Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** ResolveRadar – Online Complaint and Management System
- **Team Members:**
  - Apzana Nizam – Frontend and Project documentation
  - Bhuvana Prabha B – Backend and Project documentation
  - Upasana L – Frontend and Backend
  - Vasish Pareekshith S S – Frontend and Backend

2. Project Overview

Purpose: The Online Complaint Registration and Management System is designed to streamline the process of submitting, managing, and resolving complaints or issues. It aims to provide a centralized platform where users can easily register their complaints, track their progress, and communicate with the responsible agents. The system seeks to enhance the efficiency of complaint handling, ensure timely resolutions, and ultimately improve customer satisfaction by offering a transparent and structured approach to managing complaints.

- **Features:**

1. User Registration:

- Secure account creation with authentication methods.

- Identity verification to ensure the authenticity of complaints.

2. Complaint Submission:

- Detailed complaint submissions with descriptions.

- Categorization of complaints for organized management and routing.

3. Tracking and Notification:

- "My Complaints" dashboard section for tracking complaint status.

- Automatic updates and notifications on complaint progress.

4. Security and Confidentiality:

- Strong authentication mechanisms for user verification.

- Data encryption to protect user information and complaint details.

- Access controls to ensure only authorized personnel handle complaints.

5. Resolution and Feedback:

- Use feedback to enhance the system and complaint handling.

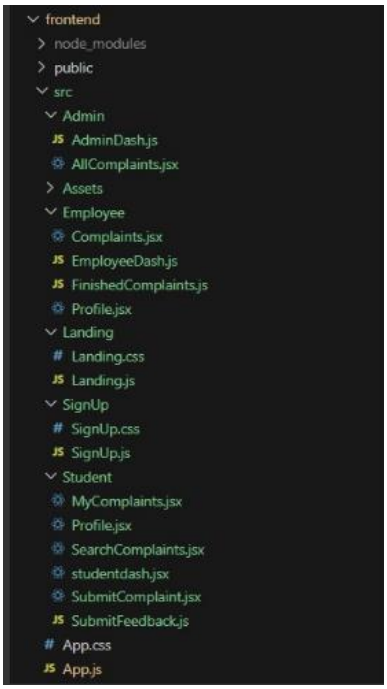
6. Admin Management Model:

- Admins monitor all complaints and assign them to agents based on workload and expertise.

3. Architecture

- **Frontend:**

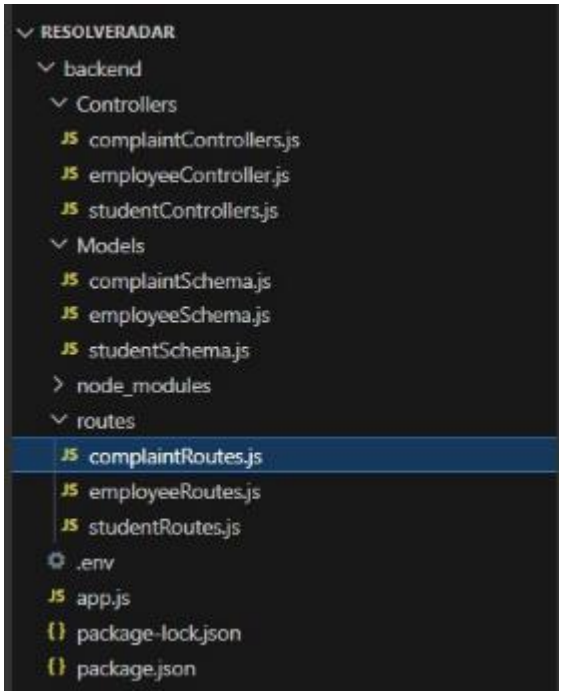
  The frontend architecture ensures a clean separation of concerns, making the application scalable and maintainable. Each user role (student, employee, admin) has its dedicated dashboard with specific functionalities, enhancing the user experience and managing the application's complexity efficiently. Here is the simple outline of our architecture.



- **Routing**: Managed by `react-router-dom` in `App.js`.
- **State Management**: Local state management using `useState` for handling dynamic data like complaints.
- **Component-Based Design**: Each dashboard and functionality is encapsulated in separate components.
- **Styling**: Utilizes `bootstrap/dist/css/bootstrap.min.css` for consistent and responsive styling.

- **Backend:**

The backend architecture for the Online Complaint Registration and Management System is built using Node.js and Express.js, with MongoDB as the database. It includes separate routes and controllers for handling complaints from students, employees, and admins. The main features include user authentication, complaint management, and secure data handling.



1. **Server Setup (`app.js`)**
    - **Express App**: Initializes the Express application.
    - **Middleware**: Includes CORS, JSON parsing, and custom error handling.
    - **MongoDB Connection**: Connects to the MongoDB database using Mongoose.
    - **Routes**: Sets up routes for students, complaints, and employees.
2. **Database (MongoDB)** – Mongo DB is used for the database
3. **Routes**
    - Separate route files for different functionalities, e.g., `studentRoutes`, `complaintRoutes`, `employeeRoutes`.
    - Each route file imports corresponding controllers and defines the endpoints.
4. **Controllers**
    - Controllers handle the logic for each route, interacting with the database and processing the requests.

- Examples: Creating a complaint, fetching all complaints, updating complaint status, deleting a complaint.

- **Database:**

The backend uses MongoDB for data storage, managed via Mongoose, which provides a clear way to define and interact with MongoDB data models. The database schema includes collections for complaints, students, and employees.

Complaint Schema

backend > Models > JS complaintSchema.js > ...

```
const mongoose = require('mongoose');


const complaintSchema = new mongoose.Schema({
    complaintText: { type: String, required: true },
    category: { type: String, required: true },
    date: { type: Date, default: () => new Date().toISOString().split('T')[0] },
    studentId: { type: String, required: true },
    status: {type:String, default:'Pending'}  // Assuming studentId is a string identifier
}, { versionKey: false });

const Complaint = mongoose.model('Complaint', complaintSchema);

module.exports = Complaint;

```

Student Schema

backend > Models > JS studentSchema.js > ...

```
const mongoose = require('mongoose');
// Define the Student Schema
const studentSchema = new mongoose.Schema({
    studentId: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    // complaints: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Complaint' }] // Array of complaint
}, { versionKey: false });
studentSchema.index({ roll: 1 }, { sparse: true });
// Create and export the Student model
const Student = mongoose.model('Student', studentSchema);

module.exports = Student;

```

Employee Schema

backend > Models > JS employeeSchema.js > ...

```
// models/employeeSchema.js

const mongoose = require('mongoose');

const employeeSchema = new mongoose.Schema({
  employeeId: {type: String, required: true, unique: true},
  email: {type: String, required: true, unique: true},
  password: {type: String, required: true,},
  // complaints: [{
  //   type: mongoose.Schema.Types.ObjectId,
  //   ref: 'Complaint',
  // }],
}, {versionKey:false} );
employeeSchema.index({ roll: 1 }, { sparse: true });
const Employee = mongoose.model('Employee', employeeSchema);
module.exports = Employee;


//{ timestamps: true }
```

The CRUD Operations performed

- - Create: Insert new documents into the `complaints`, `students`, or `employees` collections using Mongoose models.
- - Read: Queries can be based on fields like `studentId`, `EmployeeId`, or specific `complaint ID`s.
- - Update: For example, updating the status of a complaint or changing employee/student details.
- - Delete: deleting complaints or removing complaints from a student's or employee's list.
-

For example:

- - Create a Complaint: Instantiate a new Complaint document and save it to the `complaints` collection.
- - Fetch All Complaints: Retrieve all documents from the `complaints` collection.
- - Delete a Complaint: Find a complaint by its ID and remove it from the `complaints` collection.
-

Relationships and Referencing of the Documents:

- - Complaints and Students: Complaints are associated with students via the `studentId` field. This allows tracking of which student made each complaint.
- - Complaints and Employees: Employees are assigned complaints through their `assignedComplaints` field, linking them to specific complaints.
-

Error Handling and Validation

- - Validation: Mongoose schemas enforce data types and required fields, ensuring data integrity.
- - Error Handling: Errors during database operations are caught and handled, ensuring that the application can respond efficiently to issues like validation errors or connection problems.

4. Setup Instructions

- **Prerequisites:** The software dependencies required are Node.js, Express.js, MongoDB, Mongoose, CORS, Bcryptjs and .env configuration file.
-
- **Installation:**

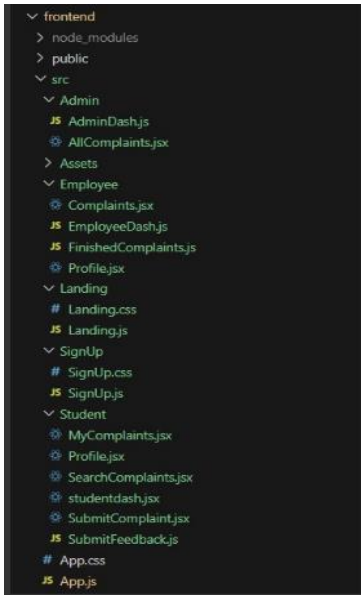
  Frontend:

  - `npm install dotenv` to install the dotenv package in order to load the required environment variables
  - `npx create-react-app .` in the current frontend directory to install the required dependencies and node modules for the react environment
  - `npm install react-bootstrap axios react-router-dom` to install the required modules used in the application.

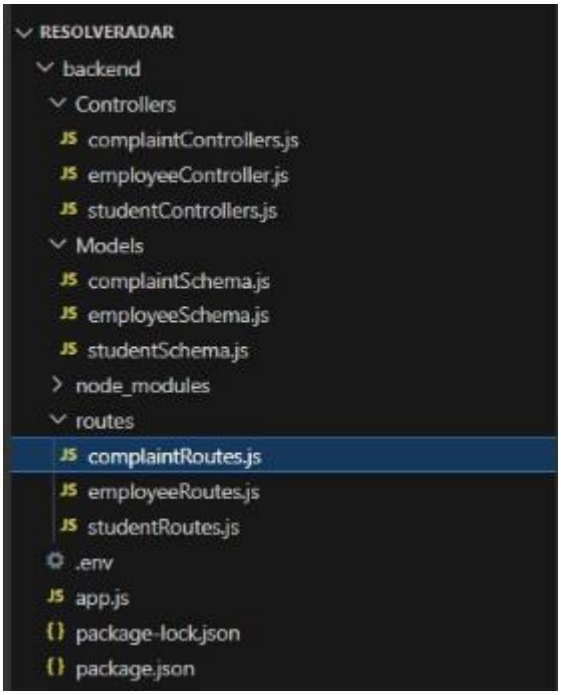
  Backend:

  - `npm install express` to install the express framework.
  - `npm install cors mongoose bcrypt` for installing the required middleware(cors), library(mongoose, bcrypt).

5. Folder Structure

- **Client:** structure of the React frontend.
- 

- **Server:** The organization of the Node.js backend.



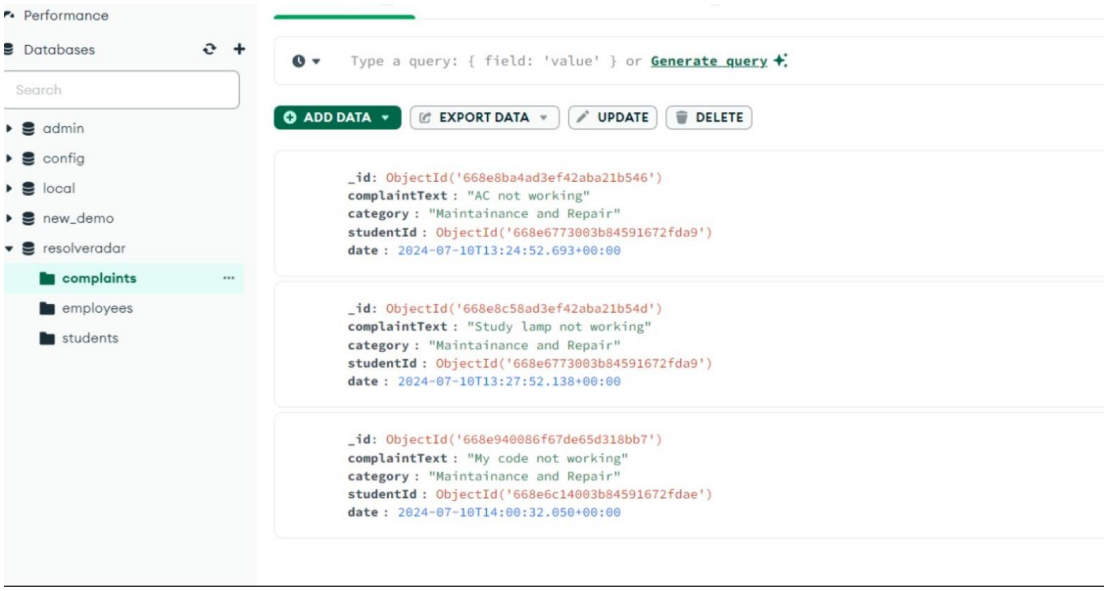
(MongoDB database and collections structure)

6. Running the Application

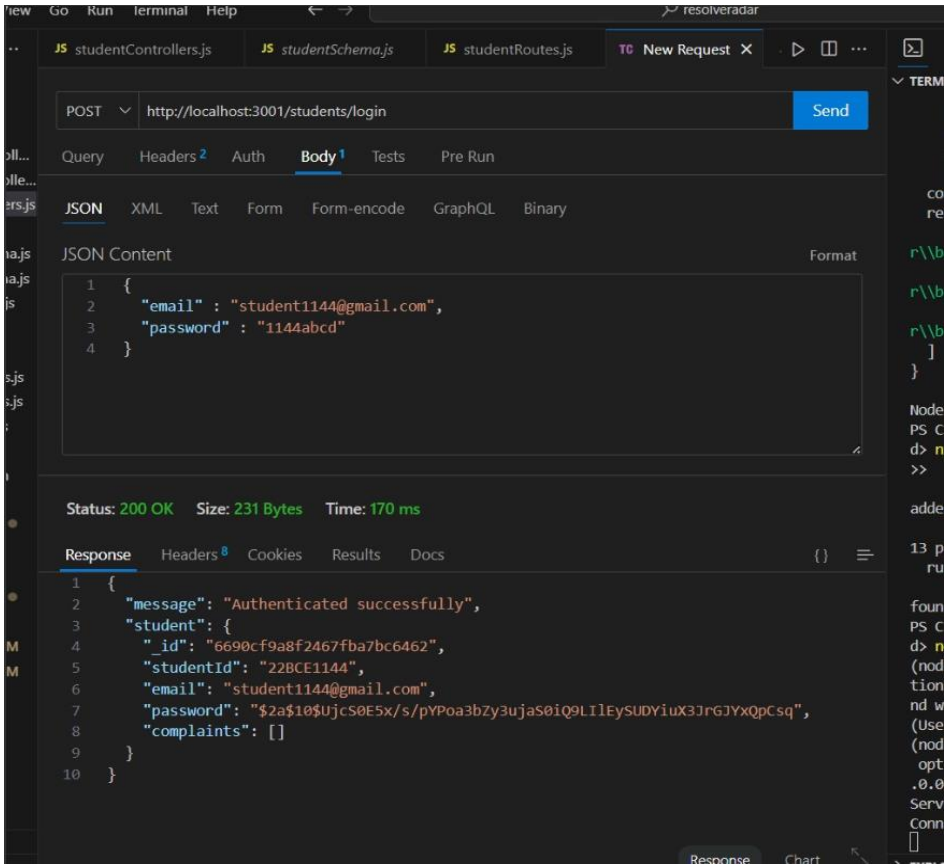
- Provide commands to start the frontend and backend servers locally.
  - **Frontend:**
  - `npm start` to start the react environment
-
  - **Backend:**
  - `node app.js` to start the server at the local port

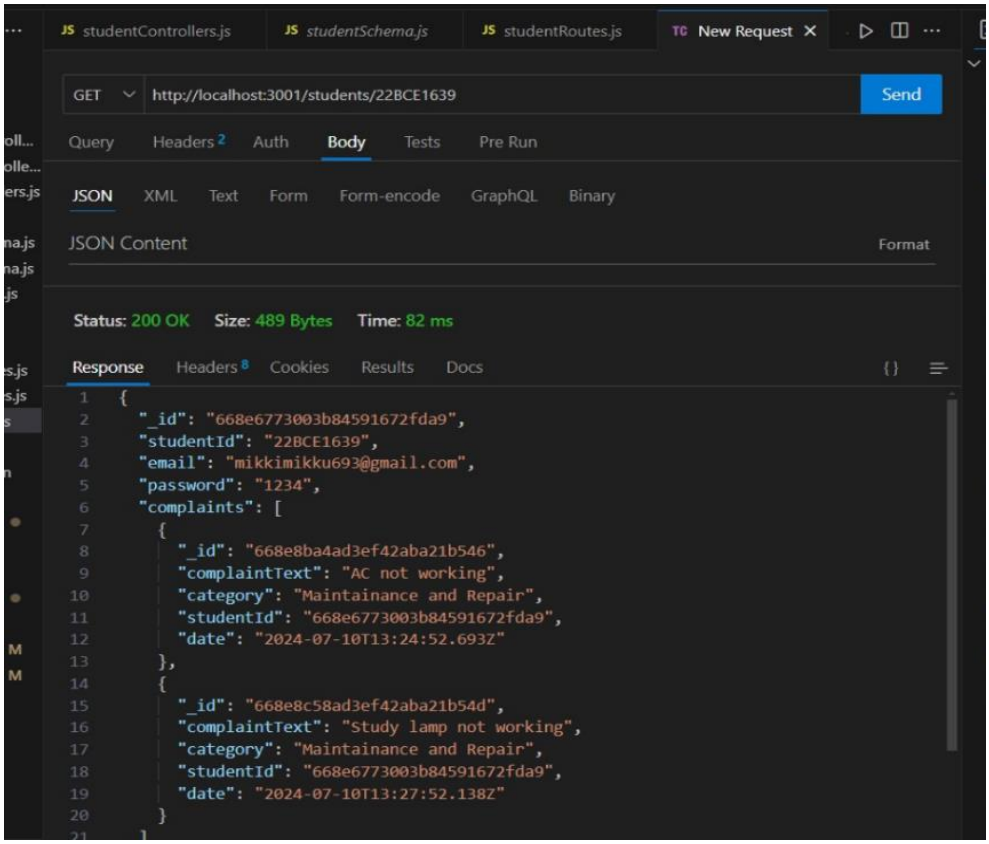
7. API Documentation

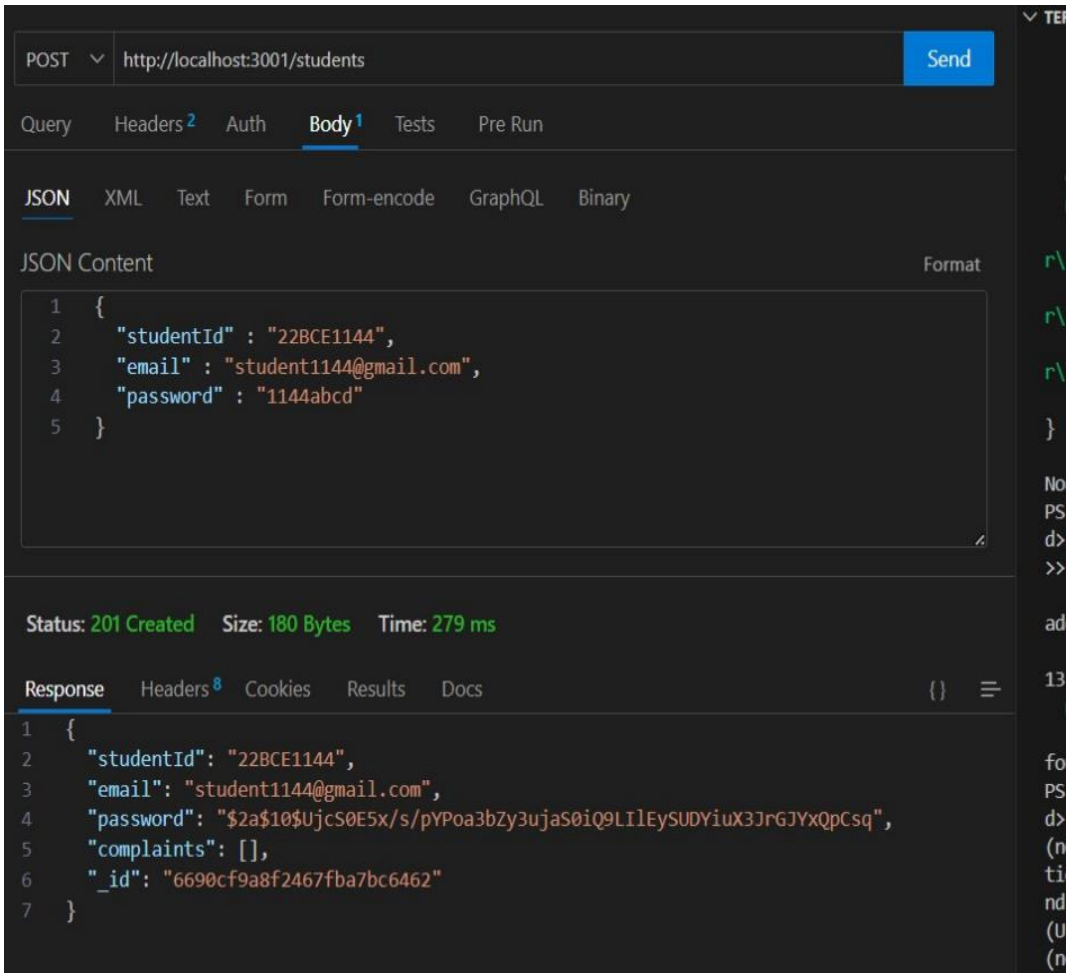
The response stored in the database collections:

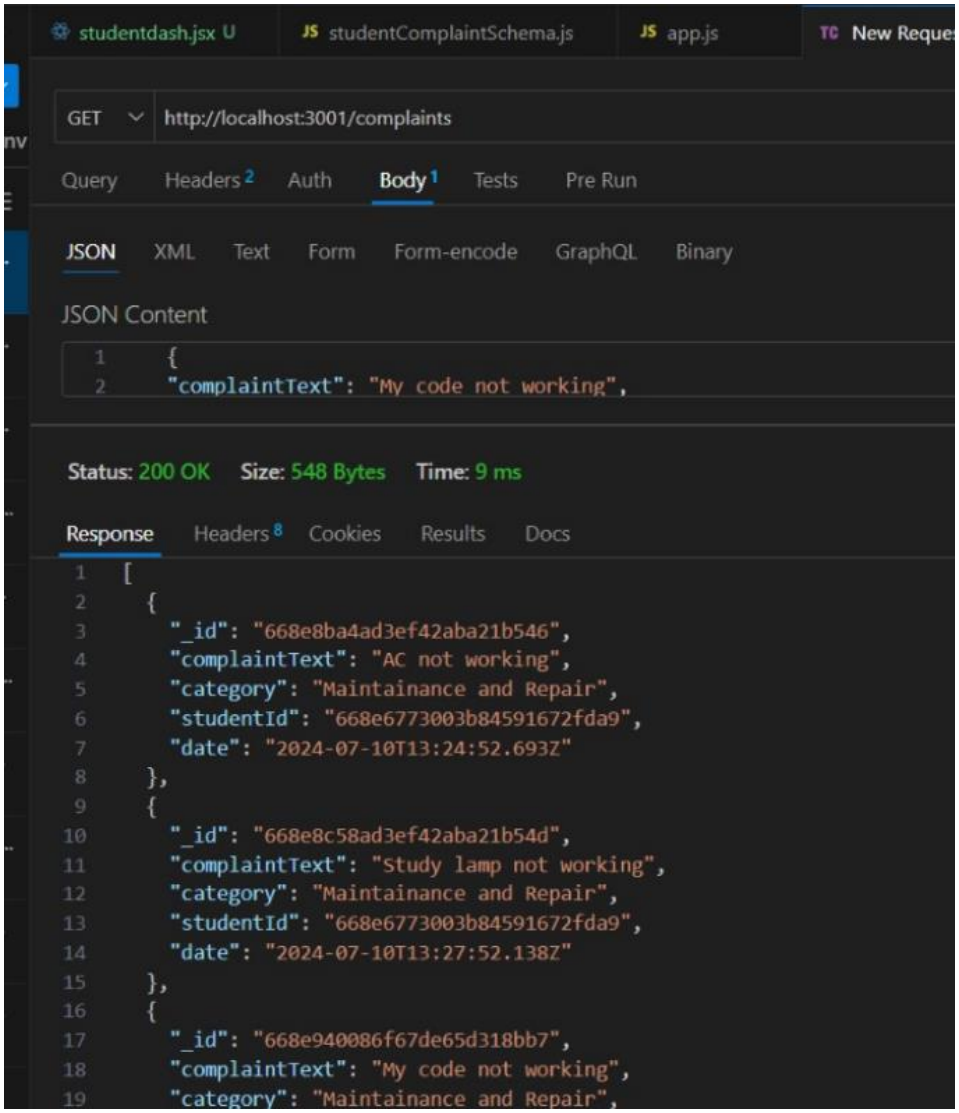


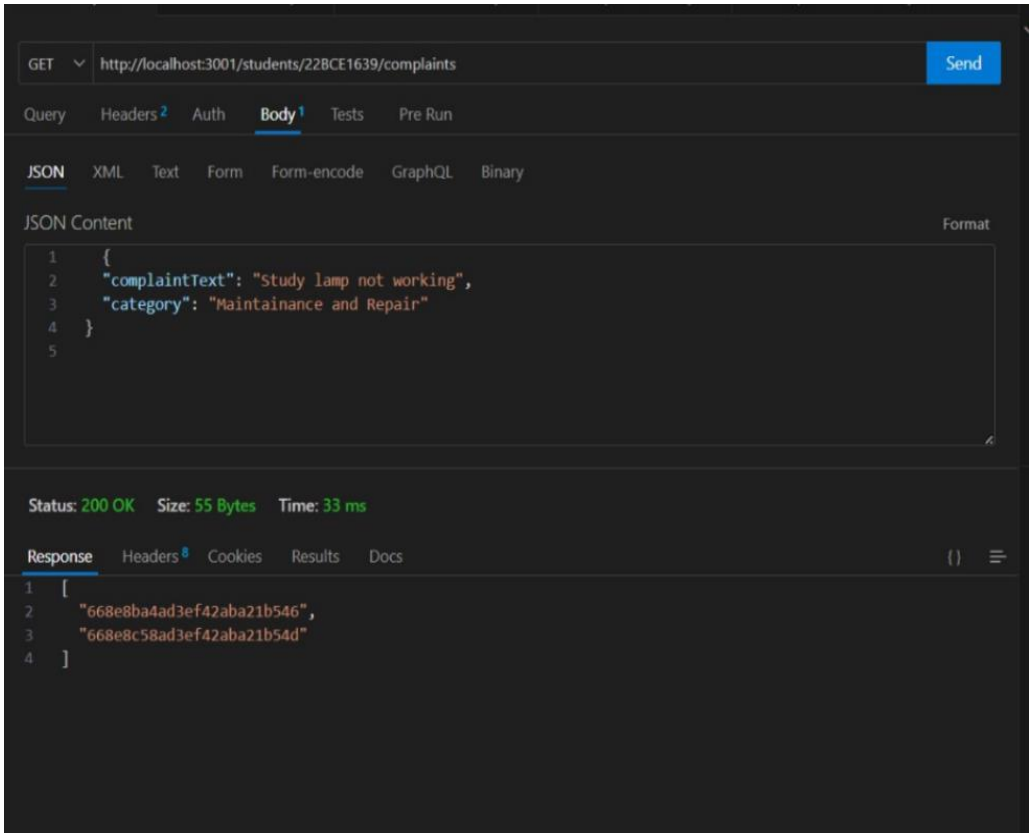
The queries and responses of the backend API endpoints**:**

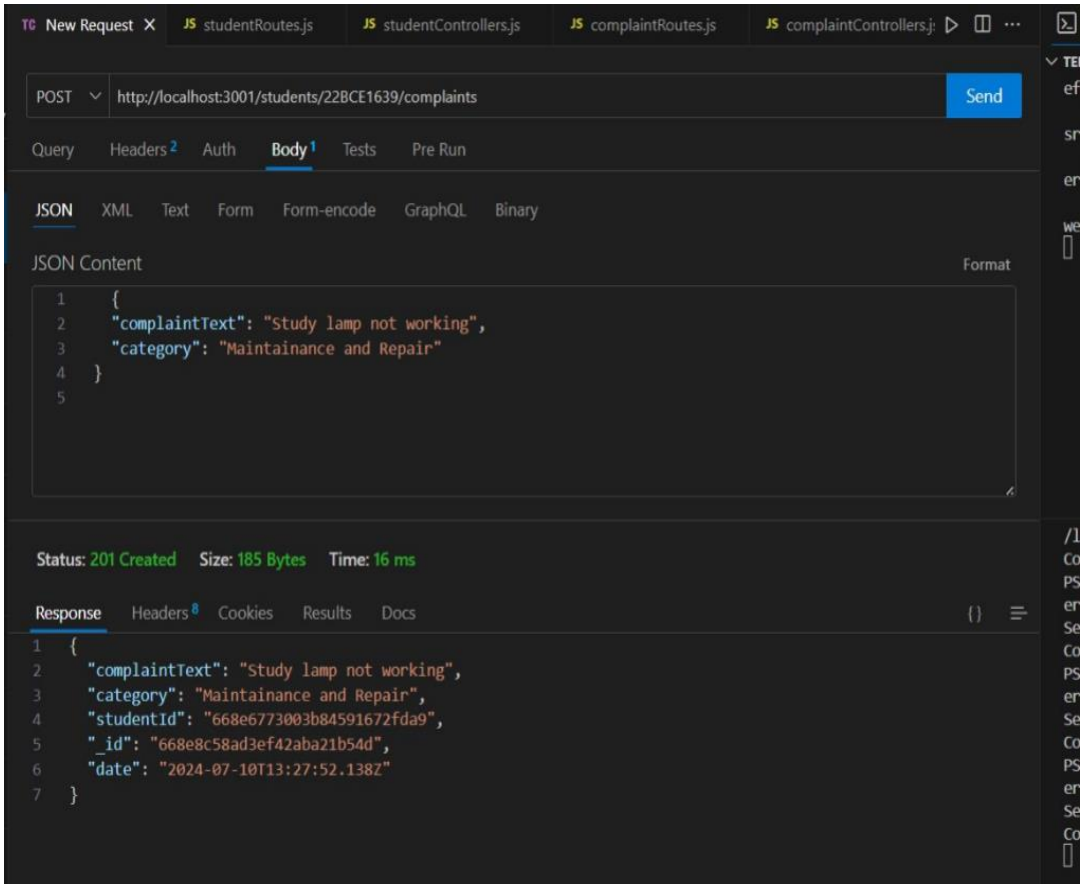


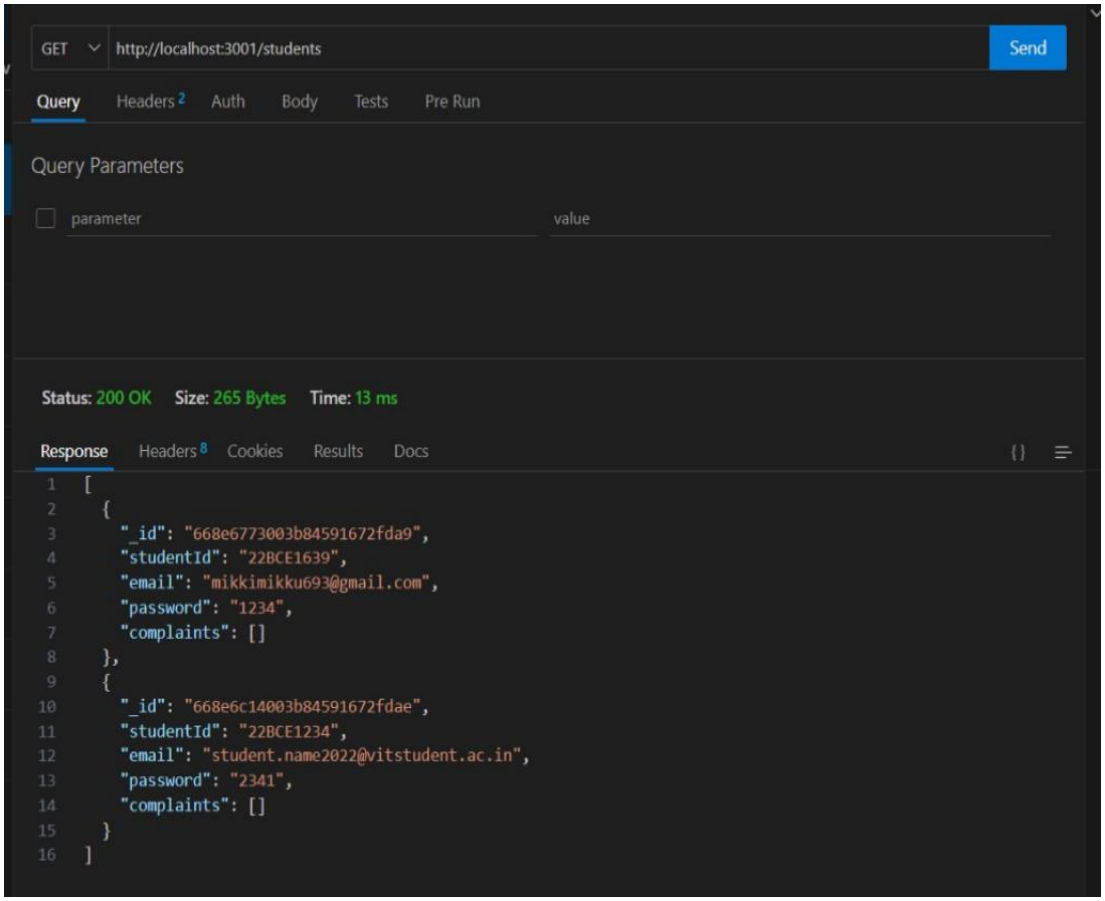

studentControllers.js
studentSchema.js
studentRoutes.js
New Request
GET
http://localhost:3001/students/22BCE1639
Send
Query
Headers 2
Auth
Body
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
Status: 200 OK
Size: 489 Bytes
Time: 82 ms
Response
Headers 8
Cookies
Results
Docs
{
"_id": "668e6773003b84591672fda9",
"studentId": "22BCE1639",
"email": "mikkimikku693@gmail.com",
"password": "1234",
"complaints": [
{
"_id": "668e8ba4ad3ef42aba21b546",
"complaintText": "AC not working",
"category": "Maintainance and Repair",
"studentId": "668e6773003b84591672fda9",
"date": "2024-07-10T13:24:52.693Z"
},
{
"_id": "668e8c58ad3ef42aba21b54d",
"complaintText": "Study lamp not working",
"category": "Maintainance and Repair",
"studentId": "668e6773003b84591672fda9",
"date": "2024-07-10T13:27:52.138Z"
}


POST
http://localhost:3001/students
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{
"studentId" : "22BCE1144",
"email" : "student1144@gmail.com",
"password" : "1144abcd"
}
Status: 201 Created
Size: 180 Bytes
Time: 279 ms
Response
Headers 8
Cookies
Results
Docs
{
"studentId": "22BCE1144",
"email": "student1144@gmail.com",
"password": "$2a$10$UjcS0E5x/s/pYPoa3bZy3ujaS0iQ9LIlEySUDYiuX3JrGJYxQpCsq",
"complaints": [],
"_id": "6690cf9a8f2467fba7bc6462"
}


studentdash.jsx U
studentComplaintSchema.js
app.js
New Reque
GET
http://localhost:3001/complaints
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
{
"complaintText": "My code not working",
Status: 200 OK
Size: 548 Bytes
Time: 9 ms
Response
Headers 8
Cookies
Results
Docs
[
{
"_id": "668e8ba4ad3ef42aba21b546",
"complaintText": "AC not working",
"category": "Maintainance and Repair",
"studentId": "668e6773003b84591672fda9",
"date": "2024-07-10T13:24:52.693Z"
},
{
"_id": "668e8c58ad3ef42aba21b54d",
"complaintText": "Study lamp not working",
"category": "Maintainance and Repair",
"studentId": "668e6773003b84591672fda9",
"date": "2024-07-10T13:27:52.138Z"
},
{
"_id": "668e940086f67de65d318bb7",
"complaintText": "My code not working",
"category": "Maintainance and Repair",


GET
http://localhost:3001/students/22BCE1639/complaints
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{
"complaintText": "Study lamp not working",
"category": "Maintainance and Repair"
}
Status: 200 OK
Size: 55 Bytes
Time: 33 ms
Response
Headers 8
Cookies
Results
Docs
[
"668e8ba4ad3ef42aba21b546",
"668e8c58ad3ef42aba21b54d"
]


New Request
studentRoutes.js
studentControllers.js
complaintRoutes.js
complaintControllers.j
POST
http://localhost:3001/students/22BCE1639/complaints
Send
Query
Headers 2
Auth
Body 1
Tests
Pre Run
JSON
XML
Text
Form
Form-encode
GraphQL
Binary
JSON Content
Format
{
"complaintText": "Study lamp not working",
"category": "Maintainance and Repair"
}
Status: 201 Created
Size: 185 Bytes
Time: 16 ms
Response
Headers 8
Cookies
Results
Docs
{
"complaintText": "Study lamp not working",
"category": "Maintainance and Repair",
"studentId": "668e6773003b84591672fda9",
"_id": "668e8c58ad3ef42aba21b54d",
"date": "2024-07-10T13:27:52.138Z"
}



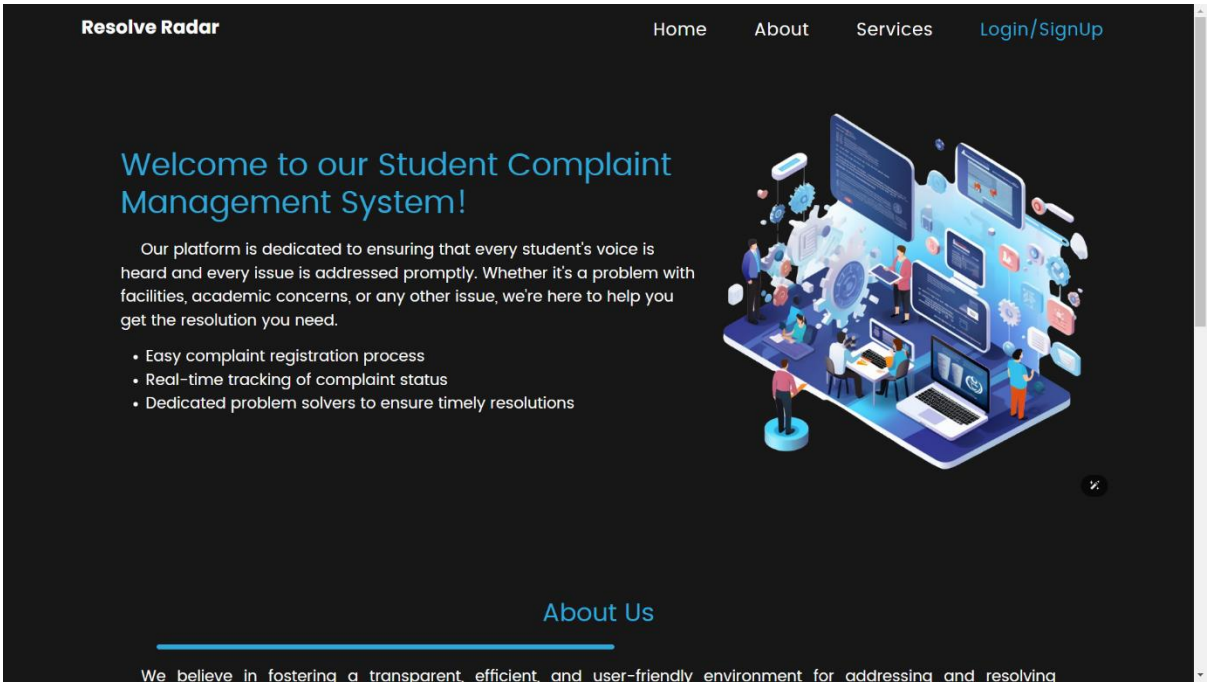
8. Authentication

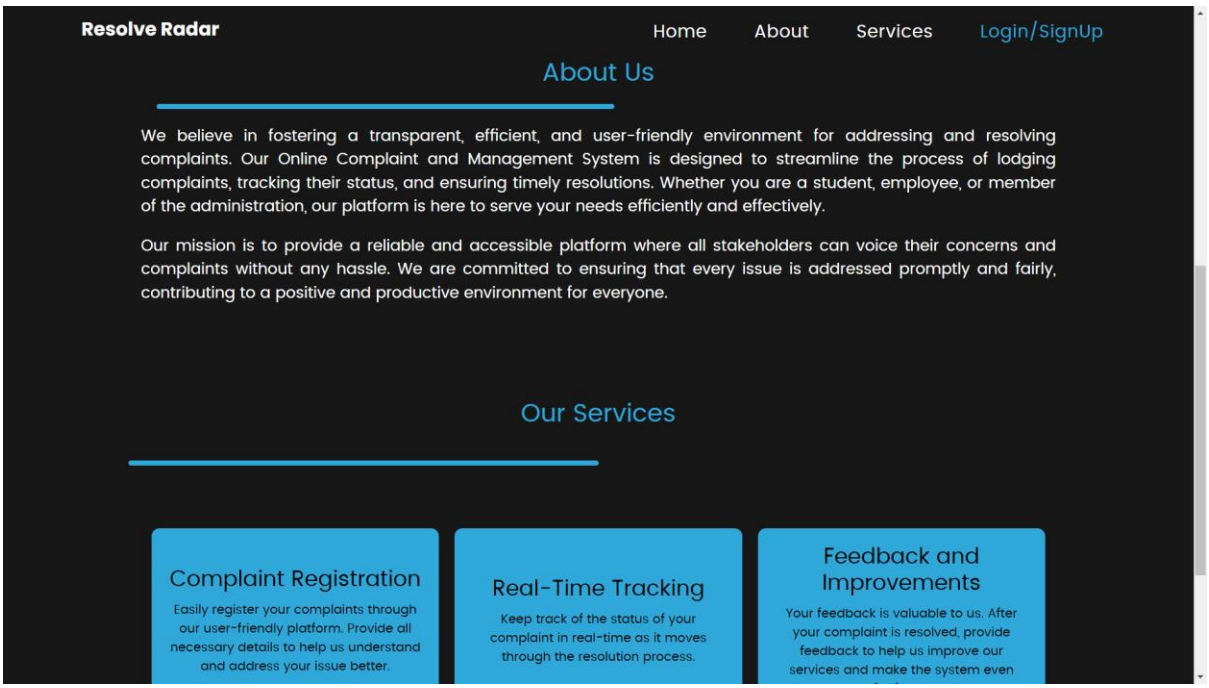
- Explain how authentication and authorization are handled in the project.
- Include details about tokens, sessions, or any other methods used.

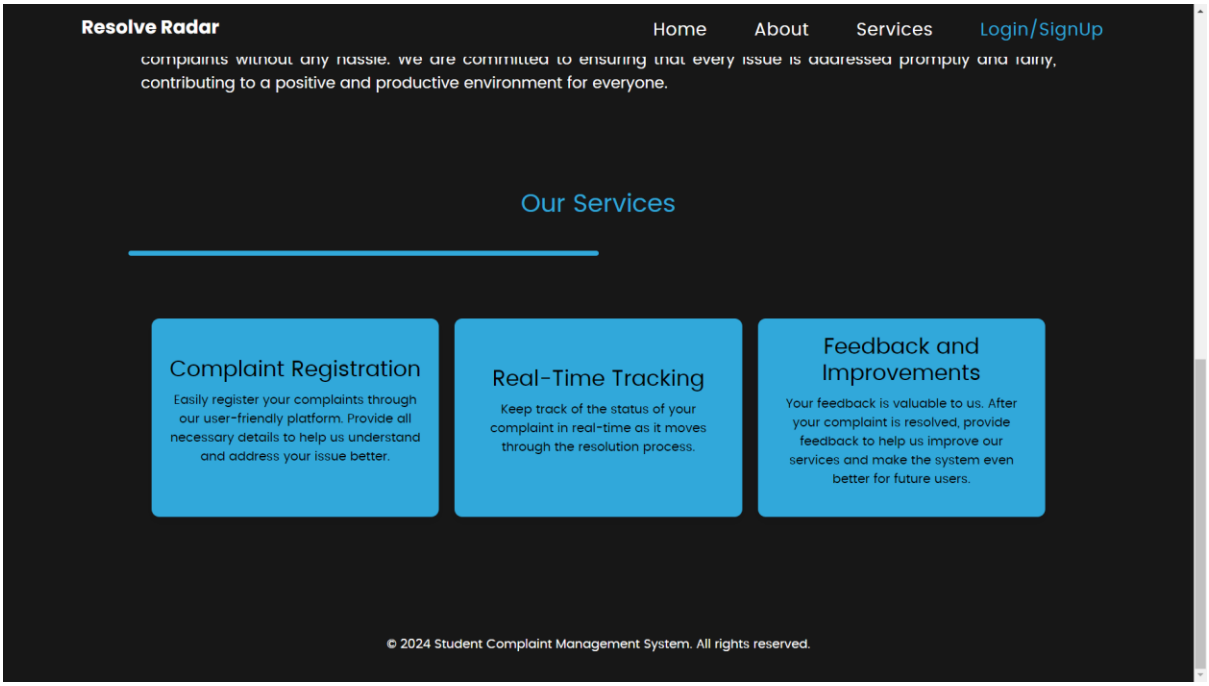
  - (We haven't integrated this part yet. As of for now, just 'database crosschecking' is used.)

9. User Interface

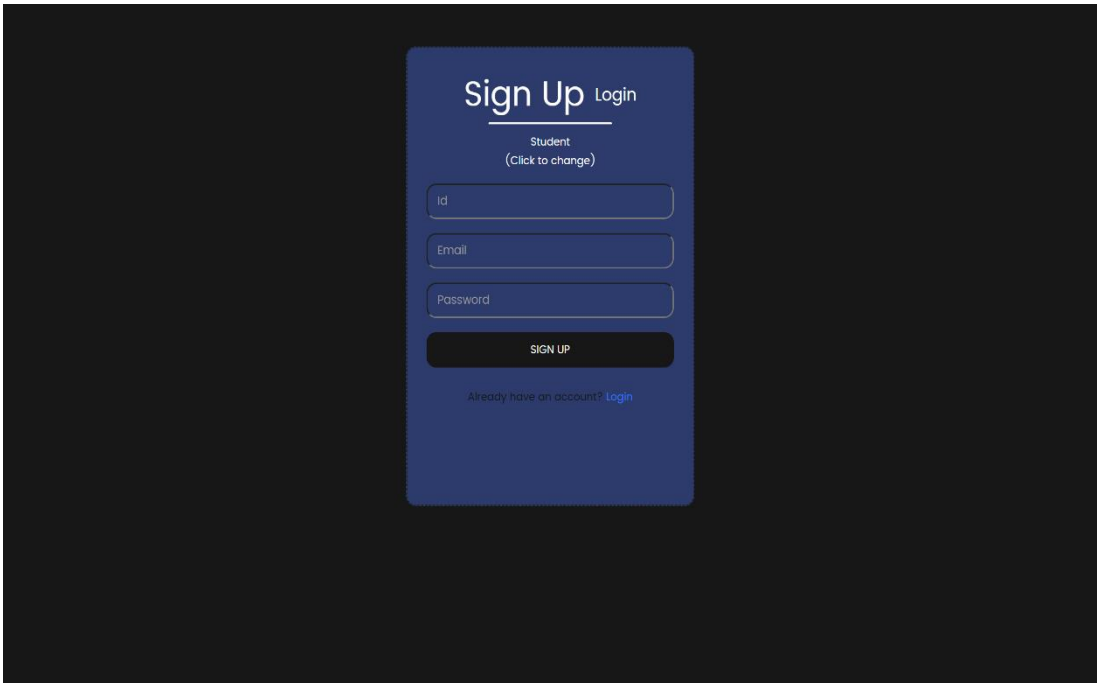
- Landing page

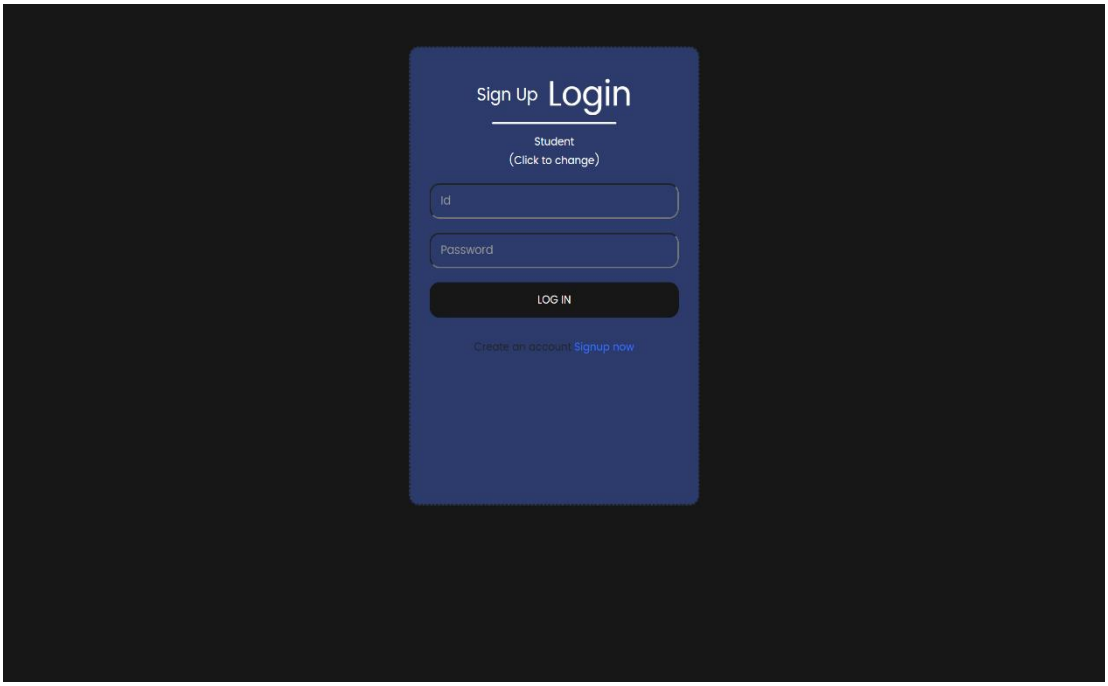

Resolve Radar
Home
About
Services
Login/SignUp
Welcome to our Student Complaint Management System!
Our platform is dedicated to ensuring that every student's voice is heard and every issue is addressed promptly. Whether it's a problem with facilities, academic concerns, or any other issue, we're here to help you get the resolution you need.
• Easy complaint registration process
• Real-time tracking of complaint status
• Dedicated problem solvers to ensure timely resolutions
About Us
We believe in fostering a transparent, efficient, and user-friendly environment for addressing and resolving


Resolve Radar
Home
About
Services
Login/SignUp
About Us
We believe in fostering a transparent, efficient, and user-friendly environment for addressing and resolving complaints. Our Online Complaint and Management System is designed to streamline the process of lodging complaints, tracking their status, and ensuring timely resolutions. Whether you are a student, employee, or member of the administration, our platform is here to serve your needs efficiently and effectively.
Our mission is to provide a reliable and accessible platform where all stakeholders can voice their concerns and complaints without any hassle. We are committed to ensuring that every issue is addressed promptly and fairly, contributing to a positive and productive environment for everyone.
Our Services
Complaint Registration
Easily register your complaints through our user-friendly platform. Provide all necessary details to help us understand and address your issue better.
Real-Time Tracking
Keep track of the status of your complaint in real-time as it moves through the resolution process.
Feedback and Improvements
Your feedback is valuable to us. After your complaint is resolved, provide feedback to help us improve our services and make the system even

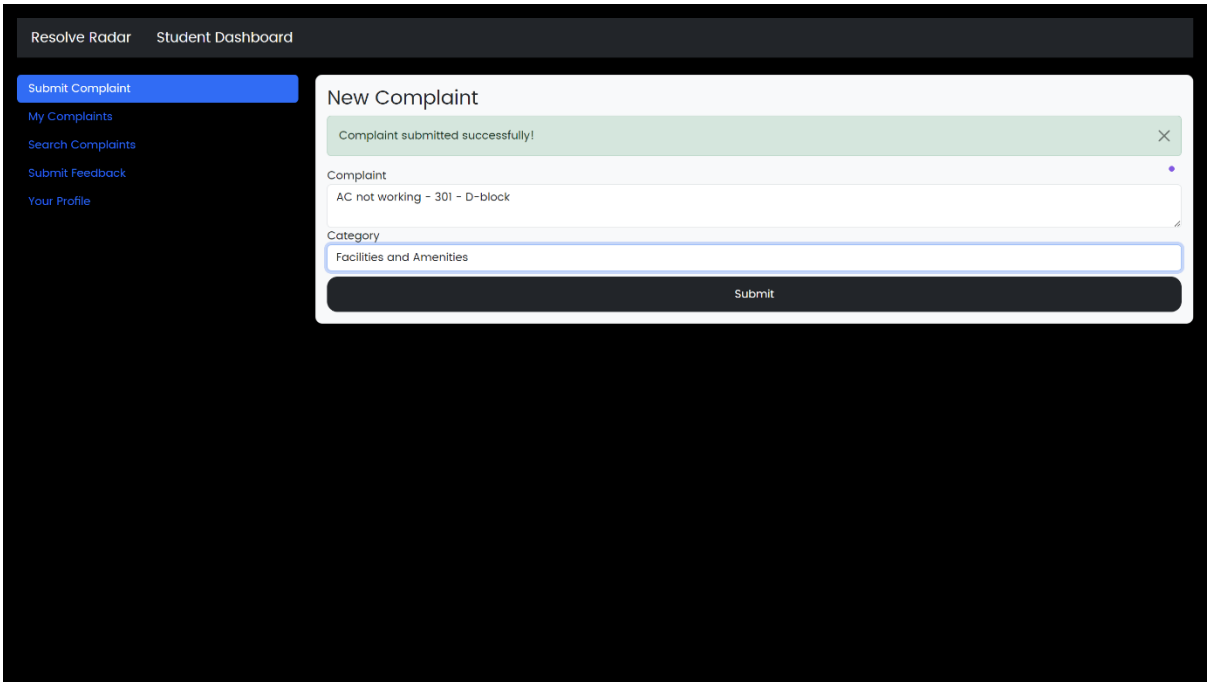


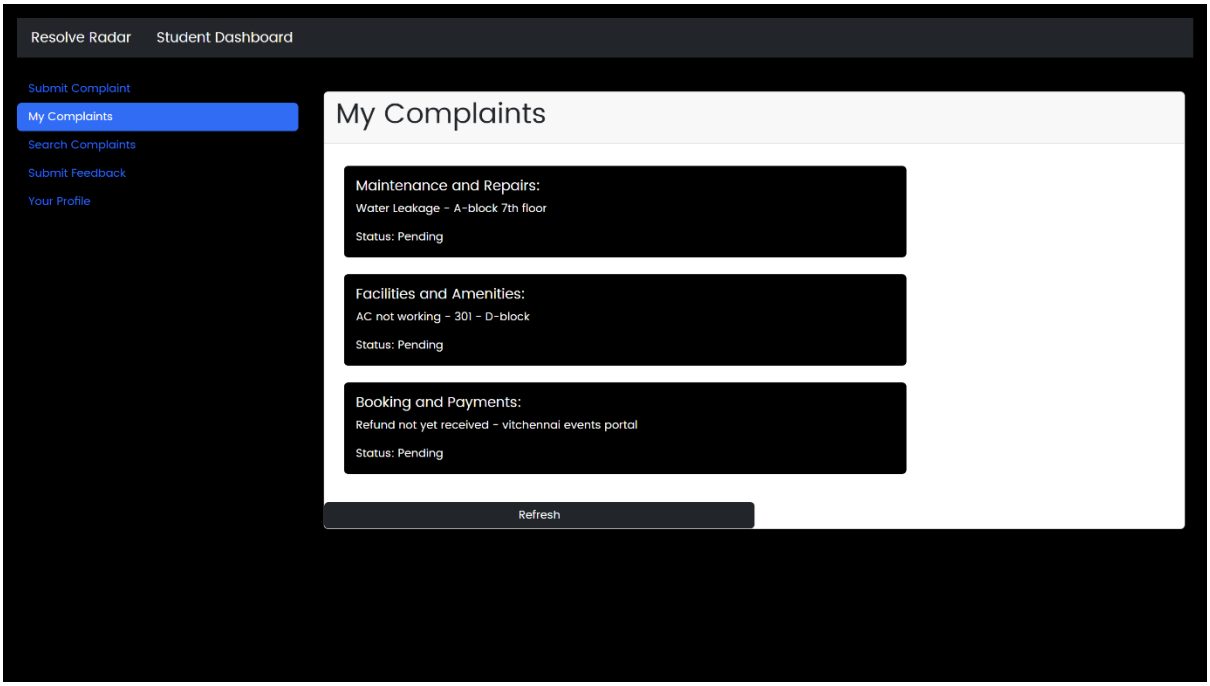
Sign-Up and Login pages:

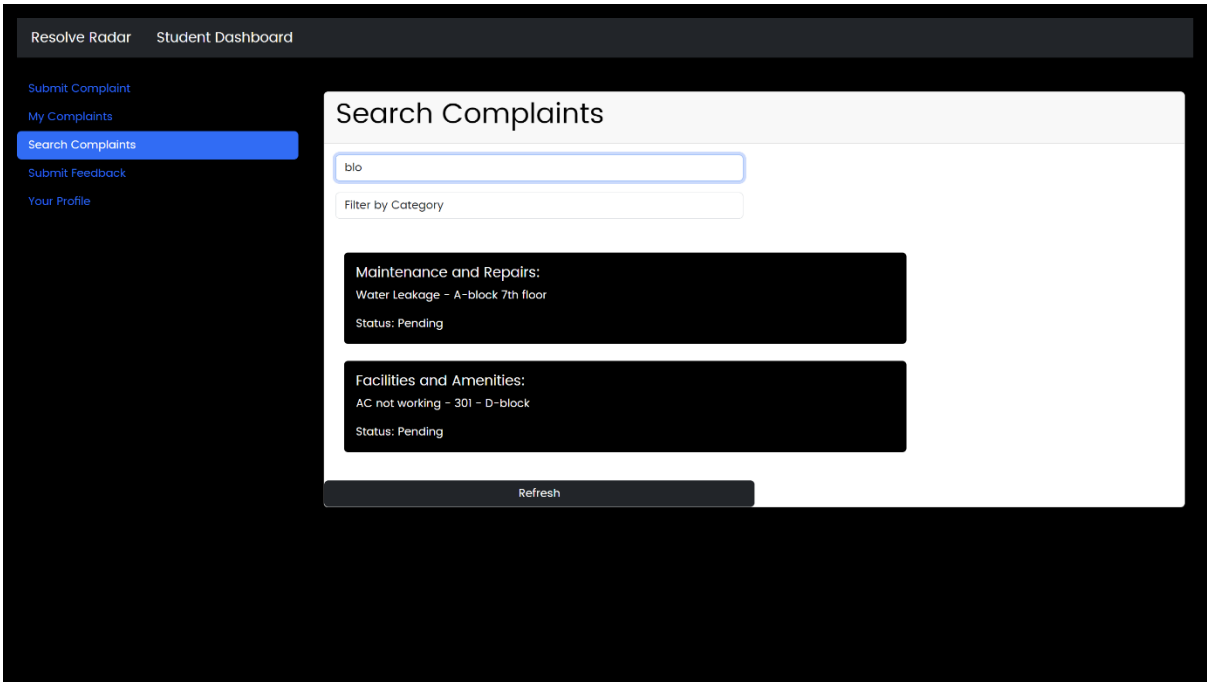


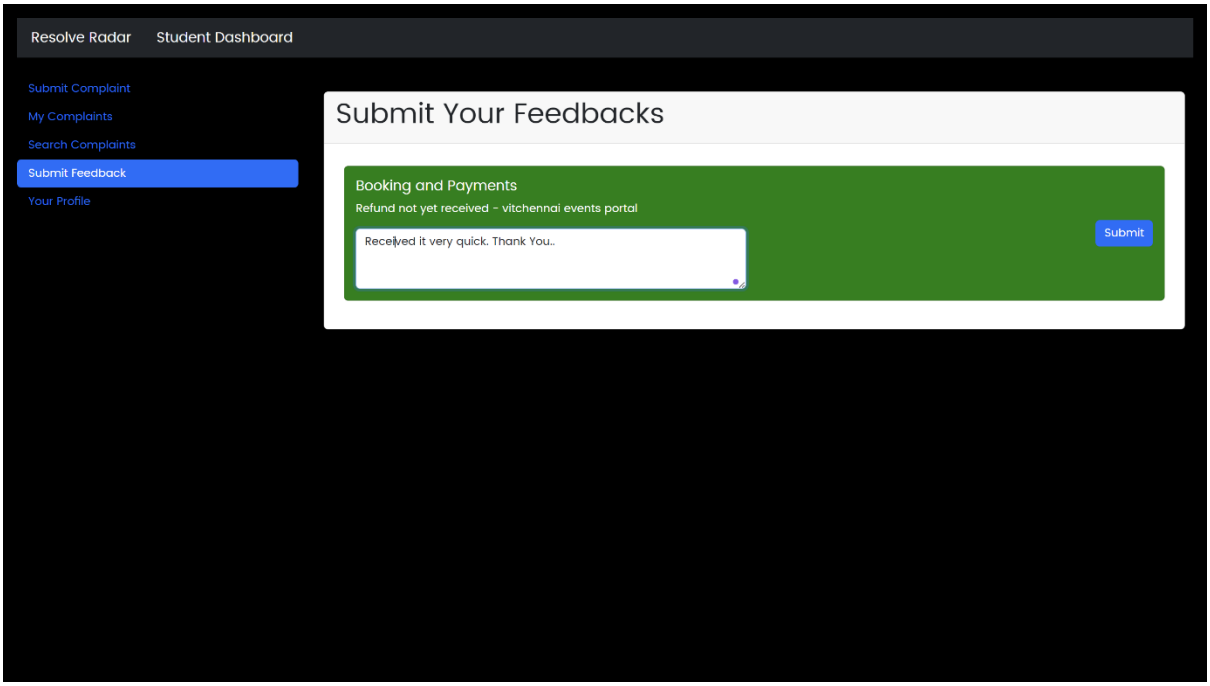


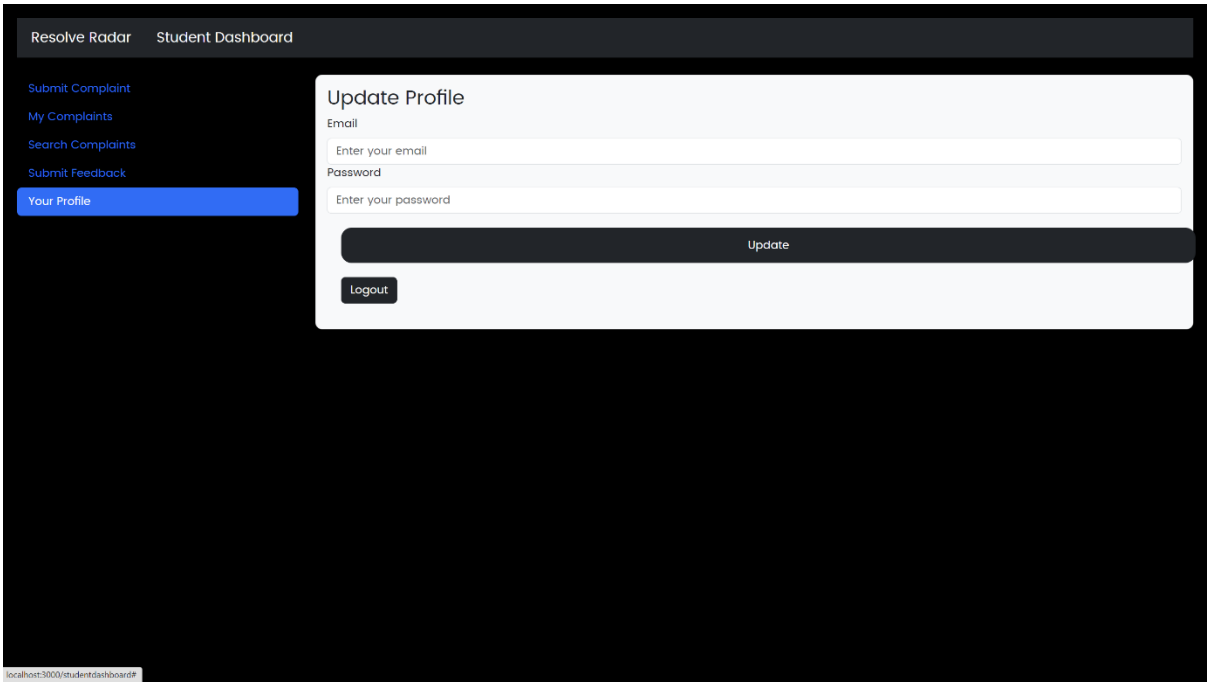
Student dashboard :



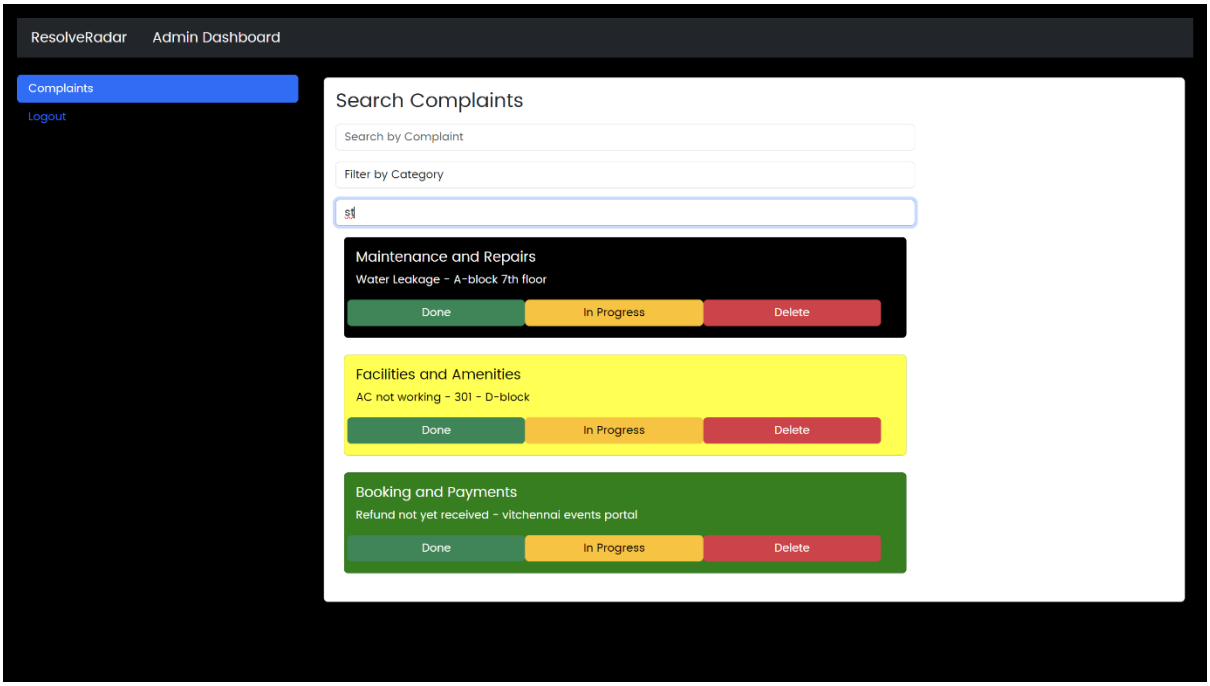

Resolve Radar
Student Dashboard
Submit Complaint
My Complaints
Search Complaints
Submit Feedback
Your Profile
My Complaints
Maintenance and Repairs:
Water Leakage - A-block 7th floor
Status: Pending
Facilities and Amenities:
AC not working - 301 - D-block
Status: Pending
Booking and Payments:
Refund not yet received - vitchennai events portal
Status: Pending
Refresh

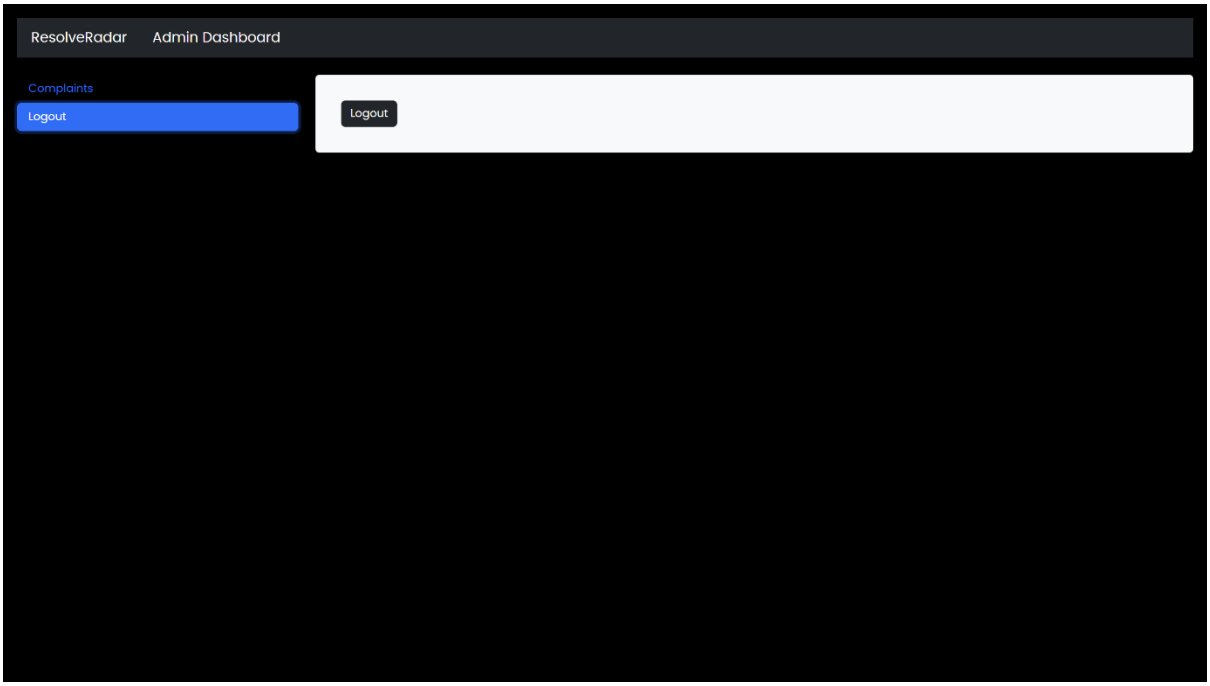

Resolve Radar
Student Dashboard
Submit Complaint
My Complaints
Search Complaints
Submit Feedback
Your Profile
Search Complaints
blo
Filter by Category
Maintenance and Repairs:
Water Leakage - A-block 7th floor
Status: Pending
Facilities and Amenities:
AC not working - 301 - D-block
Status: Pending
Refresh



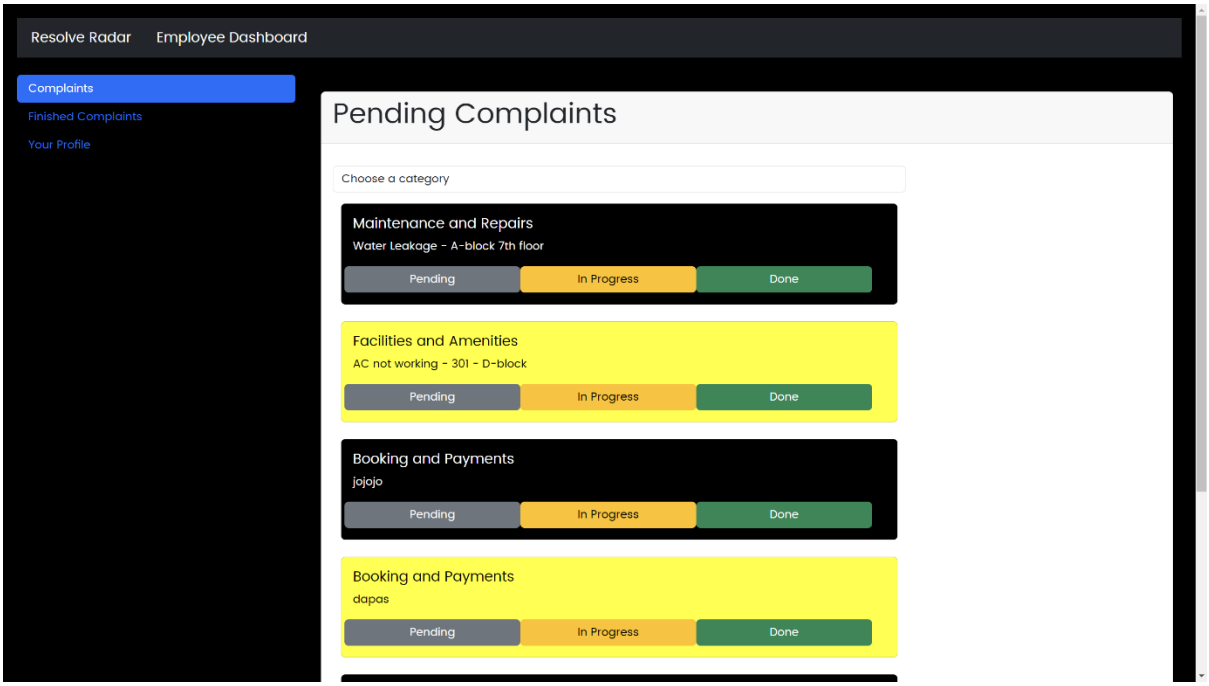


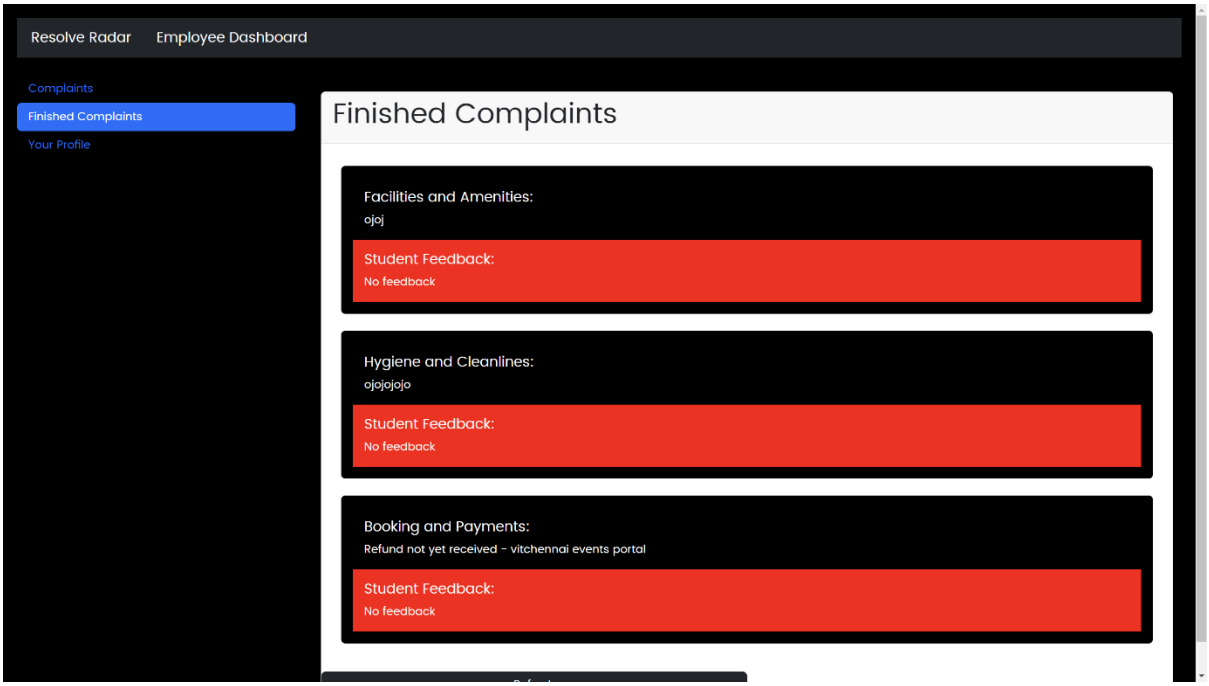
Admin dashboard:





Employee Dashboard:





10. Testing

- The testing of all the backend API endpoints were done by using the Postman extension of VS code.
- The MongoDB Compass was used to see if the collections were accurately updated upon logging in/submitting a complaint/resolving the complaint/updating the profile

11. Screenshots or Demo

- Provide screenshots or a link to a demo (if available) to showcase the application.

12. Known Issues

- Document any known bugs or issues that users or developers should be aware of.

  (As of now, none)

13. Future Enhancements

Some of the potential future features and improvements that could enhance the functionality and usability of the project:

1. User Role Management

Implement more granular user roles (e.g., admin, student, employee, guest) with specific permissions and access controls.

This enhances security and allows for more tailored user experiences.

2. Notification System

Add email or SMS notifications to inform users of important events (e.g., complaint status changes, new feedback). This helps, keeps users informed and engaged.

3. Advanced Search and Filtering

Implement advanced search and filtering options for complaints based on date range, keywords and the status of the complaint. This makes it easier for users to find specific complaints and improves data management.

4. Analytics Dashboard

Create an analytics dashboard to display statistics and trends related to complaints such as number of complaints per month, resolution times. This can be implemented in the admin dashboard which provides valuable insights to the administrators and help identify areas for improvement.

5. Mobile Application

Develop a mobile application for Android and iOS to provide a more accessible and convenient way for users to interact with the system. This increases accessibility and user engagement.

6. Complaint History and Tracking

Allow users to view the history and status changes of their complaints. This provides transparency and keeps users informed about the progress of their complaints.

7. Automated Complaint Assignment

Implement algorithms to automatically assign complaints to the most suitable employee based on predefined criteria (e.g., expertise, workload). This increases efficiency and ensures complaints are handled by the most qualified personnel.

8. Enhanced Security Features

Add features like two-factor authentication (2FA), CAPTCHA, and improved password policies. This enhances security and protects user data.

9. Multilingual Support

Provide support for multiple languages to cater to a diverse user base.

10. Chatbot Integration

Integrate a chatbot to assist users with common queries and guide them through the complaint submission process. This helps to improve the user experience and reduces the workload on support staff.

11. Gamification

Introduce gamification elements (e.g., badges, leaderboards) to encourage users to engage with the system, which in turn increases user engagement and motivation.