Fallstudie Unittest, Mock und Datenbank

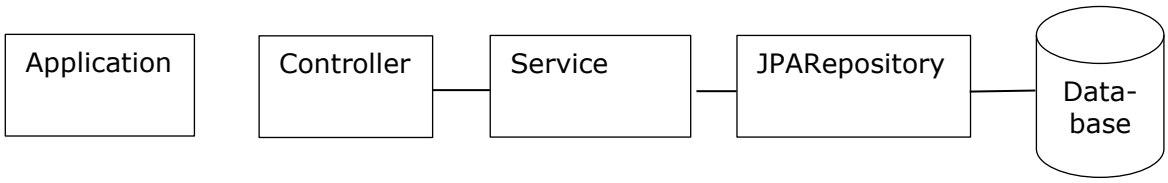
Einleitung:

Voraussetzung für diese Aufgabe ist zuvor Ihre Auseinandersetzung mit der Aufgabe *22_7_SpringBootDatenbankUnittestMockInput.pdf*

Sie werden nun die Erkenntnisse aus diese Vorprojekt in ein etwas komplexeres Projekt übertragen.

Sie bauen auf dem Projekt der Lehrperson auf. In Absprache mit der Lehrperson können Sie auch ein eigenes Projekt (bisher ohne Unittests) aus einem anderen Modul verwenden

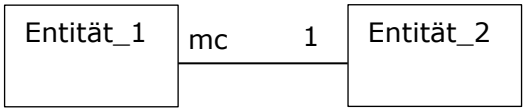
Aufbau des Projektes:



Ausgangslage der Datenbank

Die Datenbank muss *zwei Entitäten* abbilden, die über eine *Beziehung* verbunden sind.

Diese Beziehung muss via JPA in Projekt abgebildet sein.



In der Fallstudie müssen Klassen mit Hilfe von Mocks für die Tests ersetzt werden.

Dabei geht es um Unit-, Komponenten- und End-to-End-Tests.

Ziel

- ⇨ Sie können das Prinzip vom Mocken auf eine Situation mit einer Datenbank übertragen.
- ⇨ Sie können Unit-, Komponenten- und End-to-End-Tests mit Unterstützung von Mocks erstellen.
- ⇨ Sie definieren geeignete Testfälle für «Schön»- und «Schlecht»-Wetterfälle.
- ⇨ Sie werten in den Tests die erhalten Daten detailliert aus.
- ⇨ Sie testen gezielt die verschiedenen Pfade der geprüften Klassen.

Inhalte

Einleitung: 1
Ziel 1
Inhalte 1
Voraussetzung schaffen 2
Aufgabe Fallstudie 2

Voraussetzung schaffen

Haben Sie sich noch nicht mit der Aufgabe *22_7_SpringBootDatenbankUnittestMockInput.pdf auseinandergesetzt, dann tun Sie das.*

Die Fallstudie müssen Sie danach auf ein anderes Projekt anwenden.

Aufgabe Fallstudie

Nun wenden Sie die Tests an einem anderen Projekt an.

Vorbereitung

- Übernehmen Sie ein bestehendes Projekt mit Springboot und einem JPA Repository. Zur Verfügung steht 22_9_JPA-CRUD-EmployeeSPBMVC_Vorlage.zip
- Falls Sie ein eigenes Projekt nehmen wollen, dann muss es einige Kriterien erfüllen:
  - Datenbank mit mindestens zwei mit einer Beziehung verbundenen Entitäten.
  - JPA Repositories
  - Services, die den Zugriff auf die Repositories kapseln
  - Business-Service, der die Services für komplexere Anwendungen nutzt
  - Controller der die Services und den Business-Service nutzt und mit dem User interagiert.

Tests

- Entwerfen und implementieren Sie die Unittests mit Hilfe von Mocks.
- Achten Sie dabei auf die Daten, die teilweise über die Beziehung der Entitäten komplexer wird.
  Stellen Sie die Daten (inkl. Beziehung) und überprüfen Sie die Ergebnisse auf diese Daten.
- Sie müssen in der Lage sein, zu erklären, wie Sie den Transfer zu Ihrer konkreten eigenen Anwendung vollzogen haben.
- Sie müssen Tests für
  - Unittests für den EmployeeService.
    - Für diese Tests müssen die beiden Repositories *DeparmentRepository* und *EmployeeRepository* durch Mocks ersetzt werden.
  - Unittests als *Komponententests* für den BusinessService
    - ACHTUNG:
      Für diese Tests **wird nicht** EmployeeService durch einen Mock ersetzt!!.
    - Es werden wiederum die beiden Repositories *DeparmentRepository* und *EmployeeRepository* durch Mocks ersetzt. Damit erhält dieser Test den Aspekt eines Komponenten Tests.
  - Unittests als *End-To-End-Test* für den *ViewController*
    - ACHTUNG: Auch hier das Richtige mocken!!
    - Es werden wiederum die beiden Repositories *DepartmentRepository* und *EmployeeRepository* durch Mocks ersetzt. Damit erhält dieser Test den Aspekt eines End-To-End Tests.
    - Verwenden Sie hier die Möglichkeiten von mockMvc für das Testen eines Controllers an.
      (Nicht die Lösung aber eine Einführung:
      https://www.baeldung.com/integration-testing-in-spring)

- Umfang:
    - Testen Sie mindestens 4 Methoden pro Klasse
      - EmployeeService
      - BusinessService
      - *ViewController*
    - Testen Sie bei den Ergebnissen die detaillierten Daten von *Employee* und falls verbunden *Department.*
    - Testen Sie die unterschiedlichen Pfade der Methoden.
    - Testen Sie den erfolgreichen Aufruf von Methoden und den nicht erfolgreichen Aufruf von Methoden.
      ZBsp Employee mit Namen wird gefunden / Employee mit Namen wird nicht gefunden.
    - Weisen Sie in kurzen Kommentaren zu den Tests darauf hin, wo Sie obiges angewendet haben.

- Kompetenz:
    - Sie müssen fähig sein, zu erklären, was Ihre Tests machen, was die Zeilen in den Test bewirken.
    - Sie müssen fähig sein, den Transfer zu weiteren Tests und deren Test-Elementen, Begriffen zu machen und diese Tests erklären zu können.

Anwendung von KI:

- Sie dürfen KI anwenden.
- Sie müssen aber klar mit einem deutlichen Kommentar im Code deklarieren, wo Sie KI eingesetzt haben und was Ihr persönlicher Beitrag am KI-Code ist.
  Oder wenn Sie nichts dazu beigetragen haben, dann auch dies vermerken.
- Sie müssen fähig sein, den von der KI erzeugten Code zu erklären.
- Sie müssen fähig sein, den Transfer zu weiteren Tests und deren Test-Elementen, Begriffen zu machen und diese Tests erklären zu können.

Ergebnissicherung:

- Geben Sie Ihr Projekt als ZIP-Datei ab.