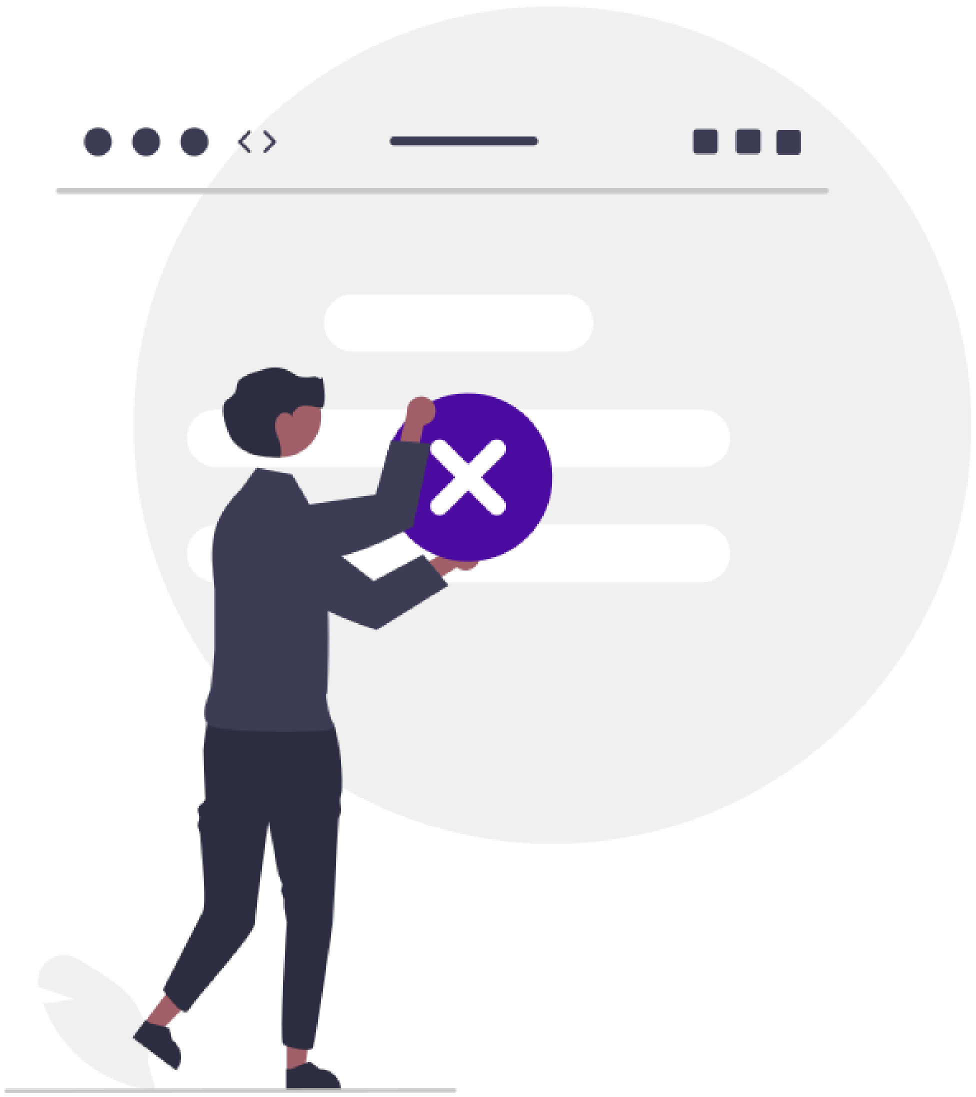
Home

Blog

About

Contact



Spring Boot – Troubleshooting

Has No Explicit Mapping for /error"

JULY 29, 2023 JAVA 0

Learn how to troubleshoot and resolve the common Spring Boot error, "This application has no explicit mapping for /error." Understand the root causes, multiple approaches to handle the issue, and best practices for global exception handling. Find detailed explanations, code examples, and important considerations to build robust and user-friendly applications.

Table of Contents

challenges, especially when it comes to error handling. One common issue that beginners face is the error message "This application has no explicit mapping for /error." In this comprehensive guide, we'll explore the reasons behind this error, various approaches to troubleshooting it, and best practices for handling exceptions in Spring Boot. Whether you're new to Spring Boot or an experienced developer, this article will equip you with the knowledge to handle this error effectively.

Understanding "This Application Has No Explicit Mapping for /error"

When you encounter the error message "This Application Has No Explicit Mapping for /error" in your Spring Boot application, it means that there is no defined mechanism to handle unhandled exceptions or errors. In other words, when an unexpected exception occurs, the application doesn't know how to handle it, resulting in the "/error" message.

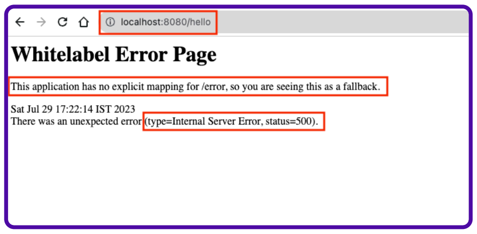
Another common reason for this error message is that Spring Boot is unable to find a handler for a specific URL or endpoint that has been requested.

Causes of the "/error" Message

1. **Unhandled Exceptions:** If your application encounters an unhandled exception, Spring Boot will attempt to handle it using its default error handling mechanism. However, if there is no explicit mapping for handling such exceptions, the "/error" message is triggered.
2. **Misconfigured Routes:** Improper configuration of routes or endpoints can also lead to this error. If the requested URL doesn't

In the below images, you will notice that the Whitelabel Error Page is displayed in two kinds of scenarios:

1. **Unhandled Exception:** In this case, a 500 Internal Server Error was thrown when an unexpected error was encountered within the application but no custom exception handler was present to handle that error.
2. **No Request Handler:** In this case, a 404 Not Found Error was thrown when a request was received with an invalid URL that does not exist within the application.

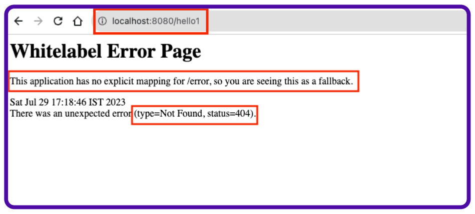


localhost:8080/hello

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Jul 29 17:22:14 IST 2023
There was an unexpected error (type=Internal Server Error, status=500).



localhost:8080/hello1

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Jul 29 17:18:46 IST 2023
There was an unexpected error (type=Not Found, status=404).

Resolving the Error

To address the “This Application Has No Explicit Mapping for /error” error, there are multiple approaches. Let’s go through each of them in detail.

In Spring Boot, requests are mapped to specific methods in controllers using annotations like `@RequestMapping` , `@GetMapping` , `@PostMapping` , and others. To avoid the error, ensure that you have correctly mapped the desired endpoint in one of your controllers.

Spring Boot Actuator provides an endpoint (`/actuator/mappings`) that allows you to view all the available mappings in your application. You can use this endpoint to ensure that your desired endpoint is mapped correctly.

Step-1: Add the following dependency to your `pom.xml` :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

```

Step-2: Enable the mappings endpoint by adding the below property in the `application.properties`

```
management.endpoints.web.exposure.include=mappings

```

Now, when you run your Spring Boot application, you can access the mappings endpoint at `http://localhost:8080/actuator/mappings` . It will show you all the available mappings in your application, helping

Solution-2: Handling Unmapped Requests

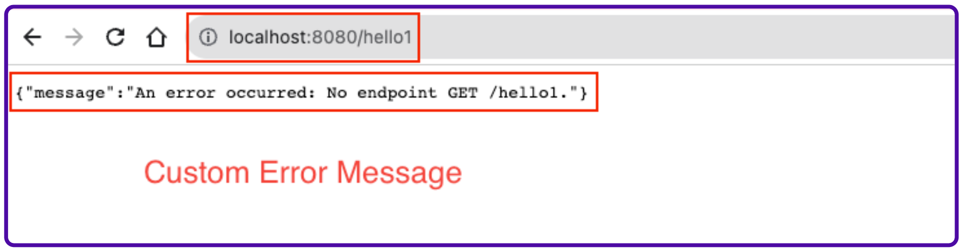
Effectively handling unmapped requests is crucial to provide a seamless user experience. One approach is to create a dedicated error controller using the `@ControllerAdvice` and `@ExceptionHandler` annotations. This allows you to customize the response for unmapped requests and offer more user-friendly error messages.

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler({NoHandlerFoundException.class})
    public ResponseEntity<ErrorResponse> handleNoHandlerFoun
            NoHandlerFoundException ex, HttpServletRequest ht
        System.out.println("No handler found exception");
        ErrorResponse error = new ErrorResponse("An error occ
        return ResponseEntity.status(HttpStatus.NOT_FOUND).co
    }
}

```

Apart from creating an exception handler to catch `NoHandlerFoundException` , some additional configuration is also required for this to work correctly. We have returned a detailed blog on this, you can check out the same by clicking here.


```
{"message":"An error occurred: No endpoint GET /hello1."}
```

Solution-3: Custom Error Page

In Solution-2, in case of 404 errors, the JSON response was displayed in the browser as well as in the case of the postman. If you want to display a custom HTML Error Page when users encounter unmapped or erroneous URLs, this is how it can be done:

Step-1: Add Thymeleaf Dependency

Add Thymelead dependency in pom.xml of your application.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

```

Step-2: Create error.html Page

Create an error page named `error.html` in the `src/main/resources/templates` directory:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initi
    <title>Error Page</title>
    <!-- Add your CSS styles and other assets here -->
</head>
<body>
<div>
    <h1>Oops! Something went wrong</h1>
    <p>We apologize for the inconvenience. The requested page
    <!-- You can add more helpful information or links to gu
</div>
</body>
```

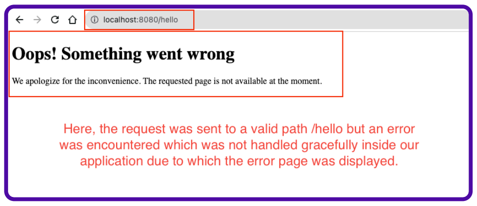
Add required properties in `application.properties` for view templates which will describe where to look for view templates and what will be the file extension for the same.

```
spring.mvc.view.prefix=/templates/
spring.mvc.view.suffix=.html

```

With this configuration, when an unmapped request occurs or when an unhandled exception is thrown, Spring Boot will use the custom error page to display the “Oops! Something went wrong” error message.

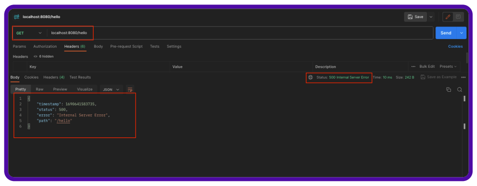
Output:


localhost:8080/hello

Oops! Something went wrong

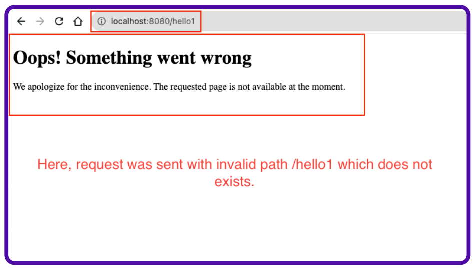
We apologize for the inconvenience. The requested page is not available at the moment.

Here, the request was sent to a valid path /hello but an error was encountered which was not handled gracefully inside our application due to which the error page was displayed.


GET localhost:8080/hello

Status: 500 Internal Server Error Time: 10 ms Size: 242 B

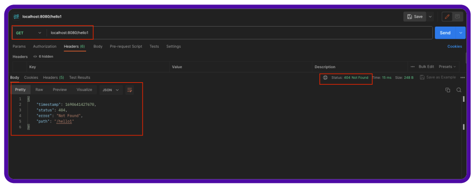
```
{
    "timestamp": 1690641583735,
    "status": 500,
    "error": "Internal Server Error",
    "path": "/hello"
}
```


localhost:8080/hello1

Oops! Something went wrong

We apologize for the inconvenience. The requested page is not available at the moment.

Here, request was sent with invalid path /hello1 which does not exists.


GET localhost:8080/hello1

Status: 404 Not Found Time: 15 ms Size: 248 B

```
{
    "timestamp": 1690641427670,
    "status": 404,
    "error": "Not Found",
    "path": "/hello1"
}
```

Solution-4: Implementing Global Exception Handling

If you want to display different error pages or if you want to send different error messages in JSON format for different kinds of errors:

Step-1: Implement the global exception handler

Create a global exception handler using Spring's `@ControllerAdvice` and `@ExceptionHandler` annotations. This handler will catch any unhandled exceptions and provide a meaningful response to the client.

Step-2: Define multiple exception handler methods

Define multiple exception handlers each one handling a specific kind of exception.

a. If you want to display the custom HTML page then the return type of these exception handler methods should be String. While returning the response, specify the template name that must be

b. If you want to send the JSON response instead of an HTML page then you can return the Java object containing the required fields.

Example:

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(NoHandlerFoundException.class)
    public String handleNotFoundError(Model model) {
        System.out.println("No handler found exception");
        String errorMessage = "OOops! Something went wrong -
        model.addAttribute("errorMessage", errorMessage);
        return "error"; // This will display the "error.html
    }

    @ExceptionHandler(CustomException.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ResponseBody
    public ErrorResponse handleCustomException(CustomExcepti
        // This will return the response in JSON format
        return new ErrorResponse("An error occurred: " + ex.
    }
}


```

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initi
    <title>Error Page</title>
    <!-- Add your CSS styles and other assets here -->
</head>
<body>
<div>
    <!-- Use Thymeleaf expression ${errorMessage} to display
    <h1 th:text="${errorMessage}">OOops! Something went wron
    <p>We apologize for the inconvenience. The requested pag
    <!-- You can add more helpful information or links to gu
</div>
</body>
</html>

```

When the `CustomException` is encountered, the JSON response will be sent to the user in the case of the **browser** as well as in the case of the **Postman**

FAQs

What is the purpose of the BasicErrorController in Spring Boot?

The `BasicErrorController` is a part of Spring Boot's default error-handling mechanism. It handles common error scenarios and provides error responses with relevant information, such as status code, error message, and error details.

How can I create a custom error page in my Spring Boot application?

To create a custom error page, you can use Thymeleaf or any other templating engine supported by Spring Boot. Create an HTML template (e.g., `error.html`) and configure the view resolver to render this custom error page when an error occurs.

they don’t conflict with each other. Be mindful of the order of exception handling to prioritize specific error scenarios.

Is it possible to customize error messages based on different types of exceptions in Spring Boot?

Yes, you can customize error messages based on different types of exceptions by using `@ExceptionHandler` for specific exception types. This allows you to handle different exception scenarios with tailored error messages.

Is it possible to add additional error information, such as a timestamp or error code, to my custom error page?

Yes, you can add additional error information to your custom error page. In the `ErrorController` , retrieve the required information and pass it as attributes to the model. Then, access these attributes in the Thymeleaf template and display them as needed.

Can I use a custom error page for specific HTTP status codes, such as 404 or 500, in my Spring Boot application?

Yes, you can create multiple custom error pages based on different HTTP status codes. Use `@ExceptionHandler` methods to handle specific exceptions that result in these status codes and return the corresponding custom error page accordingly.

experience to users.

2. **Proper Logging:** Implement robust logging to capture essential details about exceptions and errors for effective debugging.
3. **Graceful UX:** When providing error messages, make sure they are user-friendly and informative to guide users effectively.

Conclusion

In conclusion, dealing with the “This Application Has No Explicit Mapping for /error” issue in Spring Boot requires attention to URL mappings and implementing proper error handling. By creating custom error pages and using global exception handling, developers can provide better user experiences when errors occur. Understanding these troubleshooting techniques empowers developers to build more reliable and user-friendly Spring Boot applications.

Learn More

Interested in learning more?

Check out our blog on JAR vs WAR vs EAR vs Fat JAR.

Learn More