Spring Boot - REST Example

Last Updated : 23 Jul, 2025

In modern web development, most applications follow the Client-Server Architecture. The Client **(frontend)** interacts with the server **(backend)** to fetch or save data. This communication happens using the HTTP protocol. On the server, we expose a bunch of services that are accessible via the HTTP protocol. The client can then directly call the services by sending the HTTP request.

Now, this is where **REST** comes into the picture. Rest stands for **R**epresentation **S**tate **T**ransfer. It is a convention to build these HTTP services. So we use a simple HTTP protocol principle to provide support to **CREATE**, **READ**, **UPDATE,** & **DELETE** data, commonly referred to as **CRUD** operations. In short, it is a set of rules that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL.

Let's implement a REST application and understand the REST approach by creating an example where we simply return the Book data in the form of **JSON**.

Step-by-Step Guide to Build a RESTful API using SpringBoot

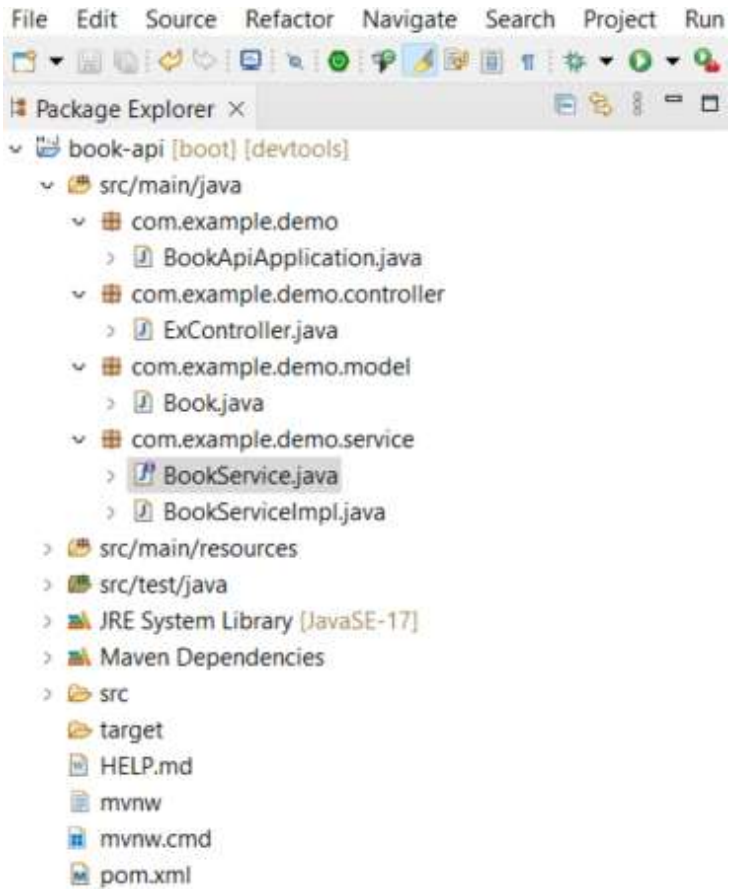
Step 1: Setting up the Spring Boot Project

So first, we will set up the spring project in STS IDE. The steps are outlined below:

- Click File -> New -> Project -> Select Spring Starter Project -> Click Next.
- A New Dialog box will open where you will provide the project-related information like project name, Java version, Maven version, and so on.
- After that, select required maven dependencies like **Spring Web**, and **Spring Boot DevTools** (which provide fast application restarts, LiveReload, and configurations for an enhanced development experience).
- Click Finish to generate the project.

Step 2: Project Structure

Once the project is created, we will see the following structure



pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="https://maven.apache.org/POM/4.0.0"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://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>3.4.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>book-api</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>book-api</name>
    <description>Demo project for Spring Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

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

</project>
```

Note: Since we are not using a database in this example, no configuration is needed in application.properties.

Step 3: Create the POJO Class

A POJO class represents the data model. In this case, we'll create a Book class.

Book.java:

```
package com.example.demo.model;

public class Book {
    private int id;
    private String title;
    private String author;

    public Book(int id, String title, String author)
    {
        this.id = id;
        this.title = title;
        this.author = author;
    }
    // Getters and Setters
    public int getId() { return id; }
    public void setId(int 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;
    }
}
```

Step 4: Create the Service Interface and Service Implementation Class

Here, we have created an interface called **BookService** which contains all the service methods that our application is going to provide to the user. And **BookServiceImpl** class that implements the BookService interface.

Java Java

```java
package com.example.demo.service;
import com.example.demo.model.Book;

import java.util.List;

public interface BookService {
    List<Book> findAllBooks();
    Book findBookById(int id);
    void deleteAllBooks();

}
```

Step 5: Create the REST Controller

The Controller layer exposes the APIs to the client. Create a RestController class to handle HTTP requests.

ExController.java:

```java
package com.example.demo.controller;

import com.example.demo.model.Book;
import com.example.demo.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
public class ExController {
     @Autowired
        private BookService bookService;

        @GetMapping("/")
        public String home() {
            return "Welcome to the Book API!";
        }

        @GetMapping("/findbyid/{id}")
        public Book findBookById(@PathVariable int id) {
```

```
            return bookService.findBookById(id);
        }

        @GetMapping("/findall")
        public List<Book> findAllBooks() {
            return bookService.findAllBooks();
        }

        @DeleteMapping("/delete")
        public String deleteAllBooks() {
            bookService.deleteAllBooks();
            return "All books have been deleted.";
        }

}
```

Note: *Here is the* ***ExController*** *class where we are exposing all the APIs which we have created.*

API's list

- ***http://localhost:8080/****: To save the data*
- ***http://localhost:8080/findbyid/2****: Find Book by id*
- ***http://localhost:8080/findall****: Find all books*
- ***http://localhost:8080/delete****: Delete all books*

Step 6: Run the Application

The main application class, BookApiApplication, is automatically generated by Spring Boot. It contains the main method to run the application.

BookApiApplication.java:

```
package com.example.demo;

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

@SpringBootApplication
public class BookApiApplication {
```

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

Step 7: Test the APIs

Test the APIs using Postman to verify they work as expected

0:00

Comment More info Advertise with us