Day14: Exception handling and input data validation

Handling the exceptions in spring boot:

By default Spring boot provides a "**BasicErrorController**" controller for /error mapping, that handles all the exceptions that occurred in our request handler methods.

This controller produces a JSON response with details of the error, the Http status and the exception messages.

Example :

```
@GetMapping("/getStudent/{roll}") public Student getStudentDetails(@PathVariable Integer
roll) { if(roll < 100) throw new IllegalArgumentException("Roll number should be greater
than 100"); return new Student(roll, name, marks); }
```

Here if we give the request as :

http://localhost:8088/getStudent/10 then this handler method will throw an exception and in spring Boot will produce a default error response in the form JSON object as follows:

```
{ "timestamp": "2022-04-28T04:30:23.360+00:00", "status": 500, "error": "Internal Server
Error", "trace": ".....", "message": "Roll number should be greater than 100", "path":
"/getStudent/10" }
```

Here the default response status code is 500.

In order to handle the exception and change the default status code 500 to error specific status code we need to define another method inside the controller class and annotate this method with **@ExceptionHandler** annotation as follows:

```
@ExceptionHandler(IllegalArgumentException.class) public ResponseEntity<String>
myExpHandler(IllegalArgumentException ie) { System.out.println("inside myHandler
method..."); return new ResponseEntity<String>(ie.getMessage(),HttpStatus.BAD_REQUEST);
```

Now if inside any request handler method throws IllegalArgumentException then this handler method will handle that exception and it will generate error specific status code as a response.

Note: We should not handle the exceptions of the request handler method inside try and catch block, instead we should let the Spring boot handle that exception and generate a error repsonse in the form of JSON object with the appropriate error status code. so that client can use that status code and display the appropriate result.

- The above Exception handler method will be applicable for the same controller class only in which it is defined.
- If we want to apply the exceptionhandler method for multiple controllers of application, we should define that method inside a separate class and annotate that class using **@ControllAdvice** annotation.
- That class annotated with **@ControllerAdvice** will be called as Global exception handler.

Example: create a separate class called GlobalExceptionHandler.java (class name could be anything) inside a package **com.masai.exception**

```
package com.masai.exception; import java.io.IOException; import
javax.servlet.http.HttpServletResponse; import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice; import
org.springframework.web.bind.annotation.ExceptionHandler; @ControllerAdvice public class
GlobalExceptionHandler { @ExceptionHandler(IllegalArgumentException.class) public
ResponseEntity<String> myExpHandler(IllegalArgumentException ie) {
System.out.println("inside myHandler method..."); return new ResponseEntity<String>
(ie.getMessage(),HttpStatus.BAD_REQUEST); } }
```

Note:- it is always recommended to through our own domain-specific custom exception class object instead of throwing, predefined Layer specific/technology specific exception class objects to achieve loose coupling among different layers.

- Create a user-defined runtime exception called: **InvalidRollNumberException**

```
InvalidRollException.java:- -------------------------------- public class
InvalidRollNumberException extends RuntimeException{ public InvalidRollNumberException()
{ } public InvalidRollNumberException(String message) { super(message); } }
```

Now throw **InvalidRollNumberExcception** from our request handling method

Exmaple:

```
@GetMapping("/getStudent/{roll}") public Student getStudentDetails(@PathVariable("roll")
Integer roll) { if(roll < 100) throw new InvalidRollNumberException("Roll number should
be greater than 100"); int result=roll/0; // another exception will be thrown return new
Student(roll, "Rahul", 500); }
```

Modify the GlobalExceptionHandler class as follows:

```
@ControllerAdvice public class GlobalExceptionHandler { //to handle specific
InvalidRollNumberException @ExceptionHandler(InvalidRollNumberException.class) public
ResponseEntity<String> myIllegalHandler(InvalidRollNumberException ie) {
System.out.println("inside myHandler method..."); return new ResponseEntity<String>
(ie.getMessage(),HttpStatus.BAD_REQUEST); } //to handle any other type of Exception
@ExceptionHandler(Exception.class) public void myExceptionHandler(Exception e) {
System.out.println("inside myHandler method..."); return new ResponseEntity<String>
(e.getMessage(),HttpStatus.BAD_REQUEST); } }
```

Customizing Error Response in user defined Format:

To customize the error response in user-defined format we need to create a Java bean class with any name:

Example:

```
//MyErrorDetails.java:- //------------------------- package com.masai.exception; import
java.time.LocalDateTime; public class MyErrorDetails { private LocalDateTime timestamp;
private String message; private String details; public MyErrorDetails() { // TODO Auto-
generated constructor stub } public MyErrorDetails(LocalDateTime timestamp, String
message, String details) { super(); this.timestamp = timestamp; this.message = message;
this.details = details; } public LocalDateTime getTimestamp() { return timestamp; }
public void setTimestamp(LocalDateTime timestamp) { this.timestamp = timestamp; } public
String getMessage() { return message; } public void setMessage(String message) {
this.message = message; } public String getDetails() { return details; } public void
setDetails(String details) { this.details = details; } }
```

Now inside the GlobalExceptionHandler class we can modify the error handler method as follows:

```
package com.masai.exception; import java.io.IOException; import
javax.servlet.http.HttpServletResponse; import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice; import
org.springframework.web.bind.annotation.ExceptionHandler; @ControllerAdvice public class
GlobalExceptionHandler { //to handle specific InvalidRollNumberException
@ExceptionHandler(InvalidRollNumberException.class) public
ResponseEntity<MyErrorDetails> myIllegalHandler(InvalidRollNumberException ie,WebRequest
req) { System.out.println("inside myHandler method..."); MyErrorDetails err=new
MyErrorDetails(LocalDateTime.now(), ie.getMessage(), req.getDescription(false));
ResponseEntity<MyErrorDetails> re=new ResponseEntity<>(err,HttpStatus.BAD_REQUEST);
return re; } //to handle generic any type of Exception
@ExceptionHandler(Exception.class) public ResponseEntity<MyErrorDetails>
myExceptionHandler(Exception e,WebRequest req) { MyErrorDetails err=new
MyErrorDetails(LocalDateTime.now(), e.getMessage(), req.getDescription(false)); return
new ResponseEntity<>(err,HttpStatus.BAD_REQUEST); } }
```

Note:- Spring boot by default does not map the custom exceptions to 404 error like Not Found exception class, if we try to call an invalid resource.(any invalid URI) for this we need to create a different exception handler as follows.

```
//to handle Not found exception @ExceptionHandler(NoHandlerFoundException.class) public
ResponseEntity<MyErrorDetails> mynotFoundHandler(NoHandlerFoundException nfe,WebRequest
req) { MyErrorDetails err=new MyErrorDetails(LocalDateTime.now(), nfe.getMessage(),
req.getDescription(false)); return new ResponseEntity<>(err,HttpStatus.BAD_REQUEST); }
```

After this we need to add following 2 entries in our application.properties file.

```
spring.mvc.throw-exception-if-no-handler-found=true spring.web.resources.add-
mappings=false
```

Request Data validation in spring boot Rest API:

We need to validate user input data (user supplied data) in the client side application using java-script, angular, react and even at server side application also.

If the user supplied data is valid then only we should call the service layer methods.

Steps to validate the request data in spring boot rest:

Step 1: add the "spring-boot-starter-validation" dependency in our project.

```
<dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-
starter-validation</artifactId> <version>2.6.7</version> </dependency>
```

- This dependency internally uses "hibernate validator api" this hibernate validator api provides lots of annotations to perform validation on request body input data.
- These annotation should be applied on the top of java bean class.

Some of the validation annotations are:

1. **@AssertTrue**

   The value of the boolean field must be true;

   Example

   ```
   @AssertTrue private boolean married;
   ```

2. **@AssertFalse**

3. **@Min**

   The value of integer type must be >= given value

   Example:

   ```
   @Min(18) private int age;
   ```

4. **@Max**

   Example:

   ```
   @Max(60) private int age;
   ```

5. **@DecimalMin and @DecimalMax**

   These annotation is for double type data:

   Example:

   ```
   @DecimalMin(12.15) private double price;
   ```

6. **@NotNull**

   The value of the field must not be null

   Example:

   ```
   @NotNull private String name;
   ```

7. **@Size**

   The length of the String must be within the given range

   Example:

   ```
   @Size(min=5,max=8) private String password;
   ```

8. **@Pattern** : it is used to validate String pattern

   Example

   ```
   @Pattern(regexp="^[A-Z][a-z]*") private String username;
   ```

9. **@Past**: the value of the Date field should be the past date

   Example:

   ```
   @Past private LocalDate dob;
   ```

10. **@Future**
11. **@Email**

    Example

    ```
    @Email private String email;
    ```

12. **@CreditCardNumber**

Step 2: add the above annotation to the bean class property, to which we want to convert our input values.

Example:

```
Student.java:- ---------------- public class Student { //@Min(100) @Min(value = 100,message = "Minimum Roll number should be 100") private Integer roll; @Size(min = 3,max = 10 ,message = "Name size must be min 3 and max 10") private String name; private Integer marks; //getters and setters }
```

Step 3: use **@Valid** annotation to activate the validation on the RequestBody (input data)

Example:

```
@PostMapping(value = "/saveStudent") public String saveStudentDetails(@Valid @RequestBody Student student) { return "Student stored ,"+student.getName(); }
```

Step 4: handle the validation exception in @ExceptionHandler method

```
@ExceptionHandler(MethodArgumentNotValidException.class) public ResponseEntity<MyErrorDetails> myMANVExceptionHandler(MethodArgumentNotValidException me) { MyErrorDetails err=new MyErrorDetails(LocalDateTime.now(),"Validation Error",me.getBindingResult().getFieldError().getDefaultMessage()); return new ResponseEntity<>(err,HttpStatus.BAD_REQUEST); }
```

Reading the validation messages from the external properties file: messages.properties file

Step1 :- create a "messages.properties" file under src/main/resources folder

```
roll.invalid=Roll number is invalid name.invalid=Name length should be min 3 and max 10
```

Step 2:- use the key of this properties file as a message inside the bean class:

Example:

```
Student.java:- ---------------- public class Student { //@Min(100) @Min(value =
100,message = "{roll.invalid}") private Integer roll; @Size(min = 3,max = 10 ,message =
"{name.invalid}") private String name; private Integer marks; }
```

Step 3: To resolve the messages from the messages.properties files we need to configure "**LocalValidatorFactoryBean**" as a Spring bean object.