Indian Citizen Complaint Box

Team members ROLES

Piyush vyas 22BAI1028

MITHUNRAJ

MOHITH GK

SPOORTHI N BHAT

**Project Overview**

Purpose:

The primary goal of this project is to develop a comprehensive public grievance resolving application for Indian citizens using the MERN stack (MongoDB, Express.js, React.js, Node.js). This platform aims to provide a streamlined and efficient way for individuals to raise complaints related to governmental works. By facilitating better communication between the public and government officials, the application seeks to ensure that grievances are addressed promptly and transparently, thereby improving public service delivery and citizen satisfaction.

Features:

1. Home Page:

- Login and Signup: Users can create a new account or log in to their existing account.

2. Tutorial Page:

- How to Use: A step-by-step guide to help users understand how to navigate and use the app effectively.

3. About Page:

- Information: Details about the purpose, goals, and benefits of the application.

4. FAQ Page:

- Common Questions: Answers to frequently asked questions to assist users with common issues and inquiries.

5. User Dashboard:

- Lodge a New Complaint:

- Form: Users can fill out a form to submit a new complaint, including demographic details, images, and comments.

- View All Complaints:

- Log: Users can view a log of all complaints they have submitted.

- Track Complaint Status:

- Status Updates: Users can track the status of their complaints to see progress and updates.

- Feedback:

- Submit Feedback: Users can provide feedback on the complaint resolution process and their overall experience with the app.

**Architecture**

Frontend:

The frontend architecture is built using React.js, divided into multiple functional components to ensure modularity and reusability. Each page and UI element is encapsulated within its own component, styled using CSS for consistency and responsiveness. The primary routing is managed through `App.js`, which uses React Router to navigate between different pages.

- Components: Individual pages (Home, Tutorial, About, FAQ, Dashboard, etc.) are structured as separate functional components.

- Styling: CSS is used for styling the components, ensuring a cohesive and user-friendly interface.

- Routing: React Router is implemented in `App.js` to handle the navigation between different components.

Backend:

The backend architecture is structured using Node.js and Express.js, providing a robust and scalable server-side framework. The backend is responsible for handling API requests, authentication, and business logic.

- Models:

- User Model: Defines the schema for storing and retrieving user data for login/signup.

- Complaint Model: Manages the schema for storing details of complaints.

- Feedback Model: Handles the schema for storing user feedback.

- Server:

- Express.js: Creates the server and manages API endpoints.

- Middleware: Implements middleware for handling requests, authentication, and error handling.

- Routes:

- User Routes: Manages endpoints for user-related actions (login, signup).

- Complaint Routes: Handles endpoints for lodging, viewing, and tracking complaints.

- Feedback Routes: Manages endpoints for submitting and retrieving feedback.

Database:

The database schema is designed using MongoDB, a NoSQL database known for its flexibility and scalability. Three primary collections are utilized:

- User Collection: Stores user data, including login credentials and profile information.

Sample document 

- Complaint Collection: Contains detailed information about each complaint, including demographic details, images, and comments.

Sample document

{"\_id":{"$oid":"668f88d3e799a94cf0a7e9d4"},"category":"Road Issues","othercategory":"","state":"Tamil Nadu","district":"Chennai","pincode":"600001","comment":"Road is heavily damaged with lot of potholes","photo":"uploads\\1720682707422-roadblock.jpg","video":"uploads\\1720682707424-road block.mp4","\_\_v":{"$numberInt":"0"}}

- Feedback Collection: Records user feedback regarding the complaint resolution process and overall experience.

Interactions:

- CRUD Operations: The backend server interacts with MongoDB to perform Create, Read, Update, and Delete (CRUD) operations on the collections.

- Mongoose: An Object Data Modeling (ODM) library for MongoDB and Node.js, is used to define schemas and interact with the database.

- Authentication: User authentication is managed through secure password hashing and token-based authentication (e.g., JWT).

This architecture ensures a seamless and efficient user experience, while also providing a robust and scalable backend to handle various functionalities related to grievance resolution.

**Setup Instructions**

Prerequisites:

Before setting up the project, ensure you have the following software dependencies installed on your machine:

1. Node.js: Download and install from [nodejs.org](https://nodejs.org/).

2. MongoDB: Download and install MongoDB from [mongodb.com](https://www.mongodb.com/).

3. Git: Download and install Git from [git-scm.com](https://git-scm.com/).

4. npm: Node.js comes with npm (Node Package Manager) installed. Verify by running `npm -v` in the terminal.

5. IDE/Text Editor: (Optional) Use an IDE or text editor like Visual Studio Code for code editing.

Installation:

Follow these step-by-step instructions to set up the project:

1. Clone the Repository:

Open your terminal and run the following command to clone the project repository:

```sh

git clone <repository-url>

```

Replace `<repository-url>` with the actual URL of your project's Git repository.

2. Navigate to Project Directory:

```sh

cd <project-directory>

```

Replace `<project-directory>` with the name of the cloned repository folder.

3. Install Frontend Dependencies:

Navigate to the frontend directory and install the necessary packages:

```sh

cd client

npm install

```

4. Install Backend Dependencies:

Navigate to the backend directory and install the necessary packages:

```sh

cd ../server

npm install

```

5. Set Up Environment Variables:

Create a `.env` file in the `server` directory and add the following environment variables:

```env

PORT=5000

MONGODB\_URI=<your-mongodb-connection-string>

JWT\_SECRET=<your-jwt-secret>

```

6. Start the MongoDB Server:

Ensure that MongoDB is running on your local machine or use a cloud-based MongoDB service.

7. Run the Backend Server:

Start the backend server by running the following command in the `server` directory:

```sh

npm start

```

8. Run the Frontend Development Server:

Open a new terminal window, navigate to the `client` directory, and start the frontend development server:

```sh

cd client

npm start

```

9. Access the Application:

Open your web browser and navigate to `http://localhost:3000` to access the application.

These steps should help you set up and run the public grievance resolving application on your local machine.

**Folder Structure**

Client:

The React frontend is structured to maintain modularity and separation of concerns, ensuring ease of development and maintenance. Here is an overview of the folder structure:

Public

- favicon.ico: Icon for the web application.

- index.html: Main HTML file for the React application.

- logo192.png & logo512.png: Logo images used in the application.

- manifest.json: Configuration file for the web application, including metadata.

- robots.txt: Instructions for web crawlers.

Src

The `src` folder contains all the source code for the React application.

1. Components:

- This folder contains reusable React components that are used across various pages.

- App.css: CSS file for styling the `App` component.

- App.js: The main component that sets up routing and renders other components.

- App.test.js: Test file for the `App` component.

- index.css: Global CSS styles.

- index.js: Entry point of the React application.

- logo.svg: Logo used in the application.

- reportWebVitals.js: Utility for measuring performance metrics.

2. Pages:

- This folder contains individual pages and their respective components.

- Assets: Contains static assets such as icons and images.

- Icons: Folder for storing icon files.

- Images: Folder for storing image files.

- `Faqwaves.png`

- `Frame 101.png`

- `Frame 152.png`

- `Frame 154.png`

- `Frame 42.png`

- `Login.jpg`

- `Rectangle 273.png`

- `Resolution.jpg`

- `Submit.jpg`

- `Track.png`

- `Update.jpg`

3. Page Components:

- ComplaintRegister.js: Component for lodging a new complaint.

- ComplaintView.css: CSS file for styling the complaint view component.

- ComplaintView.js: Component for viewing complaints.

- Content.css: CSS file for styling content.

- Dashboard.js: Main dashboard component.

- FAQ.js: FAQ page component.

- FeedbackForm.css: CSS file for styling the feedback form.

- FeedbackForm.js: Component for submitting feedback.

- Home.js: Home page component.

- LoginMainPage.css: CSS file for styling the login page.

- LoginMainPage.js: Login page component.

- Timeline.css: CSS file for styling the timeline component.

- Timeline.js: Component for displaying the complaint timeline.

- TrackComplaint.css: CSS file for styling the track complaint component.

- TrackComplaints.js: Component for tracking the status of complaints.

- Tutorial.js: Tutorial page component.

- db.css: CSS file for styling database-related components.

- faq.css: CSS file for styling the FAQ page.

This structured approach ensures that components and assets are organized logically, making the codebase easier to navigate and maintain.

**Node.js Backend Structure**

1. **Root Directory**:
   * backend: Main folder for backend code.
   * node\_modules: Contains project dependencies.
   * .gitignore: Specifies files ignored by Git.
   * package.json: Metadata and dependencies.
   * server.js: Entry point for Node.js server.
2. **Subfolders**:
   * models: Contains data models (e.g., database schemas). Includes UserModel.js.
   * routes: Handles routing logic. Includes Login.js, Signup.js, and complaints.js.

**Additional Files**:

* Login.js: Manages login-related routes.
* Signup.js: Handles user registration.
* complaints.js: Routes for user complaints.
* uploads: Possibly for file uploads

**Running the Application**

* **Frontend:** To start the frontend server locally, navigate to the client directory and run:
* sh
* Copy code
* cd client
* npm start

**Backend:** To start the backend server locally, navigate to the server directory and run:

* sh
* Copy code
* cd server
* npm start

**API Documentation**

* The backend exposes several endpoints to handle user authentication, complaint management, and feedback submission. Below is the documentation for these endpoints.

# API Documentation

## Complaints Endpoints

### 1. Create Complaint

* **Endpoint**: /complaints
* **Method**: POST
* **Description**: Registers a new complaint with optional photo and video attachments.
* **Parameters**:
  + category (string, required)
  + othercategory (string, optional)
  + state (string, required)
  + district (string, required)
  + pincode (string, required)
  + comment (string, required)
  + photo (file, optional)
  + video (file, optional)
  + phoneId (string, required)
  + date (string, required)
* **Example Request**:

json

Copy code

{

"category": "Infrastructure",

"othercategory": "",

"state": "Maharashtra",

"district": "Pune",

"pincode": "411001",

"comment": "Roads are in bad condition",

"phoneId": "12345",

"date": "2024-07-17"

}

* **Example Response**:

json

Copy code

{

"message": "Complaint registered successfully"

}

### 2. Delete Complaint

* **Endpoint**: /complaints/:id
* **Method**: DELETE
* **Description**: Deletes a complaint by its ID.
* **Parameters**:
  + id (string, required) - ID of the complaint to delete
* **Example Request**:

bash

Copy code

DELETE /complaints/60c72b1f4f1a4c001cb3eeb1

* **Example Response**:

json

Copy code

{

"message": "Complaint deleted successfully",

"deletedComplaint": { /\* complaint data \*/ }

}

### 3. Increment Likes

* **Endpoint**: /complaints/:id/likes
* **Method**: PUT
* **Description**: Increments the like count of a complaint by its ID.
* **Parameters**:
  + id (string, required) - ID of the complaint to like
* **Example Request**:

bash

Copy code

PUT /complaints/60c72b1f4f1a4c001cb3eeb1/likes

* **Example Response**:

json

Copy code

{

"message": "incrementd like",

"updatedLike": { /\* updated complaint data \*/ }

}

### 4. Update Status

* **Endpoint**: /complaints/:id/status
* **Method**: PUT
* **Description**: Updates the status of a complaint by its ID.
* **Parameters**:
  + id (string, required) - ID of the complaint
  + status (string, required) - New status of the complaint
* **Example Request**:

json

Copy code

{

"status": "Resolved"

}

* **Example Response**:

json

Copy code

{

"message": "Updation successful",

"updatedStatus": { /\* updated complaint data \*/ }

}

### 5. Fetch Complaints

* **Endpoint**: /complaints
* **Method**: GET
* **Description**: Fetches all complaints.
* **Example Request**:

bash

Copy code

GET /complaints

* **Example Response**:

json

Copy code

[

{ /\* complaint data \*/ },

{ /\* complaint data \*/ }

]

## User Endpoints

### 1. Register User

* **Endpoint**: /signup
* **Method**: POST
* **Description**: Registers a new user.
* **Parameters**:
  + firstName (string, required)
  + lastName (string, required)
  + email (string, required)
  + phone (string, required)
  + dob (string, required)
  + password (string, required)
* **Example Request**:

json

Copy code

{

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"dob": "1990-01-01",

"password": "password123"

}

* **Example Response**:

json

Copy code

{

"message": "User registered successfully"

}

### 2. User Login

* **Endpoint**: /login
* **Method**: POST
* **Description**: Logs in a user.
* **Parameters**:
  + phone (string, required)
  + password (string, required)
* **Example Request**:

json

Copy code

{

"phone": "1234567890",

"password": "password123"

}

* **Example Response**:

json

Copy code

{

"message": "Login successful"

}

## Feedback Endpoints

### 1. Submit Feedback

* **Endpoint**: /feedback
* **Method**: POST
* **Description**: Submits feedback.
* **Parameters**:
  + name (string, required)
  + email (string, required)
  + stars (number, required)
  + comments (string, required)
* **Example Request**:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com",

"stars": 5,

"comments": "Great service!"

}

* **Example Response**:

json

Copy code

{

"message": "Feedback Registered"

}

This documentation provides a clear and concise overview of each endpoint, including their methods, descriptions, parameters, and example requests and responses.

**Future Enhancement –**

Integration with aadhar api to validate user profile

Live complaint tracking with response of every progress of resolve by the admin

User can Appeal with high authority if the result are not satisfying

Data analysis according to geospatial region to find possible course correction in respective departments