Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** ResolveNow: Your Platform for Online Complaints
- **Team Members:**
  1. Yadalla Bhargavi
  2. S Hasifa
  3. Sagala Pavani
  4. Rasipoina Venkata Vinitha
  5. Seelam Aruna Shalini

2. Project Overview

- **Purpose:**

The primary purpose of **ResolveNow** is to provide an efficient, transparent platform for users to file and track complaints, ensuring that issues are addressed promptly by the relevant authorities.

-Centralized Complaint Management: To create a unified system where users can submit detailed grievances (such as road issues or water leakage) along with specific locations and comments.

-Real-Time Tracking: To allow citizens to monitor the status of their complaints—from "Pending" to "Resolved"—directly through their personal dashboard.

-Administrative Oversight: To provide administrators with a clear interface to view, manage, and update the status of all incoming complaints, facilitating better accountability.

-Technological Integration: To implement a full-stack solution using the **MERN** (MongoDB, Express, React, Node.js) stack, ensuring a scalable and responsive user experience.

- **Features:**

-Secure User Authentication: Provides a dedicated registration and login system that validates user credentials against a MongoDB database to ensure secure access to the platform.

-User Complaint Dashboard: A personalized interface where users can view the real-time status of their submitted grievances and track

progress from "Pending" to "Resolved".

-Complaint Submission Portal: A structured form-based system allowing users to file detailed complaints, which are then stored as documents in the ResolveNowDB.

-Admin Control Panel: A specialized management interface for administrators to oversee all incoming complaints, enabling them to update statuses and manage community issues effectively.

-Cross-Origin Communication (CORS): Implementation of CORS middleware to allow the React frontend and Node.js backend to communicate securely across different ports**.**

-Persistent Data Storage: Integration with MongoDB to ensure all user profiles and complaint histories are safely stored and easily retrievable.

3. Architecture

• **Frontend:**

- Component-Based UI: The interface is divided into reusable, independent components (such as Login, Register, and Dashboard), which makes the codebase maintainable and scalable.

-**Single Page Application (SPA)**: By using **React Router**, the app provides a seamless user experience where navigation occurs without full page reloads, making the platform feel fast and responsive.

-**State Management**: React handles the dynamic state of user sessions and complaint data, ensuring that the UI updates instantly when a user logs in or submits a new grievance.

-**API Integration with Axios**: The frontend uses the **Axios** library to perform asynchronous HTTP requests to your Node.js backend, handling the transfer of JSON data for authentication and complaint filing.

-**Form Validation & Handling**: Dedicated logic within components like Register.js and FileComplaint.js ensures that user input is validated before being sent to the server, reducing errors and improving data integrity.

-**Styling and UX**: The architecture likely incorporates a CSS framework (like Material UI or Bootstrap) to ensure the **ResolveNow** dashboard is clean, professional, and accessible across different device sizes.

User Flow in the Frontend

1. **Authentication**: The user interacts with the Login component, which sends credentials to the backend.

2. **Navigation**: Upon success, the user is routed to the Dashboard.

3. **Action**: The user navigates to FileComplaint to submit data, which triggers an update in the global view.

• **Backend:**

-**RESTful API Design**: The backend is organized as a series of REST endpoints (e.g., /api/auth and /api/complaints) that allow the frontend to perform CRUD (Create, Read, Update, Delete) operations.
-**Express.js Routing**: You have used Express to create a modular routing system, specifically authRoutes.js and complaintRoutes.js, which separates authentication logic from complaint management for better maintainability.
-**Asynchronous Processing**: Leveraging Node.js's non-blocking, event-driven architecture, the backend handles multiple user requests—such as concurrent logins or complaint filings—without slowing down the system.
-**Middleware Integration**: The architecture utilizes essential middleware:

- **CORS**: To allow secure communication between your frontend (Port 3000) and backend (Port 5000).
- **express.json()**: To parse incoming JSON request bodies from the frontend.

-Database Interaction via Mongoose: The backend uses the **Mongoose** library to establish a connection to **MongoDB**, providing a schema-based solution to model your application data.
-Error Handling: The backend includes try-catch blocks within the routes to catch and log errors, ensuring that the server doesn't crash if a database connection fails or a user enters invalid data.

Core Backend Flow

1. **Request**: The server listens for HTTP requests on **Port 5000**.
2. **Processing**: Express matches the request to the correct route (like /api/auth/login).
3. **Data Operations**: The logic interacts with MongoDB using Mongoose models to verify users or save new complaints.
4. **Response**: The server sends a JSON response back to the frontend with a success or error status code.

• **Database:**

The data is structured into two primary collections within the ResolveNowDB database, managed via Mongoose schemas in the backend:
-**User Collection**: Stores essential information for authentication and profile management, including fields for name, email, and password.
-**Complaint Collection**: Captures the details of grievances submitted by users. Each document typically includes:

- **Title/Issue Name**: A brief description of the problem.
- **Status**: A string field (defaulting to "Pending") that tracks the lifecycle of the complaint.
- **User Reference**: A link (ObjectId) that associates the complaint

with the specific user who filed it.

Interactions with MongoDB

The interaction between the Node.js server and MongoDB follows a specific workflow to ensure data integrity and performance:

1. **Connection Establishment**: The backend connects to the local MongoDB instance using a URI string (typically mongodb://127.0.0.1:27017/ResolveNowDB) via the Mongoose connect() method.
2. **Data Modeling**: Mongoose schemas act as a blueprint, ensuring that every document saved to the database follows the predefined structure.
3. **CRUD Operations**:
    - **Create**: When a user registers or files a complaint, the backend uses .save() or .create() to insert a new document.
    - **Read**: During login or when viewing the dashboard, the server uses .findOne() or .find() to retrieve data based on user criteria.
    - **Update**: When an admin resolves a complaint, the server uses .findByIdAndUpdate() to change the status from "Pending" to "Resolved".

4. Setup Instructions

• **Prerequisites:**

-**Node.js (v24.12.0 or later)**: The runtime environment required to execute the JavaScript code on the server side.

-**npm (Node Package Manager)**: Used to install and manage the libraries and packages for both frontend and backend.

-**MongoDB Community Server**: To host your local database instance (ResolveNowDB).

-**MongoDB Compass**: A GUI tool to visualize your data and verify that user registrations and complaints are being stored correctly.

• **Installation:**

Step 1: Clone the Repository

Open your terminal or command prompt and run:

Bash

git clone https://github.com/your-username/ResolveNow.git

cd ResolveNow

Step 2: Install Backend Dependencies

Navigate to the backend directory and install the required MERN packages (Express, Mongoose, CORS):

Bash

cd backend

npm install

Step 3: Install Frontend Dependencies

Open a new terminal tab, navigate to the frontend directory, and install the React-related packages (Axios, React-Router):

Bash

cd frontend

npm install

Environment Variable Setup

To connect your backend to the database and manage the server port, you need to configure a .env file.

1. In the backend folder, create a new file named .env.
2. Add the following variables to the file:

Code snippet

PORT=5000

MONGO_URI=mongodb://127.0.0.1:27017/ResolveNowDB

(Note: Using 127.0.0.1 is recommended for better stability on Windows systems.)

Running the Project

Once the setup is complete, you can start the system using these two commands in separate terminal tabs:

-**Backend**: node server.js (Must show "☑ MongoDB Connected!").

-**Frontend:** npm start (Will open the browser at localhost:3000).

5. Folder Structure

- **Client:**

The React frontend is organized to separate the user interface logic from the application's routing and styling.

-**public/**: Contains static assets like the index.html file and the favicon.

-**src/**: The main directory for the React source code.

- **pages/**: Contains the primary view components of the application.

  -Login.js: Handles user authentication.

  -Register.js: Manages new account creation.

  -Dashboard.js: Displays the user's complaint overview.

  -FileComplaint.js: The form used to submit new grievances.

  -MyComplaints.js: Lists the user's personal history of submitted issues.

- **App.js**: The root component that defines the application's navigation and routes.

- **index.js**: The entry point that renders the React app into the DOM.

- **App.css**: Contains global styling and layout rules for the frontend.

- **Server:**

models/: Defines the structure of your data using Mongoose schemas.
-User.js: Schema for storing user names, emails, and passwords.
-Complaint.js: Schema for storing grievance details, statuses, and user links.
routes/: Contains the API endpoints organized by functionality.
-authRoutes.js: Logic for handling login and registration requests.
-complaintRoutes.js: Logic for creating, fetching, and updating complaints.
server.js: The main entry point for the backend. It configures middleware (like CORS), establishes the MongoDB connection, and initializes the server on Port 5000.
.env: A configuration file used to store sensitive data like your MONGO_URI and port numbers.
package.json: Lists all backend dependencies and scripts required to run the Node.js environment.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.

o**Frontend:**

Open a new terminal tab in VS Code so the backend continues to run in the background.

-**Navigate to the frontend directory:**

PowerShell

cd frontend

-**Start the React development server**:

PowerShell

npm start

-**Expected Output**: After a moment, the terminal will show "Compiled successfully" and automatically open your default browser to http://localhost:3000.

o **Backend:**
1. Start the Backend Server
The backend must be initialized first to establish the database connection.
-Navigate to the backend directory:
PowerShell
cd backend
-Run the server:
PowerShell
node server.js
-Expected Output: The terminal should display **"Server is listening on Port 5000"** followed by **"MongoDB Connected!"**.

7. API Documentation

1. Authentication Endpoints

These endpoints handle user access and account creation, interacting directly with the User collection in MongoDB.

A. User Registration

- Method: POST
- Endpoint: /api/auth/register
- Parameters (JSON Body):

    - name: Full name of the user.
    - email: User's email address.
    - password: Secure password.
- Example Response (201 Created):

JSON

```
{
  "message": "User registered successfully",
  "user": { "id": "65cad...", "name": "Bhargavi", "email": "yadalla@gmail.com" }
}
```

B. User Login

- Method: POST
- Endpoint: /api/auth/login
- Parameters (JSON Body):
    - email: Registered email address.
    - password: Registered password.
- Example Response (200 OK):

JSON

```
{
  "message": "Login Successful",
  "user": { "id": "65cad...", "name": "Bhargavi", "email": "yadalla@gmail.com" }
}
```

---

2. Complaint Endpoints

These endpoints manage the lifecycle of grievances submitted through the ResolveNow portal.

A. File a New Complaint

- Method: POST
- Endpoint: /api/complaints/add
- Parameters (JSON Body):
    - name: Title or category of the issue (e.g., "Water Leakage").
    - userId: The ObjectId of the user filing the complaint.
- Example Response (201 Created):

JSON

{

"message": "Complaint Filed Successfully!",

"complaint": { "id": "78db...", "name": "Water Leakage", "status": "Pending" }

}

B. Fetch User Complaints

- Method: GET
- Endpoint: /api/complaints/user/:userId
- Example Response (200 OK):

JSON

[

{ "_id": "78db...", "name": "Water Leakage", "status": "Pending" },

{ "_id": "89ec...", "name": "Road Repair", "status": "Resolved" }

]

C. Update Complaint Status (Admin)

- Method: PUT
- Endpoint: /api/complaints/update/:id
- Parameters (JSON Body):

    - status: The updated state (e.g., "Resolved").

- Example Response (200 OK):

JSON

{ "message": "Complaint Status Updated successfully" }

8. Authentication

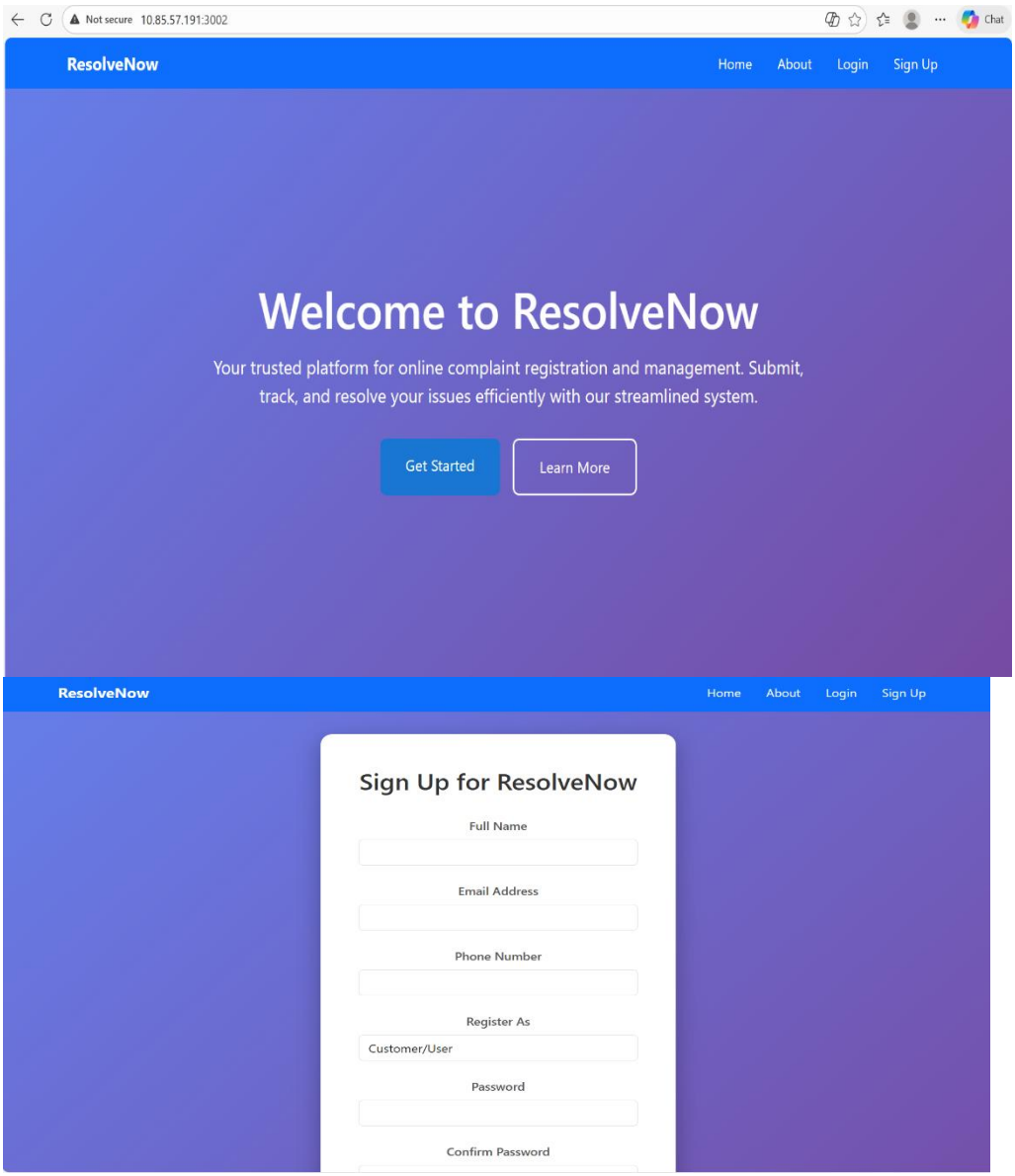
The project handles security through a structured request-response cycle between the React frontend and the Node.js backend:

- **User Registration**: New users submit their details through the Register component, which are then stored in the User collection within **MongoDB**.
- **Credential Verification**: During login, the backend receives the email and password via a POST request to the /api/auth/login endpoint. It then queries the database using Mongoose to find a matching document.
- **Session-Like Persistence**: Once the backend validates the user, it sends a successful JSON response containing the user's basic profile (ID, name, and email). The frontend then stores this data (often in localStorage or React State) to maintain a "logged-in" state across different pages like the Dashboard or MyComplaints.
- **Authorization Control**: Access to specific functionalities is governed by the user's identity. For instance, the **Admin Dashboard** is restricted to administrative accounts, while the standard **User Dashboard** only displays grievances tied to that specific user's userId.

Methods and Technologies Used

- **JSON Communication**: All authentication data is exchanged in JSON format, allowing for seamless integration between the JavaScript-based frontend and backend.
- **Cross-Origin Resource Sharing (CORS)**: The backend uses the cors middleware to authorize requests specifically from localhost:3000, preventing unauthorized external domains from accessing the API.
- **State-Based Tracking**: The frontend utilizes React's state management to track whether a user is authenticated. If a user is not logged in, the App.js router can redirect them back to the login page, protecting internal routes.
- **Secure Data Modeling**: The User schema in MongoDB ensures that every account has a unique email address, preventing duplicate registrations and maintaining data integrity.

9. User Interface



10. Testing

- **Unit Testing**: Individual backend routes, such as those in authRoutes.js, are tested to ensure they correctly validate user credentials and return appropriate status codes (e.g., 200 OK for success or 401 Unauthorized for incorrect passwords).

- **Integration Testing**: Focuses on the "bridge" between the frontend and backend, ensuring that Axios calls from React successfully reach the Express server and that **CORS** allows the data exchange.

- **End-to-End (E2E) Testing**: Simulates a complete user journey: a user registers, logs in, files a complaint, and verifies that the complaint appears on their personal dashboard.

- **Database Verification**: Using **MongoDB Compass** to manually verify that data sent from the frontend is correctly stored as documents within the ResolveNowDB collections.

Testing Tools

To execute this strategy, the following tools and methods are utilized:

- **Postman**: A vital tool for testing backend API endpoints (like POST /api/auth/login) independently of the frontend to ensure the server logic is sound before UI integration.
- **Browser Developer Tools (F12)**: Used extensively to monitor the **Network** tab for failed requests and the **Console** for JavaScript syntax errors or "Server Down" alerts.
- **MongoDB Compass**: Serves as the primary tool for data validation, allowing for a direct look at the users and complaints collections to ensure schemas are being followed.
- **React Error Overlay**: Provides immediate feedback during frontend development, highlighting syntax errors or broken components in the browser.

Testing Scenarios

Scenario	Expected Result
Login with wrong password	Backend returns 401; Frontend shows "Invalid Credentials".
File a complaint without login	System redirects user to the Login page.
Successful Database Connection	Terminal logs "☑ MongoDB Connected!".
Form Submission	New document appears in MongoDB complaints collection.

11. Screenshots or Demo

http://10.85.57.191:3002

12. Known Issues

-Backend and frontend must run in separate terminals simultaneously

-The node server.js command fails if run outside the backend folder

-Missing or extra braces in the code will crash the Node.js server

-Database connection via localhost can be unstable on Windows systems

-User login state is currently lost when the browser page is refreshed

-The previous web scraping feature was removed due to implementation difficulties

13. Future Enhancements

None