**Exercise 1: Online Bookstore - Setting Up RESTful Services**

* **Jakarta EE 9+ Support:**
* Spring Boot 3 has migrated from Java EE to Jakarta EE. This means all the packages from javax.\* are now jakarta.\*.
* **Java 17+ Baseline:**
* Spring Boot 3 requires Java 17 or higher, taking advantage of the latest language features and performance improvements.
* **Improved Native Image Support:**
* With Spring Boot 3, there’s better support for GraalVM native images, allowing applications to be compiled into native executables with significantly faster startup times.
* **Observability:**
* Enhanced support for observability through the integration of Micrometer, providing more detailed metrics and better observability tooling.
* **Declarative HTTP Clients:**
* Introduction of declarative HTTP clients, allowing for more straightforward and efficient REST client creation without the need for RestTemplate or WebClient.
* **Kotlin 1.7+ Support:**
* Spring Boot 3 now supports Kotlin 1.7 and above, aligning with the latest Kotlin features and improvements.
* **Enhanced Security Features:**
* Upgraded Spring Security to provide better integration with the latest OAuth 2.1 specifications and support for OAuth 2.0 Device Authorization Grant.
* **Better Integration with Testcontainers:**
* Enhanced integration with Testcontainers for more efficient and straightforward testing, especially for database-related tests.

**Exercise 2: Online Bookstore - Creating Basic REST Controllers**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

// GET /books - Retrieve all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET /books/{id} - Retrieve a specific book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream().filter(book -> book.getId().equals(id)).findFirst().orElse(null);

}

// POST /books - Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

books.add(book);

return book;

}

// PUT /books/{id} - Update an existing book

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

Book book = books.stream().filter(b -> b.getId().equals(id)).findFirst().orElse(null);

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

}

return book;

}

// DELETE /books/{id} - Delete a book by ID

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

books.removeIf(book -> book.getId().equals(id));

}

}

package com.example.bookstoreapi.model;

import lombok.Data;

@Data

public class Book {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

curl -X GET http://localhost:8080/books

curl -X POST http://localhost:8080/books -H "Content-Type: application/json" -d '{"id":1,"title":"Book Title","author":"Author Name","price":19.99,"isbn":"123-4567890123"}'

curl -X DELETE http://localhost:8080/books/1

**Exercise 3: Online Bookstore - Handling Path Variables and Query Parameters**

// GET /books/{id} - Retrieve a specific book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream()

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

.findFirst()

.orElse(null);

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

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

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

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

// Existing endpoints...

// GET /books/search - Retrieve books based on title and/or author

@GetMapping("/search")

public List<Book> searchBooks(@RequestParam(required = false) String title,

@RequestParam(required = false) String author) {

return books.stream()

.filter(book -> (title == null || book.getTitle().equalsIgnoreCase(title)) &&

(author == null || book.getAuthor().equalsIgnoreCase(author)))

.collect(Collectors.toList());

}

}

curl -X GET http://localhost:8080/books/1

curl -X GET http://localhost:8080/books/search?title=SomeTitle

curl -X GET http://localhost:8080/books/search?author=SomeAuthor

curl -X GET http://localhost:8080/books/search?title=SomeTitle&author=SomeAuthor

**Exercise 4: Online Bookstore - Processing Request Body and Form Data**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Customer;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

private List<Customer> customers = new ArrayList<>();

// POST /customers - Create a new customer

@PostMapping

public Customer createCustomer(@RequestBody Customer customer) {

customers.add(customer);

return customer;

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Customer;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

private List<Customer> customers = new ArrayList<>();

// Existing POST endpoint for JSON body...

// POST /customers/register - Process form data for customer registration

@PostMapping("/register")

public Customer registerCustomer(@RequestParam String name,

@RequestParam String email,

@RequestParam String password) {

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customer.setPassword(password);

customers.add(customer);

return customer;

}

}

package com.example.bookstoreapi.model;

import lombok.Data;

@Data

public class Customer {

private Long id;

private String name;

private String email;

private String password;

}

curl -X POST http://localhost:8080/customers -H "Content-Type: application/json" -d '{"id":1,"name":"John Doe","email":"john.doe@example.com","password":"password123"}'

curl -X POST http://localhost:8080/customers/register -d "name=Jane Doe&email=jane.doe@example.com&password=secret123"

**Exercise 5: Online Bookstore - Customizing Response Status and Headers**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import org.springframework.http.HttpStatus;

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

import org.springframework.web.server.ResponseStatusException;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

// Other endpoints...

// GET /books/{id} - Retrieve a specific book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream()

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

.findFirst()

.orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "Book not found"));

}

// POST /books - Add a new book

@ResponseStatus(HttpStatus.CREATED)

@PostMapping

public Book addBook(@RequestBody Book book) {

books.add(book);

return book;

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.server.ResponseStatusException;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

// Other endpoints...

// GET /books/{id} - Retrieve a specific book by ID with custom headers

@GetMapping("/{id}")

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

Book book = books.stream()

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

.findFirst()

.orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT\_FOUND, "Book not found"));

HttpHeaders headers = new HttpHeaders();

headers.add("X-Custom-Header", "CustomHeaderValue");

return new ResponseEntity<>(book, headers, HttpStatus.OK);

}

}

curl -X GET http://localhost:8080/books/1 -i

HTTP/1.1 200 OK

X-Custom-Header: CustomHeaderValue

Content-Type: application/json

Content-Length: 123

{

"id": 1,

"title": "Book Title",

"author": "Author Name",

"price": 19.99,

"isbn": "123-4567890123"

}

curl -X POST http://localhost:8080/books -H "Content-Type: application/json" -d '{"id":1,"title":"New Book","author":"New Author","price":29.99,"isbn":"987-6543210987"}'

**Exercise 6: Online Bookstore - Exception Handling in REST Controllers**

package com.example.bookstoreapi.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.server.ResponseStatusException;

import java.util.HashMap;

import java.util.Map;

@ControllerAdvice

public class GlobalExceptionHandler {

// Handle ResponseStatusException (e.g., 404 Not Found, 400 Bad Request)

@ExceptionHandler(ResponseStatusException.class)

public ResponseEntity<Map<String, String>> handleResponseStatusException(ResponseStatusException ex) {

Map<String, String> errorDetails = new HashMap<>();

errorDetails.put("message", ex.getReason());

errorDetails.put("status", ex.getStatus().toString());

return new ResponseEntity<>(errorDetails, ex.getStatus());

}

// Handle IllegalArgumentException (e.g., when invalid data is provided)

@ExceptionHandler(IllegalArgumentException.class)

public ResponseEntity<Map<String, String>> handleIllegalArgumentException(IllegalArgumentException ex) {

Map<String, String> errorDetails = new HashMap<>();

errorDetails.put("message", ex.getMessage());

errorDetails.put("status", HttpStatus.BAD\_REQUEST.toString());

return new ResponseEntity<>(errorDetails, HttpStatus.BAD\_REQUEST);

}

// Handle any other exceptions (e.g., 500 Internal Server Error)

@ExceptionHandler(Exception.class)

public ResponseEntity<Map<String, String>> handleException(Exception ex) {

Map<String, String> errorDetails = new HashMap<>();

errorDetails.put("message", "An unexpected error occurred");

errorDetails.put("status", HttpStatus.INTERNAL\_SERVER\_ERROR.toString());

return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

curl -X GET http://localhost:8080/books/999 -i

HTTP/1.1 404 Not Found

Content-Type: application/json

{

"message": "Book not found",

"status": "404 NOT\_FOUND"

}

curl -X POST http://localhost:8080/customers -H "Content-Type: application/json" -d '{"name":""}'

HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"message": "Invalid input data",

"status": "400 BAD\_REQUEST"

}

curl -X GET http://localhost:8080/books -i

HTTP/1.1 500 Internal Server Error

Content-Type: application/json

{

"message": "An unexpected error occurred",

"status": "500 INTERNAL\_SERVER\_ERROR"

}

**Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)**

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class BookDTO {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

private String name;

private String email;

}

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>3.1.0</version>

</dependency>

package com.example.bookstoreapi.config;

import org.modelmapper.ModelMapper;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ModelMapperConfig {

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import org.modelmapper.ModelMapper;

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

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

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private ModelMapper modelMapper;

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

// GET /books - Retrieve all books and convert to DTOs

@GetMapping

public List<BookDTO> getAllBooks() {

return books.stream()

.map(book -> modelMapper.map(book, BookDTO.class))

.collect(Collectors.toList());

}

// POST /books - Add a new book using DTO

@PostMapping

public BookDTO addBook(@RequestBody BookDTO bookDTO) {

Book book = modelMapper.map(bookDTO, Book.class);

books.add(book);

return modelMapper.map(book, BookDTO.class);

}

}

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

private String name;

@JsonProperty(access = JsonProperty.Access.WRITE\_ONLY)

private String email;

}

curl -X GET http://localhost:8080/books

[

{

"id": 1,

"title": "Book Title",

"author": "Author Name",

"price": 19.99,

"isbn": "123-4567890123"

}

]

curl -X POST http://localhost:8080/books -H "Content-Type: application/json" -d '{"title":"New Book","author":"New Author","price":29.99,"isbn":"987-6543210987"}'

{

"id": 2,

"title": "New Book",

"author": "New Author",

"price": 29.99,

"isbn": "987-6543210987"

}

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

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import org.modelmapper.ModelMapper;

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

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@Autowired

private ModelMapper modelMapper;

// CREATE: Add a new book

@PostMapping

public BookDTO createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = modelMapper.map(bookDTO, Book.class);

book = bookRepository.save(book);

return modelMapper.map(book, BookDTO.class);

}

// READ: Get all books

@GetMapping

public List<BookDTO> getAllBooks() {

return bookRepository.findAll().stream()

.map(book -> modelMapper.map(book, BookDTO.class))

.collect(Collectors.toList());

}

// READ: Get a book by ID

@GetMapping("/{id}")

public BookDTO getBookById(@PathVariable Long id) {

Book book = bookRepository.findById(id)

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

return modelMapper.map(book, BookDTO.class);

}

// UPDATE: Update a book

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id)

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

modelMapper.map(bookDTO, book);

book = bookRepository.save(book);

return modelMapper.map(book, BookDTO.class);

}

// DELETE: Delete a book

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

Book book = bookRepository.findById(id)

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

bookRepository.delete(book);

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository;

import org.modelmapper.ModelMapper;

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

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

@Autowired

private ModelMapper modelMapper;

// CREATE: Add a new customer

@PostMapping

public CustomerDTO createCustomer(@Valid @RequestBody CustomerDTO customerDTO) {

Customer customer = modelMapper.map(customerDTO, Customer.class);

customer = customerRepository.save(customer);

return modelMapper.map(customer, CustomerDTO.class);

}

// READ: Get all customers

@GetMapping

public List<CustomerDTO> getAllCustomers() {

return customerRepository.findAll().stream()

.map(customer -> modelMapper.map(customer, CustomerDTO.class))

.collect(Collectors.toList());

}

// READ: Get a customer by ID

@GetMapping("/{id}")

public CustomerDTO getCustomerById(@PathVariable Long id) {

Customer customer = customerRepository.findById(id)

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

return modelMapper.map(customer, CustomerDTO.class);

}

// UPDATE: Update a customer

@PutMapping("/{id}")

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

Customer customer = customerRepository.findById(id)

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

modelMapper.map(customerDTO, customer);

customer = customerRepository.save(customer);

return modelMapper.map(customer, CustomerDTO.class);

}

// DELETE: Delete a customer

@DeleteMapping("/{id}")

public void deleteCustomer(@PathVariable Long id) {

Customer customer = customerRepository.findById(id)

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

customerRepository.delete(customer);

}

}

package com.example.bookstoreapi.dto;

import lombok.Data;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

@Data

public class BookDTO {

private Long id;

@NotNull(message = "Title is required")

@Size(min = 1, max = 100, message = "Title must be between 1 and 100 characters")

private String title;

@NotNull(message = "Author is required")

@Size(min = 1, max = 100, message = "Author must be between 1 and 100 characters")

private String author;

@Min(value = 0, message = "Price must be a positive number")

private double price;

@NotNull(message = "ISBN is required")

@Size(min = 10, max = 13, message = "ISBN must be between 10 and 13 characters")

private String isbn;

}

package com.example.bookstoreapi.model;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, length = 100)

private String title;

@Column(nullable = false, length = 100)

private String author;

@Column(nullable = false)

private double price;

@Column(nullable = false, length = 13)

private String isbn;

@Version

private Long version; // Optimistic locking field

}

package com.example.bookstoreapi.model;

import lombok.Data;

import javax.persistence.\*;

@Data

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, length = 100)

private String name;

@Column(nullable = false, unique = true)

private String email;

@Version

private Long version; // Optimistic locking field

}

curl -X POST http://localhost:8080/books -H "Content-Type: application/json" -d '{"title":"New Book","author":"New Author","price":29.99,"isbn":"9876543210987"}'

{

"id": 1,

"title": "New Book",

"author": "New Author",

"price": 29.99,

"isbn": "9876543210987"

}

curl -X PUT http://localhost:8080/books/1 -H "Content-Type: application/json" -d '{"id":1,"title":"Updated Book","author":"Updated Author","price":39.99,"isbn":"9876543210987","version":1}'

{

"id": 1,

"title": "Updated Book",

"author": "Updated Author",

"price": 39.99,

"isbn": "9876543210987",

"version": 2

}

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

<dependency>

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

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

</dependency>

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import org.modelmapper.ModelMapper;

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

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

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

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

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

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@Autowired

private ModelMapper modelMapper;

// CREATE: Add a new book

@PostMapping

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

Book book = modelMapper.map(bookDTO, Book.class);

book = bookRepository.save(book);

BookDTO responseDTO = modelMapper.map(book, BookDTO.class);

// Adding HATEOAS link

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel();

responseDTO.add(selfLink);

return EntityModel.of(responseDTO);

}

// READ: Get all books

@GetMapping

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

return bookRepository.findAll().stream()

.map(book -> {

BookDTO bookDTO = modelMapper.map(book, BookDTO.class);

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel();

bookDTO.add(selfLink);

return EntityModel.of(bookDTO);

})

.collect(Collectors.toList());

}

// READ: Get a book by ID

@GetMapping("/{id}")

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

Book book = bookRepository.findById(id)

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

BookDTO bookDTO = modelMapper.map(book, BookDTO.class);

// Adding HATEOAS links

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(BookController.class).getBookById(id)).withSelfRel();

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

bookDTO.add(selfLink, allBooksLink);

return EntityModel.of(bookDTO);

}

// DELETE: Delete a book

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

Book book = bookRepository.findById(id)

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

bookRepository.delete(book);

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository;

import org.modelmapper.ModelMapper;

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

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

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

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

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

@RestController

@RequestMapping("/customers")

public class CustomerController {

@Autowired

private CustomerRepository customerRepository;

@Autowired

private ModelMapper modelMapper;

// CREATE: Add a new customer

@PostMapping

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

Customer customer = modelMapper.map(customerDTO, Customer.class);

customer = customerRepository.save(customer);

CustomerDTO responseDTO = modelMapper.map(customer, CustomerDTO.class);

// Adding HATEOAS link

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(CustomerController.class).getCustomerById(customer.getId())).withSelfRel();

responseDTO.add(selfLink);

return EntityModel.of(responseDTO);

}

// READ: Get all customers

@GetMapping

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

return customerRepository.findAll().stream()

.map(customer -> {

CustomerDTO customerDTO = modelMapper.map(customer, CustomerDTO.class);

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(CustomerController.class).getCustomerById(customer.getId())).withSelfRel();

customerDTO.add(selfLink);

return EntityModel.of(customerDTO);

})

.collect(Collectors.toList());

}

// READ: Get a customer by ID

@GetMapping("/{id}")

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

Customer customer = customerRepository.findById(id)

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

CustomerDTO customerDTO = modelMapper.map(customer, CustomerDTO.class);

// Adding HATEOAS links

Link selfLink = WebMvcLinkBuilder.linkTo(methodOn(CustomerController.class).getCustomerById(id)).withSelfRel();

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

customerDTO.add(selfLink, allCustomersLink);

return EntityModel.of(customerDTO);

}

// DELETE: Delete a customer

@DeleteMapping("/{id}")

public void deleteCustomer(@PathVariable Long id) {

Customer customer = customerRepository.findById(id)

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

customerRepository.delete(customer);

}

}

curl -X GET http://localhost:8080/books/1

{

"id": 1,

"title": "Book Title",

"author": "Author Name",

"price": 19.99,

"isbn": "123-4567890123",

"\_links": {

"self": {

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

},

"all-books": {

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

}

}

}

curl -X GET http://localhost:8080/customers

[

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"\_links": {

"self": {

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

}

}

},

{

"id": 2,

"name": "Jane Doe",

"email": "jane.doe@example.com",

"\_links": {

"self": {

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

}

}

}

]

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

<dependency>

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

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

</dependency>

package com.example.bookstoreapi.config;

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(true) // Allows content type to be specified via a query parameter (optional)

.parameterName("mediaType") // The name of the query parameter

.ignoreAcceptHeader(false) // Use the Accept header for content negotiation

.useRegisteredExtensionsOnly(false)

.defaultContentType(org.springframework.http.MediaType.APPLICATION\_JSON) // Default to JSON if no type is specified

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

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

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import org.modelmapper.ModelMapper;

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

import org.springframework.http.MediaType;

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

import javax.validation.Valid;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@Autowired

private ModelMapper modelMapper;

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

public List<BookDTO> getAllBooks() {

return bookRepository.findAll().stream()

.map(book -> modelMapper.map(book, BookDTO.class))

.collect(Collectors.toList());

}

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

public BookDTO getBookById(@PathVariable Long id) {

Book book = bookRepository.findById(id)

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

return modelMapper.map(book, BookDTO.class);

}

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

public BookDTO createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = modelMapper.map(bookDTO, Book.class);

book = bookRepository.save(book);

return modelMapper.map(book, BookDTO.class);

}

}

curl -X GET http://localhost:8080/books -H "Accept: application/json"

[

{

"id": 1,

"title": "Book Title",

"author": "Author Name",

"price": 19.99,

"isbn": "123-4567890123"

},

...

]

curl -X GET http://localhost:8080/books -H "Accept: application/xml"

<List>

<BookDTO>

<id>1</id>

<title>Book Title</title>

<author>Author Name</author>

<price>19.99</price>

<isbn>123-4567890123</isbn>

</BookDTO>

...

</List>

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

<dependency>

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

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

</dependency>

# Enable all Actuator endpoints

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

# Customize the path for Actuator endpoints (default is /actuator)

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

# Enable security for Actuator endpoints (optional)

management.endpoint.health.show-details=always

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

curl -X GET http://localhost:8080/management/health

curl -X GET http://localhost:8080/management/info

curl -X GET <http://localhost:8080/management/metrics>

package com.example.bookstoreapi.metrics;

import io.micrometer.core.instrument.MeterRegistry;

import io.micrometer.core.instrument.Tag;

import io.micrometer.core.instrument.binder.MeterBinder;

import org.springframework.stereotype.Component;

@Component

public class CustomMetrics implements MeterBinder {

@Override

public void bindTo(MeterRegistry meterRegistry) {

meterRegistry.gauge("custom.metric.book.count", Tag.of("type", "book"), 42); // Example custom metric

}

}

curl -X GET http://localhost:8080/management/metrics/custom.metric.book.count

{

"name": "custom.metric.book.count",

"measurements": [

{

"statistic": "VALUE",

"value": 42

}

],

"availableTags": [

{

"tag": "type",

"values": [

"book"

]

}

]

}

package com.example.bookstoreapi.endpoint;

import org.springframework.boot.actuate.endpoint.annotation.Endpoint;

import org.springframework.boot.actuate.endpoint.annotation.ReadOperation;

import org.springframework.stereotype.Component;

@Component

@Endpoint(id = "custom")

public class CustomEndpoint {

@ReadOperation

public String custom() {

return "This is a custom actuator endpoint";

}

}

curl -X GET http://localhost:8080/management/custom

This is a custom actuator endpoint

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

<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.config;

import com.example.bookstoreapi.security.JwtAuthenticationFilter;

import com.example.bookstoreapi.security.JwtAuthorizationFilter;

import com.example.bookstoreapi.security.UserDetailsServiceImpl;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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.builders.AuthenticationManagerBuilder;

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.crypto.bcrypt.BCryptPasswordEncoder;

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

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

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailsServiceImpl userDetailsService;

@Autowired

private JwtAuthenticationFilter jwtAuthenticationFilter;

@Autowired

private JwtAuthorizationFilter jwtAuthorizationFilter;

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

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

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)

.addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class);

// CORS configuration

http.cors();

}

@Bean

public CorsConfigurationSource corsConfigurationSource() {

CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000")); // Adjust as needed

configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));

configuration.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));

configuration.setAllowCredentials(true);

UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", configuration);

return source;

}

}

package com.example.bookstoreapi.security;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

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

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtTokenProvider {

@Value("${jwt.secret}")

private String secretKey;

@Value("${jwt.expiration}")

private long expirationTime;

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setExpiration(new Date(System.currentTimeMillis() + expirationTime))

.signWith(SignatureAlgorithm.HS512, secretKey)

.compact();

}

public String getUsername(String token) {

return Jwts.parser()

.setSigningKey(secretKey)

.parseClaimsJws(token)

.getBody()

.getSubject();

}

public boolean validateToken(String token) {

try {

Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

}

package com.example.bookstoreapi.security;

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

import org.springframework.web.util.WebUtils;

import javax.servlet.FilterChain;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

@Autowired

private JwtTokenProvider jwtTokenProvider;

@Override

public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {

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

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

token = token.substring(7);

if (jwtTokenProvider.validateToken(token)) {

String username = jwtTokenProvider.getUsername(token);

if (username != null) {

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(username, null, null);

SecurityContextHolder.getContext().setAuthentication(authentication);

}

}

}

chain.doFilter(request, response);

}

}

package com.example.bookstoreapi.security;

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

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.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class JwtAuthorizationFilter extends UsernamePasswordAuthenticationFilter {

@Autowired

private JwtTokenProvider jwtTokenProvider;

@Override

public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {

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

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

token = token.substring(7);

if (jwtTokenProvider.validateToken(token)) {

String username = jwtTokenProvider.getUsername(token);

if (username != null) {

// Add authentication to the security context

SecurityContextHolder.getContext().setAuthentication(

new UsernamePasswordAuthenticationToken(username, null, null)

);

}

}

}

chain.doFilter(request, response);

}

}

package com.example.bookstoreapi.security;

import com.example.bookstoreapi.model.User;

import com.example.bookstoreapi.repository.UserRepository;

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.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

@Autowired

private UserRepository userRepository;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), new ArrayList<>());

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.security.JwtTokenProvider;

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

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

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private JwtTokenProvider jwtTokenProvider;

@PostMapping("/login")

public String login(@RequestParam String username, @RequestParam String password) {

// Authenticate user and generate token (pseudo-code)

String token = jwtTokenProvider.generateToken(username);

return token;

}

}

curl -X GET http://localhost:8080/books -H "Authorization: Bearer <your-jwt-token>"

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

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.7.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.service.BookService;

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.http.MediaType;

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

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

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

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

import org.springframework.test.web.servlet.result.MockMvcResultHandlers;

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)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@Mock

private BookService bookService;

@InjectMocks

private BookController bookController;

@BeforeEach

public void setUp() {

MockitoAnnotations.openMocks(this);

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

}

@Test

public void testGetAllBooks() throws Exception {

when(bookService.getAllBooks()).thenReturn(List.of(new BookDTO(1L, "Title", "Author", 19.99, "123-4567890123")));

mockMvc.perform(get("/books")

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

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

.andDo(MockMvcResultHandlers.print());

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

}

@Test

public void testGetBookById() throws Exception {

when(bookService.getBookById(1L)).thenReturn(new BookDTO(1L, "Title", "Author", 19.99, "123-4567890123"));

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

.andExpect(jsonPath("$.title").value("Title"))

.andDo(MockMvcResultHandlers.print());

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

}

@Test

public void testCreateBook() throws Exception {

BookDTO bookDTO = new BookDTO(1L, "New Title", "New Author", 29.99, "987-6543210987");

when(bookService.createBook(any(BookDTO.class))).thenReturn(bookDTO);

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"New Title\", \"author\": \"New Author\", \"price\": 29.99, \"isbn\": \"987-6543210987\"}"))

.andExpect(status().isCreated())

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

.andExpect(jsonPath("$.title").value("New Title"))

.andDo(MockMvcResultHandlers.print());

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

}

@Test

public void testUpdateBook() throws Exception {

BookDTO bookDTO = new BookDTO(1L, "Updated Title", "Updated Author", 24.99, "123-4567890123");

when(bookService.updateBook(eq(1L), any(BookDTO.class))).thenReturn(bookDTO);

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

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"title\": \"Updated Title\", \"author\": \"Updated Author\", \"price\": 24.99, \"isbn\": \"123-4567890123\"}"))

.andExpect(status().isOk())

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

.andExpect(jsonPath("$.title").value("Updated Title"))

.andDo(MockMvcResultHandlers.print());

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

}

@Test

public void testDeleteBook() throws Exception {

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

.andExpect(status().isNoContent())

.andDo(MockMvcResultHandlers.print());

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

}

}

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

<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>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

# application-test.properties

# H2 Database Configuration

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.jpa.hibernate.ddl-auto=create-drop

# Enable H2 Console

spring.h2.console.enabled=true

package com.example.bookstoreapi;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

import com.example.bookstoreapi.service.BookService;

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.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;

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

import org.springframework.test.web.servlet.annotation.WebMvcTest;

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

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

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

import org.springframework.transaction.annotation.Transactional;

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

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

@SpringBootTest

@AutoConfigureMockMvc

@Transactional

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@BeforeEach

public void setUp() {

bookRepository.deleteAll(); // Ensure the database is clean before each test

}

@Test

public void testCreateBook() throws Exception {

String bookJson = "{\"title\": \"New Title\", \"author\": \"New Author\", \"price\": 29.99, \"isbn\": \"987-6543210987\"}";

mockMvc.perform(post("/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(bookJson))

.andExpect(status().isCreated())

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

.andExpect(jsonPath("$.title").value("New Title"))

.andExpect(jsonPath("$.author").value("New Author"))

.andExpect(jsonPath("$.price").value(29.99))

.andExpect(jsonPath("$.isbn").value("987-6543210987"));

}

@Test

public void testGetBookById() throws Exception {

Book book = new Book(null, "Title", "Author", 19.99, "123-4567890123");

bookRepository.save(book);

mockMvc.perform(get("/books/" + book.getId())

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

.andExpect(jsonPath("$.title").value("Title"))

.andExpect(jsonPath("$.author").value("Author"))

.andExpect(jsonPath("$.price").value(19.99))

.andExpect(jsonPath("$.isbn").value("123-4567890123"));

}

@Test

public void testUpdateBook() throws Exception {

Book book = new Book(null, "Old Title", "Old Author", 15.99, "321-6549876543");

Book savedBook = bookRepository.save(book);

String updatedBookJson = "{\"title\": \"Updated Title\", \"author\": \"Updated Author\", \"price\": 24.99, \"isbn\": \"321-6549876543\"}";

mockMvc.perform(put("/books/" + savedBook.getId())

.contentType(MediaType.APPLICATION\_JSON)

.content(updatedBookJson))

.andExpect(status().isOk())

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

.andExpect(jsonPath("$.title").value("Updated Title"))

.andExpect(jsonPath("$.author").value("Updated Author"))

.andExpect(jsonPath("$.price").value(24.99))

.andExpect(jsonPath("$.isbn").value("321-6549876543"));

}

@Test

public void testDeleteBook() throws Exception {

Book book = new Book(null, "Title to Delete", "Author", 20.00, "123-9876543210");

Book savedBook = bookRepository.save(book);

mockMvc.perform(delete("/books/" + savedBook.getId()))

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

mockMvc.perform(get("/books/" + savedBook.getId()))

.andExpect(status().isNotFound());

}

}

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

<dependency>

<groupId>org.springdoc</groupId>

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

<version>2.2.0</version>

</dependency>

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.service.BookService;

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.responses.ApiResponses;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) {

this.bookService = bookService;

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved list of books"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

@GetMapping

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

List<BookDTO> books = bookService.getAllBooks();

return ResponseEntity.ok(books);

}

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

@ApiResponses(value = {

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

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

})

@GetMapping("/{id}")

public ResponseEntity<BookDTO> getBookById(

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

BookDTO book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Book successfully created"),

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

})

@PostMapping

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

BookDTO createdBook = bookService.createBook(bookDTO);

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

}

@Operation(summary = "Update an existing book", description = "Modify an existing book's details")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Book successfully updated"),

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

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

})

@PutMapping("/{id}")

public ResponseEntity<BookDTO> updateBook(

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

@RequestBody BookDTO bookDTO) {

BookDTO updatedBook = bookService.updateBook(id, bookDTO);

return ResponseEntity.ok(updatedBook);

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "204", description = "Book successfully deleted"),

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

})

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.service.CustomerService;

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.responses.ApiResponses;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/customers")

public class CustomerController {

private final CustomerService customerService;

public CustomerController(CustomerService customerService) {

this.customerService = customerService;

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Successfully retrieved list of customers"),

@ApiResponse(responseCode = "500", description = "Internal server error")

})

@GetMapping

public ResponseEntity<List<CustomerDTO>> getAllCustomers() {

List<CustomerDTO> customers = customerService.getAllCustomers();

return ResponseEntity.ok(customers);

}

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

@ApiResponses(value = {

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

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

})

@GetMapping("/{id}")

public ResponseEntity<CustomerDTO> getCustomerById(

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

CustomerDTO customer = customerService.getCustomerById(id);

return ResponseEntity.ok(customer);

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "201", description = "Customer successfully created"),

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

})

@PostMapping

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

CustomerDTO createdCustomer = customerService.createCustomer(customerDTO);

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

}

@Operation(summary = "Update an existing customer", description = "Modify an existing customer's details")

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Customer successfully updated"),

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

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

})

@PutMapping("/{id}")

public ResponseEntity<CustomerDTO> updateCustomer(

@Parameter(description = "ID of the customer to be updated") @PathVariable Long id,

@RequestBody CustomerDTO customerDTO) {

CustomerDTO updatedCustomer = customerService.updateCustomer(id, customerDTO);

return ResponseEntity.ok(updatedCustomer);

}

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

@ApiResponses(value = {

@ApiResponse(responseCode = "204", description = "Customer successfully deleted"),

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

})

@DeleteMapping("/{id}")

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

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}