Spring Data REST + JPA + H2

Spring Data REST works by creating a layer on top of Spring Data repositories. It automatically translates calls to these repositories into appropriate web services. Here's a brief overview of how to use Spring Data REST:

1. Maven dependencies

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.2.5</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.psatraining</groupId>
	<artifactId>springdatarestdemo</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>springdatarestdemo</name>
	<description>Demo project for Spring Boot</description>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-rest -->
		<dependency>
		  <groupId>org.springframework.boot</groupId>
		  <artifactId>spring-boot-starter-data-rest</artifactId>
		  <version>3.2.5</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/jakarta.persistence/jakarta.persistence-api -->
```

```
            <dependency>
              <groupId>jakarta.persistence</groupId>
              <artifactId>jakarta.persistence-api</artifactId>
              <version>3.1.0</version>
            </dependency>

            <dependency>
             <groupId>org.springframework.data</groupId>
             <artifactId>spring-data-rest-webmvc</artifactId>
             <version>4.2.5</version>
            </dependency>

        <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
          <groupId>com.h2database</groupId>
          <artifactId>h2</artifactId>
          <scope>runtime</scope>
        </dependency>

    </dependencies>
    <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
    </build>

</project>
```

2. Define Domain Entities

First, define your domain entities using JPA annotations:

```
package com.example.demo;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Book {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
```

```
  private String title;
  private String author;

  // constructor(s), getters and setters
```

3. Create Repository Interfaces

Next, define a repository interface for each entity:

```
package com.example.demo;

import org.springframework.data.repository.CrudRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(collectionResourceRel = "books", path = "books")
public interface BookRepository extends CrudRepository<Book, Long> {
}
```

4. Configure Spring Data REST

Spring Data REST configuration is minimal, often requiring no more than the inclusion of the appropriate Spring Boot starter and some minimal application properties, if any.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApplication {
  public static void main(String[] args) {
    SpringApplication.run(MyApplication.class, args);
  }
}
```

5. Access Your RESTful API

Once the application is running, Spring Data REST exposes the CRUD operations on Book at a path like /books. You can perform standard operations such as:

```
GET /books: List all books with pagination.
POST /books: Create a new book.
GET /books/{id}: Retrieve a specific book.
PUT /books/{id}: Update an existing book.
DELETE /books/{id}: Delete a book.
```

Note:

- ✓ In sending POST/PUT/PATCH request via postman, use form-data raw json format
  Example:

```
{
        “title”:”Peter Pan”,
        “author”:”Wendy”
}
```

Spring Data REST + MySQL

1. Create mysql database
2. Add the ff dependencies in pom.xml

pom.xml

```
...
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>com.mysql</groupId>
			<artifactId>mysql-connector-j</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
...
```

3. Add application properties to connect to MySQL

src/main/resources/application.properties

```
...
# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=john
spring.datasource.password=123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver		#or you can just leave this and the preceding line active
# JPA Configuration
```

```
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update

# spring.autoconfigure.exclude =
org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
...
```

4. Define Model Class

Book.java

```
package com.example.demo;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "books")

public class Book {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	@Column(name = "title")
	private String title;

	@Column(name = "author")
	private String author;

	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 Book() {
		super();
	}

	public Book(Long id, String title, String author) {
		super();
		this.id = id;
		this.title = title;
		this.author = author;
	}

}
```

5. Create Repository Interfaces

Next, define a repository interface for each entity:

```
package com.example.demo;

import org.springframework.data.repository.CrudRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(collectionResourceRel = "books", path = "books")
public interface BookRepository extends CrudRepository<Book, Long> {
}
```

6. Configure Spring Data REST

Spring Data REST configuration is minimal, often requiring no more than the inclusion of the appropriate Spring Boot starter and some minimal application properties, if any.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApplication {
  public static void main(String[] args) {
```

```
    SpringApplication.run(MyApplication.class, args);
  }
}
```

7. Access Your RESTful API

Once the application is running, Spring Data REST exposes the CRUD operations on Book at a path like /books. You can perform standard operations such as:

```
GET /books: List all books with pagination.
POST /books: Create a new book.
GET /books/{id}: Retrieve a specific book.
PUT /books/{id}: Update an existing book.
DELETE /books/{id}: Delete a book.
```

Note:

- ✓ In sending POST/PUT/PATCH request via postman, use form-data raw json format
  Example:

```
{
        “title”:”Peter Pan”,
        “author”:”Wendy”
}
```

- ✓ For training, you may generate data for your table using online tools like: https://filldb.info/