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

**Business Scenario:**

Customize the HTTP response status and headers for the book management endpoints.

Customizing Response Status with @ResponseStatus:

@ResponseStatus(HttpStatus.CREATED)

@PostMapping("/books")

public Book createBook(@RequestBody Book book) {

books.add(book);

return book;

}

Implementation of @ResponseStatus:

@ResponseStatus(HttpStatus.NOT\_FOUND)

@GetMapping("/books/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream()

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

.findFirst()

.orElseThrow(() -> new BookNotFoundException(id));

}

Adding Custom Headers with ResponseEntity:

@PostMapping("/books")

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

books.add(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "Created a new book");

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

}

Implementation with custom headers:

@GetMapping("/books/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> {

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "Fetched book by ID");

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

})

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

Complete BookController class:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.model.Book;

import com.yourname.bookstoreapi.exception.BookNotFoundException;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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<>();

// Sample data

public BookController() {

books.add(new Book(1L, "Book One", "Author One", 9.99, "1234567890"));

books.add(new Book(2L, "Book Two", "Author Two", 14.99, "0987654321"));

}

// GET /books/{id} - Get a book by ID with custom status and headers

@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> {

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "Fetched book by ID");

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

})

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

// POST /books - Create a new book with custom status and headers

@ResponseStatus(HttpStatus.CREATED)

@PostMapping

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

books.add(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "Created a new book");

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

}

// Other endpoints (e.g., PUT, DELETE) with similar customization...

}

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

**Business Scenario:**

Implement a global exception handling mechanism for the bookstore RESTful services.

GlobalExceptionHandler Class:

package com.yourname.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.context.request.WebRequest;

@ControllerAdvice

public class GlobalExceptionHandler {

// Handle specific exceptions like BookNotFoundException

@ExceptionHandler(BookNotFoundException.class)

public ResponseEntity<String> handleBookNotFoundException(BookNotFoundException ex, WebRequest request) {

return new ResponseEntity<>("Book not found: " + ex.getMessage(), HttpStatus.NOT\_FOUND);

}

// Handle other exceptions (e.g., validation errors)

@ExceptionHandler(IllegalArgumentException.class)

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

return new ResponseEntity<>("Invalid input: " + ex.getMessage(), HttpStatus.BAD\_REQUEST);

}

// Handle all other exceptions

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGlobalException(Exception ex, WebRequest request) {

return new ResponseEntity<>("An error occurred: " + ex.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

Custom Exception Class:

package com.yourname.bookstoreapi.exception;

public class BookNotFoundException extends RuntimeException {

public BookNotFoundException(Long id) {

super("Book with ID " + id + " not found.");

}

}

Integrating the Exception Handling with Controllers:

GlobalExceptionHandler Class:

package com.yourname.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.context.request.WebRequest;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(BookNotFoundException.class)

public ResponseEntity<String> handleBookNotFoundException(BookNotFoundException ex, WebRequest request) {

return new ResponseEntity<>("Book not found: " + ex.getMessage(), HttpStatus.NOT\_FOUND);

}

@ExceptionHandler(IllegalArgumentException.class)

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

return new ResponseEntity<>("Invalid input: " + ex.getMessage(), HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGlobalException(Exception ex, WebRequest request) {

return new ResponseEntity<>("An error occurred: " + ex.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

BookNotFoundException Class:

package com.yourname.bookstoreapi.exception;

public class BookNotFoundException extends RuntimeException {

public BookNotFoundException(Long id) {

super("Book with ID " + id + " not found.");

}

}

Updated BookController:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.exception.BookNotFoundException;

import com.yourname.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 java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

public BookController() {

books.add(new Book(1L, "Book One", "Author One", 9.99, "1234567890"));

books.add(new Book(2L, "Book Two", "Author Two", 14.99, "0987654321"));

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return books.stream()

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

.findFirst()

.orElseThrow(() -> new BookNotFoundException(id));

}

@PostMapping

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

books.add(book);

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "Created a new book");

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

}

}

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

**Business Scenario:**

Use DTOs to transfer data between the client and server for books and customers.

BookDTO:

package com.yourname.bookstoreapi.dto;

public class BookDTO {

private Long id;

private String title;

private String author;

private Double price;

// Constructors, getters, and setters

public BookDTO() {}

public BookDTO(Long id, String title, String author, Double price) {

this.id = id;

this.title = title;

this.author = author;

this.price = price;

}

// Getters and setters...

}

CustomerDTO:

package com.yourname.bookstoreapi.dto;

public class CustomerDTO {

private Long id;

private String name;

private String email;

// Constructors, getters, and setters

public CustomerDTO() {}

public CustomerDTO(Long id, String name, String email) {

this.id = id;

this.name = name;

this.email = email;

}

// Getters and setters...

}

Mapping Entities to DTOs:

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.2.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.2.Final</version>

<scope>provided</scope>

</dependency>

BookMapper

package com.yourname.bookstoreapi.mapper;

import com.yourname.bookstoreapi.dto.BookDTO;

import com.yourname.bookstoreapi.model.Book;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

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

BookDTO toBookDTO(Book book);

Book toBook(BookDTO bookDTO);

}

CustomerMapper:

package com.yourname.bookstoreapi.mapper;

import com.yourname.bookstoreapi.dto.CustomerDTO;

import com.yourname.bookstoreapi.model.Customer;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {

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

CustomerDTO toCustomerDTO(Customer customer);

Customer toCustomer(CustomerDTO customerDTO);

}

Custom Serialization/Deserialization with Jackson:

**Customizing BookDTO Serialization**

package com.yourname.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

public class BookDTO {

@JsonIgnore

private Long id;

private String title;

private String author;

private Double price;

// Constructors, getters, and setters

}

Customizing CustomerDTO Deserialization

package com.yourname.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonSetter;

public class CustomerDTO {

private Long id;

private String name;

private String email;

@JsonSetter("fullName")

public void setName(String name) {

this.name = name;

}

// Constructors, getters, and setters

}

Integrating DTOs into Controllers:

package com.yourname.bookstoreapi.controller;

import com.yourname.bookstoreapi.dto.BookDTO;

import com.yourname.bookstoreapi.mapper.BookMapper;

import com.yourname.bookstoreapi.model.Book;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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<>();

@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> new ResponseEntity<>(BookMapper.INSTANCE.toBookDTO(book), HttpStatus.OK))

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

@PostMapping

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

Book book = BookMapper.INSTANCE.toBook(bookDTO);

books.add(book);

return new ResponseEntity<>(BookMapper.INSTANCE.toBookDTO(book), HttpStatus.CREATED);

}

}