Einleitung

In diesem Dokument finden Sie unser Testkonzept über das Projekt, wie wir vorgehen werden. Wir absolvieren derzeit das Modul 450 "Applikationen testen" in der Schule. Für diese Projektarbeit hatten wir etwa 20 Unterrichtseinheiten zur Verfügung. Allerdings fehlen uns einige Lektionen aufgrund von Abwesenheit. Zu Beginn des Moduls haben wir hauptsächlich Aufgaben gelöst und uns auf die LB1 vorbereitet, eine theoretische Prüfung. Die Übungen waren auch nützlich für das Projekt. Das Projekt soll ein Webshop sein, in dem Kunden ihre gewünschten Schuhe kaufen können.

Features von der Applikation

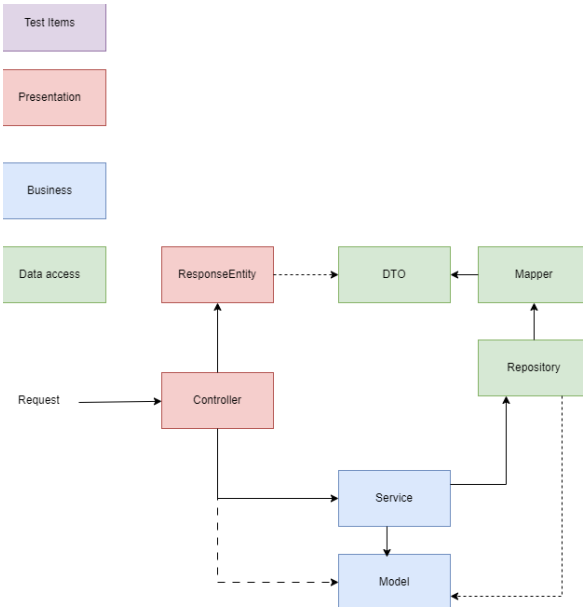
Diesen Abschnitt beschreibt die verschiedenen Features und Funktionen der Applikation eines Webshops, der sich auf den Verkauf von Schuhen spezialisiert hat. Diese Applikation bietet eine benutzerfreundliche und ansprechende Online-Shopping-Erfahrung für Schuhliebhaber und ermöglicht es ihnen, bequem von zu Hause Schuhe zu kaufen. Im Folgenden werden die wichtigsten Features der Applikation detailliert erläutert.

Produktkatalog: Die Applikation sollte eine übersichtliche Darstellung der Schuhe bieten, einschliesslich Bilder, Beschreibungen und Preise.

Warenkorb: Ein Warenkorb-Feature ermöglicht es den Benutzern, Schuhe auszuwählen und sie vor dem Kauf zu speichern. Der Warenkorb sollte die Gesamtsumme der ausgewählten Schuhe anzeigen und die Möglichkeit bieten, Artikel zu entfernen oder die Menge zu ändern.

Such- und Filterfunktion: Eine Such- und Filterfunktion ermöglicht es den Benutzern, nach bestimmten Schuhen zu suchen.

System Architektur



Request: Dies ist der Ausgangspunkt, wenn ein Client eine Anfrage an die Anwendung sendet. Die Anfrage kann verschiedene Informationen wie HTTP-Methoden (GET, POST, PUT, DELETE) enthalten.

Controller: Der Controller ist für die Entgegennahme der Anfrage verantwortlich. Er verarbeitet die Anfrage und entscheidet, welche Aktion ausgeführt werden soll.

Service: Der Service ist eine Zwischenschicht zwischen dem Controller und dem Repository. Hier werden die eigentliche Geschäftslogik und die Verarbeitung der Anfrage implementiert.

Repository: Das Repository ist eine Schicht, die den Zugriff auf die Datenbank oder andere Datenquellen abstrahiert.

Model: Das Model repräsentiert die Datenstruktur der Anwendung. Es kann Klassen oder Objekte umfassen, die die Daten in der Anwendung darstellen.

Dto (Data Transfer Object): Ein DTO ist ein Objekt, das dazu verwendet wird, Daten zwischen den Schichten der Anwendung zu übertragen.

Mapper: Der Mapper ist für die Konvertierung zwischen verschiedenen Datentypen verantwortlich.

Environmental Needs

Die Testumgebungen, die für unsere Teststrategie benötigt werden, variieren je nachdem, ob es sich um Backend- oder Frontend-Tests handelt.

Backend Unit Tests:

Hardware: Für Backend-Unit-Tests sind keine speziellen Hardwareanforderungen erforderlich. Sie können auf den meisten gängigen Entwicklungsumgebungen und Rechnern ausgeführt werden.

Software: Backend-Unit-Tests erfordern eine geeignete Entwicklungsumgebung, die die Programmiersprache und Frameworks unterstützt, in denen die Backend-Komponenten entwickelt wurden. Dies kann beispielsweise Node.js und entsprechende Testframeworks wie Mocha oder Jest für JavaScript-basierte Backend-Anwendungen umfassen.

Frontend Tests mit Cypress:

Hardware: Die Hardwareanforderungen für Frontend-Tests mit Cypress sind relativ bescheiden. Die meisten modernen Computer und Betriebssysteme sind in der Lage, Cypress auszuführen. Es ist jedoch wichtig sicherzustellen, dass die Hardware leistungsstark genug ist, um die Anwendung reibungslos auszuführen, da langsame Tests die Entwicklungszeiten verlängern können.

Software: Die Softwareanforderungen für Cypress-basierte Tests umfassen die Installation von Node.js, da Cypress als Node-Paket ausgeführt wird. Darüber hinaus müssen die erforderlichen Pakete und Abhängigkeiten installiert werden, um die Frontend-Anwendung, die getestet werden soll, auszuführen.

Test Features

- Schuhe zum Warenkorb hinzufügen (Geringfügige Fehler)
- Schuhe aus dem Warenkorb entfernen (Geringfügige Fehler)
- Navigations durch die vorhanden Seiten (Geringfügige Fehler)
- Suchfeld: Ein BEnutzer kann durch den Shop nach gewünschte Schuhe suchen (Geringfügige Fehler)

Features not to be tested

- Styling und Layout (Farben ect.)
- Statische Inhalte (Texte ect.)
- Navigations durch die vorhanden Seiten

Test Items

Unten befindet sich eine Liste der zu testenden Elemente und eine kurze Beschreibung, ob die Tests korrekt ausgeführt wurden:

1. **Beliebte Artikel auf der Startseite 4:** Die vier beliebtesten Artikel auf der Startseite. Die Tests wurden korrekt durchgeführt.
2. **Korrekte Navigation:** Die Tests überprüfen die Navigation, indem auf beliebige Schaltflächen geklickt wird, um auf eine Seite zu gelangen, und sicherstellen, dass die richtige Seite angezeigt wird. Die Tests wurden korrekt durchgeführt.
3. **Warenkorb:** Die Tests stellen sicher, dass der Warenkorb "keine Artikel im Warenkorb" anzeigt, wenn keine Artikel hinzugefügt wurden. Die Tests wurden korrekt durchgeführt.
4. **Produkt zum Warenkorb hinzufügen:** Überprüft, ob das Badge über dem Warenkorb mit der Anzahl der darin enthaltenen Artikel angezeigt wird und ob das Element im Warenkorb angezeigt wird. Die Tests wurden korrekt durchgeführt.
5. **Produkt aus dem Warenkorb entfernen:** Überprüft, ob ein Schuh aus dem Warenkorb gelöscht wird. Die Tests wurden korrekt durchgeführt.
6. **Dasselbe Produkt im Warenkorb hinzufügen:** Überprüft, ob das gleiche Produkt zweimal im Warenkorb erscheint, wenn die Schaltfläche "In den Warenkorb" erneut gedrückt wird. Die Tests wurden korrekt durchgeführt.
7. **Gesamtbetrag im Warenkorb anzeigen:** Überprüft, ob der Betrag im Warenkorb angezeigt wird. Die Tests wurden korrekt durchgeführt.
8. **Abbrechen des Checkouts:** Überprüft, ob der Abbruch des Checkouts keine Aktion durchgeführt. Die Tests wurden korrekt durchgeführt.
9. **Suchleiste - Suche nach "Air Jordan":** Überprüft, ob alle Produkte der Marke "Air Jordan" angezeigt werden, wenn nach "Nike" gesucht wird. Die Tests wurden korrekt durchgeführt.
10. **Suchleiste - Suche nach "test":** Ein falscher Input eingeben, es sollen keine Schuhe angezeigt werden.
11. **Ansichtsoptionen:** Überprüft, ob beim Klicken auf die Schaltfläche "Ansichtsoptionen" auf einem Produkt die detaillierte Produktseite angezeigt wird. Die Tests wurden korrekt durchgeführt.
12. **Reaktionsfähigkeit:** Nicht näher beschrieben, aber wahrscheinlich Tests, um sicherzustellen, dass die Anwendung auf verschiedenen Bildschirmgrössen und -geräten ordnungsgemäss reagiert.

Testing Tasks

Unsere Teststrategie umfasst mehrere Teststufen und Testarten, die darauf abzielen, die Qualität unserer Anwendung sicherzustellen. Die beschriebenen Teststufen und Testarten sind:

Unit Tests: Diese Tests konzentrieren sich auf die isolierte Überprüfung einzelner Komponenten oder Funktionen. Im Falle unserer Anwendung werden sowohl Backend- als auch Frontend-Unit-Tests durchgeführt. Diese Tests gewährleisten, dass einzelne Teile des Codes unabhängig voneinander und korrekt funktionieren.

Integrationstests: Obwohl wir nicht explizit von Integrationstests sprechen, gehen einige unserer Tests über das reine Einheitenlevel hinaus. Beispielsweise überprüfen wir das Hinzufügen von Schuhen zum Warenkorb und das Entfernen von Schuhen aus dem Warenkorb. Diese Tests können als Integrationstests betrachtet werden, da sie das reibungslose Zusammenspiel mehrerer Komponenten sicherstellen.

Funktionale Tests: Unsere funktionalen Tests konzentrieren sich darauf, die Anwendung auf einer höheren Ebene zu überprüfen. Dazu gehört das Navigieren durch vorhandene Seiten und die Verwendung des Suchfelds, um nach Schuhen zu suchen. Diese Tests stellen sicher, dass die Anwendung die erwarteten Funktionen korrekt ausführt.

Akzeptanztests: Obwohl sie in unserer Beschreibung nicht ausdrücklich erwähnt werden, könnten auch Akzeptanztests Teil unserer Teststrategie sein. Diese Tests überprüfen, ob die Anwendung den gesamten Anwendungsfall gemäss den definierten Anforderungen erfolgreich bewältigt.

Testvorgehen

Cypress:

- Installieren von Cypress und die erforderlichen Abhängigkeiten.
- Konfigurieren der Testumgebung in der `cypress.json`-Datei.
- Schreiben von Testfällen, um die Benutzeroberfläche zu überprüfen.
- Cypress-Tests ausführen, um sicherzustellen, dass die Benutzeroberfläche korrekt funktioniert.

Unit Tests:

- Installieren von Unit-Test-Framework (z. B. JUnit, Jest).
- Schreiben von Unit-Tests für einzelne Funktionen oder Komponenten der Anwendung.
- Verwenden Mocks und Testdaten, um Abhängigkeiten zu simulieren.
- Führen die Unit-Tests aus, um sicherzustellen, dass die Funktionen korrekt arbeiten.

Postman:

- Definieren von Testdaten und Testskripte in Postman, um API-Anfragen zu automatisieren und zu testen.
- Führen von Postman-Tests aus, um sicherzustellen, dass Ihre API ordnungsgemäss funktioniert.

Kriterien für erfolgreiche / nicht-erfolgreiche Tests

Kriterien für erfolgreiche Tests:

Erwartetes Verhalten: Der Test hat das erwartete Verhalten des getesteten Elements (z. B. einer Funktion, einer API oder einer Benutzeroberfläche) überprüft und das Ergebnis ist wie erwartet.

Vollständige Abdeckung: Der Test hat alle relevanten Szenarien und Funktionen abgedeckt und getestet, um sicherzustellen, dass keine kritischen Fehler übersehen wurden.

Nicht erfolgreiche Tests:

Unklare Fehlermeldungen: Die Fehlermeldungen oder Protokolle des Tests enthalten nicht genügend Informationen, um Fehler zu verstehen oder zu beheben.

Langsame Ausführung: Der Test führt inakzeptabel langsam aus und erhöht die Testlaufzeit unnötig.

Approach

Im Entwicklerteam setzen wir auf eine gründliche Testmethode, um sicherzustellen, dass unsere Software von höchster Qualität ist. Unser bevorzugter Ansatz besteht darin, Komponententests mithilfe von Unit Tests durchzuführen. Diese Tests konzentrieren sich darauf, einzelne Komponenten unserer Software, wie Funktionen, Module oder Klassen, isoliert voneinander zu überprüfen.

Ein wichtiger Bestandteil unserer Teststrategie ist die Anwendung von Test Driven Development (TDD). Dieser Ansatz erfordert, dass Tests vor der eigentlichen Implementierung des Codes geschrieben werden. Wir beginnen also nicht mit dem Schreiben von Code, sondern mit dem Definieren von Tests, die die erwarteten Verhaltensweisen unserer Software beschreiben. Erst nachdem die Tests definiert sind, erfolgt die Implementierung, um sicherzustellen, dass der Code diese Tests erfolgreich besteht.

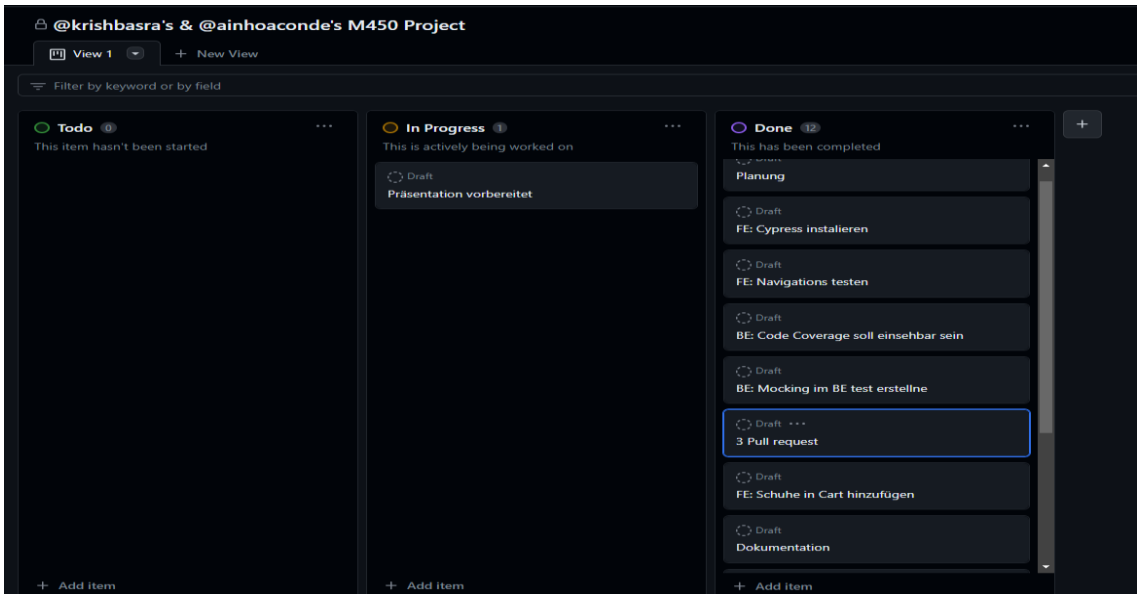
Diese Vorgehensweise bietet mehrere Vorteile. Sie fördert eine bessere Planung und strukturierte Entwicklung, da wir zuerst darüber nachdenken, wie sich unsere Software verhalten soll. Darüber hinaus ermöglicht sie eine kontinuierliche Validierung der Codequalität und minimiert die Einführung von Fehlern in den Code.

Planung

In der untere Tabelle befindet sich eine Grobe Plaung von meinen Projekt.

Datum	Aufgabe	Status
22.08.23	• Intro ins Modul erhalten • Projekt angefangen	
29.08.23	• Theorie KN 1 und 2 • Kompetenzen 1 und 2 • Abgeben von KN1 und 2	
05.09.23	• Theorie KN 3 • Kompetenzen 3 • Abgeben von KN 3	
12.09.23	• Theorie KN 4 • Kompetenzen 4	
19.09.23	• Kompetenzen 4 verbesen • KN 4 abgeben • LB1 vorbereiten	

26.09.23(LB1)	• Theorie KN 5 • Kompetenzen 5 • Testkonzept anfangen • LB 1 schreiben	
03.10.23	• Abgeben von KN 5 • Abgeben von Testkonzept • Besprechung wie weiter gehen mit Lehrer	
Herbstferien 09.10.23 - 20.10.23	Besprechung abwarten	
24.10.23	Krish (Backend) • Data sql Bugfix fixen • Test schreiben Ainhoa (Frontend) • cypress installieren • Präsentation schreiben • Dokumentation schrieben • Problem lösen mit Frontend	New York
31.10.23	Krish (Backend) • Test schreiben • Dokumentation schreiben • TDD anschauen • pipeline schreiben Ainhoa (Frontend) • cypress installieren • Cypress test schreiben • TDD anschauen • pipeline schreiben	New York
07.11.23	Präsentation	New York



Testumgebungen

Cypress: Cypress wird normalerweise in einer web-basierten Testumgebung ausgeführt, die auf Ihrem lokalen Entwicklungsrechner. In meinem Fall wird es Visual Studio Code

Unit Tests: Unit Tests werden in Ihrer Entwicklungsumgebung (IDE oder Texteditor) ausgeführt. In meinem Fall wird es Intellij sein.

Postman: Postman wird normalerweise in einer eigenen Desktop-Anwendung ausgeführt, die für das Erstellen und Ausführen von API-Tests entwickelt wurde.

Reflexion über TDD und Code Reviews

In der Welt von Test-driven Development (TDD) und Code-Reviews haben wir wertvolle Erfahrungen gesammelt. Diese sind entscheidend für die Codequalität, aber in einem verteilten Team, bei dem Ainhoa in New York war und Krish in der Schweiz waren, war die Herausforderungen spürbar.

TDD zwingt uns dazu, den Code und die Anforderungen sorgfältig zu durchdenken, bevor wir mit der Implementierung beginnen. Das führt zu besser strukturiertem Code, erfordert aber Geduld und Disziplin. Die räumliche Trennung erschwert jedoch Code-Reviews, die idealerweise von Angesicht zu Angesicht stattfinden sollten. Wir mussten auf Tools zurückgreifen, was manchmal zu Verzögerungen und Missverständnissen führte. Bei Ainhoa hat nicht mal, Teams funktioniert.

Wir haben uns schliesslich darauf geeinigt, wichtige Fragen und Kommentare über WhatsApp-Nachrichten auszutauschen, da weder Teams noch E-Mails zuverlässig funktionierten. Dies ermöglichte uns, die Kommunikation aufrechtzuerhalten und sicherzustellen, dass wir gemeinsam an

der Codequalität arbeiten. Dies zeigt, wie wichtig klare und effektive Kommunikation in verteilten Entwicklungsteams ist.

Reflexion

Die Aufgabenstellung des Projekts war herausfordernd und interessant. Ich fühlte mich in der Lage, die gestellten Aufgaben erfolgreich zu lösen und habe dabei viel gelernt. Es war eine wertvolle Erfahrung, meine Fähigkeiten in der Praxis anzuwenden und mich in verschiedenen Aspekten der Softwareentwicklung weiterzuentwickeln. Jedoch nur Aufgaben, wo man mit dem Backend zu tun hat, was schade ist.

Projektstruktur und Anforderungen:

Ein Punkt, der während des Projekts herausfordernd war, war die Klarheit in Bezug auf die Projektstruktur und die genauen Anforderungen. Ich hatte Schwierigkeiten zu verstehen, ob das Projekt sowohl das Frontend als auch das Backend umfasst oder nur eines von beiden. Eine genauere Spezifikation der Anforderungen hätte dazu beigetragen, Missverständnisse zu vermeiden und die Arbeit zu erleichtern. Wir als Team sind davon ausgegangen das es für das Backend zählt

Präsentation und Dokumentation:

Die Präsentation des Projekts und die Erstellung der Dokumentation verliefen zufriedenstellend. Ich konnte meine Arbeit klar und verständlich darstellen, und die Dokumentation war umfassend und informativ.

Herausforderungen durch die räumliche Trennung:

Eine der grössten Herausforderungen in diesem Projekt war die räumliche Trennung, insbesondere meine geografische Lage in New York. Dies führte zu Problemen, mit denen ich zuvor nicht konfrontiert war. Die Zeitunterschiede und die Schwierigkeiten bei der Kommunikation hatten Auswirkungen auf die Effizienz und die Geschwindigkeit der Projektarbeit. Leider hat dies dazu geführt, dass mehr Zeit in administrative Aufgaben investiert werden musste, anstatt sich auf die Projektinhalte zu konzentrieren.