Guide to Spring ProblemDetail with example

Description

Introduction

In this post we will go over *Problem Details for HTTP APIs* with elaborate examples. Spring framework started supporting the Problem Details starting Spring 6 and Spring Boot 3 release. We are going to go over RFC 7897 which is an error handling standard for APIs.

What is RFC 7807 problem detail

The main goal of RFC 7803 specification is to define common error formats for applications. RFC 7807 put in place a standard, to report problems which occurred as a result of invoking a REST API. The RFC 7807 defined simple JSON and XML formats which can be used to communicate problem details to API consumers. Sometimes just passing the error HTTP status code is not enough to describe that the error occurred, more details are expected at the consumer side. In order to overcome the shortcomings of RFC 7231 which was very generic, we need a more fine grained details of the error which was satisfied by RFC 7807.

Below is an example of how the response would look like after the adoption.

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json
Content-Language: en

{
 "status": 400,
 "type": "https://website.com/documents/upload",
 "title": "KYC document missing",
 "detail": "Account verification failed due to missing KYC document",
 "instance": "/documents"
}
```

Let us see what does each key in the above response signify.

- ***status***: HTTP status code returned by the server.
- ***type***: URL that will help in mitigating the problem. Default value is *-about:blank*.
- ***title***: A short summary of the problem.
- ***detail***: Detailed explanation specific to this problem.

- ***instance***: URL that has the problem. The default value is the current request URL.

Abstraction in Spring Framework to support ProblemDetails

ProblemDetail

ProblemDetail class is one of the primary object which represents the problem detail model. We can set values in the ProblemDetail class for the keys present in the above section. Those fields are also called as standard fields.

```
ProblemDetail problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.BAD_
"Account verification failed due to missing KYC document");
problemDetail.setType(URI.create("https://website.com/documents/upload"));
problemDetail.setTitle("KYC document missing");
```

If you want to set any other additional field of your choice, then you can set it as below.

```
pd.setProperty("sample-property-key", "sample-property-value");
```

ErrorResponse

Exceptions in Spring MVC are compliant with RFC 7807 specification because Spring MVC exceptions already implement ErrorResponse interface.

ErrorResponse interface exposes details such as HTTP status, response headers and body of ProblemDetail type.

ErrorResponseException

ErrorResponseException is an implementation of ErrorResponse interface.

```
ProblemDetail problemDetail = ProblemDetail.forStatusAndDetail(HttpStatus.BAD_
"Account verification failed due to missing KYC document");
problemDetail.setType(URI.create("https://website.com/documents/upload"));
problemDetail.setTitle("KYC document missing");

throw new ErrorResponseException(HttpStatus.BAD_REQUEST, problemDetail, null);
```

ProblemDetails using ResponseEntity and Exception scenarios

As you already know that ResponseEntity is used to send the response back to the caller, from the controller layer. The ResponseEntity usually contains the Response object along with the Response code. We can use ProblemDetails object along with ResponseEntity in case of error scenarios.

Let us see with an example below.

```
@GetMapping(path = "/documents/{id}")
public ResponseEntity getDocumentsById(@PathVariable("id") UUID id) {
    try {
        Documents documents = documentService.getDocumentsById(id);

        if(ObjectUtils.isEmpty(documents)) {
              ProblemDetail problemDetail = ProblemDetail.forStatusAndDetail(H
              problemDetail.setType(URI.create("https://website.com/documents/
              problemDetail.setTitle("Document Not Found");
              return ResponseEntity.status(404).body(problemDetail);
        }
        return ResponseEntity.ok(documents);
    }
    catch(Exception exp) {
        ProblemDetail problemDetail = ProblemDetail.forStatusAndDetail(HttpSta
        problemDetail.setType(URI.create("https://website.com/documents"));
        problemDetail.setTitle("API encountered a fatal error. Try again!");
        return ResponseEntity.status(500).body(problemDetail);
    }
}
```

In case of exception the response will look like below

```
{
 "status": 500,
 "type": "https://website.com/documents",
 "title": "API encountered a fatal error. Try again!",
 "detail": "Something went wrong!",
 "instance": "/documents/e8fcdb19-c653-4792-9849-50196310b155"
}
```

Adding ProblemDetail to Custom Exceptions

Almost all the projects or teams create their own user defined exceptions. User defined exceptions extends RuntimeException. Examples can be InvalidAgeException, UserNotFoundException, InsufficientBalanceException etc. User defined exceptions are more readable and easily understood.

The best place to incorporate ProblemDetails information w.r.t User defined exception is, in the class annotated with @ControllerAdvice.

```
@ControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

  @ExceptionHandler(DocumentNotFoundException.class)
  public ProblemDetail handleDocumentNotFoundException(DocumentNotFoundExcepti

    ProblemDetail detail = ProblemDetail
        .forStatusAndDetail(HttpStatus.INTERNAL_SERVER_ERROR, "Something went
    detail.setType(URI.create("https://website.com/documents"));
    detail.setTitle("API encountered a fatal error. Try again!");
    detail.setInstance(URI.create(String.format("/documents/%s",id)));
    return detail;
  }
}
```

Conclusion

By using Spring Frameworkâ€™s ProblemDetails we can set a benchmark on error response that ever microservice should follow. We learnt various ways of implementing ProblemDetails along with easy to understand examples. Feel free to comment down below on how you liked the post or any improvement suggestions.

Category

1. Spring Boot

Date Created
July 17, 2023
Author
kk-ravi144gmail-com