**WEEK 4&5**

**EXERCISE 10**

**Online Bookstore - Configuring Content Negotiation**

**Business Scenario**

The task is to configure the content negotiation in your RESTful services to support different media types such as JSON and XML. This allows clients to specify their preferred format in the Accept header of HTTP requests and receive responses in that format.

**Instructions**

**1. Content Negotiation:**

**Task:** Configure Spring Boot to support content negotiation for different media types.

**Implementation:**

* Add Dependencies: Ensure that you have dependencies for both JSON and XML support in your pom.xml or build.gradle file.

**Dependencies for Maven:**

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

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

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

</dependency>

**Dependencies for Gradle:**

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'com.fasterxml.jackson.core:jackson-databind'

implementation 'org.springframework.boot:spring-boot-starter-xml'

* **Configure Content Negotiation in application.properties or application.yml:**

**application.properties:**

spring.mvc.format.date-time=iso

spring.mvc.contentnegotiation.favor-parameter=false

spring.mvc.contentnegotiation.favor-path-extension=true

spring.mvc.contentnegotiation.ignore-accept-header=false

spring.mvc.contentnegotiation.media-types.xml=application/xml

spring.mvc.contentnegotiation.media-types.json=application/json

**application.yml:**

spring:

mvc:

format:

date-time: iso

contentnegotiation:

favor-parameter: false

favor-path-extension: true

ignore-accept-header: false

media-types:

xml: application/xml

json: application/json

**2. Accept Header:**

**Task:** Implement logic to produce and consume different media types based on the Accept header.

**Implementation:**

* **Modify Controller to Handle Different Media Types:**

**BookController.java:**

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

import org.springframework.hateoas.EntityModel;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.HttpStatus;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import javax.validation.Valid;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookService bookService;

@PostMapping(consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<BookDTO> createBook(@Valid @RequestBody BookDTO bookDTO) {

Book book = bookService.createBook(bookDTO);

BookDTO responseDTO = new BookDTO();

responseDTO.setId(book.getId());

responseDTO.setTitle(book.getTitle());

responseDTO.setAuthor(book.getAuthor());

responseDTO.setPrice(book.getPrice());

return ResponseEntity.status(HttpStatus.CREATED).contentType(MediaType.APPLICATION\_JSON).body(responseDTO);

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<BookDTO> getBookById(@PathVariable Long id, @RequestHeader(HttpHeaders.ACCEPT) String acceptHeader) {

BookDTO bookDTO = bookService.getBookById(id);

if (bookDTO != null) {

MediaType mediaType = MediaType.parseMediaType(acceptHeader);

return ResponseEntity.ok().contentType(mediaType).body(bookDTO);

} else {

return ResponseEntity.notFound().build();

}

}

@PutMapping(value = "/{id}", consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<BookDTO> updateBook(@PathVariable Long id, @Valid @RequestBody BookDTO bookDTO) {

BookDTO updatedBook = bookService.updateBook(id, bookDTO);

if (updatedBook != null) {

return ResponseEntity.ok().contentType(MediaType.APPLICATION\_JSON).body(updatedBook);

} else {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

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

boolean isDeleted = bookService.deleteBook(id);

return isDeleted ? ResponseEntity.noContent().build() : ResponseEntity.notFound().build();

}

}

**Explanation:**

* The @PostMapping, @GetMapping, and @PutMapping annotations specify the media types they can consume or produce.
* The produces attribute in @GetMapping ensures that the response format is based on the Accept header sent by the client.
* The contentType method of ResponseEntity sets the appropriate media type in the response.

**Testing:**

* Test the API using tools like Postman or cURL by setting the Accept header to application/json or application/xml and checking the response format.

**Conclusion:**

By configuring content negotiation and handling different media types, the RESTful services will be more flexible and capable of serving clients in the format they prefer.