**ONLINE COMPLAINT REGISTRATION AND MANAGEMENT SYSTEM**

**DESCRIPTION**

The Online Complaint Registration and Management System is a user-friendly software solution designed to streamline the process of submitting, tracking, and resolving complaints or issues encountered by individuals or organizations. It provides a centralized platform for efficient complaint management, allowing users to securely register complaints, track their progress in real time, and interact with assigned agents for issue resolution. With features such as automatic notifications, intelligent complaint routing, and robust security measures, this system ensures timely and effective handling of complaints while prioritizing user Details.

**SYSTEM REQUIREMENTS :**

To ensure smooth development, deployment, and usage of the Agri-Tech Web Application, certain system prerequisites must be met. These requirements are categorized into software, setup, and hardware specifications, all of which contribute to building a robust and scalable agricultural platform.

### 1. Software Requirements

These are the essential tools and platforms required to develop, test, and run the application efficiently.

#### 

* **Operating System**: Windows 10/11, macOS, or Linux — Supports cross-platform development and testing.
* **Node.js (v16 or above)**: Provides the runtime environment for building and managing frontend logic and Powers the server-side logic and handles API routing. 👉 [Download Node.js](https://nodejs.org/)
* **npm (v8 or above)**: A package manager required to install dependencies for React and related libraries.
* **React.js**: A JavaScript library for building dynamic and responsive user interfaces.
* **Browser**: Google Chrome / Firefox (latest version) — For rendering and testing the UI in real-time.
* **Express.js**: A lightweight web framework for building RESTful APIs.
* **MongoDB**: A NoSQL database used to store structured and unstructured data related to farms, users, and resources. 👉 [Download MongoDB](https://www.mongodb.com/try/download/community)
* **Postman**: Tool for testing APIs during development.
* **Visual Studio Code**: Preferred code editor with built-in Git and terminal support.
* **Git & GitHub**: For version control and collaborative development.

### 2. Hardware Requirements

Describes the minimum and recommended specifications needed to support the development and usage of the application.

* **Processor**: Intel Core i5 (8th Gen or above) / AMD Ryzen 5 or better — Ensures fast compilation and multitasking during development.
* **RAM**: Minimum 8 GB (16 GB recommended) — For handling development servers, IDEs, and browser testing simultaneously.
* **Storage**: At least 1 GB free space — Required for package installations, MongoDB setup, and local project files.
* **Display**: 1366x768 or higher — Recommended for optimal coding experience and application layout visualization.

# **SCENARIO-BASED CASE STUDY**

Scenario: John, a customer, recently encountered a problem with a product he purchased online. He notices a defect in the item and decides to file a complaint using the Online Complaint Registration and Management System.

1. **User Registration and Login:**
   * John visits the complaint management system's website and clicks on the "Sign Up" button to create a new account.
   * He fills out the registration form, providing his full name, email address, and a secure password.
   * After submitting the form, John receives a verification email and confirms his account.
   * He then logs into the system using his email and password.
2. **Complaint Submission:**
   * Upon logging in, John is redirected to the dashboard where he sees options to register a new complaint.
   * He clicks on the "Submit Complaint" button and fills out the complaint form.
   * John describes the issue in detail, attaches relevant documents or images showcasing the defect, and provides additional information such as his contact details and the product's purchase date.
   * After reviewing the information, John submits the complaint.
3. **Tracking and Notifications:**
   * After submitting the complaint, John receives a confirmation message indicating that his complaint has been successfully registered.
   * He navigates to the "My Complaints" section of the dashboard, where he can track the status of his complaint in real time.
   * John receives email notifications whenever there is an update on his complaint, such as it being assigned to an agent or its resolution status.
4. **Interaction with Agent:**
   * A customer service agent, Sarah, is assigned to handle John's complaint.
   * Sarah reviews the details provided by John and contacts him through the system's built-in messaging feature.
   * John receives a notification about Sarah's message and accesses the chat window to communicate with her.
   * They discuss the issue further, and Sarah assures John that the company will investigate and resolve the problem promptly.
5. **Resolution and Feedback:**
   * After investigating the complaint, the company identifies the defect in the product and offers John a replacement or refund.
   * John receives a notification informing him of the resolution, along with instructions on how to proceed.
   * He provides feedback on his experience with the complaint handling process, expressing his satisfaction with the prompt resolution and courteous service provided by Sarah.
6. **Admin Management:**
   * Meanwhile, the system administrator monitors all complaints registered on the platform.
   * The admin assigns complaints to agents based on their workload and expertise.
   * They oversee the overall operation of the complaint management system, ensuring compliance with platform policies and regulations.

# **PROJECT ARCHITECTURE**

# **TECHNICAL ARCHITECTURE**



The technical architecture of our online complaint registration and management app follows a client-server model, where the front end serves as the client and the back end acts as the server. The front end encompasses the user interface and presentation and incorporates the Axios library to connect with the backend easily by using RESTful Apis.

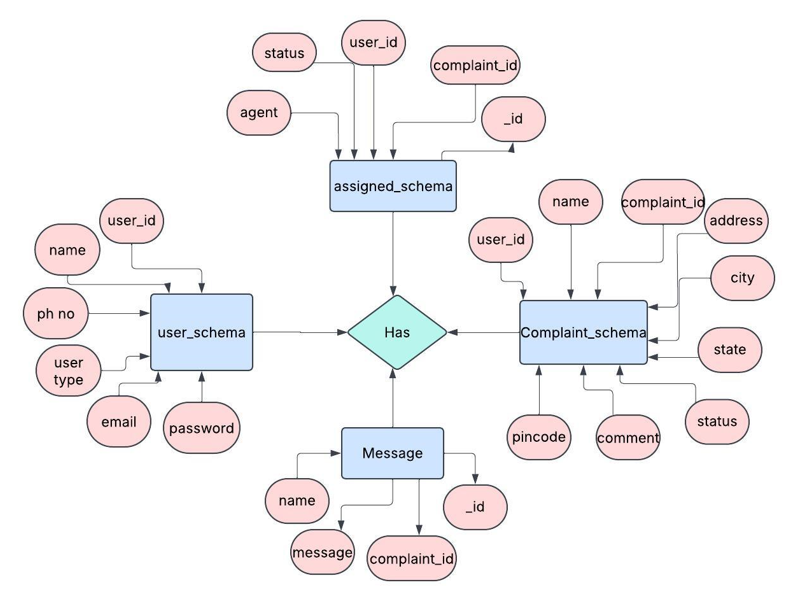
The front end utilizes the bootstrap and material UI library to establish a real-time and better UI experience for any user whether it is an agent, admin, or ordinary user working on it.

On the backend side, we employ Express.js frameworks to handle the server-side logic and communication.

For data storage and retrieval, our backend relies on MongoDB. MongoDB allows for efficient and scalable storage of user data, including user profiles, complaints registration, etc. It ensures reliable and quick access to the necessary information during registration of users or any complaints.

Together, the frontend and backend components, along with socket.io, Express.js, WebRTC API, and MongoDB, form a comprehensive technical architecture for our video conference app. This architecture enables real-time communication, efficient data exchange, and seamless integration, ensuring a smooth and immersive video conferencing experience for all users.

# **ER DIAGRAM**



This is the ER diagram of the project which shows the relationship between the user and the agent. It shows how users who have required fields can raise a complaint by filling required fields.

It illustrates how these entities relate to each other, helping us understand the underlying database structure and the flow of information within the app. He / She can also communicate with the agent with a chat window that follows the message schema which uses userId and complaintId from other schemas.

**FEATURES :**

### 1. USER REGISTRATION & PROFILE CREATION

* SECURE SIGN-UP: Users can register using email and password with secure authentication protocols.
* PROFILE CREATION: Stores user details including name, contact info, and ID details for future complaint tracking.

### 2. OFFICER BROWSING & COMPLAINT TYPE FILTERING

* SEARCH AND FILTER OFFICERS: Users can filter complaint categories or departments (e.g., Police, Electricity, Municipality).
* REAL-TIME STATUS TRACKING: Users can view officer availability or the live status of complaints (e.g., Pending, In Progress, Resolved).

### 3. COMPLAINT LODGING & MANAGEMENT

* EASY COMPLAINT FORM: Users lodge complaints by selecting category, writing a description, attaching documents/images.
* AUTOMATED UPDATES: Email/SMS notifications sent for submission, updates, and resolution to keep users informed.

### 4. OFFICER’S DASHBOARD

* MANAGE ASSIGNED COMPLAINTS: Officers can see, filter, and update complaints assigned to them.
* ACTION LOGGING: Officers can add status updates, action notes, and attach resolution reports securely.

### 5. ADMIN CONTROLS & VERIFICATION

* OFFICER APPROVAL SYSTEM: Admins verify and approve officer accounts to maintain system integrity.
* SYSTEM MONITORING: Admin has access to analytics, can resolve disputes, handle escalations, and maintain platform security.

**ROLES AND RESPONSIBILITIES :**

### 1. USER REGISTRATION

**Example:** Ravi, a citizen who wants to report a water leakage issue, visits the Online Complaint Portal. He registers as a user by providing his email, phone number, and password to create a secure account. Once registration is successful, he logs in and is greeted with a clean dashboard to begin.

### 2. BROWSING DEPARTMENTS & COMPLAINT TYPES

**Example:** Ravi browses through various departments—such as Municipal Corporation, Electricity, Police—and filters based on category, location, and urgency level. He selects Municipal Department → Water Leakage.

### 3. LODGING A COMPLAINT

**Example:** Ravi clicks on “Lodge Complaint,” selects the appropriate category, fills in the issue details, and uploads an image showing the leak. Once submitted, he gets a confirmation that the complaint has been received and is under review.

### 4. COMPLAINT CONFIRMATION

**Example:** A concerned officer from the municipal department reviews Ravi’s submission and assigns it to a field technician. Ravi receives a notification via email/SMS that his complaint has been accepted and assigned with a tracking number.

### 5. USER COMPLAINT MANAGEMENT

**Example:** Ravi can view the status of all his complaints on the dashboard—Pending, In Progress, or Resolved. He can also edit, cancel, or reopen a complaint if necessary, and communicate with the assigned officer.

### 6. ADMIN APPROVAL (BACKGROUND PROCESS)

**Example:** An admin in the backend reviews and approves officer accounts and departmental access to ensure only verified officials are part of the platform. The admin also monitors misuse or spam complaints.

### 7. PLATFORM GOVERNANCE

**Example:** The admin oversees overall platform activity, addresses escalated issues, handles user or officer disputes, enforces policies, and ensures data security and privacy compliance.

### 8. OFFICER'S DASHBOARD & TASK MANAGEMENT

**Example:** The assigned officer logs into the dashboard, views the complaint details, and updates the complaint status (e.g., "Technician Assigned", "Work Started", "Resolved"). Officers can also provide notes, attach images, or request additional info.

### 9. FIELD VISIT & COMPLAINT RESOLUTION

**Example:** A technician visits the location, fixes the leak, and updates the status in the system as Resolved. Officer uploads photos of the resolved issue for confirmation.

**USER FLOW :**

1. **Customer/Ordinary User:**
   * **Role:** Create and manage complaints, interact with agents, and manage profile information.

**Flow:**

* + 1. **Registration and Login:**
       - Create an account by providing the necessary information such as email and password.
       - Log in using the registered credentials.
    2. **Complaint Submission:**
       - Fill out the complaint form with details of the issue, including description, contact information, and relevant attachments.
       - Submit the complaint for processing.
    3. **Status Tracking:**
       - View the status of submitted complaints in the dashboard or status section.
       - Receive real-time updates on the progress of complaints.
    4. **Interaction with Agents:**
       - Connect with assigned agents directly using the built-in messaging feature.
       - Discuss complaints further and provide additional information or clarification.
    5. **Profile Management:**
       - Manage personal profile information, including details and addresses.

1. **Agent:**
   * **Role:** Manage complaints assigned by the admin, communicate with customers, and update complaint statuses.

**Flow:**

* + 1. **Registration and Login:**
       - Create an account using email and password.
       - Log in using the registered credentials.
    2. **Complaint Management:**
       - Access the dashboard to view and manage complaints assigned by the admin.
       - Communicate with customers regarding their complaints through the chat window.
    3. **Status Update:**
       - Change the status of complaints based on resolution or progress.
       - Provide updates to customers regarding the status of their complaints.
    4. **Customer Interaction:**
       - Respond to inquiries, resolve issues, and address feedback from customers.

1. **Admin:**
   * **Role:** Oversee the overall operation of the complaint registration platform, manage complaints, users, and agents, and enforce platform policies.
     1. **Management and Monitoring:**
        + Monitor and moderate all complaints submitted by users.
        + Assign complaints to agents based on workload and expertise
     2. **Complaint Assignment:**
        + Assign complaints to the desired agents for resolution.
        + Ensure timely and efficient handling of complaints.
     3. **User and Agent Management:**
        + Manage user and agent accounts, including registration, login, and profile information.
        + Enforce platform policies, terms of service, and privacy regulations.
     4. **Continuous Improvement:**
        + Implement measures to improve the platform's functionality, user experience, and security measures.
        + Address any issues or concerns raised by users or agents for better service delivery.

**PROJECT SETUP AND CONFIGURATION :**

**Creating project folder :**

* Create a folder on the desktop . (mkdir project-name)
* cd project-name
* project-name/
* │
* ├── client/ # React frontend
* ├── server/ # Node.js + Express backend
* └── README.md

**Client Setup (Installing React App) :**

* Navigate into the client directory and create the React app: ( npm create vite@latest client )
* Navigate to the client directory : ( cd client )
* Install all the required libraries, such as React, React Router, Ant Design, and others. (npm install)
* To start the frontend server and run the React application: ( npm start )

**Server Setup (npm init ) :**

* Navigate to the root and create the backend folder: ( mkdir server )
* Navigate to the server directory : ( cd server )
* Import Package.json: (npm init -y)
* Install backend dependencies:

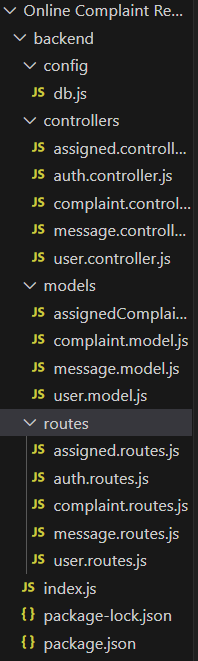
npm install express cors dotenv mongoose

npm install nodemon --save-dev

* Run the server ( npm run dev )

**BACKEND DEVELOPMENT:**

[**https://drive.google.com/file/d/1rgw5-ip9N-h3i2mB3mpwMAjcuY09NhSV/view?usp=sharing**](https://drive.google.com/file/d/1rgw5-ip9N-h3i2mB3mpwMAjcuY09NhSV/view?usp=sharing)

****

* **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, creating, assigning complaints, and chat window.
  + Implement route handlers using Express.js to handle requests and interact with the database.

#### **Implement Data Models:**

* + 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).
  + Create routes and middleware for user registration, login, and logout.
  + Set up authentication middleware to protect routes that require user authentication.

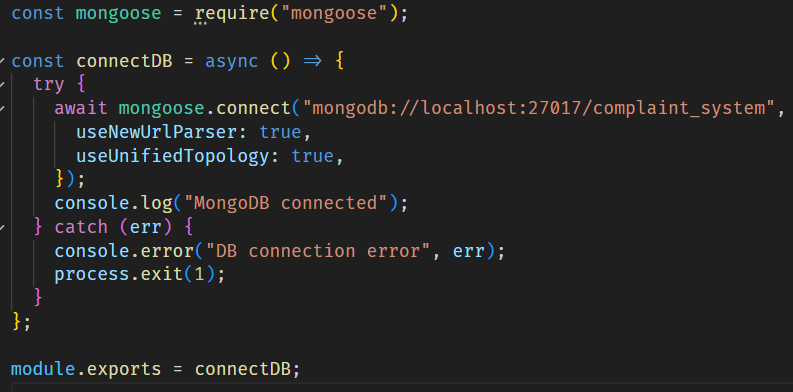
#### **Admin Functionality:**

* + Implement routes and controllers specific to admin functionalities such as fetching all the data regarding users, complaints, agents.

#### **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.

**DATABASE DEVELOPMENT :**

****

* Imports **Mongoose**, which provides schema-based modeling and convenient MongoDB operations in [Node.js](http://node.js).
* Declares a function called connectToDB. This will be called in your main server file (server.js or app.js) to initiate the DB connection.

* Connects to MongoDB using the **connection URI** stored in process.env.MONGO\_DB.

The second argument is an **options object**:

* useNewUrlParser: true ensures the new MongoDB connection string parser is used.
* useUnifiedTopology: true uses the new Server Discovery and Monitoring engine, which handles replica sets and sharding better.

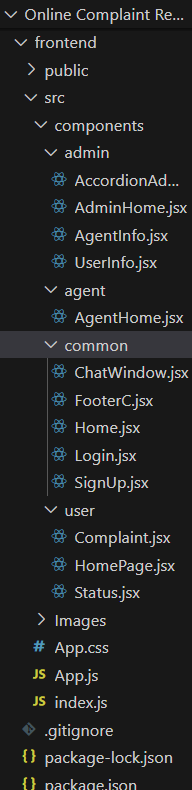
* If the connection is **successful**, it logs a confirmation message.
* If the connection **fails**, it throws an error with a message.

It’s a good practice to **handle this error gracefully** instead of just throwing it. You could console.error() and exit the process using process.exit(1) if needed.

* Exports the connectToDB function so it can be used in other files, like your server.js:

**FRONTEND DEVELOPMENT:**

[**https://drive.google.com/file/d/1K5NxXq4OoV4S\_eTGrLk\_9hMZkWRifiZm/view?usp=sharing**](https://drive.google.com/file/d/1K5NxXq4OoV4S_eTGrLk_9hMZkWRifiZm/view?usp=sharing)

****

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 front end, 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 the 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 the Customer Care Registry, like making specific complaints or managing your Product complaints.

1. **Implement frontend logic:**

In the final leg of the front-end development, we'll bridge the gap between the visual interface and the underlying data. It involves the following stages.

* + Integration with API endpoints.
  + Implement data binding.

**PROJECT EXECUTION.**

**Steps for Project execution :**

**Install Prerequisites :**

| **Tool** | **Required Version (or higher)** |
| --- | --- |
| Node.js | v14+ or v16+ |
| npm | v6+ or v7+ |

**Install Project Dependencies:**

Run the following command to install all packages listed in package.json:

**npm install**

**It will install:**

* React
* Axios
* React Bootstrap
* Ant Design
* MDB UI Kit
* MUI
* React Router DOM
* Moment
* Other necessary dependencies

**Set Up Environment Variables (Optional but Recommended):**

If you're using .env, create a .env file in the root and add:

**Run the React App :**

npm start

This will start your React development server on http://localhost:3000.

Make Sure Backend is Running

Your frontend connects to APIs like: [**http://localhost:8001/api/user/login**](http://localhost:8001/api/user/login)

so your backend server (Node.js/Express) must also be running on port 8001**.**

**Output ScreenShots :**

# **PROJECT STRUCTURE:**





The first image is of frontend part which shows all the files and folders that have been used in UI development

The second image is of the Backend part which shows all the files and folders that have been used in the backend development

# **APPLICATION FLOW:**

**Online Complaint Registration and Management System**

1. **Customer/Ordinary User:**
   * **Role:** Create and manage complaints, interact with agents, and manage profile information.
   * **Flow:**
     1. **Registration and Login:**
        + Create an account by providing the necessary information such as email and password.
        + Log in using the registered credentials.
     2. **Complaint Submission:**
        + Fill out the complaint form with details of the issue, including description, contact information, and relevant attachments.
        + Submit the complaint for processing.
     3. **Status Tracking:**
        + View the status of submitted complaints in the dashboard or status section.
        + Receive real-time updates on the progress of complaints.
     4. **Interaction with Agents:**
        + Connect with assigned agents directly using the built-in messaging feature.
        + Discuss complaints further and provide additional information or clarification.
     5. **Profile Management:**
        + Manage personal profile information, including details and addresses.
2. **Agent:**
   * **Role:** Manage complaints assigned by the admin, communicate with customers, and update complaint statuses.
   * **Flow:**
     1. **Registration and Login:**
        + Create an account using email and password.
        + Log in using the registered credentials.
     2. **Complaint Management:**
        + Access the dashboard to view and manage complaints assigned by the admin.
        + Communicate with customers regarding their complaints through the chat window.
     3. **Status Update:**
        + Change the status of complaints based on resolution or progress.
        + Provide updates to customers regarding the status of their complaints.
     4. **Customer Interaction:**
        + Respond to inquiries, resolve issues, and address feedback from customers.
3. **Admin:**
   * **Role:** Oversee the overall operation of the complaint registration platform, manage complaints, users, and agents, and enforce platform policies.
   * **Flow:**
     1. **Management and Monitoring:**
        + Monitor and moderate all complaints submitted by users.
        + Assign complaints to agents based on workload and expertise.
     2. **Complaint Assignment:**
        + Assign complaints to the desired agents for resolution.
        + Ensure timely and efficient handling of complaints.
     3. **User and Agent Management:**
        + Manage user and agent accounts, including registration, login, and profile information.
        + Enforce platform policies, terms of service, and privacy regulations.
     4. **Continuous Improvement:**
        + Implement measures to improve the platform's functionality, user experience, and security measures.
        + Address any issues or concerns raised by users or agents for better service delivery.

# **Project Flow:**

Before starting to work on this project, let’s see the demo.

**Project demo:** [**https://drive.google.com/file/d/1YwXaHRBZJL\_V7dcEK8SOmtPWZasAxccm/view?usp=drive\_link**](https://drive.google.com/file/d/1YwXaHRBZJL_V7dcEK8SOmtPWZasAxccm/view?usp=drive_link)

Use the code at: <https://github.com/awdhesh-student/complaint-registery.git>

or follow the videos below for a better understanding.

**Milestone 1:**

**Project Setup and Configuration:**

1. **Create project folders and files:**

Now, firstly create the folders for frontend and backend to write the respective code and install the essential libraries.

* + Client folders.
  + Server folders

1. **Install required tools and software:**

For the backend to function well, we use the libraries mentioned in the prerequisites. Those libraries include

* + Node.js.
  + MongoDB.
  + Bcrypt
  + Body-parser

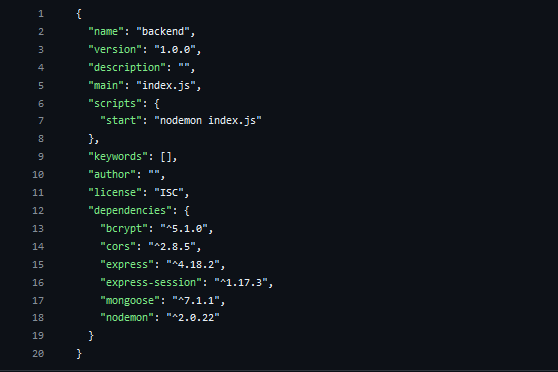
Also, for the frontend we use the libraries such as

* + React Js.
  + Material UI
  + Bootstrap
  + Axios

After the installation of all the libraries, the package.json files for the front end look like the one mentioned below.



After the installation of all the libraries, the package.json files for the backend look like the one mentioned below.



**Milestone 2:**

**Backend Development:**

* **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, creating, assigning complaints, and chat window.
  + Implement route handlers using Express.js to handle requests and interact with the database.

#### **Implement Data Models:**

* + 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).
  + Create routes and middleware for user registration, login, and logout.
  + Set up authentication middleware to protect routes that require user authentication.

#### **Admin Functionality:**

* + Implement routes and controllers specific to admin functionalities such as fetching all the data regarding users, complaints, agents.

#### **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.

Reference video for backend code:

<https://drive.google.com/file/d/19UO9XUotRVeJASTdmNIIPaYzrhlrkTAx/view?usp=drive_link>

**Milestone 3:**

**Database Development**

1. **User Schema:**
   * The user schema defines the structure of user data stored in the database. It includes fields such as name, email, password, phone, and userType.
   * Each user must provide a name, email, password, phone number, and userType (e.g., customer, agent, admin).
   * User data is stored in the "user\_Schema" collection in the MongoDB database.
2. **Complaint Schema:**
   * The complaint schema specifies the format of complaint data registered by users.
   * It contains fields like userId, name, address, city, state, pincode, comment, and status.
   * Complaints are associated with users through the userId field, and each complaint must have a name, address, city, state, pincode, comment, and status.
   * Complaint data is stored in the "complaint\_schema" collection in the MongoDB database.
3. **Assigned Complaint Schema:**
   * The assigned complaint schema defines how complaints are assigned to agents for resolution.
   * It includes fields such as agentId, complaintId, status, and agentName.
   * Each assigned complaint is linked to a specific agent (identified by agentId) and complaint (identified by complaintId).
   * The status field indicates the current status of the assigned complaint.
   * Assigned complaint data is stored in the "assigned\_complaint" collection in the MongoDB database.
4. **Chat Window Schema:**
   * The chat window schema governs the structure of messages exchanged between users and agents regarding specific complaints.
   * It comprises fields like name, message, and complaintId.
   * Messages are associated with a complaint through the complaintId field, allowing for easy tracking and retrieval of chat history for each complaint.
   * Message data is stored in the "message" collection in the MongoDB database.

Reference video for Database code:

<https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view>

**Milestone 4:**

**Frontend Development:**

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 front end, 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 the 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 the Customer Care Registry, like making specific complaints or managing your Product complaints.

1. **Implement frontend logic:**

In the final leg of the front-end development, we'll bridge the gap between the visual interface and the underlying data. It involves the following stages.

* + Integration with API endpoints.
  + Implement data binding.

Reference video for frontend code:

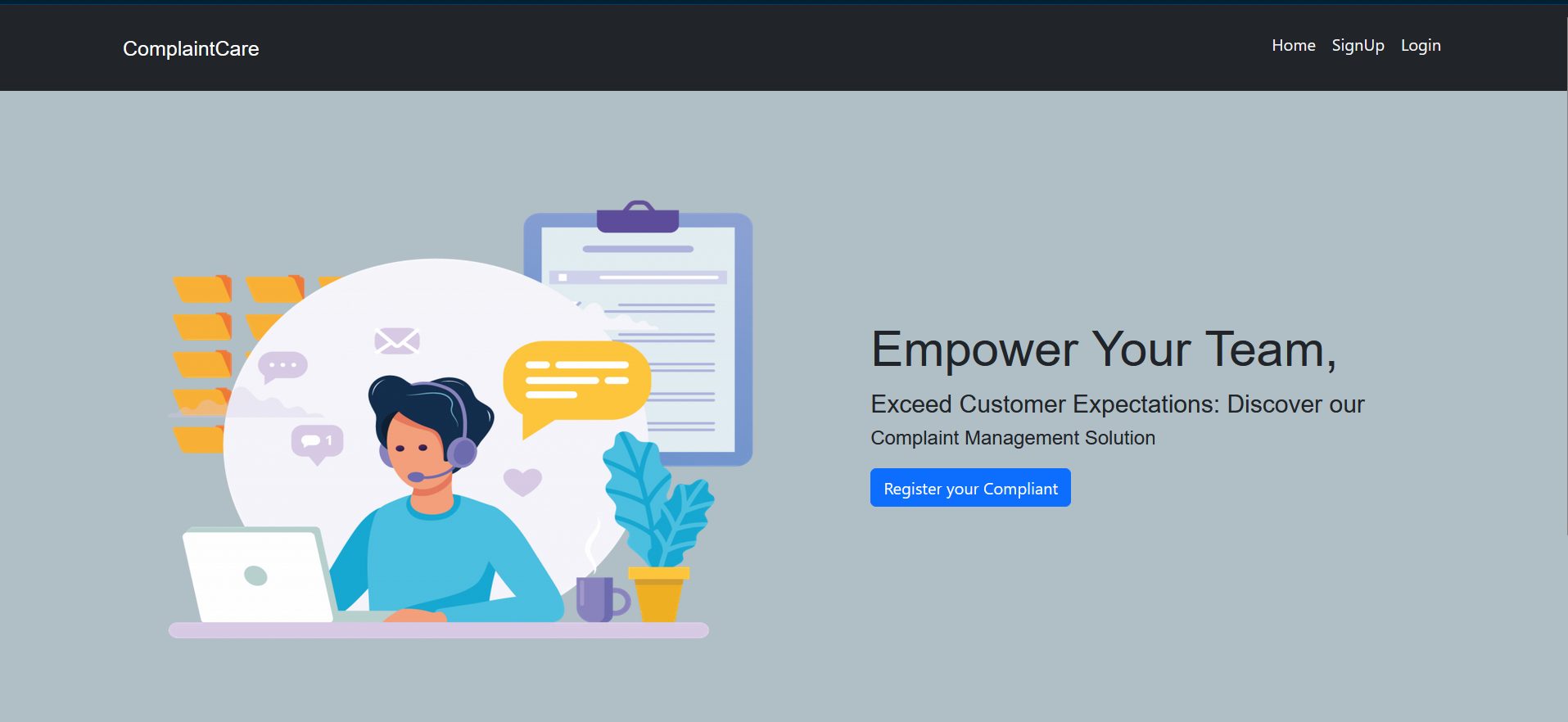
<https://drive.google.com/file/d/1k5Ww7arbOk-uWC3_ebx7X0q6dD-u-wQg/view?usp=drive_link>

**Milestone 5:**

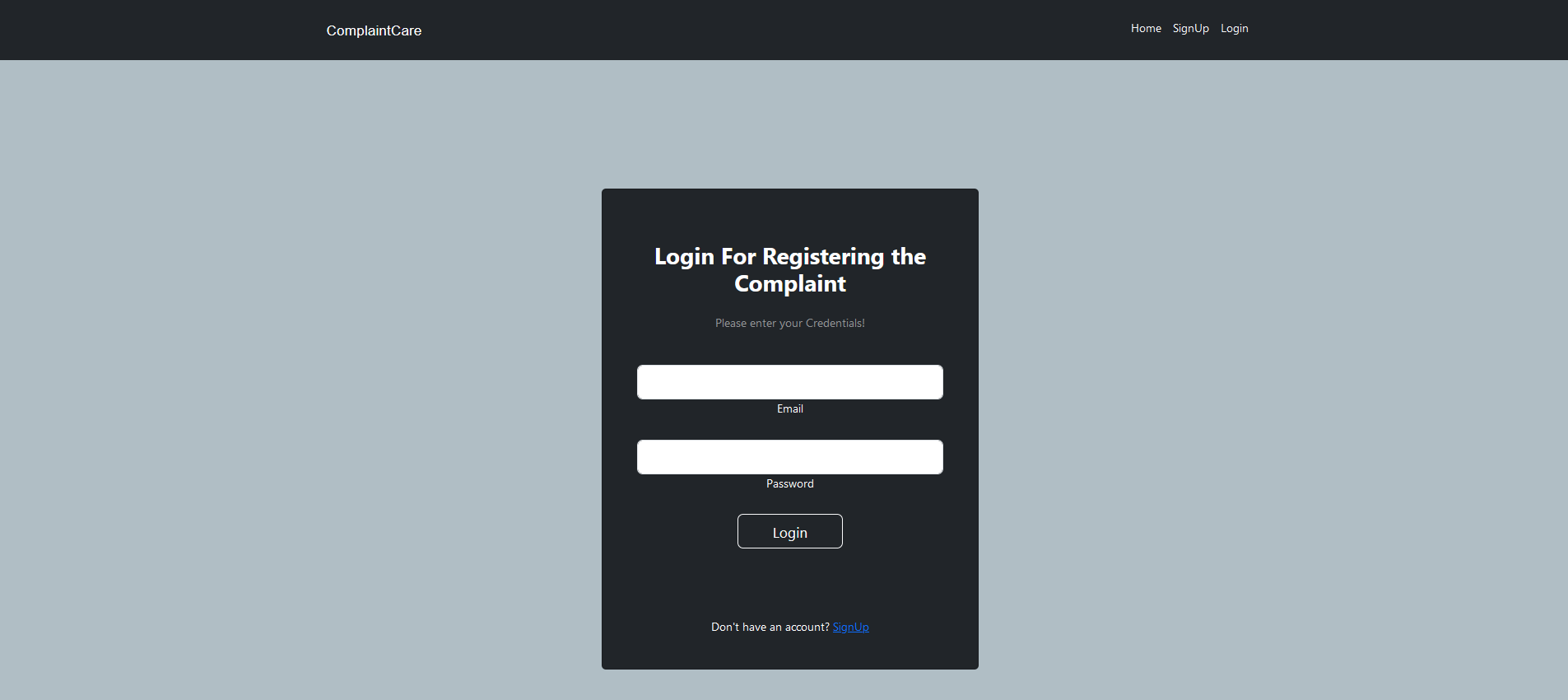
**Project Implementation:**

On completing the development part, we then ran the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one provided below.

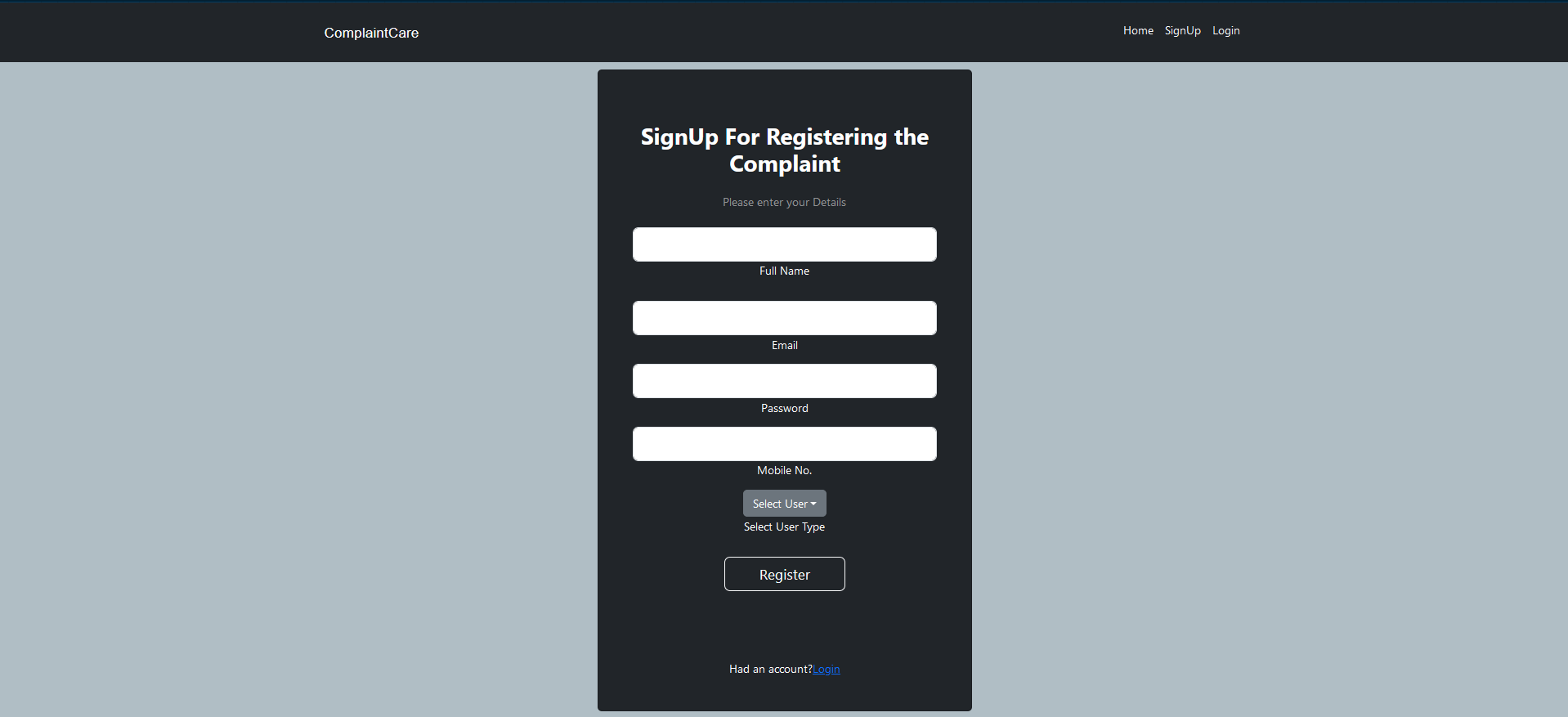
* Landing Page



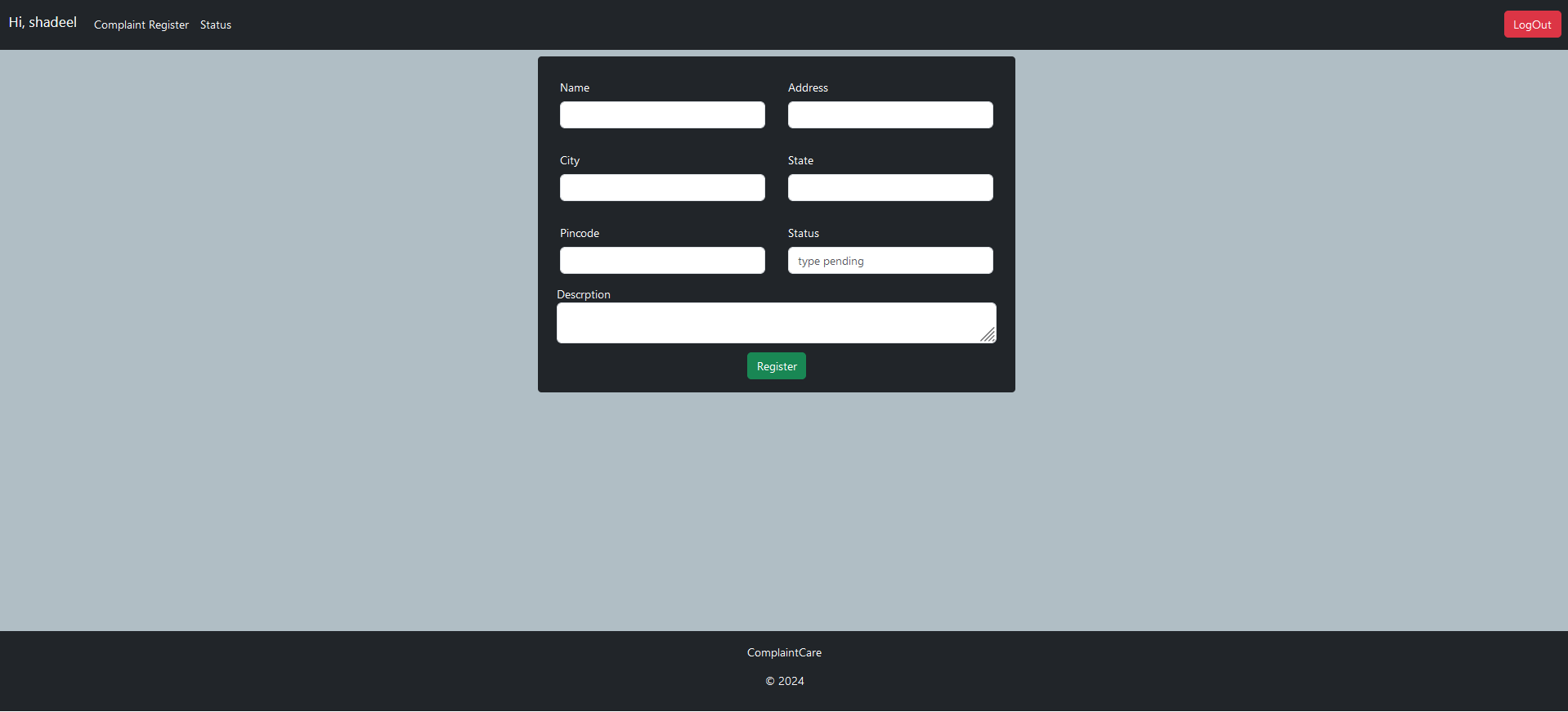
* Login Page



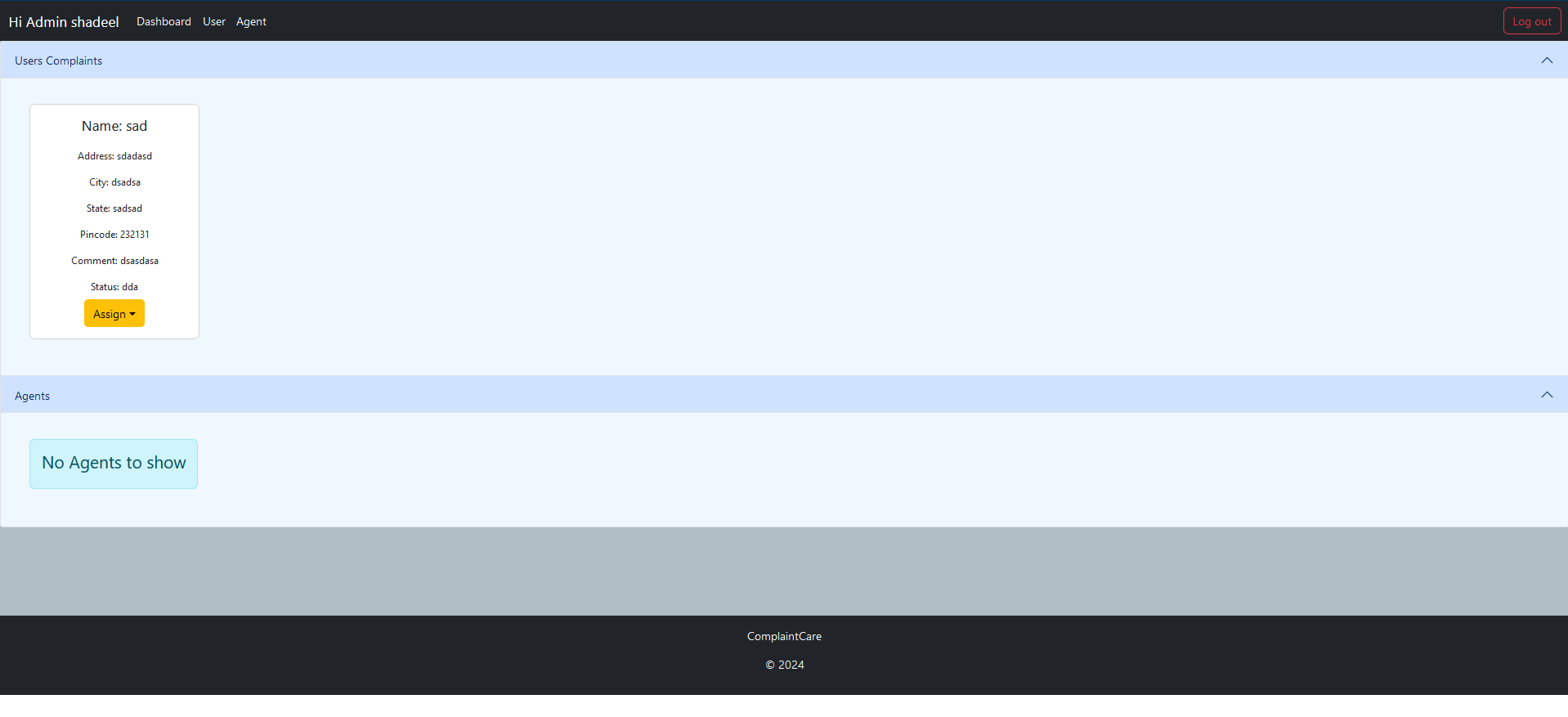
* Registration Page



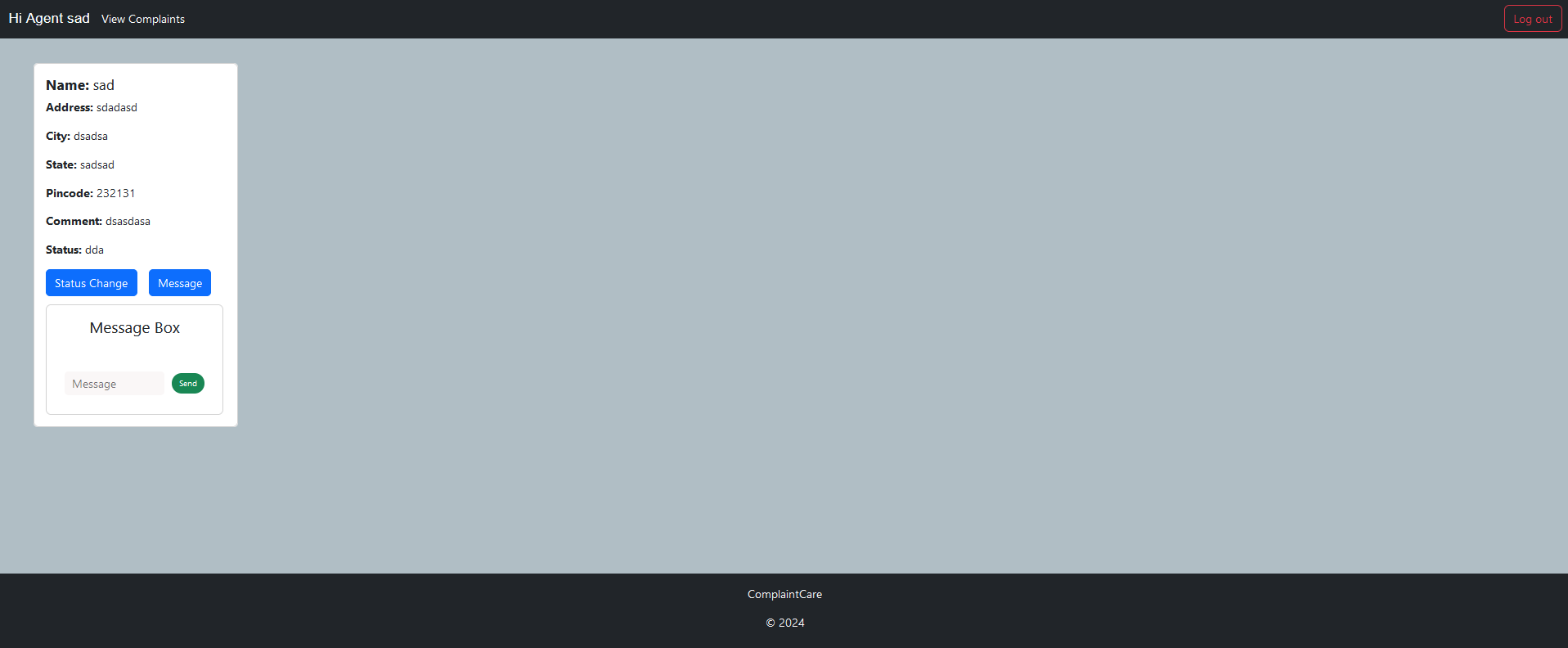
* Common Dashboard For Complaint



* Admin Dashboard



* Agent Dashboard



For any further doubts or help, please consider the code from Google Drive,

<https://drive.google.com/drive/folders/1uGwb-keRJCab88xNFCD4EoXzZsDChZe3?usp=drive_link>

The demo of the app is available at:

<https://drive.google.com/file/d/1YwXaHRBZJL_V7dcEK8SOmtPWZasAxccm/view?usp=drive_link>