V1.0

Testkonzept

Dokumenteninformation

Erstellt von: Johannes Hüttinger

Bearbeitet von: Johannes Hüttinger

Dateiname: Testkonzept_v1.0.pdf

Versionierung

Version	Datum	Bearbeiter	Änderung	Stand
1.0	17.04.2021	Johannes Hüttinger	Initiale Erstellung	Abgenommen

Inhalt

Allgemein gelten die Vorgaben und Standardrichtlinien der benutzten Frameworks. Auch dient die aus der Vorlesung bekannte Testhierarchie zur groben Orientierung bezüglich Form, Reihenfolge und Menge der Tests. Im Folgenden werden die einzelnen Stufen näher beschrieben.

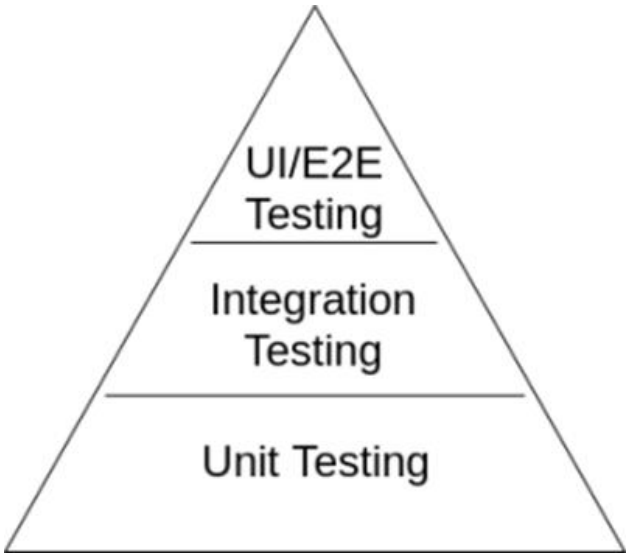

Abbildung 1 - Quelle: https://convincingbits.files.wordpress.com/2019/11/test_pyramid.png?w=300&h=265

1. Unit Tests

Jeder Entwickler sollte zu seinem Code Unit Tests schreiben, wobei die Methodik (Test first od. Test last) ihm selbst überlassen ist. Jedoch sind die Kriterien der Abnahme einzuhalten. Unit Tests werden mit dem unter dem Abschnitt Technisches genannten Framework geschrieben, wobei der angegebene Stil eingehalten werden *sollte*. Dabei wird zuerst der Name des Tests als alleinständig gut verständlicher String auf Englisch angegeben und danach die eigentliche Testmethode definiert. Mehrere Tests können in einer Klasse gebündelt werden, wobei die gewohnten Listener von JUnit (beforeClass, afterClass, usw.) genutzt werden können, um Testdaten oder Testhelfer zu initialisieren. Wo möglich ist hier auch Data Driven Testing einzusetzen, um unnötigen Code zu vermeiden und Tests übersichtlich zu gestalten.

Für Angular ist das standardmäßige enthaltene Testframework zu benutzen.

2. Integrationstests

Bei der Integration von einzelnen Modulen sollte die erfolgreiche Zusammenarbeit derselben getestet werden. Verantwortlich dafür sind die integrierenden Entwickler, die aber auch von den Entwicklern der einzelnen Untermodule unterstützt werden sollten. Für Tests des Servers “von außen” steht die Ktor-Integration von Kotest zur Verfügung, die das Simulieren von Client-Anfragen erlaubt.

3. UI-/ Systemtests

Hier wird meist aus der Perspektive des Endnutzers getestet, d. h. der Test simuliert diesen und interagiert so mit dem System. Es sind die einzelnen vereinbarten Anforderungen zu testen, wobei diese im Test aus User-Sicht durchgegangen werden. Angular bietet für UI-Tests ein eigenes E2E-Testtool an, das zu benutzen ist. Wie bei den Integrationstests sind hierfür alle beteiligten Entwickler verantwortlich.

4. Was / Wie wird getestet?

Die zu testende Funktionalität sollte sowohl nach Erfolg (“Login klappt”), als auch nach Misserfolg (“Falsches Passwort”, “User existiert nicht”) getestet werden. Dies ist gerade für UI- und E2E-Tests sehr wichtig. Tests an sich sollten immer nur eine Funktionalität überprüfen, das heißt mehrfache Assert-Statements sollten vermieden werden. Stattdessen können in einer Test-Klasse mehrere Tests hinzugefügt werden, die die unterschiedlichen Funktionalitäten prüfen.

5. Testdaten und -helfer

Generell sollte der eigentliche Code so modular wie möglich gehalten werden, um aufwändige Testhelfer zu vermeiden. Die Testdaten sollten den konkreten Anforderungen oder den User Stories (siehe Aufgabenblatt 6) entnommen werden. Falls dies nicht möglich oder passend ist, kann es sich anbieten, bei Unit oder Integrationstests mittels Data Driven Testing oder Property Testing eine (zufällige) Menge an Testdaten und -ergebnissen anzugeben bzw. diese zu generieren, anstatt nur einzelne Fälle zu testen.

6. Abnahme

Hiermit ist das Mergen in den master-Branch gemeint. Pull-Requests sollten von einem unabhängigen Entwickler überprüft und akzeptiert werden, wenn 1. alle Tests erfolgreich durchlaufen (wird automatisch von der CI-Umgebung überprüft) und 2. die Code Coverage über 75% ist. Diese Zahl gibt auch eine Orientierung bezüglich der gewünschten Menge der Tests.

7. Benennung und Dokumentation

Tests werden automatisch dokumentiert, weswegen aussagekräftige Testnamen äußerst wichtig sind. Die Ergebnisse werden automatisch auf tests.bm-games.net veröffentlicht. Ein Testname sollte sowohl die Ausgangsbedingung nennen (“Without a network connection”), als auch das erwartete Ergebnis (“my login should fail”).

8. Technisches

- **Framework:** Kotest. Für IntelliJ gibt es zusätzlich ein Plugin mit dem gleichen Namen.
- **Style:** Fun Spec
- **Angular**
- **Kotlin/Kotest-spezifische Guides:** Parameterized tests with Kotest, Data Driven Testing with Kotest