**Section 1: OOAD Lab**

Q1. Draw Class Diagram for Online Examination System. Make necessary assumptions required

**Section 2: Web Technologies Lab**

Q1. Write a program using JSP and JDBC to support editing (address modification, mobile number/ email id update) of MCA 1st Semester students of IGNOU. The program should take enrollment number or registered mobile number as input. Make necessary assumptions required.

To create a JSP program that allows editing of MCA 1st Semester students' details (address modification, mobile number/email ID update) at IGNOU using JDBC, you'll need the following components:

1. **Database setup**: You'll need a MySQL (or any other) database with a table to hold the student information.
2. **JSP pages**: To capture user input and display the results.
3. **JDBC connection**: To connect to the database and perform CRUD operations.

**Step 1: Database Setup**

Assuming you have a database named ignou\_db, you can create a table students as follows:

sql

Copy code

CREATE TABLE students (

id INT AUTO\_INCREMENT PRIMARY KEY,

enrollment\_number VARCHAR(20) UNIQUE NOT NULL,

mobile\_number VARCHAR(15),

email\_id VARCHAR(50),

address TEXT

);

**Step 2: JSP Pages**

**1. index.jsp**: This page will take enrollment number or mobile number as input to fetch student details.

jsp

Copy code

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ page import="java.sql.\*" %>

<!DOCTYPE html>

<html>

<head>

<title>Edit Student Details</title>

</head>

<body>

<h2>Edit Student Details</h2>

<form action="editStudent.jsp" method="post">

<label for="identifier">Enrollment Number or Mobile Number:</label>

<input type="text" id="identifier" name="identifier" required>

<input type="submit" value="Fetch Details">

</form>

</body>

</html>

**2. editStudent.jsp**: This page will display the student details for editing.

jsp

Copy code

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ page import="java.sql.\*" %>

<!DOCTYPE html>

<html>

<head>

<title>Edit Student Details</title>

</head>

<body>

<h2>Edit Student Details</h2>

<%

String identifier = request.getParameter("identifier");

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

try {

// Load JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ignou\_db", "username", "password");

// Query to fetch student details

String sql = "SELECT \* FROM students WHERE enrollment\_number = ? OR mobile\_number = ?";

stmt = conn.prepareStatement(sql);

stmt.setString(1, identifier);

stmt.setString(2, identifier);

rs = stmt.executeQuery();

if (rs.next()) {

String enrollmentNumber = rs.getString("enrollment\_number");

String mobileNumber = rs.getString("mobile\_number");

String emailId = rs.getString("email\_id");

String address = rs.getString("address");

%>

<form action="updateStudent.jsp" method="post">

<input type="hidden" name="enrollmentNumber" value="<%= enrollmentNumber %>">

<label for="mobile">Mobile Number:</label>

<input type="text" id="mobile" name="mobile" value="<%= mobileNumber %>" required><br>

<label for="email">Email ID:</label>

<input type="email" id="email" name="email" value="<%= emailId %>" required><br>

<label for="address">Address:</label>

<textarea id="address" name="address" required><%= address %></textarea><br>

<input type="submit" value="Update">

</form>

<%

} else {

%>

<p>No student found with the provided details.</p>

<%

}

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close resources

if (rs != null) try { rs.close(); } catch (SQLException e) {}

if (stmt != null) try { stmt.close(); } catch (SQLException e) {}

if (conn != null) try { conn.close(); } catch (SQLException e) {}

}

%>

</body>

</html>

**3. updateStudent.jsp**: This page handles the update operation.

jsp

Copy code

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ page import="java.sql.\*" %>

<!DOCTYPE html>

<html>

<head>

<title>Update Student Details</title>

</head>

<body>

<h2>Update Student Details</h2>

<%

String enrollmentNumber = request.getParameter("enrollmentNumber");

String mobile = request.getParameter("mobile");

String email = request.getParameter("email");

String address = request.getParameter("address");

Connection conn = null;

PreparedStatement stmt = null;

try {

// Load JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ignou\_db", "username", "password");

// Query to update student details

String sql = "UPDATE students SET mobile\_number = ?, email\_id = ?, address = ? WHERE enrollment\_number = ?";

stmt = conn.prepareStatement(sql);

stmt.setString(1, mobile);

stmt.setString(2, email);

stmt.setString(3, address);

stmt.setString(4, enrollmentNumber);

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

%>

<p>Student details updated successfully!</p>

<%

} else {

%>

<p>Failed to update student details.</p>

<%

}

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close resources

if (stmt != null) try { stmt.close(); } catch (SQLException e) {}

if (conn != null) try { conn.close(); } catch (SQLException e) {}

}

%>

</body>

</html>

**Step 3: Configuring the JDBC Driver**

Ensure you have the MySQL JDBC driver (mysql-connector-java-x.x.x.jar) in your classpath. If you're using Apache Tomcat, place it in the lib directory.

**Step 4: Running the Application**

1. Deploy the JSP files in a web server like Apache Tomcat.
2. Access the application via your browser using the URL (e.g., http://localhost:8080/your-app-name/index.jsp).
3. Enter an enrollment number or mobile number to fetch the student's details and make edits.

**Assumptions**

* The database is set up correctly and is running.
* The username and password for the MySQL database connection are valid.
* Error handling is minimal and can be expanded based on needs.
* The JSP files are located in the appropriate directory in the web server.

Q2. Write a program to create simple CRUD (Create, Read, Update, and Delete) application using Spring Boot and Hibernate for Online Registration and Fee Payment for a Workshop on “Web Technologies” to be organized by an IT Training Organization. Make provisions for security management in the program. Make necessary assumptions required.

Creating a simple CRUD application using Spring Boot and Hibernate for online registration and fee payment for a workshop involves several steps. Below is an overview of the components and a sample implementation.

**Prerequisites**

1. **Java Development Kit (JDK)**: Ensure you have JDK 8 or higher installed.
2. **Maven**: Use Maven for dependency management.
3. **Spring Boot**: Use Spring Boot for rapid application development.
4. **MySQL**: Use MySQL (or any other database) for data storage.

**Step 1: Create Spring Boot Project**

You can create a Spring Boot project using [Spring Initializr](https://start.spring.io/). Select the following dependencies:

* Spring Web
* Spring Data JPA
* MySQL Driver
* Spring Security (for security management)
* Spring Boot DevTools (optional, for development)

**Step 2: Project Structure**

Your project structure should look something like this:

css

Copy code

workshop-registration

├── src

│ ├── main

│ │ ├── java

│ │ │ └── com

│ │ │ └── example

│ │ │ └── workshopregistration

│ │ │ ├── controller

│ │ │ ├── model

│ │ │ ├── repository

│ │ │ ├── service

│ │ │ └── WorkshopRegistrationApplication.java

│ │ └── resources

│ │ ├── application.properties

│ │ └── templates

└── pom.xml

**Step 3: Add Dependencies in pom.xml**

xml

Copy code

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

</dependencies>

**Step 4: Configuration in application.properties**

Configure your MySQL database connection in src/main/resources/application.properties:

properties

Copy code

spring.datasource.url=jdbc:mysql://localhost:3306/workshop\_db

spring.datasource.username=root

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Step 5: Create the Model Class**

Create a model class for the Workshop Registration.

**Workshop.java** in model package:

java

Copy code

package com.example.workshopregistration.model;

import javax.persistence.\*;

@Entity

@Table(name = "registrations")

public class Workshop {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String mobile;

private double feePaid;

// Getters and Setters

}

**Step 6: Create Repository Interface**

Create a repository interface for database operations.

**WorkshopRepository.java** in repository package:

java

Copy code

package com.example.workshopregistration.repository;

import com.example.workshopregistration.model.Workshop;

import org.springframework.data.jpa.repository.JpaRepository;

public interface WorkshopRepository extends JpaRepository<Workshop, Long> {

}

**Step 7: Create Service Class**

Create a service class to handle business logic.

**WorkshopService.java** in service package:

java

Copy code

package com.example.workshopregistration.service;

import com.example.workshopregistration.model.Workshop;

import com.example.workshopregistration.repository.WorkshopRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class WorkshopService {

@Autowired

private WorkshopRepository workshopRepository;

public List<Workshop> findAll() {

return workshopRepository.findAll();

}

public Workshop findById(Long id) {

return workshopRepository.findById(id).orElse(null);

}

public Workshop save(Workshop workshop) {

return workshopRepository.save(workshop);

}

public void delete(Long id) {

workshopRepository.deleteById(id);

}

}

**Step 8: Create Controller Class**

Create a controller to handle HTTP requests.

**WorkshopController.java** in controller package:

java

Copy code

package com.example.workshopregistration.controller;

import com.example.workshopregistration.model.Workshop;

import com.example.workshopregistration.service.WorkshopService;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

import java.util.List;

@Controller

@RequestMapping("/workshops")

public class WorkshopController {

@Autowired

private WorkshopService workshopService;

@GetMapping

public String listWorkshops(Model model) {

List<Workshop> workshops = workshopService.findAll();

model.addAttribute("workshops", workshops);

return "workshop/list";

}

@GetMapping("/register")

public String showRegistrationForm(Model model) {

model.addAttribute("workshop", new Workshop());

return "workshop/register";

}

@PostMapping("/register")

public String registerWorkshop(@ModelAttribute Workshop workshop) {

workshopService.save(workshop);

return "redirect:/workshops";

}

@GetMapping("/edit/{id}")

public String showEditForm(@PathVariable("id") Long id, Model model) {

Workshop workshop = workshopService.findById(id);

model.addAttribute("workshop", workshop);

return "workshop/edit";

}

@PostMapping("/update/{id}")

public String updateWorkshop(@PathVariable("id") Long id, @ModelAttribute Workshop workshop) {

workshop.setId(id);

workshopService.save(workshop);

return "redirect:/workshops";

}

@GetMapping("/delete/{id}")

public String deleteWorkshop(@PathVariable("id") Long id) {

workshopService.delete(id);

return "redirect:/workshops";

}

}

**Step 9: Create HTML Templates**

Create the HTML templates in src/main/resources/templates/workshop/.

1. **list.html**:

html

Copy code

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Workshop Registrations</title>

</head>

<body>

<h1>Workshop Registrations</h1>

<a href="/workshops/register">Register for Workshop</a>

<table>

<tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

<th>Mobile</th>

<th>Fee Paid</th>

<th>Actions</th>

</tr>

<tr th:each="workshop : ${workshops}">

<td th:text="${workshop.id}"></td>

<td th:text="${workshop.name}"></td>

<td th:text="${workshop.email}"></td>

<td th:text="${workshop.mobile}"></td>

<td th:text="${workshop.feePaid}"></td>

<td>

<a th:href="@{/workshops/edit/{id}(id=${workshop.id})}">Edit</a>

<a th:href="@{/workshops/delete/{id}(id=${workshop.id})}">Delete</a>

</td>

</tr>

</table>

</body>

</html>

1. **register.html**:

html

Copy code

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Register for Workshop</title>

</head>

<body>

<h1>Register for Workshop</h1>

<form action="#" th:action="@{/workshops/register}" th:object="${workshop}" method="post">

<label>Name:</label>

<input type="text" th:field="\*{name}" required><br>

<label>Email:</label>

<input type="email" th:field="\*{email}" required><br>

<label>Mobile:</label>

<input type="text" th:field="\*{mobile}" required><br>

<label>Fee Paid:</label>

<input type="number" th:field="\*{feePaid}" required><br>

<button type="submit">Register</button>

</form>

<a href="/workshops">Back to List</a>

</body>

</html>

1. **edit.html**:

html

Copy code

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Edit Workshop</title>

</head>

<body>

<h1>Edit Workshop</h1>

<form action="#" th:action="@{/workshops/update/{id}(id=${workshop.id})}" th:object="${workshop}" method="post">

<label>Name:</label>

<input type="text" th:field="\*{name}" required><br>

<label>Email:</label>

<input type="email" th:field="\*{email}" required><br>

<label>Mobile:</label>

<input type="text" th:field="\*{mobile}" required><br>

<label>Fee Paid:</label>

<input type="number" th:field="\*{feePaid}" required><br>

<button type="submit">Update</button>

</form>

<a href="/workshops">Back to List</a>

</body>

</html>

**Step 10: Security Configuration**

Create a basic security configuration for user authentication.

**WebSecurityConfig.java** in a new config package:

java

Copy code

package com.example.workshopregistration.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/workshops/register", "/workshops").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.permitAll();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

**Step 11: Create Login Page**

Create a login page login.html in src/main/resources/templates/.

**login.html**:

html

Copy code

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Login</title>

</head>

<body>

<h1>Login</h1>

<form action="#" th:action="@{/login}" method="post">

<label>Username:</label>

<input type="text" name="username" required><br>

<label>Password:</label>

<input type="password" name="password" required><br>

<button type="submit">Login</button>

</form>

</body>

</html>

**Step 12: Run the Application**

Run the application by executing the main method in WorkshopRegistrationApplication.java.

**Step 13: Access the Application**

Open your browser and go to http://localhost:8080/workshops to access the application. You can register, view, edit, and delete workshop registrations.

**Assumptions**

1. **Database**: MySQL is running, and a database named workshop\_db exists.
2. **User Authentication**: Basic authentication is implemented, and a user table can be created for managing users.
3. **Error Handling**: Minimal error handling is provided. You can expand it based on your requirements.
4. **Thymeleaf**: This example uses Thymeleaf as the template engine. Ensure it’s included in your dependencies.