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

**Business Scenario:**

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

**CODE**:

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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

import java.util.List;

// Book Entity

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 100)

private String title;

@NotNull

@Size(min = 1, max = 100)

private String author;

@Min(0)

private double price;

@Version

private int version; // For Optimistic Locking

// Getters and Setters

// ...

}

// Customer Entity

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 50)

private String name;

@NotNull

@Size(min = 5, max = 100)

private String email;

@Version

private int version; // For Optimistic Locking

// Getters and Setters

// ...

}

// Book Repository

public interface BookRepository extends JpaRepository<Book, Long> {}

// Customer Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {}

// Book Service

@Service

public class BookService {

private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) {

return bookRepository.save(book);

}

@Transactional

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);

}

@Transactional

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

}

// Customer Service (similar to BookService)

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

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

}

@Transactional

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) {

Customer customer = getCustomerById(id);

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) {

Customer customer = getCustomerById(id);

customerRepository.delete(customer);

}

}

// Book Controller

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) {

this.bookService = bookService;

}

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

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

return ResponseEntity.ok(bookService.getBookById(id));

}

@PostMapping

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

return new ResponseEntity<>(bookService.createBook(book), HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

return ResponseEntity.ok(bookService.updateBook(id, bookDetails));

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

// Customer Controller (similar to BookController)

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}

@GetMapping

public List<Customer> getAllCustomers() {

return customerService.getAllCustomers();

}

@GetMapping("/{id}")

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

return ResponseEntity.ok(customerService.getCustomerById(id));

}

@PostMapping

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

return new ResponseEntity<>(customerService.createCustomer(customer), HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

return ResponseEntity.ok(customerService.updateCustomer(id, customerDetails));

}

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

// Exception Handling

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

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

**Business Scenario:**

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

**CODE:**

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

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

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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

import java.util.List;

import java.util.stream.Collectors;

// Book Entity

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 100)

private String title;

@NotNull

@Size(min = 1, max = 100)

private String author;

@Min(0)

private double price;

@Version

private int version;

// Getters and Setters

// ...

}

// Customer Entity

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 50)

private String name;

@NotNull

@Size(min = 5, max = 100)

private String email;

@Version

private int version;

// Getters and Setters

// ...

}

// Book Repository

public interface BookRepository extends JpaRepository<Book, Long> {}

// Customer Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {}

// Book Service

@Service

public class BookService {

private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) {

return bookRepository.save(book);

}

@Transactional

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);

}

@Transactional

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

}

// Customer Service (similar to BookService)

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

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

}

@Transactional

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) {

Customer customer = getCustomerById(id);

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) {

Customer customer = getCustomerById(id);

customerRepository.delete(customer);

}

}

// HATEOAS Model Assembler

@Component

public class ModelAssembler {

public Link createBookLink(Long id) {

return WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).getBookById(id)).withSelfRel();

}

public Link createCustomerLink(Long id) {

return WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerController.class).getCustomerById(id)).withSelfRel();

}

}

// Book Controller

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

private final ModelAssembler modelAssembler;

public BookController(BookService bookService, ModelAssembler modelAssembler) {

this.bookService = bookService;

this.modelAssembler = modelAssembler;

}

@GetMapping

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

return bookService.getAllBooks().stream()

.map(book -> EntityModel.of(book, modelAssembler.createBookLink(book.getId())))

.collect(Collectors.toList());

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

EntityModel<Book> resource = EntityModel.of(book, modelAssembler.createBookLink(id));

return ResponseEntity.ok(resource);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

EntityModel<Book> resource = EntityModel.of(createdBook, modelAssembler.createBookLink(createdBook.getId()));

return new ResponseEntity<>(resource, HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

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

EntityModel<Book> resource = EntityModel.of(updatedBook, modelAssembler.createBookLink(id));

return ResponseEntity.ok(resource);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

// Customer Controller (similar to BookController)

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

private final ModelAssembler modelAssembler;

public CustomerController(CustomerService customerService, ModelAssembler modelAssembler) {

this.customerService = customerService;

this.modelAssembler = modelAssembler;

}

@GetMapping

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

return customerService.getAllCustomers().stream()

.map(customer -> EntityModel.of(customer, modelAssembler.createCustomerLink(customer.getId())))

.collect(Collectors.toList());

}

@GetMapping("/{id}")

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

Customer customer = customerService.getCustomerById(id);

EntityModel<Customer> resource = EntityModel.of(customer, modelAssembler.createCustomerLink(id));

return ResponseEntity.ok(resource);

}

@PostMapping

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

Customer createdCustomer = customerService.createCustomer(customer);

EntityModel<Customer> resource = EntityModel.of(createdCustomer, modelAssembler.createCustomerLink(createdCustomer.getId()));

return new ResponseEntity<>(resource, HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

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

EntityModel<Customer> resource = EntityModel.of(updatedCustomer, modelAssembler.createCustomerLink(id));

return ResponseEntity.ok(resource);

}

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

// Exception Handling

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

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

**Business Scenario:**

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

**CODE:**

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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

import org.springframework.web.servlet.config.annotation.ContentNegotiationConfigurer;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageConverter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.util.List;

// Entity Classes

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 100)

private String title;

@NotNull

@Size(min = 1, max = 100)

private String author;

@Min(0)

private double price;

@Version

private int version;

// Getters and Setters

// ...

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 50)

private String name;

@NotNull

@Size(min = 5, max = 100)

private String email;

@Version

private int version;

// Getters and Setters

// ...

}

// Repository Interfaces

public interface BookRepository extends JpaRepository<Book, Long> {}

public interface CustomerRepository extends JpaRepository<Customer, Long> {}

// Service Layer

@Service

public class BookService {

private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) {

return bookRepository.save(book);

}

@Transactional

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);

}

@Transactional

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

}

// Customer Service (similar to BookService)

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

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

}

@Transactional

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) {

Customer customer = getCustomerById(id);

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) {

Customer customer = getCustomerById(id);

customerRepository.delete(customer);

}

}

// Controller Layer

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) {

this.bookService = bookService;

}

@GetMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

@PostMapping(consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Book createdBook = bookService.createBook(book);

return new ResponseEntity<>(createdBook, HttpStatus.CREATED);

}

@PutMapping(value = "/{id}", consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

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

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

// Customer Controller (similar to BookController)

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}

@GetMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public List<Customer> getAllCustomers() {

return customerService.getAllCustomers();

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

@PostMapping(consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

Customer createdCustomer = customerService.createCustomer(customer);

return new ResponseEntity<>(createdCustomer, HttpStatus.CREATED);

}

@PutMapping(value = "/{id}", consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

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

return ResponseEntity.ok(updatedCustomer);

}

@DeleteMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

// Content Negotiation Configuration

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorPathExtension(false)

.favorParameter(false)

.ignoreAcceptHeader(false)

.defaultContentType(MediaType.APPLICATION\_JSON)

.mediaType("json", MediaType.APPLICATION\_JSON)

.mediaType("xml", MediaType.APPLICATION\_XML);

}

@Bean

public MappingJackson2XmlHttpMessageConverter xmlConverter() {

return new MappingJackson2XmlHttpMessageConverter();

}

}

// Exception Handling

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

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

**Business Scenario:**

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

**CODE:**<!-- pom.xml -->

<dependencies>

<!-- Other dependencies -->

<dependency>

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

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

</dependency>

</dependencies>

import io.micrometer.core.instrument.Counter;

import io.micrometer.core.instrument.MeterRegistry;

import jakarta.persistence.\*;

import jakarta.validation.constraints.\*;

import org.springframework.boot.actuate.endpoint.web.annotation.RestControllerEndpoint;

import org.springframework.boot.actuate.health.Health;

import org.springframework.boot.actuate.health.HealthIndicator;

import org.springframework.boot.actuate.info.Info;

import org.springframework.boot.actuate.info.InfoContributor;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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

import java.util.HashMap;

import java.util.List;

import java.util.Map;

// Entity Classes

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 100)

private String title;

@NotNull

@Size(min = 1, max = 100)

private String author;

@Min(0)

private double price;

@Version

private int version;

// Getters and Setters

// ...

}

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 1, max = 50)

private String name;

@NotNull

@Size(min = 5, max = 100)

private String email;

@Version

private int version;

// Getters and Setters

// ...

}

// Repository Interfaces

public interface BookRepository extends JpaRepository<Book, Long> {}

public interface CustomerRepository extends JpaRepository<Customer, Long> {}

// Service Layer

@Service

public class BookService {

private final BookRepository bookRepository;

private final Counter bookCreationCounter;

public BookService(BookRepository bookRepository, MeterRegistry meterRegistry) {

this.bookRepository = bookRepository;

this.bookCreationCounter = meterRegistry.counter("book.creation.count");

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id)

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

}

@Transactional

public Book createBook(Book book) {

bookCreationCounter.increment();

return bookRepository.save(book);

}

@Transactional

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);

}

@Transactional

public void deleteBook(Long id) {

Book book = getBookById(id);

bookRepository.delete(book);

}

}

// Customer Service (similar to BookService)

@Service

public class CustomerService {

private final CustomerRepository customerRepository;

private final Counter customerCreationCounter;

public CustomerService(CustomerRepository customerRepository, MeterRegistry meterRegistry) {

this.customerRepository = customerRepository;

this.customerCreationCounter = meterRegistry.counter("customer.creation.count");

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Customer getCustomerById(Long id) {

return customerRepository.findById(id)

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

}

@Transactional

public Customer createCustomer(Customer customer) {

customerCreationCounter.increment();

return customerRepository.save(customer);

}

@Transactional

public Customer updateCustomer(Long id, Customer customerDetails) {

Customer customer = getCustomerById(id);

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

}

@Transactional

public void deleteCustomer(Long id) {

Customer customer = getCustomerById(id);

customerRepository.delete(customer);

}

}

// Custom Health Indicator

@Component

public class BookstoreHealthIndicator implements HealthIndicator {

@Override

public Health health() {

// Perform custom health checks

return Health.up().withDetail("status", "Everything is OK!").build();

}

}

// Custom Info Contributor

@Component

public class BookstoreInfoContributor implements InfoContributor {

@Override

public void contribute(Info.Builder builder) {

Map<String, Object> bookstoreDetails = new HashMap<>();

bookstoreDetails.put("application", "Online Bookstore");

bookstoreDetails.put("version", "1.0.0");

builder.withDetail("bookstore-info", bookstoreDetails);

}

}

// Custom Actuator Endpoint

@RestControllerEndpoint(id = "custom-endpoint")

public class CustomActuatorEndpoint {

@GetMapping("/status")

public ResponseEntity<String> getStatus() {

return ResponseEntity.ok("Custom Actuator Endpoint is working!");

}

}

// Actuator Configuration

@Configuration

public class ActuatorConfig {

@Bean

public MeterRegistryCustomizer<MeterRegistry> configureMetrics() {

return registry -> registry.config().commonTags("application", "Online Bookstore");

}

}

// Controller Layer

@RestController

@RequestMapping("/api/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) {

this.bookService = bookService;

}

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

return new ResponseEntity<>(createdBook, HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

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

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

// Customer Controller (similar to BookController)

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}

@GetMapping

public List<Customer> getAllCustomers() {

return customerService.getAllCustomers();

}

@GetMapping("/{id}")

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

Customer customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

@PostMapping

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

Customer createdCustomer = customerService.createCustomer(customer);

return new ResponseEntity<>(createdCustomer, HttpStatus.CREATED);

}

@PutMapping("/{id}")

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

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

return ResponseEntity.ok(updatedCustomer);

}

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

// Exception Handling

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

**Exercise 12: Online Bookstore - Securing RESTful Endpoints with Spring Security**

**Business Scenario:**

**Secure your bookstore's RESTful endpoints using Spring Security with JWT-based authentication.**

**CODE:**<!-- pom.xml -->

<dependencies>

<!-- Other dependencies -->

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId> <!-- or jjwt-gson, jjwt-orgjson, etc -->

<version>0.11.5</version>

</dependency>

</dependencies>

// Security Configuration

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.stereotype.Service;

import org.springframework.web.filter.OncePerRequestFilter;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import java.io.IOException;

import java.security.Key;

import java.util.Date;

import java.util.List;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebMvcConfigurer {

private final JwtTokenProvider jwtTokenProvider;

private final CustomUserDetailsService customUserDetailsService;

public SecurityConfig(JwtTokenProvider jwtTokenProvider, CustomUserDetailsService customUserDetailsService) {

this.jwtTokenProvider = jwtTokenProvider;

this.customUserDetailsService = customUserDetailsService;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.cors().and()

.authorizeRequests()

.antMatchers("/api/auth/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.addFilterBefore(new JwtAuthenticationFilter(jwtTokenProvider, customUserDetailsService), UsernamePasswordAuthenticationFilter.class);

}

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("http://allowed-origin.com")

.allowedMethods("GET", "POST", "PUT", "DELETE")

.allowedHeaders("\*")

.allowCredentials(true);

}

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

// JWT Token Provider

@Service

public class JwtTokenProvider {

private final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

private final long validityInMilliseconds = 3600000; // 1h

public String createToken(String username, List<String> roles) {

Claims claims = Jwts.claims().setSubject(username);

claims.put("roles", roles);

Date now = new Date();

Date validity = new Date(now.getTime() + validityInMilliseconds);

return Jwts.builder()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(validity)

.signWith(key)

.compact();

}

public boolean validateToken(String token) {

try {

Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

public String getUsername(String token) {

return Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token).getBody().getSubject();

}

}

// JWT Authentication Filter

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtTokenProvider jwtTokenProvider;

private final CustomUserDetailsService customUserDetailsService;

public JwtAuthenticationFilter(JwtTokenProvider jwtTokenProvider, CustomUserDetailsService customUserDetailsService) {

this.jwtTokenProvider = jwtTokenProvider;

this.customUserDetailsService = customUserDetailsService;

}

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

String token = resolveToken(request);

if (token != null && jwtTokenProvider.validateToken(token)) {

String username = jwtTokenProvider.getUsername(token);

UserDetails userDetails = customUserDetailsService.loadUserByUsername(username);

UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());

auth.setDetails(userDetails);

}

filterChain.doFilter(request, response);

}

private String resolveToken(HttpServletRequest request) {

String bearerToken = request.getHeader("Authorization");

if (bearerToken != null && bearerToken.startsWith("Bearer ")) {

return bearerToken.substring(7);

}

return null;

}

}

// Custom User Details Service

@Service

public class CustomUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

// For simplicity, using hardcoded users. In a real application, fetch from DB.

if ("admin".equals(username)) {

return User.withUsername(username)

.password(passwordEncoder().encode("admin123"))

.roles("ADMIN")

.build();

} else if ("user".equals(username)) {

return User.withUsername(username)

.password(passwordEncoder().encode("user123"))

.roles("USER")

.build();

} else {

throw new UsernameNotFoundException("User not found");

}

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

// Authentication Controller

@RestController

@RequestMapping("/api/auth")

public class AuthController {

private final JwtTokenProvider jwtTokenProvider;

public AuthController(JwtTokenProvider jwtTokenProvider) {

this.jwtTokenProvider = jwtTokenProvider;

}

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody AuthRequest authRequest) {

// Authentication logic can be improved by using AuthenticationManager

if ("admin".equals(authRequest.getUsername()) && "admin123".equals(authRequest.getPassword())) {

String token = jwtTokenProvider.createToken(authRequest.getUsername(), List.of("ROLE\_ADMIN"));

return ResponseEntity.ok(new AuthResponse(token));

} else if ("user".equals(authRequest.getUsername()) && "user123".equals(authRequest.getPassword())) {

String token = jwtTokenProvider.createToken(authRequest.getUsername(), List.of("ROLE\_USER"));

return ResponseEntity.ok(new AuthResponse(token));

} else {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

}

}

}

// Authentication Request & Response DTOs

public class AuthRequest {

private String username;

private String password;

// Getters and Setters

}

public class AuthResponse {

private String token;

public AuthResponse(String token) {

this.token = token;

}

// Getters

}

// Controllers for Book and Customer (Same as before with security applied)

@RestController

@RequestMapping("/api/books")

public class BookController {

// Existing code

}

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

// Existing code

}

**Exercise 13: Online Bookstore - Unit Testing REST Controllers**

**Business Scenario:**

**Write unit tests for your bookstore's REST controllers using JUnit and Mockito.**

**CODE:**

<!-- pom.xml -->

<dependencies>

<!-- Other dependencies -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

// BookControllerTest.java

import com.example.bookstore.controller.BookController;

import com.example.bookstore.model.Book;

import com.example.bookstore.service.BookService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.util.Arrays;

import java.util.List;

import static org.mockito.Mockito.\*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(BookController.class)

@ExtendWith(MockitoExtension.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@Mock

private BookService bookService;

@InjectMocks

private BookController bookController;

private Book book1;

private Book book2;

@BeforeEach

public void setUp() {

mockMvc = MockMvcBuilders.standaloneSetup(bookController).build();

book1 = new Book();

book1.setId(1L);

book1.setTitle("Book One");

book1.setAuthor("Author One");

book1.setPrice(10.0);

book2 = new Book();

book2.setId(2L);

book2.setTitle("Book Two");

book2.setAuthor("Author Two");

book2.setPrice(15.0);

}

@Test

public void testGetAllBooks() throws Exception {

List<Book> books = Arrays.asList(book1, book2);

when(bookService.getAllBooks()).thenReturn(books);

mockMvc.perform(get("/api/books")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(2))

.andExpect(jsonPath("$[0].title").value("Book One"))

.andExpect(jsonPath("$[1].title").value("Book Two"));

verify(bookService, times(1)).getAllBooks();

}

@Test

public void testGetBookById() throws Exception {

when(bookService.getBookById(1L)).thenReturn(book1);

mockMvc.perform(get("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Book One"));

verify(bookService, times(1)).getBookById(1L);

}

@Test

public void testCreateBook() throws Exception {

when(bookService.createBook(any(Book.class))).thenReturn(book1);

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"Book One\", \"author\": \"Author One\", \"price\": 10.0}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Book One"));

verify(bookService, times(1)).createBook(any(Book.class));

}

@Test

public void testUpdateBook() throws Exception {

when(bookService.updateBook(eq(1L), any(Book.class))).thenReturn(book1);

mockMvc.perform(put("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"Updated Book\", \"author\": \"Updated Author\", \"price\": 20.0}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Book One")); // since it mocks the original

verify(bookService, times(1)).updateBook(eq(1L), any(Book.class));

}

@Test

public void testDeleteBook() throws Exception {

doNothing().when(bookService).deleteBook(1L);

mockMvc.perform(delete("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNoContent());

verify(bookService, times(1)).deleteBook(1L);

}

}

// CustomerControllerTest.java

import com.example.bookstore.controller.CustomerController;

import com.example.bookstore.model.Customer;

import com.example.bookstore.service.CustomerService;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.util.Arrays;

import java.util.List;

import static org.mockito.Mockito.\*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(CustomerController.class)

@ExtendWith(MockitoExtension.class)

public class CustomerControllerTest {

@Autowired

private MockMvc mockMvc;

@Mock

private CustomerService customerService;

@InjectMocks

private CustomerController customerController;

private Customer customer1;

private Customer customer2;

@BeforeEach

public void setUp() {

mockMvc = MockMvcBuilders.standaloneSetup(customerController).build();

customer1 = new Customer();

customer1.setId(1L);

customer1.setName("Customer One");

customer1.setEmail("customer1@example.com");

customer2 = new Customer();

customer2.setId(2L);

customer2.setName("Customer Two");

customer2.setEmail("customer2@example.com");

}

@Test

public void testGetAllCustomers() throws Exception {

List<Customer> customers = Arrays.asList(customer1, customer2);

when(customerService.getAllCustomers()).thenReturn(customers);

mockMvc.perform(get("/api/customers")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(2))

.andExpect(jsonPath("$[0].name").value("Customer One"))

.andExpect(jsonPath("$[1].name").value("Customer Two"));

verify(customerService, times(1)).getAllCustomers();

}

@Test

public void testGetCustomerById() throws Exception {

when(customerService.getCustomerById(1L)).thenReturn(customer1);

mockMvc.perform(get("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Customer One"));

verify(customerService, times(1)).getCustomerById(1L);

}

@Test

public void testCreateCustomer() throws Exception {

when(customerService.createCustomer(any(Customer.class))).thenReturn(customer1);

mockMvc.perform(post("/api/customers")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Customer One\", \"email\": \"customer1@example.com\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.name").value("Customer One"));

verify(customerService, times(1)).createCustomer(any(Customer.class));

}

@Test

public void testUpdateCustomer() throws Exception {

when(customerService.updateCustomer(eq(1L), any(Customer.class))).thenReturn(customer1);

mockMvc.perform(put("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Updated Customer\", \"email\": \"updated@example.com\"}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Customer One")); // since it mocks the original

verify(customerService, times(1)).updateCustomer(eq(1L), any(Customer.class));

}

@Test

public void testDeleteCustomer() throws Exception {

doNothing().when(customerService).deleteCustomer(1L);

mockMvc.perform(delete("/api/customers/1")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNoContent());

verify(customerService, times(1)).deleteCustomer(1L);

}

}

**Exercise 14: Online Bookstore - Integration Testing for REST Services**

**Business Scenario:**

**Write integration tests for your bookstore's RESTful services.**

**CODE:**

<!-- pom.xml -->

<dependencies>

<!-- Other dependencies -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

// IntegrationTests.java

import com.example.bookstore.BookstoreApplication;

import com.example.bookstore.model.Book;

import com.example.bookstore.model.Customer;

import com.example.bookstore.repository.BookRepository;

import com.example.bookstore.repository.CustomerRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit.jupiter.SpringExtension;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest(classes = BookstoreApplication.class)

@ExtendWith(SpringExtension.class)

@AutoConfigureMockMvc

@Transactional

public class IntegrationTests {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@Autowired

private CustomerRepository customerRepository;

private Book book1;

private Customer customer1;

@BeforeEach

public void setUp() {

// Setup for Book

book1 = new Book();

book1.setTitle("Integration Test Book");

book1.setAuthor("Test Author");

book1.setPrice(29.99);

bookRepository.save(book1);

// Setup for Customer

customer1 = new Customer();

customer1.setName("Integration Test Customer");

customer1.setEmail("testcustomer@example.com");

customerRepository.save(customer1);

}

// BookController Tests

@Test

public void testGetAllBooks() throws Exception {

mockMvc.perform(get("/api/books")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(1))

.andExpect(jsonPath("$[0].title").value("Integration Test Book"));

}

@Test

public void testGetBookById() throws Exception {

mockMvc.perform(get("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Integration Test Book"));

}

@Test

public void testCreateBook() throws Exception {

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"New Book\", \"author\": \"New Author\", \"price\": 19.99}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("New Book"));

}

@Test

public void testUpdateBook() throws Exception {

mockMvc.perform(put("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"Updated Book\", \"author\": \"Updated Author\", \"price\": 39.99}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Updated Book"));

}

@Test

public void testDeleteBook() throws Exception {

mockMvc.perform(delete("/api/books/" + book1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNoContent());

Optional<Book> deletedBook = bookRepository.findById(book1.getId());

assert(deletedBook.isEmpty());

}

// CustomerController Tests

@Test

public void testGetAllCustomers() throws Exception {

mockMvc.perform(get("/api/customers")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(1))

.andExpect(jsonPath("$[0].name").value("Integration Test Customer"));

}

@Test

public void testGetCustomerById() throws Exception {

mockMvc.perform(get("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Integration Test Customer"));

}

@Test

public void testCreateCustomer() throws Exception {

mockMvc.perform(post("/api/customers")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"New Customer\", \"email\": \"newcustomer@example.com\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.name").value("New Customer"));

}

@Test

public void testUpdateCustomer() throws Exception {

mockMvc.perform(put("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"name\": \"Updated Customer\", \"email\": \"updatedcustomer@example.com\"}"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Updated Customer"));

}

@Test

public void testDeleteCustomer() throws Exception {

mockMvc.perform(delete("/api/customers/" + customer1.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNoContent());

Optional<Customer> deletedCustomer = customerRepository.findById(customer1.getId());

assert(deletedCustomer.isEmpty());

}

}

**Scenario 15: Online Bookstore - API Documentation with Swagger**

**Business Scenario:**

**Document your bookstore's REST APIs using Swagger and Springdoc.**

**CODE:**

// IntegrationTestsWithSwagger.java

package com.example.bookstore;

import com.example.bookstore.model.Book;

import com.example.bookstore.model.Customer;

import com.example.bookstore.repository.BookRepository;

import com.example.bookstore.repository.CustomerRepository;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

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

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.responses.ApiResponse;

import io.swagger.v3.oas.annotations.responses.ApiResponses;

import io.swagger.v3.oas.annotations.parameters.RequestBody as SwaggerRequestBody;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springdoc.core.GroupedOpenApi;

import org.springdoc.webmvc.ui.SwaggerConfig;

import javax.validation.Valid;

import java.util.List;

import java.util.Optional;

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

@Bean

public WebMvcConfigurer configurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("swagger-ui.html")

.addResourceLocations("classpath:/META-INF/resources/");

registry.addResourceHandler("/webjars/\*\*")

.addResourceLocations("classpath:/META-INF/resources/webjars/");

}

};

}

}

@RestController

@RequestMapping("/api/books")

class BookController {

@Autowired

private BookRepository bookRepository;

@Operation(summary = "Get all books", description = "Retrieve a list of all books")

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@Operation(summary = "Get book by ID", description = "Retrieve a book by its ID")

@ApiResponses({

@ApiResponse(responseCode = "200", description = "Book found"),

@ApiResponse(responseCode = "404", description = "Book not found")

})

@GetMapping("/{id}")

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

Optional<Book> book = bookRepository.findById(id);

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@Operation(summary = "Create a new book", description = "Add a new book to the catalog")

@PostMapping

public ResponseEntity<Book> createBook(@Valid @SwaggerRequestBody(description = "Book object to be created") @RequestBody Book book) {

Book savedBook = bookRepository.save(book);

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

}

@Operation(summary = "Update an existing book", description = "Update the details of an existing book")

@PutMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

book.setId(id);

Book updatedBook = bookRepository.save(book);

return ResponseEntity.ok(updatedBook);

}

@Operation(summary = "Delete a book", description = "Remove a book from the catalog")

@DeleteMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

@RestController

@RequestMapping("/api/customers")

class CustomerController {

@Autowired

private CustomerRepository customerRepository;

@Operation(summary = "Get all customers", description = "Retrieve a list of all customers")

@GetMapping

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

@Operation(summary = "Get customer by ID", description = "Retrieve a customer by its ID")

@ApiResponses({

@ApiResponse(responseCode = "200", description = "Customer found"),

@ApiResponse(responseCode = "404", description = "Customer not found")

})

@GetMapping("/{id}")

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

Optional<Customer> customer = customerRepository.findById(id);

return customer.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@Operation(summary = "Create a new customer", description = "Add a new customer to the system")

@PostMapping

public ResponseEntity<Customer> createCustomer(@Valid @SwaggerRequestBody(description = "Customer object to be created") @RequestBody Customer customer) {

Customer savedCustomer = customerRepository.save(customer);

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

}

@Operation(summary = "Update an existing customer", description = "Update the details of an existing customer")

@PutMapping("/{id}")

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

if (!customerRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

customer.setId(id);

Customer updatedCustomer = customerRepository.save(customer);

return ResponseEntity.ok(updatedCustomer);

}

@Operation(summary = "Delete a customer", description = "Remove a customer from the system")

@DeleteMapping("/{id}")

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

if (!customerRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

customerRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}