CERTIFICATE

Project Title: "**Resolvenow: your platform for online complaints "is** a bona fide work carried out by the following students

➢ TEAM ID: LTVIP2025TMID52273

- **Mounika Tumbalam (Team Leader)**
- **Kadiyam Tansi (Team Member)**
- **Sane ShivaPreethi (Team Member)**
- **V Shireesha (Team Member)**

TABLE OF CONTENTS

1. INTRODUCTION

Resolvenow: your platform for online complaints

An online complaint registration and management system is a software application or platform that allows individuals or organizations to submit and track complaints or issues they have encountered. It can help optimize the complaint handling process and empower organizations to develop a safety management system to efficiently resolve customer complaints, while staying in line with industry guidelines and regulatory compliance obligations. It provides a centralized platform for managing complaints, streamlining the complaint resolution process, and improving customer satisfaction.

It consists of some key features which include:

1. User registration: Users can create accounts to submit complaints and track their progress.
2. Complaint submission: Users can enter details of their complaints, including relevant information such name, description of the issue, address etc.
3. Tracking and notifications: Users can track the progress of their complaints, view updates, and receive notifications via email or SMS when there are any changes or resolutions.
4. User can interact with the agent who has assigned the complaint.
5. Assigning and routing complaints: The system assigns complaints to the appropriate department or personnel responsible for handling them. It may use intelligent routing algorithms to ensure efficient allocation of resources.
6. Security and confidentiality: The system ensures the security and confidentiality of user data and complaint information through measures such as user authentication, data encryption, access controls, and compliance with relevant data protection regulations.

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.

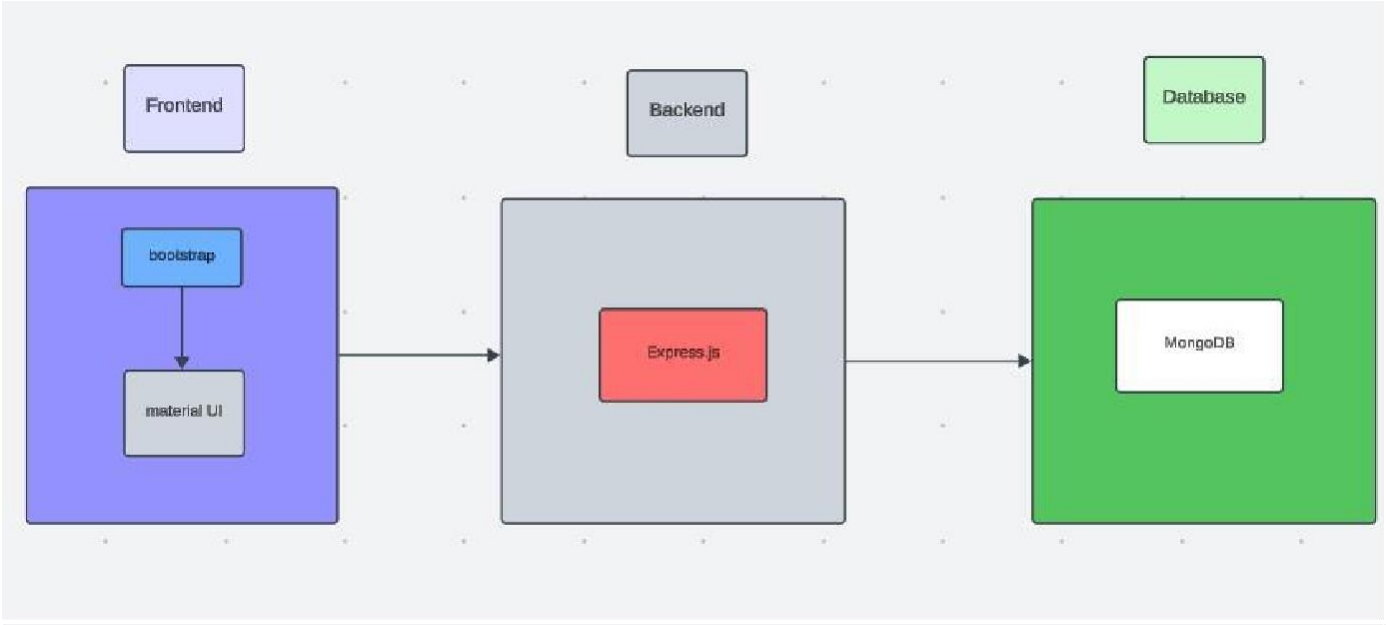
2. PROJECT OVERVIEW

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.

3. ARCHITECTURE

TECHNICAL ARCHITECTURE:



The technical architecture of our online complaint registration and management app follows a client-server model, where the frontend serves as the client and the backend acts as the server. The frontend encompasses not only the user interface and presentation but also incorporates the axios library to connect with backend easily by using RESTful Apis.

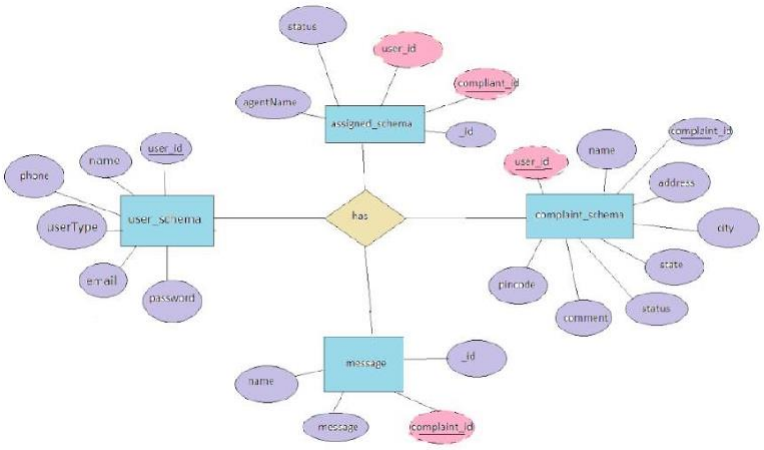
The frontend utilizes the bootstrap and material UI library to establish real-time and better UI experience for any user whether it is agent, admin or ordinary user working on it.

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, for complaints registration, etc. It ensures reliable and quick access to the necessary information during registration of user 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 user and agent It shows how user which have required fields can raise a complaint by fillings 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 chat window which follows the message schema which uses userId and complaintId from other schemas.

4. SET INSTRUCTIONS

PRE-REQUISITES/INSTALLATION:

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 serverside. 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.

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

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

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
- GIT CLONE : 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/Dabbakutivenky/Dabbakutivenky-resolvenow-your-platform-for-online-complaints.git

To run the existing Video Conference App project downloaded from GitHub: Follow below steps:
Clone the Repository:

- Open your terminal or command 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:

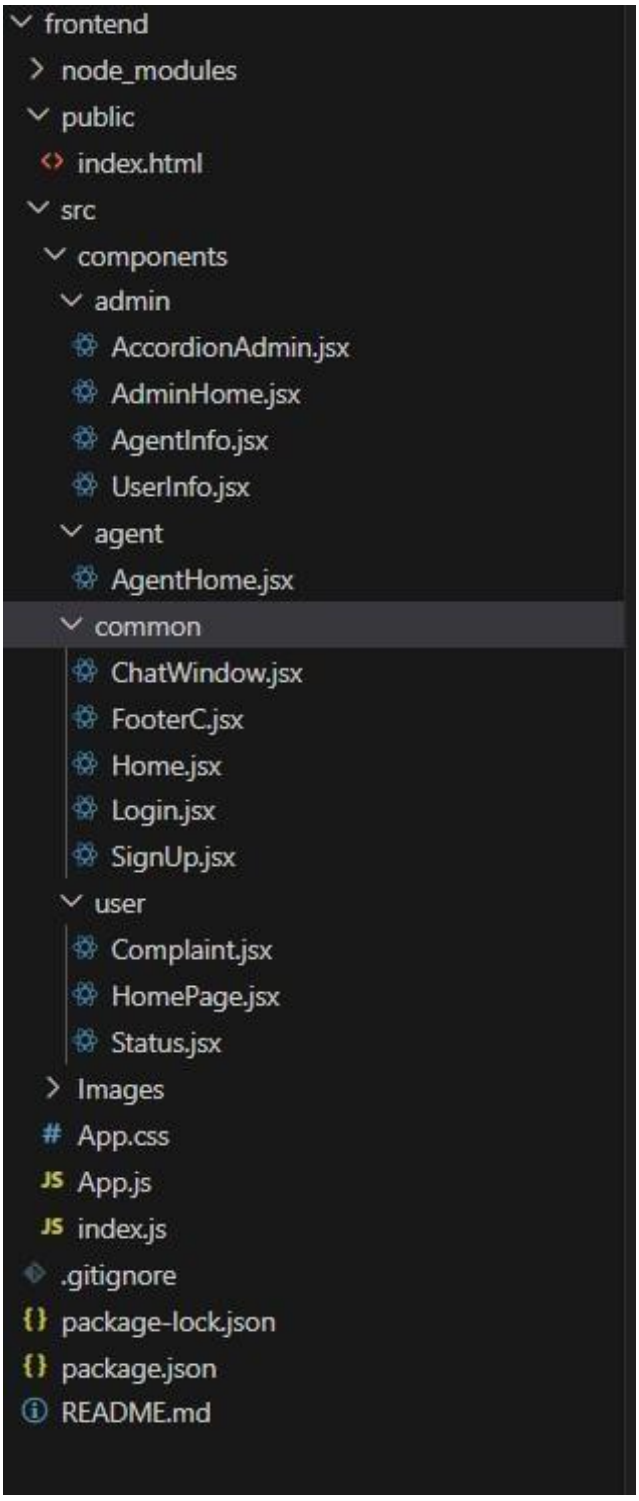
- To start the development server, execute the following command: npm start

- The online complaint registration and management app will be accessible at 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.

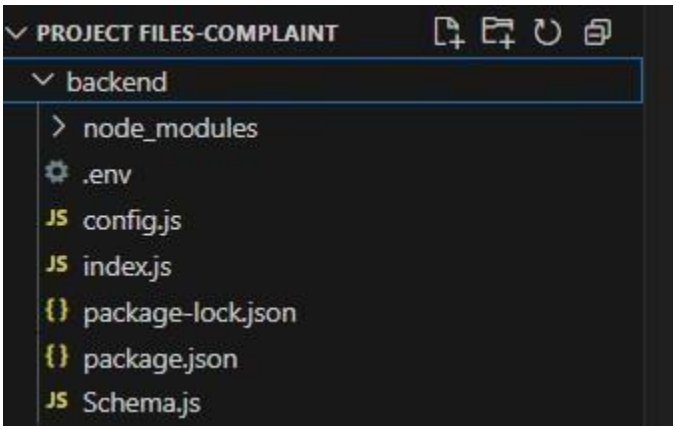
5. PROJECT STRUCTURE

Create a Project folder that contains files as shown below:

1. Frontend:


frontend
node_modules
public
index.html
src
components
admin
AccordionAdmin.jsx
AdminHome.jsx
AgentInfo.jsx
UserInfo.jsx
agent
AgentHome.jsx
common
ChatWindow.jsx
FooterC.jsx
Home.jsx
Login.jsx
SignUp.jsx
user
Complaint.jsx
HomePage.jsx
Status.jsx
Images
App.css
App.js
index.js
.gitignore
package-lock.json
package.json
README.md

2. Backend:



- The first image is of frontend part which is showing all the files and folders that have been used in UI development
- The second image is of Backend part which is showing all the files and folders that have been used in backend development

6. API DOCUMENTATION

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.

   - Frontend folders.
   - Backend folders

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

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

   - 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 frontend looks like the one mentioned below.

```
{
  "name": "task1",
  "version": "0.1.0",
  "proxy": "http://localhost:8000",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.2.3",
    "mdb-react-ui-kit": "^6.1.0",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.2",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

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

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "express-session": "^1.17.3",
    "mongoose": "^7.1.1",
    "nodemon": "^2.0.22"
  }
}
```

Backend Development:

- **Set Up Project Structure:**

o Create a new directory for your project and set up a package.json file using npm init command. o Install necessary dependencies such as Express.js, Mongoose, and other required packages.

- **Set Up Project Structure:**
  - o Create a new directory for your project and set up a package.json file using npm init command. o Install necessary dependencies such as Express.js, Mongoose, and other required packages.
- **Create Express.js Server:**
  - o Set up an Express.js server to handle HTTP requests and serve API endpoints.
  - o Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.
- **Define API Routes:**
  - o Create separate route files for different API functionalities such as authentication, stock actions, and transactions.
  - o 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 sessionbased 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.

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.

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 frontend, follow the below steps.

- Install required libraries.
- Create the structure directories.

2. **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.

3. **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.

7. AUTHENTICATION

OVERVIEW

Authentication in ResolveNow ensures that only registered users can access their respective dashboards and features. It uses JWT (JSON Web Tokens) for secure session management and role-based access.

USER ROLES:

Customer – Can register complaints and chat with agents.
Agent – Can manage assigned complaints and respond to users.
Admin – Full access to manage users, assign complaints, and monitor system.

REGISTRATION:

New users register via the /register endpoint.
They provide name, email, password, and role.
Passwords are encrypted using bcrypt before storing in the database.

LOGIN:

Users log in using /login endpoint.
If credentials are valid, a **JWT token** is generated and returned. This token is required for accessing protected routes.

TOKEN-BASED AUTHENTICATION:

JWT tokens are stored in the frontend (usually in localStorage).
For each protected API call, token is sent in the request header:

```
Authorization: Bearer <token>
```

MIDDLE WARE PROTECTION:

Backend routes are protected using an **authentication middleware**.
It checks for a valid token and extracts user info.
Unauthorized users are blocked with a 400response.

- **ROLE-BASED ACCESS CONTROL:**

After verifying token, the system checks user **role**.
Only users with the right role can access certain routes (e.g., admin dashboard, agent panel).
This prevents misuse of admin or agent APIs.

- **LOGOUT:**

Token is simply cleared from the client-side.
No server-side session is maintained, ensuring stateless design.

- **PASSWORD SECURITY:**

To ensure user data protection, ResolveNow implements secure password practices: • **Passwords are hashed** using bcrypt, a widely used hashing algorithm.

- This makes it nearly impossible to retrieve the original password from the database even if it is compromised.
- Salting is also used to enhance the strength of password hashes and prevent brute-force attacks.

- **SESSION SECURITY AND TOKEN EXPIRY:**

- JWT tokens have an **expiry time** set to limit session duration (e.g., 1 hour).
- Once the token expires, the user is logged out automatically and must reauthenticate.
- This improves security by minimizing risks of token misuse

FRONTEND TOKEN HANDLING:

- Tokens are stored in **localStorage** or **sessionStorage** on the client side.
- During logout, the token is cleared from storage to prevent unauthorized access.
- The frontend includes checks to detect expired or missing tokens and redirect users to the login page.

ERROR HANDLING:

ResolveNow provides clear and meaningful error messages in case of:

- Invalid login credentials (wrong email/password)
- Expired or missing token
- Unauthorized access to restricted routes
- Improper user roles

This helps both users and developers quickly identify and resolve issues.

SECURITY BEST PRACTICES:

To enhance the overall security of the authentication system:

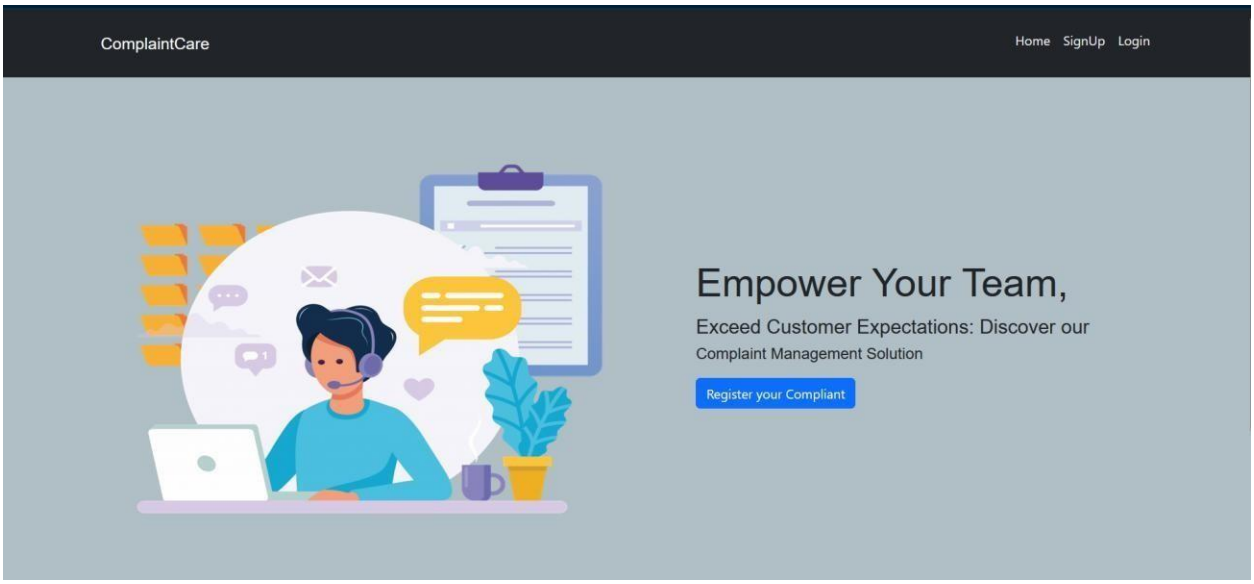
- HTTPS is used to encrypt all client-server communication.
- Input fields are validated on both client and server to prevent injection attacks.
- Rate limiting and brute-force protection mechanisms can be added to login APIs.

8. USER INTERFACE

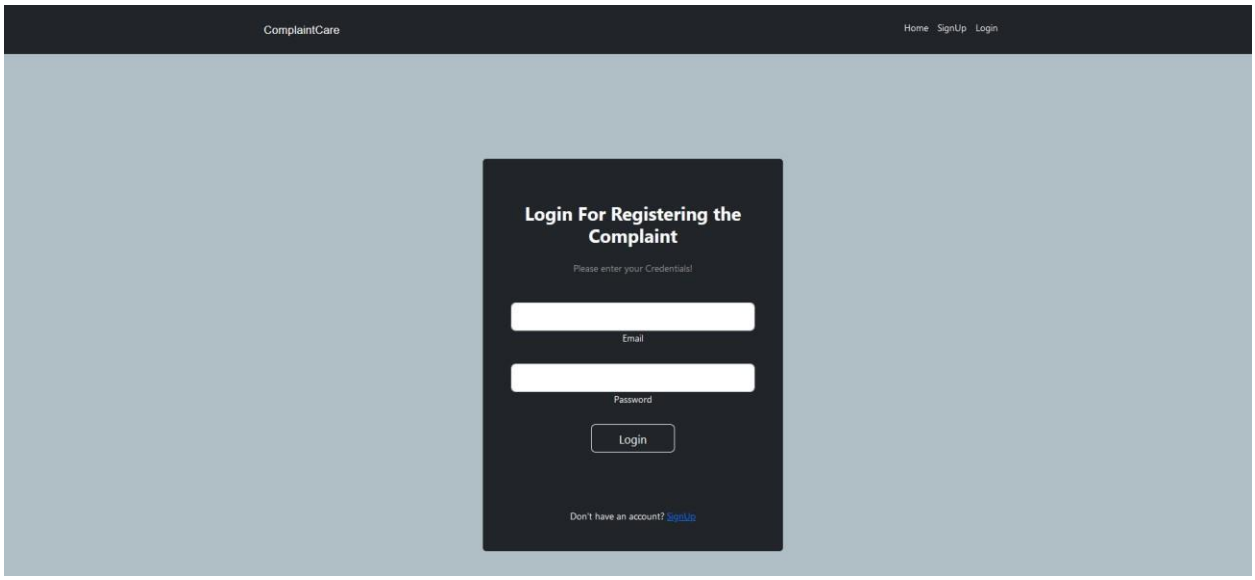
Project Implementation:

On completing the development part, we then run 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's provided below.

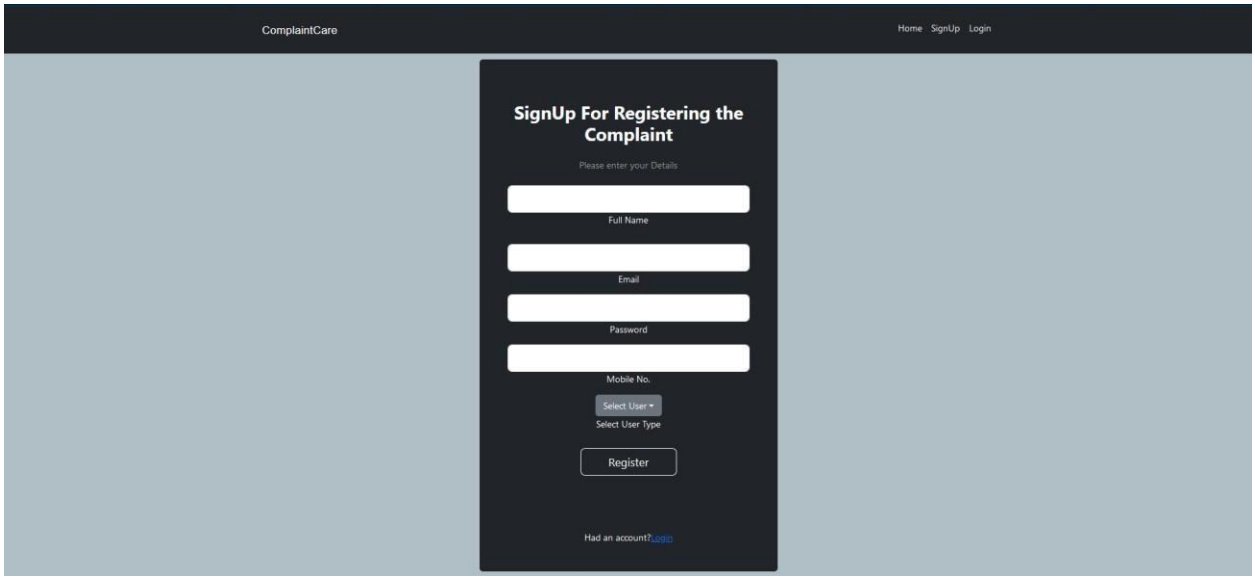
- **LANDING PAGE:**



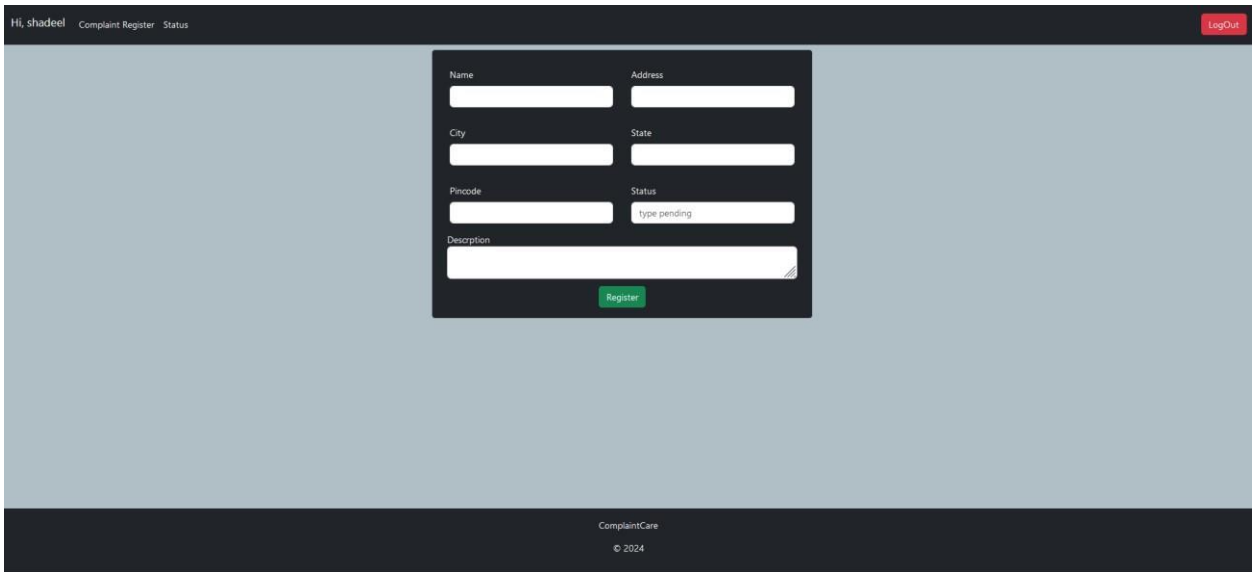
- **LOGIN PAGE:**



- **REGISTRATION PAGE:**



- **COMMON DASHBOARD FOR COMPLAINT:**



- **ADMIN DASHBOARD:**

- **AGENT DASHBOARD:**

Overview:

The user interface (UI) of **ResolveNow** is designed to be simple, user-friendly, and responsive for all types of users. It is built using **React.js**, combined with **Bootstrap** and **Material UI** libraries to enhance usability and visual appeal. The UI allows **Customers**, **Agents**, and **Admins** to interact with the system based on their roles.

Objectives of the UI:

- Provide a clean and intuitive design for users to register, log in, and perform their tasks smoothly.
- Enable complaint submission, tracking, and management through interactive dashboards.
- Support easy communication between customers and agents using a built-in chat window.
- Ensure responsive layout for desktops, tablets, and mobile devices.

Role-Based Dashboards:

Customer Dashboard:

- Register and log in as a customer.
- Submit complaints by filling a form with relevant details.
- Track complaint status (Pending, In Progress, Resolved).
- Chat with the assigned agent in real-time.

Agent Dashboard:

- View assigned complaints.
- Change complaint status (e.g., Mark as In Progress or Resolved).
- Communicate with customers via a chat interface.
- View complaint details for better understanding and resolution.

Admin Dashboard:

- View all registered complaints in the system.

- Assign complaints to agents.
- View all registered users (customers and agents).
- Monitor system activity and ensure proper workflow.

Key UI Components:

- **Navigation Bar**: Allows access to Login/Register, Home, and Dashboard.
- **Forms**: Used for complaint registration, login, and account creation.
- **Complaint Cards**: Display complaint information in an organized format.
- **Chat Window**: Real-time message exchange between customer and agent.
- **Status Labels**: Visual indicators to show the progress of complaints.

Technologies Used:

- **React.js**: For building dynamic and component-based UI.
- **Bootstrap & Material UI**: For styling, layout, and responsiveness.
- **Axios**: To connect the frontend with backend APIs.
- **React Router**: To manage navigation and routing between pages.

User Experience Goals:

- Easy navigation and minimal learning curve.
- Fast page load and responsiveness across devices.
- Visual consistency with color schemes and buttons.
- Error handling and confirmation alerts (e.g., “Complaint submitted successfully”).

9. TESTING

✅ Manual Testing:

- All modules were manually tested using real inputs.
- Features like registration, login, complaint submission, status updates, and chat were verified.

✅ API Testing with Postman:

- Each backend API was tested individually.
- Requests like POST /register, /login, /complaints, and /messages were tested using sample JSON payloads.

✅ UI Testing:

- Buttons, forms, and validations were checked for responsiveness and correctness.
- The app was tested on different screen sizes (mobile, tablet, laptop).

10. KNOWN ISSUES

- No **email verification** implemented after user registration.
- No **auto-refresh** of complaint/chat window – manual refresh required.
- **Image/file uploads** are not supported in complaint forms.
- Chat feature does not yet support **real-time sockets** (polling used).
- Role assignment is manual; no dedicated interface for admin to create roles.
- Some forms lack client-side validation feedback (e.g., “Please enter email”).

11. FUTURE ENHANCEMENTS

Here are some possible future improvements to make ResolveNow more powerful:

1. Real-time Chat using Socket.IO ○ Enables instant messaging between user and agent.
2. Email & SMS Notifications ○ Notify users when complaint is updated or resolved.
3. Complaint History & Analytics ○ Admins can view charts and stats of resolved/pending complaints.
4. File Upload Support ○ Allow users to attach documents/images when submitting complaints.
5. Multilingual Support ○ Add regional language options for better accessibility.
6. Push Notifications ○ Browser-based or mobile alerts for updates.
7. Admin Role Management Panel ○ UI for admins to create/assign/remove roles easily.
8. Mobile App Version ○ Build a React Native version for Android/iOS.

Running the Application:

A. FRONTEND:

 To run the **React frontend**-

- Open terminal and navigate to the frontend folder:

```
cd frontend
```

- Install dependencies:

```
npm install
```

- Start the frontend:

```
npm start
```

- Open browser and visit:

B. BACKEND:

- To run the **Node.js + Express backend**:
  - Open another terminal and navigate to the backend folder:

```
cd backend
```

  - Install dependencies:

```
npm install
```

  - Start the backend server:

```
npm start
```

  - Server runs at:

DEMO LINKS

- Video Demo Link:

https://drive.google.com/file/d/17mexnpYrKr6l9LiexOuVJI9CGonTgP4c/view?usp=sharing

- Project Links:

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

  https://github.com/Mounika-tumbalam/Smart-Bridge/