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

**Business Scenario:**

Implement Create, Read, Update, and Delete operations for the Book and Customer entities.

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String title;

@NotNull

@Size(min = 2, max = 100)

private String author;

@Min(0)

private Double price;

@Version

private Long version;

// Getters and Setters

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String name;

@NotNull

@Size(min = 5, max = 100)

private String email;

@Version

private Long version;

// Getters and Setters

}

Creating Repositories:

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

Creating Service Layer:

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public List<Book> findAll() {

return bookRepository.findAll();

}

public Optional<Book> findById(Long id) {

return bookRepository.findById(id);

}

public Book save(Book book) {

return bookRepository.save(book);

}

public void deleteById(Long id) {

bookRepository.deleteById(id);

}

@Transactional

public Book update(Book book) {

return bookRepository.save(book);

}

}

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public List<Customer> findAll() {

return customerRepository.findAll();

}

public Optional<Customer> findById(Long id) {

return customerRepository.findById(id);

}

public Customer save(Customer customer) {

return customerRepository.save(customer);

}

public void deleteById(Long id) {

customerRepository.deleteById(id);

}

@Transactional

public Customer update(Customer customer) {

return customerRepository.save(customer);

}

}

Creating Controller Layer

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

import org.springframework.http.ResponseEntity;

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

import jakarta.validation.Valid;

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public List<Book> getAllBooks() {

return bookService.findAll();

}

@GetMapping("/{id}")

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

return bookService.findById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

@PostMapping

public Book createBook(@Valid @RequestBody Book book) {

return bookService.save(book);

}

@PutMapping("/{id}")

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

return bookService.findById(id).map(book -> {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setPrice(bookDetails.getPrice());

return ResponseEntity.ok(bookService.update(book));

}).orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return bookService.findById(id).map(book -> {

bookService.deleteById(id);

return ResponseEntity.ok().build();

}).orElse(ResponseEntity.notFound().build());

}

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping

public List<Customer> getAllCustomers() {

return customerService.findAll();

}

@GetMapping("/{id}")

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

return customerService.findById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

@PostMapping

public Customer createCustomer(@Valid @RequestBody Customer customer) {

return customerService.save(customer);

}

@PutMapping("/{id}")

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

return customerService.findById(id).map(customer -> {

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return ResponseEntity.ok(customerService.update(customer));

}).orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return customerService.findById(id).map(customer -> {

customerService.deleteById(id);

return ResponseEntity.ok().build();

}).orElse(ResponseEntity.notFound().build());

}

}

Optimistic locking is already implemented via the @Version annotation in both the Book and Customer entities. When multiple users try to update the same record, JPA will ensure that the operation fails if the version field has changed, thus preventing lost updates.

If an optimistic locking conflict occurs, JPA will throw an OptimisticLockException, which can be handled as needed.

**Exercise 9: Online Bookstore - Understanding HATEOAS**

**Business Scenario:**

Enhance your REST API to follow HATEOAS principles for navigation through resources.

Add Spring HATEOAS Dependency:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-hateoas</artifactId>

</dependency>

Modify the Book and Customer Controllers:

**Book Controller with HATEOAS:**

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.stream.Collectors;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public CollectionModel<EntityModel<Book>> getAllBooks() {

List<EntityModel<Book>> books = bookService.findAll().stream()

.map(book -> EntityModel.of(book,

linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel(),

linkTo(methodOn(BookController.class).getAllBooks()).withRel("books")))

.collect(Collectors.toList());

return CollectionModel.of(books, linkTo(methodOn(BookController.class).getAllBooks()).withSelfRel());

}

@GetMapping("/{id}")

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

return bookService.findById(id)

.map(book -> EntityModel.of(book,

linkTo(methodOn(BookController.class).getBookById(id)).withSelfRel(),

linkTo(methodOn(BookController.class).getAllBooks()).withRel("books")))

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

@PostMapping

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

Book createdBook = bookService.save(book);

return ResponseEntity.created(linkTo(methodOn(BookController.class).getBookById(createdBook.getId())).toUri())

.body(EntityModel.of(createdBook,

linkTo(methodOn(BookController.class).getBookById(createdBook.getId())).withSelfRel(),

linkTo(methodOn(BookController.class).getAllBooks()).withRel("books")));

}

@PutMapping("/{id}")

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

return bookService.findById(id)

.map(book -> {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setPrice(bookDetails.getPrice());

Book updatedBook = bookService.update(book);

return ResponseEntity.ok(EntityModel.of(updatedBook,

linkTo(methodOn(BookController.class).getBookById(id)).withSelfRel(),

linkTo(methodOn(BookController.class).getAllBooks()).withRel("books")));

})

.orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return bookService.findById(id).map(book -> {

bookService.deleteById(id);

return ResponseEntity.noContent().build();

}).orElse(ResponseEntity.notFound().build());

}

}

**Customer Controller with HATEOAS**

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.CollectionModel;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.stream.Collectors;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping

public CollectionModel<EntityModel<Customer>> getAllCustomers() {

List<EntityModel<Customer>> customers = customerService.findAll().stream()

.map(customer -> EntityModel.of(customer,

linkTo(methodOn(CustomerController.class).getCustomerById(customer.getId())).withSelfRel(),

linkTo(methodOn(CustomerController.class).getAllCustomers()).withRel("customers")))

.collect(Collectors.toList());

return CollectionModel.of(customers, linkTo(methodOn(CustomerController.class).getAllCustomers()).withSelfRel());

}

@GetMapping("/{id}")

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

return customerService.findById(id)

.map(customer -> EntityModel.of(customer,

linkTo(methodOn(CustomerController.class).getCustomerById(id)).withSelfRel(),

linkTo(methodOn(CustomerController.class).getAllCustomers()).withRel("customers")))

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

@PostMapping

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

Customer createdCustomer = customerService.save(customer);

return ResponseEntity.created(linkTo(methodOn(CustomerController.class).getCustomerById(createdCustomer.getId())).toUri())

.body(EntityModel.of(createdCustomer,

linkTo(methodOn(CustomerController.class).getCustomerById(createdCustomer.getId())).withSelfRel(),

linkTo(methodOn(CustomerController.class).getAllCustomers()).withRel("customers")));

}

@PutMapping("/{id}")

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

return customerService.findById(id)

.map(customer -> {

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

Customer updatedCustomer = customerService.update(customer);

return ResponseEntity.ok(EntityModel.of(updatedCustomer,

linkTo(methodOn(CustomerController.class).getCustomerById(id)).withSelfRel(),

linkTo(methodOn(CustomerController.class).getAllCustomers()).withRel("customers")));

})

.orElse(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

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

return customerService.findById(id).map(customer -> {

customerService.deleteById(id);

return ResponseEntity.noContent().build();

}).orElse(ResponseEntity.notFound().build());

}

}

Testing HATEOAS Links:

{

"id": 1,

"title": "Book Title",

"author": "Author Name",

"price": 29.99,

"\_links": {

"self": {

"href": "http://localhost:8080/api/books/1"

},

"books": {

"href": "http://localhost:8080/api/books"

}

}

}

**Exercise 10: Online Bookstore - Configuring Content Negotiation**

**Business Scenario:**

Support different media types (JSON, XML) for your bookstore's RESTful services.

Maven Dependencies

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

Testing Content Negotiation:

JSON:

curl -H "Accept: application/json" <http://localhost:8080/api/books>

XML:

curl -H "Accept: application/xml" <http://localhost:8080/api/books>

When consuming the API, the client specifies the desired media type in the Accept header. The server will respond with the appropriate format (JSON or XML) based on the client's request.

**Exercise 11: Online Bookstore - Integrating Spring Boot Actuator**

**Business Scenario:**

Monitor and manage your bookstore's RESTful services using Spring Boot Actuator.

Add Spring Boot Actuator Dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

Configuring Actuator Endpoints in application.properties:

# Expose all Actuator endpoints

management.endpoints.web.exposure.include=\*

# Customize the base path for Actuator endpoints

management.endpoints.web.base-path=/actuator

# Enable or disable specific endpoints

management.endpoint.health.show-details=always

management.endpoint.metrics.enabled=true

Exposing Custom Metrics:

import io.micrometer.core.instrument.MeterRegistry;

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

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

@Component

public class CustomMetrics {

@Autowired

private MeterRegistry meterRegistry;

@Autowired

private BookRepository bookRepository;

@PostConstruct

public void registerMetrics() {

meterRegistry.gauge("bookstore.books.count", bookRepository.count());

}

}

Securing Actuator Endpoints:

# Restrict access to Actuator endpoints

management.endpoints.web.exposure.include=health,info

# Set up basic authentication for Actuator endpoints

management.endpoint.health.roles=ADMIN

management.endpoint.metrics.roles=ADMIN