Building RESTful APIs

In this section, we explored how to design and implement RESTful APIs in Spring Boot. We covered essential concepts like handling HTTP requests, structuring API responses, and performing CRUD operations.

Creating APIs

- **@RestController** – Marks the class as a REST API controller.
- **@RequestMapping** – Defines the base URL for all endpoints inside the controller.

```
@RestController
@RequestMapping("/products")
public class ProductController { }
```

Handling HTTP Requests

- **@PathVariable** – Extracts values from the URL to handle requests like fetching a product by ID.

```
@GetMapping("/{id}")
public Product getProduct(@PathVariable Long id) { }
```

- **@RequestParam** – Extracts query parameters from the URL, commonly used for filtering and sorting.

```
@GetMapping
public List<Product> getProducts(
    @RequestParam(required = false) Long categoryId,
    @RequestParam(defaultValue = "name") String sortBy) {
    // Handle filtering and sorting
}
```

- **@RequestHeader** – Reads HTTP headers, often used for authentication or metadata.

```
@GetMapping
public List<Product> getProducts(
    @RequestHeader("x-auth-token") String token) {
}
```

- **@RequestBody** – Extracts data from the request body, typically for creating or updating resources.

```
@PostMapping
public ResponseEntity<Product> createProduct(
    @RequestBody ProductDto dto) {
}
```

Handling HTTP Responses

- **ResponseEntity** – Customizes API responses, including status codes, headers, and body content.
- **HTTP Status Codes**
  - 200 OK
  - 201 Created
  - 400 Bad Request
  - 404 Not Found

```
@PostMapping
public ResponseEntity<Product> createProduct(
    @RequestBody ProductDto dto) {
    return ResponseEntity.notFound().build();
}
```

Using DTOs

- **Data Transfer Objects (DTOs)** – Prevent exposing database entities directly by defining structured response objects.
- **MapStruct** – Automatically converts entities to DTOs without manual mapping code.

```
@Mapper(componentModel = "spring")
public interface ProductMapper {
    @Mapping(source = "category.id", target = "categoryId")
    ProductDto toDto(Product product);
}
```

CRUD Operations

- **Creating resources** (POST) – Adds new data to the database.
- **Fetching resources** (GET) – Retrieves data from the database, optionally with filtering and sorting.
- **Updating resources** (PUT/PATCH) – Modifies existing records.
- **Deleting resources** (DELETE) – Removes data while handling errors properly.
- **Action-Based Updates** (POST) – Used for operations that modify state but don't fit traditional CRUD actions (e.g., changing a password).

```
@PostMapping
public ResponseEntity<ProductDto> createProduct(
    @RequestBody ProductDto dto) {
}

@GetMapping
public List<ProductDto> getAllProducts() { }

@PutMapping("/{id}")
public ResponseEntity<ProductDto> updateProduct(
    @PathVariable Long id,
    @RequestBody ProductDto dto) {
}

@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteProduct(
    @PathVariable Long id) {
}
```