* **Spring Boot WebFlux | Global Exception Handling | Functional Endpoints | JavaTechie**

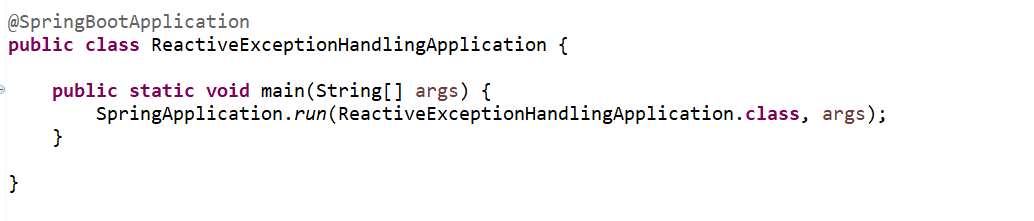
**------------------------------------------------------------------------------------------------**

In this tutorial we will understand how to handle exceptions for functional endpoints in spring webflux. As we all know, error and exceptional link is one of the key aspects and also, we can say this is one of the ways we make sure that we handle our failure consistently. So, let's quickly demonstrate global exceptional link mechanism for reactive functional endpoints.

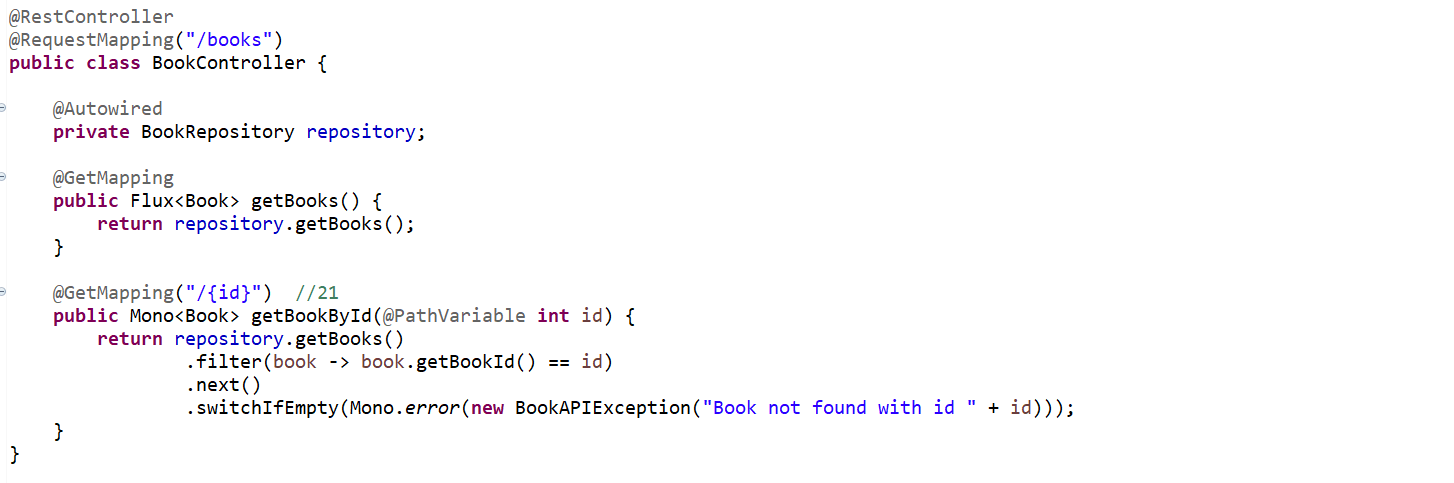
**App Name**- **reactive-exception-handling**

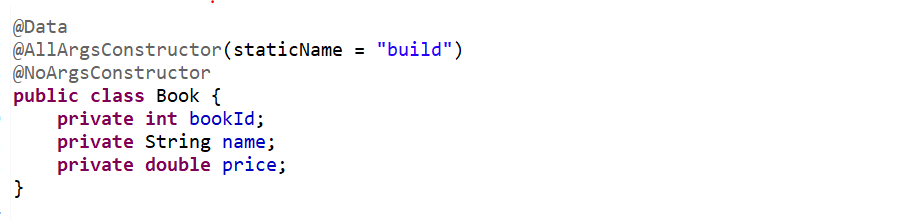
Already I created one small application using Spring Reactive. So, I just define the endpoints using the traditional approach as well as the functional endpoint. So, I'll show you both the example. Let's go to this book controller. So, this is how we can define in traditional approach. Now if we'll check in the controller, here I have two methods. We'll return all the book objects from the DB. DB means the repo. And there is another method. We'll return a single book object.

Now we know how we can handle the exception in traditional approach. So, if you go and check in this book controller, let's consider this particular example where I'm just fetching the book object by book ID. If that book ID is not found, then I'm throwing this exception. As you know in repo, we are just returning 20 book objects with 20 ID. Let's say user is giving 21. Then I'm just throwing this book API exception, book not found with ID this. Now since I'm throwing this exception from the controller, that is how I just created one handler class, which is the global exception handler you can consider. There I'm just annotated at the rate rest controller advice. Then an exception handler annotation. I'm just telling this particular spring. If the controller will throw this particular exception, then execute this method where I'm just creating a map and I'm just setting two field error message and status and that is what I'm returning. So, when book API exception occurred in your controller, immediately user will get these two response by seeing this response. User will understand, this is what the error message and this is what the status. Now this is what we can do in the traditional approach.

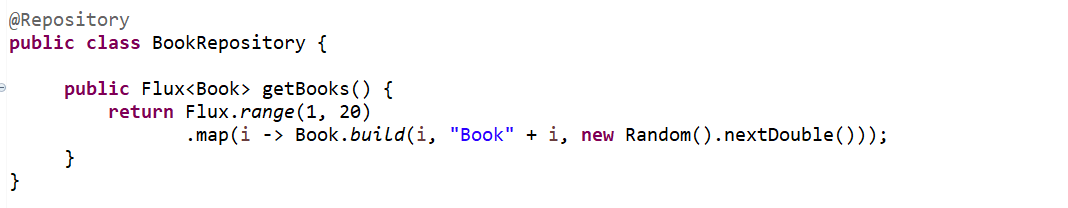


------------------------------------------------------------------------------------------------------------------------------------------

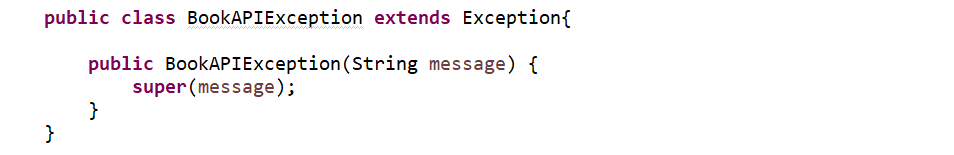




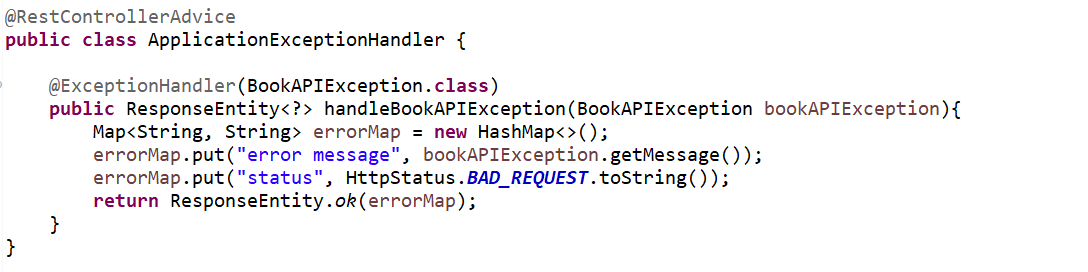
------------------------------------------------------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------------------------------------------------------

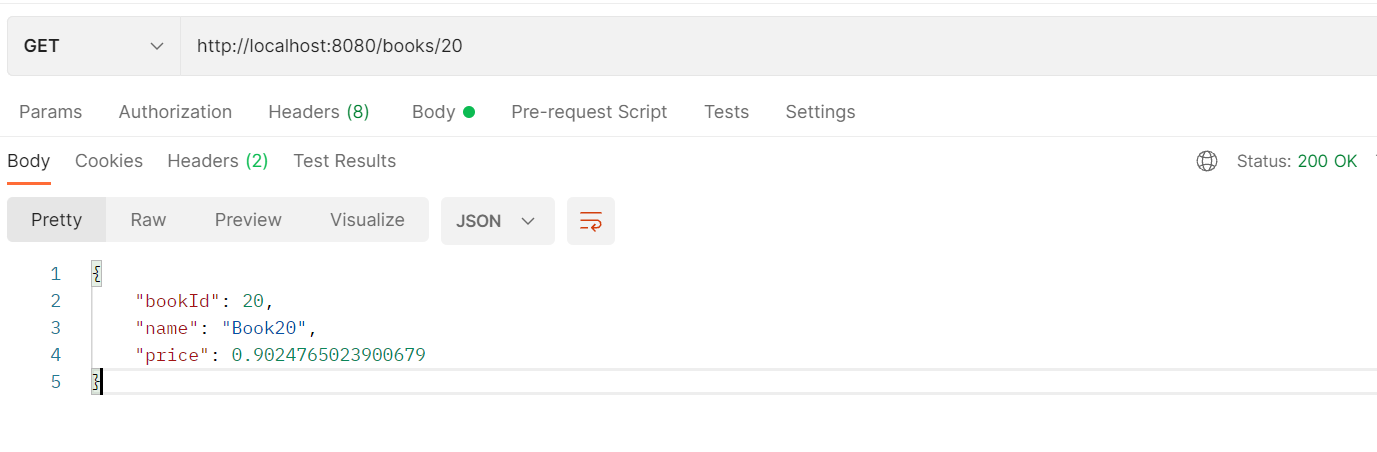


------------------------------------------------------------------------------------------------------------------------------------------

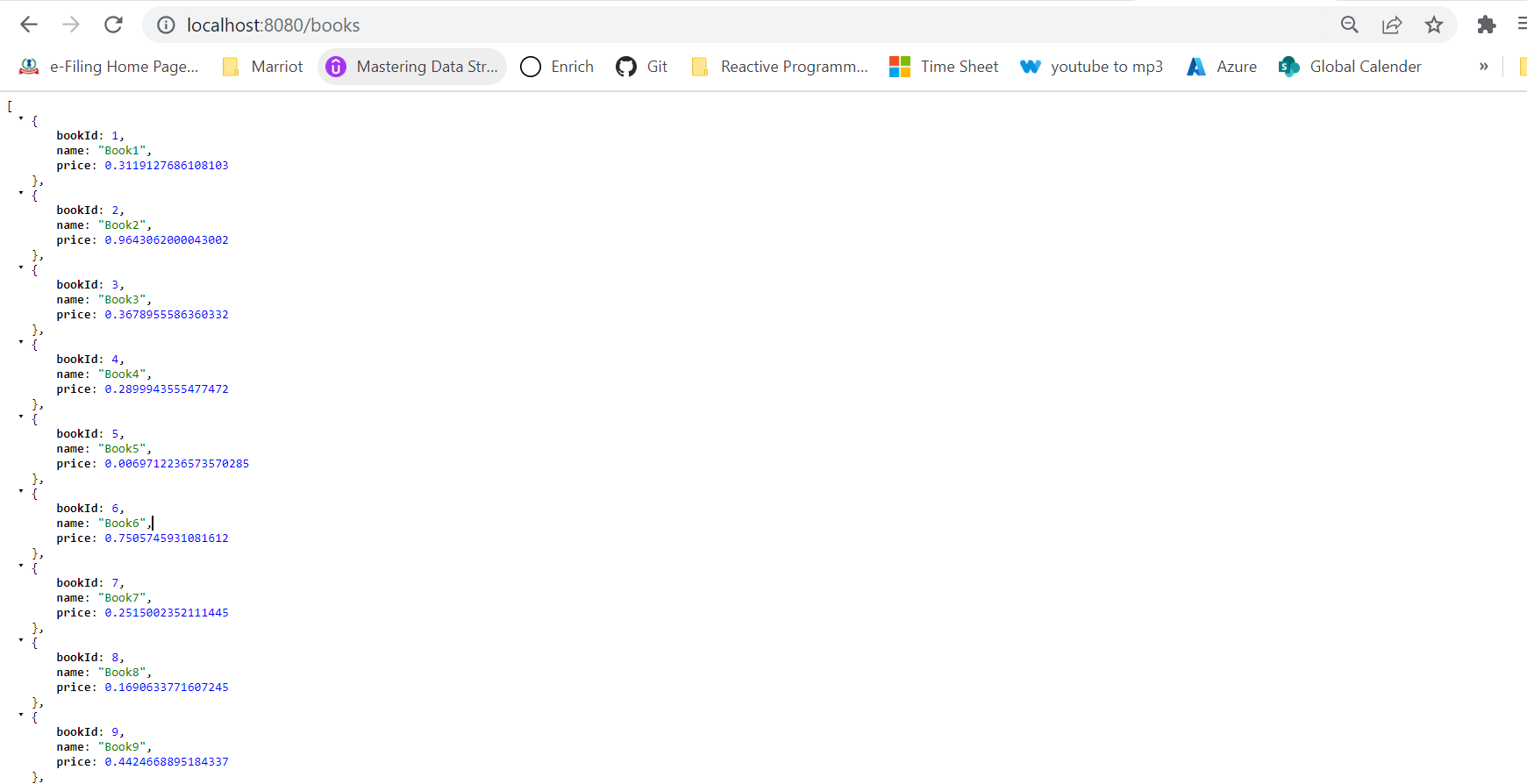


----------------------------------------------------------------------------------------------------------------------------------------

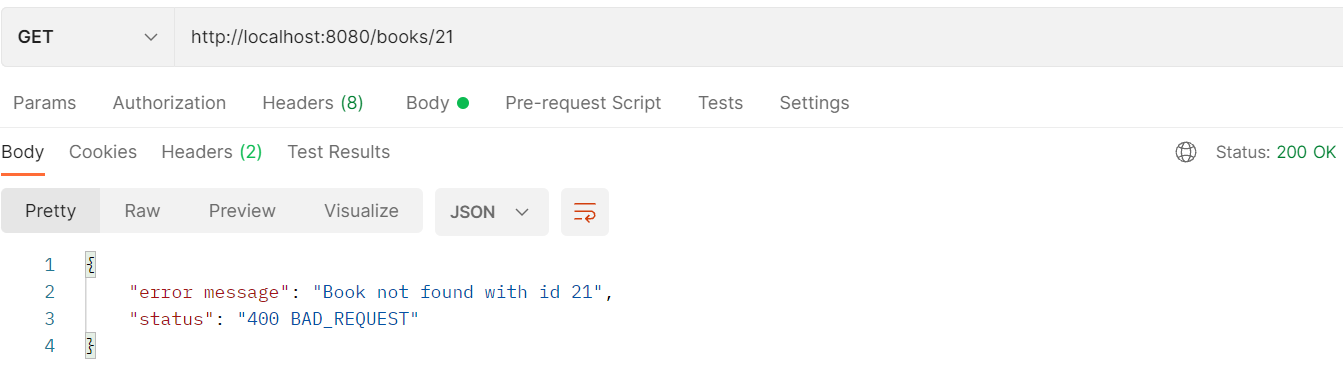
**GET** <http://localhost:8080/books/20>’



<http://localhost:8080/books>



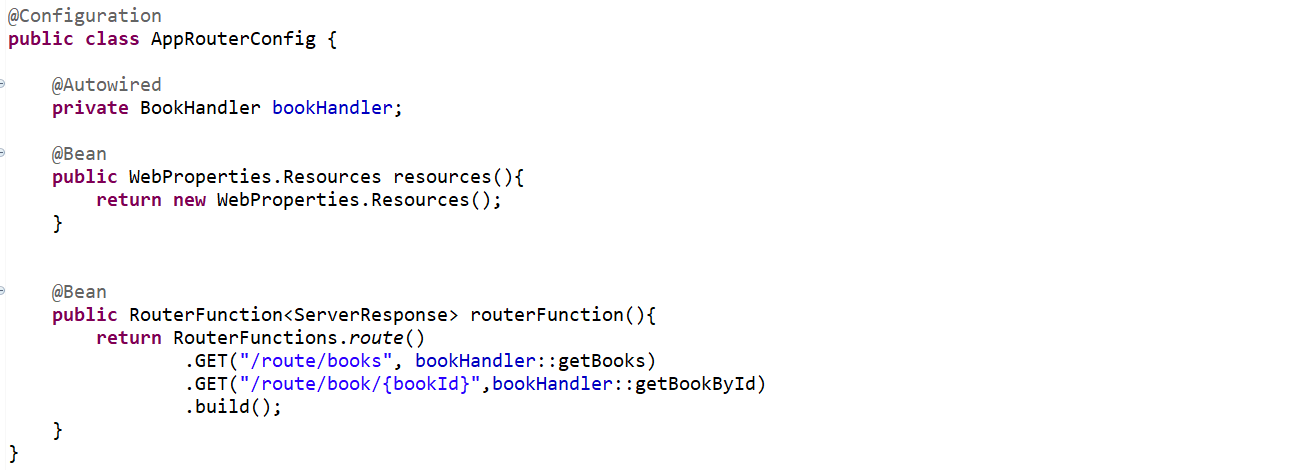
**GET** <http://localhost:8080/books/21>



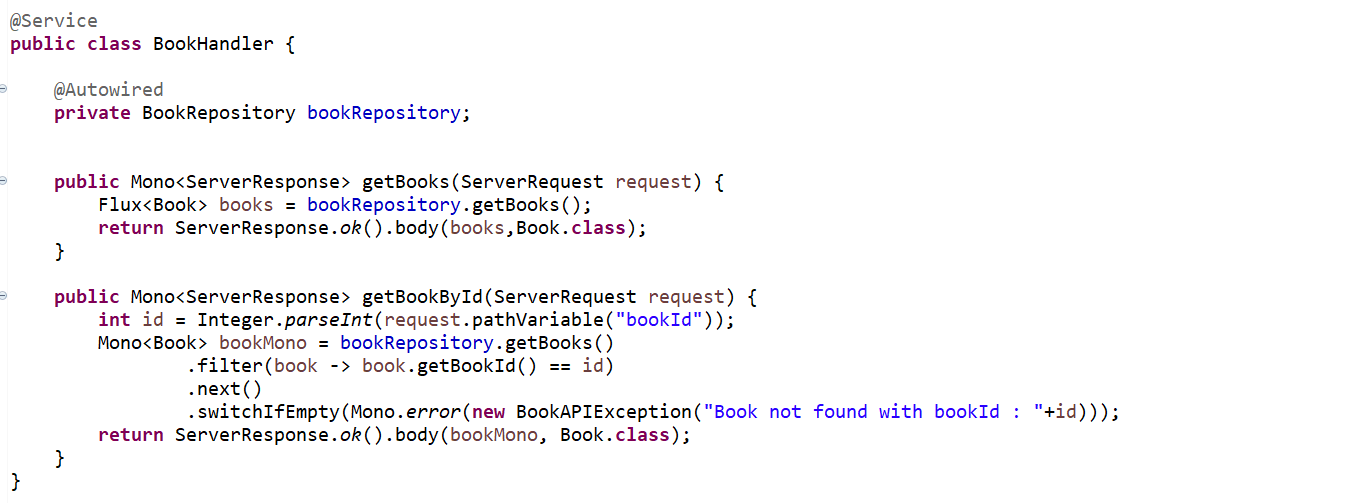
Now let's move to the functional endpoint. So, if you'll open this book handler class. If you'll go to this app router config. This is how I define my functional endpoint. If the user is giving this particular URL, slash route slash books, then I am telling this Spring Reactive, please redirect that particular request to this book handler class and this get books method. That is how I named it as a handler. You can give any name. You can give service or anything. It's just a naming convention. If you're using functional endpoint, then your service layer, it's better to give the name with handler. Now similarly, if we'll go and check, if user will give the second URL to get a book by its ID, then I'm telling just call this particular method. So, this is how we can define the functional endpoint.

But if you'll check in the functional endpoint, we don't have any HTTP method logic here. We are just defining this is what the get method and just call this particular handler, but we don't have any option to handle the exception or to delegate the exception from my endpoint to the next global handler? So how we can do that, that is what we just want to cover in this particular tutorial.

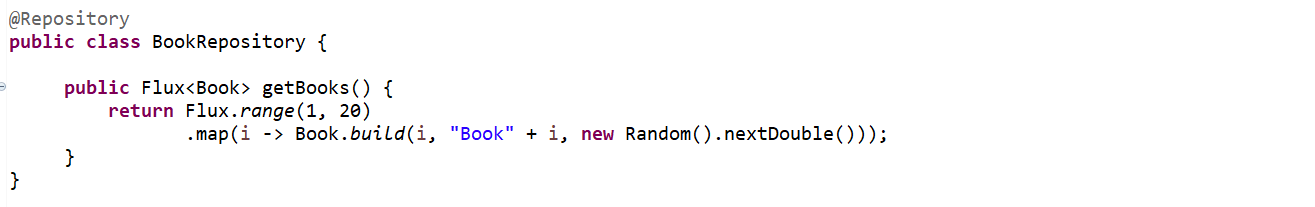
Now, let's see how it behaves in functional endpoint. So let me verify the endpoint URL. Go to the app route config slash route books will return the list of book object or flux of book object, then route slash book, then book ID. Then send it. We'll get all the 20-book object. Now I'll just search with book ID. Let's say 20. I'll get the result. Now if I'll search with 21, we don't have any logic to handle the exception in functional endpoint so far. Now let's see how we are getting the exception or how we can visualize it. Then I will guide you the next step, how we can handle it. Just hit with 21. This ID is not present. We are not getting anything right. We are just getting 200. This route config, then go to book handler. Now here we're just returning the next. Here I can, I need to validate actually switch if empty. Then I just need to return the error. Mono dot error. I'll just return the exception new book API exception with a message book not found with book ID. Just pass the ID. I'll just add a colon here. Earlier we don't have any validation logic. Now we just added. If that book is not found, then throw this exception. Now we'll see how it will display on our console.



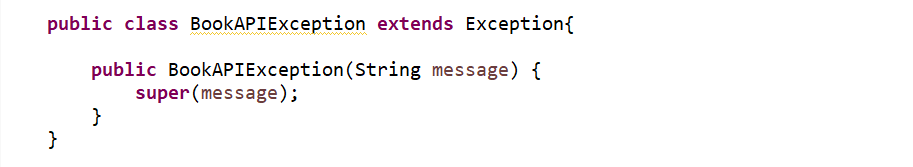
------------------------------------------------------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------------------------------------------------------

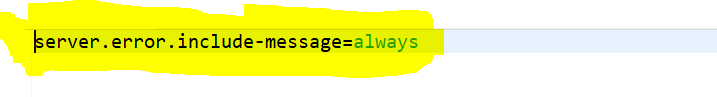


------------------------------------------------------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------------------------------------------------------

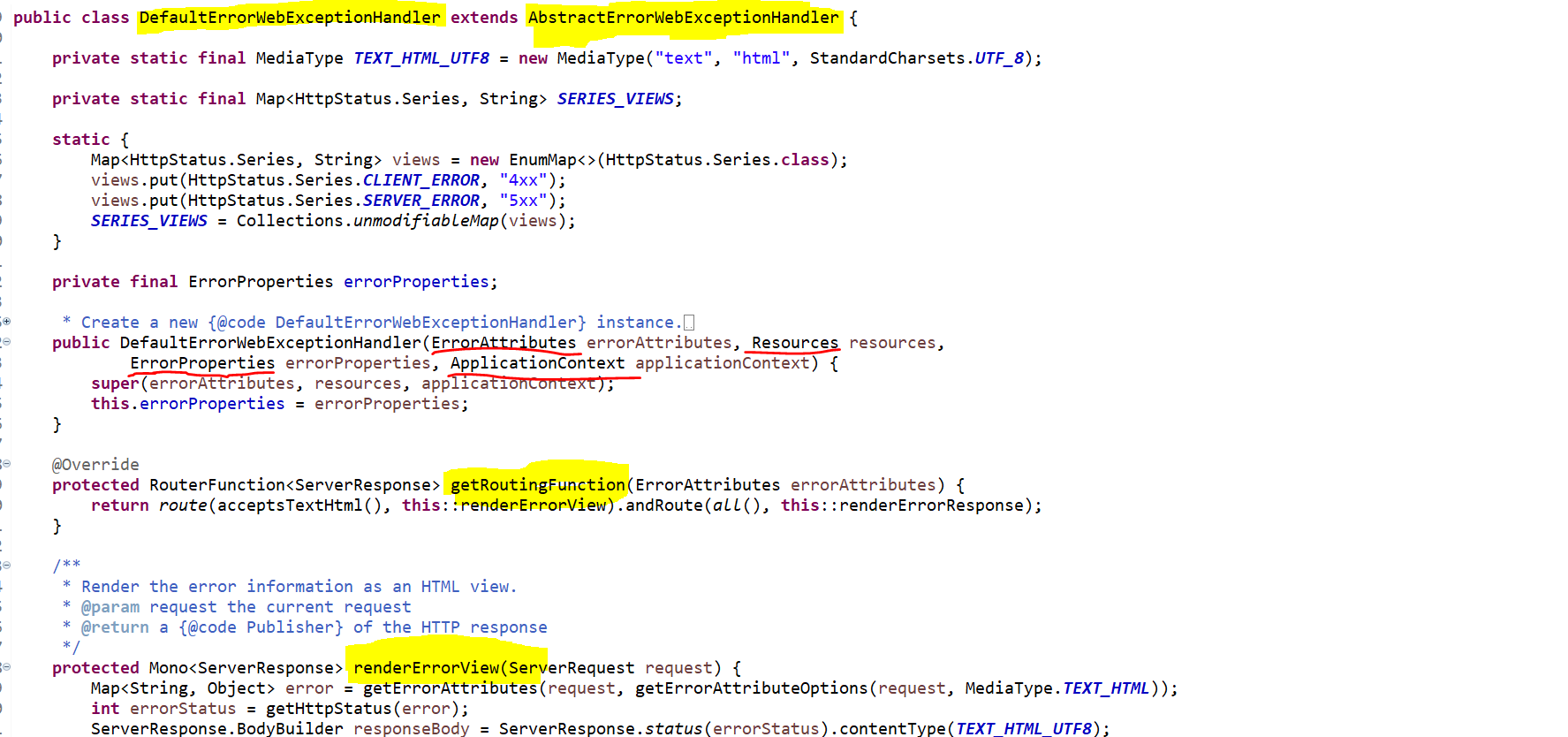
So, application started here on port 8080. Now go to the postman and let's hit the same endpoint. We are getting this error. Path is this, timestamp is this and status code is 500 error messages, internal server error and this is what something requests ID. But this is incorrect. We are expecting the status should be 400 and the error message should be book not found with the ID this. By default, message won't display in the console or in this particular response. So how we can override that? You need to add one key and value in your application dot properties file or application dot YML file. Just go to the one of them and just add this particular key and value. Always I want to display the message. The message which I'm returning or throwing from my application, I just want to display that particular message. So that is the reason you need to add this particular key and value. Now let me restart this.



Now you can see the message here, Book not found with book ID. This is what the particular message we are throwing from our handler. You can see here status error message request ID timestamp. All those information I don't want to return to the end user. I just want to return the message or the path what user is trying or any status code. These are the field enough to display as a response so that end user can easily understand.

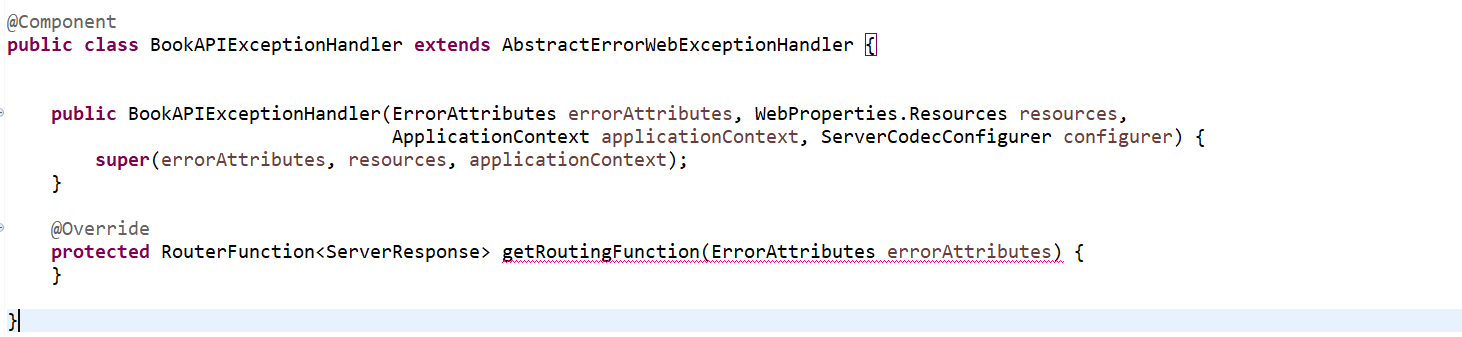
I need to modify my request. Now the question here, how we can override this to override this first we should know from where spring getting those information and displaying as a console. So, to override that first we should know how it works internally.

Spring reactive internally use one class something called **DefaultErrorWebExceptionHandler**.



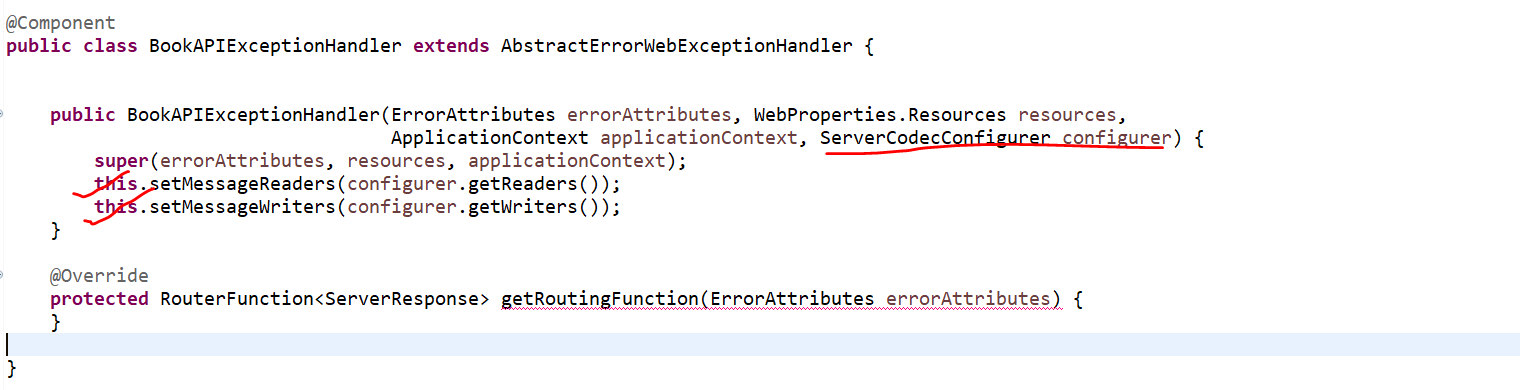
if we'll go to this class, this class extends from **AbstractErrorWebExceptionHandler**. So, to override that we need to create a class then we need to extend from this particular **AbstractErrorWebExceptionHandler**. Then we need to override this render method, **renderErrorView** and **getRouting** function.

Now what we can do to override this default behavior. Let's create a class and extends it from **AbstractErrorWebExceptionHandler**. Then we'll modify as per our need. I'll just create a package **exception.handler**. I'll just create a class **BookAPIExceptionHandler** and then I just need to extend that from the class **AbstractErrorWebExceptionHandler**. Now it will force you to override the method.



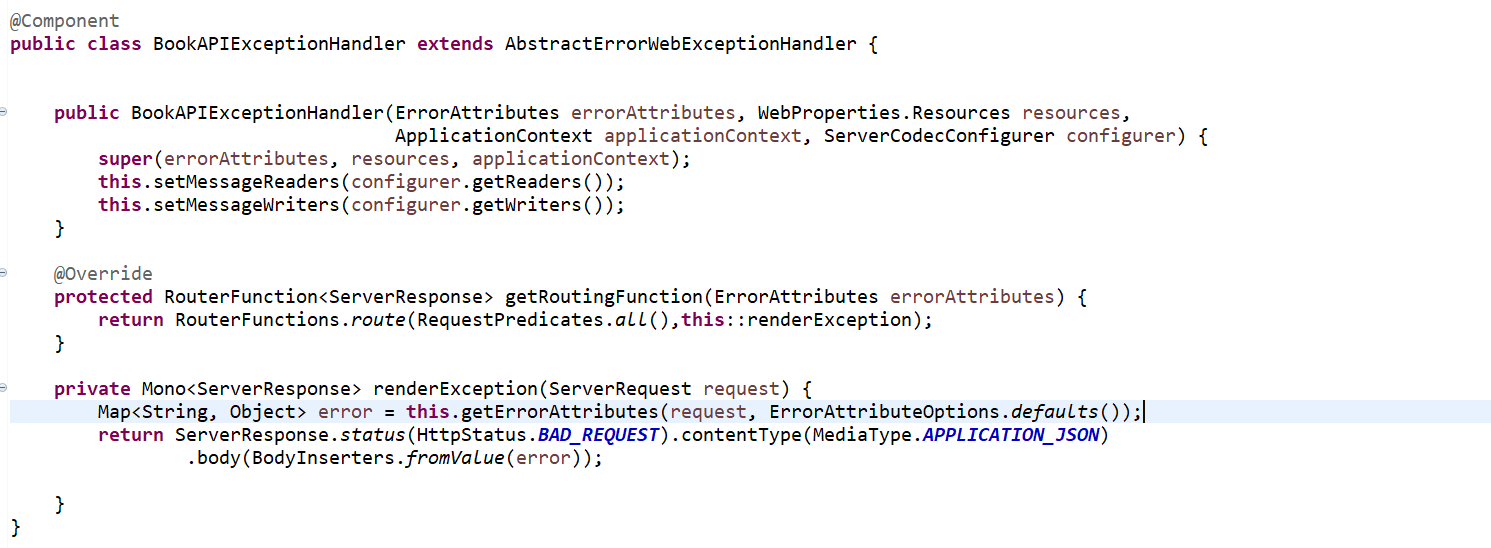
Now you can see here it will force you to override the method. Just implement that get routing function. So just add this super constructor. Now here we need to add another class to just define our reader and writer so that we can write our own exception handling mechanism and we can read it because we don't run this default message. We can customize using our own reader and writer. So there is a class, something called server, something like code configurer. Yeah. Server codec configurer. I'll just name it configurer.

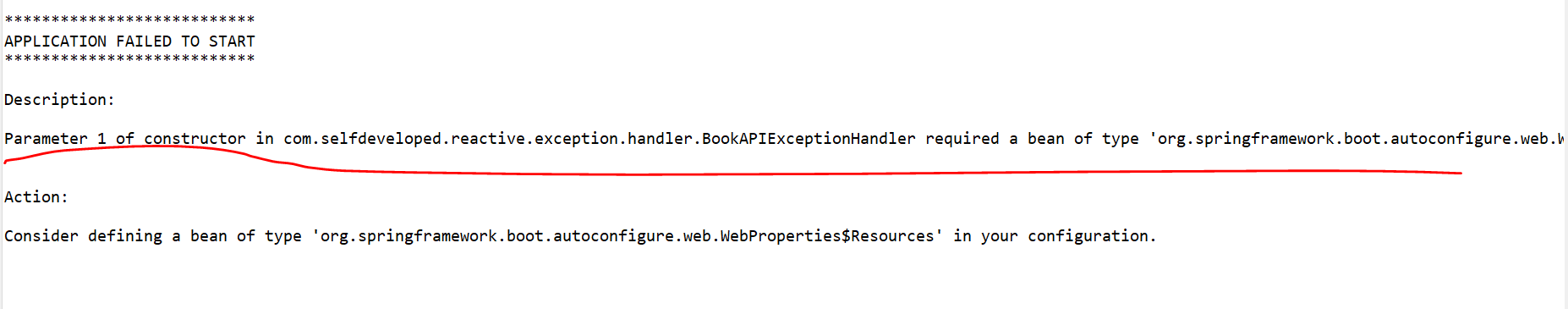
Now here, what do you need to do? Just this dot set message reader, get it from configurer dot get readers. Similarly, you can do this dot set message writers, get it from get writers from configure.



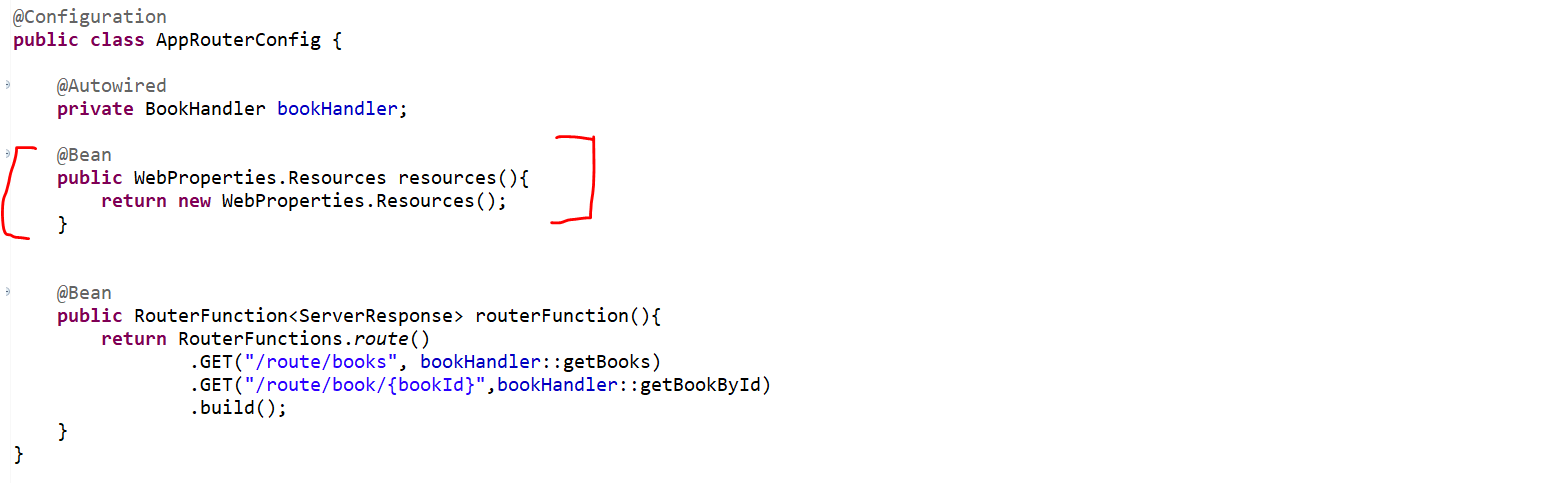
Now we have defined our own reader and writer. Now in this get routing function, which is the overridden method from the super class, which is **AbstractErrorWithExceptionHandler**. There we just need to tell for what all ends point you want to implement this custom error handling mechanism. So, what we can do here, we can define router functions dot route. If you observe route is overloaded method, you can define this, then you can define the method type for what exact method you want. Let me do that, but we'll not do this particular pattern because I just want to implement this custom exception handling mechanism for all the end point. You can define the URL here and you can define the method to render the exception message. So, I'll show you how you can do that, but now we'll do for all. So how we can do that? There is something called request predicate or something like that. Predicates dot all. In fact, here also you have message to define for each HTTP method type. But for this particular example, I just want to implement for all the method. Because there is nothing makes sense to make it for all because we have only one method in our router config.

Now here, what do you need to do? Just define a private method. I'll define something like render exception. I'll just create this private method. Now here this should be one of server response. Now here first we need to get all the error attributes. So, whatever the key value we are seeing, those are the error attributes. First, we need to get that from the default implementation. Then only we can customize it.



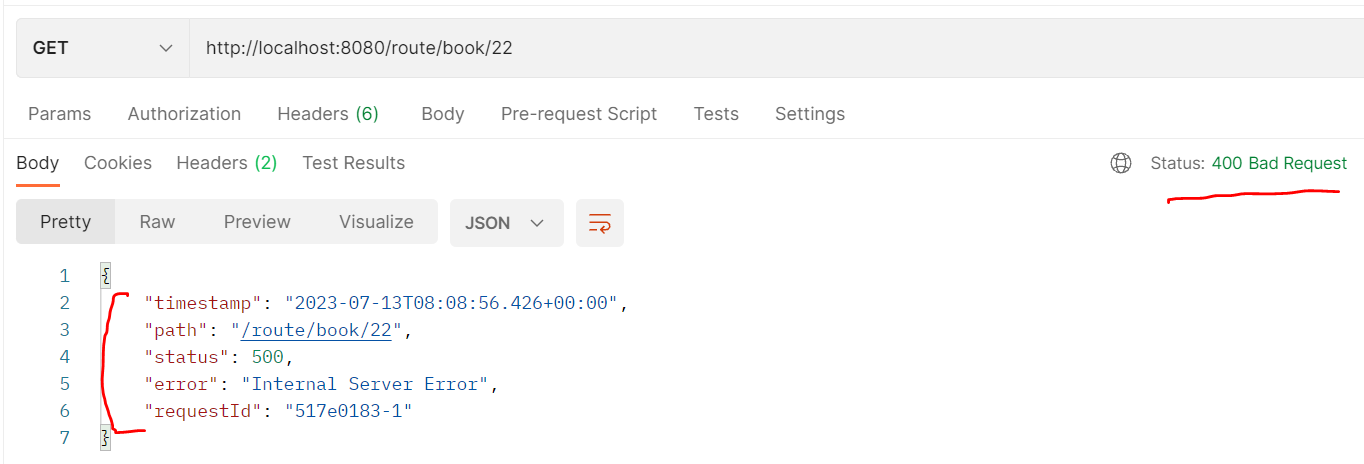


---------------------------------------------------------------------------------------------------------------------------------------



Now let’s try and hit the endpoint.

**GET** <http://localhost:8080/route/book/22>

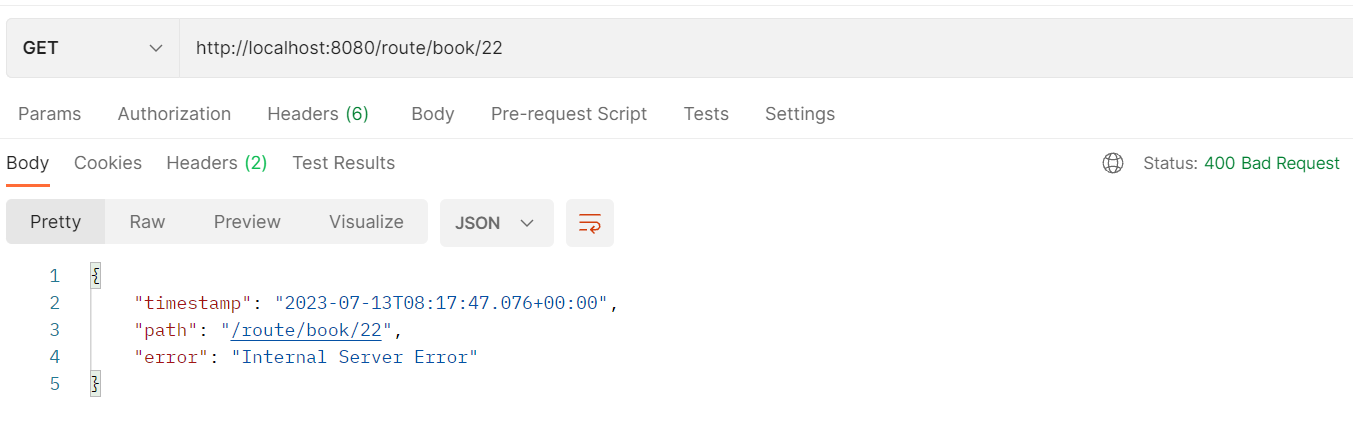


Now you can see something like path, timestamp, error, request ID. Again, this is not something relevant. Now, you might have a question. Why we are getting this value? So, if you'll go and check in your implementation, we are not doing anything. Whatever the gate error attributes we are getting, same attribute or same error attributes we are just setting as a body. That is what we can see here. If you don't want anything from this map, you can simply do error.remove. You can take the key from this map, which you don't want to display. Let's say request ID. But if you observe, we are getting the correct status code, which is 400 bad requests. If you don't want anything to be displayed as part of the default error attributes, you can directly remove because this is the map from this map. We can remove the key. If you remove the key, that particular object will be not available, and you won't see those two fields as part of your response. But that is not our solution.



------------------------------------------------------------------------------------------------------------------------------------------

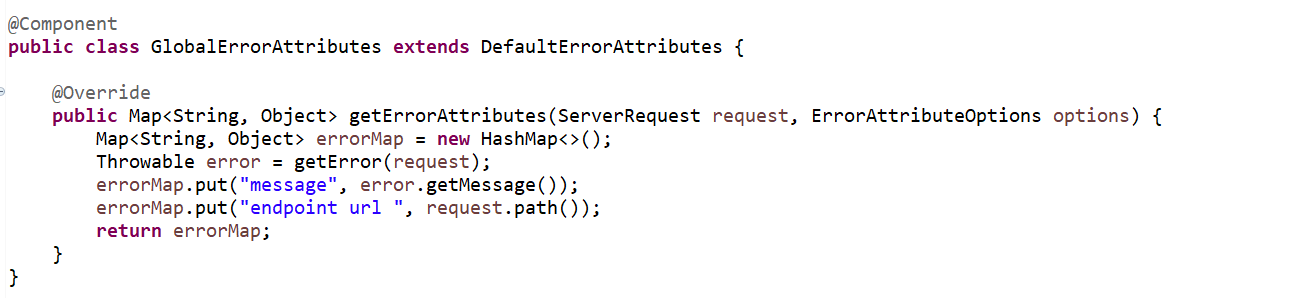
**GET**  <http://localhost:8080/route/book/22>

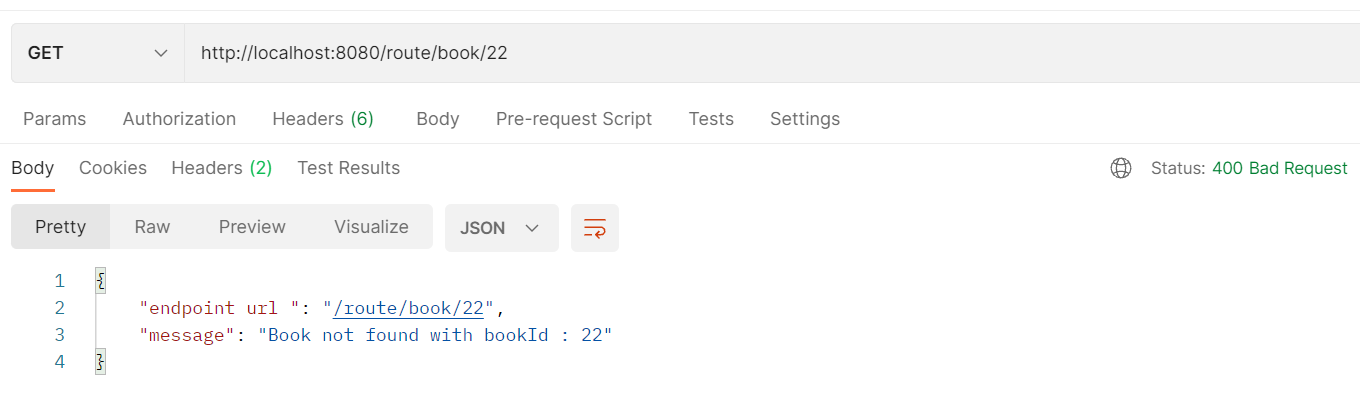


------------------------------------------------------------------------------------------------------------------------------------------

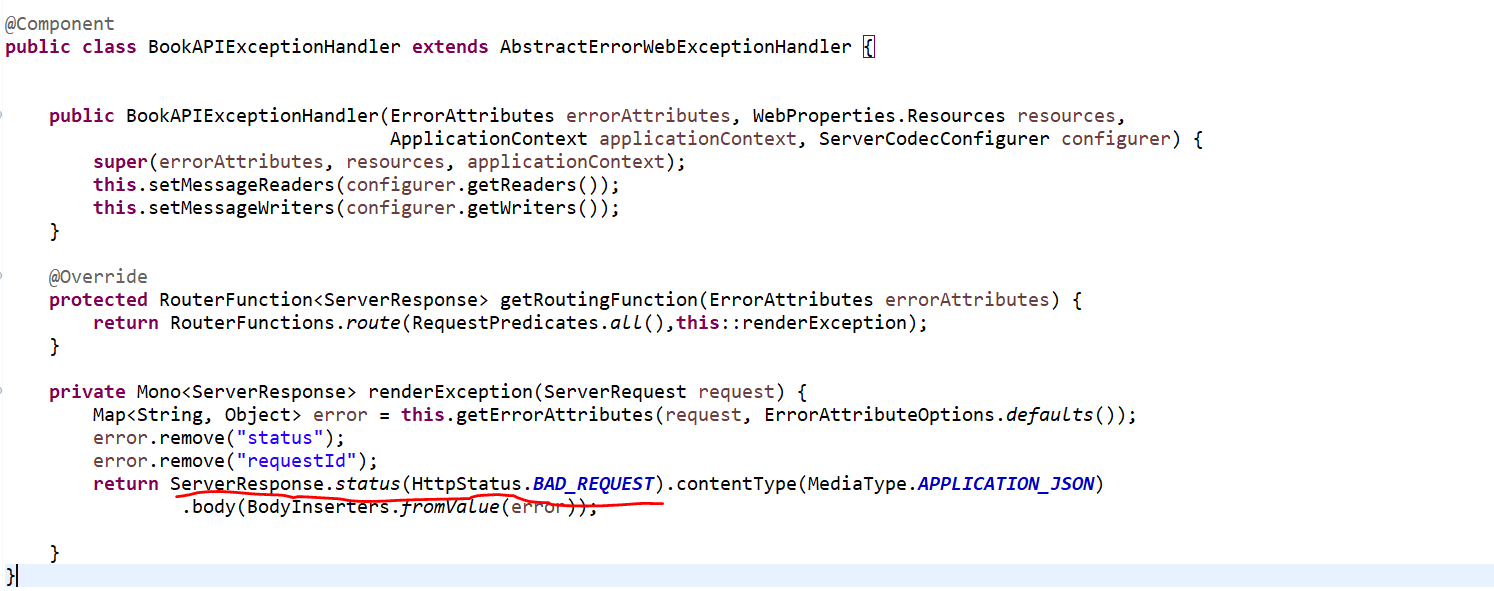
We can see only timestamp path and error. This is the internal server error message is displaying that is expected because we are just overriding the default behavior. That is what the default behavior. this.**getErrorAttributes**. Now the main thing you need to understand how I can define my own error attributes.

So, to do that, what you can do, I'll just create another class here. let give the name **GlobalErrorAttributes**. Then you need to extend that class from **DefaultErrorAttributes**. There are some method need to override **getErrorAttributes**.



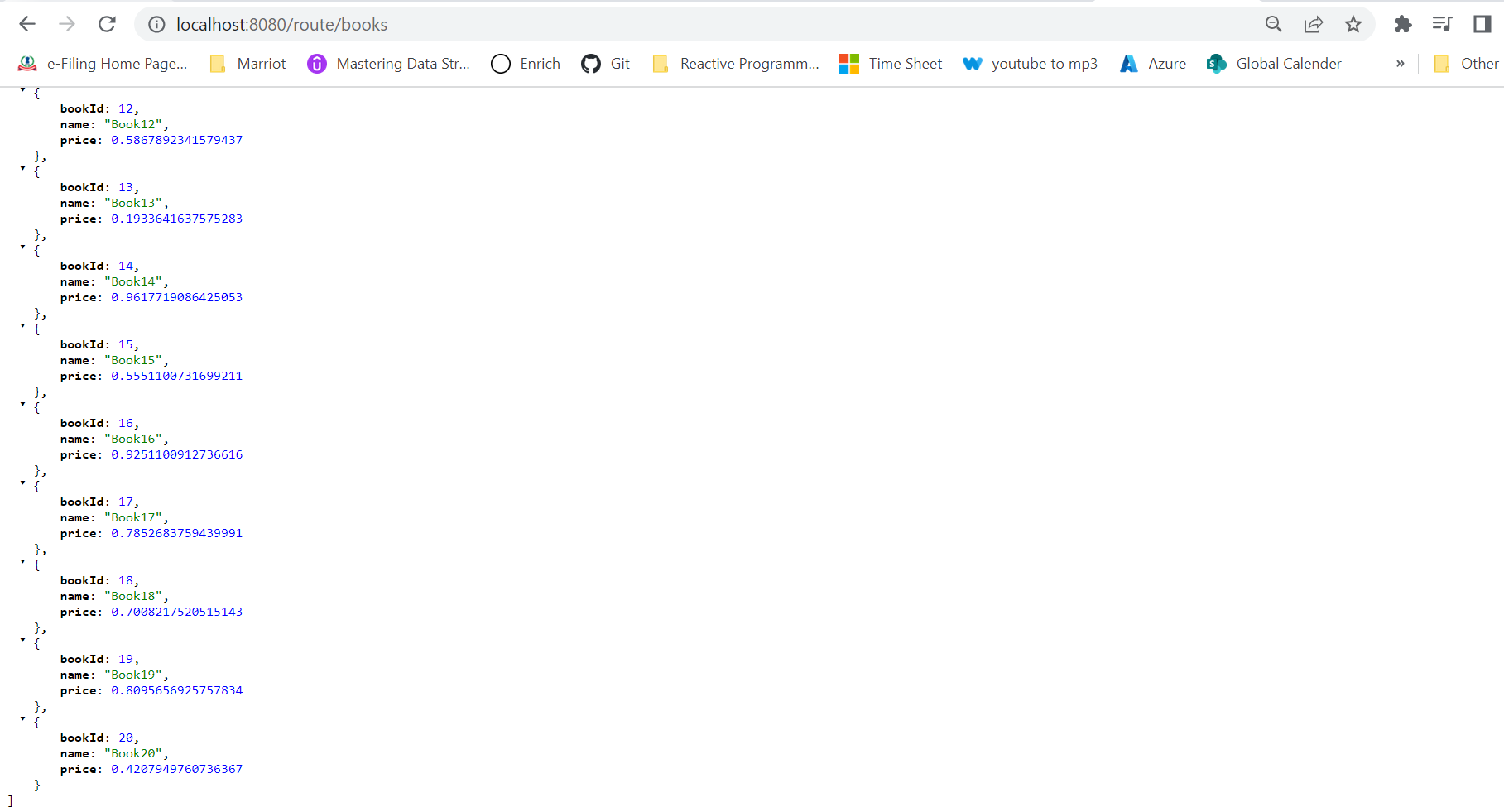


If you observe here, message and endpoint URL. That is what we are getting here. Also, we are getting the status code as 400. Since this is the global approach, rather than hard code, the status code, you can just set it dynamically. Since this is the demo app and I have only one endpoint, functional endpoint to show you, I just keep a exception and I forcefully set the status code as 400. So, if you remember, we created a class. we just keep the status code as bad request. It is up to you. You can create n number of custom exceptions to handle different status code, or you can make this field dynamic.

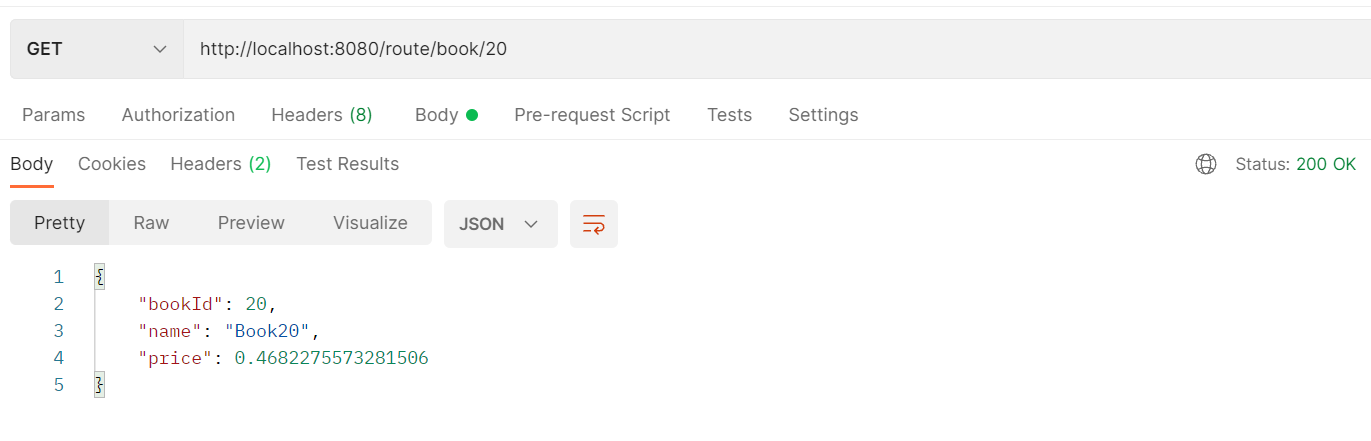


So, this is how you can handle the error or exception at global level using the Spring Reactive Functional Endpoint approach. There is not straight forward approach. You need to create two class **BookAPIExceptionHandler** extend from this and for customizing the error attributes, you need to create another class that is called **GlobalErrorAttributes**, or you can give any name and that need to be extends from **DefaultErrorAttributes**. But make sure to import the correct package statement. This should be from reactive not from servlet.

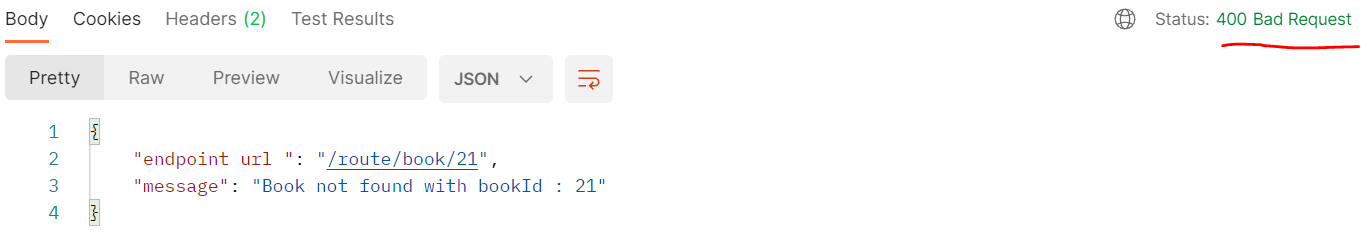
**GET** <http://localhost:8080/route/books>



<http://localhost:8080/route/book/20>



<http://localhost:8080/route/book/21>



For more Details Please Refer ->

<https://www.baeldung.com/spring-webflux-errors>

**Handling** Errors **in Spring WebFlux**