# S02 SPINGBOOT BACKEND

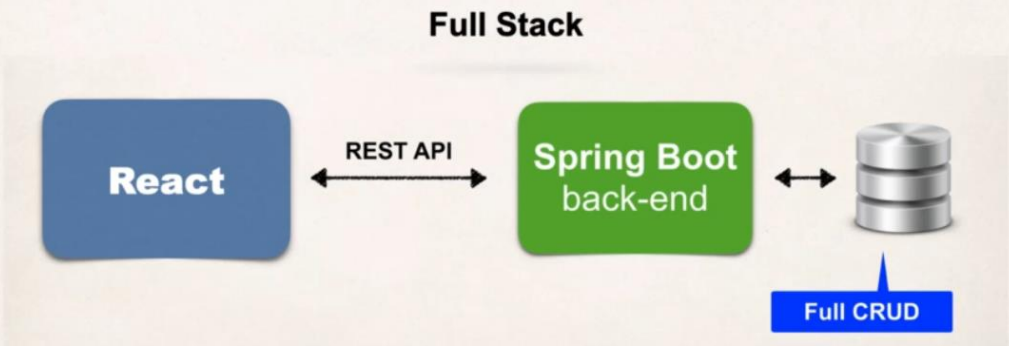
## 1. Preparation Tools

### Java Development Environment

- We assume that you are already an experienced Spring Boot Developer
- You should have the following items already installed
  - Java Development Kit (JDK)
  - Java IDE (we'll use IntelliJ in the videos, but any Java IDE will work)
  - Maven
  - MySQL Database and MySQL Workbench

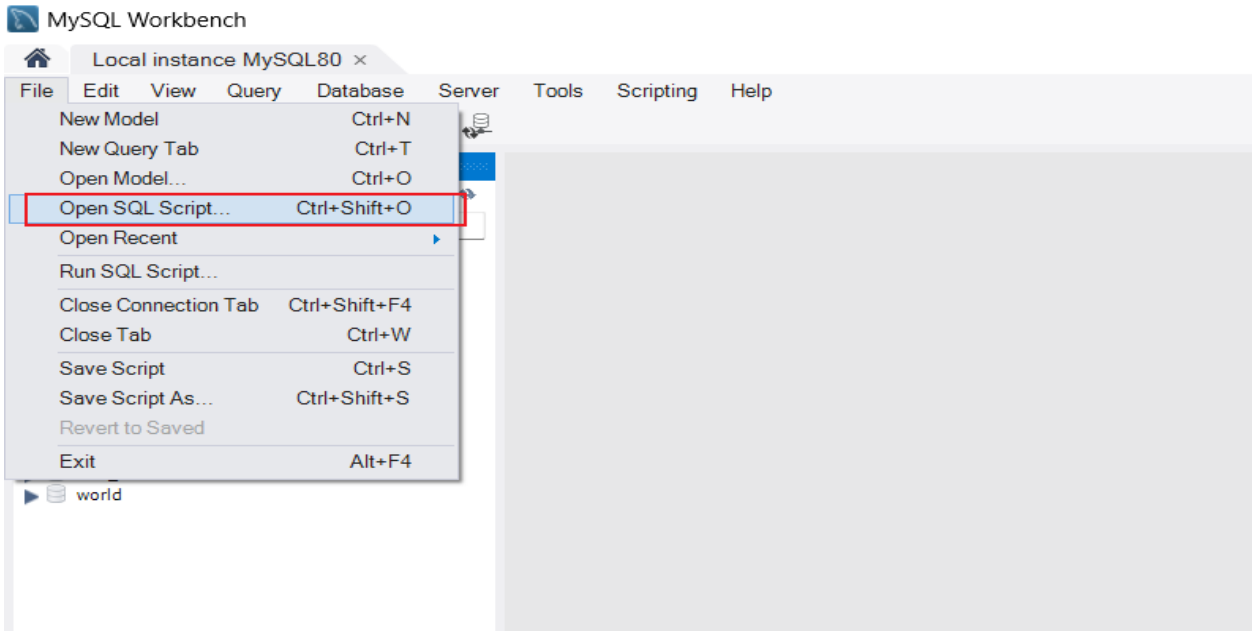
### Spring Boot Back End

- Leverage Spring Data REST for REST API
- Minimizes the coding for Spring Boot back end

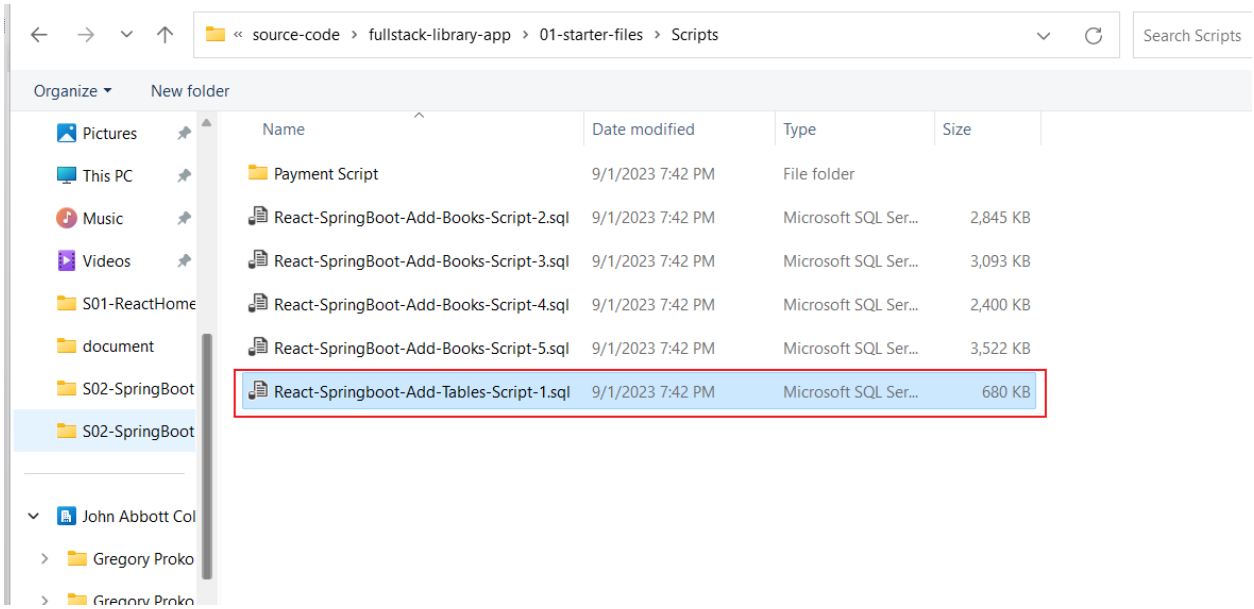


# 2. Setup Database

open MySQL Workbench, click on “file”, choose “open SQL Script”




choose the script 1 in our “starter-files”, then click on “open”.



Click on the “execute sign” , to execute our scripts.

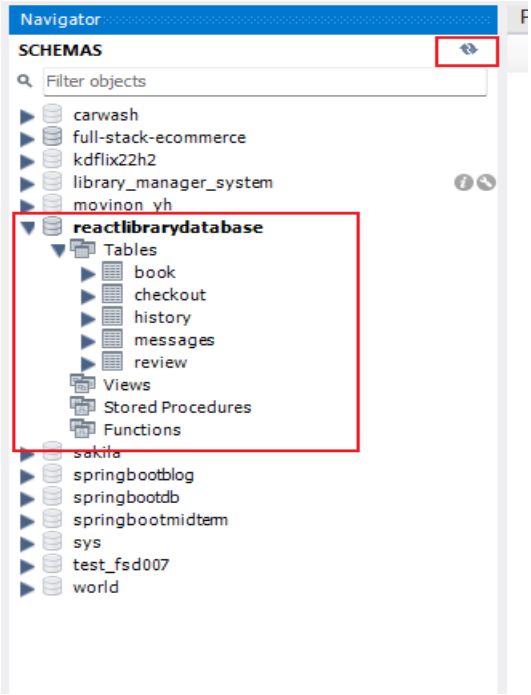
```
/*!40101 SET character_set_client = @saved_cs_client */;

-- INSERT INTO `checkout` VALUES
--  (1, 'example1user@email.com', '2022-05-22', '2022-06-25', 1),
--     (2, 'example2user@email.com', '2022-05-22', '2022-06-26', 1),
--     (3, 'example1user@email.com', '2022-05-22', '2022-06-01', 2);



DROP TABLE IF EXISTS `messages`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `messages` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `user_email` varchar(45) DEFAULT NULL,
  `title` varchar(45) DEFAULT NULL,
  `question` text DEFAULT NULL,
  `admin_email` varchar(45) DEFAULT NULL,
  `response` text DEFAULT NULL,
  `closed` tinyint(1) DEFAULT 0,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

-- INSERT INTO `messages` VALUES
--  (1, 'example3user@email.com', 'What should we do here?', 'I have been trying to work this out for a long time and no ma
--     (2, 'example2user@email.com', 'What should we do here example 2 user?', 'I have been trying to work this out for a l


DROP TABLE IF EXISTS `history`;
```

Output

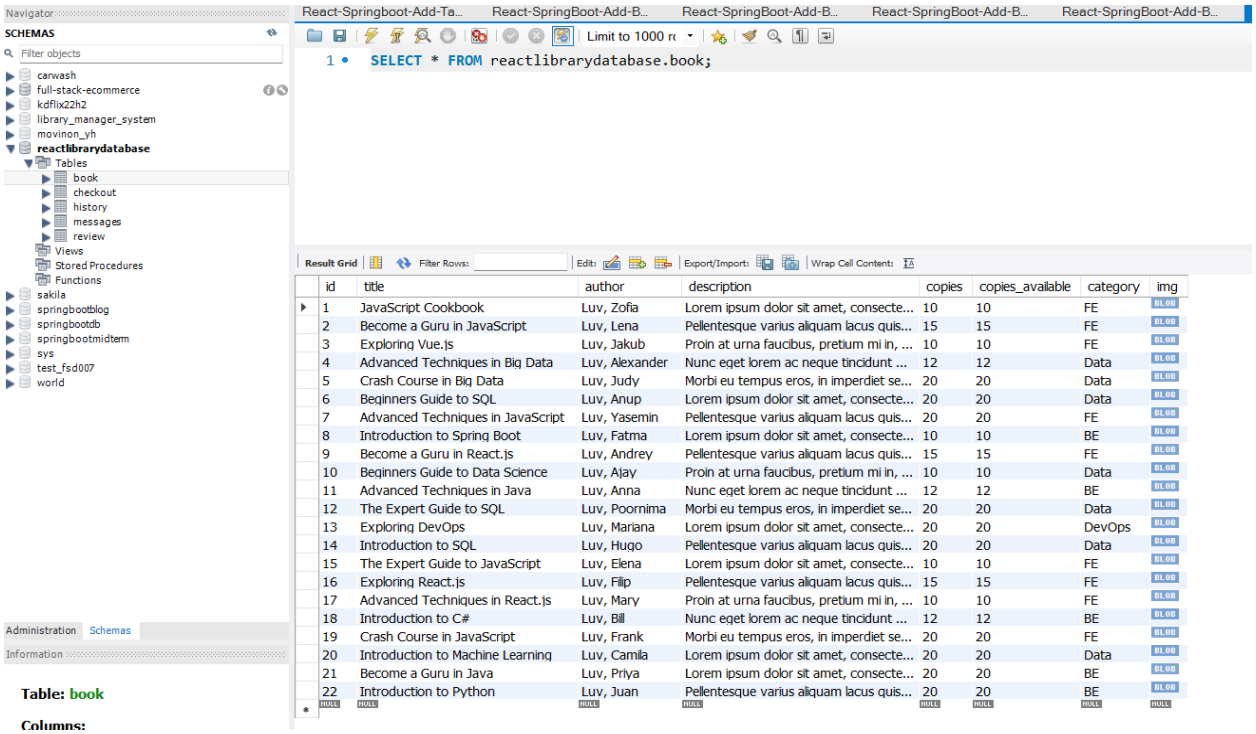
Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 36 | 19:20:14 | /*!40101 SET @saved_cs_client = @@character_set_client */ | 0 row(s) |
| 37 | 19:20:14 | /*!40101 SET character_set_client = utf8 */ | 0 row(s) |
| 38 | 19:20:14 | CREATE TABLE `history` ( `id` BIGINT(20) NOT NULL AUTO_INCREMENT, `user_email` varchar(45) DEFAULT NULL, `checkout_date` varchar(45) D... | 0 row(s) |
| 39 | 19:20:14 | /*!40101 SET character_set_client = @saved_cs_client */ | 0 row(s) |
| 40 | 19:20:14 | UNLOCK TABLES | 0 row(s) |
| 41 | 19:20:14 | /*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */ | 0 row(s) |
| 42 | 19:20:14 | /*!40101 SET SQL_MODE=@OLD_SQL_MODE */ | 0 row(s) |
| 43 | 19:20:14 | /*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */ | 0 row(s) |
| 44 | 19:20:14 | /*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */ | 0 row(s) |
| 45 | 19:20:14 | /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */ | 0 row(s) |
| 46 | 19:20:14 | /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */ | 0 row(s) |
| 47 | 19:20:14 | /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */ | 0 row(s) |
| 48 | 19:20:14 | /*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */ | 0 row(s) |

Do the same thing until we execute all the 5 scripts, refresh the schemas, and we can get the new database, and inside the tables, we can see there're five tables.



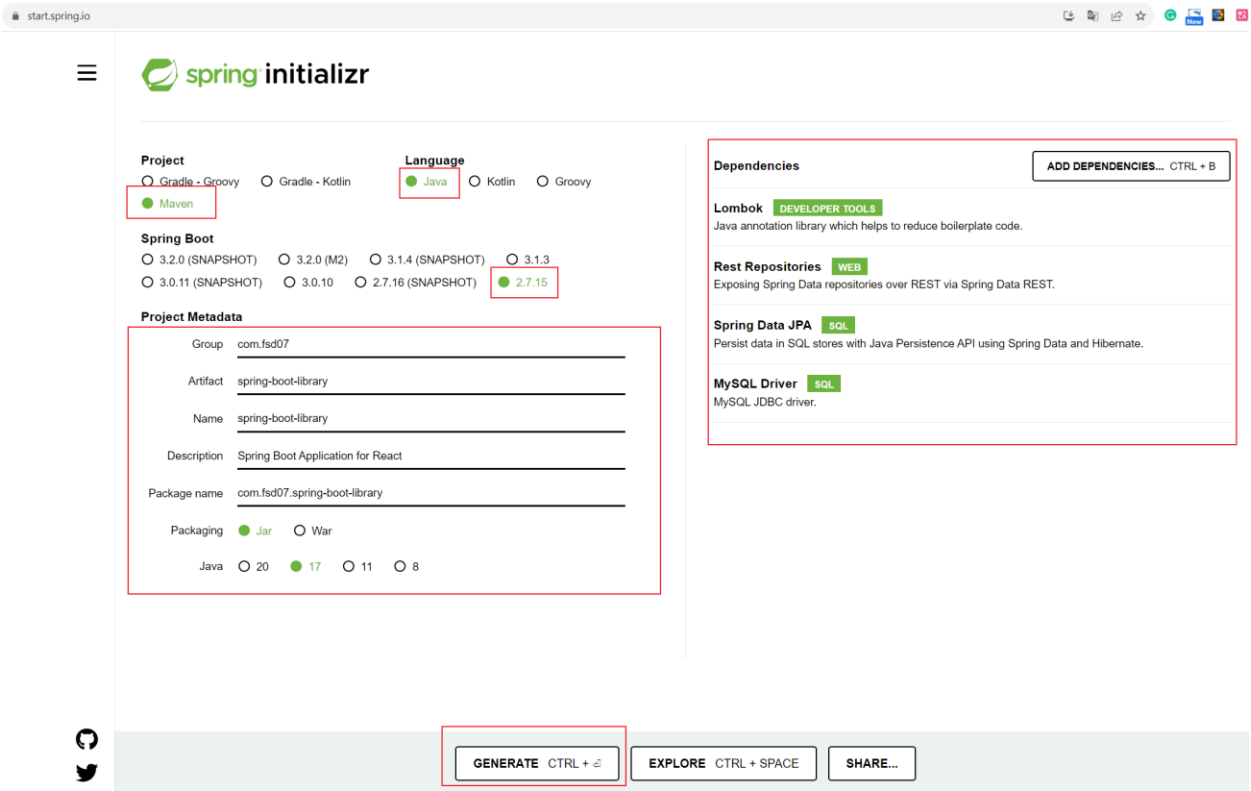

Check the book table, we can get the result like this



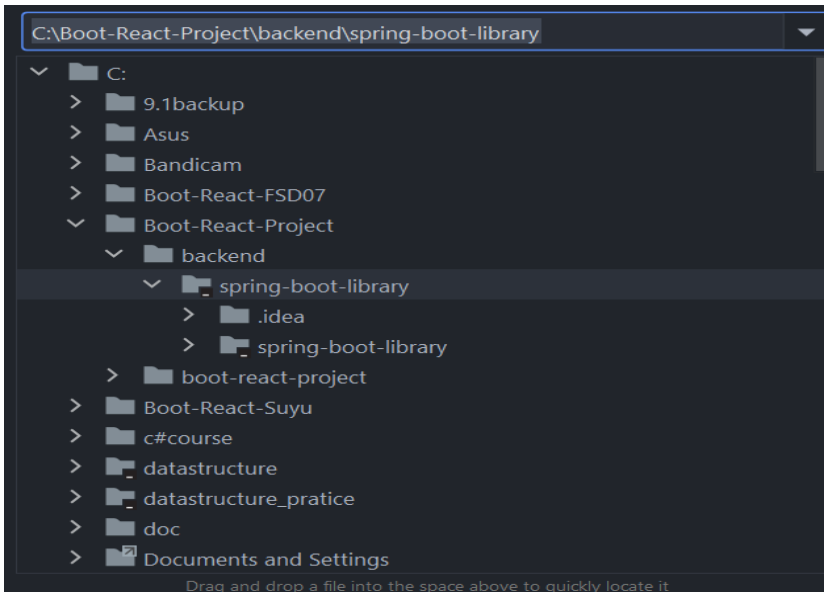

| id | title | author | description | copies | copies_available | category | img |
|---|---|---|---|---|---|---|---|
| 1 | JavaScript Cookbook | Luv, Zofia | Lorem ipsum dolor sit amet, consecte... | 10 | 10 | FE | BLOB |
| 2 | Become a Guru in JavaScript | Luv, Lena | Pellentesque varius aliquam lacus quis... | 15 | 15 | FE | BLOB |
| 3 | Exploring Vue.js | Luv, Jakub | Proin at urna faucibus, pretium mi in, ... | 10 | 10 | FE | BLOB |
| 4 | Advanced Techniques in Big Data | Luv, Alexander | Nunc eget lorem ac neque tincidunt ... | 12 | 12 | Data | BLOB |
| 5 | Crash Course in Big Data | Luv, Judy | Morbi eu tempus eros, in imperdiet se... | 20 | 20 | Data | BLOB |
| 6 | Beginners Guide to SQL | Luv, Anup | Lorem ipsum dolor sit amet, consecte... | 20 | 20 | Data | BLOB |
| 7 | Advanced Techniques in JavaScript | Luv, Yasemin | Pellentesque varius aliquam lacus quis... | 20 | 20 | FE | BLOB |
| 8 | Introduction to Spring Boot | Luv, Fatma | Lorem ipsum dolor sit amet, consecte... | 10 | 10 | BE | BLOB |
| 9 | Become a Guru in React.js | Luv, Andrey | Pellentesque varius aliquam lacus quis... | 15 | 15 | FE | BLOB |
| 10 | Beginners Guide to Data Science | Luv, Ajay | Proin at urna faucibus, pretium mi in, ... | 10 | 10 | Data | BLOB |
| 11 | Advanced Techniques in Java | Luv, Anna | Nunc eget lorem ac neque tincidunt ... | 12 | 12 | BE | BLOB |
| 12 | The Expert Guide to SQL | Luv, Poornima | Morbi eu tempus eros, in imperdiet se... | 20 | 20 | Data | BLOB |
| 13 | Exploring DevOps | Luv, Mariana | Lorem ipsum dolor sit amet, consecte... | 20 | 20 | DevOps | BLOB |
| 14 | Introduction to SQL | Luv, Hugo | Pellentesque varius aliquam lacus quis... | 20 | 20 | Data | BLOB |
| 15 | The Expert Guide to JavaScript | Luv, Elena | Lorem ipsum dolor sit amet, consecte... | 10 | 10 | FE | BLOB |
| 16 | Exploring React.js | Luv, Filip | Pellentesque varius aliquam lacus quis... | 15 | 15 | FE | BLOB |
| 17 | Advanced Techniques in React.js | Luv, Mary | Proin at urna faucibus, pretium mi in, ... | 10 | 10 | FE | BLOB |
| 18 | Introduction to C# | Luv, Bill | Nunc eget lorem ac neque tincidunt ... | 12 | 12 | BE | BLOB |
| 19 | Crash Course in JavaScript | Luv, Frank | Morbi eu tempus eros, in imperdiet se... | 20 | 20 | FE | BLOB |
| 20 | Introduction to Machine Learning | Luv, Camila | Lorem ipsum dolor sit amet, consecte... | 20 | 20 | Data | BLOB |
| 21 | Become a Guru in Java | Luv, Priya | Lorem ipsum dolor sit amet, consecte... | 20 | 20 | BE | BLOB |
| 22 | Introduction to Python | Luv, Juan | Pellentesque varius aliquam lacus quis... | 20 | 20 | BE | BLOB |

# 3. SETUP BACKEND APP

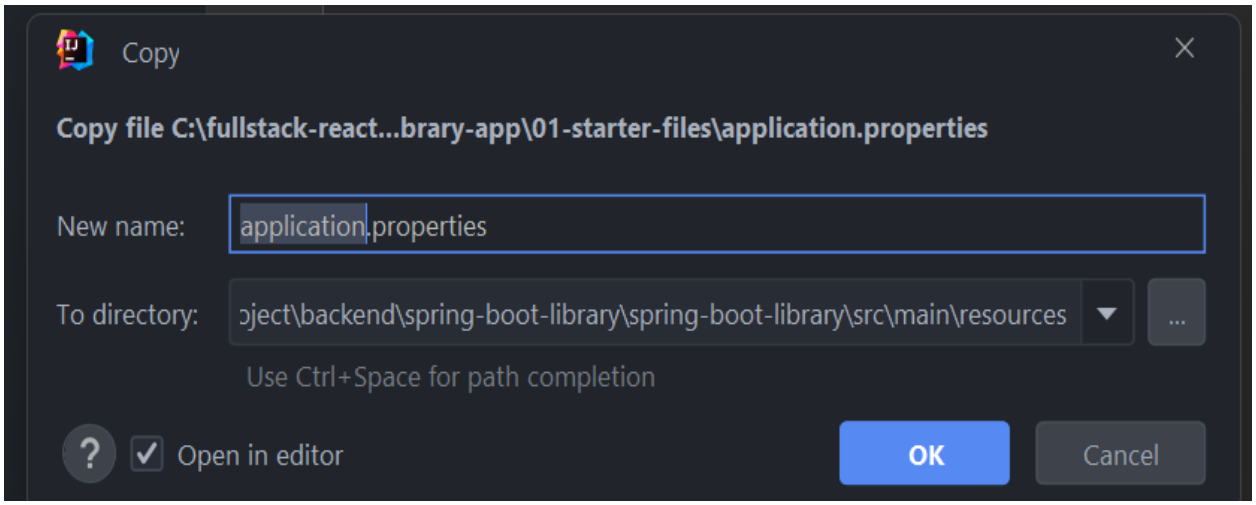
go to [https://start.spring.io/](https://start.spring.io/), configure our app. Then click on “Generate”, the website will generate a zip file and download it for us automatically. Unzip the file.



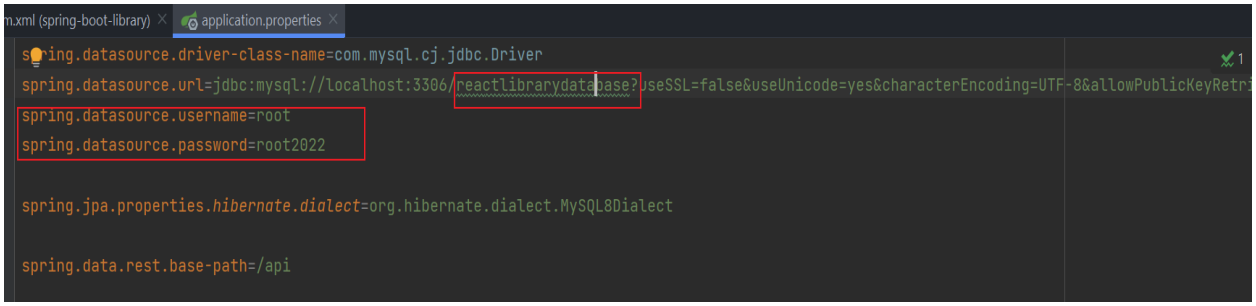

open Intellij, click on “file”, choose “open file or project”, choose the unzipped file that the website generated for us.



copy the application.properties file in our starter-file folder into our app. It will show a dialogue box, click on “ok”.

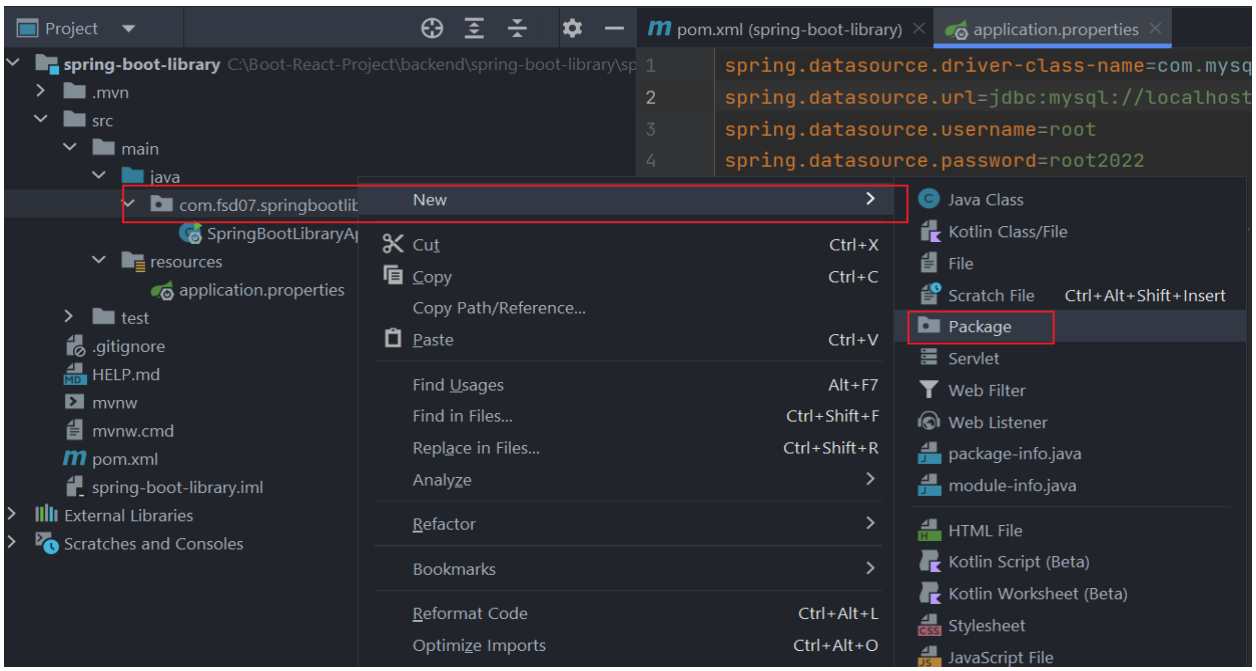



go to application.properties. Change the spring.datasource.username and spring.datasource.password to your own username and password. Make sure the database name is right

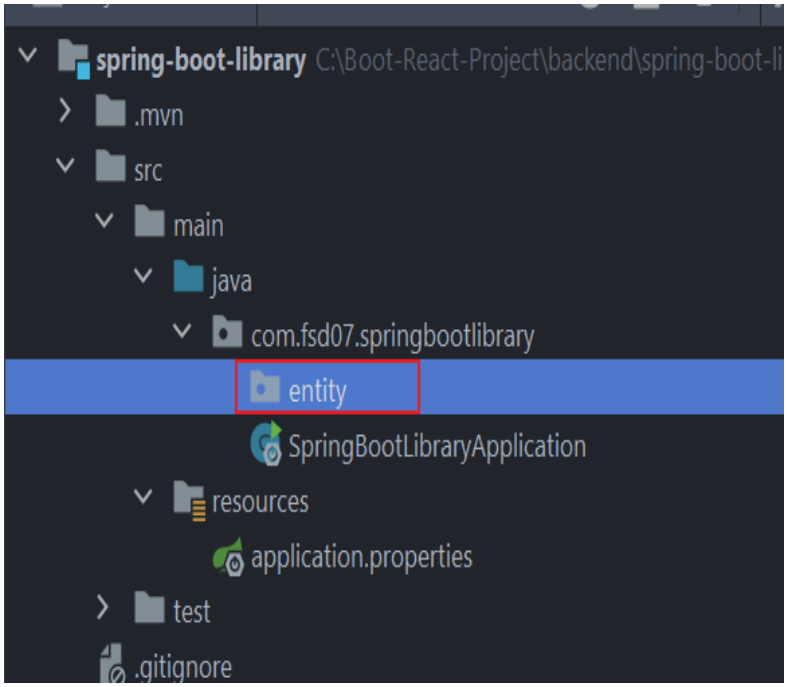


# 4. ENTITY SETUP

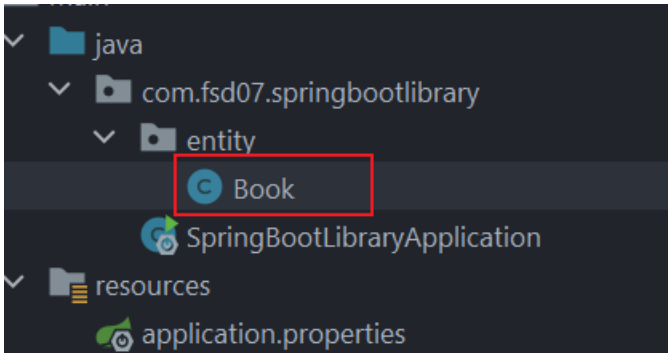
right click on the root package, choose New->package



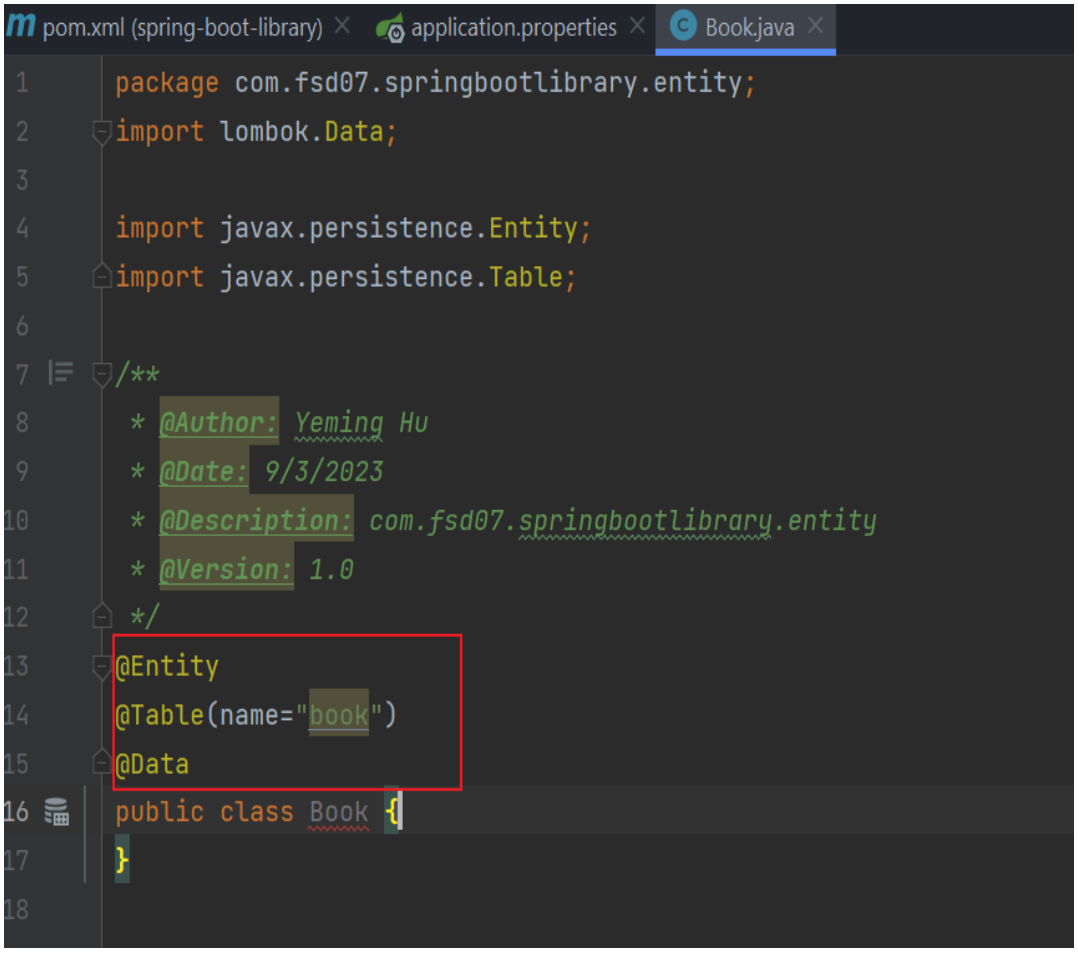

create a new package entity.



Create a new Java class file Book.




add annotations to the Book class.



add properties to the class which are mapping to our table column.

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

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

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

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

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

    @Column(name = "copies_available")
    private int copiesAvailable;

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

    @Column(name = "img")
    private String img;
```

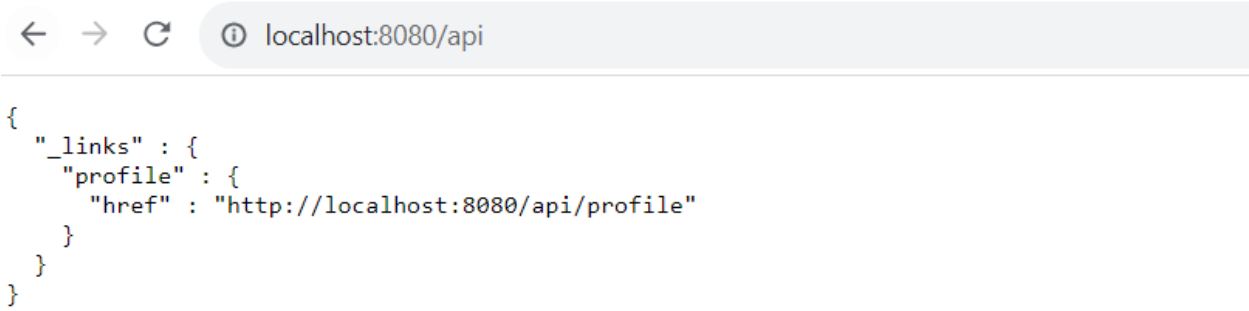
Now run the application. We can see our app runs successfully.

```java
    @Column(name = "copies")
    private int copies;

    @Column(name = "copies_available")
    private int copiesAvailable;

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

    @Column(name = "img")
    private String img;
}
```

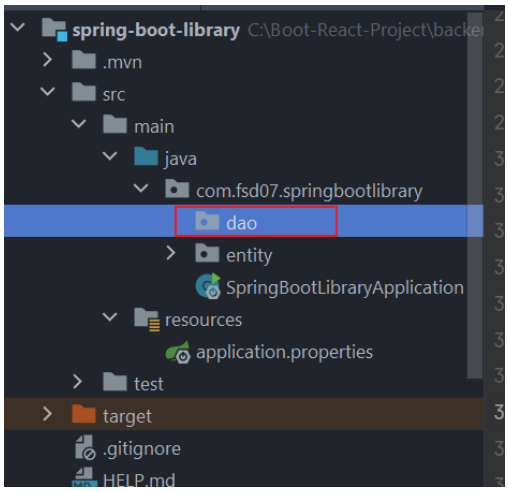
```
INFO 17160 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 21 ms. Found 1 JPA repository int
INFO 17160 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
INFO 17160 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
INFO 17160 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.79]
INFO 17160 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
INFO 17160 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 870 ms
INFO 17160 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
INFO 17160 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
INFO 17160 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
INFO 17160 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.15.Final
INFO 17160 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
INFO 17160 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
INFO 17160 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transactio
INFO 17160 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
WARN 17160 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries ma
INFO 17160 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
INFO 17160 --- [           main] c.f.s.SpringBootLibraryApplication       : Started SpringBootLibraryApplication in 2.96 seconds (JVM running for 3.447)
```

# 5. CREATE NEW API INTERFACE

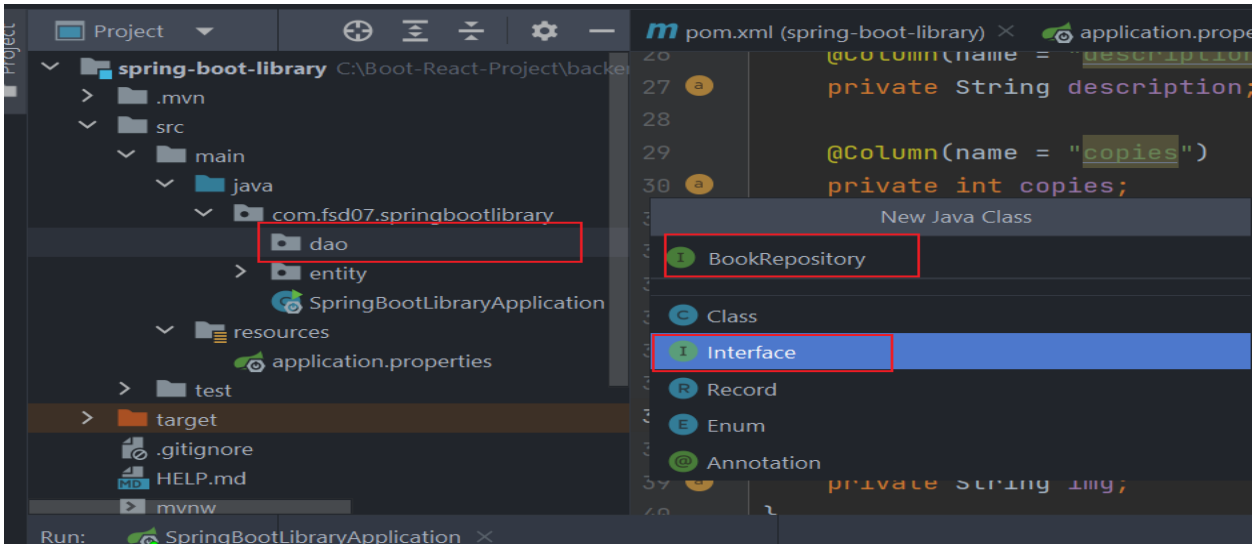
Go to the browser, run http://localhost:8080/api/, we can see the page:




Create a new package “dao”



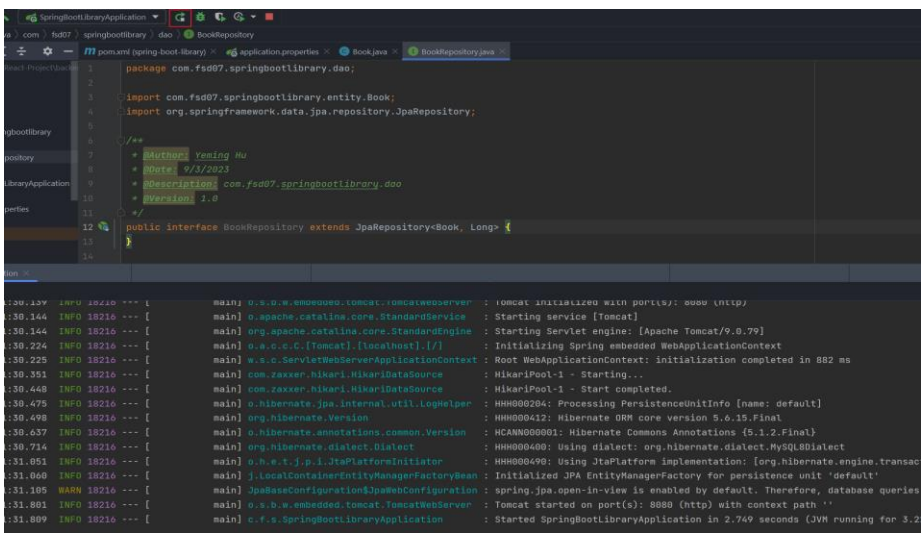

Create an interface BookRepository in the dao folder.



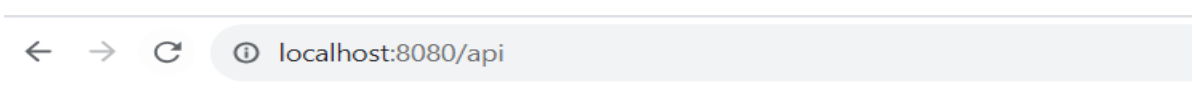
In this BookRepository file. Add “extend JpaRepository”. And import our book entity class.

```java
package com.fsd07.springbootlibrary.dao;

import com.fsd07.springbootlibrary.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * @Author: Yeming Hu
 * @Date: 9/3/2023
 * @Description: com.fsd07.springbootlibrary.dao
 * @Version: 1.0
 */
public interface BookRepository extends JpaRepository<Book, Long> {
}
```

Then we rerun our application.



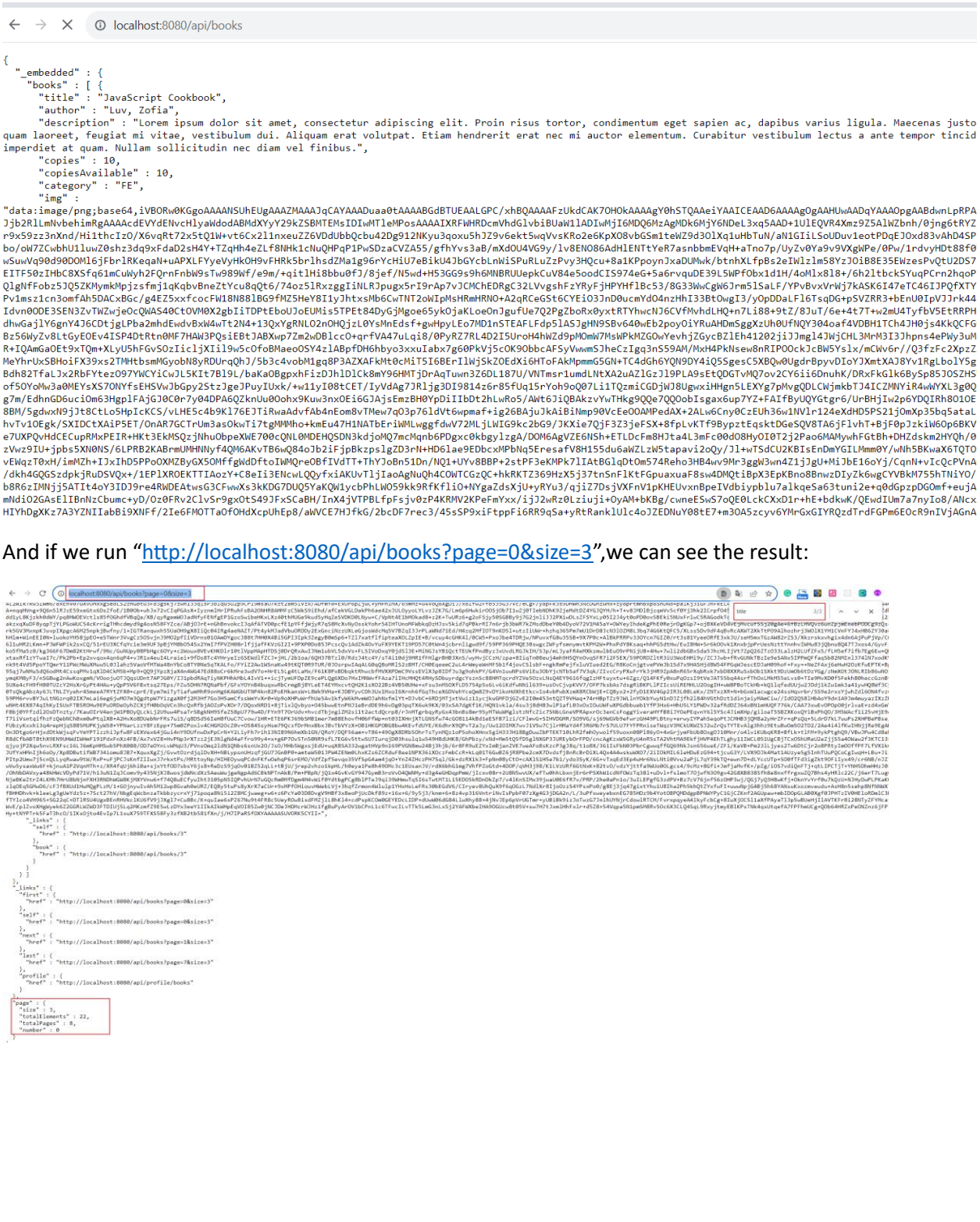
Go to the browser, run “http://localhost:8080/api/”,we can get the result like this.





```
{
  "_links" : {
    "books" : {
      "href" : "http://localhost:8080/api/books{?page,size,sort}",
      "templated" : true
    },
    "profile" : {
      "href" : "http://localhost:8080/api/profile"
    }
  }
}
```

Then we run “http://localhost:8080/api/books”, we can get the books information from our database.




And if we run “http://localhost:8080/api/books?page=0&size=3”,we can see the result:

# 6. READ ONLY CONFIGURATION

## REST API

- Spring Data REST will expose ALL these endpoints for free!

| HTTP Method | | CRUD Action |
|---|---|---|
| POST | /books | Create a new book |
| GET | /books | Read a list of books |
| GET | /books/{id} | Read a single book |
| PUT | /books/{id} | Update an existing book |
| DELETE | /books/{id} | Delete an existing book |

## REST API

I don't want that!
I want the REST API as READ-ONLY

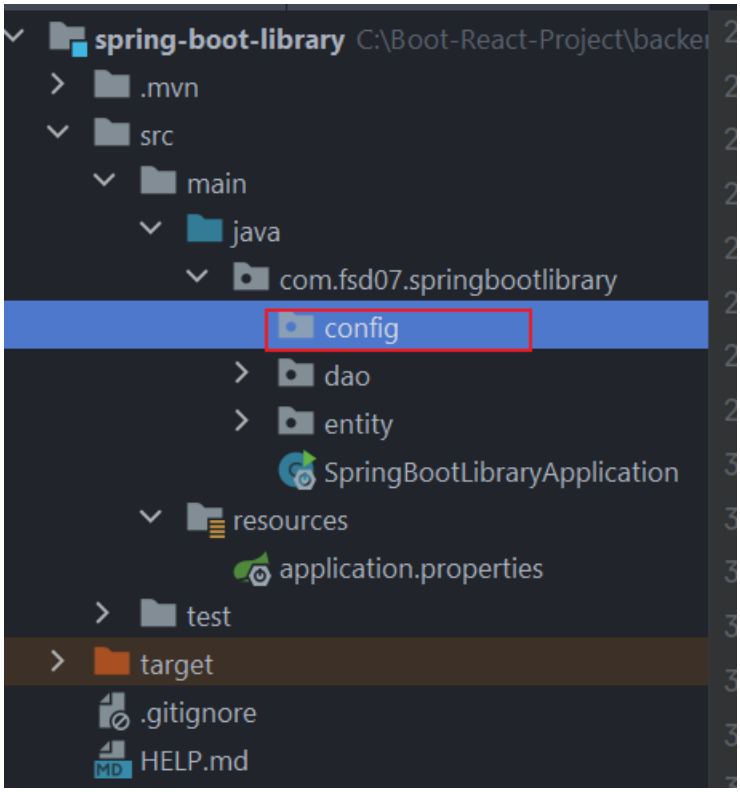
- Spring Data REST will expose ALL these endpoints for free!

| HTTP Method | | CRUD Action |
|---|---|---|
| ✗ POST | /books | Create a new book |
| ✓ GET | /books | Read a list of books |
| ✓ GET | /books/{id} | Read a single book |
| ✗ PUT | /books/{id} | Update an existing book |
| ✗ DELETE | /books/{id} | Delete an existing book |

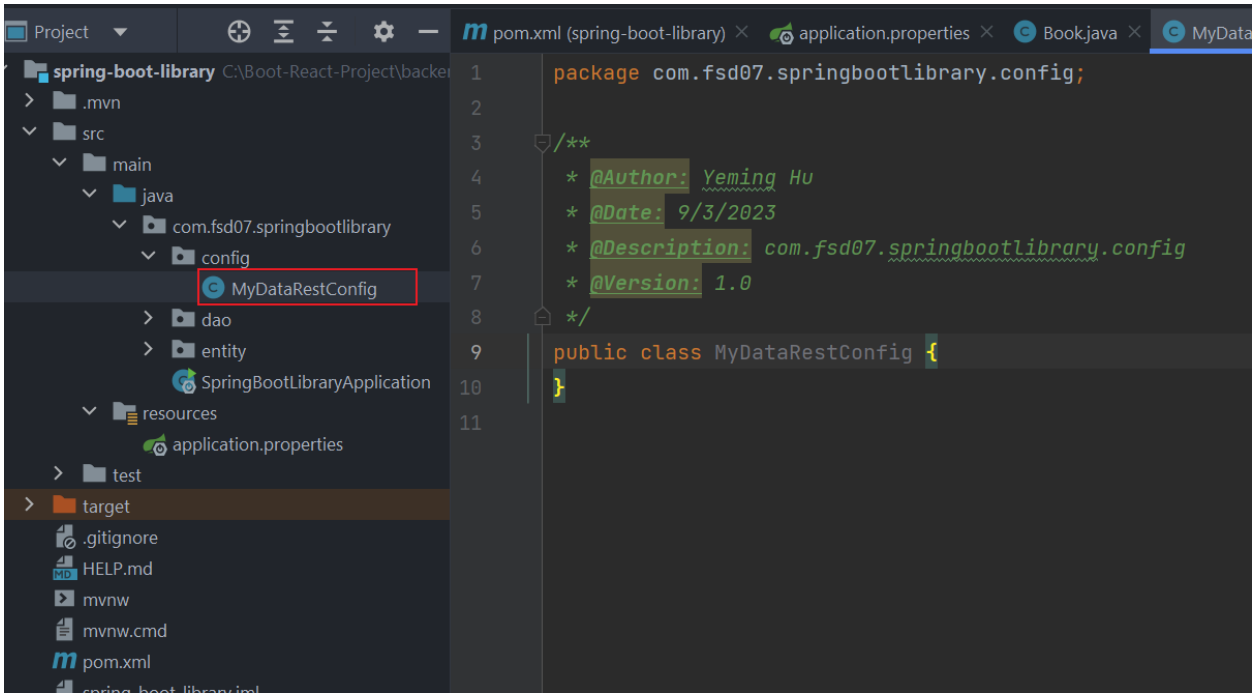
## Possible Solutions

1. Option 1: Don't use Spring Data REST
   1. Manually create our own @RestController
   2. Manually define methods for access: @GetMapping
   3. But we lose the Spring Data REST support for paging, sorting etc :-(
2. Option 2: Use Spring Data REST — Choose this one!
   1. Configure to disable certain HTTP methods: POST, DELETE etc...

Create a new config package.




Then in the config package, we create a MyDataRestConfig



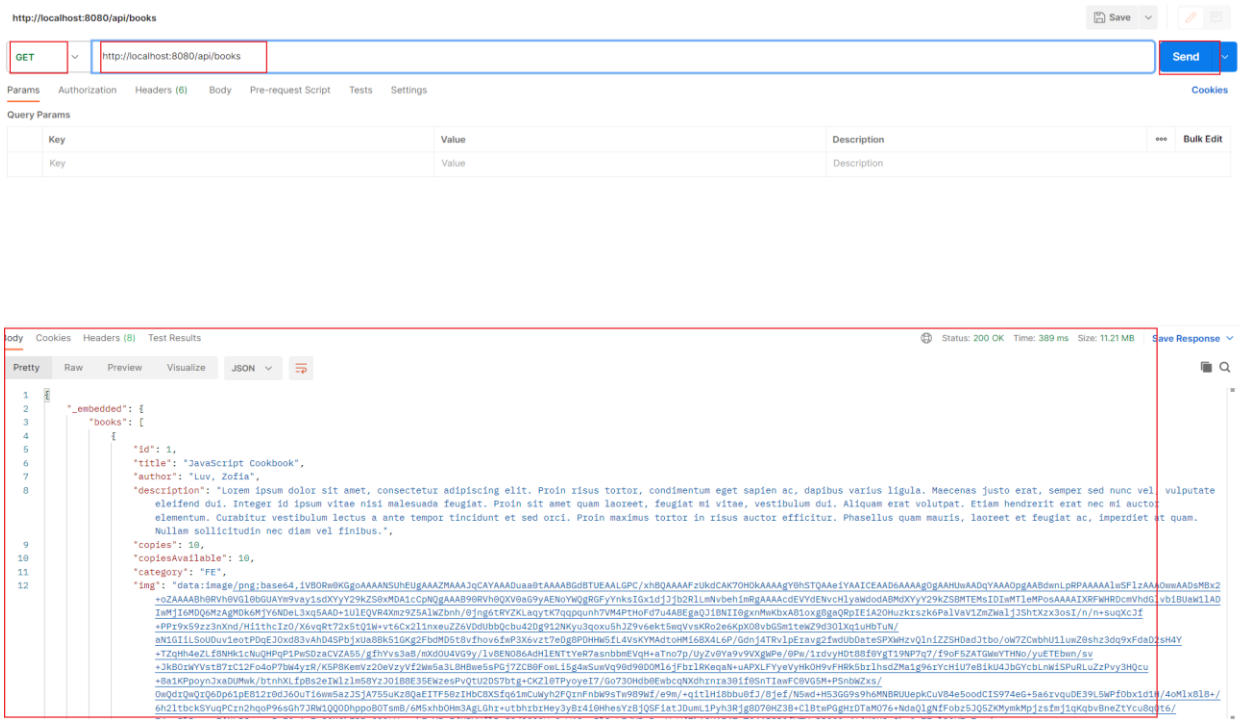
add annotation “configuration” to the class and make it “implements RepositoryRestConfigurer”.

```java
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;

/**
 * @Author: Yeming Hu
 * @Date: 9/3/2023
 * @Description: com.fsd07.springbootlibrary.config
 * @Version: 1.0
 */
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {
}
```

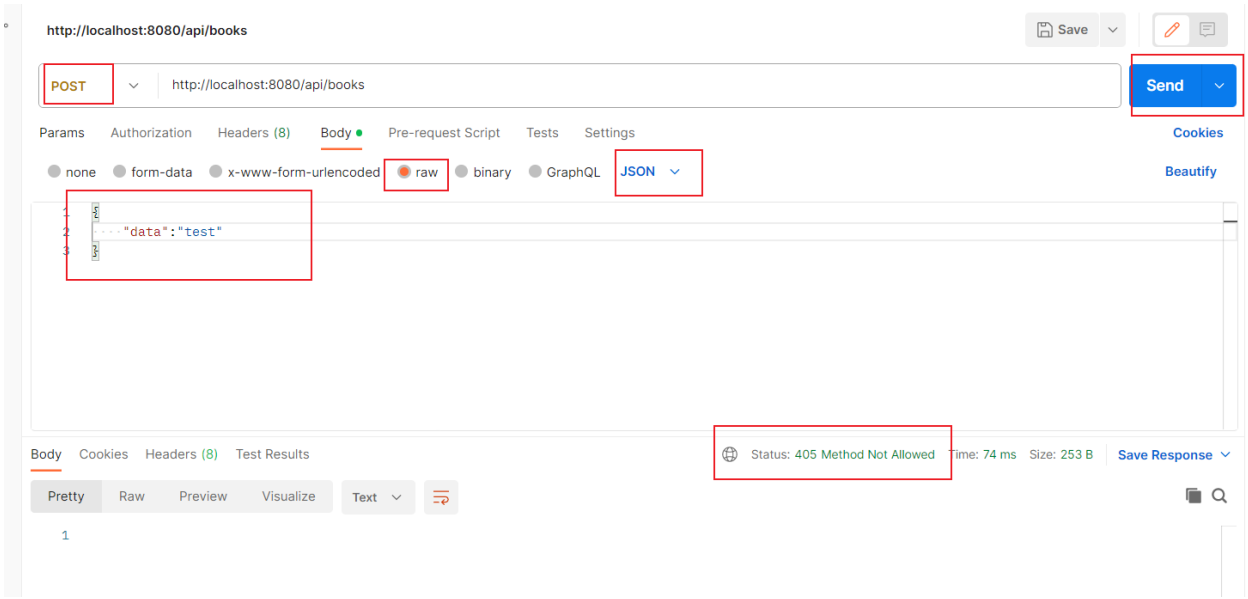
ExposeIds for the specified class and implement disableHttpMethods. Here, we set the Post, Delete, Patch and Put be the unsupported actions for Book class.

```java
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {
    1 usage
    private String theAllowedOrigins = "http://localhost:3000";

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config,
                                                     CorsRegistry cors) {
        HttpMethod[] theUnsupportedActions = {
                HttpMethod.POST,
                HttpMethod.PATCH,
                HttpMethod.DELETE,
                HttpMethod.PUT};

        config.exposeIdsFor(Book.class);

        disableHttpMethods(Book.class, config, theUnsupportedActions);

        /* Configure CORS Mapping */
        cors.addMapping( pathPattern: config.getBasePath() + "/**")
                .allowedOrigins(theAllowedOrigins);
    }

    1 usage
    private void disableHttpMethods(Class theClass,
                                    RepositoryRestConfiguration config,
                                    HttpMethod[] theUnsupportedActions) {
        config.getExposureConfiguration() ExposureConfiguration
                .forDomainType(theClass) ExposureConfigurer
                .withItemExposure((metdata, httpMethods) ->
                        httpMethods.disable(theUnsupportedActions))
                .withCollectionExposure((metdata, httpMethods) ->
                        httpMethods.disable(theUnsupportedActions));
    }
}
```

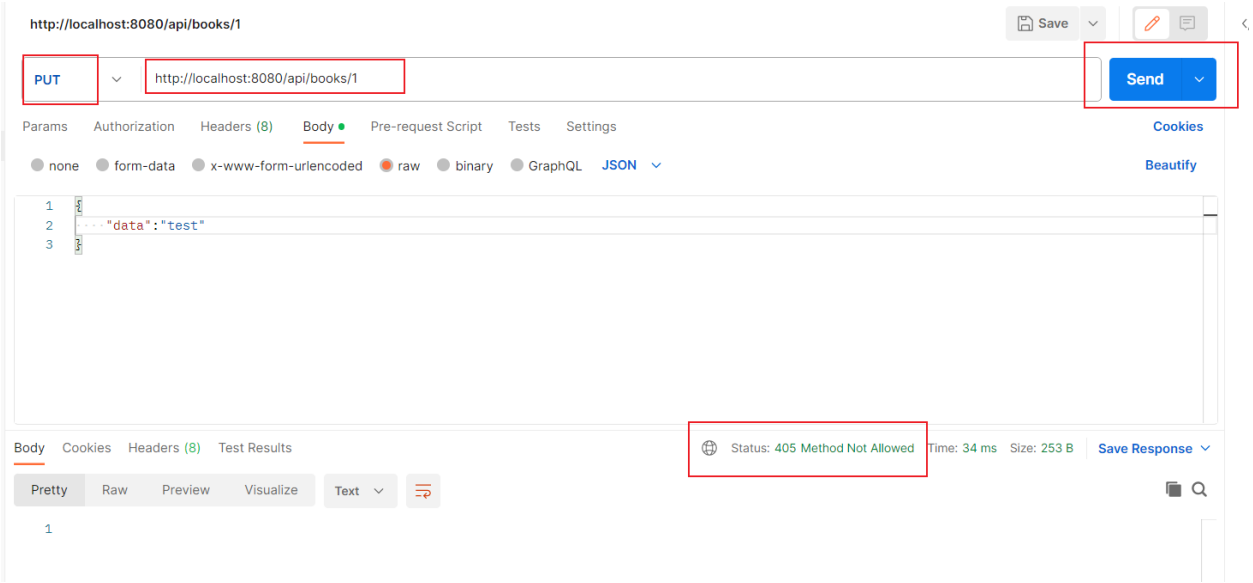
Then run the application, let's test in Postman. We input http://localhost:8080/api/books, we can get the result like this:




If we choose "POST". In body, choose "raw"and "JSON" , then click on "send", we will get a "405 method not allowed" result. Which means this API can use Post method.



Try with “PUT” , also returns “405 method not allowed”




Try with “DELETE” , also returns “405 method not allowed”.