COURSE ENROLLEMENT MANAGEMENT SYSTEM

**A MINI-PROJECT REPORT**

|  |  |
| --- | --- |
| **Submitted** | **by** |
| **M. ARUN PRAKASH** | **230701037** |
| **ADITHYA SURESH** | **230701014** |

In partial fulfilment of the award of the degree of

BACHELOR OF ENGINEERING IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS) THANDALAM

CHENNAI-602105

2023- 24

# BONAFIDE CERTIFICATE

Certified that this project report “**COURSE ENROLLEMENT MANAGEMENT SYSTEM**” is the Bonafide work of **“M. ARUN PRAKASH (230701037), ADITHYA SURESH (230701014),”**

who carried out the project work under my supervision.

**Submitted for the Practical Examination held on**

**SIGNATURE**

**Ms. DHARANI DEVI**

**Assistant Professor (SG), Computer Science and Engineering, Rajalakshmi Engineering College,**

**(Autonomous), Thandalam, Chennai - 602 105**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# ABSTRACT

The **Course Enrollment Management System** is an application designed to streamline and automate the process of managing course enrollments in an educational or corporate training environment. The system allows administrators to create, update, and manage courses while enabling employees or students to enroll in these courses seamlessly. It provides a centralized platform to handle the complexities of course registration, ensuring accuracy, efficiency, and real-time tracking.

The system reduces manual efforts, minimizes errors, and provides an enhanced user experience for both administrators and users. Its modular design allows easy integration into existing educational or corporate infrastructure while ensuring flexibility for customization to meet specific needs.

# TABLE OF CONTENTS

1. INTRODUCTION
2. INTRODUCTION
3. OBJECTIVES
4. MODULES
5. SURVEY OF TECHNOLOGIES
6. SOFTWARE DESCRIPTION
7. LANGUAGES
   1. SQL
   2. JAVA
   3. REACT.JS
8. REQUIREMENTS AND ANALYSIS
9. REQUIREMENT SPECIFICATION 10.HARDWARE AND SOFTWARE REQUIREMENTS 11.ARCHITECTURE DIAGRAM
10. PROGRAM CODE
11. RESULTS AND DISCUSSION
12. CONCLUSION
13. REFERENCES

# INTRODUCTION

The COURSE ENROLLMENT MANAGEMENT SYSTEM streamlines the process of managing the course and enrollments for educational or training purposes. It enables administrators to create, update, and manage courses while allowing users to enroll seamlessly . The system ensures data security, scalability, and operational efficiency through automated workflow and a robust database. Therefore it enhances the user experience.

# SYSTEM SPECIFICATIONS

HARDWARE SPECIFICATIONS:

* PROCESSOR : Ryzen 5
* MEMORY SIZE : 8GB(Minimum)
* HARD DISK : 500 GB of free space

SOFTWARE SPECIFICATIONS:

* PROGRAMMING LANGUAGE : Java, SQL, React.js
* FRONT-END : React.js
* BACK-END : MySQL, JAVA
* OPERATING SYSTEM : Windows 11

# Program:

# Spring Boot Java Backend:

# Package model:

# Coruse.java

package com.example.demo.model;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Table;  
import jakarta.persistence.Id;  
  
@Entity  
@Table(name = "courses")  
public class Course {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int courseId;  
  
 private String courseName;  
 private String courseDescription;  
 private int duration;  
 private String category;  
 private String createdBy;  
  
 // Default no-argument constructor (required by JPA)  
 public Course(){}  
  
 // Constructor with parameters  
 public Course(String courseName, String courseDescription, int duration, String category, String createdBy) {  
 this.courseName = courseName;  
 this.courseDescription = courseDescription;  
 this.duration = duration;  
 this.category = category;  
 this.createdBy = createdBy;  
 }  
  
 // Getters and setters  
 public int getCourseId() {  
 return courseId;  
 }  
  
 public void setCourseId(int courseId) {  
 this.courseId = courseId;  
 }  
  
 public String getCourseName() {  
 return courseName;  
 }  
  
 public void setCourseName(String courseName) {  
 this.courseName = courseName;  
 }  
  
 public String getCourseDescription() {  
 return courseDescription;  
 }  
  
 public void setCourseDescription(String courseDescription) {  
 this.courseDescription = courseDescription;  
 }  
  
 public int getDuration() {  
 return duration;  
 }  
  
 public void setDuration(int duration) {  
 this.duration = duration;  
 }  
  
 public String getCategory() {  
 return category;  
 }  
  
 public void setCategory(String category) {  
 this.category = category;  
 }  
  
 public String getCreatedBy() {  
 return createdBy;  
 }  
  
 public void setCreatedBy(String createdBy) {  
 this.createdBy = createdBy;  
 }  
}

# Employee.java

package com.example.demo.model;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Table;  
import jakarta.persistence.Id;  
import java.time.LocalDate;  
  
@Entity  
@Table(name = "employees")  
public class Employee {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int employeeId;  
  
 private String firstName;  
 private String lastName;  
 private String email;  
 private String department;  
 private String position;  
 private LocalDate dateOfJoining;  
 private String educationLevel;  
 private String password;  
  
 public Employee() {}  
  
 public Employee(String firstName, String lastName, String email, String department,  
 String position, LocalDate dateOfJoining, String educationLevel,String password) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 this.department = department;  
 this.position = position;  
 this.dateOfJoining = dateOfJoining;  
 this.educationLevel = educationLevel;  
 this.password = password;  
  
 }  
  
 // Getters and Setters  
 public int getEmployeeId() {  
 return employeeId;  
 }  
  
 public void setEmployeeId(int employeeId) {  
 this.employeeId = employeeId;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getDepartment() {  
 return department;  
 }  
  
 public void setDepartment(String department) {  
 this.department = department;  
 }  
  
 public String getPosition() {  
 return position;  
 }  
  
 public void setPosition(String position) {  
 this.position = position;  
 }  
  
 public LocalDate getDateOfJoining() {  
 return dateOfJoining;  
 }  
  
 public void setDateOfJoining(LocalDate dateOfJoining) {  
 this.dateOfJoining = dateOfJoining;  
 }  
  
 public String getEducationLevel() {  
 return educationLevel;  
 }  
  
 public void setEducationLevel(String educationLevel) {  
 this.educationLevel = educationLevel;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
}

# Enrollment.java

package com.example.demo.model;  
  
import jakarta.persistence.\*;  
  
import java.time.LocalDate;  
  
@Entity  
public class Enrollment {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int enrollmentId;  
  
 @ManyToOne  
 @JoinColumn(name = "employee\_id", nullable = false) // Employee reference  
 private Employee employee;  
  
 @ManyToOne  
 @JoinColumn(name = "course\_id", nullable = false) // Course reference  
 private Course course;  
  
 private LocalDate enrollmentDate;  
  
 private String status;  
  
 // Default constructor  
 public Enrollment() {  
 }  
  
 // Getters and Setters  
  
 public int getEnrollmentId() {  
 return enrollmentId;  
 }  
  
 public void setEnrollmentId(int enrollmentId) {  
 this.enrollmentId = enrollmentId;  
 }  
  
 public Employee getEmployee() {  
 return employee;  
 }  
  
 public void setEmployee(Employee employee) {  
 this.employee = employee;  
 }  
  
 public Course getCourse() {  
 return course;  
 }  
  
 public void setCourse(Course course) {  
 this.course = course;  
 }  
  
 public LocalDate getEnrollmentDate() {  
 return enrollmentDate;  
 }  
  
 public void setEnrollmentDate(LocalDate enrollmentDate) {  
 this.enrollmentDate = enrollmentDate;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
}

# Package controller:

# CourseController.java

package com.example.demo.controller;  
  
import com.example.demo.model.Course;  
import com.example.demo.service.CourseService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/courses")  
@CrossOrigin(origins = "http://localhost:3000") // Allow requests from your frontend  
public class CourseController {  
  
 @Autowired  
 private CourseService courseService;  
  
 // Endpoint to get all courses  
 @GetMapping  
 public ResponseEntity<List<Course>> getAllCourses() {  
 List<Course> courses = courseService.getAllCourses();  
 return ResponseEntity.*ok*(courses); // Return 200 OK status with the list of courses  
 }  
  
 // Endpoint to add a course  
 @PostMapping  
 public ResponseEntity<String> addCourse(@RequestBody Course course) {  
 courseService.addCourse(course.getCourseName(), course.getCourseDescription(), course.getDuration(), course.getCategory(), course.getCreatedBy());  
 return ResponseEntity.*ok*("Course added successfully!");  
 }  
  
 // Add other endpoints (delete, update) if necessary  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteCourse(@PathVariable int id) {  
 boolean isDeleted = courseService.deleteCourse(id);  
 if (isDeleted) {  
 return ResponseEntity.*ok*("Course deleted successfully!");  
 } else {  
 return ResponseEntity.*status*(404).body("Course not found!");  
 }  
 }  
  
 @PutMapping("/{id}/duration")  
 public ResponseEntity<String> updateCourseDuration(@PathVariable int id, @RequestBody int newDuration) {  
 boolean isUpdated = courseService.updateCourseDuration(id, newDuration);  
 if (isUpdated) {  
 return ResponseEntity.*ok*("Course duration updated successfully!");  
 } else {  
 return ResponseEntity.*status*(404).body("Course not found!");  
 }  
 }  
}

# EmployeeController.java

package com.example.demo.controller;  
  
import com.example.demo.model.Employee;  
import com.example.demo.service.EmployeeService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.http.ResponseEntity;  
  
import java.util.List;  
import java.util.Map;  
import java.util.Optional;  
  
@CrossOrigin(origins = "http://localhost:3000")  
@RestController  
@RequestMapping("/api/employees")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeService employeeService;  
  
 @GetMapping  
 public List<Employee> getAllEmployees() {  
 return employeeService.getAllEmployees();  
 }  
  
 @GetMapping("/{id}")  
 public Optional<Employee> getEmployeeById(@PathVariable int id) {  
 return employeeService.getEmployeeById(id);  
 }  
 // Signup Endpoint  
 @PostMapping("/signup")  
 public ResponseEntity<?> signupEmployee(@RequestBody Employee employee) {  
 // Check if employee already exists  
 if (employeeService.findByEmail(employee.getEmail()).isPresent()) {  
 return ResponseEntity.*badRequest*().body("Email already registered.");  
 }  
  
 // Save the new employee record  
 Employee savedEmployee = employeeService.addEmployee(employee);  
 return ResponseEntity.*ok*(Map.*of*("success", true, "message", "Employee account created", "employee", savedEmployee));  
 }  
  
 @PostMapping("/login")  
 public ResponseEntity<?> loginEmployee(@RequestBody Map<String, String> loginData) {  
 String email = loginData.get("email");  
 String password = loginData.get("password");  
  
 Optional<Employee> employee = employeeService.findByEmail(email);  
 if (employee.isPresent() && employee.get().getPassword().equals(password)) {  
 return ResponseEntity.*ok*(Map.*of*("success", true, "message", "Login successful"));  
 } else {  
 return ResponseEntity.*badRequest*().body(Map.*of*("success", false, "message", "Invalid email or password"));  
 }  
 }  
  
 // @DeleteMapping("/{id}")  
 // public void deleteEmployee(@PathVariable Long id) {  
 // employeeService.deleteEmployee(id);  
 // }  
}

# EnrollmentController.java

package com.example.demo.controller;  
  
import com.example.demo.dto.EnrollmentRequest;  
import com.example.demo.model.Course;  
import com.example.demo.model.Employee;  
import com.example.demo.model.Enrollment;  
import com.example.demo.repository.CourseRepository;  
import com.example.demo.repository.EmployeeRepository;  
import com.example.demo.repository.EnrollmentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/enrollments")  
@CrossOrigin(origins = "http://localhost:3000")  
public class EnrollmentController {  
  
 @Autowired  
 private EnrollmentRepository enrollmentRepository;  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Autowired  
 private CourseRepository courseRepository;  
  
 @PostMapping  
 public ResponseEntity<String> enrollEmployee(@RequestBody EnrollmentRequest enrollmentRequest) {  
 try {  
 // Check if employee exists  
 Optional<Employee> employee = employeeRepository.findById(enrollmentRequest.getEmployeeId());  
 if (employee.isEmpty()) {  
 return ResponseEntity.*status*(404).body("Employee not found with ID: " + enrollmentRequest.getEmployeeId());  
 }  
  
 // Check if course exists  
 Optional<Course> course = courseRepository.findById(enrollmentRequest.getCourseId());  
 if (course.isEmpty()) {  
 return ResponseEntity.*status*(404).body("Course not found with ID: " + enrollmentRequest.getCourseId());  
 }  
  
 // Create and save the enrollment record  
 Enrollment enrollment = new Enrollment();  
 enrollment.setEmployee(employee.get());  
 enrollment.setCourse(course.get());  
 enrollment.setEnrollmentDate(LocalDate.*parse*(enrollmentRequest.getEnrollmentDate()));  
 enrollment.setStatus(enrollmentRequest.getStatus());  
  
 enrollmentRepository.save(enrollment);  
  
 return ResponseEntity.*status*(201).body("Enrollment successful for employee ID " + employee.get().getEmployeeId() +  
 " in course ID " + course.get().getCourseId());  
  
 } catch (Exception e) {  
 return ResponseEntity.*status*(500).body("Error enrolling in course: " + e.getMessage());  
 }  
 }  
}

# Package repository:

# CourseRepository.java

package com.example.demo.repository;  
  
import com.example.demo.model.Course;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CourseRepository extends JpaRepository<Course, Integer> {  
}

# EnrollmentRepository.java

package com.example.demo.repository;  
  
import com.example.demo.model.Enrollment;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface EnrollmentRepository extends JpaRepository<Enrollment, Long> {  
 // You can add custom queries here if needed  
}

# EmployeeRepository.java

package com.example.demo.repository;  
  
import com.example.demo.model.Employee;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import java.util.Optional;  
  
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
 // Additional query methods can be added here if needed  
 Optional<Employee> findByEmail(String email);  
}

# Package Service:

# CourseService.java

package com.example.demo.service;  
  
import com.example.demo.model.Course;  
import com.example.demo.repository.CourseRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class CourseService {  
  
 @Autowired  
 private CourseRepository courseRepository;  
  
 // Method to add a new course  
 public void addCourse(String courseName, String courseDescription, int duration, String category, String createdBy) {  
 Course course = new Course(courseName, courseDescription, duration, category, createdBy);  
 courseRepository.save(course); // Save the course to the database  
 }  
  
 // Method to get all courses  
 public List<Course> getAllCourses() {  
 return courseRepository.findAll(); // Fetch all courses  
 }  
  
 // Method to get a course by ID  
 public Course getCourseById(int courseId) {  
 return courseRepository.findById(courseId).orElse(null);  
 }  
  
 public boolean deleteCourse(int courseId) {  
 if (courseRepository.existsById(courseId)) {  
 courseRepository.deleteById(courseId);  
 return true;  
 }  
 return false;  
 }  
  
 public boolean updateCourseDuration(int courseId, int newDuration) {  
 Optional<Course> courseOptional = courseRepository.findById(courseId);  
 if (courseOptional.isPresent()) {  
 Course course = courseOptional.get();  
 course.setDuration(newDuration); // Update the duration  
 courseRepository.save(course); // Save the updated course back to the database  
 return true;  
 } else {  
 return false; // Course not found  
 }  
 }  
}

# EmployeeService.java

package com.example.demo.service;  
  
import com.example.demo.model.Employee;  
import com.example.demo.repository.EmployeeRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 public List<Employee> getAllEmployees() {  
 return employeeRepository.findAll();  
 }  
  
 public Optional<Employee> getEmployeeById(int id) {  
 return employeeRepository.findById(id);  
 }  
  
 //public void deleteEmployee(Long id) {  
 //employeeRepository.deleteById(id);  
 // }  
  
 public Optional<Employee> findByEmail(String email) {  
 return employeeRepository.findByEmail(email);  
 }  
  
 public Employee addEmployee(Employee employee) {  
 return employeeRepository.save(employee); // This will insert a new employee if it doesn't exist  
 }  
}

# EnrollmentService.java

package com.example.demo.service;  
  
import com.example.demo.model.Enrollment;  
import com.example.demo.model.Employee;  
import com.example.demo.model.Course;  
import com.example.demo.repository.EnrollmentRepository;  
import com.example.demo.repository.EmployeeRepository;  
import com.example.demo.repository.CourseRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.Optional;  
  
@Service  
public class EnrollmentService {  
  
 @Autowired  
 private EnrollmentRepository enrollmentRepository;  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Autowired  
 private CourseRepository courseRepository;  
  
 public Enrollment enrollEmployee(int employeeId, int courseId) {  
 // Check if employee exists  
 Optional<Employee> employee = employeeRepository.findById(employeeId);  
 if (employee.isEmpty()) {  
 throw new RuntimeException("Employee not found");  
 }  
  
 // Check if course exists  
 Optional<Course> course = courseRepository.findById(courseId);  
 if (course.isEmpty()) {  
 throw new RuntimeException("Course not found");  
 }  
  
 // Create a new enrollment  
 Enrollment enrollment = new Enrollment();  
 enrollment.setEmployee(employee.get());  
 enrollment.setCourse(course.get());  
 enrollment.setEnrollmentDate(LocalDate.*now*());  
 enrollment.setStatus("Ongoing");  
  
 // Save the enrollment  
 return enrollmentRepository.save(enrollment);  
 }  
}

# Application.properties:

spring.application.name=demo  
server.port=8080  
# MySQL Database Configuration  
spring.datasource.url=jdbc:mysql://localhost:3306/project  
spring.datasource.username=root  
spring.datasource.password=@Arunprakash240703  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
  
# JPA Configuration  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

# FRONT-END:

# COMPONENTS:

# AddCourse.js

import React, { useState } from 'react';

import axios from 'axios';

import { Form, Button, Container, Alert } from 'react-bootstrap';

const AddCourse = () => {

  const [courseData, setCourseData] = useState({

    courseName: '',

    courseDescription: '',

    duration: '',

    category: '',

    createdBy: ''

  });

  const [success, setSuccess] = useState('');

  const [error, setError] = useState('');

  const handleChange = (e) => {

    const { name, value } = e.target;

    setCourseData({ ...courseData, [name]: value });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      await axios.post('http://localhost:8080/api/courses', {

        ...courseData

      });

      setSuccess('Course added successfully!');

      setError('');

      setCourseData({ courseName: '', courseDescription: '', duration: '', category: '', createdBy: '' });

    } catch (err) {

      setError('An error occurred while adding the course.');

      setSuccess('');

    }

  };

  return (

    <Container className="mt-5">

      <h2>Add New Course</h2>

      {error && <Alert variant="danger">{error}</Alert>}

      {success && <Alert variant="success">{success}</Alert>}

      <Form onSubmit={handleSubmit}>

        <Form.Group controlId="formCourseName">

          <Form.Label>Course Name</Form.Label>

          <Form.Control

            type="text"

            name="courseName"

            value={courseData.courseName}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formCourseDescription" className="mt-3">

          <Form.Label>Course Description</Form.Label>

          <Form.Control

            as="textarea"

            name="courseDescription"

            value={courseData.courseDescription}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formDuration" className="mt-3">

          <Form.Label>Duration (in hours)</Form.Label>

          <Form.Control

            type="number"

            name="duration"

            value={courseData.duration}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formCategory" className="mt-3">

          <Form.Label>Category</Form.Label>

          <Form.Control

            type="text"

            name="category"

            value={courseData.category}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formCreatedBy" className="mt-3">

          <Form.Label>Instructor</Form.Label>

          <Form.Control

            type="text"

            name="createdBy"

            value={courseData.createdBy}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Button variant="primary" type="submit" className="mt-4">

          Add Course

        </Button>

      </Form>

    </Container>

  );

};

export default AddCourse;

# CoursePage.js

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import { Container, Row, Col, Card, Alert, Button } from 'react-bootstrap';

import { useNavigate } from 'react-router-dom';

const CoursePage = () => {

  const [courses, setCourses] = useState([]);

  const [error, setError] = useState('');

  const [success, setSuccess] = useState('');

  const navigate = useNavigate();

  // Fetch all courses when the component mounts

  const fetchCourses = async () => {

    try {

      const response = await axios.get('http://localhost:8080/api/courses');

      setCourses(response.data);

    } catch (err) {

      console.error('Error fetching courses:', err);

      setError('Error fetching courses.');

    }

  };

  useEffect(() => {

    fetchCourses();

  }, []);

  // Delete a course

  const deleteCourse = async (id) => {

    try {

      await axios.delete(`http://localhost:8080/api/courses/${id}`);

      setSuccess('Course deleted successfully!');

      setCourses(courses.filter(course => course.courseId !== id));

    } catch (err) {

      console.error('Error deleting course:', err);

      setError('An error occurred while deleting the course.');

    }

  };

  // Navigate to Enrollment page with courseId

  const handleEnroll = (courseId) => {

    navigate(`/enroll/${courseId}`);

  };

  return (

    <Container className="mt-5">

      <h2>Available Courses</h2>

      {error && <Alert variant="danger">{error}</Alert>}

      {success && <Alert variant="success">{success}</Alert>}

      <Row>

        {courses.length > 0 ? (

          courses.map(course => (

            <Col md={4} key={course.courseId} className="mb-4">

              <Card>

                <Card.Body>

                  <Card.Title>{course.courseName}</Card.Title>

                  <Card.Subtitle className="mb-2 text-muted">{course.category}</Card.Subtitle>

                  <Card.Text>{course.courseDescription}</Card.Text>

                  <Card.Text><strong>Duration:</strong> {course.duration} hours</Card.Text>

                  <Card.Text><strong>Instructor:</strong> {course.createdBy}</Card.Text>

                  <Button variant="danger" onClick={() => deleteCourse(course.courseId)}>Delete</Button>

                  <Button variant="primary" onClick={() => handleEnroll(course.courseId)} className="ms-2">

                    Enroll

                  </Button>

                </Card.Body>

              </Card>

            </Col>

          ))

        ) : (

          <p>No courses available at the moment.</p>

        )}

      </Row>

    </Container>

  );

};

export default CoursePage;

# Enrollment.js

import React, { useState } from 'react';

import axios from 'axios';

import { useParams } from 'react-router-dom';

import { Container, Form, Button, Alert } from 'react-bootstrap';

const Enrollment = () => {

  const { courseId } = useParams(); // Get courseId from URL

  const [employeeId, setEmployeeId] = useState('');

  const [success, setSuccess] = useState('');

  const [error, setError] = useState('');

  const handleEnroll = async (e) => {

    e.preventDefault();

    try {

        const enrollmentData = {

            employeeId: employeeId,    // This matches the DTO field name

            courseId: courseId,        // This matches the DTO field name

            enrollmentDate: new Date().toISOString().split('T')[0], // Date in YYYY-MM-DD format

            status: 'Ongoing'          // Status is directly set

          };

      const response = await axios.post('http://localhost:8080/api/enrollments', enrollmentData, {

        headers: {

          'Content-Type': 'application/json'

        }

      });

      if (response.status === 200 || response.status === 201) {

        setSuccess('Enrollment successful!');

        setError('');

      } else {

        setError('Enrollment failed: Unexpected server response');

        setSuccess('');

      }

    } catch (err) {

      console.error('Error during enrollment:', err);

      // Check if err.response exists for a more specific error message

      const errorMessage = err.response?.data?.message || err.message;

      setError(`Failed to enroll in the course: ${errorMessage}`);

      setSuccess('');

    }

  };

  return (

    <Container className="mt-5">

      <h2>Enroll in Course</h2>

      {error && <Alert variant="danger">{error}</Alert>}

      {success && <Alert variant="success">{success}</Alert>}

      <Form onSubmit={handleEnroll}>

        <Form.Group controlId="formEmployeeId">

          <Form.Label>Employee ID</Form.Label>

          <Form.Control

            type="text"

            value={employeeId}

            onChange={(e) => setEmployeeId(e.target.value)}

            required

          />

        </Form.Group>

        <Form.Group controlId="formStatus" className="mt-3">

          <Form.Label>Status</Form.Label>

          <Form.Control

            type="text"

            value="Ongoing"

            readOnly

          />

        </Form.Group>

        <Button variant="primary" type="submit" className="mt-4">

          Enroll

        </Button>

      </Form>

    </Container>

  );

};

export default Enrollment;

# HomePage.js

// HomePage.js

import React from 'react';

import { useNavigate } from 'react-router-dom';

import { Container, Button, Row, Col } from 'react-bootstrap';

const HomePage = () => {

  const navigate = useNavigate();

  return (

    <Container className="d-flex justify-content-center align-items-center" style={{ height: '100vh' }}>

      <Row>

        <Col className="text-center">

          <h1>Welcome to Our Account Creation Platform</h1>

          <p>Please log in or sign up to access our courses.</p>

          <Button variant="primary" onClick={() => navigate('/login')} className="m-2">

            Login

          </Button>

          <Button variant="success" onClick={() => navigate('/signup')} className="m-2">

            Signup

          </Button>

        </Col>

      </Row>

    </Container>

  );

};

export default HomePage;

# Login.js

import React, { useState } from 'react';

import axios from 'axios';

import { Form, Button, Container, Alert } from 'react-bootstrap';

import { useNavigate } from 'react-router-dom'

const Login = () => {

  const [formData, setFormData] = useState({

    email: '',

    password: ''

  });

  const [success, setSuccess] = useState(null);

  const [error, setError] = useState(null);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({ ...formData, [name]: value });

  };

  const navigate = useNavigate();

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post('http://localhost:8080/api/employees/login', formData);

      if (response.data.success) {

        setSuccess('Login successful!');

        setError(null);

        navigate('/courses'); // Redirect to CoursePage.js after successful login

      } else {

        setError(response.data.message || 'Invalid login credentials');

        setSuccess(null);

      }

    } catch (error) {

      setError('Invalid password or email');

      setSuccess(null);

    }

  };

  return (

    <Container className="mt-5">

      <h2 className="text-center mb-4">Employee Login</h2>

      {error && <Alert variant="danger">{error}</Alert>}

      {success && <Alert variant="success">{success}</Alert>}

      <Form onSubmit={handleSubmit}>

        <Form.Group controlId="formEmail">

          <Form.Label>Email</Form.Label>

          <Form.Control

            type="email"

            name="email"

            value={formData.email}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formPassword" className="mt-3">

          <Form.Label>Password</Form.Label>

          <Form.Control

            type="password"

            name="password"

            value={formData.password}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Button variant="primary" type="submit" className="mt-4">

          Log In

        </Button>

      </Form>

    </Container>

  );

};

export default Login;

# Login.js

import React from 'react';

import { Navbar, Nav, Container } from 'react-bootstrap';

import { NavLink } from 'react-router-dom';

const NavBarComp = () => {

  return (

    <Navbar bg="light" expand="lg">

      <Container>

        <Navbar.Brand as={NavLink} to="/">MyApp</Navbar.Brand>

        <Navbar.Toggle aria-controls="basic-navbar-nav" />

        <Navbar.Collapse id="basic-navbar-nav">

          <Nav className="me-auto">

            <Nav.Link as={NavLink} to="/" exact>Home</Nav.Link>

            <Nav.Link as={NavLink} to="/login">Login</Nav.Link>

            <Nav.Link as={NavLink} to="/signup">SignUp</Nav.Link>

            <Nav.Link as={NavLink} to="/courses">Courses</Nav.Link>

            <Nav.Link as={NavLink} to="/add-course">Add Course</Nav.Link>

          </Nav>

        </Navbar.Collapse>

      </Container>

    </Navbar>

  );

};

export default NavBarComp;

# SignUp.js

import React, { useState } from 'react';

import axios from 'axios';

import { Form, Button, Container, Alert } from 'react-bootstrap';

const Signup = () => {

  const [formData, setFormData] = useState({

    firstName: '',

    lastName: '',

    email: '',

    department: '',

    position: '',

    dateOfJoining: '',

    educationLevel: '',

    password: ''

  });

  const [success, setSuccess] = useState(null);

  const [error, setError] = useState(null);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({ ...formData, [name]: value });

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post('http://localhost:8080/api/employees/signup', formData);

      if (response.data.success) {

        setSuccess('Account created successfully!');

        setError(null);

      } else {

        setError('Error creating account');

        setSuccess(null);

      }

    } catch (error) {

      setError('An error occurred. Please try again.');

      setSuccess(null);

    }

  };

  return (

    <Container className="mt-5">

      <h2 className="text-center mb-4">Employee Signup</h2>

      {error && <Alert variant="danger">{error}</Alert>}

      {success && <Alert variant="success">{success}</Alert>}

      <Form onSubmit={handleSubmit}>

        <Form.Group controlId="formFirstName">

          <Form.Label>First Name</Form.Label>

          <Form.Control

            type="text"

            name="firstName"

            value={formData.firstName}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formLastName" className="mt-3">

          <Form.Label>Last Name</Form.Label>

          <Form.Control

            type="text"

            name="lastName"

            value={formData.lastName}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formEmail" className="mt-3">

          <Form.Label>Email</Form.Label>

          <Form.Control

            type="email"

            name="email"

            value={formData.email}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formDepartment" className="mt-3">

          <Form.Label>Department</Form.Label>

          <Form.Control

            type="text"

            name="department"

            value={formData.department}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formPosition" className="mt-3">

          <Form.Label>Position</Form.Label>

          <Form.Control

            type="text"

            name="position"

            value={formData.position}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formDateOfJoining" className="mt-3">

          <Form.Label>Date of Joining</Form.Label>

          <Form.Control

            type="date"

            name="dateOfJoining"

            value={formData.dateOfJoining}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formEducationLevel" className="mt-3">

          <Form.Label>Education Level</Form.Label>

          <Form.Control

            type="text"

            name="educationLevel"

            value={formData.educationLevel}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Form.Group controlId="formPassword" className="mt-3">

          <Form.Label>Password</Form.Label>

          <Form.Control

            type="password"

            name="password"

            value={formData.password}

            onChange={handleChange}

            required

          />

        </Form.Group>

        <Button variant="primary" type="submit" className="mt-4">

          Sign Up

        </Button>

      </Form>

    </Container>

  );

};

export default Signup;

# App.js

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import NavBarComp from './components/NavBar';

import Home from './components/HomePage';

import Login from './components/Login';

import Signup from './components/SignUp';

import CoursePage from './components/CoursePage';

import AddCourse from './components/AddCourse';

import Enrollment from './components/Enrollement';

import 'bootstrap/dist/css/bootstrap.min.css';

function App() {

  return (

    <Router>

      <NavBarComp />

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="/login" element={<Login />} />

        <Route path="/signup" element={<Signup />} />

        <Route path="/courses" element={<CoursePage />} />

        <Route path="/add-course" element={<AddCourse />} />

        <Route path="/enroll/:courseId" element={<Enrollment />} />

      </Routes>

    </Router>

  );

}

export default App;

# MySQL Database:

# Use project;

# CREATE TABLE employees (

# employee\_id INT PRIMARY KEY AUTO\_INCREMENT,

# first\_name VARCHAR(50) NOT NULL,

# last\_name VARCHAR(50) NOT NULL,

# email VARCHAR(100) NOT NULL UNIQUE,

# department VARCHAR(100),

# position VARCHAR(100),

# date\_of\_joining DATE,

# education\_level VARCHAR(50)

# );

# select \* from employees;

# Truncate table employees;

# CREATE TABLE courses (

# course\_id INT AUTO\_INCREMENT PRIMARY KEY,

# course\_name VARCHAR(255) NOT NULL,

# course\_description TEXT,

# duration INT NOT NULL,

# category VARCHAR(100),

# created\_by VARCHAR(255) NOT NULL

# );

# select \* from courses;

# CREATE TABLE enrollments (

# enrollment\_id INT AUTO\_INCREMENT PRIMARY KEY,

# employee\_id INT NOT NULL,

# course\_id INT NOT NULL,

# enrollment\_date DATE NOT NULL,

# status VARCHAR(50) NOT NULL,

# 

# FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id),

# FOREIGN KEY (course\_id) REFERENCES Courses(course\_id)

# select \* from enrollments;

# SNAPSHOTS

# Home Page

# 

# 

# 

# 

# 

# 

# RESULTS AND DISCUSSION

The **Course Enrollment Management System** successfully fulfills the requirements for managing courses and enrollments. Key functionalities, such as course creation, deletion, updating course details, and employee enrollment, are implemented and tested. The integration of a user-friendly frontend with React.js and a robust backend using Spring Boot ensures smooth and reliable operations. Data is stored securely in a MySQL database, enabling efficient handling of course and enrollment records. The system also provides real-time error handling, ensuring a seamless user experience.

**CONCLUSION**

The **Course Enrollment Management System** successfully streamlines the process of managing courses and enrolling employees, providing an efficient and user-friendly solution. By integrating React.js for the frontend, Spring Boot for the backend, and MySQL for data management, the system achieves a seamless flow of operations. It addresses key requirements such as course management, enrollment handling, and data security. While the current implementation meets foundational needs, future enhancements can introduce features like automated notifications, progress tracking, and analytics to further enrich the user experience. This project lays a strong groundwork for modernizing course enrollment and management processes in organizational and educational settings.

# REFERENCES

# MySQL Tutorial:

# <https://www.javatpoint.com/mysql-tutorial>

# Spring Boot Tutorial:

# <https://www.javatpoint.com/spring-boot-tutorial>

# JAVA Tutorial:

# <https://www.javatpoint.com/java-tutorial>

# React Tutorial:

# <https://legacy.reactjs.org/tutorial/tutorial.html>