# Resolve Smarter Not Harder

RESOLVENOW

***INTRODUCTION:***

Project Title: ResolveNow (your platform for online complaints)

Team Members: 1. Sampada. P

2.Sai. B

3.Mounika. Y

4.Govindu. K

***PROJECT OVERVIEW:***

**Purpose:**

ResolveNow is an online dispute resolution platform designed to provide a fast, fair, and cost-effective way to resolve complaints and disputes. The platform aims to bridge the gap between consumers, businesses, and dispute resolution services, promoting a more efficient and satisfactory resolution process. At its core, the ResolveNow is a centralized database meticulously recording every touchpoint between customers and support teams. From inquiries and troubleshooting sessions to resolutions and follow-ups, the registry captures the entire spectrum of interactions.

It epitomizes a commitment to not only resolving issues but understanding the customer journey holistically. These insights guide strategic decisions, aiding in the development of targeted training programs, refined service protocols, and streamlined processes.

**Key Features:**

1. Login and Sign-up Pages: Secure and user-friendly login and sign-up pages for users to create an account and access the platform.

2. Complaint Registration Page: A simple and intuitive complaint registration page where users can submit their complaints, providing necessary details such as:

- Complaint description

- Category (e.g., product, service, etc.)

- Company/organization name

- Supporting documents (e.g., images, receipts, etc.)

3. Complaint Management System: A robust complaint management system that enables users to:

- Track the status of their complaints

- Receive updates and notifications on the resolution process

- Interact with the complaint resolution team

4. Dispute Resolution Tools: A range of dispute resolution tools, including:

- Mediation and arbitration services

- Negotiation and settlement tools

- Expert opinion and advice services

***ARCHITECTURE:***

**Frontend:**

**Implement frontend logic:**

* + Integration with API endpoints.
  + Implement data binding.
  + components: Reusable React components used throughout the application.
  + containers: Higher-order components that wrap around other components to provide additional functionality.
  + actions: Redux actions used to manage application state.
  + reducers: Redux reducers used to handle actions and update application state.
  + utils: Utility functions used throughout the application.
  + styles: Global CSS styles used throughout the application.

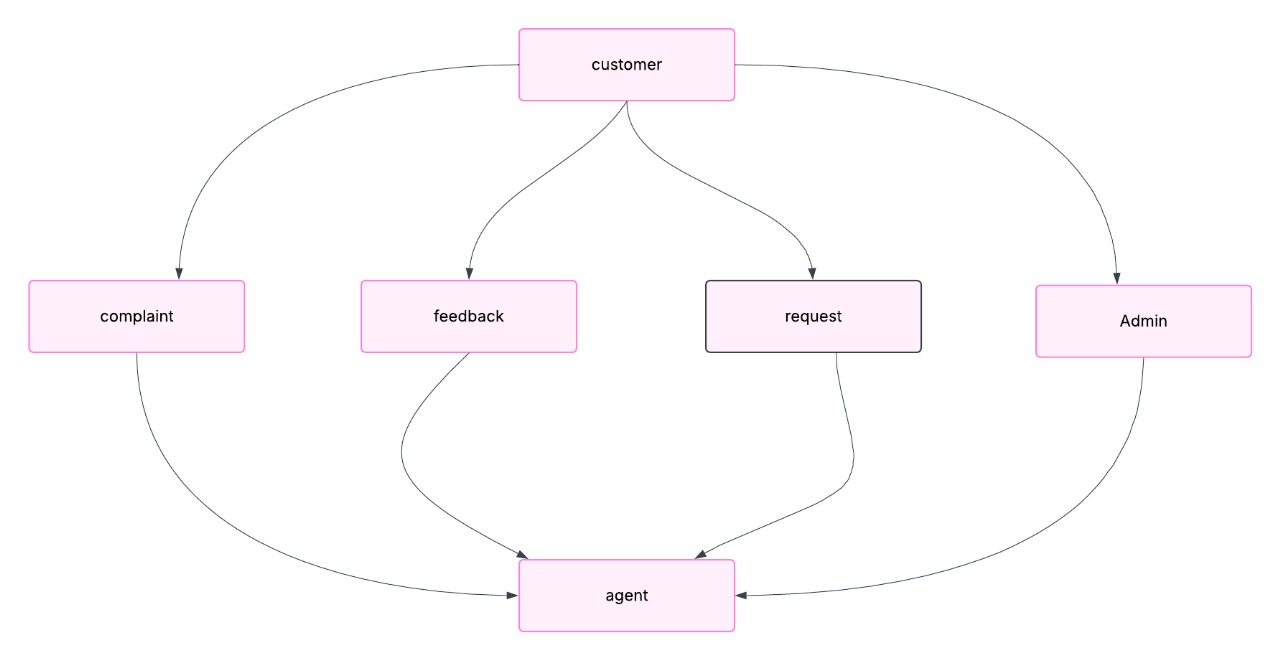
**Backend:**

**Implement backend logic:**

* Node.js as the runtime environment
* Express.js as the web framework
* MongoDB as the NoSQL database management system
* Backend architecture designed for scalability, maintainability, and performance.

The backend of our application is built using Node.js as the runtime environment and Express.js as the web framework, providing a robust and scalable architecture. We utilize MongoDB as our NoSQL database management system, allowing for flexible schema design and efficient data retrieval. To visualize and manage our MongoDB data, we employ Compass, a powerful and intuitive tool that enables us to optimize our database performance and streamline our development workflow. By leveraging this tech stack, we can build a high-performance backend that seamlessly integrates with our frontend, providing a seamless and engaging user experience.

***DATA BASE*:** The database is the central storage component that holds all customer data. It can be a relational database management system (RDBMS) like MySQL or PostgreSQL, a NoSQL database like MongoDB or Cassandra, or a combination of multiple databases depending on the requirements.



**API (Application Programming Interface):** The API serves as an interface for external systems to interact with the customer registry. It exposes a set of endpoints that allow authorized applications or services to perform operations such as creating, updating, retrieving, or deleting customer records. The API handles data validation, authentication, and authorization.

**Services:** Services are responsible for performing specific business logic and operations related to the customer registry. They can include services like customer registration,

authentication, customer profile management, data validation, and more. These services interact with the database and may communicate with each other as needed.

**User Interface:** The user interface component provides an interface for human users to interact with the customer registry system. It can be a web-based application, a mobile app, or even a command-line interface (CLI). The user interface communicates with the

API to perform actions such as searching for customers, updating information, or generating reports.

***SETUP INSTRUCTIONS:***

**Prerequisites and Installations:**

Here are some key prerequisites for building a full-stack expense tracker app: React, Node.js, MongoDB.

**Node.js Setup:**

Install Node.js

1. Go to the Node.js download page: https://nodejs.org/en/download/

2. Click on the "Windows Installer" or "macOS Installer" button, depending on your operating system.

3. Run the installer and follow the prompts to install Node.js.

**Verify Node.js Installation:**

1. Open a terminal or command prompt.

2. Type node -v and press Enter.

3. You should see the version of Node.js installed on your system.

**React-app Setup:**

**Install create-react-app:**

1. Open a terminal or command prompt.

2. Run the following command: npm install -g create-react-app

**Create a New React App:**

1. Run the following command: npx create-react-app my-app (replace "my-app" with your desired app name)

2. Follow the prompts to create a new React app.

**Change into the App Directory:**

1. Run the following command: cd my-app

**Start the Development Server:**

1. Run the following command: npm start

2. Open a web browser and navigate to http://localhost:3000 to see your React app.

**Install Additional Dependencies:**

1. Install React Router: npm install react-router-dom

2. Install Redux: npm install redux react-redux

3. Install other dependencies as needed.

**MongoDB Setup:**

*Install MongoDB Community Server*

1. Go to the MongoDB download page: https://www.mongodb.com/try/download/community

2. Select your operating system and click on the "Download" button.

3. Run the installer and follow the prompts to install MongoDB.

**Start MongoDB Service:**

1. On Windows:

- Open the Services console: services.msc

- Find the "MongoDB Server" service and start it.

2. On macOS (using Homebrew):

- Run brew services start mongodb

3. On Linux:

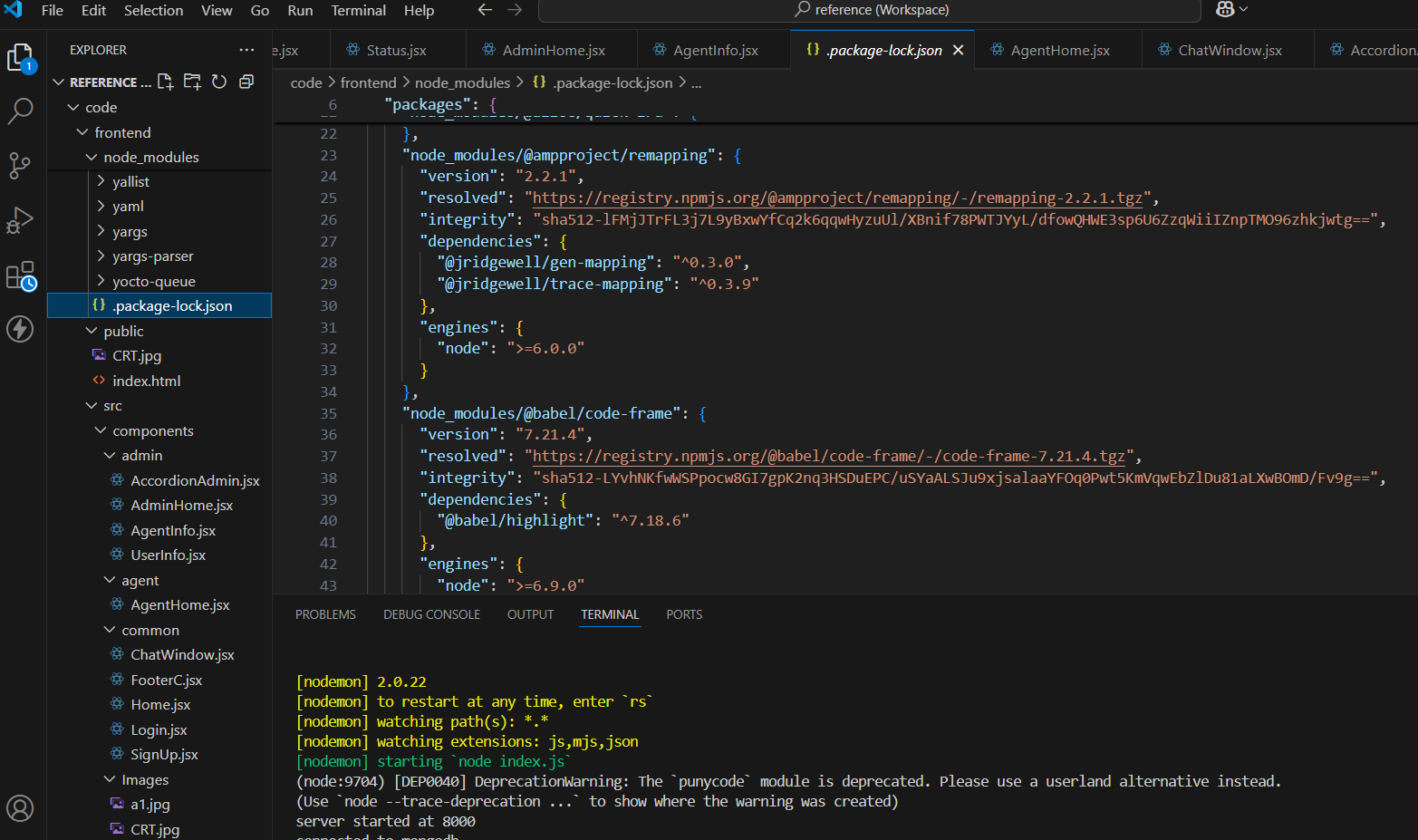
- Run sudo systemctl start mongodb

**Verify MongoDB Installation:**

1. Open a terminal or command prompt.

2. Type mongo --version and press Enter.

3. You should see the version of MongoDB installed on your system.



**FOLDER STRUCTURE FOR RESOLVENOW APP:**

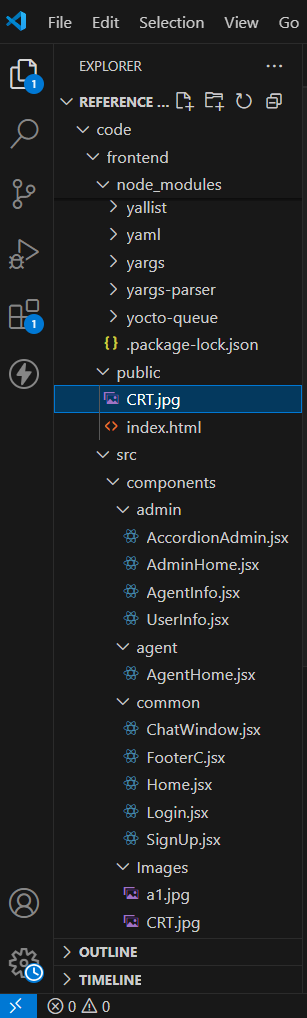
Folder structure separates the client-side and server-side code, with each side having its own subfolders for components, containers, actions, reducers, and utilities. The database and config files are stored in separate folders, and the tests are stored in a separate folder with subfolders for client-side and server-side tests.

**Client side:**

1. public/

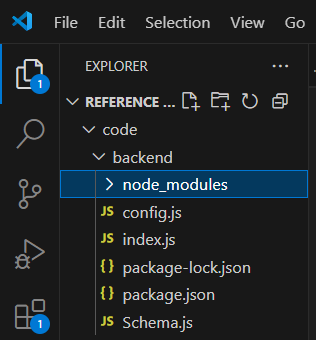
index.html

1. src/
2. components/
3. admin/
4. agent/
5. common/
6. Images/



**Server side:**

1. Config.js/
2. Index.js/
3. Schema.js/



**RUNNING THE APPLICATION FOR RESOLVENOW APP:**

*Commands to start the Frontend and Backend servers:*

**Frontend Server (Client-Side):**

Development Mode: ***npm start*** (starts the development server with hot reloading)

Production Mode: ***npm run build*** (builds the production-ready code) and then ***npm run start*** (starts the production server)

Development Mode with Debugging: ***npm run start:debug*** (starts the development server with debugging enabled)

**Backend Server (Server-Side):**

Development Mode: ***npm run server*** (starts the development server with nodemon)

Production Mode: ***npm run build:server*** (builds the production-ready server code) and then ***npm run start:server*** (starts the production server)

**API DOCUMENTATION:**

*Backend API Endpoints:*

POST /api/auth/login: Login to the application

Request Body: { username: string, password: string }

Response: { token: string, user: object }

POST /api/auth/register: Register a new user

Request Body: { username: string, password: string, email: string }

Response: { token: string, user: object }

GET /api/auth/logout: Logout from the application

Response: { message: string }

*User Endpoints:*

GET /api/users: Get all users

Response: { users: array }

GET /api/users/:id: Get a user by ID

Response: { user: object }

PUT /api/users/:id: Update a user

Request Body: { username: string, email: string }

Response: { user: object }

DELETE /api/users/:id: Delete a user

Response: { message: string }

**Methods and Parameters:**

*Methods for Login:*

POST /api/auth/login

Request Method: POST

Request Body: { username: string, password: string }

Response: { token: string, user: object }

*Methods for Register:*

POST /api/auth/register

Request Method: POST

Request Body: { username: string, password: string, email: string }

Response: { token: string, user: object }

*Methods for Logout:*

GET /api/auth/logout

- Request Method: GET

- Response: { message: string }

**AUTHENTICATION:**

Authentication is the process of verifying the identity of a user. In the ResolveNow app, authentication is handled using JSON Web Tokens (JWT).

**Authorization:**

Authorization is the process of determining what actions a user can perform on a resource. In the ResolveNow app, authorization is handled using role-based access control (RBAC).

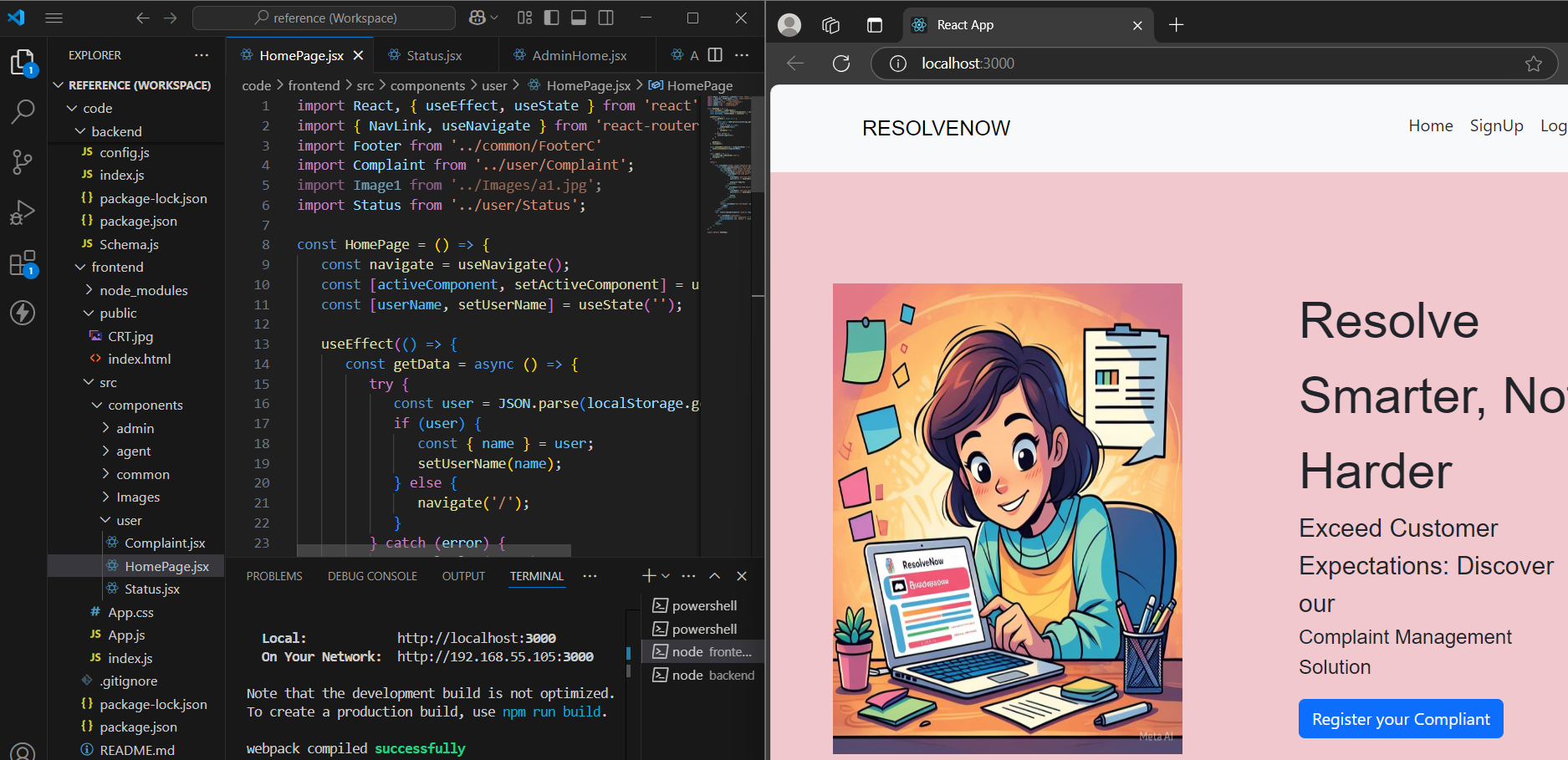
**Tokens and Sessions:**

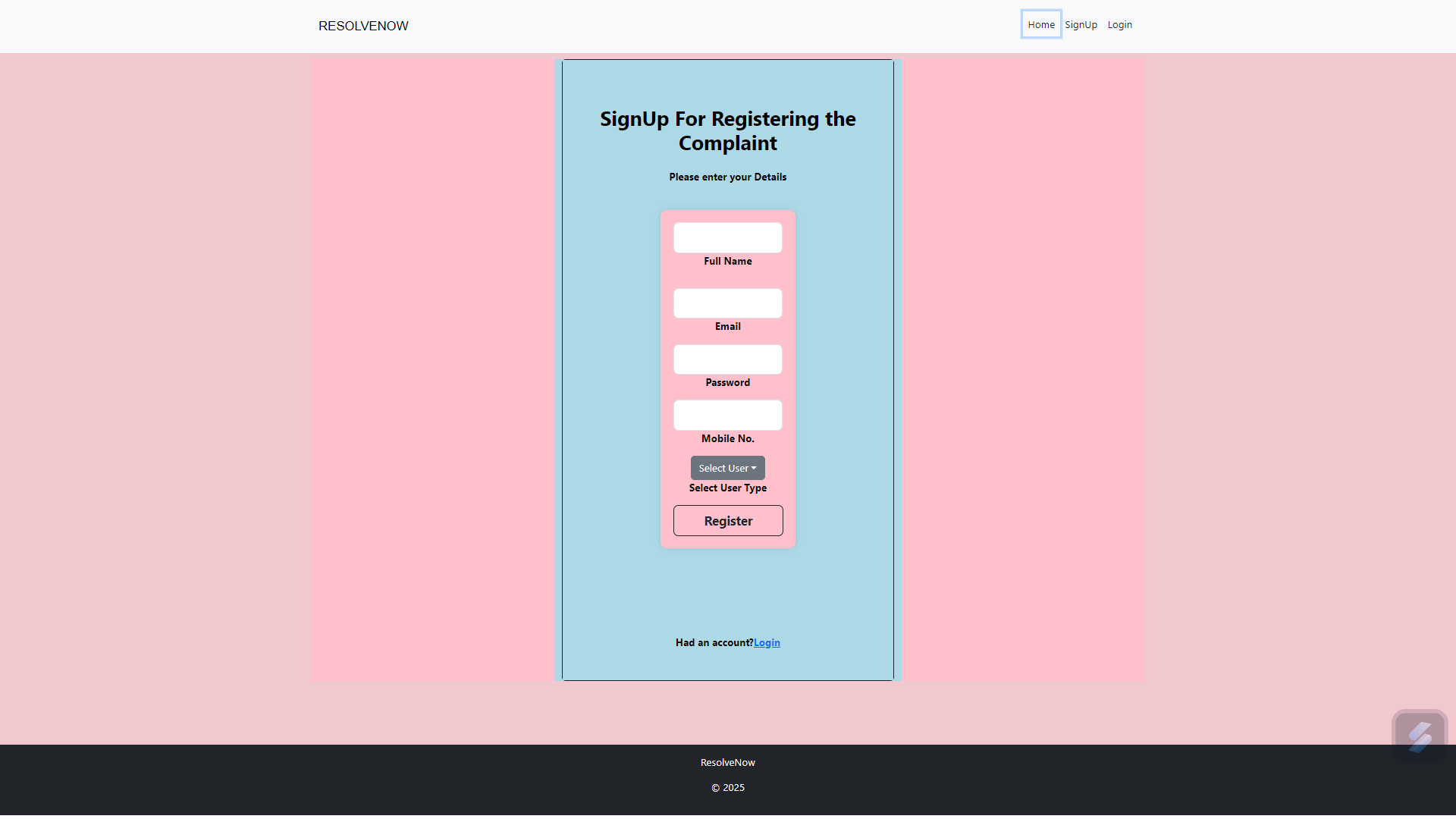
Token-Based Authentication:

The backend API uses token-based authentication to verify the identity of users. Here's how it works:

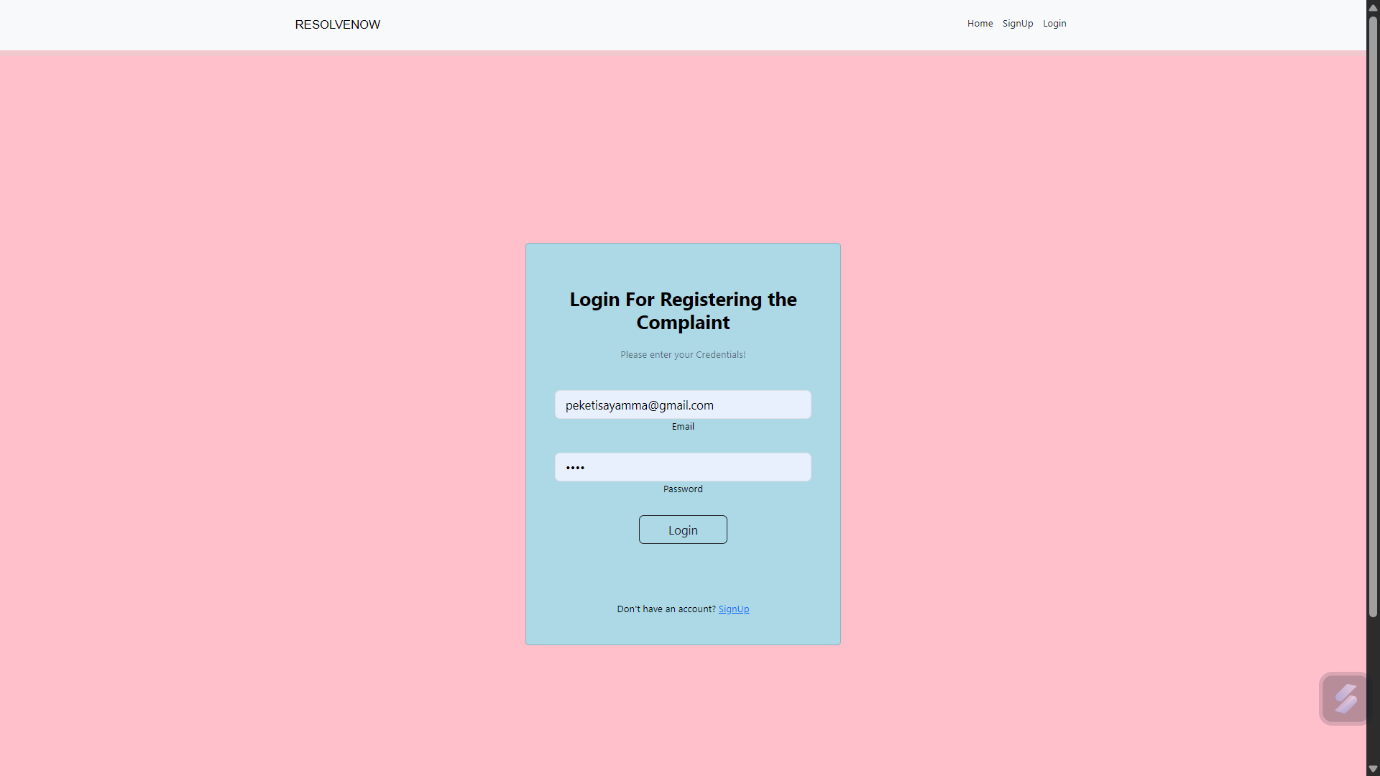
1. Token Generation: When a user logs in, the backend generates a JSON Web Token (JWT) that contains the user's ID, username, and email.
2. Token Verification: On subsequent requests, the client sends the JWT token in the Authorization header. The backend verifies the token by checking its signature and payload.
3. Token Expiration: The JWT token has a limited lifespan (e.g., 1 hour). When the token expires, the client must re-authenticate to obtain a new token.

**USER INTERFACE:**



**Signup Page :**

**Login Page :**

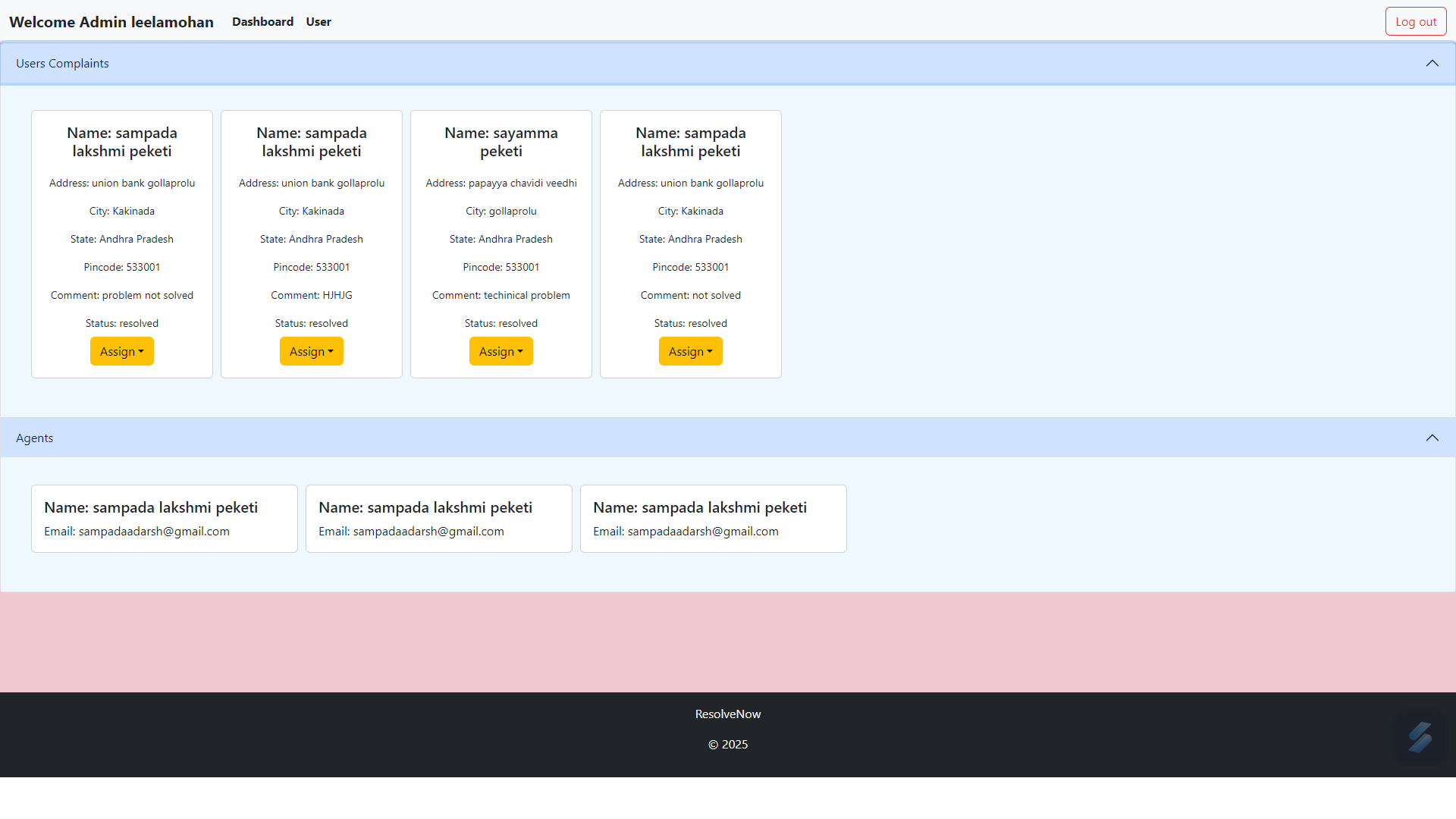


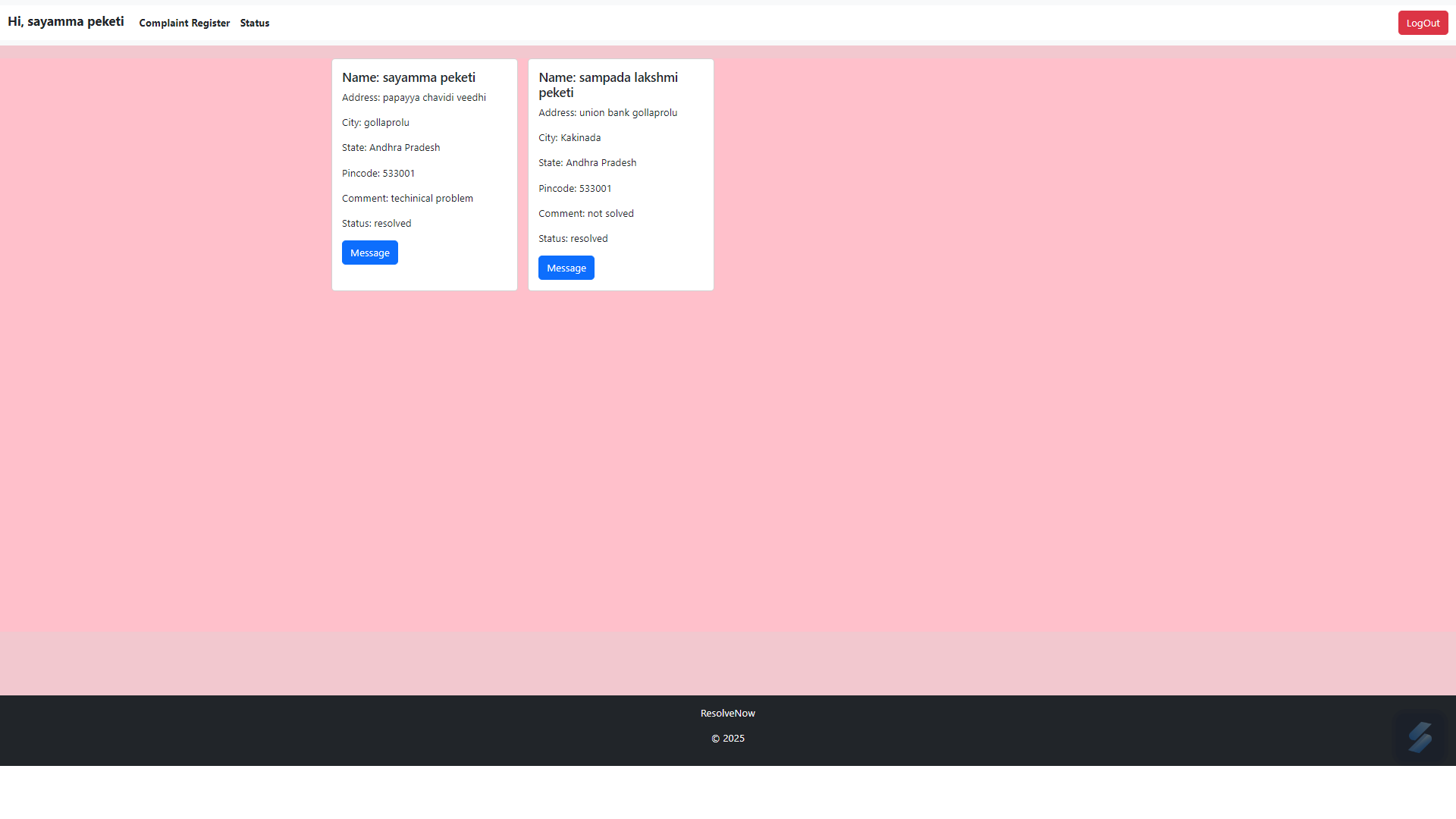
**Testing:**

1. **Frontend Testing (React):**
   * Jest: A delightful JavaScript testing framework by Facebook. Great for unit testing your React components.
   * React Testing Library: Focused on testing React components from the user's perspective.
   * Cypress: Ideal for end-to-end testing with an interactive UI.
2. **Backend Testing (Node.js/Express):**
   * Mocha: A flexible test framework for Node.js, often used with other libraries.
   * Chai: An assertion library that pairs well with Mocha.
   * Supertest: For HTTP assertions and testing your Express APIs.
3. **Database Testing (MongoDB):**
   * MongoDB Memory Server: Enables in-memory testing with a real MongoDB instance, so you don't touch your production data.
   * Mongoose Mock: Mock objects to test Mongoose models and avoid hitting the database.
4. **End-to-End Testing:**
   * Cypress (again, for its full-stack capabilities).
   * Nightwatch.js: Another option for full-stack and browser testing.

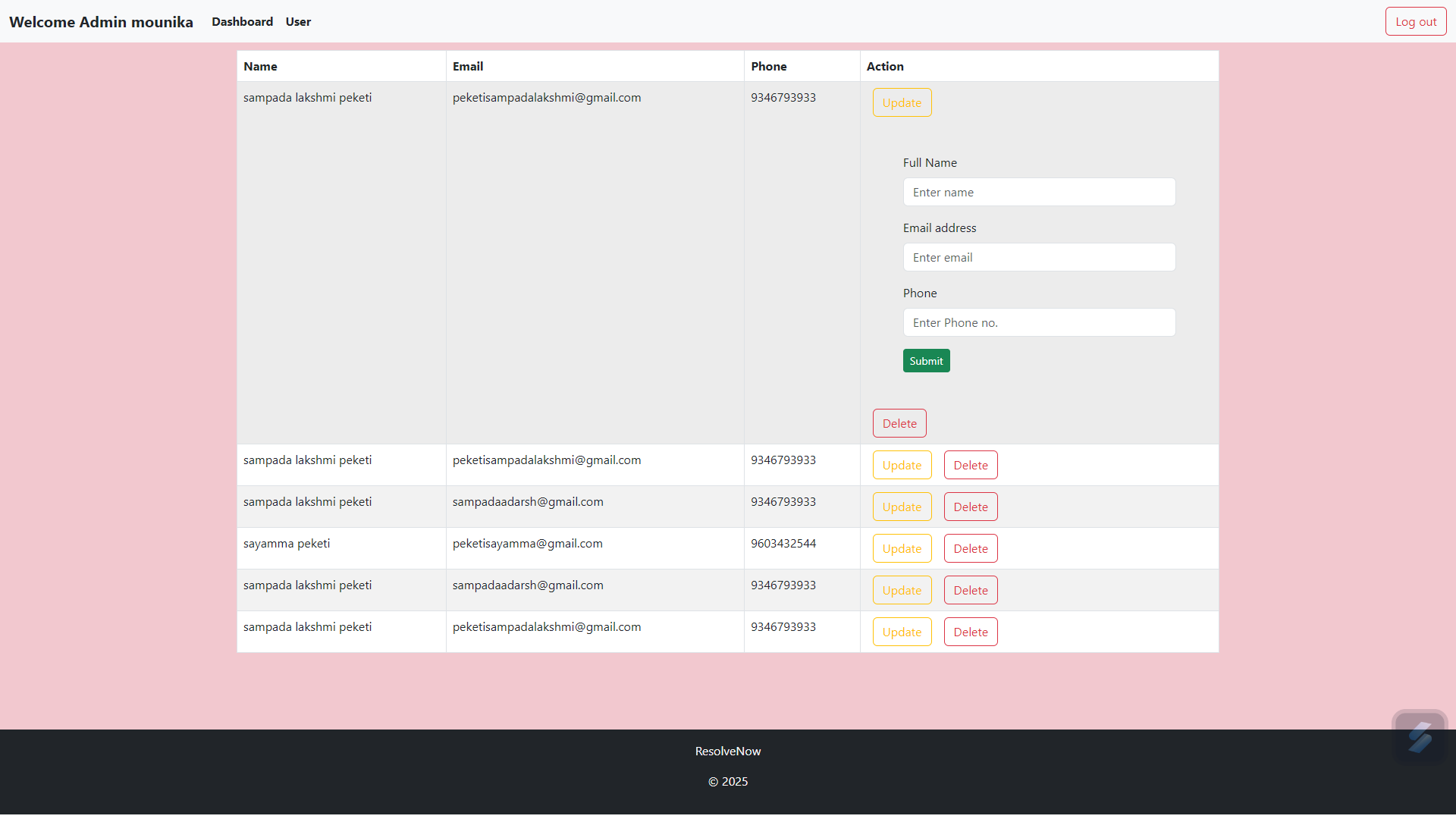
**Testing Strategies**

1. **Unit Testing:** Focus on individual components or functions. Test React components, utility functions, and API routes separately.
2. **Integration Testing**: Test the interaction between different parts of the application (e.g., frontend and backend, backend and database).
3. **End-to-End Testing:** Simulate user workflows to ensure the whole system works as expected. For example, logging in, adding a record, and viewing it.
4. **Mocking**: Use libraries like Jest or Sinon to mock external dependencies (e.g., API calls, database queries).
5. **Continuous Testing:** Integrate your tests with CI/CD pipelines using tools like GitHub Actions, Travis CI, or Jenkins. Automated tests will run whenever you push a change.
6. **Code Coverage:** Use tools like Istanbul or Jest's built-in functionality to measure and ensure high code coverage.



****

**Updating User Profile:**

****

**Known Issues:**

* Compliant gateway integration is under development.
* Occasional lag when loading search result with large data sets.

**Future Enhancements:**

* Integration with multiple online based API’s for real time Data.
* Dynamical handled by load of complaints.
* Advanced filtering based on the complaints.
* User revies and rating for complaints.