# EduHub: Collaborative Learning Platform

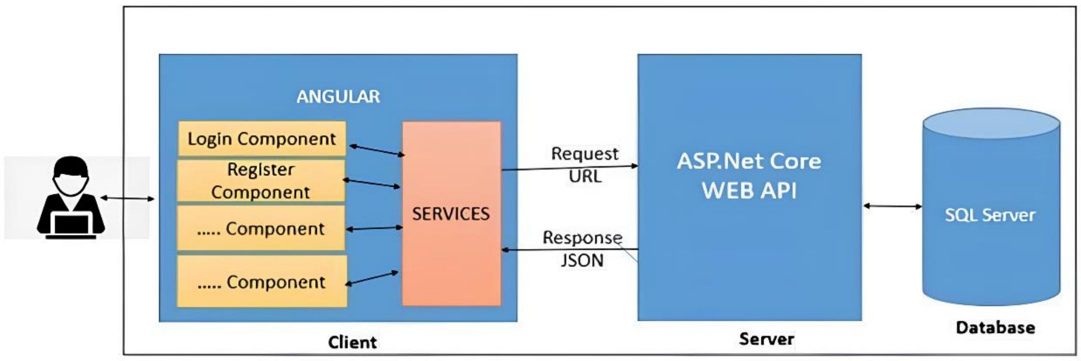
## Introduction:

EduHub is designed to empower learners by providing access to comprehensive educational resources and tools. Students can explore course materials, engage in self-paced learning, and seek clarification through inquiries. Educators, on the other hand, utilize EduHub to create and manage courses efficiently, curating content tailored to student requirements. Additionally, EduHub facilitates seamless communication between students and educators, allowing for inquiries to be submitted and addressed promptly. With its focus on personalized learning experiences and effective communication channels, EduHub enables students to thrive academically while offering educators the means to facilitate impactful teaching.

## Users of the System:

1. Educator
2. Student

## System Architectural Diagram:



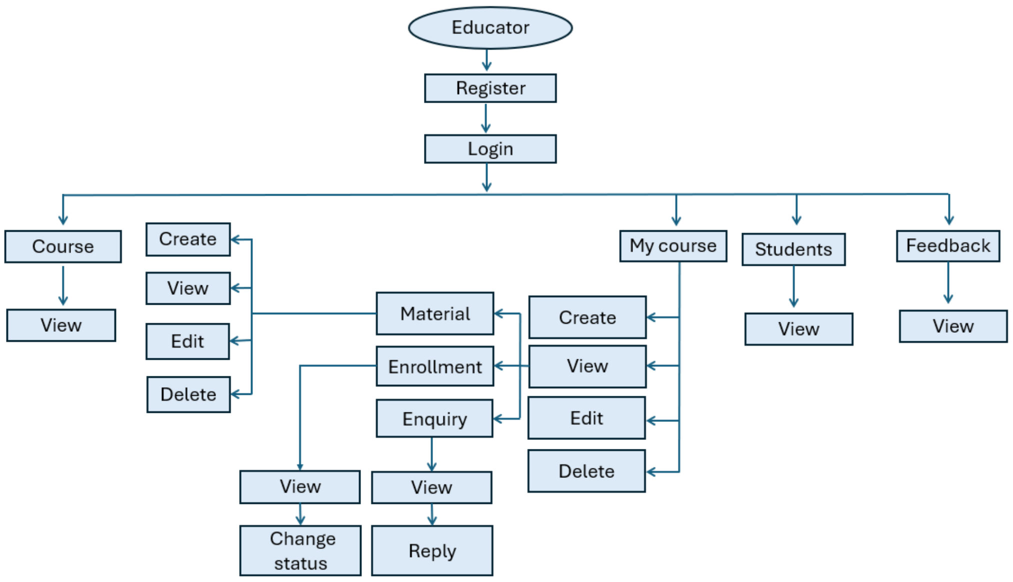
## Application Flow:

### User Authentication and Authorization:

- Users should be able to register an account with a unique email and password.
- Users should be able to log in securely using their credentials.
- Different user roles should have different levels of access (educator, student).

## Educator Actions:

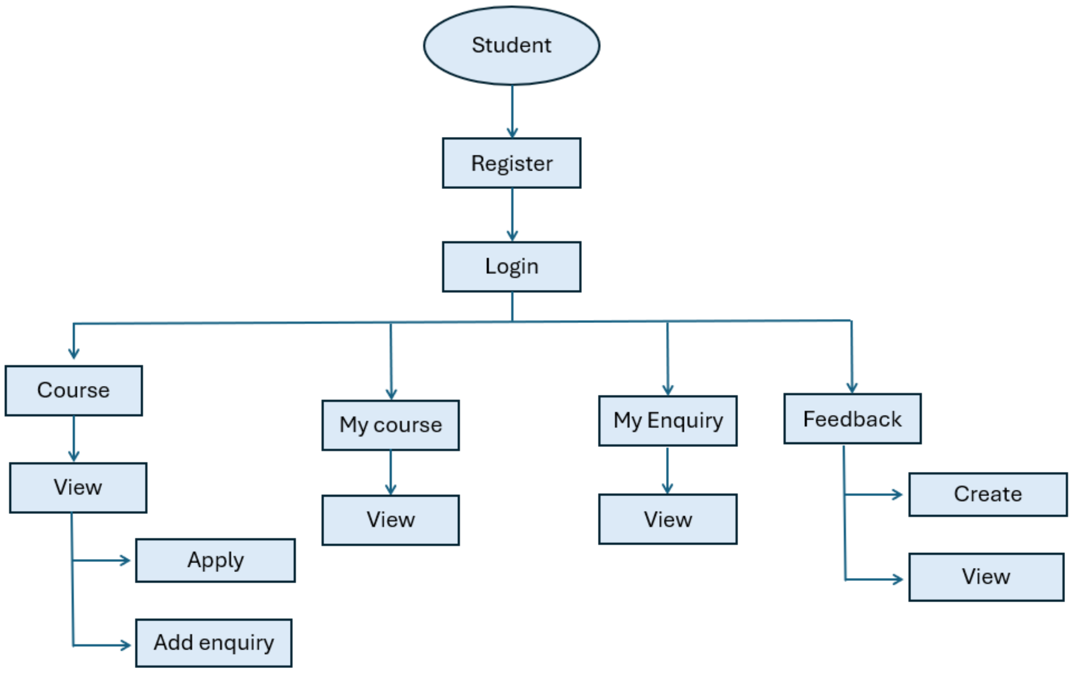
**Flow Diagram:**



- **Create Course**: Educators can create new courses by providing the title, description, start date, and end date. They are automatically assigned as the educator of the course.
- **Manage Course**: Educators can edit course details such as title, description, start date, and end date. They can also add or remove materials to the course.
- **View Enrollments**: Educators can view the list of students enrolled in the courses. They can see enrollment statuses (e.g., Accepted, Rejected).
- **Upload Materials**: Educators can upload URL learning materials/resources to their courses.
- **Respond to Enquiries**: Educators can respond to enquiries submitted by students regarding the course. They can provide answers, solutions, or additional information to address the students' queries.
- **View All Students**: Educators can view a list of all students.
- **View All Courses**: Educators can view a list of all courses.
- **Feedback**: Educators should have the option to view student's feedback.

## Student Actions:

**Flow Diagram:**



- **Enroll in Course**: Students can browse available courses and enroll in the ones they ares interested in. They must wait for the enrollment request to be accepted by the educator.
- **View Course Materials**: Once enrolled, students can access course materials uploaded by the educator. They can view resources provided for the course.
- **View His Enrolled Courses**: Students can view the list of courses they have enrolled in. They can see details such as course title, description, start date, and end date.
- **Submit Enquiries**: Students can submit enquiries regarding the course to educators.
- **View Enquiries**: Students can view enquiries they have submitted for a particular course. They can see the status of their enquiries and any responses provided by educators.
- **Feedback**: Students should have the option to view and post their feedback.

**Data Security with JWT (JSON Web Tokens):**

- **Token Generation and Validation:** Generate JWT tokens securely upon successful user authentication. Validate JWT tokens on each request to ensure their authenticity.
- **Authentication and Authorization:** Implement JWT-based authentication and ensure that only authenticated users with valid JWT tokens can access sensitive data and perform authorized actions.

## Error and Exception handling:

Employ try and catch blocks to manage potential risks, particularly during an Axios call. If an error is detected, the objective is to redirect to the page that displays the error message.

## Models:

### User Model:

1. **userId**: Primary key representing the unique identifier for each user record. It uniquely identifies each user.
2. **username**: Unique string representing the username of the user. It is required and must be unique.
3. **password**: Encrypted password for user authentication. It is required to secure user accounts.
4. **role**: String representing the role of the user (e.g., educator, student). It is required to determine user permissions.
5. **email**: Email address of the user. It is required for communication purposes and must be unique.
6. **mobileNumber**: String representing the mobile number of the user. It is required for contact purposes.
7. **profileImage**: String representing the profile image of the user encoded in base64 format. It can be used to store the image directly as a string.

### Course Model:

1. **courseId**: Primary key identifying each course uniquely.
2. **title**: Title of the course. It is required to provide a meaningful name for the course.
3. **description**: Description of the course. It provides detailed information about the course content and objectives.
4. **courseStartDate**: Date indicating when the course starts. It specifies the start date of the course.
5. **courseEndDate**: Date indicating when the course ends. It specifies the end date of the course.
6. **userId**: Foreign key referencing the user who created the course (educator). It establishes a relationship between the course and the educator who created it.
7. **category**: Category or subject area of the course. It helps in organizing and categorizing courses.
8. **level**: Difficulty level of the course (e.g., beginner, intermediate, advanced). It indicates the level of proficiency expected from the students.

**Enrollment Model:**

1. **enrollmentId**: Primary key identifying each enrollment uniquely.
2. **userId**: Foreign key referencing the enrolled user (student) . It establishes a relationship between the enrollment and the user.
3. **courseId**: Foreign key referencing the enrolled course. It establishes a relationship between the enrollment and the course.
4. **enrollmentDate**: Timestamp indicating when the user enrolled in the course. It specifies the date and time of enrollment.
5. **status**: Status of the enrollment (Accepted, Rejected, etc.). It indicates the current status of the enrollment request.

**Material Model:**

1. **materialId**: Primary key identifying each material uniquely.
2. **courseId**: Foreign key referencing the course to which the material belongs. It establishes a relationship between the material and the course.
3. **title**: Title of the material. It provides a descriptive title for the material.
4. **description**: Description of the material. It provides detailed information about the material content.
5. **URL**: External URL for the material (optional). It provides a link to external resources if applicable.
6. **uploadDate**: Date when the material was uploaded. It specifies the date and time when the material was added.
7. **contentType**: Content type of the material (e.g., lecture slides, video, quiz). It specifies the type or format of the material content.

**Enquiry Model:**

1. **enquiryId**: Primary key identifying each enquiry uniquely.
2. **userId**: Foreign key referencing the user who submitted the enquiry. It establishes a relationship between the enquiry and the user.
3. **courseId**: Foreign key referencing the course related to the enquiry (optional). It establishes a relationship between the enquiry and the course, if applicable.
4. **subject**: Subject of the enquiry. It provides a brief description of the enquiry topic.
5. **message**: Message or content of the enquiry. It contains detailed information about the enquiry.
6. **enquiryDate**: Timestamp indicating when the enquiry was submitted. It specifies the date and time of enquiry submission.
7. **status**: Status of the enquiry (e.g., Open, Closed, In Progress). It indicates the current status of the enquiry.
8. **response**: Response to the enquiry provided by administrators or educators (optional). It contains the reply or solution to the enquiry.

**Feedback model:**

1. **feedbackId:** This field represents the primary key for each record in the feedback table. It uniquely identifies each record. This field is required.
2. **userId:** The unique identifier of the user associated with the feedback. It references the User model and is required to associate the request with a specific user. This field is required.
3. **feedback:** Feedback provided by the user. This field is required.
4. **date**: The date for which the feedback is recorded. It is of type Date and is required.

## Controllers:

These controllers manage different aspects of users, courses, enrollments, materials, and enquiries within the EduHub system:

1. UserController
2. CourseController
3. EnrollmentController
4. EnquiryController
5. MaterialController
6. FeedbackController

## Endpoints:

1. Backend URL + "**/user/login**":
    a. Description: Log in a user with provided credentials.
    b. Method name: POST
    c. Request Body:
       Contains the following fields:
        i. Username: The username of the user.
        ii. Password: The password associated with the username.
    d. Response:
        i. Success (200 OK): Contains a JWT token and user details if the login is successful.
        ii. Error (401 Unauthorized): Returns an error message if the provided credentials are invalid.

2. Backend URL + **"/user/register"**:
    a. Description: Register a new user with provided details.
    b. Method name: POST
    c. Request Body:
       Contains the following fields:
        i. Username: The desired username for the new user.
        ii. Password: The password for the new user account.
        iii. Email: The email address of the new user.
        iv. Role: The role or type of the new user account (e.g., employee, manager).

        v. Mobile Number: The mobile number of the new user.
        vi. Profile Image: The profile image of the new user.

    d. Response:
        i. Success (200 Ok): Indicates that the user has been successfully registered along with user details.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure in case of registration error.

3. Backend URL + **"/user/getAllStudents"**:
    a. Description: Retrieve all students.
    b. Method name: POST
    c. Request Body:
       Contains the following fields:
        i. SearchValue
        ii. SortValue
        iii. PaginationValue
    d. Response:
        i. Success (200 Ok): List of all students.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

4. GET BackendURL + **"/course/getCourseByUserId**"
    a. Description: Retrieve courses by user ID.
    b. Request Method: GET
    c. Request Parameters:
        i. userId: The unique identifier of the user whose courses are to be retrieved.
        ii. SearchValue: Optional parameter for searching within course titles or descriptions.
        iii. SortValue: Optional parameter for sorting the results based on certain criteria.
        iv. PaginationValue: Optional parameter for paginating the results.
    d. Response:
        i. Success (200 OK): List of courses associated with the specified user ID.
        ii. Error (404 Not Found): Returns an error message if no courses are found for the given user ID.

5. GET BackendURL + **"/course/getAllCourses**"
    a. Description: Retrieve all courses.
    b. Request Method: POST
    c. Request Parameters:
        i. SearchValue: Optional parameter for searching within course titles or descriptions.

        ii. SortValue: Optional parameter for sorting the results based on certain criteria.
        iii. PaginationValue: Optional parameter for paginating the results.
    d. Response:
        i. Success (200 OK): List of all available courses.
        ii. Error (404 Not Found): Returns an error message if no courses are found.

6. POST BackendURL + "**/course/addCourse**"
    a. Description: Add a new course.
    b. Request Method: POST
    c. Request Body:
        i. Details of the new course including title, description, start date, end date, and other relevant information.
    d. Response:
        i. Success (200 OK): Details of the newly added course.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

7. GET BackendURL + "/**course/getCourseByCourseId/:courseId**"
    a. Description: Retrieve a specific course by its ID.
    b. Request Method: GET
    c. Request Parameters:
        i. courseId: The unique identifier of the course to be retrieved.
    d. Response:
        i. Success (200 OK): Details of the specified course.
        ii. Error (404 Not Found): Returns an error message if the course with the specified ID is not found.

8. PUT BackendURL + "**/course/updateCourseByCourseId/:courseId**"
    a. Description: Update an existing course by its ID.
    b. Request Method: PUT
    c. Request Parameters:
        i. courseId: The unique identifier of the course to be updated.
    d. Request Body:
        i. Contains the updated details of the course.
    e. Response:
        i. Success (200 OK): Details of the updated course.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

9. DELETE BackendURL + "**/course/deleteCourseByCourseId/:courseId**"
    a. Description: Delete a course by its ID.
    b. Request Method: DELETE
    c. Request Parameters:
        i. courseId: The unique identifier of the course to be deleted.

    d. Response:
        i. Success (200 OK): Message confirming the successful deletion of the course.
        ii. Error (404 Not Found): Returns an error message if the course with the specified ID is not found.

10. POST BackendURL + "**/material/addMaterial**"
    a. Description: Add a new material to a course.
    b. Request Method: POST
    c. Request Body:
        i. Contains the details of the new material including title, description and other relevant information.
    d. Response:
        i. Success (200 OK): Details of the newly added material.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

11. GET BackendURL + "**/material/getMaterialByMaterialID/:materialID**"
    a. Description: Retrieve a specific material by its ID.
    b. Request Method: GET
    c. Request Parameters:
        i. materialID: The unique identifier of the material to be retrieved.
    d. Response:
        i. Success (200 OK): Details of the specified material.
        ii. Error (404 Not Found): Returns an error message if the material with the specified ID is not found.

12. PUT BackendURL + "**/material/updateMaterialByMaterialID/:materialID**"
    a. Description: Update an existing material by its ID.
    b. Request Method: PUT
    c. Request Parameters:
        i. materialID: The unique identifier of the material to be updated.
    d. Request Body:
        i. Contains the updated details of the material.
    e. Response:
        i. Success (200 OK): Details of the updated material.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

13. DELETE BackendURL + "**/material/deleteMaterialByMaterialID/:materialID**"
    a. Description: Delete a material by its ID.
    b. Request Method: DELETE
    c. Request Parameters:
        i. materialID: The unique identifier of the material to be deleted.
    d. Response:

        i. Success (200 OK): Message confirming the successful deletion of the material.
        ii. Error (404 Not Found): Returns an error message if the material with the specified ID is not found.

14. GET BackendURL + "**/material/getMaterialByCourseID/:courseID**"
    a. Description: Retrieve materials associated with a specific course.
    b. Request Method: GET
    c. Request Parameters:
        i. courseID: The unique identifier of the course for which materials are to be retrieved.
    d. Response:
        i. Success (200 OK): List of materials associated with the specified course.
        ii. Error (404 Not Found): Returns an error message if no materials are found for the given course ID.

15. GET BackendURL + "**/enroll/getEnrolledCourseByUserId/:userId**"
    a. Description: Retrieve enrolled courses by user ID.
    b. Request Method: GET
    c. Request Parameters:
        i. userId: The unique identifier of the user whose enrolled courses are to be retrieved.
    d. Response:
        i. Success (200 OK): List of courses enrolled by the specified user.
        ii. Error (404 Not Found): Returns an error message if no enrolled courses are found for the given user ID.

16. GET BackendURL + "**/enroll/getAllEnrolls**"
    a. Description: Retrieve all course enrollments.
    b. Request Method: GET
    c. Request Body: None
    d. Response:
        i. Success (200 OK): List of all course enrollments.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

17. POST BackendURL + "**/enroll/addEnroll**"
    a. Description: Enroll a user in a course.
    b. Request Method: POST
    c. Request Body:
        i. Contains the details of the enrollment including the user ID and course ID.
    d. Response:
        i. Success (200 OK): Details of the newly added enrollment.

    ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

18. PUT BackendURL + "**/enroll/updateEnrollByEnrollID/:enrollID**"
    a. Description: Update an existing enrollment by its ID.
    b. Request Method: PUT
    c. Request Parameters:
        i. enrollID: The unique identifier of the enrollment to be updated.
    d. Request Body:
        i. Contains the updated details of the enrollment.
    e. Response:
        i. Success (200 OK): Details of the updated enrollment.
        ii. Error (400 Bad Request): Returns an error message if the request is invalid or missing required parameters.

19. DELETE BackendURL + "**/enroll/deleteEnrollByEnrollID/:enrollID**"
    a. Description: Delete an enrollment by its ID.
    b. Request Method: DELETE
    c. Request Parameters:
        i. enrollID: The unique identifier of the enrollment to be deleted.
    d. Response:
        i. Success (200 OK): Message confirming the successful deletion of the enrollment.
        ii. Error (404 Not Found): Returns an error message if the enrollment with the specified ID is not found.

20. GET BackendURL + "**/enquiry/getEnquiryByUserId/:userId**"
    a. Description: Retrieve enquiries by user ID.
    b. Method: GET
    c. Parameter: userId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): List of enquiries for the specified user ID.
        ii. Error (404 Not Found): If no enquiries are found for the user ID.

21. GET BackendURL + "**/enquiry/getEnquiryByCourseId/:courseId**"
    a. Description: Retrieve enquiries by course ID.
    b. Method: GET
    c. Parameter: courseId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): List of enquiries for the specified course ID.
        ii. Error (404 Not Found): If no enquiries are found for the course ID.

22. GET BackendURL + "**/enquiry/getAllEnquiry**"
    a. Description: Retrieve all enquiries.
    b. Method: POST
    c. Request Body:
        i. SearchValue: Value to filter enquiries.
        ii. SortValue: Value to sort enquiries.
        iii. PaginationValue: Value to paginate results.
    d. Response:
        i. Success (200 Ok): List of all enquiries.
        ii. Error (400 Bad Request): If there's an error in processing the request.

23. POST BackendURL + "**/enquiry/addEnquiry**"
    a. Description: Add a new enquiry.
    b. Method: POST
    c. Request Body: Contains the details of the new enquiry.
    d. Response:
        i. Success (200 Ok): Details of the newly added enquiry.
        ii. Error (400 Bad Request): If the request body is invalid.

24. GET BackendURL + "**/enquiry/getEnquiryByEnquiryID/:enquiryId**"
    a. Description: Get a specific enquiry by its ID.
    b. Method: GET
    c. Parameter: enquiryId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): Details of the specific enquiry.
        ii. Error (404 Not Found): If the enquiry with the specified ID is not found.

25. PUT BackendURL + "**/enquiry/updateEnquiryByEnquiryID/:enquiryId**"
    a. Description: Update an existing enquiry by its ID.
    b. Method: PUT
    c. Parameter: enquiryId
    d. Request Body: Contains the updated details of the enquiry.
    e. Response:
        i. Success (200 Ok): Details of the updated enquiry.
        ii. Error (400 Bad Request): If the request body is invalid.

26. DELETE BackendURL + "**/enquiry/deleteEnquiryByEnquiryID/:enquiryId**"
    a. Description: Delete an enquiry by its ID.
    b. Method: DELETE
    c. Parameter: enquiryId
    d. Request Body: None
    e. Response:
        i. Success (200 Ok): Message confirming successful deletion of the enquiry.

    ii. Error (404 Not Found): If the enquiry with the specified ID is not found.

27. Backend URL + **"/feedback/addFeedback":**
    a. Description: Add a new feedback.
    b. Method name: POST
    c. Request Body:
        i. Contains the details of the new feedback
    d. Response:
        i. Success (200 Ok): Details of the newly added feedback.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

28. Backend URL + **"/feedback /getFeedbacksByUserId/:userId":**
    a. Description: Retrieve all the feedback specific to the user.
    b. Method name: POST
    c. Parameter: userId
    d. Request Body:
        i. No request body required.
    e. Response:
        i. Success (200 Ok): List of feedbacks for the specified user ID.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

29. Backend URL + **"/feedback /getAllFeedbacks":**
    a. Description: Retrieve all feedbacks.
    b. Method name: GET
    c. Request Body:
        i. No request body required.
    d. Response:
        i. Success (200 Ok): List of all feedbacks.
        ii. Error (400 Bad Request): Returns an error message describing the reason for failure.

**Conclusion**:
EduHub offers a transformative online platform that empowers both educators and students in their educational journey. With comprehensive resources and tools, students can engage in self-paced learning while educators efficiently create and manage courses tailored to student needs. Through personalized learning experiences, EduHub fosters academic growth for students and facilitates effective teaching practices for educators, ultimately revolutionizing the landscape of education.