**Exercise 8: Online Bookstore - Implementing CRUD Operations**

1. **CRUD Endpoints for Book and Customer**

Book Entity CRUD Endpoints

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

// Create a new book

@PostMapping

public ResponseEntity<Book> createBook(@Valid @RequestBody Book book) {

Book createdBook = bookService.saveBook(book);

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

}

// Get a book by ID

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

// Update an existing book

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @Valid @RequestBody Book bookDetails) {

Book updatedBook = bookService.updateBook(id, bookDetails);

return ResponseEntity.ok(updatedBook);

}

// Delete a book

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

Customer Entity CRUD Endpoints

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

// Create a new customer

@PostMapping

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

Customer createdCustomer = customerService.saveCustomer(customer);

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

}

// Get a customer by ID

@GetMapping("/{id}")

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

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

// Update an existing customer

@PutMapping("/{id}")

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @Valid @RequestBody Customer customerDetails) {

Customer updatedCustomer = customerService.updateCustomer(id, customerDetails);

return ResponseEntity.ok(updatedCustomer);

}

// Delete a customer

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

1. **Validating Input Data**

**Book Entity Validation**

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Title is required")

@Size(min = 2, message = "Title must be at least 2 characters")

private String title;

@NotNull(message = "Author is required")

@Size(min = 2, message = "Author must be at least 2 characters")

private String author;

@Min(value = 1, message = "Price must be greater than 0")

private double price;

// Other fields and getters/setters

}

**Customer Entity Validation**

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Name is required")

@Size(min = 2, message = "Name must be at least 2 characters")

private String name;

@NotNull(message = "Email is required")

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

private String email;

// Other fields and getters/setters

}

1. **Optimistic Locking with JPA Versioning**

Book Entity with Versioning

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Title is required")

@Size(min = 2, message = "Title must be at least 2 characters")

private String title;

@NotNull(message = "Author is required")

@Size(min = 2, message = "Author must be at least 2 characters")

private String author;

@Min(value = 1, message = "Price must be greater than 0")

private double price;

@Version

private Integer version;

// Other fields and getters/setters

}

Customer Entity with Versioning

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Name is required")

@Size(min = 2, message = "Name must be at least 2 characters")

private String name;

@NotNull(message = "Email is required")

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

private String email;

@Version

private Integer version;

// Other fields and getters/setters

}

Service Layer Example

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public Book saveBook(Book book) {

return bookRepository.save(book);

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

.orElseThrow(() -> new ResourceNotFoundException("Book not found"));

}

public Book updateBook(Long id, Book bookDetails) {

Book book = getBookById(id);

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setPrice(bookDetails.getPrice());

return bookRepository.save(book);

}

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

}