**WEEK 4&5**

**EXERCISE 4**

**Online Bookstore - Processing Request Body and Form Data**

**Business Scenario**

The task is to create endpoints to accept and process JSON request bodies and form data for customer registrations in the online bookstore application.

**Instructions**

**1. Request Body:**

Task: Implement a POST endpoint to create a new customer by accepting a JSON request body.

**Implementation:**

* Use the @RequestBody annotation to map the incoming JSON data to a Customer object.
* The endpoint /customers will be used to create a new customer.

@PostMapping("/customers")

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

Customer savedCustomer = customerService.saveCustomer(customer);

return ResponseEntity.status(HttpStatus.CREATED).body(savedCustomer);

}

**Explanation:**

* The createCustomer method is mapped to the POST /customers URL.
* The @RequestBody annotation automatically deserializes the JSON request body into a Customer object.
* The method then saves the customer using the CustomerService and returns the saved customer along with a 201 Created status.

**2. Form Data:**

**Task:** Implement an endpoint to process form data for customer registrations.

**Implementation:**

* Use the @RequestParam annotation to capture form data fields.
* The endpoint /customers/register will be used to handle form submissions.

**Code Example:**

@PostMapping("/customers/register")

public ResponseEntity<Customer> registerCustomer(

@RequestParam String name,

@RequestParam String email,

@RequestParam String password) {

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPassword(password);

Customer savedCustomer = customerService.saveCustomer(customer);

return ResponseEntity.status(HttpStatus.CREATED).body(savedCustomer);

}

**Explanation:**

* The registerCustomer method is mapped to the POST /customers/register URL.
* The @RequestParam annotation is used to capture form fields name, email, and password from the request.
* These form fields are used to create a new Customer object, which is then saved using the CustomerService.
* The method returns the saved customer with a 201 Created status.

**Example Logic for the Customer Service:**

To support these endpoints, the CustomerService class can include the following logic:

import org.springframework.stereotype.Service;

import java.util.\*;

@Service

public class CustomerService {

private Map<Long, Customer> customerRepository = new HashMap<>();

private Long nextId = 1L;

// Save a new customer

public Customer saveCustomer(Customer customer) {

customer.setId(nextId++);

customerRepository.put(customer.getId(), customer);

return customer;

}

}

**Explanation:**

The saveCustomer method in the CustomerService class simulates saving a customer by assigning a unique ID and storing it in a HashMap.

**Conclusion:**

By implementing these endpoints, The RESTful service can now handle both JSON request bodies and form data, allowing for flexible customer registration processes.