HAW HAMBURG

Bachelorarbeit

Niklas Mollerus

Evaluation der Verwendung von Playwright für Keyword-Driven Testing

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Niklas Mollerus

Evaluation der Verwendung von Playwright für Keyword-Driven Testing

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 05. Dezember 2023

Niklas Mollerus

Thema der Arbeit

Evaluation der Verwendung von Playwright für Keyword-Driven Testing

Stichworte

Keyword-Driven Testing, Playwright

Kurzzusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Frage ob und wie gut sich das Playwright-Framework für das End-to-End Testen der Nutzerschnittstelle einer Webapplikation unter der Verwendung der Prinzipien des Keyword-Driven Testing eignet. Dabei werden in eine bestehende Anwendung Fehler eingebracht, welche nach Vorbild es Mutationstestens entworfen wurden. Diese werden dann mit Keyword-Tests, welche mit Playwright geschrieben wurden, gesucht.

Niklas Mollerus

Title of Thesis

Evaluation of the use of Playwright for keyword-driven testing

Keywords

Keyword-Driven Testing, Playwright

Abstract

This bachelor thesis addresses the question whether and how well the Playwright-Framework is suited to End-to-End testing of the user interface of a web application, using the principles of Keyword-Driven testing. For that an already existing application will be inserted with errors, modelled after examples of mutation testing. These errors will then be searched out by tests, using keywords written with Playwright.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Zielsetzung und Vorgehen . . . 1
1.2 Aufbau der Arbeit . . . 2
1.2.1 Güte von Tests . . . 2

2 Grundlagen **3**
2.1 Testing im Software-Lifecycle . . . 3
2.1.1 Wasserfall . . . 4
2.1.2 V-Modell . . . 5
2.1.3 Agile . . . 6
2.2 Was ist End-to-End Testing . . . 7
2.2.1 Verortung in Teststrategie (V-Modell) . . . 7
2.2.2 Verortung in Teststrategie (Agile) . . . 7
2.3 Keyword-Driven Testing . . . 8
2.3.1 Verschlagwortung . . . 8
2.3.2 Level und Layer . . . 9
2.3.3 Vorteile und Möglichkeiten . . . 10
2.3.4 Syntax . . . 11
2.4 Mutationstesten . . . 12
2.4.1 Anwendung in dieser Arbeit . . . 12
2.5 Automatisches Testen mit Playwright . . . 13
2.5.1 Keyword-Driven Testing in Playwright . . . 14
2.6 Das Demonstrationsobjekt . . . 14
2.6.1 Technischer Aufbau des Coffeemakers . . . 15
2.6.2 Komponentenbeschreibung . . . 15
2.6.3 Dokumentation der Seitenelemente . . . 17

3 Konzeptionierung der Fehler und Tests **19**
3.1 Art der zu verbauenden Fehler . . . 19

3.2 Art der zu verwendenden Tests 21
3.3 Regeln für zu verwendende Keywords 23
3.3.1 Level und Layer 23
3.3.2 Syntax 25

4 Realisierung der Tests und Fehler **26**
4.1 Spezifikation der Testfälle 26
4.2 Aufbau der Keyword-Bibliothek 31
4.2.1 Domain-Layer High-Level Keywords 31
4.2.2 Navigation-Layer High-Level Keywords 36
4.2.3 Navigation-Layer Intermediate-Level Keywords 38
4.2.4 Navigation-Layer Low-Level Keywords 46
4.2.5 Test-Interface-Layer Low-Level Keywords 49
4.3 Verbaute Fehler und erwartetes Fehlverhalten 54
4.3.1 Komponente "Rezepte einsehen" 54
4.3.2 Komponente "Rezept hinzufügen" 55
4.3.3 Komponente "Rezept brauen" 56
4.3.4 Komponente SZutaten einsehen" 57
4.3.5 Komponente SZutat nachfüllen" 58

5 Evaluation der Tests und Fehler **59**
5.1 Ergebnisse der Testausführungen 59
5.1.1 Sequenzielle Ausführung aller Tests 60
5.1.2 Einzelausführung aller Tests 63
5.2 Bewertung der Testfälle 66

6 Fazit/Ausblick **71**
6.1 Evaluation der Fragestellung 71
6.2 Probleme 72

Literaturverzeichnis **75**

A Anhang **77**
Selbstständigkeitserklärung 84

1 Einleitung

Diese Bachelorarbeit ist im Bereich des Softwaretestens angesiedelt.

Beim Testen von Software geht es um die Verifikation und Validierung des Projekts. Es wird also, abhängig von den Tests, geprüft, ob das Projekt die Wünsche des Kunden erfüllt und ob es möglichst fehlerfrei ist. Testen ist damit ein essenzieller Teil des Lebenszyklus von Software, worauf in Kapitel 2.1 weiter eingegangen wird.

Keyword-Driven Testing ist dabei ein "Testentwurfsverfahren", welches die Modularisierung bei der Spezifikation von Testfällen erleichtern soll
[Daigl und Rohner, 2022, S. 3].

Im Folgenden wird beschrieben welches Ziel diese Bachelorarbeit verfolgt, welche Frage sie versucht zu beantworten, mit welchen Mitteln dies geschieht und wie sie generell aufgebaut ist.

1.1 Zielsetzung und Vorgehen

Das Ziel dieser Bachelorarbeit ist es die Frage zu beantworten, ob sich das von Microsoft entwickelte Framework Playwright für das End-to-End Testing von Webapplikationen, unter Anwendung von Prinzipien des Keyword-Driven Testing, eignet. Dabei soll aufgezeigt werden was Keyword-Driven Testing ist, welche Vorteile es bietet, wie gut es sich im Playwright-Framework umsetzen lässt und ob die Kombination aus Playwright und Keyword-Driven Testing Probleme aufwirft.

Als primäre Quelle für Keyword-Driven Testing dient hierbei das Buch "Keyword-Driven Testing" [Daigl und Rohner, 2022].

Als Demonstrationsobjekt dient hierbei der von Christopher Dührkop in seiner Bachelorarbeit [Dührkop, 2021] erstellte Coffeemaker. Dieser stellt die Web-Oberfläche einer

potenziellen Smart-Home Kaffeemaschine dar. Über diese kann Kaffee zum einen per Knopfdruck gebraut werden, zum anderen bietet sie die Möglichkeit die Zutatenbestände einzusehen und nachzufüllen. Weiter eingegangen wird hierauf in Kapitel 2.6.

Um die Webanwendung letztlich mit keywordbasierten Tests zu überprüfen, wird diese zunächst gezielt mit Fehlern versetzt. Diese werden nach Vorbild des Mutationstestens gewählt.

Zum Erstellen der Tests ist es jedoch notwendig festzustellen, wie die Webinterface-Elemente auf Playwright-Funktionen gemappt werden müssen. Dies wird in Kapitel 2.6.3 erläutert.

1.2 Aufbau der Arbeit

Die Arbeit beschäftigt sich zunächst mit den Grundlagen. Dazu gehören Testing allgemein und End-to-End im Speziellen, sowie die verwendeten Prinzipen des Keyword-Driven Testing und Mutationstestens. Weiterhin werden das Playwright-Framework, dessen Verwendung in dieser Arbeit, sowie das verwendete Demonstrationsobjekt beschrieben. Danach geht es um die Konzeptionierung der Keyword-Driven Tests und der mit Hilfe von Mutationstesten erstellten Fehler, welche in das Demonstrationsobjekt eingebracht werden. Letztlich wird die Realisierung besagter Tests und Fehler betrachtet und die Tests werden auf ihre Güte hin bewertet.

1.2.1 Güte von Tests

Die Güte der Tests wird hier so definiert, dass sie zum einen den Prinzipen des Keyword-Driven Testing folgend (vgl. Kapitel 2.3) gut lesbar sind. Zum anderen sollten die Tests durch möglichst wenige äußere Umstände - wie andere fehlgeschlagene Tests - fehlschlagen.
Lesbarkeit liegt natürlich im Auge der Leserschaft. Die erwarteten und gefundenen Fehler aber lassen sich im Aufbau dieser Arbeit gut definieren. Weiterhin lässt sich beschreiben, ob sich Testfälle unterschiedlich verhalten, wenn sie allein, oder im Zusammenspiel mit anderen Tests, ausgeführt werden.

2 Grundlagen

Im Folgenden werden die der Bachelorarbeit zugrundeliegenden Themen kurz erläutert. Dazu gehören sowohl Begrifflichkeiten wie Keyword-Driven Testing und Mutationstesten als auch Testen im Allgemeinen und End-to-End Testen im Speziellen. Weiterhin gibt es einen kurzen Überblick über des Playwright-Framework und das Demonstrationsobjekt.

2.1 Testing im Software-Lifecycle

Als Software-Lifecycle versteht man allgemein die Gesamtheit aller Aktivitäten, die ein Softwareprojekt durchläuft. Diese Aktivitäten können wie folgt aussehen.

Der Lebenszyklus eines Softwareprojekts beginnt mit der grundlegenden Planung, also der Zielsetzung des Projekts. Darauf folgt eine Analyse der Anforderungen, aus welcher Spezifikationen hervorgehen, die dann für das Design der Architektur der Software verwendet werden. Die nächsten Schritte sind die Implementierung der Software in Code und das Testen der Komponenten der Software, allein und im Zusammenspiel miteinander. Letztlich folgt das Ausrollen der Software, die Integration und das Testen der Software im Zusammenspiel mit etwaigen anderen Systemen und die Wartung des aktiven Systems. Erweiterungen des Systems beginnen den Zyklus von neuem [Sommerville, 2018, S. 58].

Auch wenn das Testen ein grundsätzlicher Bestandteil dieses Lebenszyklus ist, variiert die Art, was wann getestet wird, abhängig von der Entwicklungsstrategie.

Im Folgenden wird der Vorgang des Testens in verschiedenen Modellen des Software-Lebenszyklus kurz zusammengefasst.

2.1.1 Wasserfall

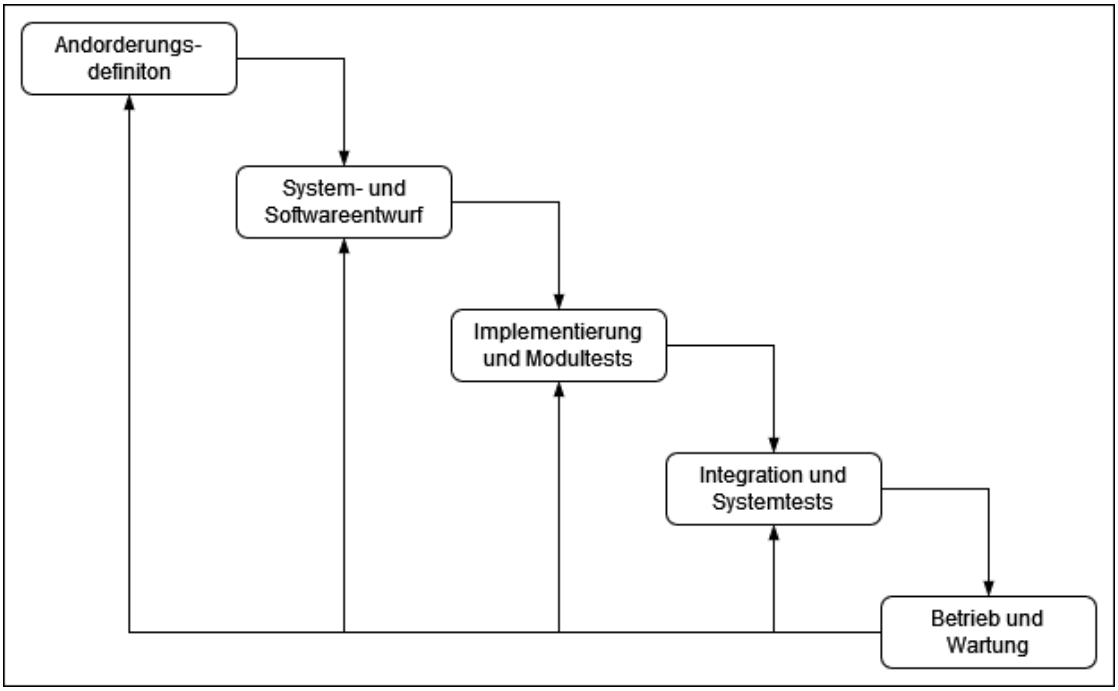

Abbildung 2.1: Das Wasserfallmodell nach [Sommerville, 2018, S. 58, Abbildung 2.1]

Aufgrund der geradlinigen Strukturierung des Wasserfallmodells, findet das Testen hier nur einmal, gegen Ende des Software-Lebenszyklus, statt. Das kann dazu führen, dass grundlegende Fehler, wie eine falsche Anforderungsdefinition, erst sehr spät in der Entwicklung gefunden werden, was bedeutet, dass große Teile eines Projekts vollständig überarbeitet werden müssen. Im schlimmsten Fall kommt dies einem Neubeginn des Projektes gleich [Sommerville, 2018, S. 57-60].

Die Tests selbst sind hier nicht weiter, oder nur grob, unterteilt. Dabei gäbe es Unittests, um die grundlegende Funktionalität der Komponenten zu sichern, Anwendungstests durch Entwickler*innen oder dedizierte testende Personen, sowie Abnahmetests durch Kunden.

2.1.2 V-Modell

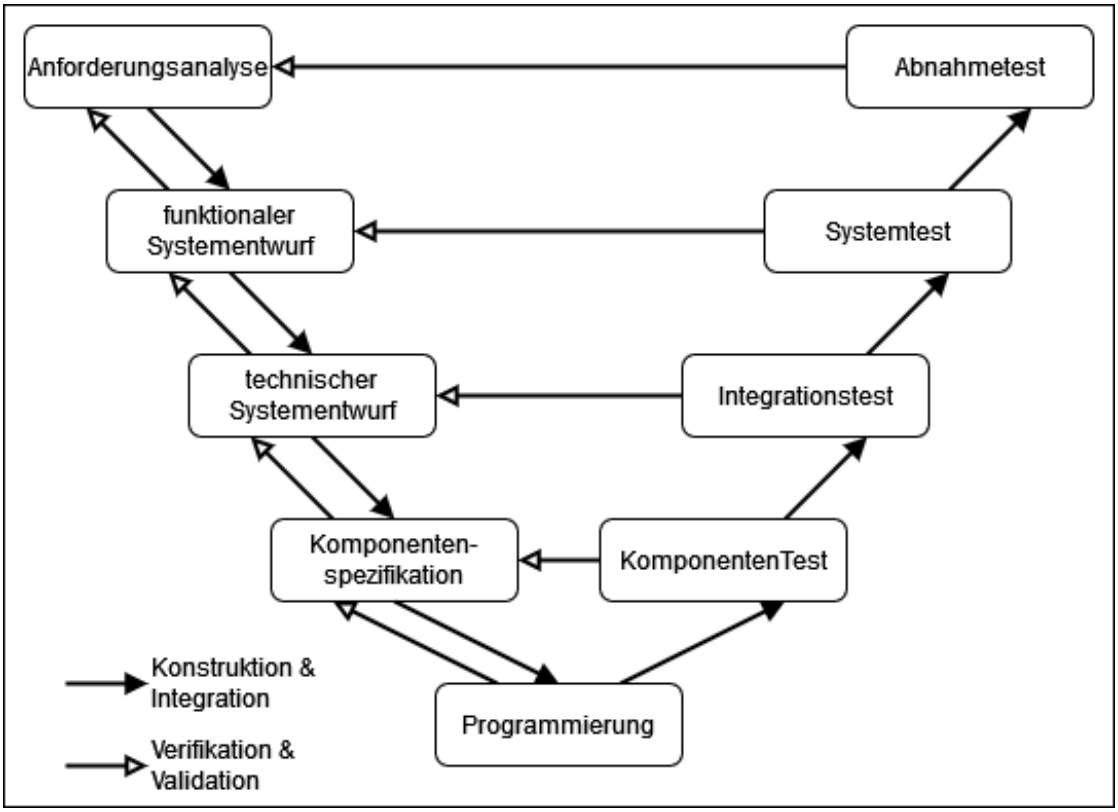

Abbildung 2.2: Allgemeines V-Modell nach [Spillner und Linz, 2019, S. 55, Abb. 3-2]

Das V-Modell ist eine erweiterte Art des Wasserfallmodells, bei dem jedem Entwicklungsschritt ein entsprechender Testschritt entgegengestellt wird. So wird zum Beispiel die Anforderungsanalyse durch Akzeptanztests und das technische Design durch Integrationstests validiert. Das bedeutet, sollten Integrationstests fehlschlagen, muss nicht das gesamte Projekt überdacht werden, sondern es reicht das technische Design zu überarbeiten [Sommerville, 2018, S. 71].

2.1.3 Agile

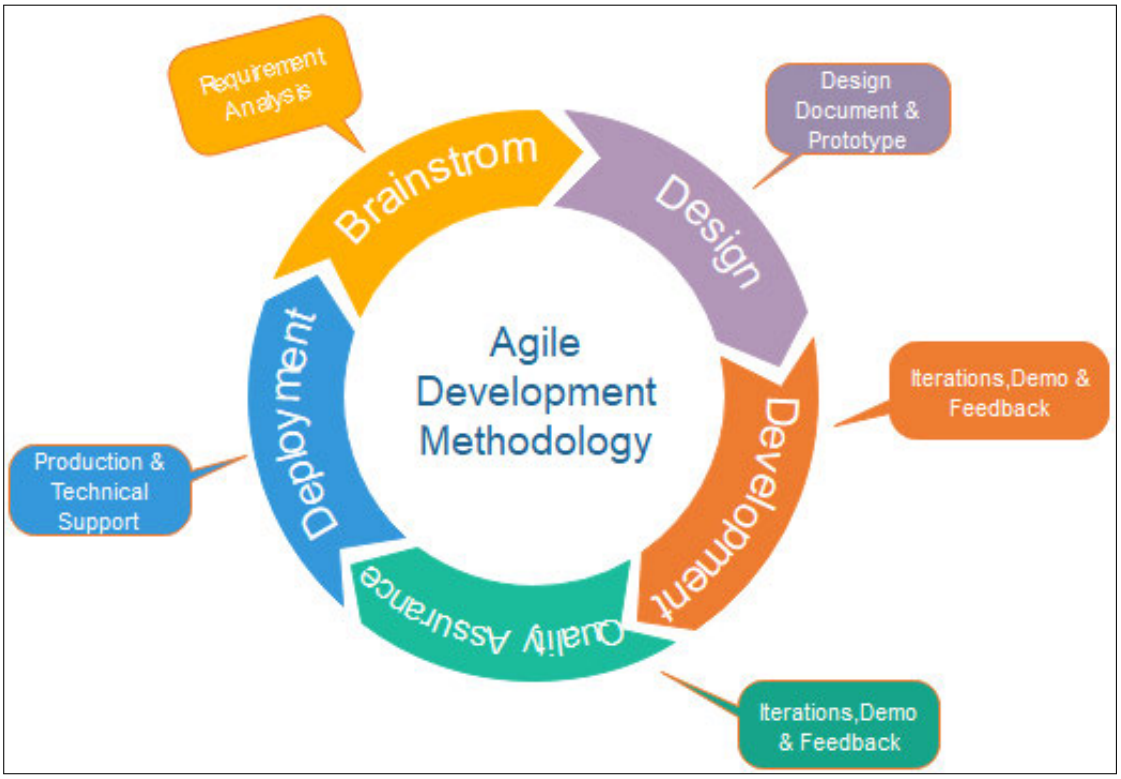

Abbildung 2.3: Agile Model [AgileCycleModel]

Die Agile-Softwareentwicklung basiert darauf, ein Projekt in mehreren kleineren Iterationen zu entwickeln, anstatt es, wie beispielsweise beim Wasserfallmodell, in einem Durchlauf, in einer Iteration, aller Phasen auf die Beine zu stellen. Dabei beinhaltet jede einzelne Iteration alle Schritte des Software-Lebenszyklus, von der Anforderungsanalyse bis zum Deployment, was zur Folge hat, dass auch die Teststrategie entsprechend ausgelegt sein muss. Dabei muss zwischen Regressionstests, also dem erneuten Ausführen bestehender Tests, und neuen Tests unterschieden werden. So müssen beispielsweise für eine Komponente, die bisher noch nicht ausgearbeitet war, nicht unbedingt neue Tests geschrieben werden, sofern keine neuen - bisher nicht von Tests abgedeckten - Funktionalitäten, ergänzt wurden. Es reicht in diesem Fall, wenn die bereits bestehenden Tests weiterhin, oder ab dann, bestanden werden. Sollten aber dem Programm neue Funktionalitäten hinzugefügt worden sein, so müssen neue Tests geschrieben werden, um diese zu testen [Spillner und Linz, 2019, S. 58-60].

2.2 Was ist End-to-End Testing

End-to-End Testing beschreibt eine Art von Softwaretests, bei denen eine vollständige Anwendung, inklusive all ihrer Komponenten, getestet wird [EndToEndTesting].

Dabei wird häufig aus Anwendersicht getestet. Im Falle einer Webanwendung können hierfür Frameworks wie Selenium, Cypress oder Playwright zur Automatisierung verwendet werden. Diese machen es möglich, einen Browser durch die Tests anzusteuern, indem Mausklicks und Tastatureingaben simuliert werden. Auf diese Weise können vollständige Geschäftsabläufe getestet werden. Ein Beispiel hier für wäre das Einloggen in einem Webshop, gefolgt vom Suchen eines Artikels, dem Hinzufügen des Artikels in den Warenkorb, dem Abschließen des Kaufs und dem Ausloggen. Überprüft wird dabei, ob alle involvierten Komponenten des Systems korrekt miteinander zusammenarbeiten.
Diese Tests erfassen allerdings nur indirekt die inneren Abläufe der Anwendung, da sie sie nur von außen ansteuern.

2.2.1 Verortung in Teststrategie (V-Modell)

Nach dem Testen der Software durch Komponenten und Integrationstests, welche sicherstellen, dass alle Komponenten an sich und miteinander arbeiten, wird das System in seiner Gesamtheit durch Systemtests, wie zum Beispiel End-to-End Tests, überprüft. Sollten die Tests fehlschlagen, muss gegebenenfalls das funktionale Design überarbeitet werden, was auch Änderungen im technischen Design und den Komponentenspezifikationen zur Folge haben kann. Nachdem alle notwendigen Änderungen vorgenommen wurden, kann das System erneut, beginnend bei den Komponenten, getestet werden.

2.2.2 Verortung in Teststrategie (Agile)

Da in der agilen Softwareentwicklung in mehreren Iterationen gearbeitet wird, sind nicht in jeder Iteration unbedingt alle Komponenten vertreten. Somit kann es sein, dass in einigen Iterationen keine End-to-End Tests notwendig sind. Sollte das System in einer gegebenen Iteration jedoch über entsprechend testbare Komponenten verfügen, müssen diese getestet werden. Sind diese Komponenten neu, müssen neue Tests dafür geschrieben werden, stammen sie aus einer vorherigen Iteration, müssen sie als Regressionstests erneut ausgeführt werden.

2.3 Keyword-Driven Testing

Keyword-Driven Testing ist eine - im Softwaretesten eingesetzte - Methode, um die Spezifikation von Testfällen zu modularisieren [Daigl und Rohner, 2022, S. 3]. Wichtig zu beachten ist jedoch, dass es sich dabei nicht um eine Testtechnik handelt, sondern nur um eine spezielle Art des Aufschreibens und Dokumentierens der Testfälle [Daigl und Rohner, 2022, S. 4].

Dabei werden Tests aus Bausteinen, sogenannten Keywords, zusammengesetzt. Jeder Baustein beschreibt dabei in Prosa, in wenigen Worten, mindestens eine Aktivität, die bei der Testausführung auszuführen ist. Auf technischer Ebene werden diese Keywords in Funktionen umgesetzt, die dann bei der Testausführung genutzt werden. Wobei ein Keyword allerdings auch andere Keywords referenzieren kann und somit nicht unbedingt den ganzen Umfang in einer neuen Funktion beschrieben haben muss.

2.3.1 Verschlagwortung

Beim Komprimieren von Anweisungen in Schlüssel- oder Schlagworte, muss unter anderem bedacht werden, dass ihre Bedeutung beim Lesen eindeutig ersichtlich wird [Daigl und Rohner, 2022, S. 27-31].

Dazu kommt, dass man angelegte Keywords schnell in einer Keyword-Bibliothek finden können muss. Das erleichtert nicht nur das Schreiben von Tests mit den Keywords, sondern verhindert auch, dass sich Dubletten - Keywords mit der gleichen Bedeutung - ansammeln.

Es ist also wichtig, klare Regeln für das Erstellen von Keywords aufzustellen. Solche Regeln umfassen die gewählte Sprache, Formulierungsschemata und Länge der zu erstellenden Keywords. Solch eine Regel könnte beispielsweise sein, dass Keywords grundsätzlich im englischen Imperativ zu verfassen sind. Es ist auch nützlich ein Glossar aufzubauen, um Missverständnisse durch Synonyme zu vermeiden.

Um die Keyword-Bibliothek sauber zu halten ist eine Kontrollinstanz denkbar [Daigl und Rohner, 2022, S. 89-91]. Diese, bestehend aus möglichst wenigen Mitarbeiter*innen, verwaltete dann das Hinzufügen neuer Keywords und wäre Ansprechpartner für alle keywordspezifischen Fragen. Weiterhin würde die Kontrollinstanz regelmäßig eine Keyword-Review durchführen, um dennoch entstandene Fehler zu beheben.

2.3.2 Level und Layer

Im Folgenden werden die Level und Layer, in die Keywords aufgeteilt werden, beschrieben [Daigl und Rohner, 2022, S. 33-41]. Dabei gehört jedes Keyword sowohl zu einem Layer, als auch zu einem Level.

Layer beschreiben dabei, wie technisch oder fachlich ein Keyword ist, ob es also fachlich ein Teil der Beschreibung eines Geschäftsprozesses ist oder technisch die Umsetzung eines Tests beschreibt.

Dabei gibt es drei Layer die, von fachlich zu technisch aufgezählt, Domain-Layer, Navigation-Layer und Test-Interface-Layer genannt werden. Ein Domain-Layer Keyword würde beispielsweise benutzt, um eine Testsequenz so zu beschreiben, dass sie von den Fachkräften, die sich gut mit den zu beschreibenden Geschäftsprozessen auskennen, leicht zu verstehen ist. Diese Beschreibung ist gut nutzbar für manuelle Tests, nicht aber für die Testautomatisierung. Keywords des Navigation-Layer sind technischer als die des Domain-Layer, sind aber, im Vergleich zu denen des Test-Interface-Layer, anwendungsspezifisch. Für die Testautomatisierung werden Keywords des Test-Interface-Layer benötigt. Diese sind spezifisch für das Test-Interface, beispielsweise eine grafische Oberfläche. Dort wäre beispielsweise "Press Button" ein mögliches Keyword in diesem Layer, da dieses keinerlei fachlichen Bezug hat.

Level andererseits beschreiben die Hierarchie der Keywords untereinander [Daigl und Rohner, 2022, S. 33].

High-Level Keywords sind die, die in der Beschreibung von Tests verwendet werden [Daigl und Rohner, 2022, S. 64-65]. Sie haben meist eine hohe Abstraktionsebene und sind eher fachlich. Für fachliche Tests ist es also sinnvoll, dass sie aus einem fachlichen Layer, zum Beispiel dem Domain-Layer, stammen, sowie es sinnvoll ist Keywords aus einem technischen Layer für technische Tests zu verwenden. Auch orientieren sie sich allgemein an der Wortwahl der Anwenderdomäne. Im High-Level sollte primär gefragt werden was mit einem Test getestet wird und nicht, wie dieses Ziel technisch zu erreichen ist.

Low-Level Keywords sind meist sehr technisch und beschreiben die Elemente des Test-Interface [Daigl und Rohner, 2022, S. 60-64]. Sie sind meist atomar, also nicht aus anderen Keywords zusammengesetzt.

Für manuelle Tests ist es wichtig zu wissen wie versiert die Testenden sind, um bestimmen zu können wie feingranular die Keywords gestaltet werden müssen. Testende Personen, die sich fachlich gut auskennen, benötigen für eine Login-Anweisung keine schrittweise Anleitung in den Keywords, während weniger versierte Testende feingranularere Anweisungen benötigt. Bei automatischen Tests hingegen müssen alle Spezifikationen immer möglichst genau sein.

Für komplexere Projekte ist es sinnvoll Intermediate-Level zu benutzen, um High- und Low-Level Keywords miteinander zu verbinden [Daigl und Rohner, 2022, S. 66-68].

Diese Intermediate-Level Keywords bestehen dann aus Low-Level Keywords und werden ihrerseits von High-Level Keywords verwendet. Sie helfen dabei die Komplexität der Technik abzukapseln, so dass sich ein Vorgang wie "Login User" im High-Level nicht mit dem Finden von Textfeldern und Drücken von Knöpfen befassen muss.

2.3.3 Vorteile und Möglichkeiten

Der initiale Aufwand Keyword-Driven Testing einzuführen ist vergleichsweise groß, weil damit ein zusätzlicher Schritt im Test-Design notwendig ist [Daigl und Rohner, 2022, S. 13-20].

Level und Layer müssen definiert werden, Regeln müssen aufgestellt und eingehalten werden und die entstehende Keyword-Bibliothek muss gewartet werden.

Dennoch bietet diese Methode viele Vorteile, wie nicht zuletzt eine vereinfachte und beschleunigte Testspezifikation durch bereitstehende Keywords.

Ein weiterer Vorteil von Keyword-Driven Testing ist die mit wohldefinierten Keywords einhergehende Klarheit. Jede Beschreibung eines Testfalls in Prosa hat Ungenauigkeiten durch Synonyme und Mehrdeutigkeiten von Wörtern. Das gleiche Problem haben auch Keywords, doch diese werden mit einer klaren Bedeutung definiert, welche dann gelernt werden kann. Dadurch verbessert sich auch die Les- und Wartbarkeit der Testspezifikationen.

Durch eine bessere Lesbarkeit der Tests verbessert sich auch deren Wartbarkeit. Die Wartbarkeit verbessert sich weiterhin durch die Modularisierung, die Keywords mit sich bringen. Während in einer Implementierung ohne Keywords bei einer Änderung von Testlogik, Testdaten oder Testimplementierung im Zweifelsfall mehrere Testskripte angepasst

werden müssen, müssen bei entsprechender Modularisierung nur noch wenige Keywords geändert werden. Dadurch skaliert der Wartungsaufwand nicht mehr mit der Anzahl der Tests, sondern mit der Anzahl der Keywords [Daigl und Rohner, 2022, S. 15-16].

Weiterhin ermöglichen Keywords es Expert*innen verschiedener Felder besser zusammenzuarbeiten. Während Fachexpert*innen mit Keywords Vorgänge beschreiben, können Testautomatisierer*innen, ohne fachliche Expertise, die Keywords als Tests umsetzen.

2.3.4 Syntax

Die Syntax der Keywords ist maßgeblich von der gewählten Sprache abhängig.

Ein allgemein guter Ansatz ist es, den Imperativ der gewählten Sprache zu verwenden; "open Browser" funktioniert hier ebenso gut wie "öffne Browser" [Daigl und Rohner, 2022, S. 76-79]. Der Imperativ bietet sich an, da Keywords Anweisungen an die Testinstanz, sei diese ein Mensch oder eine Maschine, geben.

Im Deutschen kann der Imperativ durch die dem Deutschen eigenen zusammengesetzten Wörter dennoch Probleme verursachen, da die Lesbarkeit der Keywords darunter leiden kann. So ist der Unterschied zwischen "Melde Benutzer an" und "Melde Benutzer ab" nicht schon beim ersten Wort erkennbar [Daigl und Rohner, 2022, S. 78].

Dies könnte behoben werden, in dem man die Grammatikregeln nicht allzu genau nimmt und in eine Sprache verfällt, wie man sie aus alten Point-and-Click-Adventures wie "Monkey Island" kennt [Daigl und Rohner, 2022, S. 79]. So würde aus "Melde Benutzer an" "Melde an Benutzer" und aus "Melde Benutzer ab" "Melde ab Benutzer". Doch derartige Änderungen mögen ihrerseits für einige Personen Lesbarkeitsprobleme mit sich bringen.

Die deutsche Sprache bietet jedoch mit dem Infinitiv eine alternative Möglichkeit Keywords zu schreiben [Daigl und Rohner, 2022, S. 80].

Dabei steht das Nomen des Keywords vorne an und somit im Fokus. Ein großer Vorteil des Infinitives ist, dass alle Keywords, die ein bestimmtes Objekt betreffen, zum Beispiel den Benutzer, alphabetisch immer zusammen stehen.

Dies kann das Finden bestehender Keywords vereinfachen und somit auch das Schreiben neuer Keywords erleichtern, oder sogar überflüssig machen, da bereits ein passendes existiert.

2.4 Mutationstesten

Mutationstesten ist, nach der Definition von Michael Wittner [Wittner], eine Form des Whitebox-Testens bei dem, durch gezielte Veränderungen des Codes die Qualität bestehender Tests geprüft werden kann.

Durch die Nutzung von Mutationstesten können effektfreie Tests aussortiert und fehlerhafte oder ungenügende Tests ausgebessert werden. Auch können so neue, bisher unentdeckte, Problemstellungen gefunden werden, für welche dann Tests geschrieben werden können.

Erreicht wird dies durch gezielte Veränderungen im Code. Beispielsweise können logische oder arithmetische Operatoren vertauscht oder verändert werden. Genauso ist es möglich numerische Werte abzuändern, in dem beispielsweise Standardwerte anders definiert werden, oder nach einem Argument oder Rückgabewert ein fester Wert aufaddiert wird. Dabei ist jedoch wichtig zu beachten, dass beim Einfügen der Fehler keine syntaktischen Fehler auftreten und der Code somit weiterhin ausführbar ist.

Die durch Mutationstesten entstehenden Codevarianten werden als "Mutanten" bezeichnet. Um feststellen zu können welcher Test welche der künstlich eingefügten Fehler findet, enthält jede der Mutanten genau einen der Fehler. Somit werden Seiteneffekte vermieden und die Qualität der Tests kann bewertet werden.

Dadurch lässt sich für jeden Test ein Mutationswert berechnen. Dieser ergibt sich als Prozentsatz der beseitigten Mutanten, also derjenigen Codeversionen, in denen der Test einen Fehler detektiert hat, im Vergleich zu allen getesteten Mutanten. Hat ein Test keinen der Fehler gefunden, er hat also einen Mutationswert von null, weist er potenziell Schwächen auf und sollte überprüft werden. Wenn eine Mutante mit keinem der bestehenden Tests gefunden wurde, sollten neue Tests dafür geschrieben werden.

2.4.1 Anwendung in dieser Arbeit

Da es in dieser Arbeit nicht um die Qualität bestehender Tests - sondern um das Schreiben neuer Tests mit Keyword-Driven Testing - geht, wird hier kein Mutationstesten im eigentlichen Sinne angewendet. Dennoch werden Prinzipien des Mutationstestens verwendet.

Mit Hilfe einer manuellen Codeanalyse werden Stellen gefunden, an denen Fehler, wie sie im Mutationstesten üblich sind, eingebaut werden können. Dazu gehören das Vertauschen von arithmetischen Operatoren wie Plus und Minus oder logischen wie Gleich und Ungleich. Weitere Möglichkeiten das Addieren von Konstanten auf bestehende Werte oder das Ergänzen von Zeichenketten um zusätzliche Zeichen.

Jeder der gewählten Fehler wird dann so im Code umgesetzt, dass dieser mit jeweils nur einem der Fehler ausgeführt werden kann. Diese nun fehlerhafte Codeversion liefert dann die Grundlage für die Tests, welche in der Bachelorarbeit geschrieben werden.

2.5 Automatisches Testen mit Playwright

Playwright ist ein von Microsoft entwickeltes und Anfang 2022 veröffentlichtes [PlaywrightGitHubReleases] End-to-End Testframework, welches in verschiedenen Programmiersprachen wie TypeScript, Python oder Java eingesetzt werden kann. Playwright unterstützt dabei alle Chromium basierten Browser wie Google Chrome oder Microsoft Edge, sowie Firefox und Apple Safari. In dieser Arbeit wird Playwright mit Version 1.38.1 in TypeScript verwendet. Als Editor dient mir hierbei Visual Studio Code mit Version 1.82.2.

Nach der Installation von Visual Studio Code kann darin die "Playwright Extension" [VSCodePlaywrightExtension] installiert werden. Mit dieser ist es dann möglich einem Projekt Playwright hinzuzufügen. Dies geschieht durch Ausführen des Befehls ">Test: Install Playwright".

Die Tests werden vom Playwright-Framework im Ordner "Tests" verwaltet (vgl. Abbildung A.1). Hier können in mehreren Dateien jeweils mehrere Tests geschrieben werden. Diese können dann über die "Testing-Kategorie", auf der linken Seite des Visual Studio Code Fensters (vgl. Abbildung A.2; rot markiert), gestartet werden. Möglich ist dies als Ausführung eines einzelnen Tests, als sequenzielle Ausführung aller Tests in einer Datei oder als sequenzielle Ausführung aller Tests in allen Dateien.

2.5.1 Keyword-Driven Testing in Playwright

Zu meiner Umsetzung von Keyword-Driven Testing mit dem Playwright Framework in TypeScript, werden die Keywords als Funktionen in Dokumenten definiert, von denen aus diese in die Tests importieren werden können. Die Keywords haben dabei sprechende Namen, nach der Namenskonvention von TypeScript [TypeScriptStyleGuide], die ihre Funktion beschreiben, wie zum Beispiel "pressButtonByName(name)".

Um die Wartbarkeit zu verbessern werden Funktionalitäten, die in mehreren Keywords auftreten, in Funktionen umgesetzt, welche nicht exportiert werden.

Um die Hierarchien der Keyword-Level, sowie die Unterschiede der Keyword-Layer, darstellen zu können werden diese in verschiedenen Ordner und Dateien umgesetzt. Dabei hat jeder Layer einen eigenen Ordner, in dem jeweils eine Datei für jedes Level liegt.

Playwright liefert hierbei die Low-Level Keywords des Test-Interface-Layers. Diese bedienen die Seitenelemente der Web-Oberfläche des Testobjektes.

Sie werden von Intermediate- oder High-Level Keywords verwendet, die ihrerseits von anderen High-Level Keywords oder den Testfällen genutzt werden (vgl. Kapitel 2.3.2).

2.6 Das Demonstrationsobjekt

In seiner Bachelorarbeit [Dührkop, 2021] beschäftigt sich Christopher Dührkop mit der Automatisierung der Tests eines schichtenübergreifenden Systems, mit Hilfe von Cypress als Testautomatisierungsframework. Sein Ziel war es festzustellen, wie gut sich Cypress eignet, um ein schichtenübergreifendes System zu testen und ob es für eine teamübergreifende Nutzung empfohlen werden kann. Dafür wurde zuerst eine, sich am Schichtenmodell orientierende, Smart-Home-Anwendung als Demonstrationsobjekt entwickelt, welches darauffolgend mit automatisierten Tests im Cypress-Framework getestet wurde.

In dieser Bachelorarbeit wird das von Christopher Dührkop kreierte Demonstrationsobjekt verwendet. Die Datenbank des Demonstrationsobjekts wird hier nicht mit spezifischen Tests, wie zum Beispiel für Daten-Integrität, geprüft. Andere Resultate seiner Arbeit, sowie Schlüsse, die er daraus zog, spielen hier keine Rolle. Weiterhin spielen das von Dührkop verwendete Cypress-Framework und die von ihm geschriebenen Tests hier

keine Rolle. Im Folgenden wird das Demonstrationsobjekt mit seinem technischen Aufbau und seinen funktionalen Komponenten beschrieben. Auch wird beleuchtet, wie sich die auf der aus dem Demonstrationsobjekt resultierenden Website angezeigten Elemente, identifizieren lassen.

Abbildung A.3 zeigt das von Christopher Dührkop für seinen Coffeemaker verwendete Klassendiagramm. Darin, in Rot umrandet, sind die Klassen der Anwendung markiert, die hier keine Bewandtnis haben.

Um das Demonstrationsobjekt - in seiner für diese Arbeit angepassten Version - zu installieren, ist der Code von GitHub [GitHubCoffeemaker, 2023] zu downloaden und den Anweisungen in der enthaltenden Datei "README.md" zu folgen.

2.6.1 Technischer Aufbau des Coffeemakers

Der von Christopher Dührkop geschriebene Coffeemaker verwendet das Angular Framework. Darin wird mit JavaScript oder, wie hier mit TypeScript, nicht nur die Logik des Projekts gecodet, sondern auch, mit entsprechenden Befehlen, die HTML-Repräsentation des Frontends gestaltet. Für die Datenhaltung wurde mit "json-server" die Datenbank gestubbt. "json-server" nutzt ein JSON-Dokument, um die gewünschten Daten abzulegen.

2.6.2 Komponentenbeschreibung

Der von Christopher Dührkop geschriebene Coffeemaker lässt sich in zwei große Bereiche unterteilen. Im ersten Bereich geht es um die Rezepte. Er enthält als Komponenten das Hinzufügen neuer, sowie das Einsehen und Brauen bestehender Rezepte.

Zum Hinzufügen gibt es im oberen Bereich der "Recipes"-Unterseite eine Eingabemaske, welche sowohl ein Textfeld für den Namen eines neuen Rezepts, als auch ein Nummernfeld für jede Zutat beinhaltet (vgl. Abbildung A.4). Sobald ein Name von mindestens 2 bis maximal 20 Zeichen Länge, sowie für mindestens eines der Zutatenfelder ein gültiger Wert eingegeben wurde, kann durch einen Klick auf "Create new Recipe" ein neues Rezept angelegt werden. Nach Erstellen des Rezepts sollte dieses im unteren Bereich der "Recipes"-Unterseite zu sehen sein (vgl. Abbildung A.5). Durch einen Klick auf den "Brew"-Knopf eines jeden der dort angezeigten Rezepte wird der entsprechende

Brauvorgang eingeleitet. Dabei wird überprüft, ob die vorrätigen Zutaten für das Brauen ausreichen. Ist das der Fall, wird das Rezept gebraut, andernfalls wird eine Fehlermeldung angezeigt [Dührkop, 2021, S. 54-56].

Auch wenn die Rezepte in einer realen Anwendung vermutlich ebenfalls in einer Datenbank abgelegt wären, würde an dieser Stelle dennoch die physische Kaffeemaschine angesteuert, um das Rezept letztlich zu brauen.

Der zweite Bereich befasst sich mit den Zutaten. Funktionell umfasst er das Einsehen der Füllstände, sowie das Nachfüllen der Zutaten.

Dafür gibt es die "Ingredients"-Unterseite.

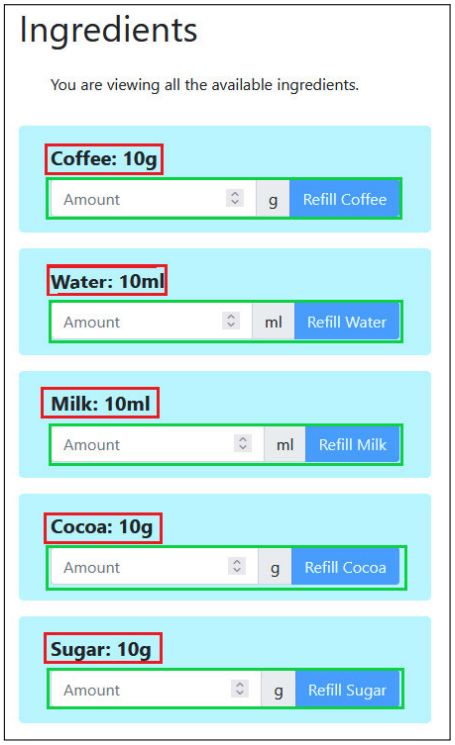

Abbildung 2.4: "Ingredients"-Komponente des Coffeemakers

Hier wird zu jeder Zutat der entsprechende Füllstand angezeigt (vgl. Abbildung 2.4; rot markiert). Ebenfalls gibt es zu jeder Zutat ein Eingabefeld - mit einem "Refill"Knopf - um diese nachzufüllen (vgl. Abbildung 2.4; grün markiert).

Dazu muss ein Wert größer 0 angegeben werden der, in Summe mit dem aktuellen Füllstand, das Maximum von 1500 nicht überschreitet. Folgend kann der entsprechende "Refill"-Knopf betätigt werden, um den Nachfüllvorgang abzuschließen [Dührkop, 2021, S. 59-60].

In einer realen Kaffeemaschine würden die Füllstände vermutlich nicht in einer Datenbank abgelegt, sondern über Sensoren aus den entsprechenden Lagereinheiten ausgelesen werden. Das würde dann natürlich auch die Funktion des Nachfüllens ersetzen, da für eine reale Kaffeemaschine realer Kaffee benötigt wird.

2.6.3 Dokumentation der Seitenelemente

Zum Testen einer Website müssen die, im folgenden als Seitenelemente benannten, Bestandteile der Website, wie Knöpfe, erwartete Alerts und Pop-ups oder Eingabefelder, bekannt sein.

Nur wenn klar ist, wie man in den Testfällen relevante Seitenelemente findet, können diese gezielt angesteuert werden. In einem normalen Entwicklungsprojekt gibt es Konventionen für die Vergabe von Namen und die Dokumentation des Projektes, worüber sich die Seitenelemente während des Schreibens der Tests identifizieren lassen.

Im Falle des hier genutzten Coffeemakers existiert keine konkrete Dokumentation dieser Art. Da es dennoch notwendig ist, die Seitenelemente eindeutig identifizieren zu können, müssen sie nachträglich dokumentiert werden.

Als mögliche Quellen stellen sich dabei der Code und die Website dar.
Im Code sollten sich HTML-Dateien finden, in denen der Aufbau der Komponenten der Website definiert ist. Somit sollten sich dort auch Namen und IDs, oder zumindest dafür vorgesehene Variablen, finden lassen.

Um aus der Website selbst die notwendigen Informationen zu extrahieren, bieten sich Werkzeuge wie der "Page Inspector" von Firefox an.
Dieser lässt sich entweder über die "Werkzeuge für Web-Entwickler" im "Extras"-Menü des Browsers oder über den Punkt "Untersuchen" im Rechtsklickmenü auf der Website öffnen.

Im Inspektor wird der, der aktuellen Seite zugrunde liegende, HTML-Code dargestellt. Durch das Bewegen der Maus über einzelne Zeilen wird der betroffene Bereich auf der Website farblich hervorgehoben. Somit ist es leicht zu identifizieren, welche Zeilen des HTML-Codes der Seite - und damit verbunden welche Namen und IDs - zu welchem Seitenelement gehören.

Eine andere - für den Gebrauch mit Playwright sehr nützliche - Möglichkeit ist es, die von der Playwright Extension in Visual Studio Code [VSCodePlaywrightExtension] gelieferte "Pick locator"-Funktion zu nutzen.

Dazu muss in einem Playwright Projekt in Visual Studio Code - wie in Kapitel 2.5 beschrieben - in der Kategorie "Testing", links unter dem Dateiexplorer (vgl. A.2; rot markiert), unter "Playwright" "Pick locator" ausgewählt werden (vgl. A.2; grün markiert).

Dadurch öffnet sich ein Browserfenster, in dem man auf die gewünschte Seite navigieren kann. Dort wird nun, wenn man den Mauszeiger über ein Seitenelement bewegt, der Befehl angezeigt, mit dem man in Playwright das entsprechende Element finden kann.

Um mein Vorgehen beim Identifizieren der Seitenelemente des Coffeemakers zu strukturieren, habe ich zunächst Prozesse - wie in Kapitel 2.6.2 beschrieben - definiert, die zusammengenommen alle Seitenelemente bedienen. Dann habe ich mit dem "Page Inspector" des Browsers die Namen oder IDs der entsprechenden Elemente bestimmt und mit einem kurzen Python-Skript und dem Playwright-Framework, diese Elemente angesteuert. Das hat mir erlaubt, mein Verständnis über den HTML-Code der Seite und das Playwright-Framework, zu vertiefen.

Mit meinem vertieften Wissen habe ich darauffolgend die essenziellen Zeilen, also nicht nur die ID oder den Namen, des HTML-Codes der Website nach ihren Komponenten sortiert und dann extrahiert. Weiterhin konnte ich nun nach diesen Zeilen, oder Teilen davon, im Code suchen und so die entsprechenden Stellen dort finden.

3 Konzeptionierung der Fehler und Tests

Dieses Kapitel beschäftigt sich mit den Konzepten die den Fehlern und Tests zugrunde liegen, die später um das Testobjekt herum aufgebaut werden.

Es wird beschrieben wie die in Kapitel 2.4.1 angesprochene Codeanalyse abläuft und welche Fehlerkategorien daraus resultieren. Weiterhin wird definiert welche Tests welche Komponenten des Coffeemakers testen sollen und wie die Syntax der, in Kapitel 2.3 angesprochenen, Keywords aussehen soll.

Dabei sollen die Tests jede Komponente des Coffeemakers betrachten und in ihrer Funktionalität testen. Sie werden entsprechend Komponente für Komponente aufgebaut.

Um eine Basis für alle Tests und Fehler zu schaffen wird im Folgenden angenommen, dass die von Christopher Dührkop gegebene Version des Coffeemakers fehlerfrei ist.

3.1 Art der zu verbauenden Fehler

Um systematisch Fehler zu generieren, werden Teile der Methodik des Mutationstestens verwendet (siehe Kapitel 2.4.1).

Der Ausgangspunkt für die Suche nach Punkten im Programmcode, an denen Fehler eingebracht werden können, ist eine Tabelle, die für jedes Feature der Anwendung die Komponenten und die Funktionen, in denen das Feature im Code steht, enthält (siehe Abbildung A.6).

In dieser sind die folgenden fünf Spalten zu sehen.

Feature enthält den hier für eine Komponente oder Funktionalität gewählten Namen.

Methodenname enthält den Namen der Methode im Code, sowie, durch ein oder mehrere ">" gekennzeichnet, welche Methoden darin aufgerufen werden.

Datei und Zeile enthält für jeden Methodennamen die Datei und Zeile, im Format "Datei:Zeile", in der die entsprechende Methode im Code zu finden ist.

Mögliche einbaubare Fehler enthält kurze Beschreibungen möglicher Fehler, die sich in die jeweilige Methode einbauen lassen könnten.

Erwarteter Fehler enthält für jeden der möglichen einzubauenden Fehler die erwartete Auswirkung auf den Code.

Jeder generierte Fehler wird auf eine Art und Weise umgesetzt, die es erlaubt ihn, unabhängig von den anderen Fehlern, zu aktivieren oder zu deaktivieren (vgl. Kapitel 2.4).

Letztlich soll für jede Komponente mindestens ein Fehler eingebaut werden, welcher deren Funkion möglichst vollständig abgedeckt.

In den Kategorien "Rezepte sehen" und "Zutaten einsehen" bietet der Code jeweils nur die Möglichkeit, ohne größere Änderungen vorzunehmen, die verwendeten URL-Konstanten abzuändern, wodurch die entsprechende Datenbankanbindung unterbrochen wird.

Die Kategorie "Rezept hinzufügen" kann durch das Entfernen der "Reload-Funktion" sabotiert werden. Da dieser Fehler nicht unbedingt direkt auffällt, scheint es besser zu sein, die dem Rezept übergebenen Werte zu vertauschen.

Die Kategorie "Rezept brauen" bietet in den Methoden "brewRecipe" und "enoughIngredientsAvailable" Möglichkeiten das Brauverhalten zu sabotieren. Durch das Austauschen von Ungleich zu Gleich in ersterer wird das Brauverhalten invertiert, ergo wird alles

braubare nicht gebraut und alles nicht braubare gebraut. Das Ändern des Standardrückgabewertes in der zweiten Methode sorgt wiederum dafür, dass in "brewRecipe" niemals der Fall eintreten kann, dass das Brauen erfolgreich ist.

Andere Möglichkeiten in dieser Komponente wären es die Methode "useIngredient" so zu ändern das sie, statt Zutaten zu verbrauchen, Zutaten nachfüllt, oder die Methode "useNeededIngredients" dahingehend zu ändern, dass sie nicht mehr die gewünschten, sondern um beispielsweise 1 abgeänderten, Zutatenwerte verwendet.

3.2 Art der zu verwendenden Tests

Da es sich bei den hier zu konzeptionierenden Tests um End-to-End Tests handelt, sollten diese alle Komponenten des Coffeemakers, welche über das Webinterface aufgerufen werden können, (siehe Kapitel 3.1) abdecken. Dabei kann man sich an den "CRUD"-Operationen, wie von Christopher Dührkop in seiner Bachelorarbeit definiert (vgl. [Dührkop, 2021, S. 54-56]), orientieren. Dieses Akronym steht für "Create", "Read", "Update" und "Delete". All diese Bereiche der zu testenden Software sollten von den Tests abgedeckt sein.

In einer sequenziellen Ausführung aller Testfälle muss jeder Test mit dem Zustand des Systems arbeiten, der vom vorherigen hinterlassen wurde. Dies ist der Fall, da die Anwendung über ihr Webinterface getestet wird und somit nicht zwischen der Ausführung zweiter Testfälle zurückgesetzt werden kann.

Dabei werden die Tests entweder als vollständige Testsuite, mit allen 11 Tests, oder alleine Ausgeführt.

Rezepte sehen
Für diese Komponente sollte getestet werden, ob alle Standard-Rezepte, also "Americano", "Espresso", "Latte Macchiato" und "Hot Chocolate", erwartungsgemäß anzeigt werden.

Dies fällt in die "Read"-Kategorie der "CRUD"-Operationen.

Rezept brauen
Dies kann weiterhin in die Komponente "Rezept brauen" übertragen werden, wo getestet werden sollte, ob der "Brew"-Knopf bei Betätigung die erwartete Reaktion liefert. Diese Reaktion ist allerdings abhängig von den verfügbaren Zutaten, weshalb es für die Tests für das "Rezept brauen" Feature notwendig ist, die Zutaten zunächst auf einen passenden Füllstand zu bringen. Entsprechend sollte jeder Knopf auch ein Mal mit und ohne ausreichende Zutaten getestet werden.

Dies fällt in die "Update"-Kategorie der "CRUD"-Operationen.

Rezept hinzufügen
Diese Komponente sollte getestet werden, indem ein neues Rezept angelegt und danach auf seine Sichtbarkeit und Funktionalität (siehe oben) überprüft wird. Weiterhin sollte geprüft werden, dass das System keine Werte für Zutaten akzeptiert, die außerhalb der zulässigen Werte liegen, also kleiner als null oder größer als 1000 sind.

Dies fällt in die "Create"-Kategorie der "CRUD"-Operationen.

Zutaten einsehen
Für diese Komponente sollte überprüft werden, ob alle Zutaten mit ihren entsprechenden Maßeinheiten angezeigt werden, was in die "Read"-Kategorie der "CRUD"-Operationen fällt.

Zutaten nachfüllen
Für diese Komponente muss für jede Zutat überprüft werden, ob die Menge nach dem Nachfüllen um den erwarteten Betrag erhöht wurde. Ebenso sollte überprüft werden, ob für jede Zutat die Maximal- und Minimalwerte, hier 0 und 1500 Milliliter oder Gramm, überschritten werden können und es ist sicherzustellen, dass keine negativen Werte akzeptiert werden.

Was in die "Update"-Kategorie der "CRUD"-Operationen fällt.

Zusätzlich können vollständige Vorgänge getestet werden. Ein Beispiel hierfür wäre ein wegen fehlender Zutaten missglückter Brauversuch. In diesem Fall würde maximal eine der fehlenden Zutaten als fehlend, durch die Fehlermeldung, angezeigt. Dann würde der

nächste Schritt des Vorgangs die als fehlend gemeldete Zutat ergänzen und einen weiteren Brauversuch starten. Dieser Zyklus müsste sich wiederholen bis alle Zutaten bis zur Füllgrenze aufgefüllt wurden. Erst wenn der Brauversuch auch dann noch fehlschlägt, kann der Test als fehlgeschlagen angesehen werden. Ein derartiger Testablauf würde Teile der "Read"- und "Update"-Kategorie der "CRUD"-Operationen enthalten.

Grundsätzlich sollte die als fehlerfrei angesehene Version des Testobjekts ebenfalls mit allen Tests getestet werden, um Ausgangswerte für den Vergleich mit den, gezielt mit Fehlern versehenen, Versionen des Testobjekts zu liefern.

Da das Feature zum Entfernen eines Rezeptes nicht implementiert wurde, kann die "Delete"-Kategorie nicht getestet werden.

3.3 Regeln für zu verwendende Keywords

Im Folgenden soll beschrieben werden, wie die für die Tests verwendeten Keywords zu gestalten sind (vgl. Kapitel 2.3). Es wird kurz etabliert aus welchen Layern und Levels sie stammen und wie sie syntaktisch aufgebaut sein werden.

3.3.1 Level und Layer

Beim Testobjekt, dem Coffeemaker, handelt es sich um eine Controller Software, die über ein Webinterface angesteuert und damit auch getestet wird.

Playwright ist ein Framework mit dem eben solche Webinterfaces angesteuert werden können. Die Low-Level Test-Interface-Layer Keywords können daher direkt in Playwright-Funktionen übersetzt werden. Aus diesem Grund wird Playwright hier als Test-Interface-Layer verwendet.

Auf dem Domain-Layer wird die Businesslogik der Anwendung beschrieben. Hier bezieht sich der Domain-Layer also auf potenzielle Coffeemaker im Allgemeinen. Dementsprechend wird sich größtenteils auf Rezepte und Zutaten - hier "Recipes" und "Ingredients" - bezogen.

Der Navigation-Layer beschäftigt sich mit Christopher Dührkops Coffeemaker im Speziellen. Hier ist die Rede von den, für diese Anwendung spezifischen, Rezepten und Zutaten.

Der Test-Interface-Layer beschäftigt sich mit der technischen Umsetzung der Web-Oberfläche. Hier erfolgt der Zugriff auf bestimmte Seitenelemente, wie Knöpfe oder Eingabefelder.

Abbildung 3.1: "Ingredients"-Komponente des Coffeemakers

Als Beispiel dazu Abbildung 3.1.
Darin sind in Blau die gesamte Komponente als Bezug für den Domain-Layer, in Grün die Umsetzung einer Zutat für den Navigation-Layer, sowie in Rot der entsprechende "Refill"-Knopf als Bezug für den Test-Interface-Layer markiert.

Um die hierarchischen Abhängigkeiten vorzudefinieren sei hier gesagt das, nach Möglichkeit, High-Level Domain-Layer Keywords nur auf Intermediate-Level Navigation-Layer Keywords, High-Level Navigation-Layer Keywords nur auf Low-Level Navigation-Layer Keywords und Low- sowie Intermediate-Level Navigation-Layer Keywords nur auf die von Playwright gegebenen Low-Level Test-Interface-Layer Keywords zugreifen.

Da der Coffeemaker kein besonders kompliziertes Projekt ist, bietet es sich nicht an, ein Intermediate-Level einzuziehen. Dennoch ist es für die Veranschaulichung in dieser Bachelorarbeit sinnvoll, diesen zusätzlichen Schritt zu gehen.

Low-Level Keywords des Navigation-Layer sollten hier verwendet werden, um die von Playwright auf dem Test-Interface-Layer liegenden Keywords, zu kapseln. Da es hier um automatisierte und nicht um manuelle Tests geht, muss die Spezifikation der Low-Level Keywords sehr genau sein. Ein Beispiel dafür könnte hier etwa sein, ein Keyword zu haben, dass eine bestimmte Art Seitenelement, wie die Füllstandsanzeige, findet und zurückgibt.

Intermediate-Level Keywords werden hier, soweit möglich, als Zwischenschritt zwischen High- und Low-Level verwendet. Beispielsweise könnte ein Low-Level Keyword für das allgemeine Drücken von Knöpfen existieren.
Dieses würde dann von einem Intermediate-Level Keyword verwendet, welches das Drücken

eines speziellen Knopfes beschreibt. Dieses wiederum würde dann von einem High-Level Keyword genutzt, um eben diesen speziellen Knopf auszulösen. Um das Beispiel aus dem letzten Absatz fortzusetzen, könnte ein Intermediate-Level Keyword Daten aus der Füllstandsanzeige entnehmen und so den aktuellen Füllstand einer Zutat liefern.

High-Level Keywords sollten größtenteils auf dem Domain-Layer angesiedelt sein (vgl. Kapitel 2.3.2). Sie beschreiben sachlich Vorgänge auf der Website der Applikation und werden direkt in den Testfällen verwendet. Beispiele hierfür wären das Anlegen von Rezepten, ebenso wie das Eintragen von Werten in eine Eingabemaske. Um erneut das Beispiel des letzten Absatzes fortzusetzen, könnte der von dem Intermediate-Level Keyword gelieferte Wert dann mit einem Erwartungswert verglichen werden.

3.3.2 Syntax

Wie bereits in Kapitel 2.3.4 gesagt ist die Syntax der Keywords von der gewählten Sprache abhängig.

Da die meisten Programmiersprachen Englisch verwenden, Playwright auf Englisch ist und auch der Coffeemaker auf Englisch geschrieben ist, fällt die Wahl auf Englisch als Sprache für die hier zu verwendenden Keywords leicht. Daraus folgend wird der englische Imperativ für die Keywords verwendet.

Die Schreibweise der Keywords ergibt sich hier daraus, dass sie als TypeScript-Funktionen implementiert werden.
Dem TypeScript Styleguide folgend, werden Funktionen und Methoden, ebenso wie Variablen, in "Lower Camel Case" geschrieben [TypeScriptStyleGuide].

Die Keywords sollten hier nach Möglichkeit kurze, beschreibende Sätze bilden. Ein Beispiel dazu wäre "checkThatButtonIsVisible". Aus dem Imperativ folgt, dass Keywords mit Verben wie "check" oder "get" beginnen sollten. Darauf sollte das betroffene Objekt, wie ein Knopf, mit möglichst eindeutigem Namen folgen. Bei "check"-Keywords sollte das Keyword mit einer kurzen Beschreibung der überprüften Eigenschaft, wie zum Beispiel "IsVisible", enden.

Die Beschreibung sollte dabei möglichst genau sein. Entsprechend ist das Keyword "checkThatButtonIsVisible" dem Keyword "checkButtonVisibility" vorzuziehen. Aus zweiterem ist nicht zu entnehmen, ob erwartet wird, dass der Knopf sichtbar ist oder nicht.

4 Realisierung der Tests und Fehler

Dieses Kapitel beschäftigt sich mit der Realisierung der in Kapitel 3 beschriebenen Testfälle, der damit zusammenhängenden Keywords, sowie den Fehlern, die in die Testanwendung eingebaut werden sollen. Dazu wird zunächst der Aufbau jedes einzelnen Testfalls mit den darin verwendeten Keywords beschrieben. Darauf folgen die Beschreibungen der Keywords selbst und letztlich wird beschrieben, welche Fehler in die Anwendung eingeführt werden, welches Fehlverhalten dadurch zu erwarten ist und folglich welche Testfälle einen Fehler erwartungsgemäß aufgreifen sollten.

4.1 Spezifikation der Testfälle

Nachstehend werden die Testfälle spezifiziert. Dazu wird die Funktion jedes Testfalls beschrieben und sein Aufbau aus Keywords dargelegt. Diese Keywords kommen, wie in Kapitel 3.3 beschrieben, möglichst aus dem High-Level und dem Domain-Layer. Dabei ist zu bedenken, dass die Anwendung nur über das Webinterface angesteuert wird. Das hat zur Folge, dass sie zwischen den Testfällen, wenn diese sequenziell ausgeführt werden, nicht zurückgesetzt werden kann. Entsprechend bauen die Testfälle aufeinander auf, sie arbeiten also jeweils auf einem System, dessen Zustand von vorherigen Testfällen verändert wurde. Das heißt, dass der Fehlschlag eines Testfalls dazu führen kann, dass der nächste ebenfalls fehlschlägt, da die für diesen notwendigen Voraussetzungen nicht mehr gegeben sind. Um derartige Abhängigkeiten zu vermeiden, überprüfen die ersten 5 Tests die Erreichbarkeit der Website, sowie die korrekte Darstellung der notwendigen Features. Weiterhin wird vorausgesetzt, dass die Zutatenwerte in der Datenbank des Coffeemakers zu Beginn der Tests auf null stehen.

Test 01: Check that recipe and ingredient page are available.

Hier soll überprüft werden, ob die Website des Testobjektes erreichbar ist. Da dieser Testfall grundlegend für alle anderen ist, wird genauer getestet, ob auch die URLs für die "Recipes"- und "Ingredients"-Unterseiten erreichbar sind. Dazu wird erst mit dem Keyword "goToCoffeemakerWebsite" die Hauptseite geladen, welche dann mit "checkThatWebsiteIsCoffeemaker" darauf hin überprüft wird, ob die URL auch richtig geladen wurde. Anschließend wird mit "switchToRecipesTab" auf die "Recipes"-Unterseite gewechselt, welche ihrerseits, mit "checkThatWebsiteIsRecipes", daraufhin überprüft wird, ob die URL der Erwartung entspricht. Das Gleiche geschieht daraufhin mit "switchToIngredientsTab" und "checkThatWebsiteIsIngredients" für die "Ingredients"-Unterseite. Wurden alle Seiten erfolgreich geladen, war auch der Test erfolgreich.

Test 02: Check that all ingredients are shown.

Hier soll überprüft werden, ob alle Zutaten auf der "Ingredients"-Seite vollständig angezeigt werden. Dazu wird zunächst, mit "goToCoffeemakerWebsite", die Website geladen. Danach wird für jede Zutat, also "Coffee", "Water", "Milk", "Cocoa" und "Sugar", mit den Keywords "checkThatCoffeeIngredientIsVisible", "checkThatWaterIngredientIsVisible", "checkThatMilkIngredientIsVisible", "checkThatCocoaIngredientIsVisible" und "checkThatSugarIngredientIsVisible" überprüft, ob sie, wie erwartet, mit Namen, Maßeinheit, Wert und "Refill"-Knopf, angezeigt werden.

Test 03: Check that all ingredients are empty.

Hier soll überprüft werden, dass - wie oben gefordert - alle Zutaten leer sind. Dazu wird zunächst, mit "goToCoffeemakerWebsite", die Website geladen. Das Überprüfen, ob die Zutaten leer sind, erfolgt mit dem Keyword "checkThatAllIngredientsAreEmpty".

Test 04: Check that default recepies are shown.

Hier soll verifiziert werden, dass alle 4 Standard-Rezepte, "Americano", "Espresso", "Latte Macchiato" und "Hot Chocolate", mit ihrem Namen und dem dazu gehörenden "Brew"-Knopf auf der "Recipes"-Unterseite angezeigt werden. Dazu wird zunächst, mit "goToCoffeemakerWebsite", die Website geladen, wonach mit der Verwendung von allen Keywords der Kategorie "checkThatRecipe<X>IsVisible" überprüft wird, ob Namen und Knöpfe angezeigt werden.

Test 05: Check that recipe input form is shown.

Hier soll die Sichtbarkeit der Eingabemaske für neue Rezepte, mit allen nötigen Eingabefeldern und dem dazu gehörenden Knopf, überprüft werden. Dazu wird zunächst, mit "goToCoffeemakerWebsite", die Website geladen. Die Sichtbarkeit jedes Eingabefeldes wird mit dem Keyword "checkThatFormCreateRecipeIsVisible" überprüft.

Test 06: Check that correct ingredient amounts are added.

In diesem Testfall soll überprüft werden das, nach dem Nachfüllen von Zutaten, die korrekten, zu erwartenden Mengen angezeigt werden. Dazu wird zunächst, mit "goToCoffeemakerWebsite", die Website geladen. Folgend wird das Keyword "checkThatIngredientAmountsAreCorrectlyAdded" verwendet, um für jede Zutat das korrekte Hinzufügen zu überprüfen.

Test 07: Check that success message is shown when brewing with sufficient ingredients.

Hier wird überprüft, dass beim Brauen eines Rezeptes, während genügend Zutaten vorhanden sind, die korrekte Erfolgsmeldung angezeigt wird. Dazu wird, nach Laden der Website mit "goToCoffeemakerWebsite", mit den Keywords "checkThatIngredientLevelsAreSufficientToBrewAmericanoOrStockUp" und "checkForSuccessMessageForBrewingAmericano" zunächst sichergestellt, dass ausreichend Zutaten für das Rezept, hier "Americano", vorhanden sind, wonach es gebraut wird,

um die darauf folgende Mitteilung auszuwerten. Das Gleiche geschieht auch für die anderen Standard-Rezepte mit den Keywords "checkThatIngredientLevelsAreSufficientToBrewEspressoOrStockUp" und "checkForSuccessMessageForBrewingEspresso", "checkThatIngredientLevelsAreSufficientToBrewLatteMacchiatoOrStockUp" und "checkForSuccessMessageForBrewingLatteMacchiato", sowie "checkThatIngredientLevelsAreSufficientToBrewHotChocolateOrStockUp" und "checkForSuccessMessageForBrewingHotChocolate".

Test 08: Check that error message is shown when brewing without sufficient ingredients.

Ähnlich wie in Test 7 (siehe oben), wird hier getestet, ob beim Versuch ein Rezept ohne die notwendigen Zutaten zu brauen, die korrekte Fehlermeldung angezeigt wird. Nach Laden der Website mit "goToCoffeemakerWebsite" wird dazu, mit den Keywords "check-ThatIngredientLevelsAreTooLowToBrewAmericano" und "checkForErrorMessageForBrewingAmericano", erst sichergestellt, dass die Zutaten nicht ausreichen um das Rezept, hier "Americano", zu brauen. Danach wird versucht es zu brauen, um zu sehen, ob die zu erwartende Fehlermeldung angezeigt wird. Auf die gleiche Art werden auch die anderen Standard-Rezepte, mit den Keywords "checkThatIngredientLevelsAreTooLowToBrewEspresso" und "checkForErrorMessageForBrewingEspresso", "checkThatIngredientLevelsAreTooLowToBrewLatteMacchiato" und "checkForErrorMessageForBrewingLatteMacchiato", "checkThatIngredientLevelsAreTooLowToBrewHotChocolate" und "checkForErrorMessageForBrewingHotChocolate", überprüft.

Test 09: Add recipe and check that it has been added to the list.

Hier soll getestet werden, ob ein neu angelegtes Rezept erfolgreich, mit den Standard-Rezepten, angezeigt wird. Dazu wird, nach Laden der Website durch "goToCoffeemaker-Website", mit "createTestRecipeWithFormCreateRecipe" die Eingabemaske gefüllt und auf den "Create new Recipes"-Knopf gedrückt, wodurch ein neues Rezept angelegt wird. Um die Rezept-Anzeige neu zu laden wird dann die Website durch das Keyword "reload-Website" aktualisiert. Ob das neue Rezept angezeigt wird, wird dann mit dem Keyword "checkThatRecipeTestRecipeIsVisibleAndCorrect", ähnlich wie in Test 4, überprüft.

Test 10: Ckeck that the correct ingredient amounts have been removed after brewing.

In diesem Testfall wird überprüft das, für jedes Standard-Rezept, die richtige Menge an Zutaten verbraucht wird. Nach Laden der Website mit dem Keyword "goToCoffeemakerWebsite", wird mit "checkThatIngredientLevelsAreSufficientToBrewAmericanoOrStockUp" sichergestellt, dass erfolgreich gebraut werden kann. Danach werden die aktuellen Zutatenfüllstände mit "getIngredientLevels" ausgelesen, zwischengespeichert und dann an das Keyword "checkThatIngredientLevelsAreAsExpectedAfterBrewingAmericano" weitergegeben. Dort wird das Rezept gebraut, wonach überprüft wird, ob die neuen Zutatenfüllstände den Erwartungen entsprechen. Auf die gleiche Art und Weise werden die andern 3 Rezepte getestet. Die dabei verwendeten Keywords sind "checkThatIngredientLevelsAreSufficientToBrewEspressoOrStockUp" und "checkThatIngredientLevelsAreAsExpectedAfterBrewingEspresso", "checkThatIngredientLevelsAreSufficientToBrewLatteMacchiatoOrStockUp" und "checkThatIngredientLevelsAreAsExpectedAfteBrewingLatteMacchiato", sowie "checkThatIngredientLevelsAreSufficientToBrewHotChocolateOrStockUp" und "checkThatIngredientLevelsAreAsExpectedAfterBrewingHotChocolate". Die aktuellen Zutatenfüllstände werden jeweils mit "getIngredientLevels" ausgelesen.

Test 11: Brew Americano whilst refilling missing ingredients.

Hier soll nun ein vollständiger Betriebsvorgang getestet werden. Dazu wird ein Rezept, hier "Americano", gebraut. Nach jedem fehlgeschlagen Brauversuch wird der Fehlermeldung die fehlende Zutat entnommen, welche dann, dem Rezept entsprechend, nachgefüllt wird. Dann wird erneut versucht das Rezept zu brauen. Nach maximal 5 fehlgeschlagenen Brauversuchen endet der Test, denn zu diesem Zeitpunkt wurden alle Zutaten nachgefüllt, was zur Folge hat, dass der 6. Brauversuch funktioniert oder der Testfall fehlschlägt. Dazu wird, nach Laden der Website mit "goToCoffeemakerWebsite", das Keyword "tryToBrewAmericanoAndRefillMissingIngredients" verwendet.

4.2 Aufbau der Keyword-Bibliothek

Hier werden für alle Keywords - kategorisiert nach Layer und Level - deren Funktionen beschrieben. Dabei ist zu beachten, dass nur High-Level Keywords in den Testfällen selbst (siehe Kapitel 4.1) verwendet werden. Weiterhin werden hier keine Keywords des Test-Interface-Layer aufgeführt, da diese direkt aus Playwright stammen. (vgl. Kapitel 3.3.1) Dennoch werden sie in Kapitel 4.2.5 kurz in ihrer Funktion erläutert. Aufgrund der inhärenten Ähnlichkeiten einiger Keywords werden diese in Kategorien zusammengefasst. Diese Kategorien haben ein "<X>" im Namen, welches ein Platzhalter für den Namen eines Rezeptes oder einer Zutat ist. Die Funktion der Keywords geht aus ihren sprechend gewählten Namen und aus ihren Beschreibungen im Aufbau der Tests (siehe Kapitel 4.1) hervor.

4.2.1 Domain-Layer High-Level Keywords

Keywords des Domain-Layer sind fachspezifisch und beziehen sich hier auf Kaffeemaschinen, oder deren potenzielle Online-Interfaces, im Allgemeinen.

Ihre Verwendung wurde in Kapitel 4.1 genannt. Hier wird beschrieben welche Keywords sie zu welchem Zweck verwenden.

High-Level Keywords nutzen dabei Keywords aus dem Intermediate- und, wenn notwendig, dem Low-Level.

checkThat<X>IngredientIsVisible

Zu dieser Kategorie gehören die Keywords "checkThatCoffeeIngredientIsVisible", "checkThatWaterIngredientIsVisible", "checkThatMilkIngredientIsVisible", "checkThatCocoaIngredientIsVisible" und "checkThatSugarIngredientIsVisible". Sie werden in Testfall 2 verwendet. Am Beispiel für die Zutat Kaffee, wird zunächst das Keyword "switchToIngredientsTab" genutzt, um auf die "Ingredients"-Unterseite zu wechseln. Danach werden die Keywords "checkThatCoffeeLevelIndicatorIsVisible" und "checkThatButtonRefillCoffeeIsVisible" verwendet, um die Sichtbarkeit des Füllstand-Indikators, also des aktuellen Werts, der dazu gehörenden Maßeinheit, des Eingabefeldes und des "Refill"-Knopfes, für die Zutat "Coffee", zu überprüfen.

checkThatAllIngredientsAreEmpty

Dieses Keyword wird in Testfall 3 verwendet. Hier wird das Keyword "switchToIngredientsTab" genutzt, um auf die "Ingredients"-Unterseite zu wechseln. Danach werden alle Keywords der Kategorie "checkThat<X>ValueIsZero" verwendet, um zu überprüfen, dass die entsprechenden Zutatenfüllstände auf null stehen.

checkThatRecipe<X>IsVisible

Zu dieser Kategorie gehören die Keywords "checkThatRecipeAmericanoIsVisible", "checkThatRecipeEspressoIsVisible", "checkThatRecipeLatteMacchiatoIsVisible" und "checkThatRecipeHotChocolateIsVisible", welche in Testfall 4 verwendet werden, sowie "checkThatRecipeTestRecipeIsVisibleAndCorrect", welches in Testfall 9 verwendet wird. Am Beispiel des Rezeptes "Americano" wird hier zunächst der Name des Rezeptes definiert, mit welchem die Funktion "checkThatRecipeIsVisible" aufgerufen wird. In dieser wird das Keyword "switchToRecipesTab" genutzt, um auf die "Recipes"-Unterseite zu wechseln. Folgend wird mit den Keywords "checkThatRecipeNameIsVisible" und "checkThatButtonBrewRecipeIsVisible" überprüft, ob der Rezeptname und der dazu gehörende Brau-Knopf sichtbar sind. Für Keyword "checkThatRecipeTestRecipeIsVisibleAndCorrect" wird weiterhin die Funkion "checkThatRecipeIsCorrect" aufgerufen, um zu prüfen, dass die angegebenen Zutatenwerte korrekt angezeigt werden. Darin werden zunächst mit dem Keyword "getIngredientsForRecipe" und dem Namen des Rezeptes die dazugehörenden Daten ausgelesen. Beide Rezepte werden daraufhin dem Keyword "checkThatRecipesAreEqual" übergeben, wo sie auf ihre Gleichheit überprüft werden.

checkThatFormCreateRecipeIsVisible

Dieses Keyword wird in Testfall 5 verwendet. Hier wird erst mit dem Keyword "switchToRecipesTab" auf die "Recipes"-Unterseite gewechselt, wonach alle Keywords der Kategorie "checkThatRecipeInputField<X>IsVisible", sowie das Keyword "checkThatRecipeButtonCreateNewRecipeIsVisible" verwendet werden. Diese prüfen ob jedes einzelne Eingabefeld der Eingabemaske zum Erstellen neuer Rezepte angezeigt wird.

checkThatIngredientAmountsAreCorrectlyAdded

Dieses Keyword wird in Testfall 6 verwendet. Dabei wird zuerst die "Ingredients"-Unterseite aufgerufen, indem das "switchToIngredientsTab" Keyword verwendet und eine Konstante definiert wird, welche den auf die Zutaten aufzuaddierenden Betrag hält. Danach werden alle Keywords der Kategorie "checkThat<X>AmountIsCorrectlyAdded" verwendet um, für jede Zutat, zu testen, ob der Betrag korrekt aufaddiert wird.

checkThatIngredientLevelsAreSufficientToBrew<X>OrStockUp

Zu dieser Kategorie gehören die Keywords "checkThatIngredientLevelsAreSufficientToBrewAmericanoOrStockUp", "checkThatIngredientLevelsAreSufficientToBrewEspressOrStockUp", "checkThatIngredientLevelsAreSufficientToBrewLatteMacchiatoOrStockUp" und "checkThatIngredientLevelsAreSufficientToBrewHotChocolateOrStockUp". Sie werden alle in Testfall 7, ersteres aber auch noch in Testfall 10, verwendet. Am Beispiel des Rezeptes "Americano", wird zunächst der Name des Rezeptes definiert, mit welchem dann die Funktion "checkThatIngredientLevelsAreSufficientToBrewRecipeOrStockUp" aufgerufen wird. In dieser wird das Keyword "switchToRecipesTab" verwendet, um auf die "Recipes"-Unterseite zu navigieren, wo dann mit dem Keyword "getIngredientsForRecipe" die Daten des Rezeptes ausgelesen werden. Folgend wird mit "switchToIngredientsTab" auf die "Ingredients"-Unterseite gewechselt, wo mit dem Keyword "refillIngredientsForRecipeIfNecessary" Zutaten, dem Rezept entsprechend, nachgefüllt werden, sollte der Füllstand zu gering sein. Dann wird mit dem Keyword "checkThatThereAreSufficientIngredientsForRecipe" überprüft, dass die Zutatenfüllstände ausreichen, um das Rezept zu brauen.

checkThatIngredientLevelsAreTooLowToBrew<X>

Zu dieser Kategorie gehören die Keywords "checkThatIngredientLevelsAreTooLowToBrewAmericano", "checkThatIngredientLevelsAreTooLowToBrewEspresso", "checkThatIngredientLevelsAreTooLowToBrewLatteMacchiato" und "checkThatIngredientLevelsAreTooLowToBrewHotChocolate". Sie werden in Testfall 8 verwendet. Am Beispiel des Rezeptes "Americano" wird zuerst der Name des Rezeptes definiert, mit welchem dann die Funktion "checkThatIngredientLevelsAreTooLowToBrewRecipe" aufgerufen wird. In dieser wird das Keyword "switchToRecipesTab" verwendet, um auf die "Recipes"-

Unterseite zu navigieren, wo dann mit dem Keyword "getIngredientsForRecipe" die Daten des Rezeptes ausgelesen werden. Folgend wird mit "switchToIngredientsTab" auf die "Ingredients"-Unterseite gewechselt, wo mit dem Keyword "checkThatThereAreNotSufficientIngredientsForRecipe" geprüft wird, dass die Zutatenfüllstände nicht ausreichen, um das Rezept zu brauen.

checkForSuccessMessageForBrewing<X>

Zu dieser Kategorie gehören die Keywords "checkForSuccessMessageForBrewingAmericano", "checkForSuccessMessageForBrewingEspresso", "checkForSuccessMessageForBrewingLatteMacchiato" und "checkForSuccessMessageForBrewingHotChocolate". Sie werden in Testfall 7 verwendet. Am Beispiel des Rezeptes "Americano" wird zunächst der Name des Rezeptes definiert, mit welchem dann die Funktion "checkForSuccessMessageForBrewingRecipe" aufgerufen wird. In dieser wird mit "switchToRecipesTab" auf die "Recipes"-Unterseite navigiert, wo dann mit dem Keyword "checkThatBrewingWillSucceed" eine Überprüfung angesetzt wird, die das zu erwartende Dialogfenster, nach Betätigen des "Brew"-Knopfes durch Verwenden des Keywords "tryToBrewRecipe", auswertet. Dabei wird dann überprüft, ob das ausgelöste Dialogfenster die erwartete Erfolgsmeldung enthält.

checkForErrorMessageForBrewing<X>

Zu dieser Kategorie gehören die Keywords "checkForErrorMessageForBrewingAmericano", "checkForErrorMessageForBrewingEspresso", "checkForErrorMessageForBrewingLatteMacchiato" und "checkForErrorMessageForBrewingHotChocolate". Diese werden in Testfall 8 verwendet. Sie sind im Kern identisch mit dem Aufbau der Keywords der Kategorie "checkForSuccessMessageForBrewing<X>". Der einzige Unterschied ist, dass hier nicht die Funktion "checkForSuccessMessageForBrewingRecipe", sondern die Funktion "checkForErrorMessageForBrewingRecipe" aufgerufen wird. Der Unterschied zwischen beiden Funktionen ist, dass letztere das Keyword "checkThatBrewingWillFail" verwendet anstatt "checkThatBrewingWillSucceed".

createTestRecipeWithFormCreateRecipe

Dieses Keyword wird in Testfall 9 verwendet. Hier wird, nach einem Wechsel auf die "Recipes"-Unterseite mit dem "switchToRecipesTab" Keyword, die Eingabemaske zum Erstellen eines neuen Rezeptes gefüllt. Dafür werden die Keywords "fillRecipeInputFieldRecipeName", mit dem Rezeptnamen, "fillRecipeInputFieldSugarAmount", mit der Zuckermenge des Rezeptes, "fillRecipeInputFieldCoffeeAmount", mit der Kaffeemenge des Rezeptes, "fillRecipeInputFieldWaterAmount", mit der Wassermenge des Rezeptes, "fillRecipeInputFieldMilkAmount", mit der Milchmenge des Rezeptes und "fillRecipeInputFieldCocoaAmount", mit der Kakaomenge des Rezeptes, aufgerufen. Nachdem so die gesamte Eingabemaske gefüllt ist, wird mit "pressButtonCreateNewRecipe" der Knopf zum Erstellen des Rezeptes gedrückt.

getIngredientLevels

Dieses Keyword wird in Testfall 10 verwendet. Im Gegensatz zu den meisten Keywords hat dieses einen Rückgabewert. Nach einem Wechsel auf die "Ingredients"-Unterseite mit dem "switchToIngredientsTab" Keyword wird das Keyword "getCurrentIngredientAmount", mit jeder der 5 Zutaten, aufgerufen, um die entsprechenden Zutatenfüllstände auszulesen. Diese werden dann als Liste, in der die Zutaten in der Reihenfolge Zucker, Kaffee, Wasser, Milch, Kakao stehen, zurückgegeben.

checkThatIngredientLevelsAreAsExpectedAfterBrewing<X>

Zu dieser Kategorie gehören die Keywords "checkThatIngredientLevelsAreAsExpectedAfterBrewingAmericano", "checkThatIngredientLevelsAreAsExpectedAfterBrewingEspresso", "checkThatIngredientLevelsAreAsExpectedAfterBrewingLatteMacchiato" und "checkThatIngredientLevelsAreAsExpectedAfterBrewingHotChocolate". Sie werden in Testfall 10 verwendet. Am Beispiel des Rezeptes "Americano" wird zunächst der Name des Rezeptes definiert, mit welchem dann die Funktion "checkThatIngredientLevelsAreAsExpectedAfterBrewingRecipe" aufgerufen wird. In dieser wird zunächst mit dem Keyword "switchToRecipesTab" auf die "Recipes"-Unterseite gewechselt. Dann werden mit Hilfe von "getIngredientsForRecipe" die Daten des Rezeptes ausgelesen. Darauf folgend wird mit dem Keyword "tryToBrewRecipe" der Brauvorgang für das Rezept gestartet. Nach einem weiteren Wechsel auf die "Ingredients"-Unterseite wird das Keyword

"checkThatIngredientLevelsAreAsExpected" genutzt um zu überprüfen, ob die Zutaten korrekt, dem Rezept folgend, abgezogen wurden.

4.2.2 Navigation-Layer High-Level Keywords

Keywords des Navigation-Layer sind spezifisch für das Testobjekt, hier also den Coffeemaker von Christopher Dührkop.

Ihre Verwendung wurde in Kapitel 4.2.1 genannt. Hier wird beschrieben welche Keywords sie zu welchem Zweck verwenden.

goToCoffeemakerWebsite

Dieses Keyword wird in allen Testfällen verwendet. Hier wird das Keyword "loadWebsite" mit der URL des Coffeemakers aufgerufen. Auf den ersten Blick scheint dies etwas sinnfrei zu sein, doch um die Hierarchie der Keywords aufrechtzuerhalten, war es unmöglich in diesem High-Level Keyword Low-Level Befehle, hier aus Playwright, zu verwenden.

checkThatWebsiteIs<X>

Zu dieser Kategorie gehören die Keywords "checkThatWebsiteIsCoffeemaker", "checkThatWebsiteIsRecipes" und "checkThatWebsiteIsIngredients". Diese werden in Testfall 1 verwendet. Dabei wird das Keyword "checkThatWebsiteHasURL" mit der URL des Coffeemakers, der "Recipes"- oder der "Ingredients"-Unterseite aufgerufen.

switchTo<X>Tab

Zu dieser Kategorie gehören die Keywords "switchToRecipesTab" und "switchToIngredientsTab". Diese werden in Testfall 1, sowie mehreren High-Level Keywords des Navigation- und Domain-Layer verwendet. Dabei wird das Low-Level Navigation-Layer Keyword "clickRecipesLink" oder "clickIngredientsLink" verwendet.

reloadWebsite

Dieses Keyword wird in Testfall 9 verwendet. Hier wird, obwohl es sich um ein High-Level Keyword handelt, direkt mit einem Low-Level Test-Interface-Layer Keyword, also einer Playwright Funktion, gearbeitet. Dies ist der Fall, da die Testfälle nur High-Level Keywords verwenden sollen und sich der Name des Keywords auf geringeren Levels nicht verändert hätte. Gleichzeitig dürfen keine zwei Keywords denselben Namen tragen. Die verwendete Playwright Funktion ist "page.reload()". Dabei wird das "Page" Objekt, das die aktuell offene Website beschreibt, neu geladen.

tryToBrewAmericanoAndRefillMissingIngredients

Dieses Keyword wird in Testfall 11 verwendet. Es gestaltet sich, aufgrund technischer Notwendigkeiten, weitaus komplexer, als die anderen Keywords. Wie in Testfall 11 (vgl. Kapitel 4.1) beschrieben, soll hier ein Betriebsvorgang abgebildet werden. Dieser ist im Anhang, in Abbildung A.7, zur Verdeutlichung als Sequenzdiagramm dargestellt. In diesem Betriebsvorgang wird zunächst der Rezeptname definiert, hier "Americano", wonach mit "switchToRecipesTab" auf die "Recipes"-Unterseite gewechselt wird. Dort werden mit Nutzung des Keywords "getIngredientsForRecipe" die Daten für das Rezept ausgelesen. Weiterhin werden die Rückgabewerte, der Boolean "brewingFailed" und der String "missingIngredientName", für das Keyword "findMissingIngredient" definiert, wonach dieses aufgerufen wird. Daran anschließend beginnt die Schleife, welche maximal 6 Mal durchlaufen wird, um die Möglichkeit zu bieten, jede einzelne der 5 Zutaten nachzufüllen und dann noch einen finalen Brauversuch zu starten. Darin wird zunächst auf die "Recipes"-Unterseite gewechselt ("switchToRecipesTab") und es wird mit "tryToBrewRecipe" versucht, das Rezept zu brauen. Dann werden die Variablen "brewingFailed" und "missingIngredientName" aktualisiert, da sie sich durch den Brauversuch verändert haben sollten. Abhängig von der Variable "brewingFailed" wird dann entschieden, ob die Schleife unterbrochen werden soll, im Falle das der Brauvorgang erfolgreich war, oder ob eine Zutat nachgefüllt werden muss. Wenn eine Zutat nachgefüllt werden soll, steht deren Namen in der Variable "missingIngredientName". Die nachzufüllende Menge wird, mit Hilfe eines Switch-Case-Statements, dem zuvor bestimmten Rezept entnommen. Menge und Zutatenname werden, nach einem Wechsel auf die "Ingredients"-Unterseite ("switchToIngredientsTab"), dem Keyword "addAmountToIngredient" übergeben, um die Zutat nachzufüllen. Dieser Vorgang wird wiederholt bis die Schleife beendet ist,

sei es durch Ablauf des Zählers oder durch das von "brewingFailed" abhängige Break-Statement. In diesem Fall wird die Variable "brewingFailed" auf ihre Falschheit getestet, denn nur wenn sie als "false" ausgewertet wird, war der Brauvorgang erfolgreich.

4.2.3 Navigation-Layer Intermediate-Level Keywords

Keywords des Navigation-Layer sind spezifisch für das Testobjekt, hier also den Coffeemaker von Christopher Dührkop. Intermediate-Level Keywords werden von High-Level Keywords verwendet und kapseln dahingehend das Low-Level weiter ab.

Ihre Verwendung wurde in Kapitel 4.2.2 genannt. Hier wird beschrieben welche Keywords sie zu welchem Zweck verwenden.

checkThat<X>LevelIndicatorIsVisible

Zu dieser Kategorie gehören die Keywords "checkThatCoffeeLevelIndicatorIsVisible", "checkThatWaterLevelIndicatorIsVisible", "checkThatMilkLevelIndicatorIsVisible", "checkThatCocoaLevelIndicatorIsVisible" und "checkThatSugarLevelIndicatorIsVisible". Sie werden in den High-Level Navigation-Layer Keywords der Kategorie "checkThat<X>-IngredientIsVisible" verwendet. Dabei wird die Zutat, hier am Beispiel von Kaffee, mit Einheitensymbol ("g") und Namen ("Coffee"), definiert. Mit dieser Zutat wird dann das Keyword "checkThatIngredientLevelIndicatorIsVisible" aufgerufen. In dieser wird ein regulärer Ausdruck generiert, welcher auf den Namen, gefolgt von einem Doppelpunkt, einem Leerzeichen, mindestens einer Ziffer und dem Einheitensymbol, prüft. Dieser reguläre Ausdruck wird dann der Playwright-Funktion "page.getByText()" übergeben, die einen Zeiger auf ein Seitenelement liefert. Dieser wird dann der Playwright-Funktion "expect().toBeVisible()" übergeben um zu überprüfen, ob das Seitenelement sichtbar ist.

checkThatButtonRefill<X>IsVisible

Zu dieser Kategorie gehören die Keywords "checkThatButtonRefillCoffeeIsVisible", "checkThatButtonRefillWaterIsVisible", "checkThatButtonRefillMilkIsVisible", "checkThatButtonRefillCocoaIsVisible" und "checkThatButtonRefillSugarIsVisible". Sie werden in den High-Level Navigation-Layer Keywords der Kategorie "checkThat<X>-

IngredientIsVisible" verwendet. Dabei wird die Zutat, hier am Beispiel von Kaffee, mit Einheitensymbol ("g") und Namen ("Coffee"), definiert und dann dem Keyword "checkThatButtonRefillIngredientIsVisible" übergeben. In dieser wird mit der Playwright-Funktion "page.locator()", welche mit dem konkatenierten String aus "id=button-" und dem Zutatennamen aufgerufen wird, ein Zeiger auf das Seitenelement des "Refill"-Knopfes der Zutat generiert. Dieser Zeiger wird dann der Playwright-Funktion "expect().toBeVisible()" übergeben, um zu überprüfen, ob der Knopf sichtbar ist.

checkThat<X>ValueIsZero

Zu dieser Kategorie gehören die Keywords "checkThatCoffeeValueIsZero", "checkThatWaterValueIsZero", "checkThatMilkValueIsZero", "checkThatCocoaValueIsZero" und "checkThatSugarValueIsZero". Sie werden im High-Level Domain-Layer Keyword "checkThatAllIngredientsAreEmpty" verwendet. Zunächst wird, hier am Beispiel von Kaffee, die Zutat definiert, welche dann in der Funktion "checkThatIngredientValueIsZero" dahingehend überprüft wird, ob ihr Füllstand null beträgt. Dazu wird der aktuelle Füllstand mit "getCurrentIngredientAmount" ausgelesen und mit der Playwright-Funktion "expect().toBe(0)" überprüft.

checkThatRecipeNameIsVisible

Dieses Keyword wird in der High-Level Domain-Layer Funktion "checkThatRecipeIsVisible" verwendet. Hier wird der Playwright-Funktion "page.locator()" ein konkatenierter String aus "id=button-" und dem, dem Keyword übergebenen Namen, übergeben, um einen Zeiger auf den Rezeptnamen zu generieren. Dieser Zeiger wird dann der Playwright-Funktion "expect().toBeVisible()" übergeben, um zu überprüfen, ob der Knopf sichtbar ist.

checkThatButtonBrewRecipeIsVisible

Dieses Keyword wird in der High-Level Domain-Layer Funktion "checkThatRecipeIsVisible" verwendet. Dazu wird der "Brew"-Knopf mit Hilfe des Keywords "getButtonBrew" gefunden. Dieser wird dann mit der Playwright-Funktion "expect().toBeVisible()" auf seine Sichtbarkeit hin überprüft.

checkThatRecipeInputField<X>IsVisible

Zu dieser Kategorie gehören die Keywords "checkThatRecipeInputFieldRecipeNameIsVisible", "checkThatRecipeInputFieldSugarAmountIsVisible", "checkThatRecipeInputFieldCoffeeAmountIsVisible", "checkThatRecipeInputFieldWaterAmountIsVisible", "checkThatRecipeInputFieldMilkAmountIsVisible" und "checkThatRecipeInputFieldCocoaAmountIsVisible". Sie werden im High-Level Domain-Layer Keyword "checkThatFormCreateRecipeIsVisible" verwendet. Dabei wird, abhängig vom Keyword dieser Kategorie, dem Low-Level Navigation-Layer Keyword "checkThatRecipeInputFieldIsVisible" ein String übergeben. Für RecipeName ist das "name", für SugarAmount "sugarAmount", für CoffeeAmount "coffeeAmount", für WaterAmount "waterAmount", für MilkAmount "milkAmount" und für CocoaAmount "cocoaAmount".

checkThatRecipeButtonCreateNewRecipeIsVisible

Dieses Keyword wird im High-Level Domain-Layer Keyword "checkThatFormCreateRecipeIsVisible" verwendet. Dabei wird das Keyword "getButtonCreateNewRecipe" aufgerufen, dessen Rückgabewert dann der Playwright-Funktion "expect().toBeVisible()" zum Überprüfen der Sichtbarkeit übergeben wird.

checkThat<X>AmountsAreCorrectlyAdded

Zu dieser Kategorie gehören die Keywords "checkThatCoffeeAmountIsCorrectlyAdded", "checkThatCocoaAmountIsCorrectlyAdded", "checkThatSugarAmountIsCorrectlyAdded", "checkThatWaterAmountIsCorrectlyAdded" und "checkThatMilkAmountIsCorrectlyAdded". Sie werden im High-Level Domain-Layer Keyword "checkThatIngredientAmountsAreCorrectlyAdded" verwendet. Hier wird zunächst die Zutat definiert, zum Beispiel Kaffee, welche mit dem, dem Keyword übergebenen, zu ergänzenden Wert an die Intermediate-Level Navigation-Layer Funktion "checkThatIngredientAmountIsCorrectlyAdded" übergeben wird.

getIngredientsForRecipe

Dieses Keyword wird in den High-Level Domain-Layer Funktionen "checkThatIngredientLevelsAreTooLowToBrewRecipe", "checkThatIngredientLevelsAreSufficientToBrewRecipeOrStockUp" und "checkThatIngredientLevelsAreAsExpectedAfterBrewingRecipe", dem High-Level Navigation-Layer Keyword "tryToBrewAmericanoAndRefillMissingIngredients", sowie der High-Level Domain-Layer Funktion "checkThatRecipeIsCorrect" verwendet. Um die Bestandteile eines Rezeptes von der Seite auszulesen wird zunächst, mit der Playwright-Funktion "page.locator('id=recipe-'+recipeName)", ein Zeiger auf das Rezept im Ganzen generiert. In diesem Rezept sind die Zutaten in einer Tabelle aufgelistet. Diese hat 5 Zeilen, für jede Zutat eine, sowie 2 Spalten, die erste für den Namen der Zutat und die zweite für den zum Brauen benötigten Wert. Die Daten für jede Zutat werden mit einem Befehl wie ".locator('tr').nth(0).locator('td').nth(1).textContent().then()" ausgelesen. Dabei wird der erste Zeiger ("locator") von dem oben gefundenen Rezept aus aufgerufen. ".locator('tr')" liefert hierbei einen Zeiger auf die Tabellenzeilen. Davon ausgehend wird mit ".nth(0)" die erste Zeile spezifiziert, in dieser steht der Wert für Zucker. Mit ".locator('td')" wird dann ein Zeiger auf die Daten der gewählten Zeile generiert, von welchen mit ".nth(1)" auf den Zutatenwert zugegriffen wird. Aus diesem Datum wird dann mit ".textContent()" ein String ausgelesen. Aufgrund der asynchronen Natur von Playwright wird allerdings kein String, sondern ein aus TypeScript stammender "Promise<string>", zurückgegeben. Um diesen in einen String umzuwandeln ist noch das ".then()" notwendig. Analog geschieht dies auch mit den Daten für die anderen Zutaten, wobei sich die Zahl im ".nth()" verändert, um die anderen Zeilen anzuwählen. Letztlich werden die gesammelten Daten in einem definierten Format zurückgegeben.

checkThatThereAreSufficientIngredientsForRecipe

Dieses Keyword wird in der High-Level Domain-Layer Funktion "checkThatIngredientLevelsAreSufficientToBrewRecipeOrStockUp" verwendet. Hier wird die Playwright-Funktion "expect().toBeGreaterThanOrEqual()" verwendet, um den aktuell Füllstand jeder Zutat mit den für ein Rezept benötigten Werten zu vergleichen. Dazu wird der Füllstand jeder Zutat mit dem Keyword "getCurrentIngredientAmount" ausgelesen und dann mit dem gegebenen, zum Brauen notwendigen, Wert vergleichen. Dies geschieht für jede der 5 Zutaten.

checkThatThereAreNotSufficientIngredientsForRecipe

Dieses Keyword wird in der High-Level Domain-Layer Funkion "checkThatIngredientLevelsAreTooLowToBrewRecipe" verwendet. Ähnlich wie im Keyword "checkThatThereAreSufficientIngredientsForRecipe" (siehe oben) werden auch hier die aktuellen Füllstände der 5 Zutaten mit den Anforderungen eines Rezeptes verglichen. Dazu wird zunächst der Füllstand jeder Zutat mit dem Keyword "getCurrentIngredientAmount" ausgelesen. Abhängig von dem vom Rezept geforderten Wert wird dann die Playwright-Funktion "expect().toBe()" oder "expect().toBeLessThan()" ausgeführt. Sollte der geforderte Wert null sein, muss der aktuelle Wert ebenfalls null sein, da er nicht kleiner sein kann, um den Test zu bestehen. Ist der geforderte Wert allerdings größer als null, muss der aktuelle Wert kleiner sein. Diese Fallunterscheidung ist notwendig da, im Falle eines geforderten Wertes größer als null, die Gleichheit der Werte ebenfalls zum Fehlschlagen des Testes führen muss.

checkThatBrewingWill<X>

Die Keywords dieser Kategorie sind "checkThatBrewingWillFail" und "checkThatBrewingWillSucceed". Sie werden jeweils in den High-Level Domain-Layer Funktionen "checkForErrorMessageForBrewingRecipe" und "checkForSuccessMessageForBrewingRecipe" verwendet. Dabei wird mit der Playwright-Funktion "page.once()" ein Listener aufgesetzt, welcher auf ein Event reagiert und danach abgemeldet wird. Das erste Argument dieser Funktion ist "dialog", was den Listener auf Dialogfenster reagieren lässt. Im zweiten Argument der Funktion steht eine anonyme Funktion, in welcher der Inhalt des Dialogfensters mit der Playwright-Funktion "expect().toContain()" auf die zu erwartende Fehler- oder Erfolgsmeldung überprüft wird. Darauf folgend wird das Dialogfenster mit der ".accept()" Methode geschlossen. Somit wird das nächste Dialogfenster, welches geöffnet wird, durch diese anonyme Funktion verarbeitet.

tryToBrewRecipe

Dieses Keyword wird in den High-Level Domain-Layer Funktionen "checkForErrorMessageForBrewingRecipe" und "checkForSuccessMessageForBrewingRecipe", dem High-Level Navigation-Layer Keyword "tryToBrewAmericanoAndRefillMissingIngredients", sowie der High-Level Domain-Layer Funktion "checkThatIngredientLevelsAreAsExpectedAfterBrewingRecipe" verwendet. Zunächst wird mit dem Keyword "getButtonBrew" der Brau-Knopf für das, dem Keyword übergebene, Rezept gefunden. Auf diesem wird dann die Playwright-Funktion ".click()" ausgeführt, was den Knopf betätigt. Darauf folgend wird für 3000 Millisekunden (3 Sekunden) gewartet. Dies dient dazu dem Listener (siehe oben) ausreichend Zeit zum Arbeiten zu geben.

refillIngredientsForRecipeIfNecessary

Dieses Keyword wird in der High-Level Domain-Layer Funktion "checkThatIngredientLevelsAreSufficientToBrewRecipeOrStockUp" verwendet. Um alle Zutaten nachzufüllen, wenn nötig, wird zunächst, für jede Zutat, der aktuelle Füllstand mit dem Keyword "getCurrentIngredientAmount" abgefragt. Aus der Subtraktion dieses Wertes, von der für das Rezept notwendigen Menge, ergibt sich die nachzufüllende Menge. Ist diese kleiner oder gleich null, muss nichts nachgefüllt werden. Andernfalls wird das Keyword "addAmountToIngredient" mit dem errechneten Wert für die entsprechende Zutat ausgeführt. Dies geschieht ebenfalls für alle 5 Zutaten.

fillRecipeInputField<X>

Zu dieser Kategorie gehören die Keywords "fillRecipeInputFieldRecipeName", "fillRecipeInputFieldSugarAmount", "fillRecipeInputFieldCoffeeAmount", "fillRecipeInputFieldWaterAmount", "fillRecipeInputFieldMilkAmount" und "fillRecipeInputFieldCocoaAmount". Sie werden im High-Level Domain-Layer Keyword "createTestRecipeWithFormCreateRecipe" verwendet. Zum Ausfüllen des entsprechenden Feldes in der Eingabemaske wird das Low-Level Navigation-Layer Keyword "fillRecipeInputField" mit dem, dem Keyword der Kategorie entsprechenden, String und dem gegebenen Wert "value" aufgerufen. Dieser Name ist für "fillRecipeInputFieldRecipeName" "name", für "fillRecipeInputFieldSugarAmount" "sugarAmount", für "fillRecipeInputFieldCoffeeAmount"

"coffeeAmount", für "fillRecipeInputFieldWaterAmount" "waterAmount", für "fillRecipeInputFieldMilkAmount" "milkAmount" und für "fillRecipeInputFieldCocoaAmount" "cocoaAmount".

pressButtonCreateNewRecipe

Dieses Keyword wird im High-Level Domain-Layer Keyword "createTestRecipeWithFormCreateRecipe" verwendet. Zunächst wird mit dem Keyword "getButtonCreateNewRecipe" der "Create new Recipe"-Knopf gefunden. Auf diesem wird dann die Playwright-Funktion ".click()" ausgeführt, was den Knopf betätigt.

checkThatIngredientLevelsAreAsExpected

Dieses Keyword wird von der Funkion "checkThatIngredientLevelsAreAsExpectedAfterBrewingRecipe" im High-Level Domain-Layer verwendet. Diese wiederum wird von den Keywords der Kategorie "checkThatIngredientLevelsAreAsExpectedAfterBrewing<X>" verwendet. Das Keyword wird mit einem verwendeten Rezept und einer Liste von Zutatenfüllständen aufgerufen. Getestet werden soll, ob die aktuellen Zutatenfüllstände dem jeweiligen Füllstand minus dem vom Rezept verwendeten Wert entsprechen. Dazu wird für jede Zutat der aktuelle Füllstand mit "getCurrentIngredientAmount" ausgelesen und mit der Playwright-Funkion "expect().toBe()" mit der Subtraktion des alten Füllstands und des vom Rezept verwendeten Wertes verglichen.

addAmountToIngredient

Dieses Keyword wird im High-Level Navigation-Layer Keyword "tryToBrewAmericanoAndRefillMissingIngredients", sowie dem Intermediate-Level Navigation-Layer Keyword "refillIngredientsForRecipeIfNecessary" und der Funktion "checkThatIngredientAmountIsCorrectlyAdded" im Intermediate-Level Navigation-Layer verwendet. Das Keyword wird mit dem Namen einer Zutat ("ingredientName") und einer zu addierenden Menge ("addAmount") aufgerufen. Mit dem Namen wird mit der Playwright-Funktion "page.locator('id=amount-' + ingredientName)" ein Zeiger auf das entsprechende Eingabefeld generiert, welches mit "fill(addAmount.toString())", mir der zu addierenden Menge gefüllt wird. Darauf folgend wird mit "page.locator('id=button-' + ingredientName)" ein Zeiger auf den "Refill"-Knopf erzeugt, welcher mit ".click()" betätigt wird.

findMissingIngredient

Dieses Keyword wird im High-Level Navigation-Layer Keyword "tryToBrewAmericanoAndRefillMissingIngredients" verwendet. Als Parameter wird hier eine Liste übergeben, welche für die Rückgabewerte genutzt wird. Dabei wird sich die Funktionalität von TypeScript zunutze gemacht das, im Gegensatz zu primitiven Datentypen, Listen als Objekte mit "Call-by-Reference" aufgerufen werden. Somit bleiben alle Änderungen an dem Objekt auch außerhalb der Funktion erhalten. Dies ist notwendig für die Funktion des Keywords "tryToBrewAmericanoAndRefillMissingIngredients" (vgl. Kapitel 4.2.2). Wie in Keyword "checkThatBrewingWillFail" wird auch hier ein Listener erstellt, der auf das nächste Dialogfenster reagiert und danach abgemeldet wird. In diesem werden zunächst die Rückgabewerte, ein Boolean für das Fehlschlagen des Brauvorgangs und ein String für den Namen der fehlenden Zutat auf "false" und einen leeren String zurückgesetzt. Danach wird die Nachricht des Dialogs daraufhin überprüft, ob sie einen fehlgeschlagenen Brauversuch meldet. Ist dies der Fall, wird der Rückgabe-Boolean auf "true" und der Rückgabe-String auf den Zutatennamen, welcher dem Text der Nachricht entnommen wird, gesetzt. Ist der Brauversuch nicht fehlgeschlagen, geschieht dies nicht. In jedem Fall wird danach das Dialogfenster geschlossen.

getCurrentIngredientAmount

Dieses Keyword wird im High-Level Domain-Layer Keyword "getIngredientLevels", den Intermediate-Level Navigation-Layer Keywords "checkThatIngredientLevelsAreAsExpected", "refillIngredientsForRecipeIfNecessary", "checkThatThereAreSufficientIngredientsForRecipe" und "checkThatThereAreNotSufficientIngredientsForRecipe", sowie den Funktionen "checkThatIngredientValueIsZero" und "checkThatIngredientAmountIsCorrectlyAdded" im Intermediate-Level Navigation-Layer verwendet. Dabei wird dem Keyword die Zutat, mit Einheitensymbol und Name, übergeben. Mit diesen wird das Keyword "getIngredientLevelIndicator" aufgerufen, welches einen Zeiger auf die Füllstandsanzeige der entsprechenden Zutat liefert. Aus diesem wird mit ".innerText()" der Text mit Namen, Füllstand und Einheit entnommen. Diesem Text wird dann mit der ".split()" Funktion aus TypeScript der Füllstand entnommen, welcher zurückgegeben wird.

checkThatRecipesAreEqual

Dieses Keyword wird in der High-Level Domain-Layer Funktion "checkThatRecipeIsCorrect" verwendet. Dabei wird für jede der 5 Zutaten, also Kaffee, Kakao, Milch, Wasser und Zucker, sowie für den Namen, die Playwright-Funkion "expect().toBe()" aufgerufen. Darin werden dann die jeweiligen Werte beider übergebener Rezepte miteinander verglichen.

4.2.4 Navigation-Layer Low-Level Keywords

Keywords des Navigation-Layer sind spezifisch für das Testobjekt, hier also den Coffeemaker von Christopher Dührkop. Low-Level Keywords werden von High- oder Intermediate-Level Keywords verwendet.

Ihre Verwendung wurde in Kapitel 4.1 genannt. Hier wird beschrieben welche Keywords sie zu welchem Zweck verwenden.

Sie verwenden selbst nur andere Low-Level Keywords, was hier allgemein Playwright Funktionen sind.

loadWebsite

Dieses Keyword wird im High-Level Navigation-Layer Keyword "goToCoffeemakerWebsite" verwendet. Dabei wird mit der, dem Keyword übergebenen URL, die Playwright-Funkion "page.goto()" aufgerufen. Dadurch wird die aktuelle Seite zur gegebenen URL gewechselt.

checkThatWebsiteHasURL

Dieses Keyword wird in den High-Level Navigation-Layer Keywords der Kategorie "checkThatWebsiteIs<X>" verwendet. Dabei wird die Playwright-Funktion "expect().toHaveURL()" mit der aktuellen Seite und der gegebenen URL aufgerufen. So wird geprüft, ob die aktuelle Seite die erwartete URL hat.

click<X>Link

Die Keywords dieser Kategorie sind "clickRecipesLink" und "clickIngredientsLink". Sie werden in den High-Level Navigation-Layer Keywords der Kategorie "switchTo<X>Tab" verwendet. Hier wird zunächst der Name des zu klickenden Links ("name") als "Recipes" oder "Ingredients" definiert, mit welchem dann, die Playwright-Funktion ".getByRole('link', { name: name })", einen Zeiger auf den gewünschten Link erzeugt. Dieser wird folgend mit ".click()" betätigt.

checkThatRecipeInputFieldIsVisible

Dieses Keyword wird in den Intermediate-Level Navigation-Layer Keywords der Kategorie "checkThatRecipeInputField<X>IsVisible" verwendet. Dabei wird zunächst mit dem Low-Level Navigation-Layer Keyword "getFormCreateNewRecipe" ein Zeiger auf die Eingabemaske zum Erstellen eines neuen Rezeptes generiert. Davon ausgehend wird mit der Playwright-Funkion ".locator('id='+name)" ein Zeiger auf das, dem übergebenen Namen entsprechende, Eingabefeld gefunden. Dessen Sichtbarkeit wir dann mit "expect().toBeVisible()" getestet.

checkThatIngredientLevelIndicatorIsVisible

Dieses Keyword wird in den Intermediate-Level Navigation-Layer Keywords der Kategorie "checkThat<X>LevelIndicatorIsVisible" verwendet. Das Keyword wird mit der Zutat, mit Einheitensymbol und Namen, aufgerufen. Diese werden an das Keyword "getIngredientLevelIndicator" übergeben, welches einen Zeiger auf die Füllstandsanzeige der entsprechenden Zutat liefert. Dieser Zeiger wird dann mit der Playwright-Funkion "expect().toBeVisible()" auf seine Sichtbarkeit überprüft.

checkThatButtonRefillIngredientIsVisible

Dieses Keyword wird im Intermediate-Level Navigation-Layer Keywords der Kategorie "checkThatButtonRefill<X>IsVisible" verwendet. Das Keyword wird mit der Zutat, mit Einheitensymbol ("ingredient.unit") und Namen ("ingredient.name") aufgerufen. Der Name wird genutzt, um mit der Playwright-Funktion ".locator('id=button-' + ingredient.name)" einen Zeiger auf den "Refill"-Knopf zu generieren. Dieser wird dann mit "expect().toBeVisible()" auf seine Sichtbarkeit überprüft.

fillRecipeInputField

Dieses Keyword wird in den Intermediate-Level Navigation-Layer Keywords der Kategorie "fillRecipeInputField<X>" verwendet. Dabei wird zunächst mit dem Keyword "getFormCreateNewRecipe" ein Zeiger auf die Eingabemaske zum Erstellen eines neuen Rezeptes generiert. Von diesem aus wird mit dem, dem Keyword übergebenen Zutatennamen ("ingredientName") und einzugebenden Wert ("value"), sowie der Playwright-Funkion ".locator('id='+ingredientName).locator('input')", ein Zeiger auf das Eingabefeld gefunden, welches durch ".fill(value)" mit dem Wert beschrieben wird.

getButtonBrew

Dieses Keyword wird in den Intermediate-Level Navigation-Layer Keywords "checkThatButtonBrewRecipeIsVisible" und "tryToBrewRecipe" verwendet. Zum Finden des "Brew"-Knopfes wird erst der Name des Knopfes generiert. Dazu wird "Brew " mit dem, dem Keyword übergebenen Namen, konkateniert. Mit dem generierten Namen wird dann, mit der Playwright-Funktion "page.getByRole('button', { name: buttonName })", wobei "buttonName" für den Namen steht, ein Zeiger auf den Knopf generiert, welcher zurückgegeben wird.

getButtonCreateNewRecipe

Dieses Keyword wird im Intermediate-Level Navigation-Layer Keyword "pressButtonCreateNewRecipe" verwendet. Dabei wird zunächst, mit dem Keyword "getFormCreateNewRecipe", ein Zeiger auf das Eingabeformular generiert. Auf diesem wird dann die Playwright-Funktion ".getByRole('button', { name: 'Create new Recipe' })" aufgerufen, welche einen Zeiger auf den entsprechenden Knopf liefert. Dieser wird anschließend zurückgegeben.

getFormCreateNewRecipe

Dieses Keyword wird in den Low-Level Navigation-Layer Keywords "checkThatRecipeInputFieldIsVisible", "fillRecipeInputField" und "getButtonCreateNewRecipe" verwendet. Dabei wird mit der Playwright-Funkion "page.locator('id=form-new-recipe')" ein Zeiger auf die Eingabemaske zum Erstellen eines neuen Rezeptes zurückgegeben.

getIngredientLevelIndicator

Dieses Keyword wird im Intermediate-Level Navigation-Layer Keyword "getCurrentIngredientAmount", sowie dem Low-Level Navigation-Layer Keyword "checkThatIngredientLevelIndicatorIsVisible" verwendet. Dabei werden dem Keyword die Zutat, mit Einheitensymbol ("ingredient.unit") und Namen ("ingredient.name"), übergeben. Mit diesen wird ein regulärer Ausdruck generiert, welcher auf den Namen, gefolgt von einem Doppelpunkt, einem Leerzeichen, mindestens einer Ziffer und dem Einheitensymbol, prüft. Mit diesem regulären Ausdruck wird dann die Playwright-Funktion "page.getByText()" aufgerufen, die einen Zeiger auf das entsprechende ein Seitenelement liefert. Dieses wird danach zurückgegeben.

4.2.5 Test-Interface-Layer Low-Level Keywords

Keywords des Test-Interface-Layer stammen direkt aus Playwright, sind also Playwright-Funktionen. Sie dienen dem direkten Ansprechen des Webinterface und sollen hier ebenfalls kurz in ihrer Funktion erklärt werden. Alle Informationen sind ebenfalls leicht in der API-Dokumentation von Playwright zu finden [PlaywrightAPI].

Page

Die grundlegende Klasse in Playwright heißt "Page" [PlaywrightAPIPage]. Sie beschreibt eine Website - wie ein Tab - in einem Browser und liefert Funktionen, um mit dieser zu interagieren.
In dieser Arbeit wurden die folgenden dieser Funktionen verwendet:

- .close()
  Diese Funktion schließt das aktuelle Browserfenster.

- .getByRole(role, {name})
  Diese Funktion liefert einen "Locator", welcher mit der entsprechenden Rolle und dem Namen spezifiziert ist. Ein Beispiel, aus dem Low-Level Navigation-Layer Keyword "clickIngredientsLink", ist "page.getByRole('link', { name: 'Recipes' })". Hiermit wird ein "Locator" geliefert, welcher auf einen Link zeigt, der auf der Web-Oberfläche "Recipes" heißt.
  In dieser Arbeit wurden die Rollen "link" und "button" verwendet; alle verfügbaren Rollen sind der API zu entnehmen.

- .getByText(text)
  Diese Funktion liefert einen "Locator", welcher mit dem entsprechenden Text spezifiziert ist. Der Text kann hierbei ein normaler String oder ein regulärer Ausdruck sein. Ein Beispiel findet sich im Low-Level Navigation-Layer Keyword "getIngredientLevelIndicator", wo der Funktion ein regulärer Ausdruck übergeben wird, welcher den Text der Füllstandsanzeige beschreibt.

- .goto(url)
  Diese Funktion ändert die URL der Website. Damit wird auf eine andere Website gewechselt.

- .locator(selector)
  Diese Funktion liefert einen "Locator", welcher über den gegebenen "Selector"-String spezifiziert wird. Dabei muss ein "Locator" nicht eindeutig sein, sondern kann auch auf mehrere Seitenelemente - wie beispielsweise alle Knöpfe - verweisen. Der "selector" beschreibt dabei einen Teil des HTML-Codes der Seite. Ein Beispiel - aus dem Low-Level Navigation-Layer Keyword "checkThatButtonRefillIngredientIsVisible" - ist "page.locator('id=button-' + ingredient.name)", wobei

"ingredient" eine Variable und "name" eine Eigenschaft dieser Variablen ist. Angenommen es gilt "ingredient.name = 'Coffee'", dann wird damit ein "Locator" geliefert, welcher auf ein Seitenelement zeigt, welches den String "id=button-Coffee" enthält.

- .reload()
  Diese Funktion lädt die aktuelle Seite erneut, auf die gleiche Weise, als hätte ein Benutzer auf "Seite neu Laden" geklickt.

- .once('dialog', dialog => {dialog.accept();})
  Die Funktion ".on('dialog', dialog => {dialog.accept();})" reagiert auf sich öffnende Dialogfenster in der Web-Oberfläche. Dabei ist der Teil innerhalb der geschwungenen Klammern eine anonyme Funktion, der das Dialogobjekt "dialog" übergeben wird. Dieses beinhaltet unter anderem eine Nachricht - "dialog.message()" - welche den Inhalt des Dialogfensters enthält. Nachdem mit ".on('dialog', dialog => {})" ein Listener gestartet wurde, reagiert dieser auf jedes sich öffnende Dialogfenster. Darin muss das Dialogfenster letztlich mit "dialog.accept()" oder "dialog.dismiss()" geschlossen werden, da sonst das Browserfenster nicht mehr reagiert. Die "once"-Funktion verhält sich wie die "on"-Funkion, nur das sie lediglich auf das nächste Event reagiert, bevor sich der jeweilige Listener abmeldet.

- .waitForTimeout(timeout)
  Diese Funktion pausiert alle Operationen auf der aktuellen Seite für die in "timeout" gegebene Anzahl von Millisekunden. Es wird davon abgeraten, diese Funktion zu nutzen, da Playwright-Funktionen allgemein von selbst warten. Der Einsatz nach dem Drücken von Knöpfen ließ sich hier aber scheinbar nicht umgehen, da der Testfall beendet wurde, bevor die asynchrone Auswertung des Dialogfensters beendet wurde.

Locator

Von "Page" abgeleitet wird die Klasse "Locator" [PlaywrightAPILocator], welche oben als Zeiger bezeichnet wurde. "Locator" verweisen auf ein oder mehrere Seitenelemente und können mit weiteren Funktionen, welche "Locator" zurückgeben, spezifiziert werden. Auf einem "Locator" können verschiedene Funktionen aufgerufen werden, um mit Seitenelementen zu interagieren. Von diesen wurden in dieser Arbeit die Folgenden verwendet:

- .click()
  Mit dieser Funktion wird auf das - von dem "Locator" definierte - Seitenelement geklickt. Sie unterstützt verschiedene Optionen, um beispielsweise die Häufigkeit oder die zu verwendende Maustaste, zu definieren.

- .fill(value)
  Mit dieser Funktion wird ein Eingabefeld mit dem als "value" übergebenen String gefüllt.

- .getByRole(role, {name})
  Diese Funktion verhält sich hier genauso wie oben beschrieben.

- .getByText(text)
  Diese Funktion verhält sich hier genauso wie oben beschrieben.

- .innerText()
  Diese Funktion gibt den Text eines Seitenelements, sei es der Text einer Überschrift oder die Beschriftung eines Knopfes, als String zurück.

- .nth(index)
  Da - wie oben erwähnt - ein "Locator" auf mehr als ein Seitenelement zeigen kann, kann dieser mit der Funktion ".nth()" auf ein einzelnes Element spezialisiert werden. Dabei werden die Seitenelemente als Liste angenommen, auf die mit der Funktion zugegriffen wird. Die Nummer "index" beschreibt dabei das wievielte Element zurückgegeben werden soll, wobei mit ".nth(0)" auf das erste Element zugegriffen wird.

Assertions

Im Folgenden sollen die verwendeten "Assertions" - also Playwrights Testabfragen - aufgegriffen werden [PlaywrightAPIAssertions]. Diese werden mit der Playwright-Funkion "expect(value)" aufgerufen, wobei "value" keinen eindeutigen Datentypen hat. Bei Fehlschlag eines "expect"-Statements bricht der umgebende Testfall ab und wird als fehlerhaft gekennzeichnet. Auf "expect(value)" können verschiedene Funktionen aufgerufen werden, von welchen die Folgenden in dieser Arbeit verwendet wurden:

- .toBe(expected)
  Diese Funktion vergleicht "value" mit dem übergebenen Wert "expected". Sie ist erfolgreich, wenn beide Werte gleich sind, andernfalls schlägt sie fehl.

- .toBeFalsy()
  Diese Funktion erwartet das "value" als "falsy" ausgewertet wird - schlägt also fehl, ist dies nicht der Fall. "falsy" ist jeder Wert der - bei Nutzung als Boolean Wert - als "false" erkannt wird. Beispiele hierfür sind ein leerer String, "null", "false" oder der Zahlenwert null. Sie ist erfolgreich, wenn beide Werte gleich sind, andernfalls schlägt sie fehl.

- .toBeGreaterThanOrEqual(expected)
  Diese Funktion vergleicht "value" mit dem übergebenen Wert "expected". Dabei wird erwartet, dass beide Werte Zahlen sind. Sie ist erfolgreich, wenn der Wert von "value" größer oder gleich dem Wert von "expected" ist, andernfalls schlägt sie fehl.

- .toBeLessThan(expected)
  Diese Funktion vergleicht "value" mit dem übergebenen Wert "expected". Dabei wird erwartet, dass beide Werte Zahlen sind. Sie ist erfolgreich, wenn der Wert von "value" kleiner als der Wert von "expected" ist, andernfalls schlägt sie fehl.

- .toBeVisible()
  Diese Funktion erwartet, dass es sich bei "value" um einen "Locator" handelt und dieser sichtbar ist. Ist er nicht sichtbar, schlägt die Funktion fehl.

- .toContain(expected)
  Diese Funktion erwartet, dass es ich bei "value" um eine Liste - oder einen vergleichbaren Container - oder einen String handelt. Sie prüft ob "expected" ein Teil des Strings oder ein Element der Liste ist. Ist dies nicht der Fall, schlägt sie fehl.

- .toHaveURL(urlOrRegExp)
  Diese Funktion erwartet, dass "value" ein "Page"-Objekt ist. Der Parameter "urlOrRegExp" kann entweder ein String oder ein regulärer Ausdruck sein. Die Funktion überprüft dann, ob die URL der Seite dem Parameter entspricht. Ist dies nicht der Fall, schlägt sie fehl.

4.3 Verbaute Fehler und erwartetes Fehlverhalten

Hier soll beschrieben werden, welche Fehler in die Anwendung des Coffeemakers eingebracht werden (vgl. Kapitel 3.1), an welchen Stellen im Code des Coffeemakers sie umgesetzt werden, sowie welche Auswirkungen durch sie zu erwarten sind. Dazu gehört auch, welche der Testfälle aus Kapitel 4.1 die neuen Fehler abfangen sollten.

Umgesetzt werden diese Fehlerfälle direkt im Code. Zum Ausführen ist es vorgesehen, dass jeweils alle bis auf einen Fehlerfall auskommentiert sind. Die fehlerhaften Funktionen sind, als Kommentar, entsprechend mit dem unten genannten Namen (Fehler 1 - 8) gekennzeichnet.

Ziel ist es hierbei, dass jede Komponente - jede Funkion - des Coffeemakers mit mindestens einem Fehler versehen wird.

4.3.1 Komponente "Rezepte einsehen"

Die Umsetzung dieser Komponente erfolgt letztlich in den Zeilen 16-20 der Datei ".//src//app//components//recipes//recipes.component.ts". Die Funktion "ngOnInit" wird beim Start der Anwendung ausgeführt und ruft dabei die Funktion "getRecipes" auf. Diese findet sich in den Zeilen 31-33 der Datei ".//src//app//services//machine.service.ts".

Fehler 1
Um hier einen Fehler einzubringen, wird die in "getRecipes" verwendete URL durch ein Suffix ergänzt. Dies simuliert einen Tippfehler in der gegebenen Konstante.

Infolgedessen sollte die Kommunikation zwischen der Datenbank und der Anwendung gestört sein, was dazu führt, dass die Rezepte nicht geladen werden können.

Dies sollte primär in Testfall 4 auffallen, jedoch auch die Funktion der Testfälle 7, 8, 9, 10 und 11 verhindern. All diese Testfälle basieren auf der in Testfall 4 geprüften Sichtbarkeit der Rezepte.

4.3.2 Komponente "Rezept hinzufügen"

Diese Komponente wird in der Funkion "odAddSubmit", in den Zeilen 25-36 der Datei ".//src//app//components//add-recipe//add-recipe.component.ts", umgesetzt.
In dieser wird die Funkion "addRecipe", aus der Datei ".//src//app//services//machine.service.ts" (Zeilen 36-39), aufgerufen.

Fehler 2
Um hier einen Fehler einzubringen wird Zeile 31 der Datei ".//src//app//components//add-recipe//add-recipe.component.ts" von "cocoaAmount: this.cocoaAmountValue ? this.cocoaAmountValue : 0," zu "cocoaAmount: this.coffeeAmountValue ? this.coffeeAmountValue : 0," geändert.

So etwas könnte ebenfalls durch einen Kopierfehler entstehen, wobei die programmierende Person, aufgrund des ähnlichen Aussehens von "cocoa" und "coffee", vergisst, die Namen entsprechend zu ändern, nachdem die Zeile kopiert wurde.

Auffallen sollte dieser Fehler in Testfall 9.

4.3.3 Komponente "Rezept brauen"

Die Umsetzung dieser Komponente beginnt in den Zeilen 18-20 der Datei ".//src//app//components//recipe-item//recipe-item.component.ts". Hier wird die Funkion "brewRecipe", aus der Datei ".//src//app//services//machine.service.ts" (Zeilen 42-51), aufgerufen. Die Funkion "brewRecipe" wiederum ruft die Funktion "enoughIngredientsAvailable" (Zeilen 54-69) auf, welche einen String mit dem Namen der fehlenden Zutat zurückgibt, wenn denn eine fehlt. Ist das der Fall, wird die Funktion "useNeededIngredients" (Zeilen 72-83) aufgerufen, welche mit der Funktion "useIngredient" (Zeile 97-104) die Zutatenwerte aus der Datenbank abzieht.

Fehler 3
Ein zu verbauender Fehler ist das "!=" in der Funkion "brewRecipe" durch ein "==" zu ersetzen.

Das hätte zur Folge, dass jedes braubare Rezept als gebraut gelten würde, während jedes nicht braubare als braubar gälte.

Auffallen sollte dies in den Testfällen 7, 8, 10 und 11. Hier wird jeweils versucht ein Rezept zu brauen, während die richtige Reaktion des Coffeemakers geprüft wird.

Fehler 4
Einen ähnlichen Effekt hätte es, würde man den Standard-Rückgabewert der Funkion "enoughIngredientsAvailable", von der Konstante "NONE_STRING", zu einem leeren String ändern. Dieser Fehler ist nicht sonderlich abwegig, da TypeScript einen leeren String, bei Nutzung als Boolean Wert, als "false" auswertet.

Dies hätte zur Folge, dass alle Brauversuche fehlschlagen.

Auffallen würde dies primär in Testfall 7, da dieser die Fehlermeldungen prüft. Jedoch auch die Testfälle 9, 10 und 11 basieren auf erfolgreichen Brauversuchen und würden entsprechend fehlschlagen.

Fehler 5
Ein weiterer Fehler ist das Austauschen von "-=" zu "+=" in der Funkion "useIngredient".

Dadurch wird beim erfolgreichen Brauen der Füllstand der Zutaten erhöht und nicht verringert.

Auffallen würde dies primär in Testfall 10.

4.3.4 Komponente "Zutaten einsehen"

Die Umsetzung dieser Komponente erfolgt in den Zeilen 17-22 der Datei ".//src//app//components//ingredients//ingredients.component.ts". Die Funktion "ngOnInit" wird beim Start der Anwendung ausgeführt und ruft dabei die Funktion "getIngredients()" auf. Diese findet sich in den Zeilen 117-119 der Datei ".//src//app//services//machine.service.ts".

Fehler 6
Wie oben, in Fehler 1, wird hier die in "getIngredients" verwendete URL durch ein Suffix ergänzt. Dies simuliert einen Tippfehler in der gegebenen Konstante.

Entsprechend sollte die Kommunikation zwischen der Datenbank und der Anwendung gestört sein, was hier dazu führt, dass die Zutaten nicht angezeigt werden können.

Dies sollte primär in Testfall 2 auffallen, jedoch auch die Funktion der Testfälle 3, 6, 7, 8, 10 und 11 verhindern. All diese Testfälle setzen voraus, dass die Zutaten auf der Unterseite sichtbar sind, was in Testfall 2 geprüft wird.

4.3.5 Komponente "Zutat nachfüllen"

Die Umsetzung dieser Komponente beginnt in den Zeilen 22-25 der Datei ".//src//app//components//ingredient-item//ingredient-item.component.ts", in der Funktion "onRefillSubmit". Dort wird die Funkion "refillIngredient", aus der Datei ".//src//app//services//machine.service.ts" (Zeilen 107-114), aufgerufen.

Fehler 7
Ein möglicher Fehler ist hier das Austauschen von "+=" zu "-=" in der Funkion "refillIngredient".

Dies hätte zur Folge, dass beim Nachfüllen von Zutaten die Füllstände abnähmen, statt zu steigen.

Auffallen würde dies primär in Testfall 6, da dieser das Nachfüllen direkt testet. Weiterhin würde aber auch die Testfälle 7, 10 und 11 nicht mehr funktionieren, da diese ebenfalls Zutaten nachfüllen.

Fehler 8
Ein anderer Fehler ist das Addieren von 1 auf den nachzufüllenden Wert in der Funkion "onRefillSubmit".

Das hätte zur Folge, dass immer eine Einheit der Zutat zu viel nachgefüllt würde.

Auffallen sollte dies direkt nur in Testfall 6.

5 Evaluation der Tests und Fehler

Dieses Kapitel befasst sich mit dem Ausführen der Testfälle auf allen Versionen des Testobjektes. Die Versionen sind hierbei die als fehlerfrei angenommene Basisversion, unverändert von Christopher Dührkop, sowie je eine Version pro Fehlerfall (vgl. Kapitel 4.3).

Weiterhin wird jeder Testfall ein Mal im sequenziellen Durchlauf aller Testfälle, so wie ein Mal alleinstehend geprüft. Jeder sequenzielle Durchlauf aller Testfälle, als auch jeder alleinstehende Durchlauf eines Testfalls, gilt hierbei als eine Testausführung. Vor jeder Testausführung wird die Prämisse sichergestellt, dass alle Zutatenwerte in der Datenbank auf null stehen.

5.1 Ergebnisse der Testausführungen

Die Ergebnisse jedes Testdurchlaufs sind in Matrizen festgehalten.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Test 11
Ohne Fehler	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS
Fehler 1	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 2	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS
Fehler 3	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 4	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 5	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	SUCCESS
Fehler 6	SUCCESS	FAILED	FAILED	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 7	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 8	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	FAILED	FAILED	SUCCESS	SUCCESS

Abbildung 5.1: Ergebnisse der sequenziellen Ausführungen aller Tests

In dieser Matrix (Abbildung 5.1) ist, für jede Code- oder Fehlerversion, zu sehen welche Testfälle bei sequenzieller Ausführung aller Testfälle bestanden wurden.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Test 11
Ohne Fehler	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS
Fehler 1	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 2	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS
Fehler 3	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	SUCCESS
Fehler 4	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	FAILED	FAILED	FAILED
Fehler 5	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	SUCCESS
Fehler 6	SUCCESS	FAILED	FAILED	SUCCESS	SUCCESS	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED
Fehler 7	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	FAILED	SUCCESS	FAILED	FAILED	FAILED
Fehler 8	SUCCESS	SUCCESS	SUCCESS	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS	FAILED	SUCCESS	SUCCESS

Abbildung 5.2: Ergebnisse der Einzelausführungen aller Tests

In dieser Matrix (Abbildung 5.2) ist, für jede Code- oder Fehlerversion, zu sehen welche Testfälle bei Einzelausführung bestanden wurden, wurden diese allein - ohne die anderen - ausgeführt.

Im Folgenden soll nun für jede Codeversion beschrieben werden, welche Testfälle den entsprechenden Fehler gefunden haben. Dazu ist zu sagen, dass die als fehlerfrei angenommene Basisversion des Codes auch nicht alle Testfälle bestanden hat. Die Funktion zum Hinzufügen eines neuen Rezeptes scheint nicht korrekt implementiert zu sein. Dies ist auch durch manuelle Tests zu bestätigen. Entsprechend wird Testfall 9 im Folgenden nicht näher erörtert. Alle anderen Tests wurden in der Basisversion bestanden.

Die Ausführung der Tests geschieht jeweils - pro Fehlerfall - für die sequenzielle aller Tests, sowie die Einzelausführung jedes einzelnen Tests.

5.1.1 Sequenzielle Ausführung aller Tests

Im Folgenden eine Darstellung der in Matrix 5.1 festgehaltenen Ergebnisse. Dabei wird die fehlerfreie Version - ebenso wie Testfall 9 - wie oben beschrieben, allgemein außer Acht gelassen.

Fehler 1

Dieser Fehler macht die Rezepte der "Recipes"-Unterseite unerreichbar.
Aufgefallen ist er in der sequenziellen Ausführung aller Tests in den Testfällen 4, 7, 8, 10 und 11.

Wie in Kapitel 4.3.1 erwartet, ist der Fehler in Testfall 4 aufgefallen und hat ebenfalls die, von der Sichtbarkeit der Rezepte abhängigen, Testfälle 7 bis 11 ausgelöst.

Fehler 2

Dieser Fehler verhindert das korrekte Anlegen von neuen Rezepten.
Er hätte in Testfall 9 auffallen sollen; da dessen Funktion allerdings nicht sichergestellt werden kann (siehe oben), wird auch dieser Fehler obsolet.

Fehler 3

Dieser Fehler invertiert das Brauverhalten. Entsprechend wird gebraut was nicht genügend Zutaten hat, während alles mit genügend Zutaten nicht gebraut werden kann.
Aufgefallen ist er in der sequenziellen Ausführung aller Tests in den Testfällen 7, 8, 10 und 11.

Die Tests 7 und 8 prüfen auf die Erfolgs- oder Misserfolgsmeldung, bekommen jedoch durch den Fehler das Gegenteil von dem, was sie erwarten. Test 10 füllt die Zutaten auf und erwartet folgend ebenfalls einen erfolgreichen Brauvorgang. Testfall 11 schlägt hier fehl, da die Zutatenfüllstände zu Beginn des Tests nicht, wie erwartet, auf null stehen. Durch den Fehler schlägt das Brauen allerdings dennoch fehl, was dazu führt, dass nach einer nicht existierenden Zutat "none" gesucht wird, dies verursacht dann einen Fehler im Test.

Wie in Kapitel 4.3.3 erwartet ist der Fehler in den Testfällen 7, 8, 10 und 11 aufgefallen, die jeweils versuchen ein Rezept zu brauen, während sie die richtige Reaktion des Coffeemakers prüfen.

Fehler 4

Dieser Fehler sorgt dafür, dass jeder Brauversuch fehlschlägt.
Aufgefallen ist er in der sequenziellen Ausführung aller Tests in den Testfällen 7, 8, 10 und 11.

Die Erwartung in Kapitel 4.3.3 war, dass der Fehler in den Testfällen 7, 10 und 11 auffallen würde, da diese erfolgreiche Brauversuche voraussetzen. Unerwartet ist, dass

der Fehler ebenfalls in Test 8 aufgefallen ist. Dies liegt allerdings daran, dass nach dem Fehlschlag von Test 7 die Zutatenfüllstände nicht mehr auf null stehen, was in Test 8 überprüft wird und weshalb dieser folglich fehlschlägt.

Fehler 5

Dieser Fehler sorgt dafür, dass beim Brauen die Zutaten aufgefüllt statt verbraucht werden.
Er wird in der sequenziellen Ausführung aller Tests in den Testfällen 8 und 10 gefunden.

Wie in Kapitel 4.3.3 erwartet, ist der Fehler in Testfall 10 aufgefallen. Weiterhin schlägt aber auch Testfall 8 fehl. Dies liegt daran, dass er darauf angewiesen ist, dass die Zutaten zu Beginn des Testfalls leer sind, was nach Test 7 nicht mehr gegeben wäre.

Fehler 6

Dieser Fehler macht die Zutaten auf der "Ingredients"-Unterseite unerreichbar.
Er wird in der sequenziellen Ausführung aller Tests in den Testfällen 2, 3, 6, 7, 8, 10 und 11 gefunden.

Wie in Kapitel 4.3.4 erwartet, fällt der Fehler in Test 2 auf. Ebenfalls wie erwartet, fällt er in den Testfällen 3, 6, 7, 8, 10 und 11 auf, da diese auf die Rezepte zugreifen müssen.

Fehler 7

Dieser Fehler sorgt dafür, dass beim Nachfüllen von Zutaten die Werte verringert anstatt erhöht werden.
Er wird in der sequenziellen Ausführung aller Tests in den Testfällen 6, 7, 10 und 11 gefunden.

Wie in Kapitel 4.3.5 erwartet, fällt der Fehler in Test 6 auf, da hier das korrekte Nachfüllen getestet wird. Ebenfalls wie erwartet, fällt er in den Testfällen 7, 10 und 11 auf, da auch diese Zutaten nachfüllen.

Allerdings fällt hier nicht auf, dass die Werte falsch addiert werden. Stattdessen kann die Füllstandsanzeige nicht mehr erkannt werden, da sie nach dem ersten Nachfüllen negative

Werte enthält und das Keyword, welches verwendet wird, um sie auf der Website zu finden, nicht in der Lage ist, negative Werte zu erkennen.

Aus diesem Grund schlägt hier unerwarteterweise auch Testfall 8 fehl, denn auch dieser versucht den Füllstand auszulesen.

Fehler 8

Dieser Fehler addiert 1 zu jedem Nachfüllvorgang.
In der sequenziellen Ausführung aller Tests fällt dieser Fehler nur in den Testfällen 6 und 8 auf.

Das Auffallen in Testfall 6 geschieht wie in Kapitel 4.3.5 erwartet, da hier überprüft wird, ob die korrekten Werte nachgefüllt werden.

Das auch Testfall 8, unerwarteterweise, diesen Fehler bemerkt liegt daran, dass nach den vorangegangenen Tests die Füllstände nicht mehr auf null stehen, da bei jedem Nachfüllen der nachgefüllte Wert um 1 größer ist als er sein sollte.

5.1.2 Einzelausführung aller Tests

Im Folgenden eine Darstellung der in Matrix 5.2 festgehaltenen Ergebnisse. Dabei wird die fehlerfreie Version - ebenso wie Testfall 9 - wie oben beschrieben, allgemein außer Acht gelassen.

Fehler 1

Dieser Fehler macht die Rezepte der "Recipes"-Unterseite unerreichbar.
Aufgefallen ist er in der Einzelausführung der Tests in den Testfällen 4, 7, 8, 10 und 11.

Wie in Kapitel 4.3.1 erwartet, ist der Fehler in Testfall 4 aufgefallen und hat ebenfalls die, von der Sichtbarkeit der Rezepte abhängigen, Testfälle 7 bis 11 ausgelöst.

Fehler 2

Dieser Fehler verhindert das korrekte Anlegen von neuen Rezepten.
Er hätte in Testfall 9 auffallen sollen; da dessen Funktion allerdings nicht sichergestellt werden kann (siehe oben), wird auch dieser Fehler obsolet.

Fehler 3

Dieser Fehler invertiert das Brauverhalten, entsprechend wird gebraut was nicht genügend Zutaten hat, während alles mit genügend Zutaten nicht gebraut werden kann.
In den Einzelausführungen fällt er in den Tests 7, 8 und 10 auf.

Die Tests 7 und 8 prüfen auf die Erfolgs- oder Misserfolgsmeldung, bekommen jedoch durch den Fehler das Gegenteil von dem, was sie erwarten. Test 10 füllt die Zutaten auf und erwartet folgend ebenfalls einen erfolgreichen Brauvorgang.

Wie in Kapitel 4.3.3 erwartet ist der Fehler in den Testfällen 7, 8 und 10 aufgefallen, die jeweils versuchen ein Rezept zu brauen, während sie die richtige Reaktion des Coffeemakers prüfen. Allerdings ist der Fehler in der Einzelausführung nicht in Testfall 11 aufgefallen, was erwartet wurde. Der Grund hierfür ist, dass die Zutaten zu Beginn des Tests auf null standen und der Brauversuch also fehlschlagen sollte. Da das Brauverhalten allerdings durch den Fehler invertiert wurde, gelang der Brauversuch und der Test terminierte erfolgreich.

Fehler 4

Dieser Fehler sorgt dafür, dass jeder Brauversuch fehlschlägt.
Aufgefallen ist er in den Einzelausführungen der Tests 7, 10 und 11.

Wie in Kapitel 4.3.3 erwartet, ist der Fehler in den Testfällen 7, 10 und 11 aufgefallen, da diese erfolgreiche Brauversuche voraussetzen.

Fehler 5

Dieser Fehler sorgt dafür, dass beim Brauen die Zutaten aufgefüllt statt verbraucht werden.
Er wird in der Einzelausführung nur von Testfall 10 gefunden. Dieser überprüft, ob die Füllstände nach einem Nachfüllvorgang korrekt sind.

Dies ist wie in Kapitel 4.3.3 erwarten.

Fehler 6

Dieser Fehler macht die Zutaten auf der "Ingredients"-Unterseite unerreichbar.
Er fällt in den Einzelausführungen in den Tests 2, 3, 6, 7, 8, 10 und 11 auf.

Wie in Kapitel 4.3.4 erwartet, fällt der Fehler in Test 2 auf. Ebenfalls wie erwartet, fällt er in den Testfällen 3, 6, 7, 8, 10 und 11 auf, da diese auf die Rezepte zugreifen müssen.

Fehler 7

Dieser Fehler sorgt dafür, dass beim Nachfüllen von Zutaten die Werte verringert anstatt erhöht werden.
Er fällt in den Einzelausführungen der Tests 6, 7, 10 und 11 auf.

Wie in Kapitel 4.3.5 erwartet, fällt der Fehler in Test 6 auf, da hier das korrekte Nachfüllen getestet wird. Ebenfalls wie erwartet, fällt er in den Testfällen 7, 10 und 11 auf, da auch hier Zutaten nachgefüllt werden.

Allerdings fällt hier nicht auf, dass die Werte falsch addiert werden. Stattdessen kann die Füllstandsanzeige nicht mehr erkannt werden, da sie nach dem ersten Nachfüllen negative Werte enthält und das Keyword, welches verwendet wird, um sie auf der Website zu finden, nicht in der Lage ist, negative Werte zu erkennen.

Fehler 8

Dieser Fehler addiert 1 zu jedem Nachfüllvorgang.
Er fällt in der Einzelausführung nur in Testfall 6 auf.

Dies entspricht der Erwartung aus Kapitel 4.3.5.

5.2 Bewertung der Testfälle

Hier sollen die Testfälle bewertet werden.
Dafür werden die in Kapitel 1.2.1 genannten Kriterien angewendet. Da die Lesbarkeit allerdings eine sehr subjektive Bewertung ist und ich die von mir gewählten Keywordnamen, in meinen Augen, so gut lesbar wie möglich gewählt habe, wird die Lesbarkeit hier nicht näher betrachtet. Die zu erwartenden Fehler sind Kapitel 4.3 zu entnehmen.

Numerisch werden die Testfälle anhand folgender Formel bewertet:

$$\frac{\text{Gefundene erwartete Fehler}}{\text{Erwartete Fehler}} - \frac{\text{Gefundene nicht erwartete Fehler}}{\text{Nicht erwartete Fehler}} \tag{5.1}$$

Dabei ist 1,0 das beste und 0,0 das schlechteste Ergebnis.

Testfall 1
Dieser Testfall prüft die grundsätzliche Erreichbarkeit des Webinterface.
Er findet keinen der Fehler. Gleichzeitig beeinflusst keiner der Fehler die generelle Erreichbarkeit des Webinterface, da dadurch alle Testfälle immer fehlschlagen würden. Die Sinnhaftigkeit dieses Testfalls ist damit zu hinterfragen, seine Funktionalität lässt sich hier aber nicht bewerten.

Testfall 2
Dieser Testfall überprüft die Sichtbarkeit der Zutaten auf der "Ingredients"-Unterseite.
Er hat Fehler 6 gefunden, der den Zugriff auf die Zutaten verhindert, was erwartet wurde.

Testfall 2 hat somit alle von ihm erwarteten Fehler gefunden und wurde von keinem anderen Fehler ausgelöst. Er hat also einen von einem erwarteten Fehler und null der

restlichen sieben Fehler gefunden. Numerisch heißt das, dass er ein Ergebnis von 1,0 erreicht hat und damit sehr gut ist.

Testfall 3

Dieser Testfall überprüft, dass die Zutaten auf der "Ingredients"-Unterseite auf null stehen.
Er findet Fehler 6, welcher den Zugriff auf die Zutaten verhindert. Dies war zwar zu erwarten, wird aber nicht nur dann ausgelöst, wenn die Zutaten nicht auf null stehen, sondern auch, wenn sie generell nicht zu sehen sind.

Testfall 3 hat damit zwar den von ihm erwarteten Fehler gefunden, dies geschieht aber nicht, weil er seine eigentliche Funkion erfüllt, sondern weil er durch einen äußeren Einfluss ausgelöst wird.

Weil keiner der Fehler die Prämisse der Tests verändert, kann Test 3 nicht wirklich effektiv bewertet werden. Da er in einer manuellen Ausführung, mit entsprechend veränderter Prämisse - nichtleeren Zutaten in der Datenbank - allerdings anschlägt, würde ich ihn dennoch als nicht schlecht beschreiben, auch wenn keine Bewertung errechnet werden kann.

Testfall 4

Dieser Testfall überprüft die Sichtbarkeit der Standard-Rezepte auf der "Recipes"-Unterseite. Er hat Fehler 1 gefunden, der den Zugriff auf die Rezepte verhindert.

Damit hat Testfall 4 einen von einem erwarteten Fehler und null der restlichen sieben Fehler gefunden. Numerisch heißt das, dass er ein Ergebnis von 1,0 erreicht hat und damit sehr gut ist.

Testfall 5

Dieser Testfall prüft die Sichtbarkeit der Eingabemaske für das Erstellen neuer Rezepte. Er findet keinen der Fehler. Allerdings beeinflusst auch keiner der Fehler die Sichtbarkeit der Eingabemaske, weshalb dieses Ergebnis zu erwarten war. Dennoch würde ich diesen Testfall nicht als sinnlos bezeichnen, seine Funktionalität lässt sich hier aber dennoch nicht bewerten.

Testfall 6

Dieser Testfall überprüft, dass die richtigen Zutatenmengen beim Nachfüllen ergänzt werden.
Hier wurden die Fehler 6, 7 und 8 gefunden. Diese verhindern entweder den Zugriff auf die Zutaten oder sabotieren das Nachfüllen.

Mit Finden der Fehler 7 und 8 hat Testfall 4 zwei von zwei erwarteten Fehlern. Durch das Finden von Fehler 6 findet er allerdings auch einen der sechs nicht erwarteten Fehler. Numerisch heißt das, dass er ein Ergebnis von 0,8333 erreicht hat und damit gut ist.

Testfall 7

Dieser Testfall füllt Zutaten nach und überprüft den Brauprozess auf die zu erwartende Erfolgsmeldung.
Er findet die Fehler 1, 3, 4, 6 und 7, wobei das Finden der Fehler 3, 4 und 7 erwartet ist. Dabei sabotieren 3 und 4 den Brauvorgang, während 7 den Nachfüllprozess invertiert.

Fehler 7 wird allerdings nicht gefunden, weil falsche Zutatenwerte detektiert werden, sondern weil die Zutatenfüllstände mit negativen Werte nicht mehr identifiziert werden können. Damit sollte das Finden von Fehler 7 nicht als Erfolg gewertet werden.

Das Finden der Fehler 1 und 6, welche die Zutaten oder Rezepte gänzlich verbergen, ist zwar nicht unerwartet, aber dennoch unerwünscht.

Folglich hat Testfall 7 zwei von drei erwarteten Fehler, sowie zwei von verbleibenden fünf nicht erwarteten Fehler gefunden. Numerisch heißt das, dass er ein Ergebnis von 0,2666 erreicht hat, womit er nicht als gut beschrieben werden kann.

Testfall 8

Dieser Testfall prüft die Zutatenfüllstände und überprüft den Brauprozess auf die zu erwartenden Fehlermeldungen.
Er findet die Fehler 1, 3 und 6 in Einzelausführung und die Fehler 1, 3, 4, 5, 6, 7, und 8 in der sequenziellen Ausführung aller Tests.

Das Finden von Fehler 3 ist erwartet, da hier das Brauverhalten invertiert wird.

Das die Fehler 1 und 6 gefunden werden ist zwar unerwünscht, aber zu erwarten, da sie den Zugriff auf die Rezepte oder Zutaten verhindern.

Die Fehler 4, 5, 7 und 8 sollten von diesem Testfall nicht gefunden werden können. Weiterhin fällt es negativ auf, dass dieser Testfall unterschiedliche Fehler findet, abhängig davon, ob er einzeln ausgeführt wird, oder in der sequenziellen Folge aller Tests.

Folglich hat Testfall 8 einen von einem erwarteten Fehler, sowie zwei oder sechs der sieben nicht erwarteten Fehler gefunden. Numerisch heißt das, dass er ein Ergebnis von 0,7143 oder 0,1429 erreicht hat - wobei ich den Fokus hier auf das schlechtere Ergebnis legen würde - womit er als schlecht gewertet werden muss.

Testfall 9

Dieser Testfall prüft, ob mit der Eingabemaske für das Erstellen neuer Rezepte neue Rezepte korrekt angelegt werden können.
Er wird von allen Fehlern ausgelöst. Wie einleitend in Kapitel 5.1 gesagt, ist die Implementierung dieser Komponente fehlerhaft.

Damit kann der Testfall nicht sinnvoll bewertet werden.

Testfall 10

Dieser Testfall prüft die Zutatenfüllstände nach einem Brauvorgang.
Hier wurden die Fehler 1, 3, 4, 5, 6 und 7 gefunden, wobei das Finden der Fehler 3 und 5 erwartet war, da diese den Zutatenverbrauch beim Brauen, oder den Wert beim Nachfüllen verändern.

Das die Fehler 1 und 6 gefunden werden ist zwar unerwünscht, aber zu erwarten, da sie den Zugriff auf die Rezepte oder Zutaten verhindern. Fehler 4 macht es unmöglich, einen Brauversuch erfolgreich abzuschließen, was - auch wenn es nicht Ziel des Tests ist - hier jedoch vorausgesetzt wird. Fehler 7 sorgt dafür das beim Nachfüllen die Füllstände sinken anstatt zu steigen. Dies führt zu negativen Füllständen welche folgend nicht mehr ausgelesen werden können, was dann einen Fehler im Test zur Folge hat.

Folglich hat Testfall 10 zwei von zwei erwarteten Fehlern, sowie vier der sechs nicht erwarteten Fehler gefunden. Numerisch heißt das, dass er ein Ergebnis von 0,3333 erreicht hat. Damit kann man ihn als bestenfalls mittelmäßig beschreiben.

Testfall 11

Dieser Testfall stellt einen vollständigen Betriebsvorgang dar, mit Brauversuchen und Nachfüllen fehlender Zutaten.
Er findet die Fehler 1, 3, 4, 6 und 7, wobei das Finden von Fehler 3 erwartet ist. Dieser wird dabei nur in der sequenziellen Ausführung aller Tests, nicht aber in der Einzelausführung gefunden.

Das Finden der Fehler 1 und 6 ist auch weiterhin unerwünscht, wenn auch nicht unerwartet. Fehler 7 fällt auch hier dadurch auf, dass er negative Werte in der Datenbank verursacht. Dies führt zu einem Fehler, da die - nun negativen - Füllstände nicht mehr erkannt werden können. Fehler 4 sorgt dafür, dass alle Brauversuche fehlschlagen. Test 11 setzt jedoch mindestens einen erfolgreichen Brauversuch voraus.

Nebst diesen unerwartet gefundenen Fehlern fällt es weiterhin negativ auf, dass dieser Testfall unterschiedliche Fehler findet, abhängig davon, ob er einzeln ausgeführt wird, oder in der sequenziellen Folge aller Tests.

Folglich findet Testfall 11 den einen erwarteten, sowie vier der sieben nicht erwarteten Fehler. Numerisch heißt das, dass er ein Ergebnis von 0,4286 erreicht hat. Damit würde ich ihn als mittelmäßig beschreiben.

6 Fazit/Ausblick

Im Folgenden soll die Fragestellung aus Kapitel 1.1 evaluiert werden. Weiterhin sollen die Probleme, die beim Erstellen der Arbeit aufgetreten sind aufgegriffen und kurz erläutert werden.

6.1 Evaluation der Fragestellung

Die Fragestellung dieser Arbeit war, ob sich zum einen Playwright für Keyword-Driven Testing eignet und zum anderen, ob sich Keyword-Driven Testing für End-to-End Tests einsetzen lässt.

Die in dieser Arbeit geschriebenen Keywords legen in meinen Augen klar dar, dass es ohne Probleme möglich ist, Keyword-Driven Testing mit Playwright umzusetzen. Playwright bietet hier ein gut brauchbares, vorimplementiertes Test-Interface-Layer, mit einer Vielzahl von Low-Level Keywords. Weiterhin bietet Playwright den Vorteil nicht nur in TypeScript, sondern auch in anderen Sprachen zur Verfügung zu stehen. Auch wenn sich die Befehle, auf Grund verschiedener Syntax, natürlich unterscheiden, ist es dennoch leicht möglich sie von einer in eine andere Sprache zu übersetzen. Dazu bietet die Einbindung von Playwright in Visual Studio Code eine gute Umgebung zum Starten der Tests.

Keyword-Driven Testing ist hervorragend geeignet, um End-to-End Tests zu schreiben. Keywords können hierbei durch ihren Namen allein ihre Funktion beschreiben und sind somit auch für Laien les- und schreibbar. Die weitergehende Unterteilung - in immer konkretere und technischere Keywords - ist danach ebenfalls für alle Programmierer*innen möglich. Dies erlaubt in einem größeren Team eine bessere Arbeitsteilung, auch wenn es natürlich bedeutet, dass die Organisation und Bereinigung der Keywords ein zusätzlicher Aufwand ist.

Der Einführungsaufwand für Keyword-Driven Testing ist nicht unerheblich. Das Aufstellen der Regeln, sowie das Warten der Keywords im Nachhinein kosten Zeit, die sonst anderweitig genutzt werden könnte. Dieser initiale Aufwand skaliert jedoch nicht wirklich mit der Größe eines Projektes, sondern eher mit der Größe des Teams, das in die Methode eingeführt werden muss.

Die dadurch - im Verlauf des Projektes - gesparte Zeit sollte dies, bei einem hinreichend großen Projekt, aber wieder wettmachen. Dazu kommt, wenn erst einmal eine Methodik für Keyword-Driven Testing entwickelt ist, sollte es nicht schwerfallen diese in weiteren Projekten zu nutzen. Die grundsätzlichen Arbeitsabläufe, sowie die Organisation von Layern und Levels, sollte sich ohne weiteres übertragen lassen. Auch die Syntax dürfte keine zu großen Anpassungen benötigen. Was verbleibt, ist das Anpassen und Definieren der für die Businesslogik (Domain-Layer) oder Anwendung (Navigation-Layer) spezifischen Keywords.

Der in Kapitel 2.3.3 angesprochene Vorteil der Wartbarkeit wurde in dieser Arbeit ebenfalls deutlich. Nach dem initialen Schreiben der Keywords fielen immer wieder Codesegmente auf, die an mehreren Stellen vorkamen. Dies war allgemein sehr simpel und schnell umzusetzen. Dazu kommt, dass Fehler im Code allgemein einem sehr direkten Pfad durch die Keyword-Hierarchie folgen. Da die Keywords selbst meist sehr kurz sind - häufig laufen sie hier auf nur eine Zeile hinaus - ist ein Fehler darin auch leichter zu finden.

6.2 Probleme

Die meisten Probleme sind beim Einrichten der Software aufgetreten. Am schwierigsten war es dabei, die veralteten Versionen der im Coffeemaker verwendeten Dependencies zu updaten, ohne dabei den Code zu beschädigen.

Eine andere Problemquelle war die Implementierung des Coffeemakers.
Beispielsweise war es relativ aufwendig herauszufinden, auf welche Art und Weise die Brau-Erfolgs- oder Misserfolgs-Meldung ausgelesen werden kann. Dabei war allerdings nicht ersichtlich, dass es sich hier nicht um Pop-ups, sondern um Browser-Alerts handelte. Es ist also definitiv sinnvoll sich, zumindest kurz, mit der Vielzahl von verschiedenen Browser-Alerts auseinander zu setzen, damit man weiß, womit man es zu tun hat.

Ein ähnliches Problem war, dass die Knöpfe zum Brauen von Rezepten und die zum Nachfüllen von Zutaten, nicht auf die gleiche Weise implementiert sind. Die Brau-Knöpfe lassen sich mit der Playwright-Funktion ".getByRole('button', name)" finden, während die Nachfüll-Knöpfe nur über ihre ID zu finden sind.

Letztere haben auch noch das Problem, das sie zwar deaktiviert sein können, dies aber nicht über einen Wert in ihren Attributen festgehalten wird. Stattdessen wird, meines Erachtens, das Attribut deklariert aber nicht mit einem Wert definiert. Dieser fehlende Wert wird in Playwright als leerer String interpretiert. Das gleiche Ergebnis gibt es allerdings auch, wenn das Attribut nicht deklariert ist. Entsprechend habe ich keinen Weg gefunden im Test herauszufinden, ob ein solcher Knopf deaktiviert ist oder nicht.

Rückblickend wäre es besser gewesen, die Testfälle nicht erst alle zu konzeptionieren und dann nacheinander zu implementieren, sondern jeden Testfall direkt nach dem Konzeptionieren auch zu implementieren. Dadurch hätte die Erfahrung aus dem Erstellen des einen Testfalls, in der Konzeption des nächsten geholfen.

Vielleicht wäre es auch einfacher gewesen, wäre die Unterteilung in Layer und Level eindeutig gewesen. Das hieße hier, es gäbe nur High-Level Domain-Layer, Intermediate-Level Navigation-Layer und Low-Level Test-Interface-Layer Keywords. Die anderen Bereiche wie Low- und High-Level Navigation-Layer fielen dann weg.

Es viel mir unerwartet schwer mich vollständig an die gesetzten Richtlinien zu halten. Meistens misslang dies, wenn nicht eindeutig war, was ins Intermediate-Level gehörte. Für eine derart kleine Anwendung war dieses zusätzliche Level eher hinderlich. Besser wäre es gewesen, nur High-Level Domain-Layer und Low-Level Navigation-Layer Keywords zu haben, welche dann die Low-Level Test-Interface-Layer Keywords kapseln.

Weiterhin sind zu lange Keywords nicht hilfreich. Sie erlauben zwar eine genaue Beschreibung ihrer Funktion, sind aber generell schwer zu lesen. Wenn Funktionalitäten scheinbar direkt verbunden sind, wie zum Beispiel in den Keywords der Kategorie "checkThatIngredientLevelsAreSufficientToBrew<X>OrStockUp" (vgl. Kapitel 4.2.1), wäre es sinnvoller zwei Keywords daraus zu bauen. Diese könnten hier zum Beispiel "checkIfIngredientLevelsAreSufficientToBrew<X>" und "stockUpIngredientsForRecipe<X>" sein, wobei ersteres keine Prüfung über die "expect"-Playwright-Funkion ausführen dürfte, da diese sonst bei einem Fehlschlag den Test unterbricht.

Es kann argumentiert werden, dass Bindewörter ebenfalls zu vermeiden sind. Ich halte sie dennoch für sinnvoll, da sie etwaige Ungenauigkeiten vermeiden.

Rückblickend wäre es wahrscheinlich auch möglich gewesen, die Keywords noch weiter zu optimieren. Beispielsweise hätte das Finden der Knöpfe mit Optionen wie "hasText" - die es ermöglicht den sichtbaren Text auf dem Knopf ebenfalls abzufragen - einfacher sein können. Weiterhin ist die Verwendung von "waitForTimeout" - einer Funkion, von deren Verwendung in der Dokumentation abgeraten wird - etwas, dass mich stört. Leider habe ich keine einfache Möglichkeit gefunden die Testfälle davon abzuhalten sich zu beenden, bevor die Auswertung der Browser-Alerts beendet wurde. Eventuell wäre dies möglich gewesen, indem auf eine Referenz auf das Dialogfenster gewartet würde. Dies sähe dann ähnlich aus wie die Rückgabewerte des Keywords "tryToBrewAmericanoAndRefillMissingIngredients" (vgl. Kapitel 4.2.2).

In Hinsicht auf Test 11 (vgl. Kapitel 5.2) wäre es wahrscheinlich logischer - sicher aber einfacher - würde nicht die für das Rezept benötigte Zutatenmenge nachgefüllt, sondern ein konstanter Wert, ein imaginärer Löffel Kaffee oder ein Schluck Wasser. Das hätte die Logik des Keywords "tryToBrewAmericanoAndRefillMissingIngredients" (vgl. Kapitel 4.2.2) deutlich vereinfacht, da ein Großteil des Code-Volumens aus der Identifikation der korrekten Zutatenmenge besteht.

Literaturverzeichnis

[AgileCycleModel] : *Agile Model*. – URL https://www.javatpoint.com/software-engineering-agile-model. – Zugriffsdatum: 2023-12-01

[Angular] : *Angular - What is Angular?*. – URL https://angular.io/guide/what-is-angular. – Zugriffsdatum: 2023-09-09

[PlaywrightAPIAssertions] : *Assertions | Playwright*. – URL https://playwright.dev/docs/api/class-locator. – Zugriffsdatum: 2023-12-02

[EndToEndTesting] : *E2E Testing - Code With Engineering Playbook*. – URL https://microsoft.github.io/code-with-engineering-playbook/automated-testing/e2e-testing/. – Zugriffsdatum: 2023-11-23

[TypeScriptStyleGuide] : *Google TypeScript Style Guide*. – URL https://google.github.io/styleguide/tsguide.html#identifiers. – Zugriffsdatum: 2023-12-01

[PlaywrightAPILocator] : *Locator | Playwright*. – URL https://playwright.dev/docs/api/class-locator. – Zugriffsdatum: 2023-12-02

[PlaywrightAPIPage] : *Page | Playwright*. – URL https://playwright.dev/docs/api/class-page. – Zugriffsdatum: 2023-12-02

[PlaywrightAPI] : *Playwright Library | Playwright*. – URL https://playwright.dev/docs/api/class-playwright. – Zugriffsdatum: 2023-12-02

[VSCodePlaywrightExtension] : *Playwright Test for VSCode - Visual Studio Marketplace*. – URL https://marketplace.visualstudio.com/items?itemName=ms-playwright.playwright

[PlaywrightGitHubReleases] : *Releases · Microsoft/Playwright*. – URL https://github.com/microsoft/playwright/releases?page=12. – Zugriffsdatum: 2023-09-25

[GitHubCoffeemaker 2023] : *GitHub - Orovo/angular-coffeemaker-kdt-playwright*. 12 2023. – URL https://github.com/Orovo/angular-coffeemaker-kdt-playwright/tree/main

[Daigl und Rohner 2022] Daigl, Matthias ; Rohner, René: *Keyword-Driven Testing: Grundlage für effiziente Testspezifikation und Automatisierung*. 1. dpunkt.verlag GmbH, 3 2022

[Dührkop 2021] Dührkop, Christopher: *Auf Cypress basierendes schichtenübergreifendes Testkonzept für Webapplikationen im Kontext von Smart-Home-Anwendungen*. 12 2021

[Sommerville 2018] Sommerville, Ian: *Software Engineering*. 10. Pearson Studium, 10 2018

[Spillner und Linz 2019] Spillner, Andreas ; Linz, Tilo: *Basiswissen Softwaretest*. 6. dpunkt.verlag, 6 2019

[Wittner] Wittner, Michael: *Wie Mutationstests die Software-Testqualität verbessern*. – URL https://www.all-electronics.de/elektronik-entwicklung/wie-mutationstests-die-software-testqualitaet-verbessern.html. – Zugriffsdatum: 2023-03-02

A Anhang

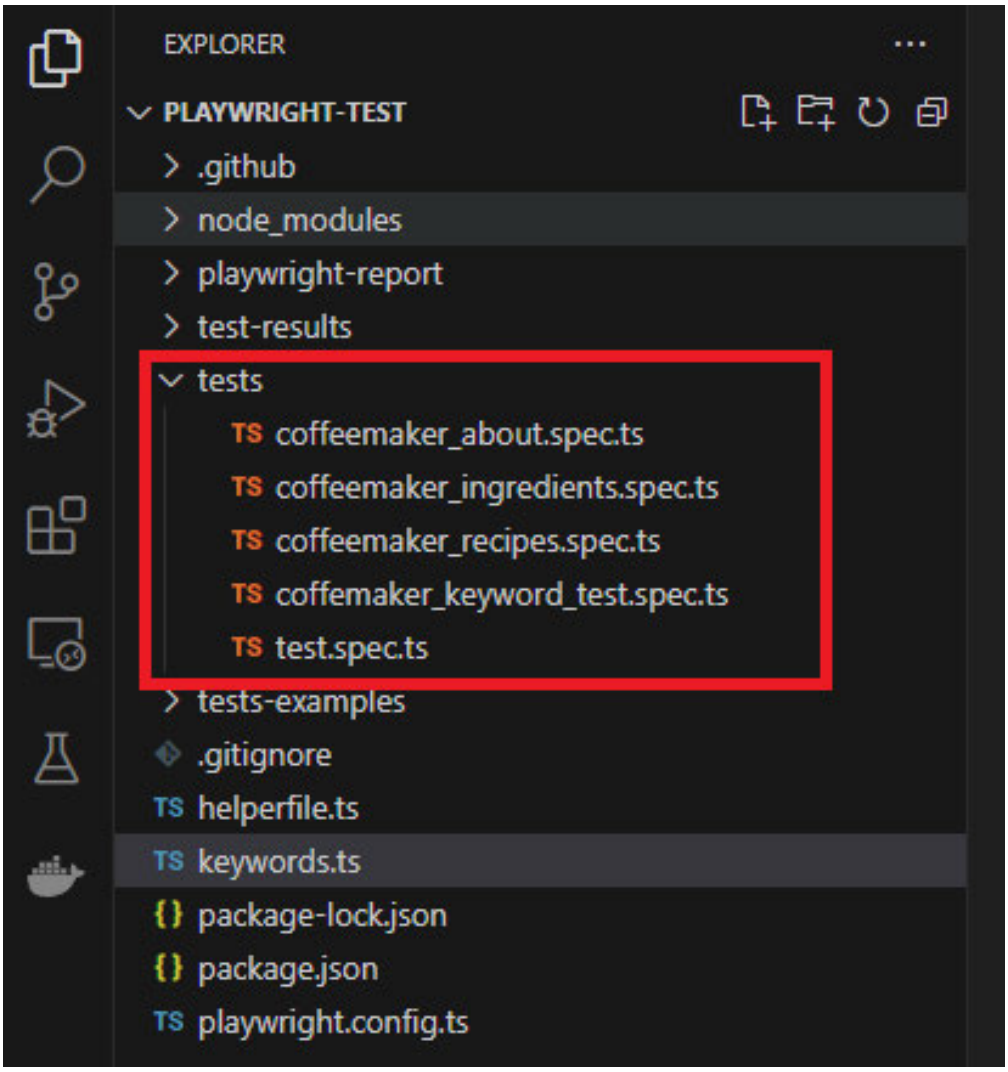

Abbildung A.1: Testdateien für Playwright im Dateiexplorer von Visual Studio Code

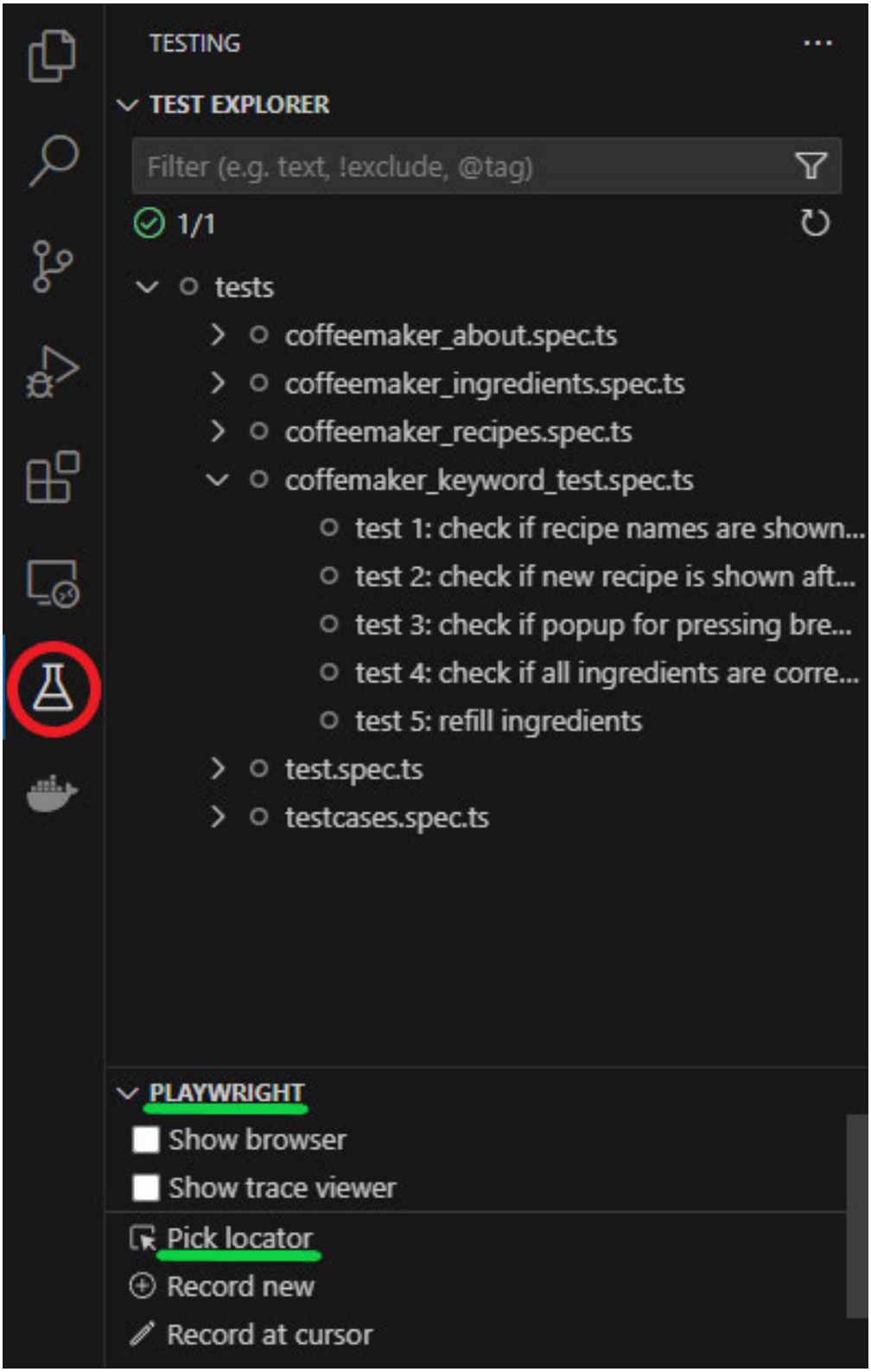

Abbildung A.2: Tests in der Testing-Kategorie (rot) von Visual Studio Code, sowie der Playwright-"Pick locator" (grün)

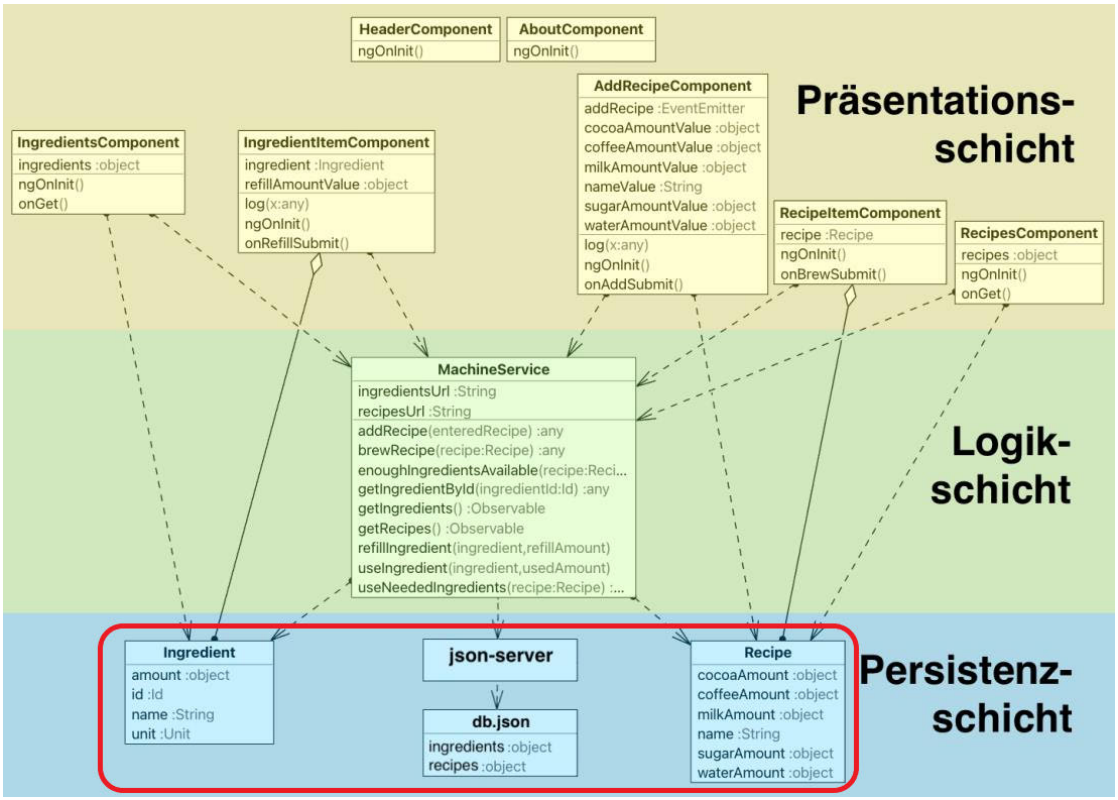

Abbildung A.3: Klassendiagramm des Coffeemakers von Christopher Dührkop [Dührkop, 2021, S. 62, Abbildung 12]; rot markiert sind die Klassen die in dieser Arbeit hier keine Bewandtnis haben

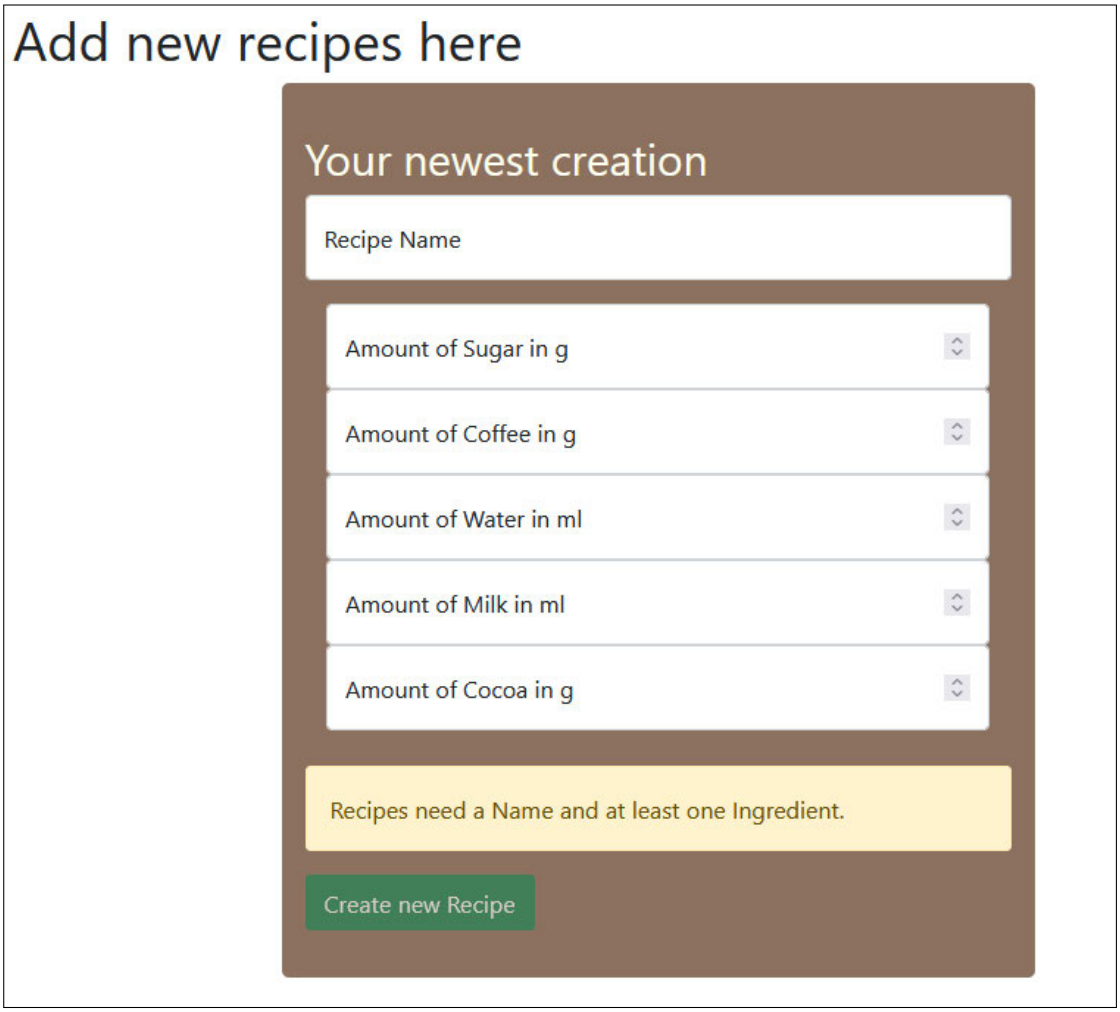

Abbildung A.4: Eingabemaske der "Add Recipe"-Komponente des Coffeemakers

Abbildung A.5: "Recipes"-Komponente des Coffeemakers mit den Standard-Rezepten

Feature	Methodenname	Datei und Zeile	Mögliche einbaubare Fehler	Erwarteter Fehler
Rezepte sehen	RecipesComponent.ngOnInit()	recipes.component.ts:16		
	> MachineService.getRecipes()	machine.service.ts:32	Ergänze URL um Präfix oder Suffix	URL für Rezepte kann nicht gefunden werden, daher keine Rezepte anzeig- und braubar
Rezept hinzufügen	AddRecipeComponent.onAddSubmit()	add-recipe.component.ts:25	Entferne Zeile 35 (location.reload())	Seite wird nicht neu geladen, neues Rezept wird nicht angezeigt
			Vertausche oder weise Werte an falscher Stelle zu	Falsche Werte in Rezept, daher Brauverhalten nicht wie vorgesehen
	> MachineService.addRecipe(Recipe)	machine.service.ts:36		
Rezept brauen	RecipeItemComponent.onBrewSubmit()	recipe-item.component.ts:18		
	> MachineService.brewRecipe(Recipe)	machine.service.ts:42	Ändere '!=' zu '==' in Zeile 45	Brauverhalten ist invertiert
	> > MachineService.enoughIngredientsAvailable(Recipe)	machine.service.ts:54	Ändere 'result' in Zeile 55 von 'NONE_STRING' zu bspw. "None_Missing" oder ""	Brauvorgang kann niemals gelingen
	> > > MachineService.getIngredientById(Id)	machine.service.ts:90		
	> > MachineService.useNeededIngredients(Recipe)	machine.service.ts:72	Addiere 1 auf Werte (bswp. 'coffeeAmount')	Falscher Wert wird abgezogen, resultiert in negativen Werten in Datenbank
	> > > MachineService.getIngredientById(Id)	machine.service.ts:90		
	> > > MachineService.useIngredient(Ingredient, number)	machine.service.ts:97	Ändere '-=' zu '+=' in Zeile 100	Werte in Datenbank steigen nur noch
Zutaten einsehen	IngredientsComponent.ngOnInit()	ingredients.component.ts:17		
	> MachineService.getIngredients()	machine.service.ts:117	Ergänze URL um Präfix oder Suffix	URL für Zutaten kann nicht gefunden werden, daher keine Zutaten anzeig- und nachfüllbar
Zutat nachfüllen	IngredientItemComponent.onRefillSubmit()	ingredient-item.component.ts:22	Addiere 1 oder -1 auf 'refillAmountValue'	Falscher Wert wird in DB übertragen. Rezepte die möglich sein sollten sind nicht möglich oder Rezepte die nicht möglich sein sollten sind es
	> MachineService.refillIngredient(Ingredient, number)	machine.service.ts:107	Ändere '+=' zu '-=' in Zeile 110	Werte in Datenbank steigen nur noch, daher negative Werte in Datenbank

Abbildung A.6: Matrix mit Komponenten des Coffeemakers, sowie möglichen Fehlern

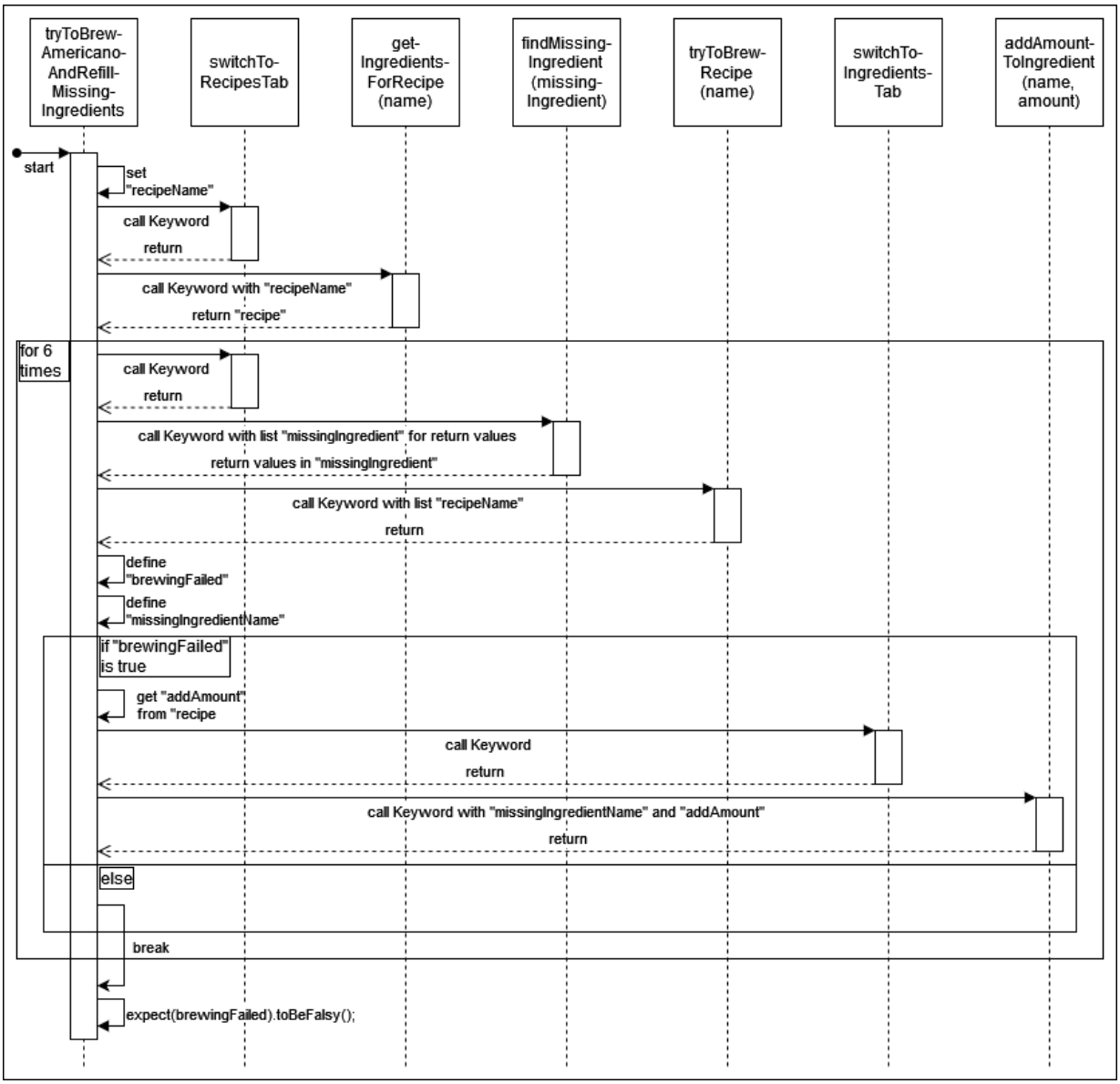

Abbildung A.7: Sequenzdiagramm für das Keyword "tryToBrewAmericanoAndRefillMissingIngredients"

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original