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

#### **1.1 Setup Spring Boot Project:**

**Project Structure:** The generated project structure should look like this:  
plaintext  
   
bookstoreapi/

├── src/

│ ├── main/

│ │ ├── /

│ │ │ └── com/

│ │ │ └── example/

│ │ │ └── bookstoreapi/

│ │ │ └── BookstoreApiApplication.

│ │ └── resources/

│ │ ├── application.properties

│ └── test/

│ ├── /

│ │ └── com/

│ │ └── example/

│ │ └── bookstoreapi/

│ │ └── BookstoreApiApplicationTests.

└── pom.xml

#### **1.2 What's New in Spring Boot 3:**

* **Documenting New Features in Spring Boot 3:**
  + Improved support for GraalVM native images.
  + Enhanced observability and metrics support.
  + New configurations for HTTP/2.
  + Upgraded dependency versions for Spring Framework, Spring Data, Spring Security, etc.
  + Full support for Jakarta EE 9.

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

#### **2.1 Create Book Controller:**

**Book Entity:**   
   
package com.example.bookstoreapi.entity;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import x.persistence.Entity;

import x.persistence.GeneratedValue;

import x.persistence.GenerationType;

import x.persistence.Id;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

**BookController Class:**   
   
package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Book;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import .util.ArrayList;

import .util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

@GetMapping

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

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

}

@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

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

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

}

@PostMapping

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

books.add(book);

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

}

@PutMapping("/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

@DeleteMapping("/{id}")

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

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

if (removed) {

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

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

}

}

#### **2.2 Return JSON Responses:**

* The controller above ensures that the responses are in JSON format using Spring Boot's default message converters.

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

#### **3.1 Path Variables:**

**Fetch Book by ID:**   
   
@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

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

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

}

#### **3.2 Query Parameters:**

**Filter Books by Title and Author:**   
   
@GetMapping("/search")

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

@RequestParam(required = false) String author) {

List<Book> filteredBooks = books.stream()

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

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

.toList();

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

}

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

#### **4.1 Request Body:**

**Create a New Customer:**   
   
@PostMapping("/customers")

public ResponseEntity<Customer> createCustomer(@RequestBody Customer customer) {

// Save the customer to a database or list

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

}

#### **4.2 Form Data:**

**Process Form Data:**   
   
@PostMapping("/customers/register")

public ResponseEntity<Customer> registerCustomer(@RequestParam String name,

@RequestParam String email) {

Customer customer = new Customer(name, email);

// Save the customer

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

}

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

#### **5.1 Response Status:**

**Custom Status Codes:**   
   
@ResponseStatus(HttpStatus.CREATED)

@PostMapping("/books")

public Book createBook(@RequestBody Book book) {

books.add(book);

return book;

}

#### **5.2 Custom Headers:**

**Add Custom Headers:**   
   
@GetMapping("/books/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> ResponseEntity.ok()

.header("Custom-Header", "CustomValue")

.body(book))

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

}

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

#### **6.1 Global Exception Handler:**

**GlobalExceptionHandler Class:**   
   
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;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<Object> handleResourceNotFoundException(ResourceNotFoundException ex) {

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

}

@ExceptionHandler(Exception.class)

public ResponseEntity<Object> handleGeneralException(Exception ex) {

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

}

}

**Custom Exception Class:**   
   
package com.example.bookstoreapi.exception;

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

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

#### **7.1 Create DTOs:**

**BookDTO Class:**   
   
package com.example.bookstoreapi.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class BookDTO {

private Long id;

private String title;

private String author;

private double price;

}

#### **7.2 Mapping Entities to DTOs:**

* **Using MapStruct:**

Add dependency in pom.xml:  
xml  
   
<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.3.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.3.Final</version>

</dependency>

**BookMapper Interface:**   
   
package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.entity.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);

}

#### **7.3 Custom Serialization/Deserialization:**

**Using Jackson Annotations:**   
   
package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class BookDTO {

private Long id;

private String title;

private String author;

@JsonIgnore

private double price;

}

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

#### **8.1 CRUD Endpoints:**

**Book CRUD Operations:**   
   
@RestController

@RequestMapping("/books")

public class BookController {

// CREATE

@PostMapping

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

books.add(book);

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

}

// READ

@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

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

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

}

// UPDATE

@PutMapping("/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

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

})

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

}

// DELETE

@DeleteMapping("/{id}")

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

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

if (removed) {

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

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

}

}

#### **8.2 Validating Input Data:**

**Validation Annotations:**   
   
@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String title;

@NotNull

@Size(min = 2, max = 100)

private String author;

@Min(1)

private double price;

@NotNull

@Pattern(regexp = "^[0-9]{13}$")

private String isbn;

}

#### **8.3 Optimistic Locking:**

**Enable Optimistic Locking:**   
   
@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Version

private Integer version;

// Other fields...

}

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

#### **9.1 Add Links to Resources:**

**Using Spring HATEOAS:**   
   
import org.springframework.hateoas.Link;

import org.springframework.hateoas.RepresentationModel;

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

@RestController

@RequestMapping("/books")

public class BookController {

@GetMapping("/{id}")

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

return books.stream()

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

.findFirst()

.map(book -> {

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

return ResponseEntity.ok(new RepresentationModel<>(book).add(selfLink));

})

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

}

}

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

#### **10.1 Content Negotiation:**

**Configure Spring Boot to Support Content Negotiation:**   
   
@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorParameter(true)

.parameterName("mediaType")

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

}

#### **10.2 Accept Header:**

**Implement Logic for Media Types:**   
   
@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id, @RequestHeader(value = "Accept", required = false) String acceptHeader) {

Book book = books.stream()

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

.findFirst()

.orElse(null);

if (book == null) {

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

}

if ("application/xml".equals(acceptHeader)) {

return ResponseEntity.ok().header(HttpHeaders.CONTENT\_TYPE, MediaType.APPLICATION\_XML\_VALUE).body(book);

}

return ResponseEntity.ok().header(HttpHeaders.CONTENT\_TYPE, MediaType.APPLICATION\_JSON\_VALUE).body(book);

}

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

#### **11.1 Add Actuator Dependency:**

**Add to pom.xml:**xml  
   
<dependency>

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

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

</dependency>

#### **11.2 Expose Actuator Endpoints:**

**Enable Endpoints:**properties  
   
management.endpoints.web.exposure.include=\*

#### **11.3 Custom Metrics:**

**Expose Custom Metrics:**   
   
@RestController

@RequestMapping("/metrics")

public class MetricsController {

@GetMapping("/custom")

public ResponseEntity<String> getCustomMetrics() {

return ResponseEntity.ok("Custom metrics data");

}

}

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

#### **12.1 Add Spring Security:**

**Add to pom.xml:**xml  
   
<dependency>

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

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

</dependency>

#### **12.2 JWT Authentication:**

**JWT Utility Class:**   
   
@Service

public class JwtUtil {

private String SECRET\_KEY = "secret";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

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

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

public String generateToken(UserDetails userDetails) {

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

return createToken(claims, userDetails.getUsername());

}

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

return Jwts.builder().setClaims(claims).setSubject(subject)

.setIssuedAt(new Date(System.currentTimeMillis()))

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

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY).compact();

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

private Boolean isTokenExpired(String token) {

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

}

private Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

}

#### **12.3 CORS Handling:**

**Enable CORS Globally:**   
   
@Configuration

public class CorsConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*").allowedOrigins("\*").allowedMethods("GET", "POST", "PUT", "DELETE");

}

}

### **Exercise 13: Online Bookstore - Pagination and Sorting**

#### **13.1 Enable Pagination and Sorting:**

**Repository Interface:**   
   
@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

#### **13.2 Implement Pagination:**

**Controller Method:**   
   
@GetMapping

public Page<Book> getBooks(Pageable pageable) {

return bookRepository.findAll(pageable);

}

#### **13.3 Implement Sorting:**

**Sorting Example:**   
   
@GetMapping

public List<Book> getBooks(@RequestParam String sortBy) {

return bookRepository.findAll(Sort.by(sortBy));

}

### **Exercise 14: Online Bookstore - Versioning RESTful Services**

#### **14.1 URI Versioning:**

**Versioning Example:**   
   
@GetMapping("/v1/books/{id}")

public ResponseEntity<BookV1> getBookV1(@PathVariable Long id) {

return booksV1.stream()

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

.findFirst()

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

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

}

@GetMapping("/v2/books/{id}")

public ResponseEntity<BookV2> getBookV2(@PathVariable Long id) {

return booksV2.stream()

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

.findFirst()

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

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

}

#### **14.2 Parameter Versioning:**

**Parameter-Based Versioning:**   
   
@GetMapping(value = "/books/{id}", params = "version=1")

public ResponseEntity<BookV1> getBookV1(@PathVariable Long id) {

return booksV1.stream()

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

.findFirst()

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

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

}

@GetMapping(value = "/books/{id}", params = "version=2")

public ResponseEntity<BookV2> getBookV2(@PathVariable Long id) {

return booksV2.stream()

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

.findFirst()

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

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

* }