Full Stack Development with MERN

PROJECT - ResolveNow: Your Platform for Online Complaints

TEAM ID

LTVIP2025TMID50838

TEAM MEMBERS

- Kovuru Lakshmaiah – Backend Developer, Team Lead
- M Bhavana – Frontend Developer
- M Hemasri – Frontend Developer

ResolveNow: Your Platform for Online Complaints

Introduction

The Online Complaint Management Platform is a robust full-stack web application developed to simplify and digitize the process of registering, managing, and resolving user complaints. Designed with user convenience and administrative efficiency in mind, this system allows for smooth interaction between customers, service agents, and administrators — all through an intuitive and secure digital interface.
Users can register complaints, track their progress in real time, and communicate with assigned agents, making the resolution process transparent and responsive. Agents are equipped with tools to manage assigned complaints, update statuses, and respond to user queries. Meanwhile, administrators have a comprehensive view of all system operations, enabling them to assign tasks, monitor performance, and ensure effective platform governance.

This platform supports role-based access control with dedicated dashboards for Users, Agents, and Admins, providing each with tailored functionalities. Whether it's uploading complaint-related documents, receiving real-time updates, or engaging in secure in-app chat, the platform ensures a seamless experience across all roles.

The application is built using the MERN stack, comprising MongoDB, Express.js, React.js, and Node.js, with JWT-based authentication for secure access. Frontend design is handled with modern libraries like Material UI or TailwindCSS, offering a responsive and engaging user experience. With scalable architecture and modular components, the platform is designed for maintainability, performance, and future expansion.

In response to the growing need for transparent, trackable, and digitally managed complaint systems, this project provides a secure, real-time solution that enhances user satisfaction and improves service delivery across organizations.

KEY FEATURES

User Registration & Profile Management

- Secure Sign-Up using JWT-based authentication ensures encrypted login and registration for all users.
- Profile Creation allows users to update and maintain contact information and view their complaint history in one place.

Complaint Submission & Tracking

- Intuitive Complaint Form enables users to submit complaints with detailed descriptions and optional file attachments (e.g., images, documents).
- Real-Time Tracking lets users monitor the status of their complaints (e.g., Submitted, In Progress, Resolved) through a dynamic dashboard.

Role-Based Dashboards

- Separate Dashboards for Users, Agents, and Admins, each tailored to the needs of their role for a personalized experience.
- Dynamic Interface built using React.js and TailwindCSS or Material UI for easy navigation and responsiveness.

In-App Messaging System

- Built-In Chat Module allows seamless communication between users and agents for clarifications or updates.
- Real-Time Message Delivery ensures that users and agents stay informed throughout the complaint resolution process.

Agent Tools & Complaint Handling

- Agents Can View & Manage all assigned complaints, update complaint statuses (e.g., In Review, Resolved), and communicate with users directly.
- Dashboard Insights help agents stay organized and prioritize workload effectively.

Admin Controls & Platform Management

- Administrators Oversee the Entire Platform, including user and agent management, complaint assignment, and system monitoring.
- Assign Complaints to Agents efficiently based on availability and type, improving resolution speed and accuracy.
- System Maintenance Tools for dispute handling, user blocking/unblocking, and complaint analytics.

Responsive & Modular Ui

- Built Using Modern Frontend Libraries to ensure mobile-friendly layouts and scalable components.
- Easy to Extend UI Structure, allowing integration of future enhancements like push notifications or advanced filtering.

DESCRIPTION

The **Online Complaint Management Platform** is a user-focused digital solution built to simplify the complaint registration and resolution process across organizations. It bridges the communication gap between users, service agents, and administrators by providing a centralized, role-based system for complaint handling and tracking. Through its clean interface and robust backend, users can file complaints, upload supporting documents, and track resolution progress in real time.

For users, the platform offers secure registration, profile management, and a dynamic dashboard to view complaint status updates. An integrated messaging feature ensures seamless communication with assigned agents for clarifications or updates. Agents benefit from a dedicated dashboard to view and manage complaints, update progress statuses, and directly interact with users. Administrators have full control over the platform — including complaint assignment, user and agent management, and overall system monitoring — ensuring a smooth and well-regulated experience.

The platform is built using **React.js** with **Material UI** or **TailwindCSS** to provide a responsive and modern user experience. **Axios** facilitates efficient communication between the frontend and backend, while **Express.js** and **MongoDB** form the foundation of a scalable and secure backend architecture. **Multer** handles file uploads for complaint attachments, and **JWT authentication** ensures data integrity and secure access across roles.

With its emphasis on accessibility, transparency, and operational efficiency, the Online Complaint Management Platform empowers organizations to deliver faster, more accountable service, while providing users with a convenient and traceable way to address their concerns.

SCENARIO-BASED CASE STUDY

1. **User Registration & Login:**

Ravi, a resident in an urban area, faces irregular garbage collection. He visits the Online Complaint Management Platform and registers as a User by providing his name, email, phone number, and password. Once registered, he logs in securely using JWT-based authentication and accesses his personal dashboard.

2. Complaint Submission:

On the dashboard, Ravi navigates to the “Submit Complaint” section. He fills in a title like “Garbage Not Collected for 3 Days”, writes a description, and uploads a photo of the overflowing bin. After submitting the form, he sees a confirmation and the complaint appears under “My Complaints” with status: Submitted.

3. Real-Time Tracking:

Ravi checks the dashboard regularly. The complaint card shows a timeline and status updates:
Submitted → Assigned to Agent → In Progress → Resolved
Each update is accompanied by a timestamp and notifications.

4. Admin Complaint Assignment:

On the Admin panel, Meena, the platform Admin, sees a new complaint from Ravi. After verifying the complaint details, she assigns it to Agent Raju, who handles that area. Meena changes the complaint status to Assigned, and the system notifies both Ravi (User) and Raju (Agent).

5. Agent Dashboard Actions:

Agent Raju logs into his Agent Dashboard and sees a new assigned complaint. He checks the location, user details, and attached image. He visits the site, initiates the cleanup, and updates the status to In Progress, adding remarks like “Cleanup team dispatched.”

6. In-App Messaging Between User & Agent:

Ravi receives an update and uses the built-in chat to ask Raju about the estimated time of resolution. Raju replies, “Work in progress. It will be done by 5 PM today.” This two-way in-app messaging feature ensures direct, secure communication.

7. Complaint Resolution:

By the evening, Raju completes the work and updates the complaint status to Resolved, attaching a photo of the cleaned area and a closing note. Ravi gets a notification, checks the images, and is satisfied. He marks the complaint as “Closed” and leaves a positive rating.

8. Admin Management & Monitoring:

Admin Meena monitors all complaint activities. She views logs, status updates, and agent performance metrics. In this case, she sees that the complaint was resolved in under 24 hours — an ideal benchmark. She uses analytics to track such metrics for monthly reports.

9. Handling User Feedback Or Reopening:

If Ravi had not been satisfied, he could have clicked “Reopen Complaint,” which would reset the complaint to In Progress. The Admin would then review the issue and reassign or escalate as needed. This feature ensures accountability and user satisfaction.

10. Platform Governance & Maintenance (Admin Role):

- Behind the scenes, the Admin handles:
- Approving new agent registrations
- Managing user and agent accounts
- Monitoring all chats and complaint flows
- Handling system disputes
- Generating performance analytics and reports
- Ensuring platform security and compliance with data protection policies

Summary of Role-Based Workflows:

Role	Core Responsibilities
User	Register/Login, Submit Complaints, Track Status, Chat with Agent, Rate/Close Complaint
Agent	View Assigned Complaints, Update Status, Upload Images, Chat with Users
Admin	Assign Complaints, Manage Users/Agents, Monitor Platform, Resolve Disputes, View Reports

TECHNICAL ARCHITECTURE

The Online Complaint Management Platform is designed using a modern client-server architecture, ensuring a scalable, secure, and responsive experience for users, agents, and administrators. The system is structured with a clear separation between frontend and backend responsibilities, promoting maintainability and performance.

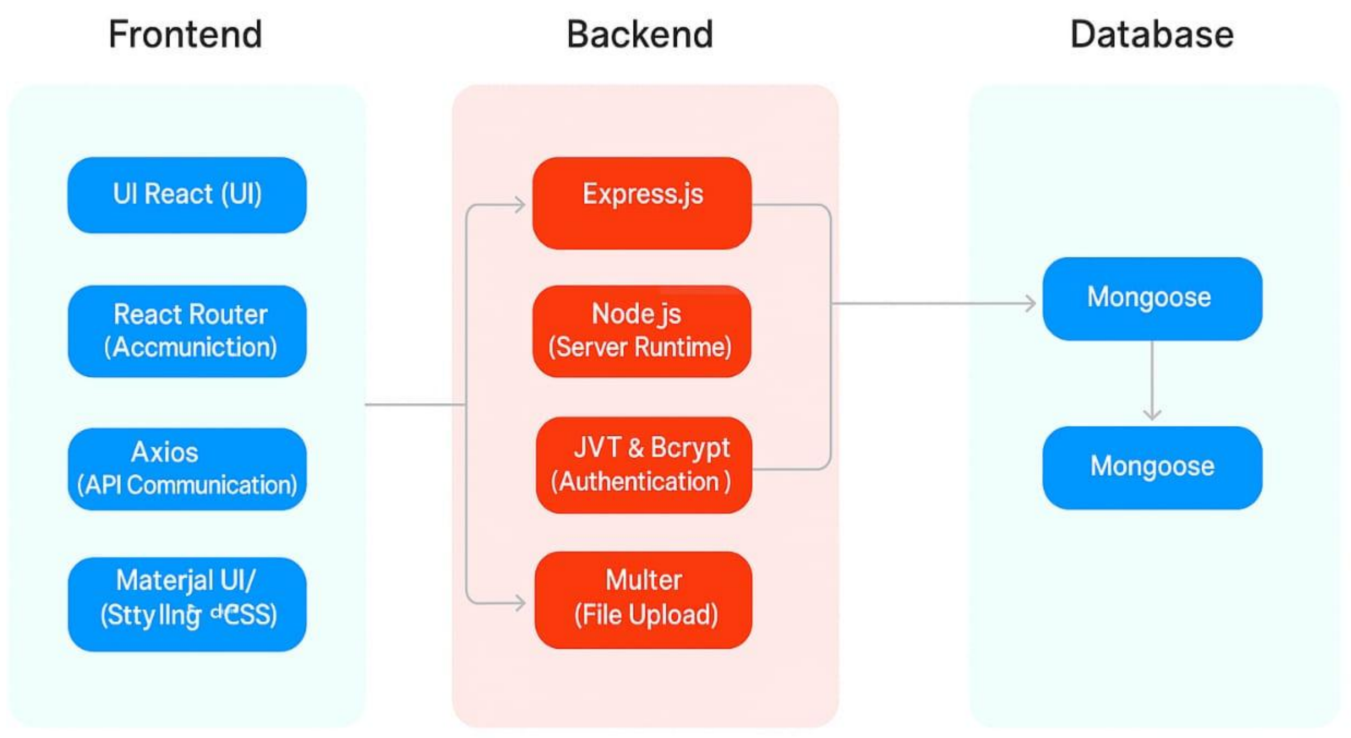
The frontend is developed using React.js, enhanced with either Material UI or TailwindCSS to deliver a clean, responsive, and user-friendly interface. Navigation is handled via React Router, and real-time communication with the backend is managed using Axios, enabling efficient API interactions for data submission, retrieval, and updates.

The backend is powered by Node.js and Express.js, handling core business logic, request routing, complaint processing, role-based data management, and communication flows. User data, complaint records, status updates, and messaging logs are stored securely in MongoDB, a flexible NoSQL database integrated using Mongoose ORM.

For authentication and session security, the platform implements JWT (JSON Web Tokens) for token-based access control and bcrypt for encrypting user passwords. This ensures secure login and protects sensitive data across the system. Multer is used to handle file uploads, enabling users and agents to attach supporting documents or images with complaints.

The system enforces Role-Based Access Control (RBAC) to restrict and manage feature access across Users, Agents, and Admins. Each role operates within its own dashboard environment, accessing only the functionality relevant to their responsibilities.

Performance optimization is supported through modular API design, efficient MongoDB querying, and potential load balancing strategies for handling increased platform activity. The architecture supports future integrations like real-time push notifications, analytics dashboards, and multi-language support.
With this robust and scalable technical architecture, the platform ensures seamless interaction, secure data flow, and a foundation built for maintainability and future enhancement.


Frontend
Backend
Database
UI React (UI)
React Router (Accmuniction)
Axios (API Communication)
Materjal UI/ (Sttyllng dCSS)
Express.js
Node js (Server Runtime)
JVT & Bcrypt (Authentication)
Multer (File Upload)
Mongoose
Mongoose

Frontend Technologies

- React.js with CSS: Used for building a responsive and interactive user interface that adapts seamlessly across devices and screen sizes. CSS enhances the design with appealing layouts for users, agents, and admins.
- Axios: A promise-based HTTP client used to make API requests to the backend. It ensures efficient and consistent communication between the frontend and server for operations like complaint submission, status updates, and admin actions.

Backend Framework

- Express.js: A minimal and flexible Node.js web application framework used for handling all API routing, middleware, and server-side logic. It supports RESTful services for complaint management, user/agent registration, and admin controls.
- Node.js: Powers the backend runtime environment for high-performance and scalable server-side processing.

Database And Authentication

- MongoDB: A NoSQL database used to store user profiles, complaint records, agent details, and admin data. Its flexible document structure supports real-time updates and efficient queries.
- Mongoose: An ODM (Object Data Modeling) library for MongoDB, used for schema-based modeling and validation.
- JWT (JSON Web Tokens): Used for secure user authentication and role-based session handling. Tokens are passed with each request to ensure authorized access to protected routes.
- Bcrypt: Used for hashing and salting passwords before storing them in the database, adding an essential layer of data security.

Admin Panel & Governance

- Admin Dashboard: Allows the admin to manage user and agent accounts, assign complaints, monitor statuses, generate reports, and handle escalations.
- Role-Based Access Control (RBAC): Ensures specific access levels for users (User, Agent, Admin), maintaining data privacy and controlling permissions across the platform.

Scalability And Performance

- MongoDB Atlas (Optional Cloud Hosting): Supports auto-scaling and global replication for better availability and performance.
- Load Balancing: Distributes incoming requests across multiple servers (or instances) to ensure consistent app performance.
- Caching (Optional with Redis or In-Memory): Used to store frequently accessed data like status filters or user sessions to reduce DB load and improve response time.

Time Management And Scheduling

- Moment.js / Day.js: Used for handling date/time formats, complaint registration timestamps, resolution deadlines, and sorting complaint history based on time.

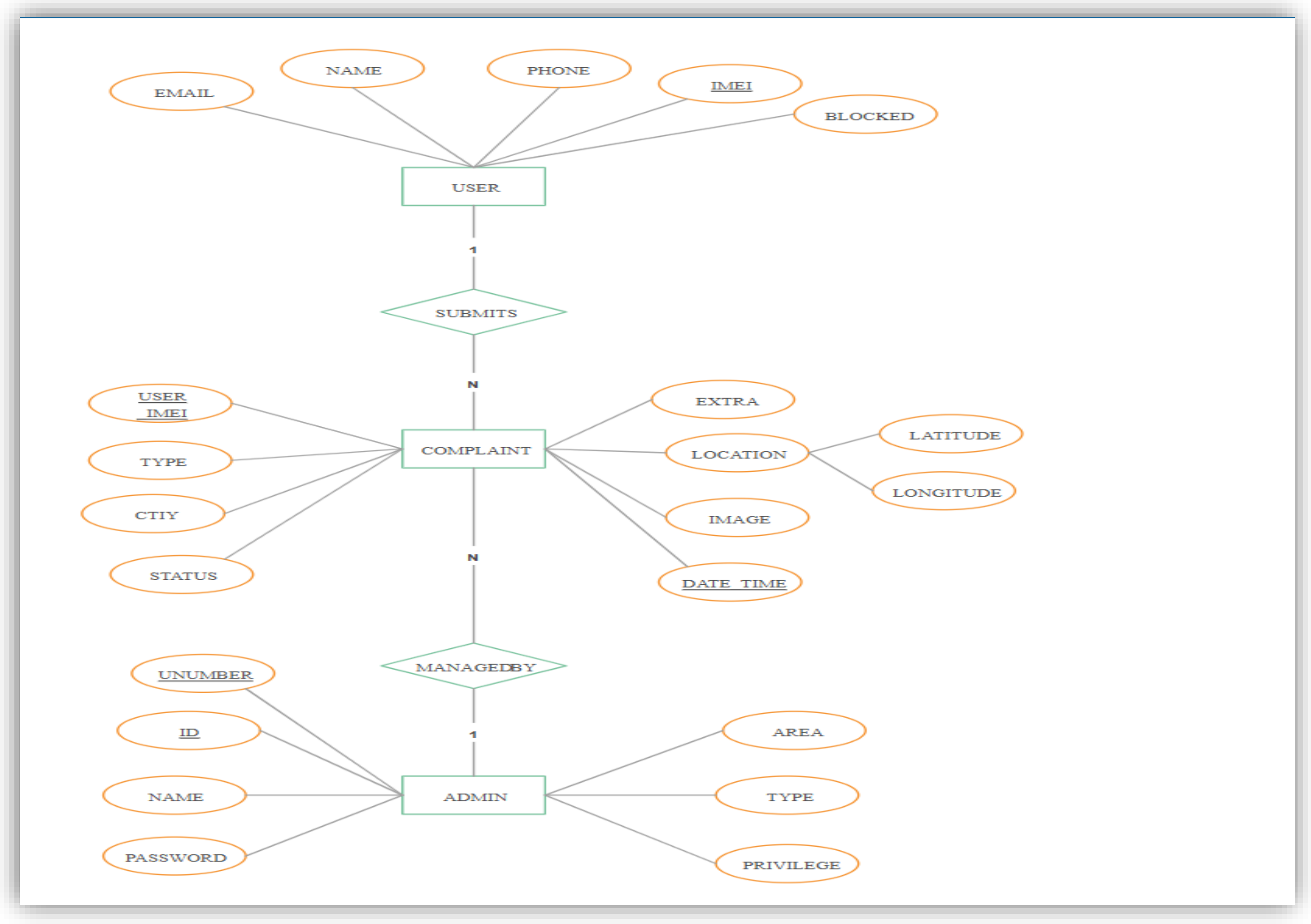
Security Features

- **HTTPS:** All data transmission between client and server is secured using SSL encryption to prevent data interception.
- **Input Validation & Sanitization:** All form data (e.g., complaint descriptions, contact info) is validated and sanitized to prevent attacks like SQL injection or XSS.
- **Data Encryption**: Sensitive complaint data and user credentials are encrypted both in transit and in storage, ensuring compliance with data protection regulations.

Notifications And Reminders

- Email/SMS Notifications: Users and agents receive automated messages for complaint registration, status updates (Accepted, In Progress, Resolved), or rejections. Admins may get alerts for unassigned or delayed complaints.
- Dashboard Alerts: Real-time dashboard notifications for logged-in users to monitor their complaint status and agent actions.

ER-DAIGRAM:-



- Users Can File Complaints
  Each user (complainant) can file multiple complaints. A one-to-many relationship exists between User and Complaint.
- Complaints Belong to Categories
  Every complaint is classified under a specific Category or Department (like electricity, water, etc.), helping in better organization and assignment.
- Admins Handle Complaints
  Admins (or staff) are responsible for responding to complaints. They view complaints, send replies, and update statuses.
- Messages Track Interactions
  A complaint can have multiple Messages, which include replies or updates from either the User or Admin—enabling a full communication history.
- Status Management
  Each complaint has a Status (like Pending, Resolved, In Progress), helping track its lifecycle from creation to closure.
- Complaint History and Reposting
  The system may include a RepostFlag or indicator to show if the user has resubmitted a complaint—useful for identifying unresolved or repeated issues.

PRE-REQUISITES

NODE.JS AND NPM

- **Node.js** is a JavaScript runtime that allows you to run JavaScript code on the server-side. It provides a scalable platform for network applications.
- **npm (Node Package Manager)** is required to install libraries and manage dependencies.
- **Download Node.js**: Node.js Download
- **Installation Instructions**: Installation Guide
- Run the following to initialize your project and create a package.json file: `npm init -y`

EXPRESS.JS

- Express.js is a web application framework for Node.js that helps build APIs and backend services.
- Install Express to manage routing and endpoints:
  ```
  npm install express
  ```

MONGODB

- MongoDB is a NoSQL database that stores data in a flexible, JSON-like format—ideal for user profiles, appointments, etc.
- **Download MongoDB**: MongoDB Download
- **Installation Guide**: MongoDB Manual

REACT.JS

- React.js is a front-end JavaScript library for building dynamic and reusable user interfaces.
- Create a new React app:
  ```
  npx create-react-app your-app-name
  ```
- React Docs

HTML, CSS, AND JAVASCRIPT

- Basic knowledge of **HTML** (structure), **CSS** (styling), and **JavaScript** (functionality) is required for building the user interface.

DATABASE CONNECTIVITY (MONGOOSE)

- Mongoose is an Object-Document Mapping (ODM) tool that connects Node.js applications to MongoDB.
- Simplifies CRUD operations and schema modeling.
- Install using:
  ```
  npm install mongoose
  ```
- Mongoose Guide

FRONTEND FRAMEWORKS AND LIBRARIES

- **React.js** is used for the client-side interface.
- You may optionally use:
  - **Material UI**:
    ```
    npm install @mui/material @emotion/react @emotion/styled
    ```
  - **Bootstrap**:
    ```
    npm install bootstrap
    ```

SETUP & INSTALLATION INSTRUCTIONS

CLONE THE PROJECT REPOSITORY

```
git clone https://github.com/Bhavana029/online-complaint-system.git
cd your-repo-name
```

INSTALL DEPENDENCIES

Frontend:

```
cd frontend
npm install
```

Backend:

```
cd backend
npm install
```

START THE DEVELOPMENT SERVERS

Backend:

```
node index.js
```

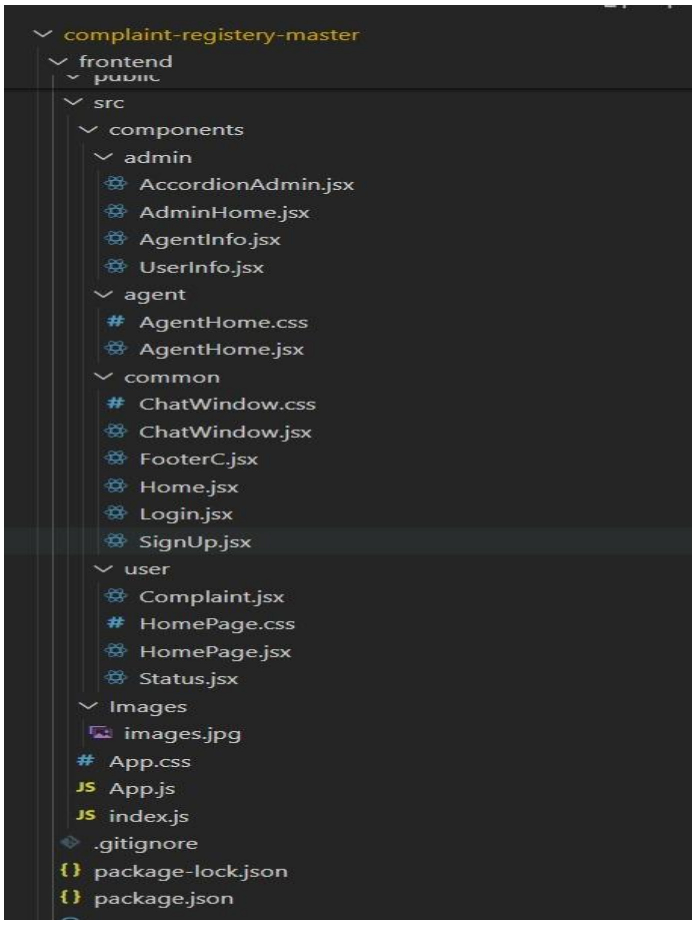
Frontend:

```
npm start
```

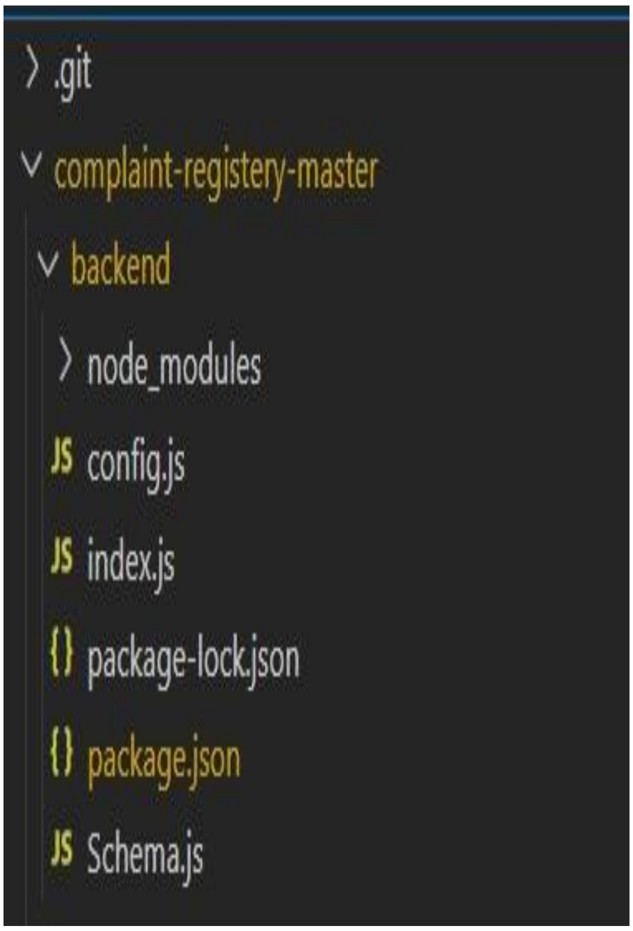
ACCESS THE APPLICATION

- **Frontend**: Open http://localhost:3000 in your browser.
- **Backend**: Check backend API at http://localhost:8001

PROJECT STRUCTURE :



```
complaint-registery-master
  frontend
    public
    src
      components
        admin
          AccordionAdmin.jsx
          AdminHome.jsx
          AgentInfo.jsx
          UserInfo.jsx
        agent
          AgentHome.css
          AgentHome.jsx
        common
          ChatWindow.css
          ChatWindow.jsx
          FooterC.jsx
          Home.jsx
          Login.jsx
          SignUp.jsx
        user
          Complaint.jsx
          HomePage.css
          HomePage.jsx
          Status.jsx
      Images
        images.jpg
      App.css
      App.js
      index.js
    .gitignore
    package-lock.json
    package.json
```



```
> .git
complaint-registery-master
  backend
    > node_modules
    config.js
    index.js
    package-lock.json
    package.json
    Schema.js
```

PROJECT FOLDER STRUCTURE

```
complaint-registry-master/
├── backend/
│   ├── config.js
│   ├── index.js
│   ├── Schema.js
│   ├── package.json
├── frontend/
│   ├── public/
│   └── src/
│       ├── components/
│       │   ├── admin/
│       │   ├── agent/
│       │   ├── common/
│       │   └── user/
│       ├── Images/
│       ├── App.js
│       ├── index.js
├── package.json (frontend)
```

FRONTEND (React.js)

- Role-based UI for Admin, Agent, and User.
- Routing via React Router for login, signup, dashboard, etc.
- State Management to track sessions and roles.
- Components Folder Includes:
  - admin: Admin-related views like AdminHome, UserInfo
  - agent: Agent-specific pages like AgentHome
  - common: Shared UI (e.g., Login, SignUp, ChatWindow)
  - user: Complaint view, HomePage, Status
- Styling using CSS & Ant Design

BACKEND (Node.js + Express.js)

- Handles:
  - Authentication (login/signup)
  - Complaint registration and status tracking
  - Admin & Agent role-based routing
- Files:
  - index.js: Main server file
  - Schema.js: Mongoose schema definitions
  - config.js: Database connection setup
- API Endpoints:
  - /register, /login, /complaints, /status, etc.

USER ROLES

USER (Complainant)

- Register/Login

- Submit complaints
- View status (Pending / Resolved)
- Chat with agents/admin
- Cancel or update complaint

AGENT

- View assigned complaints
- Update status
- Chat with users
- Forward to admin if needed

ADMIN

- Monitor entire platform
- Manage users and agents
- Approve, escalate, or close complaints
- View user and agent details

SETUP & CONFIGURATION

FRONTEND

```
cd complaint-registry-master/frontend
npm install
npm start
# Runs on http://localhost:3000
```

BACKEND

```
cd complaint-registry-master/backend
npm install
npm start
# Runs on http://localhost:5000 or configured port
```

DATABASE (MongoDB)

- Create .env file in /backend:

```
env
MONGO_URI=mongodb://localhost:27017/complaintDB
PORT=5000
JWT_SECRET=your_jwt_secret
```

RUNNING BOTH SERVERS TOGETHER

Optionally install:

```
npm install concurrently --save-dev
```

In root package.json:

```
json
CopyEdit
"scripts": {
  "start": "concurrently \"npm run server\" \"npm run client\"",
  "server": "node backend/index.js",
  "client": "npm start --prefix frontend"
}
```

Run:

```
npm start
```

FINAL TESTING

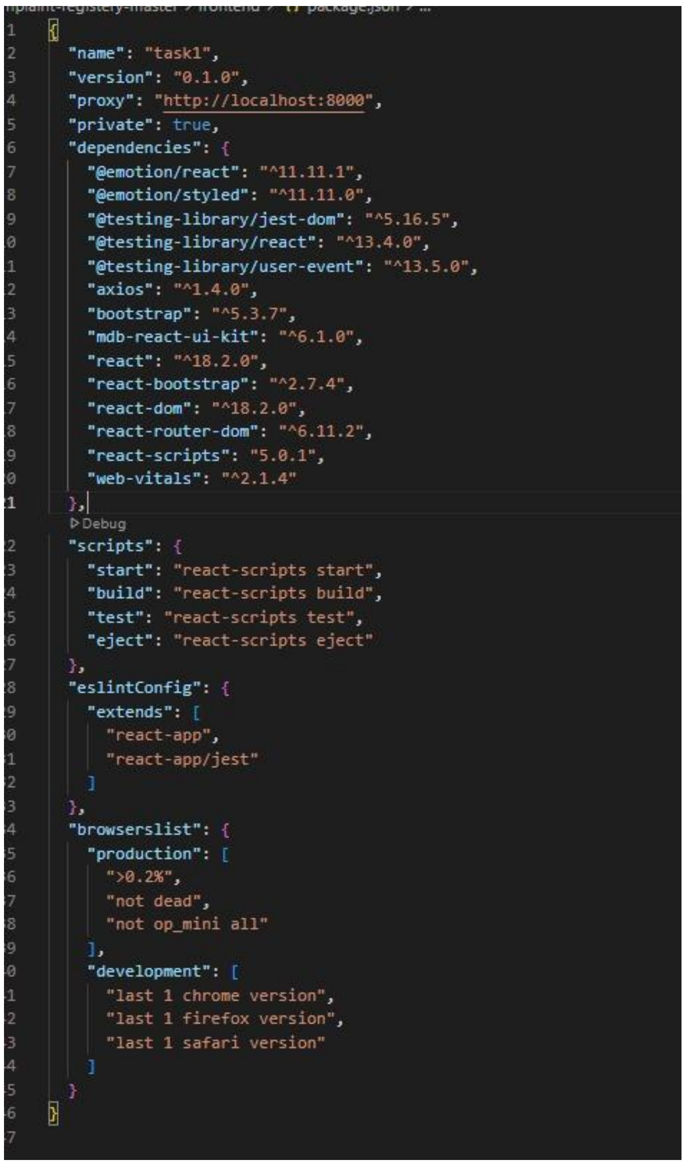
- Frontend: Open http://localhost:3000
- Backend: Test APIs using Postman at http://localhost:5000
- Login/Register, Submit and Track Complaints

PROJECT DEMO & RESOURCES

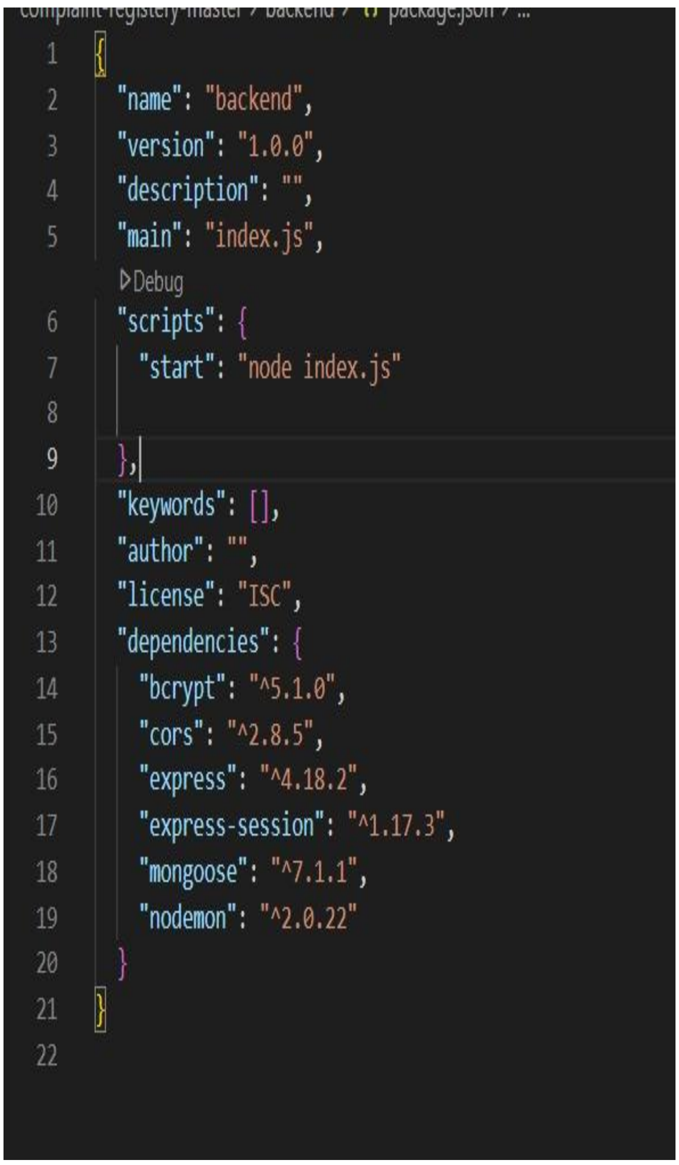
- Demo Video: Google Drive Demo Link
- GitHub Repo: *https://github.com/Bhavana029/online-complaint-system*
- Tutorial Videos:
  - Setup Backend
  - Connect MongoDB
  - Build Frontend
  - Test Endpoints
  -

MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

● Setting up a structured environment is crucial for any application. By organising the project into separate folders for the frontend and backend, we ensure that the codebase is manageable and scalable.



```json
{
  "name": "task1",
  "version": "0.1.0",
  "proxy": "http://localhost:8000",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.7",
    "mdb-react-ui-kit": "^6.1.0",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```



```json
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"

  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "express-session": "^1.17.3",
    "mongoose": "^7.1.1",
    "nodemon": "^2.0.22"
  }
}
```

PROJECT FOLDER STRUCTURE:

Frontend Folder:

- Contains the entire user interface code, built using React.js.
- Utilizes UI frameworks like Material UI and Bootstrap to enhance user experience.
- Maintains separation from backend logic for modularity.

Backend Folder:

- Includes server-side logic built using Node.js and Express.js.
- Manages routing, APIs, database connectivity, and user authentication.
- Enables scalability and clean separation of logic.

LIBRARY AND TOOL INSTALLATION:

Backend Libraries:

- **Node.js:** Server-side runtime for JavaScript.
- **MongoDB:** NoSQL database for dynamic and schema-less data.
- **Bcrypt:** Password hashing for secure authentication.
- **Body-parser:** Parses incoming JSON payloads.
- **CORS:** Enables secure communication between frontend and backend.

Frontend Libraries:

- **React.js:** Component-based frontend framework.
- **Material UI & Bootstrap:** For responsive and consistent UI design.
- **Axios:** For handling HTTP requests and communication with backend APIs.

MILESTONE 2: BACKEND DEVELOPMENT

Express.js Server Setup:

- Acts as a request-response handler.
- Uses middleware like body-parser and cors for parsing and CORS handling.

API Route Definitions:

- Routes categorized by functionality (auth, complaints, appointments).
- Each route file handles specific CRUD operations via Express route handlers.

Data Models with Mongoose:

- **User Schema:** Handles personal information and role-based access.
- **Complaint & Appointment Models:** Manage relationships and operations.
- **CRUD operations:** Enable structured Create, Read, Update, Delete actions.

User Authentication:

- Uses **JWT** for sessionless secure authentication.
- Protects routes via token verification.

Admin and Transaction Handling:

- Admins handle user roles, appointments, and complaint monitoring.
- Users manage bookings and history through secure APIs.
-

Error Handling:

- Middleware captures and logs errors with proper HTTP status codes.

MILESTONE 3: DATABASE DEVELOPMENT

Schemas for MongoDB Collections:

- **User Schema:** Stores name, email, password, and userType.
- **Complaint Schema:** Connects users to their complaints with statuses.
- **Chat Window Schema:** Manages user-agent communication.

Database Collections:

- Collections include users, complaints, appointments, and messages.
- Schema-less structure allows flexible scaling.

MILESTONE 4: FRONTEND DEVELOPMENT

React App Setup:

- Organized into components, pages, and services.
- Uses reusable UI elements for forms, dashboards, and modals.

UI Component Libraries:

- Material UI and Bootstrap for responsive and intuitive design.

Frontend Logic Implementation:

- **API Integration:** Axios communicates with backend.
- **State Management:** React state keeps UI in sync with backend data.

MILESTONE 5: PROJECT IMPLEMENTATION & TESTING

Functionality Testing:

- Ensures the entire stack (frontend, backend, database) works smoothly.
- Tests all flows like login, registration, appointment booking, and complaint submission.

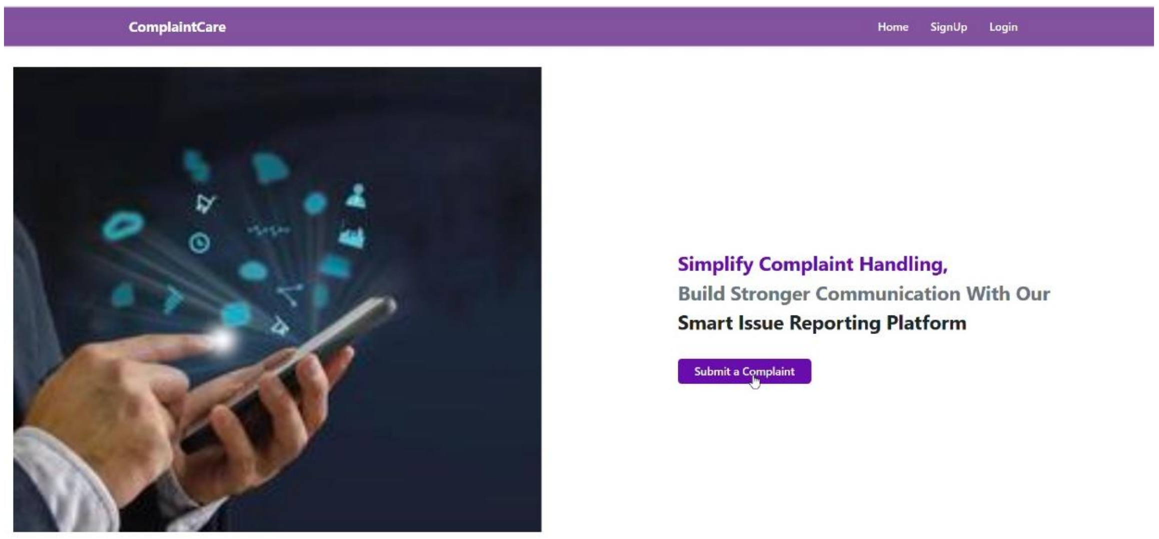
UI Testing:

- Verifies layout, responsiveness, and cross-device compatibility.

Final Deployment:

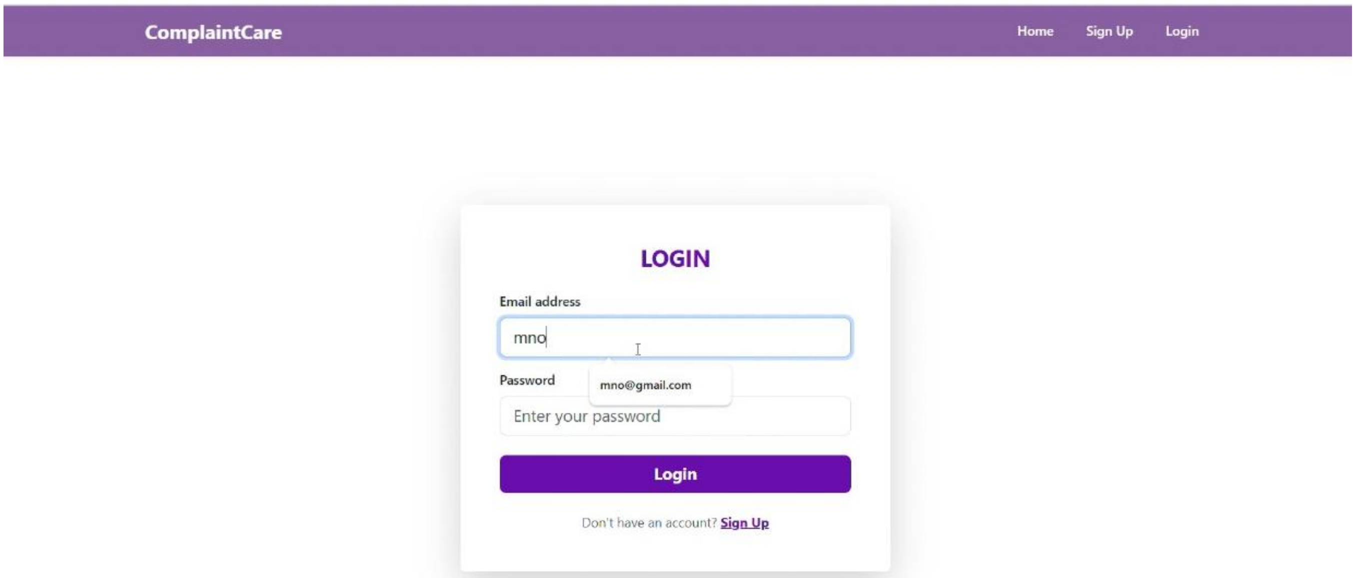
- After all tests pass, project is deployed to a production server.
- Application becomes publicly accessible with full functionality.

LANDING PAGE :

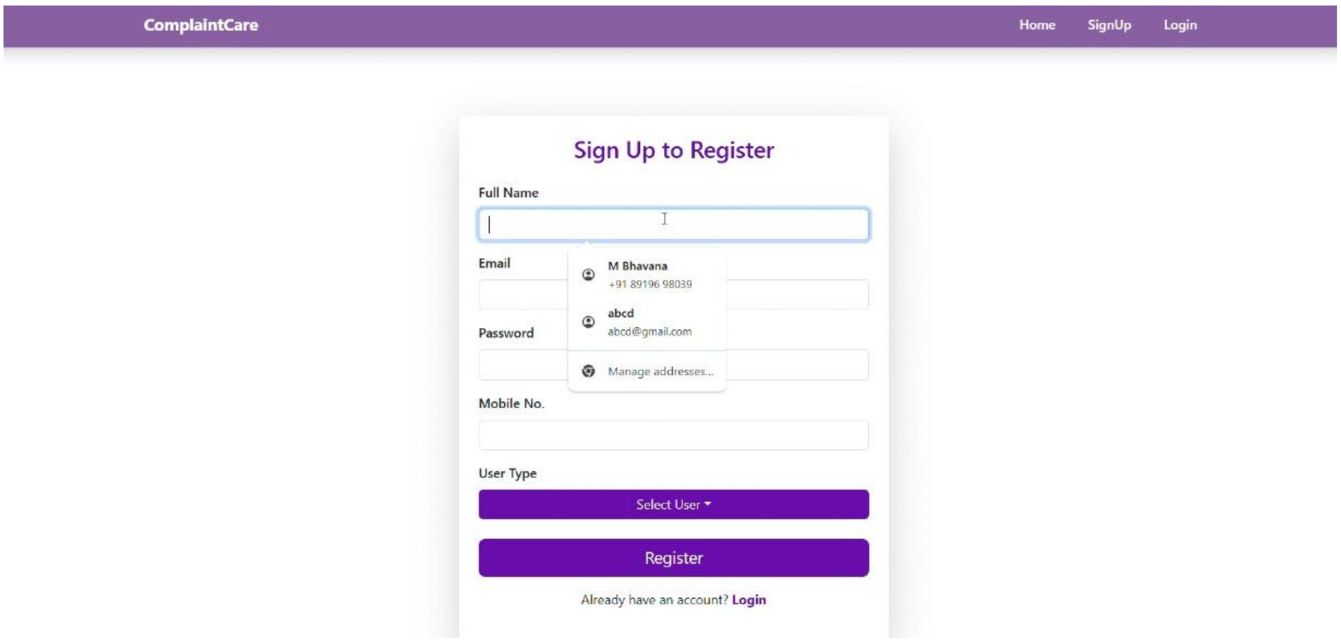


Simplify Complaint Handling,
Build Stronger Communication With Our
Smart Issue Reporting Platform

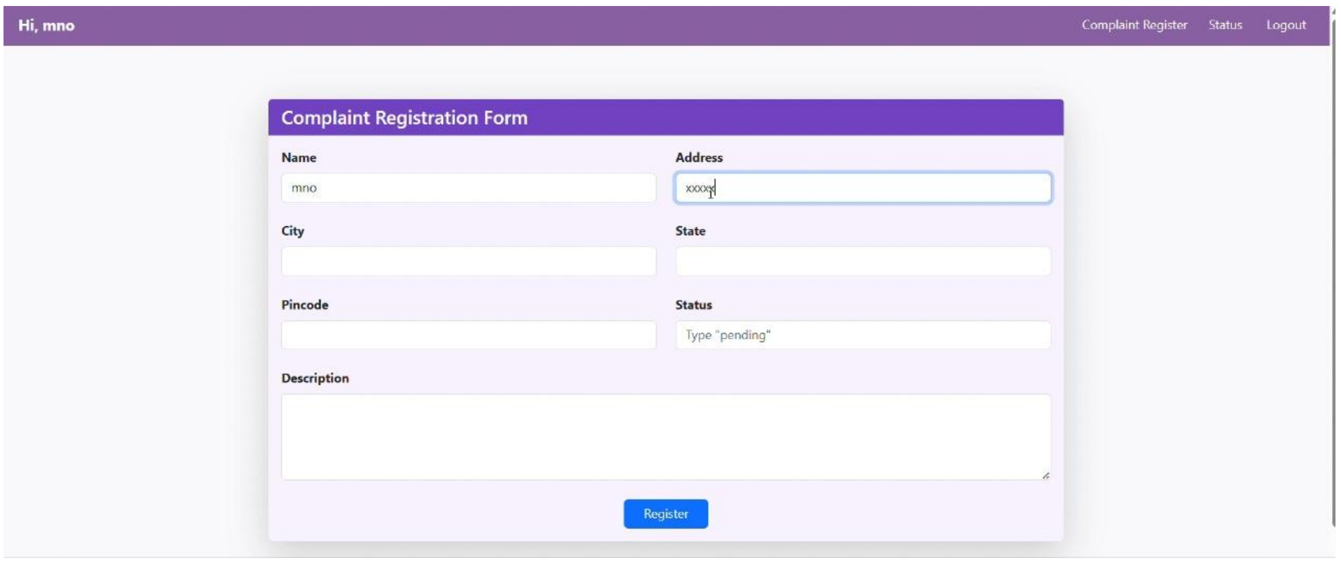
LOGIN PAGE :



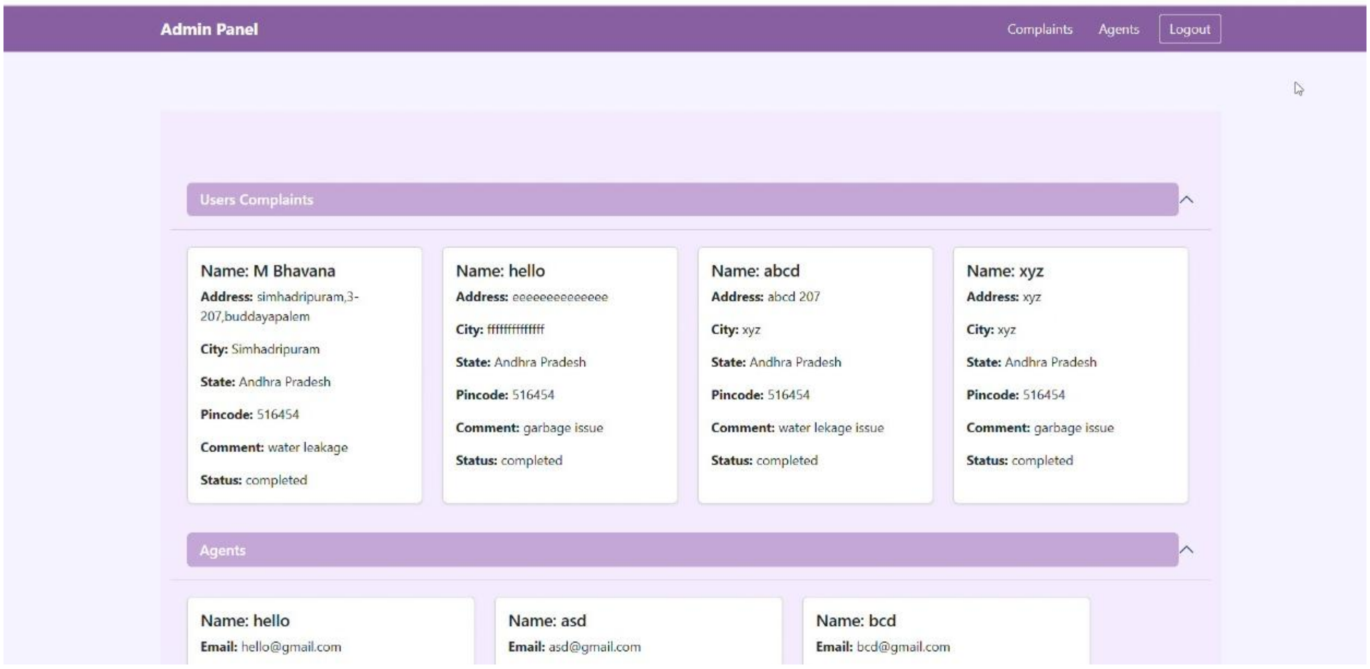
REGISTRATION PAGE :



USER PAGE :



ADMIN PAGE :



Name: M Bhavana
Address: simhadripuram,3-207,buddayapalem
City: Simhadripuram
State: Andhra Pradesh
Pincode: 516454
Comment: water leakage
Status: completed

Name: hello
Address: eeeeeeeeeeeeee
City: fffffffffffff
State: Andhra Pradesh
Pincode: 516454
Comment: garbage issue
Status: completed

Name: abcd
Address: abcd 207
City: xyz
State: Andhra Pradesh
Pincode: 516454
Comment: water lekage issue
Status: completed

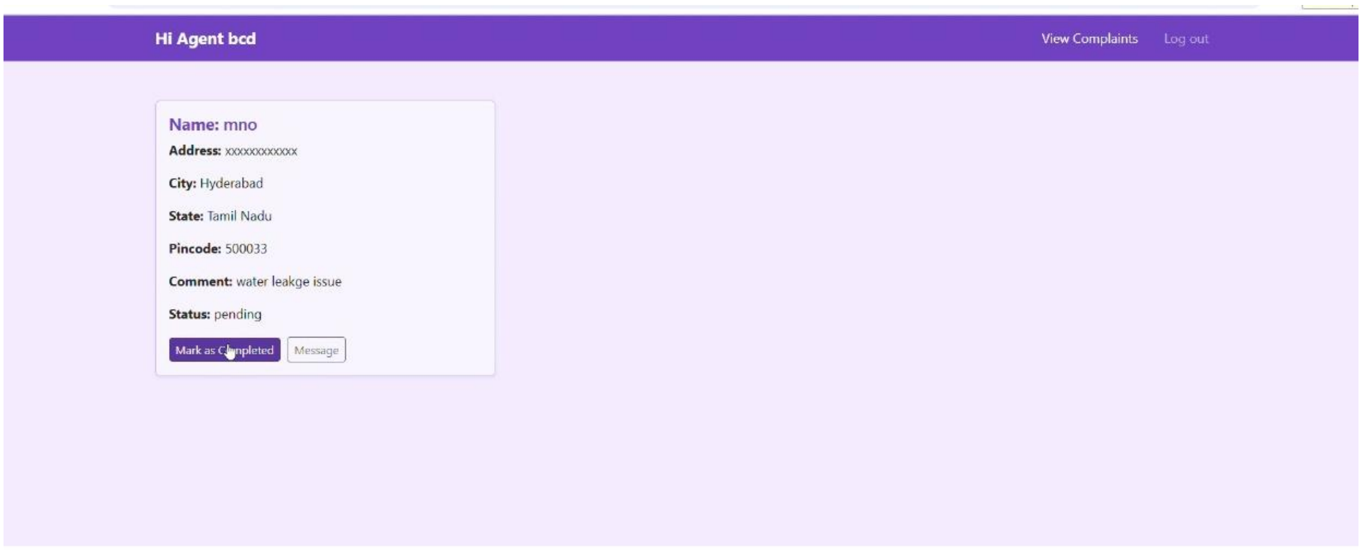
Name: xyz
Address: xyz
City: xyz
State: Andhra Pradesh
Pincode: 516454
Comment: garbage issue
Status: completed

Name: hello
Email: hello@gmail.com

Name: asd
Email: asd@gmail.com

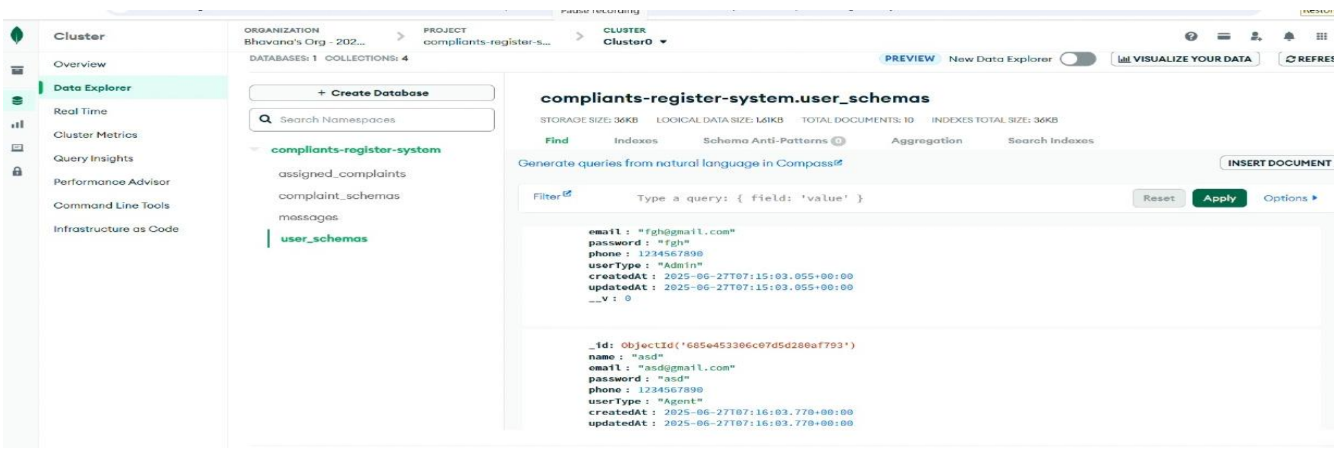
Name: bcd
Email: bcd@gmail.com

AGENT PAGE:-



Name: mno
Address: xxxxxxxxxxxx
City: Hyderabad
State: Tamil Nadu
Pincode: 500033
Comment: water leakge issue
Status: pending

DATABASE STORAGE:



DATABASES: 1 COLLECTIONS: 4

compliants-register-system
- assigned_complaints
- complaint_schemas
- messages
- user_schemas

compliants-register-system.user_schemas

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.61KB TOTAL DOCUMENTS: 10 INDEXES TOTAL SIZE: 36KB

```
email : "fgh@gmail.com"
password : "fgh"
phone : 1234567890
userType : "Admin"
createdAt : 2025-06-27T07:15:03.055+00:00
updatedAt : 2025-06-27T07:15:03.055+00:00
__v : 0
```

```
_id: ObjectId('685e453306c07d5d280af793')
name : "asd"
email : "asd@gmail.com"
password : "asd"
phone : 1234567890
userType : "Agent"
createdAt : 2025-06-27T07:16:03.770+00:00
updatedAt : 2025-06-27T07:16:03.770+00:00
```

CONCLUSION

This project successfully demonstrates the development of a full-stack **Complaint and Appointment Management System** using the MERN stack. The system incorporates multiple user roles—**Admin, Customer, and Doctor**—with distinct functionalities tailored to each role. Key milestones included setting up the frontend using **React with Material UI and Bootstrap**, backend development using **Node.js and Express.js**, and data management using **MongoDB** with Mongoose schemas for users, complaints, appointments, and messages.

Security measures like **JWT authentication** and **bcrypt password encryption** were implemented to ensure safe user interaction. The use of **Axios** for frontend-backend communication and middleware like **body-parser and CORS** facilitated efficient API handling and cross-origin support.

Thorough testing and modular structuring allowed for a scalable and maintainable system. The final outcome is a responsive and interactive application that provides users with real-time booking, complaint tracking, secure authentication, and admin-level control—making it a robust and user-friendly solution.