PROJECT REPORT

VELLORE INSTITUTE OF TECHNOLOGY, BHOPAL

Team ID: SWTID1743161557
Project Title: Online Complaint Registration and Management System (OCRMS)
Program: Full Stack Developement MERN_VIT (2025)

Mentor: Pratiksha Amale

April, 2025

Contents

Full Stack Development (MERN)

1. Introduction

Project Title: Online Complaint Registration And Management System (OCRMS)
Team ID : SWTID1743161557
Team Members:

1. **Team Leader :** Biprajeet Sen
2. **Team Member :** Manas Maheshwari
3. **Team Member :** Saptarshi Das
4. **Team Member :** Amritangshu Dey

2. Project Overview

The **Online Complaint Registration and Management System (OCRMS)** is a software solution designed to streamline the process of lodging, tracking, and resolving complaints. Built with the MERN stack (MongoDB, Express, React.js, Node.js), OCRMS centralizes complaint handling, ensuring efficiency, regulatory compliance, and improved user satisfaction. It is ideal for organizations aiming to enhance issue resolution and safety management in line with industry standards. OCRMS incorporates several key features:

A. **User Registration:** Users can create accounts to securely submit complaints and track their progress.
B. **Complaint Submission:** Users can provide detailed descriptions of their issues, along with supplementary information like name, address, and relevant attachments such as documents or images.
C. **Tracking and Notifications:** Real-time updates allow users to monitor complaint statuses and receive notifications via email or SMS for every significant action taken, such as assignment or resolution.
D. **Agent Interaction:** Users can directly communicate with the assigned agent to discuss and resolve complaints effectively.
E. **Intelligent Complaint Routing:** The system ensures complaints are routed to the appropriate personnel or department based on predefined criteria, optimizing resource allocation.
F. **Security and Confidentiality:** OCRMS prioritizes data security and compliance with privacy regulations by employing robust measures like user authentication, data encryption, and access controls.

Online Complaint Redressal Management System (OCRMS) is designed to streamline the process of lodging, tracking, and resolving complaints efficiently. The goal of this system is to provide a transparent, structured, and user-friendly platform where users can report grievances, monitor their status, and receive timely resolutions.

Many organizations and government bodies struggle with inefficient complaint management due to manual processing, delayed responses, and lack of accountability. OCRMS aims to eliminate these issues by digitizing and automating the complaint redressal system, ensuring a faster, more reliable, and transparent process.

2.1 Key Objectives of the Project:

- Enable users to submit complaints online with ease.
- Provide a centralized dashboard for tracking complaint progress.
- Implement an automated workflow to assign complaints to the appropriate department.
- Ensure timely communication through notifications and updates.
- Allow authorities to analyze complaint trends for better decision-making.
- Offer feedback mechanisms to assess user satisfaction.

OCRMS can be deployed in corporate environments, government offices, educational institutions, and customer service sectors, ensuring enhanced efficiency, accountability, and improved user experience.

2.2 Features & Functionalities

OCRMS is equipped with several advanced features to ensure a seamless user experience for both complainants and administrators. Below are the key features of the system:

A. User Registration & Authentication

- Users can **sign up** using their email, phone number, or social media accounts.
- Multi-factor authentication (MFA) for added security.
- Role-based access for **complainants, administrators, and higher authorities**.

B. Complaint Submission Portal

- Easy-to-use complaint filing interface with category selection.
- Users can attach images, videos, or documents as proof.
- Auto-suggestions for frequently reported issues.

C. Complaint Tracking System

- **Real-time status updates** via SMS and email notifications.
- Unique tracking ID assigned to each complaint.
- **Color-coded priority levels** (Low, Medium, High, Urgent).

D. Automated Workflow & Complaint Assignment

- **AI-powered categorization** to route complaints to the relevant department.
- **Automated ticket assignment** based on department workload.
- **Deadline reminders** to ensure timely resolution.

E. Communication & Notifications

- Users receive instant notifications on complaint progress.
- In-app messaging system for direct communication with authorities.
- Escalation mechanism for unresolved complaints.

F. Feedback & Rating System

- Users can rate the resolution process after complaint closure.
- Feedback analytics to identify common issues & service gaps.

G. Reporting & Analytics Dashboard

- Admins can **generate reports** on complaint trends and resolution times.
- AI-based insights for **predictive analysis** and efficiency improvements.
- Graphical representations for **easy decision-making**.

H. Multilingual & Accessibility Support

- Support for multiple languages for wider usability.
- Voice-enabled complaint filing for users with disabilities.

I. Security & Privacy Compliance

- Data encryption & GDPR compliance to ensure user data protection.
- Audit logs to track administrative actions and maintain accountability.

J. Integration with External Systems

- Can be integrated with chatbots, helplines, and government databases.
- API support for third-party applications.

3. Methodology

3.1 Approach

The Online Complaint Registration and Management System (OCRMS) is designed using the MERN stack, a collection of powerful, open-source JavaScript technologies. This stack is ideal for developing modern, dynamic, and scalable web applications. By leveraging MongoDB, Express.js, React.js, and Node.js, OCRMS provides an efficient, robust, and extensible solution for managing user complaints. The following subsections detail each of the core components of the MERN stack:

A. MongoDB: MongoDB is a highly scalable NoSQL database that stores data in flexible, JSON-like documents. Its schema-less structure allows for rapid prototyping and quick adjustments during development. Unlike traditional relational databases, MongoDB is particularly well-suited for managing semi-structured or unstructured data. This flexibility helps OCRMS easily accommodate changing data requirements as the system evolves. Furthermore, MongoDB's built-in support for horizontal scaling ensures that the system can efficiently handle an increase in data as the application grows, making it suitable for large-scale complaint management.

B. Express.js: Express.js is a minimal yet highly extensible framework for Node.js, making it an ideal choice for building web applications and APIs. It simplifies the creation of robust server-side logic, enabling the seamless processing of HTTP requests, man-aging routes, and integrating middleware. Express.js allows the creation of RESTful APIs, which facilitate communication between the frontend and backend, making it a core component of OCRMS for handling user requests such as submitting complaints, updating complaint statuses, and managing user profiles.

Additionally, Express's middleware system allows the system to integrate important features like request logging,data validation, and error handling with minimal effort.

C. React.js: React.js is a powerful JavaScript library that specializes in building complex user interfaces, especially single-page applications (SPAs). React's declarative nature and component-based architecture allow developers to build interactive and reusable UI components. Each component in React encapsulates its own logic and UI, ensuring maintainability and reusability. React's Virtual DOM mechanism improves performance by updating only the parts of the UI that have changed, leading to faster rendering times. In OCRMS, React.js is utilized to create a dynamic and responsive frontend where users can register complaints, track their status, interact with agents, and receive real-time updates.

D. Node.js: Node.js is an open-source, cross-platform runtime environment that allows for the execution of JavaScript code outside of the browser. It is built on Google Chrome's V8 JavaScript engine and uses an event-driven, non-blocking I/O model, making it lightweight and efficient. This architecture is well-suited for building scalable network applications that require high throughput and real-time communication. OCRMS lever- ages Node.js to handle backend processes, including server-side logic, API integration, and real-time data handling. Its asynchronous nature enables OCRMS to handle many user requests concurrently, ensuring smooth operations even under high traffic conditions.

4. System Architecture

The architecture of OCRMS is designed using a client-server model, where the frontend (client-side), backend (server-side), and database are clearly separated, allowing for better scalability, modularity, and ease of maintenance. The architecture ensures smooth communication between all components, enabling a responsive and efficient application that can handle a high volume of user interactions. Below is a detailed breakdown of the OCRMS architecture:

1. Frontend (Client-Side): The frontend of OCRMS is the user-facing interface, built using React.js. It provides users with a dynamic and responsive experience, allowing them to register complaints, track complaint statuses, communicate with agents, and view their complaint history. The key technical highlights of the frontend include:

Continued...

a. **Component-Based Architecture:** React's component-based structure allows for the development of reusable UI components such as forms, modals, dashboards, and tables. This promotes consistency across the application and reduces redundancy in code.

b. **UI/UX Design:** The frontend incorporates modern design principles and responsive UI frameworks such as Bootstrap and Material UI to ensure a seamless user experience across a variety of devices, including desktops, tablets, and smart-phones.

c. **State Management:** React's state management system ensures that the application maintains and updates its data dynamically. Using state management libraries such as Redux or React's built-in hooks allows for better control over the flow of data between components, leading to an optimized user experience.

d. **Real-Time Features:** React is integrated with WebSockets, specifically Socket.IO, to enable real-time communication features. These include live updates for complaint statuses, direct communication with agents, and instant notifications for new messages or updates.

e. **RESTful API Integration:** The frontend communicates with the backend through RESTful APIs. Axios, a promise-based HTTP client, facilitates asynchronous communication between the frontend and backend for submitting complaints, fetchingdata, and updating complaint statuses.

2. Backend (Server-Side): The backend of OCRMS is built with Node.js and Express.js,forming the backbone of the system's logic and data processing. The backend handles user authentication, communication between the client and the database, and the execution of business logic. Key features of the backend include:

a. **RESTful API Design:** The backend exposes a series of RESTful APIs that enable the frontend to interact with the system. These APIs facilitate CRUD operations on entities like complaints, users, and agents. The endpoints are designed to be secure, efficient, and easily extendable.

b. **Middleware Architecture:** Express.js allows the use of middleware functions to handle various server-side operations, such as request validation, user authentication, logging, and error handling. Middleware functions enhance security, ensure data integrity, and streamline server processing.

c. **Authentication and Authorization:** To ensure secure access to the application, the backend employs JSON Web Tokens (JWT) for stateless authentication. This ensures that only authorized users (e.g., regular users, agents, or administrators) can access the appropriate system features. Additionally, role-based access control (RBAC) is implemented to restrict access based on user roles.

d. **Real-Time Interaction:** Socket.IO enables bi-directional communication between the frontend and backend, allowing the system to push real-time updates to users. This is critical for notifying users about changes in complaint statuses, new messages, or agent availability.

3. Database (Data Storage): MongoDB serves as the database for OCRMS, offering flexibility, scalability, and ease of use. It stores data in collections of JSON-like documents, which can represent dynamic entities such as users, complaints, and agents. Key aspects of the database design include:

a. **Data Model:** OCRMS uses MongoDB collections to store documents for various entities, including users, complaints, agents, and administrators. Each document contains key fields necessary for the operation of OCRMS, such as complaint description, user details, complaint status, and timestamps.

b. **Indexing and Query Optimization:** MongoDB supports indexing, which enhances the speed of data retrieval. Indexes are created for frequently queried fields, such as complaint status, user ID, and timestamps, allowing for faster searches and efficient data access.

c. **Data Integrity:** Mongoose, an ODM (Object Data Modeling) library for MongoDB, is used to define schemas and enforce data validation rules. Mongoose ensures that the data stored in the database adheres to the required structure,which maintains the integrity of the application data.

d. **Scalability:** MongoDB is horizontally scalable, allowing it to handle large volumes of data as the application grows. The system can be distributed across multiple servers using sharding, ensuring high availability and fault tolerance.

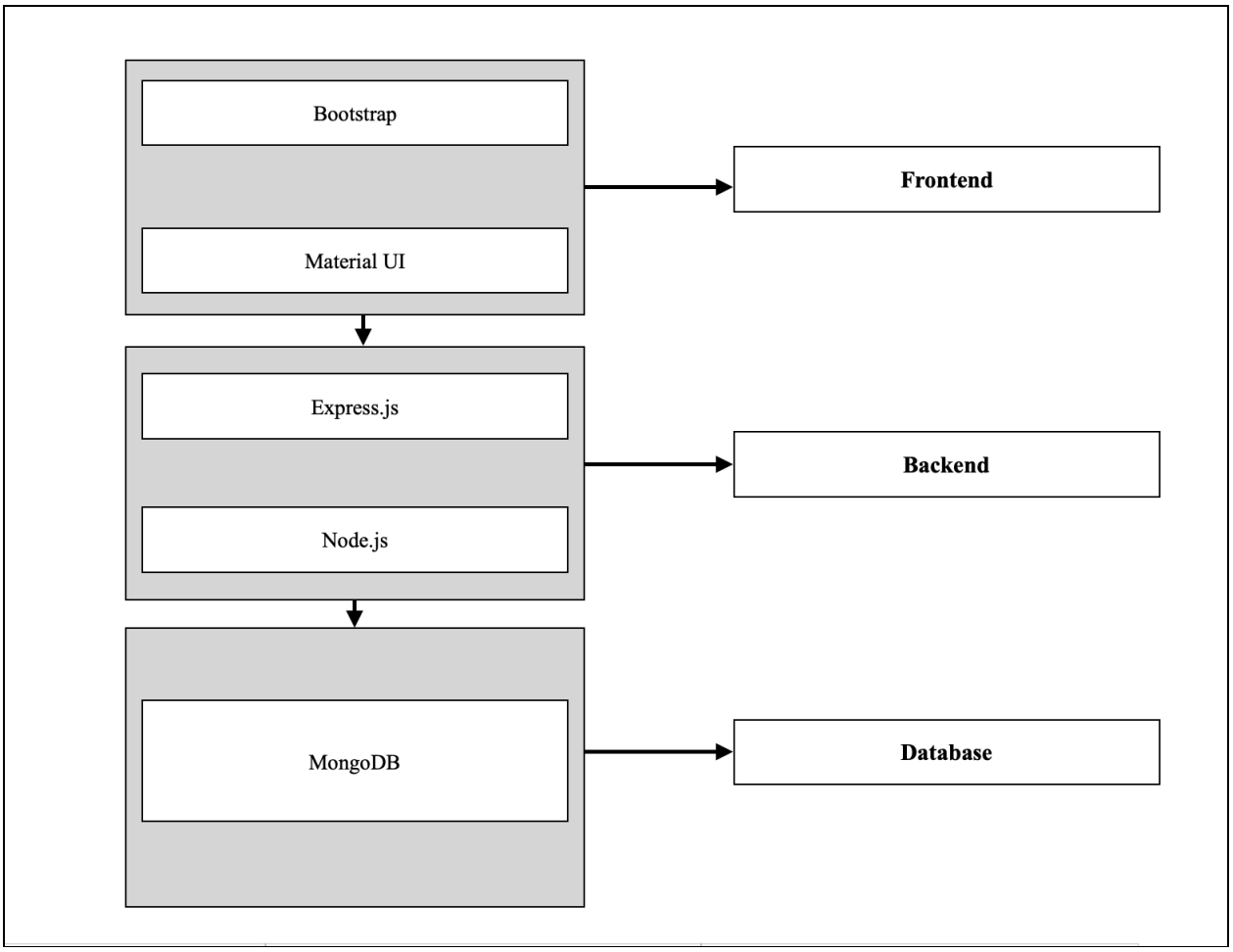


Figure 1: System Architecture Diagram for OCRMS

This architecture diagram shows a **3-Tier MERN-like stack**, divided into:

1. Frontend

- **Technologies:** Bootstrap and Material UI

2. Backend

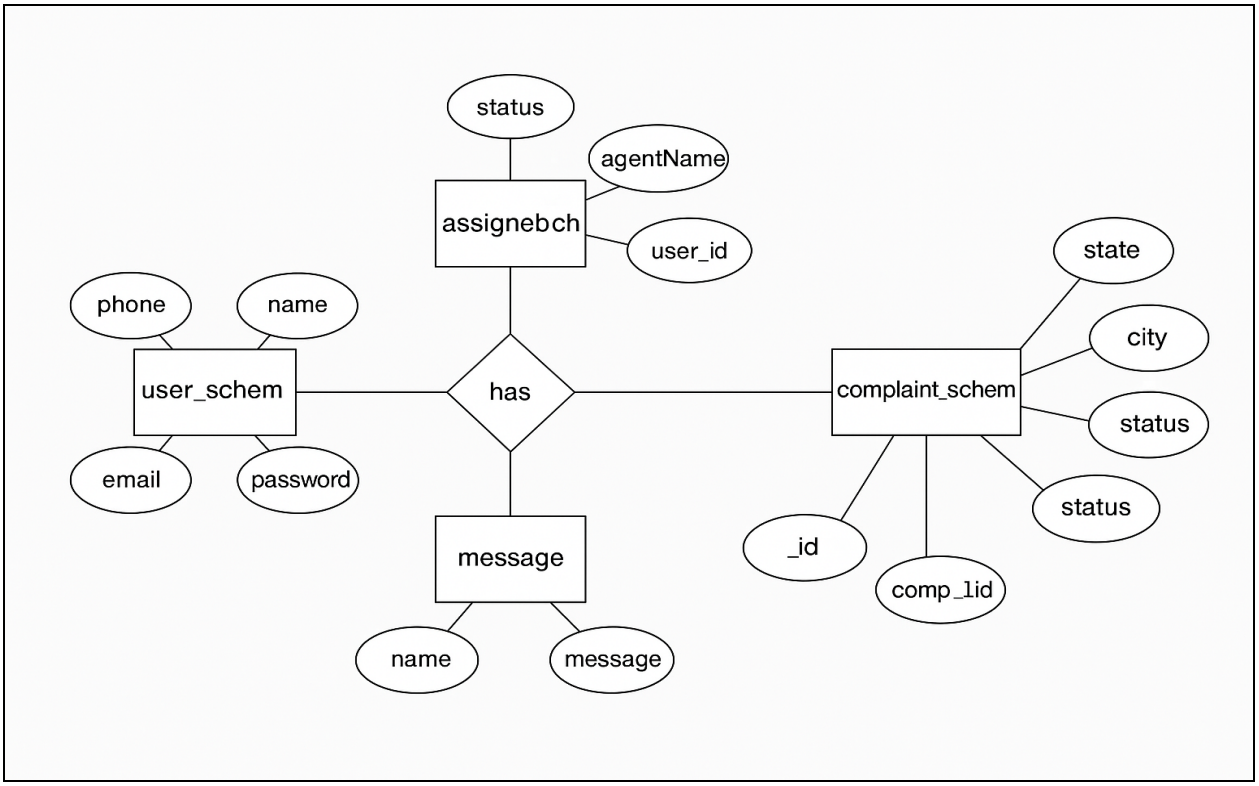
- **Technologies:** Node.js, Express.js

3. Database

- **Technology:** MongoDB

Continued...

4.1 ER Diagram



This ER diagram outlines how users, complaints, assignments, and messages are related in the OCRMS database.

Entities and Their Attributes

1. user_schem (User Schema)

Represents the **users** of the system.

Attributes:

- name: Name of the user
- email: Email ID (used for login/identification)
- password: Encrypted password
- phone: Contact number

Continued...

2. complaint_schem (Complaint Schema)

Represents the **complaints** filed by users.

Attributes:

- _id: Unique identifier for each complaint (MongoDB default)
- comp_lid: Complaint Local ID (possibly user-defined or easier to reference)
- state: State of the complainant
- city: City of the complainant
- status: Current status (e.g., pending, in-progress, resolved)

3. assignebch (Assignment Schema)

Represents the **assignment** of a complaint to a backend agent.

Attributes:

- agentName: Name of the assigned agent
- user_id: ID of the user who created the complaint (foreign key)
- status: Status of the assignment (e.g., assigned, completed)

4. message

Represents communication/messages related to complaints.

- Attributes:
  - name: Name of the person sending the message (could be agent or user)
  - message: The actual message text

5. Relationships

- The **diamond "has"** represents a **relationship** among:
  - user_schem
  - complaint_schem
  - assignebch
  - message

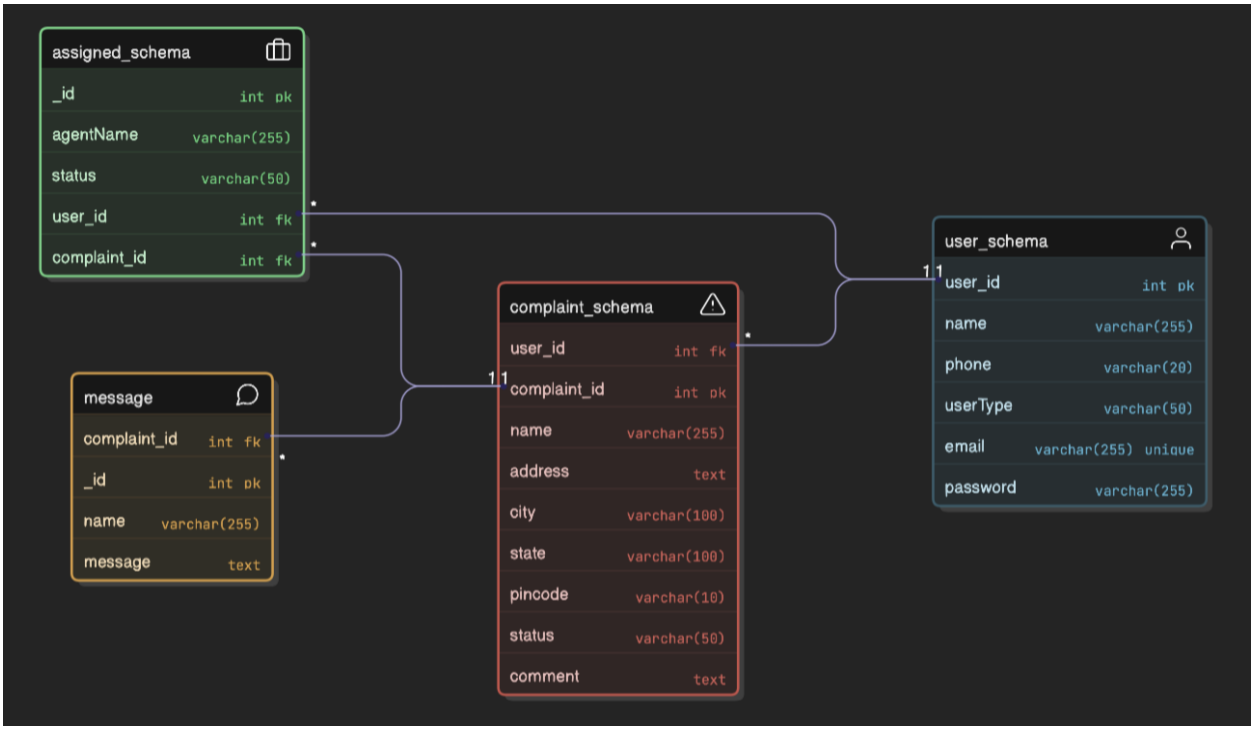
This suggests:

- A **user** can **file complaints**.
- A **complaint** can be **assigned** to an agent.
- A **message** can be **associated** with both the user and complaint (for tracking or progress).

Summary of Data Flow

a. **Users** file **complaints**.
b. Each complaint is stored in the complaint_schem.
c. Complaints are then **assigned** to agents via assignebch.
d. Users and/or agents can exchange **messages** about the complaint.

4.2 UML Class Diagram



This is an **UML Class Diagram** for the Online Complaint Registration Management System (OCRMS) database schema. It visually represents the structure and relationships between the entities (tables) used in the system.

Entities & Relationships Explaination

1. **user_schema**

Purpose: Stores information about users (citizens, admins, agents).
Attributes:

- user_id (PK) – Unique identifier for each user.

- name, phone, email – Basic contact details.
- userType – Role of user (admin, agent, user, etc.).
- password – For authentication.
- email is unique.

Relationships:

- One user can submit **multiple complaints** → complaint_schema.user_id (1-to-many).
- One user can be linked to **multiple assignments** → assigned_schema.user_id (1-to-many).

2. complaint_schema

Purpose: Stores complaint details raised by users.
Attributes:

- complaint_id (PK) – Unique ID for each complaint.
- user_id (FK) – Links complaint to the user.
- name, address, city, state, pincode – Details of the complainant and location.
- status, comment – Complaint tracking and remarks.

Relationships:

- One complaint can have **many messages** → message.complaint_id (1-to-many).
- One complaint can be **assigned** via assigned_schema.complaint_id (1-to-1 or 1-to-many based on use).

3. assigned_schema

Purpose: Links agents to specific complaints.
Attributes:

- _id (PK) – Assignment ID.
- agentName – Name of the assigned agent.
- status – Current status of the assignment (Assigned, Resolved, etc.).
- user_id (FK) – User who raised the complaint.
- complaint_id (FK) – Complaint that’s being handled.

Relationships:

- Links to both **user** and **complaint** tables through foreign keys.

4. message

Purpose: Stores messages exchanged regarding a specific complaint (e.g., between user and agent).
Attributes:

- _id (PK) – Message ID.
- complaint_id (FK) – Links to the related complaint.
- name – Who sent the message (user/agent/admin).
- message – The actual message text.

Relationships:

- One complaint can have **multiple messages**.

Relationships:

Relationship	Type
user → complaint	1-to-many
user → assigned_schema	1-to-many
complaint → message	1-to-many
complaint → assigned_schema	1-to-1 (or 1-to-many in extended use)

5. Project Structure

The OCRMS project is organized into three primary components, each serving a distinct function in the system:

Setup of Frontend:

- The React.js codebase, structured into functional and class components, ensures that the application is modular, maintainable, and scalable.
- The frontend includes pages for registering complaints, tracking status, user profiles, and live chat features.
- State management and data fetching are handled using React's hooks and Axios to ensure efficient communication with the backend and real-time updates.

The frontend is built using the following steps:

A. Initial Setup & Libraries: Before diving into the creation of UI components, we first need to install the required libraries and set up the basic project structure.

```
npx create-react-app customer-care-registry
cd customer-care-registry
```

B. Required Libraries: Install libraries that will help in building the user interface and handling various tasks such as routing, state management, and API calls. Key libraries include:

```
npm install axios react-router-dom redux react-redux material-ui
@mui/icons-material
```

- **axios:** For making HTTP requests to communicate with the backend API.
- **react-router-dom:** For handling navigation between different pages in the app.
- **redux and react-redux:** For state management across the application.
- **material-ui:** A UI component library to help design a modern and responsive interface.

Setup of Backend:

- Built using Node.js and Express.js, the backend exposes a suite of RESTful APIs for handling user requests such as complaint registration, status updates, and user management.

- The backend is also responsible for authentication and authorization using JWT, ensuring secure access.

Setup of Database:

MongoDB stores user profiles, complaint details, and agent information in collections, ensuring fast and scalable data access.

The database schema is defined using Mongoose to enforce data integrity and consistency.

5.1 Technical Workflow

The following sequence of actions illustrates the technical workflow of OCRMS, integrating the frontend, backend, and database layers:

a. A user accesses the frontend and submits a complaint or views their complaint status.

b. The frontend sends an authenticated request to the backend via Axios.

c. The backend processes the request, applies business logic, and communicates with MongoDB to fetch or update data.

d. The backend responds with the requested data, and real-time updates (such as changes in complaint status) are pushed to the frontend using Socket.IO.

e. Data security is ensured through encryption protocols, authentication, and access control policies.

This technical workflow ensures that OCRMS is both user-friendly and scalable, capable of handling high traffic and providing real-time, efficient complaint management.

6. Setup Instructions

6.1 Prerequisites

Before setting up the project, ensure that the following software dependencies and tools are installed on your system:

- **Node.js** (v14 or above)
  Download Node.js – Required for running both frontend and backend environments using npm.
- **MongoDB** (v4.4 or above)
  Download MongoDB Community Server – Required to store and manage complaint data.
- **Code Editor** (e.g., Visual Studio Code)
  Download VS Code – Recommended for editing the codebase.

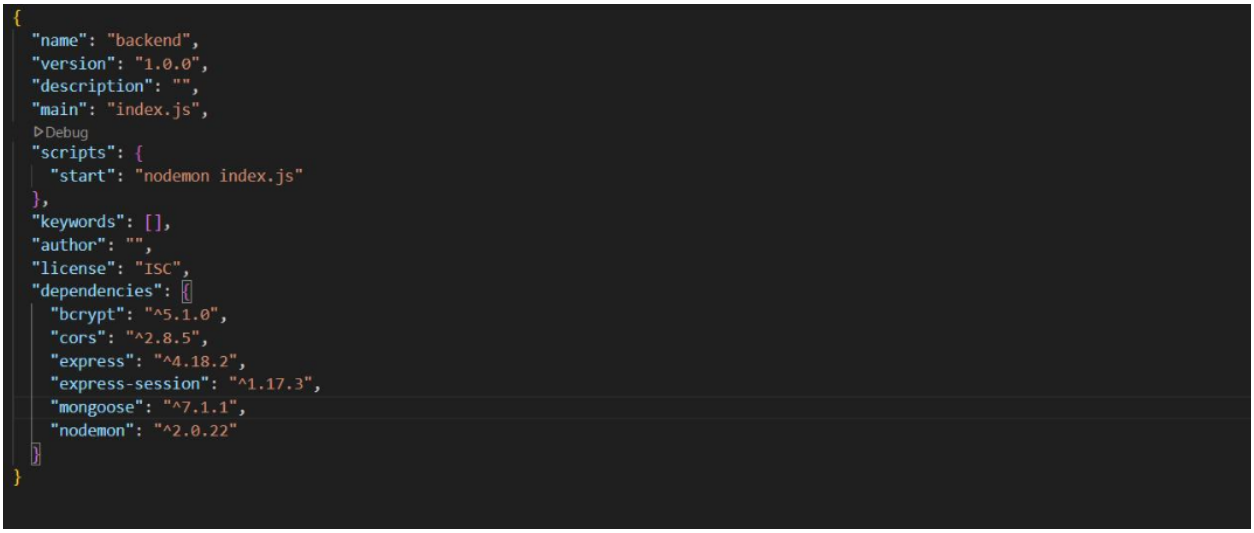
[Frontend Dependencies (frontend/package.json)]

- **React –** Frontend library
- **Redux & React-Redux –** For state management
- **React Router DOM –** For routing between components
- **Material UI –** UI components and styling
- **Axios –** For making HTTP requests
- **Socket.io-client –** For real-time communication with the backend
- **WebRTC –** For peer-to-peer communication (real-time video/chat features)

```
{
  "name": "task1",
  "version": "0.1.0",
  "proxy": "http://localhost:8000",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.13.3",
    "@emotion/styled": "^11.13.0",
    "@mui/icons-material": "^6.1.7",
    "@mui/material": "^6.1.7",
    "@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": "^5.2.3",
    "framer-motion": "^11.11.17",
    "mdb-react-ui-kit": "^6.1.0",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-confirm-alert": "^3.0.6",
    "react-dom": "^18.2.0",
    "react-hot-toast": "^2.4.1",
    "react-loading-skeleton": "^3.5.0",
    "react-router-dom": "^6.11.2",
    "react-scripts": "^5.0.1",
    "react-spinners": "^0.14.1",
    "react-toastify": "^10.0.6",
    "tailwindcss": "^3.4.15",
    "web-vitals": "^2.1.4"
  }
```

[Backend Dependencies (backend/package.json)]

- **Express –** Web framework for Node.js
- **Mongoose –** MongoDB ODM (Object Data Modeling)
- **Cors –** To allow cross-origin requests
- **Dotenv –** For environment variables
- **Socket.io –** Real-time communication (used for WebSockets)
- **Bcryptjs –** For password hashing
- **Jsonwebtoken –** For authentication using JWT
- **Node.js –** Required to run the backend server and manage packages via npm



6.2 Installation:

Step-by-step guide to clone, install dependencies, and set up the environment variables.

Follow these step-by-step instructions to set up the project on your local machine.

NOTE: This is a Group Project (Uploaded By Group Lead: Biprajeet Sen) on Github.

Step 1: Clone the Repository

```
git clone https://github.com/biprajeetvit22/SmartInternz_MERN_VITB25/tree/main/PROJECT.git
cd PROJECT
```

Make sure the cloned folder contains both the client (React frontend) and server (Node.js backend) directories.

Step 2: Setting Up the Backend

1. Navigate to the server directory:

```
cd server
```

2. Install backend dependencies:

```
npm install
```

3. Create a .env file in the server folder and configure the following environment variables:

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/project
```

4. Start the backend server:

```
npm start
```

The server should now be running on http://localhost:5000.

Step 3: Setting Up the Frontend

1. Open a new terminal and navigate to the client directory:

cd client

2. Install frontend dependencies:

npm install

3. Start the React development server:

npm start

This will launch the frontend at http://localhost:3000.

6.3 Sample Folder Layout After Setup:

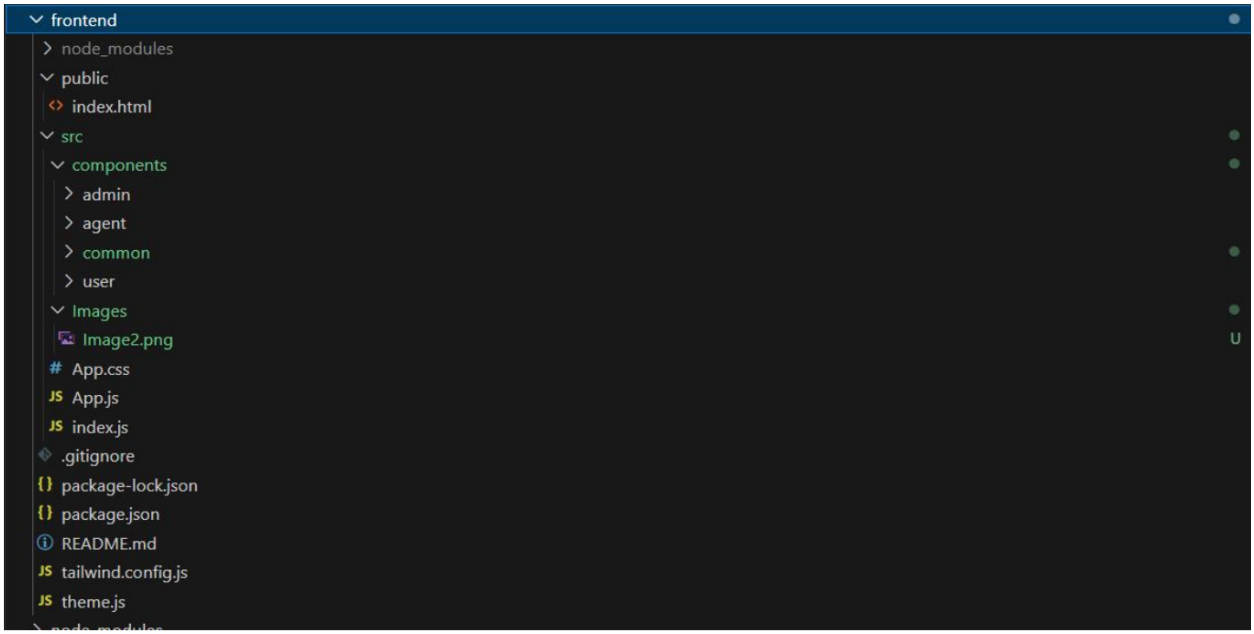
```
PROJECT/
├── backend/
│   ├── node_modules/
│   ├── Schema.js
│   ├── config.js
│   ├── index.js
│   ├── package-lock.json
│   └── package.json
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── Images/
│   │   ├── components/
│   │   ├── App.css
│   │   ├── App.js
│   │   └── index.js
│   ├── .gitignore
│   ├── package-lock.json
│   ├── package.json
│   ├── tailwind.config.js
│   └── theme.js
├── node_modules/
├── package-lock.json
├── package.json
└── README.md
```

7. Folder Structure

A. Client Server: (React Frontend)

: Located in the frontend/ directory, structured as follows:

- **public/ –** Static assets and HTML template.
- **src/ –** Main React source code.
- **components/ –** Reusable UI components like Navbar, Footer, Sidebar.
- **pages/ –** Full-page views like Home, Login, Register, Dashboard, Admin Panel.
- **redux/ –** Redux store, actions, and reducers for state management.
- **api/ –** Axios configuration and API helper functions.
- **App.js –** Entry point with routing.
- **index.js –** Renders React app to DOM.



B. Server (Node.js + Express + MongoDB Backend)

: Located in the backend/ directory, structured like this:

- **config/ –** MongoDB connection setup.
- **controllers/ –** Logic for handling API requests.
- **models/ –** Mongoose schemas for users, complaints, etc.
- **routes/ –** Express route definitions for users, complaints, admin.
- **middleware/ –** Custom middleware (e.g., auth, error handling).

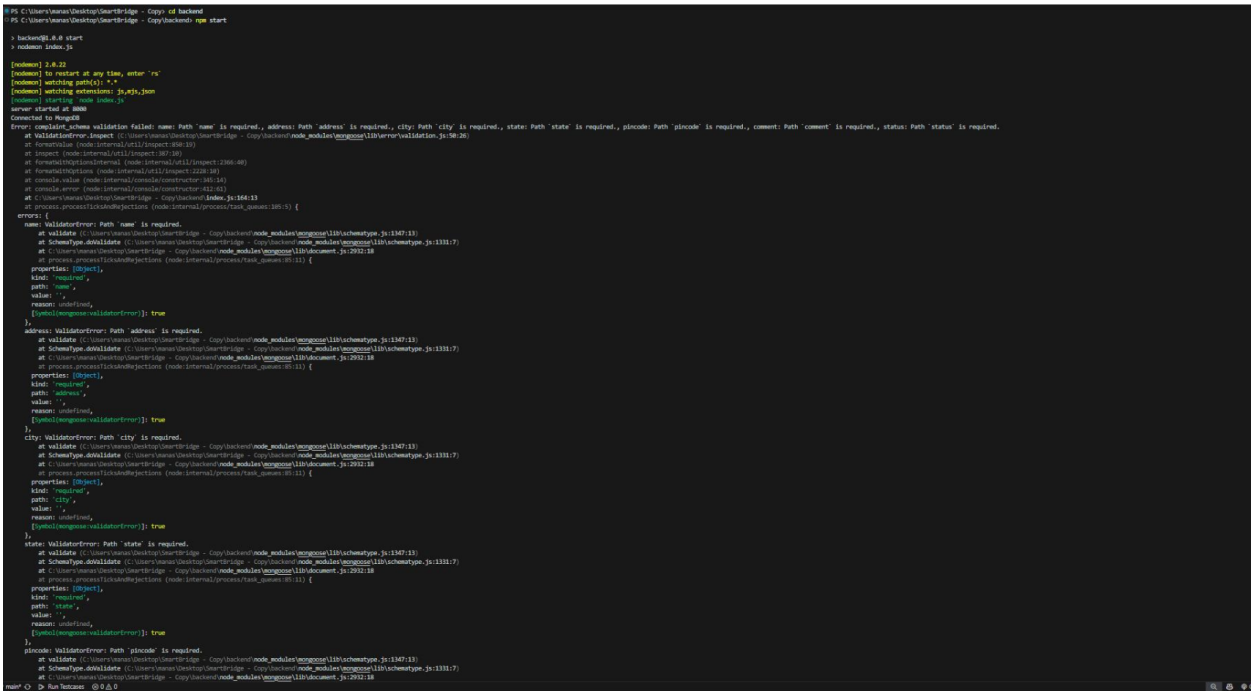
- **server.js –** Main entry point of the backend server.

8. Running the Application

To run the OCRMS application locally, follow these steps:

A. <u>Backend Server (Node.js + Express + MongoDB):</u>

- ➤ Navigate to the backend directory.
- ➤ Install the required dependencies using npm install.
- ➤ Start the server with npm start. By default, the server will run on http://localhost:5000 or server starts at 8000.



B. <u>Frontend Server (React.js):</u>

- ➤ Navigate to the frontend directory.
- ➤ Install the required dependencies using npm install.
- ➤ Start the development server with npm start. By default, the frontend will run on http://localhost:3000.

9. API Documentation

This system exposes a RESTful API that enables core functionalities such as user registration, authentication, complaint registration, agent assignment, and real-time messaging between users and agents. The API is structured around standard HTTP methods (GET, POST, PUT, DELETE), with intuitive endpoints categorized by function—such as user management, complaint handling, and message exchange. At present, all API routes are publicly accessible for testing, as no token-based authentication is implemented yet. Each endpoint accepts JSON-formatted input and returns responses in JSON as well, making it easy to integrate with frontend clients or third-party systems. For instance, users can register or log in, submit complaints, view status updates, and communicate with agents through associated complaint threads. Administrators can assign complaints to agents, while agents can update statuses and view assigned tasks. The API responds with standard HTTP status codes like 200 OK, 201 Created, 400 Bad Request, and 404 Not Found to indicate the result of operations. Errors and validation issues are returned with descriptive messages to aid in debugging. Although currently open, the system is designed to be extended with security layers such as JWT-based authentication in the future. This will ensure secure access control as the platform scales to production use.

9.1 Authentication

Currently, no token-based or session-based authentication is implemented. All endpoints are public.

9.2 User Authentication & Management

A. POST /SignUp

- **Input:** { name, email, password, phone, userType }
- **Output:** Saved user document
- **Purpose:** Register a new user

B. POST /Login

- **Input:** { email, password }
- **Output:** User object or error
- **Purpose:** User login

C. GET /AgentUsers

- **Output:** List of users with userType: "Agent"
- **Purpose:** Get all agent users

D. GET /AgentUsers/:agentId

- **Output:** Agent user details
- **Purpose:** Fetch a specific agent user

E. GET /OrdinaryUsers

- **Output:** List of users with userType: "Ordinary"
- **Purpose**: Get all ordinary users

F. DELETE /OrdinaryUsers/:id

- **Output:** Deletion confirmation
- **Purpose:** Delete an ordinary user and their complaint

G. PUT /user/: id

- **Input**: { name, email, phone }
- **Output:** Updated user object
- **Purpose:** Update user details

9.3 Complaint Management

A. POST /Complaint/:id

- **Input:** Complaint data (name, address, city, etc.)
- **Output:** Complaint document
- **Purpose:** Register a complaint (by user ID)

B. GET /status/:id

- **Output:** Complaints of a user
- **Purpose:** View complaint status by user ID

C. GET /status

- **Output:** All complaints
- **Purpose:** Admin view of all complaints

D. PUT /complaint/:complaintId

- **Input:** { status }
- **Output:** Updated complaint
- **Purpose:** Update complaint status (used by agent)

9.4 Complaint Assignment (Admin → Agent)

A. POST /assignedComplaints

- **Input:** { agentId, complaintId, status, agentName }
- **Output:** 201 Created
- **Purpose:** Assign complaint to agent

B. GET /allcomplaints/:agentId

- **Output:** Complaints assigned to a specific agent, enriched with complaint info
- **Purpose:** Agent dashboard

9.5 Messaging (Chat System)

A. POST /messages

- **Input**: { name, message, complaintId }
- **Output:** Saved message
- **Purpose:** Send message on a complaint thread

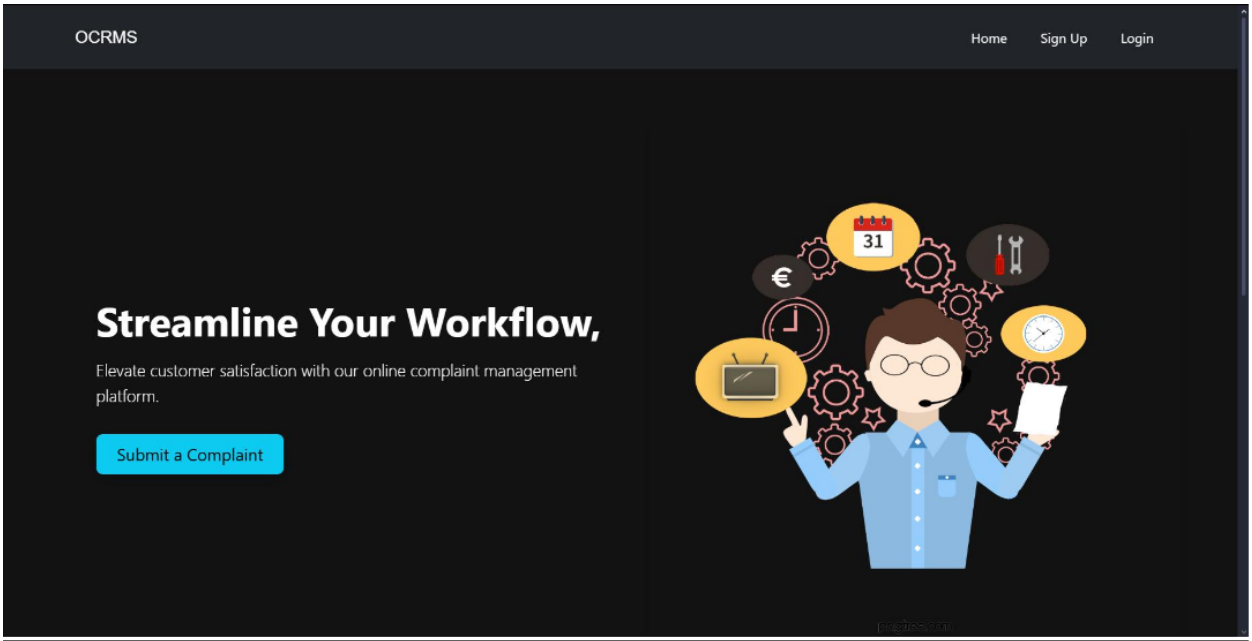
B. GET /messages/:complaintId

- **Output:** All messages for a complaint
- **Purpose:** Fetch messages related to a complaint

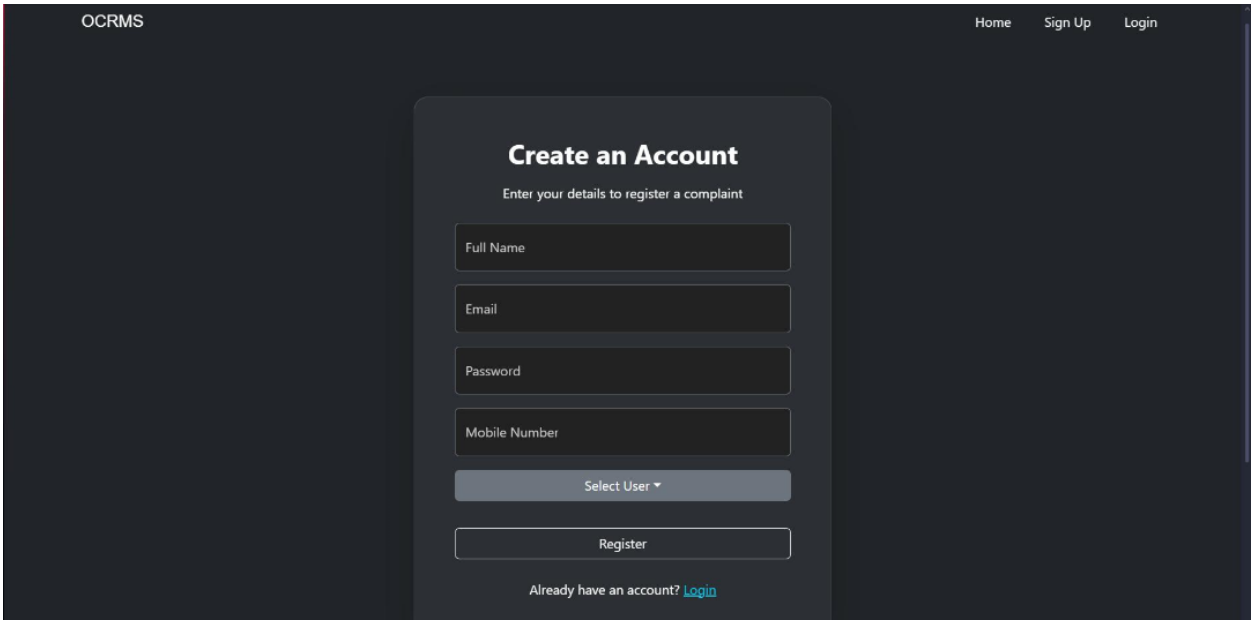
10. User Interface

10.1 Screenshots

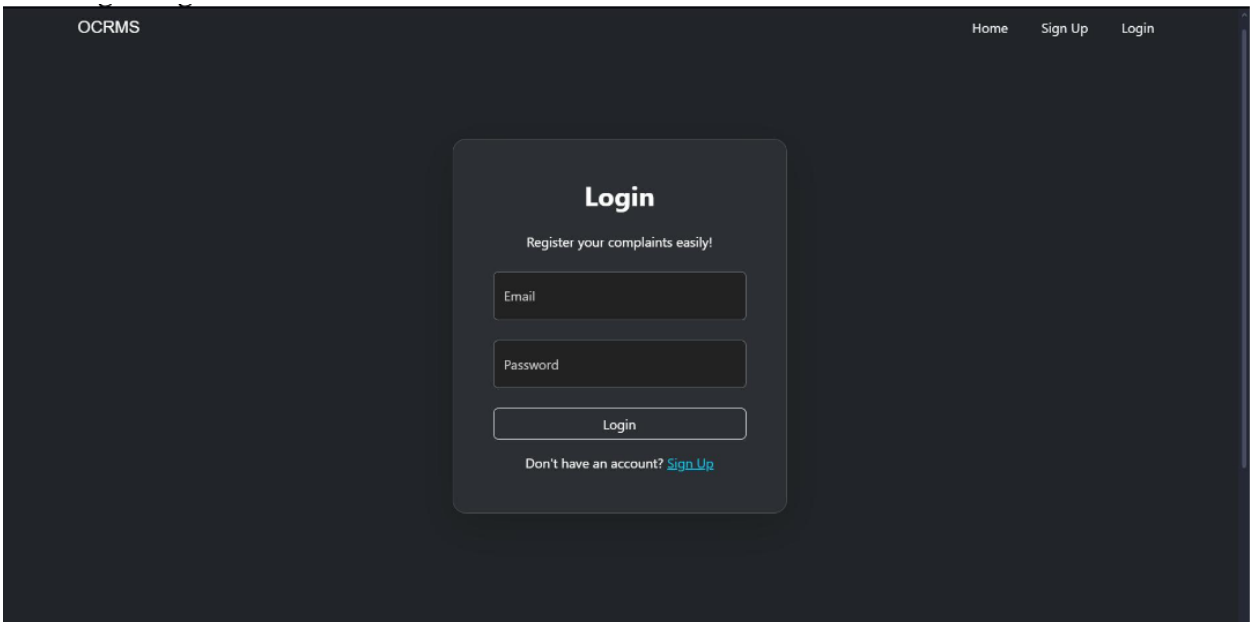
A. Home Page:



B. User Registration Page:

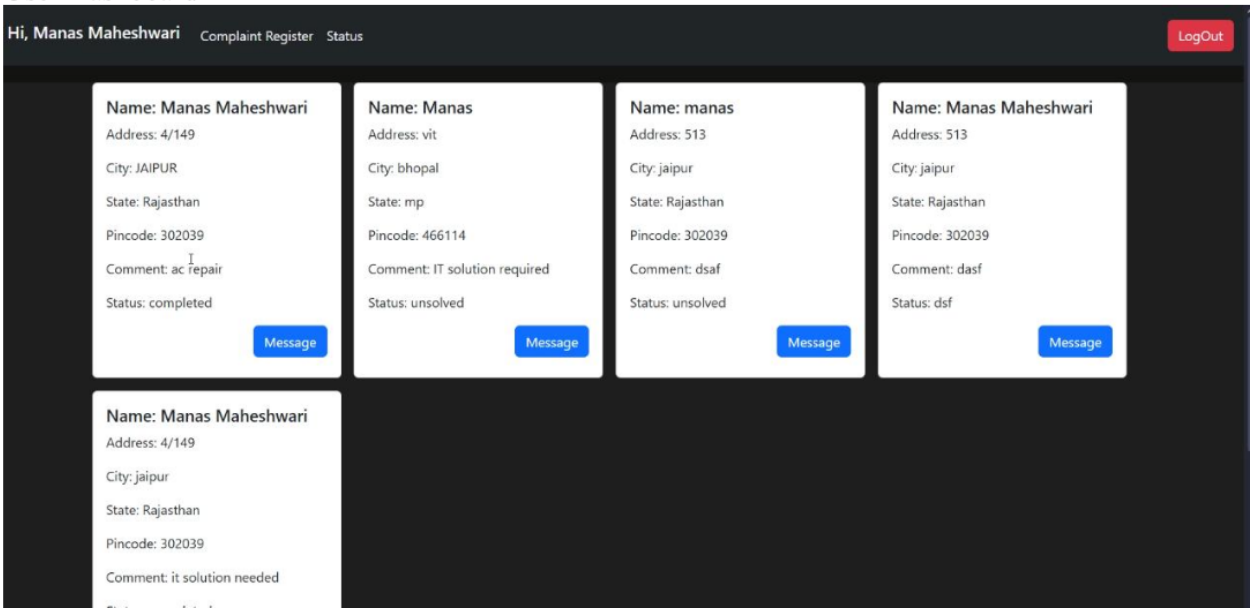


C. User Login Page:

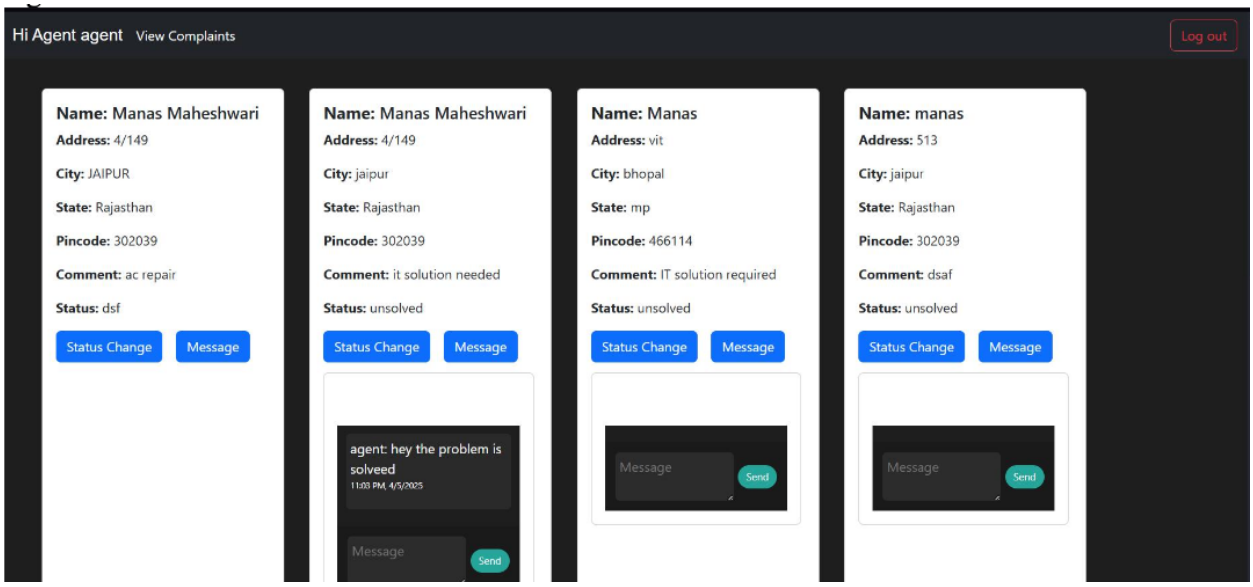


D. Dashboard (User Panel)

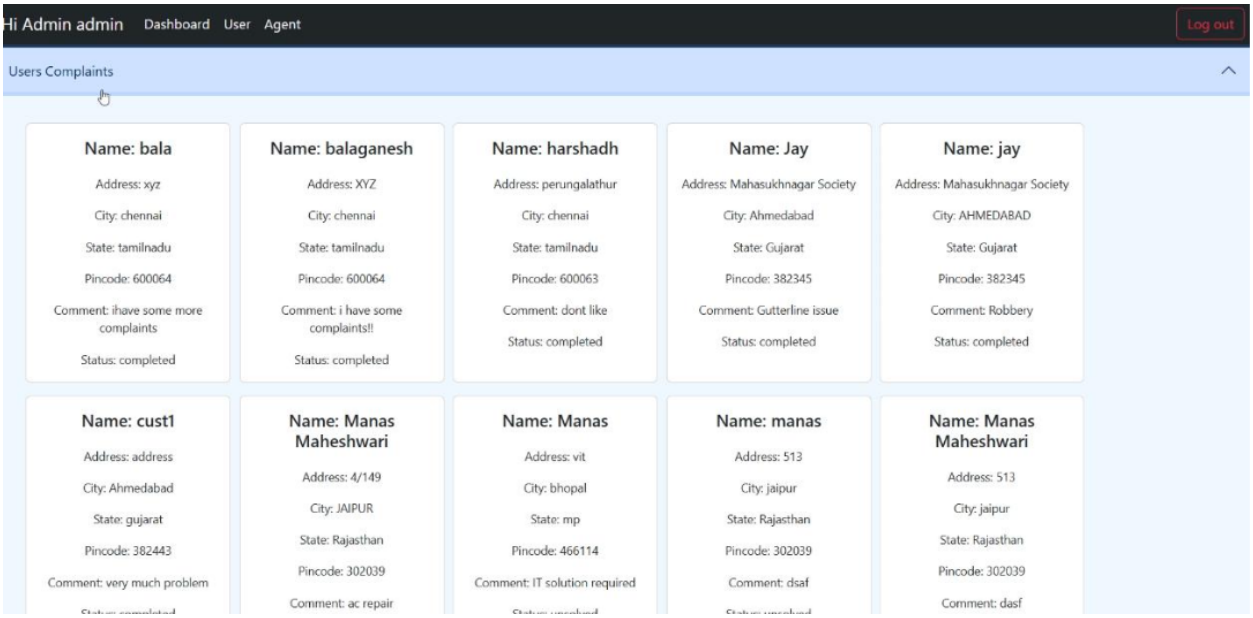
i. User Dashboard:



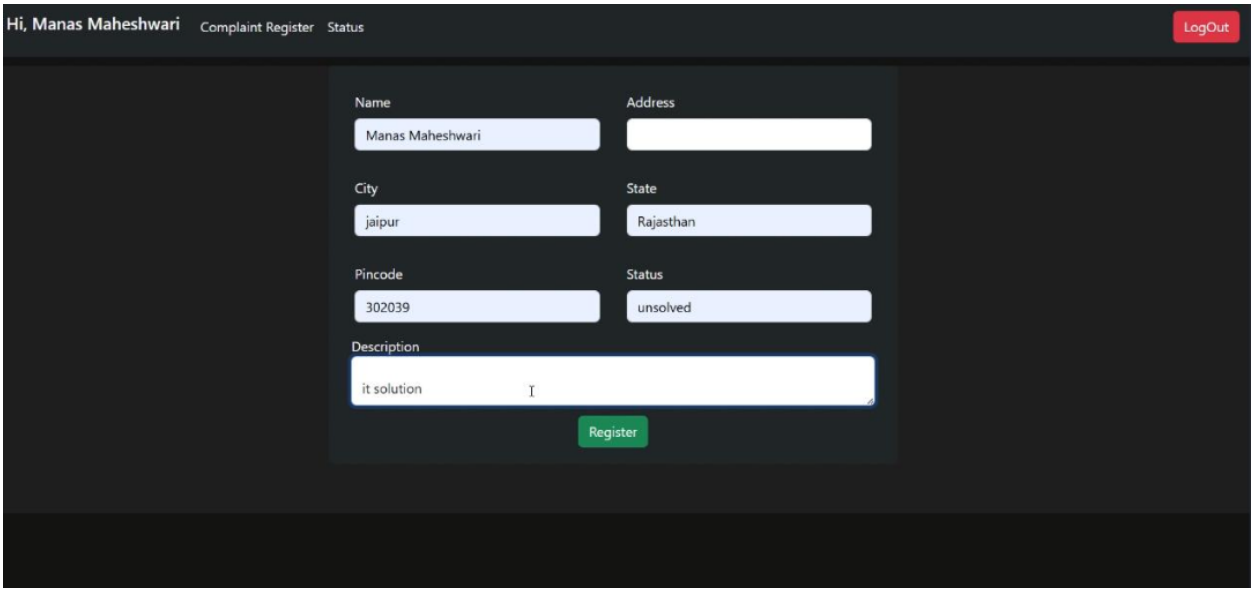
ii. Agent Dashboard:



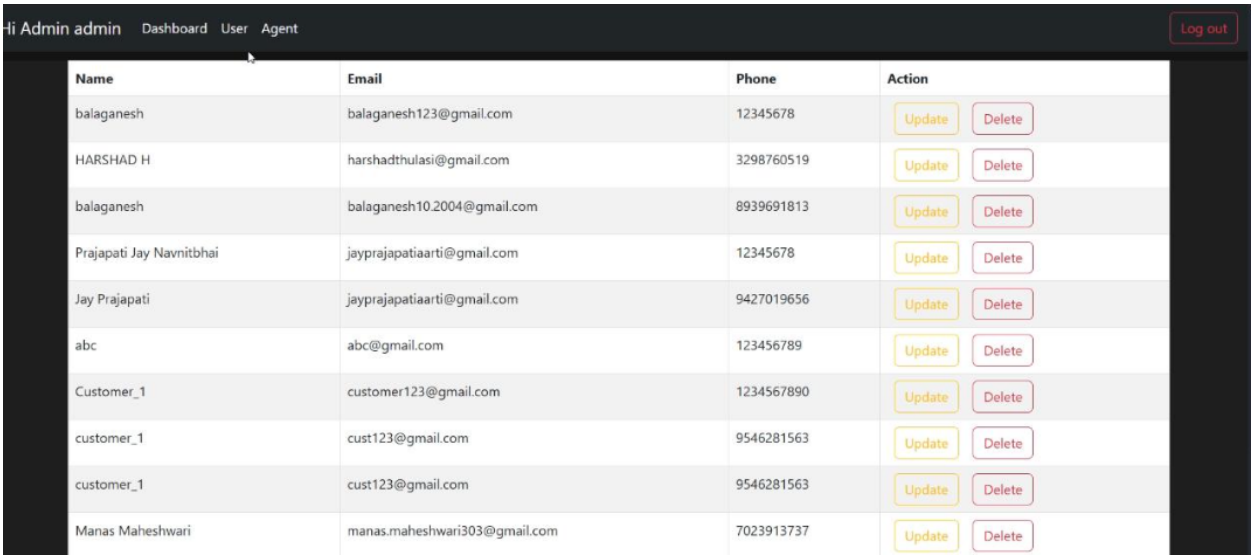
iii. Admin Dashboard



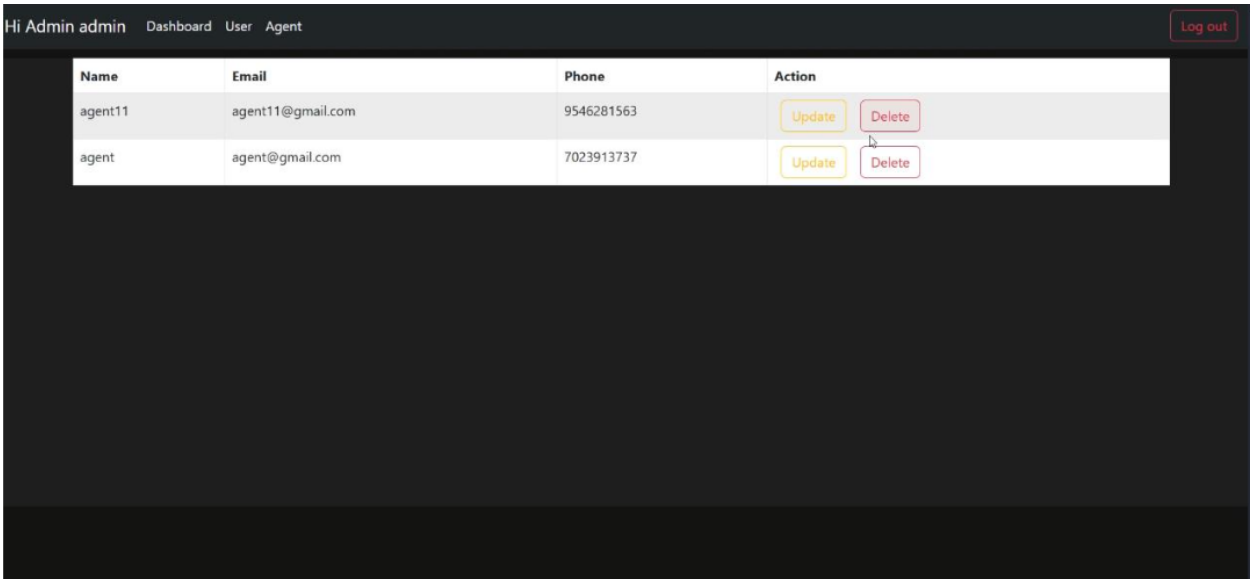
E. Submit Complaint Form



F. Admin -> User Data



G. Admin -> Agent Data



11. Demo Link

: Drive Link of OCRMS Demo -
[https://drive.google.com/file/d/1f6PEfakGXl0_U_y2VDusp7r_6EtYc-Ts/view?usp=sharing]

12. Known Issues

A. Limited Role-Based Access Control

Currently, the role-based access is implemented only for users, agents, and admins. However, deeper permission granularity (e.g., restricting certain admin operations or sub-agent roles) is not yet fully developed.

B. No File Type Validation for Uploads

The system allows image/file uploads with complaints, but there's limited validation of file types or size, which could lead to performance issues or security vulnerabilities.

C. Email Notification Delays

At times, email notifications (if configured) experience delays due to SMTP server response times or lack of queuing mechanisms.

D. Lack of Mobile Responsiveness in Some Pages

Although the frontend is built using React, some components are not fully optimized for small screen devices, which might affect usability on mobile phones or tablets.

E. Missing Logs and Error Reporting

The backend does not currently implement a centralized logging or error reporting mechanism, making it harder to trace issues in production environments.

F. Manual Environment Configuration

Developers must manually configure environment variables in local .env files. Missing or incorrect variables can lead to application crashes without clear error messages.

G. Basic Validation on User Inputs

While there is some form validation on the client side, it lacks advanced sanitization or backend validation in some forms, potentially leading to data quality issues or vulnerabilities.

13. Future Enhancements

A. Real-Time Notifications:
Implement real-time notification features via email, SMS, or in-app push messages to inform users and agents about complaint registration, assignment, status updates, and resolution. This will enhance communication, reduce uncertainty, and keep all stakeholders informed.

B. Complaint Categorization & Priority Tagging:
Add a system to categorize complaints (e.g., electrical, water, maintenance) and allow users or admins to assign priority levels (low, medium, high). This will help the backend to prioritize urgent issues and assign them to the appropriate agent teams faster and more efficiently.

C. Geo-Tagging & Location Integration:
Integrate Google Maps API or similar geolocation tools to allow users to pin the exact location of the complaint. This helps field agents to locate the issue easily and respond promptly, especially in large campuses or city-wide systems.

D. Mobile Application Support:
Develop dedicated mobile applications for both users and agents (Android/iOS). This would allow for complaint submission on-the-go, access to complaint status, agent task lists, and even uploading complaint-related images directly from mobile devices.

E. AI-Based Auto-Assignment of Agents:
Introduce an AI-powered assignment system that automatically allocates complaints to available agents based on their location, workload, complaint type, and historical performance. This reduces manual effort and ensures optimal resource utilization.

F. User Feedback and Rating Mechanism:
After resolution of a complaint, users should be allowed to rate the service and provide feedback. This feature will help the administration monitor agent performance, gather suggestions, and continuously improve the quality of service delivery.

G. Admin Analytics Dashboard:
Build a comprehensive dashboard for administrators to monitor system statistics such as:

- Number of complaints per category
- Average resolution time
- Agent-wise workload and performance
- Complaint heatmaps based on areas
- User satisfaction scores

14. Conclusion

The Online Complaint Registration Management System (OCRMS) has been developed as a centralized digital platform aimed at streamlining the process of complaint registration, assignment, and resolution within organizations or municipal services. The system addresses many of the traditional challenges associated with manual complaint handling by providing a transparent, user-friendly, and efficient interface that benefits both users and administrators alike. OCRMS empowers users by offering them a convenient way to lodge complaints online at any time, without the need for physical visits or cumbersome paperwork. With essential features like real-time status updates, complaint history tracking, and feedback submission, the platform encourages active participation and builds user trust in the system. The seamless login and registration modules ensure only authenticated access while maintaining user privacy and data security.

On the administrative front, the system simplifies workload management by allowing administrators to view, filter, and assign complaints to agents based on priority and category. Agents are given a structured view of their assigned tasks and can easily update the status of a complaint once resolved. This transparent and accountable system reduces communication gaps and ensures faster resolution, thereby improving overall service quality.

The use of modern technologies such as **React.js** for the frontend and **Node.js with Express** for the backend makes the platform robust, responsive, and scalable. With **MongoDB** serving as the NoSQL database, the application handles dynamic data effectively, while offering the flexibility to accommodate evolving requirements.

Security and role-based access control are fundamental to the system’s design, ensuring that each user (admin, agent, or general user) only accesses features pertinent to their role. Although currently implemented at a basic level, the groundwork for a secure and role-driven architecture has been laid, which can be expanded in future iterations. Despite its strengths, the system does have a few known limitations, such as limited mobile responsiveness on certain components, lack of advanced file validation, and missing logging tools for backend diagnostics. These areas present opportunities for further refinement and optimization in upcoming versions.

In conclusion, the OCRMS project marks a substantial step toward digital governance and smart public service delivery. It reflects the core objectives of e-governance - efficiency, transparency, and accountability—by providing a structured and reliable complaint redressal mechanism. With continued development, testing, and community feedback, OCRMS can become a powerful solution not just for educational campuses or organizations, but also for larger urban and rural public services. This project stands as an example of how digital platforms can transform manual processes into intelligent, agile systems that serve communities more effectively.

- END OF REPORT -