

**FULL STACKDEVELOPMENT WITH MERN**

**PROJECT TITLE : Online Complaint Registration And Management System**

**Team Members :**

**1.Sneha.N[TL]**

**2.Abitha sathya.S[FRONT END DEVELOPER]**

**3.Sarath.T [BACK END DEVELOPER]**

**4.Ajith.B [QUALITY ASSURANCE]**

**Abstract :** The main purpose of this project is to help the public in knowing their place details and getting their problems solved in online without going to the officer regularly until the problem is solve.By this system the public can save his time and eradicate corruption in government offices.Its main purpose is to provide a smart and easy way through android or web Application for Complaint registration and its Tracking and eradicating system and thus to prevent Corruption.As technology is growing rapidly, we are also moving to a technical world where everything we want is to be online.The system will be developed using HTML, CSS, JavaScript and using Php MyAdmin**By**

**INTRODUCTION :**

The main purpose of this project is to help the public in knowing their place details and getting their problems solved in online without going to the ofﬁcer regularly until the problem is solved.We want to develop web application for Complaint management .To transform the existing manual compliant management system into an automate system. For the better management of complaints to improve efﬁciency.

The main purpose of this project is to help the public in knowing their place details and getting their problems solved in online without going to the officer regularly until the problem is solved.

**SCOPE OF THE PROJECT :**

The objective of this website is to help the public in knowing their place details and getting their problems solved in online without going to the ofﬁcer regularly until the problem is solved.

**Project Overview**

**Purpose :** Online Complaint Registration And Management System provides an online way of solving the problems faced by the public by saving time and eradicate corruption , And The ability of providing many of the reports on the system , and add to Facilitate the process of submitting a complaint.Online Complaint Management System provides an online way of solving the problems faced by the public by saving time and eradicate corruption , And The ability of providing many of the reports on the system , and add to Facilitate the process of submitting a complaint

**Features:**

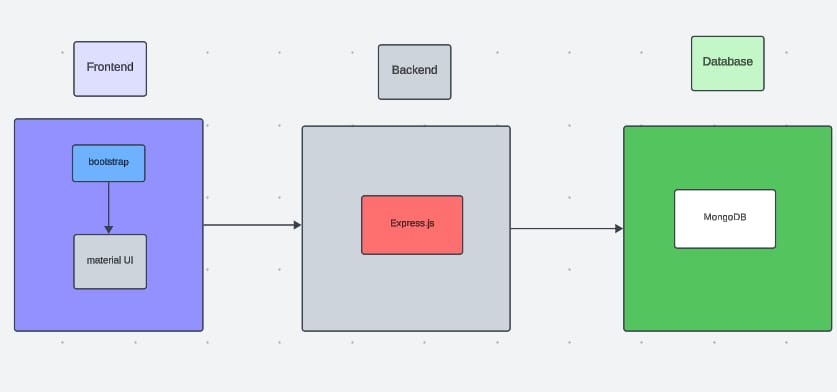
* Dynamic creation of customer and product master during complaint registration .
* Transparent resentment administration.
* Configuration Escalation Matrix.
* Structured information flow and analytics.
* Case**/**Complaint wisetracking of spare parts with price.
* Real-time notification via mail,message and more .
* Case traceability.
* Non follow p of SLA ,bad customer feedback escalates the request.
* Status wise tracking of complaints.

**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 proflies,for complaints registration,etc.It ensures reliable and quick access to the necessary information during registration of user or any complaints.

Together,this frontend and backend components,along with socket.io,Epress.js,WebRTC API,and MongoDB,form a comprehensive technical architecture for our video conferences app.



**Backend (Node.js, Express.js, MongoDB)**

**Dependencies**

- express: for creating the RESTful API

- mongodb: for interacting with the MongoDB database

- mongoose: for modeling and interacting with the MongoDB database

- bcrypt: for password hashing

- jsonwebtoken: for authentication and authorization

Models (Mongoose Schemas)

**User Model**

const userSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

role: String

});

**Complaint Model**

const complaintSchema = new mongoose.Schema({

title: String,

description: String,

category: String,

status: String,

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

**Comment Model**

const commentSchema = new mongoose.Schema({

text: String,

complaintId: { type: mongoose.Schema.Types.ObjectId, ref: 'Complaint' },

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

**API Endpoints**

- POST /api/register: register a new user

- POST /api/login: login an existing user

- POST /api/complaints: create a new complaint

- GET /api/complaints: retrieve all complaints

- GET /api/complaints/:id: retrieve a single complaint by ID

- PUT /api/complaints/:id: update a single complaint by ID

- DELETE /api/complaints/:id: delete a single complaint by ID

- POST /api/comments: create a new comment

- GET /api/comments: retrieve all comments

- GET /api/comments/:id: retrieve a single comment by ID

**Authentication and Authorization**

- Use JSON Web Tokens (JWT) to authenticate and authorize users

- Verify user role and permissions for each API endpoint

**Frontend (React.js)**

**Dependencies**

- react: for building the user interface

- react-router-dom: for client-side routing

- axios: for making API requests to the backend

**Components**

- App.js: the main application component

- Login.js: the login component

- Register.js: the registration component

- Complaints.js: the complaints component

- Complaint.js: the single complaint component

- Comments.js: the comments component

- Comment.js: the single comment component

**Routing**

- Use react-router-dom to define client-side routes

- Define routes for login, registration, complaints, and comments

**API Requests**

- Use axios to make API requests to the backend

- Make requests to retrieve and update complaints and comments

**State Management**

- Use React state to manage the application's state

- Use React context API to share state between components

**Database**

**Backend (Node.js, Express.js, MongoDB)**

**Dependencies**

- express: for creating the RESTful API

- mongodb: for interacting with the MongoDB database

- mongoose: for modeling and interacting with the MongoDB database

- bcrypt: for password hashing

- jsonwebtoken: for authentication and authorization

**Models (Mongoose Schemas)**

**User Model**

const userSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

role: String

});

**Complaint Model**

const complaintSchema = new mongoose.Schema({

title: String,

description: String,

category: String,

status: String,

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

**Comment Model**

const commentSchema = new mongoose.Schema({

text: String,

complaintId: { type: mongoose.Schema.Types.ObjectId, ref: 'Complaint' },

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }

});

**API Endpoints**

- POST /api/register: register a new user

- POST /api/login: login an existing user

- POST /api/complaints: create a new complaint

- GET /api/complaints: retrieve all complaints

- GET /api/complaints/:id: retrieve a single complaint by ID

- PUT /api/complaints/:id: update a single complaint by ID

- DELETE /api/complaints/:id: delete a single complaint by ID

- POST /api/comments: create a new comment

- GET /api/comments: retrieve all comments

- GET /api/comments/:id: retrieve a single comment by ID

**Authentication and Authorization**

- Use JSON Web Tokens (JWT) to authenticate and authorize users

- Verify user role and permissions for each API endpoint

**Frontend (React.js)**

**Dependencies**

- react: for building the user interface

- react-router-dom: for client-side routing

- axios: for making API requests to the backend

**Components**

- App.js: the main application component

- Login.js: the login component

- Register.js: the registration component

- Complaints.js: the complaints component

- Complaint.js: the single complaint component

- Comments.js: the comments component

- Comment.js: the single comment component

**Routing**

- Use react-router-dom to define client-side routes

- Define routes for login, registration, complaints, and comments

**API Requests**

- Use axios to make API requests to the backend

- Make requests to retrieve and update complaints and comments

**State Management**

- Use React state to manage the application's state

- Use React context API to share state between components

**Database (MongoDB)**

**Collections**

- users: stores user data

- complaints: stores complaint data

- comments: stores comment data

**Schema**

- Use Mongoose to define the schema for each collection

- Define fields and data types for each collection

This is a high-level overview of an online complaint registration and management system using the MERN stack.

**Setup Instruction**

**Prerequisties :**

**Hardware Requirements**

1. Computer or laptop with a minimum of 4GB RAM and 256GB storage

2. Reliable internet connection

**Software Requirements**

1. Node.js: Install the latest version of Node.js (LTS or Current) from the official Node.js website.

2. MongoDB: Install MongoDB Community Server (latest version) from the official MongoDB website.

3. Express.js: Install Express.js using npm (Node Package Manager) by running the command npm install express in your terminal.

4. React.js: Install React.js using npm by running the command npm install react in your terminal.

5. Visual Studio Code or any other code editor of your choice.

6. Postman or any other API testing tool of your choice.

**Technical Skills**

1. Basic knowledge of HTML, CSS, and JavaScript.

2. Understanding of React.js and its ecosystem (e.g., JSX, components, state, props).

3. Familiarity with Node.js and Express.js for building RESTful APIs.

4. Knowledge of MongoDB and Mongoose for database modeling and interaction.

5. Understanding of authentication and authorization concepts (e.g., JWT, sessions).

6. Familiarity with API testing tools like Postman.

**Project Requirements**

1. Create a new Node.js project using npm init.

2. Install required dependencies using npm.

3. Set up a MongoDB database and connect it to your Node.js application.

4. Design and implement the API endpoints for complaint registration, management, and retrieval.

5. Implement authentication and authorization using JSON Web Tokens (JWT).

6. Build the frontend using React.js, including components for complaint registration, management, and retrieval.

7. Test the application using Postman or other API testing tools.

By fulfilling these prerequisites, you'll be well-prepared to build a comprehensive online complaint registration and management system using the MERN stack.

**Installation**

**Installing Node.js and npm**

1. Download and install Node.js from the official Node.js website: (link unavailable)

2. Once installed, open a terminal or command prompt and verify the installation by running the command: node -v

3. npm (Node Package Manager) is included with Node.js. Verify the installation by running the command: npm -v

**Installing MongoDB**

1. Download and install MongoDB Community Server from the official MongoDB website: (link unavailable)

2. Follow the installation instructions for your operating system.

3. Once installed, start the MongoDB service by running the command: mongod (on macOS/Linux) or net start mongodb (on Windows)

4. Verify the installation by running the command: mongo

**Installing Express.js**

1. Create a new directory for your project and navigate into it: mkdir complaint-management-system && cd complaint-management-system

2. Initialize a new Node.js project by running the command: npm init

3. Install Express.js by running the command: npm install express

4. Create a new file called server.js and add the following code:

const express = require('express');

const app = express();

const port = 3000;

app.listen(port, () => {

console.log(`Server started on port ${port}`);

});

1. Start the server by running the command: node server.js

**Installing React.js**

1. Install React.js by running the command: npx create-react-app client

2. Navigate into the newly created client directory: cd client

3. Start the React development server by running the command: npm start

**Installing Mongoose**

1. Install Mongoose by running the command: npm install mongoose

2. Create a new file called models/Complaint.js and add the following code:

const mongoose = require('mongoose');

const complaintSchema = new mongoose.Schema({

title: String,

description: String,

category: String,

status: String

});

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

module.exports = Complaint;

1. Create a new file called models/User.js and add the following code:

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

name: String,

email: String,

password: String,

role: String

});

const User = mongoose.model('User', userSchema);

module.exports = User;

**Installing JSON Web Tokens (JWT)**

1. Install JWT by running the command: npm install jsonwebtoken

2. Create a new file called utils/jwt.js and add the following code:

const jwt = require('jsonwebtoken');

const secretKey = 'your-secret-key';

const generateToken = (user) => {

const token = jwt.sign(user, secretKey, { expiresIn: '1h' });

return token;

};

const verifyToken = (token) => {

try {

const decoded = jwt.verify(token, secretKey);

return decoded;

} catch (error) {

return null;

}

};

module.exports = { generateToken, verifyToken };

Replace 'your-secret-key' with a secure secret key.

**Folder structure :**

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

Client (React Frontend)

The client-side code is organized into the following folders:

**Public**

- Contains static assets, such as images and fonts

- index.html: The main entry point of the application

**src**

- components: Reusable React components

- **ComplaintForm.js:** Form component for submitting complaints

- **ComplaintList.js:** List component for displaying complaints

**containers:** Higher-order components that wrap around other components

- **`App.js`:** Main application component

- **`ComplaintDashboard.js**`: Dashboard component for viewing complaints

**actions:** Action creators for Redux

- **complaintActions.js:** Action creators for complaint-related actions

**reducers:** Redux reducers

**- complaintReducer.js:** Reducer for complaint-related state

**utils**: Utility functions

- **api.js**: API utility functions

**index.js:** Main entry point of the application

**store.js:** Redux store configuration

**Server (Node.js Backend)**

The server-side code is organized into the following folders:

**config**

**database.js:** Database configuration

**auth.js:** Authentication configuration

**controllers**

**complaintController.js:** Controller for complaint-related routes

**userController.js:** Controller for user-related routes

**models**

**complaintModel.js**: Mongoose model for complaints

**userModel.js**: Mongoose model for users

**routes**

- **complaintRoutes.js:** Routes for complaint-related endpoints

- **userRoutes.js**: Routes for user-related endpoints

**services**

- **complaintService.js:** Service for complaint-related business logic

- **userService.js**: Service for user-related business logic

**utils**

- **auth.js:** Authentication utility functions

- errorHandler.js: Error handling utility functions

**app.js**

- Main entry point of the server-side application

- Sets up Express.js app and routes

**Running the Application**

To run the Online Complaint Registration and Management System, follow these steps:

**Starting the Frontend Server**

1. Navigate to the `client` directory:

bash

cd client

1. Start the frontend server using npm:

npm start

This will start the React development server, and you can access the application at `http://localhost:3000`.

**Starting the Backend Server**

1. Navigate to the `server` directory:

bash

cd server

1. Start the backend server using npm:

npm start

This will start the Node.js server, and you can access the API endpoints at `http://localhost:5000`.

**Accessing the Application**

Once both servers are running, you can access the application at `http://localhost:3000`. You can register complaints, view complaint lists, and perform other actions as described in the application's documentation.

**API Documentation:**

**Complaints**

**Get All Complaints**

- **URL: `**/complaints`

**- Method**: `GET`

**- Parameters**: None

**- Response**:

[

{

"id": 1,

"title": "Example Complaint",

"description": "This is an example complaint.",

"status": "Open",

"created\_at": "2022-01-01T12:00:00.000Z"

},

{

"id": 2,

"title": "Another Example Complaint",

"description": "This is another example complaint.",

"status": "Closed",

"created\_at": "2022-01-02T12:00:00.000Z"

}

]

**Get Complaint by ID**

**URL:** `/complaints/:id`

**Method:** `GET`

**Parameters:**

- id: The ID of the complaint to retrieve.

**Response:**

{

"id": 1,

"title": "Example Complaint",

"description": "This is an example complaint.",

"status": "Open",

"created\_at": "2022-01-01T12:00:00.000Z"

}

**Create New Complaint**

**URL**: `/complaints`

**Method**: `POST`

**Parameters:**

- title: The title of the complaint.

- description: The description of the complaint.

**Response:**

{

"id": 3,

"title": "New Complaint",

"description": "This is a new complaint.",

"status": "Open",

"created\_at": "2022-01-03T12:00:00.000Z"

}

**Update Existing Complain**t

**URL:** `/complaints/:id`

**Method:** `PUT`

**Parameters:**

- id: The ID of the complaint to update.

- title: The updated title of the complaint.

- description: The updated description of the complaint.

**Response:**

{

"id": 1,

"title": "Updated Complaint",

"description": "This is an updated complaint.",

"status": "Open",

"created\_at": "2022-01-01T12:00:00.000Z"

}

Delete Complaint

**URL:** `/complaints/:id`

**Method** :DELETE**`**

**Parameters:**

id: The ID of the complaint to delete.

**Response**:

{

"message": "Complaint deleted successfully."

}

**Users**

**Get All Users**

**URL:** `/users`

**Method:** `GET`

**Parameters**: None

**Response**:

[

{

"id": 1,

"name": "John Doe",

"email": "johndoe@example.com"

},

{

"id": 2,

"name": "Jane Doe",

"email": "janedoe@example.com"

}

]

**Get User by ID**

**URL**: `/users/:id`

**Method:** `GET`

**Parameters:**

- `id`: The ID of the user to retrieve.

**Response:**

{

"id": 1,

"name": "John Doe",

"email": "johndoe@example.com"

}

**Create New User**

**URL**: `/users`

**Method**: `POST`

**Parameters:**

- `name`: The name of the user.

- `email`: The email of the user.

**Response:**

{

"id": 3,

"name": "New User",

"email": "newuser@example.com"

}

**Update Existing User**

**URL:** `/users/:id`

**Method: `**PUT`

**Parameters:**

- `id`: The ID of the user to update.

- `name`: The updated name of the user.

- `email`: The updated email of the user.

**Response:**

{

"id": 1,

"name": "Updated User",

"email": "updateduser@example.com"

}

**Delete User**

**- URL: `**/users/:id`

**- Method:** `DELETE`

**Parameters:**

- **id**: The ID of the user to delete.

**Response:**

{

**"message":** "User deleted successfully."

}

**Authentication**

The project uses JSON Web Tokens (JWT) for authentication. Here's a step-by-step overview of the authentication process:

**1**. **User Registration:** When a user registers, their credentials are stored in the database.

**2**. **User Login**: When a user logs in, the system checks their credentials against the stored data in the database.

**3**. **Token Generation:** If the credentials are valid, the system generates a JWT token containing the user's ID and other relevant information.

**4. Token Verification:** The token is then verified on each subsequent request to ensure that the user is authenticated.

**Authorization**

The project uses Role-Based Access Control (RBAC) for authorization. Here's an overview of the authorization process:

**1. User Roles**: Each user is assigned a role (e.g., admin, moderator, user) when they register.

**2. Role-Based Permissions**: Each role has specific permissions that determine what actions the user can perform.

**3. Permission Checks:** On each request, the system checks the user's role and permissions to determine whether they are authorized to perform the requested action.

**Token Storage**

The JWT token is stored in the user's browser using HTTP-only cookies. This ensures that the token is secure and cannot be accessed by malicious scripts.

**Token Validation**

On each request, the system validates the JWT token by checking its signature, expiration time, and other relevant information. If the token is invalid or has expired, the system returns an error response.

**Session Management**

The project does not use traditional sessions. Instead, it relies on JWT tokens for authentication and authorization. This approach is more suitable for modern web applications, as it allows for better scalability and flexibility.

**Security Measures**

The project implements the following security measures to protect against common web vulnerabilities:

**-HTTPS:** The application uses HTTPS to encrypt data transmitted between the client and server.

- **CSRF Protection**: The application uses CSRF tokens to protect against cross-site request forgery attacks.

**- Input Validation**: The application validates user input to prevent SQL injection and cross-site scripting (XSS) attacks.

**- Password Hashing:** The application uses a secure password hashing algorithm to protect user passwords.

**Testing Strategy**

The testing strategy for the Online Complaint Registration and Management System involves a combination of manual and automated testing. The goal is to ensure that the system meets the required functionality, usability, and performance standards.

**Manual Testing**

Manual testing involves testing the system manually by simulating real-user scenarios. This includes:

**- Unit Testing:** Testing individual components or units of code to ensure they function as expected.

**- Integration Testing**: Testing how different components or units of code interact with each other.

- **System Testing:** Testing the entire system to ensure it meets the required functionality and performance standards.

**- User Acceptance Testing (UAT):** Testing the system with real users to ensure it meets their expectations and requirements.

**Automated Testing**

Automated testing involves using tools and scripts to test the system automatically. This includes:

**- Unit Testing**: Using testing frameworks like Jest or Mocha to write and run unit tests.

**- Integration Testing:** Using testing frameworks like Cypress or Selenium to write and run integration tests.

**- API Testing**: Using tools like Postman or Swagger to test APIs.

**Testing Tools**

The following testing tools are used for the Online Complaint Registration and Management System:

**- Jest**: A JavaScript testing framework used for unit testing and integration testing.

**- Cypress:** A JavaScript testing framework used for integration testing and end-to-end testing.

- **Postman:** A tool used for API testing and documentation.

- **Selenium:** An open-source tool used for automating web browsers and testing web applications.

**- Swagger:** An open-source tool used for API documentation and testing.

**Testing Environment**

The testing environment for the Online Complaint Registration and Management System includes:

**- Development Environmen**t: A local development environment set up on each developer's machine.

- **Staging Environment**: A staging environment set up on a remote server to mimic the production environment.

**- Production Environment:** The live production environment where the system is deployed.

**Testing Schedule**

The testing schedule for the Online Complaint Registration and Management System includes:

**- Daily Testing:** Daily testing and review of code changes.

- **Weekly Testing:** Weekly testing and review of system functionality.

**- Monthly Testing**: Monthly testing and review of system performance and security.

**Testing Deliverables**

The testing deliverables for the Online Complaint Registration and Management System include:

- **Test Plans:** Detailed test plans outlining the testing approach, test cases, and test data.

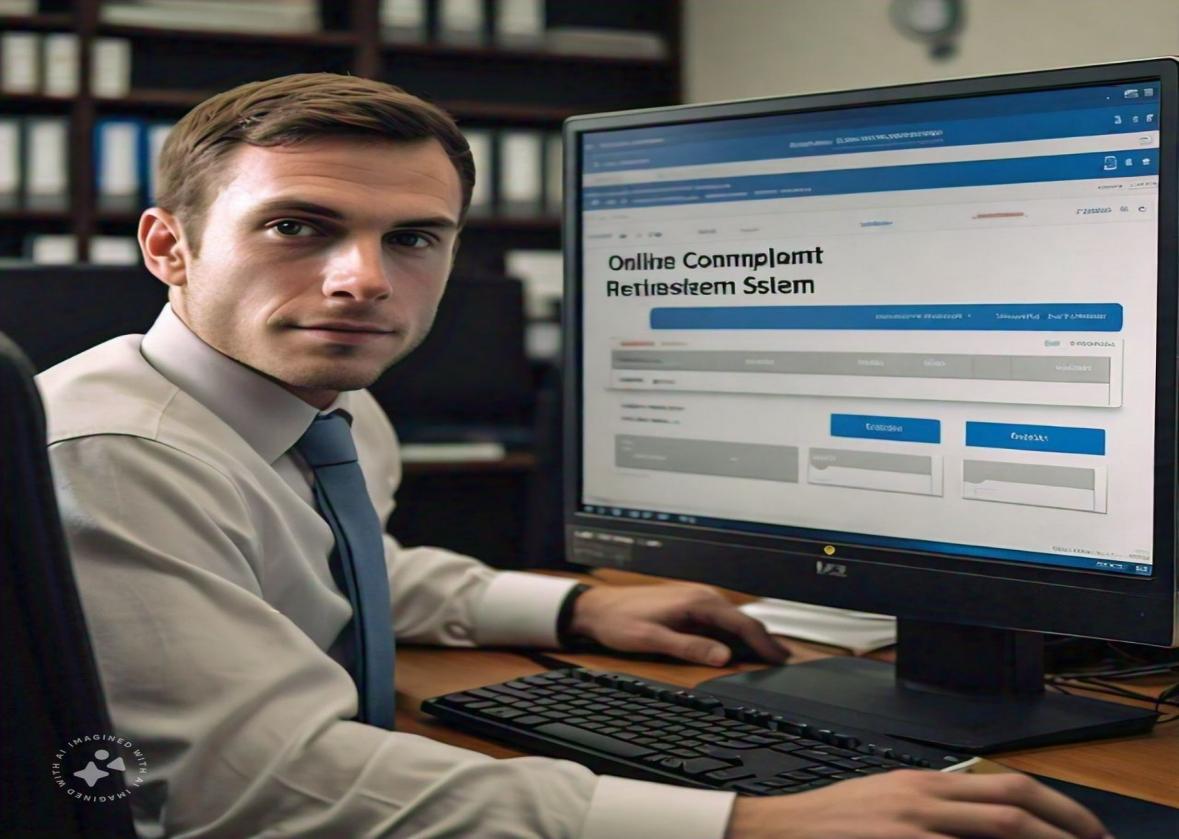
**- Test Reports:** Test reports summarizing the testing results, including defects found and defects fixed.

- **Test Cases:** Test cases outlining the steps to test specific system functionality.

**- Defect Reports:** Defect reports detailing defects found during testing, including steps to reproduce and expected results.

**User Interface**

Provide screenshots or GIFs showcasing different UI features.



**Next Steps**

1. Implement the complaint registration and management system using the installed dependencies.

2. Design and implement the API endpoints for complaint registration, management, and retrieval.

3. Implement authentication and authorization using JSON Web Tokens (JWT).

4. Build the frontend using React.js, including components for complaint registration, management, and retrieval.

5. Test the application using Postman or other API testing tools.

**MODULE DESCRIPTION**

**USER**

* Register
* Post Complaint
* Location Mark in Google Map View complaint status Feedback
* Get Admin Contact detail**s**

**OFFICER**

* Register
* View Complaint Update status
* Update proof of work

**ADMIN**

* Generate id & password for ofﬁcer
* The administrator has the full-ﬂedged rights over the OES. Create/delete an account.
* view the accounts.
* Change the password.
* Hide any kind of features from the both of users. Insert the information available on OES.
* Access all the accoun**ts** of the ofﬁcer.

**MODULE DESCRIPTION :**

**USER**

**Registration Module**

Another main function of our proposed system is registration, in order to register with the unique application details such as name; password, email, place and time are required.

**Post Complaint**

User posts their complaint through this application. That is they can’t get water regular manner in their area and about the problem description.

**Location Mark in Google Map**

In this module is used to the user marks their location in the Google Map for the betterment of the complaint registration.

**View Status**

If the user to compliant the problems they need to a current status of the compliant so it helps to user views the status about their complaint.

**Feedback**

It helps to give a feedback for after completed the compliant the user to give a feedback about the action what they done in a compliant.

**OFFICER**

**View Complaint**

In this module is used to ofﬁcer view the complaint details who posted the user complaint using the Google map location mark.

**Update status**

In this module is used to ofﬁcer updates status for user compliant, update the current status what going on in a compliant.

**Update completed proof**

It used to if the related compliant work is completed they give a proof using photo format, Ofﬁcer Update photo proof of work.

**ADMIN**

**Generate id & password for ofﬁcer**

In this module is used for generate the unique id and password for the Public Work Department (PWD) ofﬁcer.

**Create/ delete an account**

The admin can create an account for new user and also if it’s not need an account, admin can remove the user account.

**View the accounts**

In this module used for admin to view all the users account for the reference, it helps to know the users strength.

**Change the password**

In this module is used for to change the password for the security purpose.

**Insert the information of available on OES**

Admin want to insert the information of available on Ofﬁce Economic Stabilization (OES).

**PROPOSED SYSTEM**

We want to replace existing manual CMS (Complaint Management System) to an android application changing the way of maintaining the society complaints will also prove to be beneﬁcial, improve efﬁciency and save us time.By using this application people can register their complaints in easy and proper format. Mainly they can mark their location in Google Map while placing the complaint so that it will help the people in easy manner.They will also well aware about their complaints progress.They can also provide feedback about their complaints progress weather they are satisﬁed or not.Also they user can post their requirements through this system and they will receive needed items by admin within couple of hours ,its depending on the needed item and you can also look your status about your requirements.These user complaints, needs requirements maintain by admin. The User post feedback of these CMS system and admin can view this feedback.

**ADVANTAGES OF PROPOSED SYSTEM**

* To compliant our problem at any time.
* Minimum time needed for the various processing.
* The system is greater efﬁciency and better service .
* User friendliness and interactive .
* Minimum time required and also minimise manual data entry.

**Practice to be presented**

1. To solve complaint the officers have their own rules and standards written in document
2. form for complaint user.
3. The complaint decision is announced to user on notice board.
4. To face complaint users bring a applicants letter from anywhere physically to officers.
5. Officers allow to access information about the authority of the user.
6. The employee of officers must have full reason to solve their complain.
7. If the user wants to face their complain different reason first they must apply to lowest level and then continue to next branch of office such as department head ,college dean ,HRM, Finance Directorate, and etc.
8. If one wants to leave from DBU before he/she fills the form leave form he/she must
9. return all working material to respected department otherwise they will be rejected.

**Forms and other documents of existing system**

There are a number of forms and documents which are used by the existing system.The forms are used during the period of accepting and solving process for the employee to control business rules of the system.The uses of these forms are to assure the correctness of their activity and to generate reports.

The following is sample form taken from Finance Directorate.

**System Requirement Specification :**

**Functional Requriments :**

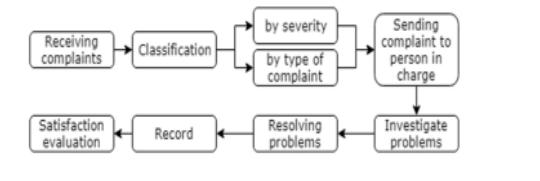
* Dart 2.15.1 or Higher
* Git / Github
* MongoDB 5.0
* **Code Editer :** Flutter, Andriod Studio, Visual Studio

**Non -Functional Requriements :**

* **Operatingn System :** Windows 7 or later(64-bit)
* **Disk Space :** 128 GB high speed storage(m.2 SSDs)

**System Architecture**

**Data Flow Diagram**

****

**Implementation**

Login Page

import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {

const MyApp({Key? key}) : super(key: key);

static const String \_title = 'Sample App';

@override

Widget build(BuildContext context) {

return MaterialApp(

title: \_title,

home: Scaffold(

appBar: AppBar(title: const Text(\_title)), body: const MyStatefulWidget(),

),

);

}

class MyStatefulWidget extends StatefulWidget {

const MyStatefulWidget({Key? key}) : super(key: key);

@override

State<MyStatefulWidget> createState() => \_MyStatefulWidgetState();}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

TextEditingController nameController = TextEditingController();

TextEditingController passwordController = TextEditingController();

@override

Widget build(BuildContext context) {

return Padding(

padding: const EdgeInsets.all(10),

child: ListView(

children: <Widget>[

Container(

alignment: Alignment.center,

padding: const EdgeInsets.all(10),

child: const Text(

'TutorialKart',

style: TextStyle(

color: Colors.blue,

fontWeight: FontWeight.w500,

fontSize: 30),

)),

Container(

alignment: Alignment.center…

padding: const EdgeInsets.all(10),

child: const Text(

'Sign in',

style: TextStyle(fontSize: 20),

)),

Container(

padding: const EdgeInsets.all(10),

child: TextField(

controller: nameController,

decoration: const InputDecoration(

border: OutlineInputBorder(),

labelText: 'User Name',

),

),

),

Container(

padding: const EdgeInsets.fromLTRB(10, 10, 10, 0),

child: TextField(

obscureText: true,

controller: passwordController,

decoration: const InputDecoration(

border: OutlineInputBorder(),

labelText: 'Password',

),

),

),

TextButton(

onPressed: () {

//forgot password screen

child: const Text('Forgot Password',),

),

Container(

height: 50,

padding: const EdgeInsets.fromLTRB(10, 0, 10, 0), child: ElevatedButton(

child: const Text('Login'),

onPressed: () {

print(nameController.text);

print(passwordController.text);

},

)

),

Row(

children: <Widget>[

const Text('Does not have account?'),

TextButton(

child: const Text(

'Sign in',

style: TextStyle(fontSize: 20),

),

onPressed: () {

//signup screen

},

)

],

mainAxisAlignment: MainAxisAlignment.center,

),

],

));

}

}

**Complaint Registration Page :**

import 'package:flutter/material.dart';

class SignupPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

resizeToAvoidBottomInset: false,

resizeToAvoidBottomPadding: false,

backgroundColor: Colors.white,

appBar: AppBar(

elevation: 0,

brightness: Brightness.light,

backgroundColor: Colors.white,

leading:

IconButton( onPressed: (){

Navigator.pop(context);

},icon:Icon(Icons.arrow\_back\_ios,size: 20,color: Colors.black,)),

),

body: SafeArea(

child: SingleChildScrollView(

child: Container(

height: MediaQuery.of(context).size.height,

width: double.infinity,

child: Column(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: [

Column(.

children: [

Column(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

Text ("Sign up", style: TextStyle(

fontSize: 30,

fontWeight: FontWeight.bold,

),),

SizedBox(height: 20,),

Text("Create an Account,Its free",style: TextStyle(

fontSize: 15,

color: Colors.grey[700],

),),

SizedBox(height: 30,)

],

),

Padding(

padding: EdgeInsets.symmetric(

horizontal: 40

),

child: Column(

children: [

makeInput(label: "Email"),

makeInput(label: "Password",obsureText: true),

makeInput(label: "Confirm Pasword",obsureText: true) ],

),

),

Padding(

padding: EdgeInsets.symmetric(horizontal: 40), child: Container(

padding: EdgeInsets.only(top: 3,left: 3),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(40),

border: Border(

bottom: BorderSide(color: Colors.black), top: BorderSide(color: Colors.black),

right: BorderSide(color: Colors.black),

left: BorderSide(color: Colors.black)

)

),

child: MaterialButton(

minWidth: double.infinity,

height:60,

onPressed: (){},

color: Colors.redAccent,

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(40)

),

child: Text("Sign Up",style: TextStyle(

fontWeight: FontWeight.w600,fontSize: 16,

),),

),

),

),

SizedBox(height: 20,),

Row(

mainAxisAlignment: MainAxisAlignment.center, children: [

Text("Already have an account? "),

Text("Login",style: TextStyle(

fontWeight: FontWeight.w600,

fontSize: 18

),),

],

)

],

),

],

),

),

),

),

);

}

}

Widget makeInput({label,obsureText = false}){

return Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(label,style:TextStyle(

fontSize: 15,

fontWeight: FontWeight.w400,

),),

SizedBox(height: 5,),

TextField(

obscureText: obsureText,

decoration: InputDecoration(

contentPadding: EdgeInsets.symmetric(vertical: 0,horizontal: 10),

enabledBorder: OutlineInputBorder(

borderSide: BorderSide(

color: Colors.grey[400],

),

),

border: OutlineInputBorder(

borderSide: BorderSide(color: Colors.grey[400])

),

),

),

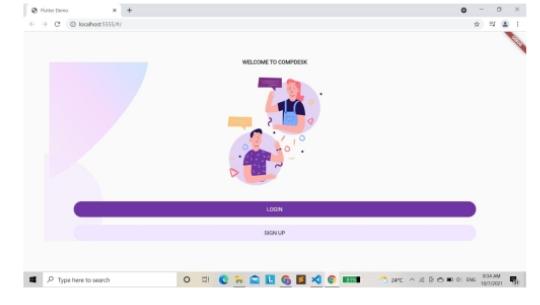
SizedBox(height: 30,)

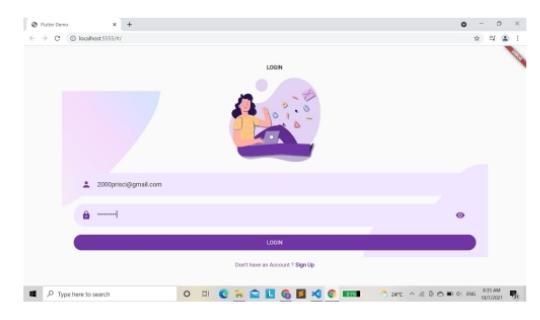
],

);

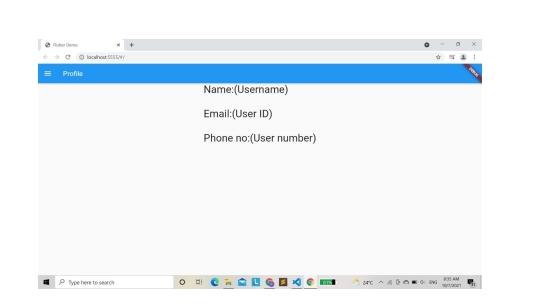
}

**Screen Shots**

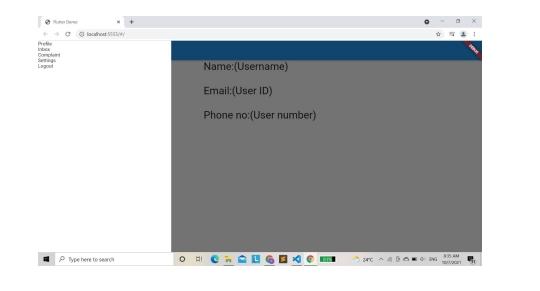
****

****

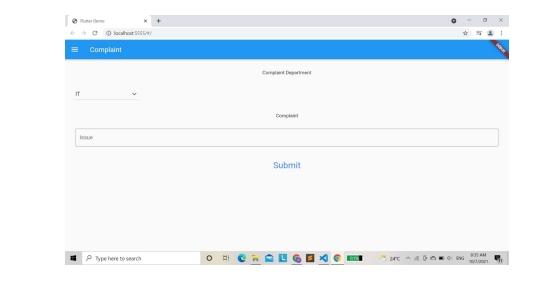
**Login Page**

****

**User Page**

****

**User Profile Page**

****

**Complaint Page**

**Known issues**

**Backend Issues**

**1. Authentication and Authorization**: Issues with authentication and authorization may arise, such as incorrect password hashing or inadequate role-based access control.

**2. Database Connection:** Problems with connecting to the MongoDB database may occur, such as incorrect connection strings or inadequate database permissions.

**3. API Endpoints:** Issues with API endpoints may arise, such as incorrect routing or inadequate error handling.

**4. Error Handling:** Inadequate error handling may lead to issues with error messages, logging, and debugging.

**Frontend Issues**

**1. Rendering and Layout:** Issues with rendering and layout may arise, such as incorrect component rendering or inadequate styling.

**2. State Management:** Problems with state management may occur, such as incorrect state updates or inadequate state synchronization.

**3. Event Handling:** Issues with event handling may arise, such as incorrect event listeners or inadequate event handling logic**.**

**4. Browser Compatibility:** Inadequate browser compatibility may lead to issues with rendering, layout, and functionality.

**Database Issues**

**1. Data Modeling:** Issues with data modeling may arise, such as incorrect schema design or inadequate data normalization.

**2. Data Validation**: Problems with data validation may occur, such as incorrect data types or inadequate validation logic.

**3. Data Indexing:** Issues with data indexing may arise, such as incorrect indexing or inadequate indexing strategies.

**4. Data Backup and Recovery:** Inadequate data backup and recovery strategies may lead to issues with data loss and system downtime.

**Security Issues**

**1. Authentication and Authorization:** Issues with authentication and authorization may arise, such as incorrect password hashing or inadequate role-based access control.

**2. Input Validation and Sanitization:** Problems with input validation and sanitization may occur, such as incorrect input validation or inadequate sanitization logic.

**3. Cross-Site Scripting (XSS):** Issues with XSS may arise, such as incorrect output encoding or inadequate XSS protection.

**4. Cross-Site Request Forgery (CSRF)**: Problems with CSRF may occur, such as incorrect token validation or inadequate CSRF protection.

**Performance Issues**

**1. Slow API Endpoints:** Issues with slow API endpoints may arise, such as incorrect database queries or inadequate caching strategies.

**2. Inadequate Indexing:** Problems with inadequate indexing may occur, such as incorrect indexing or inadequate indexing strategies.

**3. Inadequate Caching:** Issues with inadequate caching may arise, such as incorrect caching strategies or inadequate cache invalidation.

**4. High Server Load:** Problems with high server load may occur, such as incorrect server configuration or inadequate load balancing strategies.

**Future Enhancements:**

**Integrations**

**1. Social Media Integration:** Integrate the system with social media platforms to allow users to register complaints directly from social media.

**2. Email Integration:** Integrate the system with email services to send automated notifications and updates to users and administrators.

**3. SMS Integration:** Integrate the system with SMS services to send automated notifications and updates to users and administrators.

**4. Google Maps Integration:** Integrate the system with Google Maps to allow users to register complaints with location-specific information.

**Artificial Intelligence (AI) and Machine Learning (ML) Integration**

**1. Chatbots:** Integrate chatbots to provide automated support and assistance to users.

**2. Sentiment Analysis:** Integrate sentiment analysis to analyze user feedback and complaints.

**3. Predictive Analytics:** Integrate predictive analytics to predict and prevent potential complaints.

**Security Enhancements**

**1. Two-Factor Authentication (2FA):** Implement 2FA to provide an additional layer of security for user authentication.

**2. Encryption:** Implement encryption to protect sensitive user data.

**3. Regular Security Audits:** Perform regular security audits to identify and address potential security vulnerabilities.

**User Experience (UX) Enhancements**

**1. Mobile App:** Develop a mobile app to provide users with a more convenient and accessible way to register complaints.

**2. Personalized Dashboard**: Implement a personalized dashboard **to** provide users with a customized view of their complaints and updates.

**3. Real-time Updates:** Implement real-time updates to provide users with instant notifications and updates on their complaints.

**Reporting and Analytics**

**1. Customizable Reports: Implement customizable reports to provide administrators with a more detailed and accurate view of complaints and trends.**

**2. Real-time Analytics:** Implement real-time analytics to provide administrators with instant insights and trends on complaints and user b**e**havior.

**3. Data Visualization:** Implement data visualization to provide administrators with a more intuitive and interactive way to view and analyze complaint data.

**Scalability and Performance**

**1. Load Balancing:** Implement load balancing to distribute traffic and improve system performance.

**2. Caching:** Implement caching to reduce the load on the database and improve system performance.

**3. Cloud Hosting:** Consider hosting the system on a cloud platform to improve scalability and performance.Integrations

**Conculsion**

Complaint Management System was developed to enhance the current scenario by using the mobile application. Therefore, Complaint Management System was able to provide several channels for filing the complaint, which enables users to send the complaint easier, and also provides the channel for progress tracking by using the mobile application. Moreover, the system was capable of classifying the complaint and directly sending to the appropriate responsible department, therefore, the system could reduce the cost of hiring the staff and time of the operation. In addition, this system could decrease the duplicate complaints by suggesting the similar complaint to users. Finally, the system generates the data visualization for the summary of complaint data.