**Full Stack Development with MERN**

**Database Design and Development Report**

|  |  |
| --- | --- |
| Date | 20-07-2024 |
| Team ID | SWTID1720177025 |
| Project Name | online complaint registration and management system |
| Maximum Marks |  |

**PROJECT TITLE: ONLINE COMPLAINT REGISTRATION AND MANAGEMENT SYSTEM**

**Date: : 20-07-2024**

**Prepared by: Janani.R.S**

**Objective**

The objective of this report is to outline the database design and implementation details for the "Online Complaint Registration and Management System" project. This includes developing a comprehensive schema, selecting and integrating a suitable DBMS, and ensuring efficient data operations. Additionally, the report will cover measures for data integrity and security, ensuring a robust and scalable system capable of handling large volumes of complaints and user interactions.

**Technologies Used**

* **Database Management System (DBMS):** MongoDB
* **Object-Document Mapper (ODM):** Mongoose

**Design the Database Schema**

The database schema is designed to accommodate the following entities and relationships:

**1. Users**

- Attributes: userID, name, email, password, phone, user\_type (e.g., user, admin), dateRegistered, and lastLogin..

**2. Posts**

- Attributes complaintID, userID , category, description, status, dateSubmitted, dateResolved, and priority.

Top of Form

Bottom of Form

**3. Comments**

- Attributes: commentID, complaintID (references Complaint), userID (references User), commentText, datePosted, and dateUpdated

**Implement the Database using MongoDB**

The MongoDB database is implemented with the following collections and structures:

Database Name: online complaint

1. Collection: users

- Schema:

const userSchema=new mongoose.Schema({

name:String,

email:String,

password:String,

user\_type:String,

phone:Number

})

2. Collection: posts

- Schema:

```

{

\_id: ObjectId,,

content: String,

author: ObjectId (references users),

type:string,

createdAt: Date,

updatedAt: Date

}

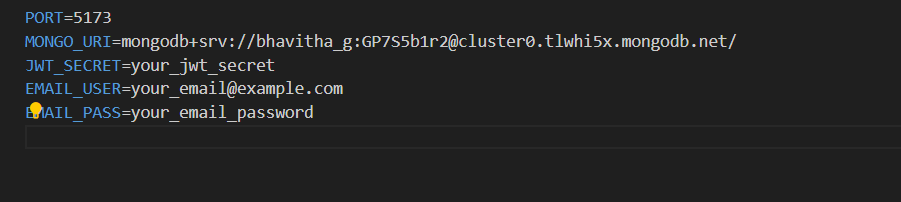
**Integration with Backend**

* Database connection:

Config/db.js



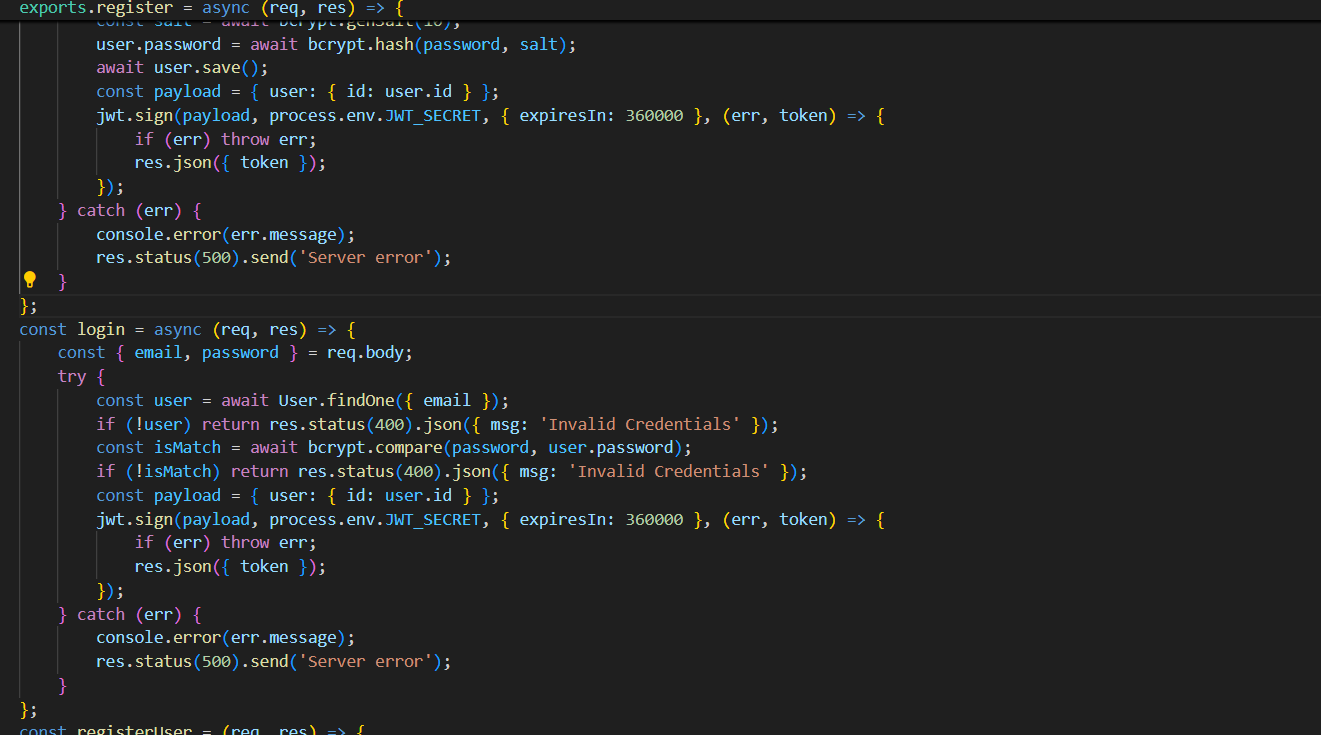
.env



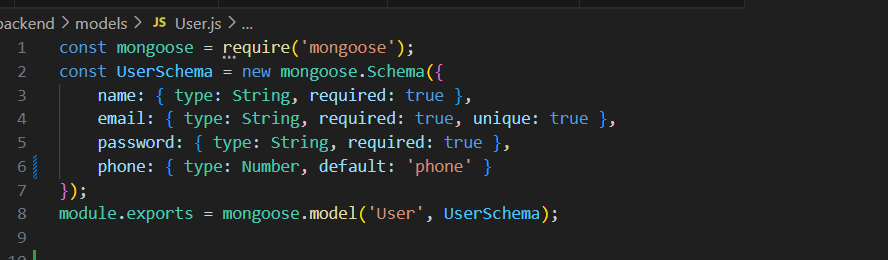
User management database code involve

Controllers/userControl

This involve both login and register



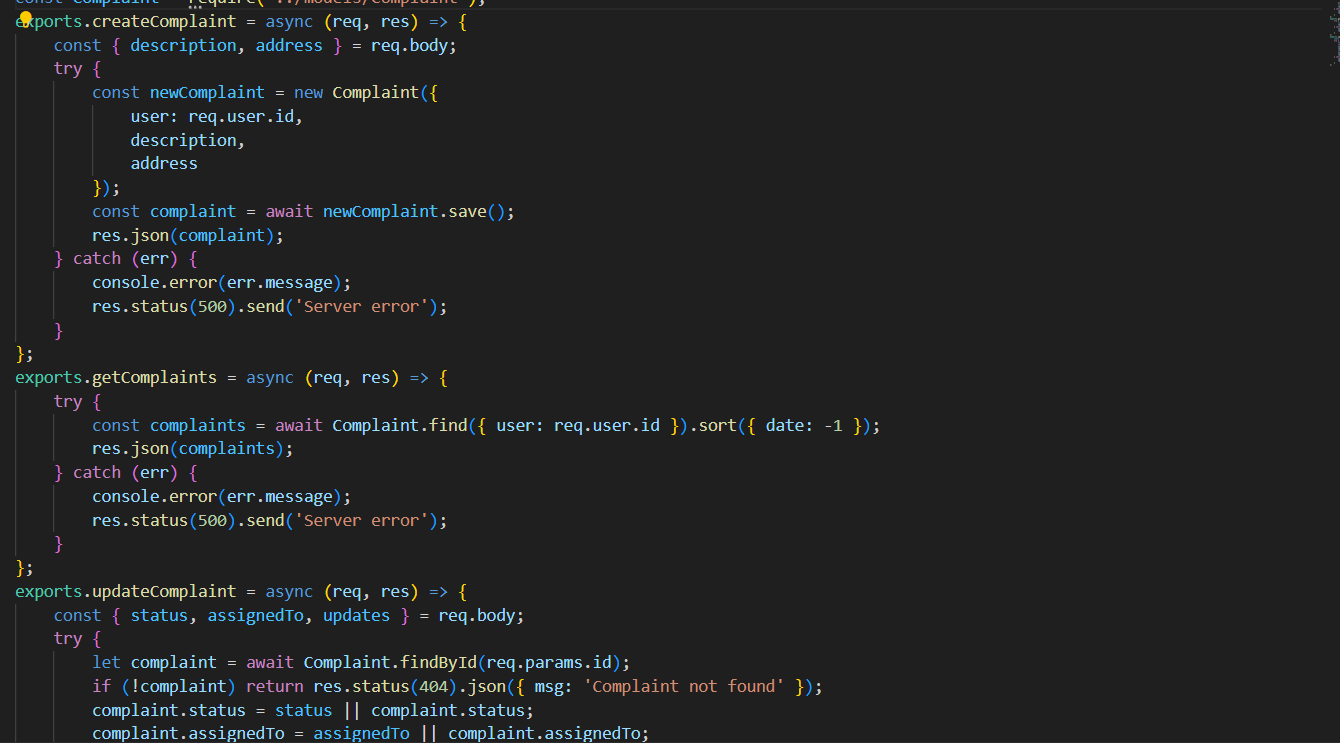
/models and /routes





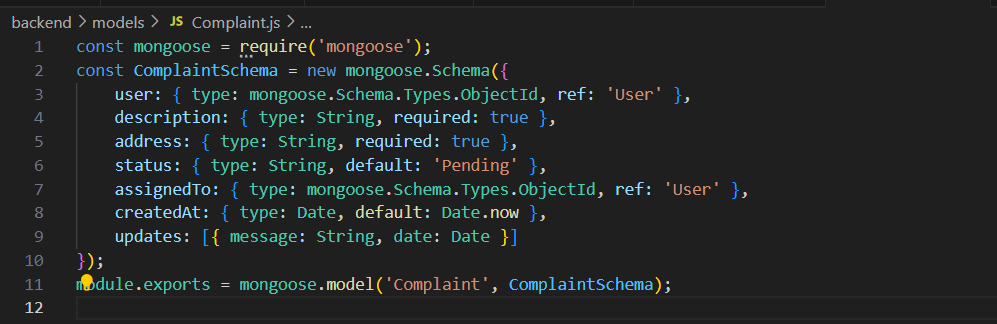
Complaint management involve CRUD operations for management ,add,view with user authentication.

Controllers/ComplaintControl

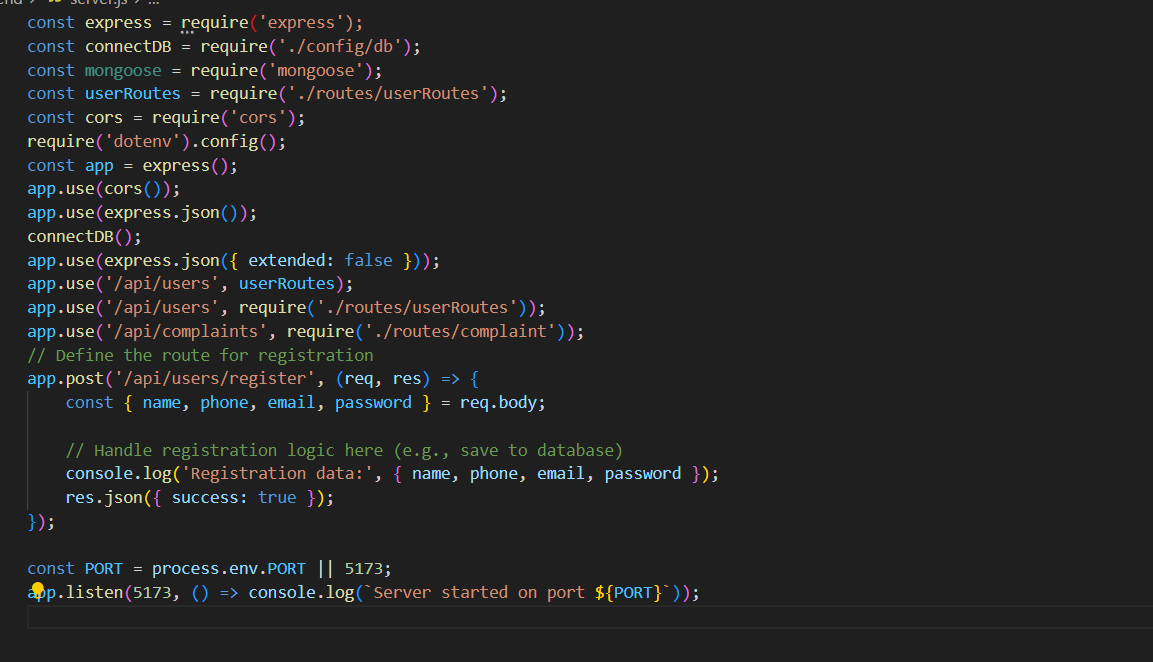


/models and /routes





Entery loop for backend ( /server)



**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 20-07-2024 |
| Team ID | SWTID1720177025 |
| Project Name | Online Complaint Registration And Management System |
| Maximum Marks |  |

**PROJECT TITLE: ONLINE COMPLAINT REGISTRATION AND MANAGEMENT SYSTEM**

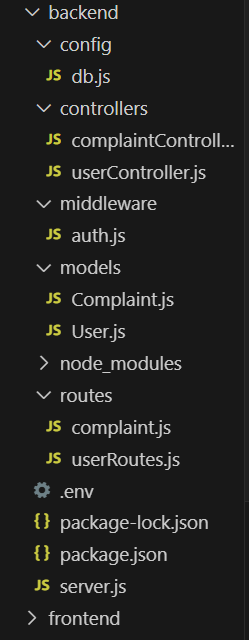
**Date: 20-07-2024  
Prepared By: Guru Prasaath.D**

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Online Complaint Registration And Management System Using Mern project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB

**Project Structure**



**Key Directories and Files**

1. **/Controllers**
   * Contains functions to handle requests and responses.
   * Complaintcontroller.js manages complaints by creating new ones, retrieving existing ones for a specific user, and updating complaint details in a Node.js application using Mongoose
   * usercontroller.js performs registration and login
2. **/models**
   * Includes Mongoose schemas and models for MongoDB collections.
   * Complaint.js defines a Mongoose schema for a **Complaint** model, specifying the structure and default values for complaints stored in a MongoDB database.
   * user.js defines a Mongoose schema for a **User** model, specifying the structure and default values for user accounts stored in a MongoDB database.
3. **/routes**
   * Defines the API endpoints and links them to controller functions.
   * Complaint.js sets up Express routes for creating, retrieving, and updating complaints, using authentication middleware and corresponding controller functions.
   * user.js API endpoints (`/register`, `/login`) in an Express router and links them to corresponding controller functions (`Register`, `Login`) from `userController`. Exports the router for use in the main application.

**API Endpoints**  
A summary of the main API endpoints and their purposes:

**User Authentication**

* **POST** **/register -** Registers a new user
* **GET /login** - Authenticates a user

**Home page**

* **GET /home** - displays quote and buttons
* **GET /about**– Retrieves all details about us

**Complaint registration Page**

* **GET /details** – User describes about their complaint
* **GET /category** – Add the category of the complaint
* **GET /urgency –** Select the urgency of the Complaint to be solved
* **POST/submit –** Retrieves the user complaint and send to the database

**Complaint Status Page**

* **POST/status –** Display the complaint details and the status of the complaint

**Integration with Frontend**  
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

* **User Authentication:** Tokens are passed between frontend and backend to handle authentication.
* **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

**Error Handling and Validation**  
Describe the error handling strategy and validation mechanisms:

* **Error Handling:** Centralized error handling using middleware.



Token Retrieval and Initial Check

Token Verification and Error Handling

* **Validation:** Input validation using libraries like Joi or express-validator.

**Security Considerations**  
Outline the security measures implemented:

* **Authentication:** Secure token-based authentication.

bcryptjs for hashing passwords ,jsonwebtoken for handling JWT

* **Data Encryption:** Encrypt sensitive data at rest and in transit.

**Bcryptjs -**bcryptjs is typically used for hashing passwords rather than encryption. Hashing is a one-way function, meaning you cannot retrieve the original data from the hashed data. This is commonly used for securely storing passwords.