

Abschlussprüfung

Fachinformatikerin für Anwendungsentwicklung

Projektdokumentation zum Thema:

## **Umstellung von Cypress Testwerkzeug auf ein Open-Source Akzeptanztest-Framework**

Abgabedatum:

Team:

Projektbetreuerin:

Prüfungsbewerberin:

Ausbildungsbetrieb:

## Inhaltsverzeichnis

Abbildungsverzeichnis .....	<a href="#">IV</a>
Tabellenverzeichnis .....	<a href="#">IV</a>
Abkürzungsverzeichnis .....	<a href="#">V</a>
<b>1. Einleitung .....</b>	<b><a href="#">1</a></b>
1.1 Projektumfeld .....	<a href="#">1</a>
1.2 Projektbegründung und -ziele .....	<a href="#">1</a>
1.3 Projektschnittstellen .....	<a href="#">5</a>
1.4 Projektabgrenzung .....	<a href="#">6</a>
<b>2. Projektplanung .....</b>	<b><a href="#">6</a></b>
2.1 Agile Projektplanung .....	<a href="#">6</a>
2.2 Zeitplanung .....	<a href="#">7</a>
2.3 Personalplanung .....	<a href="#">9</a>
2.4 Sachmittelplanung .....	<a href="#">9</a>
<b>3. Analysephase .....</b>	<b><a href="#">9</a></b>
3.1 Ist-Analyse .....	<a href="#">9</a>
3.2 Anwendungsfälle .....	<a href="#">12</a>
3.3 Nutzwertanalyse .....	<a href="#">12</a>
3.4 Wirtschaftlichkeitsanalyse .....	<a href="#">13</a>
3.4.1 „Make or buy“ Entscheidung .....	<a href="#">13</a>
3.4.2 Projektkosten .....	<a href="#">14</a>
3.4.3 Amortisationsdauer.....	<a href="#">15</a>
<b>4. Entwurfsphase .....</b>	<b><a href="#">15</a></b>
4.1 Qualitätssicherung .....	<a href="#">15</a>
4.2 Use Case Diagramme .....	<a href="#">16</a>
<b>5. Implementierungsphase .....</b>	<b><a href="#">16</a></b>
5.1 Implementierung des Puppeteer Tests .....	<a href="#">16</a>
5.2 Implementierung des Playwright Tests .....	<a href="#">16</a>
<b>6. Testphase .....</b>	<b><a href="#">17</a></b>

7. Deploymentphase .....	<u>17</u>
7.1 Durchführung Deployment .....	<u>17</u>
8. Dokumentationsphase.....	<u>17</u>
9. Fazit .....	<u>18</u>
9.1 Soll/Ist-Vergleich .....	<u>18</u>
9.2 Lessons learned .....	<u>18</u>
9.3 Ausblick .....	<u>18</u>
 Anhangsverzeichnis .....	<u>VII</u>
Eidesstattliche Erklärung .....	<u>VII</u>
Quellenverzeichnis .....	<u>VIII</u>

## Abbildungsverzeichnis

Abbildung 1: Cypress Notiz-Screenshot aus dem Ticket „RRC-9265“ .....	<u>2</u>
Abbildung 2: Cypress Notiz-Screenshot aus dem Ticket „RRC-9265“ .....	<u>2</u>
Abbildung 3: Cypress Notiz-Screenshot aus dem Ticket „RRC-9265“ .....	<u>3</u>
Abbildung 4: Manueller Puppeteer Test .....	<u>16</u>
Abbildung 5: Die maui Frontend Login-Seite und der VoucherJS dahinter .....	<u>XI</u>
Abbildung 6: Beispiel eines VoucherJS Tests in Playwright .....	<u>XI</u>
Abbildung 7: VoucherJS Test Szenarien .....	<u>XI</u>
Abbildung 8: Screenshot der JSDoc .....	<u>XIV</u>
Abbildung 9: Screenshot der Dokumentation in Kiwipedia.....	<u>XIV</u>

## Tabellenverzeichnis

Tabelle 1: Angaben zur pauschalen Zeitplanung des Projekts .....	<u>8</u>
Tabelle 2: Cypress Tests pro Woche .....	<u>13</u>
Tabelle 3: Personalkosten .....	<u>15</u>
Tabelle 4: Vergleich zwischen Test-Framework.....	<u>X</u>
Tabelle 5: Angaben zur pauschalen Zeitplanung des Projekts .....	<u>XII</u>
Tabelle 6: Zeitplanung Soll-/Ist-Vergleich .....	<u>XIII</u>

## Abkürzungsverzeichnis

API	-----	Application Programming Interface
BDD	-----	Behaviour-Driven Development
CI / CD	-----	Continuous Integration / Continuous Delivery*
CLI	-----	Command Line Interface
DLS	-----	Digital Lifestyle
ECM	-----	Enterprise Content Management
E2E Tests	-----	End-to-End Tests
Freenet	-----	freenet DLS GmbH
HTML	-----	Hypertext Markup Language
IDE	-----	Integrated Development Environment
JS	-----	JavaScript
JSDocs	-----	JavaScript Documentation Generator
MVP	-----	Minimum Viable Product
OSS	-----	Open-Source Software
PDF	-----	Portable Document Format
SPA	-----	Single-Page Application
SSR	-----	Server-Side Rendering
TDD	-----	Test-Driven Development
UML	-----	Unified Modeling Language
<u>YAML</u>	-----	YAML Ain't Markup Language™

## Anhangsverzeichnis

1	UML Diagramm Use Case Puppeteer .....	<u>IX</u>
2	UML Diagramm Use Case Playwright .....	<u>IX</u>
3	Vergleich der Testframeworks Funktionalitäten .....	<u>X</u>
4	Anhang 4 .....	<u>XI</u>
5	Projektphasen ausführlich .....	<u>XII</u>
6	Soll-/Ist-Vergleich .....	<u>XIII</u>
7	Dokumentation .....	<u>XIV</u>

## Eidesstattliche Erklärung

Hiermit erkläre ich eidesstattlich, die vorliegende Dokumentation zur betrieblichen Projektarbeit mit dem Thema

*Umstellung von Cypress Testwerkzeug auf ein Open-Source Akzeptanztest-Framework*

im Rahmen meiner Prüfung selbständig verfasst habe.

Alle Stellen, die im Wortlaut oder dem Sinne nach anderen Texten entnommen sind, wurden unter Angaben hier verwendeter Quellen, und alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet.

Dies gilt auch für bildliche Darstellungen, Tabellen und dergleichen.

Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschungsversuch behandelt werden und zum Nichtbestehen der Prüfung führen können.

Diese Arbeit wird in dieser Form zum ersten Mal eingereicht und wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rendsburg,

Ort, Datum

Unterschrift

## Quellenverzeichnis

Freenet Group GitHub Repository des Voucher.js Cypress Tests. [online]:

<https://github.com/freenet-group/maui-cypressintegrationtest> [01.04.2024]

Cypress Dokumentation. [online]: <https://docs.cypress.io> [01.04.2024]

Cypress Dokumentation – Preise. [online]: <https://www.cypress.io/pricing#monthly> [01.04.2024]

Manikanta Rajkumar Seka. 9 Best Playwright Alternatives and Competitors. [online]:

<https://www.softwaretestingmaterial.com/playwright-alternatives/> [03.04.2024]

Puppeteer Dokumentation. [online]: <https://pptr.dev/> [04.04.2024]

GitHub Actions Dokumentation. [online]: <https://docs.github.com/en/actions> [09.04.2024]

Seb Rose. Introduction to TDD and BDD.(15 Mai 2017)[online]: <https://cucumber.io/blog/bdd/intro-to-bdd-and-tdd/> [10.04.2024]

Unsere Agile Methoden. [firmeninternes Intranet]:

<https://freenetgroup.sharepoint.com/sites/AgileTransformation/SitePages/Unsere%20Agilit%C3%A4t.aspx> [12.04.2024]

Open-Source Software. [online]: [https://en.wikipedia.org/wiki/Open-source\\_software](https://en.wikipedia.org/wiki/Open-source_software) [12.04.2024]

Hoffmann, Nagel, Zhou (16. Januar 2024). The Value of Open-Source Software. Zusammenfassung.

[online]: [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=4693148](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4693148) [12.04.2024]



# 1. Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojekts, welches die Autorin [REDACTED] im Rahmen ihrer Ausbildung zur Fachinformatikerin für Anwendungsentwicklung durchgeführt hat.

## 1.1 Projektumfeld

Das Projekt wurde angefertigt im Ausbildungsbetrieb der *freenet DSL GmbH*, am Standort Büdelsdorf, dem Hauptsitz des Unternehmens mit ca. 650 Mitarbeitern.

Das Unternehmen ist ein Tochterunternehmen der *freenet AG* und der größte netzunabhängige Anbieter mobiler Kommunikation in Deutschland, mit weiteren „Digital Lifestyle“ Geschäftsfeldern: Internet und TV-Entertainment, sowie Energie.

Die Bereiche des Unternehmens sind wie folgt gegliedert:

- Software Development
- Portfolio & Agile
- Billing & SAP
- Infrastructure

Die Ausbildung bei der *freenet DSL GmbH* hat die Autorin im Fachbereich „Software Development“ in der Abteilung *eCommerce & API* begonnen, danach in der Abteilung *Retail & Enterprise Content Management (ECM)* weiterverfolgt, innerhalb der auch diese Projektarbeit gefertigt wurde.

## 1.2 Projektbegründung und -ziele

*freenet DSL GmbH* nutzt seit 2019 in allen Bereichen die Grundlagen der Agilen Arbeitsweise.

In der Praxis nützen wir dafür die Continuous Integration/ Continuous Delivery (CI/CD)\* : kontinuierliche Integration/ kontinuierliche, iterative Entwicklung, kontinuierliches Testen, kontinuierliche Integration der Produkte in kurzen Zyklen. Automatisierte Tests überprüfen die Softwarefunktionalität, so wird die Produktivität erhöht. Der Einsatz der Agilen Methode über Scrum-Framework bei *freenet DSL GmbH*, sowie der einzelnen Testwerkzeuge, wird im [Kapitel 2.1 Agile Projektplanung](#) näher erläutert.

Im Rahmen der CI/CD und der automatisierten Tests, wird das auf JavaScript basierte Testwerkzeug Cypress genutzt.

Cypress hat ab der Version v13 begonnen, die Verwendung von Plug-ins von Drittanbietern zu blockieren.

Ein Plug-in (wahlweise: Add-on, Add-in) ist eine Software-Erweiterung, die eine spezifische zusätzliche Funktionalität einem existierenden Computerprogramm hinzufügt.

Das Programm selbst bereitet Dienste, die von Plug-ins genutzt werden, denn ein Plug-in kann nicht selbständig funktionieren, es dient nur der Anpassung des „Host-Programms“, in unserem Fall – des Cypress.

Ab der Version v13, hat das Framework Änderungen vorgenommen, die eine neue Konfiguration notwendig gemacht haben:

```
-----
From the Cypress Team:

Beginning June 30, 2023, recording Cypress tests to self-hosted solutions now requires the Cypress Gateway Connector.

To continue recording, please visit the link below:

https://on.cypress.io/gateway-connector?id=cmZ6biV0c2t0dXR3eWZxJVdqbdHdqeHhudHMLTE5Z

-----

Errors:

[
  "Action Required: Unsupported recording service"
]

Request Sent:

{
  "projectId": "maui Infoportal Regression GIT",
  "url": "****"
}
```

**Abb 1.:** Cypress Notiz-Screenshots aus dem Ticket „RRC-9265“ vom 21.September – 21. Dezember 2023: eine neue Konfiguration ist notwendig, um das Werkzeug weiterhin nutzen zu können.

```
72 > DEBUG=currents:specs cypress-cloud run --record --parallel --ci-build-id "SBUILD_ID" --spec ./cypress/integration/Auftragsbeaufkunding_Sanity/*.feature
73
74 Using config file: file:///w/maui-cypressintegrationtest/maui-cypressintegrationtest/currents.config.js
75
76 Copyright (C) 2023 Currents Software Inc https://currents.dev
77 This is free software, and you are welcome to redistribute it under certain
78 conditions. This program comes with no warranty. Parts of this program are MIT
79 licensed. Refer to the license for details
80 https://github.com/currents-dev/cypress-cloud/blob/main/LICENSE.md
81
82 Cypress stdout:
83
84 Cypress stderr:
85
86 DevTools listening on ws://127.0.0.1:39313/devtools/browser/95906875-15d7-4675-ae1d-cdb5a96530b1
87 [581:1120/134231.774192:ERROR:gpu_memory_buffer_support_x11.cc(44)] dr13 extension not supported.
88
89 We've detected that you're using a 3rd party library that is not supported by Cypress: cypress-cloud
90
91 To continue running Cypress, please remove this library and its configuration files or contact us for help migrating.
92
93 https://on.cypress.io/unsupported-third-party-library?id=w6f0m80vw6MK3D8q=w530s80qw6zDn80tw63Cp80dw6bDnc0vw57CusKswgJCqskowgCp80cw5%2EDrs0hwgJCq%3D%3D
94
```

**Abb 2.:** Cypress Notiz-Screenshots aus dem Ticket „RRC-9265“ vom 21.September – 21. Dezember 2023: Die Aufnahmen der Tests sind nicht mehr möglich, Aufgrund der Blockade der Drittanbieter Plug-ins.

Laut Meldung von [REDACTED]

„Ab Version 13 blockiert Cypress aktiv die Plugins:

- @deploysentinel/cypress-parallel
- cypress-debugger
- @currents/cypress-debugger-support
- @currents/cypress-debugger-plugin
- cypress-cloud
- cypress-debug
- cypress-vscode “

Auch das in Cypress integrierte Kommandozeilen-Modul / Command Line Interface (CLI)\* cy2, das den Entwicklern die Arbeit erleichterte, wurde abgekündigt.

Dies machte die täglichen Tests schwierig bis unmöglich, jedoch die von Cypress ausgesprochene Empfehlung lautete, auf Cypress-Cloud als Werkzeug umzustellen.

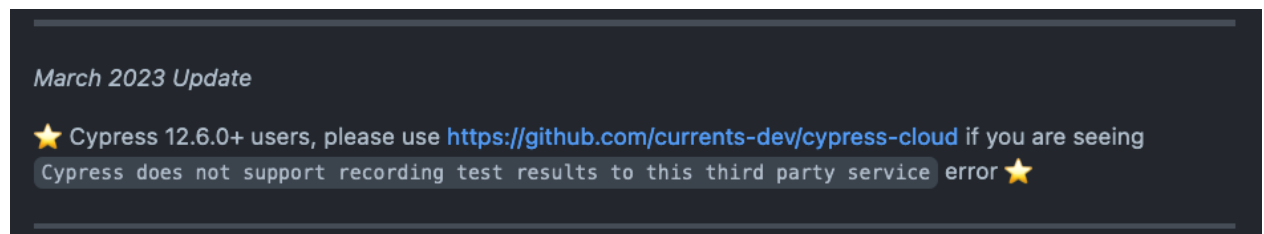


Abb 3.: Cypress Notiz-Screenshots aus dem Ticket „RRC-9265“ vom 21. September – 21. Dezember 2023: Die Aufnahmen der Tests sind nicht mehr möglich, Aufgrund der Blockade der Drittanbieter.

Das hätte nicht nur die monatlichen Test-Kosten um ca. 300 Euro pro Abteilung erhöht, sondern auch die Nutzung der Drittanbieter-Services erschwert, sodass die Tests und auch GitHub-Workflows in jedem Team umgeschrieben werden müssten. Die Workflows und ihre ausschlaggebende Rolle bei der Testautomatisierung werden im Kapitel ausführlicher betrachtet.

Es wurde im Team Retail Core eine „Hilfskonstruktion“ als vorübergehende Lösung hierfür entwickelt, die immer noch im Einsatz ist: über einen Mirror-Site, bei dem die Version v12 installiert wurde und von dort in den Test-Prozess eingebunden.

Aus diesem Grund möchten wir analysieren, wie Tests in anderen, Open-Source Testwerkzeugen funktionieren, um diese, anstatt von Cypress, einsetzen zu können

Eine der Grundlagen der Softwareentwicklung im Rahmen der agilen Arbeitsweise ist das Testen: Unit-Tests, End-to-End-Tests (E2E), Integration-Tests und Test-Automation. In diesem Falle im Bereich „Software Development“ / Abteilung „Retail & ECM“ wird das E2E-Testing-Werkzeug Cypress genutzt.

Diese Komponenten und Test-Arten werden umfassender im Abschnitt [Agilen Projektplanung\(S. 7\)](#).

Aus zeitlichen Gründen haben wir uns entschieden, zwei Open Source Testing-Werkzeuge auszuprobieren, um dann uns auf eines davon festzulegen. Es wurden Puppeteer und Playwright ausgesucht. Begründung:

Puppeteer ist eine NodeJS Bibliothek, die als Projekt von Google unterstützt wurde. Anfangs wurde sie von Andrey Lushnikov entwickelt, der mittlerweile an Playwright arbeitet. Puppeteer hat eine flexible, intuitive Application Programming Interface (API), die es einfach macht, Skripte für die Automatisierung von Browser-Interaktionen zu schreiben. Es nutzt das Sprachen- und Plattform-neutrale Standard des WebDriver-Protokolls der Browser-Automatisierung, um sich mit dem Browser zu verbinden und Interaktionen mit HTML zu simulieren.

Puppeteer kann dann Screenshots und PDFs der Webseiten erstellen, die Single Page Applications (SPAs) durchgehen und vorgerenderten Content generieren, auch diese Apps, die Server Side Rendering (SSR) Methode nutzen, automatisiert Formulare füllen, User Interface testen, Tastatur Eingaben testen usw., und all das im Browser. Das macht ihn auch sehr schnell.

Es kann auch einen zeitlichen Ablauf der Interaktionen der Webseite erstellen, so ist eine Diagnose der Performance möglich.

Puppeteer als Bibliothek würde ein schnell integrierbares Werkzeug abgeben, das keine Lernkurve für die Teams darstellen würde, es basiert auf JavaScript, arbeitet gut mit TypeScript und Cucumber zusammen und ist gut geeignet für Cross-Browser Testautomatisierung.

Playwright ist ein E2E-Framework für moderne Anwendungen, das sehr flexibel in jedem Browser und auf jeder Plattform genutzt werden kann. Es wird von Microsoft unterstützt.

Es nutzt nur eine API und die Tests sind isoliert – das bedeutet, was immer in einem Test getestet wird, bleibt es auch in diesem Test und beeinflusst nicht weitere Tests.

Tests können parallel, simultan in verschiedenen Browsern laufen, mit verschiedenen Cookies, Caches und anderen Speicher konfiguriert sein. Das macht Playwright auch sehr schnell und vielseitig. Es hat sehr leistungsfähige Tools und ist für viele Programmiersprachen anwendbar: Python, TypeScript, JavaScript, .NET, Java.

Es kann auch als VSCode-Erweiterung (Plug-in) genutzt werden. VSCode ist eines der populärsten Entwicklungsumgebungsprogramme, das auch in Teams der freenet gerne genutzt wird.

Alle diese Komponenten und Begriffe werden umfassender im Abschnitt der [Agilen Projektplanung](#) beschrieben.

Ziel des Projekts ist die Analyse und Umstellung unserer aktuellen End-to-End-Testumgebung von Cypress auf eine dieser zwei Open-Source Alternativen. Es soll eine flexible und vielseitige Testumgebung geschaffen werden, wobei untersucht wird, welcher der Frameworks in unsere bestehenden Workflow-Aktionen am besten integriert werden kann, welche Unterschiede es zu Cypress gibt, und ein besonderes Augenmerk auf die bestehende Integration von *Cucumber* (Adapter-Framework für Umsetzung der Test Szenarien- Schritte) gelegt wird, das schon in Test-Workflows genutzt wird, denn diese Behaviour-Driven-Development (BDD)-Ansätze sollen für alle Team Applikationen beibehalten werden.

BDD-Ansatz: bei dem Ansatz ist es wichtig die kollaborativ zu arbeiten und die Nutzer/Unternehmen-Anforderung und Perspektive bei der Entwicklung von Programmen einzunehmen, bevor die tatsächliche Entwicklungsarbeit begonnen hat.

Aus einer festgelegten Anforderung / Spezifikation ergeben sich Szenarien, die in *.feature* Dateien geschrieben werden und als Grundlage für einzelnen Schritte eines automatisierten Tests dienen.

Hierfür nutzen wir bei unserem Voucher.js das Werkzeug Cucumber innerhalb des Cypress Frameworks für Szenarien der Spezifikation, und Gherkin als die strukturierte Klartext-Sprache mit der Syntax, die für diese Szenarien der „Grammatik“ festlegt. So können auch Nichtprogrammierer Szenarien schreiben, die dann als *.features* für Testautomatisierung gespeichert und genutzt werden.

BDD-Szenarien bestehen in der Regel aus drei Abschnitten (Gherkin-Syntax):

1. Gegeben (Given) – Beschreibung des Zustands, bevor ein Verhalten ausgelöst ist;
2. Wenn (When) – Beschreibung der Aktionen, die ein Verhalten auslösen;
3. Dann (Then) – Beschreibung des erweiterten Ergebnisses des Verhaltens.
4. Und (And), But (Aber) – Diese Schlagwörter fügen zusätzliche Schritte hinzu und werden zur besseren Lesbarkeit eingesetzt.

Beispiel ein Szenario in der *Voucher.feature* Datei [Zitat]<sup>1</sup>:

„Szenario: Erfassung eines Vouchers mit gültiger Coupon ID

Angenommen die Voucher Erfassung wurde aufgerufen

Und die Felder "Coupon-ID" mit Daten befüllt wurden

Und der aktive Button "Coupon Aktivieren" betätigt wird

Dann ist die Voucher Aktivierungsbestätigungsseite ersichtlich“

### 1.3 Projektschnittstellen

Die Entwickler der Abteilung Retail & ECM betreuen zahlreiche Anwendungen, und sind dementsprechend in Projektteams aufgeteilt.

Das Entwicklerteam Auftragserfassung betreut unter anderem auch die Voucher Applikation, die innerhalb dieses Projekts getestet wird - eine Frontend-Anwendung, über die das Unternehmen das Marketing, Verkauf im Handel und digitale Dienstleistung in einer Anwendung zusammenbringt:

Ein physischer Coupon/Voucher versehen mit einem QR-Code wird im Handel vermarktet, ein Verkäufer scannt den Gutschein-Code ein, der dadurch automatisch in das Frontend-Formular eingefügt wird und nach der Eingabe und Verifizierung des Rabatt-Code, kann der Kunde zu einem vergünstigten Preis ein DSL-Vertrag abschließen, während im Hintergrund/ Backend für den Promotor des Vouchers die Provision gebucht wird.

Die Voucher Anwendung ist die neueste in der Reihe und E2E-Tests sind unerlässlich.

---

<sup>1</sup> Ausschnitt aus der *Voucher.feature* Datei des Cypress Tests der freenet-group Repository auf GitHub

## 1.4 Projektabgrenzung

Im Rahmen dieses Projekts, wird die Voucher-Anwendung mit Puppeteer und Playwright getestet. Aufgrund der begrenzten Zeit von 80 Stunden, wurden bestimmte Einschränkungen und Priorisierungen vorgenommen:

Es ist nicht vorgesehen weitere E2E Test-Frameworks außer Puppeteer und Playwright zu betrachten.

Auch wird nur das Frontend der Voucher getestet, E2E Tests für das gesamte Team sind zum späteren Zeitpunkt nach dem Projekt geplant.

## 2. Projektplanung

### 2.1 Agile Projektplanung

Freenet DLS GmbH nutzt seit 2019 Agile Methoden und hat dafür vier Leitlinien entwickelt, nach denen gearbeitet wird:

- Sprint-Synchronisation
- Minimum Viable Product
- Boards
- Rollen & Verantwortlichkeiten

Alle Teams der Software-Entwicklung arbeiten in Sprints, was den Umgang mit teamübergreifenden und auch bereichsübergreifenden Abhängigkeiten und Ressourcen erleichtert. So können sie schnell zugewiesen werden, wenn sie im Verlauf eines Projekts gebraucht werden. Dies wird dadurch erleichtert, dass alle Teams am gleichen Tag der gleichen Woche ihre Sprints starten.

Boards werden zur Information und Transparenz genutzt, sodass alle in Teams wissen, in welcher Phase sich Projekte und einzelne Aufgaben befinden, sowie auch wer die Ansprechpartner für verschiedene Verantwortungsbereiche sind.

Für dieses Abschlussprojekt ist die wichtigste Leitlinie die des Minimum Viable Product – MVP.

Sie hilft uns [Zitat]<sup>2</sup> „... möglichst frühzeitig den potenziellen Kundennutzen (zu) erzeugen“. Ein Produkt wird durch IT-Systeme mit kleinstmöglichem Entwicklungsaufwand umgesetzt für Endkunden, firmeninterne Kunden oder IT-Kunden.

Das bedeutet auch dass ein Produkt durch die gesamte Prozesskette (Verkauf, Buchung, Anpassung, Stornierung, Beendigung) bis in Backend-Komponenten (Netzpartner, Provisionssystem, Bilanzierung) betrachtet wird, und immer wieder getestet wird.

Manuelle Aufwände sind anfangs in Kauf genommen, durch die iterative Entwicklung und Tests werden sie eliminiert, wo es möglich ist. Die einzelnen Phasen werden immer unter Anforderern und IT-Kollegen besprochen, so lernen alle gemeinsam aus den Erkenntnissen, passen das Produkt an, bis zum Deployment und Ausbau.

---

<sup>2</sup> Laut firmeninternen Intranet Seite „Rund um die freenet“. Jan-Hendrik Wolf (2023). „Agile Transformation – Unsere agilen Grundlagen“ [intranet]: <https://freenetgroup.sharepoint.com/sites/AgileTransformation/SitePages/Unsere%20Agilit%C3%A4t.aspx>

Das Testen ein wesentlicher Bestandteil der Produktion und wird in allen Formen angewandt: es werden Unit-Tests, E2E-Tests, Integrationstests durchgeführt.

Unit-Tests testen einzelne Software- oder Programm-Komponenten auf ihre Korrektheit - dabei wird jede kleinste Einheit (Unit) des Codes oder einer Code-Komponente geprüft und getestet, damit Fehler so früh wie möglich entdeckt und korrigiert werden. Das sichert die Code-Qualität und Wiederverwendbarkeit von Komponenten, was langfristig Zeit und weitere Ressourcen spart. Bei freenet werden hierfür zum Beispiel Frameworks Mockito, Mocha, Jasmine und junit5 in Kombination mit SonarQube eingesetzt.

Hiermit werden auch Integrationstests durchgeführt, um zu testen, wie einzelne Komponenten oder Module miteinander funktionieren oder ob die Interaktionen zwischen Softwarekomponenten und Datenbanken fehlerfrei verlaufen. Das ist auch essenziell und zeitsparend, vor allem dann, wenn mehrere Entwickler / Entwicklerteams an verschiedenen Teilen der Software arbeiten.

Schließlich, nachdem Unit Tests und Integrationstest erfolgreich bestanden worden sind, werden E2E Tests durchgeführt. Sie sichern eine ganzheitliche Verifizierung von Code und Software innerhalb der Geschäfts- und Prozessabläufe aus der Benutzersicht. Der erwartete Anwenderfluss wird z.B. vom Frontend zum Backend und zurück geprüft. Ein echtes Anwendungsszenario wird simuliert, damit Fehler, die bei dem Ablauf der Anwendung durch das komplexe Softwaresystem und deren Abhängigkeiten, offenbar werden, rechtzeitig von dem Deployment behoben werden können.

Hierfür wird das Cypress Framework benutzt, das von Brian Mann im Jahr 2017 veröffentlicht wurde. Es ist ein sehr leistungsfähiges Werkzeug, dass die Applikationen und Software im Browser testet, und nicht nur auf JavaScript basierende Testing-Bibliothek und APIs bietet, die unzählige Codezeilen anderer Testwerkzeuge ersetzt, sondern auch eine ganze Plattform und Dashboard, um alle Testvorgänge an einer Stelle zu beaufsichtigen. Es ist sehr konfigurierbar, hat viele Vorteile und eine umfangreiche Dokumentation. Es kann zum Unit-Testing und Integrationstests genutzt werden.

Im Team Auftragserfassung der Abteilung Retail & ECM wurde es und wird es sehr gut an die Continuous Integration (CI) Produktionslinie mit GitHub Workflows angepasst: Tests können automatisch gestartet werden, sobald ein Code auf GitHub geschickt wurde - über vom Team vorkonfigurierten GitHub Workflow und vom Team angepasste Cypress GitHub Action wird ein Cypress Test in der GitHub Repository gestartet. Die visuelle Rückmeldung über die Testergebnisse wird im Browser der Wahl angezeigt (zurzeit Chrome, Firefox und Edge), und kann auch auf einer Zeitachse als Video angesehen werden.

Somit möchte die Autorin darstellen, dass eine gut funktionierende Testing-Umgebung und Praxis für die agilen Prozesse des Unternehmens unverzichtbar sind. Nähere Erläuterungen zu der Verwendung einzelnen Werkzeuge, Frameworks und Technologien, z.B. GitHub Workflows, wird im Kapitel [3.1 Ist-Analyse](#) beschrieben.

## 2.2 Zeitplanung

Der Durchführungszeitraum für das Projekt ist durch die IHK-Vorgabe auf achtzig Stunden festgelegt. Dadurch, dass ich die Projektarbeit einzeln bearbeite, habe ich mich für die agile Methode des

Kanban entschieden, um effizient, methodisch und fokussiert voranzukommen. Kanban funktioniert über Visualisierung des Projektfortschritts in einer Tabelle / einem Board, was meiner visuellen Präferenz entspricht und die festgelegte Aufteilung der Aufgaben eignet sich gut für die Kanban Fortschritt-Verfolgung. Zudem ist der Scrum Sprint Rhythmus, bei freenet in 2-wöchigen Zyklen, nicht ganz geeignet für die 80 zugewiesenen Arbeitsstunden.

Eine pauschale Zeitplanung folgt in der Tabelle 1 (fortgesetzt auf folgender Seite):

Tab.1: Angaben zur pauschalen Zeitplanung des Projekts

Projektphase	Benötigte Zeit
<b>Analyse</b>	<b>9h</b>
Analyse der aktuellen <i>Cypress</i> Tests für <i>Voucher.js</i>	3h
Vergleich der Funktionen von OSS Testing-Frameworks	4h
Untersuchung von Integrationsmöglichkeit von <i>Cucumber</i>	2h
<b>Konzeptentwicklung</b>	<b>12h</b>
Entwurf des Testframework-Konzepts	3h
Planung der <i>Cucumber</i> / <i>Gherkin</i> Integration	4h
Testfälle für jedes der festgelegten Frameworks	5h
<b>Implementierung</b>	<b>37h</b>
Entwicklung von Testskripte in Open-Source Testframeworks	25h
Einrichtung der Testinfrastruktur und Integration in GitHub Workflow-Aktionen	12h
<b>Testphase</b>	<b>10h</b>
Durchführung von End-To-End-Tests	6h
Vergleich mit <i>Cypress</i> Testwerkzeug	4h
<b>Dokumentation</b>	<b>12h</b>
Erstellung Projektdokumentation	6h
Anpassung Entwickler- und Anwenderdokumentation	6h
<b>Gesamt</b>	<b>80h</b>



## 2.3 Personalplanung

Die Autorin arbeitet als Einzelperson an diesem Projekt und die [REDACTED] und [REDACTED] stehen für Nachfragen zur Verfügung. Regelmäßige Treffen und Revision mit Kanban-Board sind auch Teil der Projektentwicklung, um die Anforderungen immer wieder zu überprüfen und die Planung entsprechend anzupassen.

## 2.4 Sachmittelplanung

Im Rahmen der Ausbildung wurde mir ein Entwickler-Notebook DELL Latitude 5531 mit dem Betriebssystem Windows 10 Pro zur Verfügung gestellt, sowie der Zugang zu allen für agile Methoden notwendigen Technologien und Softwareprodukten:

- Jira, Confluence, SharePoint;
- MS Office365;
- GitHub, Cypress Version v12, NodeJS Version v18;
- Integrierte Entwicklungsumgebung IntelliJ IDEA 2022.3.3 (Ultimate Edition).

# 3. Analysephase

## 3.1 Ist-Analyse

Für das Voucher-Frontend im Infoportal des maui-Werkzeugs für den freenet-Fachhandel wird zum integrativen Testen das Cypress Testing-Werkzeug für Frontends genutzt, implementiert durch GitHub Actions Workflow auf dem maui Cypress-Repository. Die Tests erfolgen automatisch, aus der Konsole, täglich, können aber auch manuell gestartet werden.

Cypress als Testing Framework stellt eigene vorgefertigte, konfigurierbare GitHub-Action Workflows zur Verfügung, die jeder für eigene Tests anpassen kann.

GitHub ist eine offene Plattform für Versionsverwaltung, die aber auch gleichzeitig wie ein soziales Netzwerk funktioniert. GitHub existiert seit 2007, und wurde 2018 von Microsoft übernommen. Es basiert auf Linus Torvalds Git Versionsverwaltungssoftware: jeder kann einen Account eröffnen und dort seinen Code und Versionen von Code verwalten in einer Quelltext-Datenbank, die Repository genannt wird. Anlegen von neuem Code erfolgt in Branches (Zweigen), die zu einem großen Projekt zusammengeführt werden (Merge) und können auch abgespalten werden in sogenannten Forks, was Zusammenarbeit auch mit anderen Repositories vereinfacht. Änderungen werden den Branches oder den Forks hinzugefügt und dann wird eine Anfrage an den Autor gestellt, die Änderungen zu übernehmen (Pull Request).

GitHub-Actions ist eine in GitHub eingebaute Plattform für Continuous Integration (CI): sie ermöglicht den Entwicklern die Automatisierung der Softwareentwicklungsphasen und Prozesse – entweder komplett oder Teile davon - von der Idee / Aufbau (Build-Pipeline [Englisch]), über das Testen bis zur Integration in die Produktion. GitHub Actions nutzt Workflows (Ablaufplan [Englisch]), die wiederum aus einzelnen Jobs (Aufgaben) bestehen und jeder Job hat einzelne Steps (Schritte). Workflows können von Entwicklern / Entwicklerteams an eigene Bedürfnisse angepasst werden, sie werden in einer .yaml\*-Datei in einem .github/workflows Ordner im Repository gespeichert. Jede Repository kann mehrere Workflows haben und jeder davon kann eigenen Satz Aufgaben erledigen: für jeden

Prozess der Entwicklung - z.B. Workflow für Tests und Pull Requests, oder einen für Bereitstellung der Applikation, jedes Mal, wenn eine neue Version in Produktion gestellt wird. Workflows können mehrfach genutzt werden, d.h., ein Workflow kann von einem anderen referenziert werden. So werden zusätzlich Zeit und Ressourcen gespart.

GitHub-Actions sind eigene kleine Anwendungen, die genutzt werden, um komplexe, aber sich häufig wiederholende Aufgaben zu automatisieren. Entwickler können sie selbst schreiben oder auf GitHub bereitgestellte finden, darunter auch Cypress GitHub-Actions, und für das eigene Projekt anpassen.

Cypress GitHub-Action wird für E2E- und Komponenten-Tests benutzt, und ist gänzlich konfigurierbar. Sie wird über die Zeile `use: cypress-io/github-action@v6` in das Repository Workflow eingebunden und installiert dann die notwendigen Dependencies/Abhängigkeiten. Danach laufen die E2E-Tests im Browser und die Ausgabe von Test-Ergebnissen. Die Entwickler der Abteilung haben hierfür die Nutzung von Chrome Browser konfiguriert. Auch alle weiteren Abhängigkeiten und Befehle wurden vom Team konfiguriert.

Hier wird der Vorteil der Nutzung von Open Source Software sehr deutlich:

Open-Source Software (OSS) bietet die Flexibilität und Anpassungsmöglichkeiten, mit einem zunächst höheren Aufwand, durch das Lernen, Anpassen und Supportbedarf, dies ist jedoch auch der Fall bei kostenintensiveren kommerziellen Frameworks. Somit ist auch die Wirtschaftlichkeit ein Faktor, denn die Anpassung erfolgt meistens nur am Anfang, und die Tests und Workflows werden festgelegt, dann laufen sie weiter automatisch, mit einer schnellen Amortisation.

Open-Source Software wird unter der Lizenz herausgegeben, bei der die Urheberrechtsinhaber den Nutzern das Recht einräumen, die Software und ihren Quellcode zu nutzen, studieren, zu ändern und verbreiten. Sie wird oft in offener Zusammenarbeit erstellt – was Nutzern die Möglichkeit der Mitwirkung bietet. Sie kann an persönliche Bedürfnisse angepasst werden. Änderungen können von verschiedenen Nutzern als Forks veröffentlicht werden und Verbesserungen als Pull Request eingereicht werden.

Es wird geschätzt, dass Open Source Software einen Wert für Unternehmen von ca. \$8.2 Billionen darstellt, sonst müssten sie ohne OSS etwa 3,5-mal höheren Betrag aufwenden für die Software, die sie nutzen, wäre sie kommerziell zu erwerben<sup>3</sup>.

Die Überlegung, OSS zu nutzen hat sich aus wirtschaftlichen Gründen und guten Erfahrungen ergeben, denn die Transparenz, gute Dokumentation und das hohe Maß an Anpassungsmöglichkeit an die Softwareentwicklungsprozesse der Abteilungen, machen OSS wie z.B. GitHub mittlerweile zu Standards, vor allem auch in Hinblick auf dynamische Auslastung der IT-Abteilungen.

So hat auch die Umstellung des Cypress Frameworks auf erzwungene Nutzung von (bezahlten) Cloud Lösungen dazu geführt, uns die OSS-Werkzeuge wie die Puppeteer Bibliothek und das Playwright Framework näher anzuschauen und auszutesten.

Unter Bibliotheken und Frameworks im Kontext der Softwareentwicklung verstehen wir wiederverwendbare Code-Module, die mit Standard entsprechenden, vorgefertigten

<sup>3</sup> Laut Wikipedia Notiz[online] [https://en.wikipedia.org/wiki/Open-source\\_software#cite\\_note-4](https://en.wikipedia.org/wiki/Open-source_software#cite_note-4) [16.04.2024] und der Kurzfassung der Studie von Hoffmann, Nagel, Zhou ( 16. Januar 2024). "The Value of Open Source Software." [online] [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=4693148](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4693148)

Funktionalitäten, Klassen und Funktionen den Entwicklern helfen, skalierbare, wartbare Applikationen schneller zu bauen.

Eine Bibliothek ist jedoch im Unterschied zu einem Framework auf eine spezifische Aufgabe orientiert – ausgestattet mit fertigen Code-Operationen, die Entwickler abrufen können, um spezifische Aufgaben in ihrem Programm zu erledigen: zeitsparend, durch vorgefertigte Klassen und Funktionen, die ein Entwickler nicht selbst schreiben muss. So ist eine Bibliothek sehr flexibel: Entwickler können sie in ihr Code einbauen, in der Art und Weise, wie sie es für ihre Applikation sinnvoll halten.

Das ist auch einer der Hauptunterscheidungen zwischen eine Bibliothek / Library und einem Framework:

Ein Framework (Rahmenwerk) ist ein ganzer Satz Werkzeuge und Bibliotheken, Templates/Vorlagen und Leitlinien für die Entwicklung der Software, was eine Struktur bietet, gerade bei größeren und komplexeren Projekten die Entwicklung zu beschleunigen.

Somit hat Puppeteer als Bibliothek keine Funktionalitäten, die sonst in einem kompletten Testing-Framework zu erwarten sind: z.B. die Assert oder Expect API - die Interfaces, die im Testcode genutzt werden, um die Testfälle zu simulieren, bei Fehlern eine Ausführung unterschiedlich zu behandeln und eine festgelegte Rückmeldung zu geben. Daraus ergibt sich die Notwendigkeit, Puppeteer mit einem sog. Test-Runner zu nutzen. Oft werden dafür Jest oder Mocha und Chai JavaScript-Bibliotheken genommen, und in unserem Fall ist das die Kombination mit Mocha / Chai.

Playwright hingegen ist ein ganzes Framework, mit einer API und isolierten Tests, das auf allen Plattformen und für viele Sprachen funktioniert, wie schon im Kapitel 1.2 Projektbegründung und -Ziele **erwähnt**. Die Installation ist sehr schnell, mit automatisierten, konfigurierbaren Optionen, z.B. die GitHub-Action Dateien gleich mit der Installation zu generieren, sodass eine CI-Integration und das Testen sehr schnell erfolgen können. Eine der weiteren Funktionalitäten ist das Debugging-Fenster, das Schritt-für-Schritt das Test Skript verfolgt, um die Fehlerbeseitigung zu erleichtern.

Für Tester, die sich in der Playwright eigenen Syntax von Locators (Keywords, die Elemente auf einer zu testenden Seite finden) nicht zurechtfinden, ist eine Syntax-Vervollständigung vorhanden, sowie ein Test-Skript Generierungswerkzeug Codegen vorhanden, sodass eine Test-Spezifikation durch Zusammenklicken der Begriffe erstellt werden kann: die Interaktionen des Testers mit einer Webseite werden automatisch in ein Testskript in eine .spec.js oder spec.ts Datei generiert.

Ein zusätzliches Modul in Playwright ist „UI Mode“: eine Benutzeroberfläche/Dashboard, die im Browser Fenster dem Tester erlaubt alle Tests anzusehen, zu filtern und wie in einer Videodatei schneller oder langsamer abzuspielen, mit der Hervorhebung im Code während das Video spielt, was dem Framework auch den Namen gegeben hatte: Playwright / [Deutsch]Dramatiker, Drehbuchautor. Denn in der Filmindustrie werden Filme und Drehbücher auch so getestet: die Bildleiste und Skript laufen parallel sichtbar auf dem Bildschirm.

Mit dieser Aufzählung wird deutlich, dass ein Framework ein leistungsfähiges Werkzeug ist, aber eher geeignet für große, komplexe Projekte und Teams.

### 3.2 Anwendungsfälle

Das Test-Werkzeug hat zwei Anwendungsfälle:

- Ausführung von Testszenarien durch die Beschreibung der festgelegten Testskripte, in diesem Fall in JavaScript, durch eine .spec.js Datei und
- Integration in die CI durch eine GitHub .feature Datei in der freenet Repository und eine Rückgabe der Ergebnisse.

### 3.3 Nutzwertanalyse

Wie beschrieben in 2.1 und 3.1, für die agile Arbeitsweise im IT-Fachbereich und vor allem auch der Abteilung Retail & ECM, ist das Testing ein unerlässlicher Teil der täglichen Prozesse.

Im Projekt wird beispielhaft für die Voucher-Frontend, für das Team Auftragserfassung, eine Analyse der OSS-Testwerkzeuge vorgenommen, um die bisherige Nutzung von Cypress als Framework, das seine Dienste in die Cloud verlagert hat und für unsere Entwickler nicht mehr ohne einen Umweg über eine Mirror-Site einsetzbar ist, auszuwerten.

Puppeteer hat sich als Bibliothek bewährt, und bietet ähnliche Konfigurationsmöglichkeiten im CI, es ist weiterhin Cucumber mit Gherkin einsetzbar, allerdings sind die für ein Testwerkzeug erwarteten Funktionalitäten, wie Assert und Expect nicht Teil von Puppeteer und müssen aus anderen Bibliotheken, für unser Team wären das Mocha.js und Chai.js, hinzugefügt werden. Das betrifft auch Test-Runner – das wesentliche Teil jedes Testing-Frameworks, dass die Tests in deren CI-Umgebung orchestriert.

Deshalb hat mich Playwright als Werkzeug besser überzeugt: es ist sehr leistungsfähig, schnell und schnell einsetzbar, ohne zusätzlichen Bibliotheken, die für Funktionalitäten notwendig sind. Es kann die Menge der Tests im Team Auftragserfassung, aber vor allem auch in allen anderen Entwicklerteams, bzw. im gesamten IT-Fachbereich eingesetzt werden, da es auf allen Plattformen (MacOS, Windows,

Linux) mit fast allen Sprachen (JavaScript, TypeScript, Python, Java, .NET) eingesetzt werden kann, unterstützt die Gherkin-Syntax der BDD-Szenarien und bietet bei der Installation eine schon generierte GitHub-Workflow Datei, die dann vom Entwicklerteam konfiguriert in der GitHub Repository eingesetzt werden kann.

Obwohl beide Werkzeuge schnell auf NodeJS installiert werden können, und die Lernkurve, bzw. die Umstellungszeit für beide Werkzeuge für die Entwicklerteams niedrig wäre, ist die Umstellung von Cypress auf Puppeteer umständlicher:

die Syntax erfordert keine großen Änderungen (JavaScript), aber aufgrund der Abhängigkeiten von mehreren Bibliotheken, birgt die Nutzung Instabilitäts-Risiken mit sich, falls eine der Bibliotheken inkompatibel ist mit bestehendem System eines Teams ist oder die Unterstützung eingestellt wird.

Playwright ist schnell, stabil, leistungsfähig, wird von Microsoft unterstützt und bietet eine einzigartige graphische Oberfläche (Dashboard) für die Tester, mit der Möglichkeit granular die verschiedenen Test-Typen und Dateien zu filtern, sortieren, beobachten und aufnehmen.

Die Gegenüberstellung der drei Testing-Werkzeuge wird im [Anhang 3](#) abgebildet.

### 3.4 Wirtschaftlichkeitsanalyse

Die Abteilung Retail Core & ECM besteht aus verschiedenen Entwicklerteams, die unterschiedliche Applikationen unter der Abteilungsführung betreuen, wie z.B. eCare, maui Inforportal, mauiBKB, maui Auftragserfassung, das vom Team Auftragserfassung betreut wird und wo sich auch die Voucher-Frontend befindet.

Um eine approximative Anzahl der Tests die z. B. in einer Woche durchgeführt werden, habe ich eine Kollegin im Team maui befragt und einen Kollegen im Team eCare befragt.

Für das Team maui gilt, dass täglich mindestens ein automatischer Test gestartet wird, und bei jedem Deployment in die GIT-Umgebung werden die Tests angestoßen (je nach Applikation). Ein manuelles Anstoßen der Tests ist auch jederzeit möglich. Ähnlich ist das mit Team eCare, wobei sie viel öfter Änderungen an Apps vornehmen und deshalb auch die Testanzahl viel höher ist.

Datum	maui Auftragserfassung (Auftragsbeauskunftung, ZBO, usw) mit Voucher Frontend	maui Inforportal	eCare
Dienstag, 26.03.2024	5	1	~20
Montag, 25.03.2024	4	1	~25
Sonntag, 24.03.2024	2	1	-
Samstag, 23.03.2024	2	1	-
Freitag, 22.03.2024	2	5	~28

Tab.2: Cypress Tests pro Woche (Angaben nach Auswertung/ Einschätzung der KollegInnen, Retail Core & ECM Team)

Cypress Preise für Cloud-Service ab der Version v13 würden für das gesamte Team Retail Core & ECM und seine verschiedenen Projektgruppen/ Teams, die Kosten von \$300 / Monat, (per Umrechnungskurs vom 16.04.2024) ca. 282.17 Euro bedeuten.

#### 3.4.1 „Make or Buy“ Entscheidung

Aufgrund der Kosten-Unterschiede zwischen Cypress Cloud Services und der Analyse der Funktionalitäten und anderer Faktoren, wie im [Anhang 3](#) dargelegt, ist die Tendenz stark, Playwright als OSS-Testingframework zu nutzen, Puppeteer scheidet dann aus, aufgrund der technischen Limits der Abhängigkeiten und Unwägbarkeiten, die damit verbunden sind.

Kosten: Cypress Cloud ist ein kommerzieller Service, für den bei weiterer Nutzung für das Team Retail & ECM monatliche Kosten von ca. ~ €282 entstehen würden, dies würde Support und andere Funktionalitäten beinhalten.

Funktionalität: Cypress bietet eine verwaltete Lösung mit Fokus auf Integration und Skalierbarkeit. Playwright hat eine umfangreiche Cross-Browser Unterstützung und leistungsfähige Automatisierungsmöglichkeiten, die gut geeignet sind für komplexe Szenarien und Multi-Plattform-Testing.

Integration: Nahtlos mit Cypress-basierten Tests, und anderen Werkzeugen. Gut für Workflow und Kollaboration. Playwright arbeitet sehr gut mit anderen OSS-Werkzeugen, somit ist die Entwicklung-Pipeline komplett konfigurierbar was unter Umständen mehr Zeit und Arbeit bei der Einrichtung bedeutet.

Lernkurve: beide Frameworks sind einfach einzurichten, mit ausgeprägt benutzerfreundlichen Oberflächen.

Support: Der Support-Service ist beim kommerziellen Produkt strukturierter und folgt den Standards, die Playwright Open-Source-Community ist auch sehr aktiv.

Skalierbarkeit: Mit der Cloud-basierten Infrastruktur und eingebundenem Support ist Cypress ideal für Projekte, die Skalierung eingeplant haben. Playwright ist auch fähig mit den Infrastrukturen mitzuwachsen, wobei zusätzliche Konfiguration und Instandsetzung notwendig sind.

Zukunftsfähigkeit: Ein kommerzielles Produkt wie Cypress hat regelmäßige Updates und Support. Preisänderungen oder Änderungen der Konfigurierbarkeit bringen Risiken mit sich, wie unsere Abteilung gerade erlebt hatte. Bei einem OSS-Produkt wie Playwright ist das auch möglich: Entwickleraktivität ist größerer Fluktuation unterworfen, aber die Entwickler-Community ist sehr aktiv und deren Unterstützung durch Microsoft suggeriert Stabilität.

**Buy** (Cypress Cloud): für eine skalierbare, verwaltete, kommerzielle Lösung mit Support sind die Kosten von ca. ~ €280/Monat hoch, aber erwartbar. Weitere Faktoren neben der Anzahl der Teams und Tests / Monat könnten aber auch dieses Budget sprengen.

**Make** (Playwright): Unsere Entwicklerteams haben die technische Expertise, um die Infrastruktur und Konfiguration von Playwright genauso wie bei Cypress einzurichten oder sogar schneller, und hätten damit ein flexibles Werkzeug für komplexe Testing Szenarien, und volle Kontrolle über die CI-Testing Umgebung.

**Schlussfolgerung:** Cypress Cloud bietet kommerziellen Support an; einer Umstellung von bisherigen Cucumber/Gherkin Plugins auf eine neue Version ist integriert in diesen Prozess. Die Nutzung der Cypress Cloud Services ist mit hohen Kosten für die IT-Abteilung verbunden.

Umstellung auf Playwright bedeutet statt direkter finanzieller Kosten, das Einplanen von Umstellungs- und Lernzeit.

### 3.4.2 Projektkosten

Für das Projekt entstehen keine direkten finanziellen Kosten, sie drücken sich in den laufenden Kosten für die vorhanden Hard- und Software und den entsprechenden Personalkosten zusammen. Aus datenschutzrechtlichen Gründen werden keine realen Angaben gemacht, sondern eine Berechnung mit pauschalen Werten:  
IT-Mitarbeiter mit Stundensatz von €48 / Stunde und €20 / Stunde für Auszubildenden.

In diese Kosten sind Sozialversicherungsanteile der Arbeitgeber berücksichtigt.

Person	Arbeitszeit	Stundesatz	Kosten
Entwicklerin (Autorin)	80 Stunden	€20,00 / Std.	€1.600,00
	10 Stunden	€ 48,00 / Std.	€480,00
Summe			€2.080,00

Tab.3: Personalkosten

### 3.4.3 Amortisationsdauer

Die laufenden Kosten für die Umstellung von Cypress auf Playwright würden sich auf schätzungsweise 3 Tagessätze pro IT-Mitarbeiter, verursacht durch die Lernzeit für das neue Test-Werkzeug, wie folgt darstellen:

$$\sim 32 \text{ Stunden} * €48,00 = €1.152,00$$

Bisherige Kosten für die Entwicklung einer vorübergehenden Lösung für Cypress v12 über die Mirror-Site betragen ca.

$$\sim 17 \text{ Stunden} * €48,00 = €816,00$$

Somit würde sich in anderthalb Wochen, konservativ gerechnet - unter Annahme der Verzögerungen durch andere Projekte oder Prioritäten - zwei Wochen, die Entwicklungszeit amortisieren.

## 4. Entwurfsphase

### 4.1 Qualitätssicherung

Wie schon erläutert, das Testen ist eine der Grundlagen der Qualitätssicherung bei der Softwareentwicklungsarbeit unserer IT-Abteilung: es werden kombiniert alle Arten von Tests durchgeführt: manuell, automatisiert, Unit- und Integrationstests, E2E-Tests. Sie sind notwendig, um die hohe Qualität und Funktionalität des Codes und der IT-Produkte und Services zu gewährleisten.

Somit dient die Projektarbeit der Autorin einer Qualitätssicherung des E2E-Testing-Prozesses innerhalb des Teams Auftragserfassung der Abteilung Retail & ECM.

Manuelle Tests werden von mir als Entwicklerin durchgeführt durch die Ausbilderin bei den Reviews und der Abnahme.

Die automatisierten Tests werden zu einem späteren Zeitpunkt im GitHub Repository integriert.

## 4.2 Use Case Diagramme

Use Case Diagramme der geplanten Testabläufe sind im [Anhang 1](#) und [Anhang 2](#) dargestellt.

# 5. Implementierungsphase

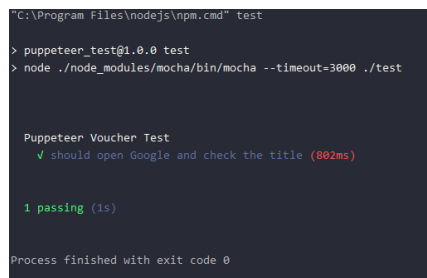
## 5.1 Implementierung der Puppeteer Tests

Puppeteer-Installation hatte zunächst mit der NodeJS Version Schwierigkeiten, sowie mit Voreinstellungen bei der integrierten Entwicklungsumgebung bereitet. Trotz verschiedener Ansätze wie z.B.

- dem Downgrading/der Herabstufung der NodeJS Version,
  - Herausnehmen der "script:"-Zeile aus der "package.json", gab es zumindest in unserem Fall, Konflikte der Dependencies, der NodeJS Version, node\_modules und unbekannter Parameter.
- Nach einigen Stunden Recherche, wurden die Konflikte gelöst.

Die Syntax ist relativ einfach zu beherrschen, dennoch besitzen Bibliotheken Chai und Mocha eigene Begriffe, die nachgeschlagen werden müssen bei den ersten Test-Anwendungen. Dies hat zu Folge, dass bei der ersten Implementierung mehr Zeit notwendig war, um alles anzupassen.

Die Probe-Tests waren erfolgreich, wie erwähnt, GitHub Workflows wurden nicht implementiert.



```
C:\Program Files\nodejs\npm.cmd test
> puppeteer_test@1.0.0 test
> node ./node_modules/mocha/bin/mocha --timeout=3000 ./test

Puppeteer Voucher Test
  ✓ should open Google and check the title (802ms)

1 passing (1s)

Process finished with exit code 0
```

**Abb. 4:** Die ersten Puppeteer-Tests nach der Installation waren erfolgreich, auch wenn manuell das --timeout Parameter gesetzt werden sollte, damit der Test-Code das Aufladen der Seite abwartet.

## 5.2 Implementierung des Playwright Test

Die Installation und Implementierung von Playwright Tests war erheblich einfacher und schneller: das Framework bietet eine VSCode Erweiterung, die alles halb-automatisch einrichtet, bzw. viele Voreinstellungen vereinfacht und fast selbsterklärend macht. Das Testen kann sehr schnell beginnen. Aus zeitlichen Gründen hat sich die Autorin entschieden, die Tests für die VoucherJS in JavaScript zu schreiben und im Firefox Browser zu beobachten, auch wenn die Möglichkeit besteht in TypeScript und allen gängigen Browsern zu testen.

Playwright benutzt eigene sogenannte Locators-Syntax, um die Elemente der Seite anzusprechen und das Playwright-eigene Page Object Model (POM), um Wartbarkeit, Lesbarkeit und Erweiterbarkeit von Testskripten zu gewährleisten.



Dadurch werden Seitenobjekte klar strukturiert: wiederkehrende Aktionen können leicht in verschiedenen Testszenarien wiederverwendet werden. Durch die Nutzung der Locators, und durch die Kombination dieser Locators Syntax, wird ein großer Wert daraufgelegt, dass eine Webseite zuallererst leicht zugänglich ist (accessible).

Nach der Installation, (NodeJS vorhanden, Playwright Erweiterung installiert), wurde erneut die maui Login-Seite getestet, die zu der VoucherJS Oberfläche führt, siehe [Anhang 4](#).

## 6. Testphase

Die manuellen Tests der Playwright Umgebung und des Codes innerhalb der maui Umgebung wurden wie folgt durchgeführt: Die manuellen Tests innerhalb des Reviews und des Entwicklungsprozesses umfassten unter anderem die folgenden Testfälle. Diese konnten alle bestanden werden.

- Öffnen der maui Webseite (J/N)
- Korrekte NutzerID (J/N)
- Einloggen möglich oder nicht
- Öffnen der VoucherJS (J/N)
- Voucher Code eingeben (Code korrekt, Code nicht korrekt, Code verbraucht, Code aktivieren)

Siehe [Abbildung 7](#) in Anhang 4.

## 7. Deploymentphase

### 7.1 Durchführung Deployment

Die manuellen Tests für Playwright wurden für die URLs

und aufgesetzt und die Ergebnisse im Firefox Browser betrachtet, wie im Anhang 4, Abbildung 6 dargestellt.

```
test('navigate to voucher', async ({ page }) => {  
  //öffne die Login Seite  
  await page.goto('https://maui-git.md.de/#/login');  
  //klicke auf das Feld Benutzername  
  await page.locator('div').filter({ hasText: /^Benutzername \*$/  
}).nth(2).click();
```

GitHub Workflows sind für den späteren Zeitpunkt geplant.

## 8. Dokumentationsphase

Das Projekt wurde von mir mit JSDoc\*, dem Dokumentationsgenerator, im Code verfasst und generiert. (Anhang 6, [Abb.8](#)). Eine README.md, bzw. eine NOTE.md Datei im GitHub Repository ist angelegt, um Notizen, die Ausnahmefälle bei der Installation beschreiben, mit Lösungsansätzen.

Und für die KiWI (Konzernweite Wissensplattform für Prozesse, Projekte und Produkte) im Abschnitt Kiwipedia freentGroup IT, eine Einleitung zur Playwright Installation verfasst. (Siehe Anhang 6, [Abb.9](#))

## 9. Fazit

### 9.1 Soll/ Ist- Vergleich

Im Rahmen des Abschlussprojekts wurde ein Teil der Funktionalitäten umgesetzt, während andere Aspekte zu einem späteren Zeitpunkt implementiert werden:

Es wurde ursprünglich geplant, bei beiden Testing-Werkzeugen - Puppeteer und Playwright - auch die GitHub-Actions Workflows anzupassen. Aber bei der Zeitberechnung und -Erfassung wurde schnell deutlich, dass dies erst nachträglich implementiert werden kann (Siehe [Tabelle 6](#)).

Auch hatte sich im Laufe der Vergleichsarbeit herausgestellt, dass Playwright die bessere Option für ein Testingframework für das Team Auftragserfassung ist.

Somit bleibt als nächster Schritt nur die Anpassung der Github-Workflows für Playwright offen.

### 9.2 Lessons Learned

Die Erfahrung der Projektplanung im IT-Bereich des eigenen Teams durchzuführen, mit einer vorgegebenen Aufgabe, von Start bis zum Abschluss innerhalb der vorgegebenen Zeit, mit vergleichsweise zu Kollegen wenig vorheriger Erfahrung in der parallelen Durchführung der Tests in verschiedenen Testing-Werkzeugen, und unter Einbeziehung der Inhalte und Themen der Ausbildung, war sehr wertvoll und lehrreich. Auch wenn die Zeitplanung nicht vollends zutraf, die agile Arbeitsweise und das Prinzip von MVP waren sehr hilfreich, den entscheidenden Teil der komparativen Arbeit und der Dokumentation als eine Betrachtungseinheit abzuschließen.

Die Planungsabweichungen sind der [Tabelle 6, Soll-/Ist-Vergleich, S.XIII](#) zu entnehmen.

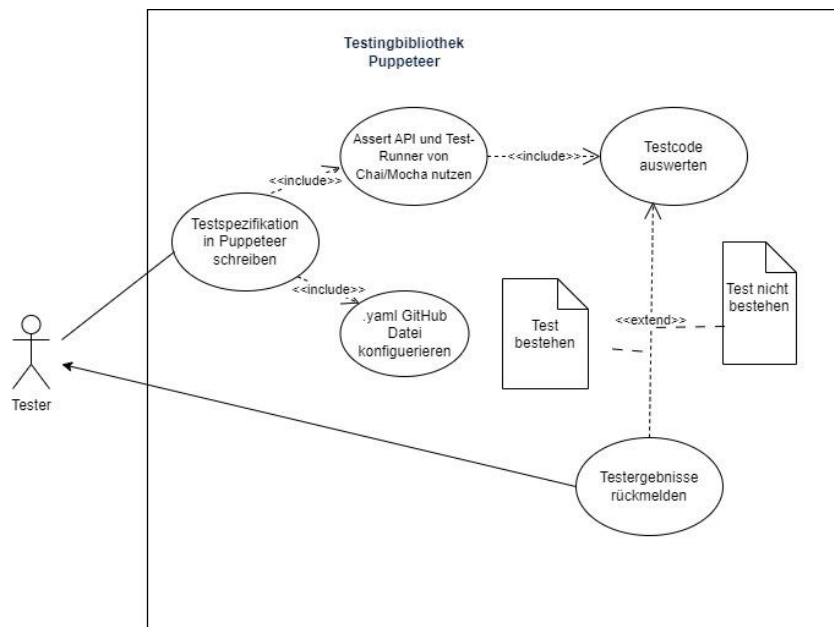
Die ständige Kommunikation mit der Ausbilderin, nach Abschluss jeder kleinen Einheit, entsprach im kleinsten Rahmen der üblichen Scrum-Arbeitsweise eines Teams an Tickets, mit kurzer Rückmeldung (Feedback) oder Review (Rückschau des Tages / der Woche), hier stattdessen mit einem Kanban-Board. Diese Arbeitsweise ist sehr förderlich, nicht nur für die Produkte und Ergebnisse, die produziert werden, sondern auch für die persönliche Entwicklung und eine schnelle Integration in ein Team oder Unternehmen.

### 9.3 Ausblick

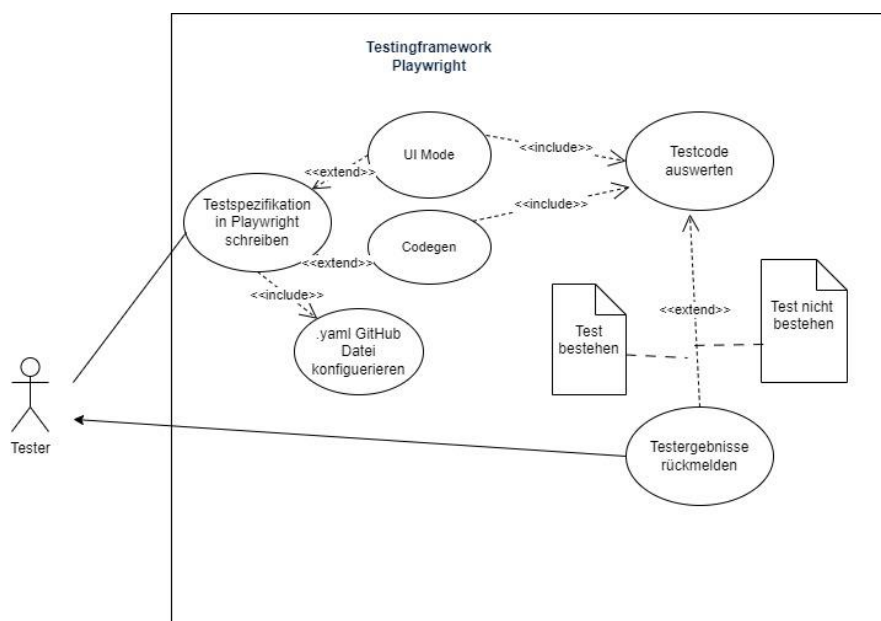
Der ursprüngliche Ansatz, GitHub Workflows für beide Testing-Werkzeuge Puppeteer und Playwright für das Team zu integrieren wurde zunächst geändert, da sich im Laufe der Vergleiche herausgestellt hatte, dass Playwright die bessere Wahl für das Team Auftragserfassung ist.

Die Aufgabe bleibt, noch Tests mit anderen Komponenten durchzuführen, auch mit TypeScript zu testen, danach zunächst Playwright in alle Workflows des GitHub-Repository im Team Retail & ECM Auftragserfassung und sukzessive in Workflows anderer Teams zu integrieren.

## Anhang 1: UML-Anwendungsfalldiagramm (Use Case Diagramm) Puppeteer



## Anhang 2: UML-Anwendungsfalldiagramm (Use Case Diagramm) Playwright



## Anhang 3:

Funktion	Cypress	Puppeteer	Playwright
Integrierter Test Runner	●	×	●
Assert / Expect API	●	×	●
Unterstützte Browser	Chrome Edge Firefox	Chrome Firefox	Chrome Edge Firefox Safari Opera
Unterstützte Sprachen	Java, JavaScript, TypeScript, C#, Python	JavaScript, TypeScript	Java, JavaScript, TypeScript, .NET, Python
Mocking von API-Calls	●	●	●
Automatische Retry Ability/ Erneute Testversuche	●	● <sup>4</sup>	●
Integriertes Dashboard	●	×	×
Integrierter Steps -Debugger	●	×	●
Screenshots	●	●	●
Videoaufnahme	●	×	●
Code Abdeckung/ Code Coverage	●	●	●
Automatisiertes Onboarding	●	×	●

Tab.4: Vergleich der Funktionalitäten nach Kriterien, die bei verschiedenen vergleichbaren Funktionen, die für Tester von Bedeutung sind.

<sup>4</sup> Retries – erneutes Versuchen von Tests, die nicht erfolgreich waren, ist bei Puppeteer relativ neu als Funktionalität, noch nicht ganz stabil, deshalb wird sie in der Puppeteer Dokumentation als “experimental” genannt: [Zitat]: „Locators is a new, experimental API“ [online] <https://pptr.dev/guides/page-interactions> [19.04.2024]

## Anhang 4:

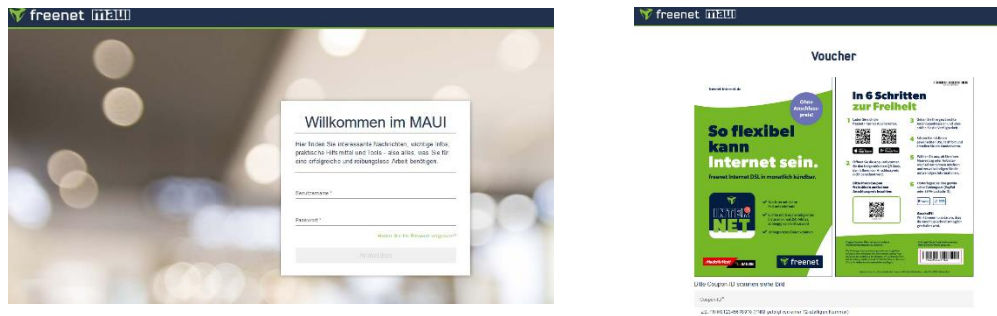


Abb. 5: Die maui Frontend Login-Seite und der VoucherJS dahinter, mit dem Voucher Code der von Mitarbeitern eingegeben oder eingescannt wird.

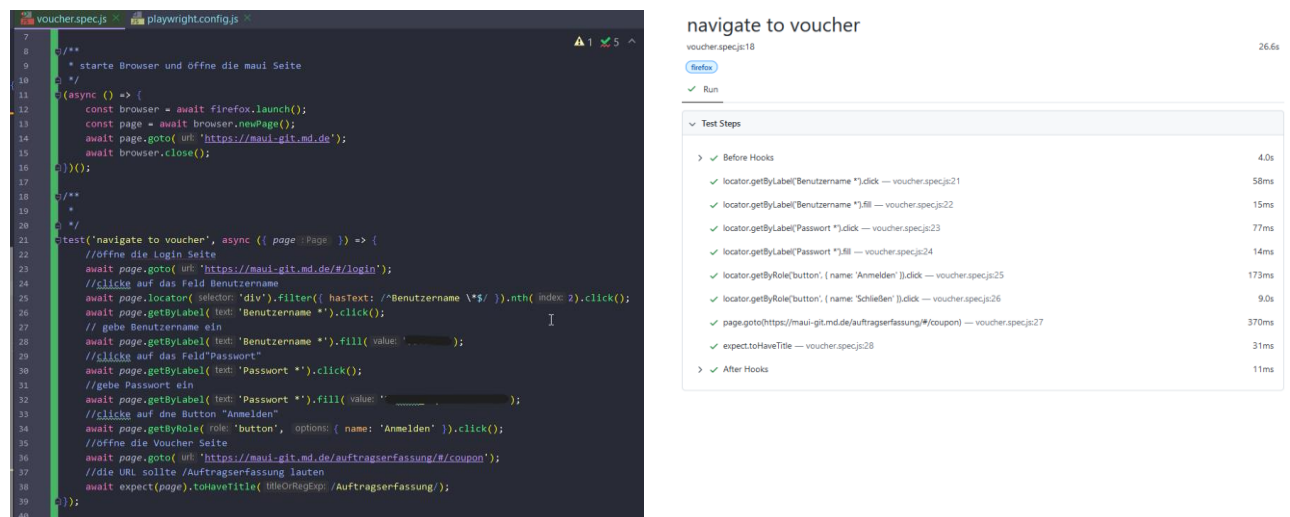


Abb. 6: Beispiel eines VoucherJS Tests in Playwright mit Ergebnissen dargestellt im Firefox Browser.

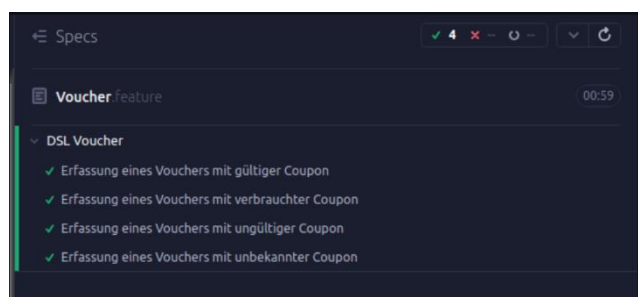


Abb. 7: VoucherJS Test Szenarien, die aus Cypress ins Playwright übernommen werden.

## Anhang 5: Projektphasen ausführlich

Analysephase	11h
Analyse des Ist-Zustandes	2h
Anwendungsfälle	1h
Nutzwertanalyse	3h
Wirtschaftlichkeitsanalyse	3h
Projektkosten	1h
Amortisationsdauer	1h
Konzeptentwicklung	7h
Qualitätssicherung	6h
Sequenzdiagramm	1h
Implementierung	38h
Implementierung <i>Puppeteer</i> Tests	18h
Implementierung <i>Playwright</i> Tests	20h
Testphase	6h
Deploymentphase	8h
Dokumentationsphase	10h
Erstellung Projektdokumentation	6h
Anpassung Anwenderdokumentation	3h
Anpassung Entwicklerdokumentation	1h
<b>Gesamt</b>	<b>80h</b>

Tab.5: Angaben zur pauschalen Zeitplanung des Projekts

## Anhang 6: Soll-/Ist-Vergleich

Projektphase	Soll	Ist	Differenz
Analyse	11h	11h	
Konzeptentwicklung	7h	4h	-3h
Implementierung	38h	40h	+2h
Testphase	10h	8h	-2h
Deployment	8h	5h	-3h
Dokumentation	10h	12h	+2h
Gesamt	80h	80h	

Tab.6: Zeitplanung Soll-/Ist-Vergleich

## Anhang 7: Dokumentation

```
/**
 *
 * @type {{vouchernumber: RegExp}}
 */
const pattern = {
  vouchernumber: /^FNI[A-Z]\d{12}$/ ,
}

/**
 *
 * @param idvalid
 * @param idinvalid
 * @param field
 * @param fieldName
 */
const validatingVoucher = function(idvalid, idinvalid, field, fieldName){
  if (pattern[fieldName].test(field.value)){
```

Abb. 8: Screenshot der JSDoc für den lokalen Voucher Test

The screenshot shows the freenet documentation interface. At the top, there is a navigation bar with links like 'Startseite', 'Zuletzt verwendet', 'Bereiche', 'Teams', 'Apps', 'Vorlagen', and a '+ Erstellen' button. Below the navigation bar, there is a search bar and a 'Gespeichert' button. The main content area displays instructions for installing Playwright and Cucumber. The text reads: 'Danach installiere die Cucumber Erweiterung.' followed by 'Oder installiere in der cmd:'. Below this, there are two code blocks: '1 npm init playwright@latest' and '1 npm i @cucumber/cucumber'.

Erstelle Ordner "features" und "steps" unter dem "tests" Folder

„features“: der Ordner für die „features“ Datei

Steps können wie folgt übernommen werden:

The screenshot shows a code editor with a Gherkin script. The script is written in German and describes a scenario for creating a voucher. The text reads: '1 #language: de', '2', '3 Funktionalität: DSL Voucher', '4', '5', '6 Grundlage: Ich logge mich als MSD Shop Benutzer ein', '7 Angenommen Ich logge mich als MSD Shop Benutzer ein', '8', '9 Szenario: Erfassung eines Vouchers mit gültiger Coupon', '10 Angenommen die Voucher Erfassung wurde aufgerufen', '11', '12', '13', '14'. The script is highlighted in yellow.

Abb. 9: Screenshot der Dokumentation in Kiwipedia der freenet DSL