### **Exercise 1: Setting Up RESTful Services**

1. **Setup Spring Boot Project:**You can use Spring Initializr to create the project:
   * **Dependencies:** Spring Web, Spring Boot DevTools, Lombok.

bash  
   
mvn spring-boot:run

1. This command will start your Spring Boot application.
2. **Project Structure:**The structure generated by Spring Boot includes:
   * src/main/java: Your Java code.
   * src/main/resources: Your configuration files like application.properties.
   * src/test/java: Your test cases.
3. **What's New in Spring Boot 3:**
   * Improved support for GraalVM native images.
   * Java 17 as the baseline.
   * Enhanced observability with Micrometer.

### **Exercise 2: Creating Basic REST Controllers**

**Create Book Controller:**java  
   
@RestController

@RequestMapping("/books")

public class BookController {

@GetMapping

public List<Book> getAllBooks() {

// Logic to return all books

}

@PostMapping

public Book createBook(@RequestBody Book book) {

// Logic to create a new book

}

@PutMapping("/{id}")

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

// Logic to update a book

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

// Logic to delete a book

}

}

1. **Handle HTTP Methods:**The methods in the BookController handle different HTTP requests like GET, POST, PUT, and DELETE.
2. **Return JSON Responses:**
   * The @RestController annotation automatically converts returned objects to JSON.

java  
   
@Data

@AllArgsConstructor

@NoArgsConstructor

public class Book {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

### **Exercise 3: Handling Path Variables and Query Parameters**

**Path Variables:**java  
   
@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

// Logic to fetch book by ID

}

**Query Parameters:**java  
   
@GetMapping("/search")

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

// Logic to search books by title and author

}

### **Exercise 4: Processing Request Body and Form Data**

**Request Body:**java  
   
@PostMapping("/customers")

public Customer createCustomer(@RequestBody Customer customer) {

// Logic to create a new customer

}

**Form Data:**java  
   
@PostMapping("/register")

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

// Logic to register a customer using form data

}

### **Exercise 5: Customizing Response Status and Headers**

**Response Status:**java  
   
@ResponseStatus(HttpStatus.CREATED)

@PostMapping("/books")

public Book createBook(@RequestBody Book book) {

// Logic to create a new book

}

**Custom Headers:**java  
   
@PutMapping("/{id}")

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

// Logic to update a book

return ResponseEntity.ok()

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

.body(updatedBook);

}

### **Exercise 6: Exception Handling in REST Controllers**

**Global Exception Handler:**java  
   
@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(BookNotFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

public String handleBookNotFound(BookNotFoundException ex) {

return ex.getMessage();

}

}

* + The @ControllerAdvice annotation is used to handle exceptions globally.

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

**Create DTOs:**java  
   
@Data

@AllArgsConstructor

@NoArgsConstructor

public class BookDTO {

private Long id;

private String title;

private String author;

}

**Mapping Entities to DTOs:**You can use MapStruct or ModelMapper for this:  
java  
   
BookDTO bookDTO = modelMapper.map(book, BookDTO.class);

**Custom Serialization/Deserialization:**java  
   
@JsonSerialize(using = CustomBookSerializer.class)

public class Book { ... }

### **Exercise 8: Implementing CRUD Operations**

**CRUD Endpoints:**java  
   
@PostMapping("/books")

public Book createBook(@RequestBody Book book) { ... }

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

public Book getBookById(@PathVariable Long id) { ... }

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

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

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

public void deleteBook(@PathVariable Long id) { ... }

**Validating Input Data:**java  
   
public class Book {

@NotNull

private String title;

@Size(min = 10, max = 13)

private String isbn;

}

**Optimistic Locking:**java  
   
@Version

private Long version;

### **Exercise 9: Understanding HATEOAS**

**Add Links to Resources:**java  
   
EntityModel<Book> resource = EntityModel.of(book);

resource.add(Link.of("/books/" + book.getId()).withSelfRel());

1. **Hypermedia-Driven APIs:**Spring HATEOAS provides tools to easily create hypermedia-driven APIs.

### **Exercise 10: Configuring Content Negotiation**

**Content Negotiation:**java  
   
@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void configureContentNegotiation(ContentNegotiationConfigurer configurer) {

configurer.favorPathExtension(true)

.favorParameter(true)

.ignoreAcceptHeader(false)

.useRegisteredExtensionsOnly(false)

.defaultContentType(MediaType.APPLICATION\_JSON)

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

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

}

}

1. **Accept Header:**The controller will automatically handle different media types based on the Accept header.

### **Exercise 11: Integrating Spring Boot Actuator**

**Add Actuator Dependency:**xml  
   
<dependency>

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

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

</dependency>

**Expose Actuator Endpoints:**Configure endpoints in application.properties:  
properties  
   
management.endpoints.web.exposure.include=\*

**Custom Metrics:**Use MeterRegistry to expose custom metrics:  
java  
   
@Autowired

private MeterRegistry meterRegistry;

public void recordCustomMetric() {

meterRegistry.counter("custom.metric.name").increment();

}

### **Exercise 12: Securing RESTful Endpoints with Spring Security**

**Add Spring Security:**xml  
   
<dependency>

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

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

</dependency>

**JWT Authentication:**Implement JWT filter and configure security:  
java  
   
@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests().ant

.

Here’s the coding implementation for each of the exercises:

### **Exercise 12: Securing RESTful Endpoints with Spring Security**

**1. Add Spring Security:**

xml

* + <!-- Add to pom.xml -->
  + <dependency>
  + <groupId>org.springframework.boot</groupId>
  + <artifactId>spring-boot-starter-security</artifactId>
  + </dependency>
  + <dependency>
  + <groupId>io.jsonwebtoken</groupId>
  + <artifactId>jjwt</artifactId>
  + <version>0.9.1</version>
  + </dependency>

**2. JWT Authentication:**

* **Security Configuration:**

java

* + import org.springframework.context.annotation.Bean;
  + import org.springframework.context.annotation.Configuration;
  + import org.springframework.security.authentication.AuthenticationManager;
  + import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
  + 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;
  + import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
  + import org.springframework.security.crypto.password.PasswordEncoder;
  + import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
  + @Configuration
  + @EnableWebSecurity
  + public class SecurityConfig extends WebSecurityConfigurerAdapter {
  + @Override
  + protected void configure(AuthenticationManagerBuilder auth) throws Exception {
  + // Configure authentication manager with user details service
  + }
  + @Override
  + protected void configure(HttpSecurity http) throws Exception {
  + http.csrf().disable()
  + .authorizeRequests()
  + .antMatchers("/auth/login").permitAll()
  + .anyRequest().authenticated()
  + .and()
  + .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
  + http.addFilterBefore(jwtFilter(), UsernamePasswordAuthenticationFilter.class);
  + }
  + @Bean
  + public JwtFilter jwtFilter() {
  + return new JwtFilter();
  + }
  + @Bean
  + public PasswordEncoder passwordEncoder() {
  + return new BCryptPasswordEncoder();
  + }
  + @Override
  + @Bean
  + public AuthenticationManager authenticationManagerBean() throws Exception {
  + return super.authenticationManagerBean();
  + }
  + }
* **JWT Utility Class:**

java

* + import io.jsonwebtoken.Claims;
  + import io.jsonwebtoken.Jwts;
  + import io.jsonwebtoken.SignatureAlgorithm;
  + import org.springframework.security.core.userdetails.UserDetails;
  + import org.springframework.stereotype.Service;
  + import java.util.Date;
  + import java.util.HashMap;
  + import java.util.Map;
  + import java.util.function.Function;
  + @Service
  + public class JwtUtil {
  + private String SECRET\_KEY = "secret";
  + public String extractUsername(String token) {
  + return extractClaim(token, Claims::getSubject);
  + }
  + public Date extractExpiration(String token) {
  + return extractClaim(token, Claims::getExpiration);
  + }
  + 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();
  + }
  + private Boolean isTokenExpired(String token) {
  + return extractExpiration(token).before(new Date());
  + }
  + 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));
  + }
  + }
* **JWT Filter:**

java

* + import org.springframework.beans.factory.annotation.Autowired;
  + import org.springframework.security.core.context.SecurityContextHolder;
  + import org.springframework.security.core.userdetails.UserDetails;
  + import org.springframework.security.core.userdetails.UserDetailsService;
  + import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
  + import org.springframework.stereotype.Component;
  + import org.springframework.web.filter.OncePerRequestFilter;
  + import io.jsonwebtoken.ExpiredJwtException;
  + import javax.servlet.FilterChain;
  + import javax.servlet.http.HttpServletRequest;
  + import javax.servlet.http.HttpServletResponse;
  + import java.io.IOException;
  + @Component
  + public class JwtFilter extends OncePerRequestFilter {
  + @Autowired
  + private JwtUtil jwtUtil;
  + @Autowired
  + private UserDetailsService userDetailsService;
  + @Override
  + protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
  + throws IOException, ServletException {
  + final String authorizationHeader = request.getHeader("Authorization");
  + String username = null;
  + String jwt = null;
  + if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
  + jwt = authorizationHeader.substring(7);
  + try {
  + username = jwtUtil.extractUsername(jwt);
  + } catch (ExpiredJwtException e) {
  + // Handle the exception
  + }
  + }
  + if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
  + UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);
  + if (jwtUtil.validateToken(jwt, userDetails)) {
  + UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =
  + new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
  + usernamePasswordAuthenticationToken
  + .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
  + SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
  + }
  + }
  + chain.doFilter(request, response);
  + }
  + }

**3. CORS Handling:**

java

* + @Override
  + protected void configure(HttpSecurity http) throws Exception {
  + http.cors().and().csrf().disable()
  + .authorizeRequests()
  + .antMatchers("/auth/login").permitAll()
  + .anyRequest().authenticated()
  + .and()
  + .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
  + http.addFilterBefore(jwtFilter(), UsernamePasswordAuthenticationFilter.class);
  + }
  + @Bean
  + public CorsConfigurationSource corsConfigurationSource() {
  + CorsConfiguration configuration = new CorsConfiguration();
  + configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000")); // Change this to your frontend origin
  + configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
  + configuration.setAllowedHeaders(Arrays.asList("Authorization", "Cache-Control", "Content-Type"));
  + UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
  + source.registerCorsConfiguration("/\*\*", configuration);
  + return source;
  + }

### **Exercise 13: Unit Testing REST Controllers**

**1. JUnit Setup:**

xml

* + <!-- Add to 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>

**2. MockMvc:**

java

* + import static org.mockito.Mockito.when;
  + import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
  + import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
  + import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
  + 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.boot.test.mock.mockito.MockBean;
  + import org.springframework.test.web.servlet.MockMvc;
  + @WebMvcTest(BookController.class)
  + public class BookControllerTest {
  + @Autowired
  + private MockMvc mockMvc;
  + @MockBean
  + private BookService bookService;
  + @Test
  + public void testGetBookById() throws Exception {
  + when(bookService.getBookById(1L)).thenReturn(new Book(1L, "Effective Java", "Joshua Bloch"));
  + mockMvc.perform(get("/books/1"))
  + .andExpect(status().isOk())
  + .andExpect(jsonPath("$.title").value("Effective Java"))
  + .andExpect(jsonPath("$.author").value("Joshua Bloch"));
  + }
  + }

**3. Test Coverage:**

* Ensure you have similar tests for different endpoints, including error scenarios (e.g., book not found, invalid input).

### **Exercise 14: Integration Testing for REST Services**

**1. Spring Test:**

java

* + import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
  + import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
  + import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
  + import org.junit.jupiter.api.Test;
  + import org.springframework.beans.factory.annotation.Autowired;
  + import org.springframework.boot.test.context.SpringBootTest;
  + import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
  + import org.springframework.test.web.servlet.MockMvc;
  + import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
  + @SpringBootTest
  + @AutoConfigureMockMvc
  + public class BookControllerIntegrationTest {
  + @Autowired
  + private MockMvc mockMvc;
  + @Test
  + public void testCreateBook() throws Exception {
  + String newBookJson = "{\"title\":\"Clean Code\",\"author\":\"Robert Martin\"}";
  + mockMvc.perform(post("/books")
  + .contentType("application/json")
  + .content(newBookJson))
  + .andExpect(status().isOk())
  + .andExpect(jsonPath("$.title").value("Clean Code"))
  + .andExpect(jsonPath

**Exercise 15: API Documentation with Swagger**

### **1. Add Swagger Dependency**

* **pom.xml:**

xml

Copy code

<!-- Add to pom.xml -->

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.7.0</version>

</dependency>

### **2. Document Endpoints**

* **Example REST Controller:**

java

Copy code

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

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

import io.swagger.v3.oas.annotations.responses.ApiResponse;

import io.swagger.v3.oas.annotations.responses.ApiResponses;

@RestController

public class BookController {

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

@ApiResponses(value = {

@ApiResponse(responseCode = "200", description = "Found the book"),

@ApiResponse(responseCode = "404", description = "Book not found")

})

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

public Book getBookById(@PathVariable Long id) {

// Implementation here

return new Book(id, "Effective Java", "Joshua Bloch");

}

// Other endpoints can be documented similarly

}

* **Book Class:**

java

Copy code

public class Book {

private Long id;

private String title;

private String author;

// Constructors, Getters, and Setters

public Book(Long id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

// Getters and Setters

}

### **3. API Documentation**

* **Access the Swagger UI:**
  + After starting your Spring Boot application, access the Swagger UI at http://localhost:8080/swagger-ui/index.html.
  + This URL provides an interactive interface where you can view and interact with the API documentation generated from your annotated REST controllers.