Table of Contents

Step 1: Create Spring Boot Project

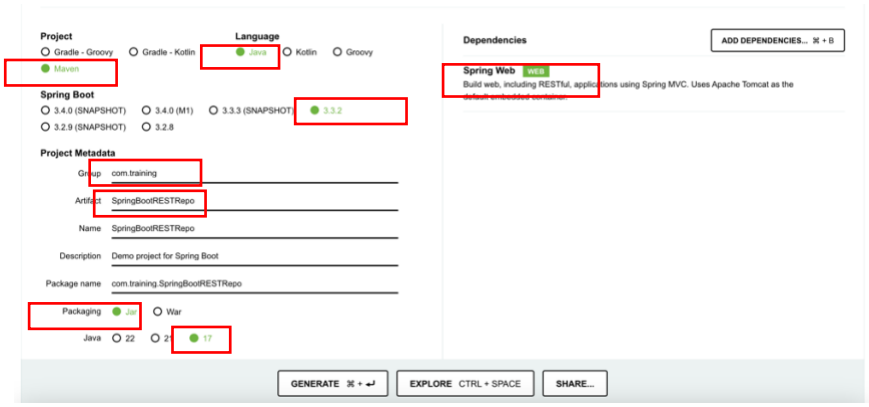
1. Bitbucket URL: https://bitbucket.org/shalini-ws/springbootrestrepo/src/main/

2. Please download **POSTMAN** on your VM.
   https://www.postman.com/downloads/

3. Open below url on the browser:

   https://start.spring.io/

4. Create spring boot project as shown on the screen below:

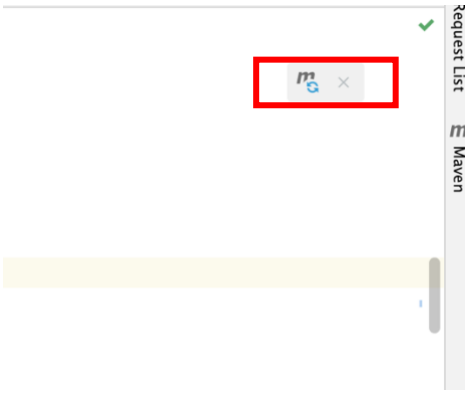


5. Click on Generate, it will download the zip. Extract and open project using IntelliJ

6. 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:



7. 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

8. 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

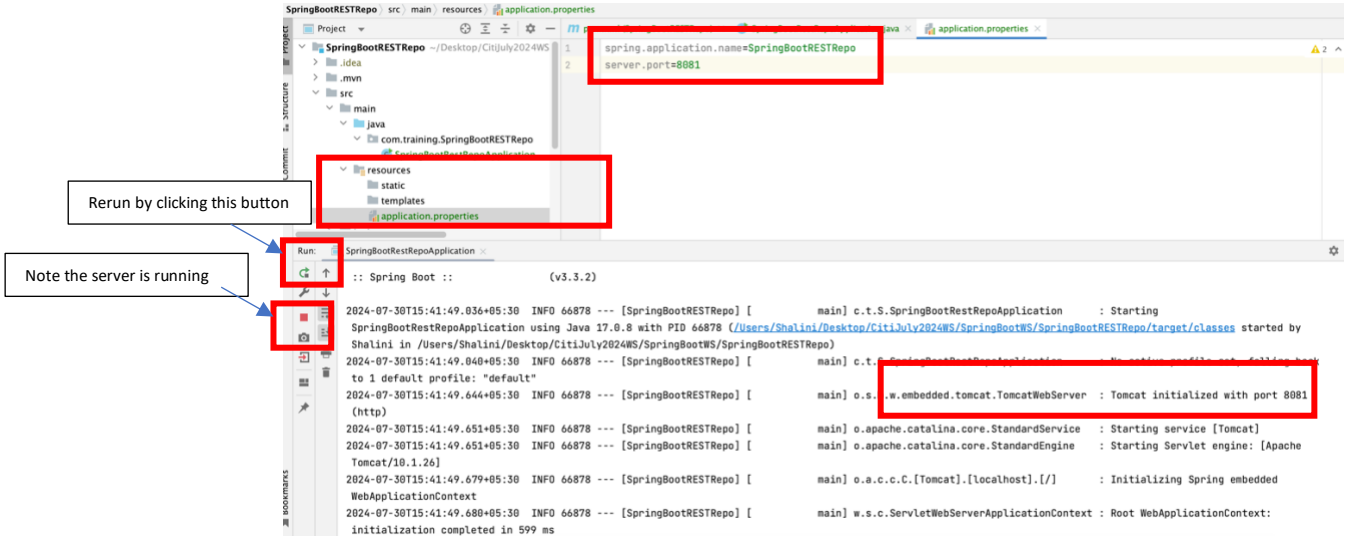
b. @ComponentScan: enable @Component scan on the package where the application is located.
c. @Configuration: allow to register extra beans in the context or import additional configuration classes

3. Run the main method and observe the console.

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



Step 3: Understand Spring object creation

1.Create a class Book as follows:

```
package com.training.SpringBootRESTRepo.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 + '\'' +
                ", author='" + author + '\'' +
                ", desc='" + desc + '\'' +
                ", 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;
    }
}
```

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

```
package com.training.SpringBootRESTRepo.service;

import com.training.SpringBootRESTRepo.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");

    }
}
```

3. 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
}
```

4. The classes that are loaded and instantiated by spring are called as **“SPRING MANAGED BEANS”**

5. Just Rerun the server and should see output from the constructor on the console

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

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

```
package com.training.SpringBootRESTRepo.rest;

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

@RestController
public class BookRestController {

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

}
```

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

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.

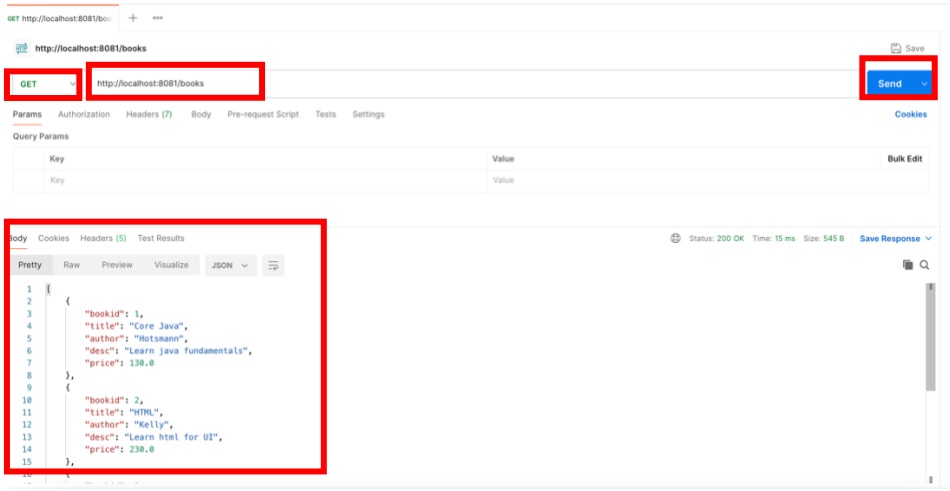
```
@RestController
@RequestMapping("/books")
public class BookRestController {
    // other parts are same

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

MAKE SURE TO RESTART THE SERVER

It will be available at http://localhost:8080/books.

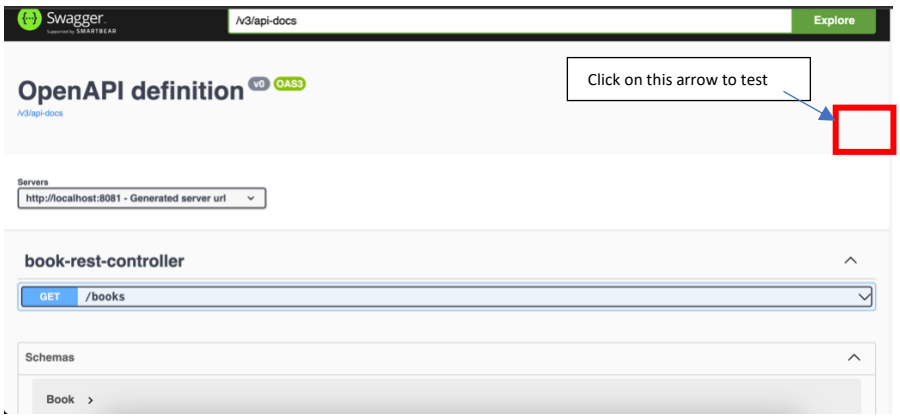
8. To check if it is working open POSTMAN and check if REST endpoint is working as follows:
    a. Type url in the address bar
    b. Make sure to select GET from dropdown
    c. Click on Send
    d. Should see the output as follows:



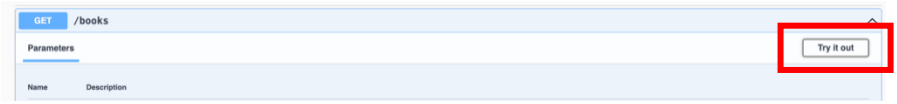
9. Alternatively it can be checked using swagger. Swagger is used for REST endpoints documentation and testing. Go on to browser and type in the below url:

http://localhost:8081/swagger-ui/index.html#/

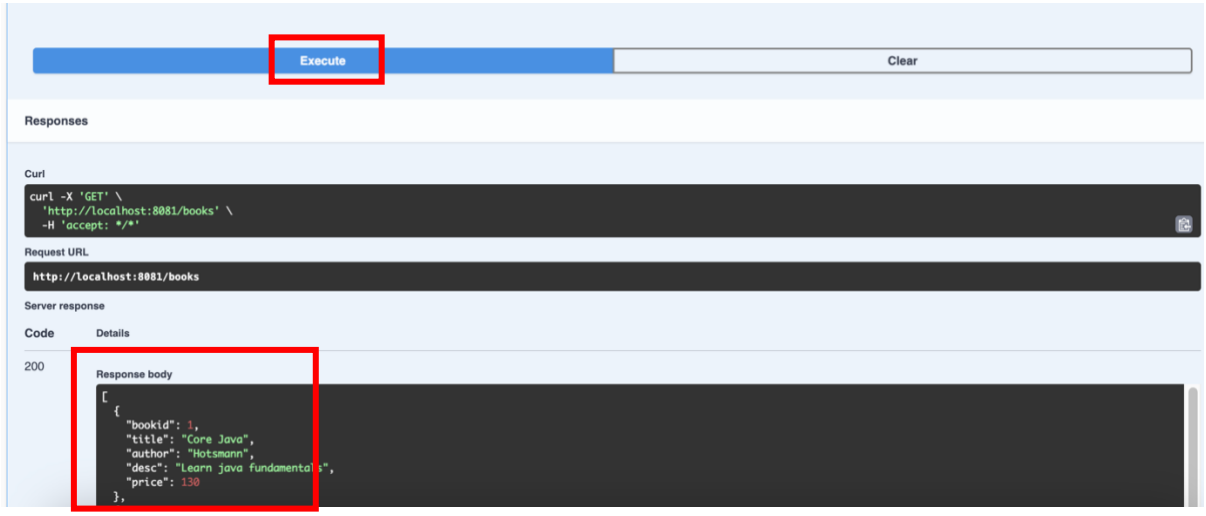
Should see the output as below:



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:

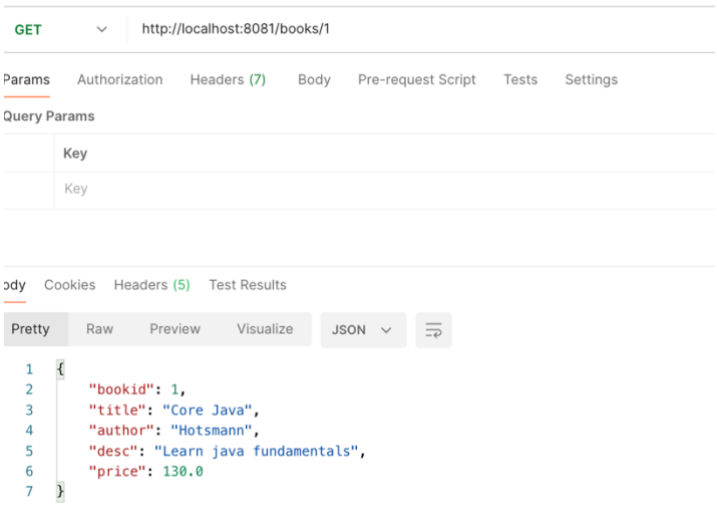
```
public Book getBookById(int id){
	return bookService.getBookById(id);
}
```

2. 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);
}
```

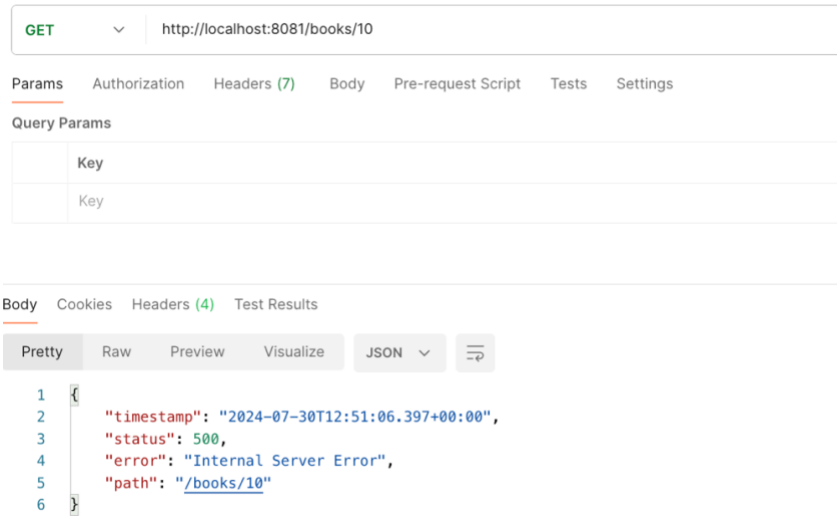
MAKE SURE TO RESTART THE SERVER

To access type in postman url : http://localhost:8080/books/1
Make sure GET is selected in the dropdown and click on send. Should see the details of book by id 1.

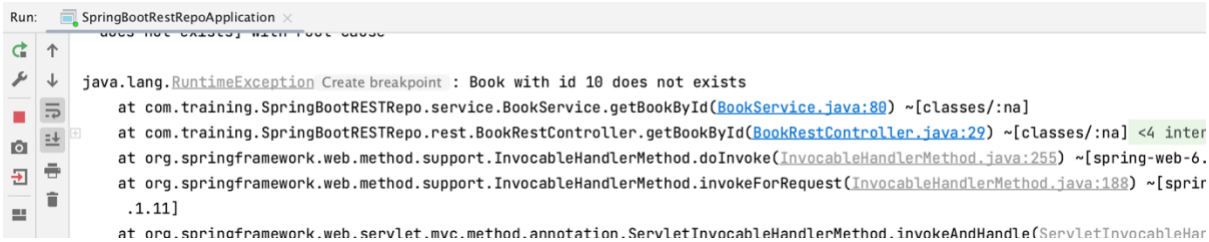


{} -> is the placeholder for the value [1] passed in the url.
{id} is mapped to method parameter id using @PathVariable annotation.

3. 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:

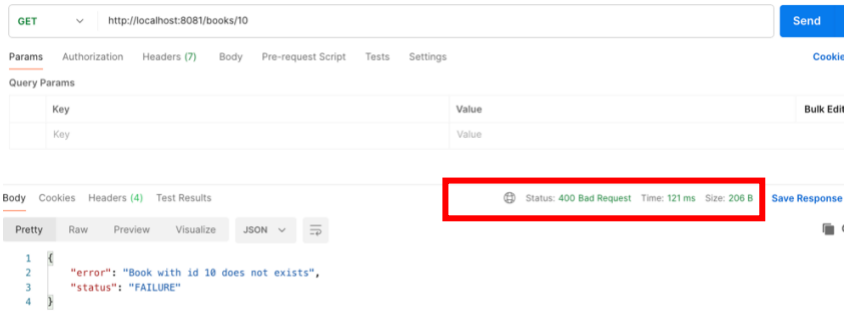


4. Displaying internal server error is not a good practice. Let's modify the code to handle the exception and return an appropriate response along with respective status code. Spring provides with ResponseEntity class to wrap the data and any extra information to be returned .

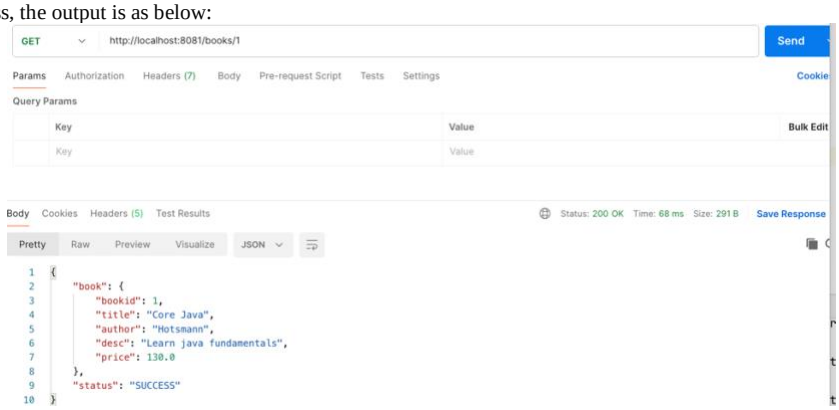
```
@GetMapping("/{id}")
public ResponseEntity<Object> getBookById(@PathVariable  int id){
   Map<String, Object> map = new HashMap<>();
   try {
      map.put(AppConstants.STATUS, Status.SUCCESS);
      map.put("book",bookService.getBookById(id) );
      return ResponseEntity.ok(map);
   }
   catch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

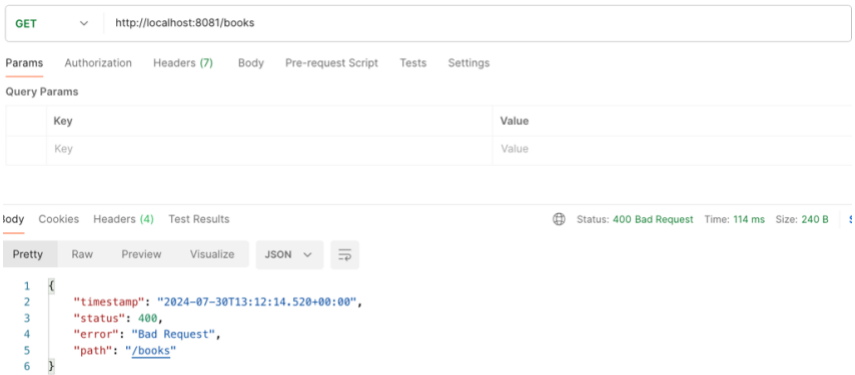
1. Get books method returns all the books. How about we need to give users choice to get books filtered by author? This has to be optional if no filter provided then return all books. Use RequestParam annotation for the same: Update the method as follows:

```
@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:8080/books
You will get below error as value for author was not provided.



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

2. 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:8080/books

Step 7: @PostMapping

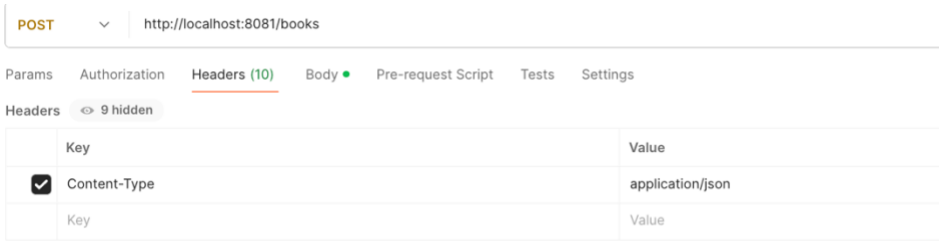
1. To add new book we use @PostMapping annotation. Add below method in controller:

```
@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:8080/books

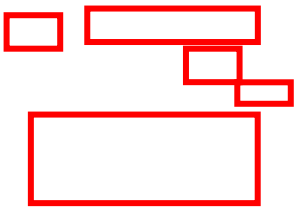
ALSO PLEASE UPDATE HEADER AS FOLLOWS:

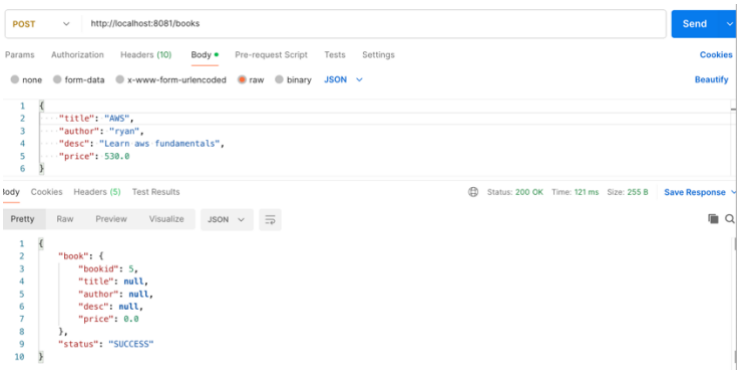


Add below JSON in body

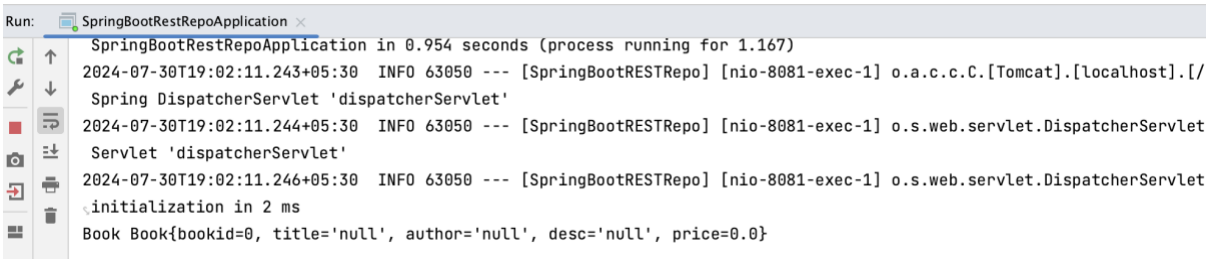
```
{
   "title": "AWS",
   "author": "ryan",
   "desc": "Learn aws fundamentals",
   "price": 530.0
}
```

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

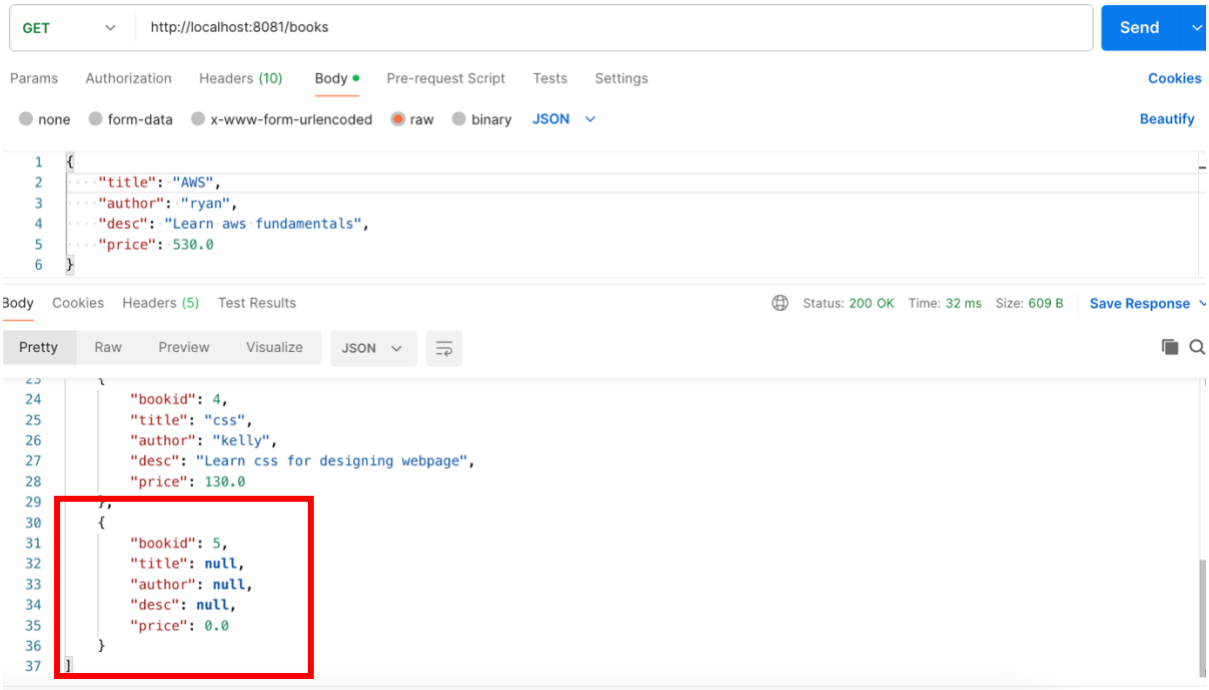




Check the IDE console. WHAT??? Book data is null 😉



Check fetching records : Make a GET request to http://localhost:8080/books



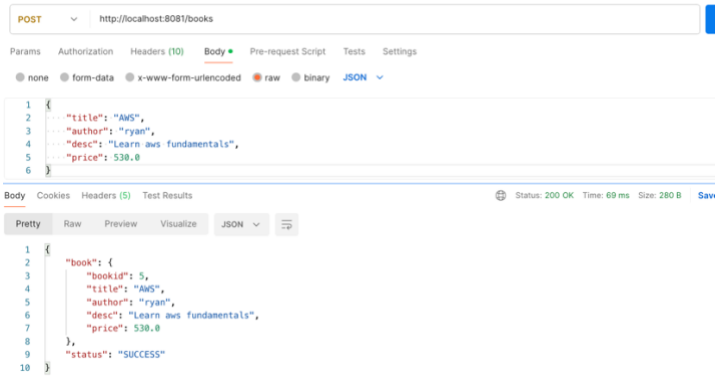
2. Looks like spring was not able to map the data coming in the request to java class. We need to add @RequestBody in the method parameter for spring to know to do the mapping of JSON data to java class.

```
@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);
```

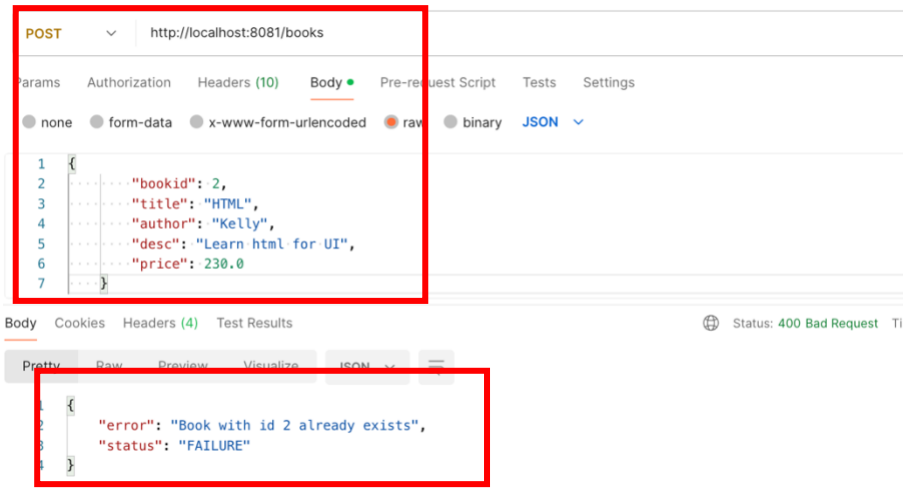
```
        map.put("book",bookService.addNewBook(book) );
        return ResponseEntity.ok(map);
    }
    catch (RuntimeException e){
        map.put(AppConstants.STATUS, Status.FAILURE);
        map.put("error",e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}
```

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

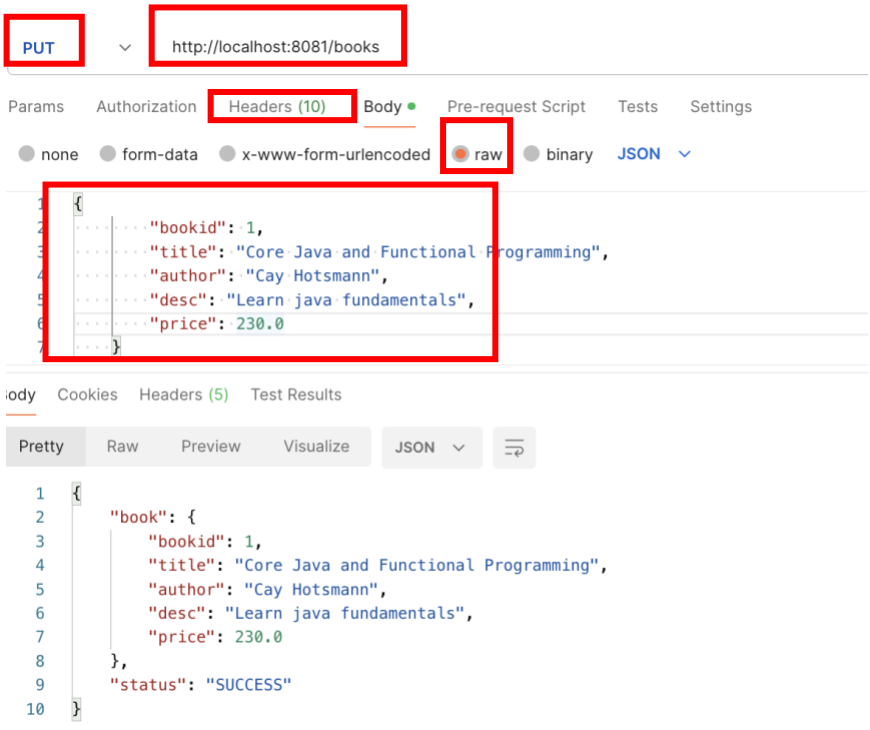
1. To update a book add below method:

```
@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);
    }
}
```

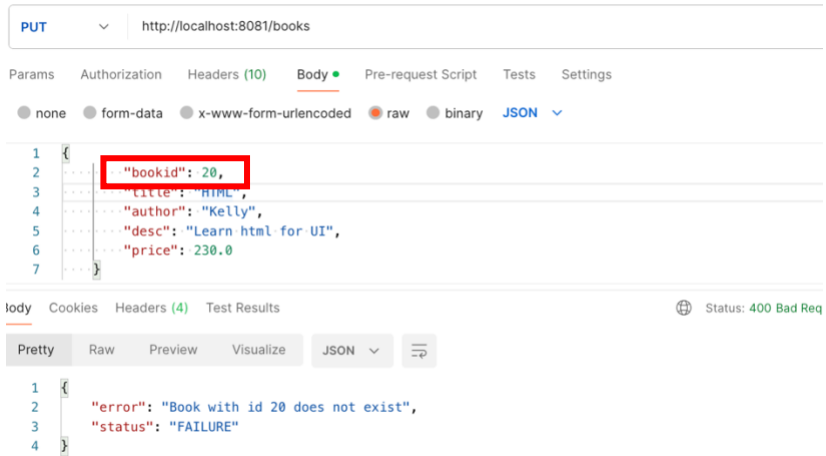
MAKE SURE TO RESTART THE SERVER

ALSO PLEASE UPDATE HEADER :

Content-Type: application/json



Try to update a book that does not exist:



Step 9: @DeleteMapping

1. To delete a book add below method:

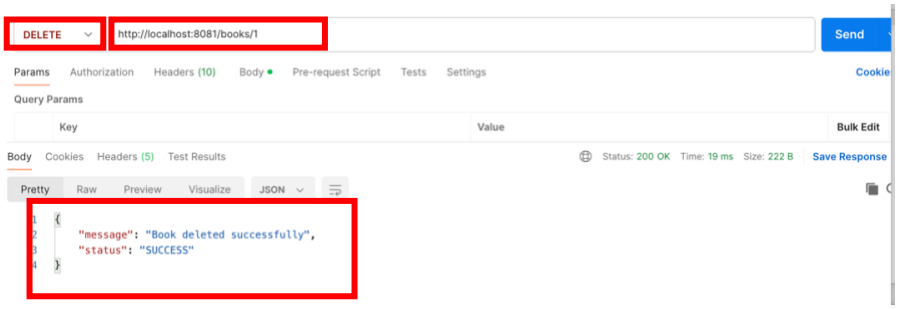
```
@DeleteMapping ("/{id}")
public ResponseEntity<Object> deleteBook(@PathVariable  int id){
   Map<String, Object> map = new HashMap<>();
   try {
      map.put(AppConstants.STATUS, Status.SUCCESS);
      if(bookService.deleteBook(id)) {
         map.put("message", "Book deleted successfully");
         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 exists with the id: