# Exercise 8: Online Bookstore - Implementing CRUD Operations

##### CRUD Endpoints

package com.example.bookstore.controller;

import com.example.bookstore.dto.BookDTO;

import com.example.bookstore.mapper.BookMapper; import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import com.example.bookstore.exception.ResourceNotFoundException; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid; import java.util.List;

import java.util.Optional;

@RestController @RequestMapping("/books") public class BookController {

@Autowired

private BookRepository bookRepository;

@Autowired

private BookMapper bookMapper;

// Create a new book @PostMapping

public ResponseEntity<BookDTO> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookMapper.bookDTOToBook(bookDTO); Book savedBook = bookRepository.save(book);

return ResponseEntity.status(201).body(bookMapper.bookToBookDTO(savedBook));

}

// Read all books @GetMapping

public List<BookDTO> getAllBooks() { return bookRepository.findAll().stream()

.map(bookMapper::bookToBookDTO)

.toList();

}

// Read a single book by ID @GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(@PathVariable Long id) { Book book = bookRepository.findById(id)

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

return ResponseEntity.ok(bookMapper.bookToBookDTO(book));

}

// Update a book by ID @PutMapping("/{id}")

public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @Valid @RequestBody BookDTO bookDTO) {

Book existingBook = bookRepository.findById(id)

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

Book book = bookMapper.bookDTOToBook(bookDTO); book.setId(id);

Book updatedBook = bookRepository.save(book);

return ResponseEntity.ok(bookMapper.bookToBookDTO(updatedBook));

}

// Delete a book by ID @DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) { if (bookRepository.existsById(id)) {

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

throw new ResourceNotFoundException("Book not found with id " + id);

}

}

}

##### CustomerController

package com.example.bookstore.controller; import com.example.bookstore.dto.CustomerDTO;

import com.example.bookstore.mapper.CustomerMapper; import com.example.bookstore.model.Customer;

import com.example.bookstore.repository.CustomerRepository;

import com.example.bookstore.exception.ResourceNotFoundException; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid; import java.util.List;

@RestController @RequestMapping("/customers") public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

@Autowired

private CustomerMapper customerMapper;

// Create a new customer @PostMapping

public ResponseEntity<CustomerDTO> createCustomer(@Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = customerMapper.customerDTOToCustomer(customerDTO);

Customer savedCustomer = customerRepository.save(customer); return

ResponseEntity.status(201).body(customerMapper.customerToCustomerDTO(savedC ustomer));

}

// Read all customers @GetMapping

public List<CustomerDTO> getAllCustomers() { return customerRepository.findAll().stream()

.map(customerMapper::customerToCustomerDTO)

.toList();

}

// Read a single customer by ID @GetMapping("/{id}")

public ResponseEntity<CustomerDTO> getCustomerById(@PathVariable Long id) { Customer customer = customerRepository.findById(id)

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

return ResponseEntity.ok(customerMapper.customerToCustomerDTO(customer));

}

// Update a customer by ID @PutMapping("/{id}")

public ResponseEntity<CustomerDTO> updateCustomer(@PathVariable Long id, @Valid @RequestBody CustomerDTO customerDTO) {

Customer existingCustomer = customerRepository.findById(id)

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

Customer customer = customerMapper.customerDTOToCustomer(customerDTO);

customer.setId(id);

Customer updatedCustomer = customerRepository.save(customer); return

ResponseEntity.ok(customerMapper.customerToCustomerDTO(updatedCustomer));

}

// Delete a customer by ID @DeleteMapping("/{id}")

public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) { if (customerRepository.existsById(id)) {

customerRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

throw new ResourceNotFoundException("Customer not found with id " + id);

}

}

}

##### Validating Input Data

package com.example.bookstore.dto;

import javax.validation.constraints.NotNull; import javax.validation.constraints.Size; import javax.validation.constraints.Min;

public class BookDTO {

private Long id;

@NotNull

@Size(min = 1, max = 100) private String title;

@NotNull

@Size(min = 1, max = 50) private String author;

@NotNull @Min(0)

private Double price;

@NotNull

@Size(min = 10, max = 13) private String isbn;

// Constructors, Getters, and Setters

}

**CustomerDTO Example:**

package com.example.bookstore.dto;

import javax.validation.constraints.NotNull; import javax.validation.constraints.Size; import javax.validation.constraints.Email;

public class CustomerDTO {

private Long id;

@NotNull

@Size(min = 1, max = 50) private String name;

@NotNull @Email

private String email;

@Size(min = 10, max = 15) private String phoneNumber;

// Constructors, Getters, and Setters

}

##### Implementing Optimistic Locking

package com.example.bookstore.model;

import javax.persistence.\*;

import javax.validation.constraints.NotNull; import javax.validation.constraints.Size; import javax.validation.constraints.Min;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

@Version

private Long version;

@NotNull

@Size(min = 1, max = 100) private String title;

@NotNull

@Size(min = 1, max = 50) private String author;

@NotNull @Min(0)

private Double price;

@NotNull

@Size(min = 10, max = 13) private String isbn;

// Constructors, Getters, and Setters

}

**Customer Entity Example:**

package com.example.bookstore.model; import javax.persistence.\*;

import javax.validation.constraints.NotNull; import javax.validation.constraints.Size; import javax.validation.constraints.Email;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Version

private Long version;

@NotNull

@Size(min = 1, max = 50) private String name;

@NotNull @Email

private String email;

@Size(min = 10, max = 15) private String phoneNumber;

// Constructors, Getters, and Setters

}

# Exercise 9: Online Bookstore - Understanding HATEOAS

##### Add Spring HATEOAS Dependency

<dependency>

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

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

</dependency>

##### Create Resource Assemblers

package com.example.bookstore.assembler;

import com.example.bookstore.controller.BookController;

import com.example.bookstore.dto.BookDTO; import com.example.bookstore.model.Book; import org.springframework.hateoas.EntityModel; import org.springframework.hateoas.Link;

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

@Component

public class BookResourceAssembler {

public EntityModel<BookDTO> toModel(BookDTO bookDTO) { EntityModel<BookDTO> bookResource = EntityModel.of(bookDTO); Link selfLink =

WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).get BookById(bookDTO.getId())).withSelfRel();

Link allBooksLink = WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).get AllBooks()).withRel("all-books");

bookResource.add(selfLink, allBooksLink); return bookResource;

}

}

**CustomerResourceAssembler**

package com.example.bookstore.assembler;

import com.example.bookstore.controller.CustomerController; import com.example.bookstore.dto.CustomerDTO;

import com.example.bookstore.model.Customer; import org.springframework.hateoas.EntityModel; import org.springframework.hateoas.Link;

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

@Component

public class CustomerResourceAssembler {

public EntityModel<CustomerDTO> toModel(CustomerDTO customerDTO) {

EntityModel<CustomerDTO> customerResource = EntityModel.of(customerDTO);

Link selfLink = WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerC ontroller.class).getCustomerById(customerDTO.getId())).withSe lfRel();

Link allCustomersLink = WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(CustomerC ontroller.class).getAllCustomers()).withRel("all-customers");

customerResource.add(selfLink, allCustomersLink); return customerResource;

}

}

##### Modify Controllers to Include Links

package com.example.bookstore.controller;

import com.example.bookstore.dto.BookDTO;

import com.example.bookstore.assembler.BookResourceAssembler; import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import com.example.bookstore.exception.ResourceNotFoundException; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.hateoas.EntityModel;

import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid; import java.util.List;

import java.util.Optional;

@RestController @RequestMapping("/books") public class BookController {

@Autowired

private BookRepository bookRepository;

@Autowired

private BookResourceAssembler bookResourceAssembler;

@PostMapping

public ResponseEntity<EntityModel<BookDTO>> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = new Book(); // Assuming you have a method to convert DTO to Entity book.setTitle(bookDTO.getTitle());

book.setAuthor(bookDTO.getAuthor()); book.setPrice(bookDTO.getPrice()); book.setIsbn(bookDTO.getIsbn());

Book savedBook = bookRepository.save(book);

BookDTO savedBookDTO = new BookDTO(savedBook.getId(), savedBook.getTitle(), savedBook.getAuthor(), savedBook.getPrice(), savedBook.getIsbn());

return ResponseEntity.status(201).body(bookResourceAssembler.toModel(savedBookDTO));

}

@GetMapping

public List<EntityModel<BookDTO>> getAllBooks() { return bookRepository.findAll().stream()

.map(book -> bookResourceAssembler.toModel(new BookDTO(book.getId(), book.getTitle(), book.getAuthor(), book.getPrice(), book.getIsbn())))

.toList();

}

@GetMapping("/{id}")

public ResponseEntity<EntityModel<BookDTO>> getBookById(@PathVariable Long id) { Book book = bookRepository.findById(id)

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

BookDTO bookDTO = new BookDTO(book.getId(), book.getTitle(), book.getAuthor(), book.getPrice(), book.getIsbn());

return ResponseEntity.ok(bookResourceAssembler.toModel(bookDTO));

}

@PutMapping("/{id}")

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

Book existingBook = bookRepository.findById(id)

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

existingBook.setTitle(bookDTO.getTitle()); existingBook.setAuthor(bookDTO.getAuthor()); existingBook.setPrice(bookDTO.getPrice()); existingBook.setIsbn(bookDTO.getIsbn());

Book updatedBook = bookRepository.save(existingBook);

BookDTO updatedBookDTO = new BookDTO(updatedBook.getId(), updatedBook.getTitle(), updatedBook.getAuthor(), updatedBook.getPrice(), updatedBook.getIsbn());

return ResponseEntity.ok(bookResourceAssembler.toModel(updatedBookDTO));

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) { if (bookRepository.existsById(id)) {

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

throw new ResourceNotFoundException("Book not found with id " + id);

}

}

}

##### CustomerController

package com.example.bookstore.controller;

import com.example.bookstore.dto.CustomerDTO;

import com.example.bookstore.assembler.CustomerResourceAssembler; import com.example.bookstore.model.Customer;

import com.example.bookstore.repository.CustomerRepository;

import com.example.bookstore.exception.ResourceNotFoundException; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.hateoas.EntityModel;

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

import javax.validation.Valid; import java.util.List;

@RestController @RequestMapping("/customers") public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

@Autowired

private CustomerResourceAssembler customerResourceAssembler;

@PostMapping

public ResponseEntity<EntityModel<CustomerDTO>> createCustomer(@Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = new Customer(); // Assuming you have a method to convert DTO to Entity

customer.setName(customerDTO.getName()); customer.setEmail(customerDTO.getEmail()); customer.setPhoneNumber(customerDTO.getPhoneNumber()); Customer savedCustomer = customerRepository.save(customer);

CustomerDTO savedCustomerDTO = new CustomerDTO(savedCustomer.getId(), savedCustomer.getName(), savedCustomer.getEmail(), savedCustomer.getPhoneNumber());

return ResponseEntity.status(201).body(customerResourceAssembler.toModel(savedCusto merDTO));

}

@GetMapping

public List<EntityModel<CustomerDTO>> getAllCustomers() { return customerRepository.findAll().stream()

.map(customer -> customerResourceAssembler.toModel(new CustomerDTO(customer.getId(), customer.getName(), customer.getEmail(), customer.getPhoneNumber())))

.toList();

}

@GetMapping("/{id}")

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

Customer customer = customerRepository.findById(id)

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

CustomerDTO customerDTO = new CustomerDTO(customer.getId(), customer.getName(), customer.getEmail(), customer.getPhoneNumber());

return ResponseEntity.ok(customerResourceAssembler.toModel(customerDTO));

}

@PutMapping("/{id}")

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

Customer existingCustomer = customerRepository.findById(id)

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

existingCustomer.setName(customerDTO.getName()); existingCustomer.setEmail(customerDTO.getEmail()); existingCustomer.setPhoneNumber(customerDTO.getPhoneNumber());

Customer updatedCustomer = customerRepository.save(existingCustomer); CustomerDTO updatedCustomerDTO = new

CustomerDTO(updatedCustomer.getId(), updatedCustomer.getName(), updatedCustomer.getEmail(), updatedCustomer.getPhoneNumber());

return ResponseEntity.ok(customerResourceAssembler.toModel(updatedCustomerDTO));

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) { if (customerRepository.existsById(id)) {

customerRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

throw new ResourceNotFoundException("Customer not found with id " + id);

}

}

}

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

##### Conﬁgure Content Negotiation in Spring Boot

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration; 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.favorParameter(false)

.ignoreAcceptHeader(false)

.defaultContentType(org.springframework.http.MediaType.APPLICATION\_J

SON)

.mediaType("json", org.springframework.http.MediaType.APPLICATION\_JSON)

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

}

}

##### Implement Logic Based on Accept Header

import org.springframework.http.MediaType;

import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import java.util.HashMap;

import java.util.Map;

@RestController @RequestMapping("/books") public class BookController {

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

public Map<String, String> getBooks() {

Map<String, String> books = new HashMap<>(); books.put("1", "Spring Boot in Action"); books.put("2", "Effective Java");

return books;

}

}

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

##### Add Actuator Dependency

<dependency>

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

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

</dependency>

##### Expose Actuator Endpoints

# Enable all actuator endpoints management.endpoints.web.exposure.include=\*

# Enable specific actuator endpoints management.endpoints.web.exposure.include=health,info,metrics management:

endpoints:

web:

exposure:

include: health,info,metrics management.endpoints.web.base-path=/actuator management.endpoints.web.path-mapping.health=health-check

management:

endpoints:

web:

base-path: /actuator path-mapping:

health: health-check

##### 4. Expose Custom Metrics

import io.micrometer.core.instrument.MeterRegistry; import org.springframework.stereotype.Component;

@Component

public class CustomMetrics {

private final MeterRegistry meterRegistry;

public CustomMetrics(MeterRegistry meterRegistry) { this.meterRegistry = meterRegistry; this.registerCustomMetrics();

}

private void registerCustomMetrics() {

meterRegistry.gauge("custom.metric", 42); // Register a simple gauge metric

}

}

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

##### Add Spring Security to Your Project

###### <dependency>

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

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

**</dependency>**

###### b. Create a Security Configuration Class:

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration; import

org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuil der;

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

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

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

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

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

@Configuration @EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception { http.csrf().disable()

.authorizeRequests()

.antMatchers("/public/\*\*").permitAll() // Allow public endpoints

.anyRequest().authenticated() // Secure all other endpoints

.and()

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

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

// Configure authentication provider

}

@Bean

public JwtAuthenticationFilter jwtAuthenticationFilter() { return new JwtAuthenticationFilter();

}

}

##### Implement JWT-Based Authentication

import io.jsonwebtoken.Claims; import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm; import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

private String secretKey = "your\_secret\_key"; // Use a strong secret key

public String generateToken(String username) { return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60)) // 1 hour

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

public Claims extractClaims(String token) { return Jwts.parser()

.setSigningKey(secretKey)

.parseClaimsJws(token)

.getBody();

}

public String extractUsername(String token) { return extractClaims(token).getSubject();

}

public boolean isTokenExpired(String token) {

return extractClaims(token).getExpiration().before(new Date());

}

public boolean validateToken(String token, String username) {

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

}

}

###### b. Create JWT Authentication Filter:

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

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter; import javax.servlet.FilterChain;

import javax.servlet.ServletException; import javax.servlet.ServletRequest; import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest; import java.io.IOException;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

private JwtUtil jwtUtil;

public JwtAuthenticationFilter(JwtUtil jwtUtil) { this.jwtUtil = jwtUtil;

}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

String authHeader = httpRequest.getHeader("Authorization");

if (authHeader != null && authHeader.startsWith("Bearer ")) { String token = authHeader.substring(7);

if (jwtUtil.validateToken(token, jwtUtil.extractUsername(token))) { SecurityContextHolder.getContext().setAuthentication(jwtUtil.getAuthentication(token));

}

}

chain.doFilter(request, response);

}

}

1. Configure CORS Handling

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

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) { registry.addMapping("/\*\*")

.allowedOrigins("\*")

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

.allowedHeaders("\*")

.allowCredentials(true);

}

}

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

##### JUnit and Mockito Setup

<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>

<version>4.0.0</version> <!-- or the latest version -->

<scope>test</scope>

</dependency>

##### Use MockMvc to Write Unit Tests

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.test.web.servlet.MockMvc;

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

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.status;

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

@WebMvcTest(BookController.class) public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@BeforeEach void setUp() {

MockitoAnnotations.openMocks(this);

}

@Test

void testGetBookById() throws Exception {

// Mock the service layer

when(bookService.getBookById(1L)).thenReturn(new Book(1L, "Eﬀective Java", "Joshua Bloch"));

// Perform the request and verify the response mockMvc.perform(get("/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Eﬀective Java"))

.andExpect(jsonPath("$.author").value("Joshua Bloch"));

}

@Test

void testCreateBook() throws Exception {

Book book = new Book(1L, "Clean Code", "Robert C. Martin");

// Mock the service layer when(bookService.createBook(any(Book.class))).thenReturn(book);

// Perform the request and verify the response mockMvc.perform(post("/books")

.contentType("application/json")

.content("{\"title\":\"Clean Code\",\"author\":\"Robert C. Martin\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Clean Code"))

.andExpect(jsonPath("$.author").value("Robert C. Martin"));

}

}

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

##### Set Up Spring Test

<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>

##### MockMvc Integration

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

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

import org.springframework.test.context.ActiveProfiles;

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

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

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.status;

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

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

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

@SpringBootTest @AutoConfigureMockMvc @ActiveProfiles("test")

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository; // Assuming you use Spring Data JPA

@BeforeEach

public void setUp() {

// Clear the database before each test if necessary bookRepository.deleteAll();

}

@Test

void testGetBookById() throws Exception {

// Arrange: Set up your test data

Book book = new Book(1L, "Eﬀective Java", "Joshua Bloch"); bookRepository.save(book);

// Act & Assert: Perform the request and verify the response mockMvc.perform(get("/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Eﬀective Java"))

.andExpect(jsonPath("$.author").value("Joshua Bloch"));

}

@Test

void testCreateBook() throws Exception {

// Act & Assert: Perform the request and verify the response mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\":\"Clean Code\",\"author\":\"Robert C. Martin\"}"))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Clean Code"))

.andExpect(jsonPath("$.author").value("Robert C. Martin"));

}

}

##### Database Integration

spring.datasource.url=jdbc:h2:mem:testdb spring.datasource.driver-class-name=org.h2.Driver spring.datasource.username=sa spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect spring.h2.console.enabled=true

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

##### Add Swagger or Springdoc Dependency

<dependency>

<groupId>org.springdoc</groupId>

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

<version>2.0.0</version> <!-- Use the latest version -->

</dependency>

##### Document Endpoints

import org.springframework.web.bind.annotation.\*; import io.swagger.v3.oas.annotations.Operation; import io.swagger.v3.oas.annotations.Parameter;

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

@RestController @RequestMapping("/books")

@Tag(name = "Book Controller", description = "APIs for managing books") public class BookController {

@GetMapping("/{id}") @Operation(summary = "Get a book by ID",

description = "Retrieve the details of a book by its ID", responses = {

@ApiResponse(responseCode = "200", description = "Book found"), @ApiResponse(responseCode = "404", description = "Book not found")

})

public Book getBookById(

@Parameter(description = "ID of the book to be retrieved") @PathVariable Long

id) {

// Implementation

}

@PostMapping

@Operation(summary = "Create a new book", description = "Add a new book to the collection", responses = {

@ApiResponse(responseCode = "201", description = "Book created"), @ApiResponse(responseCode = "400", description = "Invalid input")

})

public Book createBook( @RequestBody Book book) {

// Implementation

}

}

###### b. Document Models:

import io.swagger.v3.oas.annotations.media.Schema;

@Schema(description = "Book model") public class Book {

@Schema(description = "ID of the book", example = "1") private Long id;

@Schema(description = "Title of the book", example = "Effective Java") private String title;

@Schema(description = "Author of the book", example = "Joshua Bloch") private String author;

// Getters and Setters

}

##### Generate and Review API Documentation

http://localhost:8080/swagger-ui.html