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

1. **Setup Spring Boot Project:**
   * Initialize a new Spring Boot project named BookstoreAPI.
   * Add dependencies:
     + Spring Web
     + Spring Boot DevTools
     + Lombok
2. **Project Structure:**
   * Familiarize yourself with the generated structure, including:
     + src/main/java: contains application logic.
     + src/main/resources: contains configuration files like application.properties.
     + src/test/java: contains test files.
3. **What's New in Spring Boot 3:**
   * **Java 17 support**: Spring Boot 3 now requires Java 17 as the minimum version.
   * **New observability features**: Integrated Micrometer and Prometheus support for application metrics.
   * **Spring Native**: Improved native image support with GraalVM for faster startup times.
   * **Improved Docker support**: With build packs that allow creating OCI-compliant images directly from Spring Boot applications.

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

1. **Create Book Controller:**

@RestController

@RequestMapping("/books")

public class BookController {

}

1. **Handle HTTP Methods:**

@GetMapping

@RestController

@RequestMapping("/books")

public class BookController {

private final BookService bookService;

// Constructor-based dependency injection

public BookController(BookService bookService) {

this.bookService = bookService;

}

// GET - Retrieve all books

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

// POST - Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookService.addBook(book);

}

// PUT - Update an existing book by ID

@PutMapping("/{id}")

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

return bookService.updateBook(id, updatedBook);

}

// DELETE - Delete a book by ID

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookService.deleteBook(id);

}

}

public List<Book> getAllBooks() {

}

@PostMapping

public Book addBook(@RequestBody Book book) {

}

@PutMapping("/{id}")

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

// Logic for updating a book

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

// Logic for deleting a book

}

1. **Return JSON Responses:**
   * By default, Spring Boot REST controllers return JSON responses.
2. **Define Book Entity:**

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Book {

private Long id;

private String title;

private String author;

private Double price;

private String isbn;

}

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

1. **Path Variables:**

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

}

1. **Query Parameters:**

@GetMapping("/search")

public List<Book> searchBooks(@RequestParam String title, @RequestParam String author) {

}

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

1. **Request Body:**

@PostMapping("/customers")

public Customer createCustomer(@RequestBody Customer customer) {

}

1. **Form Data:**

@PostMapping("/customers/form")

public Customer registerCustomer(@RequestParam String name, @RequestParam String email) {

}

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

1. **Response Status:**

@ResponseStatus(HttpStatus.CREATED)

@PostMapping

public Book createBook(@RequestBody Book book) {

}

1. **Custom Headers:**

@GetMapping("/{id}")

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

HttpHeaders headers = new HttpHeaders();

headers.add("Custom-Header", "BookStore");

return ResponseEntity.ok().headers(headers).body(book);

}

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

1. **Global Exception Handler:**

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<String> handleNotFoundException(ResourceNotFoundException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(ex.getMessage());

}

}

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

1. **Create DTOs:**

public class BookDTO {

private Long id;

private String title;

private String author;

private Double price;

}

public class CustomerDTO {

private Long id;

private String name;

private String email;

}

1. **Mapping Entities to DTOs:**
   * Use MapStruct:

@Mapper

public interface BookMapper {

BookDTO toDto(Book book);

Book toEntity(BookDTO bookDto);

}

1. **Custom Serialization/Deserialization:**
   * Use Jackson annotations like @JsonProperty, @JsonIgnore, etc.

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

1. **CRUD Endpoints:**  
   Implement the basic CRUD operations for books and customers similar to Exercise 2.
2. **Validating Input Data:**

public class BookDTO {

@NotNull @Size(min = 1)

private String title;

@Min(0)

private Double price;

}

1. **Optimistic Locking:**
   * Add a @Version field to entities for concurrency control.

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

1. **Add Links to Resources:**

Resource<Book> resource = new Resource<>(book);

resource.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

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

1. **Content Negotiation:**
   * Configure Spring Boot to support XML and JSON by adding:

spring:

mvc:

content-negotiation:

favor-parameter: true

favor-path-extension: false

media-types:

json: application/json

xml: application/xml

1. **Accept Header:**
   * Use @RequestMapping(produces = {"application/json", "application/xml"}).

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

1. **Add Actuator Dependency:**
   * Add spring-boot-starter-actuator to pom.xml.
2. **Expose Actuator Endpoints:**
   * Enable specific endpoints in application.properties.

management:

endpoints:

web:

exposure:

include: "\*"

1. **Custom Metrics:**
   * Implement custom metrics using Micrometer.

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

1. **Add Spring Security:**
   * Add spring-boot-starter-security dependency.
2. **JWT Authentication:**
   * Implement JWT token generation and validation in a filter.
3. **CORS Handling:**
   * Use @CrossOrigin to configure CORS for your endpoints.

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

1. **JUnit Setup:**
   * Include spring-boot-starter-test and mockito-core dependencies.
2. **MockMvc:**
   * Write tests using MockMvc.

@Test

public void testGetBooks() throws Exception {

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.length()").value(3));

}

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

1. **Spring Test:**
   * Set up Spring Test with @SpringBootTest.
2. **MockMvc Integration:**
   * Use MockMvc for integration testing as in Exercise 13.
3. **Database Integration:**
   * Use an H2 in-memory database for tests.

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

1. **Add Swagger Dependency:**
   * Add springdoc-openapi-ui dependency.
2. **Document Endpoints:**
   * Use annotations like @Operation, @Parameter.
3. **API Documentation:**
   * Access the documentation at /swagger-ui.html or /v3/api-docs.