**Online Learning Platform**

**NAME : SRI HARSHANJALI.V**

**Roll No: BU21CSEN0300418**

**PROBLEM STATEMENT:**

**Problem:** Create a platform where instructors can upload courses, and students can enroll in and complete courses with progress tracking.

**Requirements:**

o Front-end: Course browsing, enrolling, and video streaming interface.

o Back-end: REST API for course management, progress tracking, and user

enrolment.

o Database: Store users, courses, and student progress.

o Video Streaming: Integrate with a cloud video streaming service.

**SOLUTION:**

**Objective:** To create an interactive online learning platform where instructors can upload and manage courses, and students can browse, enroll in, and complete these courses while tracking their progress. Here I am using techniques are Frontend is HTML, Backend is Java and Database is MySQL.

**Key Features:**

1. **Course Management:**
   * Instructors should have the ability to create, update, and delete courses.
   * Each course should include a title, description, instructional materials (such as videos), and progress tracking.
2. **User Management:**
   * Users must be able to register, log in, and manage their profiles.
   * There will be two types of users: instructors and students, each with different permissions.
3. **Course Browsing:**
   * Students should have a user-friendly interface to browse available courses.
   * Courses should be filterable by category, popularity, and rating.
4. **Enrollment and Progress Tracking:**
   * Students can enroll in courses and start learning at their own pace.
   * The platform must track student progress, including completed modules and time spent on each course.
5. **Video Streaming:**
   * The platform should support video streaming, allowing instructors to upload course videos and students to watch them seamlessly.
   * Integration with a cloud video streaming service (e.g., AWS S3 + CloudFront) for scalable storage and distribution.

**Technical Requirements:**

* **Frontend:**
  + The user interface should be developed using HTML, CSS, and JavaScript, ensuring a responsive design that works on various devices.
* **Backend:**
  + A RESTFUL API should be developed using Java with Spring Boot, allowing for CRUD operations on courses and user management.
  + Hibernate for ORM to interact with the MySQL database.
* **Database:**
  + A MySQL database to store user information, course details, and enrollment data.

**Use Case:**

* An instructor logs into the platform, uploads a new course with instructional videos, and makes it available to students.
* A student logs in, browses the course catalog, enrolls in a selected course, watches videos, and tracks their progress through a dashboard.

**Implementation Steps:**

**Step 1: Define Project Scope**

* **Identify Stakeholders:** Understand who will be using the platform (instructors, students, administrators).
* **Gather Requirements:** Collaborate with stakeholders to gather detailed functional and non-functional requirements.

**Step 2: Design the System Architecture**

* **Choose an Architecture Pattern:** Consider using an MVC (Model-View-Controller) architecture for separation of concerns.
* **Define API Endpoints:** Outline the RESTful API endpoints required for course management, user management, and enrollment.

**Step 3: Set Up Development Environment**

* **Tools and Frameworks:** Set up Java with Spring Boot, MySQL, and a suitable IDE (like IntelliJ IDEA or Eclipse).
* **Version Control:** Initialize a Git repository for version control.

**Step 4: Database Design**

* **Create Database Schema:**
  + Design tables for users, courses, and enrollments.
  + Use MySQL Workbench or similar tools to create the database structure.

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL,

role ENUM('INSTRUCTOR', 'STUDENT') NOT NULL

);

CREATE TABLE courses (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100) NOT NULL,

description TEXT NOT NULL,

instructor\_id INT,

FOREIGN KEY (instructor\_id) REFERENCES users(id)

);

CREATE TABLE enrollments (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

course\_id INT,

progress INT DEFAULT 0,

FOREIGN KEY (user\_id) REFERENCES users(id),

FOREIGN KEY (course\_id) REFERENCES courses(id)

);

**Step 5: Implement Backend**

* **Create Spring Boot Application:**
  + Initialize a new Spring Boot project using Spring Initializr with dependencies for Spring Web, Spring Data JPA, and MySQL Driver.
* **Implement REST API:**
  + Create controllers for handling requests (e.g., CourseController, UserController).
  + Implement services for business logic (e.g., CourseService, UserService).
  + Use Hibernate for data access.

**Example of a Course Controller:**

@RestController

@RequestMapping("/api/courses")

public class CourseController {

@Autowired

private CourseService courseService;

@GetMapping

public List<Course> getCourses() {

return courseService.getAllCourses();

}

@PostMapping

public void createCourse(@RequestBody Course course) {

courseService.createCourse(course);

}

@PostMapping("/enroll/{courseId}")

public void enrollUser(@PathVariable Long courseId, @RequestBody User user) {

courseService.enrollUser(courseId, user);

}

}

**Step 6: Implement Frontend**

* **Set Up Frontend Framework:**
  + Create the HTML/CSS/JavaScript structure.
  + Use AJAX (or Fetch API) to interact with the back-end REST API.
* **Course Browsing Page:**
  + Build a dynamic course listing page with options to enroll.
* **Profile Management:**
  + Create user profile management features for viewing and editing user information.

**Step 7: Integrate Video Streaming**

* **Choose Video Hosting Solution:** Decide on a cloud provider like AWS S3 for video storage.
* **Set Up CloudFront:** Configure AWS CloudFront for content delivery.
* **Implement Video Upload/Streaming Logic:**
  + Use AWS SDK for Java to upload videos to S3 and generate secure URLs for stream.

**Example of video upload code:**

public void uploadVideo(File file) {

AmazonS3 s3Client = AmazonS3ClientBuilder.standard().build();

ObjectMetadata metadata = new ObjectMetadata();

metadata.setContentLength(file.length());

s3Client.putObject("your-bucket-name", file.getName(), file, metadata);

}

**Step 8: Implement User Authentication**

* **User Registration/Login:**
  + Implement registration and login features with password hashing (e.g., using BCrypt).
* **Session Management:**
  + Use Spring Security for session management and role-based access control.

**Step 9: Testing**

* **Unit Testing:** Write unit tests for both front-end and back-end components.
* **Integration Testing:** Ensure that different components work well together.
* **User Acceptance Testing:** Gather feedback from real users to make improvements.

**Step 10: Deployment**

* **Choose Hosting Platform:** Consider options like AWS, Heroku, or DigitalOcean.
* **Deploy Backend and Frontend:**
  + Use tools like Docker for containerization or deploy directly on a cloud server.
* **Set Up CI/CD Pipeline:** Use tools like GitHub Actions or Jenkins for continuous integration and deployment.

**Step 11: Monitor and Maintain**

* **Performance Monitoring:** Use monitoring tools to keep track of server performance and user activities.
* **Feedback Loop:** Continuously gather user feedback for future improvements.
* **Regular Updates:** Keep the system updated with security patches and new features.

**1. Frontend Development**

**HTML Structure for Course Browsing and Enrolment**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Online Learning Platform</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<header>

<h1>Online Learning Platform</h1>

<nav>

<a href="#courses">Courses</a>

<a href="#profile">Profile</a>

</nav>

</header>

<section id="courses">

<h2>Available Courses</h2>

<div id="course-list"></div>

</section>

<script>

async function fetchCourses() {

const response = await fetch('/api/courses');

const courses = await response.json();

const courseList = document.getElementById('course-list');

courses.forEach(course => {

const courseItem = document.createElement('div');

courseItem.innerHTML = `

<h3>${course.title}</h3>

<p>${course.description}</p>

<button onclick="enroll('${course.id}')">Enroll</button>

`;

courseList.appendChild(courseItem);

});

}

async function enroll(courseId) {

await fetch(`/api/enroll/${courseId}`, { method: 'POST' });

alert('Enrolled successfully!');

}

fetchCourses();

</script>

</body>

</html>

**2. Backend Development**

**Java Spring Boot REST API**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api")

public class CourseController {

@Autowired

private CourseService courseService;

@GetMapping("/courses")

public List<Course> getCourses() {

return courseService.getAllCourses();

}

@PostMapping("/enroll/{courseId}")

public void enroll(@PathVariable Long courseId, @RequestBody User user) {

courseService.enrollUser(courseId, user);

}

}

@Service

public class CourseService {

@Autowired

private CourseRepository courseRepository;

public List<Course> getAllCourses() {

return courseRepository.findAll();

}

public void enrollUser(Long courseId, User user) {

// Logic to enroll user in the course and track progress

}

}

**3. Database Management**

**MySQL Schema**

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL

);

CREATE TABLE courses (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100) NOT NULL,

description TEXT NOT NULL

);

CREATE TABLE enrollments (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

course\_id INT,

progress INT DEFAULT 0,

FOREIGN KEY (user\_id) REFERENCES users(id),

FOREIGN KEY (course\_id) REFERENCES courses(id)

);

**4. Video Streaming Integration**

For video streaming, you can integrate AWS S3 and CloudFront.

**Example Code for Uploading Video to S3**

import com.amazonaws.services.s3.AmazonS3;

import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import com.amazonaws.services.s3.model.ObjectMetadata;

import java.io.File;

public class VideoUploadService {

private final AmazonS3 s3Client = AmazonS3ClientBuilder.standard().build();

private final String bucketName = "your-bucket-name";

public void uploadVideo(File file) {

ObjectMetadata metadata = new ObjectMetadata();

metadata.setContentLength(file.length());

s3Client.putObject(bucketName, file.getName(), file, metadata);

}

}

**Output Example**

* **Course Browsing**: Users will see a list of courses with titles and descriptions, along with an "Enroll" button.
* **Enrollment**: Upon clicking "Enroll," an alert will show "Enrolled successfully!" after successfully calling the API.

### 1. HTML Output:

When you open the HTML file in a browser, you would see a simple layout like this:

#### Browser View:

markdown

Copy code

------------------------------------------------

| Online Learning Platform |

| [ Courses ] [ Profile ] |

------------------------------------------------

Available Courses

-----------------------------------------------

| Course Title 1 |

| Description of course 1 |

| [ Enroll ] |

-----------------------------------------------

| Course Title 2 |

| Description of course 2 |

| [ Enroll ] |

-----------------------------------------------

| Course Title 3 |

| Description of course 3 |

| [ Enroll ] |

-----------------------------------------------

### 2. Enrollment Success

When a user clicks on the "Enroll" button for a specific course, they would see an alert pop-up:

------------------------------------------------

| Alert |

| Enrolled successfully! |

------------------------------------------------

### 3. REST API Responses

Assuming the back-end is functioning properly, here’s how the API responses would look when fetching courses:

#### GET /api/courses Response:

{

"id": 1,

"title": "Course Title 1",

"description": "Description of course 1"

},

{

"id": 2,

"title": "Course Title 2",

"description": "Description of course 2"

},

{

"id": 3,

"title": "Course Title 3",

"description": "Description of course 3"

}

]

### 4. Database Output

Once a user enrolls in a course, the database would reflect this change. For example, if user with id = 1 enrolls in course\_id = 1, the enrollments table would look like this:

#### enrollments Table:

| id | user\_id | course\_id | progress |

|----|---------|-----------|----------|

| 1 | 1 | 1 | 0 |

### Conclusion:

This output showcases how the user interface, API responses, and database interactions would work together to form the online learning platform. As users interact with the platform, they would be able to browse courses, enroll, and their progress would be tracked in the database. This example serves as a foundational layout for building a complete online learning platform. You can expand upon this with additional features like user authentication, detailed progress tracking, and video streaming capabilities.