**Full Stack Development with MERN**

**1. Introduction**

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

Divyanshi Mittal(Leader) – Frontend developer

V Sree Teja Pradeep – Backend developer

Manas Kamal Das – Frontend developer

Shreya Mahakur – Frontend developer

**2. Project Overview**

**Purpose:** The Online Complaint Registration and Management System is designed to provide a streamlined and efficient way for users to register complaints and for administrators and agents to manage and resolve these complaints. The primary goal of the project is to enhance user satisfaction by ensuring timely and organized handling of complaints while reducing the administrative burden on support staff.

**Goals:**

* Enable users to easily register and track complaints.
* Provide administrators and agents with tools to efficiently manage and resolve complaints.
* Ensure transparency and accountability in the complaint handling process.
* Improve overall user experience and satisfaction.

**Features:**

1. **User Authentication**
   1. Signup: Allows new users to create an account.
   2. Signin: Allows existing users to log in to their account.
2. **User Profile Management**
   1. Get Profile: Enables users to view their profile information.
3. **Complaint Registration and Management**
   1. Register Complaint: Users can submit new complaints specifying issue details.
   2. View User Complaints: Users can view the status and details of **their** submitted complaints.
   3. View All Complaints: Administrators can access a comprehensive list of all complaints.
   4. View Pending Complaints: Administrators can filter and view only the complaints that are pending**.**
4. **Agent Management**
   1. Get Agents: Administrators can view a list of all available agents.
   2. Assign Agent: Administrators can assign agents to specific complaints.
   3. View Agent Complaints: Agents can view all complaints assigned to them.
5. **Complaint Status Management**
   1. Update Complaint Status: Administrators and agents can update the status of complaints to reflect their current state.
6. **User-Friendly Interface**
   1. Intuitive and responsive user interface for easy navigation and operation.
   2. Dashboard for administrators to get an overview of the complaint status and workload.

**3. Architecture**

**Frontend Architecture using React:**

* **Component-based Structure**: The frontend is structured using React, where the user interface is broken down into reusable components.
  + **Pages**: High-level components corresponding to different views or routes in the application (e.g., LoginPage, SignupPage, UserDashboard, AdminDashboard).
  + **Components**: Smaller, reusable components that are used within pages (e.g., Navbar, Footer, ComplaintForm, ComplaintList).
* **Routing**: Implemented using react-router-dom to handle navigation between different pages and components.
* **API Integration**: Communication with the backend is handled through a centralized apiService file where all the API requests are defined.
* **Styling**: CSS modules or styled-components are used to apply scoped styles to components, ensuring maintainability and modularity.
* **Responsiveness**: Design is made responsive using CSS Grid, Flexbox, and vh units for consistent sizing across different screen sizes.

**Backend Architecture using Node.js and Express.js:**

* **Express Server**: The backend server is built using Express.js, a minimal and flexible Node.js web application framework.
  + **Routing**: Organized using Express routers to manage different sets of endpoints (e.g., userRoutes, complaintRoutes, agentRoutes).
  + **Middleware**: Middleware functions are used for handling errors
* **Controllers**: Business logic and request handling are separated into controller files.
  + **AdminController**: Handles admin-related operations.
  + **AgentController**: Manages agent-related operations.
  + **UserComplaintController**: Manages user complaint-related operations.
  + **UserController**: Handles user-related operations.
* **Models**: Mongoose schemas define the structure of the data stored in MongoDB.
  + **User.js**: Schema for user details.
  + **UserComplaint.js**: Schema for user complaints.
* **Utils**: Utility functions such as password hashing and token generation.
  + **bcryptUtils.js**: Contains functions for hashing and comparing passwords.
* **Environment Configuration**: .env file is used to store environment variables such as database connection strings and secret keys.

**Database Schema and Interactions with MongoDB:**

* **MongoDB**: A NoSQL database is used for storing application data.
  + **Mongoose**: An Object Data Modeling (ODM) library for MongoDB and Node.js, providing a schema-based solution to model application data.

**Schemas:**

1. **User Schema** (User.js):
   * **Fields**:

* {
* \_id:{type:String, required:true},
* name: { type: String, required: true },
* email: { type: String, required: true },
* address: { type: String, required: true },
* mobileNumber: { type: Number, required: true },
* role:{type: String, required:true},
* username: { type: String, required: true },
* password: { type: String, required: true },
* }
  + **Methods**: Password hashing before saving a user, and password comparison for authentication.

1. **User Complaint Schema** (userComplaint.js):
   * **Fields**:

* {
* ticketId:{type:String,required:true},
* name:{type:String, required:true},
* phoneNum:{type:Number,required:true},
* email:{type:String,required:true},
* complaintTitle:{type:String,required:true},
* complaintDescription:{type:String,required:true},
* status:{type:String,required:true},
* userId:{type:String,required:true},
* agentId:{type:String,required:false}
* }
  + **Methods**: Methods to update complaint status and assign agents.

**Interactions**:

* **User Registration**: When a user signs up, their details are saved in the users collection.
* **User Login**: Upon login, user credentials are verified, and a JWT token is generated and returned.
* **Complaint Registration**: User complaints are saved in the userComplaints collection.
* **Retrieving Complaints**: Various endpoints allow for fetching complaints based on user ID, agent ID, and complaint status.
* **Assigning Agents**: Admins can assign agents to complaints, and this information is updated in the userComplaints collection.
* **Updating Status**: Agents and admins can update the status of complaints, which is then reflected in the database.

**4. Setup Instructions**

* **Prerequisites(dependencies):** Node Js, React, MongoDB, mongoose, MUI, Axios, bcrypt, body parser, dotenv, express, uuid
* **Installation:** clone the repository, open the app.js file in both front end and back end projects and type ‘npm install’ to install the dependencies in the terminal of the frontend and the backend directory, create an env file mentioning the local mongodb url with the variable "MONGO\_URI" and port number to start your backend server as "PORT"and a variable called salt\_rounds with the value ranging from 10-12.

**5. Folder Structure**

**Client:**

**node\_modules/**: This directory contains all the dependencies and modules installed via npm.

**public/**: This directory typically contains the static assets of our project like the index.html file.

**src/**: This is the source directory where all our React code resides.

* **api/**: This directory contains the file related to API calls and services.
* **components/**: This directory is intended for storing reusable React components.
* **pages/**: This directory is intended for storing different pages of our application, separating them from reusable components.
* **App.css**: A CSS file for styling the App component.
* **App.js**: The main component file for your application, the root component.
* **App.test.js**: A file for testing the App component.
* **index.css**: A CSS file for global styles.
* **index.js**: The entry point of the React application. This file is responsible for rendering the App component into the DOM.
* **logo.svg**: An SVG file for the logo, typically used in the App component.
* **reportWebVitals.js**: A file for measuring the performance of your app.**setupTests.js**: A file for setting up testing configurations.

**Server:**

* **src/config/database.js**:

Manages database configuration and connection setup.

* **src/controllers/adminController.js**:

Handles admin-related API requests and business logic.

* **src/controllers/agentController.js**:

Manages agent-related API requests and business logic.

* **src/controllers/userComplaintController.js**:

Processes user complaint-related API requests and business logic.

* **src/controllers/userController.js**:

Handles user-related API requests and business logic.

* **src/middlewares/errorHandler.js**:

Contains middleware for handling errors across the application.

* **src/models/User.js**:

Defines the User model schema and interactions with the database.

* **src/models/userComplaint.js**:

Defines the UserComplaint model schema and interactions with the database.

* **src/routes/userRoutes.js**:

Sets up and manages user-related API routes.

* **src/utils/bcryptUtils.js**:

Provides utility functions for password hashing and validation using bcrypt.

* **src/app.js**:

Main entry point of the application, sets up the server and integrates middleware.

* **.env**:

Stores environment variables.

* **.gitignore**:

Specifies files and directories to be ignored by Git.

* **package-lock.json**:

Manages the exact versions of installed Node.js dependencies.

* **package.json**:

Contains metadata about the project and lists dependencies and scripts.

**6. Running the Application**

* Provide commands to start the frontend and backend servers locally.
  + **Frontend:** npm start in the frontend directory.
  + **Backend:** npm start in the backend directory.

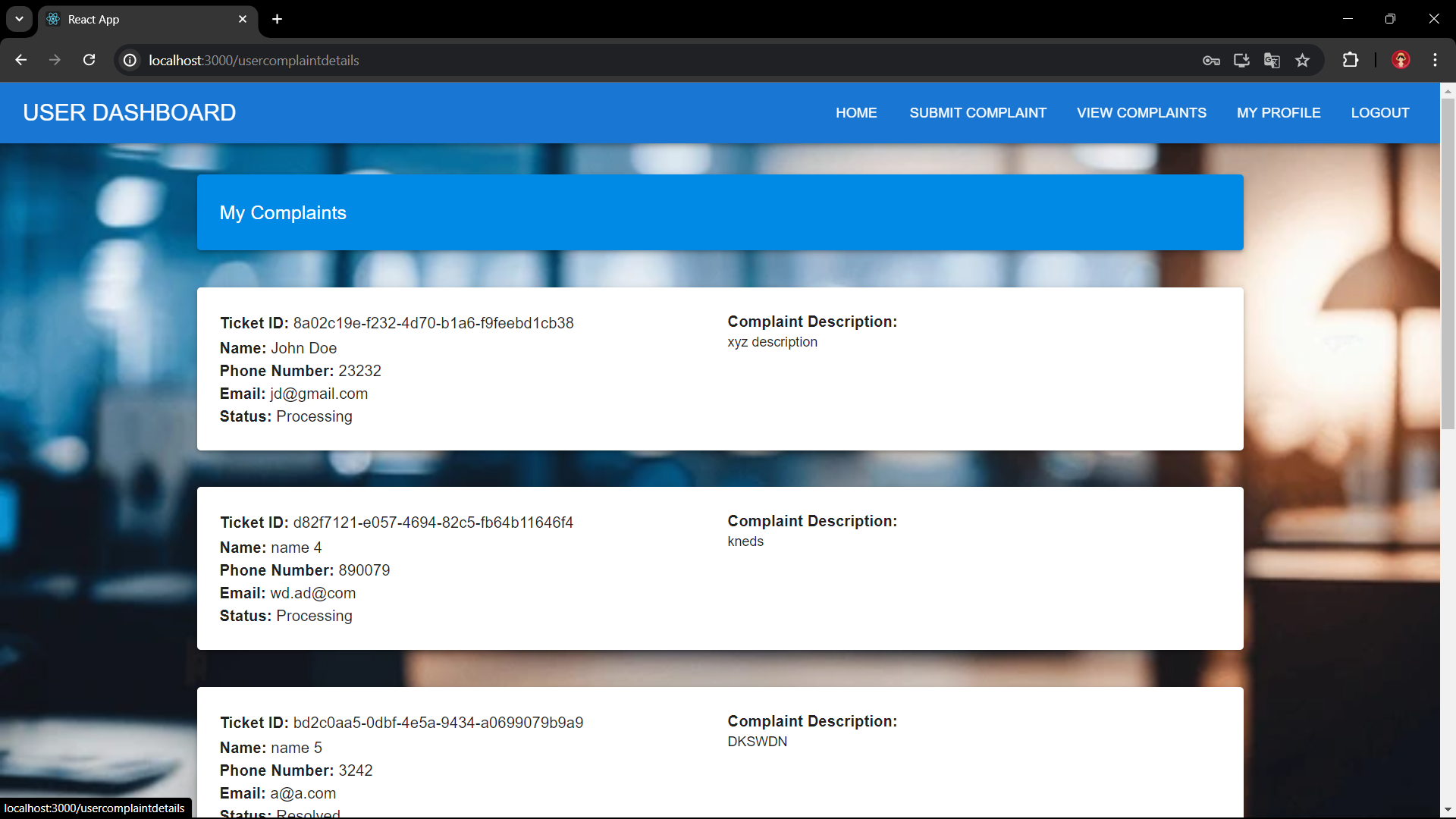
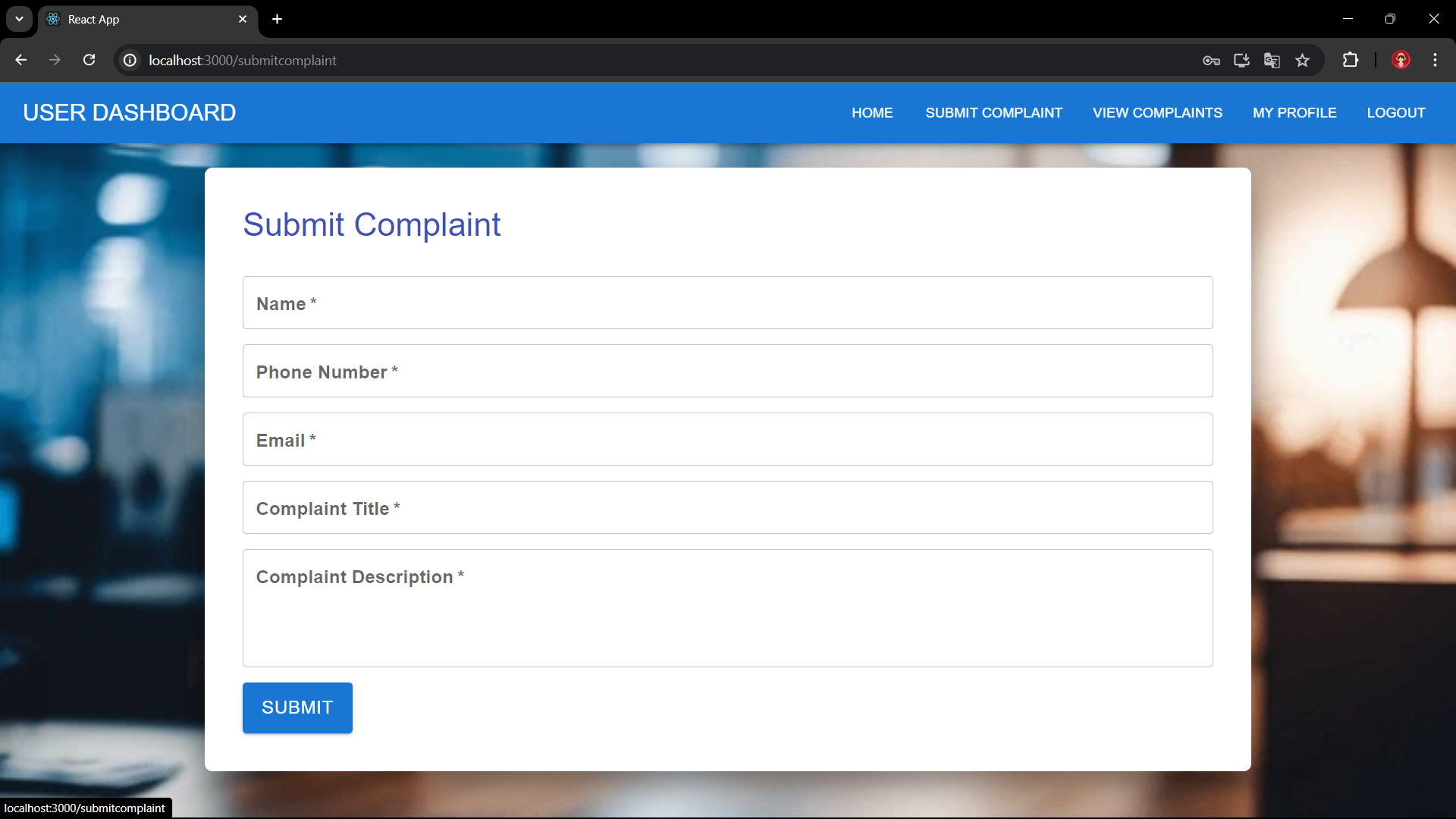
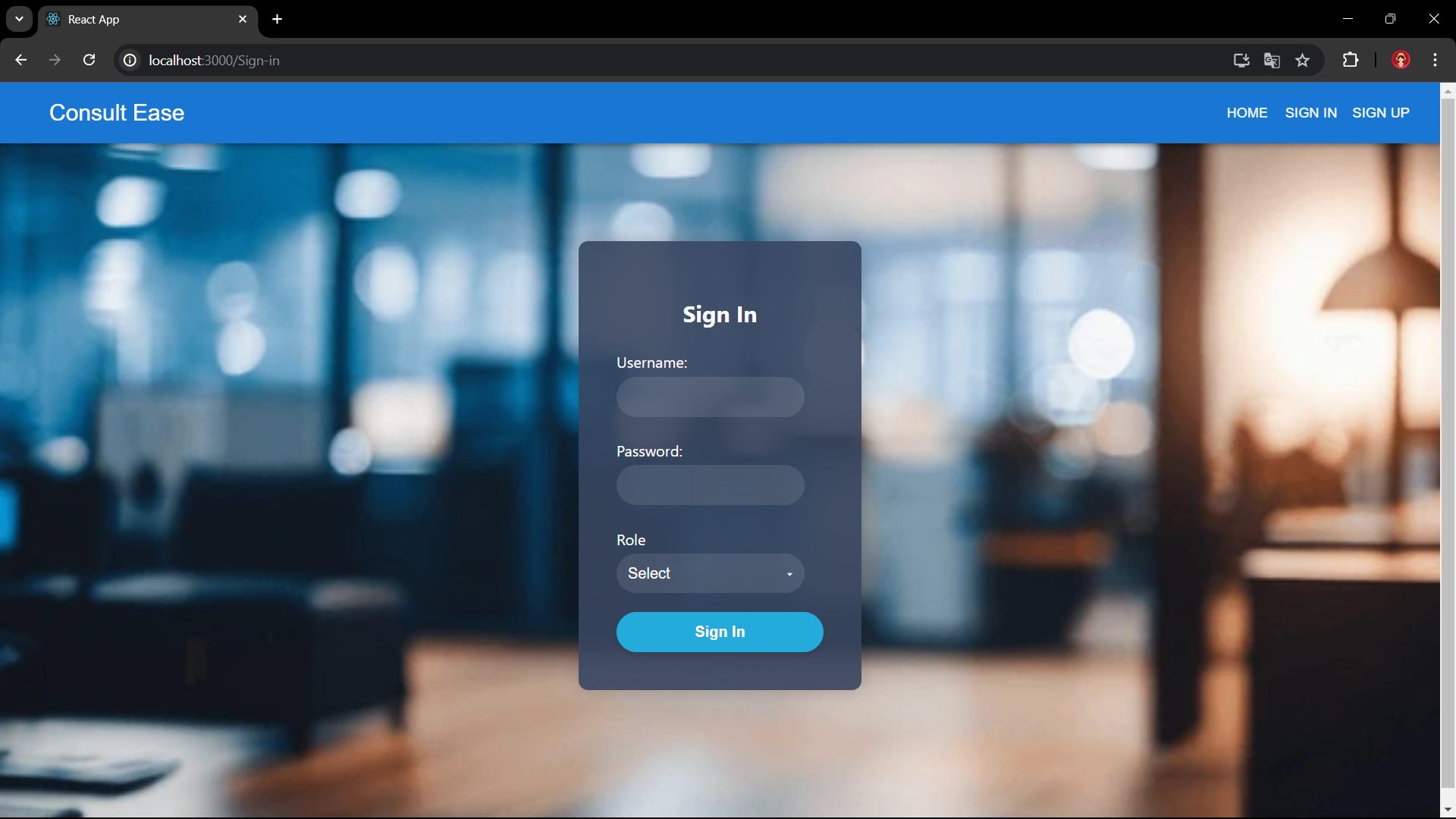
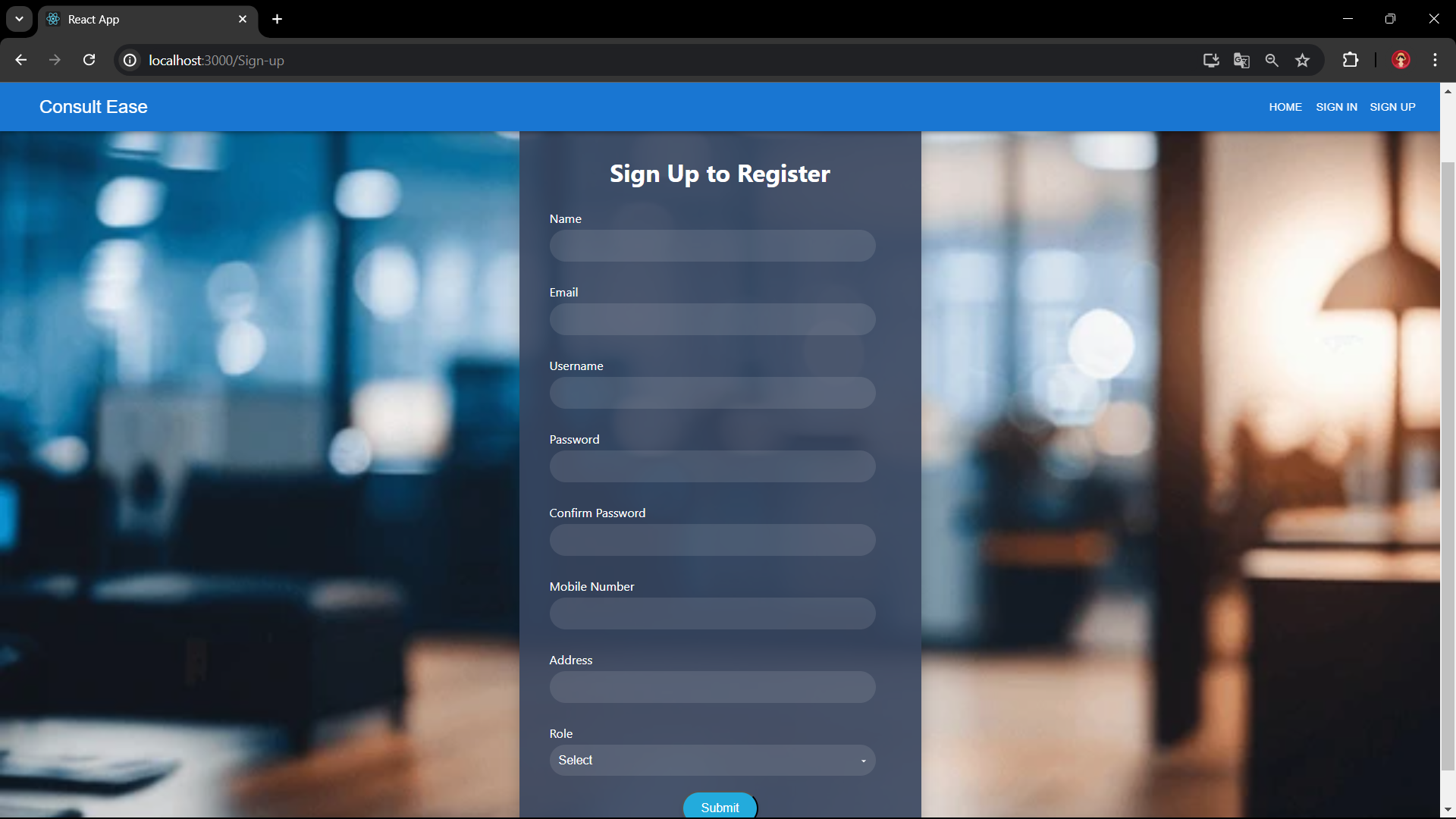
1. **API Documentation**

* A summary of the main API endpoints and their purposes:
* **User Authentication**
* **- POST /api/signup -** Registers a new user.
* **- POST /api/signin -** Authenticates a user and returns a token.
* **User Profile**
* **- GET /api/getprofile/:userId -** Retrieves user profile information by user ID.
* **User Complaints**
* **- POST /api/complaintReg -** Registers a new user complaint.
* **- GET /api/getusercomplaints/:userId** - Retrieves complaints made by a specific user.
* **- GET /api/getallcomplaints-** Retrieves all user complaints.
* **- GET /api/pendingcomplaints -** Retrieves all pending complaints.
* **Agent Management**
* **- GET /api/getagents -** Retrieves all agents.
* **- POST /api/assignagent -** Assigns an agent to a complaint.
* **- GET /api/viewagentcomplaints/:agentId -** Retrieves complaints assigned to a specific agent.
* **- POST /api/updatecomplaintstatus -** Updates the status of a complaint. Integration with Frontend

**8. Authentication**

We are using bcrypt module to compare the stored hashed password and the entered password and authenticating the users

**9. User Interface**



**10. Testing**

Tested all the possible edge cases, tested backend api using postman service, and tested frontend responsiveness using chrome browser.

**11. Screenshots or Demo**

**Link for the demo video:** [CLICK HERE TO SEE THE DEMO VIDEO](https://drive.google.com/drive/folders/1WpYrDhX8WPAzC6RvRC1C1uUV819IpQGT)

**12. Known Issues**

**-**A minor issue, with the responsiveness of home page.

**13. Future Enhancements**

Could implement a feature for the user and agent to chat using socket.io .