

Spring Boot Test Customization



I have to test a class without mock in spring boot based project with test profile.

How to customise to provide success, failure and exception



To test a class without using mocks in a Spring Boot-based project while leveraging the test profile, you can customize the behavior of your test by providing real instances with different configurations for success, failure, and exception scenarios. Here's how you can achieve this:

1. Set Up a Test Profile

Create a `test` profile in your `application.yml` or `application.properties` to define configurations specific to testing.

For example, in `application-test.yml`:

```
yaml

spring:
  datasource:
    url: jdbc:h2:mem:testdb
    username: sa
    password: password
    driver-class-name: org.h2.Driver
  jpa:
    hibernate:
      ddl-auto: update
```

Activate the `test` profile in your test



Write a helper class which will return success, failure or exception on basis of json content..



Here's a helper class that can parse JSON content and return a response (success, failure, or exception) based on the JSON's structure or content:

Helper Class Implementation

```
java

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonResponseHelper {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    /**
     * Processes the input JSON and returns a response based on its content.
     *
     * @param jsonContent The JSON content as a String.
     * @return Response message based on JSON content.
     * @throws IllegalArgumentException If the input JSON indicates an exception.
     */
    public static String processJson(String jsonContent) {
        try {
            // Parse the JSON content
            JsonNode jsonNode = objectMapper.readTree(jsonContent);

            // Check for success
            if (jsonNode.has("status") && "success".equalsIgnoreCase(jsonNode.get("status").asText())) {
                return "Success: " + jsonNode.get("message").asText();
            }

            // Check for failure
            if (jsonNode.has("status") && "failure".equalsIgnoreCase(jsonNode.get("status").asText())) {
                return "Failure: " + jsonNode.get("error").asText();
            }

            // Check for exception
            if (jsonNode.has("status") && "exception".equalsIgnoreCase(jsonNode.get("status").asText())) {
                throw new IllegalArgumentException("Exception: " + jsonNode.get("error").asText());
            }

            // Default case
            return "Unknown status in JSON content";
        } catch (Exception e) {
            throw new RuntimeException("Error processing JSON: " + e.getMessage(), e);
        }
    }
}
```

Example Usage

You can write unit tests or use this helper in your main code to process JSON.

Test Scenarios

```
java

public class JsonResponseHelperTest {

    @Test
    public void testSuccessJson() {
        String successJson = "{\"status\": \"success\", \"message\": \"Operation completed successfully.\"}";
        String response = JsonResponseHelper.processJson(successJson);
        assertEquals("Success: Operation completed successfully.", response);
    }

    @Test
    public void testFailureJson() {
        String failureJson = "{\"status\": \"failure\", \"error\": \"Something went wrong.\"}";
        String response = JsonResponseHelper.processJson(failureJson);
        assertEquals("Failure: Something went wrong.", response);
    }

    @Test
    public void testExceptionJson() {
        String exceptionJson = "{\"status\": \"exception\", \"error\": \"Critical error occurred.\"}";
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            JsonResponseHelper.processJson(exceptionJson);
        });
        assertEquals("Exception: Critical error occurred.", exception.getMessage());
    }

    @Test
    public void testUnknownStatusJson() {
        String unknownJson = "{\"status\": \"unknown\", \"message\": \"Unrecognized status.\"}";
        String response = JsonResponseHelper.processJson(unknownJson);
        assertEquals("Unknown status in JSON content", response);
    }
}
```

Key Points

- JSON Parsing:** Uses Jackson's `ObjectMapper` to parse JSON.
- Content-Based Logic:** Determines the response based on the value of the `status` field in the JSON.
- Error Handling:** Throws an exception for "exception" status and catches parsing issues.



The exception could be anything



If the exception could be dynamic (different types of exceptions), you can enhance the helper class to support a configurable or dynamic exception mechanism. Here's how you can achieve this:

Enhanced Helper Class

```
java

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonResponseHelper {

    private static final ObjectMapper objectMapper = new ObjectMapper();

    /**
     * Processes the input JSON and returns a response based on its content.
     *
     * @param jsonContent The JSON content as a String.
     * @return Response message based on JSON content.
     * @throws Exception If the input JSON indicates an exception with a specific type.
     */
    public static String processJson(String jsonContent) throws Exception {
        try {
            // Parse the JSON content
            JsonNode jsonNode = objectMapper.readTree(jsonContent);

            // Check for success
            if (jsonNode.has("status") && "success".equalsIgnoreCase(jsonNode.get("status").asText())) {
                return "Success: " + jsonNode.get("message").asText();
            }

            // Check for failure
            if (jsonNode.has("status") && "failure".equalsIgnoreCase(jsonNode.get("status").asText())) {
                return "Failure: " + jsonNode.get("error").asText();
            }

            // Check for exception and dynamically throw the exception
            if (jsonNode.has("status") && "exception".equalsIgnoreCase(jsonNode.get("status").asText())) {
                String exceptionType = jsonNode.get("exceptionType").asText();
                String exceptionMessage = jsonNode.get("error").asText();
                String errorMessage = jsonNode.get("message").asText();
                // Dynamically throw the exception based on the type
                throwExceptionDynamically(exceptionType, exceptionMessage);
            }

            // Default case
            return "Unknown status in JSON content";
        } catch (Exception e) {
            throw new RuntimeException("Error processing JSON: " + e.getMessage(), e);
        }
    }

    /**
     * Dynamically throws an exception based on the type and message.
     *
     * @param exceptionType The fully qualified name of the exception class.
     * @param message The exception message.
     * @throws Exception The dynamically created exception.
     */
    private static void throwExceptionDynamically(String exceptionType, String message) throws Exception {
        try {
            Class<?> exceptionClass = Class.forName(exceptionType);
            if (Throwable.class.isAssignableFrom(exceptionClass)) {
                throw (Throwable) exceptionClass.getConstructor(String.class).newInstance(message);
            } else {
                throw new IllegalArgumentException("Invalid exception type: " + exceptionType);
            }
        } catch (ClassNotFoundException e) {
            throw new IllegalArgumentException("Unknown exception type: " + exceptionType, e);
        }
    }
}
```

JSON Input Examples

Success Example:

```
json

{
  "status": "success",
  "message": "Operation completed successfully."
}
```

Failure Example:

```
json

{
  "status": "failure",
  "error": "Something went wrong."
}
```

Exception Example with Custom Exception:

```
json

{
  "status": "exception",
  "exceptionType": "java.lang.IllegalArgumentException",
  "error": "Invalid input provided."
}
```

Exception Example with Default Exception:

```
json

{
  "status": "exception",
  "error": "Critical error occurred."
}
```

Test Scenarios

```
java

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class JsonResponseHelperTest {

    @Test
    public void testSuccessJson() throws Exception {
        String successJson = "{\"status\": \"success\", \"message\": \"Operation completed successfully.\"}";
        String response = JsonResponseHelper.processJson(successJson);
        assertEquals("Success: Operation completed successfully.", response);
    }

    @Test
    public void testFailureJson() throws Exception {
        String failureJson = "{\"status\": \"failure\", \"error\": \"Something went wrong.\"}";
        String response = JsonResponseHelper.processJson(failureJson);
        assertEquals("Failure: Something went wrong.", response);
    }

    @Test
    public void testCustomExceptionJson() {
        String exceptionJson = "{\"status\": \"exception\", \"exceptionType\": \"java.lang.IllegalArgumentException\", \"error\": \"Invalid input provided.\"}";
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            JsonResponseHelper.processJson(exceptionJson);
        });
        assertEquals("Invalid input provided.", exception.getMessage());
    }

    @Test
    public void testDefaultExceptionJson() {
        String exceptionJson = "{\"status\": \"exception\", \"error\": \"Critical error occurred.\"}";
        Exception exception = assertThrows(Exception.class, () -> {
            JsonResponseHelper.processJson(exceptionJson);
        });
        assertEquals("Critical error occurred.", exception.getMessage());
    }

    @Test
    public void testUnknownExceptionType() {
        String unknownExceptionJson = "{\"status\": \"exception\", \"exceptionType\": \"com.example.NonExistingException\", \"error\": \"Custom exception occurred.\"}";
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            JsonResponseHelper.processJson(exceptionJson);
        });
        assertTrue(exception.getMessage().contains("Unknown exception type"));
    }
}
```

Key Features

- Dynamic Exception Handling:** Throws exceptions dynamically based on the `exceptionType` field in the JSON.
- Fallback to Default Exception:** If `exceptionType` is not specified, defaults to `java.lang.Exception`.
- Error Safety:** Handles unknown exception types gracefully by throwing an `IllegalArgumentException`.

This implementation provides flexibility to handle various success, failure, and exception scenarios dynamically based on JSON input.