**Online Complaint Registration and management System with MERN**

**1. Introduction**

* **Project Title:** [Online Complaint Registration And Management System]
* **Team Members:** **Priyangha - Project Manager & Backend Developer**
* Responsible for overseeing the project development, coordinating with team members, and ensuring project milestones are met.
* Leads backend development, including setting up the server, creating API endpoints, and integrating with MongoDB.
* **Varsha shree - Frontend Developer**
* Handles the development of the React frontend, including creating UI components, implementing user flows, and ensuring responsiveness.
* Collaborates with the backend team to integrate API endpoints and data fetching.
* **Vigneshwari - UI/UX Designer**
* Focuses on designing an intuitive and user-friendly interface for the application.
* Responsible for creating wireframes, mockups, and implementing design elements using tools like Tailwind CSS for a polished user experience.
* **Saktheesh - Quality Assurance & Tester**
* Ensures the application meets quality standards by creating and executing test cases for both frontend and backend.
* Responsible for identifying bugs, documenting issues, and conducting final testing before deployment.

**2. Project Overview**

* **Purpose:** The purpose of the *Online Complaint Registration and Management System* (Customer Care Registry) is to provide a streamlined, user-friendly platform where customers can register complaints, and customer support teams (agents and admins) can manage and resolve those complaints efficiently. This system is designed to facilitate better communication between customers and support teams, enhance accountability, and improve overall customer satisfaction. Key goals include:
* **Enhanced Customer Experience**: Customers can easily lodge complaints and track their resolution progress.
* **Efficient Complaint Management**: The system ensures complaints are directed to the right personnel for timely action.
* **Transparency and Accountability**: Both customers and agents can track complaint status, creating transparency in the process.
* **Data-Driven Insights**: Administrators can analyze complaints data to improve service quality and identify recurring issues.
* **Features:** The system includes the following features to meet its purpose effectively:
* **1. Customer/Ordinary User Features**
* **Complaint Registration**: Users can log complaints through a straightforward form, providing essential details like issue description, type of complaint, and urgency.
* **Complaint Tracking**: Users can view the status of their complaints (e.g., pending, in-progress, resolved) and receive updates on progress.
* **User Dashboard**: A personalized dashboard displays complaint history, status summaries, and resolution timelines.
* **2. Agent Features**
* **Complaint Assignment and Handling**: Agents receive complaints in their queue, organized by priority and type, and can update the complaint status (e.g., in-progress, resolved).
* **Real-Time Notifications**: Agents receive notifications for new complaint assignments, feedback from users, and changes in complaint priority.
* **3. Admin Features**
* **Role and Access Management**: Admins can manage user roles, assigning permissions for customers, agents, and other admins.
* **4. Additional General Features**
* **User Authentication and Security**: Secure login for all users, ensuring data privacy and preventing unauthorized access.
* **Responsive Design**: The system provides an optimized experience on different devices, ensuring ease of use for all users.
* **Detailed Reporting**: The system generates periodic reports for performance insights and helps in identifying frequent complaints or bottlenecks in the process.
* **3. Architecture**
* **Frontend:** React handles the user interface, manages state, and communicates with the backend via API calls.
* **Backend:** Node.js and Express.js handle API routing, authentication, authorization, and business logic.
* **Database:** MongoDB stores application data, with Mongoose providing structured interaction between backend and database.
* **4. Setup Instructions**
* **Prerequisites:** Before setting up the project, make sure you have the following software dependencies installed:
* **Node.js** (v14 or higher)
* Required for running the backend server (Node and Express).
* Download from [Node.js official website](https://nodejs.org/).
* **npm** (Node Package Manager)
* Comes with Node.js and is required to install project dependencies.
* To verify, run: node -v and npm -v in your terminal.
* **MongoDB** (Community Edition or MongoDB Atlas for cloud)
* Used for database management to store complaints, user data, and more.
* For local setup, download from [MongoDB official website](https://www.mongodb.com/try/download/community).
* Alternatively, set up a MongoDB Atlas account for a cloud database [here](https://www.mongodb.com/cloud/atlas).
* **Git**
* Required to clone the project repository from a version control platform (e.g., GitHub).
* Download from [Git official website](https://git-scm.com/).
* **Installation:** Follow these steps to set up the project locally:
* **Clone the Project Repository**
* Open your terminal and navigate to the directory where you want to clone the project. Then run:
* git clone < https://github.com/Saktheesh15/Online-Complaint-Registration-And-Management-System.git >
* **Navigate to Project Directory**
* After cloning, navigate to the project directory:
* cd Online-Complaint-Registration-and-Management-System
* **Install Dependencies**
* There are dependencies for both frontend and backend, so follow these steps:
* **Frontend**: Navigate to the frontend folder and install dependencies.
* cd frontend
* npm install
* **Backend**: Open a new terminal, navigate to the backend folder, and install dependencies.
* cd ../backend
* npm install
* **Set Up Environment Variables**
* Create .env files in both the *frontend* and *backend* folders to configure environment-specific variables. Here’s how:
* **Backend Environment Variables**: In the backend folder, create a .env file and add the following variables:
* PORT=3000 # Port for backend server

MONGO\_URI=< mongodb+srv://Saktheesh:Sakthee123@cluster0.qdjlr.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0> # MongoDB URI for database connection

* JWT\_SECRET=<your\_jwt\_secret \_key> # Secret key for token authentication
* **Frontend Environment Variables**: In the frontend folder, create a .env file to define the API endpoint:
* REACT\_APP\_API\_URL=http://localhost:3000/api # Base URL for backend API
* **Running the Project**
* Once the setup is complete, start the project by running the frontend and backend servers.
* **Run the Backend Server**:
* In the backend folder, run:
* npm start
* The backend server should start on http://localhost:3000.
* **Run the Frontend Server**:
* Open a new terminal, navigate to the frontend folder, and run:
* npm start
* The frontend server should start on http://localhost:3001 or another available port.

**5. Folder Structure**

**Frontend: React with TypeScript and CSS**

* frontend/
* │
* ├── public/
* │ ├── favicon.ico # Favicon for the application
* │ ├── index.html # Main HTML file, root for the React app
* │ └── manifest.json # Web app manifest for PWA support
* │
* ├── src/
* │ ├── components/ # Reusable UI components in TypeScript
* │ │ ├── AuthContent.tsx # Component for authentication-related content
* │ │ ├── ComplaintForm.tsx # Form component for submitting complaints
* │ │ ├── ComplaintList.tsx # Component to list complaints
* │ │ ├── LoginForm.tsx # Login form component
* │ │ └── SignupForm.tsx # Signup form component
* │ │
* │ ├── contexts/ # Context API setup for global state management
* │ │ └── AuthContext.tsx # Context for managing user authentication
* │ │
* │ ├── App.tsx # Root component with routing and layout
* │ ├── index.tsx # Entry point for rendering the React app
* │ ├── index.css # Global styling, integrates Tailwind CSS
* │ ├── react-app-env.d.ts # React environment type definitions
* │
* ├── tailwind.config.js # Tailwind CSS configuration file
* ├── tsconfig.json # TypeScript configuration for frontend
* ├── package-lock.json # Lock file for frontend dependencies
* └── package.json # Lists frontend dependencies and scripts
* **src/components/**: Contains TypeScript components such as AuthContent.tsx, ComplaintForm.tsx, and others, which handle UI rendering for various sections.
* **src/contexts/**: Houses AuthContext.tsx for managing authentication state across the application.
* **tailwind.config.js**: Configuration file for Tailwind CSS, where custom theme settings and plugins are defined.
* **tsconfig.json**: TypeScript configuration file to define rules for type-checking, paths, and compilation settings.

**Backend: Node.js and Express**

**backend/**

**│**

**├── .env # Environment variables for backend configuration**

**├── package-lock.json # Package lock file for exact dependency versions**

**├── package.json # Lists backend dependencies and scripts**

**└── server.js # Main backend server file, configures Express app**

* **server.js: Entry point for the backend server. It configures Express, connects to MongoDB, and defines middleware.**
* **.env: Stores sensitive information like database URIs and JWT secrets.**

**6. Running the Application**

* Commands to start the frontend and backend servers locally.
  + **Frontend: Open a new terminal** and navigate to the frontend directory:
  + cd frontend
  + **Start the frontend server**:
  + npm start
  + This command will start the React application on http://localhost:3001 (or another available port).
  + The frontend server communicates with the backend API using the REACT\_APP\_API\_URL set in the frontend .env file.
  + **Backend:** **Starting the Backend Server**
  + **Navigate to the backend directory**:
  + cd backend
  + **Start the backend server**:
  + npm start
  + This command will start the backend server on the port specified in your .env file (e.g., PORT=3000).
  + Ensure that your MongoDB server is running

**7. API Documentation**

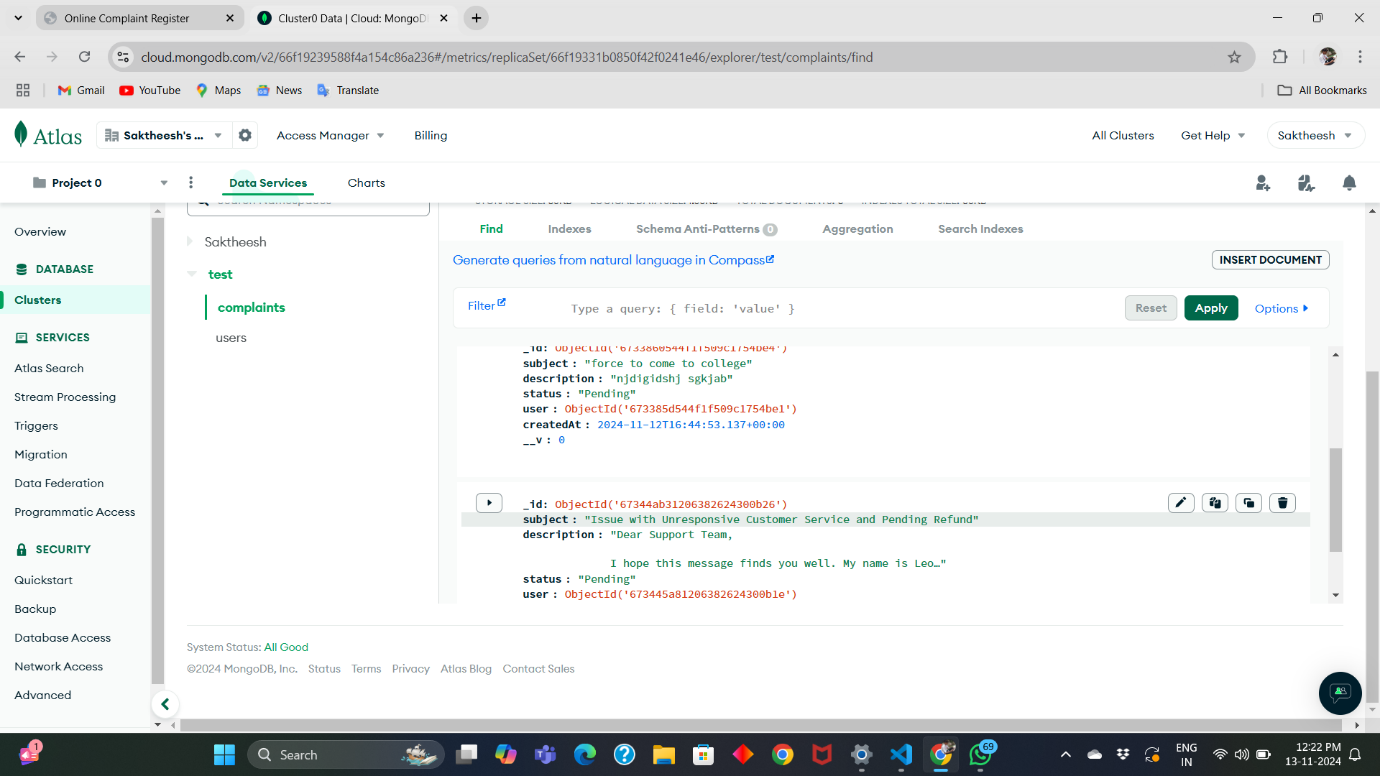
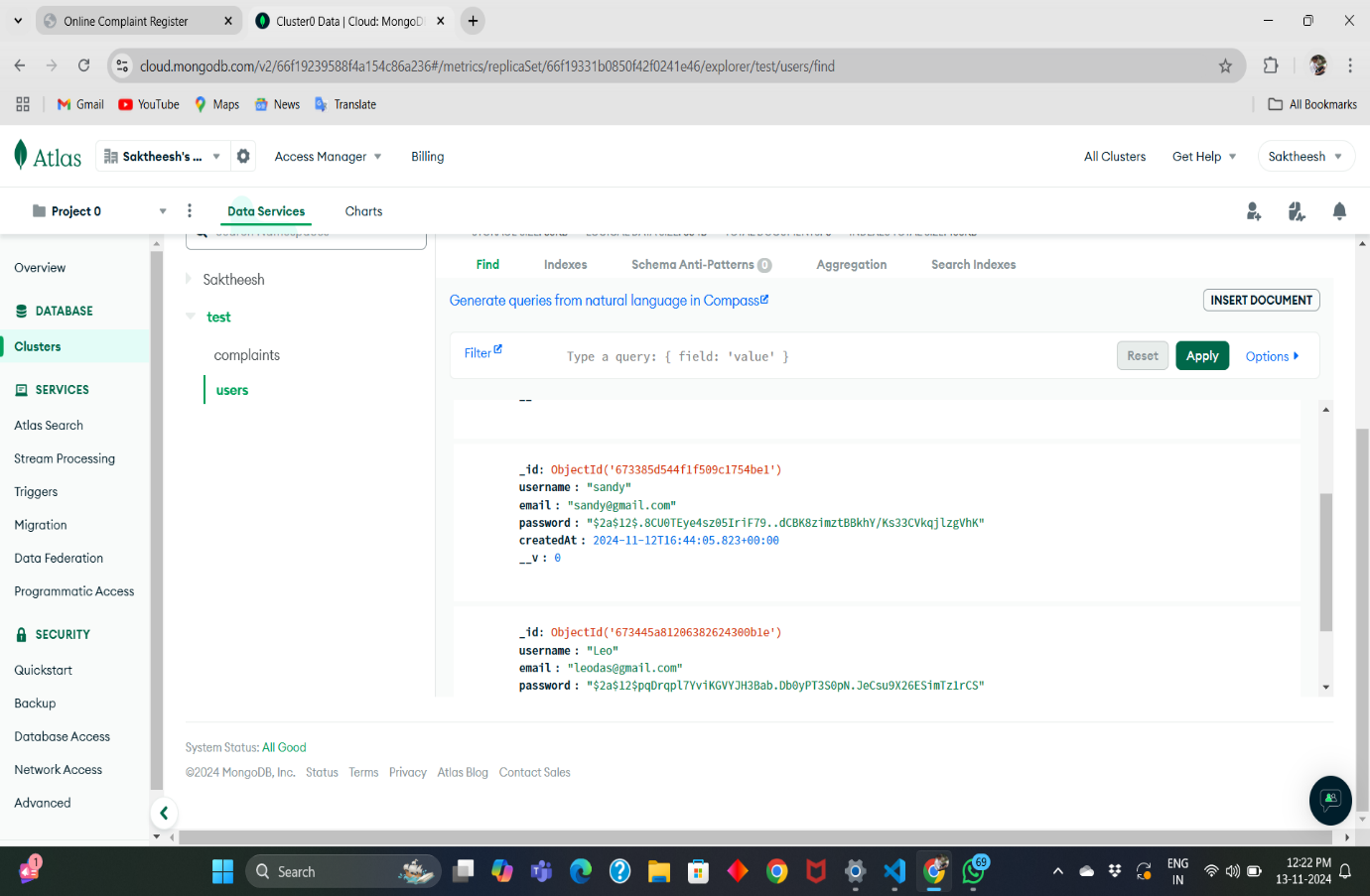
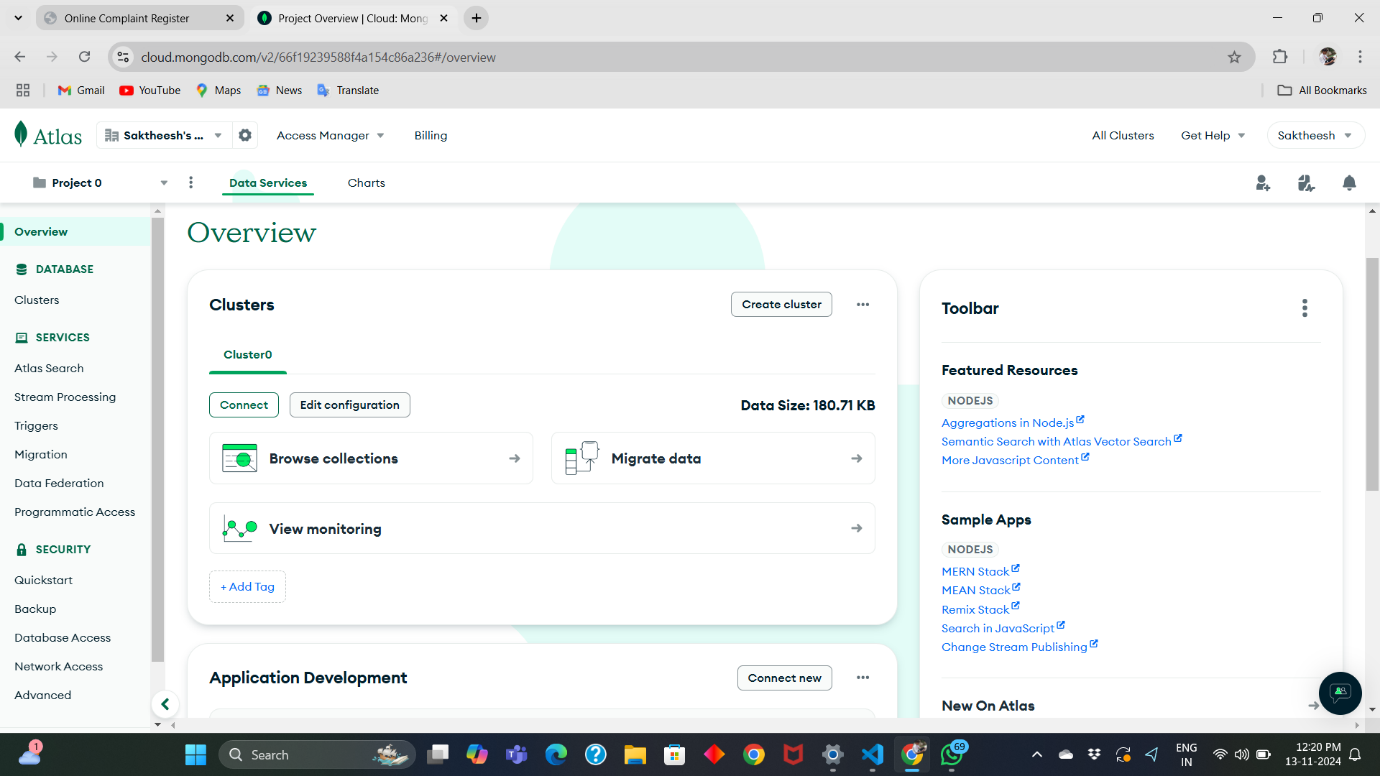
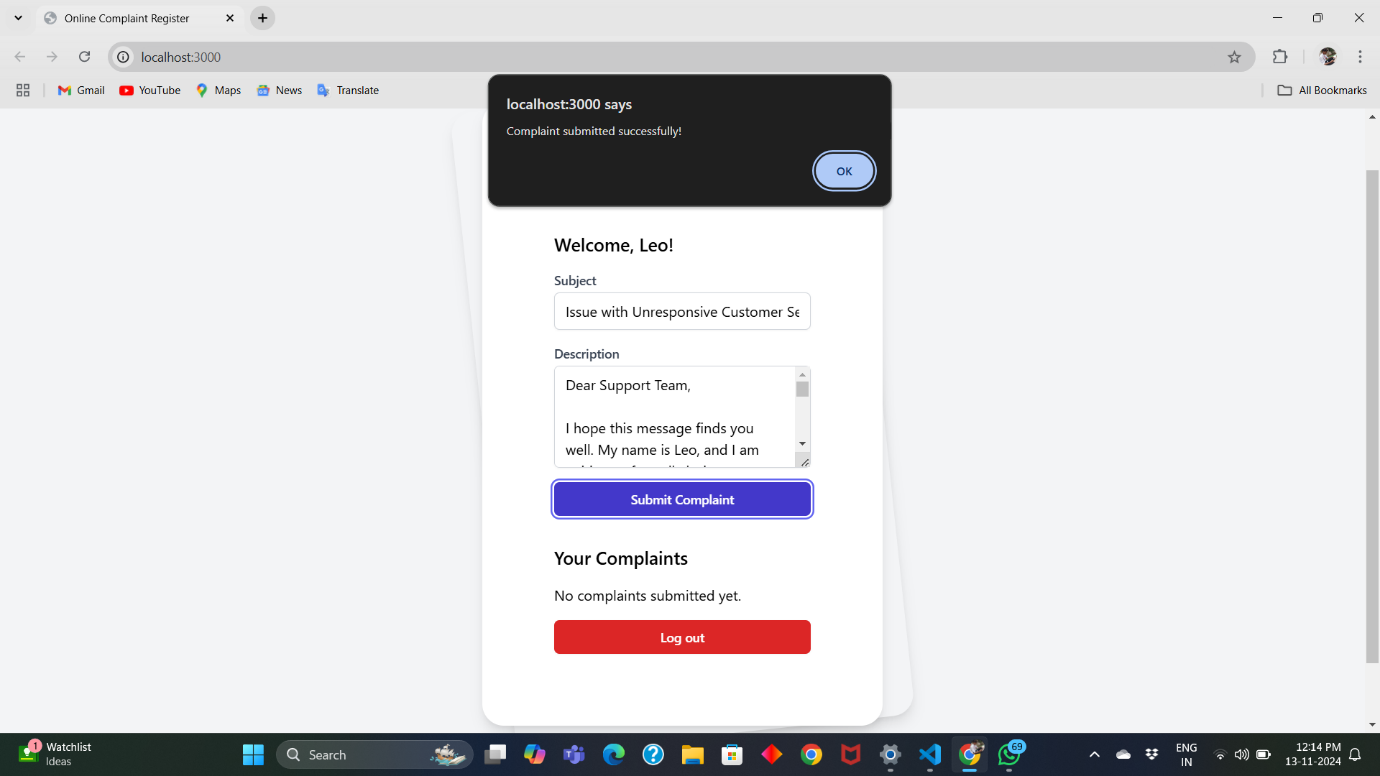
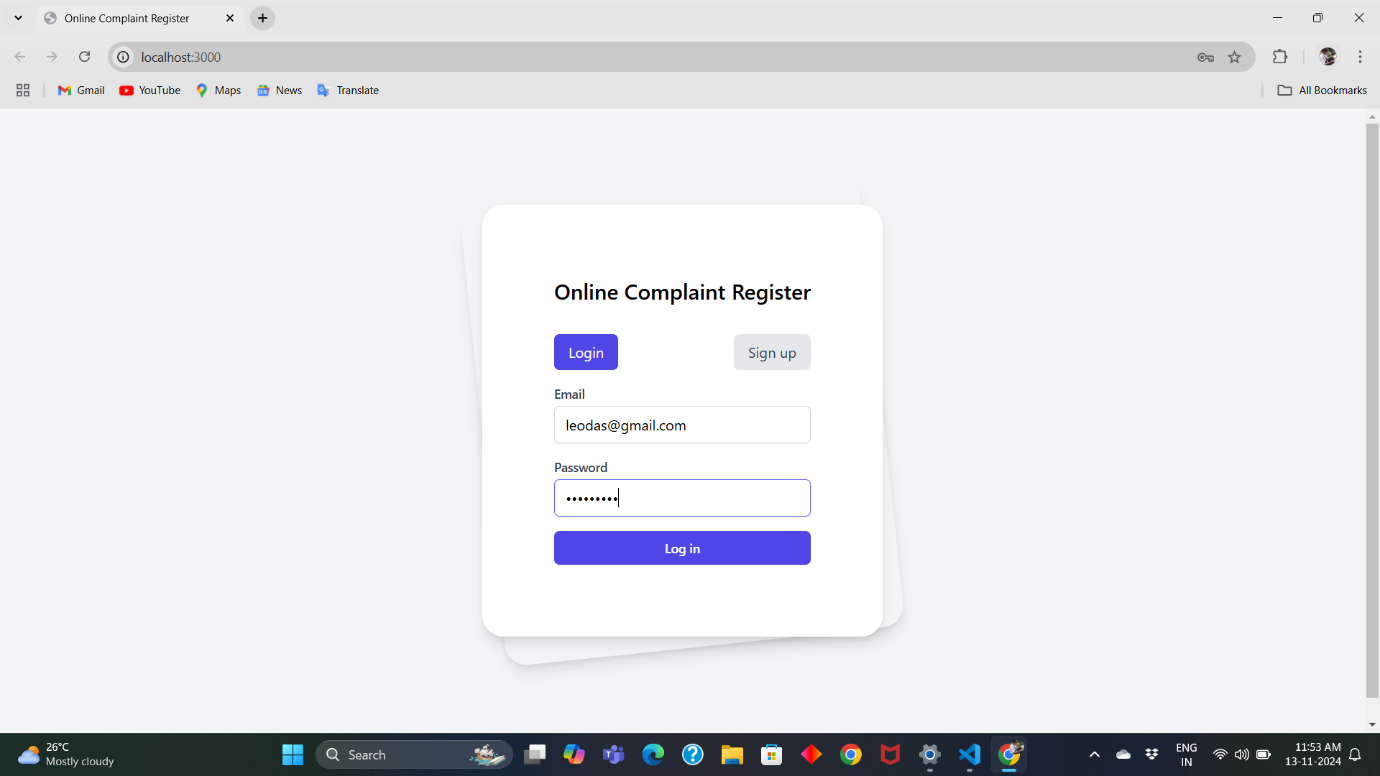
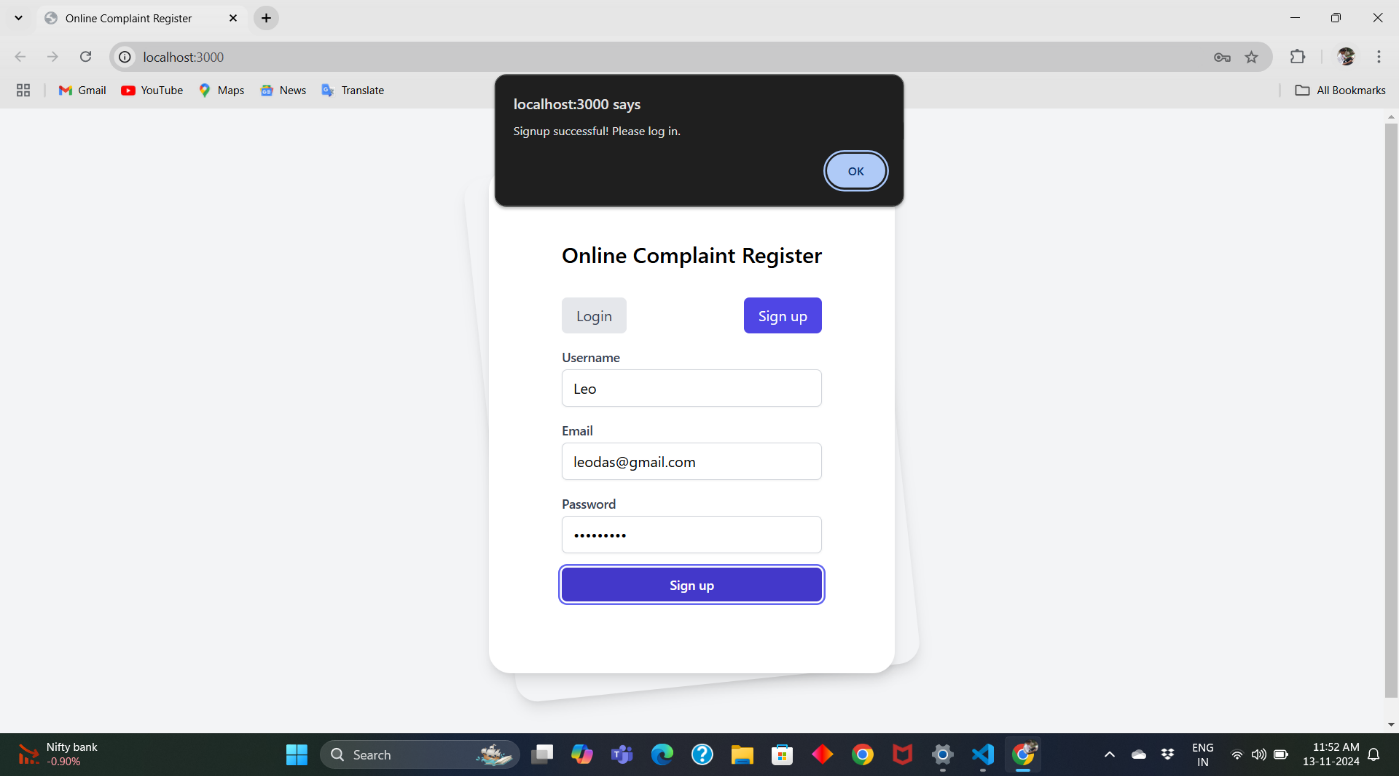
* **Authentication**: POST /api/auth/register, POST /api/auth/login for user registration and login.
* **Complaints**: CRUD operations on complaints, including submitting, retrieving, updating, and listing.
* **User Profiles**: Fetch and update user profile information.Include request methods, parameters, and example responses.

**8. Authentication**

* **Authentication**: Handled using JWTs, where each authenticated user receives a token after logging in, which they include in the headers of subsequent requests.
* **Authorization**: Controlled by role-based access, enforced through middleware to restrict access to certain routes and actions based on the user's role.

This combination of JWT-based authentication and role-based authorization ensures that only verified users have access to the application’s resources, and their permissions are aligned with their roles. This setup improves security and keeps sensitive data protected within the application.

**9. User Interface Screenshots or Demo**

* Screenshots or GIFs showcasing different UI features.

**10. Testing**

**Unit Testing**: Using Jest and React Testing Library to test individual functions and components in isolation.

**Integration Testing**: Using Jest, React Testing Library, and Supertest to test interactions between components and backend APIs.

**End-to-End Testing**: Using Cypress to simulate real user workflows and verify that the application functions correctly from start to finish

**11. Known Issues**

Session Expiry Handling

Slow Loading on Initial Complaint List Fetch

Form Validation Messages Not Clearing

Incomplete Mobile Responsiveness

Error Handling for Database Connection Issues

Delayed Status Update Notifications

Limited Role-Based Authorization on Frontend

These known issues don’t prevent the application from functioning but may affect user experience or efficiency. Developers are encouraged to address these bugs in future updates to improve usability and performance. Users should be aware of these limitations while using the system.

**12. Future Enhancements**

Real-Time Notifications

Enhanced Mobile Responsiveness

Dashboard with Analytics and Insights

Automated Email Notifications

Audit Log for Admins

These potential future features and improvements could significantly enhance the functionality, user experience, and scalability of the *Online Complaint Registration and Management System.*