Spring Boot 3 - REST CRUD APIs

What are REST Services?

- **REST**
  - **RE**presentational **S**tate **T**ransfer
  - Lightweight approach for **communicating between applications**
  - **Language independent**
    - **Client** application and **server** can use **any language**
  - Can use any **data format**
    - Commonly **XML** and **JSON**
      - **J**ava**S**cript **O**bject **N**otation

JSON Basics

- What is **JSON**?
  - **J**ava**S**cript **O**bject **N**otation
  - Lightweight **data format** for **storing** and **exchanging data** → **Plain text**
  - **Language independent**
- **JSON Structure**
  - **Curley braces** define **objects** in **JSON**
  - **Object members** are **name/value pairs**
    - Delimited by **colons**
  - **Name** is always in **double-quotes**
- **JSON Values**
  - **Numbers** → No quotes
  - **String** → Double Quotes
  - **Boolean** → true, false
  - Nested **JSON Object**
  - **Array**
  - **null**

Spring Boot REST HTTP Basics

- **REST Over HTTP**
  - Most common use of **REST** is over **HTTP**
  - Leverage **HTTP methods** for **CRUD operations**

HTTP Method	CRUD Operation
POST	**Create** a new entity
GET	**Read** a list of entities or single entity
PUT	**Update** an existing entity
DELETE	**Delete** an existing entity

- HTTP **Request Message**
  - **Request Line** → The HTTP command
  - **Header Variables** → Request metadata
  - **Message Body** → Contents of message
- HTTP **Response Message**
  - **Response Line** → Server protocol and status code
  - **Header Variables** → Response metadata
  - **Message Body** → Contents of message
- HTTP Response - **Status Codes**

Code Range	Description
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client error

500 - 599	Server error

- **MIME Content Types**
  - The **message format** is described by **MIME content type**
    - **M**ultipurpose **I**nternet **M**ail-**E**xtension
  - Basic Syntax → **type/sub-type**
  - Examples
    - text/html
    - text/plain
    - application/json
    - application/xml
- **Client Tool**
  - **Send HTTP requests** to the **REST Web Service / API**

Spring Boot Rest Controller

- **Web Browser vs Postman**
  - For **simple REST testing** for GET requests
    - **Web browser** and **Postman** are **similar**
  - For **advanced REST testing** → POST, PUT, etc.
    - **Postman** has much better support
    - **POSTing JOSN data, setting content type**
    - **Passing HTTP request headers, authentication**, etc
- **Development Process**
  - Add **Maven dependency** for **Spring Boot Starter Web**
  - Create **Spring REST Service** using **@RestController**

JSON Jackson Data Binding

- **Java JSON Data Binding**
  - **Data binding** is the process of **converting JSON data** to a **Java POJO**
  - AKA → **mapping**, **serialization/deserialization**, **marshalling/unmarshalling**
- **JSON Data Binding with Jackson**

    - **Spring** uses the **Jackson Project** behind-the-scenes
    - **Jackson** handles **data binding** between **JSON** and **Java POJO**
- **Jackson Data Binding**
    - By default, **Jackson** will call the appropriate **getter/setter** method
- **Spring and Jackson Support**
    - **Spring** will **automatically** handle **Jackson** integration
    - **JSON data** being passed to **REST controller** is converted to **POJO**
    - **Java object** being returned from **REST controller** is converted to **JSON**

Spring Boot REST POJO

- **Development Process**
    - Create **Java POJO** class for **Student**
    - Create **Spring REST Service** using **@RestController**

Spring Boot REST Path Variables

- **Development Process**
    - Add **request mapping** to **Spring REST Service**
        - Bind **path variable** to **method parameter** using **@PathVariable**

Spring Boot REST Exception Handling

- We want to **handle exceptions** and **return errors** as **JSON**
- **Development Process**
    - Create a **custom error response class**
    - Create a **custom exception class**
    - Update **REST service** to **throw exception**
    - Add an **exception handler method** using **@ExceptionHandler**

Spring Boot Rest Global Exception Handling

- **Global Exception Handlers**
    - Promote **reuse**
    - **Centralizes** exception handling
- **Spring @ControllerAdvice**
    - Similar to **interceptor/filter**

    - **Pre-process requests** to controllers
    - **Post-process responses** to handle exceptions
    - Perfect for **global exception handling**
    - Real-time use of **Aspect-Oriented Programming (OAP)**
- **Development Process**
    - Create new **@ControllerAdvice**
    - **Refactor REST service** → remove exception handling code
    - Add exception handling code to **@ControllerAdvice**

Spring Boot Rest API Design - Best Practices

- **REST API Design**
    - **Who** will use your API?
    - **How** will the API be used?
    - Design the API based on **requirements**
- **API Design Process**
    - Review API **requirements**
    - **Identify** main resource / entity
    - Use HTTP methods to **assign action** on resource
- Step 1: **Review API Requirements**
    - Determine **what APIs must be created**
- Step 2: **Identify main resource / entity**
    - Look for the most prominent **noun** in the problem
    - Convention is to use **plural form** of resource
        - **.../api/employees**
- Step 3: **Use HTTP methods to assign action on resource**

HTTP Method	CRUD Action
POST	**C**reate a new entity
GET	**R**ead a list of entities or single entity

PUT	**U**pdate an existing entity
DELETE	**D**elete an existing entity

- **Antipatterns**
  - **Don't include action verbs** in the endpoint
    - .../**api/deleteEmployee**

Spring Boot REST Project Overview

- **API Requirements**
  - **Get** a list of **employees**
    - GET
    - .../api/employees
  - **Get** a single **employee** by **ID**
    - GET
    - .../api/employees/{employeeId}
  - **Add** a new **employee**
    - POST
    - .../api/employees → With a body
  - **Update** an **employee**
    - PUT
    - .../api/employees
  - **Delete** an **employee**
    - DELETE
    - .../api/employees/{employeeId}
- **Development Process**
  - Set up **database development environment**
  - Create a **Spring Boot** project using **Spring Initializr**
  - **Get** a **list** of **employees**
  - **Get** a single **employee** by **ID**
  - **Add** a new **employee**
  - **Update** an existing **employee**

    - **Delete** an existing **employee**

Spring Boot Define Service Layer

- **Purpose** of **Service Layer**
    - **Service Facade** design pattern
    - Intermediate layer for **custom business logic**
    - **Integrate data** from **multiple sources** (DAO/repositories)
- Most times → **Delegate calls**
- Specialized **annotation** for **services**
    - **@Service** annotation
    - **Spring** will **automatically register** the **Service implementation**
        - Due to **component scanning**

Spring Boot DAO: Add, Update, Delete

- **Service Layer - Best Practice**
    - Apply **transactional boundaries** at the **service layer**
    - It's the **service layer's responsibility** to **manage transaction boundaries**
    - For **implementation code**
        - Apply **@Transactional** on **service methods**
        - Remove **@Transactional** on **DAO methods** if they already exist

Spring Boot REST: Spring Data JPA

- **Spring Data JPA**
    - Create a **DAO**, plugging in your **entity type** and **primary key**
    - **Spring** will return a **CRUD implementation**
        - **Minimizes** boiler-plate DAO **code**
        - More than **70% reduction** in **code**
- **JpaRepository**
    - **Spring Data JPA** provides the **JpaRepository** interface
    - It exposes **methods** (some by **inheritance** from **parents**)
    - You only need the **entity** and **primary key**
- **Development Process**
    - Extend **JpaRepository** interface

    - Use your **repository** in the app
- **Advanced Features**
    - **Extending** and **adding custom queries** with **JPQL**
    - **Query Domain Specific Language (Query DSL)**
    - Defining **custom methods** (low-level coding)
    - **No need** for **@Transactional**

Spring Boot REST: Spring Data REST

- **Spring Data Rest**
    - Leverages your existing **JpaRepository**
    - **Spring** provides a **REST CRUD implementation**
        - **Minimizes** boiler-plate **REST code**
- **How** does it work?
    - **Spring Data REST scans** your **project** for **JpaRepository**
    - **Expose REST APIs** for each **entity type** for your **JpaRepository**
- **REST Endpoints**
    - By default, **Spring Data Rest** will **create endpoints** based on **entity type**
    - **Simple pluralized form**
        - **First character** of entity type is **lowercase**
        - **Append** entity with **"s"**
- **Development Process**
    - Add **Spring Data REST** to your **Maven POM** file
- What we **need**
    - Entity: **Employee**
    - JpaRepository: **EmployeeRepository extends JpaRepository**
    - Maven POM Dependency for: **spring-boot-starter-data-rest**
- **HATEOAS**
    - **Spring Data REST** endpoints are **HATEOAS** compliant
        - **H**ypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate
    - **Hypermedia-driven sites** provide information to **access REST interfaces**

        - Like **meta-data** for REST data
    - For a **collection**, **meta-data** includes **page size**, **total elements**, **pages**, etc.
    - Uses **Hypertext Application Language (HAL)** data format
- **Advanced Features**
    - **Pagination, sorting,** and **searching**
    - **Extending** and **adding custom queries** with **JPQL**
    - **Query Domain Specific Language (Query DSL)**

Spring Boot REST: Spring Data REST Configs and Sorting

- **REST Endpoints**
    - By default, **Spring Data REST** creates endpoints based on **entity** type
    - Simple, **pluralized form**
        - **First character** of **Entity** type is **lowercase**
        - **Append** with **"s"**
        - Spring Data REST **doesn't handle complex pluralized forms**
- Can **specify name/path** with an **annotation** on your **repository**
    - **@RepositoryRestResource(path="XXX")**
- **Pagination**
    - By default, **Spring Data REST** returns the **first 20 elements**
        - Page size = 20
    - Can **navigate pages** with **query params**
        - Pages are **zero-based**
- **Spring Data REST Configuration**

Name	Description
spring.data.rest.base-path	Base path used to expose repository resources

spring.data.rest.default-page-size	Default size of pages
spring.data.rest.max-page-size	Maximum size of pages
...	...

- **Sorting**
  - Can **sort by property names** of your **entity**
  - **Ascending** is default