Spring Boot: Basics

Development:

Marcelo Eduardo Guillen Castillo

Java Monterrey Academy

September 5,, 2024

Introduction

Business case

Spring is a framework very well known for Back-End development and web services, which can have many dependencies that can add new functionalities or improvements into the application service, in this case, we are going to make a simple Spring Boot application that could maintain a simple service.
Our company needs an application web that could make access to their library and make all the simplest actions possible for a CRUD application. Now we are on team A, and are in charge of creating a little service of access and modification of the library books, so our employees could have access to their information and modifications, meanwhile other teams work on other projects.

Objectives

In order to have a simple application which could supply its services for like the following 4 weeks of its life, while our team B is planning for the new software with Spring Data JPA (which is an improvement of our actual service in progress). By this way, our company could provide a temporary service for our employees meanwhile team B is planning to create a better and refactored version of our team A project.

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_CRUDAPI</artifactId>
       <version>0.0.1-SNAPSHOT</version>
       <properties>
              <java.version>17</java.version>
       </properties>
       <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>
```

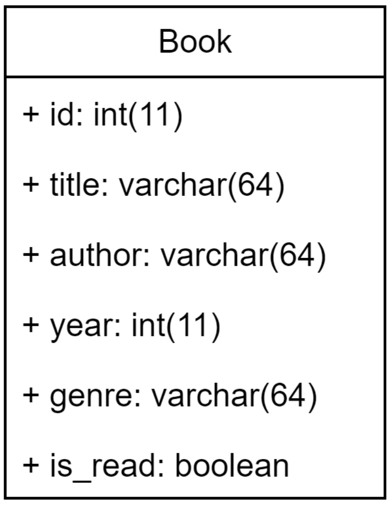
```
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <scope>provided</scope>
            </dependency>
        </dependencies>

        <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

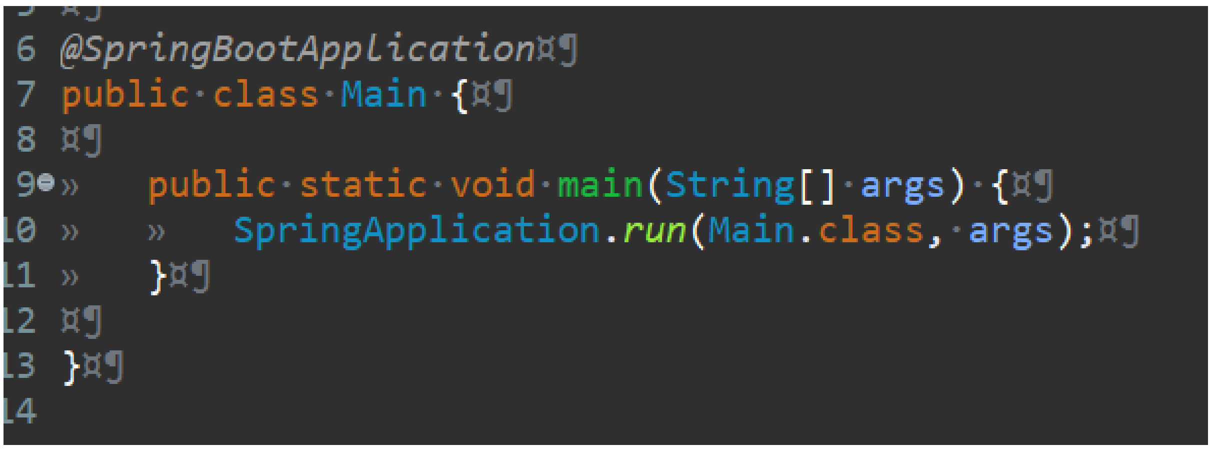
Our library can handle books as its data, and has the following characteristics:

- 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.



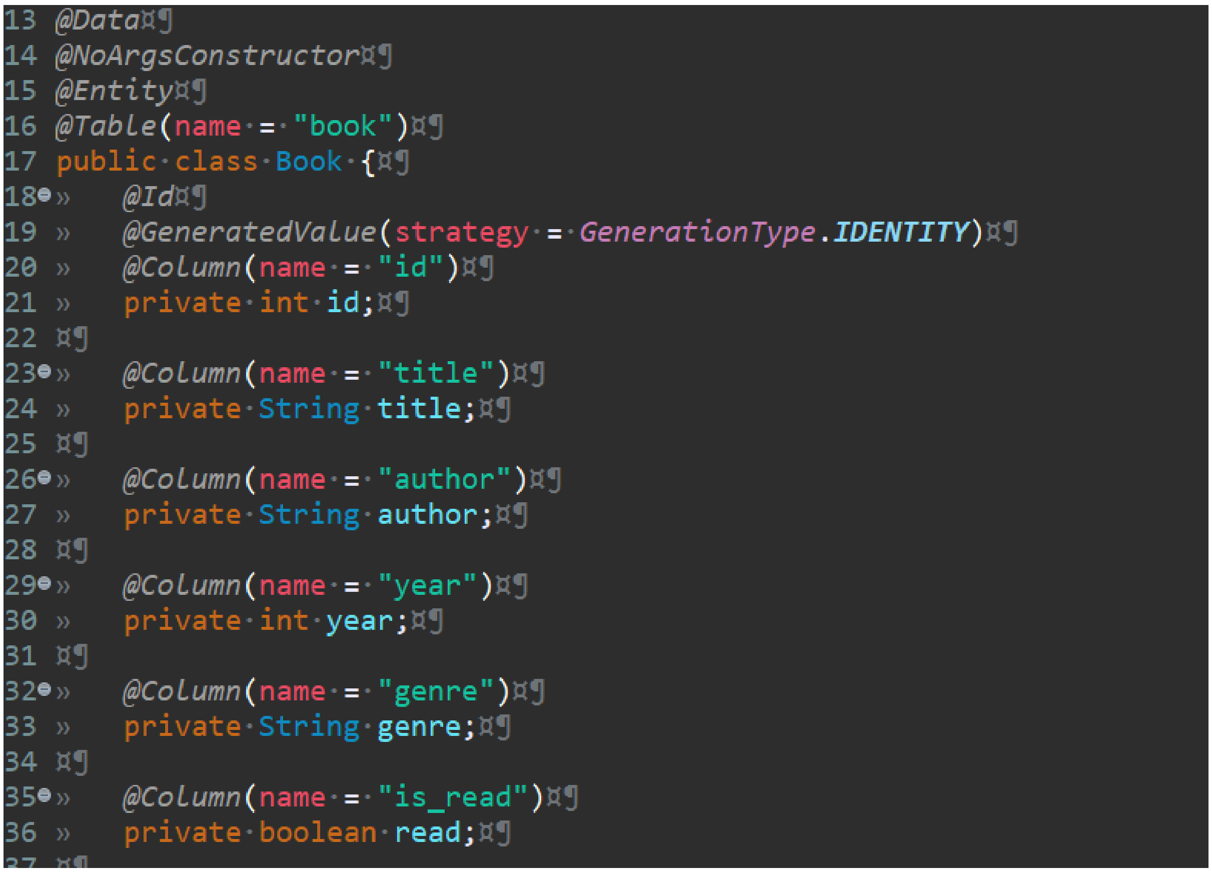
Demonstrations

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



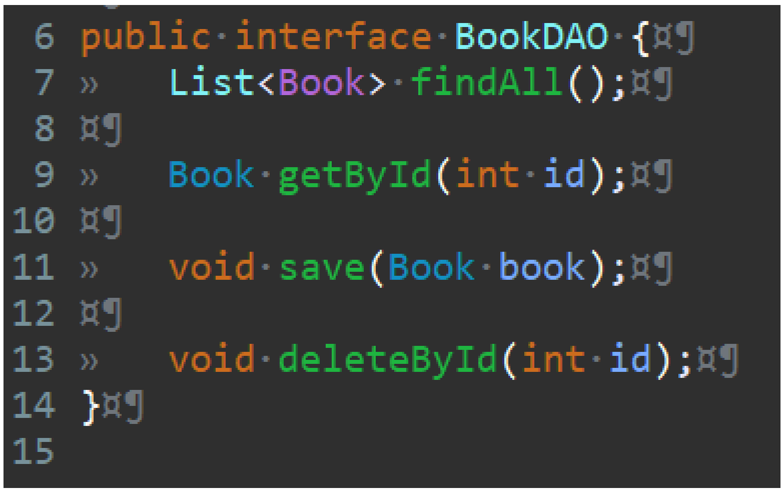
```java
@SpringBootApplication
public class Main {

	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.



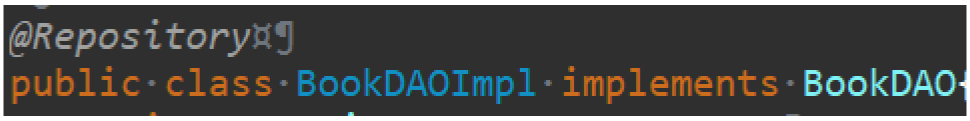
```java
@Data
@NoArgsConstructor
@Entity
@Table(name = "book")
public class Book {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private int id;

	@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;
```

We need an interface of the data access object or DAO, which will make the work to persist and modify the data directly into the database.



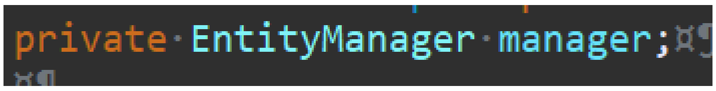
```java
public interface BookDAO {
	List<Book> findAll();

	Book getById(int id);

	void save(Book book);

	void deleteById(int id);
}
```

Then we create the implementation of that DAO, where we need to autowire the constructor in order to make sure Spring will instance.

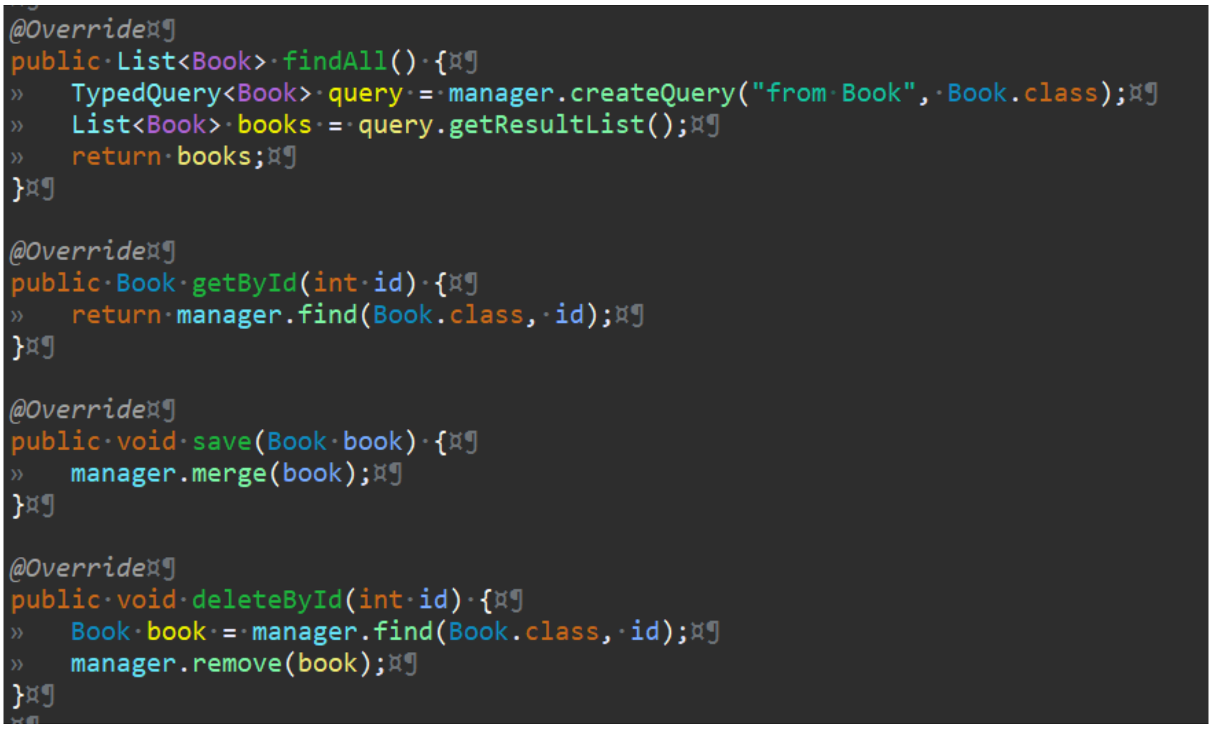


```java
@Repository
public class BookDAOImpl implements BookDAO
```

It will need an Entity Manager, the object that will interact with the database.

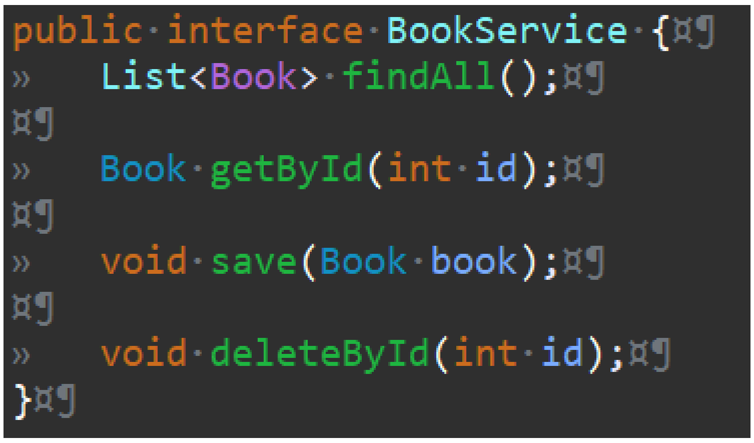


```java
private EntityManager manager;
```



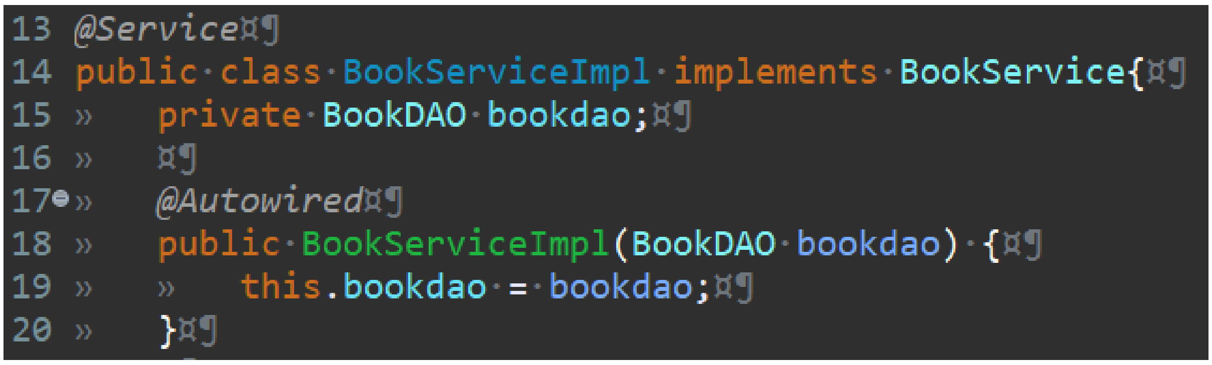
```java
@Override
public List<Book> findAll() {
	TypedQuery<Book> query = manager.createQuery("from Book", Book.class);
	List<Book> books = query.getResultList();
	return books;
}

@Override
public Book getById(int id) {
	return manager.find(Book.class, id);
}

@Override
public void save(Book book) {
	manager.merge(book);
}

@Override
public void deleteById(int id) {
	Book book = manager.find(Book.class, id);
	manager.remove(book);
}
```

Then we need to create the interface of the service for the books, with its methods.



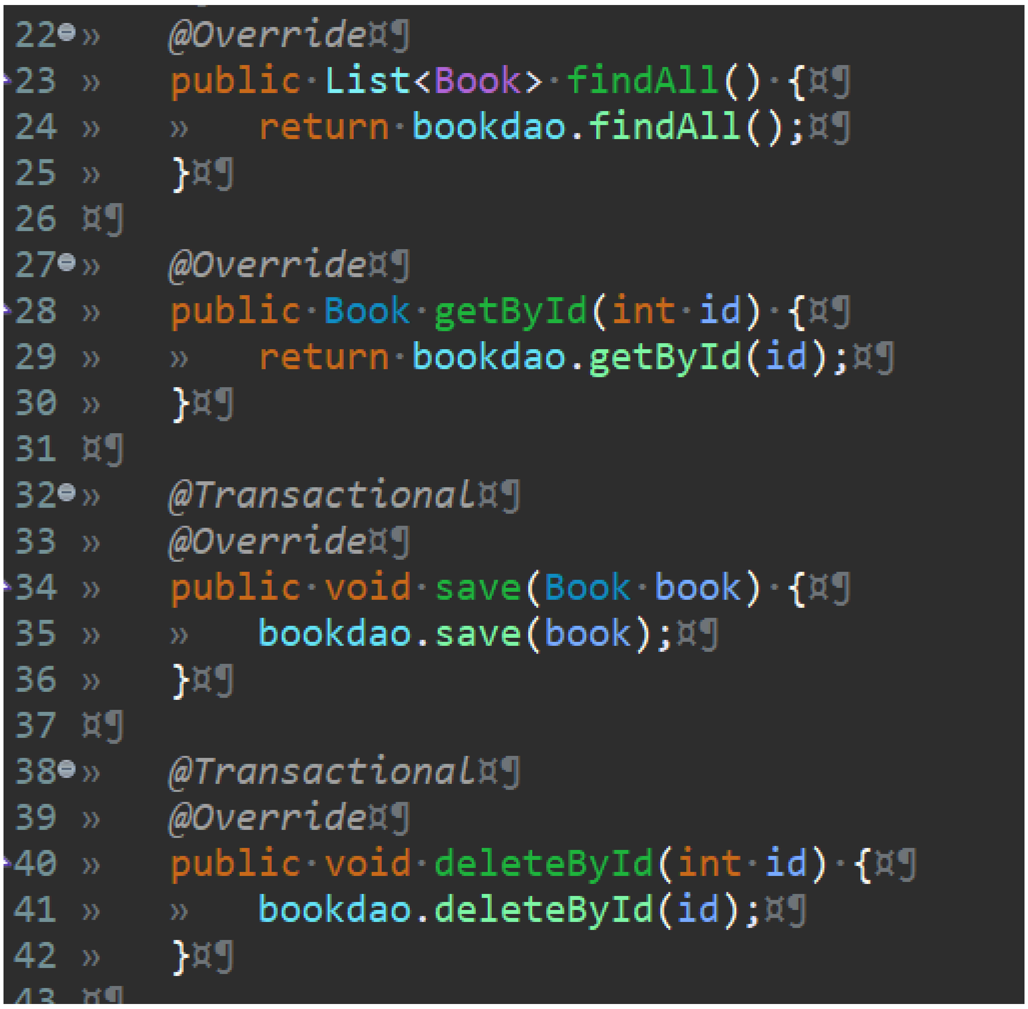
```java
public interface BookService {
	List<Book> findAll();

	Book getById(int id);

	void save(Book book);

	void deleteById(int id);
}
```

Then on the implementation of this class we need to declare the DAO of the book, by autowire.



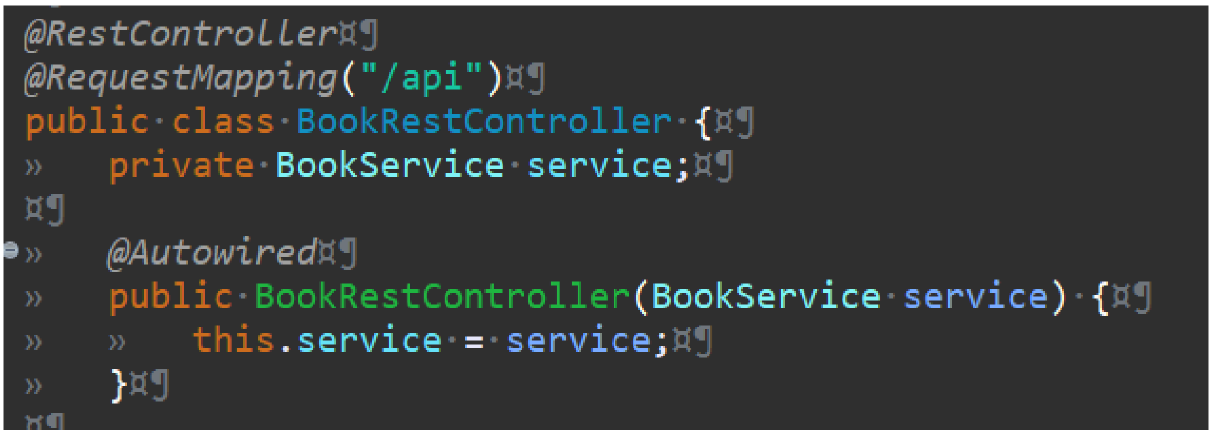
```java
@Service
public class BookServiceImpl implements BookService{
	private BookDAO bookdao;
	
	@Autowired
	public BookServiceImpl(BookDAO bookdao) {
		this.bookdao = bookdao;
	}
```

And then create the methods, indicating which ones we want to be transactional.



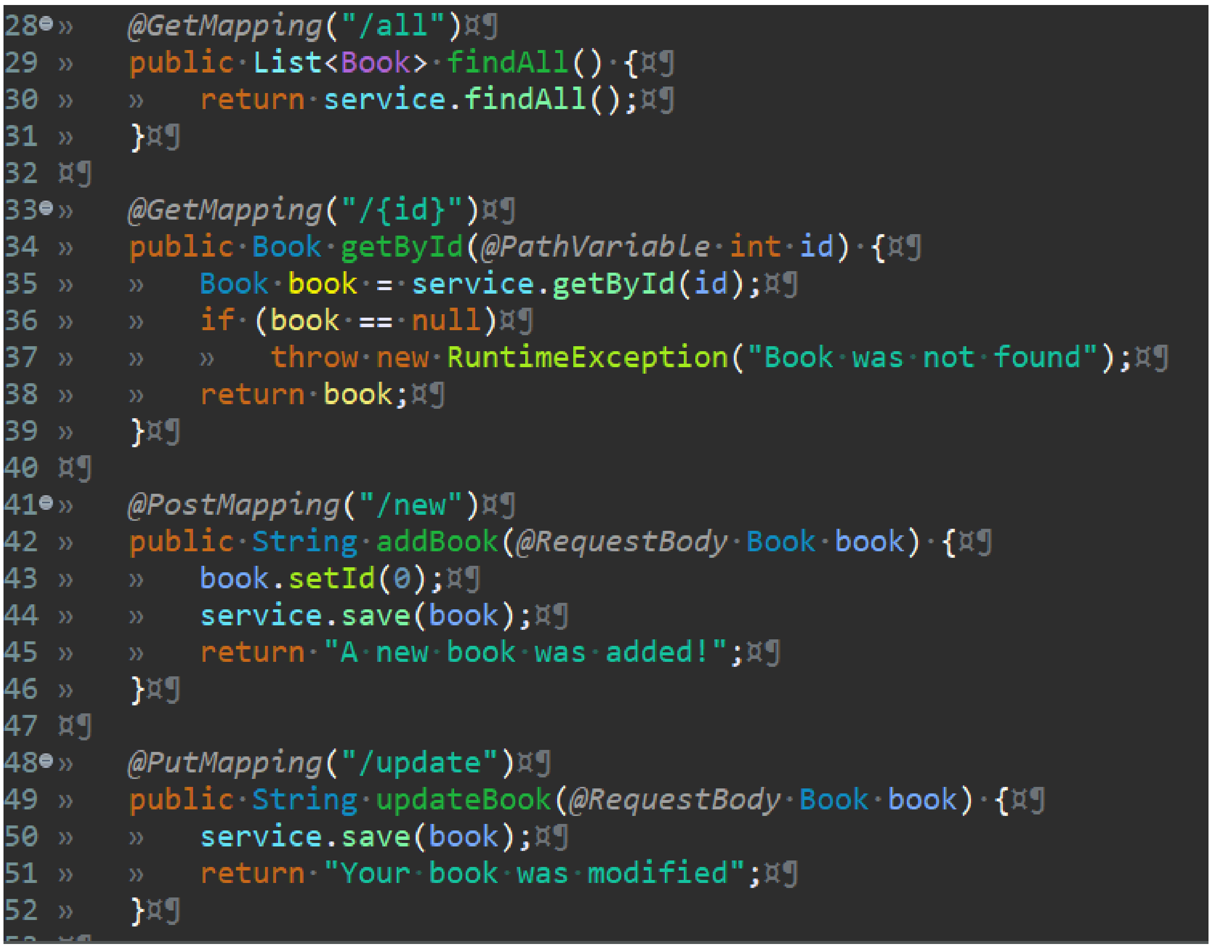
```
	@Override
	public List<Book> findAll() {
		return bookdao.findAll();
	}

	@Override
	public Book getById(int id) {
		return bookdao.getById(id);
	}

	@Transactional
	@Override
	public void save(Book book) {
		bookdao.save(book);
	}

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

```

And finally we create the implementation of the Rest Controller, which will create the http methods for our Postman (or any tool for web testing) to make use of it. We will need to create a book service attribute and inject it with Spring annotations.



```
@RestController
@RequestMapping("/api")
public class BookRestController {
	private BookService service;

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

```

Then we create all the methods that our web application will support.



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

	@GetMapping("/{id}")
	public Book getById(@PathVariable int id) {
		Book book = service.getById(id);
		if (book == null)
			throw new RuntimeException("Book was not found");
		return book;
	}

	@PostMapping("/new")
	public String addBook(@RequestBody Book book) {
		book.setId(0);
		service.save(book);
		return "A new book was added!";
	}

	@PutMapping("/update")
	public String updateBook(@RequestBody Book book) {
		service.save(book);
		return "Your book was modified";
	}
```

Conclusions

Web technologies are now more accessible and easier to use than before, and thanks to the “magic” of these tools, we can build great applications and services with less effort than before and in a more productive way, less time consuming and resources.