**WEEK 4&5**

**EXERCISE 9**

**Online Bookstore - Understanding HATEOAS**

**Business Scenario**

The task is to enhance the REST API to follow HATEOAS (Hypermedia as the Engine of Application State) principles for navigation through resources.

**Instructions**

**1. Add Links to Resources:**

**Task:** Use Spring HATEOAS to add links to resources in your API responses.

**Implementation:**

* HATEOAS provides links within the responses to guide clients on how to interact with the API.

**Code Example:**

**BookDTO.java:**

import org.springframework.hateoas.RepresentationModel;

public class BookDTO extends RepresentationModel<BookDTO> {

private Long id;

private String title;

private String author;

private double price;

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**BookController.java:**

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

import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.Link;

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

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

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());

// Add link to self

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

responseDTO.add(selfLink);

return ResponseEntity.status(HttpStatus.CREATED).body(responseDTO);

}

@GetMapping("/{id}")

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

BookDTO bookDTO = bookService.getBookById(id);

if (bookDTO != null) {

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

bookDTO.add(selfLink);

return ResponseEntity.ok(bookDTO);

} else {

return ResponseEntity.notFound().build();

}

}

@PutMapping("/{id}")

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

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

if (updatedBook != null) {

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

updatedBook.add(selfLink);

return ResponseEntity.ok(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 BookDTO class extends RepresentationModel<BookDTO>, which provides methods to add links to the resource.
* In the BookController, after creating or retrieving a book, a link to the resource itself is added using WebMvcLinkBuilder.linkTo().
* The withSelfRel() method adds a self-referential link, allowing clients to navigate to the current resource.

**2. Hypermedia-Driven APIs:**

**Task:** Build and consume hypermedia-driven APIs.

**Implementation:**

* Hypermedia-driven APIs provide navigation through resources using links in the API responses.

**Example of Hypermedia-Driven API Consumption:**

**Client-Side Example:**

import org.springframework.hateoas.EntityModel;

import org.springframework.web.client.RestTemplate;

public class BookClient {

private final RestTemplate restTemplate = new RestTemplate();

public BookDTO getBook(Long id) {

EntityModel<BookDTO> bookEntity = restTemplate.getForObject("/books/" + id, EntityModel.class);

BookDTO bookDTO = bookEntity.getContent();

if (bookDTO != null) {

System.out.println("Book Title: " + bookDTO.getTitle());

}

return bookDTO;

}

}

**Explanation:**

* The client retrieves a BookDTO using a RestTemplate and processes the response.
* The client can use the links provided in the BookDTO to navigate to related resources.

**Conclusion:**

By implementing HATEOAS, we can enhance the REST API to be more navigable and self-descriptive, improving the client experience by providing meaningful links for resource interaction.