Final Capstone Project (Day 8 to Day 12)

Day 8 - 10

Assignment 1: Coffee Shop Management System (CSMS) API with Spring Boot and REST APIs

Objective:

Develop a RESTful API for the Coffee Shop Management System (CSMS) using **Spring Boot**. This API will allow clients to perform CRUD operations for managing customers, menu items, orders, and payments in a coffee shop system.

Requirements:

Using the knowledge from **Day 8-10** on **Spring Boot** and **REST APIs**, implement an API for CSMS that:

- Manages operations for customers, menu items, orders, and payments.
- Follows REST principles with CRUD endpoints.
- Applies key Spring Boot concepts: Controllers, Services, Repositories, Exception Handling, and Validation.

Problem Statement:

Create a Coffee Shop Management API that allows clients to interact with a coffee shop's data, including:

1. **Customer Management**: Register new customers, update their details, retrieve customer data, and view order history.
2. **Menu Management**: List, add, update, and delete menu items.
3. **Order Management**: Place orders, retrieve order details, and view all orders.
4. **Payment Processing**: Implement different payment methods and provide receipts.

This API should handle:

- **CRUD operations** for each major entity: Customer, MenuItem, Order, and Payment.

- **Exception handling** to return meaningful error responses for bad requests.
- **Data validation** to ensure that requests are valid.
- **Business Logic**: Ensure that an order cannot be placed without at least one menu item, and total cost should be calculated automatically.
- **Custom Endpoints**: Some custom endpoints for specific operations, such as viewing a customer's order history or filtering orders by status.

Part 1: Setting Up the Project

1. **Create a Spring Boot Project**:
    - Use Spring Initializr to create a new Spring Boot project with dependencies:
        - Spring Web
        - Spring Data JPA
        - Spring Boot DevTools
        - H2/ Hyper SQL/ Postgres SQL Database (or another preferred database)
        - Spring Validation

Part 2: API Design

Define the following entities and their relationships.

1. Customer:

- **Attributes**: customerId, name, email, phoneNumber
- **Endpoints**:
    - POST /customers: Add a new customer
    - GET /customers/{id}: Retrieve a customer by ID
    - PUT /customers/{id}: Update customer details
    - DELETE /customers/{id}: Delete a customer
    - GET /customers/{id}/orders: Get order history for a specific customer

2. MenuItem:

- **Attributes**: itemId, name, price, type (e.g., regular or specialty)

- **Endpoints**:
    - POST /menu-items: Add a new menu item
    - GET /menu-items: Retrieve all menu items
    - GET /menu-items/{id}: Retrieve a specific menu item by ID
    - PUT /menu-items/{id}: Update menu item details
    - DELETE /menu-items/{id}: Delete a menu item

3. Order:

- **Attributes**: orderId, customerId, items (list of MenuItem IDs), totalAmount, orderStatus (e.g., pending, completed)
- **Endpoints**:
    - POST /orders: Place a new order (assign to a customer and calculate total)
    - GET /orders/{id}: Retrieve a specific order by ID
    - GET /orders: Retrieve all orders (support filtering by status)
    - PUT /orders/{id}: Update order details (e.g., change status to completed)

4. Payment:

- **Attributes**: paymentId, orderId, amount, paymentType (e.g., credit_card, cash), status (e.g., successful, failed)
- **Endpoints**:
    - POST /payments: Process a payment for an order
    - GET /payments/{id}: Retrieve payment details
    - GET /payments: Retrieve all payments, with optional filtering by status

Part 3: Detailed Instructions

1. **Create Entity Classes**:
    - Define the entities for Customer, MenuItem, Order, and Payment using **JPA annotations** to map them to database tables.
    - Use **appropriate relationships** (e.g., OneToMany for Customer to Order).

2. **Repository Layer**:
    - Create repository interfaces (e.g., CustomerRepository, MenuItemRepository) by extending JpaRepository for CRUD operations.
3. **Service Layer**:
    - Implement service classes for CustomerService, MenuItemService, OrderService, and PaymentService.
    - Add business logic here, such as calculating order totals, validating order items, and managing payments.
    - Ensure thread safety for methods where necessary.
4. **Controller Layer**:
    - Implement REST controllers (CustomerController, MenuItemController, OrderController, PaymentController) with mappings for each endpoint.
    - Use @RequestMapping and @PathVariable to route HTTP requests and bind them to service methods.
    - Use @RequestBody to handle JSON data input and @ResponseEntity to customize HTTP responses.
5. **Exception Handling**:
    - Create custom exceptions, such as CustomerNotFoundException, InvalidOrderException, and PaymentException.
    - Use @ControllerAdvice and @ExceptionHandler to manage exceptions globally and return meaningful error responses.
6. **Data Validation**:
    - Use **Hibernate Validator** annotations (e.g., @NotNull, @Email, @Min, @Max) in entity classes to validate incoming data.
    - Add validation in controllers to check input before processing.
7. **Database Configuration**:
    - Use **H2 database/Hyper SQL/ Postgres SQL** for testing. Set up sample data in data.sql or import.sql.

    - Configure different flavour of DB for persistence and use Spring Boot @Profile feature to connect to multiple DB in when the active Profile/Env is dev, test, prod.

Part 4: Testing the API

1. **Postman Collection**:
    - Create a **Postman** collection with requests for each endpoint and sample JSON payloads for POST and PUT requests.
    - Test each endpoint to ensure functionality.
2. **Unit and Integration Tests**:
    - To demonstrate unit and integration testing only, no need to develop for all scenarios. Choose any class and a method.
    - Write unit tests for service layer methods.
    - Write integration tests to verify that all controllers work correctly and validate response structures.
    - Use **Spring Boot Test** and **Mockito** for dependency mocking.

Assessment Criteria:

1. **Functionality (40%)**
    - Are all required endpoints implemented?
    - Do endpoints handle CRUD operations correctly?
    - Is business logic like order total calculation implemented properly?
2. **Error Handling and Validation (20%)**
    - Are validation annotations used effectively in entities?
    - Are meaningful error messages returned for invalid requests?
    - Are exceptions handled properly with custom messages?
3. **Code Quality (20%)**
    - Is the code well-structured into layers (Controller, Service, Repository)?
    - Are methods modular and easy to understand?
    - Is proper use of annotations and configurations demonstrated?

4. **Testing (20%)**
    - Are unit tests for services written and passing?
    - Are integration tests for controllers implemented?
    - Is the Postman collection complete and functional?

Day 11 and 12

Assignment 2: Web Automation with Selenium

Objective:

Develop a series of Selenium scripts to automate web applications. This assignment will help you practice the fundamental and advanced concepts of Selenium, including locating elements, navigating web pages, handling dynamic elements, and applying the Page Object Model (POM) for maintainable test scripts.

Problem Statement:

You are tasked with automating a demo web application to perform the following tasks:

1. **Login Automation:**
    - Navigate to the login page of a web application.
    - Use sendKeys() to enter username and password.
    - Click the login button using click().
    - Verify the login by asserting the presence of a welcome message or user dashboard.
2. **Web Elements Interaction:**
    - Navigate to a page with a form containing the following elements:
        - Input fields (e.g., name, email).
        - Checkboxes.
        - Dropdowns.
        - Radio buttons.
    - Automate the process of filling out the form using Selenium WebDriver.

    - Submit the form and validate the success message.
3. **Dynamic Web Element Handling:**
    - Interact with dynamically loaded content (e.g., AJAX elements, alerts).
    - Handle a JavaScript alert and print its message to the console.
    - Switch to an iframe, locate an element within it, and interact with it.
4. **Multiple Windows and Tabs:**
    - Automate a scenario where a button opens a new browser window/tab.
    - Switch to the new window, perform an action (e.g., clicking a link), and return to the original window.
5. **Page Object Model Implementation:**
    - Refactor the scripts for the Login Automation and Form Interaction tasks using the Page Object Model (POM).
    - Create page classes to encapsulate locators and reusable methods for each page.
6. **TestNG Framework:**
    - Create a TestNG suite to organize and run the test cases created in the previous steps.
    - Use assertions to validate test outcomes.
    - Group tests (e.g., "LoginTests", "FormTests") and run them selectively.

Detailed Requirements:

Part 1: Basic Setup

1. Set up Selenium WebDriver in your development environment.
    - Use Maven for dependency management.
    - Install and configure a browser driver (e.g., ChromeDriver, EdgeDriver).
2. Configure TestNG for running test cases.

Part 2: Automating Web Tasks

1. **Login Automation:**
    - URL: Use any demo login application (e.g., https://the-internet.herokuapp.com/login).
    - Automate the login process with valid and invalid credentials.
2. **Form Interaction:**
    - URL: Use any sample form page (e.g., https://www.selenium.dev/selenium/web/web-form.html, https://practice.expandtesting.com/inputs, techlistic.com/p/selenium-practice-form.html).
    - Fill out the form with valid data.
    - Automate validations for mandatory fields.
3. **Dynamic Elements:**
    - Handle a pop-up alert by clicking a button to trigger it.
    - Switch to an iframe and locate a specific element inside it.
4. **Multiple Windows:**
    - URL: Use a site with a "New Window" button (e.g., https://the-internet.herokuapp.com/windows).
    - Automate switching between windows and interacting with elements in the new window.

Part 3: Advanced Automation with POM

- Refactor the Login and Form scripts to follow the Page Object Model.
- Create a LoginPage and a FormPage class.
- Use reusable methods for actions like enterUsername, clickLoginButton, and submitForm.

Part 4: TestNG Suite

1. Organize test cases using TestNG.
2. Group tests and execute selected groups.
3. Add assertions to verify test results.
4. Generate a detailed test report after execution.

Assessment Criteria:

1. **Functionality (40%):**
    - Correct implementation of automation scripts for the given tasks.
    - Handling of dynamic elements, alerts, and iFrames.
2. **Code Quality (30%):**
    - Use of meaningful variable names and methods.
    - Proper application of Page Object Model (POM).
3. **TestNG Usage (20%):**
    - Correct organization of tests using TestNG.
    - Use of annotations, assertions, and grouping.
4. **Documentation (10%):**
    - Clear and concise code comments.
    - Proper structuring of the project.