

S02 SPINGBOOT BACKEND

1. Preparation Tools

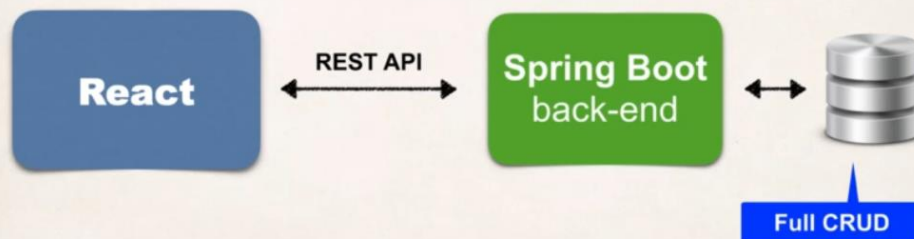
Java Development Environment

- 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

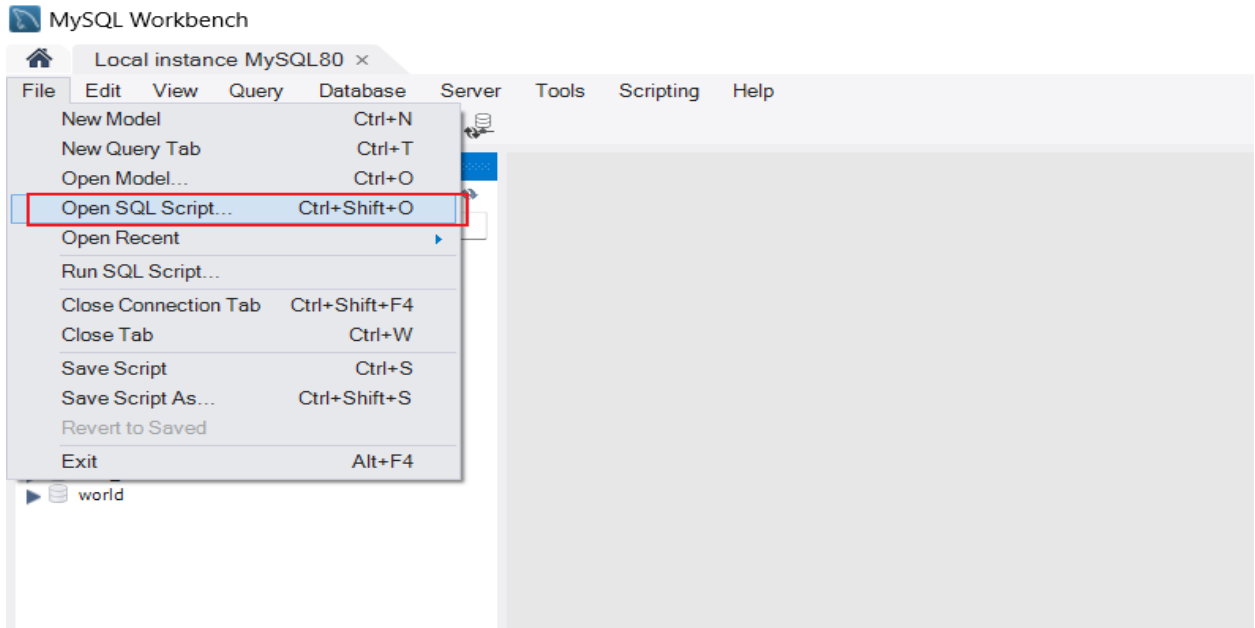
- Leverage Spring Data REST for REST API
- Minimizes the coding for Spring Boot back end

Full Stack

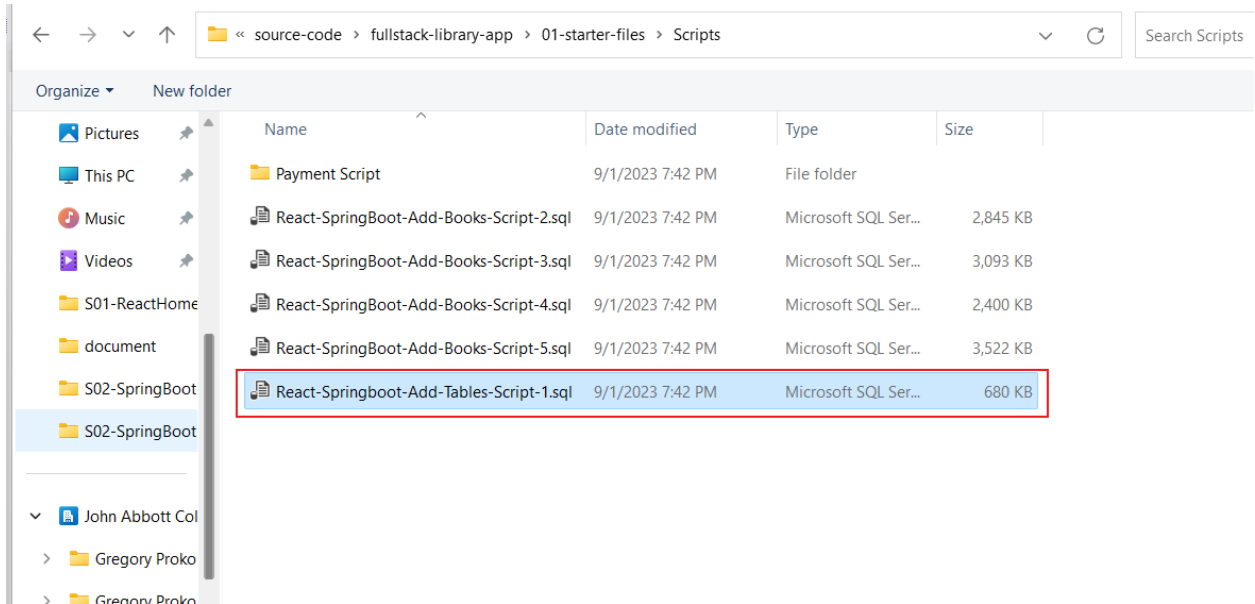


2. Setup Database

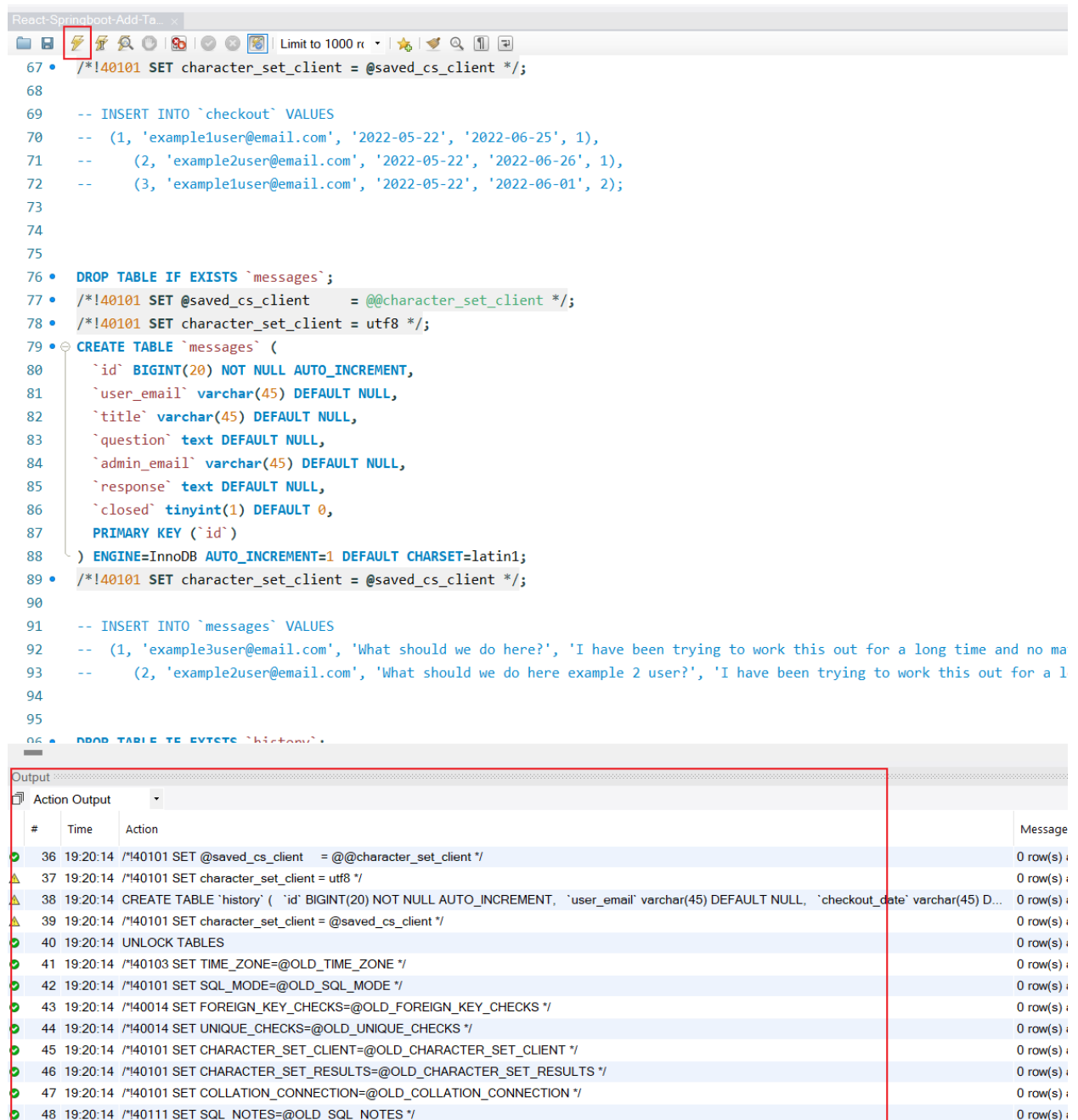
open MySQL Workbench, click on “file”, choose “open SQL Script”



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



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



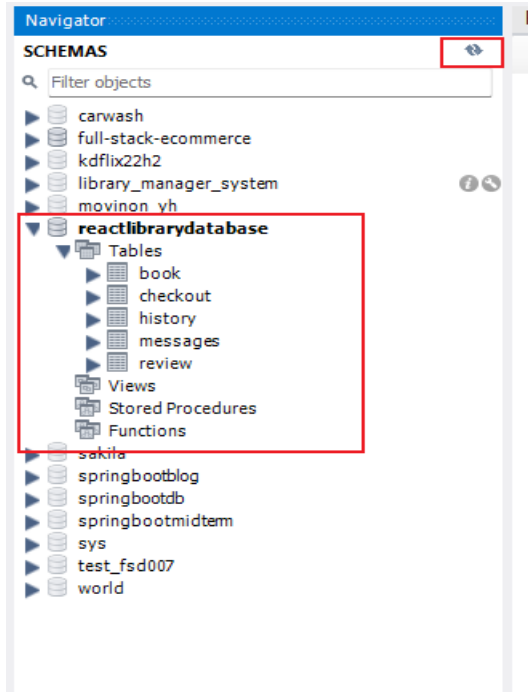
The screenshot shows a SQL IDE window titled "React-Springboot-Add-Ta...". The script editor contains SQL code for setting character set, inserting data into a 'checkout' table, dropping a 'messages' table, creating a 'messages' table, and inserting data into it. The 'execute sign' (a lightning bolt icon) is highlighted with a red box. Below the script editor, the 'Output' panel is visible, showing a table with columns '#', 'Time', 'Action', and 'Message'. The output shows the execution of the script, with the 'CREATE TABLE' statement highlighted in yellow.

```
67 • /*!40101 SET character_set_client = @saved_cs_client */;
68
69 -- INSERT INTO `checkout` VALUES
70 -- (1, 'example1user@email.com', '2022-05-22', '2022-06-25', 1),
71 -- (2, 'example2user@email.com', '2022-05-22', '2022-06-26', 1),
72 -- (3, 'example1user@email.com', '2022-05-22', '2022-06-01', 2);
73
74
75
76 • DROP TABLE IF EXISTS `messages`;
77 • /*!40101 SET @saved_cs_client = @@character_set_client */;
78 • /*!40101 SET character_set_client = utf8 */;
79 • CREATE TABLE `messages` (
80   `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
81   `user_email` varchar(45) DEFAULT NULL,
82   `title` varchar(45) DEFAULT NULL,
83   `question` text DEFAULT NULL,
84   `admin_email` varchar(45) DEFAULT NULL,
85   `response` text DEFAULT NULL,
86   `closed` tinyint(1) DEFAULT 0,
87   PRIMARY KEY (`id`)
88 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
89 • /*!40101 SET character_set_client = @saved_cs_client */;
90
91 -- INSERT INTO `messages` VALUES
92 -- (1, 'example3user@email.com', 'What should we do here?', 'I have been trying to work this out for a long time and no ma
93 -- (2, 'example2user@email.com', 'What should we do here example 2 user?', 'I have been trying to work this out for a l
94
95
96 • DROP TABLE IF EXISTS `history`;
```

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

The screenshot shows the SQL IDE interface. The SQL query editor contains the following query:

```
1 • SELECT * FROM reactlibrarydatabase.book;
```

The 'Result Grid' shows the results of the query. The columns are: id, title, author, description, copies, copies_available, category, and img. The results are as follows:

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

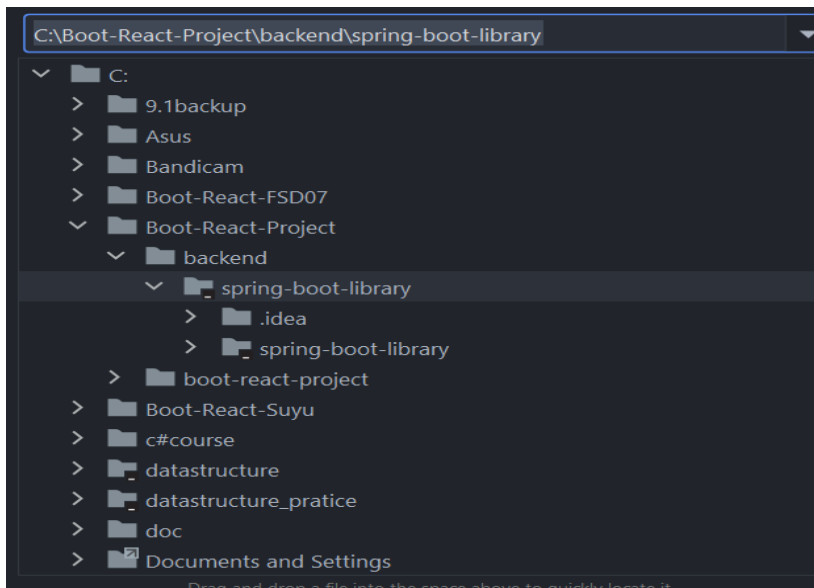
The 'Table: book' is highlighted in green. The 'Columns:' section is also highlighted in green.

3. SETUP BACKEND APP

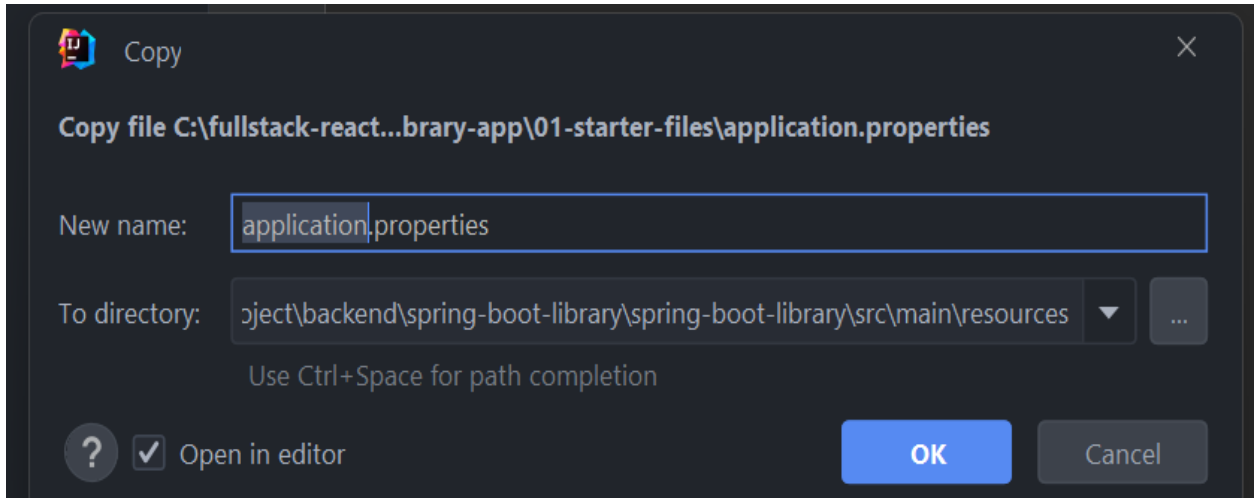
go to <https://start.spring.io/>, configure our app. Then click on “Generate”, the website will generate a zip file and download it for us automatically. Unzip the file.

The screenshot shows the Spring Initializr website interface. The 'Project' section has 'Maven' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.7.15' selected. The 'Project Metadata' section shows fields for Group, Artifact, Name, Description, and Package name, all filled with default values. The 'Packaging' section has 'Jar' selected. The 'Dependencies' section shows 'Lombok', 'Rest Repositories', 'Spring Data JPA', and 'MySQL Driver' selected. The 'GENERATE' button is highlighted with a red box.

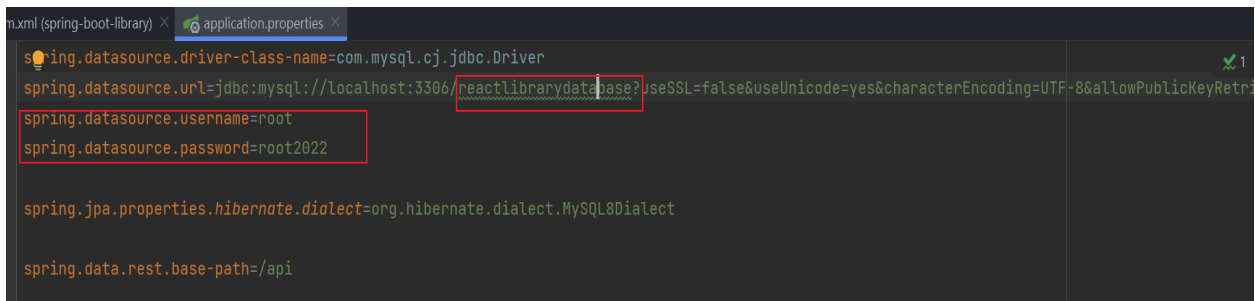
open IntelliJ, click on “file”, choose “open file or project”, choose the unzipped file that the website generated for us.



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

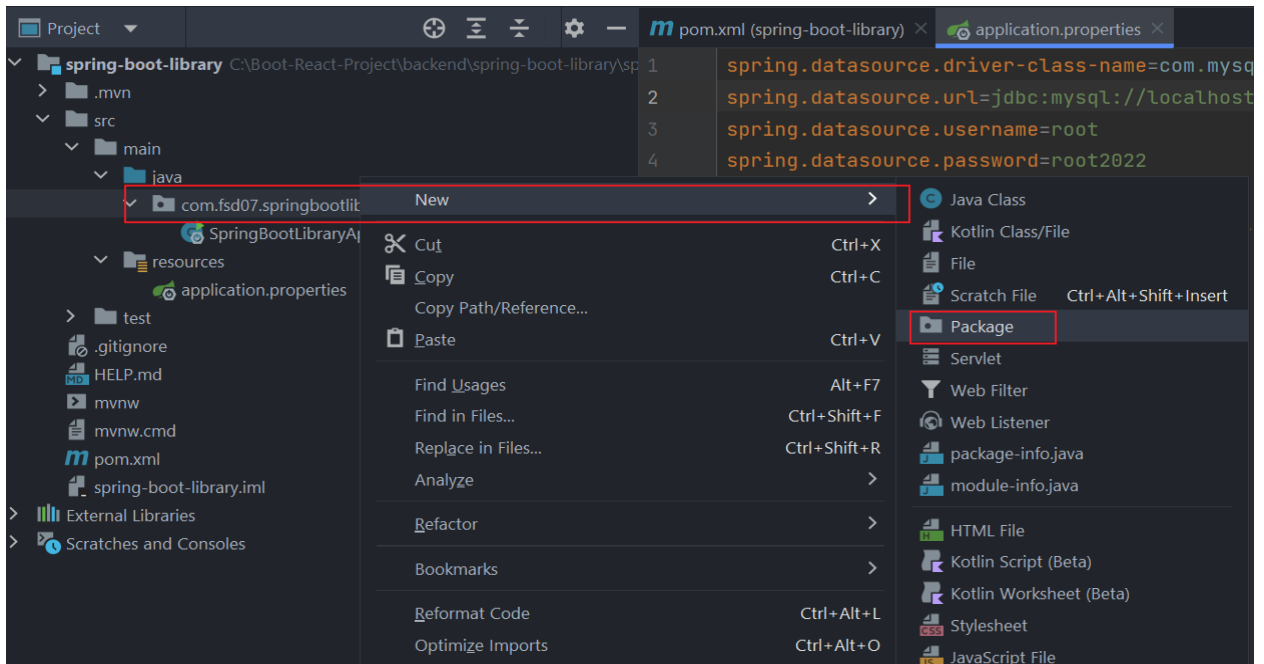


go to application.properties. Change the spring.datasource.username and spring.datasource.password to your own username and password. Make sure the database name is right

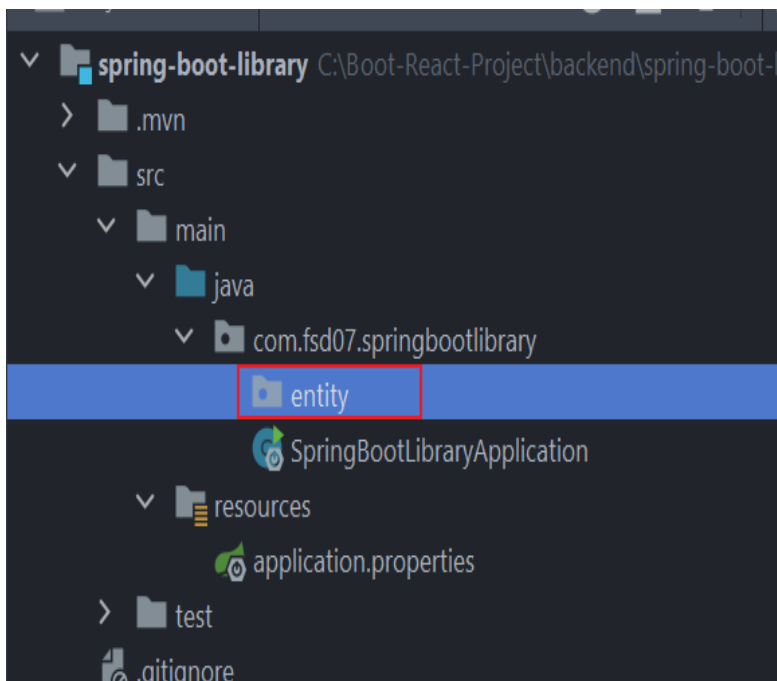


4. ENTITY SETUP

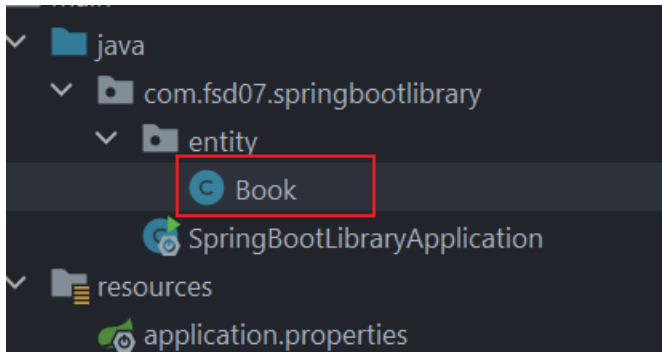
right click on the root package, choose New->package



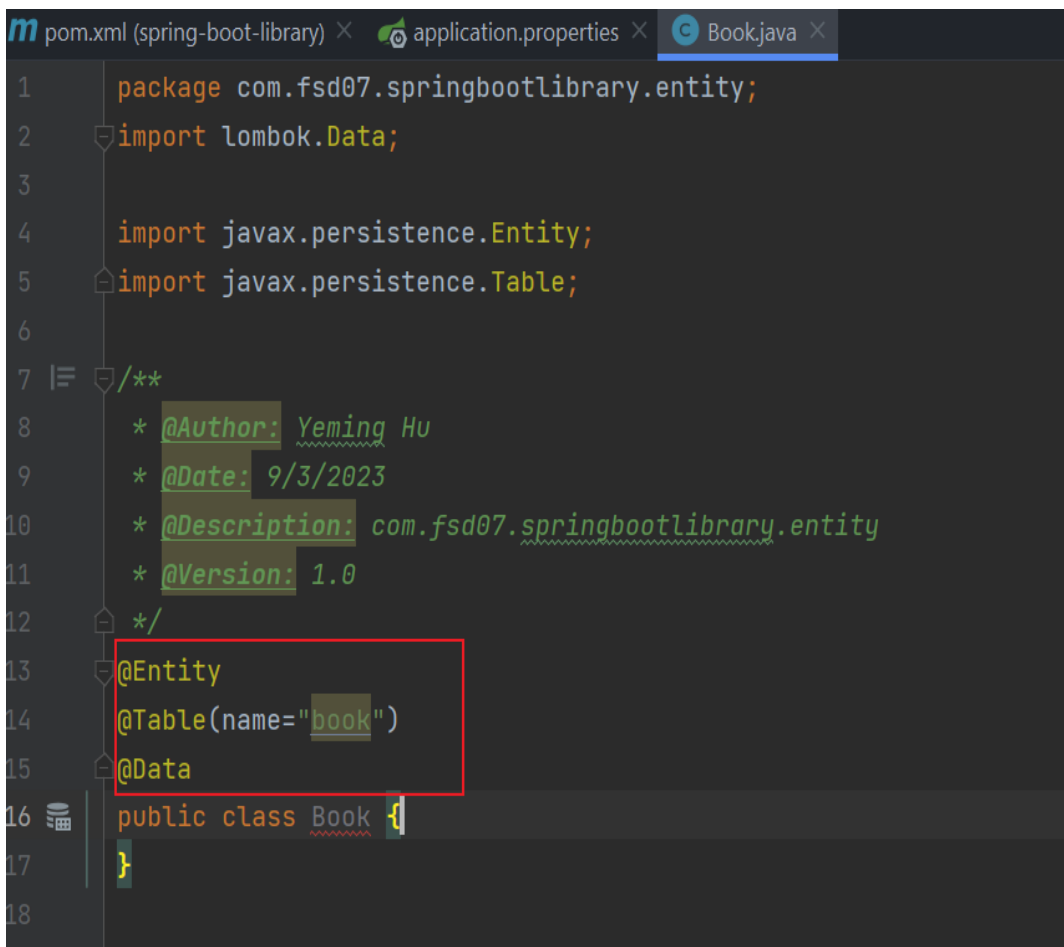
create a new package entity.



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.

```
@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;
}
```

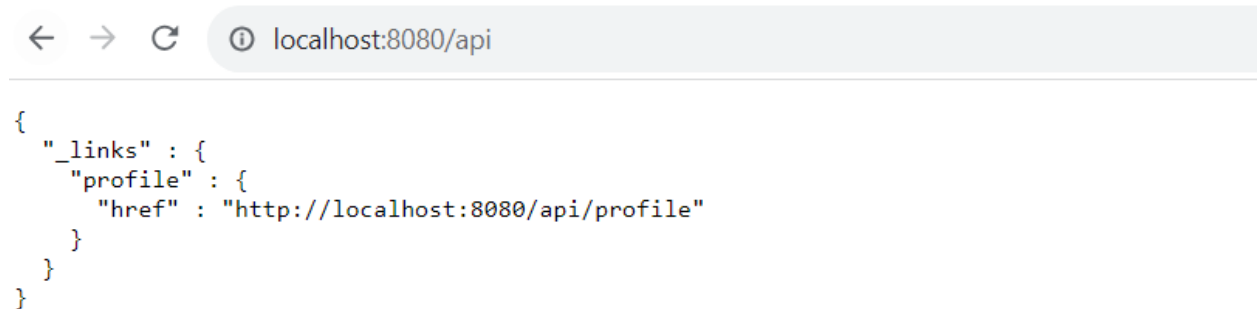
Now run the application. We can see our app runs successfully.

```
springbootlibrary > entity > Book > img
pom.xml (spring-boot-library) x application.properties x Book.java x BookRepository.java x
29 @Column(name = "copies")
30 private int copies;
31
32 @Column(name = "copies_available")
33 private int copiesAvailable;
34
35 @Column(name = "category")
36 private String category;
37
38 @Column(name = "img")
39 private String img;
40 }

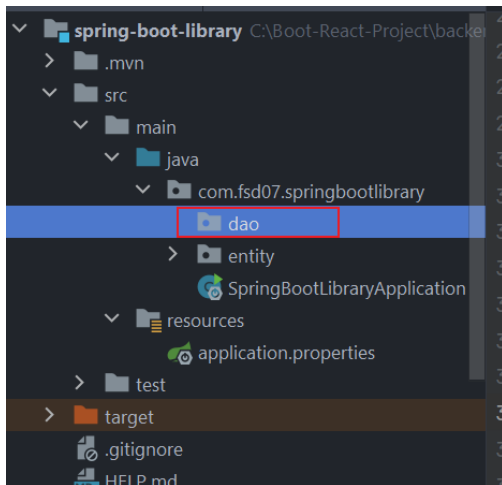
INFO 17160 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 21 ms. Found 1 JPA repository int
INFO 17160 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
INFO 17160 --- [main] o.apache.catalina.core.StandardService : 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.transaction.jta.platform.internal.NoJtaPlatform]
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 may run longer than
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

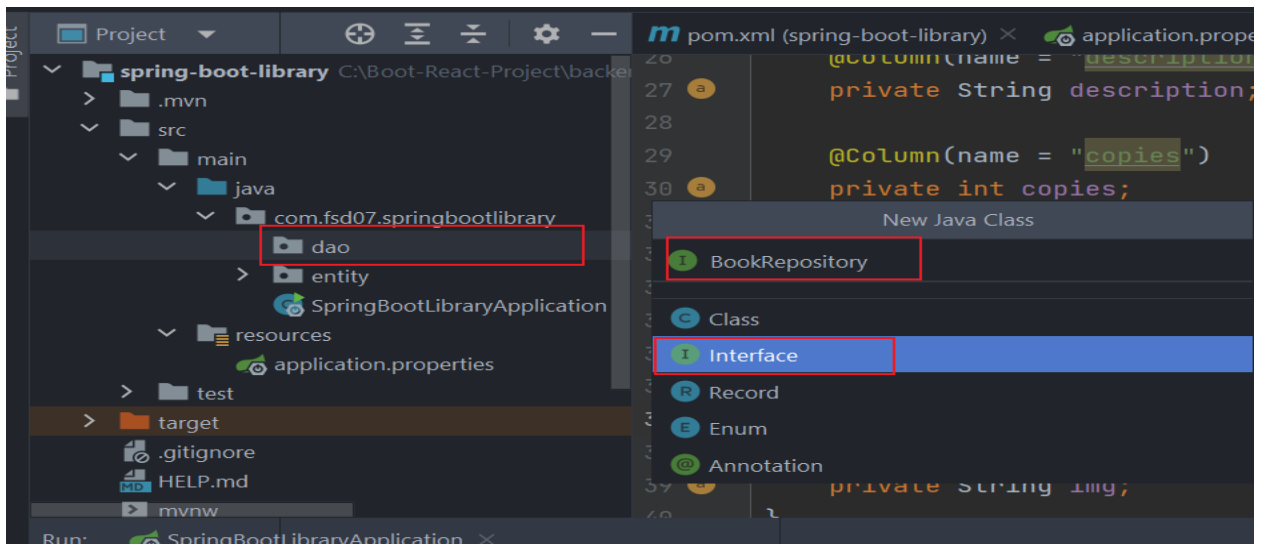
Go to the browser, run <http://localhost:8080/api/>, we can see the page:



Create a new package “dao”



Create an interface `BookRepository` in the dao folder.



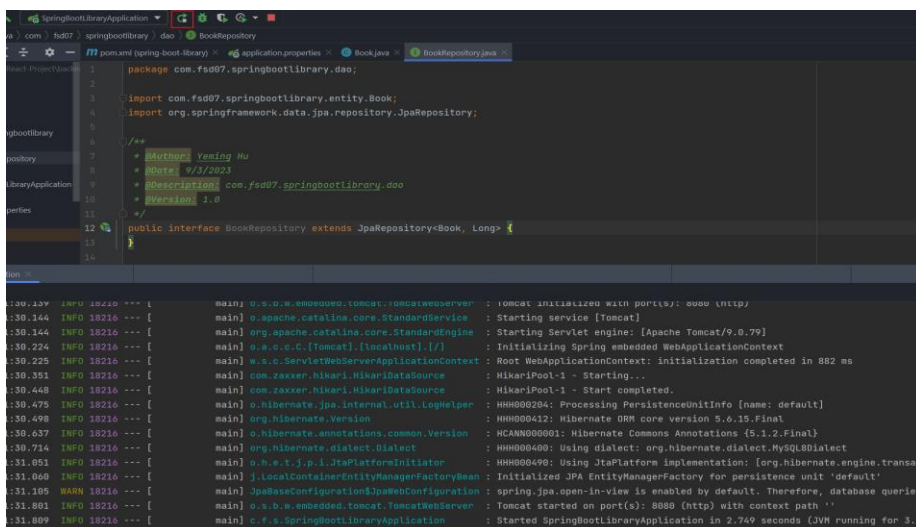
In this BookRepository file. Add “extend JpaRepository”. And import our book entity class.

```
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.



The screenshot shows an IDE with the BookRepository.java file open. The file contains the following code:

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

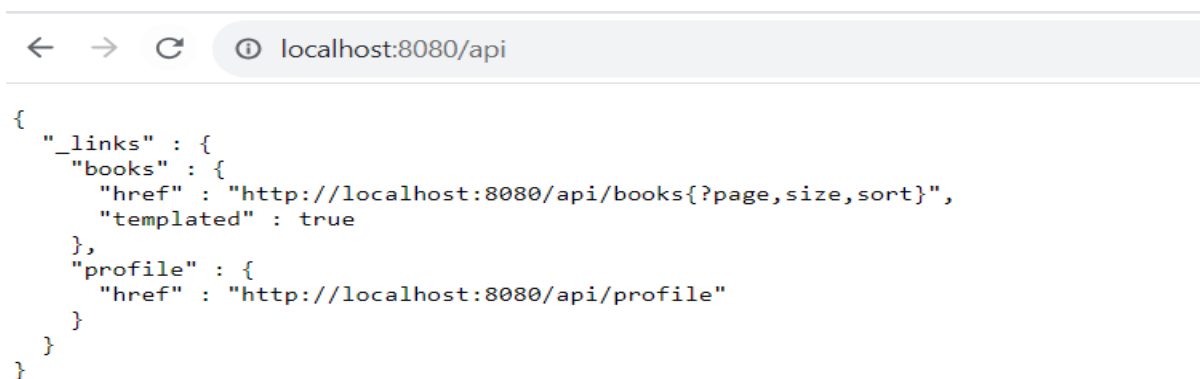
import com.fsd07.springbootlibrary.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

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

The console output shows the application's startup logs, including the following lines:

```
main | 0.0.0.0:8080:org.springframework.boot.web.embedded.tomcat.TomcatWebServer : tomcat initialized with port(s): 8080 (http)
main | 0.0.0.0:8080:org.apache.catalina.core.StandardService : Starting service [Tomcat]
main | 0.0.0.0:8080:org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.79]
main | 0.0.0.0:8080:org.springframework.boot.web.embedded.tomcat.TomcatWebServer : Initializing Spring embedded WebApplicationContext
main | 0.0.0.0:8080:org.springframework.boot.web.embedded.tomcat.TomcatWebServer : Root WebApplicationContext: initialization completed in 882 ms
main | 0.0.0.0:8080:com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
main | 0.0.0.0:8080:com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
main | 0.0.0.0:8080:org.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
main | 0.0.0.0:8080:org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.15.Final
main | 0.0.0.0:8080:org.hibernate.annotations.common.Version : HHH00000001: Hibernate Commons Annotations (5.1.2.Final)
main | 0.0.0.0:8080:org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
main | 0.0.0.0:8080:org.hibernate.jpa.internal.util.LogHelper : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
main | 0.0.0.0:8080:org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main | 0.0.0.0:8080:org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
main | 0.0.0.0:8080:org.springframework.boot.web.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
main | 0.0.0.0:8080:org.springframework.boot.SpringApplication : Started SpringBootApplication in 2.749 seconds (JVM running for 3.2)
```

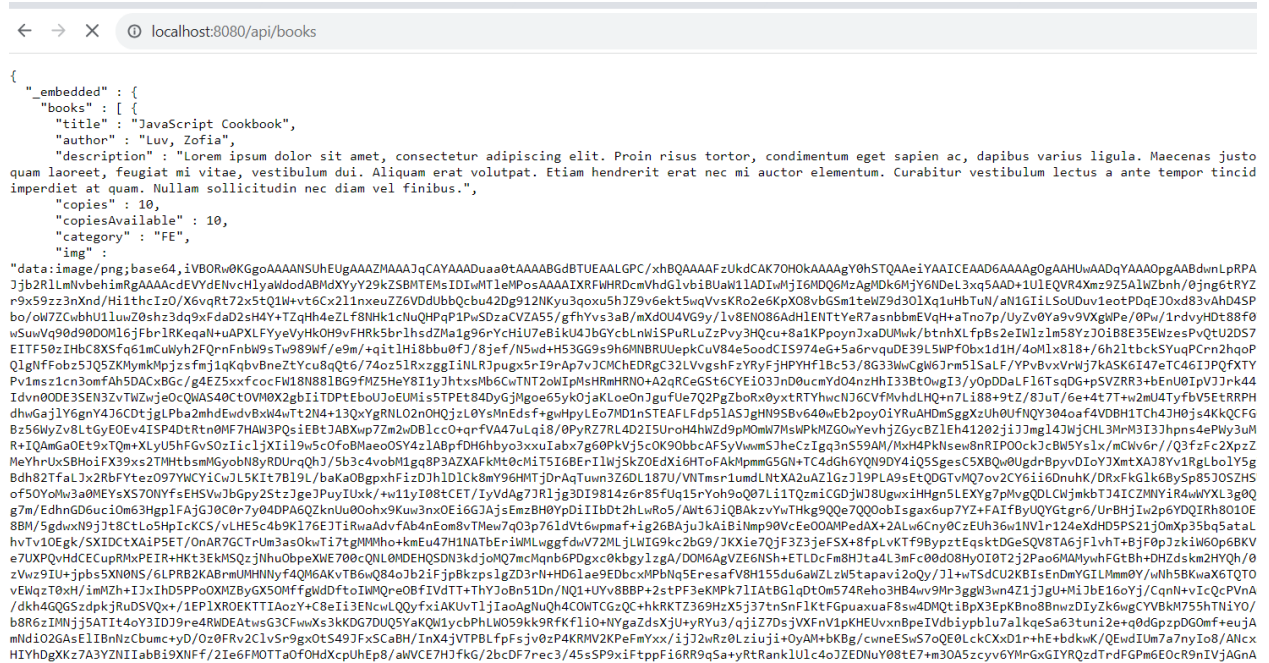
Go to the browser, run “<http://localhost:8080/api/>”,we can get the result like this.



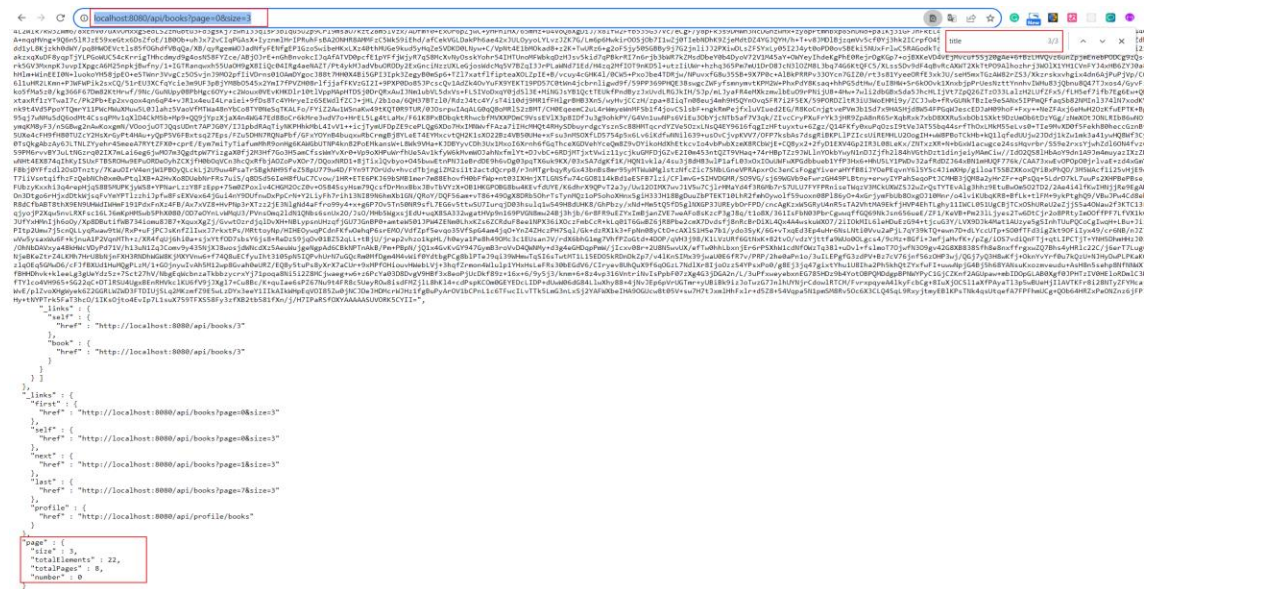
The screenshot shows a web browser with the address bar displaying <http://localhost:8080/api/>. The response is a JSON object:

```
{
  "_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	<u>C</u> reate a new book
GET	/books	<u>R</u> ead a list of books
GET	/books/{id}	<u>R</u> ead a single book
PUT	/books/{id}	<u>U</u> pdate an existing book
DELETE	/books/{id}	<u>D</u> elete 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	<u>C</u> reate a new book
✅ GET	/books	<u>R</u> ead a list of books
✅ GET	/books/{id}	<u>R</u> ead a single book
❌ PUT	/books/{id}	<u>U</u> pdate an existing book
❌ DELETE	/books/{id}	<u>D</u> elete an existing book

Possible Solutions

1. Option 1: Don't use Spring Data REST

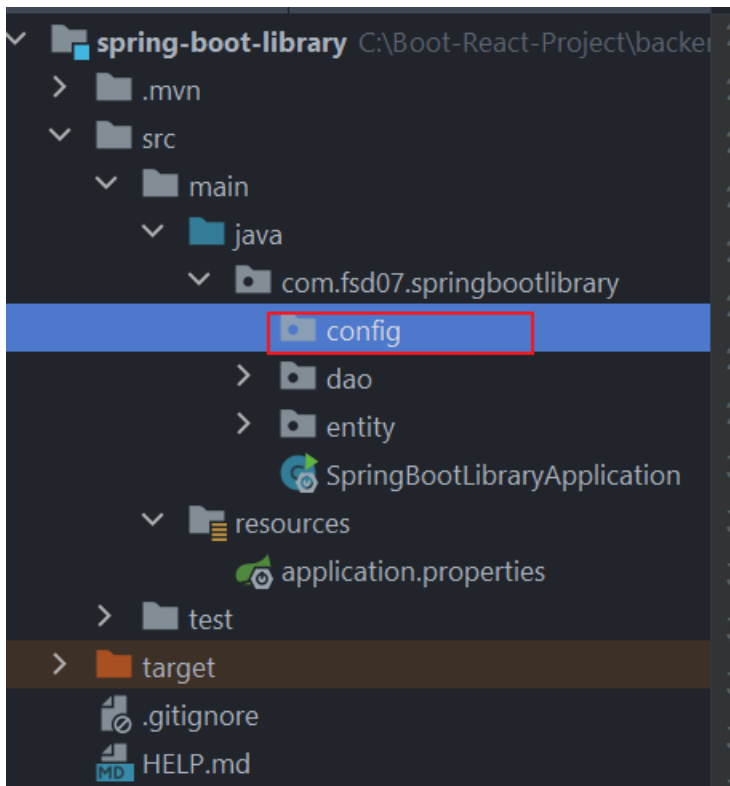
- Manually create our own @RestController
- Manually define methods for access: @GetMapping
- But we lose the Spring Data REST support for paging, sorting etc :-(

2. Option 2: Use Spring Data REST

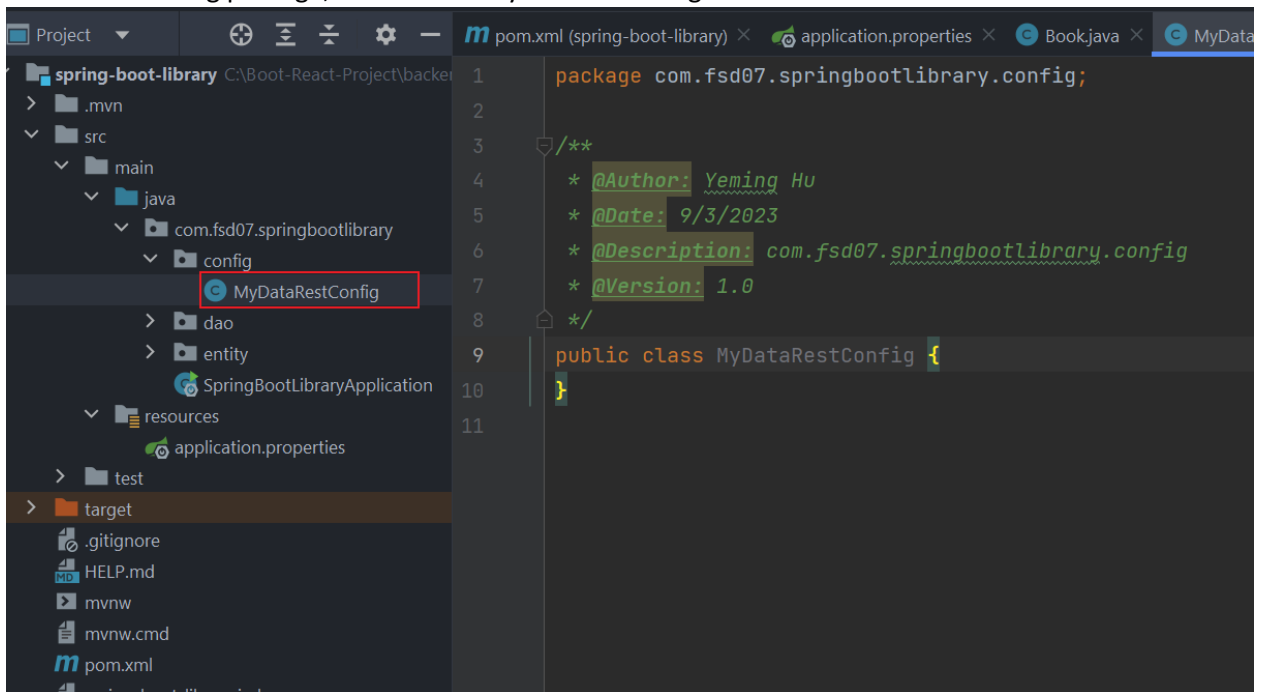
Choose this one!

- 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".

```
4 import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurer;
5
6 /**
7  * @Author: Yeming Hu
8  * @Date: 9/3/2023
9  * @Description: com.fsd07.springbootlibrary.config
10  * @Version: 1.0
11  */
12 @Configuration
13 public class MyDataRestConfig implements RepositoryRestConfigurer {
14 }
15
```

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

```
@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);

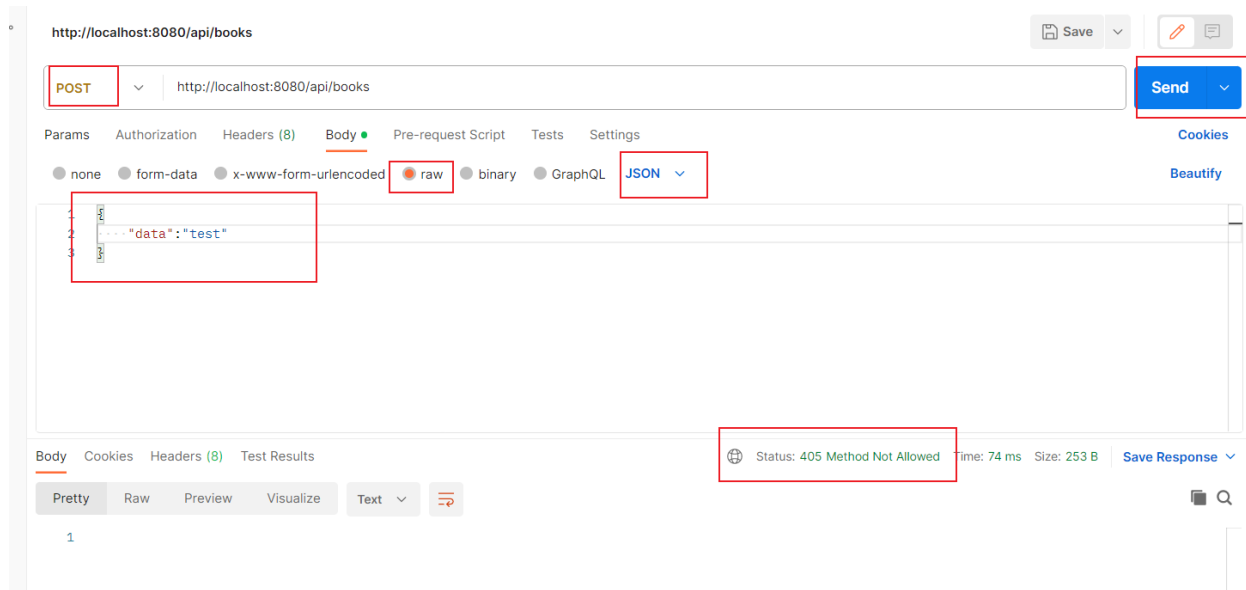
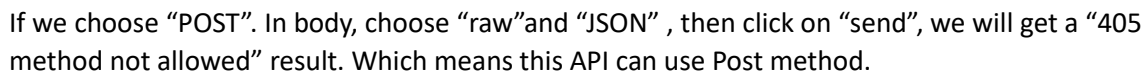
        /* Configure CORS Mapping */
        cors.addMapping(pathPattern: config.getBasePath() + "/*")
            .allowedOrigins(theAllowedOrigins);
    }

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



The screenshot shows the Swagger UI interface for the endpoint `http://localhost:8080/api/books`. The `GET` method is selected. The `Send` button is highlighted with a red box. Below the endpoint, the `Query Params` section is visible, showing a table structure for key-value pairs.

Key	Value	Description
Key	Value	Description



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

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/api/books/1`
- Method:** `PUT`
- Body:**

```
{ 1 2   "data": "test" 3 }
```
- Response:** Status: 405 Method Not Allowed, Time: 34 ms, Size: 253 B

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

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/api/books/1`
- Method:** `DELETE`
- Body:**

```
{ 1 2   "data": "test" 3 }
```
- Response:** Status: 405 Method Not Allowed, Time: 8 ms, Size: 253 B