

Table of Contents

1.Introduction.....	1
2.Project Overview.....	1
3.Architecture.....	2
4.Setup Instructions.....	3
5.Folder Structure.....	4
6.Running the Application.....	5
7.API Documentation.....	6
8.Authentication.....	7
9.User Interface.....	8
10. Testing.....	9
11.Screenshots(or) Demo.....	10
12.Known Issues.....	12
13.Future Enhancement.....	13

1.Introduction

- **Project Title:** Online Complaint Registration and Management System.
- **Team Roles:**
 - **Sachin Anandharaj :** Database design, and implementation.
 - **M. P. Praveen Raja :** Backend development and implementation.
 - **K. P. Tharun :** Database design and implementation.
 - **K. Navin Raj :** Backend development and implementation.

2. Project Overview

Purpose

The system is designed to serve as a centralized platform for efficient complaint management, streamlining resolution processes and enhancing customer satisfaction. Its core features include user registration, complaint submission, tracking, and real-time notifications. The platform facilitates seamless user-agent interactions and leverages intelligent routing algorithms to assign complaints to the most suitable personnel, ensuring prompt and effective handling. With a strong emphasis on security, the system incorporates robust authentication mechanisms, data encryption, and compliance with data protection regulations to safeguard user confidentiality.

Features

- **User Registration and Authentication:**

Users can securely register and login to manage their complaints and monitor progress.
- **Complaint Submission:**

Users can submit complaints by providing details such as their name, a description of the issue, and their address.
- **Progress Tracking and Notifications:**

Users can track the status of their complaints, view updates, and receive notifications via email or SMS regarding any changes or resolutions.
- **Agent Interaction:**

Users can directly communicate with the assigned agent to discuss or clarify complaint-related matters.
- **Complaint Assignment and Routing:**

The system assigns complaints to the appropriate department or personnel based on intelligent routing algorithms to optimize resource allocation and ensure efficient resolution.

3.Architecture

Frontend Architecture

The frontend is designed with React.js, focusing on modularity and reusability. The application follows a structured directory setup, ensuring clean organization and scalability. Core functionalities are implemented as reusable UI components (e.g., forms, buttons, navigation bars), promoting a consistent and intuitive design. React's state management and data binding enable dynamic updates and seamless integration with backend APIs for real-time interactions. Key features, including complaint registration, tracking, and admin dashboards, are styled for responsiveness and visual appeal using Material-UI and Bootstrap.

Backend Architecture

The backend is powered by Node.js and Express.js, providing a robust and scalable server-side framework. The architecture adopts a well-structured directory hierarchy with modular routes for functionalities like authentication, complaint management, and admin operations. APIs interact with the MongoDB database via Mongoose for efficient data handling. Middleware such as body-parser and CORS ensures smooth data parsing and secure cross-origin communication.

Database Design

The database leverages MongoDB for scalable and efficient data storage. Entities like users, complaints, assigned complaints, and messages are structured using Mongoose schemas, enabling seamless CRUD operations. Relationships are established through object references



complaintId) to link users with their complaints and assigned agents. Collections are optimized for performance, supporting scalability, complex queries, and robust data retrieval, ensuring smooth database operations.

4. Setup Instructions

Prerequisites

To ensure the backend and frontend function seamlessly, the following libraries are required:

Backend Libraries:

1. Node.js: Server-side JavaScript runtime.
2. MongoDB: NoSQL database for application data storage.
3. Bcrypt: For secure password hashing.

Frontend Libraries:

1. React.js: JavaScript library for building user interfaces.
2. Material-UI: React component library for responsive design and styling.
3. Bootstrap: CSS framework for styling and layout.
4. Axios: Library for handling HTTP requests.

Installation:

1. **Node.js & npm:**

Install Node.js to run server-side scripts and manage dependencies.

2. **Express.js:**

Set up the web application framework for APIs and routing using:

```
npm install express
```

3. **MongoDB:**

Install the NoSQL database for storing data. React.js:

Install React.js for building dynamic user interfaces.

4. **Database Connectivity:**

Connect the Node.js server to MongoDB using Mongoose for seamless CRUD operations.

5. **Libraries:**

Integrate Material-UI and Bootstrap for enhanced styling and responsiveness.

6. **Version Control:**

Use Git for managing code versions. Download Git.

Use Github for the version control system.

7. **Development Tools:**

Utilize Visual Studio Code as the code editor for efficient development.

5. Folder Structure

Client (Frontend)

The React-based frontend is structured for modularity and scalability, with reusable components ensuring maintainability and a consistent design. Key features include:

1. Component-Based Architecture:

UI is broken into modular components like `Header`, `Footer`, `ComplaintForm`, and `AdminDashboard`, each handling specific functionality.

2. Routing:

Navigation between pages such as complaint submission, status tracking, and admin dashboards is implemented using React Router.

3. State Management:

State is managed locally within components or globally using Context API for shared data across the application.

4. Styling:

Material-UI and Bootstrap are used for a responsive and visually consistent user interface.

Server (Backend)

The Node.js backend is designed with the MVC (Model-View-Controller) pattern, ensuring scalability, maintainability, and a clean codebase. Key elements include:

1. Routing:

Express.js handles API endpoints for user management, complaint processing, and admin functions.

2. Controllers:

Centralized business logic processes incoming requests, sends appropriate responses to the client.

3. Models:

Mongoose schemas define the structure of MongoDB collections, such as `User` and `Complaint`.

4. Database Integration:

MongoDB is connected via Mongoose for efficient and seamless CRUD operations.

5. Environment Variables:

Used to securely manage configuration details, such as database URIs and API keys.

6. Running the Application

Commands to Start Frontend and Backend Servers Locally

Frontend

1. **Navigate to the Frontend Directory**

Change the terminal directory to the frontend folder to access the client-side code.

2. **Install Dependencies**

Ensure all necessary packages required for the frontend are installed to enable functionality.

3. **Start the Development Server**

Launch the React development server to run and preview the user interface locally.

4. **Access the Application**

By default, the frontend can be accessed at: <http://localhost:3080>.

Backend

1. **Navigate to the Backend Directory**

Move to the backend directory to manage server-side functionalities and APIs.

2. **Install Dependencies**

Install all required libraries, such as Express.js and Mongoose, to ensure seamless backend operations.

3. **Set Up Environment Variables**

Configure the `.env` file with key details, including the database connection string (`MONGO_URI`), JWT secret key (`JWT_SECRET`), and server port (`PORT`).

4. **Start the Backend Server**

Launch the backend development server to enable API connectivity and interactions.

5. **Access the Backend**

The backend will be accessible at the port specified in the `.env` file. For example, the default location is <http://localhost:3080>.

7. API Documentation

Backend API Documentation

- a. **User Authentication**

Register Endpoint: Enables users to create an account by providing required details like name, email, and password.

b. Login Endpoint:

Allows registered users to authenticate themselves and receive an access token for secure interactions with the application.

c. Complaint Management

Submit Complaint: Enables users to register complaints by providing details like issue description, address, and user ID.

d. View Complaint Status:

Provides users with the ability to track the progress of their complaints, including updates on the assigned agent and status.

e. Search Complaints

Enables users and admins to search for complaints based on keywords, complaint IDs, or other relevant details. This feature helps in efficiently locating specific complaints.

f. Admin Operations:

- i. Assign Complaint: Allows admins to allocate complaints to specific agents for resolution.
- ii. View All Complaints: Enables administrators to review and manage all submitted complaints, including their status and assigned agents.

g. Resolve Complaint

Allows an admin to mark a complaint as resolved once the assigned agent has addressed the issue. The complaint status is updated accordingly, and a resolution message is recorded.

8. Authentication

Authorization ensures that users access only the resources or perform actions for which they have

permission. The system enforces this using **Role-Based Access Control (RBAC)**, granting different levels of access based on user roles.

1. Roles

The system supports the following user roles, each with distinct levels of access:

- **Customer:** Can create complaints, view their own complaints, and chat with agents.
- **Agent:** Can view complaints assigned to them and communicate with customers.
- **Admin:** Can access all complaints, assign complaints to agents, view user data, and manage the entire system.

2. Protected Routes

Certain API endpoints are safeguarded to ensure access only by authenticated and authorized users:

- **Admins:** Can access routes to view all complaints, assign complaints, and manage users.
- **Agents:** Can only view complaints assigned to them and interact with customers.

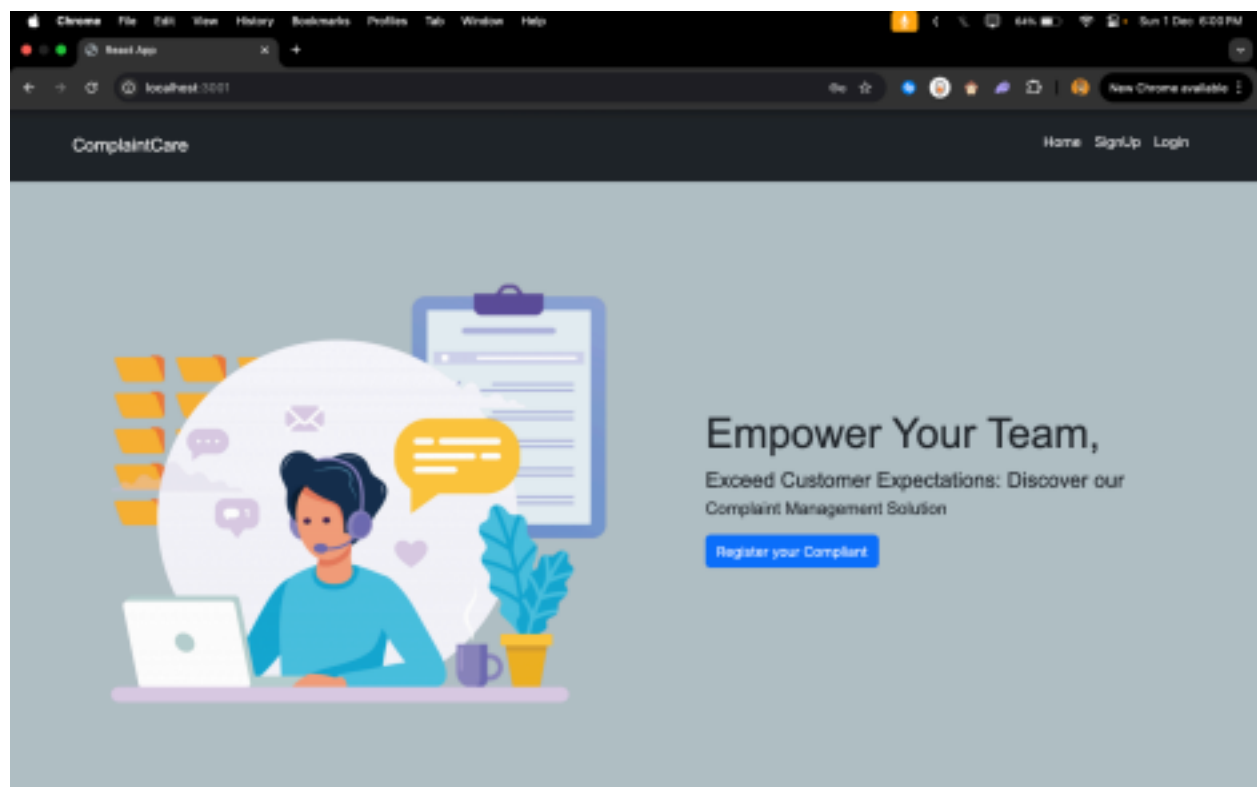
Before granting access to a protected route, the server validates the **JWT token** and checks if the user's role matches the required role for that specific endpoint. If the token is invalid or the role is incorrect, the request is denied.

3. JWT Token Validation

To ensure that only authenticated and authorized users can access protected routes, the system enforces **JWT token validation**. Here's how it works:

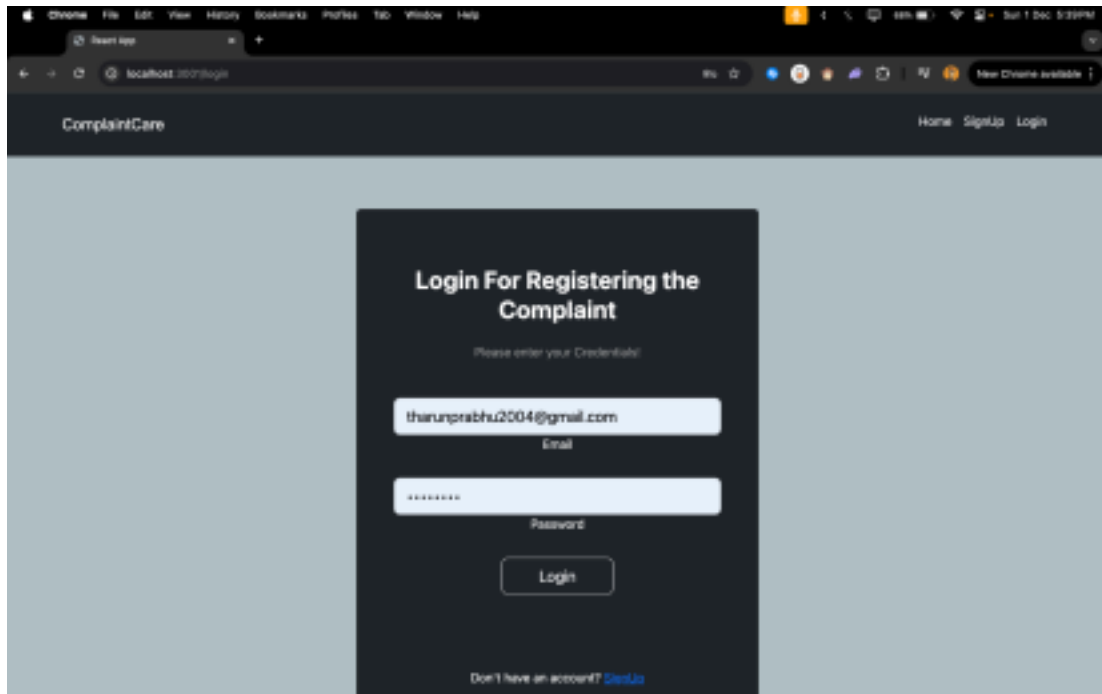
- When a user logs in, the system generates a JWT token, which is sent back to the user. This token is then included in the HTTP header of subsequent API requests under the **Authorization** field.

9. User Interface



The screenshot shows the "SignUp For Registering the Complaint" form. The form is centered on a dark background. It includes the following fields and elements:

- Title:** "SignUp For Registering the Complaint"
- Subtitle:** "Please enter your Details"
- Full Name:** Input field containing "Sachin".
- Email:** Input field containing "sachin.anandharaj@gmail.com".
- Password:** Input field containing "*****".
- Mobile No.:** Input field containing "9150254021".
- Select User Type:** A dropdown menu with "Select User" as the placeholder text.
- Register:** A button at the bottom of the form.



10. Testing

1. Unit Testing

Objective: Validate individual components, functions, or modules in isolation to ensure each part performs as intended.

2. Integration Testing

Objective: Test the interaction and data flow between different modules or services to ensure they work together as expected.

3. End-to-End (E2E) Testing

Objective: Simulate real-world user interactions with the system to ensure the entire application functions seamlessly.

4. Performance Testing

Objective: Evaluate the application's performance under varying conditions to ensure responsiveness and scalability.

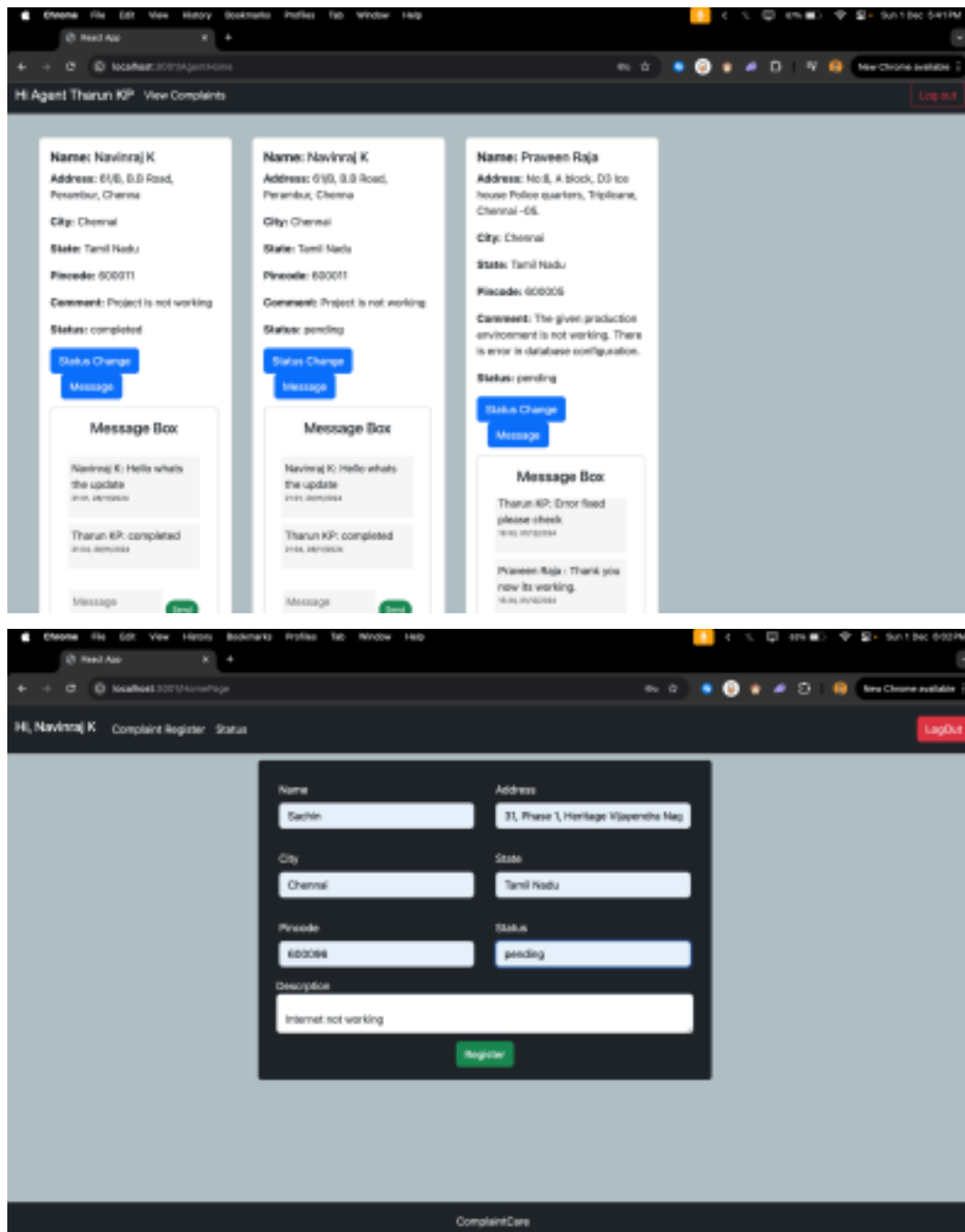
5. Security Testing

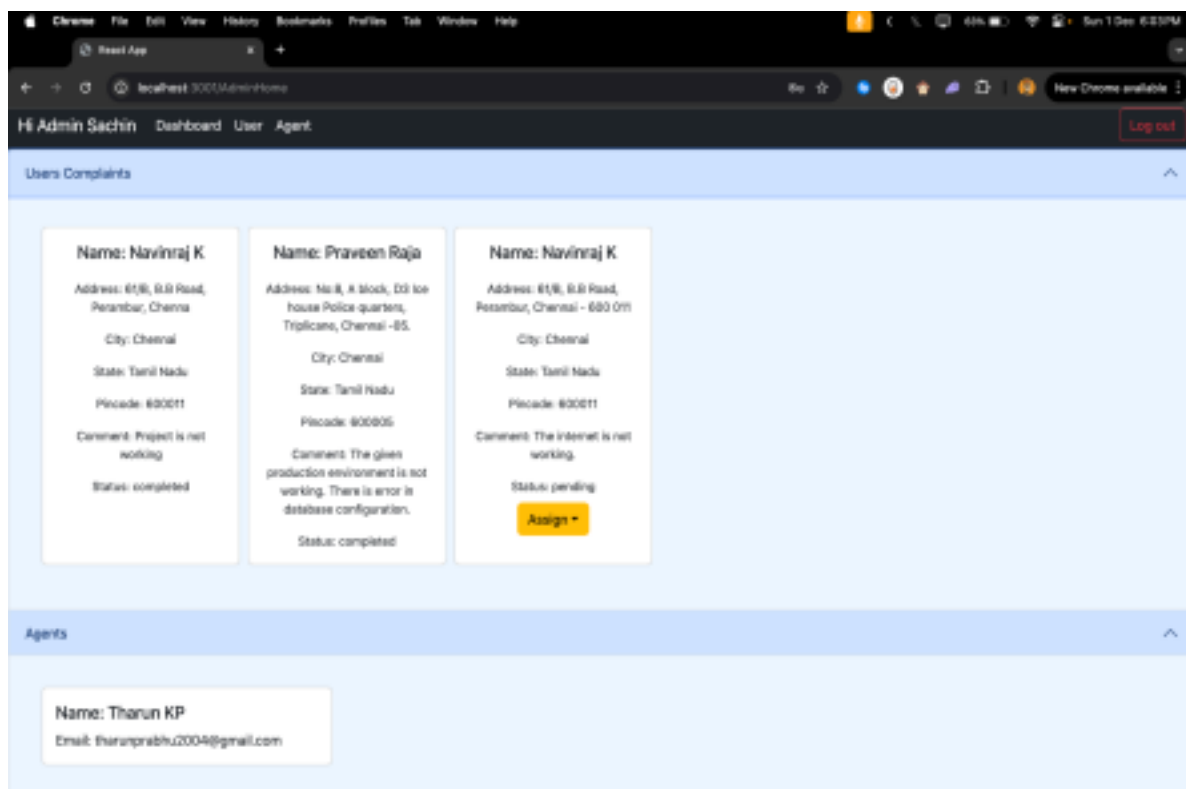
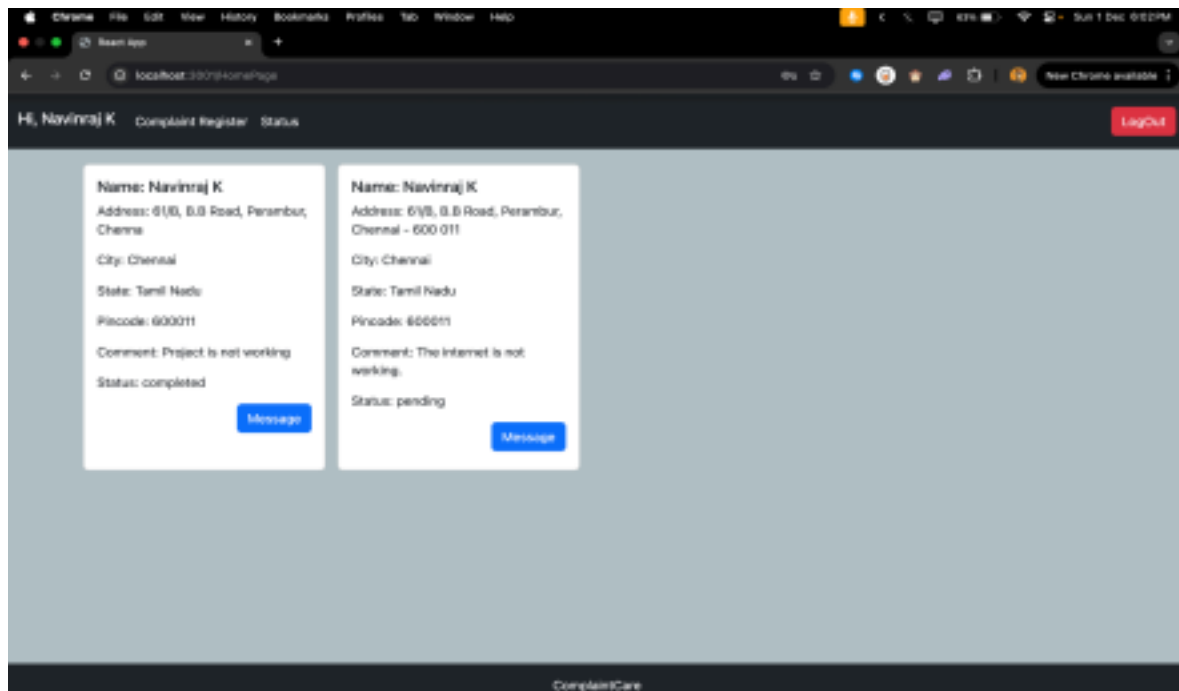
Objective: Identify potential vulnerabilities and ensure the system's data security protocols are effective.

6. Manual Testing

Objective: Perform exploratory testing to identify edge cases, unexpected behaviors, and usability issues.

11.Screenshot or Demo:





12. Known Issues

1. Delayed Real-Time Updates

In certain cases, the chat interface does not update immediately after a new message is sent. This issue may be caused by delays in message synchronization between the client and server. Users may experience a brief lag before seeing the newly sent message appear in the chat window.

Workaround: To see the most recent messages, refresh the chat page manually. A permanent solution is being explored to improve real-time message synchronization.

2. Pagination Not Implemented

Currently, large datasets, such as complaints, are displayed in full without pagination. This can cause the dashboard to become sluggish when browsing through large amounts of data, leading to slower load times and potential performance issues, especially on lower-spec devices or networks.

Impact: Users may experience significant delays in data retrieval when attempting to view or filter large complaint records.

Workaround: Use filters to narrow down the data displayed, minimizing the impact of the unpaginated data. We are working on implementing pagination to enhance performance and improve the user experience when dealing with large datasets.

3. Error Message Clarity

Some error responses generated by the system do not provide clear or user-friendly messages. As a result, end-users may find it difficult to understand the issue or how to resolve it. These vague or technical error messages may lead to confusion, especially for non-technical users.

Examples: Generic error messages like "An error occurred" or "Something went wrong" offer no context or guidance on how to resolve the issue.

Plan: We are in the process of improving error messages to provide more specific, actionable descriptions. This will help users understand the nature of the issue and take the necessary steps to resolve it, reducing confusion and support requests.

4. Slow Search Functionality

The search functionality may be slow or unresponsive when querying large datasets or when searching for terms that are more complex or broad in scope. This can result in delays in retrieving the relevant search results, negatively impacting the user experience.

Workaround: Try refining your search query with more specific terms or filters to speed up the results. A performance optimization for the search feature is currently being developed.

13.Future Enhancements

1. Improved Chat Functionality

To enhance the communication experience, we plan to introduce advanced features for the chat interface. This includes real-time message syncing, multimedia support (e.g., image, video, and document sharing), typing indicators, and read receipts. The goal is to make the chat system more interactive and responsive, ensuring seamless communication between users (customers, agents, and admins).

Future Plans: Implement features such as chat history search, chat archiving, and improved notification settings for better user control over conversations.

2. Advanced Analytics for Admin

Currently, the admin panel offers basic management features, but we plan to expand its capabilities by adding advanced analytics tools. These tools will allow admins to view detailed reports and insights on system usage, complaint trends, user activity, and agent performance.

Key Features:

- Real-time dashboards to track complaint status and agent workload.
- Data-driven insights on customer satisfaction and complaint resolution efficiency.
- Customizable reporting tools to filter and analyze data based on various parameters.

3. Mobile App Integration

To enhance accessibility and user experience, we are working towards developing a mobile application version of the complaint registration and management system. The app will allow users to access all the key features of the platform on their mobile devices, improving the convenience of submitting and managing complaints while on the go.

Key Features:

- Push notifications for real-time updates.
- Mobile-optimized user interface for smooth navigation.
- Offline functionality, allowing users to view complaints and submit them once they are back online.

4. Enhanced User Notifications

We plan to introduce a more robust notification system that will keep users informed about updates related to their complaints, messages, or system activities. This will include customizable notifications via email, SMS, and in-app alerts.

Future Features:

- Customizable alert settings, allowing users to choose when and how they are notified.
- Enhanced notification system to update users on complaint status changes (e.g., when an agent is assigned or when a complaint is resolved).