**Web Technology**

**Problem Statement No.1**

Develop a simple online bookstore's inventory management system using JSP and JSTL. The bookstore has various genres of books, and you need to provide an interface for both users and administrators.

**Requirements:**

1. **User Interface:** Create a JSP page that displays a list of books available in the bookstore. Each book should have the following details:
   * Title
   * Author
   * Genre
   * Price
   * Availability (In stock/Out of stock)
2. **Filtering:** Implement filtering options using JSTL to allow users to filter books by genre. The genres should be displayed as checkboxes on the page.
3. **Admin Functionality:** Create an admin JSP page where administrators can add new books to the inventory. Include a form that collects the necessary information about the book.
4. **Data Source:** Use an in-memory list or a mock database (like an array or a list) to manage the book data, with sample data provided.

**Problem Statement 2: User Registration and Login System**

**Background:** You are developing a user registration and login system for a small web application. The application allows users to create an account and log in to access personalized content.

**Requirements:**

1. **Registration Page:** Create a JSP page for user registration that collects the following information:
   * Username
   * Password
   * Email
   * Confirm Password
2. **Validation:** Use JSTL to validate the user inputs. Implement checks to ensure that:
   * All fields are filled out.
   * The password and confirm password fields match.
   * The email format is valid.
3. **Login Page:** Create a separate JSP page for users to log in. The login form should ask for the username and password.
4. **Session Management:** Once a user successfully logs in, create a session and display a welcome message on a new JSP page. If the login fails, display an error message using JSTL.
5. **User Data Storage:** For simplicity, use a Java List to store user data in memory. Implement a way to check for existing usernames during registration.

**Problem Statement 3: Task Management Application**

**Background:** You are developing a task management application that allows users to create, update, and delete tasks. Users can also mark tasks as complete and filter tasks based on their status.

**Requirements:**

1. **Task Entity:** Create a Task entity class with the following attributes:
   * ID (auto-generated)
   * Title
   * Description
   * Status (Pending, In Progress, Completed)
   * Due Date
2. **Repository Layer:** Implement a Spring Data JPA repository to handle CRUD operations for the Task entity.
3. **Service Layer:** Create a service class that provides methods for:
   * Creating a new task
   * Updating an existing task
   * Deleting a task
   * Retrieving all tasks
4. **Controller Layer:** Develop a Spring MVC controller with the following endpoints:
   * GET /tasks - Retrieve all tasks.
   * GET /tasks/{status} - Filter tasks by status.
   * POST /tasks - Create a new task.
   * PUT /tasks/{id} - Update an existing task.
   * DELETE /tasks/{id} - Delete a task.
5. **View Layer:** Use Thymeleaf to create HTML templates for displaying the task list and a form for creating/updating tasks.

**Problem Statement 4: E-Commerce Product Catalogue**

**Background:** You are tasked with building an e-commerce application that features a product catalogue where users can view, add, update, and delete products.

**Requirements:**

1. **Product Entity:** Create a Product entity class with the following attributes:
   * ID (auto-generated)
   * Name
   * Description
   * Price
   * Stock Quantity
   * Category
2. **Repository Layer:** Implement a Spring Data JPA repository for managing the Product entity.
3. **Service Layer:** Create a service class that includes methods for:
   * Adding a new product
   * Updating an existing product
   * Deleting a product
   * Retrieving all products
4. **Controller Layer:** Develop a Spring MVC controller with the following endpoints:
   * GET /products - Retrieve and display all products.
   * GET /products/{id} - Retrieve product details by ID.
   * POST /products - Add a new product.
   * PUT /products/{id} - Update an existing product.
   * DELETE /products/{id} - Delete a product.
5. **View Layer:** Use Thymeleaf to create HTML templates for displaying the product catalog, product details, and forms for adding/updating products.
6. **Data Validation:** Implement validation for product fields using Spring’s validation annotations.

**Problem Statement 5: Online Grocery Store API**

**Background:** You are tasked with developing a RESTful API for an online grocery store. The API should allow users to manage grocery items, including searching, adding, updating, and deleting items from the inventory.

**Requirements:**

1. **Grocery Item Entity:** Create a GroceryItem entity class with the following attributes:
   * ID (auto-generated)
   * Name
   * Description
   * Price
   * Quantity in Stock
   * Category
2. **Repository Layer:** Use Spring Data JPA to create a repository interface for managing GroceryItem entities.
3. **Service Layer:** Implement a service class that provides methods for:
   * Adding a new grocery item
   * Updating an existing grocery item
   * Deleting a grocery item by ID
   * Retrieving all grocery items
   * Searching for items by name or category
4. **Controller Layer:** Develop a RESTful controller with the following endpoints:
   * GET /api/grocery-items - Retrieve all grocery items.
   * GET /api/grocery-items/{id} - Retrieve a specific item by ID.
   * POST /api/grocery-items - Add a new grocery item.
   * PUT /api/grocery-items/{id} - Update an existing grocery item.
   * DELETE /api/grocery-items/{id} - Delete a grocery item.
5. **Data Validation:** Implement input validation for the GroceryItem fields using Spring's validation annotations.
6. **Exception Handling:** Create a global exception handler to manage errors and return appropriate HTTP status codes and messages.

**Problem Statement 6: Employee Management System**

**Background:** You are developing an Employee Management System to manage employee records within a company. The system should allow HR personnel to perform CRUD operations on employee data.

**Requirements:**

1. **Employee Entity:** Create an Employee entity class with the following attributes:
   * ID (auto-generated)
   * First Name
   * Last Name
   * Email
   * Phone Number
   * Position
   * Salary
2. **Repository Layer:** Use Spring Data JPA to create a repository interface for managing Employee entities.
3. **Service Layer:** Implement a service class that provides methods for:
   * Adding a new employee
   * Updating an existing employee
   * Deleting an employee by ID
   * Retrieving all employees
   * Retrieving an employee by ID
4. **Controller Layer:** Develop a RESTful controller with the following endpoints:
   * GET /api/employees - Retrieve all employees.
   * GET /api/employees/{id} - Retrieve a specific employee by ID.
   * POST /api/employees - Add a new employee.
   * PUT /api/employees/{id} - Update an existing employee.
   * DELETE /api/employees/{id} - Delete an employee.

**Problem Statement 7: Library Management System**

**Background:** You are tasked with developing a library management system that allows users to manage books and their associated authors. The system should facilitate operations like adding new books, updating existing ones, and retrieving information about books and authors.

**Requirements:**

1. **Entities:**
   * **Book Entity:** Create a Book entity class with the following attributes:
     + ID (auto-generated)
     + Title
     + ISBN
     + Published Date
     + Number of Copies
   * **Author Entity:** Create an Author entity class with the following attributes:
     + ID (auto-generated)
     + Name
     + Biography
     + List of Books (one-to-many relationship with Book)
2. **Mapping:** Use Hibernate annotations to establish a one-to-many relationship between Author and Book. An author can write multiple books, while a book has only one author.
3. **Repository Layer:** Create a repository interface for both entities to perform CRUD operations using Hibernate.
4. **Service Layer:** Implement a service class that provides methods for:
   * Adding a new book and author
   * Updating book details
   * Retrieving all books and their authors
   * Searching for books by title or author name
5. **Transaction Management:** Ensure that all operations that modify data are wrapped in a transaction to maintain data integrity.

**Problem Statement 8: Online Course Registration System**

**Background:** You are developing an online course registration system that allows students to enroll in courses. The system should manage courses, students, and their registrations efficiently.

**Requirements:**

1. **Entities:**
   * **Course Entity:** Create a Course entity class with the following attributes:
     + ID (auto-generated)
     + Title
     + Description
     + Credits
     + List of Students (many-to-many relationship)
   * **Student Entity:** Create a Student entity class with the following attributes:
     + ID (auto-generated)
     + Name
     + Email
     + List of Courses (many-to-many relationship)
2. **Mapping:** Use Hibernate annotations to establish a many-to-many relationship between Student and Course. Each student can enroll in multiple courses, and each course can have multiple students.
3. **Repository Layer:** Create repository interfaces for both entities to perform CRUD operations using Hibernate.
4. **Service Layer:** Implement a service class that provides methods for:
   * Enrolling a student in a course
   * Unenrolling a student from a course
   * Retrieving all courses for a student
   * Retrieving all students enrolled in a course
5. **Transaction Management:** Ensure that all enrollment and unenrollment operations are wrapped in a transaction to maintain consistency.