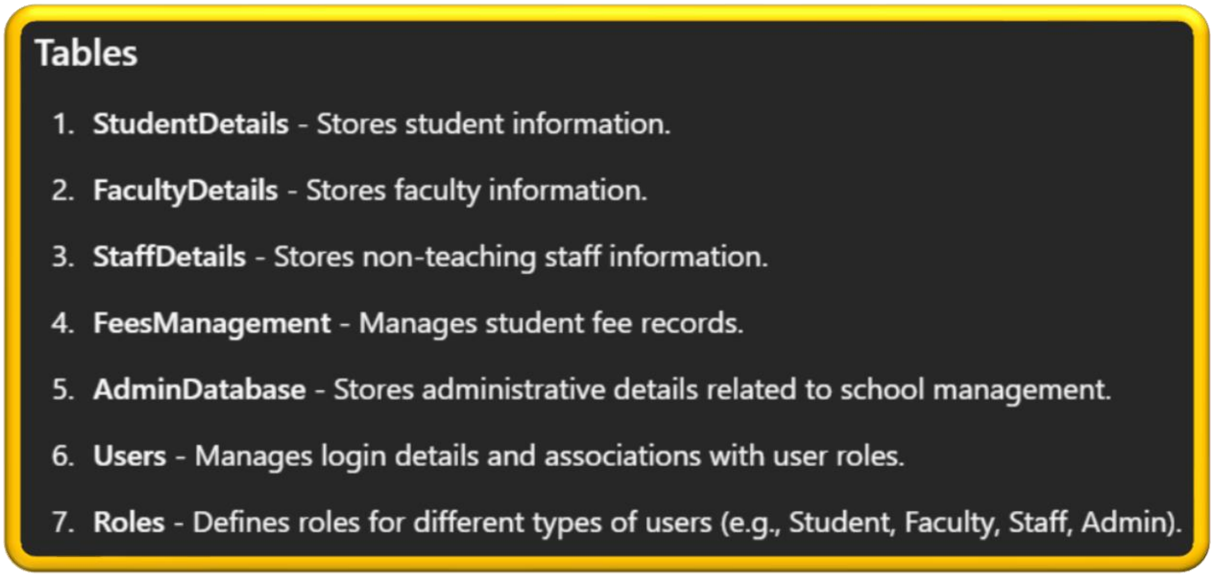
School Management System Usecase – Spring Boot Application

1. **School Management Usecase**

   Questions: The school management system will be able to handle new registration of students, teachers and staff. It also helps to maintain the fees structure of students etc. The relationships between tables are designed to capture real-world associations, like teachers belonging to particular department, multiple students linked to same teacher, and so on. As an initial MVP you are required to develop a restful API backend application in springboot.

   Here is the requirement for the application

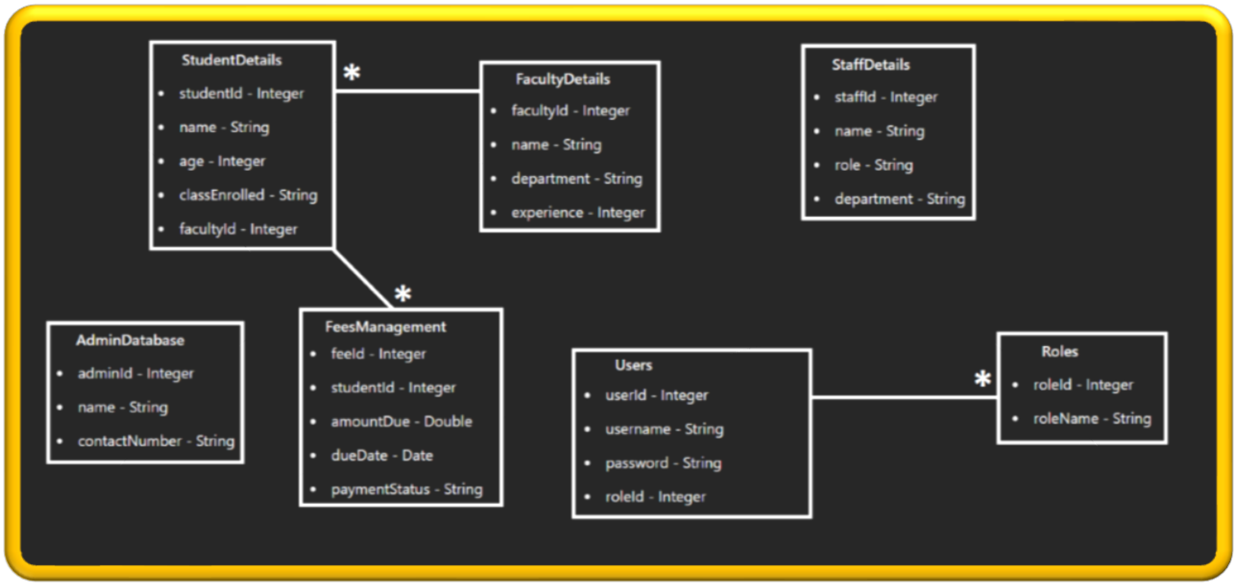
   A. Databases present in the application



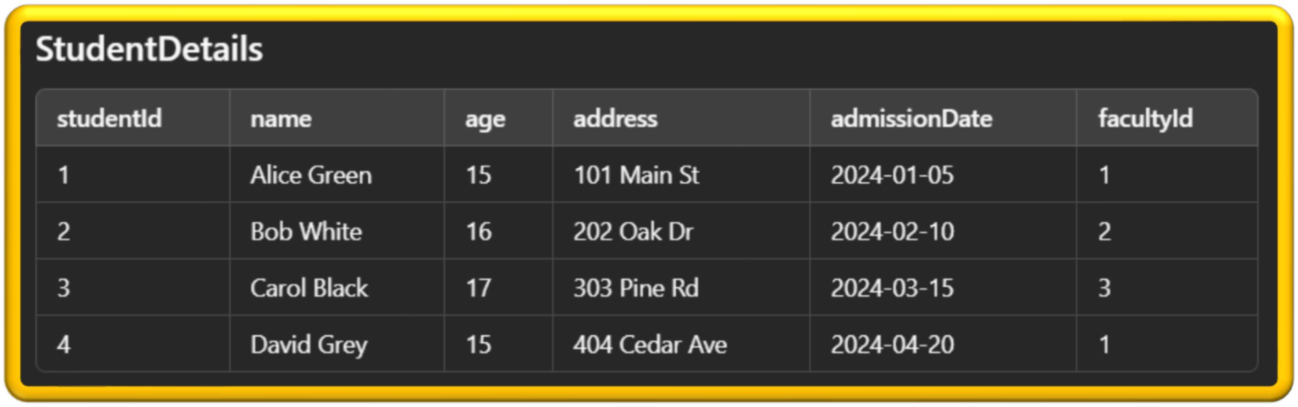
Tables

1. **StudentDetails** - Stores student information.
2. **FacultyDetails** - Stores faculty information.
3. **StaffDetails** - Stores non-teaching staff information.
4. **FeesManagement** - Manages student fee records.
5. **AdminDatabase** - Stores administrative details related to school management.
6. **Users** - Manages login details and associations with user roles.
7. **Roles** - Defines roles for different types of users (e.g., Student, Faculty, Staff, Admin).

   B. Relationship between Databases present in the application

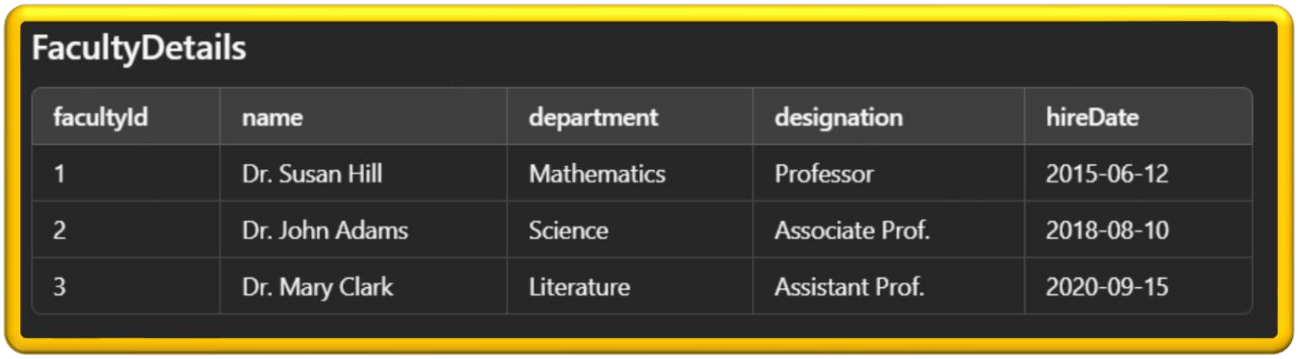


C. Populate the table with below sample data



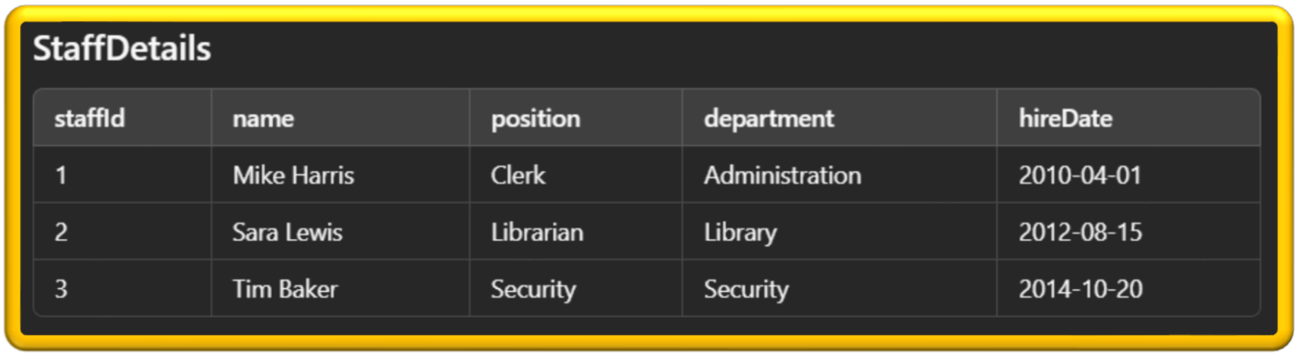
StudentDetails

studentId	name	age	address	admissionDate	facultyId
1	Alice Green	15	101 Main St	2024-01-05	1
2	Bob White	16	202 Oak Dr	2024-02-10	2
3	Carol Black	17	303 Pine Rd	2024-03-15	3
4	David Grey	15	404 Cedar Ave	2024-04-20	1



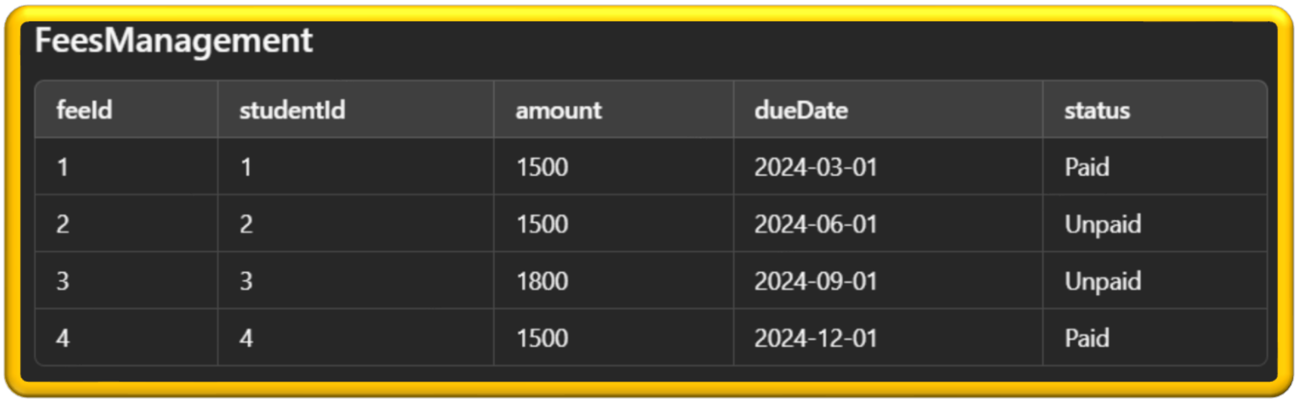
FacultyDetails

facultyId	name	department	designation	hireDate
1	Dr. Susan Hill	Mathematics	Professor	2015-06-12
2	Dr. John Adams	Science	Associate Prof.	2018-08-10
3	Dr. Mary Clark	Literature	Assistant Prof.	2020-09-15



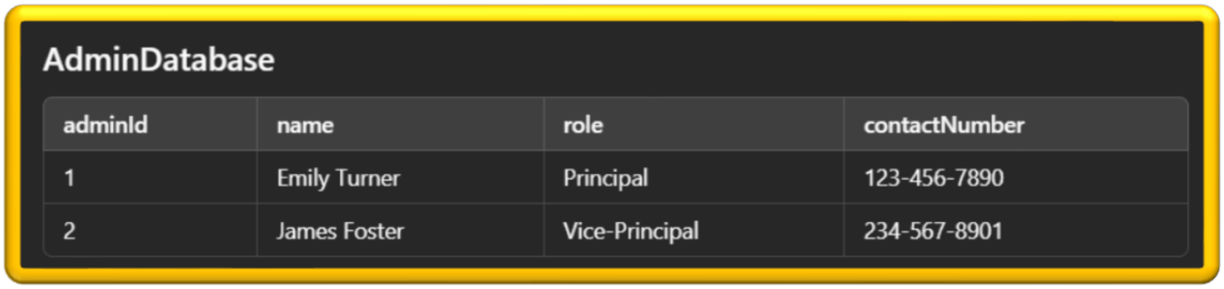
StaffDetails

staffId	name	position	department	hireDate
1	Mike Harris	Clerk	Administration	2010-04-01
2	Sara Lewis	Librarian	Library	2012-08-15
3	Tim Baker	Security	Security	2014-10-20



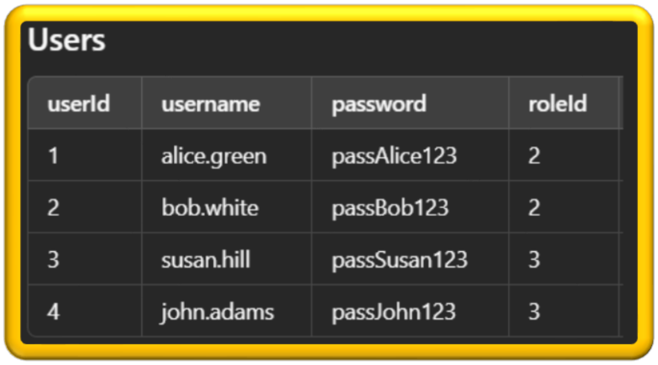
FeesManagement

feeId	studentId	amount	dueDate	status
1	1	1500	2024-03-01	Paid
2	2	1500	2024-06-01	Unpaid
3	3	1800	2024-09-01	Unpaid
4	4	1500	2024-12-01	Paid



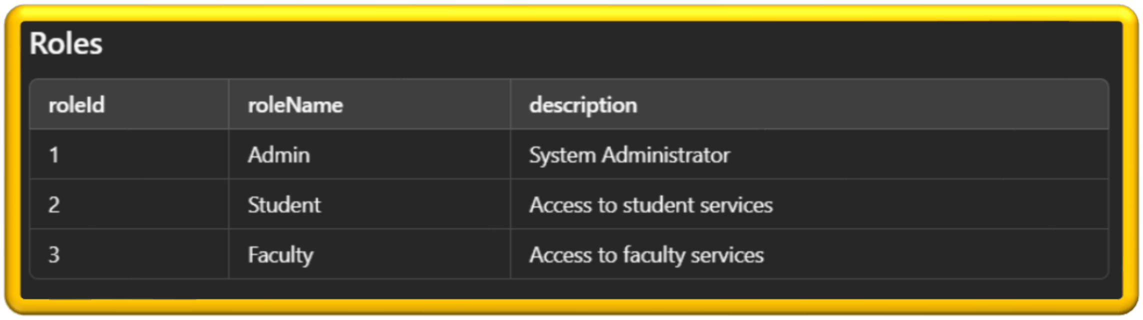
AdminDatabase

adminId	name	role	contactNumber
1	Emily Turner	Principal	123-456-7890
2	James Foster	Vice-Principal	234-567-8901



Users

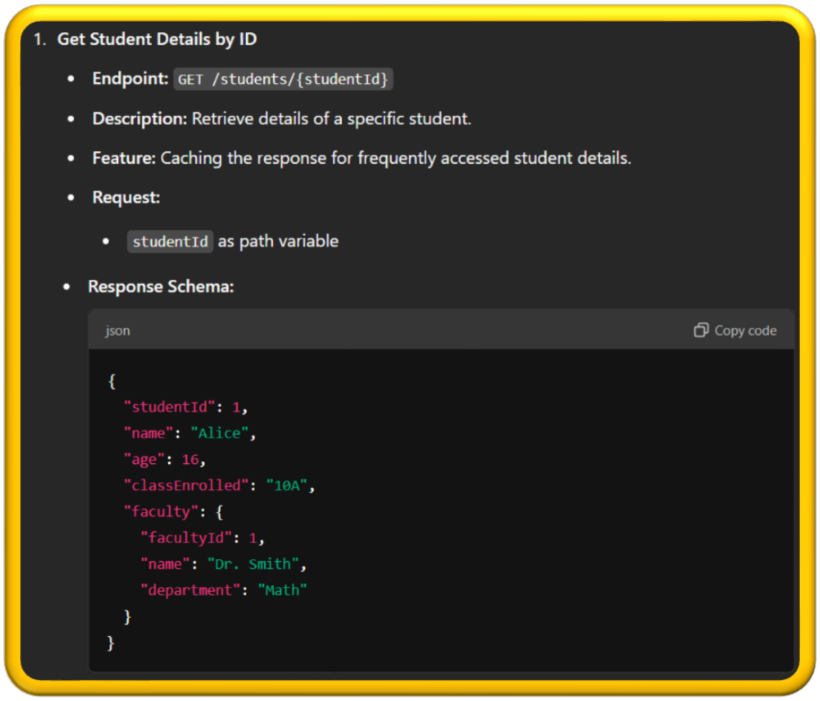
userId	username	password	roleId
1	alice.green	passAlice123	2
2	bob.white	passBob123	2
3	susan.hill	passSusan123	3
4	john.adams	passJohn123	3



Roles

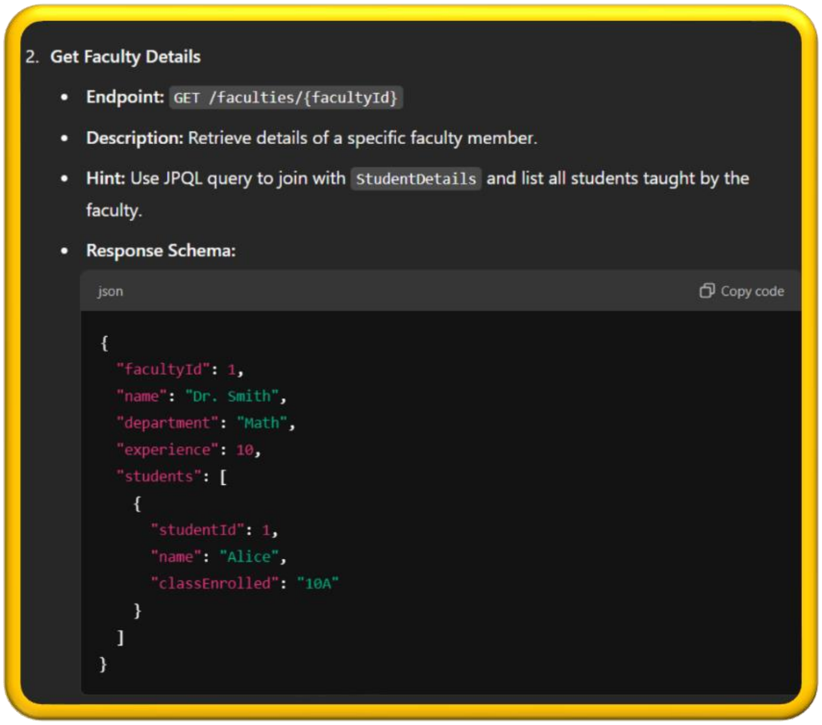
roleId	roleName	description
1	Admin	System Administrator
2	Student	Access to student services
3	Faculty	Access to faculty services

D. Endpoint Details are as Shown Below:



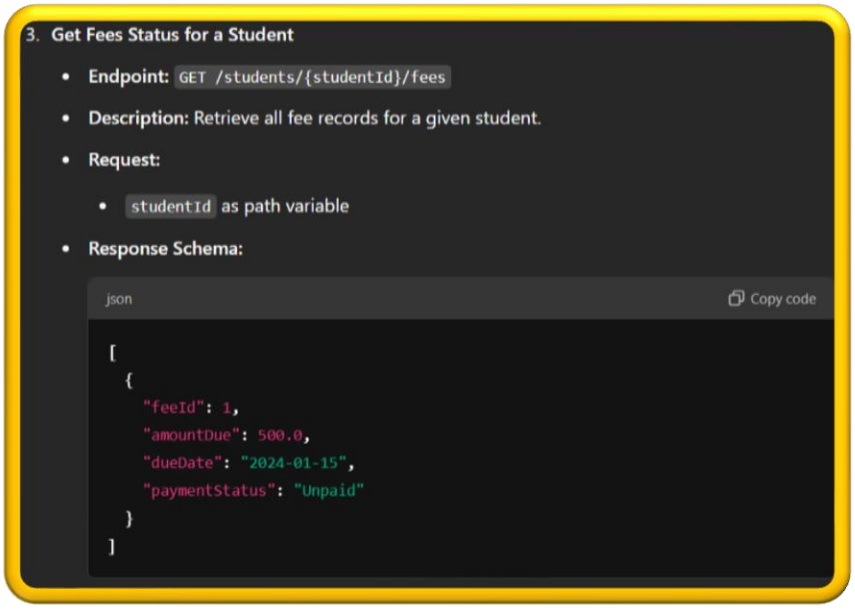
1. Get Student Details by ID
   - **Endpoint:** `GET /students/{studentId}`
   - **Description:** Retrieve details of a specific student.
   - **Feature:** Caching the response for frequently accessed student details.
   - **Request:**
     - `studentId` as path variable
   - **Response Schema:**

```json
{
  "studentId": 1,
  "name": "Alice",
  "age": 16,
  "classEnrolled": "10A",
  "faculty": {
    "facultyId": 1,
    "name": "Dr. Smith",
    "department": "Math"
  }
}
```


2. Get Faculty Details

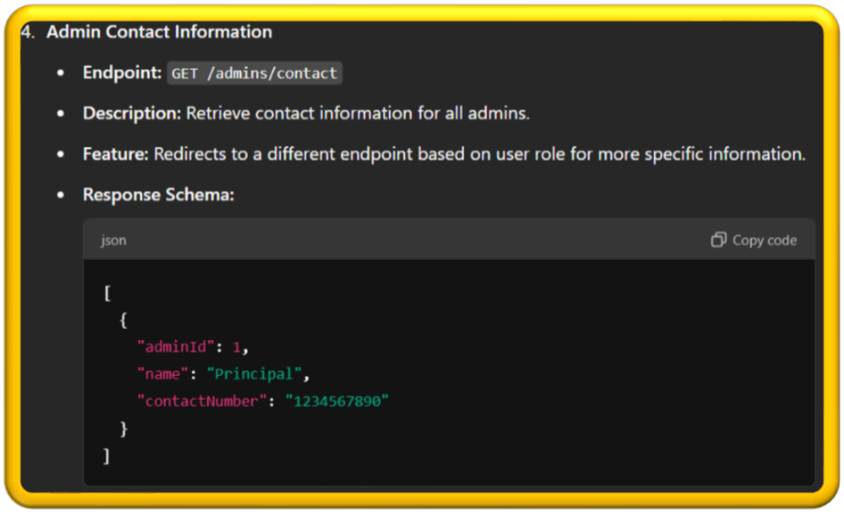
- **Endpoint:** `GET /faculties/{facultyId}`
- **Description:** Retrieve details of a specific faculty member.
- **Hint:** Use JPQL query to join with `StudentDetails` and list all students taught by the faculty.
- **Response Schema:**

json Copy code
```json
{
  "facultyId": 1,
  "name": "Dr. Smith",
  "department": "Math",
  "experience": 10,
  "students": [
    {
      "studentId": 1,
      "name": "Alice",
      "classEnrolled": "10A"
    }
  ]
}
```


3. Get Fees Status for a Student

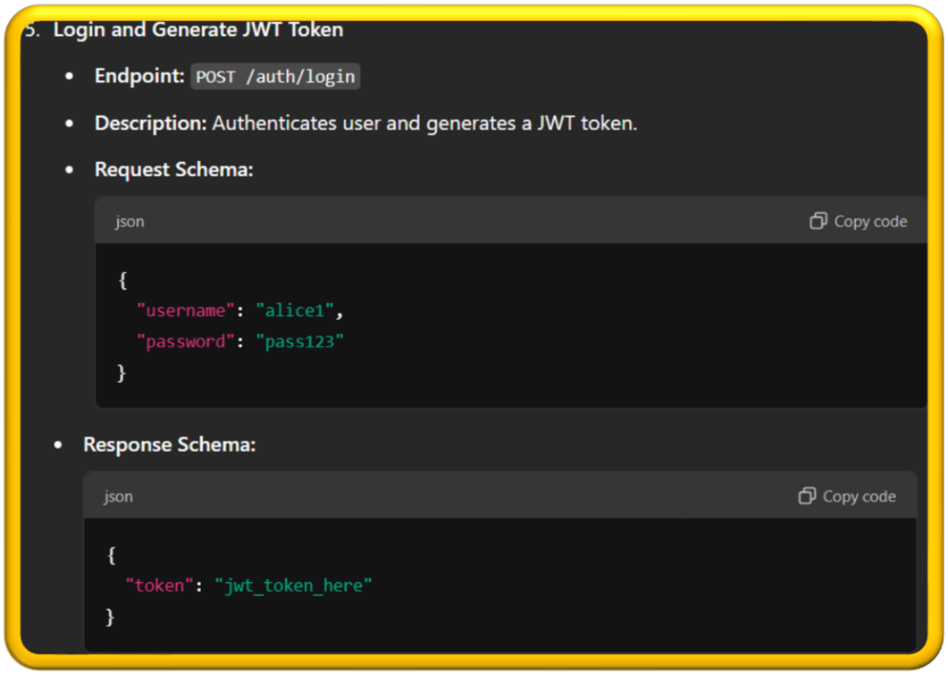
- **Endpoint:** `GET /students/{studentId}/fees`
- **Description:** Retrieve all fee records for a given student.
- **Request:**
  - `studentId` as path variable
- **Response Schema:**

json Copy code
```json
[
  {
    "feeId": 1,
    "amountDue": 500.0,
    "dueDate": "2024-01-15",
    "paymentStatus": "Unpaid"
  }
]
```


4. Admin Contact Information

- **Endpoint:** `GET /admins/contact`
- **Description:** Retrieve contact information for all admins.
- **Feature:** Redirects to a different endpoint based on user role for more specific information.
- **Response Schema:**

json Copy code
```json
[
  {
    "adminId": 1,
    "name": "Principal",
    "contactNumber": "1234567890"
  }
]
```



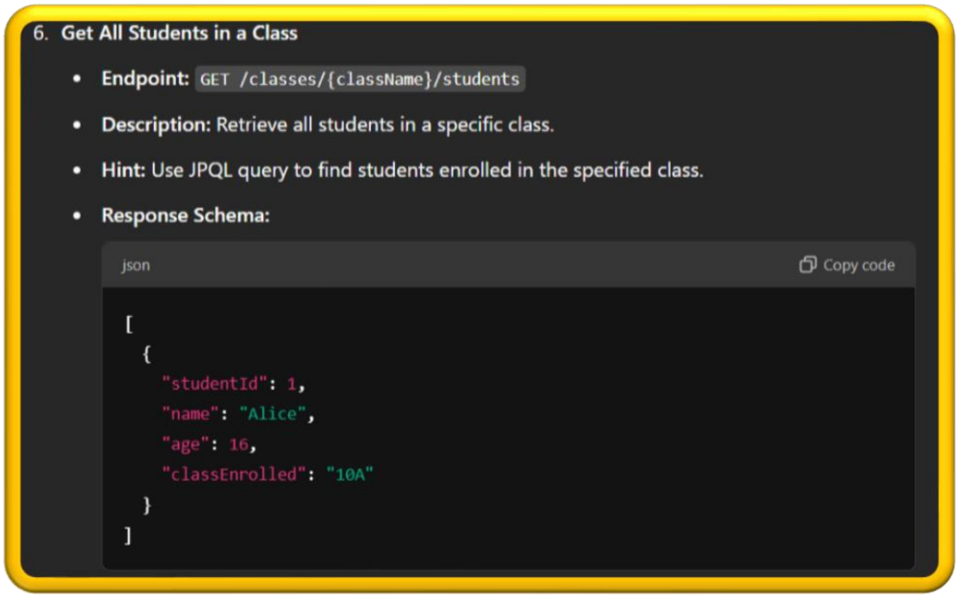
5. Login and Generate JWT Token

- **Endpoint:** `POST /auth/login`
- **Description:** Authenticates user and generates a JWT token.
- **Request Schema:**

```json
{
  "username": "alice1",
  "password": "pass123"
}
```

- **Response Schema:**

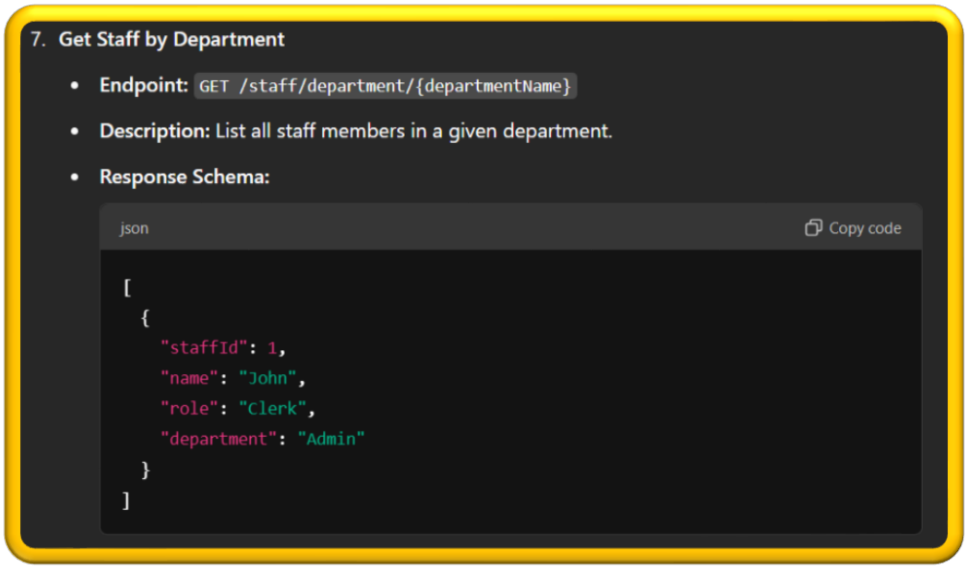
```json
{
  "token": "jwt_token_here"
}
```



6. Get All Students in a Class

- **Endpoint:** `GET /classes/{className}/students`
- **Description:** Retrieve all students in a specific class.
- **Hint:** Use JPQL query to find students enrolled in the specified class.
- **Response Schema:**

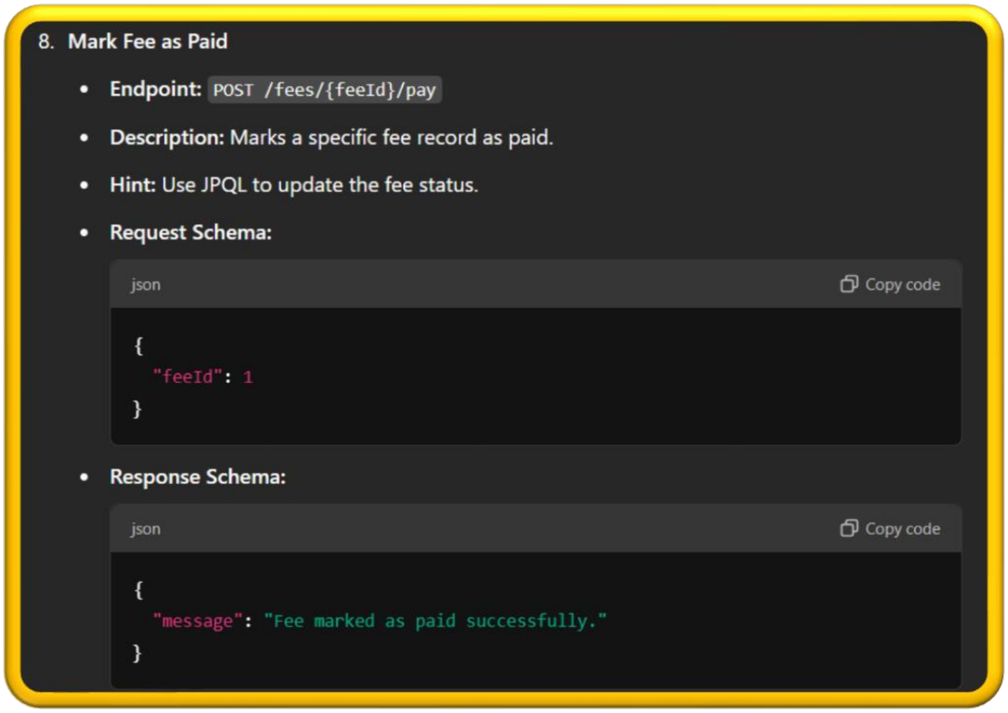
```json
[
  {
    "studentId": 1,
    "name": "Alice",
    "age": 16,
    "classEnrolled": "10A"
  }
]
```



7. Get Staff by Department

- **Endpoint:** `GET /staff/department/{departmentName}`
- **Description:** List all staff members in a given department.
- **Response Schema:**

```json
[
  {
    "staffId": 1,
    "name": "John",
    "role": "Clerk",
    "department": "Admin"
  }
]
```



8. Mark Fee as Paid

- **Endpoint:** `POST /fees/{feeId}/pay`
- **Description:** Marks a specific fee record as paid.
- **Hint:** Use JPQL to update the fee status.
- **Request Schema:**

```json
{
  "feeId": 1
}
```

- **Response Schema:**

```json
{
  "message": "Fee marked as paid successfully."
}
```