**WEEK 4&5**

**EXERCISE 8**

**Online Bookstore - Implementing CRUD Operations**

**Business Scenario**

The task is to implement Create, Read, Update, and Delete operations for the Book and Customer entities in the online bookstore application.

**Instructions**

**1. CRUD Endpoints:**

**Task:** Implement endpoints for creating, reading, updating, and deleting books and customers.

**Implementation:**

* Create: POST /books and POST /customers
* Read: GET /books/{id} and GET /customers/{id}
* Update: PUT /books/{id} and PUT /customers/{id}
* Delete: DELETE /books/{id} and DELETE /customers/{id}

**Code Example:**

**BookController.java:**

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

import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@PostMapping

public ResponseEntity<BookDTO> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookService.createBook(bookDTO);

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

}

@GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(@PathVariable Long id) {

BookDTO bookDTO = bookService.getBookById(id);

return bookDTO != null ? ResponseEntity.ok(bookDTO) : ResponseEntity.notFound().build();

}

@PutMapping("/{id}")

public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @Valid @RequestBody BookDTO bookDTO) {

BookDTO updatedBook = bookService.updateBook(id, bookDTO);

return updatedBook != null ? ResponseEntity.ok(updatedBook) : ResponseEntity.notFound().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

boolean isDeleted = bookService.deleteBook(id);

return isDeleted ? ResponseEntity.noContent().build() : ResponseEntity.notFound().build();

}

}

**CustomerController.java:**

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

import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid;

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping

public ResponseEntity<CustomerDTO> createCustomer(@Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = customerService.createCustomer(customerDTO);

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

}

@GetMapping("/{id}")

public ResponseEntity<CustomerDTO> getCustomerById(@PathVariable Long id) {

CustomerDTO customerDTO = customerService.getCustomerById(id);

return customerDTO != null ? ResponseEntity.ok(customerDTO) : ResponseEntity.notFound().build();

}

@PutMapping("/{id}")

public ResponseEntity<CustomerDTO> updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO customerDTO) {

CustomerDTO updatedCustomer = customerService.updateCustomer(id, customerDTO);

return updatedCustomer != null ? ResponseEntity.ok(updatedCustomer) : ResponseEntity.notFound().build();

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) {

boolean isDeleted = customerService.deleteCustomer(id);

return isDeleted ? ResponseEntity.noContent().build() : ResponseEntity.notFound().build();

}

}

**2. Validating Input Data:**

**Task:** Use validation annotations like @NotNull, @Size, and @Min to validate input data.

**Implementation:**

* Use annotations to ensure that the data provided by the client meets the required constraints.

**Code Example:**

**BookDTO.java:**

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

import javax.validation.constraints.Min;

public class BookDTO {

private Long id;

@NotNull(message = "Title cannot be null")

@Size(min = 1, max = 100, message = "Title must be between 1 and 100 characters")

private String title;

@NotNull(message = "Author cannot be null")

@Size(min = 1, max = 50, message = "Author must be between 1 and 50 characters")

private String author;

@Min(value = 0, message = "Price must be positive")

private double price;

}

**CustomerDTO.java:**

import javax.validation.constraints.Email;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class CustomerDTO {

private Long id;

@NotNull(message = "Name cannot be null")

@Size(min = 1, max = 100, message = "Name must be between 1 and 100 characters")

private String name;

@NotNull(message = "Email cannot be null")

@Email(message = "Email should be valid")

private String email;

}

**3. Optimistic Locking:**

**Task:** Implement optimistic locking for concurrent updates using JPA versioning.

**Implementation:**

* Use a @Version field in your entity classes to handle concurrent updates.

**Code Example:**

**Book.java:**

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Version;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private double price;

@Version

private Integer version;

}

**Customer.java:**

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Version;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@Version

private Integer version;

}

**Explanation:**

* The @Version annotation is used to mark the version field, which JPA uses to detect concurrent modifications.
* When an update occurs, JPA checks the version to ensure no other updates have been made to the entity since it was last read.

**Conclusion:**

By implementing these CRUD operations, validating input data, and using optimistic locking, we can ensure robust and reliable management of your Book and Customer entities.