# Q8.

package com.example.bookstoreapi.model;

import org.springframework.hateoas.RepresentationModel;

public class BookModel extends RepresentationModel<BookModel> {

private Long id;

private String title;

private String author;

private Double price;

private String isbn;

// Constructors, getters, and setters

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.BookModel;

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.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

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

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> bookList = new ArrayList<>();

public BookController() {

bookList.add(new Book(1L, "The Great Gatsby", "F. Scott Fitzgerald", 10.99, "9780743273565"));

bookList.add(new Book(2L, "1984", "George Orwell", 8.99, "9780451524935"));

}

// GET all books with HATEOAS links

@GetMapping

public List<BookModel> getAllBooks() {

return bookList.stream()

.map(book -> {

BookModel bookModel = BookMapper.INSTANCE.bookToBookDTO(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

return bookModel;

})

.collect(Collectors.toList());

}

// GET a single book by ID with HATEOAS links

@GetMapping("/{id}")

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

Book book = bookList.stream()

.filter(b -> b.getId().equals(id))

.findFirst()

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

BookModel bookModel = BookMapper.INSTANCE.bookToBookDTO(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(id)).withSelfRel());

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

return ResponseEntity.ok(bookModel);

}

}

Update the BookMapper to convert Book to BookModel:

java

Copy code

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.BookModel;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

BookDTO bookToBookDTO(Book book);

Book bookDTOToBook(BookDTO bookDTO);

// Add method to convert Book to BookModel

BookModel bookToBookModel(Book book);

}

Update the BookMapper to convert Book to BookModel:

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.BookModel;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

BookDTO bookToBookDTO(Book book);

Book bookDTOToBook(BookDTO bookDTO);

// Add method to convert Book to BookModel

BookModel bookToBookModel(Book book);

}

When a client fetches all books, it will receive a response like this:

[

{

"id": 1,

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"price": 10.99,

"isbn": "9780743273565",

"\_links": {

"self": {

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

}

}

},

{

"id": 2,

"title": "1984",

"author": "George Orwell",

"price": 8.99,

"isbn": "9780451524935",

"\_links": {

"self": {

"href": "http://localhost:8080/books/2"

}

}

}

]

# Q10.

To support XML in addition to JSON, add the spring-boot-starter-xml dependency to pom.xml:

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.3.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.3.Final</version>

<scope>provided</scope>

</dependency>

To explicitly configure content negotiation, create a configuration class:

package com.example.bookstoreapi.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.MediaType;

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

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

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.BookModel;

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

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

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

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> bookList = new ArrayList<>();

public BookController() {

bookList.add(new Book(1L, "The Great Gatsby", "F. Scott Fitzgerald", 10.99, "9780743273565"));

bookList.add(new Book(2L, "1984", "George Orwell", 8.99, "9780451524935"));

}

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

public List<BookModel> getAllBooks() {

return bookList.stream()

.map(book -> {

BookModel bookModel = BookMapper.INSTANCE.bookToBookModel(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

return bookModel;

})

.collect(Collectors.toList());

}

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

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

Book book = bookList.stream()

.filter(b -> b.getId().equals(id))

.findFirst()

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

BookModel bookModel = BookMapper.INSTANCE.bookToBookModel(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(id)).withSelfRel());

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

return ResponseEntity.ok(bookModel);

}

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

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

book.setId((long) (bookList.size() + 1));

bookList.add(book);

BookModel bookModel = BookMapper.INSTANCE.bookToBookModel(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

return ResponseEntity.ok(bookModel);

}

}

# Q11.

Add the following dependency to the pom.xml:

<dependency>

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

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

</dependency>

After adding the dependency, make sure to refresh your Maven project so that the dependencies are downloaded and applied.

**Example Configuration in application.properties:**

properties

# Enable all endpoints

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

# Or, you can specify particular endpoints like this:

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

# Set the base path for actuator endpoints (default is /actuator)

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

# Customize health check endpoints

management.endpoint.health.show-details=always

**Example Configuration in application.yml:**

management:

endpoints:

web:

exposure:

include: "\*"

base-path: "/actuator"

endpoint:

health:

show-details: "always"

package com.example.bookstoreapi.metrics;

import io.micrometer.core.instrument.MeterRegistry;

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

import org.springframework.context.annotation.Configuration;

@Configuration

public class CustomMetrics {

@Autowired

public CustomMetrics(MeterRegistry meterRegistry) {

// Counter example: Track the number of books added

meterRegistry.counter("bookstore.books.added");

// Gauge example: Track the number of books in the store

meterRegistry.gauge("bookstore.books.count", BookController.getBookList(), List::size);

// Timer example: Track the time taken to process a book creation request

meterRegistry.timer("bookstore.books.creation.timer");

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.BookModel;

import io.micrometer.core.instrument.MeterRegistry;

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

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

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

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

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> bookList = new ArrayList<>();

private final MeterRegistry meterRegistry;

@Autowired

public BookController(MeterRegistry meterRegistry) {

this.meterRegistry = meterRegistry;

bookList.add(new Book(1L, "The Great Gatsby", "F. Scott Fitzgerald", 10.99, "9780743273565"));

bookList.add(new Book(2L, "1984", "George Orwell", 8.99, "9780451524935"));

}

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

public List<BookModel> getAllBooks() {

return bookList.stream()

.map(book -> {

BookModel bookModel = BookMapper.INSTANCE.bookToBookModel(book);

bookModel.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

return bookModel;

})

.collect(Collectors.toList());

}

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

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

Book book = bookList.stream()

.filter(b -> b.getId().equals(id))

.findFirst()

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

BookModel bookModel = BookMapper.INSTANCE.bookToBookModel(book);

# Q12.

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

package com.example.bookstoreapi.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

@Component

public class JwtUtil {

private String secret = "bookstore\_secret\_key";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

private Boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(String username) {

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

return createToken(claims, username);

}

private String createToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.HS256, secret).compact();

}

public Boolean validateToken(String token, String username) {

final String extractedUsername = extractUsername(token);

return (extractedUsername.equals(username) && !isTokenExpired(token));

}

}

package com.example.bookstoreapi.security;

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

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

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

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

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserDetailsService userDetailsService;

@Override

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

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

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

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwt, userDetails.getUsername())) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request, response);

}

}

package com.example.bookstoreapi.security;

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

import org.springframework.context.annotation.Bean;

import org.springframework.security.authentication.AuthenticationManager;

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.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

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

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

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

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailsService userDetailsService;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().ant

# Q13.

<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>

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.service.BookService;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

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

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

import org.springframework.boot.test.mock.mockito.MockBean;

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.BDDMockito.given;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@Autowired

private ObjectMapper objectMapper;

private Book book1;

private Book book2;

@BeforeEach

public void setUp() {

book1 = new Book(1L, "The Great Gatsby", "F. Scott Fitzgerald", 10.99, "9780743273565");

book2 = new Book(2L, "1984", "George Orwell", 8.99, "9780451524935");

}

@Test

public void testGetAllBooks() throws Exception {

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

given(bookService.getAllBooks()).willReturn(allBooks);

mockMvc.perform(get("/books")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$[0].title").value("The Great Gatsby"))

.andExpect(jsonPath("$[1].title").value("1984"));

}

@Test

public void testCreateBook() throws Exception {

Book newBook = new Book(null, "Brave New World", "Aldous Huxley", 12.99, "9780060850524");

given(bookService.createBook(newBook)).willReturn(new Book(3L, "Brave New World", "Aldous Huxley", 12.99, "9780060850524"));

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(newBook)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Brave New World"));

}

}

# Q14.

<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>

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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.web.servlet.MockMvc;

import static org.hamcrest.Matchers.hasSize;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest

@AutoConfigureMockMvc

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private ObjectMapper objectMapper;

@Autowired

private BookRepository bookRepository;

@BeforeEach

public void setUp() {

bookRepository.deleteAll();

}

@Test

public void testGetAllBooks() throws Exception {

Book book1 = new Book(null, "The Great Gatsby", "F. Scott Fitzgerald", 10.99, "9780743273565");

Book book2 = new Book(null, "1984", "George Orwell", 8.99, "9780451524935");

bookRepository.save(book1);

bookRepository.save(book2);

mockMvc.perform(get("/books")

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$", hasSize(2)))

.andExpect(jsonPath("$[0].title").value("The Great Gatsby"))

.andExpect(jsonPath("$[1].title").value("1984"));

}

@Test

public void testCreateBook() throws Exception {

Book newBook = new Book(null, "Brave New World", "Aldous Huxley", 12.99, "9780060850524");

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(newBook)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Brave New World"));

// Verify that the book was saved in the database

Book savedBook = bookRepository.findAll().get(0);

assert savedBook.getTitle().equals("Brave New World");

}

}

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

@Test

public void testFindBookById() throws Exception {

Book book = new Book(null, "The Catcher in the Rye", "J.D. Salinger", 9.99, "9780316769488");

Book savedBook = bookRepository.save(book);

mockMvc.perform(get("/books/{id}", savedBook.getId())

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("The Catcher in the Rye"));

}

# Q15.

Add the Springdoc OpenAPI dependency to your `pom.xml` file:

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.2.0</version>

</dependency>

2.package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.service.BookService;

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

import io.swagger.v3.oas.annotations.tags.Tag;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/books")

@Tag(name = "Books", description = "Operations related to books in the online bookstore")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

@Operation(summary = "Get all books", description = "Fetches all the books available in the bookstore")

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

return ResponseEntity.ok(bookService.getAllBooks());

}

@GetMapping("/{id}")

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

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

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

}

@PostMapping

@Operation(summary = "Create a new book", description = "Adds a new book to the bookstore")

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

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

}

@PutMapping("/{id}")

@Operation(summary = "Update a book", description = "Updates an existing book in the bookstore")

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

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

}

@DeleteMapping("/{id}")

@Operation(summary = "Delete a book", description = "Removes a book from the bookstore by its ID")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

```

Generate and Review API Documentation

Run your Spring Boot application, and access the Swagger UI to review the API documentation.

1. \*\*Start the Application\*\*: Run your application as you normally would (`mvn spring-boot:run` or using your IDE).

2. \*\*Access Swagger UI\*\*:

- Open your web browser and navigate to `http://localhost:8080/swagger-ui.html`.

- This page will show the generated API documentation, including all the endpoints, their descriptions, request parameters, and response types.

3.2: Customize API Documentation

You can further customize your API documentation using additional annotations and configurations.

- \*\*`@Parameter`\*\*: Used to describe parameters for your endpoints.

- \*\*`@ApiResponse`\*\*: Used to provide metadata about the possible responses from an endpoint.

- \*\*`@Schema`\*\*: Used to describe the schema of request or response bodies.

@GetMapping("/{id}")

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

@ApiResponse(responseCode = "200", description = "Successfully retrieved book")

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

public ResponseEntity<Book> getBookById(

@Parameter(description = "ID of the book to be fetched") @PathVariable Long id) {

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

}