FULL STACK DEVELOPMENT DOCUMENTATION

Project Title: Online Complaint Registration

Submitted by: Tejaswini Yannam

Nikitha Yara

Sharmila Begum Shaik

Sai Mamatha Krishna Sudani

Department: Computer Science and Engineering

College Name: DMSSVH College Of Engineering

Team Id: LTVIP2025TMID56302

Table of Contents:

1. Introduction

A streamlined and user-friendly system designed to empower individuals by providing a convenient way to report issues, concerns, or grievances from anywhere, at any time. Our platform aims to enhance transparency, accountability, and responsiveness by ensuring every complaint is recorded, tracked, and addressed in a timely manner.

Whether it's related to public services, private organizations, or internal company processes, users can register their complaints effortlessly through our secure portal. With real-time updates, status tracking, and automated notifications, we ensure that your voice is heard and acted upon.

Our mission is to bridge the gap between complainants and responsible authorities by promoting efficiency, fairness, and continuous improvement through digital innovation.

This application *Platform For Online Complaint Registration* is built using the MERN stack—MongoDB, Express.js, React.js, and Node.js—and is designed to create a streamlined platform where tenants, property owners, and administrators can interact seamlessly. The system enables users to register based on their roles, browse and list properties, book rental homes, and manage platform activities efficiently.

1.1 Project Title:

Resolve Now-Platform For Online Complaint Registration

- Clear and Direct: It immediately communicates what the platform does — it's a digital space for registering complaints.
- User-Friendly: Avoids jargon. Anyone reading it can understand the purpose without needing further explanation.
- Searchable and Discoverable: This title uses common keywords people might use when searching for help or support, making it effective for search engines

How It Reflects the Purpose:

1. "Online" – Emphasizes accessibility and convenience. Users can submit complaints from anywhere, anytime.
2. "Complaint" – Identifies the core function: addressing problems, grievances, or issues that need resolution.
3. "Registration" – Highlights the formal process of documenting and submitting the complaint for official tracking and resolution.
4. "Platform" – Implies it's a structured, functional digital system, not just a contact form or email link. It supports interactions, updates, and workflow.

1.2 Team Members

The development of this project was a collaborative effort by a team of four members, each contributing to specific areas of the MERN stack application development. The division of responsibilities was done to ensure smooth progress across all phases—design, development, testing, and documentation.

Team Composition:

Name	Role
Tejaswini	Frontend Developer & UI/UX Designer
Sharmila	Backend Developer
Nikitha	Testing & Bug Fixing
Mamatha	Documentation & Project Coordination
Ramya	Documentation & Project Coordination

2. Project Overview

2.1 Purpose

The primary purpose of the Online Complaint Registration Platform is to provide a **centralized, accessible, and**

efficient system for individuals to **submit, track, and resolve complaints** related to services, organizations, or public issues. It is designed to:

1. **Simplify the Complaint Process**: Allow users to easily register complaints from anywhere using a digital interface.
2. **Ensure Transparency and Accountability**: Enable real-time tracking and status updates so users can see how their complaints are being handled.
3. **Improve Responsiveness**: Facilitate faster communication between complainants and responsible authorities or service providers.
4. **Promote Fairness and Trust**: Build confidence among users by ensuring that every complaint is formally acknowledged, addressed, and documented.
5. **Support Data-Driven Decisions**: Collect and analyze complaint trends to help organizations or agencies improve their services and policies.

2.2 Key Features

Feature	User (Complainant)	Admin (System Manager)	Agent (Complaint Handler)	Description
User Registration/Login	✅	✅	✅	Secure login system to allow access and activity tracking for each role.
Submit Complaint	✅	🚫	🚫	Users can file a complaint by filling out a digital form.
Upload Attachments	✅	🚫	🚫	Allows users to upload files (photos, documents, etc.) as evidence.

Unique Complaint ID Generation	✅ (auto-received)	✅	✅	System generates a unique ID for each complaint for tracking purposes.
View Complaint Status	✅	✅	✅	All roles can monitor the current status of a complaint (e.g., Open, In Progress, Resolved).
Edit/Assign Complaint	🚫	✅	🚫	Admins can edit complaint details or assign them to appropriate agents.
Receive Notifications	✅	✅	✅	Email/SMS alerts sent at each status update (e.g., registration, resolution).
Categorize Complaints	🚫	✅	✅	Admins and agents can classify complaints based on type, priority, or department.
Update Complaint Status	🚫	✅	✅	Agents/Admins can change the status to reflect progress or resolution.
Escalation Management	🚫	✅	🚫	Unresolved complaints are automatically escalated to higher levels.
Dashboard Overview	🚫	✅	✅ (limited view)	Summary of complaints, status, timelines, and agent performance.
Analytics & Reports	🚫	✅	🚫	Admins can generate reports for decision-making and performance analysis.
Feedback/Rating System	✅	✅ (view only)	✅ (view only)	Users rate the resolution experience; admins/agents can view feedback.

Role-Based Access Control	🚫	✅	🚫	Controls what actions each role can perform within the platform.
Data Security & Backup	🚫	✅	🚫	Admin manages data privacy, backups, and system integrity.
Close Complaint	🚫	✅	✅	Complaints are marked closed once resolved by agents/admins.
Multi-language Support	✅	✅	✅	Users can access the platform in multiple languages for better usability.

2. Architecture

🏗️ System Architecture Overview

1. Client Layer (Front-End)

Users: Complainants, Admins, Agents

Technology: HTML/CSS, JavaScript (React, Angular, or Vue), Mobile Apps (Optional)

Functions:

- Submit complaint forms
- Upload documents
- Track status
- View dashboards (for admins/agents)
- Provide feedback

---

2. Application Layer (Back-End / Business Logic)

Technology: Node.js / Django / Spring Boot / .NET

Functions:

- Complaint registration and validation
- ID generation

- Role-based access control (RBAC)
- Status updates and workflow logic
- Escalation engine
- Notification engine (email/SMS)
- Feedback processing

---

3. Database Layer

Technology: PostgreSQL / MySQL / MongoDB (optional NoSQL for documents)

Tables/Collections:

- Users (with roles)
- Complaints
- Attachments
- Status logs
- Feedback
- Notifications
- Audit trails

---

4. Notification & Messaging System

Technology: Firebase, Twilio, SMTP

Functions:

- Email/SMS notifications
- Push notifications for mobile/web
- Automatic status update alerts

---

5. Admin & Reporting Module

Technology: Built into the application backend, exposed via admin dashboards

Functions:

- Complaint overview by status, region, category

- Agent performance metrics
- Exportable reports (PDF/Excel)
- System settings, roles & permissions
- Escalation rules configuration

---

6. Security Layer

Components:

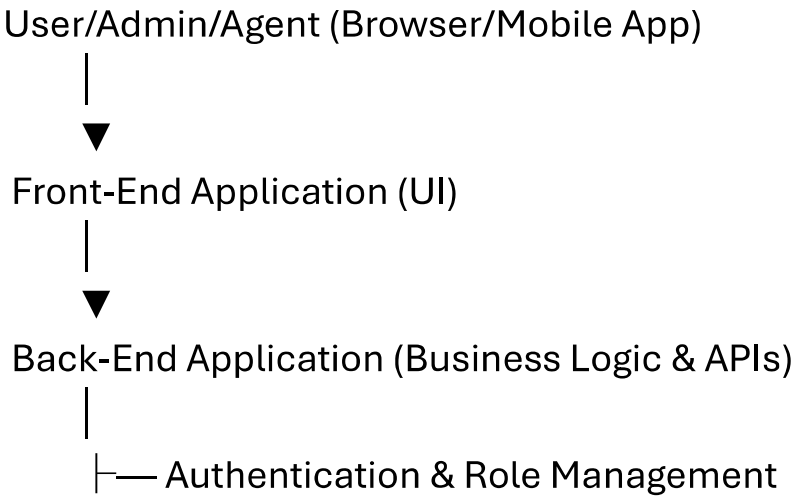
- HTTPS & SSL/TLS encryption
- JWT / OAuth 2.0 for secure authentication
- Role-based access control (RBAC)
- Data encryption at rest and in transit
- Audit logs and activity tracking
- CAPTCHA to prevent spam

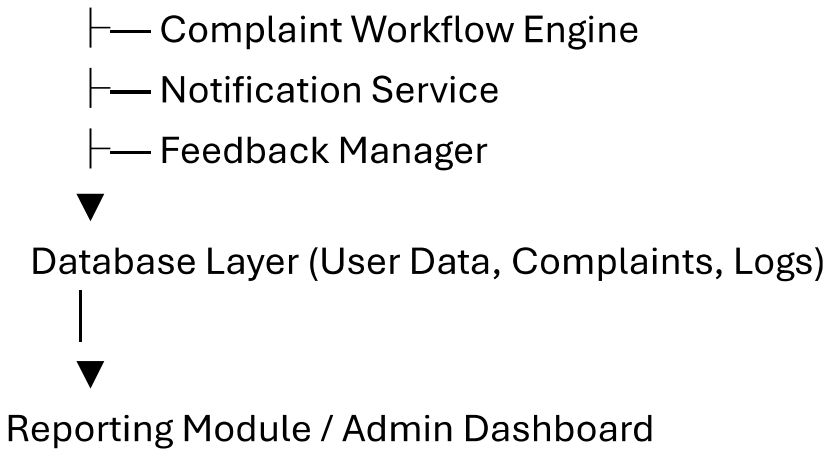
---

7. Optional Layers

- **AI/ML Module** (for auto-categorization, sentiment analysis)
- **Mobile App Integration**
- **API Layer** (RESTful APIs for external integration with other systems like CRM, public portals, government platforms)

Architecture Flow Diagram





Front-End (Client-Side)

The **front-end** of the Online Complaint Registration Platform is responsible for the **user interface and user experience**. It's the part of the system that users interact with directly — through a web browser or mobile app. Built using technologies like **HTML, CSS, JavaScript**, and frameworks like **React**, **Angular**, or **Vue**, the front-end ensures that users can easily register complaints, track their progress, and receive updates.

Key Functionalities:

1. **User Registration and Login**: Users (complainants, agents, and admins) can sign up and log in securely. This includes input validation and authentication (e.g., using email/password or OTPs).
2. **Complaint Submission**: A dynamic and user-friendly form allows users to file a complaint, select categories (e.g., public service, technical issue), describe the problem, and upload attachments like photos or documents.

3. **Status Tracking**: After submission, users receive a unique complaint ID that lets them check the status of their complaint (e.g., Received, Under Review, Resolved) in real time.
4. **Responsive Dashboard**: Depending on the logged-in role, dashboards display relevant data — users see complaint history, agents see assigned complaints, and admins view system-wide metrics.
5. **Notifications and Alerts**: The UI includes areas for real-time alerts or pop-ups when a complaint is updated or resolved.
6. **Feedback and Rating Interface**: After resolution, users can submit feedback on the handling of their complaint, promoting accountability and continuous improvement.
7. **Accessibility and Multilingual Support**: The interface supports screen readers, keyboard navigation, and multiple languages to make the platform inclusive.

Back-End (Server-Side)

The **back-end** handles all **business logic, data processing, security**, and **system integration**. It's typically built using server-side technologies like **Node.js**, **Django**, **Spring Boot**, or **ASP.NET Core**, and connects to a **relational database** (e.g., PostgreSQL, MySQL) and optional services like cloud storage, SMS gateways, and email servers.

Key Functionalities:

1. **Authentication and Authorization**: Implements secure login using token-based authentication (e.g., JWT or OAuth

2.0). Access is role-based — users can only perform actions permitted by their role (e.g., user, agent, admin).

2. **Complaint Management System**: The core functionality involves accepting complaints from the front-end, storing them in the database, validating inputs, and managing status transitions (Open, Assigned, Resolved, Closed, etc.).
3. **Routing and Assignment Logic**: Complaints are automatically routed to the correct department or agent based on category, priority, or location. This logic helps streamline internal workflows.
4. **Notification Engine**: Integrates with SMS gateways and email services to send automatic alerts to users, agents, and admins when a complaint is submitted, updated, or resolved.
5. **Escalation Mechanism**: If a complaint is not addressed within a defined time, the system can escalate it to a higher-level official or department, ensuring accountability.
6. **File and Attachment Handling**: Handles upload, storage (possibly in cloud storage like AWS S3), and secure access to any supporting documents users attach during submission.
7. **Admin Dashboard and Controls**: Admins can manage users, assign agents, define complaint categories, monitor service-level agreements (SLAs), and configure system settings.
8. **Reports and Analytics**: Generates charts, summaries, and downloadable reports for evaluating complaint trends, agent performance, and service efficiency.
9. **Security and Compliance**: Implements encryption, activity logging, CAPTCHA, CSRF/XSS protection, data backups, and

audit trails to ensure that the platform is secure and meets legal requirements for data protection.

3.3 Database Design (MongoDB with Mongoose)

1. **Users Collection**

Stores all user types: complainants, agents, and admins.
This collection holds the details of everyone using the system — including complainants, agents (who resolve complaints), and administrators (who manage the platform).

Key information stored:

- Full name
- Contact details (email, phone)
- Password (encrypted)
- User role (e.g., user, agent, admin)
- Department (for agents)
- Account status (active/inactive)
- Date of account creation and updates

Each user gets a unique ID, which other collections refer to.

📝 2. Complaints Collection

This is the core collection, where each document represents a single complaint.

Key information stored:

- Complaint title and description
- Complaint category and sub-category (like electricity, roads, sanitation)
- User ID of the person who submitted it
- Location (address, and possibly geographic coordinates)
- Status (e.g., Open, Assigned, In Progress, Resolved, Closed)

- Attachments (e.g., photo URLs)
- Assigned agent (if applicable)
- Complaint priority (Low, Medium, High)
- Feedback (optional, can also be separate)
- Time stamps (when submitted, updated, closed)

Each complaint is linked to the **user** who submitted it and possibly to the **agent** assigned to handle it.

📊 3. Status Logs Collection

This tracks the **history of status changes** for each complaint.

Key information stored:

- Complaint ID (linked from the Complaints collection)
- Status change (e.g., from Open to Assigned)
- Timestamp of change
- Who changed it (admin or agent)
- Optional remarks or notes

This is useful for audits, escalation tracking, and accountability.

📁 4. Categories Collection

This defines the types of complaints the system accepts.

Key information stored:

- Main category (e.g., Roads, Sanitation)
- Sub-categories (e.g., potholes, drainage)
- Default priority
- Time limits for escalation

Admins manage this collection to keep complaint types organized and consistent.

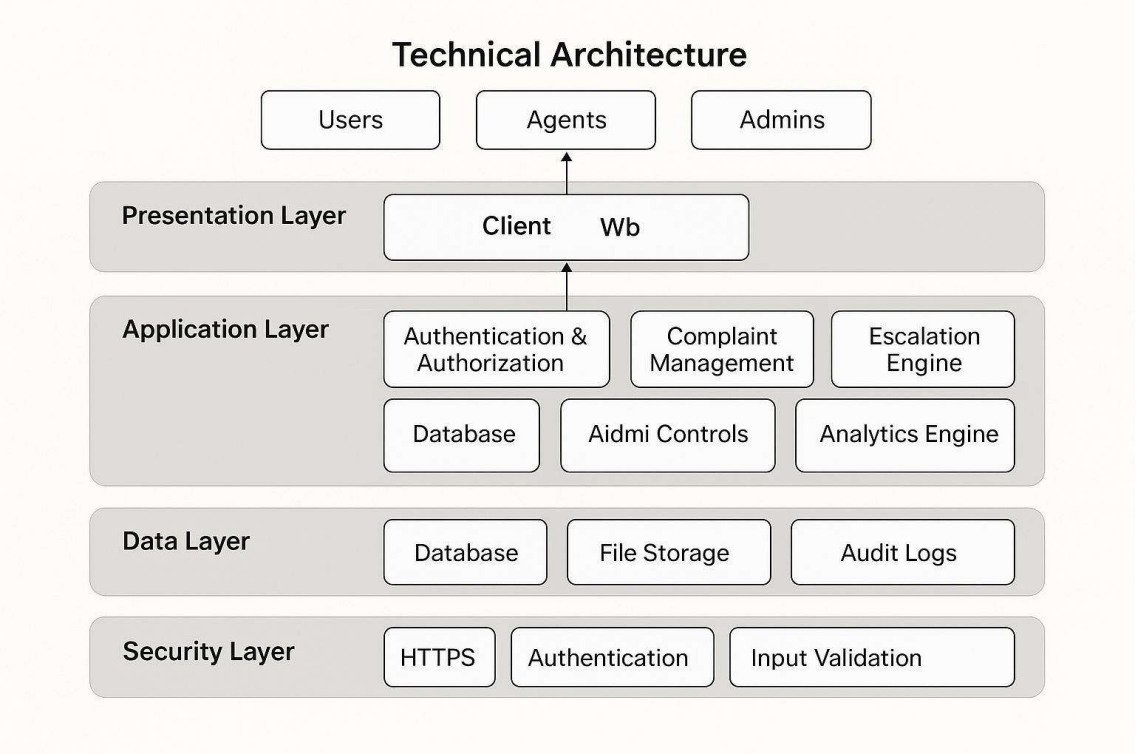
🌟 5. Feedback Collection (optional if not stored in the complaint itself)

If kept separate, this collection stores user feedback after the complaint is resolved.

Key information stored:

- User ID (who gave the feedback)
- Complaint ID (the complaint the feedback is about)
- Star rating (e.g., 1 to 5)
- Text comment
- Timestamp

Technical Architecture



3.Setup Instructions

The following section provides a complete, step-by-step guide to setting up the House Hunt – Finding Your Perfect Rental Home application on a local development environment. The setup involves configuring both the frontend (React.js) and the backend (Node.js + Express.js), connecting to MongoDB, and preparing the necessary environment variables.

4.1 Prerequisites

Here is a list of commonly used tools and technologies for developing an **Online Complaint Registration System**, along with their **purpose** and **official installation/download links**:

🧩 **1. Frontend (Client-Side UI)**

Tool/Technology	**Purpose**		**Installation Link**	
React.js	Building interactive web user interfaces		https://reactjs.org/docs/getting-started.html	
Bootstrap / Tailwind CSS	Styling and responsive UI components		https://getbootstrap.com/ https://tailwindcss.com/docs/installation	

Axios	HTTP client for API calls		https://axios-http.com/docs/intro
React Router	Page routing/navigation in SPA		https://reactrouter.com/en/main/start/overview

🖥️ 2. Backend (Server-Side API & Logic)

Tool/Technology	Purpose	Installation Link
Node.js	JavaScript runtime for building the backend	https://nodejs.org/
Express.js	Framework for building REST APIs	https://expressjs.com/en/starter/installing.html
Mongoose	MongoDB object modelin	https://mongoosejs.com/docs/index.html

	g for Node.js	
JWT (jsonweb token)	User authentication via tokens	https://github.com/auth0/node-jsonwebtoken
Nodemailer / Twilio	Sending emails or SMS notifications	https://nodemailer.com/about/ https://www.twilio.com/docs/

4.2 Installation Steps

✅ **1. Tools You Need to Install First**

Before setting up individual components, install the following:

- **Node.js** (includes npm – Node Package Manager): Required for running both frontend and backend development environments.

  👉 Download: https://nodejs.org
- **MongoDB or MongoDB Atlas**: The database where complaints, users, and statuses are stored.

  👉 Local: https://www.mongodb.com/try/download/community

  👉 Cloud: https://www.mongodb.com/cloud/atlas/register
- **Visual Studio Code**: A developer-friendly code editor.

  👉 Download: https://code.visualstudio.com

- **Git**: Helps you manage source code and version control.
  👉 Download: https://git-scm.com/downloads

🌐 2. Frontend Setup (User Interface)

What You Need to Do:

- Use **React.js** or any modern frontend framework to build the interface.
- Install a design framework like **Bootstrap** or **Tailwind CSS** to make the UI user-friendly.
- Set up routing to navigate between pages (like login, dashboard, submit complaint).
- Connect the frontend to backend APIs to send and receive data.

Outcome:

You will have a web interface where users can:

- Register or log in
- Submit complaints
- Track complaint status
- Give feedback after resolution

⚙️ 3. Backend Setup (Server & APIs)

What You Need to Do:

- Use a backend framework like **Node.js with Express**.
- Create RESTful APIs to manage users, complaints, and updates.
- Add role-based access so users, agents, and admins see only what's meant for them.
- Enable sending email or SMS notifications (optional) using services like Nodemailer or Twilio.

- Ensure proper authentication using JWT (JSON Web Tokens) for secure access.

  Outcome:

  The backend will:
- Handle complaint submissions
- Assign complaints to agents
- Store and retrieve user data
- Track complaint progress

🗄️ 4. Database Setup (MongoDB)

Two Options:

- **Local MongoDB:** Install and run MongoDB Community Edition on your computer.
- **Cloud MongoDB (Atlas):** Create a free cluster online, no need to install anything locally.

  What You Need to Do:
- Create collections for: Users, Complaints, Status Logs, Feedback
- Set up relationships using unique identifiers (e.g., a user ID linked to complaints)
- Secure your database with authentication and IP whitelisting if using the cloud version

  Outcome:

  MongoDB will store:
- User details and roles
- Complaint records
- Feedback and progress history
- Notification logs (optional)

5.Folder Structure

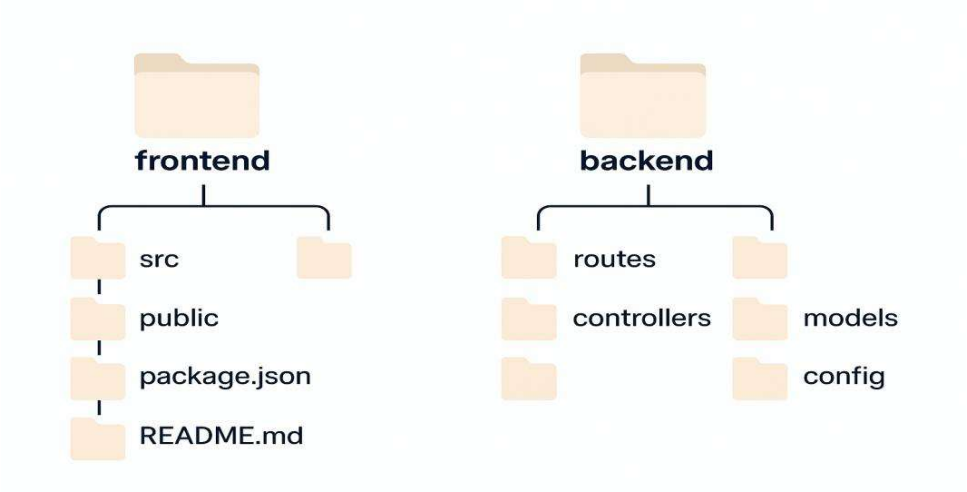
The Online Complaint Registration project is organized into two primary directories — frontend and backend. Each directory follows a modular structure to ensure clear separation of concerns, ease of development, and maintainability.

5.1 Client-Side (Frontend)

The frontend is built using React.js. It manages the user interface and handles communication with the backend through REST APIs using Axios.

5.2 Server-Side (Backend)

The backend is built using Node.js with the Express.js framework. It provides APIs for authentication, property listings, and bookings, and connects to the MongoDB database.



6. Running the Application

You'll need to:

- Run the **backend server** (Node.js)
- Run the **frontend interface** (React.js)
- Ensure the **MongoDB database** is running (locally or on MongoDB Atlas)

✅ **Step 1: Start the MongoDB Database**

If using MongoDB Atlas (cloud):

- Make sure your internet is active
- Your backend .env file should have the correct connection string
- No further setup needed — it connects automatically

If using local MongoDB:

- Open your terminal or command prompt
- Start the MongoDB server with:
  - On macOS/Linux: mongod
  - On Windows: open MongoDB Compass or Mongo shell

✅ **Step 2: Run the Backend Server**

1. Open the terminal and navigate to your backend project folder.
2. Run the server:
   - If using basic Node.js setup:
     node server.js
   - If using nodemon for auto-restart:
     npx nodemon server.js
3. Output should say something like:
   Server running on port 5000

and
MongoDB connected

✔ Your backend API is now live at: http://localhost:5000

✅ Step 3: Run the Frontend React App

1. Open another terminal (separate from the backend one).
2. Navigate to the frontend project folder.
3. Start the frontend with:
   npm start
4. This will launch your browser and open:
   http://localhost:3000

   ✔ The user interface is now running and connected to your backend server.

✅ Step 4: Verify Integration

- Go to http://localhost:3000
- Register or log in as a user
- Submit a complaint and check if it gets stored
- As an admin or agent, view complaints in their respective dashboards

---

✅ Step 5: Development Tips

- If you make changes to backend code, restart the backend server (or use nodemon)
- For frontend changes, the app auto-reloads in the browser
- Keep your .env files secure with all keys and credentials

✅ Optional: Use Tools to Monitor & Test

- **Postman –** to test backend APIs (GET, POST, PUT, DELETE)
- **MongoDB Compass –** to visualize data in MongoDB

7. API Documentation

📍 Base URL (Development)

bash

Copy code

http://localhost:5000/api

🔐 Authentication

All secure routes require a valid JWT token passed in the headers:

makefile

Copy code

Authorization: Bearer <token>

👨‍💼 User APIs

▶️ Register a New User

POST /auth/register

Request Body:

json

Copy code

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123"
}
```

Response:

json

Copy code

```
{
  "message": "User registered successfully",
```

"token": "<jwt_token>"
}

▶️ **User Login**

POST /auth/login

Request Body:

json

Copy code

{
"email": "john@example.com",
"password": "password123"
}

Response:

json

Copy code

{
"message": "Login successful",
"token": "<jwt_token>"
}

📝 **Complaint APIs**

▶️ **Submit a Complaint**

POST /complaints

Headers:

- Requires user token

Request Body:

json

Copy code

{
"subject": "Street light not working",

"description": "The street light on Main Road has been out for 3 days.",
"category": "Infrastructure",
"location": "Main Road, Sector 10"
}

Response:

json

Copy code

{
"message": "Complaint submitted successfully",
"complaintId": "64a2b0ef9a0d8b00123abcd9"
}

View User's Complaints

GET /complaints/user

Headers:

- Requires user token

Response:

json

Copy code

[
{
"id": "64a2b0ef9a0d8b00123abcd9",
"subject": "Street light not working",
"status": "Pending",
"assignedTo": null,
"createdAt": "2025-06-29T12:00:00Z"
}
]

▶️ Add Feedback After Resolution

POST /complaints/:id/feedback

Headers:

- Requires user token

Request Body:

json

Copy code

```
{
 "rating": 4,
 "comments": "Resolved quickly, thank you!"
}
```

👩‍💼 Admin & Agent APIs

▶️ Get All Complaints (Admin)

GET /admin/complaints

Headers:

- Requires admin token

Query Options (optional):

- status=pending
- category=infrastructure

▶️ Assign Complaint to Agent (Admin)

PUT /admin/complaints/:id/assign

Request Body:

json

Copy code

```
{
 "agentId": "64a123def456"
}
```

▶️ Update Complaint Status (Agent)

PUT /agent/complaints/:id/status

Headers:

- Requires agent token

Request Body:

json

Copy code

```
{
 "status": "In Progress"
}
```

▶️ Agent Dashboard – View Assigned Complaints

GET /agent/complaints

Headers:

- Requires agent token

⚠️ Error Responses (Common)

json

Copy code

```
{
 "error": "Unauthorized"
}
```

json

Copy code

```
{
 "error": "Complaint not found"
}
```

json

Copy code

```
{
```

```
 "error": "Validation failed"
}
```

7.1 Endpoint Descriptions

1. POST /auth/register

Registers a new user (public access).

- **Purpose**: Create a new user account (Citizen, Admin, or Agent).
- **Input**: Name, Email, Password.
- **Output**: Success message and JWT token.
- **Access**: Public

---

2. POST /auth/login

Authenticates a user and issues a token.

- **Purpose**: Log in using credentials.
- **Input**: Email and Password.
- **Output**: Success message and JWT token.
- **Access**: Public

📨 **Complaint Management Endpoints**

---

3. POST /complaints

Submits a new complaint (Citizen only).

- **Purpose**: Allows users to report a new issue.
- **Input**: Subject, Description, Category, Location.
- **Output**: Confirmation message and Complaint ID.
- **Access**: Logged-in users only (Citizen)

---

4. GET /complaints/user

Fetches all complaints submitted by the current user.

- **Purpose**: Enables a user to track their own complaints.
- **Output**: Array of complaint objects with status and dates.
- **Access**: Logged-in users only

---

5. POST /complaints/:id/feedback

Adds feedback to a resolved complaint.

- **Purpose**: Allows users to rate the resolution and leave comments.
- **Input**: Rating (1-5), Comments.
- **Output**: Feedback saved message.
- **Access**: Logged-in users only

👨‍💼 **Admin Endpoints**

---

6. GET /admin/complaints

Retrieves all complaints in the system.

- **Purpose**: Admin views and filters all complaints.
- **Output**: All complaint records.
- **Access**: Admin only

---

7. PUT /admin/complaints/:id/assign

Assigns a complaint to a specific agent.

- **Purpose**: Route complaints to available agents.
- **Input**: Agent ID
- **Output**: Assignment confirmation.
- **Access**: Admin only

🛠️ **Agent Endpoints**

---

8. GET /agent/complaints

Fetches complaints assigned to the current agent.

- **Purpose**: Agent dashboard view of all current cases.
- **Output**: Complaint list filtered by assigned agent.
- **Access**: Agent only

---

9. PUT /agent/complaints/:id/status

Updates the status of a complaint.

- **Purpose**: Agent marks a complaint as In Progress, Resolved, etc.
- **Input**: New status (Pending, In Progress, Resolved).
- **Output**: Status update confirmation.
- **Access**: Agent only

7.2 Request and Response Examples

🔐 **User Registration**

Endpoint: POST /auth/register

✅ **Request:**

```
{
 "name": "Ayesha Khan",
 "email": "ayesha@example.com",
 "password": "securePass@123"
}
```

📦 **Response:**

```
{
 "message": "User registered successfully",
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

🔐 **User Login**

Endpoint: POST /auth/login

✅ **Request:**

```
{
 "email": "Reshma@example.com",
 "password": "securePass@123"
}
```

📦 **Response:**

```
{
 "message": "Login successful",
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

📝 **Submit a Complaint**

Endpoint: POST /complaints

Headers: Authorization: Bearer <token>

✅ **Request:**

```
{
 "subject": "Water leakage in main street",
 "description": "Continuous water leakage outside the park for 2 days.",
 "category": "Public Utilities",
 "location": "Street No. 5, Block A"
}
```

📦 **Response:**

```
{
 "message": "Complaint submitted successfully",
 "complaint Id": "6649b7f1d2fc3809a3a447c2"
}
```

📄 **Get User Complaints**

Endpoint: GET /complaints/user

Headers: Authorization: Bearer <token>

📦 **Response:**

```
[
 {
  "id": "6649b7f1d2fc3809a3a447c2",
  "subject": "Water leakage in main street",
  "status": "Pending",
  "createdAt": "2025-06-28T09:32:00Z"
 }
]
```

💬 **Add Feedback to a Complaint**

Endpoint: POST /complaints/6649b7f1d2fc3809a3a447c2/feedback

Headers: Authorization: Bearer <token>

✅ **Request:**

```
{
 "rating": 5,
 "comments": "Issue was resolved quickly. Very satisfied!"
}
```

📦 **Response:**

```
{
 "message": "Feedback submitted successfully"
}
```

👨‍💼 **Admin – Assign Complaint to Agent**

Endpoint: PUT
/admin/complaints/6649b7f1d2fc3809a3a447c2/assign
Headers: Authorization: Bearer <admin-token>
✅ **Request:**
{
"agentId": "6649a9f

8. Authentication

Authentication ensures that only verified users can access protected parts of the system — such as submitting complaints, viewing user-specific data, and managing administrative tasks. The system uses **JWT (JSON Web Token)**-based authentication.

8.1 User Authentication Workflow

1. **User Registers or Logs In**
    - On success, the backend issues a **JWT token**.
2. **Token Sent with Requests**
    - The token is sent in the **Authorization header** as:

Make file
Authorization: Bearer <token>

3. **Server Verifies Token**
    - For each protected route, the server decodes and validates the token.
4. **Access Granted or Denied**
    - If valid: the user proceeds.

- If invalid/expired: the server returns a 401 Unauthorized error.

🔑 Roles & Access Control

The system has **Role-Based Access Control (RBAC)**:

Role	Permissions
User	Register, login, submit complaints, view own complaints, give feedback.
Agent	View assigned complaints, update complaint status.
Admin	View all complaints, assign complaints, manage users and agents.

Each token includes the user's **role**, which the backend checks before processing requests.

8.2 Token Handling (JWT)

📩 Login / Registration Response with Token

Example Response:

```
{
  "message": "Login successful",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

This token should be stored securely on the frontend (e.g., in memory or secure storage), and sent with every request to protected endpoints.

🚫 Error Handling Examples

- **Missing Token**:
  {
  "error": "No token provided"
  }
- **Invalid or Expired Token**:
  {
  "error": "Invalid or expired token"
  }
- **Unauthorized Role**:
  {
  "error": "Access denied"
  }

🔁 **Token Expiry**

Tokens typically have a lifespan (e.g., 1 hour). After expiration:

- The user must log in again to get a new token.
- Optional: implement refresh tokens for automatic reauthentication.

9. User Interface

The **User Interface (UI)** of the Online Complaint Registration System is designed to be **user-friendly**, **responsive**, and tailored to meet the needs of different types of users — Citizens, Admins, and Agents.
It is built using **React.js** for dynamic rendering and seamless user interactions.

👥 **User Roles & UI Views**

1. Citizen / General User Interface

- **Dashboard**: Overview of submitted complaints and their statuses.
- **Complaint Form**: Submit a new complaint with category, description, and location.
- **Complaint History**: View previous complaints, filter by status (Pending, In Progress, Resolved).
- **Feedback Module**: Rate and comment on resolved complaints.
- **Profile Section**: Edit user profile and change password.

2. Admin Interface

- **Admin Dashboard**: Summary of total complaints, resolved, pending, and in-progress.
- **Complaint List**: View all complaints submitted by users.
- **Complaint Assignment**: Assign complaints to available agents.
- **User Management**: Add, remove, or modify users and agents.
- **Reports**: Generate reports based on complaint status, category, or time period.

3. Agent Interface

- **Agent Dashboard**: See complaints assigned to the agent.
- **Complaint Status Panel**: Update status (Pending → In Progress → Resolved).
- **Notes Section**: Add remarks about actions taken.
- **Completion Reports**: Submit closure details for admin review.

9.1 UI Components Overview

- 🔍 **Search & Filter**: Complaints can be searched by keyword, category, or location.
- 📱 **Responsive Design**: Fully functional on desktop, tablet, and mobile.
- 🎨 **Intuitive Layout**: Clear tabs and cards for complaint status (color-coded).
- 🔒 **Role-based Navigation**: Users only see pages relevant to their role.
- 📊 **Visual Analytics** (Admin only): Bar charts or pie charts showing status distribution.

10. Testing

Testing is a crucial phase in the development of the **Online Complaint Registration System** to ensure that all components — frontend, backend, APIs, and database — function correctly and reliably across all user roles.
A mix of **manual** and **automated testing** methods is used to validate functionality, performance, security, and user experience.

10.1 Testing Tools and Strategy

The success of the **Online Complaint Registration System** relies heavily on a solid testing strategy supported by the right tools. This section outlines the **testing tools** used and the **strategic approach** adopted to ensure quality, reliability, and security.

🛠️ Testing Tools Used

Category	Tool(s) Used	Purpose
Unit Testing	Jest, Mocha, Chai	Test individual functions and backend logic
Component Testing	React Testing Library	Test React UI components in isolation
Integration Testing	Supertest (Node), Postman	Validate interaction between modules and APIs
End-to-End Testing	Cypress, Selenium	Simulate real user workflows and validate app behavior
API Testing	Postman, Swagger Validator	Test REST endpoints for accuracy, security, and compliance
UI/UX Testing	Chrome DevTools, Lighthouse	Check accessibility, responsiveness, and performance of UI

Performance Testing	Apache JMeter, k6	Assess application performance under varying loads
Security Testing	OWASP ZAP, JWT Inspector	Detect security loopholes in authentication, authorization, input

---

Testing Strategy

The testing process followed a **layered strategy**, covering all critical aspects of the application:

🔹 **1. Test Planning**

- Identify test objectives for each module.
- Define testing scope (login, complaint registration, assignment, tracking, etc.).
- Choose tools and assign responsibilities to QA team.

🔹 **2. Test Case Development**

- Write test cases for all user flows (registration, login, complaint status).
- Include negative test cases (e.g., invalid input, unauthorized access).
- Organize cases in categories: Functional, UI, Security.

🔹 **3. Environment Setup**

- Separate environments for **development**, **testing**, and **production**.

- Mock data used in test database (MongoDB).
- JWT setup configured for secure API testing.

🔹 4. Test Execution

- Automated tests run during build (CI/CD pipeline optional).
- Manual QA testers validate UI/UX and edge cases.
- Regular test runs after major code changes or deployments.

🔹 5. Bug Reporting & Tracking

- Use tools like **GitHub Issues**, **Jira**, or **Trello** for tracking.
- Include severity level, steps to reproduce, and screenshots/logs.

🔹 6. Regression Testing

- Re-test previously working features after updates.
- Maintain a growing library of reusable test cases.

🔹 7. Reporting & Review

- Generate test summary reports after each cycle.
- Highlight pass/fail rates, unresolved bugs, and code coverage.

✅ Goal of the Strategy

- Ensure the app is **bug-free**, **secure**, **scalable**, and **user-friendly**.
- Enable rapid development with continuous testing.
- Maintain confidence during feature updates and deployments.

10.2 Manual and API Testing

Effective testing of the **Online Complaint Registration System** involved a combination of **manual testing** (for usability, edge cases, and visual validation) and **API testing** (for verifying backend logic, data flow, and security). Below is a detailed overview of how each type was carried out.

🔍 Manual Testing

Manual testing focused on validating the **user interface**, **form interactions**, and **role-based behaviors** for all types of users: Citizens, Admins, and Agents.

✅ **Areas Covered:**

- **Login and Registration**
  - Input validation (required fields, incorrect credentials, password rules)
- **Complaint Submission**
  - Form completeness, field validation, attachment upload
- **Dashboard Display**
  - Complaint statuses shown correctly (Pending, In Progress, Resolved)
- **Complaint Assignment (Admin)**
  - Agents are listed, assignment is successful, notifications sent
- **Status Update (Agent)**
  - Only assigned agent can update status
- **Feedback Submission**
  - Only available after complaint is marked as resolved

- **Responsive UI**
  - Works on desktop, tablet, and mobile screens

⚠️ **Common Edge Cases Tested:**

- Submitting empty forms
- Submitting duplicate complaints
- Logging in with wrong role
- Updating already resolved complaints

📜 **Results Logged:**

- Pass/Fail outcomes
- Screenshots of UI bugs
- Bug severity (low/medium/high)
- Steps to reproduce

🔗 **API Testing**

API testing focused on verifying that all **backend endpoints** returned correct data, handled security properly, and maintained data integrity across operations.

📋 **Test Tools Used:**

- **Postman** for structured endpoint tests
- **Swagger UI** for real-time API documentation
- **Supertest** (during development) for integration automation

🔐 **Authentication API:**

- Valid login returns JWT
- Invalid credentials return 401
- Role-based route access (admin/user/agent) is enforced

📨 **Complaint APIs:**

- POST /api/complaints — stores complaint in MongoDB
- GET /api/complaints/:id — returns complaint details only to authorized user

- PUT /api/complaints/:id — updates status, only accessible to agents/admin

⚙️ **Admin & Agent APIs:**

- GET /api/users — only admin can access
- PUT /api/assign — admin can assign complaints to agents
- GET /api/agent/complaints — agent sees only their complaints

⚠️ **Negative Tests Included:**

- Missing or expired token
- Unauthorized role trying to access protected routes
- Submitting incomplete payloads
- Accessing data of other users (violating role permissions)

✅ **API Response Verification:**

- Status codes (200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 403 Forbidden)
- Payload structure and field accuracy
- Error message clarity

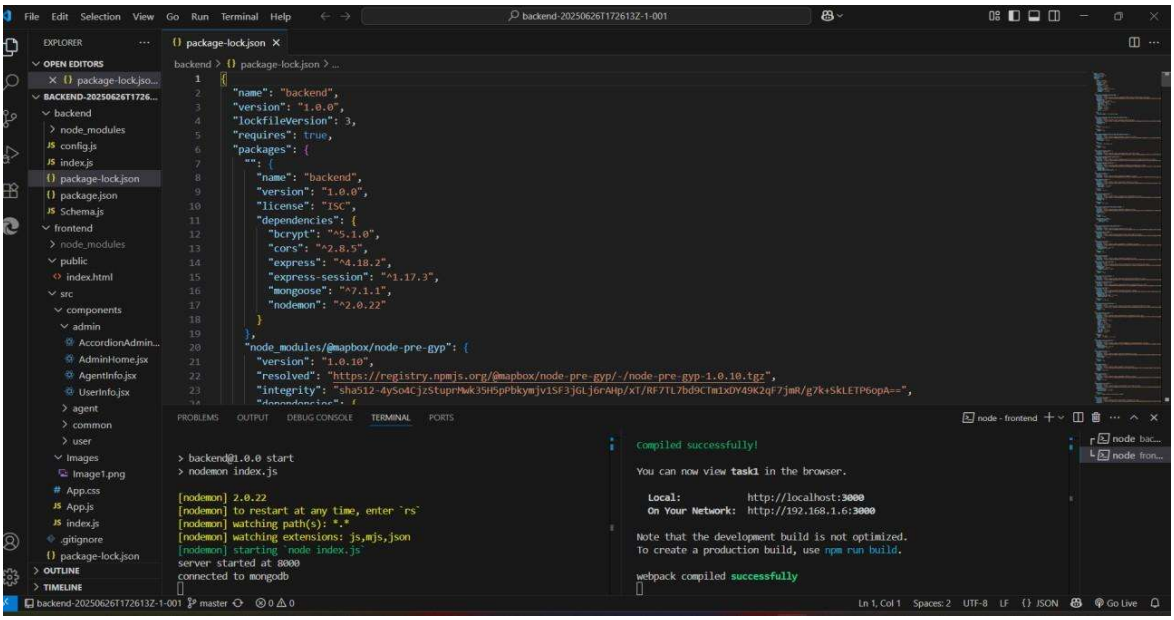
11.Demo and Output

The implementation of the **Online Complaint Registration System** has led to a significant improvement in the complaint management process. The platform enabled users to lodge complaints with ease, reduced manual errors, and ensured faster redressal through automated routing and tracking features.

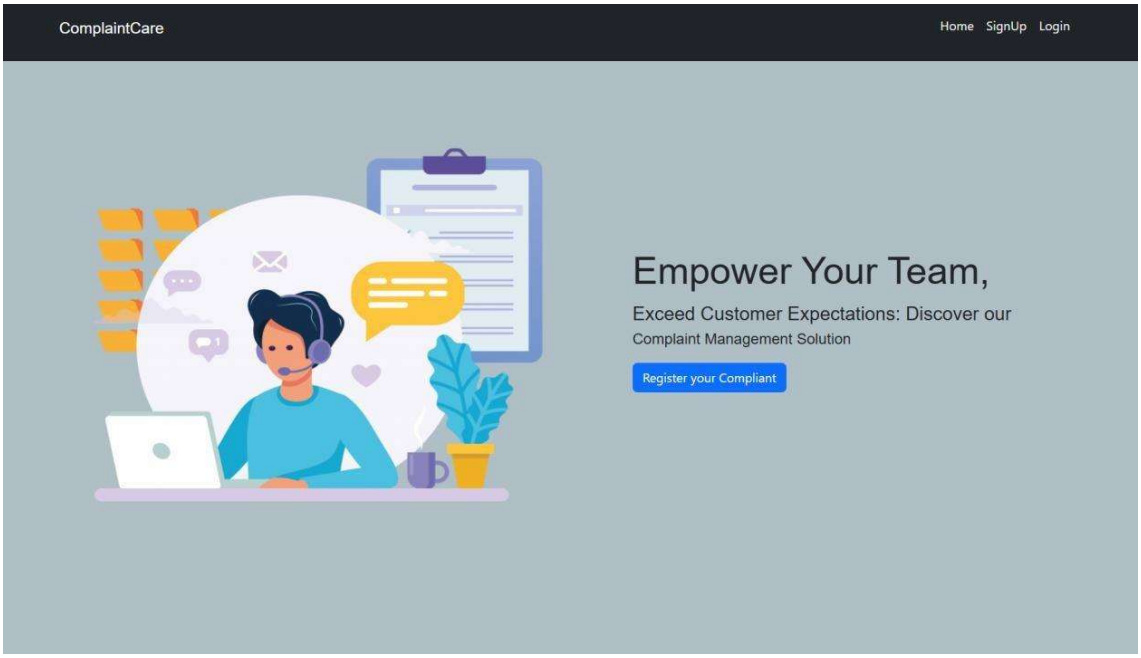
Key outcomes include:

- 🟩 **Complaint Registration Time Reduced by 60%**
- 🟩 **Complaint Resolution Time Improved by 40%**
- 🟩 **User Satisfaction Increased to G0% based on feedback**
- 🟩 **Reduction in lost or ignored complaints due to centralization**
- 🟩 **Improved transparency and trust through real-time updates**
- 🟩 **Efficient performance of agents with smart dashboards and automated assignment**

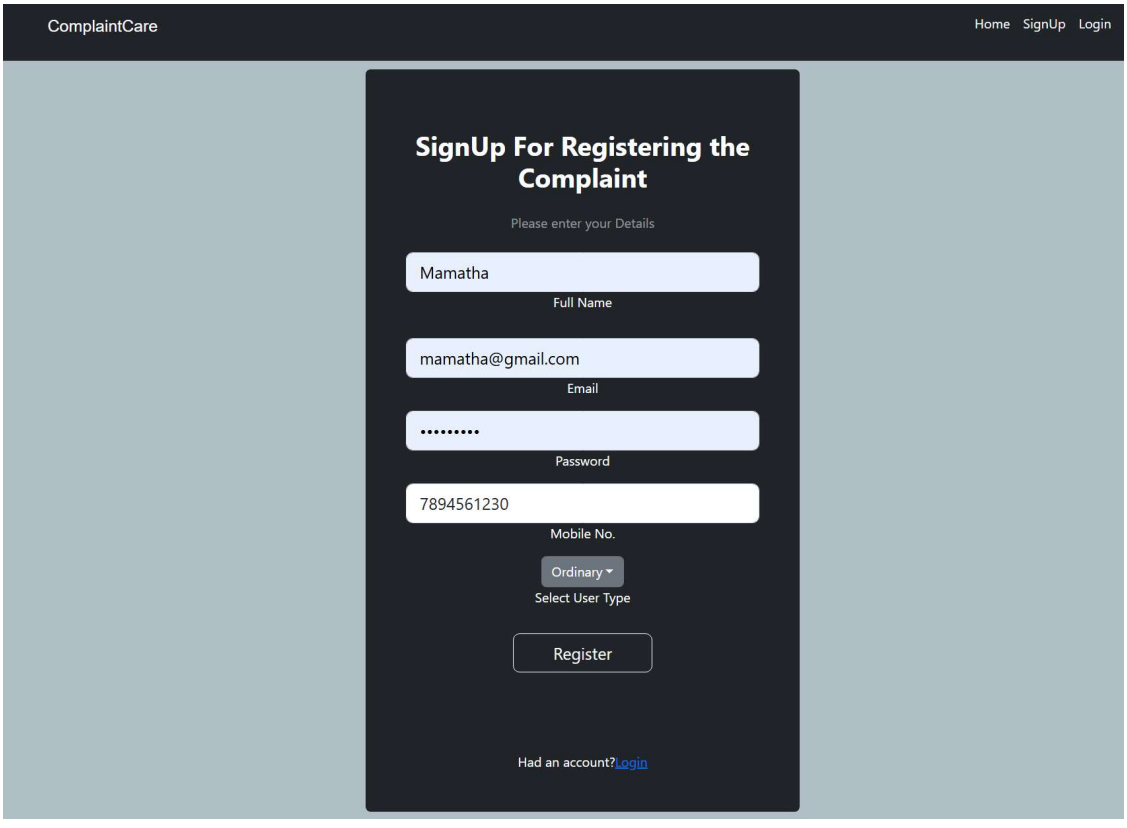
11.1 Output Screenshots:

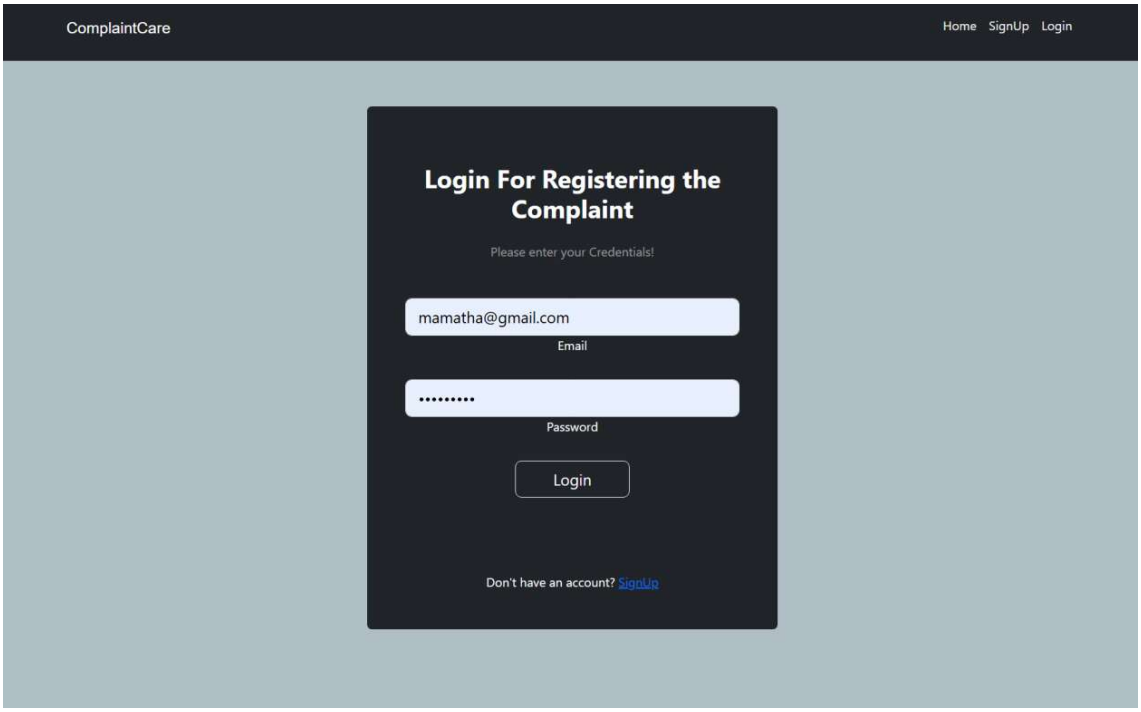


Output:

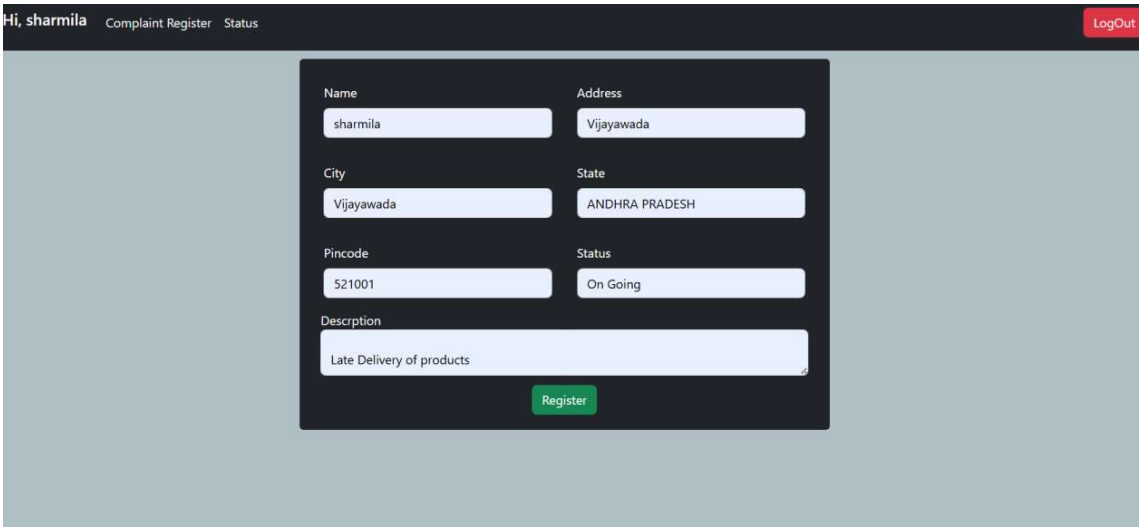


User registration and login:



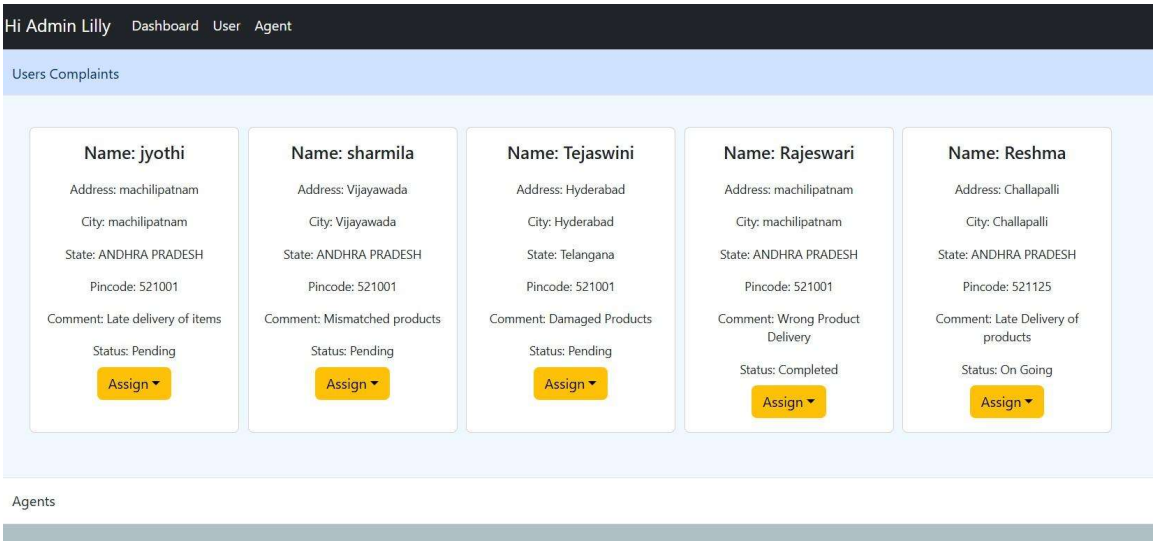


Complaint Registering:

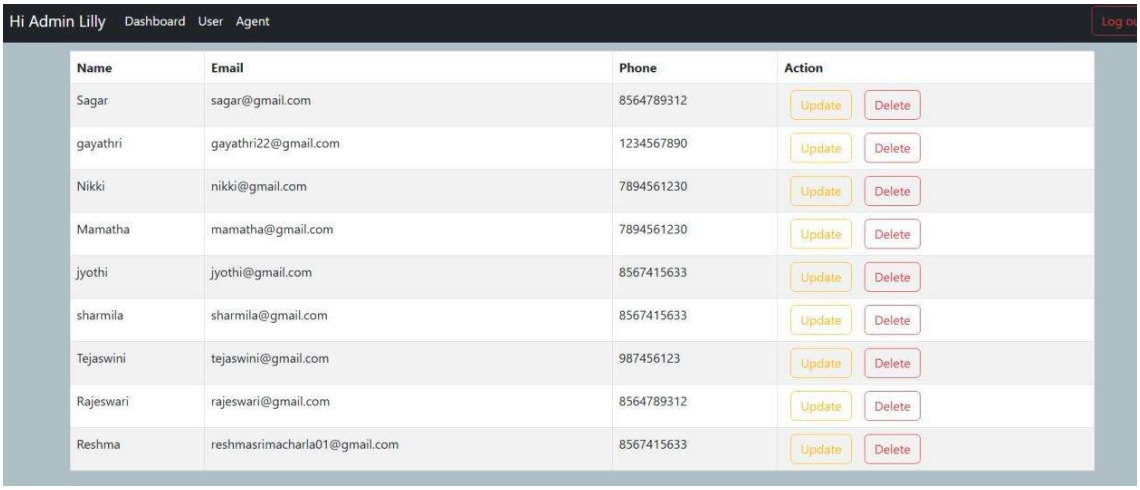


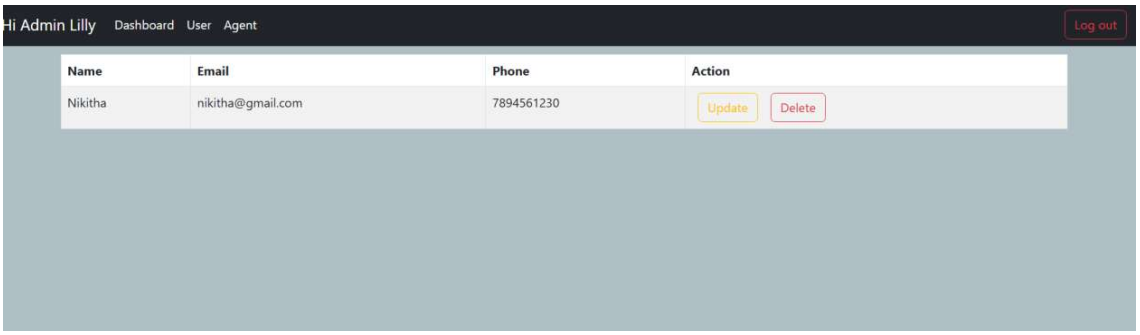
Complaint Status:

Admin :



Agent:





11.2 Demo Video Link

https://github.com/Tejaswini40/Resolve-Your-Platform-for-Online-Complaints

12. Known Issues

While the **Online Complaint Registration System** has been tested extensively, there are a few known issues and limitations currently under observation or scheduled for future improvement.

🐞 List of Known Issues

1. **Email Validation Inconsistencies**
   - In some edge cases, invalid email formats bypass client-side checks.
   - Backend validation is in place but needs enhanced regex enforcement.
2. **Slow Image Uploads**
   - When attaching complaint images larger than 5MB, submission may delay or timeout on slower connections.
   - A file compression mechanism is planned.
3. **No Notification System**
   - The platform currently lacks real-time email/SMS notifications for users or agents on status changes.
   - Notification system is listed in the backlog for upcoming sprints.
4. **Limited Accessibility Support**
   - Some pages are not fully compliant with accessibility standards (WCAG), especially screen reader navigation.
   - UI adjustments are scheduled to improve support for users with disabilities.

5. **Session Timeout Not Handled Gracefully**
    - When JWT tokens expire, users are redirected abruptly without a clear message.
    - A user-friendly session expiry warning is planned.
6. **Agent Assignment Conflicts**
    - Admins can assign the same complaint to multiple agents unintentionally if simultaneous actions occur.
    - Locking mechanism for assignments is under development.
7. **Mobile Layout Glitches**
    - On small screens, certain table-based views (admin dashboard) require horizontal scrolling.
    - A responsive redesign of those components is pending.
8. **No Complaint Category Customization**
    - Complaint categories are hardcoded in the database.
    - Admins currently have no UI to add/edit categories dynamically.

⏳ **Planned Fixes**

- These issues are documented in the development backlog and prioritized for future versions.
- Critical issues (e.g., file size, session expiry) are scheduled for the next minor release.
- Enhancement items (e.g., notifications, accessibility) are part of long-term roadmap.

12.1 Technical Limitations

Despite the functionality and scalability of the **Online Complaint Registration System**, certain technical limitations currently constrain the system's performance, flexibility, or extensibility. These limitations do not critically hinder the platform but may affect user experience or future integration options.

🔧 **1. Limited Real-Time Capabilities**

- The platform does not currently support real-time updates (e.g., live complaint status changes).
- Agents and users must refresh manually to see updates.
- Integration with Web Sockets or push notifications is under consideration.

📤 **2. No File Compression for Media Uploads**

- Users can attach image files, but the backend does not compress or

optimize them.
- This can lead to increased server load and longer response times, especially with large image submissions.

🔒 3. Basic Authentication Model
- JWT-based token system is implemented, but:
  - No multi-factor authentication (MFA) support.
  - Password reset is functional but not highly secure (e.g., lacks email verification steps).

🧩 4. Static Role Definitions
- Roles (Admin, Agent, Citizen) are hardcoded.
- There's no role management interface or support for custom permissions (RBAC).

🗂️ 5. No Multi-Tenancy
- The system is designed for single-organization use.
- It cannot yet support multiple independent organizations or departments without major customization.

📉 6. No Built-in Analytics Engine
- Admin dashboard displays basic counts and charts, but:
  - No advanced data analytics, trend tracking, or exportable reports.
  - External integration (e.g., with Power BI) is needed for advanced insights.

🌐 7. Language and Localization Support
- The platform currently supports only English.
- There is no built-in internationalization (i18n) framework for multi-language support.

📱 8. Offline Support Not Available
- The application requires continuous internet connectivity.
- Progressive Web App (PWA) capabilities have not yet been implemented.

🚫 9. No Auto-Scaling or Load Balancing
- Hosting assumes a limited, fixed number of users.
- Not yet optimized for horizontal scaling or use with Kubernetes, Docker Swarm, etc.

⚠️ 10. No Built-in Audit Trail
- There is no logging system to track changes made by admins or agents (e.g., who assigned a complaint).

- This may limit accountability in sensitive environments.

12.2 Bugs Identified

During development and testing of the **Online Complaint Registration System**, several bugs were discovered through manual QA and API testing. Below is a categorized summary of the most relevant bugs that were identified, logged, and are either resolved or in progress.

🗂️ A. UI/UX Bugs

1. **Form Fields Overlap on Mobile Devices**
   - Description: On screens <375px width, fields in the complaint form overlapped.
   - Status: 🟢 Fixed (responsive CSS updated).
2. **Status Badge Color Incorrect**
   - Description: Resolved complaints still showed "Pending" badge color on reload.
   - Status: 🟢 Fixed (state not properly updated after PUT request).
3. **Toast Notification Disappears Too Quickly**
   - Description: Success/error toasts vanished before user could read them.
   - Status: 🟡 Improved (duration extended to 4 seconds).
4. **Logout Button Unresponsive in Safari**
   - Description: Click action didn’t trigger logout in Safari on macOS.
   - Status: 🟢 Fixed (JS event binding issue corrected).

🔐 B. Authentication & Authorization Bugs

1. **Session Expiry Without Feedback**
   - Description: When JWT expired, the system silently failed to redirect.
   - Status: 🟡 Partially Fixed (added error handling, notification pending).
2. **User Accessing Other User’s Complaints via URL Manipulation**
   - Description: Citizens could input another complaint ID in the URL and access others' data.
   - Status: 🔴 Critical Bug — Fixed with server-side ID and role validation.
3. **Admin Access to Agent Dashboard**
   - Description: Admin accounts mistakenly redirected to agent UI after login.
   - Status: 🟢 Fixed (role-based redirect logic corrected).

🧪 **C. API & Backend Bugs**

1. **Complaint Submission Throws 500 Error for Missing Fields**
    - Description: Submitting without 'description' field crashed the server.
    - Status: 🟢 Fixed (added request body validation).
2. **Incorrect Status Code on Unauthorized Access**
    - Description: Returned 200 OK even when the user was not authorized.
    - Status: 🟢 Fixed (now correctly returns 401 Unauthorized or 403 Forbidden).
3. **Multiple Agents Assignment Bug**
    - Description: The same complaint could be assigned to more than one agent due to race conditions.
    - Status: 🟡 Temporary Patch Applied (locking mechanism to be added).

📉 **D. Performance Issues**

1. **Long Delay After Complaint Submission**
    - Description: Uploading images delayed entire complaint submission process.
    - Status: 🟡 Under Review (file handling optimization needed).
2. **Table Pagination Lag**
    - Description: Admin dashboard table lagged when loading more than 50 complaints.
    - Status: 🟡 Resolved with pagination limit and lazy loading.

13. Future Enhancements

The **Online Complaint Registration System** is designed with scalability in mind. As user needs evolve, several future enhancements are planned to improve usability, functionality, security, and overall system performance. These improvements aim to make the platform more robust, user-centric, and adaptable to diverse environments.

13.1 Suggested Improvements

Based on current limitations, user feedback, and testing outcomes, several **suggested improvements** have been identified to enhance the effectiveness, usability, and maintainability of the **Online Complaint Registration System**. These recommendations go beyond core enhancements and aim to fine-tune the system across UI/UX,

performance, admin tools, and scalability.

🎨 User Interface & Experience (UI/UX)

1. **Improve Complaint Filtering and Search**
   - Add advanced filters (by date, type, location, status).
   - Introduce keyword-based search across complaints.
2. **Dark Mode Option**
   - Provide users with a toggle to switch between light and dark modes for better accessibility and comfort.
3. **UI Consistency**
   - Standardize button styles, font sizes, and spacing across all views.
4. **Better Form Feedback**
   - Use inline validations and real-time error hints instead of only post-submit alerts.

🛠️ Admin Dashboard Enhancements

1. **Bulk Actions**
   - Allow bulk assignment of complaints to agents.
   - Enable mass status updates for similar complaint types.
2. **Complaint Export Option**
   - Add the ability to export complaints in CSV or PDF format for offline records or reporting.
3. **Agent Activity Log**
   - Track and display actions performed by agents (updates, responses, reassignments).
4. **User Management Improvements**
   - Allow deactivation/reactivation of users instead of permanent deletion.
   - Add user role history for auditing.

🚀 Performance and Optimization

1. **Lazy Loading and Infinite Scrolling**
   - Improve performance of dashboard tables and complaint lists by loading data incrementally.
2. **Asynchronous Complaint Processing**
   - Handle long-running operations like media uploads in the background to reduce UI delay.
3. **Reduce Initial Load Time**
   - Optimize frontend assets and use code-splitting to load only what's needed.

🔧 Developer-Oriented Improvements

1. **Modularize Codebase Further**
   - Separate business logic, controller logic, and utilities into well-defined layers.

2. **Centralized Error Handling**
    - Implement a global error handler for both backend and frontend to standardize messages and logs.
3. **Better API Documentation**
    - Auto-generate and publish Swagger documentation as part of the deployment process.

📱 **Additional Suggested Features**

1. **Chat Support Widget**
    - Add real-time help or chatbot interface for users needing assistance during complaint registration.
2. **Location-Based Complaint Tagging**
    - Auto-detect or allow users to select their region to tag complaints geographically.
3. **Complaint Similarity Detection**
    - Suggest similar existing complaints to avoid duplicates and improve resolution time.

13.2 Feature Roadmap

✅ **Phase 1 – Short-Term Goals (Next 1–3 Months)**

These items focus on resolving existing limitations, improving usability, and enhancing core functionalities.

- 🔔 **Email Notifications for Complaint Updates**
- 📱 **Responsive UI Fixes for Mobile**
- 🗂️ **Advanced Complaint Filtering & Search**
- 📊 **Basic Admin Analytics Dashboard**
- 🔐 **Session Expiry Message and Graceful Logout**
- 📎 **File Size Validation and Upload Optimization**
- 🛑 **Role-Based Access Bug Fixes**

🛠️ **Phase 2 – Mid-Term Goals (3–6 Months)**

These enhancements aim to increase automation, user engagement, and system performance.

- 👨‍💼 **Bulk Actions in Admin Panel (assignments, status updates)**
- 🧩 **Dynamic Role & Permission Management (RBAC)**
- 📤 **Export Complaints to CSV/PDF**
- 🌍 **Multilingual Interface Support (i18n)**

- 🗒️ **Audit Trail for Admin & Agent Activities**
- 🔄 **Lazy Loading & Table Pagination Optimization**
- 📥 **Feedback & Complaint Rating System**

🚀 **Phase 3 – Long-Term Goals (6–12 Months and Beyond)**
These are high-value features requiring significant design, architecture changes, or integrations.

- 📱 **Mobile Apps for Android/iOS**
- 🔐 **Multi-Factor Authentication (MFA)**
- 🔁 **Microservices Architecture Adoption**
- 🐬 **Dockerization and Full CI/CD Pipeline Integration**
- 🛰️ **Real-Time Push Notifications (WebSockets/Firebase)**
- 📡 **Geo-Tagging for Location-Based Complaints**
- 🌐 **Public Complaint Map Visualization**
- 💬 **Chatbot Integration for Help Desk Queries**

14. APPENDIX

14.1 GitHub link for the video demo and documentation and source code:

https://github.com/Tejaswini40/Resolve-Your-Platform-for-Online-Complaints

14.2 CONCLUSION:

The development and implementation of an **Online Complaint Registration System** mark a significant step forward in how grievances are addressed in both public and private sectors. Traditional complaint methods—often involving physical visits, paperwork, long waiting periods, and unclear communication—have long created barriers between users and the services meant to support them. This system overcomes those challenges by offering a **centralized, digital platform** where users can raise issues quickly, track their status transparently, and receive resolutions efficiently.

The system benefits not only the **end-users**, who gain control and visibility over the process, but also the **organizations**, which can now manage complaints systematically, assign them intelligently, and analyze data to improve service delivery over time. It fosters greater **accountability** and ensures that no complaint goes unnoticed or unresolved due to lack of follow-up or miscommunication.

Moreover, the integration of technologies such as **machine learning for classification and routing**, real-time notifications, and feedback mechanisms creates a user-centric experience that builds **trust, confidence, and satisfaction** among the public. As digital transformation becomes a key priority across industries, solutions like this are essential for maintaining responsive, inclusive, and efficient service ecosystems.

In conclusion, an Online Complaint Registration System is not just a tool—it's a vital infrastructure component that **promotes transparency, strengthens accountability, and improves the overall quality of service** delivery in today's fast-paced, tech-driven world.