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

1. **Add Links to Resources:**
   * **Code:** You need to add HATEOAS support to your entities by creating resource assemblers that add links to your responses.

**BookResourceAssembler**

import org.springframework.hateoas.EntityModel;

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

import org.springframework.stereotype.Component;

@Component

public class BookResourceAssembler extends RepresentationModelAssemblerSupport<Book, EntityModel<Book>> {

public BookResourceAssembler() {

super(BookController.class, EntityModel.class);

}

@Override

public EntityModel<Book> toModel(Book book) {

EntityModel<Book> bookResource = EntityModel.of(book);

bookResource.add(WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).getBook(book.getId())).withSelfRel());

bookResource.add(WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(BookController.class).getAllBooks()).withRel("books"));

return bookResource;

}

}

Hypermedia-Driven APIs:

BookController:

import org.springframework.hateoas.EntityModel;

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

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@Autowired

private BookResourceAssembler assembler;

@GetMapping("/{id}")

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

Book book = bookService.findById(id);

return ResponseEntity.ok(assembler.toModel(book));

}

@GetMapping("/")

public ResponseEntity<CollectionModel<EntityModel<Book>>> getAllBooks() {

List<EntityModel<Book>> books = bookService.findAll().stream().map(assembler::toModel).collect(Collectors.toList());

return ResponseEntity.ok(CollectionModel.of(books, linkTo(methodOn(BookController.class).getAllBooks()).withSelfRel()));

}

}

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

**Content Negotiation:**

Code: Enable content negotiation by configuring ContentNegotiationConfigurer in your Spring configuration.

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)

.parameterName("mediaType")

.ignoreAcceptHeader(false)

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

}

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

**Add Actuator Dependency:**

<dependency>

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

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

</dependency>

**Expose Actuator Endpoints:**

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

management.endpoint.health.show-details=always

**Custom Metrics:**

**Code: You can create custom metrics using MeterRegistry.**

import io.micrometer.core.instrument.MeterRegistry;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class MetricsConfig {

@Bean

public MeterRegistryCustomizer<MeterRegistry> metricsCommonTags() {

return registry -> registry.config().commonTags("application", "Online Bookstore");

}

}

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

**Add Spring Security:**

Code: Add the following dependency to pom.xml

<dependency>

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

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

</dependency>

**JWT Authentication:**

Code: Implement JWT authentication by creating filters and configuring SecurityConfig

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.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

// Add JWT token filter here

}

}

CORS Handling:

Code: Configure CORS in your security configuration.

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.cors().and()

.csrf().disable()

// ... rest of the configuration

}

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

**JUnit Setup:**

Code: Add the following dependencies to your pom.xml

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<scope>test</scope>

</dependency>

**MockMvc:**

Code: Use MockMvc in your test classes to write unit tests.

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

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

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.test.web.servlet.MockMvc;

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testGetBook() throws Exception {

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

.andExpect(status().isOk());

}

}

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

**Spring Test:**

Code: Already included with the spring-boot-starter-test dependency added in Exercise 13.

**MockMvc Integration:**

Code: Similar to unit testing, but with full application context.

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit.jupiter.SpringExtension;

@SpringBootTest

public class BookIntegrationTest {

// Use full application context for integration testing

}

**Database Integration:**

**Code:** Use an in-memory database like H2 for integration tests.

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

**Document Endpoints:**

**Code:** Annotate your REST controllers and methods

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

@RestController

@RequestMapping("/api/books")

@Tag(name = "Book", description = "The Book API")

public class BookController {

@Operation(summary = "Get a book by its id")

@GetMapping("/{id}")

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

// ...

}

}

**API Documentation:**

* **Code:** Swagger UI will be available at http://localhost:8080/swagger-ui.html or http://localhost:8080/swagger-ui/index.html.