Erklärung der automatischen Tests

Schriftliche Ausarbeitung

vorgelegt von

Bjarne Christel
aus Mönchengladbach
geboren am: 29.03.1999
Matrikelnr.: 1344413

Tobias Piepers
aus Mönchengladbach
geboren am: 19.01.1998
Matrikelnr.: 1359763

Louisa Schmitz
aus Krefeld
geboren am: 11.07.2000
Matrikelnr.: 1319646

Hochschule Niederrhein
Fachbereich Wirtschaftswissenschaften
Studiengang M. Sc. Wirtschaftsinformatik
Wintersemester 2023/ 2024
Prüfer: Prof. Dr. Schekelmann

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In Aufgabe 5 wurden automatische Tests für die Fachdomäne implementiert. Das Domain-Driven-Design ermöglicht es, die fachlichen Komponenten in Aggregates, Entities, Value Objects und Domain Services aufzuteilen und separat zu testen.
Dies ermöglicht eine Isolierung des zu testenden Quellcodes und vereinfacht die Identifizierung von Fehlern sowie die Verbesserung der Code-Qualität. Zudem ermöglicht die fachliche Trennung eine reibungslosere Wartung als auch Erweiterung des Systems, ohne einen direkten Einfluss auf das gesamte System zu nehmen. Dadurch resultiert ein stabileres und verlässlicheres System.

2 Recherche

Es wurde eine Internetrecherche durchgeführt, um herauszufinden, wie die Microservices separat getestet werden können. Es wurde nach bereits durchgeführten Implementierungen sowie den notwenigen Grundlagen gesucht. Dabei konnten zwei GitHub-Profile gefunden und für die JUnit-Tests genutzt werden.

1. https://github.com/DomingoAlvarez99/ddd-example
2. https://github.com/VaughnVernon/IDDD_Samples/blob/master/iddd_collaboration/src/test/java/com/saasovation/collaboration/domain/model/calendar/CalendarTest.java

3 Aufbau

Die Architektur der Paketstruktur aus der vorherigen Teilaufgabe wurde bei der Implementierung der Test-Klassen übernommen. Dabei wurde die Trennung der Fachlichkeit berücksichtigt. Ausführlichere Details zur Aufteilung können der Ausarbeitung „Package-Struktur in Eclipse“ entnommen werden.
Die Annotation "@SpringBootTest" ermöglicht die Ausführung der gesamten Klasse und die Erstellung des ApplicationContexts für den Test. Die weitere Annotation „@Test“ kennzeichnet eine Testmethode und ermöglicht den Zugriff auf SpringBoot. Wenn das Testergebnis negativ ist, wird eine Exception ausgelöst und der Test als nicht bestanden markiert und in der Konsole dementsprechend dargestellt. Die Tests können mit der Methode „C:\apache-maven-3.9.5\bin\mvn test“ in der Konsole durchgeführt werden (vgl. Abbildung 1).

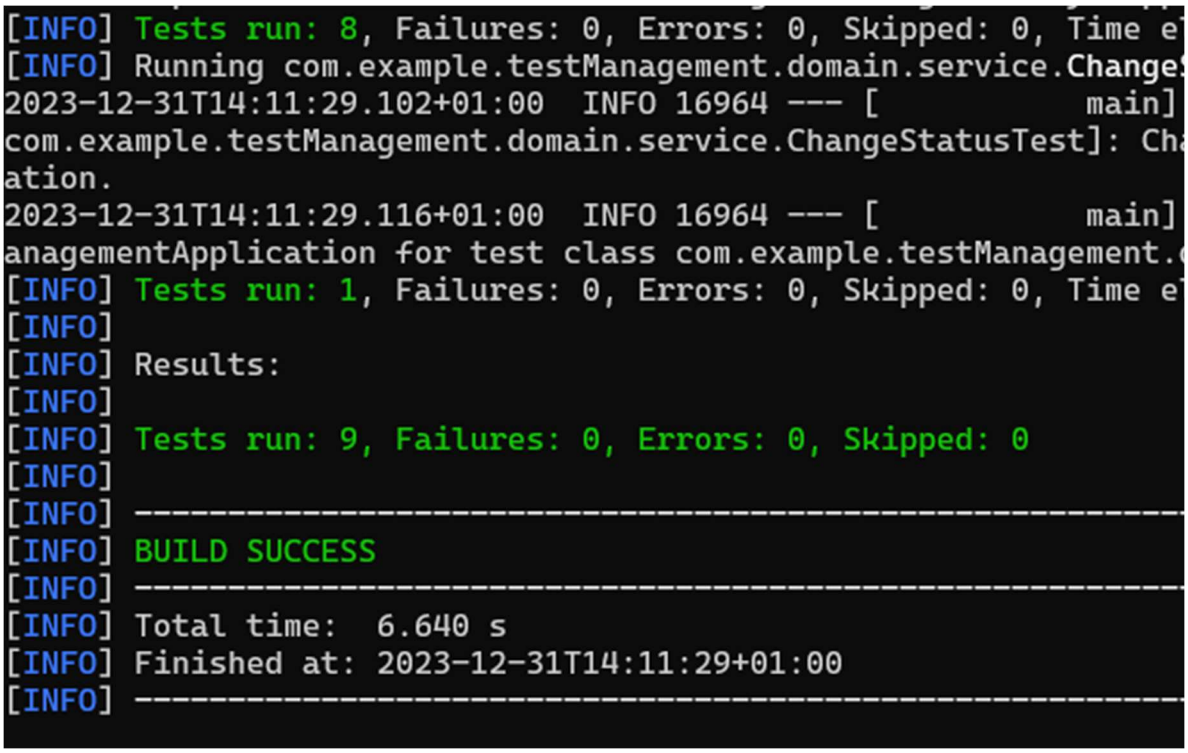

Abbildung 1: Testdurchlauf; Quelle: eigene Darstellung

4 Fachliche Test:

Um alle Klassenfunktionen zu testen, wurden für jede Klasse mehrere Testmethoden erstellt. Diese lassen sich wie folgt aufteilen:

Der Equals-Test überprüft, ob zwei verschiedene Variablen derselben Klasse, die mit inhaltsgleichen Parametern erstellt wurden, nicht identisch sind. Jeder Wert hat einen eindeutigen Hashwert, der jeweils nur einmal vergeben wird (vgl. Abbildung 2).

```java
@Test
public void testTestCaseEquals() throws Exception {
    TestCase testCase1 = new TestCase(new TestCaseId(100), "Test");
    TestCase testCase2 = new TestCase(new TestCaseId(100), "Test");

    assertNotSame(testCase1, testCase2);
}
```

Abbildung 2: Equals-Test; Quelle: eigene Darstellung

Bei einem Status-Test wird überprüft, ob der Status einer Variablen korrekt geändert wird, nachdem sie weiterverarbeitet wurde (vgl. Abbildung 3).

```java
@Test
public void testChangeTestCaseStatus() throws Exception {
    testCase.changeTestStatus(StoryStatus.READY_FOR_TEST);
    assertEquals("ready for test", testCase.getTestStatus().toString());
}
```

Abbildung 3: Status-Test; Quelle: eigene Darstellung

Der Description-Test überprüft, ob die Beschreibung der Test-Cases korrekt geändert und gespeichert wird (vgl. Abbildung 4).

```java
@Test
public void testChangeTestCaseDescription() throws Exception {
    testCase.describeTest("Test");
    assertEquals("Test", testCase.getTestDescription());
}
```

Abbildung 4: Beschreibungs-Test; Quelle: eigene Darstellung

Die Methode assertTrue() überprüft, ob ein bestimmter Ausdruck wahr ist, während die Methode assertEquals() überprüft, ob zwei Werte gleich sind. Wenn diese Bedingungen nicht erfüllt sind, gilt der Test als nicht bestanden.

Die Methode fail() wird genutzt, um einen Test bewusst scheitern zu lassen. In diesem Fall wird sie verwendet, um sicherzustellen, dass die erwartete Ausnahme auftritt. Wenn keine Ausnahme ausgelöst wird, gilt der Test als nicht bestanden.