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

#### Setup Spring Boot Project:

BookstoreApiApplication.java

package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApiApplication { public static void main(String[] args) {

SpringApplication.run(BookstoreApiApplication.class, args);

}

}

BookController.java

package com.example.bookstoreapi.controller;

import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import lombok.Getter;

import lombok.Setter; import lombok.ToString; import java.util.List;

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

@GetMapping

public List<Book> getBooks() {

// This is just a placeholder. In a real application, you'd fetch data from a database. return List.of(

new Book(1L, "Effective Java", "Joshua Bloch"),

new Book(2L, "Spring Boot in Action", "Craig Walls")

);

}

}

**Book.java** (Inside controller package) package com.example.bookstoreapi.controller;

import lombok.Getter;

import lombok.Setter; import lombok.ToString;

@Getter @Setter @ToString

public class Book { private Long id; private String title; private String author;

public Book(Long id, String title, String author) { this.id = id;

this.title = title; this.author = author;

}

}

**pom.xml** (Dependencies section)

<dependencies>

<!-- Spring Boot Starter Web for building web, including RESTful, applications using Spring MVC -->

<dependency>

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

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

</dependency>

<!-- Spring Boot DevTools for automatic restart and live reload -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- Lombok for reducing boilerplate code -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.24</version>

<scope>provided</scope>

</dependency>

</dependencies>

**Project Structure:**

**BookstoreAPI/**

##### │

**├── src/**

##### │ ├── main/

**│ │ ├── java/**

##### │ │ │ └── com/

**│ │ │ └── example/**

##### │ │ │ └── bookstoreapi/

**│ │ │ ├── BookstoreApiApplication.java**

##### │ │ │ └── controller/

**│ │ │ └── BookController.java**

##### │ │ └── resources/

**│ │ ├── application.properties**

##### │ │ └── static/

**│ └── test/**

##### │ └── java/

**│ └── com/**

##### │ └── example/

**│ └── bookstoreapi/**

##### │ └── BookstoreApiApplicationTests.java

**└── pom.xml**

1. **What's New in Spring Boot 3:**

Spring Boot 3 introduced several new features and improvements. Here’s an overview of the key enhancements and updates:

##### Java 21 Support

Spring Boot 3 requires Java 21 or later, aligning with the latest Java LTS (Long-Term Support) version. This update allows leveraging new language features and performance improvements in Java.

##### Spring Framework 6 Integration

Spring Boot 3 is built on Spring Framework 6, which includes:

* + **Jakarta EE 9+ Support**: Migrates from javax.\* to jakarta.\* namespaces.
  + **Native Compilation Support**: Enhanced support for GraalVM native images.
  + **Enhanced Configuration**: Improvements in the configuration and property binding.

##### Enhanced Observability

* + **Micrometer Integration**: Improved integration with Micrometer for metrics and monitoring.
  + **Observability Support**: Enhanced support for distributed tracing, metrics, and logging.

##### Improved Dependency Management

* + **Dependency Upgrades**: Updated versions of key dependencies to improve security and performance.
  + **Simplified Dependency Resolution**: Enhanced dependency management with clearer dependency versions.

##### Native Compilation Support

* + **GraalVM Native Image**: Enhanced support for GraalVM to build native executables, improving startup times and reducing memory consumption.

# Exercise 2: Online Bookstore - Creating Basic REST Controllers

1. **Define the Book Entity**

package com.example.bookstore.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;

@Entity

public class Book {

@Id

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

private String title; private String author; private Double price; private String isbn;

// Getters and Setters public Long getId() {

return id;

}

public void setId(Long id) { this.id = id;

}

public String getTitle() { return title;

}

public void setTitle(String title) { this.title = title;

}

public String getAuthor() { return author;

}

public void setAuthor(String author) { this.author = author;

}

public Double getPrice() { return price;

}

public void setPrice(Double price) { this.price = price;

}

public String getIsbn() { return isbn;

}

public void setIsbn(String isbn) { this.isbn = isbn;

}

}

1. **Create the BookRepository**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

1. **Implement the BookController**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.http.HttpStatus;

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

import java.util.List; import java.util.Optional;

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

@Autowired

private BookRepository bookRepository;

// GET all books

@GetMapping

public List<Book> getAllBooks() { return bookRepository.findAll();

}

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

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

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

// POST a new book @PostMapping

public ResponseEntity<Book> createBook(@RequestBody Book book) { Book savedBook = bookRepository.save(book);

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

}

// PUT update an existing book @PutMapping("/{id}")

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

Optional<Book> existingBook = bookRepository.findById(id); if (existingBook.isPresent()) {

Book book = existingBook.get(); book.setTitle(bookDetails.getTitle()); book.setAuthor(bookDetails.getAuthor());

book.setPrice(bookDetails.getPrice()); book.setIsbn(bookDetails.getIsbn()); bookRepository.save(book);

return ResponseEntity.ok(book);

}

return ResponseEntity.notFound().build();

}

// 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();

}

return ResponseEntity.notFound().build();

}

}

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

##### Fetch a Book by Its ID Using a Path Variable

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

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

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

##### Filter Books Based on Query Parameters

// GET books with optional query parameters @GetMapping("/search")

public List<Book> searchBooks(

@RequestParam(value = "title", required = false) String title, @RequestParam(value = "author", required = false) String author) {

if (title != null && author != null) {

return bookRepository.findByTitleAndAuthor(title, author);

} else if (title != null) {

return bookRepository.findByTitle(title);

} else if (author != null) {

return bookRepository.findByAuthor(author);

} else {

return bookRepository.findAll();

}

}

1. **Update the BookRepository with Custom Queries**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByTitle(String title);

List<Book> findByAuthor(String author);

List<Book> findByTitleAndAuthor(String title, String author);

}

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

1. **Define the Customer Entity**

package com.example.bookstoreapi.model;

import lombok.AllArgsConstructor; import lombok.Getter;

import lombok.NoArgsConstructor; import lombok.Setter;

import lombok.ToString;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;

@Entity @Getter @Setter

@NoArgsConstructor @AllArgsConstructor @ToString

public class Customer { @Id

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

private String name; private String email; private String phone; private String address;

}

1. **Create the CustomerRepository Interface**

package com.example.bookstoreapi.repository;

import com.example.bookstoreapi.model.Customer;

import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.stereotype.Repository;

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

1. **Create the CustomerController**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Customer;

import com.example.bookstoreapi.repository.CustomerRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.http.HttpStatus;

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

import javax.servlet.http.HttpServletRequest; import java.util.Optional;

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

@Autowired

private CustomerRepository customerRepository;

// Create a new customer using JSON request body @PostMapping("/json")

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) { Customer savedCustomer = customerRepository.save(customer);

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

}

// Register a new customer using form data @PostMapping("/form")

public ResponseEntity<Customer> registerCustomer(HttpServletRequest request) { String name = request.getParameter("name");

String email = request.getParameter("email"); String phone = request.getParameter("phone"); String address = request.getParameter("address");

Customer customer = new Customer(); customer.setName(name); customer.setEmail(email); customer.setPhone(phone); customer.setAddress(address);

Customer savedCustomer = customerRepository.save(customer);

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

}

}

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

1. **Customizing HTTP Status Codes with @ResponseStatus**

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

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

@Autowired

private BookRepository bookRepository;

// GET a single book by ID with custom status @GetMapping("/{id}")

@ResponseStatus(HttpStatus.OK) // Default status code for successful retrieval public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id)

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

}

// POST a new book with custom status @PostMapping

@ResponseStatus(HttpStatus.CREATED) // Default status code for resource creation public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

// PUT update an existing book with custom status @PutMapping("/{id}")

@ResponseStatus(HttpStatus.OK) // Default status code for successful update

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) { Book book = bookRepository.findById(id)

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

book.setAuthor(bookDetails.getAuthor()); book.setPrice(bookDetails.getPrice()); book.setIsbn(bookDetails.getIsbn()); return bookRepository.save(book);

}

// DELETE a book by ID with custom status @DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.NO\_CONTENT) // Default status code for successful deletion with no content

public void deleteBook(@PathVariable Long id) { if (bookRepository.existsById(id)) {

bookRepository.deleteById(id);

} else {

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

}

}

}

##### Custom Exception Handling

// Custom exception class @ResponseStatus(HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException { public ResourceNotFoundException(String message) {

super(message);

}

}

// Global exception handler @ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class) @ResponseStatus(HttpStatus.NOT\_FOUND)

public ResponseEntity<String> handleResourceNotFoundException(ResourceNotFoundException ex) {

return ResponseEntity

.status(HttpStatus.NOT\_FOUND)

.body(ex.getMessage());

}

}

1. **Adding Custom Headers Using ResponseEntity**

import org.springframework.http.HttpHeaders; import org.springframework.http.ResponseEntity;

// GET all books with custom headers @GetMapping

public ResponseEntity<List<Book>> getAllBooks() { List<Book> books = bookRepository.findAll(); HttpHeaders headers = new HttpHeaders(); headers.add("Custom-Header", "HeaderValue");

return new ResponseEntity<>(books, headers, HttpStatus.OK); // HTTP 200 OK with custom headers

}

// POST a new book with custom headers @PostMapping

public ResponseEntity<Book> createBook(@RequestBody Book book) { Book savedBook = bookRepository.save(book);

HttpHeaders headers = new HttpHeaders(); headers.add("Location", "/books/" + savedBook.getId());

return new ResponseEntity<>(savedBook, headers, HttpStatus.CREATED); // HTTP 201 Created with Location header

}

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

##### Deﬁne the Global Exception Handler

package com.example.bookstore.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.bind.annotation.ResponseStatus;

import org.springframework.web.bind.MethodArgumentNotValidException; import org.springframework.web.bind.annotation.ResponseBody;

import javax.validation.ConstraintViolationException;

// Global Exception Handler @ControllerAdvice

public class GlobalExceptionHandler {

// Handle resource not found exceptions @ExceptionHandler(ResourceNotFoundException.class) @ResponseStatus(HttpStatus.NOT\_FOUND)

@ResponseBody

public ResponseEntity<ErrorResponse> handleResourceNotFoundException(ResourceNotFoundException ex) {

ErrorResponse errorResponse = new ErrorResponse("Resource Not Found", ex.getMessage());

return new ResponseEntity<>(errorResponse, HttpStatus.NOT\_FOUND);

}

// Handle validation exceptions @ExceptionHandler(MethodArgumentNotValidException.class) @ResponseStatus(HttpStatus.BAD\_REQUEST)

@ResponseBody

public ResponseEntity<ErrorResponse> handleValidationException(MethodArgumentNotValidException ex) {

String errorMessage = ex.getBindingResult().getAllErrors().stream()

.map(error -> error.getDefaultMessage())

.reduce((message1, message2) -> message1 + ", " + message2)

.orElse("Validation error");

ErrorResponse errorResponse = new ErrorResponse("Validation Error", errorMessage);

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

}

// Handle constraint violations (e.g., @Valid constraints) @ExceptionHandler(ConstraintViolationException.class) @ResponseStatus(HttpStatus.BAD\_REQUEST)

@ResponseBody

public ResponseEntity<ErrorResponse> handleConstraintViolationException(ConstraintViolationException ex) {

String errorMessage = ex.getConstraintViolations().stream()

.map(violation -> violation.getMessage())

.reduce((message1, message2) -> message1 + ", " + message2)

.orElse("Constraint violation");

ErrorResponse errorResponse = new ErrorResponse("Constraint Violation", errorMessage);

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

}

// Handle any other exceptions @ExceptionHandler(Exception.class) @ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR) @ResponseBody

public ResponseEntity<ErrorResponse> handleGenericException(Exception ex) {

ErrorResponse errorResponse = new ErrorResponse("Internal Server Error", ex.getMessage());

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

}

}

1. **Define the ErrorResponse Class**

package com.example.bookstore.exception;

public class ErrorResponse {

private String error; private String message;

public ErrorResponse(String error, String message) { this.error = error;

this.message = message;

}

// Getters and Setters public String getError() {

return error;

}

public void setError(String error) { this.error = error;

}

public String getMessage() { return message;

}

public void setMessage(String message) { this.message = message;

}

}

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

##### Define DTO Classes

package com.example.bookstore.dto;

public class BookDTO {

private Long id; private String title; private String author; private Double price; private String isbn;

// Constructors public BookDTO() {

}

public BookDTO(Long id, String title, String author, Double price, String isbn) { this.id = id;

this.title = title; this.author = author; this.price = price; this.isbn = isbn;

}

// Getters and Setters public Long getId() {

return id;

}

public void setId(Long id) { this.id = id;

}

public String getTitle() { return title;

}

public void setTitle(String title) { this.title = title;

}

public String getAuthor() { return author;

}

public void setAuthor(String author) { this.author = author;

}

public Double getPrice() { return price;

}

public void setPrice(Double price) { this.price = price;

}

public String getIsbn() { return isbn;

}

public void setIsbn(String isbn) { this.isbn = isbn;

}

}

###### CustomerDTO

package com.example.bookstore.dto;

public class CustomerDTO {

private Long id; private String name; private String email;

private String phoneNumber;

// Constructors

public CustomerDTO() {

}

public CustomerDTO(Long id, String name, String email, String phoneNumber) { this.id = id;

this.name = name;

this.email = email; this.phoneNumber = phoneNumber;

}

// Getters and Setters public Long getId() {

return id;

}

public void setId(Long id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

public String getPhoneNumber() { return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber;

}

}

###### Mapping Entities to DTOs Using MapStruct or ModelMapper

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.5.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.5.Final</version>

<scope>provided</scope>

</dependency>

###### Create Mapper Interfaces

package com.example.bookstore.mapper;

import com.example.bookstore.dto.BookDTO; import com.example.bookstore.model.Book; import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

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

BookDTO bookToBookDTO(Book book);

Book bookDTOToBook(BookDTO bookDTO);

}

package com.example.bookstore.mapper;

import com.example.bookstore.dto.CustomerDTO; import com.example.bookstore.model.Customer; import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {package com.example.bookstore.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();

}

}

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

CustomerDTO customerToCustomerDTO(Customer customer);

Customer customerDTOToCustomer(CustomerDTO customerDTO);

}

###### Using ModelMapper

<dependency>

<groupId>org.modelmapper</groupId>

<artifactId>modelmapper</artifactId>

<version>3.1.1</version>

</dependency>

###### Configure ModelMapper

package com.example.bookstore.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();

}

}

###### Use ModelMapper for Mapping

import org.modelmapper.ModelMapper;

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

@Service

public class BookService {

@Autowired

private ModelMapper modelMapper;

public BookDTO convertToDto(Book book) { return modelMapper.map(book, BookDTO.class);

}

public Book convertToEntity(BookDTO bookDTO) { return modelMapper.map(bookDTO, Book.class);

}

}

##### Custom Serialization/Deserialization Using Jackson Annotations

package com.example.bookstore.dto;

import com.fasterxml.jackson.annotation.JsonFormat; import com.fasterxml.jackson.annotation.JsonProperty;

public class BookDTO {

private Long id;

@JsonProperty("book\_title") private String title;

private String author;

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "$#.00") private Double price;

private String isbn;

// Constructors, Getters, and Setters

}