Testing

Since our development process was TDD (**t**est-**d**riven-**d**evelopment), tests were implemented before any implementation. To explain this process of development in detail, an [example is provided further below](), which dives into the development of the branch `feature/api-message-reset`.

In addition to our process, we needed to implement **c**ontinous **i**ntegration (CI) and **c**ontinous **d**eployment (CD) pipelines. Those pipelines used the default maven commands, to build our project for artifacts and execute the available unit tests on them.
The details about that, are in our CI/CD documentation.

In the following sections, we document what our unit tests do. The sections are seperated by feature branch, meaning each section contains a different API path with different corresponding testcases.

Since the internal state of the application stays the same throughout test execution, we needed to set the annotation `@DirtiesContext` for some testcases, otherwise one test could affect the expected initial state of another.

feature/api-message

The `message` feature returns either the current service message or the default service message, if the `current` message hasn't been set.

GetMessageTest

This testcase verifies the existance and functionality of the APi-path `api/message`.
On initial start of the webpage, the service doesn't have an initialized `message`, which is why we provide a default message.
The initial default message is statically set to `Status OK`.

We expect a positive response `200` of the API call and verify the contents of the response against our expected static message.

```
@Test
@Order(1)
void GetMessageTest() throws Exception {
    mockMvc.perform(get("/api/message"))
            .andExpect(status().is2xxSuccessful())
            .andExpect(content().string("Status Ok"));
}
```

GetMessageWithoutDefaultTest

This testcase verifies the expected failure of `api/message`.

We expect a failure of `api/message` when the `default` message was set to a blank message, without having called `api/message/set` to initialize the `current` message.

This failure prevents a segmentation fault, because when trying to return a `null` current message, it would mean we have an access violation.

```
@Test
@Order(2)
// Makes context dirty because: defaultMessage will be blank
@DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
void GetMessageWithoutDefaultTest() throws Exception {
    mockMvc.perform(get("/api/message/default?msg="))
            .andExpect(status().is2xxSuccessful())
            .andExpect(content().string(""));
    mockMvc.perform(get("/api/message"))
            .andExpect(status().is5xxServerError())
            .andExpect(content().string("No default message or message set"));
}
```

feature/api-message-set

The `set` feature is supposed to **set** the current message of the server to a provided message. We did not set any limitations due to the small scale of the project, as such the API simply sets the message without checking the provided string.

GetMessageSetTest

This test verifies the normal behaviour of the API call, by setting the current message to a string and verifying the response header as well as the content, which should correspond to the message set.

```
@Test
@Order(1)
void GetMessageSetTest() throws Exception {
    mockMvc.perform(get("/api/message/set?m=Giraffe"))
        .andExpect(status().is2xxSuccessful())
        .andExpect(content().string("Giraffe"));
}
```

GetMessageSetMultipleWordsTest

This test verifies if the message can be set to a message containing spaces, which implicitly gets parsed to a correct URL by the mocking client.
In practice, the used test string would look like so:

`/api/message/set?m=The+giraffe+is+a+large+African+hoofed+mammal+belonging+to+the+genus+Giraffa`

The message is again verified in the response header as well as in the content.

```
@Test
@Order(2)
void GetMessageSetMultipleWordsTest() throws Exception {
    mockMvc.perform(get("/api/message/set?m=The giraffe is a large African
hoofed mammal belonging to the genus Giraffa"))
        .andExpect(status().is2xxSuccessful())
        .andExpect(content().string("The giraffe is a large African hoofed
mammal belonging to the genus Giraffa"));
}
```

feature/api-message-reset

The `reset` feature is supposed to **reset** the current message to a default message of the server. We adapted the requirement a little bit, which is why we also added an API path for a `default` message, so we can make sure that there's a message to begin with, without requiring the implementation of `set`.

GetMessageResetTest

This testcase verifies the existence and functionality of the API-path `/api/message/reset` by expecting a response of `200` when calling it with the `MockMvc` client.

By checking the response, we successfully tested the functionality of `reset`, since that's what it responds with.

This test is executed **first**.
It affects the internal state of `currentApiMessage` by setting it to the content of `apiMessageDefault`.

```
@Test
@Order(1)
// Makes context dirty because: currentMessage will be set to "Status Ok"
@DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
void GetMessageResetTest() throws Exception {
    mockMvc.perform(get("/api/message/reset"))
            .andExpect(status().is2xxSuccessful());
}
```

GetMessageDefaultTest

This testcase verifies the `default` API-path functionality.

It sets the default message, verifies the response of the API call and then verifies the contents of the response.
After successful execution of this path, it resets the default message to the original state and verifies the result.

This test is executed **second**.
It affects the internal state of `apiMessageDefault` by setting it to a blank string.

```
@Test
@Order(2)
// Makes context dirty because: defaultMessage will be blank
@DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
void GetMessageDefaultTest() throws Exception {
    mockMvc.perform(get("/api/message/default?msg=Hello"))
            .andExpect(status().is2xxSuccessful())
            .andExpect(content().string("Hello"));
    mockMvc.perform(get("/api/message/default?msg="))
            .andExpect(status().is2xxSuccessful())
            .andExpect(content().string(""));
}
```

GetMessageResetWithoutDefaultTest

This testcase verifies the expected failure of the `reset` API-path.

When `reset` is called without a `default` message, an internal server error occurs, since it requires the default message to be set.
We defined this testcase, to make sure that the implementation will use the default message for calling `reset`, because we defined that `reset` doesn't clear the message, but instead resets the message to a default one. Without a default message, this API cannot be called.

This test is executed **third**.
It affects the internal state of `apiMessageDefault` by setting it to a blank string.

```
@Test
@Order(3)
// Makes context dirty because: defaultMessage will be blank
@DirtiesContext(methodMode = DirtiesContext.MethodMode.AFTER_METHOD)
void GetMessageResetWithoutDefaultTest() throws Exception {
    mockMvc.perform(get("/api/message/default?msg="))
            .andExpect(status().is2xxSuccessful())
            .andExpect(content().string(""));
    mockMvc.perform(get("/api/message/reset"))
            .andExpect(status().is5xxServerError())
            .andExpect(content().string("Default message is not set."));
}
```

Example process in feature branch

To implement a test, we used the testing features of the [spring boot framework](#).
Initially a test class looks like the following example:

Add MessageResetTests class to test /api/message/reset

development (#16) + documentation (#19, #16) + feature/api-message (#16) + feature/api-message-reset (#16)

HackXIt committed on Oct 4

Showing **1 changed file** with **13 additions** and **0 deletions**.

13 src/test/java/at/fhtw/bic/slmstudyproject/controller/MessageResetTests.java

```
@@ -0,0 +1,13 @@
+ package at.fhtw.bic.slmstudyproject.controller;
+
+ import org.junit.jupiter.api.Test;
+ import org.springframework.beans.factory.annotation.Autowired;
+ import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
+ import org.springframework.boot.test.context.SpringBootTest;
+ import org.springframework.test.web.servlet.MockMvc;
+
+ @WebMvcTest(controllers = MessageController.class)
+ @SpringBootTest
+ public class MessageResetTests {
+
+ }
```

As intended in TDD, even this simple test class will initially fail:

```
Build: Build Output
SLM-Study-Project: build failed At 30/11/2022 16:33 with 1 error  11 sec, 970 ms
MessageResetTests.java src\test\java\at\fhtw\bic\slmstudyproject\controller 1 error
cannot find symbol class MessageController :9

D:\#Git-Stash\SLM-Study-Project\src\test\java\at\fhtw\bic\slmstudyproject\controller
\MessageResetTests.java:9:27
java: cannot find symbol
  symbol: class MessageController

Build completed with 1 error and 0 warnings in 11 sec, 970 ms (moments ago)
14:1 (81 chars, 5 line breaks)  CRLF  UTF-8  4 spaces  f3303672
```

The process of TDD dictates, that we now refactor our code to fix this initial error:

MessageController.java

```
1  package at.fhtw.bic.slmstudyproject.controller;
2
   HackXIt
3  public class MessageController {
4  }
5
```

```
Run: MessageResetTests
No tests were found
C:\Users\rini\.jdks\openjdk-17.0.1\bin\java.exe ...

Process finished with exit code 0

Tests passed: 0 (moments ago)
5:1  CRLF  UTF-8  4 spaces  6188d69a
```

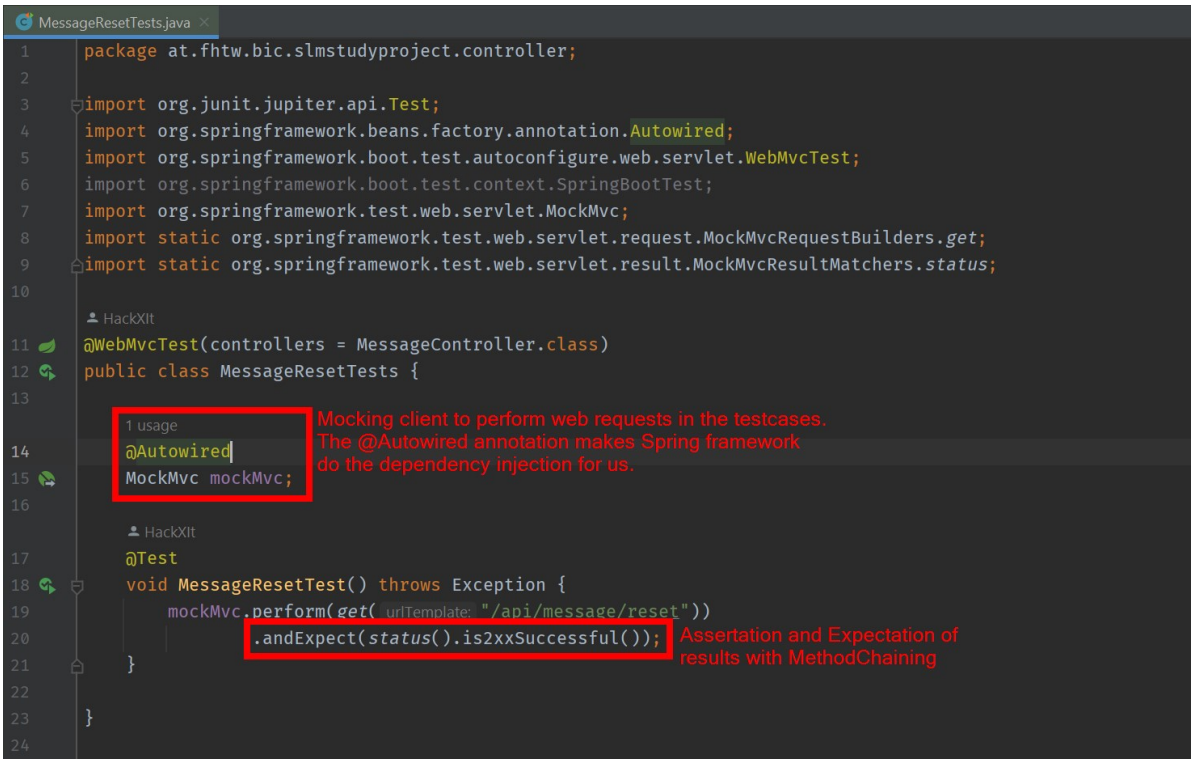
```
* 6188d69 Fix build-error, add MessageController class
* f330367 Add MessageResetTests class to test /api/message/reset
```

Great!

But we still don't have any relevant tests, which were implemented using the same principle as before:

1. Implement a test
2. Execute test and expect failure
3. Refactor code until Expectation succeeds.
4. Repeat.

To help us implement our testcases for the actual API paths, we required some additional help of the Spring Framework, since we need a client to make API requests.
For this we used the provided `MockMvc`, which is a complete `mocking` client to do web requests. The client is very useful, since it provides mechanism to `assert` and `expect` results from the reponse of the API, which allowed us to design our API with TDD as process.



In the feature branch, the process was continued until all requirements of the feature were implemented and commited.

For our example feature `feature/api-message-reset`, our resulting tests were:
(`MessageResetTests` @ `f91fff5`)

```
package at.fhtw.bic.slmstudyproject.controller;

import org.junit.jupiter.api.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest(controllers = MessageController.class)
```

```java
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class MessageResetTests {

    @Autowired
    MockMvc mockMvc;

    // Using test order, since when GetMessageResetTest is run between, it will
fail,
    // since Initial Default Message was already reset

    @Test
    @Order(1)
    void GetMessageResetTest() throws Exception {
        mockMvc.perform(get("/api/message/reset"))
                .andExpect(status().is2xxSuccessful());
    }

    @Test
    @Order(2)
    void GetMessageDefaultTest() throws Exception {
        mockMvc.perform(get("/api/message/default?msg=Hello"))
                .andExpect(status().is2xxSuccessful())
                .andExpect(result ->
result.getResponse().getContentAsString().contentEquals("Hello"));
        mockMvc.perform(get("/api/message/default?msg="))
                .andExpect(status().is2xxSuccessful())
                .andExpect(result ->
result.getResponse().getContentAsString().isBlank());
    }

    @Test
    @Order(3)
    void GetMessageResetWithoutDefaultTest() throws Exception {
        mockMvc.perform(get("/api/message/default?msg="))
                .andExpect(status().is2xxSuccessful())
                .andExpect(result ->
result.getResponse().getContentAsString().isBlank());
        mockMvc.perform(get("/api/message/reset"))
                .andExpect(status().is5xxServerError())
                .andExpect(result ->
result.getResponse().getContentAsString().contentEquals("Default message is not
set."));

    }

}
```

And here is the final implementation of these tests: (`MessageController.java` @ `f91fff5`)

```java
package at.fhtw.bic.slmstudyproject.controller;

import org.springframework.beans.factory.annotation.Required;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.HttpMediaTypeException;
```

```java
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/message")
public class MessageController {

    private String apiMessageDefault = "Status Ok";
    private String currentApiMessage;

    @GetMapping("/reset")
    public ResponseEntity<String> MessageReset() {
        if(apiMessageDefault.isBlank()) {
            return new ResponseEntity<>("Default message is not set.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
        currentApiMessage = apiMessageDefault;
        return new ResponseEntity<>(currentApiMessage, HttpStatus.OK);
    }

    @GetMapping("/default")
    public ResponseEntity<String> SetMessageDefault(@RequestParam(required=true)
String msg) {
        apiMessageDefault = msg;
        return new ResponseEntity<>(apiMessageDefault, HttpStatus.OK);
    }

}
```