Full Stack Development with MERN

# Introduction

**Project Title:** Online Complaint Registration and Management System

**Team Lead:**   
 Selvakumar P (311521104052)

**Team Members:**

Prasanth S (311521104305)

Aswin H (311521104303)

Abbhinandan G (311521104001)

# Project Overview

The **Online Complaint Registration and Management System** is an advanced digital platform designed to streamline the complaint handling process for organizations and enhance user satisfaction. Its primary objective is to provide a structured and efficient method for users to submit complaints, track progress, and engage with support agents throughout the resolution process. By offering users a transparent and responsive experience, the system aims to bridge the communication gap between users and organizations, facilitating faster, more organized complaint management and improving the quality of services.

## 2.1 Purpose and Benefits

The purpose of this system is twofold. For organizations, it is a powerful tool for managing customer complaints, allowing them to address issues swiftly and effectively while improving customer relations. It aids in upholding compliance standards, enhancing transparency, and reducing response times—all of which are critical to building trust and reputation. By enabling easy tracking and providing real-time updates, the system makes users feel empowered and confident in the organization’s commitment to resolving their concerns.

For users, the system offers a simplified, intuitive process for complaint submission and tracking. They can securely submit details of their complaints, attach necessary documentation, and even specify locations. This streamlined experience reduces the frustration often associated with complaint resolution, as users can monitor the status of their complaints and receive updates at each stage. Additionally, direct communication with support agents creates a more interactive, personalized experience, further enhancing user satisfaction.

## Key Features

1. **User Registration and Login:** The system provides a secure, user-friendly registration process. Users can create accounts to submit and track complaints, benefiting from multi-factor authentication for enhanced security. This feature is foundational, as it ensures that only authorized users have access to sensitive complaint information. After registration, users can log in anytime to view complaint history or check updates.
2. **Complaint Submission:** The complaint submission feature allows users to log complaints quickly and comprehensively. Through a detailed form, users provide essential information such as the complaint category, a description, relevant documents or images, contact details, and location data. This organized input equips support agents with all necessary information upfront, minimizing back-and-forth communications and expediting the resolution process.
3. **Real-Time Complaint Tracking and Notifications:** With real-time tracking, users can monitor the status of their complaints at every stage. They receive notifications via email or SMS whenever there is a status update, such as the complaint being assigned, in progress, or resolved. This feature reduces user anxiety by keeping them informed and involved, ensuring they know their complaint is actively being addressed.
4. **Interaction with Assigned Agents:** Direct communication with assigned support agents is facilitated through an in-platform messaging system. This feature allows users to clarify details, provide additional information, or receive answers to their questions during the resolution process. Such two-way interaction makes the process more transparent and responsive, as users feel directly involved in resolving their issues.
5. **Intelligent Complaint Routing:** Complaints are intelligently routed to the appropriate department or support personnel based on criteria such as complaint type, priority, and agent workload. This automated routing ensures that each complaint reaches the most qualified agent without manual intervention, reducing delays and increasing the likelihood of a timely resolution. By ensuring that complaints are always directed to the appropriate team, this feature maximizes the system’s efficiency.
6. **Administrative Control and Monitoring:** A centralized dashboard gives administrators a comprehensive view of all complaints registered on the platform. They can monitor complaint statuses, assign tasks, and track system performance metrics. The dashboard offers valuable insights, such as response times and agent workloads, which allow administrators to identify bottlenecks and optimize the complaint resolution process.
7. **Feedback Collection and Analysis:** Users can provide feedback once a complaint is resolved, rating the handling process and offering comments on their experience. This feedback loop is vital for continuous improvement, as it helps organizations gauge user satisfaction, evaluate agent performance, and pinpoint areas for improvement. By analyzing this data, organizations can implement changes that lead to better customer service.
8. **Enhanced Security and Confidentiality:** The system employs multiple layers of security, including user authentication, data encryption, and strict access controls. These measures comply with data protection regulations to safeguard user information and complaint details. Confidentiality is prioritized, ensuring that users trust the system to handle their sensitive information with care. Only authorized personnel have access to specific data, and all interactions are logged for accountability.
9. **Data Analytics and Reporting Tools:** Analytical tools within the system generate reports on key metrics, such as complaint trends, resolution times, and customer satisfaction levels. These insights empower organizations to make data-driven decisions, addressing recurring issues proactively. By tracking patterns and identifying common complaints, the organization can adjust its strategies to enhance overall user satisfaction and operational efficiency.
10. **Multi-Platform Accessibility:** Built with a responsive design, the system is accessible across multiple devices, including desktops, tablets, and smartphones. This multi-platform compatibility ensures that users can access the system from any device with ease, whether they are at home, at work, or on the go. This convenience is critical to maintaining a positive user experience in a modern, mobile-driven world.

## Expected Outcomes and Impact

The **Online Complaint Registration and Management System** is expected to have a significant positive impact on both users and organizations. For users, it provides a seamless experience from complaint submission to resolution, fostering trust and satisfaction. They no longer need to make repeated inquiries about complaint status, as real-time updates and transparent communication keep them informed. For organizations, the system’s comprehensive features help manage complaints more effectively, reducing response times, increasing efficiency, and improving overall service quality. This approach strengthens customer loyalty and helps build a reputation for responsiveness and reliability.

## Future Expansion Possibilities

As the system matures, additional functionalities can be integrated to further enhance its value. These may include advanced AI-driven insights, predictive analytics to identify potential issues before they arise, and an escalation management feature for high-priority cases. Integrating multilingual support could expand the system’s accessibility, while incorporating machine learning algorithms could enable personalized complaint resolution suggestions, further refining user experience.

# 3. Architecture

## Frontend

The frontend of the **Online Complaint Registration and Management System** is built using **React**, a powerful JavaScript library ideal for creating dynamic and responsive user interfaces. React’s component-based architecture ensures the application is modular, maintainable, and scalable, providing a seamless user experience.

1. **Component-Based Structure:** The frontend is divided into reusable, independent components that manage their own state. Components like **Header**, **Footer**, **LoginForm**, **ComplaintForm**, **Dashboard**, and **ChatBox** are designed to handle specific functionalities. This modularity enhances code organization, making it easy for developers to update or reuse components across different parts of the application.
2. **Single-Page Application (SPA):** By utilizing **React Router**, the app functions as a **Single-Page Application (SPA)**. This means that users can navigate between different pages (dashboard, complaint submission form, and tracking page) without reloading the entire page. The SPA approach minimizes load times and provides a seamless experience by updating only necessary content.
3. **State Management:** React’s **useState** and **useContext** hooks are used for managing component-level and global states, respectively. For larger-scale state management, **Redux** or **React’s Context API** can be incorporated to centralize control over state across components, ensuring consistency in data like user authentication, complaint status, and notifications.
4. **API Integration with Axios:** The frontend communicates with the backend through **RESTful APIs** using **Axios**, a popular HTTP client for handling requests and responses. Axios is configured globally to manage requests related to complaint submission, tracking, user authentication, and notifications, with built-in support for error handling, token-based authentication, and response management.
5. **Responsive Design with Material-UI and Bootstrap:** To provide a responsive and visually appealing interface, the app utilizes **Material-UI** and **Bootstrap**. Material-UI, based on Google’s Material Design principles, ensures a clean and accessible design, while Bootstrap adds additional styling capabilities, ensuring the app’s accessibility across a wide range of devices, including desktops, tablets, and mobile phones.
6. **Real-Time Updates with WebSockets:** For real-time interactions such as agent-user communication and live complaint status updates, **Socket.IO** is integrated with React. This allows bidirectional communication between the frontend and backend, enabling instant updates to the complaint status, notifications, and chat messages without page reloads, improving user engagement.
7. **Form Validation and Error Handling: Formik** and **Yup** are used for efficient form validation, ensuring users input correct and complete information during complaint registration or profile updates. This reduces errors and enhances data quality before submission.
8. **Security Measures:** Security is implemented on the frontend by adopting best practices such as **token-based authentication** and **secure data handling**. Sensitive data, like authentication tokens, are stored in **HTTP-only cookies** or secure storage solutions to prevent unauthorized access. CSRF protection and error handling mechanisms are also in place to prevent vulnerabilities.
9. **Optimized Performance with Code Splitting and Lazy Loading:** To optimize performance, **code splitting** and **lazy loading** are employed. Components are dynamically loaded only when needed, such as loading the chat interface when a user starts a conversation. This approach minimizes initial load time, keeping the application responsive.
10. **Testing and Debugging with React Developer Tools and Jest:** The frontend is thoroughly tested using **Jest** and **React Testing Library** to validate component functionality and UI behavior. **React Developer Tools** help in debugging by allowing developers to inspect component hierarchies, track states, and identify issues.

## Backend

The backend of the **Online Complaint Registration and Management System** is developed using **Node.js** and **Express.js** to provide efficient, scalable, and fast server-side operations. This architecture enables real-time processing, secure data management, and robust support for a variety of client-side interactions.

* 1. **Express.js Framework for Routing and Middleware**

**Express.js** serves as the core framework, managing all HTTP routing and middleware. Each API endpoint (e.g., **/login, /submit-complaint, /track-complaint**) follows a RESTful design for predictable and maintainable requests. Middleware functions handle error management, logging, authentication, and request validation.

* 1. **Modular Structure**

The backend follows a modular structure for better organization:

* **Controllers:** Handle incoming requests, interact with services for business logic, and return appropriate responses.
* **Services:** Contain core business logic for operations like complaint submission, tracking, user registration, and agent assignment.
* **Models:** Define the structure of MongoDB collections using **Mongoose** schemas, ensuring data integrity and consistency.
  1. **Database Management with MongoDB and Mongoose**

MongoDB, a **NoSQL database**, stores user profiles, complaints, statuses, and interactions. Mongoose serves as the **ODM** (Object Data Modeling) library, defining schemas, enforcing validation, and streamlining database interactions.

* 1. **RESTful API Design**

The backend exposes **RESTful APIs** for key operations such as user registration, complaint submission, and tracking. Each API endpoint includes input validation and error handling to maintain data accuracy and provide useful error messages to the frontend.

* 1. **Authentication and Authorization with JWT**

The backend uses **JWT** (JSON Web Tokens) for user authentication. Upon successful login, a JWT is generated, stored securely (e.g., in **HTTP-only cookies**), and used to validate the user's identity for subsequent requests. Role-based access control (e.g., user, agent, admin) restricts certain actions to authorized users only.

* 1. **Real-Time Communication with Socket.IO**

For real-time communication, such as status changes and live chat between users and agents, the backend integrates **Socket.IO** to establish bidirectional communication, enabling instant updates and interactive complaint resolution.

* 1. **Error Handling and Logging**

Centralized error-handling middleware formats error responses and sends them back to the client. Winston is used for logging, recording request logs, errors, and critical events, which are monitored to assist with debugging and system maintenance.

* 1. **Data Validation with Joi**

The Joi library is employed to validate incoming data for each API request, ensuring that data adheres to required formats and fields before processing, thus improving data quality and reducing errors.

* 1. **Security Practices**

To secure data and protect against common vulnerabilities:

* **Input Sanitization:** Prevents attacks like **SQL** injection and **XSS**.
* **CORS Configuration:** Restricts access to trusted domains only.
* **Rate Limiting:** Prevents abuse by limiting the number of requests a client can make in a specified time period.
  1. **Scalability and Load Balancing**

The backend is designed to scale horizontally by adding more server instances as traffic increases. Load balancers distribute incoming requests across these instances, ensuring optimal performance during high-traffic periods.

## Database

The Online Complaint Registration and Management System uses MongoDB, a NoSQL database, to store and manage data efficiently. Its document-based structure allows for flexible and scalable storage of diverse data types. Mongoose, an ODM library, interacts with the MongoDB database, streamlining schema definitions and operations.

### Database Schema

The database schema consists of several collections that manage different aspects of the complaint management system.

#### Users Collection

The Users collection stores data related to users who register and submit complaints.

**const UserSchema = new mongoose.Schema({**

**username: { type: String, required: true, unique: true },**

**email: { type: String, required: true, unique: true },**

**password: { type: String, required: true },**

**role: { type: String, enum: ['user', 'agent', 'admin'], default: 'user' },**

**complaints: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Complaint' }],**

**createdAt: { type: Date, default: Date.now }**

**});**

**Fields:**

* **username:** Unique identifier for the user.
* **email:** User's email address for communication.
* **password:** Encrypted password for user login.
* **role:** Defines user roles (admin, agent, user).
* **complaints:** References complaints submitted by the user.
* **createdAt:** Date of registration.

#### Complaints Collection

The Complaints collection stores complaint details.

**const ComplaintSchema = new mongoose.Schema({**

**user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**title: { type: String, required: true },**

**description: { type: String, required: true },**

**status: { type: String, enum: ['submitted', 'in-progress', 'resolved', 'closed'], default: 'submitted' },**

**priority: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },**

**assignedAgent: { type: mongoose.Schema.Types.ObjectId, ref: 'Agent' },**

**createdAt: { type: Date, default: Date.now },**

**updatedAt: { type: Date },**

**resolutionDetails: { resolutionStatus: { type: String, enum: ['resolved', 'unresolved'], default: 'unresolved' }, resolutionDate: { type: Date }, notes: { type: String } }**

**});**

**Fields:**

* **user:** Reference to the user submitting the complaint.
* **title:** Short description of the complaint.
* **description:** Detailed complaint information.
* **status:** Current complaint status.
* **priority:** Complaint priority (low, medium, high).
* **assignedAgent:** Agent responsible for resolving the complaint.
* **createdAt:** Date of complaint submission.
* **updatedAt:** Date when the complaint was last updated.
* **resolutionDetails:** Contains resolution status and notes for closed complaints.

### 3.3.2. Database Interactions with MongoDB

The **Online Complaint Registration and Management System** uses MongoDB in combination with **Mongoose** to efficiently store and retrieve data. The backend performs **CRUD (Create, Read, Update, Delete)** operations to manage collections such as **Users**, **Complaints**, and **Agents**. Below are the key interactions within the database.

#### 3.3.2.1. User Registration

* When a new user registers, a new document is created in the **Users** collection, with a unique identifier (user.\_id) generated automatically.
* Passwords are hashed before storage to ensure security.
* Example code for registration:

**const newUser = new User({**

**username,**

**email,**

**password: hashedPassword, // Using bcrypt for hashing**

**role: 'user'**

**});**

**await newUser.save();**

#### 3.3.2.2. Complaint Submission

* On complaint submission, a new document is created in the **Complaints** collection. This document includes fields like title, description, status, and priority.
* The user field in the **Complaints** collection links each complaint to the user who submitted it. The status field is initially set to submitted.
* If an agent is assigned at submission, the assignedAgent field is populated accordingly.

#### 3.3.2.3. Complaint Status Update

* As complaints are processed, the status field (submitted, in-progress, resolved, closed) and updatedAt timestamp are updated using Mongoose’s **update** or **findOneAndUpdate** methods.
* Example:

**Complaint.findOneAndUpdate(**

**{ \_id: complaintId },**

**{ status: newStatus, updatedAt: Date.now() }**

**);**

#### 3.3.2.4. Agent Assignment

* Admins or the system can assign complaints to agents by updating the assignedAgent field of the complaint document.
* Correspondingly, the agent’s **assignedComplaints** field is updated with the complaint’s \_id.
* Example:

**Complaint.findByIdAndUpdate(**

**complaintId,**

**{ assignedAgent: agentId }**

**);**

**Agent.findByIdAndUpdate(**

**agentId,**

**{ $push: { assignedComplaints: complaintId } }**

**);**

#### 3.3.2.5. Complaint Resolution

* Once resolved, the **resolutionDetails** field of the complaint document is updated with details such as resolutionStatus, resolutionDate, and any notes provided by the agent. The status is also updated to resolved.

#### 3.3.2.6. User-Complaint Relationship

* The **Users** collection includes a reference to each user’s complaints through a complaints field, which is an array of **ObjectId** references to the **Complaints** collection.
* This setup allows for easy querying of all complaints filed by a specific user.

#### 3.3.2.7. Data Retrieval

* Data retrieval is handled using Mongoose queries such as **find()**, **findOne()**, and **populate()** to gather related documents.
* Example: To fetch a complaint along with its assigned agent, the query may look like:

**Complaint.findById(complaintId).populate('assignedAgent');**

### 3.3.3. Indexes for Optimization

To optimize query performance, especially for frequently accessed fields, MongoDB indexes are used on the following fields:

* **Users Collection:** The email field is indexed to speed up user lookup by email.

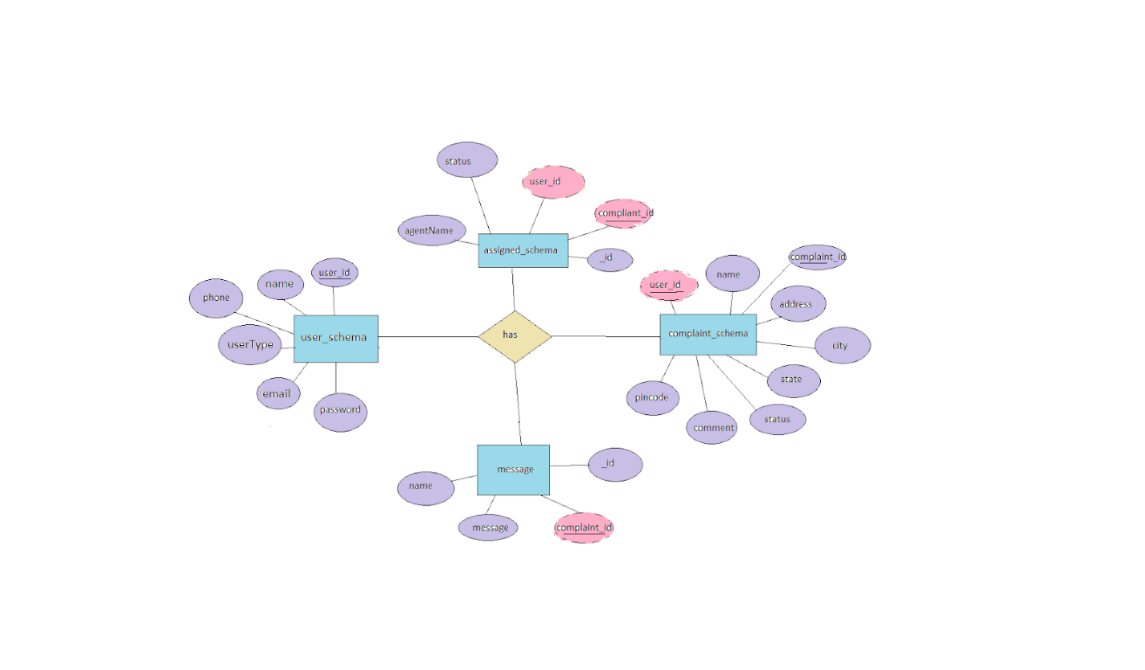
**UserSchema.index({ email: 1 });**

* **Complaints Collection:** The status field is indexed to allow efficient filtering and retrieval of complaints based on their status (e.g., retrieving only in-progress complaints).

**ComplaintSchema.index({ status: 1 });**

## ER Diagram

**ER DIAGRAM:-**



## Technical Architecture

**TECHNICAL ARCHITECURE:-**



# Setup Instructions

## Prerequisites

Here are the key prerequisites for developing a full-stack application using **Node.js**, **Express.js**, **MongoDB**, **React.js**:

**Node.js and npm:**

**Node.js** is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

Download: <https://nodejs.org/en/download/>

Installation instructions: https://nodejs.org/en/download/package-manager/

**Express.js:**

**Express.js** is a fast and minimalist web application framework for **Node.js**. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.

Install **Express.js**, a web application framework for **Node.js**, which handles server-side routing, middleware, and API development.

**Installation:** Open your command prompt or terminal and run the following command:

**npm install express**

**MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

Download: <https://www.mongodb.com/try/download/community>

Installation instructions: https://docs.mongodb.com/manual/installation/ 

**React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

**HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

**Database Connectivity**: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:

[https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/](https://www.section.io/engineering-education/nodejs-%20mongoosejs-mongodb/ )

**Front-end Framework**: Utilize Reactjs to build the user-facing part of the application, including entering complaints, status of the complaints, and user interfaces for the admin dashboard.

For making better UI we have also used some libraries like material UI and boostrap.

**Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

Git: Download and installation instructions can be found at:

<https://git-scm.com/downloads>

**Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

Visual Studio Code: Download from <https://code.visualstudio.com/download>

To run the existing Video Conference App project downloaded from GitHub:

Follow below steps:

Clone the Repository:

* Open your terminal or command prompt.
* Navigate to the directory where you want to store the e-commerce app.
* Execute the following command to clone the repository:

**git clone**: <https://github.com/awdhesh-student/complaint-registery.git>

**Install Dependencies:**

* + Navigate into the cloned repository directory:

**cd complaint-registery**

* + Install the required dependencies by running the following commands:

**cd frontend**

**npm install**

**cd ../backend**

**npm install**

* + Start the Development Server:
    - To start the development server, execute the following command:

**npm start**

* + - The online complaint registration and management app will be accessible at [http://localhost:3000](http://localhost:3000/)

You have successfully installed and set up the online complaint registration and management app on your local machine. You can now proceed with further customization, development, and testing as needed.

## Installation

Follow these steps to set up the **Online Complaint Registration and Management System** locally:

### 4.2.1. Clone the Repository

**Clone the project repository:**

git clone **https://github.com/your-username/complaint-management-system.git**

**cd complaint-management-system**

### 4.2.2. Install Dependencies

* Navigate to the **client** directory and install frontend dependencies:

**cd client**

**npm install**

* Navigate to the **server** directory and install backend dependencies:

**cd ../server**

**npm install**

### 4.2.3. Set Up Environment Variables

1. In the **server** directory, create a .env file with the following variables:

***MONGO\_URI=mongodb://localhost:27017/complaintDB***

***JWT\_SECRET=your-secret-key***

***PORT=5000***

1. Replace **mongodb://localhost:27017/complaintDB** with your actual MongoDB URI if using MongoDB Atlas.

# 5. Folder Structure

* **Client:**

The **client** directory follows a typical React project structure:

graphql

Copy code

client/

│

├── public/ # Static assets (e.g., index.html, images)

├── src/ # React components, state management, and hooks

│ ├── components/ # Reusable UI components

│ ├── pages/ # Individual pages (e.g., Home, Login, Dashboard)

│ ├── services/ # API calls to backend

│ ├── App.js # Main app component

│ └── index.js # Entry point for React app

└── package.json # Frontend dependencies and scripts

* **Server:**

The **server** directory is structured as follows:

bash

Copy code

server/

│

├── models/ # Mongoose models (e.g., User, Complaint, Agent)

├── routes/ # API routes (e.g., /auth, /complaints, /agents)

├── controllers/ # Business logic for each route

├── middlewares/ # Authentication, error handling, etc.

├── config/ # Configuration files (e.g., DB connection)

├── server.js # Main entry point for the server

└── .env # Environment variables

# 6. Running the Application

## Frontend

1. **Setup React Application:**

Bringing Customer Care Registry to life involves a three-step development process. First, a solid foundation is built using React.js. This includes creating the initial application structure, installing necessary libraries, and organizing the project files for efficient development. Next, the user interface (UI) comes to life. To start the development process for the frontend, follow the below steps.

* Install required libraries.
* Create the structure directories.

1. **Design UI components:**

Reusable components will be created for all the interactive elements you'll see on screen, from stock listings and charts to buttons and user profiles. Next, we'll implement a layout and styling scheme to define the overall look and feel of the application. This ensures a visually-appealing and intuitive interface.  Finally, a navigation system will be integrated, allowing you to effortlessly explore different sections of Customer Care Registry, like making specific complaints or managing your Product complaints.

1. **Implement frontend logic:**

In the final leg of the frontend development, we'll bridge the gap between the visual interface and the underlying data. It involves the below stages.

* Integration with API endpoints.
* Implement data binding.
* In the **client** directory:

**npm start**

## Backend

* **Set Up Project Structure:**
* Create a new directory for your project and set up a package.json file using npm init command.
* Install necessary dependencies such as Express.js, Mongoose, and other required packages.
* **Set Up Project Structure:**
  + Create a new directory for your project and set up a package.json file using npm init command.
  + Install necessary dependencies such as Express.js, Mongoose, and other required packages.
* **Create Express.js Server:**
  + Set up an Express.js server to handle HTTP requests and serve API endpoints.
  + Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.
* **Define API Routes:**
  + Create separate route files for different API functionalities such as authentication, stock actions, and transactions.
  + Implement route handlers using Express.js to handle requests and interact with the database.
* **Implement Data Models:**
  + Define Mongoose schemas for the different data entities like Bank, users, transactions, deposits and loans.
  + Create corresponding Mongoose models to interact with the MongoDB database.
  + Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
* **User Authentication:**
  + Implement user authentication using strategies like JSON Web Tokens (JWT) or session-based authentication.
  + Create routes and middleware for user registration, login, and logout.
  + Set up authentication middleware to protect routes that require user authentication.
* **Handle new transactions:**
  + Allow users to make transactions to other users using the user’s account id.
  + Update the transactions and account balance dynamically in real-time.
* **Admin Functionality:**
  + Implement routes and controllers specific to admin functionalities such as fetching all the data regarding users, transactions, stocks and orders.
* **Error Handling:**
  + Implement error handling middleware to catch and handle any errors that occur during the API requests.
  + Return appropriate error responses with relevant error messages and HTTP status codes.
* In the **server** directory:

**npm start**

# 7. API Documentation

## User Registration

* **POST** /api/auth/register
  + **Request Body:**

{

"username": "john\_doe",

"email": "john@example.com",

"password": "securePassword123"

}

* + **Response:**

{

"message": "User registered successfully"

}

## User Login

* **POST** /api/auth/login
  + **Request Body:**

{

"email": "john@example.com",

"password": "securePassword123"

}

* + **Response:**

{

"token": "JWT\_TOKEN"

}

## Submit Complaint

* **POST** /api/complaints
  + **Request Body:**

{

"title": "Defective Product",

"description": "The product received was damaged.",

"priority": "high"

}

* + **Response:**

{

"message": "Complaint submitted successfully"

}

# 8. Authentication

Authentication is handled using **JSON Web Tokens (JWT)**:

* **Login:**  
  When a user logs in, a JWT token is generated and returned.
* **Authorization:**  
  The JWT token is included in the Authorization header for protected routes (e.g., submitting complaints, updating profiles). The backend verifies the token and grants access if valid.
* **Token Expiry:**  
  Tokens are set to expire after a predefined duration (e.g., 1 hour). Users must log in again once expired.

# 9. User Interface

Provide visual representations of key user interfaces, including:

* Registration Page
* Login Page
* Dashboard (Complaint List)
* Complaint Submission Form
* Admin Dashboard for Monitoring Complaints

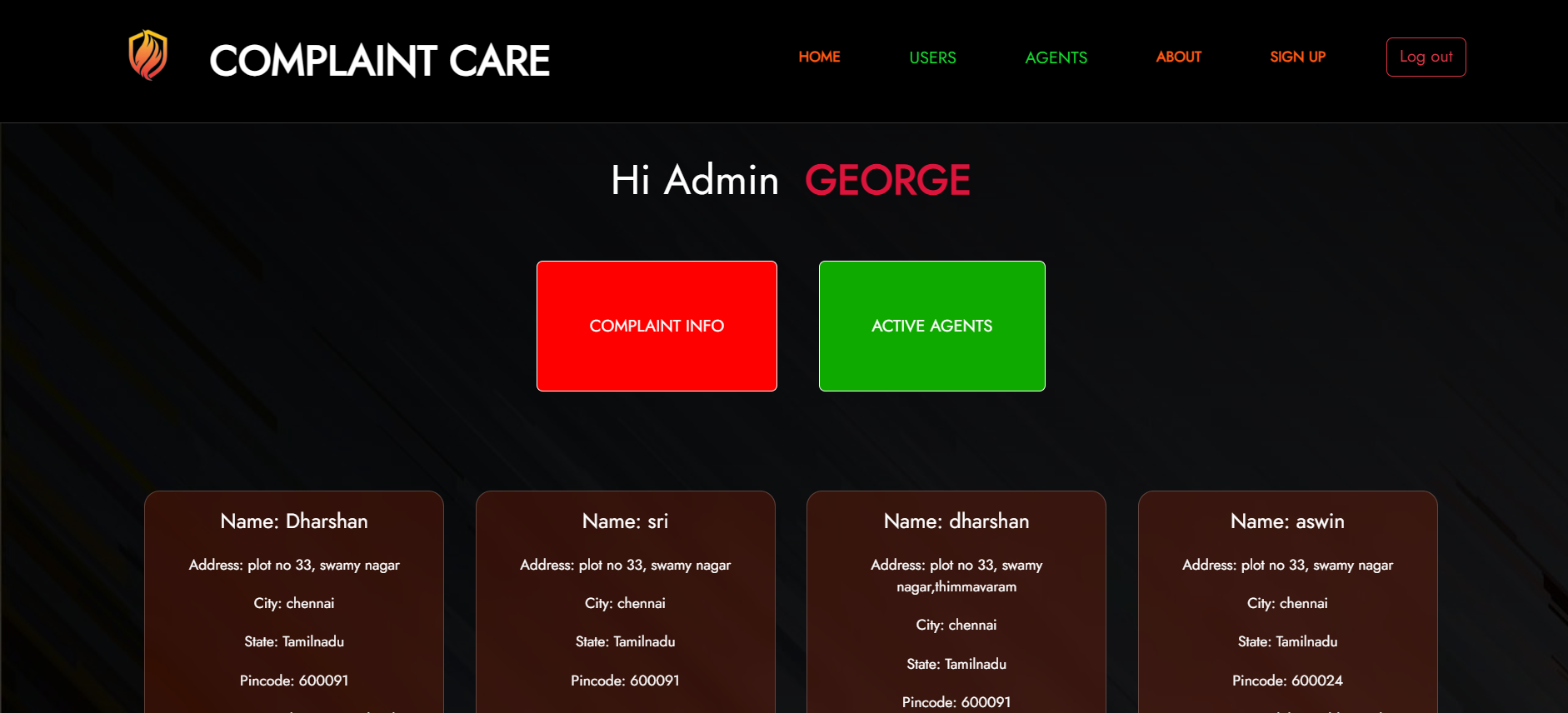
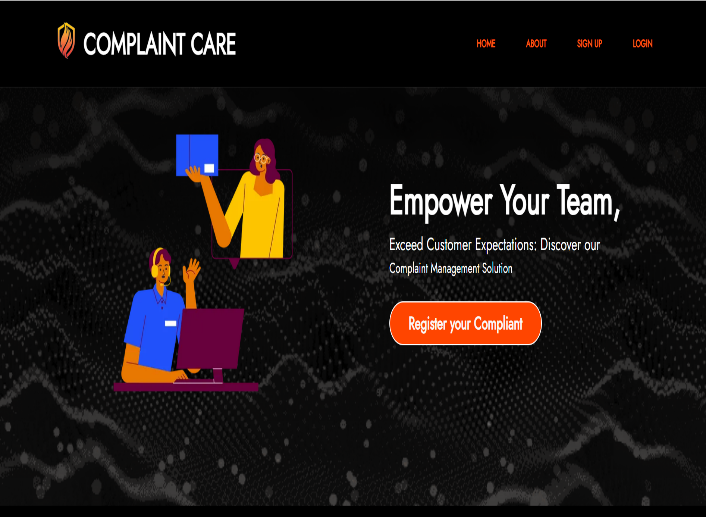
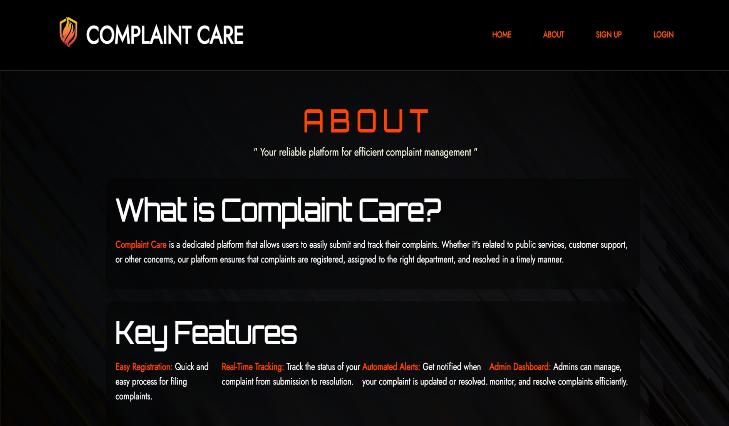
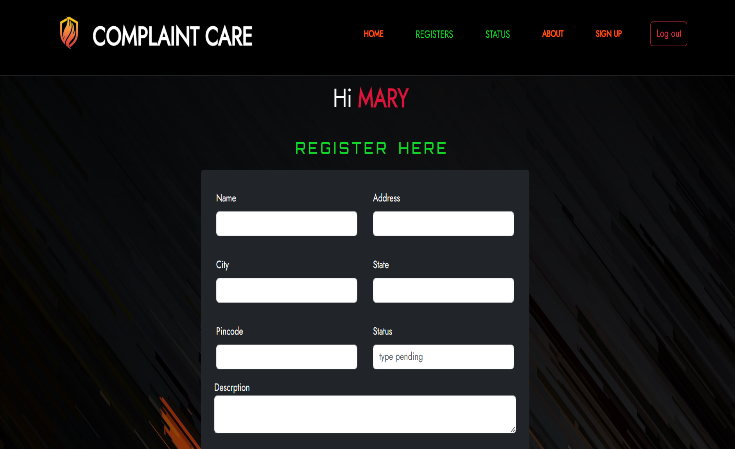
# 10. Testing

* **Testing Strategy:**
  + **Unit Tests**: Used for testing individual functions, such as user login or complaint submission.
  + **Integration Tests**: Ensures that the frontend and backend communicate as expected.
* **Tools Used:**
  + **Jest** for unit and integration tests.
  + **Supertest** for API endpoint testing.

To run tests:

**npm test**

# 11. Screenshots or Demo



# **LINK :** **https://drive.google.com/file/d/1rtJ0aiGEIS0Q7zgW8gzGUmRUap09YiqF/view?usp=sharing**

# 12. Known Issues

Document any known bugs or limitations, such as:

* Bug 1: Notifications not sent for some complaint updates.
* Bug 2: Complaints not being sorted by priority correctly.

# 13. Future Enhancements

Outline potential future features or improvements:

* **Real-Time Chat:** Allow users and agents to interact through live chat.
* **Complaint Analytics:** Add graphs or reports to track complaint statistics.
* **Admin Dashboard Enhancements:** Improve complaint categorization and filtering for admins.