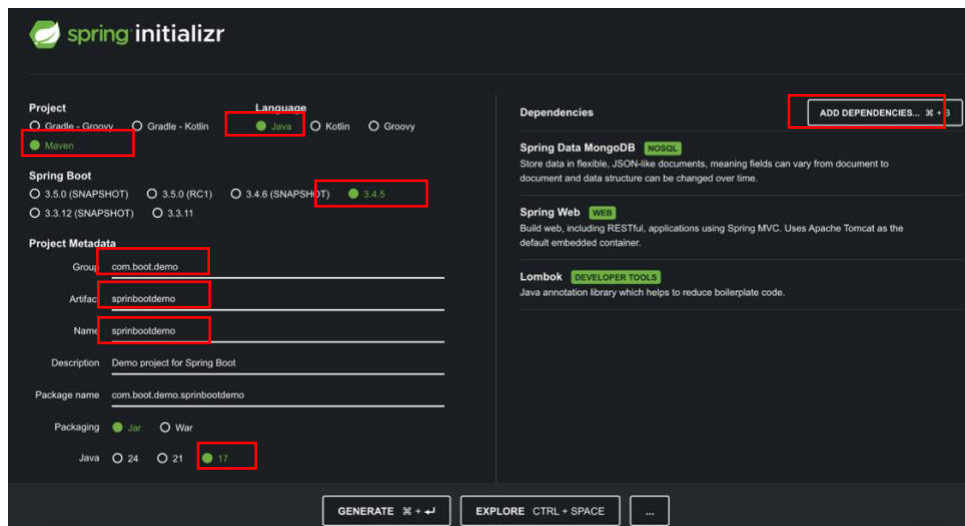


## Table of Contents

|  |           |
|--|-----------|
| <b>Step 1: Create Spring Boot Project .....</b>  | <b>2</b>  |
| <b>Step 2: Understand Spring Boot as opinionated.....</b>                              | <b>3</b>  |
| <b>Step 3: Understand Spring object creation .....</b>                                 | <b>4</b>  |
| <b>Step 4: Understand DI (@Autowired ) , @RestController and @RequestMapping .....</b> | <b>6</b>  |
| <b>Step 5: @PathVariable .....</b>   | <b>8</b>  |
| <b>Step 6: @RequestParam .....</b>   | <b>10</b> |
| <b>Step 7: @PostMapping .....</b>  | <b>11</b> |
| <b>Step 8: @PutMapping.....</b>  | <b>14</b> |
| <b>Step 9: @DeleteMapping .....</b>  | <b>15</b> |
| <b>Step 10: XML Response.....</b>  | <b>16</b> |
| <b>Step 11: Exception Handling.....</b>  | <b>17</b> |

## Step 1: Create Spring Boot Project

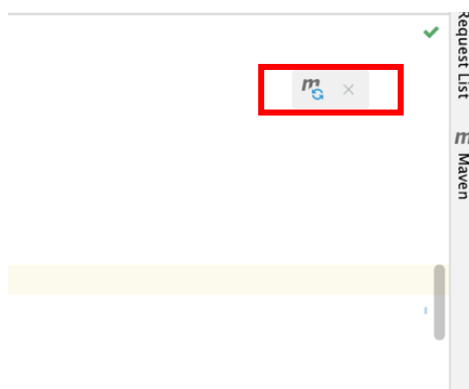
1. Please download **POSTMAN** on your VM.  
<https://www.postman.com/downloads/>
2. Open below url on the browser:  
<https://start.spring.io/>
3. Create spring boot project as shown on the screen below:



4. Click on Generate, it will download the zip. Extract and open project using IntelliJ
5. Once opened on INTELLIJ, please add below dependency in pom.xml: What this is for will discuss at later stage.

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.0.3</version>
</dependency>
```

After adding the dependency, you will get an option to reload in pom.xml as follows: Please click on that:



6. The embedded tomcat with spring boot web includes a light weight server which is the tomcat core and is capable of processing HTTP requests and send JSON as a response
7. **Following the package structure is very important with spring boot for it to follow the default configurations.**

## Step 2: Understand Spring Boot as opinionated

1. View the pom.xml file that has spring boot starter parent.

It is a special starter project that provides default configurations for our application and a complete dependency tree to quickly build our Spring Boot project.

It also provides default configurations for Maven plugins, such as maven-failsafe-plugin, maven-jar-plugin, maven-surefire-plugin, and maven-war-plugin.

Beyond that, it also inherits dependency management from spring-boot-dependencies, which is the parent to the spring-boot-starter-parent.

2. @SpringBootApplication on the class with the main method:

This annotation is used to enable three features, that is:

- a. @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
- b. @ComponentScan: enable @Component scan on the package where the application is located.
- c. @Configuration: allow to register extra beans in the context or import additional configuration classes

3. Since boot is opinionated framework, it looks for database configuration as it found dependency in the build path. Add below in application.properties:

```
spring.data.mongodb.uri=mongodb+srv://<username>:<password>@democluster.6c2vj.mongodb.net/?retryWrites=true&w=majority&appName=DemoCluster
```

# for local mongodb

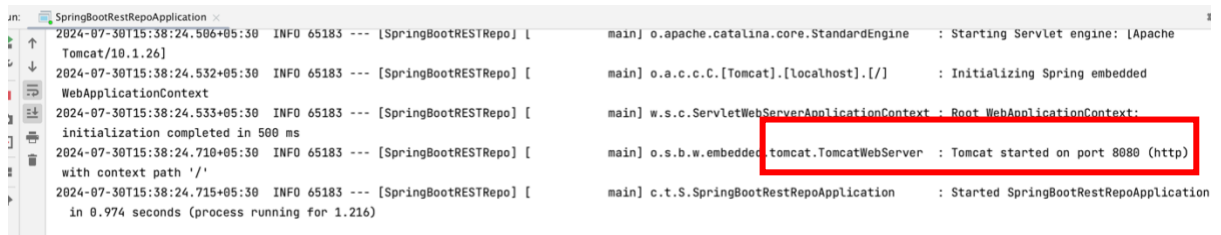
# spring.data.mongodb.uri=mongodb://localhost:27017/telecomdb

logging.level.org.springframework.data.mongodb.core.MongoTemplate=DEBUG

logging.level.org.mongodb.driver.protocol.command=DEBUG

spring.data.mongodb.database=telecomdb

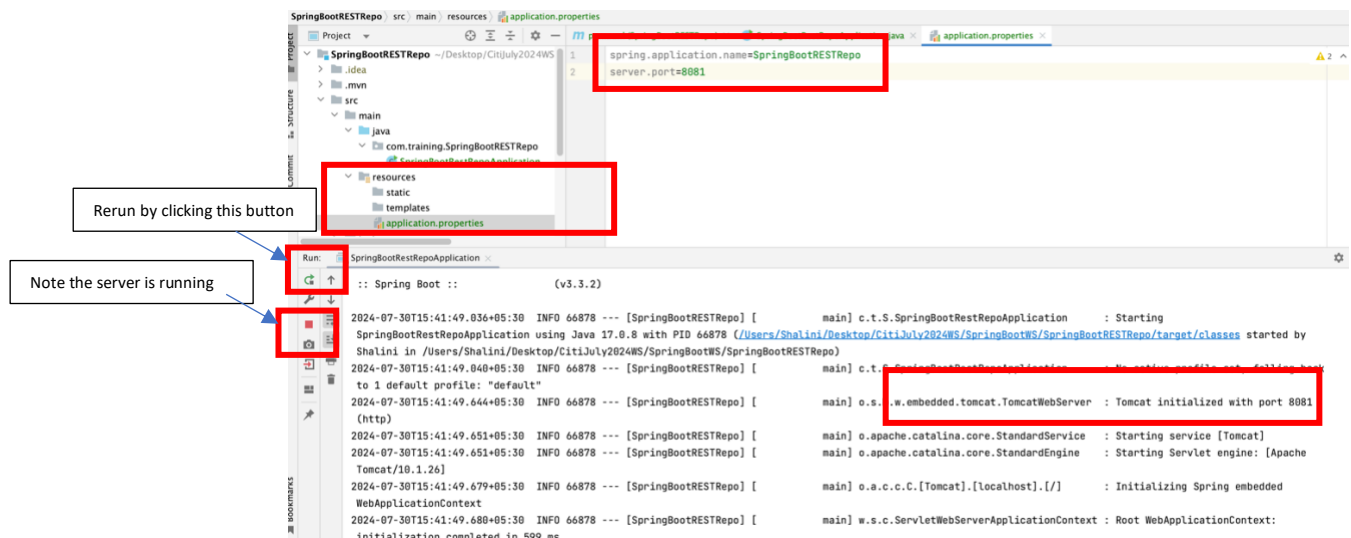
4. Run the main method and observe the console.



```
2024-07-30T15:38:24.586+05:30 INFO 65183 --- [SpringBootREStRepo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.26]
2024-07-30T15:38:24.532+05:30 INFO 65183 --- [SpringBootREStRepo] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-07-30T15:38:24.533+05:30 INFO 65183 --- [SpringBootREStRepo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 500 ms
2024-07-30T15:38:24.710+05:30 INFO 65183 --- [SpringBootREStRepo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2024-07-30T15:38:24.715+05:30 INFO 65183 --- [SpringBootREStRepo] [main] c.t.S.SpringBootRestRepoApplication : Started SpringBootRestRepoApplication in 0.974 seconds (process running for 1.216)
```

- a. Notice tomcat is running on port 8080
- b. Tomcat server by default runs on port 8080. To change the port add below in application.properties file  
server.port=8081

Rerun the project and you will see the output as below



Rerun by clicking this button

Note the server is running

```
spring.application.name=SpringBootREStRepo
server.port=8081
```

```
2024-07-30T15:41:49.036+05:30 INFO 66878 --- [SpringBootREStRepo] [main] c.t.S.SpringBootRestRepoApplication : Starting SpringBootRestRepoApplication using Java 17.0.8 with PID 66878 (/Users/Shalini/Desktop/CitiJuly2024WS/SpringBootWS/SpringBootREStRepo/target/classes started by ShaLini in /Users/Shalini/Desktop/CitiJuly2024WS/SpringBootWS/SpringBootREStRepo)
2024-07-30T15:41:49.040+05:30 INFO 66878 --- [SpringBootREStRepo] [main] c.t.S.SpringBootRestRepoApplication : No active profile found, using default: "default"
2024-07-30T15:41:49.644+05:30 INFO 66878 --- [SpringBootREStRepo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8081 (http)
2024-07-30T15:41:49.651+05:30 INFO 66878 --- [SpringBootREStRepo] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-07-30T15:41:49.651+05:30 INFO 66878 --- [SpringBootREStRepo] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.26]
2024-07-30T15:41:49.679+05:30 INFO 66878 --- [SpringBootREStRepo] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-07-30T15:41:49.680+05:30 INFO 66878 --- [SpringBootREStRepo] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 599 ms
```

### Step 3: Understand Spring object creation

1. Create a class Book as follows:

```
package com.boot.demo.springbootdemo.entity;

public class Book {

    private int bookid;
    private String title;
    private String author;
    private String desc;
    private double price;

    public Book() {

    }

    public Book(int bookid, String title, String author, String desc, double price) {
        this.bookid = bookid;
        this.title = title;
        this.author = author;
        this.desc = desc;
        this.price = price;
    }

    public Book(String title, String author, String desc, double price) {
        this.title = title;
        this.author = author;
        this.desc = desc;
        this.price = price;
    }

    @Override
    public String toString() {
        return "Book{" +
            "bookid=" + bookid +
            ", title=" + title + "\n" +
            ", author=" + author + "\n" +
            ", desc=" + desc + "\n" +
            ", price=" + price +
            "}";
    }

    public int getBookid() {
        return bookid;
    }

    public void setBookid(int bookid) {
        this.bookid = bookid;
    }

    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;
    }

    public String getDesc() {
        return desc;
    }
}
```

```

    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}

```

- 1 . Create service class as follows that will provide with book details:

```

package com.boot.demo.springbootdemo.service;

import com.boot.demo.springbootdemo.entity.Book;
import java.util.ArrayList;
import java.util.List;

public class BookService {

    private List<Book> bookList;

    public BookService() {
        System.out.println("Book service default constructor");
        bookList = new ArrayList<>();
        bookList.add(
            new Book(1, "Core Java", "Hotsmann", "Learn java fundamentals", 130.0));
        bookList.add(
            new Book(2, "HTML", "Kelly", "Learn html for UI", 230.0));
        bookList.add(
            new Book( 3, "python", "ryan", "Learn python fundamentals", 130.0));
        bookList.add(
            new Book( 4, "css", "kelly", "Learn css for designing webpage", 130.0));
    }

    public long getTotalBookCount(){
        return bookList.size();
    }
    public List<Book> getAllBooks(){
        return bookList;
    }
    public Book addNewBook(Book book){

        for (Book ob : bookList){
            if(ob.getBookid() == book.getBookid())
                throw new RuntimeException("Book with id "+book.getBookid()+" already exists");
        }
        Book lastBook = bookList.get(bookList.size()-1);
        int id = lastBook.getBookid()+ 1;
        book.setBookid(id);
        bookList.add(book);
        return book;
    }

    public Book updateBook(Book book){
        for (int i=0;i<bookList.size();i++){
            if(bookList.get(i).getBookid() == book.getBookid()) {
                bookList.set(i, book);
                return book;
            }
        }
        throw new RuntimeException("Book with id "+book.getBookid()+" does not exist");
    }
    public boolean deleteBook(int id){
        for (int i=0;i<bookList.size();i++){
            if(bookList.get(i).getBookid() == id) {
                bookList.remove(i);
                return true;
            }
        }
    }
}

```

```

        throw new RuntimeException("Book with id "+id+" does not exist");
    }
    public List<Book> getBooksByAuthor(String author){
        List<Book> booksByAuthor = new ArrayList<>();
        for (Book ob : bookList){
            if(ob.getAuthor().equalsIgnoreCase (author))
                booksByAuthor.add(ob);
        }
        return booksByAuthor;
    }
    public Book getBookById(int id){

        for (Book ob : bookList) {
            if (ob.getBookid() == id)
                return ob;
        }
        throw new RuntimeException("Book with id "+id+" does not exists");
    }
}

```

- Using spring specific annotations will automatically load and instantiate the classes. Since we need BookService class object just annotate as follows:

```

@Service
public class BookService {

    private List<Book> bookList;
    // other parts are same
}

```

- The classes that are loaded and instantiated by spring are called as “**SPRING MANAGED BEANS**”
- Just Rerun the server and should see output from the constructor on the console

```

Run: SpringBootRestRepoApplication
Spring embedded WebApplicationContext
2024-07-30T16:44:18.482+05:30 INFO 97019 --- [SpringBootRestRepo] [main] w.s.c.ServletWebServerApplicationContext : Root
WebApplicationContext: initialization completed in 529 ms
Book service default constructor
2024-07-30T16:44:18.518+05:30 INFO 97019 --- [SpringBootRestRepo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat sta
port 8081 (http) with context path '/'
2024-07-30T16:44:18.574+05:30 INFO 97019 --- [SpringBootRestRepo] [main] c.t.S.SpringBootRestRepoApplication : Started
SpringBootRestRepoApplication in 1.073 seconds (process running for 1.467)

```

#### Step 4: Understand DI (@Autowired) , @RestController and @RequestMapping

- To inform spring that a class is exposing data over HTTP protocol, we need to use RestController annotation on the class.
- Create a class BookRestController as follows:

```

package com.boot.demo.springbootdemo.rest;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class BookRestController {

    public BookRestController() {
        System.out.println("Book Rest Controller default constructor");
    }

}

```

- Rerun the application and should see the output from Rest Controller class as follows:

```

Run: SpringBootRestRepoApplication
2024-07-30T16:58:14.977+05:30 INFO 3994 --- [SpringBootRestRepo] [main] w.s.c.ServletWebServerApplicationContext : Root
WebApplicationContext: initialization completed in 502 ms
Book Rest Controller default constructor
Book service default constructor
2024-07-30T16:58:15.157+05:30 INFO 3994 --- [SpringBootRestRepo] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on
port 8081 (http) with context path '/'
2024-07-30T16:58:15.159+05:30 INFO 3994 --- [SpringBootRestRepo] [main] c.t.S.SpringBootRestRepoApplication : Started
SpringBootRestRepoApplication in 0.969 seconds (process running for 1.18)

```

4. This class needs reference of BookService class to get the data. Update BookRestController as follows:

```
@RestController
public class BookRestController {

    private BookService bookService;
    // other parts are same

    public List<Book> getBooks() {
        return bookService.getAllBooks();
    }
}
```

5. Normally we provide dependencies as follows:

```
BookService bs = new BookService();
BookRestController ob = new BookRestController(bs);
```

6. Since BookService is a spring managed bean, we need to tell spring to inject this dependency. Update the code as follows:

```
@RestController
public class BookRestController {

    @Autowired
    private BookService bookService;
    // other parts are same
}
```

@Autowired annotation tells spring about the dependency and it looks for the bean within its context and if found inject it.

7. To expose data annotate the class with @RequestMapping to specify the exposed endpoint URI and update method with @GetMapping annotation to fetch data.

```
@RestController
@RequestMapping("/books")
public class BookRestController {
    // other parts are same

    @GetMapping
    public List<Book> getBooks() {
        return bookService.getAllBooks();
    }
}
```

#### MAKE SURE TO RESTART THE SERVER

It will be available at <http://localhost:8081/books>.

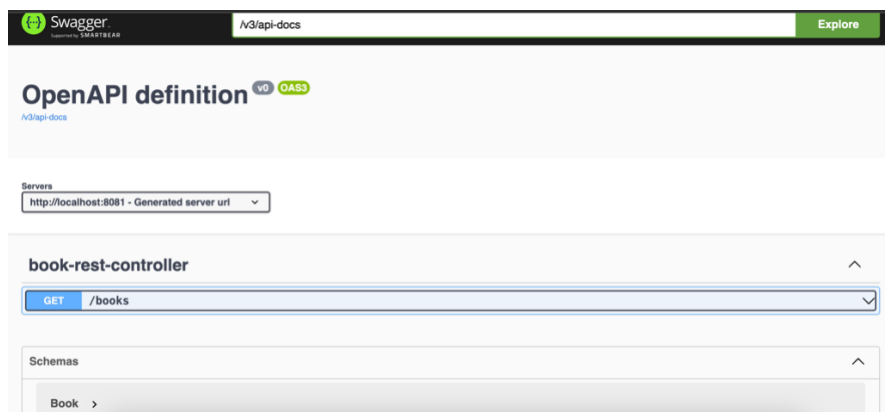
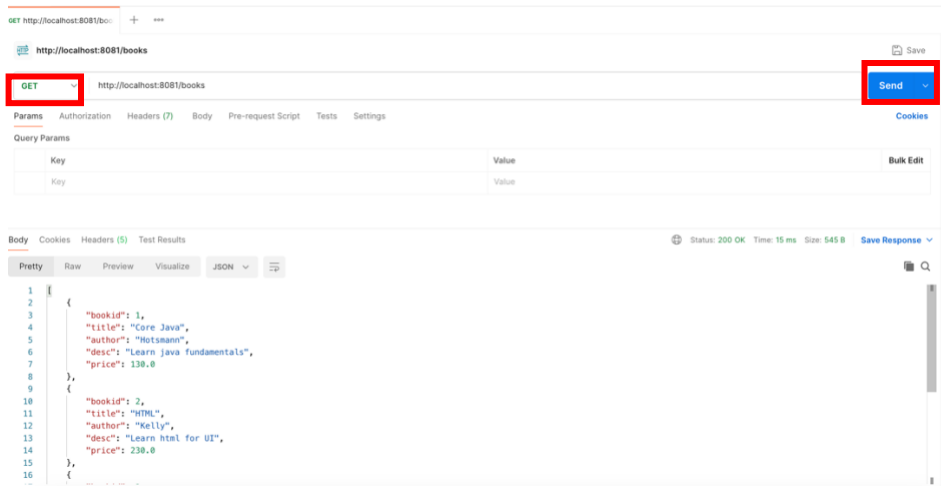
8. To check if it is working open POSTMAN and check if REST endpoint is working as follows:

- Type url in the address bar
- Make sure to select GET from dropdown
- Click on Send
- Should see the output as follows:

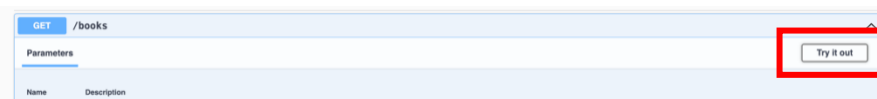
9. Alternatively it can be checked using swagger. Swagger is used for REST endpoints documentation and testing. Go on to browser and type in the below url:

<http://localhost:8081/swagger-ui/index.html/>

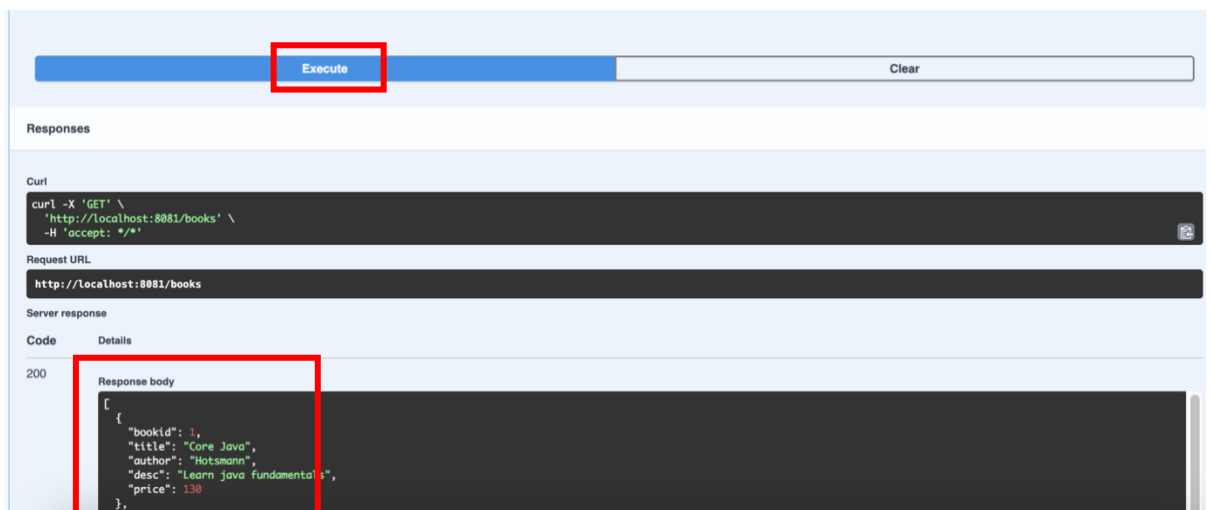
Should see the output as below:



When you click on the arrow, you get Try it out. Click on that will have an option to Execute.



Click on Execute and should see the JSON response as shownn below:



#### Step 5: @PathVariable

1. Update Controller and add below method to return book by id as follows:

```
public Book getBookById(int id){
```



```

        return bookService.getBookById(id);
    }

```

- To make this method available at REST API endpoint and return book for a specific id, update the method as follows:

```

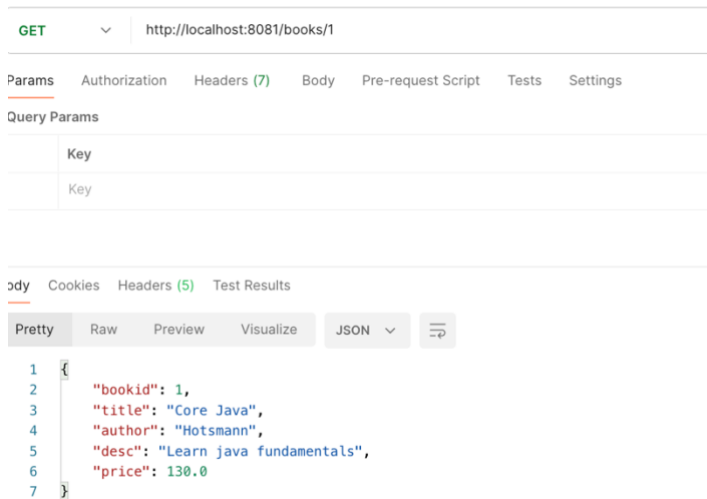
@GetMapping("/{id}")
public Book getBookById(@PathVariable int id){
    return bookService.getBookById(id);
}

```

### MAKE SURE TO RESTART THE SERVER

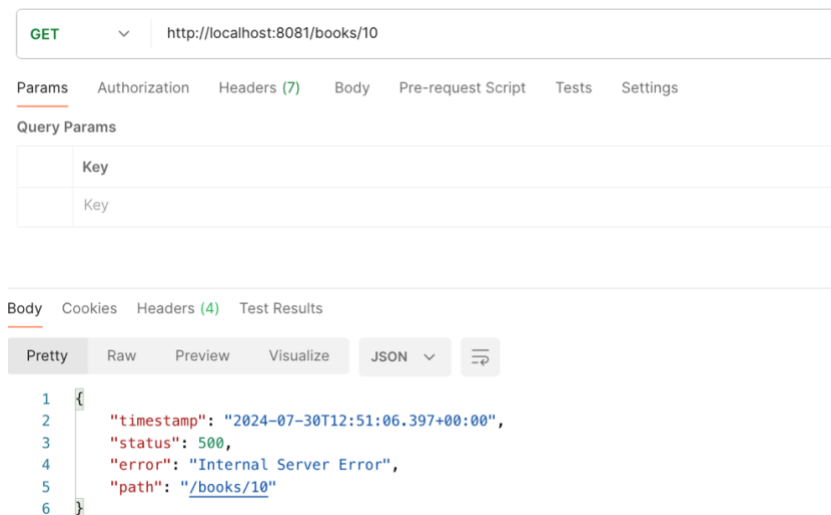
To access type in postman url : <http://localhost:8081/books/1>

Make sure GET is selected in the dropdown and click on send. Should see the details of book by id 1.

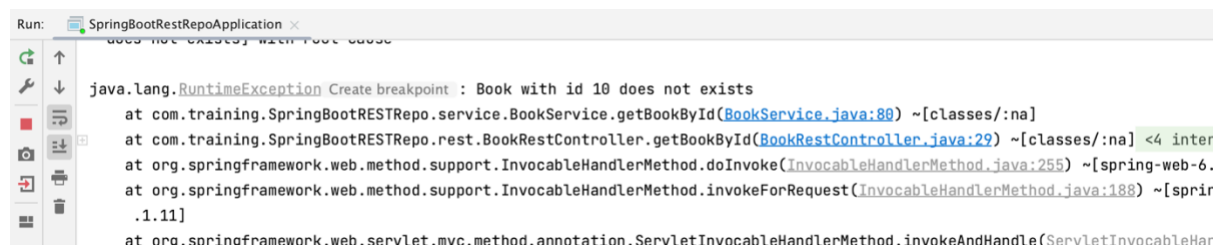


`{}` -> is the placeholder for the value [1] passed in the url.  
`{id}` is mapped to method parameter id using `@PathVariable` annotation.

- Now try to access a book that does not exists: You should get following screen on postman:



And on IDE console, you will see the exception:



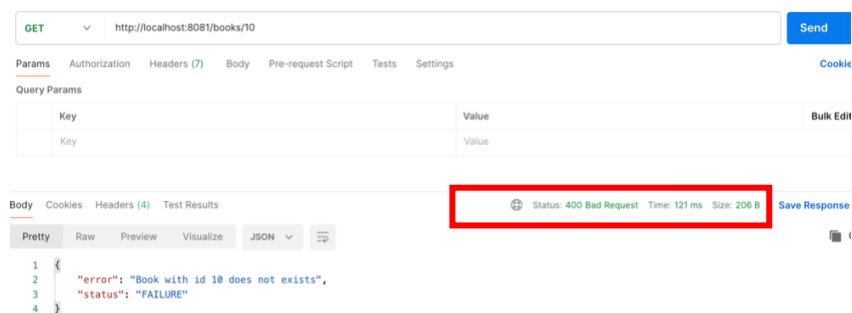
```
Run: SpringBootRestRepoApplication
java.lang.RuntimeException Create breakpoint : Book with id 10 does not exists
    at com.training.SpringBootRESTRepo.service.BookService.getBookById(BookService.java:80) ~[classes/:na]
    at com.training.SpringBootRESTRepo.rest.BookRestController.getBookById(BookRestController.java:29) ~[classes/:na]
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:255) ~[spring-web-6.1.11]
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:188) ~[spring-web-6.1.11]
    at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:118) ~[spring-web-6.1.11]
```

4. Displaying internal server error is not a good practice. Let's modify the code to handle the exception and return an appropriate response along with respective status code. Spring provides with `ResponseEntity` class to wrap the data and any extra information to be returned.

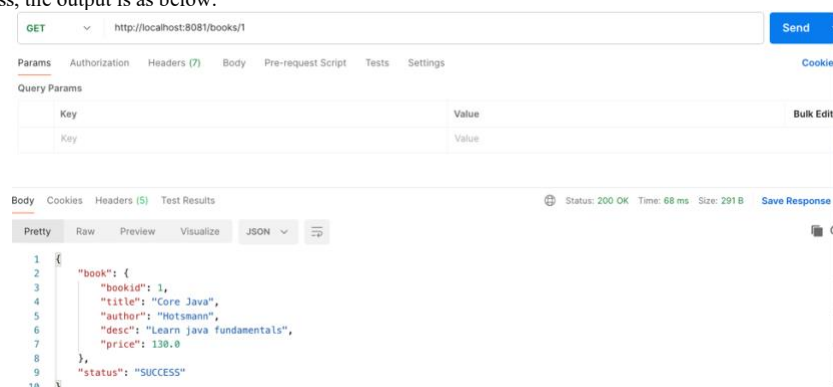
```
@GetMapping("/{id}")
public ResponseEntity<Object> getBookById(@PathVariable int id){
    Map<String, Object> map = new HashMap<>();
    try {
        map.put(AppConstants.STATUS, Status.SUCCESS);
        map.put("book", bookService.getBookById(id));
        return ResponseEntity.ok(map);
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}
```

DO CHECK THE UTILITY PACKAGE FOR `AppConstants` and `Status` used here.

## MAKE SURE TO RESTART THE SERVER



For success, the output is as below:



## Step 6: @RequestParam

1. Get books method returns all the books. How about we need to give users choice to get books filtered by author? This has to be optional if no filter provided then return all books. Use `RequestParam` annotation for the same: Update the method as follows:

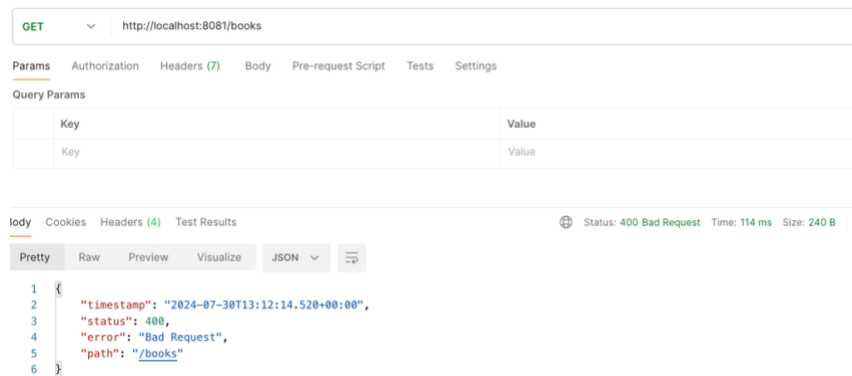
```

@GetMapping
public List<Book> getBooks(@RequestParam String author){
    if(author==null)
        return bookService.getAllBooks();
    return bookService.getBooksByAuthor(author);
}

```

### MAKE SURE TO RESTART THE SERVER

To access type in postman url : <http://localhost:8081/books>  
 You will get below error as value for author was not provided.



Now access with this url : <http://localhost:8081/books?author=kelly>

- But the problem is providing value for author is mandatory. Update the method to make author as required false.

```

@GetMapping
public List<Book> getBooks(@RequestParam(required = false) String author){
    if(author==null)
        return bookService.getAllBooks();
    return bookService.getBooksByAuthor(author);
}

```

### MAKE SURE TO RESTART THE SERVER

Now this url works just fine without providing the value for author : <http://localhost:8081/books>

## Step 7: @PostMapping

- To add new book we use @PostMapping annotation. Add below method in controller:

```

@PostMapping
public ResponseEntity<Object> addBook(Book book){
    System.out.println("Book "+book);
    Map<String, Object> map = new HashMap<>();
    try {
        map.put(AppConstants.STATUS, Status.SUCCESS);
        map.put("book",bookService.addNewBook(book) );
        return ResponseEntity.ok(map);
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error",e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}

```

### MAKE SURE TO RESTART THE SERVER

Try this url in postman. Make sure to select **POST** from dropdown of POSTMAN :  
<http://localhost:8081/books>

**ALSO PLEASE UPDATE HEADER AS FOLLOWS:**

POST http://localhost:8081/books

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Headers 9 hidden

| Key  | Value            |
|--|------------------|
| <input checked="" type="checkbox"/> Content-Type | application/json |
| Key  | Value            |

Add below JSON in body

```
{
  "title": "AWS",
  "author": "ryan",
  "desc": "Learn aws fundamentals",
  "price": 530.0
}
```

You will get below output on postman: HMMMM??? 😊

POST http://localhost:8081/books

Body raw JSON

```
{
  "title": "AWS",
  "author": "ryan",
  "desc": "Learn aws fundamentals",
  "price": 530.0
}
```

Status: 200 OK Time: 121 ms Size: 255 B

Response Body (Pretty):

```
{
  "book": {
    "bookid": 5,
    "title": null,
    "author": null,
    "desc": null,
    "price": 0.0
  },
  "status": "SUCCESS"
}
```

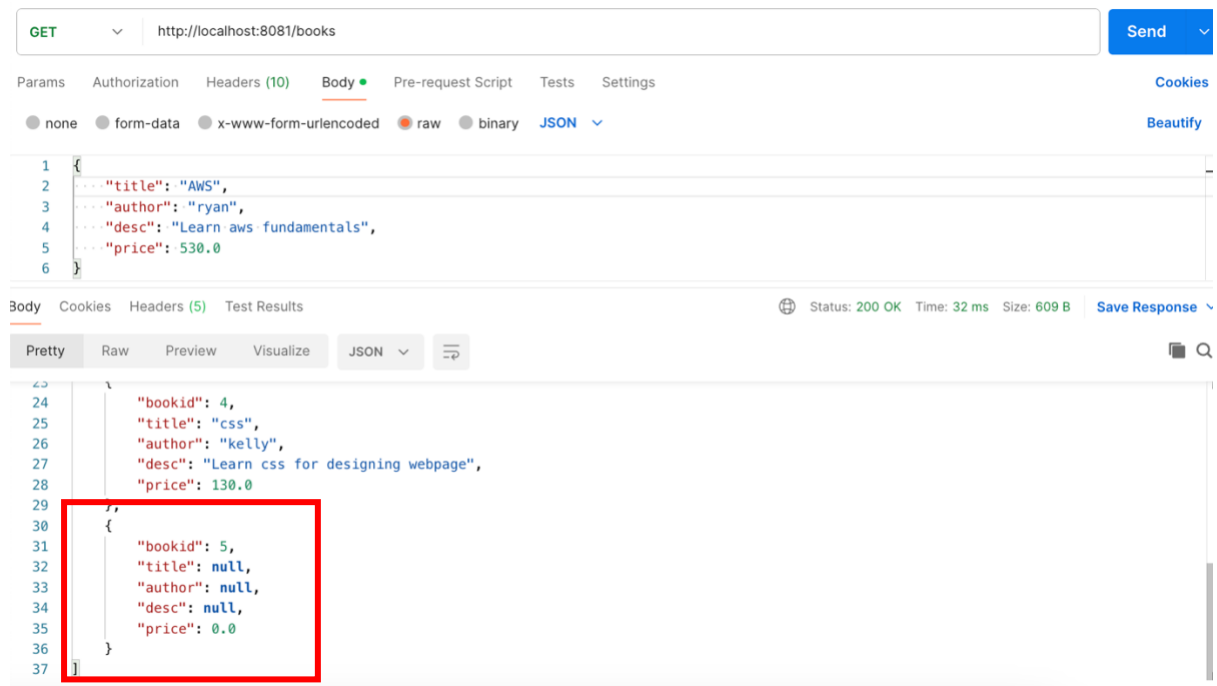
Check the IDE console. WHAT??? Book data is null 😊

Run: SpringBootRestRepoApplication

SpringBootRestRepoApplication in 0.954 seconds (process running for 1.167)

```
2024-07-30T19:02:11.243+05:30 INFO 63050 --- [SpringBootRESTRepo] [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/
Spring DispatcherServlet 'dispatcherServlet'
2024-07-30T19:02:11.244+05:30 INFO 63050 --- [SpringBootRESTRepo] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
Servlet 'dispatcherServlet'
2024-07-30T19:02:11.246+05:30 INFO 63050 --- [SpringBootRESTRepo] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet
Initialization in 2 ms
Book Book{bookid=0, title='null', author='null', desc='null', price=0.0}
```

Check fetching records : Make a GET request to <http://localhost:8081/books>



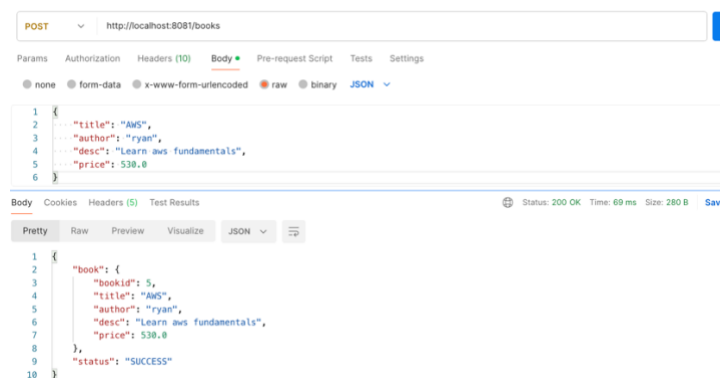
- Looks like spring was not able to map the data coming in the request to java class. We need to add `@RequestBody` in the method parameter for spring to know to do the mapping of JSON data to java class.

#### @PostMapping

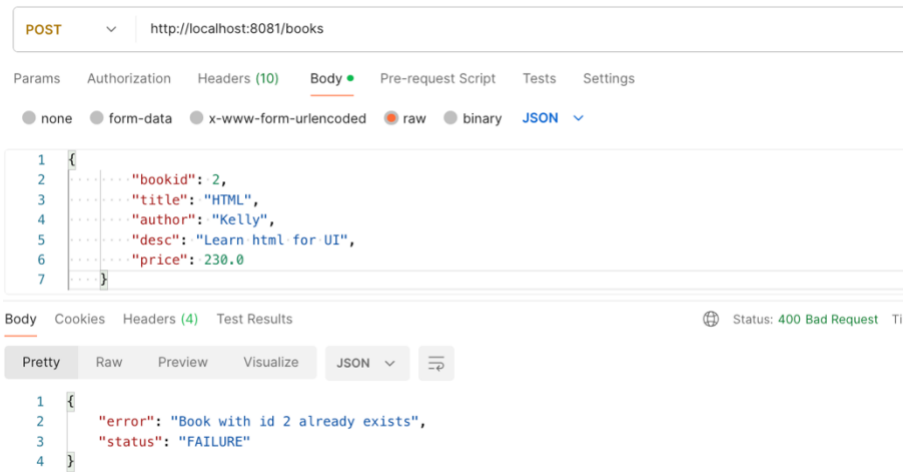
```
public ResponseEntity<Object> addBook(@RequestBody Book book){
    System.out.println("Book "+book);
    Map<String, Object> map = new HashMap<>();
    try {
        map.put(AppConstants.STATUS, Status.SUCCESS);
        map.put("book",bookService.addNewBook(book) );
        return ResponseEntity.ok(map);
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error",e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}
```

#### MAKE SURE TO RESTART THE SERVER

Now checking the above url's for POST will work as expected: DO NOT FORGET THE **HEADER**



Also do check for adding an already existing book. Should get output as follows: DO NOT FORGET THE **Header**



## Step 8: @PutMapping

1. To update a book add below method:

```
@PutMapping
public ResponseEntity<Object> updateBook(@RequestBody Book book){
    System.out.println("Book "+book);
    Map<String, Object> map = new HashMap<>();
    try {
        map.put(AppConstants.STATUS, Status.SUCCESS);
        map.put("book",bookService.updateBook(book) );
        return ResponseEntity.ok(map);
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error",e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}
```

**MAKE SURE TO RESTART THE SERVER**

**ALSO PLEASE UPDATE HEADER :**

**Content-Type: application/json**

PUT http://localhost:8081/books

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```

1 {
2   "bookid": 1,
3   "title": "Core Java and Functional Programming",
4   "author": "Cay Hotsmann",
5   "desc": "Learn java fundamentals",
6   "price": 230.0
7 }

```

body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "book": {
3     "bookid": 1,
4     "title": "Core Java and Functional Programming",
5     "author": "Cay Hotsmann",
6     "desc": "Learn java fundamentals",
7     "price": 230.0
8   },
9   "status": "SUCCESS"
10 }

```

Try to update a book that does not exist:

PUT http://localhost:8081/books

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```

1 {
2   "bookid": 20,
3   "title": "HTML",
4   "author": "Kelly",
5   "desc": "Learn html for UI",
6   "price": 230.0
7 }

```

body Cookies Headers (4) Test Results Status: 400 Bad Req

Pretty Raw Preview Visualize JSON

```

1 {
2   "error": "Book with id 20 does not exist",
3   "status": "FAILURE"
4 }

```

## Step 9: @DeleteMapping

1. To delete a book add below method:

```

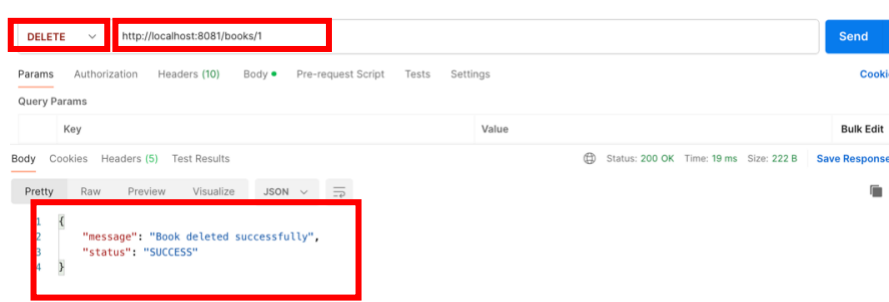
@DeleteMapping("/{id}")
public ResponseEntity<Object> deleteBook(@PathVariable int id){
    Map<String, Object> map = new HashMap<>();
    try {
        map.put(AppConstants.STATUS, Status.SUCCESS);
        if(bookService.deleteBook(id)) {
            map.put("message", "Book deleted successfully");
            return ResponseEntity.ok(map);
        }
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error", e.getMessage());
    }
}

```

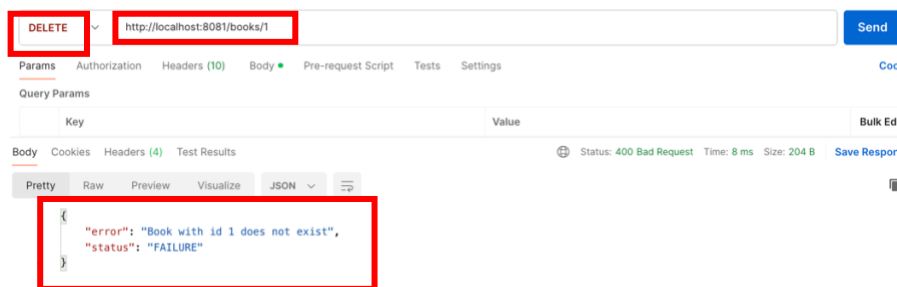
```
        return ResponseEntity.badRequest().body(map);
    }
}
```

**MAKE SURE TO RESTART THE SERVER**

Try for success deletion for a book that exists with the id:



Try for failure deletion for a book that does not exist with the id:



#### Step 10: XML Response

1. Add below dependency in pom.xml

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

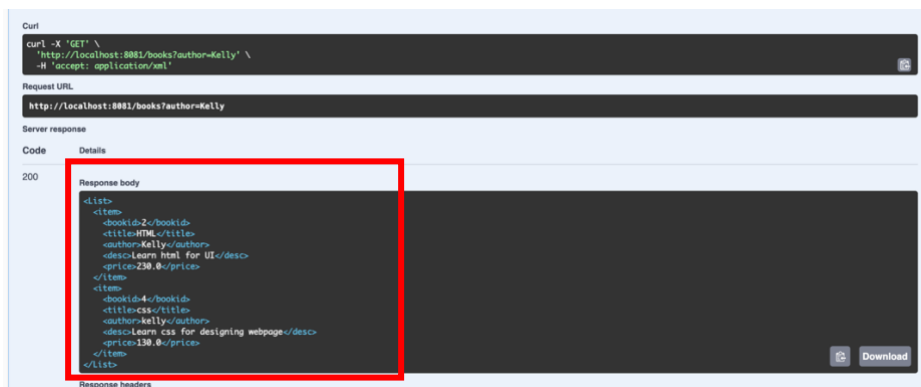
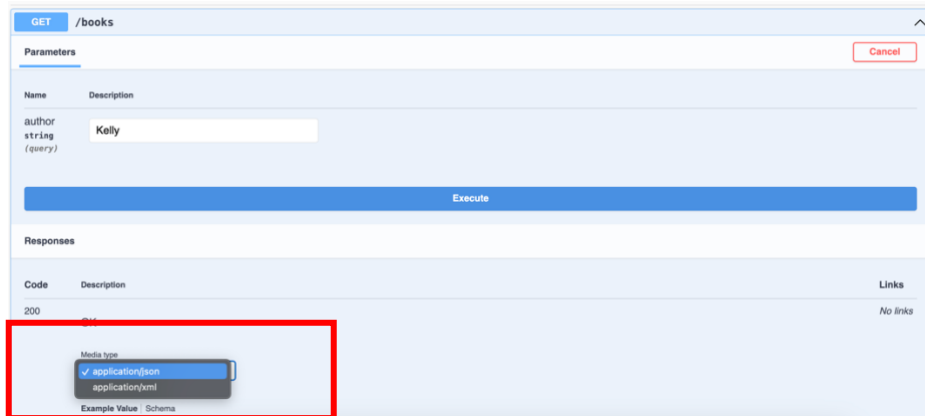
2. Modify the `getBooks` method as follows to produce data of type JSON and XML both:

```
@GetMapping(produces = {"application/json", "application/xml"})
public List<Book> getBooks (@RequestParam(required = false, defaultValue = "Kelly") String author) {
    if(author.equalsIgnoreCase("all"))
        return this.bookService.getAllBooks();
    return this.bookService.getBooksByAuthor(author);
}
```

3. Restart the server and try on postman or Swagger and you should get XML data as follows:

Swagger provides you with the option to specify the response type as follows: Then when you choose xml or json and click on Execute button, the response will be of the type specified.





## Step 11: Exception Handling

### 1.1. Add below dependency in pom.xml

In previous BookRestController, every method is responsible for handling the exceptions. There is lots of repetition for the exception handler. Spring boot provides a central exception handler in 2 ways as follows:

#### 1.1.1. Use @ExceptionHandler

Spring Boot provides the @ExceptionHandler annotation to handle exceptions thrown by a specific controller method. This annotation can be used to provide customized error responses for specific exceptions.

#### 1.1.2. Create a new Rest Controller as follows and look at the method with @ExceptionHandler

```
package com.boot.demo.springbootdemo.rest;
import com.boot.demo.springbootdemo.entity.Book;
import com.boot.demo.springbootdemo.service.BookServiceRepo;
import com.boot.demo.springbootdemo.utility.AppConstants;
import com.boot.demo.springbootdemo.utility.Status;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
@RestController
@RequestMapping("/books")
public class BookRestControllerExceptionHandler {
```

```

@Autowired
private BookServiceRepo bookService;
public BookRestControllerExceptionHandler() {
    System.out.println("Book Rest Controller default constructor");
}

```

```

@ExceptionHandler(RuntimeException.class)
public ResponseEntity<Object> handleResourceNotFoundException(RuntimeException ex)
{
    Map<String, Object> body = new HashMap<>();
    body.put("error", ex.getMessage());
    return new ResponseEntity<>(body, HttpStatus.NOT_FOUND);
}

```

```

@GetMapping
public List<Book> getBooks(@RequestParam(required = false) String author){
    if(author==null)
        return bookService.getAllBooks();
    return bookService.getBooksByAuthor(author);
}

```

```

@GetMapping("/{id}")
public ResponseEntity<Object> getBookById(@PathVariable int id){
    Map<String, Object> map = new HashMap<>();
    map.put(AppConstants.STATUS, Status.SUCCESS);
    map.put("book",bookService.getBookById(id) );
    return ResponseEntity.ok(map);
}

```

```

@PostMapping
public ResponseEntity<Object> addBook(@RequestBody Book book){
    System.out.println("Book "+book);
    Map<String, Object> map = new HashMap<>();
    map.put(AppConstants.STATUS, Status.SUCCESS);
    map.put("book",bookService.addNewBook(book) );
    return ResponseEntity.ok(map);
}

```

```

@PutMapping
public ResponseEntity<Object> updateBook(@RequestBody Book book){
    System.out.println("Book "+book);
    Map<String, Object> map = new HashMap<>();

    map.put(AppConstants.STATUS, Status.SUCCESS);
    map.put("book",bookService.updateBook(book) );
    return ResponseEntity.ok(map);
}

```

```

@DeleteMapping("/{id}")
public ResponseEntity<Object> deleteBook(@PathVariable int id){
    Map<String, Object> map = new HashMap<>();

    map.put(AppConstants.STATUS, Status.SUCCESS);
    if(bookService.deleteBook(id)) {
        map.put("message", "Book deleted successfully");
    }
}

```

```

        return ResponseEntity.ok(map);
    }

    return ResponseEntity.badRequest().body(map);
}
}

```

#### 1.1.3. Use @ControllerAdvice

Spring Boot provides the @ControllerAdvice annotation to handle exceptions globally across all controllers. This annotation can be used to provide a centralized error handling mechanism for an entire application.

#### 1.1.4. Create a class as follows to handle global exception

```

package com.rest.SpringBootRestDemo.exception;

import jakarta.persistence.EntityNotFoundException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class MyGlobalHandler {

    MyGlobalHandler(){
        System.out.println("Global handler");
    }

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<Object>
    handleResourceNotFoundException(EntityNotFoundException ex) {
        Map<String, Object> body = new HashMap<>();
        body.put("message", ex.getMessage());

        return new ResponseEntity<>(body, HttpStatus.NOT_FOUND);
    }
}

```

---