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

**Book Entity**

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

private String title;

@NotNull

private String author;

@NotNull

@Min(0)

private double price;

@NotNull

private String isbn;

@Version

private int version;

// Getters and Setters

}

Customer Entity

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String name;

@NotNull

private String email;

@Version

private int version;

// Getters and Setters

}

BookController

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

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

return bookService.getBookById(id)

.map(ResponseEntity::ok)

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

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

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

}

@PutMapping("/{id}")

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

return bookService.updateBook(id, book)

.map(updatedBook -> ResponseEntity.ok().body(updatedBook))

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

}

@DeleteMapping("/{id}")

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

if (bookService.deleteBook(id)) {

return ResponseEntity.noContent().build();

}

return ResponseEntity.notFound().build();

}

}

CustomerController

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping

public List<Customer> getAllCustomers() {

return customerService.getAllCustomers();

}

@GetMapping("/{id}")

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

return customerService.getCustomerById(id)

.map(ResponseEntity::ok)

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

}

@PostMapping

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

Customer createdCustomer = customerService.createCustomer(customer);

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

}

@PutMapping("/{id}")

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

return customerService.updateCustomer(id, customer)

.map(updatedCustomer -> ResponseEntity.ok().body(updatedCustomer))

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

}

@DeleteMapping("/{id}")

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

if (customerService.deleteCustomer(id)) {

return ResponseEntity.noContent().build();

}

return ResponseEntity.notFound().build();

}

}

CustomerController

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping

public List<Customer> getAllCustomers() {

return customerService.getAllCustomers();

}

@GetMapping("/{id}")

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

return customerService.getCustomerById(id)

.map(ResponseEntity::ok)

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

}

@PostMapping

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

Customer createdCustomer = customerService.createCustomer(customer);

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

}

@PutMapping("/{id}")

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

return customerService.updateCustomer(id, customer)

.map(updatedCustomer -> ResponseEntity.ok().body(updatedCustomer))

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

}

@DeleteMapping("/{id}")

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

if (customerService.deleteCustomer(id)) {

return ResponseEntity.noContent().build();

}

return ResponseEntity.notFound().build();

}

}

#### ****Service Layer****

##### **BookService**

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

public Book createBook(Book book) {

return bookRepository.save(book);

}

public Optional<Book> updateBook(Long id, Book bookDetails) {

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

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setPrice(bookDetails.getPrice());

book.setIsbn(bookDetails.getIsbn());

return bookRepository.save(book);

});

}

public boolean deleteBook(Long id) {

if (bookRepository.existsById(id)) {

bookRepository.deleteById(id);

return true;

}

return false;

}

}

CustomerService

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Optional<Customer> getCustomerById(Long id) {

return customerRepository.findById(id);

}

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

public Optional<Customer> updateCustomer(Long id, Customer customerDetails) {

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

customer.setName(customerDetails.getName());

customer.setEmail(customerDetails.getEmail());

return customerRepository.save(customer);

});

}

public boolean deleteCustomer(Long id) {

if (customerRepository.existsById(id)) {

customerRepository.deleteById(id);

return true;

}

return false;

}

}

#### ****Repository Layer****

##### **BookRepository**

public interface BookRepository extends JpaRepository<Book, Long> {

}

CustomerRepository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

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

#### ****Adding Links to Resources****

To implement HATEOAS, you can use Spring HATEOAS to add links to your resources.

import org.springframework.hateoas.EntityModel;

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

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping("/{id}")

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

return bookService.getBookById(id)

.map(book -> {

EntityModel<Book> resource = EntityModel.of(book);

WebMvcLinkBuilder linkToBooks = WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(this.getClass()).getAllBooks());

resource.add(linkToBooks.withRel("all-books"));

return ResponseEntity.ok(resource);

})

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

}

}

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

#### ****Supporting Different Media Types****

To support different media types, configure content negotiation in Spring Boot.

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorParameter(true)

.parameterName("mediaType")

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

}

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

#### ****Adding Actuator Dependency and Configuring Endpoints****

Add the Actuator dependency in your pom.xml or build.gradle:

<dependency>

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

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

</dependency>

Customizing Actuator Endpoints

management:

endpoints:

web:

exposure:

include: "\*"

Exposing Custom Metrics

import io.micrometer.core.instrument.MeterRegistry;

import org.springframework.stereotype.Component;

@Component

public class CustomMetrics {

public CustomMetrics(MeterRegistry registry) {

registry.counter("books.service.count").increment();

}

}

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

#### ****Integrating Spring Security with JWT Authentication****

1. **Add Security Dependency**

<dependency>

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

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

</dependency>

JWT Security Configuration

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

}

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter();

}

}

JWT Security Configuration

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

}

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() {

return new JwtAuthenticationFilter();

}

}

JWT Token Utility

@Service

public class JwtTokenProvider {

private final String jwtSecret = "secretKey";

public String generateToken(Authentication authentication) {

Date now = new Date();

Date expiryDate = new Date(now.getTime() + 864000000);

return Jwts.builder()

.setSubject(authentication.getName())

.setIssuedAt(new Date())

.setExpiration(expiryDate)

.signWith(SignatureAlgorithm.HS512, jwtSecret)

.compact();

}

}

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

#### ****Using MockMvc for Unit Testing****

1. **JUnit Setup**

<dependency>

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

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

<scope>test</scope>

</dependency>

Unit Test for BookController

@RunWith(SpringRunner.class)

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@Test

public void testGetAllBooks() throws Exception {

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

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON));

}

}

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

**Spring Test Setup**

1. **Integration Test for BookController**

java

@RunWith(SpringRunner.class)

@SpringBootTest

@AutoConfigureMockMvc

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testGetBookById() throws Exception {

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

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON));

}

}

#### ****Database Integration with H2****

xml

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>test</scope>

</dependency>

### ****Exercise 15: Online Bookstore - API Documentation with Swagger****

#### ****Integrating Swagger****

1. **Add Swagger Dependency**

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-boot-starter</artifactId>

<version>3.0.0</version>

</dependency>

Swagger Configuration

@Configuration

@EnableSwagger2

public class SwaggerConfig {

@Bean

public Docket api() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build();

}

}

Annotate REST Controllers

@Api(tags = "Books")

@RestController

@RequestMapping("/books")

public class BookController {

// Controller methods...

}