Spring Data JPA CRUD API

Development:

Marcelo Eduardo Guillen Castillo

Java Monterrey Academy

September 5,, 2024

Introduction

Business case

Based on the project made on team A from the Spring Boot JPA project on the last document, the team B is now in charge of the refactor code from the previous one, now adding the new feature of Spring Data JPA, a dependency for the Spring framework which can make the implementation of the data persistence more efficient and can require less code for at least the basic functions of our service, in these case the service is the same as the team's A project, a simple library where our employees could access any time of the day.

Objectives

To improve the service weight on the server and process less code, also, less instructions to the server to do the same activities.
Improve the persistence of data to the database and also improve security about possible code injection.

Requirements

- A database that could store the information about the books.
- The user can access all the book's information.
- The user can access information from a concrete book.
- The user can modify the information of a book.
- The user can delete a book.

Installation

The engineer needs to install the JDK 17 to beyond, then any IDE that could maintain Maven projects, and finally install all the dependencies and plugins necessary to make the project work correctly. This is made in Maven, so please enter the following code into your POM file.

Unset
```
<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>3.3.2</version>
              <relativePath />
              <!--  lookup parent from repository  -->
       </parent>
       <groupId>group</groupId>
       <artifactId>MarceloG_HW2_SpringDataJPA</artifactId>
       <version>0.0.1-SNAPSHOT</version>
       <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>org.springframework.boot</groupId>
                     <artifactId>spring-boot-devtools</artifactId>
                     <scope>runtime</scope>
                     <optional>true</optional>
              </dependency>
              <dependency>
                     <groupId>com.mysql</groupId>
                     <artifactId>mysql-connector-j</artifactId>
                     <scope>runtime</scope>
              </dependency>
              <dependency>
                     <groupId>org.springframework.boot</groupId>
                     <artifactId>spring-boot-starter-test</artifactId>
                     <scope>test</scope>
              </dependency>
              <dependency>
                     <groupId>org.projectlombok</groupId>
                     <artifactId>lombok</artifactId>
                     <scope>provided</scope>
              </dependency>
       </dependencies>
       <build>
              <plugins>
```

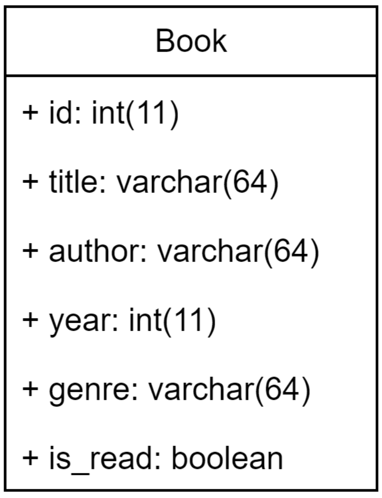
```
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Also we use a database on MySQL, being modified with MySQL Workbench, but you can use any kind of IDE for MySQL databases, on the file there are a “sql” folder which contains the sql queries for the user that this app use and another one with the table information for this service.
We use postman for our manual tests, a software for web service testing, but you can use any tool or extension you want in order to verify the correct functionalities of the service.

Project architecture

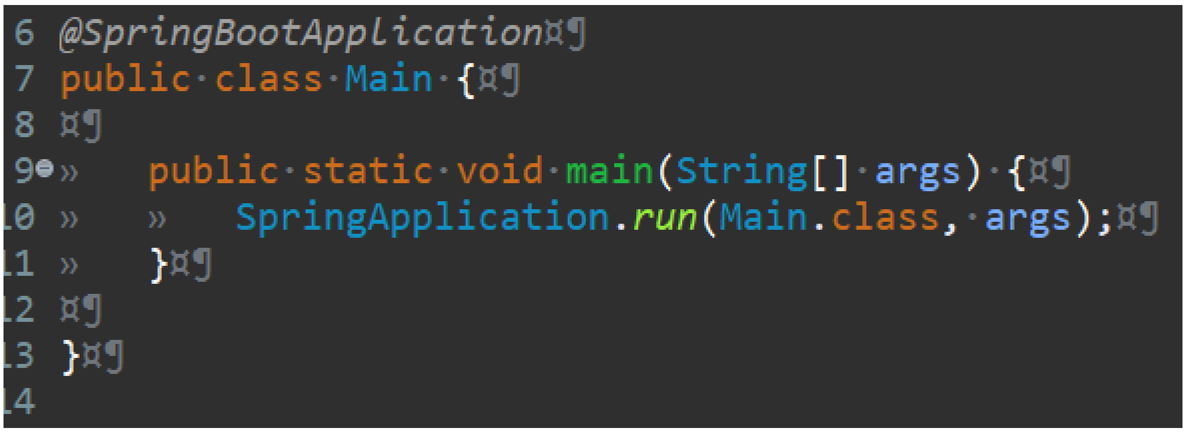
Our library can handle books as its data, and has the following characteristics:

- ID: identification number.
- Title: name of the book.
- Author: name of the author.
- Year: publication years.
- Genre: type of genre of the book.
- Is read: specifies if the book was being read.



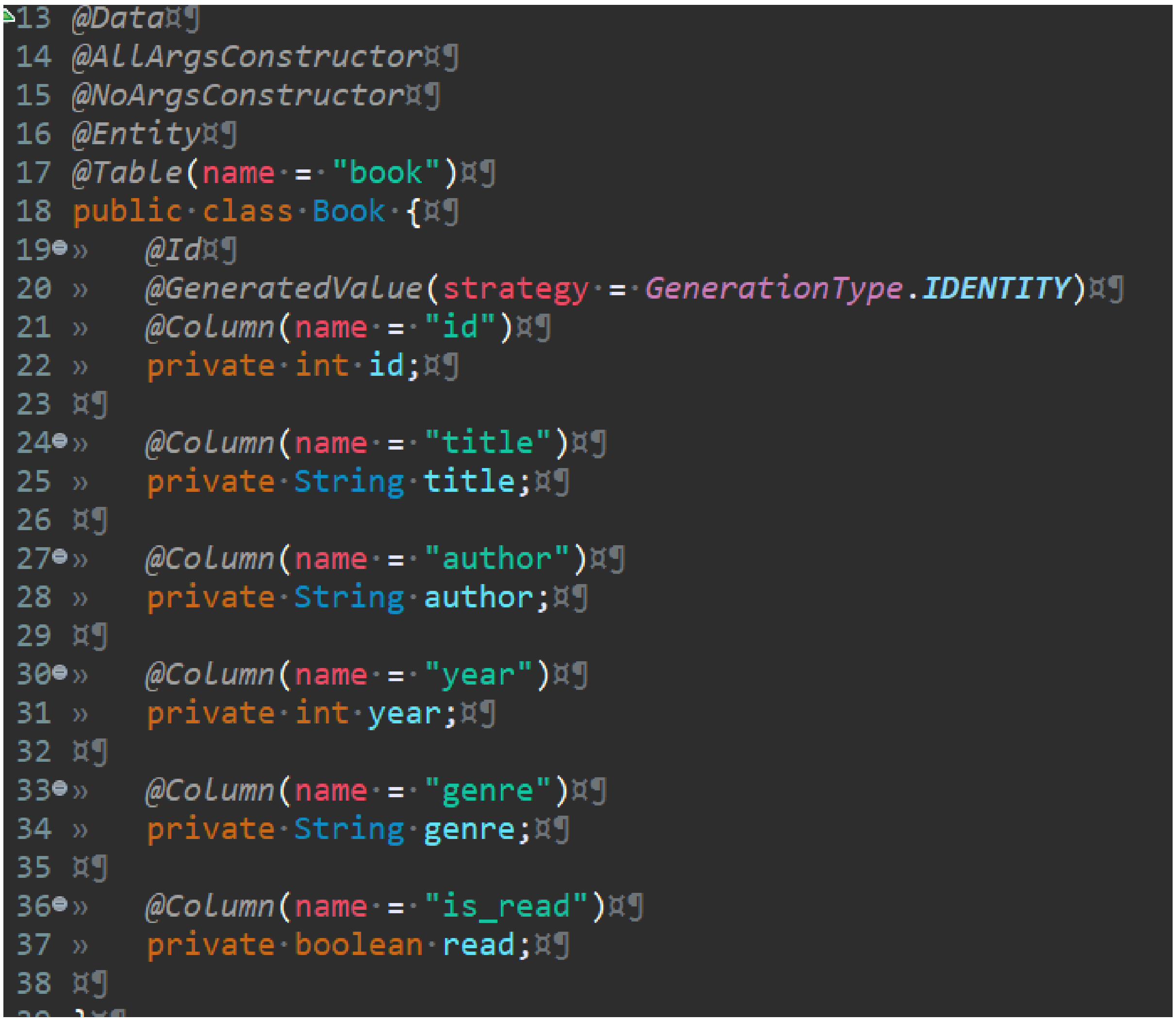
Demonstrations

Here are some code demonstrations, some of the implementations about functionalities of the software.
First of all we create a Main class with it’s typical main method, then we declare to the compiler that our Main class will be the Spring Boot Application by annotation, and then run the Spring Application with the Main class and it’s arguments



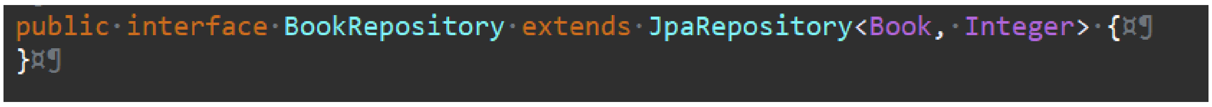
```
@SpringBootApplication
public class Main {

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

}
```

Then we create a class Book to be intended as a Spring Entity, with its attributes.



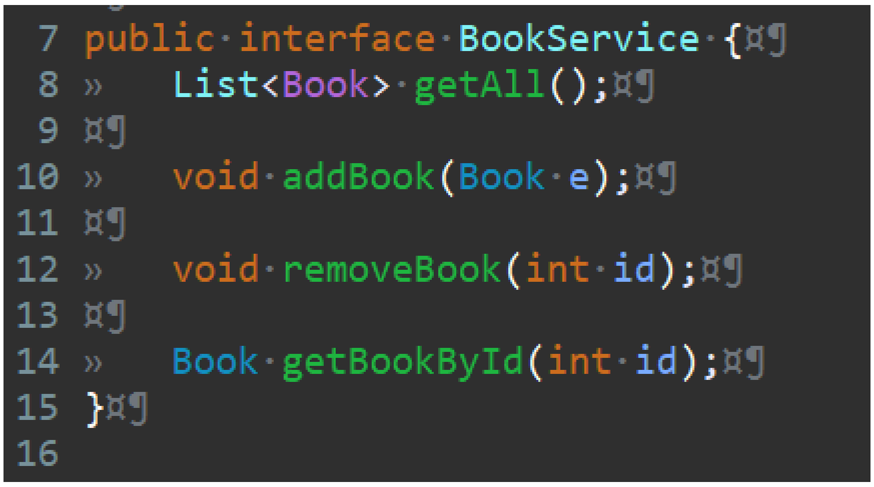
```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "book")
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private int id;

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

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

	@Column(name = "year")
	private int year;

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

	@Column(name = "is_read")
	private boolean read;

}
```

Instead of manually creating the data access object and its methods, we will make use of Spring Data JPA feature, JPA repository. A JPA Repository will integrate all basic functionalities of obtaining data from a database of that entity in particular like obtaining all data in a list, obtaining a singular data by giving its id, etc.

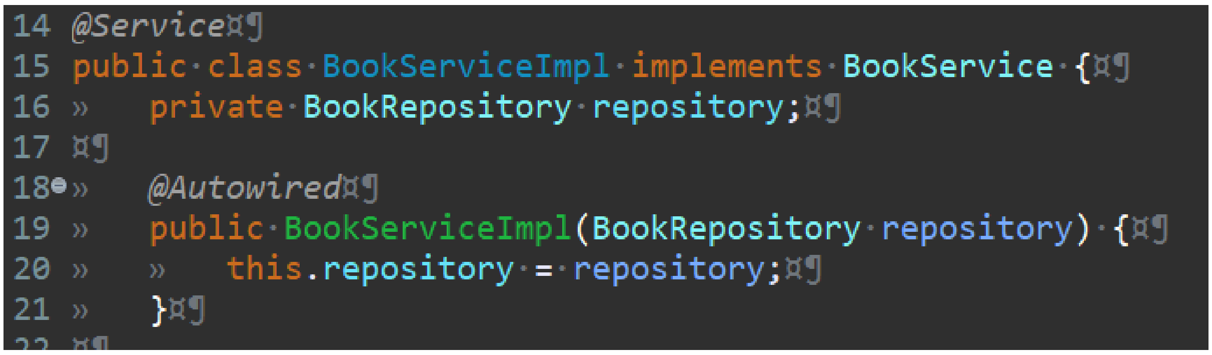


```java
public interface BookRepository extends JpaRepository<Book, Integer> {
}
```

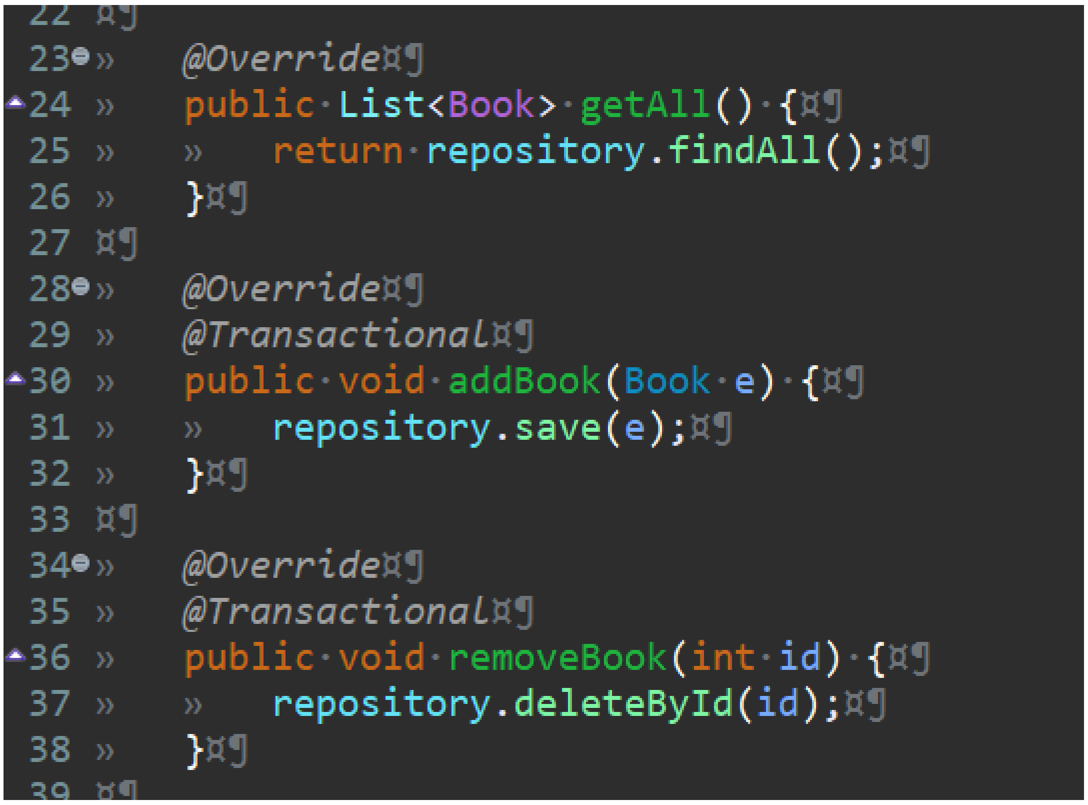
Then as usual, we create the service that will need a repository book being injected, also its methods and each one indicating if it needs to be transactional.



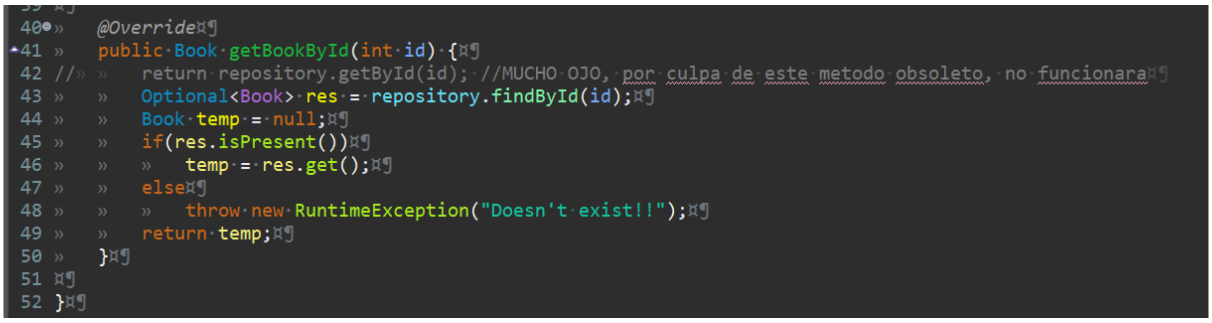
```java
public interface BookService {
	List<Book> getAll();

	void addBook(Book e);

	void removeBook(int id);

	Book getBookById(int id);
}

```



```java
@Service
public class BookServiceImpl implements BookService {
	private BookRepository repository;

	@Autowired
	public BookServiceImpl(BookRepository repository) {
		this.repository = repository;
	}
```

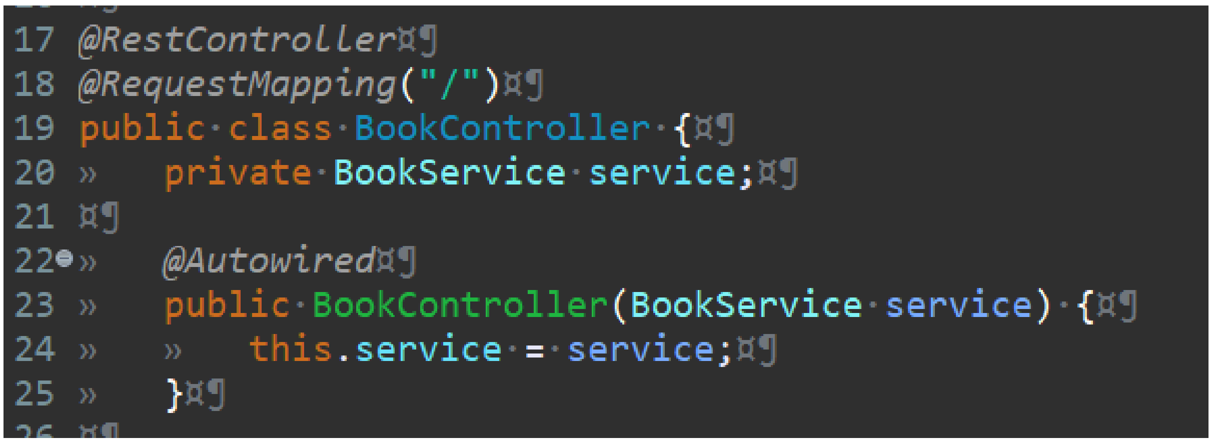


```java

	@Override
	public List<Book> getAll() {
		return repository.findAll();
	}

	@Override
	@Transactional
	public void addBook(Book e) {
		repository.save(e);
	}

	@Override
	@Transactional
	public void removeBook(int id) {
		repository.deleteById(id);
	}

```

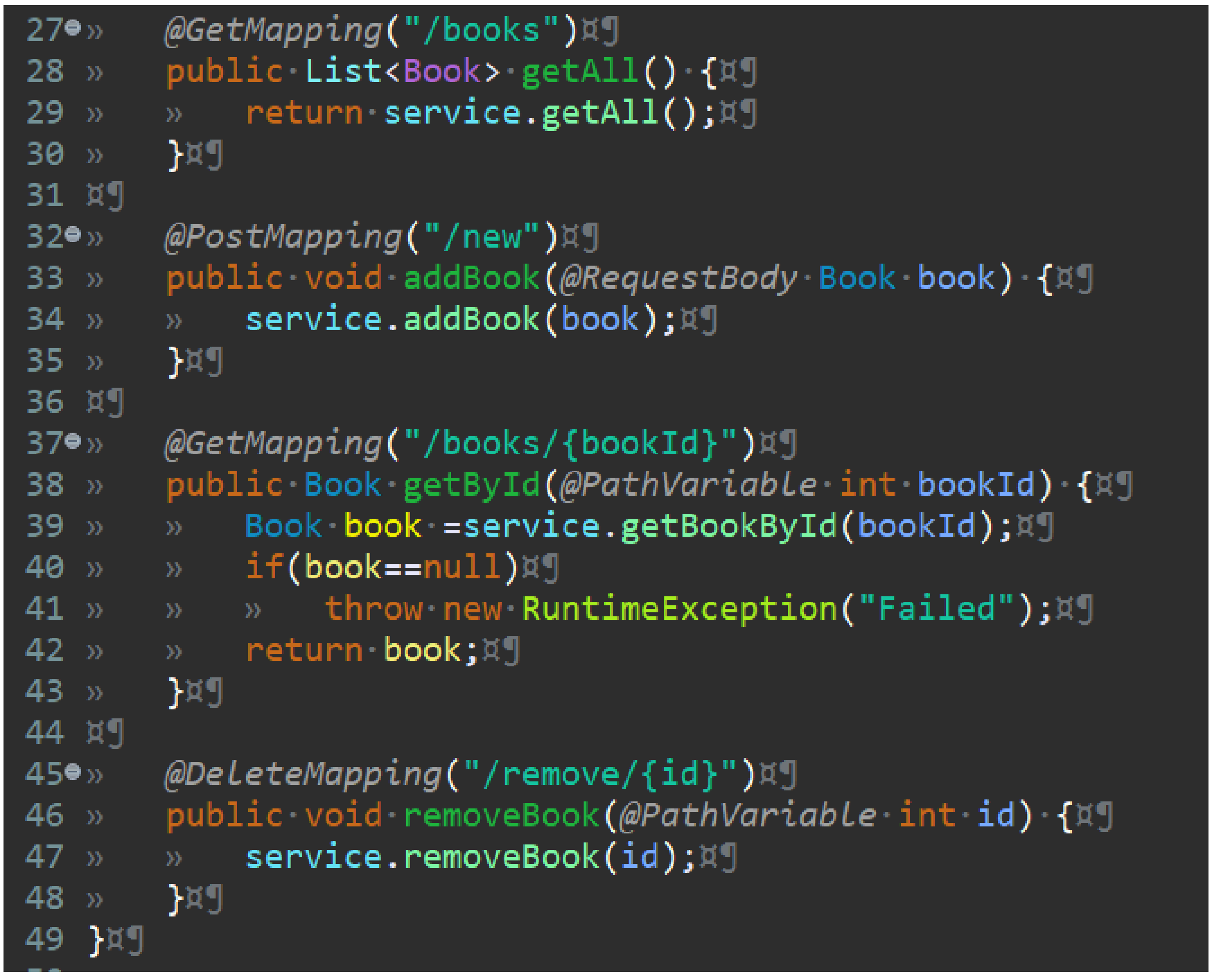


```java
    @Override
    public Book getBookById(int id) {
//      return repository.getById(id); //MUCHO OJO, por culpa de este metodo obsoleto, no funcionara
        Optional<Book> res = repository.findById(id);
        Book temp = null;
        if(res.isPresent())
            temp = res.get();
        else
            throw new RuntimeException("Doesn't exist!!");
        return temp;
    }

}
```

Then we need to create the Rest Component with its methods to run our code.



```java
@RestController
@RequestMapping("/")
public class BookController {
    private BookService service;

    @Autowired
    public BookController(BookService service) {
        this.service = service;
    }
```



```java
    @GetMapping("/books")
    public List<Book> getAll() {
        return service.getAll();
    }

    @PostMapping("/new")
    public void addBook(@RequestBody Book book) {
        service.addBook(book);
    }

    @GetMapping("/books/{bookId}")
    public Book getById(@PathVariable int bookId) {
        Book book =service.getBookById(bookId);
        if(book==null)
            throw new RuntimeException("Failed");
        return book;
    }

    @DeleteMapping("/remove/{id}")
    public void removeBook(@PathVariable int id) {
        service.removeBook(id);
    }
}
```

Conclusions

Thanks to many dependencies, creating web applications has never been so easy, you can just implement many functionalities in much less time and effort, and can even help the developer to implement less code to do the same tasks. It is just the development team's responsibility to explore all the possibilities with these tools and limitations in order to domain that technology and be very productive when designing the web service application.