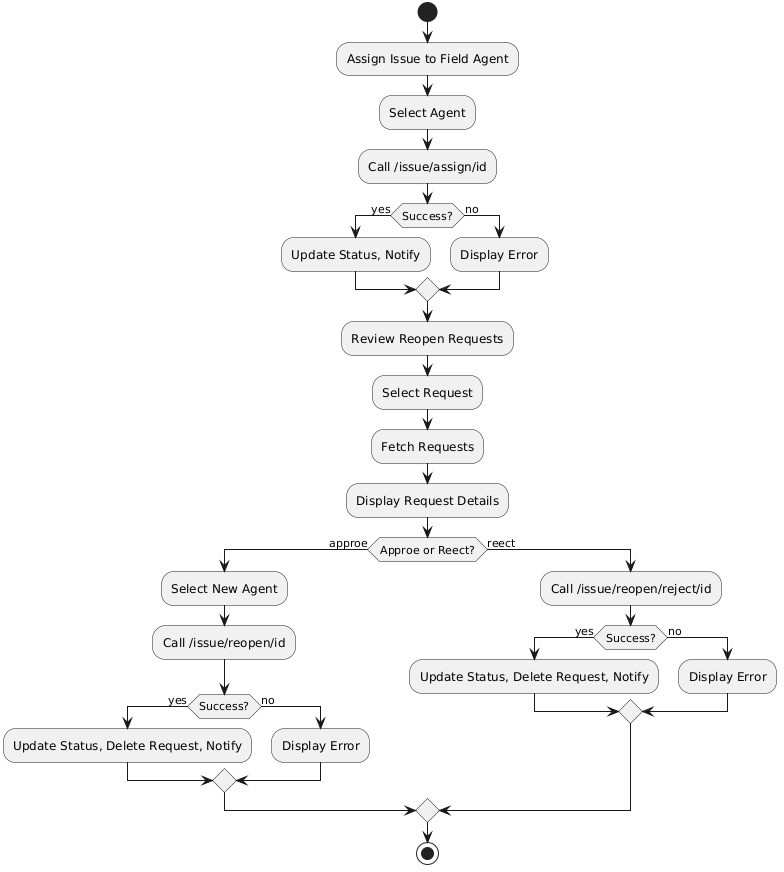
#### ANALYSIS MODELS

* + 1. USECASE DIAGRAM

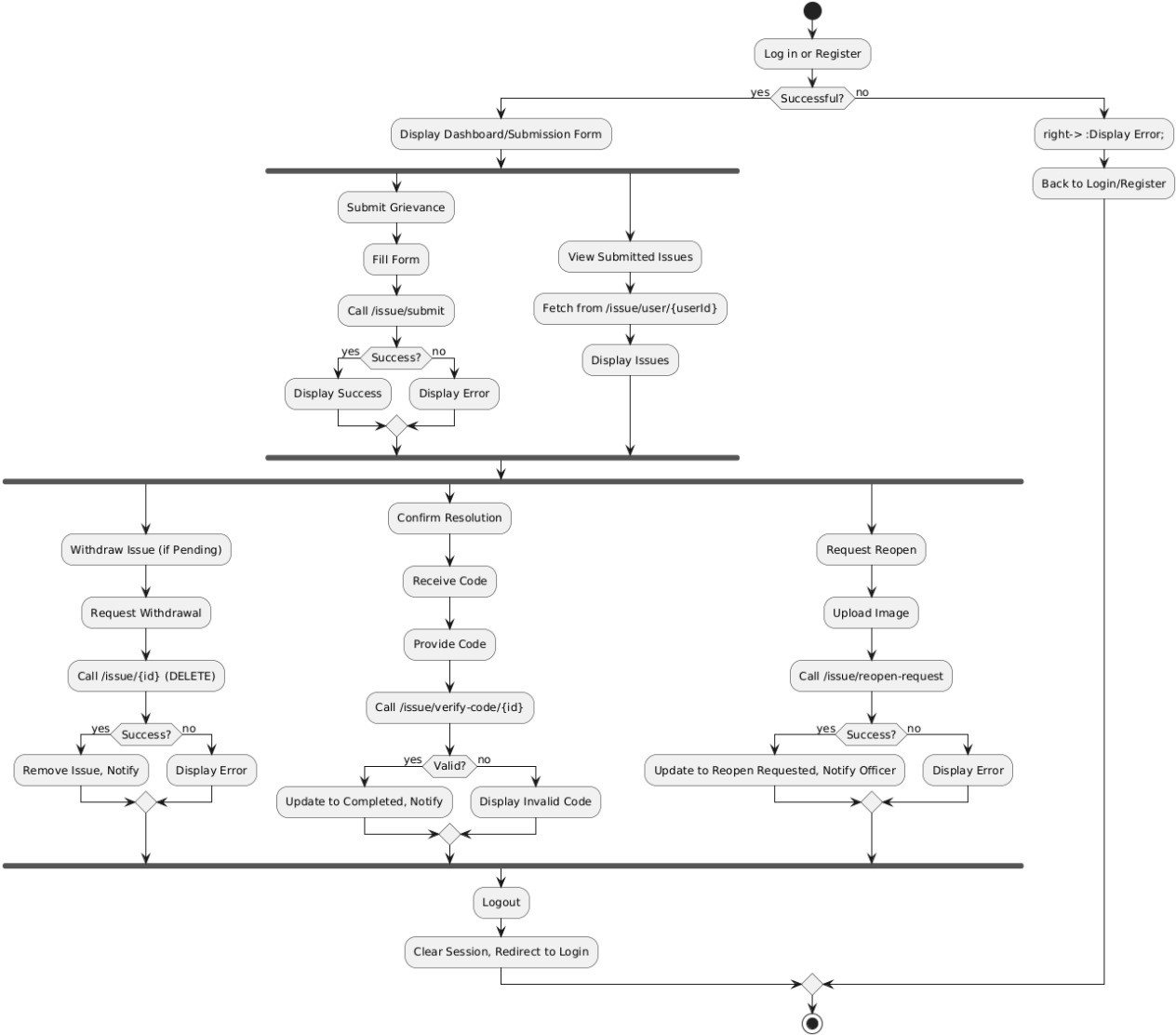


###### Figure No.3.4.1.UseCase Diagram of Urban Grievance Portal

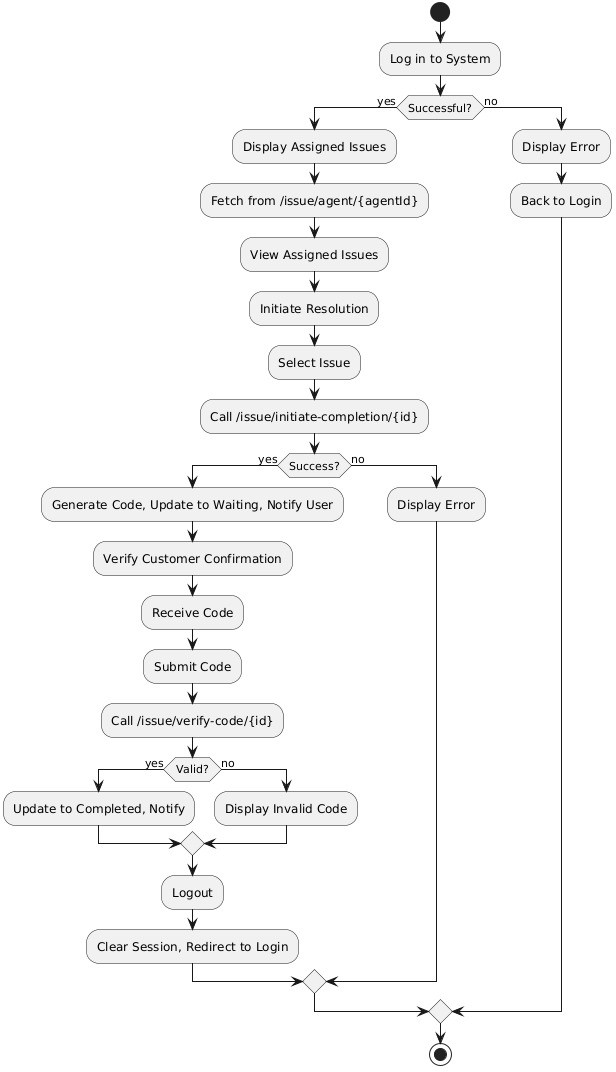
* + 1. ACTIVITY DIAGRAM



**Figure No.3.4.2.Activity Diagram of Municipal Officer**

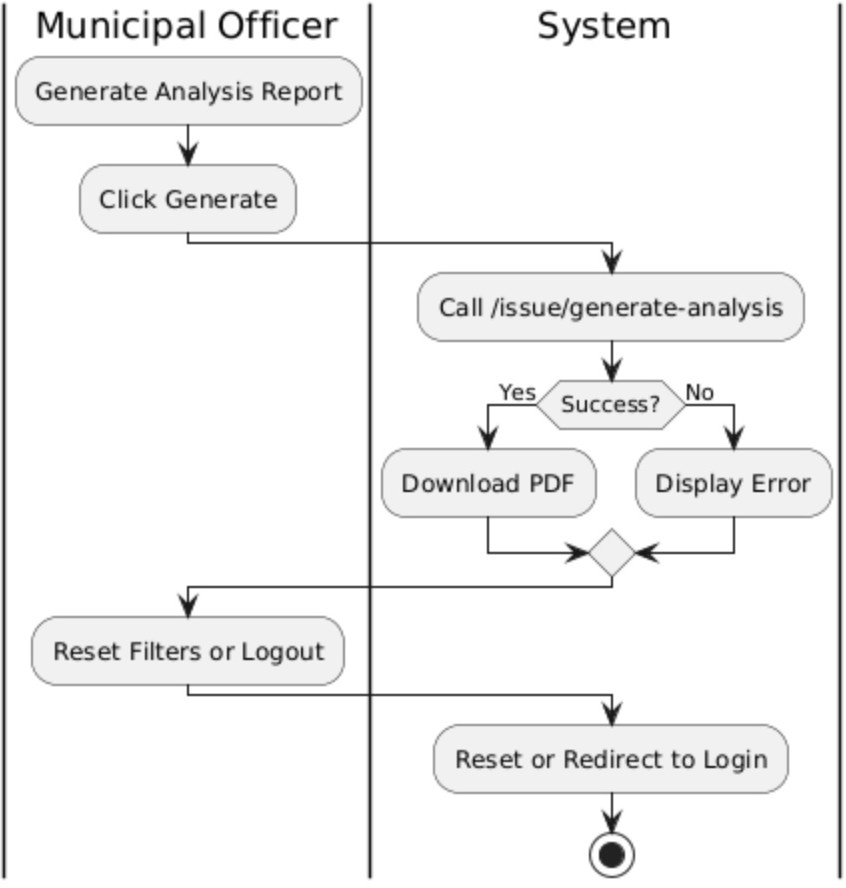


**Figure No.3.4.3.Activity Diagram of Public User**

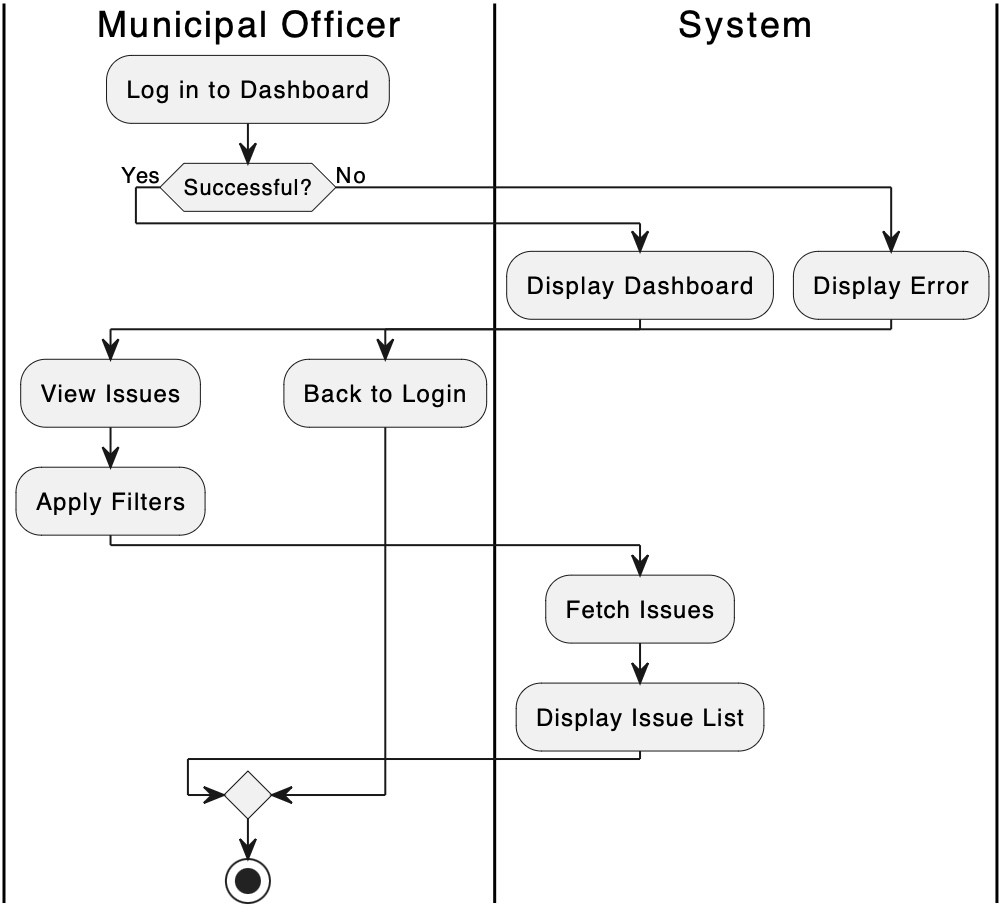


**Figure No.3.4.3.Activity Diagram of Field Service Agent**

* + 1. SWIMLANE

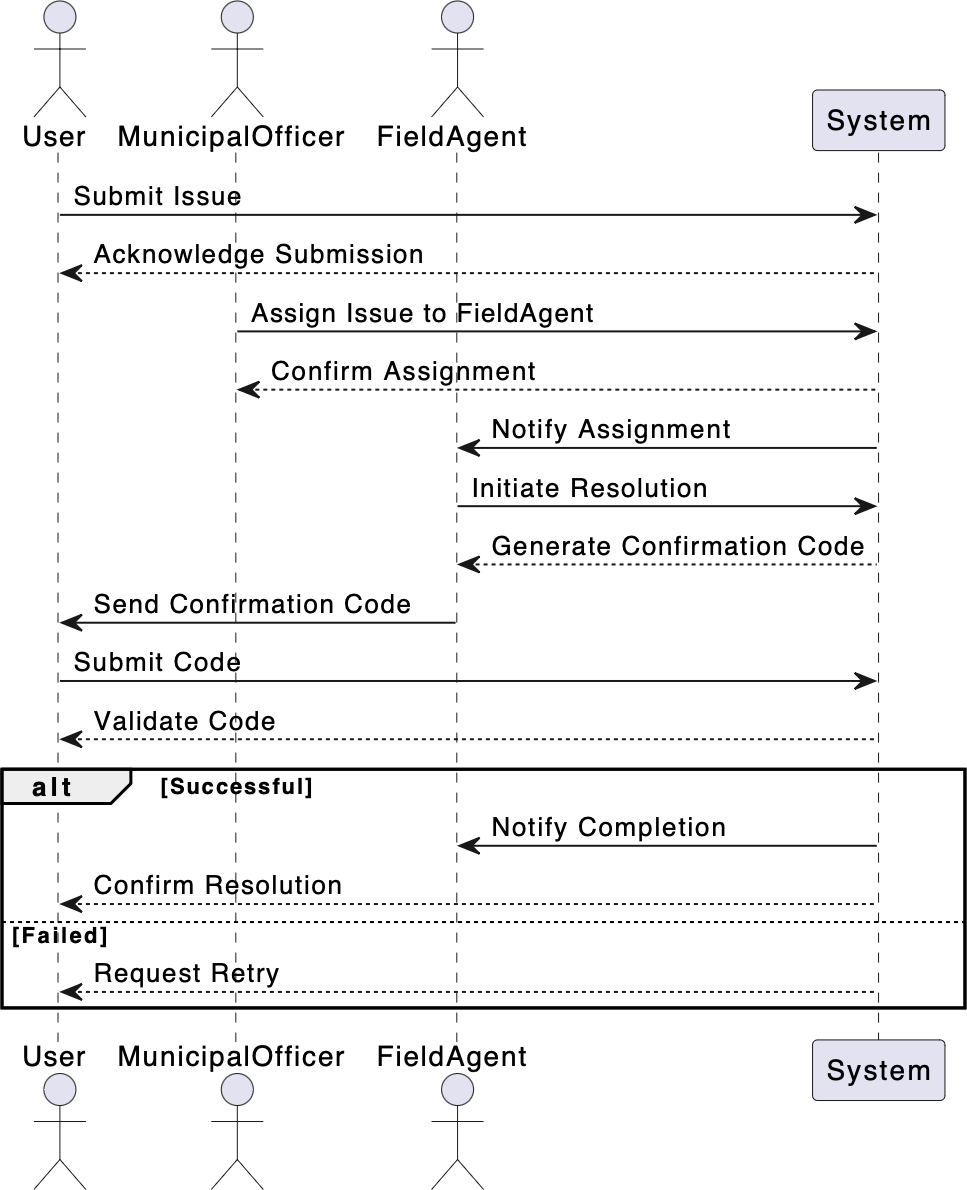


###### Figure No.3.4.4.Swinlane Diagram of Municipal Officer



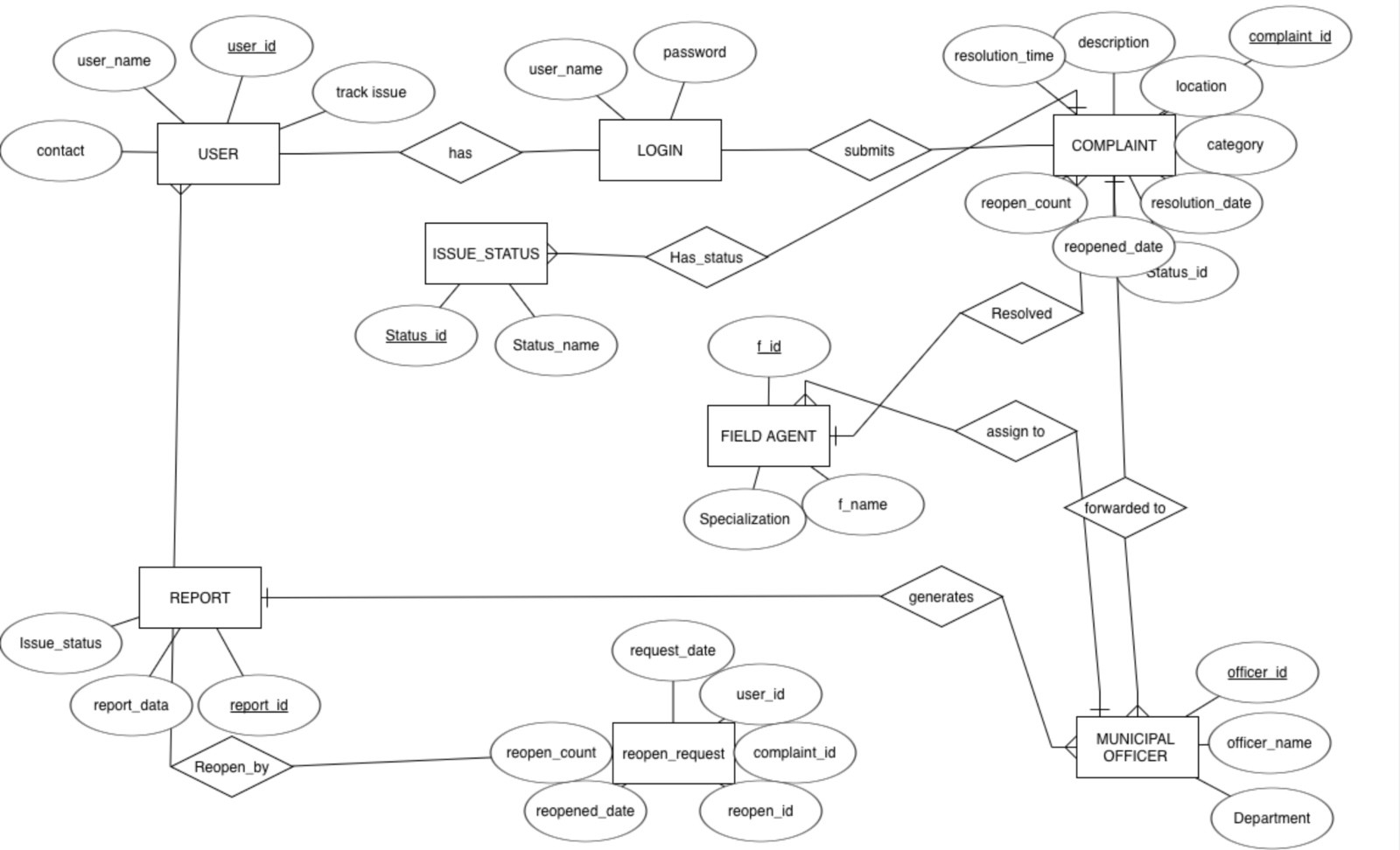
**Figure No.3.4.5.Swinlane Diagram of Municipal Officer**

* + 1. SEQUENCE-DIAGRAM



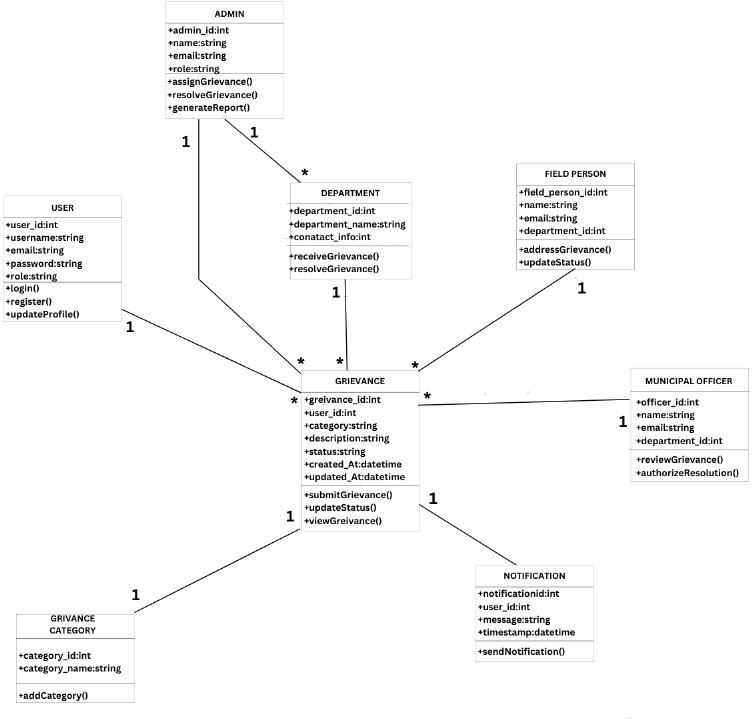
###### Figure No.3.4.6.Sequence Diagram Of Urban Grievance Portal

* + 1. ER-DIAGRAM



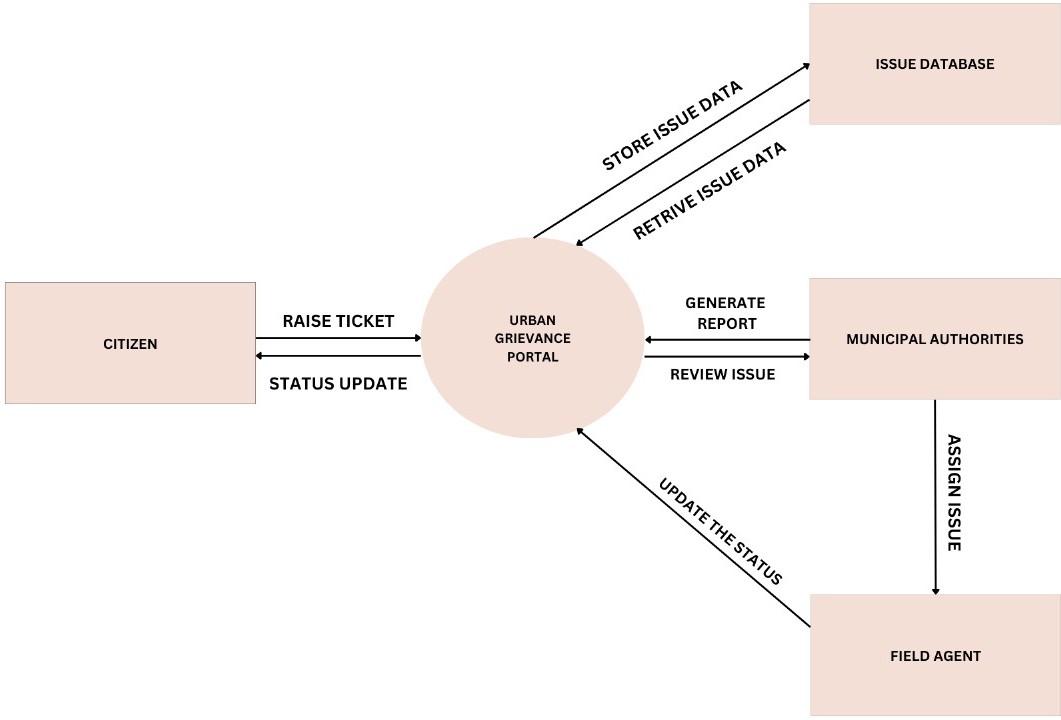
###### Figure No.3.4.7.ER Diagram Of Urban Grievance Porta

* + 1. CLASS DIAGRAM



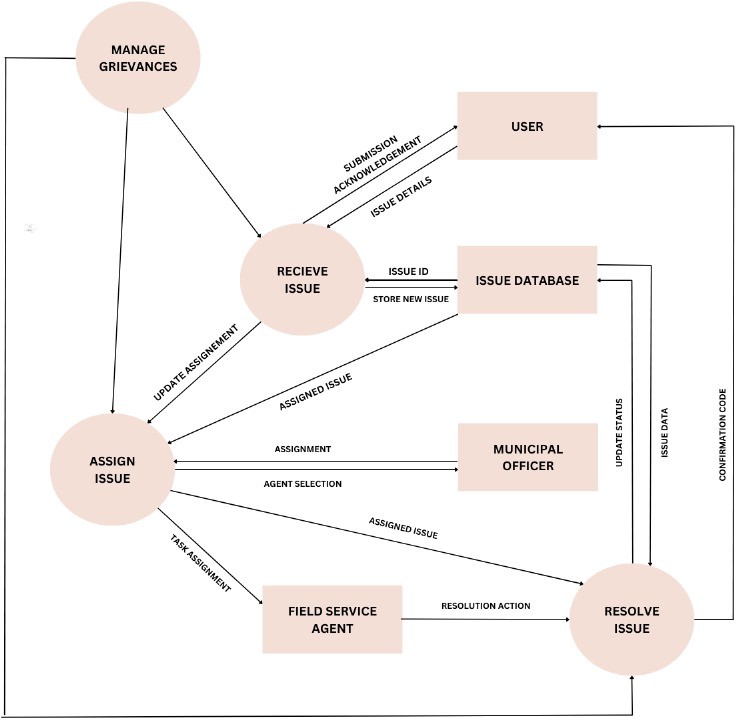
###### Figure No.3.4.8.Class Diagram Of Urban Grievance Portal

* + 1. DATA FLOW DIAGRAM
       1. DFD-LEVEL 0:



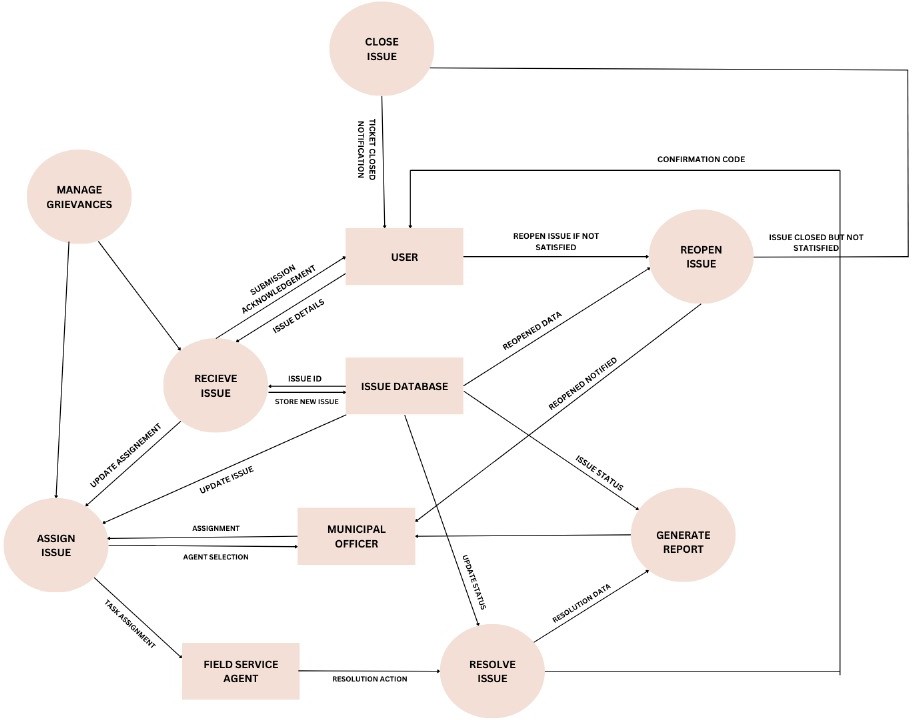
###### Figure No.3.4.9.DFD Level 0 Of Urban Grievance Portal

* + - 1. DFD LEVEL-1



###### Figure No.3.4.10.DFD Level 1 Of Urban Grievance Portal

* + - 1. DFD LEVEL-2



**Figure No.3.4.11.DFD Level 2 Of Urban Grievance Portal**

## CHAPTER 4 TESTING AND INTEGRATION

##### Testing Strategy

A comprehensive testing strategy ensured reliability and usability within the 3-month timeline.

###### Unit Testing

* + - * **Purpose**: Verified individual components.

###### Tools:

* + - * + **JUnit** and **Mockito** for Spring Boot: Tested service and repository layers (e.g., GrievanceService, GrievanceRepository).
        + **Jest** with **React Testing Library** for React: Tested UI components (e.g., grievance submission form).
      * **Coverage**: Achieved 75% code coverage, focusing on core features like grievance submission.
      * **Example**: Tested the submitGrievance API endpoint for database persistence.

###### Integration Testing

* + - * **Purpose**: Validated component interactions.

###### Tools:

* + - * + **Spring Boot Test** with an H2 database for API endpoints (e.g., /issue/submit).
        + **Cypress** for end-to-end React frontend testing.
      * **Focus**: Ensured seamless integration, such as grievance submission from React to Spring Boot.
      * **Example**: Tested the /issue/submit endpoint with React form data.

###### System Testing

* + - * **Purpose**: Confirmed overall functionality and performance.

###### Approach:

* + - * + Tested end-to-end workflows (e.g., submission to resolution).
        + Used **JMeter** to verify 200 concurrent users with a response time under 2 seconds.
      * **Example**: Simulated 50 users submitting grievances to ensure stability.

###### User Acceptance Testing (UAT)

* + - * **Purpose**: Ensured stakeholder satisfaction.

###### Approach:

* + - * + Conducted UAT with 5–10 users (public users, officers, agents) in the final week.
        + Focused on usability (e.g., intuitive tracking) and functionality (e.g., accurate reports).
      * **Feedback**: Implemented minor UI tweaks (e.g., button labels) pre-launch.

##### Cyclomatic Complexity Analysis

Cyclomatic complexity analysis was conducted to measure the complexity of the code and guide testing efforts within the 3-month timeline.

###### Purpose

* + - * Assessed the number of linearly independent paths in the codebase to identify complex modules requiring thorough testing.
      * Used to optimize unit and integration testing by focusing on high-complexity areas.

###### Methodology

* + - * **Calculation**: Cyclomatic complexity (V(G)) = E - N + 2P, where:
        + E = number of edges in the control flow graph
        + N = number of nodes
        + P = number of connected components (typically 1 for a single method)
      * **Tools**: Utilized **SonarQube** and **CodeClimate** to analyze Spring Boot services (e.g., GrievanceService) and React components.
      * **Threshold**: A complexity score above 10 was considered high, indicating a need for additional test cases.

###### Findings

* + - * **Spring Boot**: The submitGrievance method, with multiple conditional checks (e.g., input validation, status updates), had a complexity of 12, suggesting 12 test paths to cover all scenarios.
      * **React**: The grievance submission form component, with conditional rendering and state management, had a complexity of 8, manageable with existing Jest tests.
      * **Action**: Increased unit test coverage for high-complexity methods (e.g., added edge cases for submitGrievance) and monitored integration points to ensure stability.

###### Benefits

* + - * Identified critical code sections, reducing the risk of bugs in complex workflows.
      * Supported the 3-member team’s testing efforts by prioritizing resources on high-complexity areas within the tight timeline.

##### Applicability of Black-Box and White-Box Testing

###### Black-Box Testing

Black-box testing, focusing on inputs and outputs without code insight, was highly applicable:

###### Relevance:

* + - * + **User-Centric Features**: Validated functionalities like grievance submission and report generation from a user perspective, critical for public users, municipal officers, and field agents.
        + **UAT and System Testing**: Applied to ensure the system met requirements (e.g., submitting a grievance and viewing status) as experienced by stakeholders.
        + **API Testing**: Tested Spring Boot APIs (e.g., /issue/submit) by sending requests and verifying responses, focusing on functionality.
      * **Example**: In UAT, users submitted grievances via the React frontend and verified status updates, unaware of backend logic.

###### White-Box Testing

White-box testing, leveraging internal code knowledge, was equally applicable:

###### Relevance:

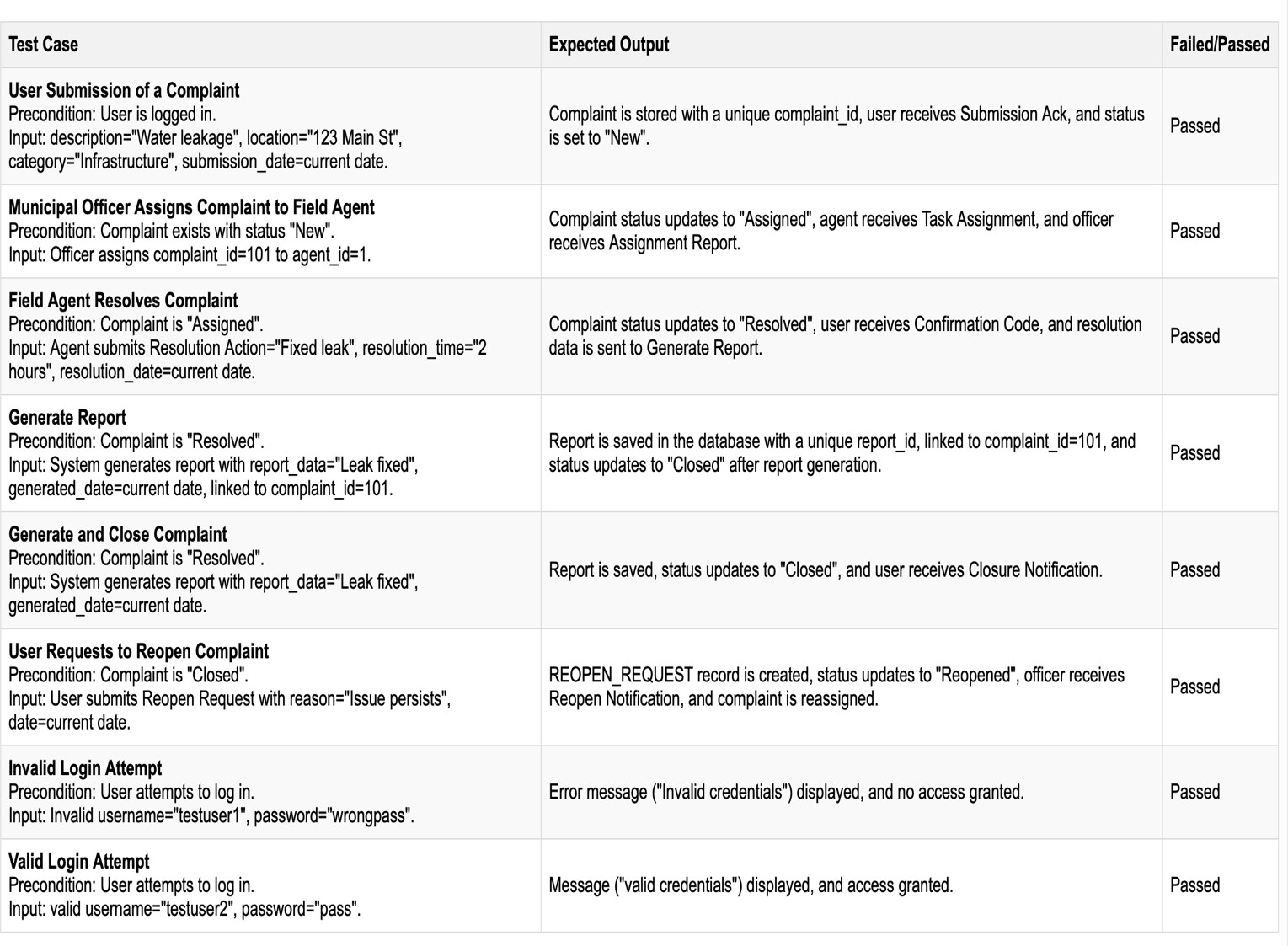
* + - * + **Unit Testing**: The team used white-box testing with JUnit and Jest to test Spring Boot services (e.g., if-else conditions in GrievanceService) and React components, ensuring code paths were covered.
        + **Integration Testing**: Applied to debug interactions between React and Spring Boot, such as ensuring fetch calls mapped correctly to APIs.
        + **Optimization**: Within the 3-month timeline, white-box testing identified performance bottlenecks, like optimizing database queries in Spring Boot.
      * **Example**: Wrote a JUnit test for the submitGrievance method to execute all error-handling branches, requiring knowledge of its logic.

###### Combined Approach

Both approaches complemented each other:

* + - * **Black-Box in UAT/System Testing**: Ensured functional and usability requirements from an external perspective.
      * **White-Box in Unit/Integration Testing**: Ensured code-level correctness and integration, vital for the Spring Boot-React architecture.
      * **Balance**: The tight 3-month schedule benefited from white-box testing for efficiency and black-box testing for user validation, ensuring a robust system.

##### Blackbox Cases

****

**Figure No.4.1.BlackBox Testing Of Urban Grievance Portal**

## CHAPTER 5 PROJECT METRICS

##### Effort Estimation Using COCOMO Model

The Constructive Cost Model (COCOMO) estimated effort, development time, and staffing for the urban grievance project.

###### Project Classification

Classified as a **Semi-Detached project** due to moderate complexity, involving user authentication, grievance submission, assignment workflows, resolution tracking, and analytics reporting for a distributed user base.

###### COCOMO Parameters

Using the Intermediate COCOMO model for a Semi-Detached project:

###### Effort (E) = a × (KLOC)^b × EAF

* + - * + a = 3.0 (coefficient for Semi-Detached)
        + b = 1.12 (exponent for Semi-Detached)
        + KLOC = 5
        + EAF (Effort Adjustment Factor) = 1.15 (moderate complexity, team cohesion, tool usage)

###### Development Time (D) = c × (E)^d

* + - * + c = 2.5
        + d = 0.35

###### Calculations

1. **Effort Calculation**:

o E = 3.0 × (5)^1.12 × 1.15

o E = 3.0 × 6.44 × 1.15 ≈ 22.23 person-months

###### Development Time:

o D = 2.5 × (22.23)^0.35

o D = 2.5 × 2.37 ≈ 5.93 months

###### Staffing:

* + Average staff = 22.23 / 5.93 ≈ 3.75 ≈ 4 people

###### Validation with Real Data

The estimate (5.93 months, 4 people) differs from the actual 3 months and 3-member team, indicating high productivity. Adjusting EAF to 0.9 to reflect efficiency:

* + - * E = 3.0 × 6.44 × 0.9 ≈ 17.39 person-months
      * D = 2.5 × (17.39)^0.35 ≈ 2.5 × 2.23 ≈ 5.58 months
      * Staffing = 17.39 / 5.58 ≈ 3.12 ≈ 3 people, aligning with reality.

The adjusted COCOMO estimate suggests **17.39 person-months**, **5.58 months**, and a team of **3 developers**, reflecting efficient execution within the 3-month timeline.

##### Function Points (FP) Estimation

Function Points (FP) analysis estimated the project’s size based on functionality.

###### FP Components

The system’s functionalities include:

1. **External Inputs (EI)**: 5 inputs (e.g., user login, grievance submission), rated as average complexity (4 FP each).

o 5 × 4 = 20 FP

1. **External Outputs (EO)**: 3 outputs (e.g., grievance status, analytics report), rated as average complexity (5 FP each).

o 3 × 5 = 15 FP

1. **External Inquiries (EQ)**: 4 inquiries (e.g., view assigned issues), rated as average complexity (4 FP each).

o 4 × 4 = 16 FP

1. **Internal Logical Files (ILF)**: 3 logical files (e.g., User, Grievance, Report databases), rated as average complexity (10 FP each).

o 3 × 10 = 30 FP

1. **External Interface Files (EIF)**: 1 interface (e.g., notification service), rated as simple complexity (5 FP).

o 1 × 5 = 5 FP

**Unadjusted Function Points (UFP)** = 20 + 15 + 16 + 30 + 5 = 86 FP

###### Complexity Adjustment

With a Value Adjustment Factor (VAF) score of 35 (moderate influence), VAF = 0.65 + (0.01 × 35)

= 1.0.

**Adjusted Function Points (FP)** = 86 × 1.0 = 86 FP

###### Effort Estimation Using FP

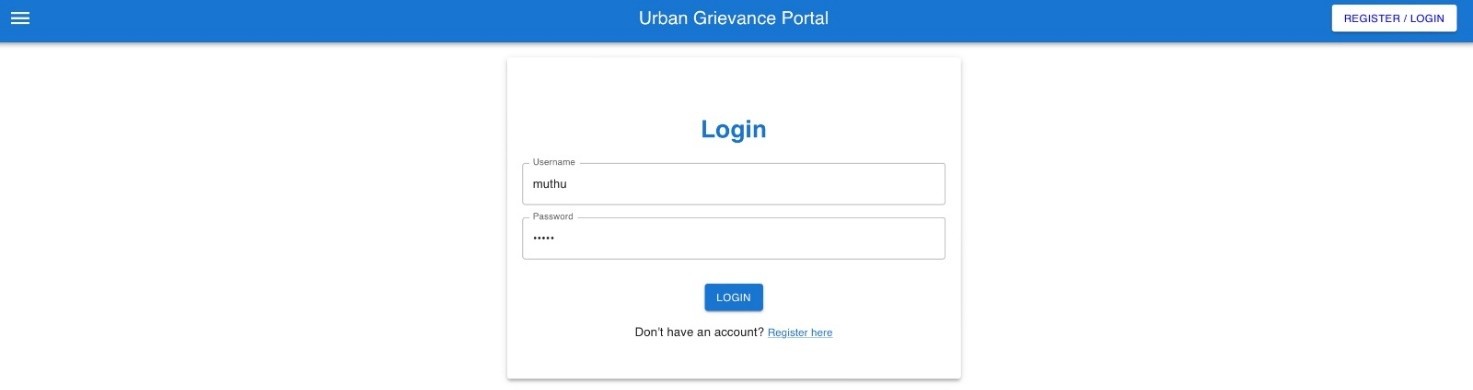
Assuming a productivity rate of 15 FP per person-month (adjusted for the 3-month completion):

* + - * **Effort** = 86 FP / 15 FP per person-month ≈ 5.73 person-months
      * **Development Time**: With 3 developers, Time = 5.73 / 3 ≈ 1.91 months

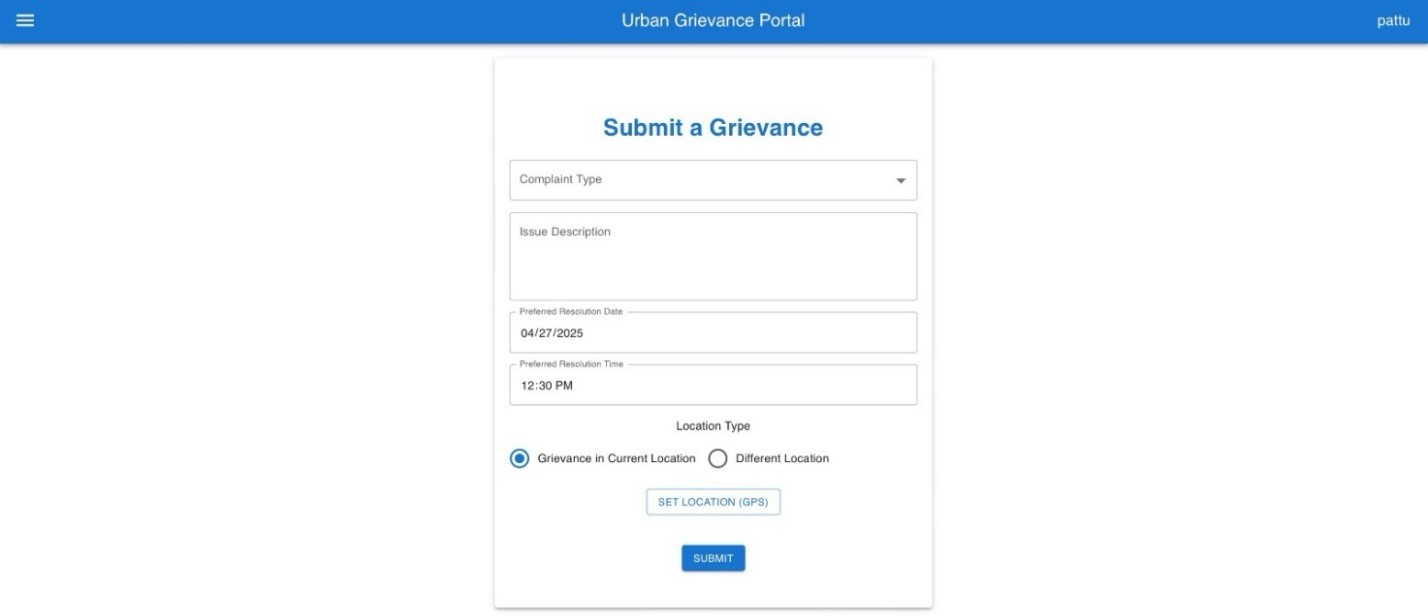
The FP analysis estimates **86 Function Points**, requiring **5.73 person-months** and **1.91 months**

with a 3-person team, aligning closely with the actual 3-month timeline.

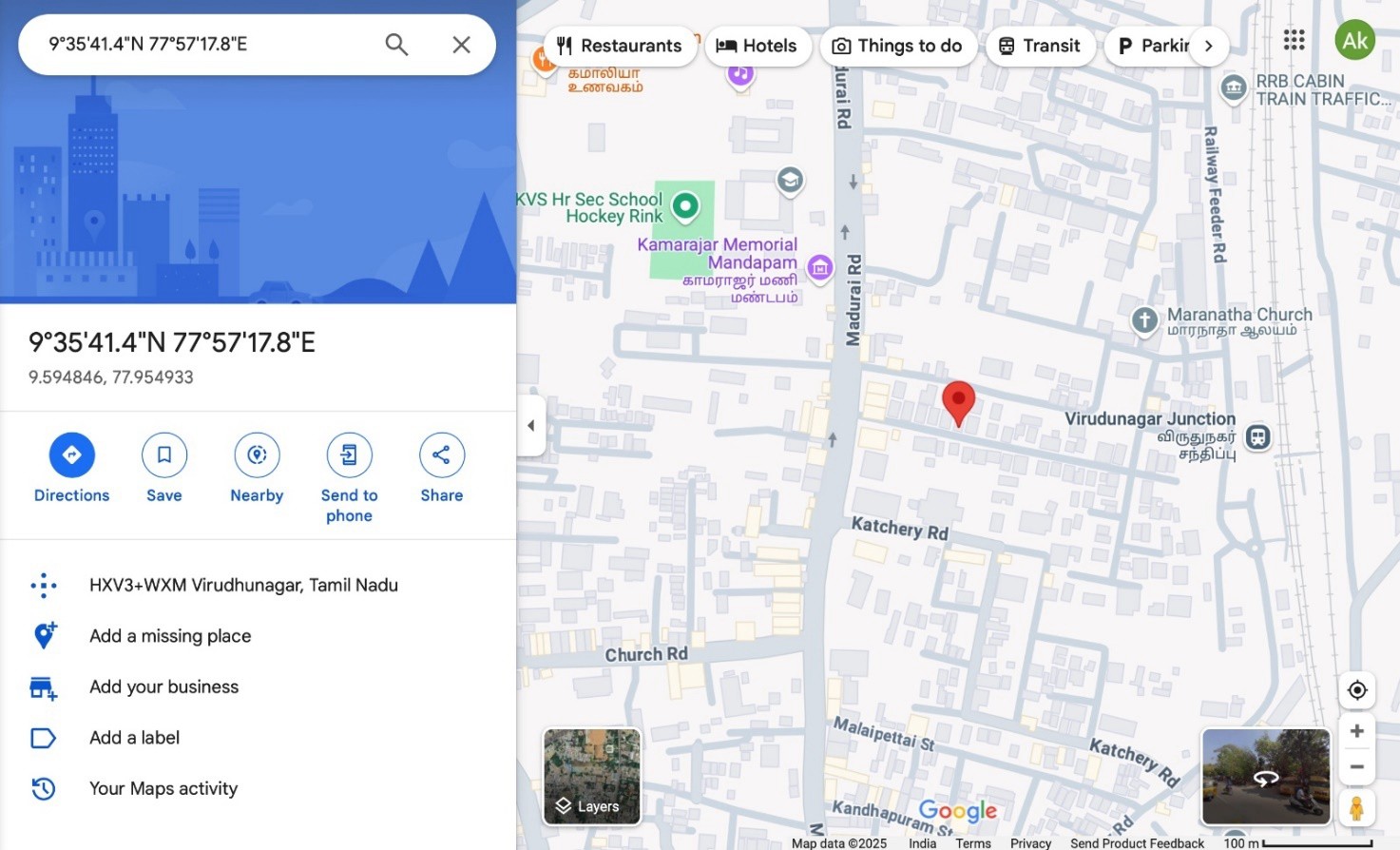
## CHAPTER 6 OUTPUTS



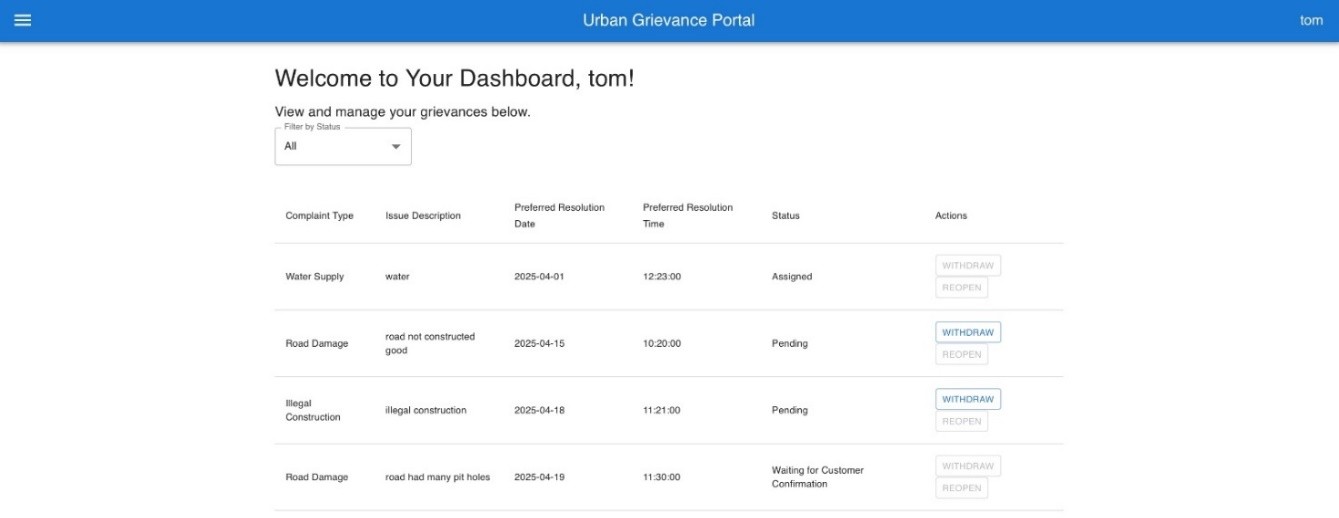
**Figure No.6.1.User Login Page**

****

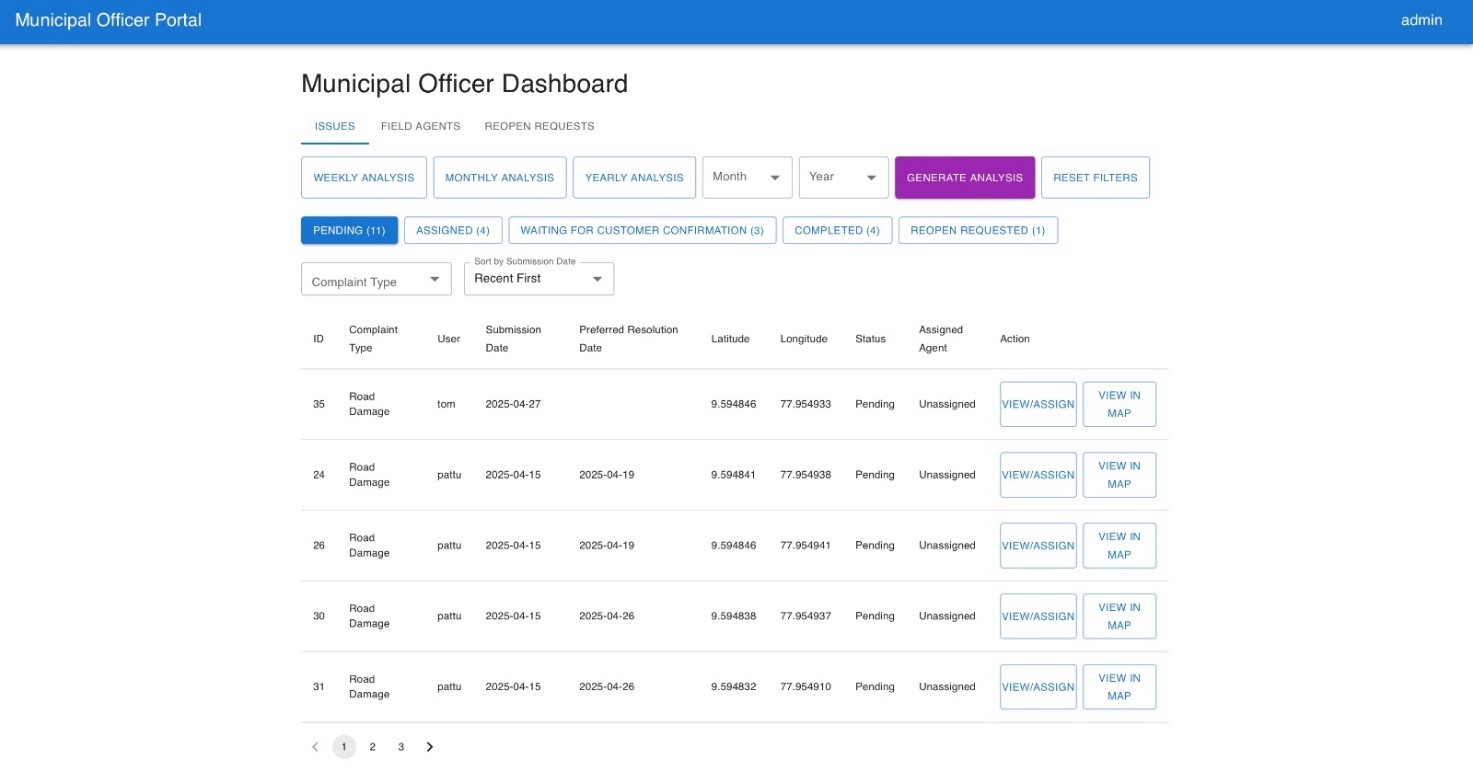
**Figure No.6.2.Grievance Submission Form**



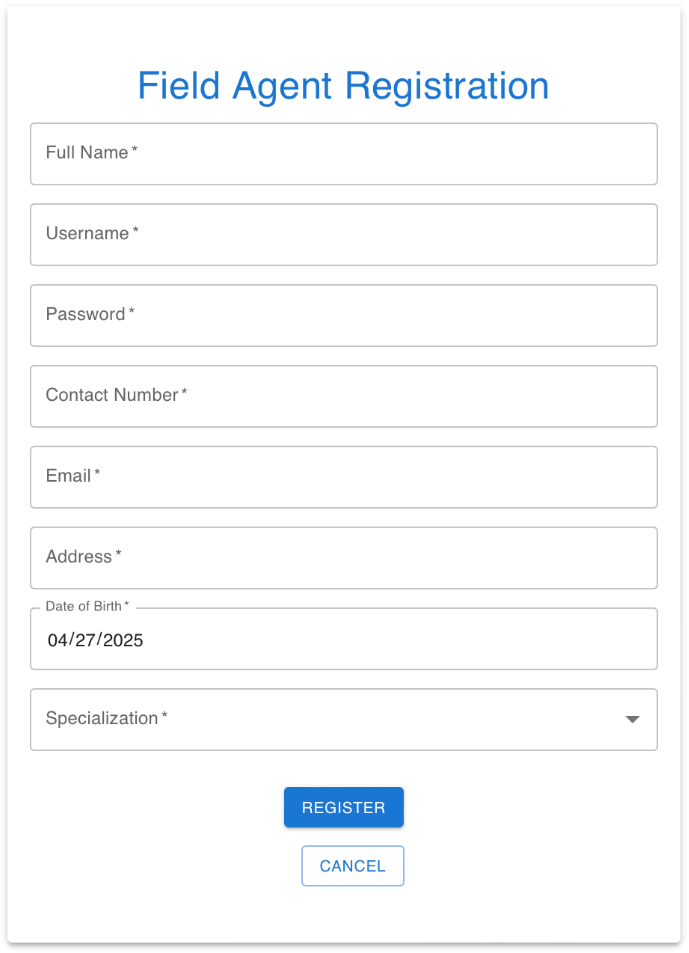
**Figure No.6.3.Location Of Issue**

****

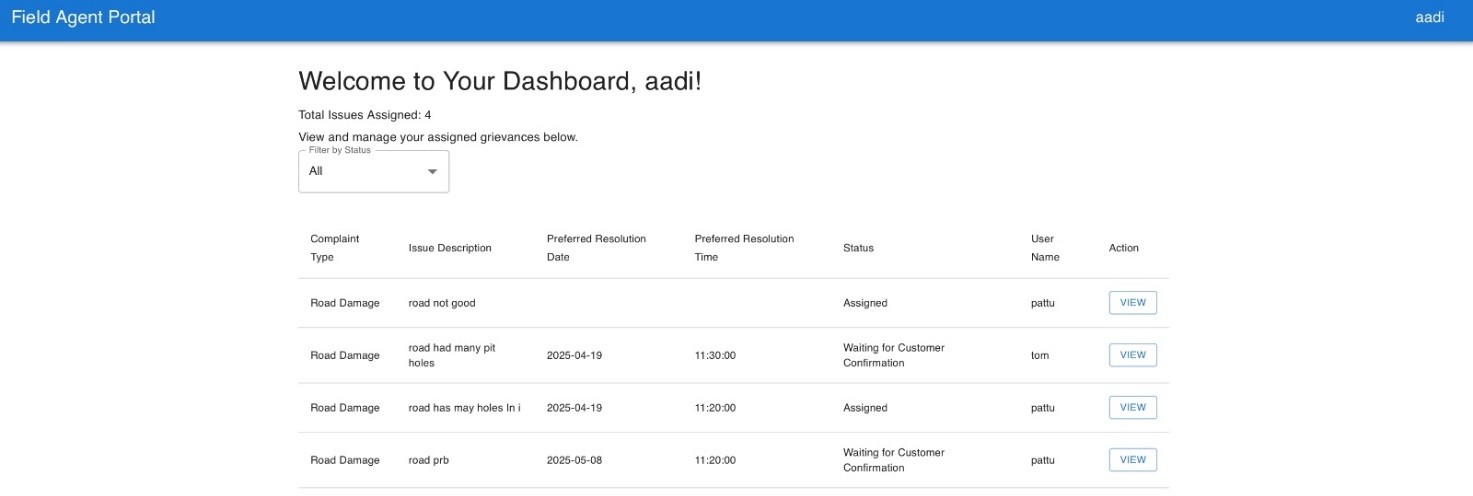
**Figure No.6.4.User Dashboard Page**



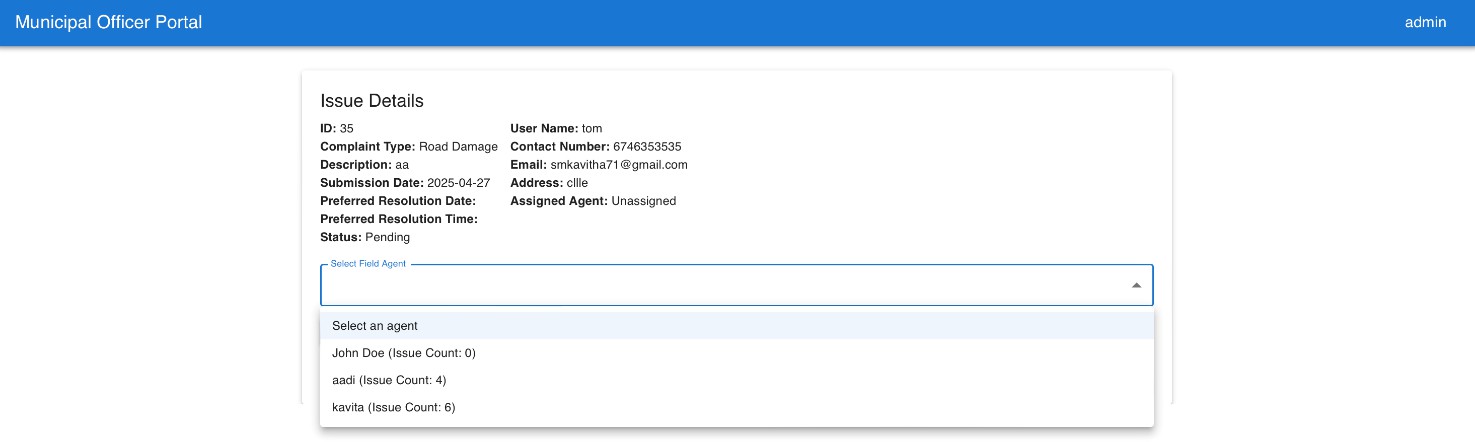
**Figure No.6.5.Municiapal Officer Dashboard Page**

****

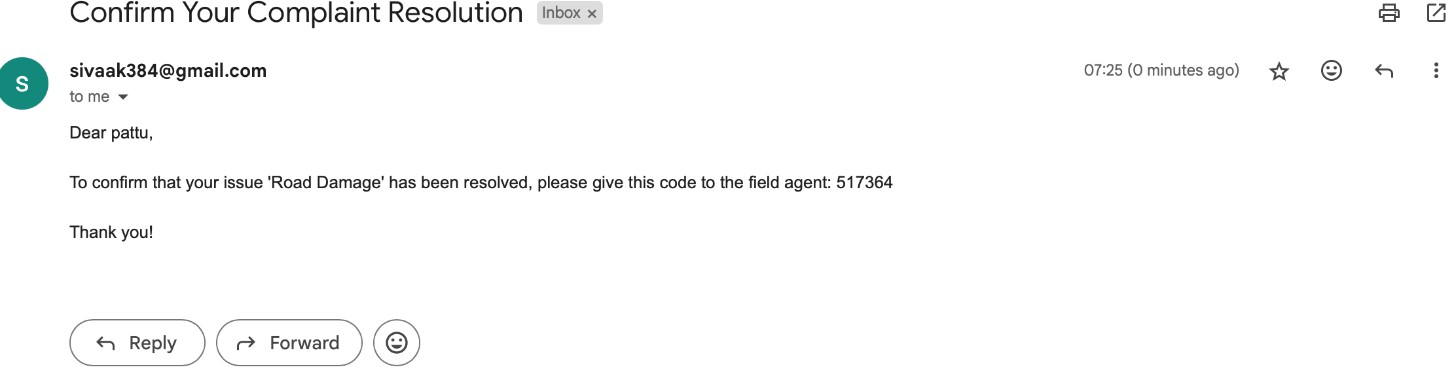
**Figure No.6.6.Field Agent Registration Page**



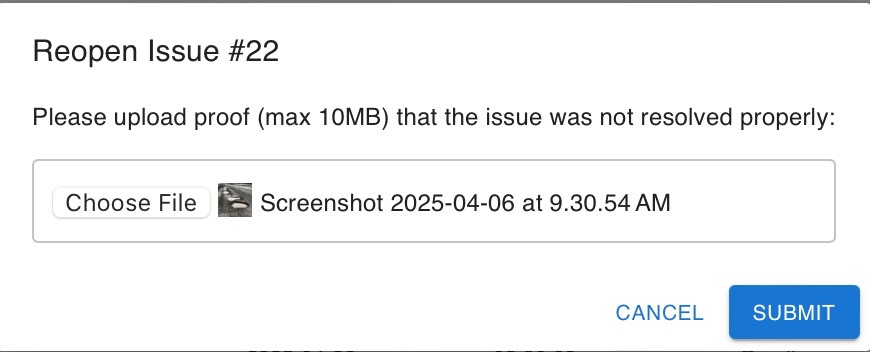
**Figure No.6.7.Field Agent Dashboard Page**



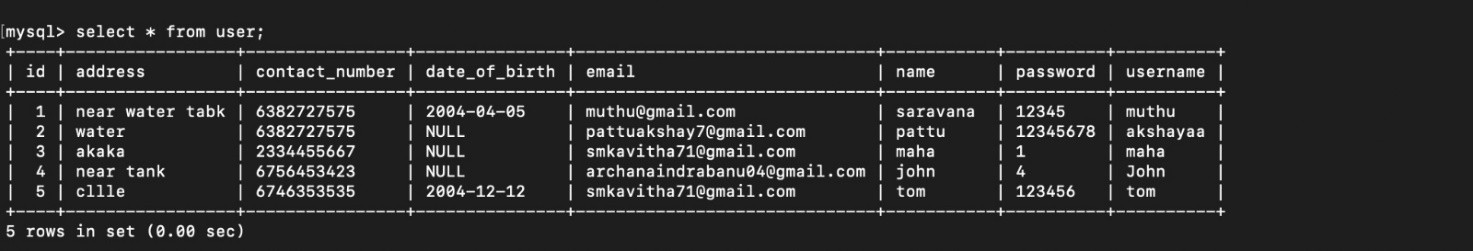
**Figure No.6.8.Assignment of Field Agent**

****

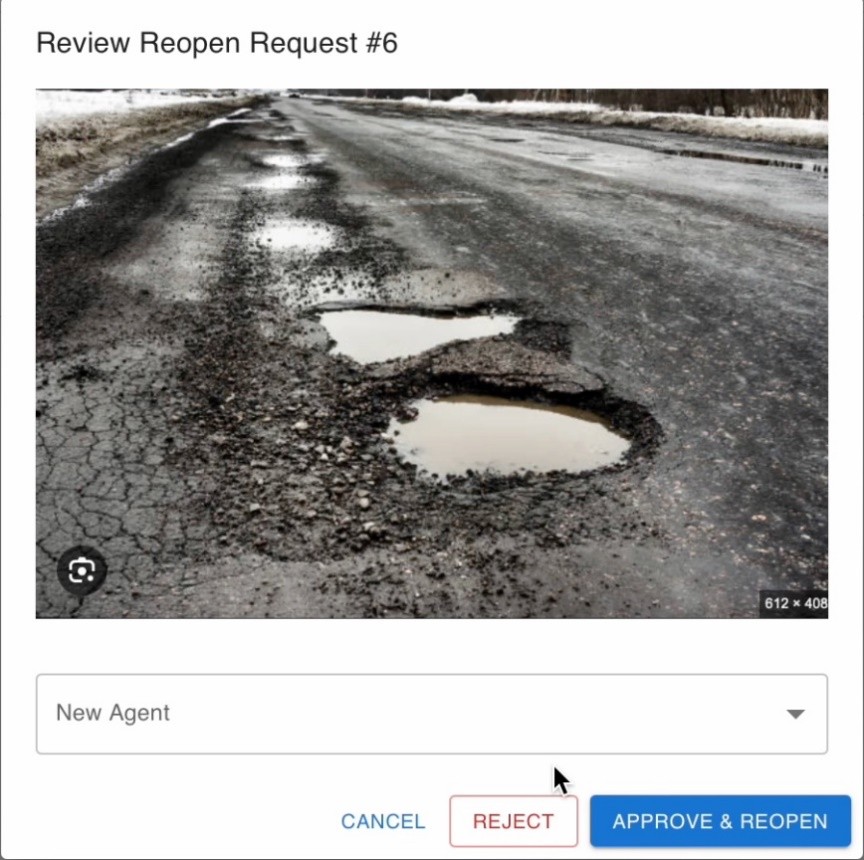
**Figure No.6.9.Email Confirmation for Issue Closing**



**Figure No.6.10.Reopening Issue**



**Figure No.6.11.Database Storage**

****

**Figure No.6.12.Reassigning new Field Agent**

CODE:

**Dashboard.js**

## CHAPTER 7 IMPLEMENTATION

import React, { useState, useEffect } from 'react'; import {

Container, Typography, Box,

Table, TableBody, TableCell, TableHead, TableRow, Select, MenuItem, FormControl, InputLabel, Button, Paper,

Grid,

Dialog, DialogTitle, DialogContent, DialogActions, TextField

} from '@mui/material';

export default function Dashboard() { const [issues, setIssues] = useState([]);

const [filteredIssues, setFilteredIssues] = useState([]); const [statusFilter, setStatusFilter] = useState('All'); const [selectedIssue, setSelectedIssue] = useState(null); const [error, setError] = useState('');

const [openReopenDialog, setOpenReopenDialog] = useState(false); const [reopenImage, setReopenImage] = useState(null);

const userId = localStorage.getItem("userId");

const userName = localStorage.getItem("userName");

const fetchIssues = async () => { try {

const response = await fetc[h(`http://localhost:8080/issue/user/${us](http://localhost:8080/issue/user/%24)erId}`); if (response.ok) {

const data = await response.json(); setIssues(data); setFilteredIssues(data);

} else {

setError("Failed to fetch issues.");

}

} catch (err) {

setError("Server error. Please try again later.");

}

};

useEffect(() => { if (!userId) {

setError("User not logged in. Please log in to view your dashboard."); return;

}

fetchIssues();

}, [userId]);

useEffect(() => {

if (statusFilter === 'All') { setFilteredIssues(issues);

} else {

setFilteredIssues(issues.filter(issue => issue.status === statusFilter));

}

setSelectedIssue(null);

}, [statusFilter, issues]);

const handleIssueClick = (issue) => { setSelectedIssue(issue);

};

const handleWithdraw = async (issueId) => { try {

const response = await fetc[h(`http://localhost:8080/issue/${i](http://localhost:8080/issue/%24)ssueId}`, { method: 'DELETE'

});

if (response.ok) {

setIssues(prevIssues => prevIssues.filter(issue => issue.id !== issueId)); setFilteredIssues(prevIssues => prevIssues.filter(issue => issue.id !== issueId)); setSelectedIssue(null);

} else {

const errorText = await response.text(); alert(`Failed to withdraw issue: ${errorText}`);

}

} catch (error) {

alert("Error withdrawing issue. Please try again later.");

}

};

const handleReopen = async (issueId) => { const issue = issues.find(i => i.id === issueId); if (issue.status !== "Completed") {

alert("Only completed issues can be reopened."); return;

}

setSelectedIssue(issue); setOpenReopenDialog(true);

};

const handleReopenSubmit = async () => { if (!reopenImage) {

alert("Please upload proof image.");

return;

}

const maxSize = 10 \* 1024 \* 1024; // 10MB in bytes if (reopenImage.size > maxSize) {

alert("File size exceeds 10MB. Please upload a smaller image."); return;

}

const formData = new FormData(); formData.append("issueId", selectedIssue.id); formData.append("image", reopenImage); formData.append("userId", userId);

try {

const response = await fetch('[http://localhost:8080/issue/reopen-request',](http://localhost:8080/issue/reopen-request%27) { method: 'POST',

body: formData,

});

if (response.ok) { setOpenReopenDialog(false); setReopenImage(null);

alert("Reopen request submitted successfully.");

// Fetch updated issues to reflect any changes (e.g., rejection) await fetchIssues();

} else {

const errorText = await response.text(); console.error("Response Status:", response.status); console.error("Response Text:", errorText);

alert(`Failed to submit reopen request: ${response.status} - ${errorText}`);

}

} catch (error) { console.error("Fetch Error:", error);

alert("Error submitting reopen request: " + error.message);

}

};

return (

<Container sx={{ mt: 4 }}>

<Box>

<Typography variant="h4" gutterBottom> Welcome to Your Dashboard, {userName}!

</Typography>

<Typography variant="h6" gutterBottom> View and manage your grievances below.

</Typography>

<FormControl sx={{ mb: 2, minWidth: 200 }}>

<InputLabel>Filter by Status</InputLabel>

<Select value={statusFilter}

onChange={(e) => setStatusFilter(e.target.value)} label="Filter by Status"

>

<MenuItem value="All">All</MenuItem>

<MenuItem value="Pending">Pending</MenuItem>

<MenuItem value="Completed">Completed</MenuItem>

<MenuItem value="Reopen Requested">Reopen Requested</MenuItem>

<MenuItem value="Rejected">Rejected</MenuItem>

</Select>

</FormControl>

{error ? (

<Typography color="error" sx={{ mt: 2 }}>

{error}

</Typography>

) : filteredIssues.length === 0 ? (

<Typography sx={{ mt: 2 }}>

No issues found for the selected status.

</Typography>

) : (

<>

<Table sx={{ mt: 2 }}>

<TableHead>

<TableRow>

<TableCell>Complaint Type</TableCell>

<TableCell>Issue Description</TableCell>

<TableCell>Preferred Resolution Date</TableCell>

<TableCell>Preferred Resolution Time</TableCell>

<TableCell>Status</TableCell>

<TableCell>Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{filteredIssues.map((issue) => (

<TableRow key={issue.id}

onClick={() => handleIssueClick(issue)}

sx={{ cursor: 'pointer', '&:hover': { backgroundColor: '#f5f5f5' } }}

>

<TableCell>{issue.complaintType}</TableCell>

<TableCell>{issue.issueDescription}</TableCell>

<TableCell>{issue.preferredResolutionDate}</TableCell>

<TableCell>{issue.preferredResolutionTime}</TableCell>

<TableCell>{issue.status}</TableCell>

<TableCell>

<Button variant="outlined" size="small"

disabled={issue.status !== "Pending"}

onClick={(e) => { e.stopPropagation(); handleWithdraw(issue.id); }} sx={{ mr: 1 }}

>

Withdraw

</Button>

<Button variant="outlined"

size="small"

disabled={issue.status !== "Completed"}

onClick={(e) => { e.stopPropagation(); handleReopen(issue.id); }}

>

Reopen

</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

<Dialog open={openReopenDialog} onClose={() => setOpenReopenDialog(false)}>

<DialogTitle>Reopen Issue #{selectedIssue?.id}</DialogTitle>

<DialogContent>

<Typography>Please upload proof (max 10MB) that the issue was not resolved properly:</Typography>

<TextField type="file"

inputProps={{ accept: "image/\*" }}

onChange={(e) => setReopenImage(e.target.files[0])} fullWidth

sx={{ mt: 2 }}

/>

</DialogContent>

<DialogActions>

<Button onClick={() => setOpenReopenDialog(false)}>Cancel</Button>

<Button onClick={handleReopenSubmit} variant="contained">Submit</Button>

</DialogActions>

</Dialog>

{selectedIssue && (

<Paper elevation={3} sx={{ mt: 4, p: 3 }}>

<Typography variant="h6" gutterBottom> Issue Details

</Typography>

<Grid container spacing={2}>

<Grid item xs={12} sm={6}>

<Typography><strong>Complaint Type:</strong>

{selectedIssue.complaintType}</Typography>

<Typography><strong>Issue Description:</strong>

{selectedIssue.issueDescription}</Typography>

<Typography><strong>Preferred Resolution Date:</strong>

{selectedIssue.preferredResolutionDate}</Typography>

<Typography><strong>Preferred Resolution Time:</strong>

{selectedIssue.preferredResolutionTime}</Typography>

<Typography><strong>Location:</strong> {selectedIssue.latitude && selectedIssue.longitude

? `${selectedIssue.latitude.toFixed(6)}, ${selectedIssue.longitude.toFixed(6)}`

: 'Not provided'}</Typography>

<Typography><strong>Status:</strong> {selectedIssue.status}</Typography>

</Grid>

<Grid item xs={12} sm={6}>

<Typography><strong>Full Name:</strong> {selectedIssue.user.name}</Typography>

<Typography><strong>Contact Number:</strong>

{selectedIssue.user.contactNumber}</Typography>

<Typography><strong>Email:</strong> {selectedIssue.user.email}</Typography>

<Typography><strong>Address:</strong> {selectedIssue.user.address}</Typography>

</Grid>

</Grid>

</Paper>

)}

</>

)}

</Box>

</Container>

);

}

###### Home.js

import React from "react";

import { Box, Grid, Typography } from "@mui/material";

export default function Home() { return (

<Box sx={{ height: "calc(100vh - 80px)", overflow: "hidden", display: "flex", alignItems: "center" }}>

<Grid container sx={{ height: "100%" }}>

{/\* Left Side: Full Height Image \*/}

<Grid item xs={5}

sx={{

display: "flex", justifyContent: "center",

alignItems: "center",

}}

>

<img src={require("../image.png")} alt="Urban Issues"

style={{

maxWidth: "100%",

maxHeight: "100%", objectFit: "contain",

}}

/>

</Grid>

{/\* Right Side: Text (Closer to Image) \*/}

<Grid item xs={7}

sx={{

display: "flex", flexDirection: "column", justifyContent: "center",

paddingLeft: "20px", // Adjusted to bring text closer paddingRight: "40px", // Prevents text from touching screen edges

}}

>

<Typography variant="h3" gutterBottom>

help

time.

Welcome to the Urban Grievance Portal

</Typography>

<Typography variant="h5" gutterBottom>

A platform for citizens to report and track urban issues. We ensure your grievances reach the right authorities for quick resolution.

</Typography>

<Typography variant="h6" paragraph>

Our system connects you directly with municipal bodies to handle issues like road damage, waste management, water supply problems, and more. Stay updated on your complaints and

make our cities better.

</Typography>

<Typography variant="h6">

Join us in creating a better urban environment by reporting and tracking grievances in real

</Typography>

</Grid>

</Grid>

</Box>

);

}

###### Login.js

import React, { useState } from 'react';

import { TextField, Container, Paper, Button, Box, Typography, Link } from '@mui/material'; import { useNavigate } from 'react-router-dom';

export default function Login() {

const [username, setUsername] = useState(''); const [password, setPassword] = useState(''); const [message, setMessage] = useState(''); const navigate = useNavigate();

const handleClick = async (e) => { e.preventDefault(); setMessage("");

const user = { username, password };

try {

const response = await fetch('[http://localhost:8080/user/checkuser',](http://localhost:8080/user/checkuser%27) { method: 'POST',

headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(user),

});

if (response.ok) {

const userData = await response.json(); localStorage.setItem("userId", userData.id); localStorage.setItem("userName", userData.name); localStorage.setItem("contactNumber", userData.contactNumber); localStorage.setItem("email", userData.email); localStorage.setItem("address", userData.address); navigate('/submit-grievance');

} else {

setMessage("Incorrect Username/Password ");

}

} catch (error) {

console.error("Error during login:", error); setMessage("Server error. Please try again later.");

}

};

return (

<Container>

<Paper elevation={3} sx={{ padding: "50px 20px", width: 600, margin: "20px auto", textAlign: "center" }}>

<h1 style={{ color: '#1976d2', marginBottom: '20px' }}>Login</h1>

<Box display="flex" flexDirection="column" alignItems="center">

<TextField label="Username" variant="outlined" fullWidth value={username} sx={{ mb: 2 }}

onChange={(e) => setUsername(e.target.value)}

/>

<TextField label="Password" variant="outlined" type="password" fullWidth value={password} sx={{ mb: 2 }}

onChange={(e) => setPassword(e.target.value)}

/>

<Button variant="contained"

sx={{ backgroundColor: "#1976d2", color: "white", mt: 2 }} onClick={handleClick}

>

Login

</Button>

{message && (

<Typography sx={{ color: message.includes("Success") ? "green" : "red", mt: 2 }}>

{message}

</Typography>

)}

<Typography sx={{ mt: 2 }}> Don't have an account?{" "}

<Link component="button" variant="body2"

onClick={() => navigate('/register')}

sx={{ color: "#1976d2", cursor: "pointer" }}

>

Register here

</Link>

</Typography>

</Box>

</Paper>

</Container>

);

}

###### SubmitGrievance.js

import React, { useState } from 'react';

import { TextField, Container, Paper, Button, Box, Typography, MenuItem, Radio, RadioGroup, FormControlLabel, FormControl } from '@mui/material';

import { useNavigate } from 'react-router-dom';

export default function SubmitGrievance() {

const [complaintType, setComplaintType] = useState(''); const [issueDescription, setIssueDescription] = useState('');

const [preferredResolutionDate, setPreferredResolutionDate] = useState(''); const [preferredResolutionTime, setPreferredResolutionTime] = useState(''); const [latitude, setLatitude] = useState(null);

const [longitude, setLongitude] = useState(null); const [coordinates, setCoordinates] = useState(''); const [message, setMessage] = useState('');

const [locationType, setLocationType] = useState('current'); // Default to 'current' const navigate = useNavigate();

const handleSetLocation = () => { if (!navigator.geolocation) {

setMessage('Geolocation is not supported by your browser.'); return;

}

console.log('Requesting GPS location...'); navigator.geolocation.getCurrentPosition( (position) => {

const lat = position.coords.latitude; const lon = position.coords.longitude;

if (lat > 90 || lat < -90 || lon > 180 || lon < -180) { setMessage('Invalid GPS coordinates received. Please try again.'); console.log('Invalid GPS coords:', { lat, lon });

return;

}

console.log('GPS location received:', { lat, lon }); setLatitude(lat);

setLongitude(lon); setCoordinates(`${lat.toString()},${lon.toString()}`);

setMessage('GPS location detected. Please submit to update database.');

},

(error) => {

console.error('Geolocation error:', error); switch (error.code) {

case error.PERMISSION\_DENIED:

setMessage('Location access denied. Please allow GPS.'); break;

case error.POSITION\_UNAVAILABLE: setMessage('Location data unavailable. Please try again.'); break;

case error.TIMEOUT:

setMessage('Request timed out. Please try again.'); break;

default:

setMessage('Unable to fetch location: ' + error.message);

}

},

{ timeout: 10000 }

);

};

const handleClearLocation = () => { setLatitude(null); setLongitude(null); setCoordinates(''); setMessage('Location cleared.'); console.log('Location cleared');

};

const handleManualLocation = () => {

const [latInput, lonInput] = coordinates.trim().split(','); const lat = parseFloat(latInput);

const lon = parseFloat(lonInput);

console.log('Attempting to set manual location:', { coordinates, lat, lon });

if (!latInput || !lonInput || isNaN(lat) || isNaN(lon) || lat > 90 || lat < -90 || lon > 180 || lon < -180) { setMessage('Invalid coordinates. Enter as "latitude,longitude" (e.g., "9.9252,78.1198" for

Madurai).');

setLatitude(null); setLongitude(null);

console.log('Validation failed:', { coordinates, lat, lon }); return;

}

setLatitude(lat); setLongitude(lon);

setMessage(`Location updated to ${lat.toFixed(6)}, ${lon.toFixed(6)}. Please submit to update database.`);

console.log('Location set successfully:', { latitude: lat, longitude: lon });

};

const handlePreviewLocation = () => {

const [latInput, lonInput] = coordinates.trim().split(','); const lat = parseFloat(latInput);

const lon = parseFloat(lonInput);

console.log('Previewing location:', { coordinates, lat, lon });

if (!latInput || !lonInput || isNaN(lat) || isNaN(lon) || lat > 90 || lat < -90 || lon > 180 || lon < -180) { setMessage('Invalid coordinates. Enter as "latitude,longitude" (e.g., "9.9252,78.1198").'); console.log('Preview validation failed:', { coordinates, lat, lon });

return;

}

setMessage('Previewing location in Google Maps.');

const mapsUrl = `https://[www.google.com/maps?q=${l](http://www.google.com/maps?q=%24)at},${lon}&z=15`;

window.open(mapsUrl, '\_blank');

};

const handleSubmit = async (e) => { e.preventDefault();

setMessage('');

const userId = localStorage.getItem("userId");

if (!userId) {

setMessage("User not logged in. Please log in to submit a grievance."); console.log('Submission failed: No userId');

return;

}

if (latitude === null || longitude === null) { setMessage("Please set a location before submitting."); return;

}

const issue = {

userId: parseInt(userId), complaintType, issueDescription, preferredResolutionDate, preferredResolutionTime, latitude,

longitude,

};

try {

console.log('Submitting issue:', issue);

const response = await fetch('[http://localhost:8080/issue/submit',](http://localhost:8080/issue/submit%27) { method: 'POST',

headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(issue),

});

if (response.ok) {

setMessage("Issue submitted successfully!"); console.log('Issue submitted:', issue); setTimeout(() => {

navigate('/dashboard');

}, 2000);

} else {

setMessage("Failed to submit issue. Please try again."); console.log('Submission failed:', response.status, await response.text());

}

} catch (error) {

console.error("Error submitting issue:", error); setMessage("Server error. Please try again later.");

}

};

return (

<Container>

<Paper elevation={3} sx={{ padding: "50px 20px", width: 600, margin: "20px auto", textAlign: "center" }}>

<h1 style={{ color: '#1976d2', marginBottom: '20px' }}>Submit a Grievance</h1>

<Box component="form" display="flex" flexDirection="column" alignItems="center" onSubmit={handleSubmit}>

<TextField select

label="Complaint Type" variant="outlined" fullWidth value={complaintType}

onChange={(e) => setComplaintType(e.target.value)} sx={{ mb: 2 }}

>

<MenuItem value="Road Damage">Road Damage</MenuItem>

<MenuItem value="Waste Management">Waste Management</MenuItem>

<MenuItem value="Water Supply">Water Supply</MenuItem>

<MenuItem value="Electricity Issues">Electricity Issues</MenuItem>

<MenuItem value="Public Transportation">Public Transportation</MenuItem>

<MenuItem value="Noise Pollution">Noise Pollution</MenuItem>

<MenuItem value="Air Pollution">Air Pollution</MenuItem>

<MenuItem value="Sewage and Drainage">Sewage and Drainage</MenuItem>

<MenuItem value="Street Lighting">Street Lighting</MenuItem>

<MenuItem value="Illegal Construction">Illegal Construction</MenuItem>

<MenuItem value="Parks and Recreation">Parks and Recreation</MenuItem>

<MenuItem value="Other">Other</MenuItem>

</TextField>

<TextField

label="Issue Description" variant="outlined" multiline

rows={4} fullWidth

value={issueDescription}

onChange={(e) => setIssueDescription(e.target.value)} sx={{ mb: 2 }}

/>

<TextField

label="Preferred Resolution Date" type="date"

variant="outlined" fullWidth

value={preferredResolutionDate}

onChange={(e) => setPreferredResolutionDate(e.target.value)} InputLabelProps={{ shrink: true }}

sx={{ mb: 2 }}

/>

<TextField

label="Preferred Resolution Time" type="time"

variant="outlined" fullWidth

value={preferredResolutionTime}

onChange={(e) => setPreferredResolutionTime(e.target.value)} InputLabelProps={{ shrink: true }}

sx={{ mb: 2 }}

/>

<FormControl component="fieldset" sx={{ mb: 2, width: '100%' }}>

<Typography sx={{ mb: 1 }}>Location Type</Typography>

<RadioGroup row

value={locationType}

onChange={(e) => setLocationType(e.target.value)}

>

<FormControlLabel value="current" control={<Radio />} label="Grievance in Current Location" />

<FormControlLabel value="different" control={<Radio />} label="Different Location" />

</RadioGroup>

</FormControl>

{locationType === 'current' ? (

<Box sx={{ mb: 2, width: '100%' }}>

<Button variant="outlined" onClick={handleSetLocation} sx={{ mb: 1 }}> Set Location (GPS)

</Button>

{latitude !== null && longitude !== null && (

<Typography sx={{ color: 'green', mt: 1 }}>

Location set to: {latitude.toFixed(6)}, {longitude.toFixed(6)}{' '}

<a

href={`https://[www.google.com/maps?q=${l](http://www.google.com/maps?q=%24)atitude},${longitude}&z=15`} target="\_blank"

rel="noopener noreferrer"

style={{ color: 'inherit', textDecoration: 'underline' }}

>

(View on Map)

</a>

</Typography>

)}

</Box>

) : (

<Box sx={{ mb: 2, width: '100%' }}>

<Box sx={{ display: 'flex', gap: 1, mb: 1 }}>

<Button variant="outlined" onClick={handleSetLocation}> Set Location (GPS)

</Button>

{latitude && longitude && (

<Button variant="outlined" color="secondary" onClick={handleClearLocation}> Clear Location

</Button>

)}

</Box>

<Typography sx={{ color: 'gray', mb: 1 }}>

Paste coordinates as "latitude,longitude" (e.g., "9.9252,78.1198" for Madurai) from Google Maps, preview to check, then update.

</Typography>

<Box sx={{ display: 'flex', gap: 1, mb: 1, alignItems: 'center' }}>

<TextField

label="Coordinates (latitude,longitude)" variant="outlined"

value={coordinates}

onChange={(e) => setCoordinates(e.target.value)} placeholder="e.g., 9.9252,78.1198"

fullWidth sx={{ mb: 1 }}

/>

<Button variant="outlined" onClick={handlePreviewLocation} sx={{ height: '56px' }}> Preview Location

</Button>

</Box>

<Button variant="outlined" onClick={handleManualLocation} sx={{ mb: 1 }}> Update Location

</Button>

{(latitude !== null && longitude !== null) ? (

<Typography sx={{ color: 'green', mt: 1 }}>

Location set to: {latitude.toFixed(6)}, {longitude.toFixed(6)}{' '}

<a

href={`https://[www.google.com/maps?q=${l](http://www.google.com/maps?q=%24)atitude},${longitude}&z=15`} target="\_blank"

rel="noopener noreferrer"

style={{ color: 'inherit', textDecoration: 'underline' }}

>

(View on Map)

</a>

</Typography>

) : (

<Typography sx={{ color: 'gray', mt: 1 }}> No location set. Update after previewing.

</Typography>

)}

</Box>

)}

<Button type="submit" variant="contained" sx={{ backgroundColor: '#1976d2', color: 'white', mt: 2 }}>

Submit

</Button>

{message && (

<Typography sx={{ color: message.includes('successfully') || message.includes('updated') || message.includes('cleared') ? 'green' : 'red', mt: 2 }}>

{message}

</Typography>

)}

</Box>

</Paper>

</Container>

);

}

MUNICIPAL OFFICER :

###### Municipalofficerdashboard.js

import React, { useState, useEffect } from 'react'; import {

Container, Typography, Box,

Tabs, Tab,

Table, TableBody, TableCell, TableHead, TableRow, Button, Grid, Pagination, Dialog, DialogTitle,

DialogContent, DialogActions, FormControl, InputLabel, Select, MenuItem,

} from '@mui/material';

import { useNavigate, useLocation } from 'react-router-dom';

export default function Dashboard() { const [tab, setTab] = useState(0); const [issues, setIssues] = useState([]);

const [filteredIssues, setFilteredIssues] = useState([]); const [fieldAgents, setFieldAgents] = useState([]);

const [filteredFieldAgents, setFilteredFieldAgents] = useState([]); const [reopenRequests, setReopenRequests] = useState([]);

const [selectedRequest, setSelectedRequest] = useState(null);

const [newAgentId, setNewAgentId] = useState(''); const [error, setError] = useState('');

const [statusFilter, setStatusFilter] = useState(['Pending', 'Reopen Requested']); const [complaintTypeFilter, setComplaintTypeFilter] = useState('');

const [sortOrder, setSortOrder] = useState('desc'); const [issuePage, setIssuePage] = useState(1); const [agentPage, setAgentPage] = useState(1);

const [selectedPeriod, setSelectedPeriod] = useState(''); const [selectedMonth, setSelectedMonth] = useState(''); const [selectedYear, setSelectedYear] = useState(''); const itemsPerPage = 5;

const navigate = useNavigate(); const location = useLocation();

const isLoggedIn = localStorage.getItem("officerLoggedIn"); if (!isLoggedIn) {

navigate('/');

}

useEffect(() => {

const fetchData = async () => { try {

const [issuesResponse, agentsResponse, reopenResponse] = await Promise.all([ fetch('[http://localhost:8080/issue/all'),](http://localhost:8080/issue/all%27))

fetch('[http://localhost:8080/field-agent/all'),](http://localhost:8080/field-agent/all%27)) fetch('[http://localhost:8080/issue/reopen-requests'),](http://localhost:8080/issue/reopen-requests%27))

]);

if (issuesResponse.ok && agentsResponse.ok && reopenResponse.ok) { const issuesData = await issuesResponse.json();

const agentsData = await agentsResponse.json(); const reopenData = await reopenResponse.json(); console.log('Issues Data:', issuesData); setIssues(issuesData);

const initialFiltered = issuesData

.filter(issue => ['Pending', 'Reopen Requested'].includes(issue.status))

.sort((a, b) => new Date(b.submissionDate) - new Date(a.submissionDate)); setFilteredIssues(initialFiltered);

setFieldAgents(agentsData); setFilteredFieldAgents(agentsData); setReopenRequests(reopenData);

} else {

setError('Failed to fetch data.');

}

} catch (err) {

setError('Server error. Please try again later.'); console.error('Fetch error:', err);

}

};

const searchParams = new URLSearchParams(location.search); if (searchParams.get('refresh') === 'true') {

fetchData();

navigate('/dashboard', { replace: true });

} else {

fetchData();

}

}, [location.search, navigate]);

useEffect(() => {

const applyFilters = () => { let filtered = [...issues];

if (statusFilter.length > 0) {

filtered = filtered.filter(issue => statusFilter.includes(issue.status));

}

if (complaintTypeFilter) {

filtered = filtered.filter(issue => issue.complaintType === complaintTypeFilter);

}

if (selectedMonth && selectedYear) { filtered = filtered.filter(issue => {

const issueDate = new Date(issue.submissionDate);

return issueDate.getMonth() + 1 === parseInt(selectedMonth) && issueDate.getFullYear()

=== parseInt(selectedYear);

});

}

filtered.sort((a, b) => sortOrder === 'asc'

? new Date(a.submissionDate) - new Date(b.submissionDate)

: new Date(b.submissionDate) - new Date(a.submissionDate)

);

setFilteredIssues(filtered);

};

applyFilters();

}, [statusFilter, complaintTypeFilter, sortOrder, issues, selectedMonth, selectedYear]);

useEffect(() => {

const applyPeriodFilter = async () => { if (selectedPeriod) {

try {

const response = await fetc[h(`http://localhost:8080/issue/issues-by-](http://localhost:8080/issue/issues-by-) period?period=${selectedPeriod}`);

if (response.ok) {

const data = await response.json();

setFilteredIssues(data.sort((a, b) => new Date(b.submissionDate) - new Date(a.submissionDate)));

} else {

setError('Failed to fetch issues for the selected period.');

}

} catch (err) {

setError('Error fetching issues: ' + err.message); console.error('Fetch error:', err);

}

}

};

applyPeriodFilter();

}, [selectedPeriod]);

const fetchIssuesByPeriod = (period) => { setSelectedPeriod(period); setSelectedMonth('');

setSelectedYear('');

};

const filterFieldAgents = () => { let filtered = [...fieldAgents];

if (complaintTypeFilter) {

filtered = filtered.filter((agent) => agent.specialization === complaintTypeFilter);

}

setFilteredFieldAgents(filtered);

};

useEffect(() => { filterFieldAgents();

}, [complaintTypeFilter]);

const handleApproveReopen = async (requestId) => { if (!newAgentId) {

alert("Please select a new agent."); return;

}

const request = reopenRequests.find((r) => r.id === requestId); const previousAgent = fieldAgents.find(

(agent) => agent.name === request.previousAgentName && request.previousAgentName !== "None"

);

if (previousAgent && previousAgent.id === parseInt(newAgentId)) { alert("Cannot assign the same agent who previously handled this issue."); return;

}

try {

const response = await fetc[h(`http://localhost:8080/issue/reopen/${re](http://localhost:8080/issue/reopen/%24)questId}`, { method: 'POST',

headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ newAgentId }),

});

if (response.ok) {

setReopenRequests((prev) => prev.filter((r) => r.id !== requestId)); setSelectedRequest(null);

const issuesResponse = await fetch('[http://localhost:8080/issue/all');](http://localhost:8080/issue/all%27)%3B) const issuesData = await issuesResponse.json(); setIssues(issuesData);

const initialFiltered = issuesData

.filter(issue => ['Pending', 'Reopen Requested'].includes(issue.status))

.sort((a, b) => new Date(b.submissionDate) - new Date(a.submissionDate)); setFilteredIssues(initialFiltered);

alert("Issue reopened successfully with new agent.");

} else {

const errorText = await response.text();

alert(`Failed to approve reopen request: ${errorText}`);

}

} catch (err) {

alert("Error approving reopen request: " + err.message);

}

};

const handleRejectReopen = async (requestId) => { if (!requestId || requestId === undefined) { console.error("Invalid requestId:", requestId);

alert("Error: No valid request ID provided for rejection."); return;

}

try {

const response = await fetc[h(`http://localhost:8080/issue/reopen/reject/${re](http://localhost:8080/issue/reopen/reject/%24)questId}`, { method: 'POST',

headers: { 'Content-Type': 'application/json' },

});

if (response.ok) {

setReopenRequests((prev) => prev.filter((r) => r.id !== requestId)); setSelectedRequest(null);

alert("Reopen request rejected successfully.");

} else {

const errorText = await response.text();

console.error(`Reject failed with status ${response.status}: ${errorText}`); alert(`Failed to reject reopen request: ${errorText}`);

}

} catch (err) {

console.error("Fetch error during rejection:", err); alert("Error rejecting reopen request: " + err.message);

}

};

const handleViewInMap = (latitude, longitude) => {

if (latitude && longitude && !isNaN(latitude) && !isNaN(longitude)) {

const mapsUrl = `https://[www.google.com/maps?q=${l](http://www.google.com/maps?q=%24)atitude},${longitude}&z=15`; window.open(mapsUrl, '\_blank');

} else {

alert('No valid location data available for this issue.');

}

};

const fetchIssuesByMonthYear = async (month, year) => { try {

const response = await fetc[h(`http://localhost:8080/issue/issues-by-month-](http://localhost:8080/issue/issues-by-month-) year?month=${month}&year=${year}`);

if (response.ok) {

const data = await response.json();

setFilteredIssues(data.sort((a, b) => new Date(b.submissionDate) - new Date(a.submissionDate)));

setSelectedMonth(month); setSelectedYear(year); setSelectedPeriod('');

} else {

setError('Failed to fetch issues for the selected month and year.');

}

} catch (err) {

setError('Error fetching issues: ' + err.message);

console.error('Fetch error:', err);

}

};

const generateComprehensiveAnalysis = async () => { try {

const response = await fetc[h(`http://localhost:8080/issue/generate-analysis](http://localhost:8080/issue/generate-analysis)`, { method: 'GET',

headers: {

'Accept': 'application/pdf',

},

});

if (response.ok) {

const blob = await response.blob();

const url = window.URL.createObjectURL(blob); const link = document.createElement('a'); link.href = url;

link.download = `comprehensive\_grievance\_analysis.pdf`; document.body.appendChild(link);

link.click(); document.body.removeChild(link); window.URL.revokeObjectURL(url);

} else {

console.error('Failed to generate PDF');

alert('Failed to generate comprehensive analysis report. Please try again.');

}

} catch (err) {

console.error('Error generating analysis:', err);

alert('Error generating comprehensive analysis report: ' + err.message);

}

};

const issuePageCount = Math.ceil(filteredIssues.length / itemsPerPage);

const agentPageCount = Math.ceil(filteredFieldAgents.length / itemsPerPage);

const paginatedIssues = filteredIssues.slice((issuePage - 1) \* itemsPerPage, issuePage \* itemsPerPage);

const paginatedAgents = filteredFieldAgents.slice((agentPage - 1) \* itemsPerPage, agentPage \* itemsPerPage);

const months = [

'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',

'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'

];

const years = Array.from({ length: 2026 - 2000 }, (\_, i) => 2000 + i);

const complaintTypes = [ "Road Damage", "Waste Management", "Water Supply", "Electricity Issues", "Public Transportation", "Noise Pollution",

"Air Pollution", "Sewage and Drainage", "Street Lighting",

"Illegal Construction", "Parks and Recreation", "Other",

];

const getStatusCount = (status) => {

return issues.filter(issue => issue.status === status).length;

};

const handleStatusFilter = (status) => { setStatusFilter([status]); setSelectedPeriod(''); setSelectedMonth(''); setSelectedYear('');

};

const resetFilters = () => { setStatusFilter(['Pending', 'Reopen Requested']); setComplaintTypeFilter('');

setSortOrder('desc'); setSelectedPeriod(''); setSelectedMonth(''); setSelectedYear('');

const initialFiltered = issues

.filter(issue => ['Pending', 'Reopen Requested'].includes(issue.status))

.sort((a, b) => new Date(b.submissionDate) - new Date(a.submissionDate)); setFilteredIssues(initialFiltered);

};

return (

<Container sx={{ mt: 4 }}>

<Typography variant="h4" gutterBottom>Municipal Officer Dashboard</Typography>

<Tabs value={tab} onChange={(e, newValue) => setTab(newValue)} sx={{ mb: 2 }}>

<Tab label="Issues" />

<Tab label="Field Agents" />

<Tab label="Reopen Requests" />

</Tabs>

<Grid container spacing={2} sx={{ mb: 2 }}>

<Grid item xs={12}>

<Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap' }}>

<Button

variant={selectedPeriod === 'weekly' ? 'contained' : 'outlined'} onClick={() => fetchIssuesByPeriod('weekly')}

sx={{ mb: 1 }}

>

Weekly Analysis

</Button>

<Button

variant={selectedPeriod === 'monthly' ? 'contained' : 'outlined'} onClick={() => fetchIssuesByPeriod('monthly')}

sx={{ mb: 1 }}

>

Monthly Analysis

</Button>

<Button

variant={selectedPeriod === 'yearly' ? 'contained' : 'outlined'} onClick={() => fetchIssuesByPeriod('yearly')}

sx={{ mb: 1 }}

>

Yearly Analysis

</Button>

<FormControl sx={{ minWidth: 120, mb: 1 }}>

<InputLabel>Month</InputLabel>

<Select value={selectedMonth}

onChange={(e) => fetchIssuesByMonthYear(e.target.value, selectedYear || new Date().getFullYear())}

label="Month"

>

<MenuItem value="">All Months</MenuItem>

{months.map((month, index) => (

<MenuItem key={month} value={index + 1}>{month}</MenuItem>

))}

</Select>

</FormControl>

<FormControl sx={{ minWidth: 120, mb: 1 }}>

<InputLabel>Year</InputLabel>

<Select value={selectedYear}

onChange={(e) => fetchIssuesByMonthYear(selectedMonth || 1, e.target.value)} label="Year"

>

<MenuItem value="">All Years</MenuItem>

{years.map((year) => (

<MenuItem key={year} value={year}>{year}</MenuItem>

))}

</Select>

</FormControl>

<Button variant="contained" color="secondary"

onClick={generateComprehensiveAnalysis} sx={{ mb: 1 }}

>

Generate Analysis

</Button>

<Button variant="outlined" onClick={resetFilters} sx={{ mb: 1 }}

>

Reset Filters

</Button>

</Box>

</Grid>

</Grid>

{tab === 0 && (

<Box>

<Grid container spacing={2} sx={{ mb: 2 }}>

<Grid item xs={12} sm={6}>

<Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap' }}>

{['Pending', 'Assigned', 'Waiting for Customer Confirmation', 'Completed', 'Reopen Requested'].map(status => (

<Button key={status}

variant={statusFilter[0] === status ? 'contained' : 'outlined'} onClick={() => handleStatusFilter(status)}

sx={{ mb: 1 }}

>

{status} ({getStatusCount(status)})

</Button>

))}

</Box>

</Grid>

<Grid item xs={12} sm={3}>

<FormControl fullWidth sx={{ minWidth: 200 }}>

<InputLabel>Complaint Type</InputLabel>

<Select value={complaintTypeFilter}

onChange={(e) => setComplaintTypeFilter(e.target.value)} label="Complaint Type"

sx={{ height: 45 }}

>

<MenuItem value="">All</MenuItem>

{complaintTypes.map((type) => (

<MenuItem key={type} value={type}>{type}</MenuItem>

))}

</Select>

</FormControl>

</Grid>

<Grid item xs={12} sm={3}>

<FormControl fullWidth sx={{ minWidth: 200 }}>

<InputLabel>Sort by Submission Date</InputLabel>

<Select value={sortOrder}

onChange={(e) => setSortOrder(e.target.value)} label="Sort by Submission Date"

sx={{ height: 45 }}

>

<MenuItem value="desc">Recent First</MenuItem>

<MenuItem value="asc">Oldest First</MenuItem>

</Select>

</FormControl>

</Grid>

</Grid>

{error ? (

<Typography color="error">{error}</Typography>

) : filteredIssues.length === 0 ? (

<Typography>No issues found.</Typography>

) : (

<>

<Table>

<TableHead>

<TableRow>

<TableCell>ID</TableCell>

<TableCell>Complaint Type</TableCell>

<TableCell>User</TableCell>

<TableCell>Submission Date</TableCell>

<TableCell>Preferred Resolution Date</TableCell>

<TableCell>Latitude</TableCell>

<TableCell>Longitude</TableCell>

<TableCell>Status</TableCell>

<TableCell>Assigned Agent</TableCell>

<TableCell>Action</TableCell>

</TableRow>

</TableHead>

<TableBody>

{paginatedIssues.map((issue) => (

<TableRow key={issue.id}>

<TableCell>{issue.id}</TableCell>

<TableCell>{issue.complaintType}</TableCell>

<TableCell>{issue.user ? issue.user.name.trim() : 'N/A'}</TableCell>

<TableCell>{issue.submissionDate}</TableCell>

<TableCell>{issue.preferredResolutionDate}</TableCell>

<TableCell>{issue.latitude ? issue.latitude.toFixed(6) : 'N/A'}</TableCell>

<TableCell>{issue.longitude ? issue.longitude.toFixed(6) : 'N/A'}</TableCell>

<TableCell>{issue.status}</TableCell>

<TableCell>

{issue.assignedAgent ? issue.assignedAgent.name : "Unassigned"}

</TableCell>

<TableCell>

<Box sx={{ display: 'flex', gap: 1 }}>

<Button variant="outlined"

onClick={() => navigate(`/issue-details/${issue.id}`)}

>

View/Assign

</Button>

{issue.latitude && issue.longitude && (

<Button variant="outlined" color="primary"

onClick={() => handleViewInMap(issue.latitude, issue.longitude)}

>

View in Map

</Button>

)}

</Box>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

<Pagination count={issuePageCount} page={issuePage}

onChange={(e, value) => setIssuePage(value)} sx={{ mt: 2 }}

/>

</>

)}

</Box>

)}

{tab === 1 && (

<Box>

<Grid container spacing={2} sx={{ mb: 2 }}>

<Grid item xs={12}>

<Button variant="contained"

onClick={() => navigate('/field-agent-register')} sx={{ mb: 2 }}

>

Add Field Agent

</Button>

</Grid>

<Grid item xs={12} sm={6}>

<FormControl fullWidth sx={{ minWidth: 200 }}>

<InputLabel>Specialization</InputLabel>

<Select value={complaintTypeFilter}

onChange={(e) => setComplaintTypeFilter(e.target.value)} label="Specialization"

sx={{ height: 45 }}

>

<MenuItem value="">All</MenuItem>

{complaintTypes.map((type) => (

<MenuItem key={type} value={type}>{type}</MenuItem>

))}

</Select>

</FormControl>

</Grid>

</Grid>

{filteredFieldAgents.length === 0 ? (

<Typography>No field agents found.</Typography>

) : (

<>

<Table>

<TableHead>

<TableRow>

<TableCell>ID</TableCell>

<TableCell>Name</TableCell>

<TableCell>Specialization</TableCell>

<TableCell>Issue Count</TableCell>

<TableCell>Contact Number</TableCell>

<TableCell>Email</TableCell>

</TableRow>

</TableHead>

<TableBody>

{paginatedAgents.map((agent) => (

<TableRow key={agent.id}>

<TableCell>{agent.id}</TableCell>

<TableCell>{agent.name}</TableCell>

<TableCell>{agent.specialization}</TableCell>

<TableCell>{agent.issueCount}</TableCell>

<TableCell>{agent.contactNumber}</TableCell>

<TableCell>{agent.email}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

<Pagination count={agentPageCount} page={agentPage}

onChange={(e, value) => setAgentPage(value)} sx={{ mt: 2 }}

/>

</>

)}

</Box>

)}

{tab === 2 && (

<Box>

{reopenRequests.length === 0 ? (

<Typography>No reopen requests pending.</Typography>

) : (

<Table>

<TableHead>

<TableRow>

<TableCell>Request ID</TableCell>

<TableCell>Issue ID</TableCell>

<TableCell>User</TableCell>

<TableCell>Previous Agent</TableCell>

<TableCell>Action</TableCell>

</TableRow>

</TableHead>

<TableBody>

{reopenRequests.map((request) => (

<TableRow key={request.id}>

<TableCell>{request.id}</TableCell>

<TableCell>{request.issueId}</TableCell>

<TableCell>{request.userName}</TableCell>

<TableCell>{request.previousAgentName}</TableCell>

<TableCell>

<Button variant="outlined"

onClick={() => setSelectedRequest(request)}

>

Review

</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

)}

<Dialog open={!!selectedRequest} onClose={() => setSelectedRequest(null)}>

<DialogTitle>Review Reopen Request #{selectedRequest?.id}</DialogTitle>

<DialogContent>

<img

src[={`http://localhost:8080/issue/reopen-image/${s](http://localhost:8080/issue/reopen-image/%24)electedRequest?.id}`} alt="Proof"

style={{ maxWidth: '100%', marginBottom: '16px' }}

/>

<FormControl fullWidth sx={{ mt: 2 }}>

<InputLabel>New Agent</InputLabel>

<Select value={newAgentId}

onChange={(e) => setNewAgentId(e.target.value)} label="New Agent"

>

<MenuItem value="">Select Agent</MenuItem>

{fieldAgents

.filter((agent) => {

const previousAgent = fieldAgents.find(

(a) => a.name === selectedRequest?.previousAgentName && selectedRequest?.previousAgentName !== "None"

);

return !previousAgent || agent.id !== previousAgent.id;

})

.map((agent) => (

<MenuItem key={agent.id} value={agent.id}>

{agent.name} ({agent.specialization})

</MenuItem>

))}

</Select>

</FormControl>

</DialogContent>

<DialogActions>

<Button onClick={() => setSelectedRequest(null)}>Cancel</Button>

<Button variant="outlined" color="error"

onClick={() => handleRejectReopen(selectedRequest?.id)}

>

Reject

</Button>

<Button variant="contained"

onClick={() => handleApproveReopen(selectedRequest?.id)}

>

Approve & Reopen

</Button>

</DialogActions>

</Dialog>

</Box>

)}

</Container>

);

}

###### FIELDAGENT

**Dashboard.js**

import React, { useState, useEffect } from 'react'; import {

Container, Typography, Box,

Table, TableBody, TableCell, TableHead, TableRow, Select, MenuItem, FormControl, InputLabel, Button

} from '@mui/material';

import { useNavigate } from 'react-router-dom';

export default function Dashboard() { const [issues, setIssues] = useState([]);

const [filteredIssues, setFilteredIssues] = useState([]); const [statusFilter, setStatusFilter] = useState('All');

const [error, setError] = useState('');

const agentId = localStorage.getItem("agentId");

const agentName = localStorage.getItem("agentName"); const navigate = useNavigate();

// Fetch issues assigned to the field agent when the component mounts useEffect(() => {

if (!agentId) {

setError("Field agent not logged in. Please log in to view your dashboard."); return;

}

const fetchIssues = async () => { try {

const response = await fetc[h(`http://localhost:8080/issue/agent/${a](http://localhost:8080/issue/agent/%24)gentId}`); if (response.ok) {

const data = await response.json(); setIssues(data); setFilteredIssues(data);

} else {

setError('Failed to fetch issues.');

}

} catch (err) {

setError('Server error. Please try again later.');

}

};

fetchIssues();

}, [agentId]);

// Fetch the agent's issue count (optional, for display purposes) const [issueCount, setIssueCount] = useState(0);

useEffect(() => {

const fetchIssueCount = async () => { try {

const response = await fetc[h(`http://localhost:8080/field-agent/${a](http://localhost:8080/field-agent/%24)gentId}`); if (response.ok) {

const agent = await response.json(); setIssueCount(agent.issueCount);

}

} catch (err) {

console.error('Error fetching issue count:', err);

}

};

if (agentId) { fetchIssueCount();

}

}, [agentId]);

// Filter issues based on status useEffect(() => {

if (statusFilter === 'All') { setFilteredIssues(issues);

} else {

setFilteredIssues(issues.filter(issue => issue.status === statusFilter));

}

}, [statusFilter, issues]);

// Handle issue click to view details

const handleViewGrievance = (issueId) => { navigate(`/view-grievance/${issueId}`);

};

return (

<Container sx={{ mt: 4 }}>

<Box>

<Typography variant="h4" gutterBottom> Welcome to Your Dashboard, {agentName}!

</Typography>

<Typography variant="body1" gutterBottom> Total Issues Assigned: {issueCount}

</Typography>

<Typography variant="body1" gutterBottom>

View and manage your assigned grievances below.

</Typography>

{/\* Status Filter \*/}

<FormControl sx={{ mb: 2, minWidth: 200 }}>

<InputLabel>Filter by Status</InputLabel>

<Select

value={statusFilter}

onChange={(e) => setStatusFilter(e.target.value)} label="Filter by Status"

>

<MenuItem value="All">All</MenuItem>

<MenuItem value="Pending">Pending</MenuItem>

<MenuItem value="Completed">Completed</MenuItem>

</Select>

</FormControl>

{/\* Error Message \*/}

{error ? (

<Typography color="error" sx={{ mt: 2 }}>

{error}

</Typography>

) : filteredIssues.length === 0 ? (

<Typography sx={{ mt: 2 }}>

No issues assigned for the selected status.

</Typography>

) : (

<Table sx={{ mt: 2 }}>

<TableHead>

<TableRow>

<TableCell>Complaint Type</TableCell>

<TableCell>Issue Description</TableCell>

<TableCell>Preferred Resolution Date</TableCell>

<TableCell>Preferred Resolution Time</TableCell>

<TableCell>Status</TableCell>

<TableCell>User Name</TableCell>

<TableCell>Action</TableCell>

</TableRow>

</TableHead>

<TableBody>

{filteredIssues.map((issue) => (

<TableRow key={issue.id}>

<TableCell>{issue.complaintType}</TableCell>

<TableCell>{issue.issueDescription}</TableCell>

<TableCell>{issue.preferredResolutionDate}</TableCell>

<TableCell>{issue.preferredResolutionTime}</TableCell>

<TableCell>{issue.status}</TableCell>

<TableCell>{issue.user.name}</TableCell>

<TableCell>

<Button variant="outlined" size="small"

onClick={() => handleViewGrievance(issue.id)}

>

View

</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

)}

</Box>

</Container>

);

}

###### Home.js

import React from 'react';

import { Container, Typography, Button, Box } from '@mui/material'; import { useNavigate } from 'react-router-dom';

export default function Home() { const navigate = useNavigate();

return (

<Container sx={{ mt: 4, textAlign: 'center' }}>

<Typography variant="h3" gutterBottom style={{ color: '#1976d2' }}> Welcome to the Field Agent Portal

</Typography>

<Typography variant="h6" gutterBottom> Please log in or register to manage grievances.

</Typography>

<Box sx={{ mt: 4 }}>

<Button variant="contained"

sx={{ backgroundColor: '#1976d2', color: 'white', mr: 2 }} onClick={() => navigate('/login')}

>

Login

</Button>

<Button variant="outlined"

sx={{ color: '#1976d2', borderColor: '#1976d2' }} onClick={() => navigate('/register')}

>

Register

</Button>

</Box>

</Container>

);

}

BACKEND

###### Issuecontroller.java

package com.arjuncodes.studentsystem.controller;

import com.arjuncodes.studentsystem.model.FieldAgent; import com.arjuncodes.studentsystem.model.Issue;

import com.arjuncodes.studentsystem.model.ReopenRequest; import com.arjuncodes.studentsystem.model.User;

import com.arjuncodes.studentsystem.service.FieldAgentService; import com.arjuncodes.studentsystem.service.IssueService; import com.arjuncodes.studentsystem.service.UserService; import com.arjuncodes.studentsystem.service.EmailService; import com.fasterxml.jackson.annotation.JsonFormat;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.format.annotation.DateTimeFormat; import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType; import org.springframework.http.ResponseEntity; import org.springframework.http.HttpHeaders; import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile; import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException; import java.nio.file.Files; import java.time.LocalDate; import java.time.LocalTime;

import java.time.temporal.ChronoUnit; import java.util.List;

import java.util.Optional; import java.util.Random;

import java.util.stream.Collectors;

import com.itextpdf.kernel.pdf.PdfDocument; import com.itextpdf.kernel.pdf.PdfWriter; import com.itextpdf.layout.Document;

import com.itextpdf.layout.element.Paragraph; import com.itextpdf.layout.element.Table;

@RestController @RequestMapping("/issue")

@CrossOrigin(origins = "[http://localhost:3000"](http://localhost:3000/)) public class IssueController {

@Autowired

private IssueService issueService;

@Autowired

private UserService userService;

@Autowired

private FieldAgentService fieldAgentService;

@Autowired

private EmailService emailService;

private static final String UPLOAD\_DIR = "/Users/akahya/grievance\_Uploads/reopen\_images/";

@PostMapping("/submit")

public ResponseEntity<String> submitIssue(@RequestBody IssueRequest issueRequest) { System.out.println("Received issue request: " + issueRequest);

try {

User user = userService.findById(issueRequest.getUserId()); if (user == null) {

return ResponseEntity.badRequest().body("User not found");

}

Issue issue = new Issue();

issue.setUser(user); issue.setComplaintType(issueRequest.getComplaintType()); issue.setIssueDescription(issueRequest.getIssueDescription()); issue.setPreferredResolutionDate(issueRequest.getPreferredResolutionDate()); issue.setPreferredResolutionTime(issueRequest.getPreferredResolutionTime()); issue.setLatitude(issueRequest.getLatitude()); issue.setLongitude(issueRequest.getLongitude());

issue.setStatus("Pending");

System.out.println("Saving issue: " + issue); issueService.saveIssue(issue);

return ResponseEntity.ok("Issue submitted successfully");

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error submitting issue: " + e.getMessage());

}

}

@PutMapping("/initiate-completion/{id}")

public ResponseEntity<String> initiateCompletion(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id);

if (optionalIssue.isPresent()) { Issue issue = optionalIssue.get();

String code = generateRandomCode(); issue.setConfirmationCode(code); issue.setStatus("Waiting for Customer Confirmation");

issueService.saveIssue(issue);

User user = issue.getUser();

String subject = "Confirm Your Complaint Resolution"; String body = "Dear " + user.getName() + ",\n\n" +

"To confirm that your issue '" + issue.getComplaintType() + "' has been resolved, " + "please give this code to the field agent: " + code + "\n\nThank you!";

emailService.sendEmail(user.getEmail(), subject, body);

return ResponseEntity.ok("Confirmation code sent to user");

}

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

@PutMapping("/verify-code/{id}")

public ResponseEntity<String> verifyCode(@PathVariable Long id, @RequestParam String code) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (optionalIssue.isPresent()) {

Issue issue = optionalIssue.get();

if (issue.getConfirmationCode() != null && issue.getConfirmationCode().equals(code)) { issue.setStatus("Completed");

issue.setConfirmationVerified(true); issueService.saveIssue(issue);

String subject = "Issue Successfully Closed";

String body = "Dear " + issue.getUser().getName() + ",\n\nYour complaint has been marked as resolved. " +

"Thank you for confirming!\n\n- Municipal Team"; emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Code verified. Issue marked as completed.");

} else {

return ResponseEntity.badRequest().body("Invalid code");

}

}

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

@DeleteMapping("/{id}")

public ResponseEntity<String> deleteIssueIfPending(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id);

if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

Issue issue = optionalIssue.get();

if (!"Pending".equalsIgnoreCase(issue.getStatus())) {

return ResponseEntity.badRequest().body("Issue cannot be withdrawn as it is already assigned or completed.");

}

issueService.deleteIssueById(id);

return ResponseEntity.ok("Issue withdrawn successfully.");

}

@GetMapping("/all")

public ResponseEntity<List<Issue>> getAllIssues() { return ResponseEntity.ok(issueService.getAllIssues());

}

@GetMapping("/user/{userId}")

public ResponseEntity<List<Issue>> getIssuesByUserId(@PathVariable int userId) { return ResponseEntity.ok(issueService.getIssuesByUserId(userId));

}

@PutMapping("/update-status/{id}")

public ResponseEntity<String> updateIssueStatus(@PathVariable Long id, @RequestParam String status) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (optionalIssue.isPresent()) {

Issue issue = optionalIssue.get(); issue.setStatus(status); issueService.saveIssue(issue);

if ("Completed".equalsIgnoreCase(status)) { User user = issue.getUser();

String subject = "Your Grievance Has Been Resolved"; String body = "Dear " + user.getName() + ",\n\n"

+ "Your grievance titled '" + issue.getComplaintType() + "' has been successfully

resolved.\n\n"

+ "Thank you for your patience.\n\n"

+ "Best regards,\nMunicipal Officer Team";

emailService.sendEmail(user.getEmail(), subject, body);

}

return ResponseEntity.ok("Issue status updated successfully");

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

@GetMapping("/status/{status}")

public ResponseEntity<List<Issue>> getIssuesByStatus(@PathVariable String status) { return ResponseEntity.ok(issueService.getIssuesByStatus(status));

}

@GetMapping("/{id}")

public ResponseEntity<Issue> getIssueById(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id); return optionalIssue.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

@PutMapping("/assign/{id}")

public ResponseEntity<String> assignIssueToAgent(@PathVariable Long id, @RequestParam int agentId) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

FieldAgent agent = fieldAgentService.findById(agentId); if (agent == null) {

return ResponseEntity.badRequest().body("Field agent not found");

}

Issue issue = optionalIssue.get();

if (!issue.getComplaintType().equals(agent.getSpecialization())) {

return ResponseEntity.badRequest().body("Field agent's specialization does not match the complaint type");

}

issue.setAssignedAgent(agent); issue.setStatus("Assigned"); issueService.saveIssue(issue); fieldAgentService.incrementIssueCount(agentId);

return ResponseEntity.ok("Issue assigned successfully");

}

@GetMapping("/agent/{agentId}")

public ResponseEntity<List<Issue>> getIssuesByAssignedAgentId(@PathVariable int agentId) { return ResponseEntity.ok(issueService.getIssuesByAssignedAgentId(agentId));

}

@GetMapping("/filter")

public ResponseEntity<List<Issue>> getIssuesWithFilters(

@RequestParam(required = false) String status, @RequestParam(required = false) String complaintType,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate submissionStartDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate submissionEndDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate resolutionStartDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate resolutionEndDate) {

return ResponseEntity.ok(issueService.getIssuesWithFilters(status, complaintType, submissionStartDate,

submissionEndDate, resolutionStartDate, resolutionEndDate));

}

@PostMapping(value = "/reopen-request", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<String> submitReopenRequest( @RequestParam("issueId") Long issueId, @RequestParam("userId") int userId, @RequestParam("image") MultipartFile image) {

try {

Optional<Issue> optionalIssue = issueService.getIssueById(issueId); if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

Issue issue = optionalIssue.get();

if (!"Completed".equalsIgnoreCase(issue.getStatus())) {

return ResponseEntity.badRequest().body("Only completed issues can be reopened.");

}

User user = userService.findById(userId); if (user == null) {

return ResponseEntity.badRequest().body("User not found");

}

File dir = new File(UPLOAD\_DIR); if (!dir.exists()) {

System.out.println("Attempting to create directory: " + dir.getAbsolutePath()); boolean created = dir.mkdirs();

if (!created) {

System.err.println("Failed to create directory: " + dir.getAbsolutePath());

return ResponseEntity.badRequest().body("Failed to create upload directory: " + UPLOAD\_DIR);

} else {

System.out.println("Directory created successfully: " + dir.getAbsolutePath());

}

} else {

System.out.println("Directory already exists: " + dir.getAbsolutePath());

}

String fileName = System.currentTimeMillis() + "\_" + image.getOriginalFilename(); File file = new File(dir, fileName);

System.out.println("Saving image to: " + file.getAbsolutePath()); image.transferTo(file);

ReopenRequest reopenRequest = new ReopenRequest();

reopenRequest.setIssue(issue); reopenRequest.setUser(user); reopenRequest.setImagePath(file.getPath());

reopenRequest.setPreviousAgentId(issue.getAssignedAgent() != null ? issue.getAssignedAgent().getId() : null);

issueService.saveReopenRequest(reopenRequest);

issue.setStatus("Reopen Requested"); issueService.saveIssue(issue);

return ResponseEntity.ok("Reopen request submitted successfully");

} catch (IOException e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error uploading image: " + e.getMessage());

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error submitting reopen request: " + e.getMessage());

}

}

@GetMapping("/reopen-requests")

public ResponseEntity<List<ReopenRequestDTO>> getAllReopenRequests() { List<ReopenRequest> requests = issueService.getAllReopenRequests(); List<ReopenRequestDTO> dtos = requests.stream().map(request -> {

String previousAgentName = "None";

Integer previousAgentId = request.getPreviousAgentId(); if (previousAgentId != null) {

FieldAgent agent = fieldAgentService.findById(previousAgentId); previousAgentName = (agent != null) ? agent.getName() : "Unknown";

}

return new ReopenRequestDTO( request.getId(), request.getIssue().getId(), request.getUser().getName(), previousAgentName

);

}).collect(Collectors.toList()); return ResponseEntity.ok(dtos);

}

@GetMapping(value = "/reopen-image/{requestId}", produces = MediaType.IMAGE\_JPEG\_VALUE)

public ResponseEntity<byte[]> getReopenImage(@PathVariable Long requestId) { Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

ReopenRequest request = optionalRequest.get(); try {

File file = new File(request.getImagePath()); if (!file.exists()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

byte[] imageBytes = Files.readAllBytes(file.toPath());

return ResponseEntity.ok().contentType(MediaType.IMAGE\_JPEG).body(imageBytes);

} catch (IOException e) {

return ResponseEntity.badRequest().build();

}

}

@PostMapping("/reopen/{requestId}")

public ResponseEntity<String> approveReopenRequest(@PathVariable Long requestId, @RequestBody ReopenApprovalRequest approvalRequest) {

if (requestId == null) {

return ResponseEntity.badRequest().body("Request ID cannot be null or undefined");

}

Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("Reopen request not found for ID: " + requestId);

}

ReopenRequest reopenRequest = optionalRequest.get(); Issue issue = reopenRequest.getIssue();

FieldAgent newAgent = fieldAgentService.findById(approvalRequest.getNewAgentId()); if (newAgent == null) {

return ResponseEntity.badRequest().body("New field agent not found");

}

if (reopenRequest.getPreviousAgentId() != null && newAgent.getId() == reopenRequest.getPreviousAgentId()) {

return ResponseEntity.badRequest().body("Cannot assign the same agent who previously handled this issue");

}

if (!issue.getComplaintType().equals(newAgent.getSpecialization())) {

return ResponseEntity.badRequest().body("New agent's specialization does not match the complaint type");

}

issue.setAssignedAgent(newAgent); issue.setStatus("Assigned"); issueService.saveIssue(issue);

issueService.deleteReopenRequestById(requestId);

String subject = "Your Issue Reopen Request Approved"; String body = "Dear " + issue.getUser().getName() + ",\n\n" +

"Your request to reopen issue '" + issue.getComplaintType() + "' has been approved. " + "A new field agent has been assigned.\n\nThank you!";

emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Issue reopened and assigned to new agent successfully");

}

@PostMapping("/reopen/reject/{requestId}")

public ResponseEntity<String> rejectReopenRequest(@PathVariable Long requestId) { if (requestId == null) {

return ResponseEntity.badRequest().body("Request ID cannot be null or undefined");

}

Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("Reopen request not found for ID: " + requestId);

}

ReopenRequest reopenRequest = optionalRequest.get(); Issue issue = reopenRequest.getIssue();

issue.setStatus("Rejected"); issueService.saveIssue(issue);

issueService.deleteReopenRequestById(requestId);

String subject = "Your Issue Reopen Request Rejected"; String body = "Dear " + issue.getUser().getName() + ",\n\n" +

"Your request to reopen issue '" + issue.getComplaintType() + "' has been reviewed and rejected. " +

"If you have further concerns, please submit a new grievance.\n\nThank you!"; emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Reopen request rejected successfully");

}

@GetMapping("/issues-by-period")

public ResponseEntity<List<Issue>> getIssuesByPeriod(@RequestParam String period) {

List<Issue> issues = issueService.getAllIssues(); LocalDate now = LocalDate.now();

List<Issue> filteredIssues;

if ("weekly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 7)

.collect(Collectors.toList());

} else if ("monthly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 30)

.collect(Collectors.toList());

} else if ("yearly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 365)

.collect(Collectors.toList());

} else {

return ResponseEntity.badRequest().body(null);

}

return ResponseEntity.ok(filteredIssues);

}

@GetMapping("/issues-by-month-year")

public ResponseEntity<List<Issue>> getIssuesByMonthYear( @RequestParam(required = false) Integer month, @RequestParam(required = false) Integer year) { List<Issue> issues = issueService.getAllIssues();

LocalDate now = LocalDate.now();

if (month != null && year != null) { List<Issue> filteredIssues = issues.stream()

.filter(issue -> {

LocalDate issueDate = issue.getSubmissionDate();

return issueDate.getMonthValue() == month && issueDate.getYear() == year;

})

.collect(Collectors.toList());

return ResponseEntity.ok(filteredIssues);

}

return ResponseEntity.ok(issues); // Return all if no filters

}

@GetMapping("/generate-analysis")

public ResponseEntity<byte[]> generateAnalysis() { try {

ByteArrayOutputStream baos = new ByteArrayOutputStream(); PdfWriter writer = new PdfWriter(baos);

PdfDocument pdf = new PdfDocument(writer); Document document = new Document(pdf);

document.add(new Paragraph("Comprehensive Grievance Analysis Report")

.setBold().setFontSize(18));

List<Issue> issues = issueService.getAllIssues(); LocalDate now = LocalDate.now();

String[] periods = {"Weekly", "Monthly", "Yearly"}; int[] days = {7, 30, 365};

for (int i = 0; i < periods.length; i++) { final int daysForPeriod = days[i];

document.add(new Paragraph(periods[i] + " Analysis")

.setBold().setFontSize(14).setMarginTop(10));

// Updated table to include new columns (8 columns total) Table table = new Table(8);

table.addCell("Period"); table.addCell("Total Issues"); table.addCell("Pending"); table.addCell("Assigned"); table.addCell("Waiting for Confirmation"); table.addCell("Completed"); table.addCell("Rejected"); table.addCell("Reopened");

table.addCell("Last " + daysForPeriod + " Days"); List<Issue> filteredIssues = issues.stream()

.filter(issue -> ChronoUnit.DAYS.between(issue.getSubmissionDate(), now) <= daysForPeriod)

.collect(Collectors.toList());

long totalIssues = filteredIssues.size();

long pending = filteredIssues.stream().filter(issue -> "Pending".equals(issue.getStatus())).count();

long assigned = filteredIssues.stream().filter(issue -> "Assigned".equals(issue.getStatus())).count();

long waitingForConfirmation = filteredIssues.stream().filter(issue -> "Waiting for Customer Confirmation".equals(issue.getStatus())).count();

long completed = filteredIssues.stream().filter(issue -> "Completed".equals(issue.getStatus())).count();

long rejected = filteredIssues.stream().filter(issue -> "Rejected".equals(issue.getStatus())).count();

long reopened = filteredIssues.stream().filter(issue -> "Reopen Requested".equals(issue.getStatus())).count();

// Debugging: Log issues that contribute to total but aren't categorized

if (totalIssues > (pending + assigned + waitingForConfirmation + completed + rejected + reopened)) {

filteredIssues.forEach(issue -> {

String status = issue.getStatus() != null ? issue.getStatus() : "null"; System.out.println("Issue ID: " + issue.getId() + ", Status: " + status + ", Submission

Date: " + issue.getSubmissionDate());

});

}

table.addCell(String.valueOf(totalIssues)); table.addCell(String.valueOf(pending)); table.addCell(String.valueOf(assigned)); table.addCell(String.valueOf(waitingForConfirmation)); table.addCell(String.valueOf(completed)); table.addCell(String.valueOf(rejected)); table.addCell(String.valueOf(reopened));

document.add(table);

reopened;

// Verify the counts add up

long sum = pending + assigned + waitingForConfirmation + completed + rejected +

document.add(new Paragraph("Verification: Pending + Assigned + Waiting for

Confirmation + Completed + Rejected + Reopened = " + sum + " (Total: " + totalIssues + ")")

.setFontSize(10).setItalic());

}

document.close();

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION\_PDF); headers.setContentDispositionFormData("attachment",

"comprehensive\_grievance\_analysis.pdf"); headers.setContentLength(baos.size());

return new ResponseEntity<>(baos.toByteArray(), headers, HttpStatus.OK);

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().build();

}

}

private String generateRandomCode() { int length = 6;

String digits = "0123456789"; Random random = new Random();

StringBuilder code = new StringBuilder(length); for (int i = 0; i < length; i++) {

code.append(digits.charAt(random.nextInt(digits.length())));

}

return code.toString();

}

}

// Inner classes (IssueRequest, ReopenApprovalRequest, ReopenRequestDTO) remain unchanged class IssueRequest {

private int userId;

private String complaintType; private String issueDescription;

@JsonFormat(pattern = "yyyy-MM-dd") private LocalDate preferredResolutionDate;

@JsonFormat(pattern = "HH:mm")

private LocalTime preferredResolutionTime;

private Double latitude; private Double longitude;

@Override

public String toString() { return "IssueRequest{" +

"userId=" + userId +

", complaintType='" + complaintType + '\'' +

", issueDescription='" + issueDescription + '\'' +

", preferredResolutionDate=" + preferredResolutionDate + ", preferredResolutionTime=" + preferredResolutionTime + ", latitude=" + latitude +

", longitude=" + longitude + '}';

}

public int getUserId() { return userId; }

public void setUserId(int userId) { this.userId = userId; }

public String getComplaintType() { return complaintType; }

public void setComplaintType(String complaintType) { this.complaintType = complaintType; }

public String getIssueDescription() { return issueDescription; }

public void setIssueDescription(String issueDescription) { this.issueDescription = issueDescription; }

public LocalDate getPreferredResolutionDate() { return preferredResolutionDate; } public void setPreferredResolutionDate(LocalDate preferredResolutionDate) {

this.preferredResolutionDate = preferredResolutionDate; }

public LocalTime getPreferredResolutionTime() { return preferredResolutionTime; } public void setPreferredResolutionTime(LocalTime preferredResolutionTime) {

this.preferredResolutionTime = preferredResolutionTime; }

public Double getLatitude() { return latitude; }

public void setLatitude(Double latitude) { this.latitude = latitude; }

public Double getLongitude() { return longitude; }

public void setLongitude(Double longitude) { this.longitude = longitude; }

}

class ReopenApprovalRequest { private int newAgentId;

public int getNewAgentId() { return newAgentId; }

public void setNewAgentId(int newAgentId) { this.newAgentId = newAgentId; }

}

class ReopenRequestDTO { private Long id;

private Long issueId; private String userName;

private String previousAgentName;

public ReopenRequestDTO(Long id, Long issueId, String userName, String previousAgentName) {

this.id = id; this.issueId = issueId;

this.userName = userName; this.previousAgentName = previousAgentName;

}

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public Long getIssueId() { return issueId; }

public void setIssueId(Long issueId) { this.issueId = issueId; }

public String getUserName() { return userName; }

public void setUserName(String userName) { this.userName = userName; }

public String getPreviousAgentName() { return previousAgentName; }

public void setPreviousAgentName(String previousAgentName) { this.previousAgentName = previousAgentName; }

}

###### Issuemodel.java

import com.arjuncodes.studentsystem.model.FieldAgent; import com.arjuncodes.studentsystem.model.Issue;

import com.arjuncodes.studentsystem.model.ReopenRequest; import com.arjuncodes.studentsystem.model.User;

import com.arjuncodes.studentsystem.service.FieldAgentService; import com.arjuncodes.studentsystem.service.IssueService; import com.arjuncodes.studentsystem.service.UserService; import com.arjuncodes.studentsystem.service.EmailService; import com.fasterxml.jackson.annotation.JsonFormat;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.format.annotation.DateTimeFormat; import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType; import org.springframework.http.ResponseEntity; import org.springframework.http.HttpHeaders;

import org.springframework.web.bind.annotation.\*; import org.springframework.web.multipart.MultipartFile; import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException; import java.nio.file.Files; import java.time.LocalDate; import java.time.LocalTime;

import java.time.temporal.ChronoUnit; import java.util.List;

import java.util.Optional; import java.util.Random;

import java.util.stream.Collectors;

import com.itextpdf.kernel.pdf.PdfDocument; import com.itextpdf.kernel.pdf.PdfWriter; import com.itextpdf.layout.Document;

import com.itextpdf.layout.element.Paragraph; import com.itextpdf.layout.element.Table;

@RestController @RequestMapping("/issue")

@CrossOrigin(origins = "[http://localhost:3000"](http://localhost:3000/)) public class IssueController {

@Autowired

private IssueService issueService;

@Autowired

private UserService userService;

@Autowired

private FieldAgentService fieldAgentService;

@Autowired

private EmailService emailService;

private static final String UPLOAD\_DIR = "/Users/akahya/grievance\_Uploads/reopen\_images/";

@PostMapping("/submit")

public ResponseEntity<String> submitIssue(@RequestBody IssueRequest issueRequest) { System.out.println("Received issue request: " + issueRequest);

try {

User user = userService.findById(issueRequest.getUserId()); if (user == null) {

return ResponseEntity.badRequest().body("User not found");

}

Issue issue = new Issue(); issue.setUser(user);

issue.setComplaintType(issueRequest.getComplaintType()); issue.setIssueDescription(issueRequest.getIssueDescription()); issue.setPreferredResolutionDate(issueRequest.getPreferredResolutionDate()); issue.setPreferredResolutionTime(issueRequest.getPreferredResolutionTime());

issue.setLatitude(issueRequest.getLatitude()); issue.setLongitude(issueRequest.getLongitude()); issue.setStatus("Pending");

System.out.println("Saving issue: " + issue); issueService.saveIssue(issue);

return ResponseEntity.ok("Issue submitted successfully");

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error submitting issue: " + e.getMessage());

}

}

@PutMapping("/initiate-completion/{id}")

public ResponseEntity<String> initiateCompletion(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id);

if (optionalIssue.isPresent()) { Issue issue = optionalIssue.get();

String code = generateRandomCode(); issue.setConfirmationCode(code); issue.setStatus("Waiting for Customer Confirmation");

issueService.saveIssue(issue);

User user = issue.getUser();

String subject = "Confirm Your Complaint Resolution"; String body = "Dear " + user.getName() + ",\n\n" +

"To confirm that your issue '" + issue.getComplaintType() + "' has been resolved, " + "please give this code to the field agent: " + code + "\n\nThank you!";

emailService.sendEmail(user.getEmail(), subject, body);

return ResponseEntity.ok("Confirmation code sent to user");

}

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

@PutMapping("/verify-code/{id}")

public ResponseEntity<String> verifyCode(@PathVariable Long id, @RequestParam String code) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (optionalIssue.isPresent()) {

Issue issue = optionalIssue.get();

if (issue.getConfirmationCode() != null && issue.getConfirmationCode().equals(code)) { issue.setStatus("Completed");

issue.setConfirmationVerified(true); issueService.saveIssue(issue);

String subject = "Issue Successfully Closed";

String body = "Dear " + issue.getUser().getName() + ",\n\nYour complaint has been marked as resolved. " +

"Thank you for confirming!\n\n- Municipal Team"; emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Code verified. Issue marked as completed.");

} else {

return ResponseEntity.badRequest().body("Invalid code");

}

}

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

@DeleteMapping("/{id}")

public ResponseEntity<String> deleteIssueIfPending(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id);

if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

Issue issue = optionalIssue.get();

if (!"Pending".equalsIgnoreCase(issue.getStatus())) {

return ResponseEntity.badRequest().body("Issue cannot be withdrawn as it is already assigned or completed.");

}

issueService.deleteIssueById(id);

return ResponseEntity.ok("Issue withdrawn successfully.");

}

@GetMapping("/all")

public ResponseEntity<List<Issue>> getAllIssues() { return ResponseEntity.ok(issueService.getAllIssues());

}

@GetMapping("/user/{userId}")

public ResponseEntity<List<Issue>> getIssuesByUserId(@PathVariable int userId) { return ResponseEntity.ok(issueService.getIssuesByUserId(userId));

}

@PutMapping("/update-status/{id}")

public ResponseEntity<String> updateIssueStatus(@PathVariable Long id, @RequestParam String status) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (optionalIssue.isPresent()) {

Issue issue = optionalIssue.get(); issue.setStatus(status); issueService.saveIssue(issue);

if ("Completed".equalsIgnoreCase(status)) { User user = issue.getUser();

String subject = "Your Grievance Has Been Resolved"; String body = "Dear " + user.getName() + ",\n\n"

+ "Your grievance titled '" + issue.getComplaintType() + "' has been successfully

resolved.\n\n"

+ "Thank you for your patience.\n\n"

+ "Best regards,\nMunicipal Officer Team";

emailService.sendEmail(user.getEmail(), subject, body);

}

return ResponseEntity.ok("Issue status updated successfully");

} else {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

}

@GetMapping("/status/{status}")

public ResponseEntity<List<Issue>> getIssuesByStatus(@PathVariable String status) { return ResponseEntity.ok(issueService.getIssuesByStatus(status));

}

@GetMapping("/{id}")

public ResponseEntity<Issue> getIssueById(@PathVariable Long id) { Optional<Issue> optionalIssue = issueService.getIssueById(id); return optionalIssue.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.status(HttpStatus.NOT\_FOUND).build());

}

@PutMapping("/assign/{id}")

public ResponseEntity<String> assignIssueToAgent(@PathVariable Long id, @RequestParam int agentId) {

Optional<Issue> optionalIssue = issueService.getIssueById(id); if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

FieldAgent agent = fieldAgentService.findById(agentId); if (agent == null) {

return ResponseEntity.badRequest().body("Field agent not found");

}

Issue issue = optionalIssue.get();

if (!issue.getComplaintType().equals(agent.getSpecialization())) {

return ResponseEntity.badRequest().body("Field agent's specialization does not match the complaint type");

}

issue.setAssignedAgent(agent); issue.setStatus("Assigned"); issueService.saveIssue(issue); fieldAgentService.incrementIssueCount(agentId);

return ResponseEntity.ok("Issue assigned successfully");

}

@GetMapping("/agent/{agentId}")

public ResponseEntity<List<Issue>> getIssuesByAssignedAgentId(@PathVariable int agentId) { return ResponseEntity.ok(issueService.getIssuesByAssignedAgentId(agentId));

}

@GetMapping("/filter")

public ResponseEntity<List<Issue>> getIssuesWithFilters( @RequestParam(required = false) String status, @RequestParam(required = false) String complaintType,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate submissionStartDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate submissionEndDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate resolutionStartDate,

@RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate resolutionEndDate) {

return ResponseEntity.ok(issueService.getIssuesWithFilters(status, complaintType, submissionStartDate,

submissionEndDate, resolutionStartDate, resolutionEndDate));

}

@PostMapping(value = "/reopen-request", consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public ResponseEntity<String> submitReopenRequest( @RequestParam("issueId") Long issueId, @RequestParam("userId") int userId, @RequestParam("image") MultipartFile image) {

try {

Optional<Issue> optionalIssue = issueService.getIssueById(issueId); if (!optionalIssue.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

Issue issue = optionalIssue.get();

if (!"Completed".equalsIgnoreCase(issue.getStatus())) {

return ResponseEntity.badRequest().body("Only completed issues can be reopened.");

}

User user = userService.findById(userId); if (user == null) {

return ResponseEntity.badRequest().body("User not found");

}

File dir = new File(UPLOAD\_DIR); if (!dir.exists()) {

System.out.println("Attempting to create directory: " + dir.getAbsolutePath()); boolean created = dir.mkdirs();

if (!created) {

System.err.println("Failed to create directory: " + dir.getAbsolutePath());

return ResponseEntity.badRequest().body("Failed to create upload directory: " + UPLOAD\_DIR);

} else {

System.out.println("Directory created successfully: " + dir.getAbsolutePath());

}

} else {

System.out.println("Directory already exists: " + dir.getAbsolutePath());

}

String fileName = System.currentTimeMillis() + "\_" + image.getOriginalFilename(); File file = new File(dir, fileName);

System.out.println("Saving image to: " + file.getAbsolutePath()); image.transferTo(file);

ReopenRequest reopenRequest = new ReopenRequest(); reopenRequest.setIssue(issue); reopenRequest.setUser(user); reopenRequest.setImagePath(file.getPath());

reopenRequest.setPreviousAgentId(issue.getAssignedAgent() != null ? issue.getAssignedAgent().getId() : null);

issueService.saveReopenRequest(reopenRequest);

issue.setStatus("Reopen Requested"); issueService.saveIssue(issue);

return ResponseEntity.ok("Reopen request submitted successfully");

} catch (IOException e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error uploading image: " + e.getMessage());

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().body("Error submitting reopen request: " + e.getMessage());

}

}

@GetMapping("/reopen-requests")

public ResponseEntity<List<ReopenRequestDTO>> getAllReopenRequests() { List<ReopenRequest> requests = issueService.getAllReopenRequests(); List<ReopenRequestDTO> dtos = requests.stream().map(request -> {

String previousAgentName = "None";

Integer previousAgentId = request.getPreviousAgentId(); if (previousAgentId != null) {

FieldAgent agent = fieldAgentService.findById(previousAgentId); previousAgentName = (agent != null) ? agent.getName() : "Unknown";

}

return new ReopenRequestDTO( request.getId(),

request.getIssue().getId(), request.getUser().getName(), previousAgentName

);

}).collect(Collectors.toList()); return ResponseEntity.ok(dtos);

}

@GetMapping(value = "/reopen-image/{requestId}", produces = MediaType.IMAGE\_JPEG\_VALUE)

public ResponseEntity<byte[]> getReopenImage(@PathVariable Long requestId) { Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

ReopenRequest request = optionalRequest.get(); try {

File file = new File(request.getImagePath()); if (!file.exists()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).build();

}

byte[] imageBytes = Files.readAllBytes(file.toPath());

return ResponseEntity.ok().contentType(MediaType.IMAGE\_JPEG).body(imageBytes);

} catch (IOException e) {

return ResponseEntity.badRequest().build();

}

}

@PostMapping("/reopen/{requestId}")

public ResponseEntity<String> approveReopenRequest(@PathVariable Long requestId, @RequestBody ReopenApprovalRequest approvalRequest) {

if (requestId == null) {

return ResponseEntity.badRequest().body("Request ID cannot be null or undefined");

}

Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("Reopen request not found for ID: " + requestId);

}

ReopenRequest reopenRequest = optionalRequest.get(); Issue issue = reopenRequest.getIssue();

FieldAgent newAgent = fieldAgentService.findById(approvalRequest.getNewAgentId()); if (newAgent == null) {

return ResponseEntity.badRequest().body("New field agent not found");

}

if (reopenRequest.getPreviousAgentId() != null && newAgent.getId() == reopenRequest.getPreviousAgentId()) {

return ResponseEntity.badRequest().body("Cannot assign the same agent who previously handled this issue");

}

if (!issue.getComplaintType().equals(newAgent.getSpecialization())) {

return ResponseEntity.badRequest().body("New agent's specialization does not match the complaint type");

}

issue.setAssignedAgent(newAgent); issue.setStatus("Assigned"); issueService.saveIssue(issue);

issueService.deleteReopenRequestById(requestId);

String subject = "Your Issue Reopen Request Approved"; String body = "Dear " + issue.getUser().getName() + ",\n\n" +

"Your request to reopen issue '" + issue.getComplaintType() + "' has been approved. " + "A new field agent has been assigned.\n\nThank you!";

emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Issue reopened and assigned to new agent successfully");

}

@PostMapping("/reopen/reject/{requestId}")

public ResponseEntity<String> rejectReopenRequest(@PathVariable Long requestId) { if (requestId == null) {

return ResponseEntity.badRequest().body("Request ID cannot be null or undefined");

}

Optional<ReopenRequest> optionalRequest = issueService.getReopenRequestById(requestId); if (!optionalRequest.isPresent()) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("Reopen request not found for ID: " + requestId);

}

ReopenRequest reopenRequest = optionalRequest.get(); Issue issue = reopenRequest.getIssue();

issue.setStatus("Rejected"); issueService.saveIssue(issue);

issueService.deleteReopenRequestById(requestId);

String subject = "Your Issue Reopen Request Rejected"; String body = "Dear " + issue.getUser().getName() + ",\n\n" +

"Your request to reopen issue '" + issue.getComplaintType() + "' has been reviewed and rejected. " +

"If you have further concerns, please submit a new grievance.\n\nThank you!"; emailService.sendEmail(issue.getUser().getEmail(), subject, body);

return ResponseEntity.ok("Reopen request rejected successfully");

}

@GetMapping("/issues-by-period")

public ResponseEntity<List<Issue>> getIssuesByPeriod(@RequestParam String period) { List<Issue> issues = issueService.getAllIssues();

LocalDate now = LocalDate.now(); List<Issue> filteredIssues;

if ("weekly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 7)

.collect(Collectors.toList());

} else if ("monthly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 30)

.collect(Collectors.toList());

} else if ("yearly".equalsIgnoreCase(period)) { filteredIssues = issues.stream()

.filter(i -> ChronoUnit.DAYS.between(i.getSubmissionDate(), now) <= 365)

.collect(Collectors.toList());

} else {

return ResponseEntity.badRequest().body(null);

}

return ResponseEntity.ok(filteredIssues);

}

@GetMapping("/issues-by-month-year")

public ResponseEntity<List<Issue>> getIssuesByMonthYear( @RequestParam(required = false) Integer month, @RequestParam(required = false) Integer year) { List<Issue> issues = issueService.getAllIssues(); LocalDate now = LocalDate.now();

if (month != null && year != null) { List<Issue> filteredIssues = issues.stream()

.filter(issue -> {

LocalDate issueDate = issue.getSubmissionDate();

return issueDate.getMonthValue() == month && issueDate.getYear() == year;

})

.collect(Collectors.toList());

return ResponseEntity.ok(filteredIssues);

}

return ResponseEntity.ok(issues); // Return all if no filters

}

@GetMapping("/generate-analysis")

public ResponseEntity<byte[]> generateAnalysis() { try {

ByteArrayOutputStream baos = new ByteArrayOutputStream(); PdfWriter writer = new PdfWriter(baos);

PdfDocument pdf = new PdfDocument(writer); Document document = new Document(pdf);

document.add(new Paragraph("Comprehensive Grievance Analysis Report")

.setBold().setFontSize(18));

List<Issue> issues = issueService.getAllIssues(); LocalDate now = LocalDate.now();

String[] periods = {"Weekly", "Monthly", "Yearly"}; int[] days = {7, 30, 365};

for (int i = 0; i < periods.length; i++) { final int daysForPeriod = days[i];

document.add(new Paragraph(periods[i] + " Analysis")

.setBold().setFontSize(14).setMarginTop(10));

// Updated table to include new columns (8 columns total) Table table = new Table(8);

table.addCell("Period"); table.addCell("Total Issues"); table.addCell("Pending"); table.addCell("Assigned"); table.addCell("Waiting for Confirmation"); table.addCell("Completed"); table.addCell("Rejected"); table.addCell("Reopened");

table.addCell("Last " + daysForPeriod + " Days"); List<Issue> filteredIssues = issues.stream()

.filter(issue -> ChronoUnit.DAYS.between(issue.getSubmissionDate(), now) <= daysForPeriod)

.collect(Collectors.toList());

long totalIssues = filteredIssues.size();

long pending = filteredIssues.stream().filter(issue -> "Pending".equals(issue.getStatus())).count();

long assigned = filteredIssues.stream().filter(issue -> "Assigned".equals(issue.getStatus())).count();

long waitingForConfirmation = filteredIssues.stream().filter(issue -> "Waiting for Customer Confirmation".equals(issue.getStatus())).count();

long completed = filteredIssues.stream().filter(issue -> "Completed".equals(issue.getStatus())).count();

long rejected = filteredIssues.stream().filter(issue -> "Rejected".equals(issue.getStatus())).count();

long reopened = filteredIssues.stream().filter(issue -> "Reopen Requested".equals(issue.getStatus())).count();

// Debugging: Log issues that contribute to total but aren't categorized

if (totalIssues > (pending + assigned + waitingForConfirmation + completed + rejected + reopened)) {

filteredIssues.forEach(issue -> {

String status = issue.getStatus() != null ? issue.getStatus() : "null"; System.out.println("Issue ID: " + issue.getId() + ", Status: " + status + ", Submission

Date: " + issue.getSubmissionDate());

});

}

table.addCell(String.valueOf(totalIssues)); table.addCell(String.valueOf(pending)); table.addCell(String.valueOf(assigned)); table.addCell(String.valueOf(waitingForConfirmation)); table.addCell(String.valueOf(completed)); table.addCell(String.valueOf(rejected)); table.addCell(String.valueOf(reopened));

document.add(table);

// Verify the counts add up

reopened;

long sum = pending + assigned + waitingForConfirmation + completed + rejected +

document.add(new Paragraph("Verification: Pending + Assigned + Waiting for

Confirmation + Completed + Rejected + Reopened = " + sum + " (Total: " + totalIssues + ")")

.setFontSize(10).setItalic());

}

document.close();

HttpHeaders headers = new HttpHeaders(); headers.setContentType(MediaType.APPLICATION\_PDF); headers.setContentDispositionFormData("attachment",

"comprehensive\_grievance\_analysis.pdf"); headers.setContentLength(baos.size());

return new ResponseEntity<>(baos.toByteArray(), headers, HttpStatus.OK);

} catch (Exception e) { e.printStackTrace();

return ResponseEntity.badRequest().build();

}

}

private String generateRandomCode() { int length = 6;

String digits = "0123456789"; Random random = new Random();

StringBuilder code = new StringBuilder(length); for (int i = 0; i < length; i++) {

code.append(digits.charAt(random.nextInt(digits.length())));

}

return code.toString();

}

}

// Inner classes (IssueRequest, ReopenApprovalRequest, ReopenRequestDTO) remain unchanged class IssueRequest {

private int userId;

private String complaintType; private String issueDescription;

@JsonFormat(pattern = "yyyy-MM-dd") private LocalDate preferredResolutionDate;

@JsonFormat(pattern = "HH:mm")

private LocalTime preferredResolutionTime;

private Double latitude; private Double longitude;

@Override

public String toString() { return "IssueRequest{" +

"userId=" + userId +

", complaintType='" + complaintType + '\'' +

", issueDescription='" + issueDescription + '\'' +

", preferredResolutionDate=" + preferredResolutionDate +

", preferredResolutionTime=" + preferredResolutionTime + ", latitude=" + latitude +

", longitude=" + longitude + '}';

}

public int getUserId() { return userId; }

public void setUserId(int userId) { this.userId = userId; }

public String getComplaintType() { return complaintType; }

public void setComplaintType(String complaintType) { this.complaintType = complaintType; }

public String getIssueDescription() { return issueDescription; }

public void setIssueDescription(String issueDescription) { this.issueDescription = issueDescription; }

public LocalDate getPreferredResolutionDate() { return preferredResolutionDate; } public void setPreferredResolutionDate(LocalDate preferredResolutionDate) {

this.preferredResolutionDate = preferredResolutionDate; }

public LocalTime getPreferredResolutionTime() { return preferredResolutionTime; } public void setPreferredResolutionTime(LocalTime preferredResolutionTime) {

this.preferredResolutionTime = preferredResolutionTime; }

public Double getLatitude() { return latitude; }

public void setLatitude(Double latitude) { this.latitude = latitude; }

public Double getLongitude() { return longitude; }

public void setLongitude(Double longitude) { this.longitude = longitude; }

}

class ReopenApprovalRequest { private int newAgentId;

public int getNewAgentId() { return newAgentId; }

public void setNewAgentId(int newAgentId) { this.newAgentId = newAgentId; }

}

class ReopenRequestDTO { private Long id;

private Long issueId; private String userName;

private String previousAgentName;

public ReopenRequestDTO(Long id, Long issueId, String userName, String previousAgentName) {

this.id = id; this.issueId = issueId;

this.userName = userName; this.previousAgentName = previousAgentName;

}

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public Long getIssueId() { return issueId; }

public void setIssueId(Long issueId) { this.issueId = issueId; }

public String getUserName() { return userName; }

public void setUserName(String userName) { this.userName = userName; }

public String getPreviousAgentName() { return previousAgentName; }

public void setPreviousAgentName(String previousAgentName) { this.previousAgentName = previousAgentName; }

}

###### Model-issue.java

package com.arjuncodes.studentsystem.model;

import jakarta.persistence.\*; import java.time.LocalDate; import java.time.LocalTime;

@Entity

@Table(name = "issue") public class Issue {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false) private User user;

@ManyToOne

@JoinColumn(name = "assigned\_agent\_id") private FieldAgent assignedAgent;

@Column(name = "complaint\_type") private String complaintType;

@Column(name = "issue\_description") private String issueDescription;

@Column(name = "preferred\_resolution\_date") private LocalDate preferredResolutionDate;

@Column(name = "preferred\_resolution\_time") private LocalTime preferredResolutionTime;

@Column(name = "latitude")

private Double latitude; // New field for GPS

@Column(name = "longitude")

private Double longitude; // New field for GPS

@Column(nullable = false) private String status = "Pending";

@Column(name = "submission\_date", nullable = false) private LocalDate submissionDate;

@Column(name = "confirmation\_code") private String confirmationCode;

@Column(name = "confirmation\_verified") private boolean confirmationVerified = false;

public Issue() {

this.submissionDate = LocalDate.now();

}

// Getters and Setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public User getUser() { return user; }

public void setUser(User user) { this.user = user; }

public FieldAgent getAssignedAgent() { return assignedAgent; }

public void setAssignedAgent(FieldAgent assignedAgent) { this.assignedAgent = assignedAgent;

}

public String getComplaintType() { return complaintType; }

public void setComplaintType(String complaintType) { this.complaintType = complaintType; }

public String getIssueDescription() { return issueDescription; }

public void setIssueDescription(String issueDescription) { this.issueDescription = issueDescription; }

public LocalDate getPreferredResolutionDate() { return preferredResolutionDate; }

public void setPreferredResolutionDate(LocalDate preferredResolutionDate) { this.preferredResolutionDate = preferredResolutionDate; }

public LocalTime getPreferredResolutionTime() { return preferredResolutionTime; } public void setPreferredResolutionTime(LocalTime preferredResolutionTime) {

this.preferredResolutionTime = preferredResolutionTime; }

public Double getLatitude() { return latitude; }

public void setLatitude(Double latitude) { this.latitude = latitude; }

public Double getLongitude() { return longitude; }

public void setLongitude(Double longitude) { this.longitude = longitude; }

public String getStatus() { return status; }

public void setStatus(String status) { this.status = status; }

public LocalDate getSubmissionDate() { return submissionDate; }

public void setSubmissionDate(LocalDate submissionDate) { this.submissionDate = submissionDate; }

public String getConfirmationCode() { return confirmationCode; }

public void setConfirmationCode(String confirmationCode) { this.confirmationCode = confirmationCode; }

public boolean isConfirmationVerified() { return confirmationVerified; }

public void setConfirmationVerified(boolean confirmationVerified) { this.confirmationVerified = confirmationVerified; }

@Override

public String toString() { return "Issue{" +

"id=" + id +

", user=" + (user != null ? user.getId() : null) +

", assignedAgent=" + (assignedAgent != null ? assignedAgent.getId() : null) + ", complaintType='" + complaintType + '\'' +

", issueDescription='" + issueDescription + '\'' +

", preferredResolutionDate=" + preferredResolutionDate + ", preferredResolutionTime=" + preferredResolutionTime + ", latitude=" + latitude +

", longitude=" + longitude + ", status='" + status + '\'' +

", submissionDate=" + submissionDate +

", confirmationCode='" + confirmationCode + '\'' + ", confirmationVerified=" + confirmationVerified + '}';

}

}

###### Issuerepository

package com.arjuncodes.studentsystem.repository; import com.arjuncodes.studentsystem.model.Issue;

import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.time.LocalDate; import java.util.List;

@Repository

public interface IssueRepository extends JpaRepository<Issue, Long> { List<Issue> findByStatus(String status);

List<Issue> findByUserId(int userId); List<Issue> findByAssignedAgentId(int agentId);

List<Issue> findAllByOrderBySubmissionDateAsc(); List<Issue> findAllByOrderBySubmissionDateDesc(); List<Issue> findByComplaintType(String complaintType);

@Query("SELECT i FROM Issue i WHERE i.submissionDate BETWEEN :startDate AND

:endDate")

List<Issue> findBySubmissionDateBetween(@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

@Query("SELECT i FROM Issue i WHERE i.preferredResolutionDate BETWEEN :startDate AND :endDate")

List<Issue> findByPreferredResolutionDateBetween(@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

@Query("SELECT i FROM Issue i JOIN FETCH i.user u WHERE " + "(:status IS NULL OR i.status = :status) AND " +

"(:complaintType IS NULL OR i.complaintType = :complaintType) AND " + "(:submissionStartDate IS NULL OR i.submissionDate >= :submissionStartDate) AND " + "(:submissionEndDate IS NULL OR i.submissionDate <= :submissionEndDate) AND " +

"(:resolutionStartDate IS NULL OR i.preferredResolutionDate >= :resolutionStartDate) AND " +

"(:resolutionEndDate IS NULL OR i.preferredResolutionDate <= :resolutionEndDate) " + "ORDER BY i.submissionDate DESC")

List<Issue> findWithFilters( @Param("status") String status,

@Param("complaintType") String complaintType, @Param("submissionStartDate") LocalDate submissionStartDate, @Param("submissionEndDate") LocalDate submissionEndDate, @Param("resolutionStartDate") LocalDate resolutionStartDate, @Param("resolutionEndDate") LocalDate resolutionEndDate

);

}

###### Issueservice.java

package com.arjuncodes.studentsystem.service;

import com.arjuncodes.studentsystem.model.Issue;

import com.arjuncodes.studentsystem.model.ReopenRequest;

import java.time.LocalDate; import java.util.List;

import java.util.Optional;

public interface IssueService { void saveIssue(Issue issue); List<Issue> getAllIssues();

Optional<Issue> getIssueById(Long id);

List<Issue> getIssuesByStatus(String status); List<Issue> getIssuesByUserId(int userId); List<Issue> getIssuesByAssignedAgentId(int agentId); List<Issue> getIssuesBySubmissionDateAsc(); List<Issue> getIssuesBySubmissionDateDesc();

List<Issue> getIssuesByComplaintType(String complaintType);

List<Issue> getIssuesBySubmissionDateBetween(LocalDate startDate, LocalDate endDate);

List<Issue> getIssuesByPreferredResolutionDateBetween(LocalDate startDate, LocalDate endDate);

List<Issue> getIssuesWithFilters(String status, String complaintType, LocalDate submissionStartDate, LocalDate submissionEndDate, LocalDate resolutionStartDate, LocalDate resolutionEndDate);

void deleteIssueById(Long id);

void saveReopenRequest(ReopenRequest reopenRequest); List<ReopenRequest> getAllReopenRequests(); Optional<ReopenRequest> getReopenRequestById(Long id); void deleteReopenRequestById(Long id);

}

###### Issueserviceimpl.java

package com.arjuncodes.studentsystem.service;

import com.arjuncodes.studentsystem.model.Issue;

import com.arjuncodes.studentsystem.model.ReopenRequest; import com.arjuncodes.studentsystem.repository.IssueRepository;

import com.arjuncodes.studentsystem.repository.ReopenRequestRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.time.LocalDate; import java.util.List;

import java.util.Optional;

@Service

public class IssueServiceImpl implements IssueService { @Autowired

private IssueRepository issueRepository; @Autowired

private ReopenRequestRepository reopenRequestRepository; @Override

public void saveIssue(Issue issue) { issueRepository.save(issue);

}

@Override

public List<Issue> getAllIssues() { return issueRepository.findAll();

}

@Override

public Optional<Issue> getIssueById(Long id) { return issueRepository.findById(id);

}

@Override

public List<Issue> getIssuesByStatus(String status) { return issueRepository.findByStatus(status);

}

@Override

public List<Issue> getIssuesByUserId(int userId) { return issueRepository.findByUserId(userId);

}

@Override

public List<Issue> getIssuesByAssignedAgentId(int agentId) { return issueRepository.findByAssignedAgentId(agentId);

}

@Override

public List<Issue> getIssuesBySubmissionDateAsc() {

return issueRepository.findAllByOrderBySubmissionDateAsc();

}

@Override

public List<Issue> getIssuesBySubmissionDateDesc() {

return issueRepository.findAllByOrderBySubmissionDateDesc();

}

@Override

public List<Issue> getIssuesByComplaintType(String complaintType) {

return issueRepository.findByComplaintType(complaintType);

}

@Override

public List<Issue> getIssuesBySubmissionDateBetween(LocalDate startDate, LocalDate endDate) {

return issueRepository.findBySubmissionDateBetween(startDate, endDate);

}

@Override

public List<Issue> getIssuesByPreferredResolutionDateBetween(LocalDate startDate, LocalDate endDate) {

return issueRepository.findByPreferredResolutionDateBetween(startDate, endDate);

}

@Override

public List<Issue> getIssuesWithFilters(String status, String complaintType, LocalDate submissionStartDate,

LocalDate submissionEndDate, LocalDate resolutionStartDate, LocalDate resolutionEndDate) {

return issueRepository.findWithFilters(status, complaintType, submissionStartDate, submissionEndDate,

resolutionStartDate, resolutionEndDate);

}

public void deleteIssueById(Long id) { issueRepository.deleteById(id);

}

@Override

public void saveReopenRequest(ReopenRequest reopenRequest) { reopenRequestRepository.save(reopenRequest);

}

@Override

public List<ReopenRequest> getAllReopenRequests() { return reopenRequestRepository.findAll();

}

@Override

public Optional<ReopenRequest> getReopenRequestById(Long id) { return reopenRequestRepository.findById(id);

}

@Override

public void deleteReopenRequestById(Long id) { reopenRequestRepository.deleteById(id);

}

}

## CHAPTER 8 CONCLUSION

The grievance management system project has successfully evolved into a robust framework designed to streamline the process of receiving, assigning, resolving, reporting, closing, and reopening complaints within a municipal context. Through the development of detailed DFDs, the system effectively models the flow of data from users submitting complaints to municipal officers and field agents managing resolutions, ensuring a clear delineation of processes such as "Receive Issue," "Assign Issue," "Resolve Issue," "Generate Report," "Close Issue," and "Reopen Issue." The ERD, enhanced with entities like USER, LOGIN, COMPLAINT, MUNICIPAL\_OFFICER, FIELD\_AGENT,ISSUE\_STATUS, REPORT, and REOPEN\_REQUEST,

provides a solid database structure to support these processes, with the addition of a proof image field in REOPEN\_REQUEST addressing the need for evidence-based reopen requests.

The Municipal Officer Dashboard, as visualized in the provided interface, offers a user-friendly platform with tabs for ISSUES, FIELD AGENTS, and REOPEN REQUESTS, along with filters and action buttons like "VIEW/ASSIGN." The integration of a "Generate Analysis" feature enhances decision-making by allowing officers to analyze complaint trends over weekly, monthly, or yearly periods. Testing strategies, including black box and white box test cases, have been developed to validate the system’s functionality, with specific focus on critical workflows such as complaint submission, assignment, resolution, report generation, and reopen request handling. The introduction of a proof image review mechanism for reopen requests (though not yet fully implemented in the interface) marks a significant step toward transparency and accountability.

Overall, the project lays a strong foundation for a functional grievance management system, demonstrating a well-thought-out design that can be implemented and scaled with additional refinement. The collaborative approach to addressing user requirements and technical challenges has resulted in a system that is poised to effectively serve municipal officers, field agents, and citizens alike.

## CHAPTER 9 REFERENCES

###### Books and Academic Resources:

* Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach. McGraw-Hill Education. (For software design and testing methodologies, including DFDs and ERDs.)
* Sommerville, I. (2015). Software Engineering. Pearson. (For system modeling and testing strategies.)

###### Online Tools and Platforms:

* ERDPlus. (n.d.). ERDPlus - Online Entity-Relationship Diagram Tool. Retrieved from https://erdplus.com/. (Used for ERD design and validation.)
* Postman. (n.d.). Postman API Testing Tool. Retrieved from https://[www.postman.com/.](http://www.postman.com/) (For testing backend APIs.)

###### Web Development and Frameworks:

* React Documentation. (n.d.). React - A JavaScript Library for Building User Interfaces.

Retrieved

from https://reactjs.org/.

* Express.js Documentation. (n.d.). Express - Fast, Unopinionated, Minimalist Web Framework for

Node.js. Retrieved from https://expressjs.com/.

* Multer Documentation. (n.d.). Multer - Middleware for Handling multipart/form-data. Retrieved from https://github.com/expressjs/multer.

General Guidelines and Standards:

* IEEE Standards Association. (1998). IEEE 829-1998 - Standard for Software and System Test Documentation. IEEE. (For structuring test cases.)
* W3C. (n.d.). HTML5 Specification. Retrieved from https://[www.w3.org/TR/html5/.](http://www.w3.org/TR/html5/) (For HTML table formatting in test case documentation.)