**Team Tapati**

**Module 1: User Authentication and Registration**

### Project: CommUnity Community Management System

This document provides a detailed overview of Module 1: User Authentication and Registration for the CommUnity Community Management System. It outlines the key functionalities, the underlying implementation, and the technologies used to achieve secure user authentication and profile management for residents and administrators.

## Objectives

1. Implement a robust user registration system.
2. Develop JWT-based authentication for secure user login.
3. Enable role-based profile management for residents and administrators.

## **Features Overview**

1. User Registration

○ Allows new users to sign up with their email, password, and role (RESIDENT or ADMIN).

○ Passwords are securely hashed before storage in the database.

1. User Login

○ Authenticates users using their email and password.

○ Issues a JSON Web Token (JWT) upon successful login, enabling stateless authentication.

1. Profile Management

○Provides an interface to manage user profiles, distinguishing between residents and admins.

**Key Components**

### Database Design

The Users table includes the following fields:

● id: Unique identifier for the user.

● email: User's email, serving as the unique username.

● password: Hashed password for secure authentication.

● role: Enum type (RESIDENT or ADMIN) for role-based access control.

Model Layer

#### Users Entity

The Users class is an implementation of Spring Security's UserDetails interface for integrating with the security framework.  
 Key highlights:

● getAuthorities(): Returns the role as ROLE\_ADMIN or ROLE\_RESIDENT.

● getUsername(): Maps the email to the username field for authentication.

#### Role Enum

The Role enum defines the available roles in the system (RESIDENT and ADMIN).

Service Layer

#### AuthenticationService

Handles registration, authentication, and user data retrieval.

1. signup()

○ Registers a new user by saving their email, hashed password, and role to the database.

1. authenticate()

○ Validates user credentials.

○ Generates a JWT token for authenticated users using JwtService.

1. allUsers()

○ Fetches all users for administrative purposes.

#### UserService

Provides additional user-related functionalities:

● getUserByRoleAndEmail(String role, String email): Retrieves a user based on their role and email.

● allUsers(): Returns a list of all registered users.

JWT Implementation

#### JwtService

Manages token generation, validation, and claims extraction.

1. Token Generation

○ Includes user roles in the claims for role-based access control.

1. Token Validation

○ Ensures the token is valid and not expired.

#### JwtUtil

Provides helper methods for generating and validating tokens.

Repository Layer

#### UserRepository

An interface extending JpaRepository for CRUD operations on Users.

● Includes methods for finding users by email and role.

Security Configuration

Spring Security is configured to:

1. Authenticate requests using JWT tokens.
2. Authorize users based on their roles.

**Endpoints**

| HTTP Method | Endpoint | Description |
| --- | --- | --- |
| POST | /auth/signup | Registers a new user. |
| POST | /auth/login | Authenticates a user and returns a JWT. |
| GET | /users/all | Fetches a list of all users (Admin only). |

**Technologies Used**

● Spring Boot: Framework for rapid development.

● Spring Security: Provides authentication and authorisation mechanisms.

● JWT: Ensures stateless, secure authentication.

● Hibernate: Manages ORM and database interactions.

● MySQL: Database for storing user data.

**Code Implementation**

### Sample User Registration Request

Endpoint: /auth/signup  
 Request Body:

JSON

{

"email": "user@example.com",

"password": "securePassword",

"role": "RESIDENT"

}

### Sample User Login Request

Endpoint: /auth/login  
 Request Body:

JSON

{

"email": "user@example.com",

"password": "securePassword"

}

Response:

JSON

{

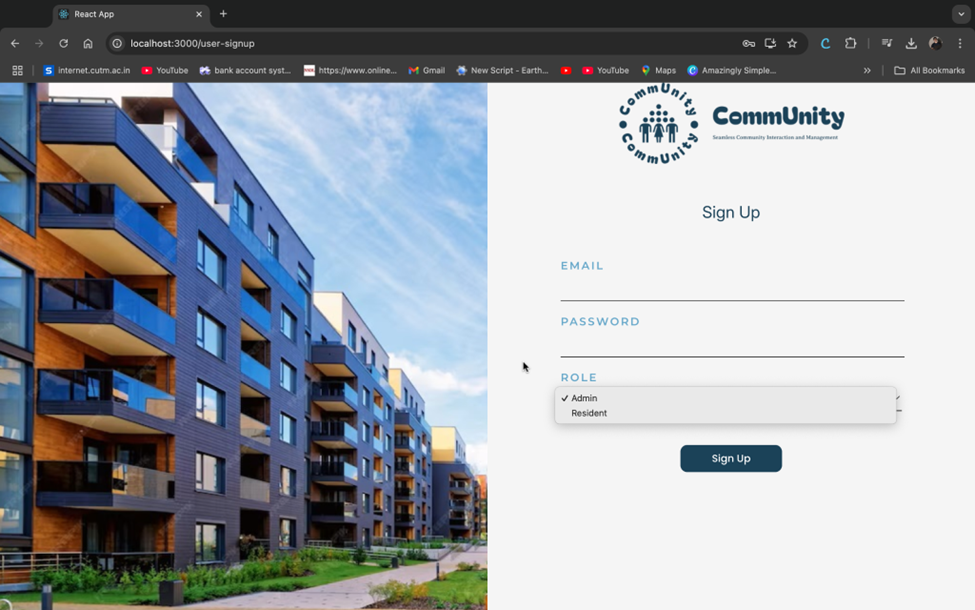
"token": "eyJhbGciOiJIUzI1NiIsInR..."

}

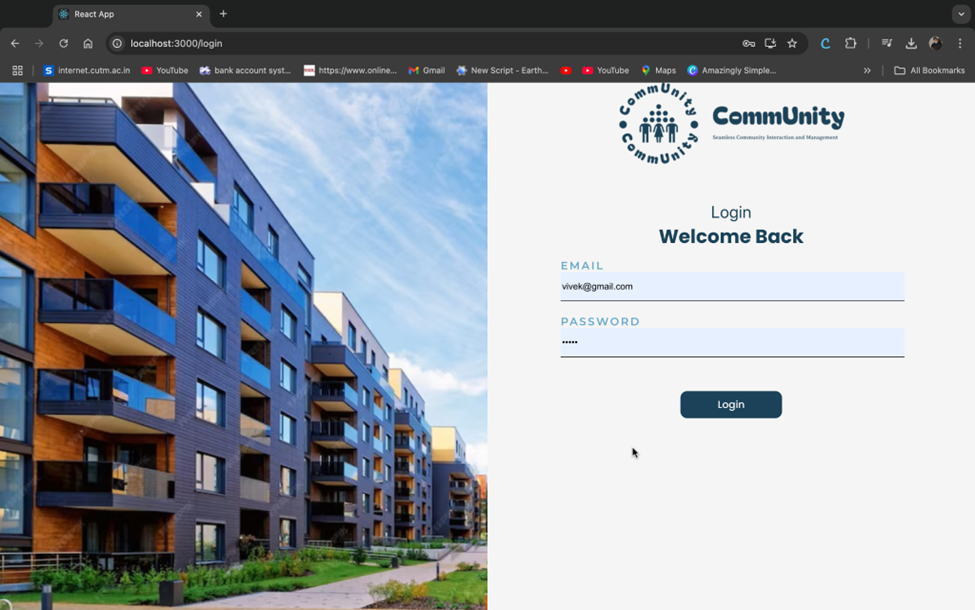
**Output Screenshot**

● User Login and Registration Page

1. Registration form with fields for email, password, and role.



1. Login form with fields for email and password.



**TEAM TAPATHI**

**TEAM MEMBERS:**

1. Vivek Singh
2. Swathi Gottipati
3. Meghana Katakam
4. Divya Rakshida
5. Tikeshwari Patel

**Milestone 2: WEEKS 3-4**

**Module 2: Backend-to-Frontend Request and Response Payload Details**

1. **Request Payload:** Frontend sends structured data (e.g., JSON) containing necessary fields like identifiers or user inputs (e.g., email, password, role) for backend processing.
2. **Response Payload:** Backend responds with status codes (e.g., 200 OK, 400 Bad Request), success messages, or data (e.g., user profile, confirmation message).
3. **Validation and Security:** Backend validates request data for integrity and security, ensuring the response contains only sanitized and necessary information.
4. **Error Handling:** Backend includes clear error messages and codes in the response to help the frontend handle errors effectively

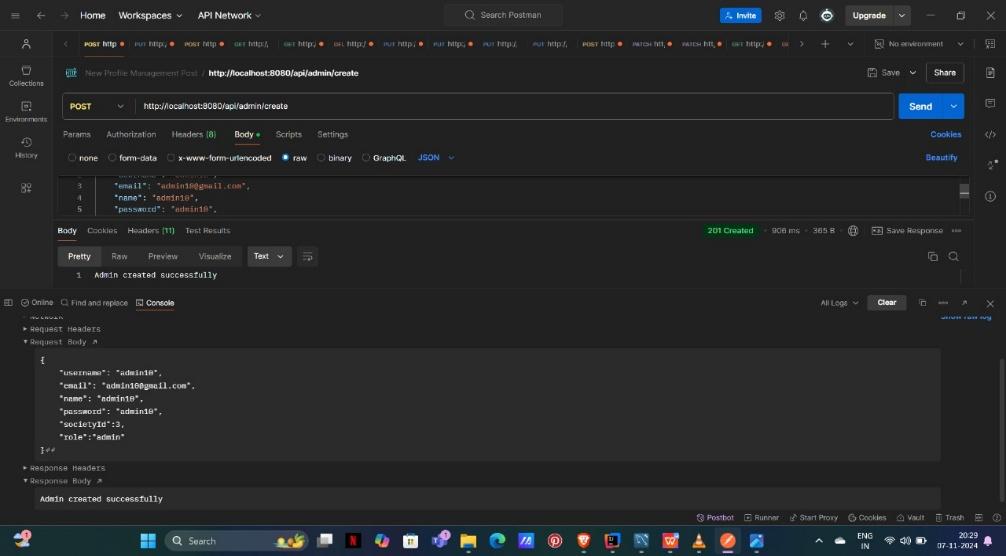
Milestone 2 focuses on building a robust system for Admin Registration, Resident Registration, Admin Login, Resident Login, Admin Profile, Resident Profile, Society Registration, Society Updation. This includes CRUD (Create, Read, Update, Delete) operations for admins, societies, and residents, along with role management and secure profile updates.

**---------------------------------------------------------------------------------**

**REGISTRATION RESPONSE**

**1.Admin Registration:**

The image shows a POST request in Postman to http://localhost:8080/api/admin/create, with a request body containing JSON data for creating an admin. The server responds with a status code 201 Created and the message "Admin created successfully."



Example: {

"username": "admin1",

"email": "admin1@gmail.com",

"name": "admin1",

"password": "admin1",

"societyId":3,

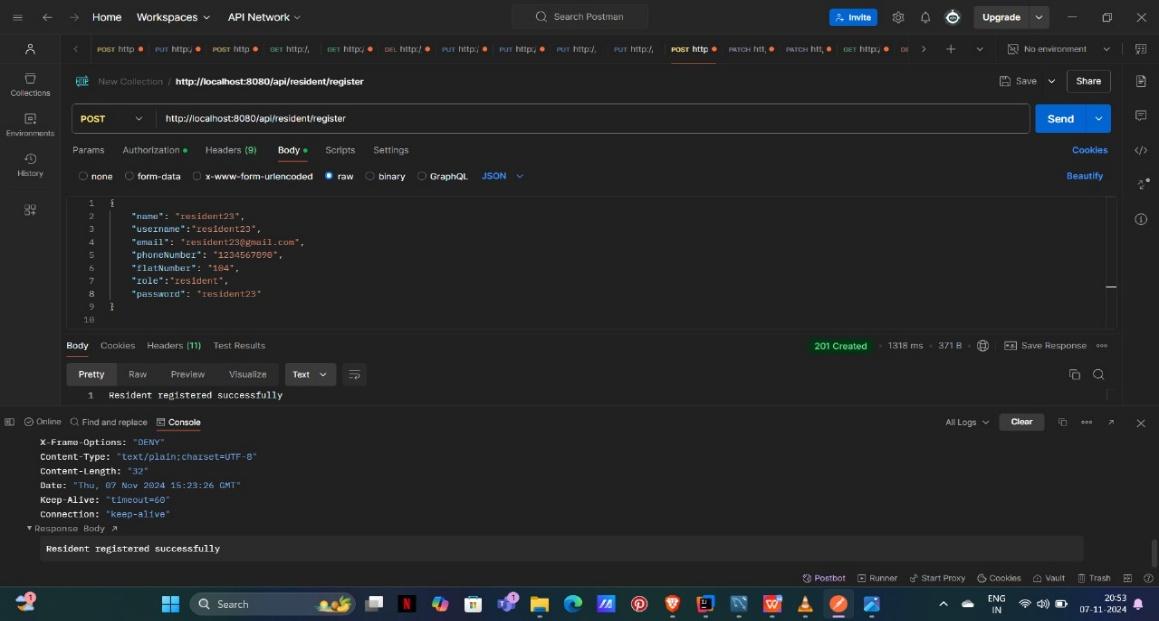
"role":"admin"

}

RESPONSE BODY:(201 Created) Admin created successfully

**2.Resident Registration:**

The image shows a POST request being made in Postman to the endpoint http://localhost:8080/api/resident/register. The request body contains JSON data with details for registering a new resident, such as name, username, email, phone number, flat number, role, and password. The server responds with a 201 Created status and the message "Resident registered successfully."



Example:

{

"name": "resident21",

"username":"resident21",

"email": "resident21@gmail.com",

"phoneNumber": "1234567890",

"flatNumber": "102",

"role":"resident",

"password": "resident21"

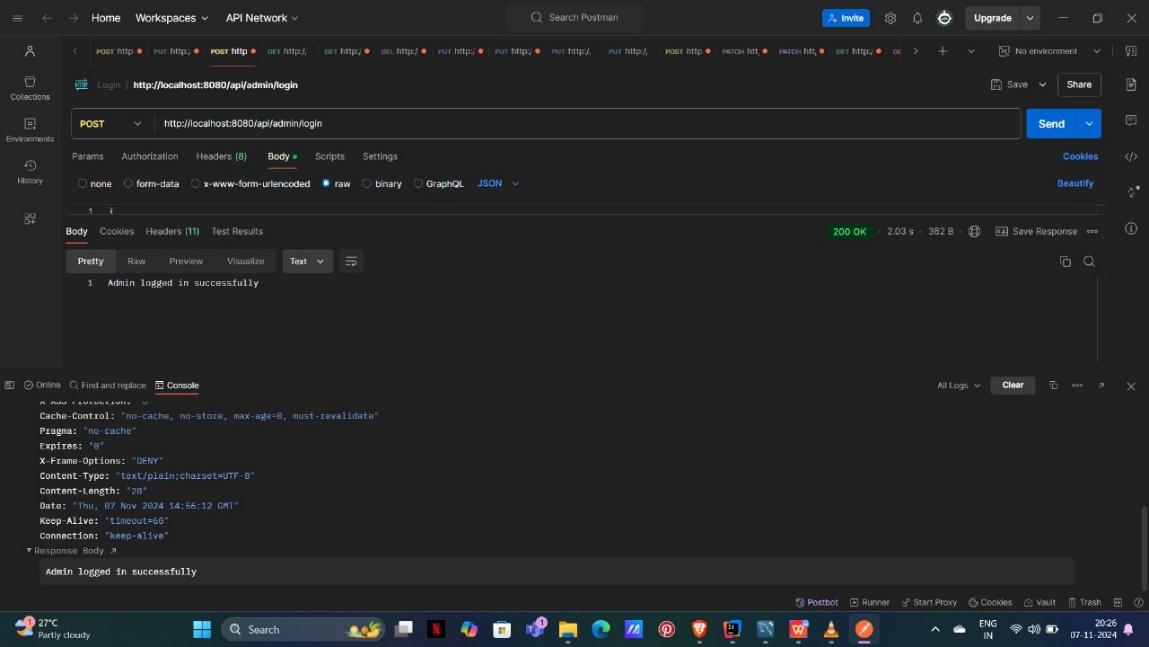
}

RESPONSE PAYLOAD:(201 Created) Resident registered successfully

**LOGIN RESPONSES**

**3.Admin Login:**

The image shows a POST request being sent to a login endpoint. The response indicates successful login, with the message "Admin logged in successfully" and various HTTP headers.



Example:

{

"username": "admin6",

"email": "admin6@gmail.com",

"name": "admin6",

"password": "admin6",

"societyId":3,

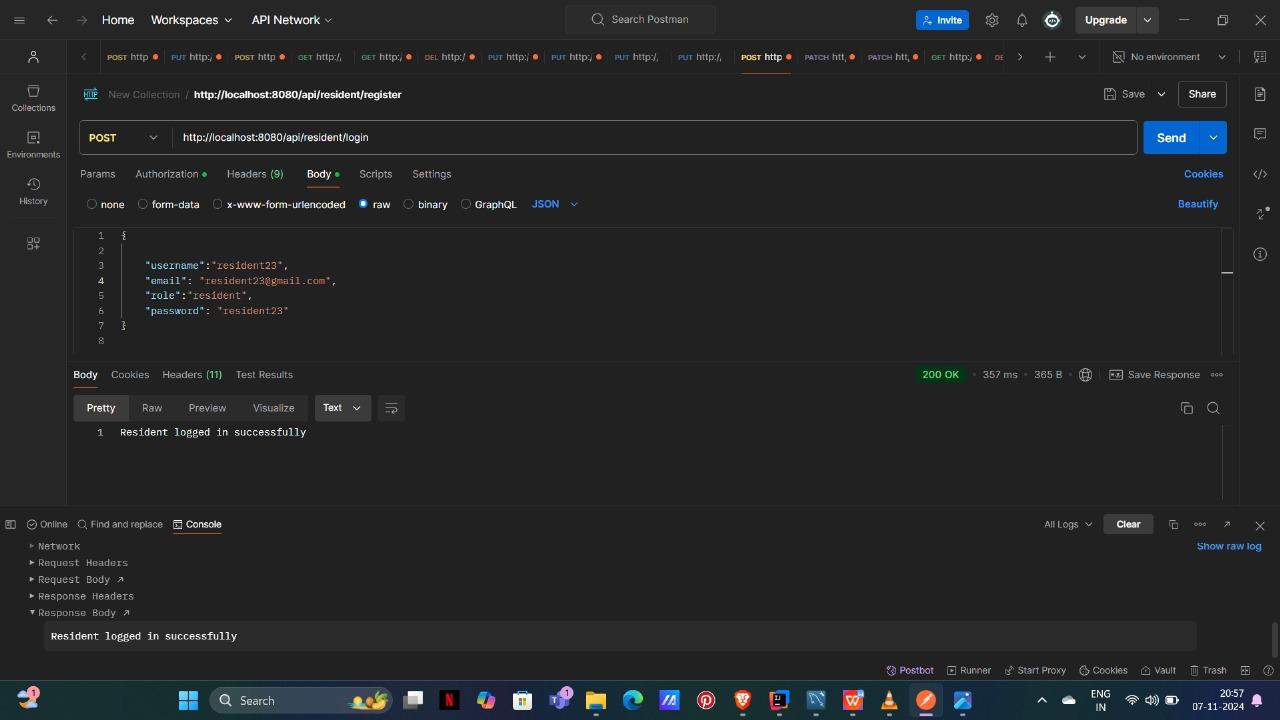
"role":"admin"

}

RESPONSE PAYLOAD: (200 Ok) Admin created successfully

**4.Resident Login:**

The image shows a POST request being sent to a login endpoint. The response indicates successful login, with the message "Resident logged in successfully" and various HTTP headers.



Example:

{

"username": "resident009",

"role":"RESIDENT",

"email": "resident009@gmail.com",

"password": "resident009"

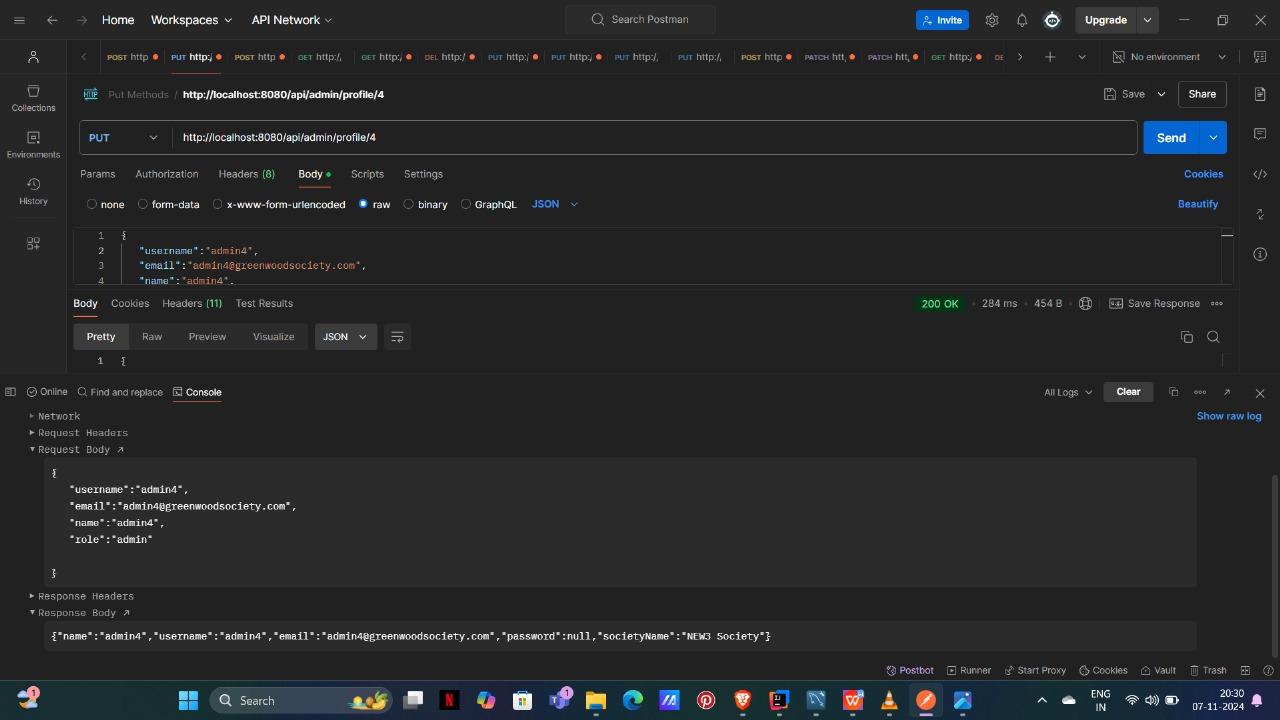
}

RESPONSE PAYLOAD:(200 ok) Resident logged in successfully

**UPDATING PROFILE**

**5.Admin profile:**

The image shows a Postman interface with a successful PUT request to update user profile data. The response status is 200 OK.



Example:

{

"username":"admin04",

"email":"admin4@greenwoodsociety.com",

"name":"admin004",// changed name from admin4 to admin004

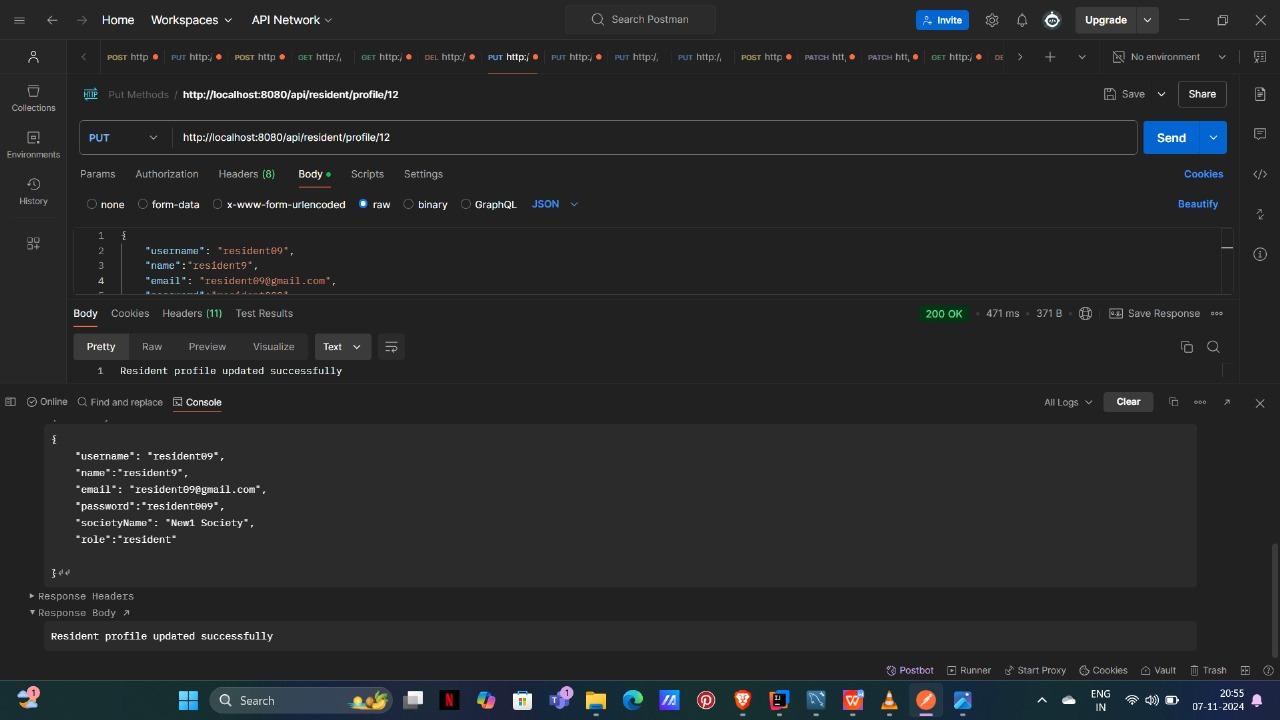
"role":"admin"

}

RESPONSE PAYLOAD: (200 OK) Admin profile updated successfully

**6.Resident profile:**

The image shows a Postman interface with a successful PUT request to update a resident profile at http://localhost:8080/api/resident/profile/12. The response status is 200 OK, with a message confirming the profile update.



Example:

{

"username": "resident009",

"name":"resident9",

"email": "resident009@gmail.com",

"societyName": "New1 Society",

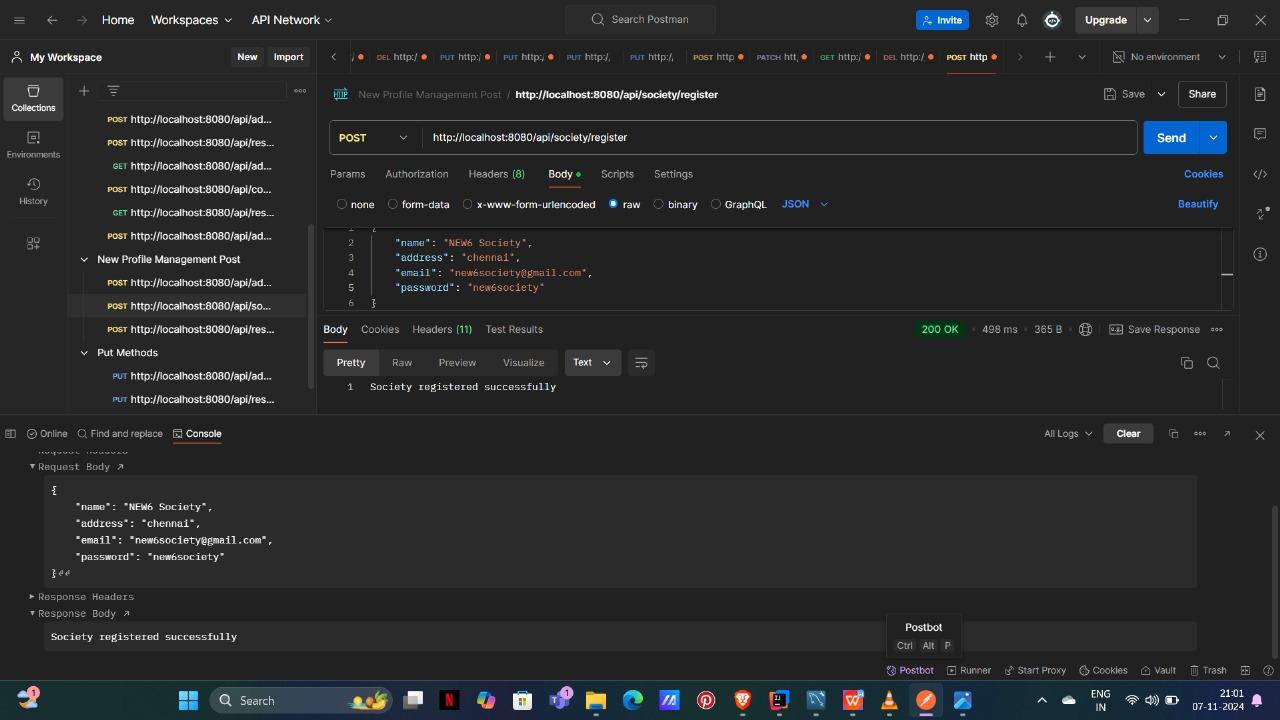
"role":"resident"

}

RESPONSE PAYLOAD:(200 OK) Resident profile updated successfully

**SOCIETY RESPONSE PAYLOADS**

**7.Society Registration:**

The image shows a Postman interface with a successful POST request to register a new society at http://localhost:8080/api/society/register. The response status is 200 OK, confirming the society registration.

Example:

{

"name": "NEW5 Society",

"address": "chennai",

"email": "new5society@gmail.com",

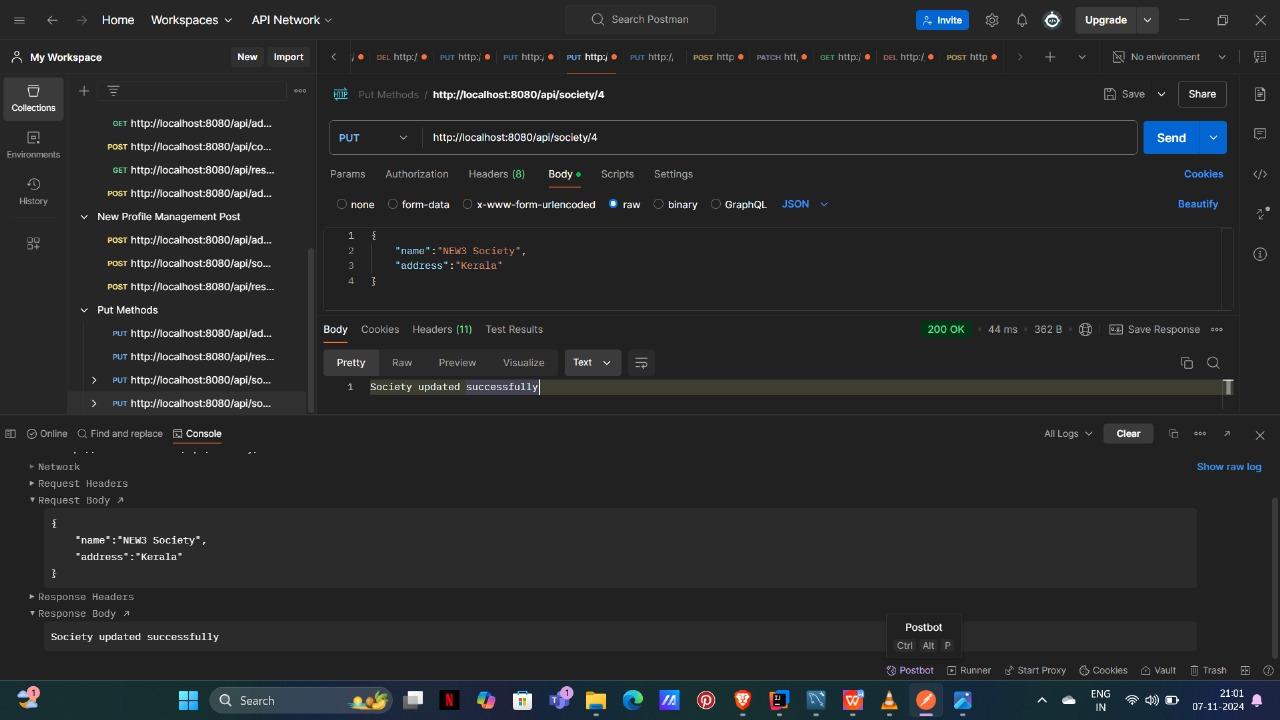
"password": "new5society"

}

RESPONSE PAYLOAD:( 200 OK) Society registered successfully

**8.Society Updatation:**

The image shows a screenshot of Postman, an API development environment. It appears to be making a PUT request to update a "Society" with the ID "4". The request body contains new values for "name" and "address". The response indicates a successful update.



Example:

{

"name":"NEW3 Society", "address":"Kerala"//address was chennai and changed to kerala

}

RESPONSE PAYLOAD:(200 OK) Society updated successfully

**TEAM TAPATI**

**MILESTONE-3**

**TASK ASSIGNED :** Team Tapati was assigned to develop backend for Notice Management ,complaint and request service playloads.

**Team Members :** Divya,Pravallika,Swathi,TikeshwariPatel,Meghana

**Team Contributions :**

Divya and Pravallika developed a whole notices part.

**NOTICE MANAGEMENT**

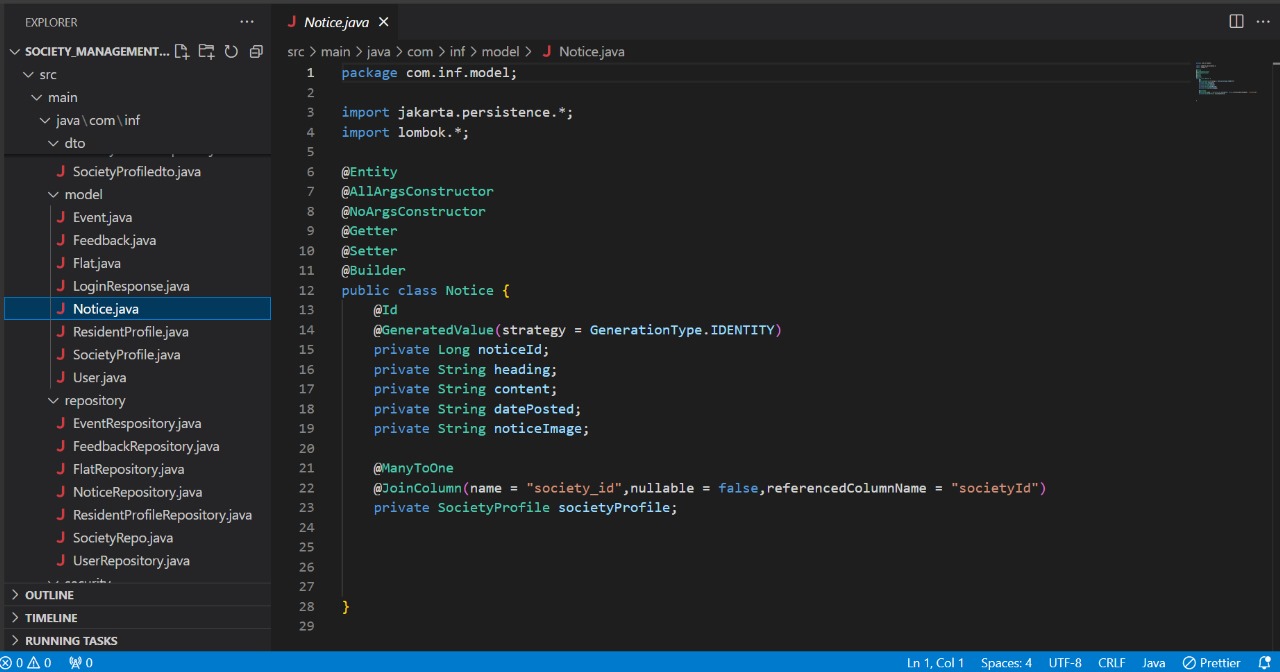
**INTRODUCTION :**

The Notice Functionality allows administrators to create, update, and delete notices for the residents of a society. Residents can view these notices via their dashboard, ensuring effective communication within the community.

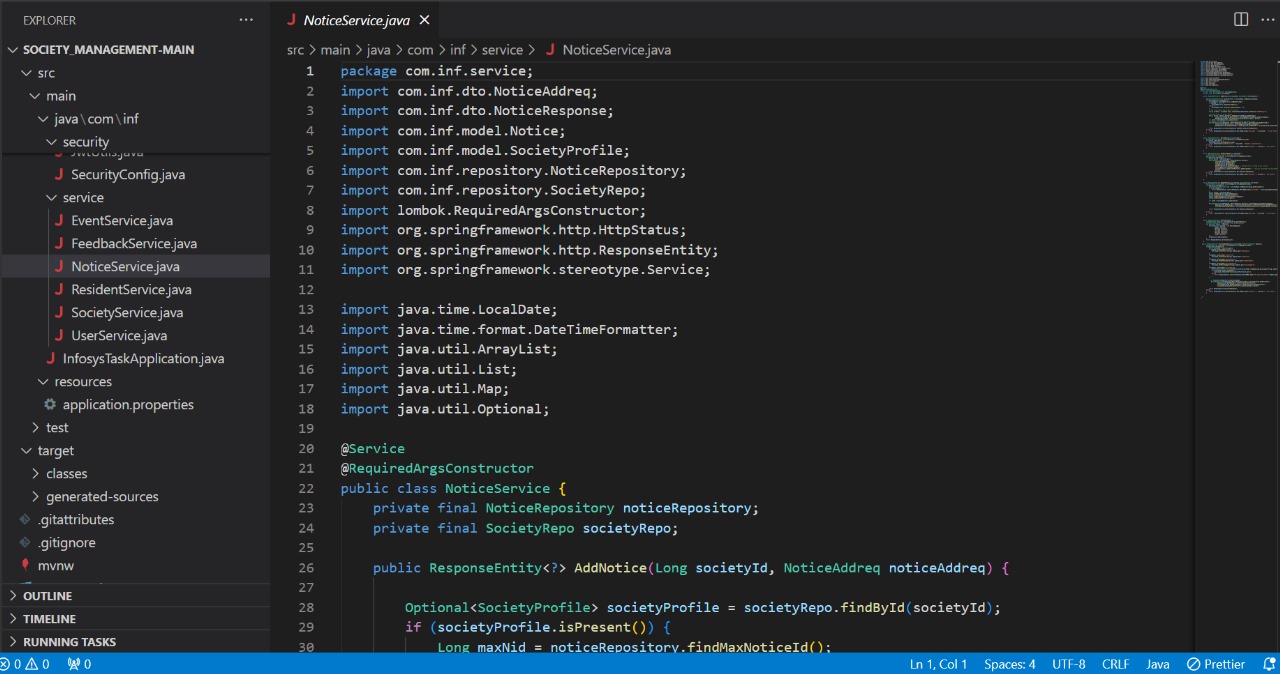
**ROLES INVOLVED :**

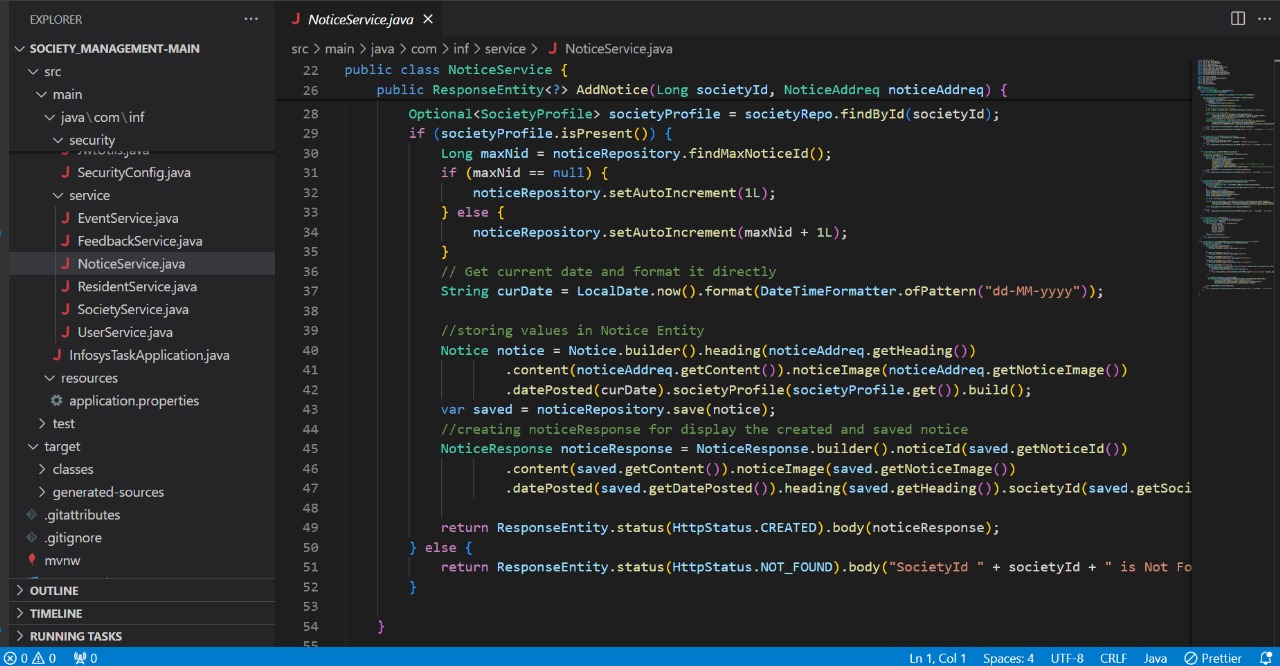
* **Admin :** Admin can add new notices,update or delete existing notices and manage notices.
* **RESIDENT :** Residents can view notices published by the admin.

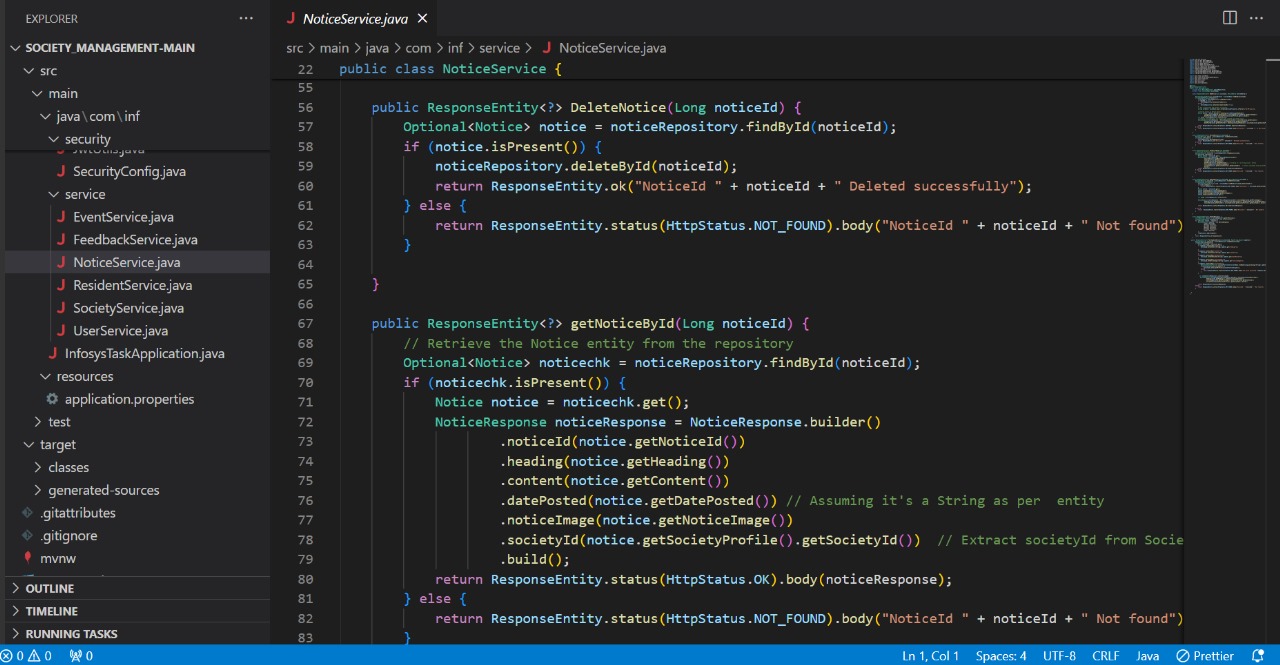
**NOTICE ENTITY :** Notice entity involves noticed,headings,content,date posted,notice image and society profile . This image provides the code for Notice entities.

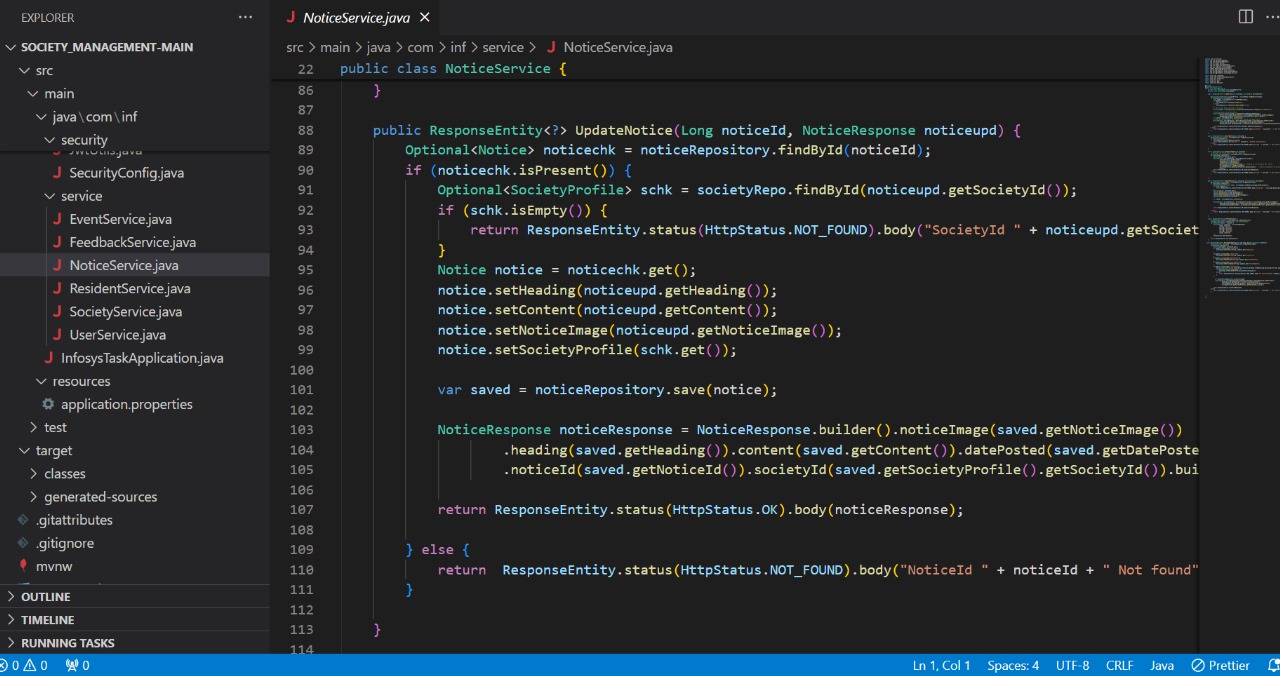
****

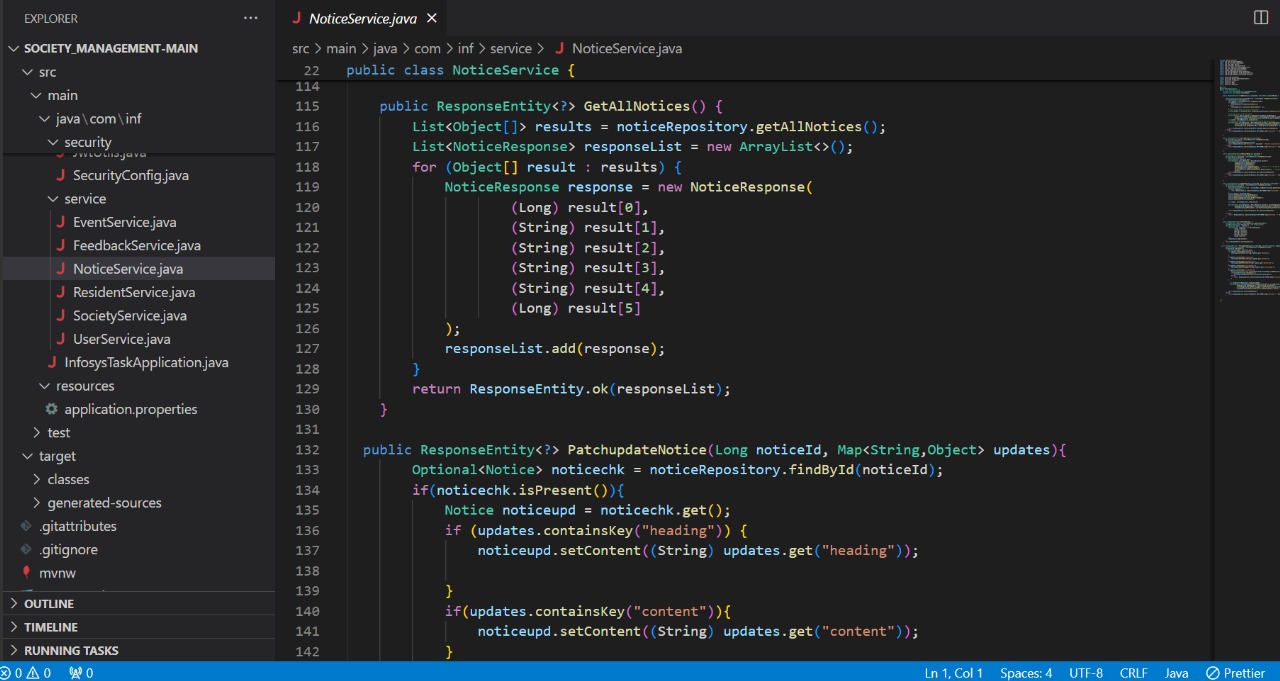
**NOTICE SERVICES :** These notice services handles all the logics for managing notices like adding,updating and deleting.These images provides the overall code for the services.

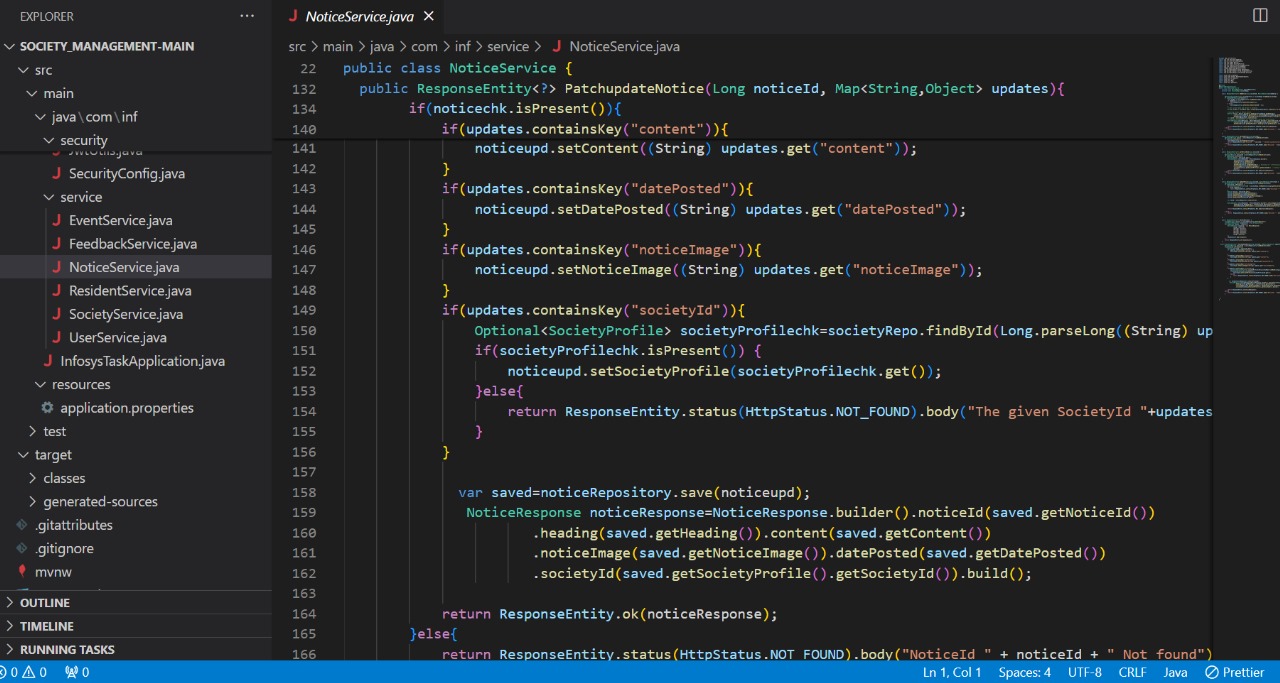




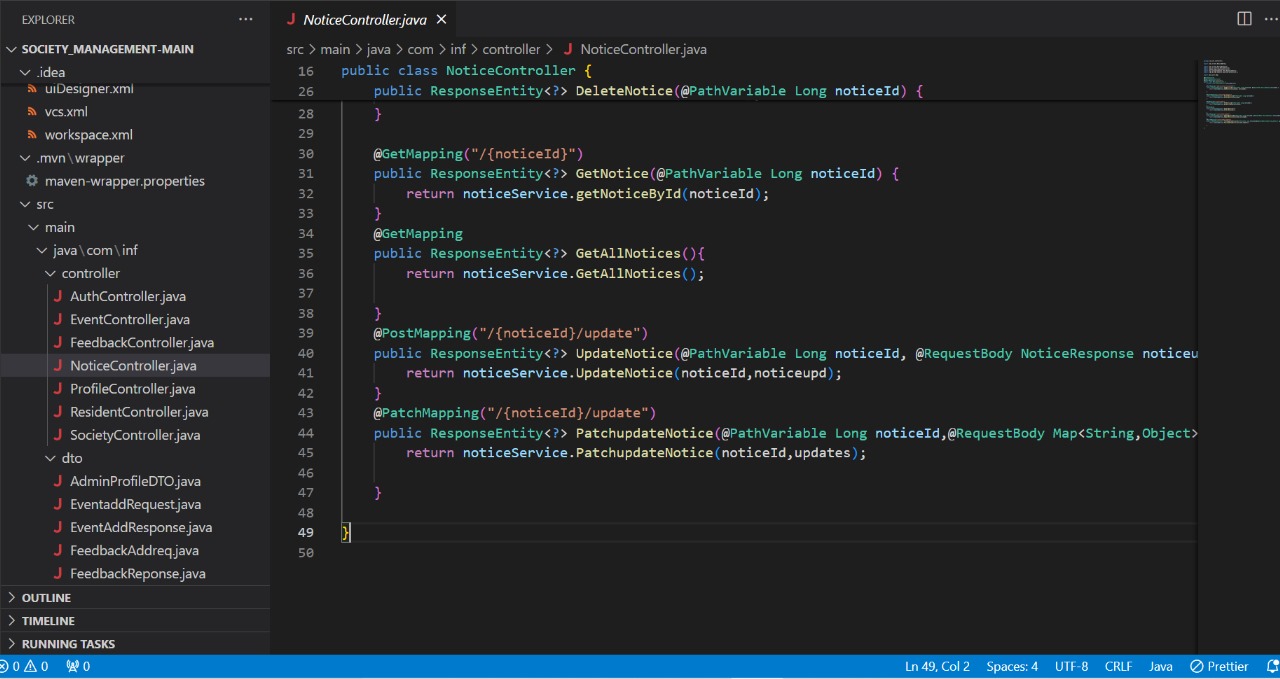








**Notice Controller :** Notice controller provides REST API’s for the frontend to interact with notice system.This image provides a detailed code for notice controller.



**API’s Endpoints :**

* **POST :** /notices/add/society/{societyId}

Adding a new notice.

* **GET** : /notices/add/society/{societyId}

Gets a specific notice by ID.

* **GET :** /notices

Get all notices.

* **POST :** /notices/{noticeId}/update

Update a notice entirely.

* **PATCH :** /notices/{noticeId}/update

Update specific fields in a notice.

* **DELETE :** /notices/{noticeId}/delete

Deletes a notice.

**DATABASE :** Database contains a **Notice Table** which contains fields noticed,headings,contents,date posted,notice image,society profile.

When invalid details are given for noticed or societyid then notice not found and society id not found error wil be occurred.

**COMPLAINT AND REQUEST SERVICE MANAGEMENT**

**Team contributions :** Divya and Pravallikadone the complaints payload and the Request Service playloads.

* **Submit Complaint**

**Endpoint:**POST /complaints/submit

**Request Payload :**

{

"societyId": "1",

"residentId": "1",

"name": "John",

"flatNo": "A101",

"title": "Water Leakage Issue",

"description": "There is water leakage in my kitchen ceiling."

}

**Response Payload:**

{

"complaintId": "101",

"name": "John",

"message": "Complaint submitted successfully."

}

## **Get Complaint Details**

**Endpoint:**GET /complaints/{complaintId}

**complaintId**: Unique ID of the complaint.

**Response Payload:**

{

"complaintId": "101",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "Water Leakage Issue",

"description": "There is water leakage in my kitchen ceiling.",

"status": "IN\_PROGRESS",

"societyId": "1"

}

* **Update Complaint**

**Endpoint:**PATCH /complaints/{complaintId}/update

**Request Payload:**

{

"residentId": "1",

"name": "John",

"flatNo": "A101",

"title": "Water Leakage Issue",

"description": "There is water leakage in my kitchen ceiling.",

"status": "COMPLETED"

}

**Response Payload:**

{

"complaintId": "101",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "Water Leakage Issue",

"description": "There is water leakage in my kitchen ceiling.",

"status": "COMPLETED",

"societyId": "1"

}

## **Submit Service Request**

**Endpoint:**POST /service-requests/submit

**Request Payload:**

{

"societyId": "1",

"residentId": "1",

"name": "John",

"flatNo": "A101",

"title": "AC Maintenance Request",

"description": "The air conditioner in my flat is not cooling properly."

}

**Response Payload:**

{

"requestId": "201",

"name": "John",

"message": "Service request submitted successfully."

}

## **Get Service Request Details**

**Endpoint :** GET /service-requests/{requestId}

**requestId :** Unique ID of the service request.

**Response Payload:**

{

"requestId": "201",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "AC Maintenance Request",

"description": "The air conditioner in my flat is not cooling properly.",

"status": "IN\_PROGRESS",

"societyId": "1",

"assignedVendor": "Cool Breeze Services"

}

* **Update Service Request**

**Endpoint :** PATCH /service-requests/{requestId}/update

**Request Payload:**

{

"residentId": "1",

"name": "John",

"flatNo": "A101",

"title": "AC Maintenance Request",

"description": "The air conditioner in my flat is not cooling properly.",

"status": "COMPLETED",

"assignedVendor": "Cool Breeze Services"

}

**Response Payload:**

{

"requestId": "201",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "AC Maintenance Request",

"description": "The air conditioner in my flat is not cooling properly.",

"status": "COMPLETED",

"societyId": "1",

"assignedVendor": "Cool Breeze Services"

}

* **Get All Complaints**

**Endpoint:** GET /complaints

**Response Payload:**

[

{

"complaintId": "101",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "Water Leakage Issue",

"description": "There is water leakage in my kitchen ceiling.",

"status": "COMPLETED",

"societyId": "1"

},

{

"complaintId": "102",

"residentId": "2",

"flatNo": "A102",

"name": "Ram",

"title": "Power Failure",

"description": "There is no power in my flat since morning.",

"status": "OPEN",

"societyId": "1"

}

]

* **Get All Service Requests**

**Endpoint:** GET /service-requests

**Response Payload:**

[

{

"requestId": "201",

"residentId": "1",

"flatNo": "A101",

"name": "John",

"title": "AC Maintenance Request",

"description": "The air conditioner in my flat is not cooling properly.",

"status": "COMPLETED",

"societyId": "1",

"assignedVendor": "Cool Breeze Services"

},

{

"requestId": "202",

"residentId": "3",

"flatNo": "A202",

"name": "Sita",

"title": "Plumbing Work",

"description": "The bathroom tap is leaking continuously.",

"status": "IN\_PROGRESS",

"societyId": "2",

"assignedVendor": "QuickFix Plumbing"

}

]

**TEAM TAPATI**

**TASK ASSIGNED:** Team Tapati was assigned to develop **playloads for vendor management and Billing system**.

**Team members :** Swathi,Divya,Pravallika,Tikkeshwari Patel,Meghana

**TEAM CONTRIBUTIONS :**

Divya and Pravallika developed playloads .

**MILESTONE- 4**

**INTRODUCTION:**

**VENDOR MANAGEMENT :** Managing vendors for service requests and complaints

**Billing and Payment System :**  Generating bills, tracking payments, and handling transactions.

**ROLES PROVIDED :**

1. ADMIN : Has responsibilities to add ,update and assign vendors.And generate bills for flats,view and manage payments.
2. RESIDENT : Views assigned vendors for service requests or complaints, View and pay bills online.

**API Overview**

**1.Vendor Management API’s**

* **Add vendor**

**Method :** By using post method we can add vendor.

**Endpoint :** /requestservices/add/vendor/society/{societyId}

* **Assigning Vendor to Request**

**Method :** By using PATCH method we can assign vendor to request.

**Endpoint :** /requestservices/vendor/assign/request/{requestId}

* **Get Vendor Details**

**Method :** By using GET method we can get details of vendor.

**Endpoint :** /requestservices/vendor/{vendorId}

* **Delete Vendor**

**Method :** By using DELETE method we can delete vendor.

**Endpoint :** /requestservices/vendor/{vendorId}/delete

**2.Billing API’s**

* **Generate Bill for Flats**

**Method :** By using POST method we can post details of residents.

**Endpoint :** /bills

* **Get Payment Details by ID**

**Method :** By using GET method we can get the payment details.

**Endpoint :** /billings/payments/{paymentId}

* **Get All Payments by Society**

**Method :** By using GET method we can get the payment details.

**Endpoint :** /billings/payments/society/{societyId}

* **Process Payment**

**Method :** By using POST we can pay the bill.

**Endpoint :** /billings/paynow

**Request and Response playloads**

* **Add Vendor**

**Request:**

{

"name": "Ram",

"company": "Ram Water Service",

"email": "ram@gmail.com",

"service": "water can",

"phoneNo": "123456890"

}

**Response:**

{

"message": "Vendor has been added successfully",

"vendorId": "1"

}

* **Generate Bill**

**Request:**

{

"paymentId": 12345,

"societyId": 1,

"amount": "550.00",

"status": "pending"

}

**Response:**

{

"paymentId": 12345,

"flatNo": "A101",

"societyId": 1,

"amount": "550.00",

"status": "pending",

"paymentDate": null,

"receiptNo": null

}

* **Payment**

**Request:**

{

"paymentId": 12345

}

**Response:**

{

"paymentId": 12345,

"status": "completed",

"receiptNo": "RE123456"

}

These provides Clear and efficient tracking of vendor actions and payment statuses.