

Fakultät für Informatik

Studiengang Wirtschaftsinformatik

Vergleich von JavaScript-E3E-Test Werkzeugen für Single-Page-Anwendungen am Bespiel einer Angular-Anwendung

Seminararbeit

von

Dominik Ampletzer

Datum der Abgabe: xx.xx.2019

Erstprüfer: Herr Reimer

Zweitprüfer: Prof. Beneken

ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim den, xx.xx.2019

Dominik Ampletzer

**Abstract**

Diese Seminararbeit beschäftigt sich mit dem Vergleich dreier E2E-Test-Werkzeuge welche auf JavaScript basieren und für den Einsatz mit Web-Anwendungen entwickelt wurden. Bei diesen Werkzeugen handelt es sich um: Protractor, Nightwatch.js und WebdriverIO. Diese wurden einen Kurzcheck unterzogen, welche u.a. Lizenz, Reifegrad, Support, Dokumentation, Bekannte Mängel, Projekt-Aktivität und Bekanntheitsgrad beinhaltet. Die zwei besten Werkzeuge wurden darauf anhand einer Angular-Anwendung an verschiedenen Anwendungsfällen getestet. Dadurch soll die Integrationsfähigkeit, Aufwand und Leistungs -bzw. Fähigkeitsumfang der Werkzeuge getestet werden. Aus dem Kurzcheck und der Testimplementierung soll ersichtlich werden, welches Werkzeug im Spezialfall von Angular-Anwendungen aber auch im Allgemeinen das bessere Werkzeug ist und demnach verwendet werden soll.

Es wurde ersichtlich, dass Protractor im Spezialfall von Angular-Anwendungen aber auch im Allgemeinen das bessere Werkzeug ist. Demnach kann Protractor vor WebdriverIO und WebdriverIO vor Nightwatch.js empfohlen werden.

Schlagworte

E2E-Werkzeuge, Angular, Protractor, Nightwatch.js, WebDriverIO

**Inhaltsverzeichnis**

[Abbildungsverzeichnis iii](#_Toc9086308)

[Tabellenverzeichnis iv](#_Toc9086309)

[Code-Verzeichnis v](#_Toc9086310)

[Glossar / Abkürzungsverzeichnis vi](#_Toc9086311)

[1 Einleitung 1](#_Toc9086312)

[1.1 Problemstellung 1](#_Toc9086313)

[1.2 Vorgehen & Ziel der Arbeit 1](#_Toc9086314)

[1.3 Rahmenbedingungen 1](#_Toc9086315)

[2 E2E-Test-Werkzeuge 1](#_Toc9086316)

[2.1 Kurzüberprüfung / Schnelltest 3](#_Toc9086317)

[2.2 Protractor 3](#_Toc9086318)

[2.2.1 Lizenz: 3](#_Toc9086319)

[2.2.2 Reifegrad 3](#_Toc9086320)

[2.2.3 Stabile-Version-Veröffentlicht 4](#_Toc9086321)

[2.2.4 Support 4](#_Toc9086322)

[2.2.5 Dokumentation 4](#_Toc9086323)

[2.2.6 Bekannte Mängel/Fehler/Sicherheitslöcher 4](#_Toc9086324)

[2.2.7 Qualität des Projekts 4](#_Toc9086325)

[2.2.8 Aktivitäten 4](#_Toc9086326)

[2.2.9 Bekanntheitsgrad 4](#_Toc9086327)

[2.2.10 Unterstützer des Projekts 5](#_Toc9086328)

[2.3 Nightwatch.js 5](#_Toc9086329)

[2.3.1 Lizenz: 5](#_Toc9086330)

[2.3.2 Reifegrad 5](#_Toc9086331)

[2.3.3 Stabile-Version-Veröffentlicht 5](#_Toc9086332)

[2.3.4 Support 5](#_Toc9086333)

[2.3.5 Dokumentation 5](#_Toc9086334)

[2.3.6 Bekannte Mängel/Fehler/Sicherheitslöcher 5](#_Toc9086335)

[2.3.7 Qualität des Projekts 5](#_Toc9086336)

[2.3.8 Aktivitäten 6](#_Toc9086337)

[2.3.9 Bekanntheitsgrad 6](#_Toc9086338)

[2.3.10 Unterstützer des Projekts 6](#_Toc9086339)

[2.4 WebDriver.IO 6](#_Toc9086340)

[2.4.1 Lizenz: 6](#_Toc9086341)

[2.4.2 Reifegrad 6](#_Toc9086342)

[2.4.3 Stabile-Version-Veröffentlicht 6](#_Toc9086343)

[2.4.4 Support 6](#_Toc9086344)

[2.4.5 Dokumentation 6](#_Toc9086345)

[2.4.6 Bekannte Mängel/Fehler/Sicherheitslöcher 7](#_Toc9086346)

[2.4.7 Qualität des Projekts 7](#_Toc9086347)

[2.4.8 Aktivitäten 7](#_Toc9086348)

[2.4.9 Bekanntheitsgrad 7](#_Toc9086349)

[2.4.10 Unterstützer des Projekts 7](#_Toc9086350)

[2.5 Zusammenfassung/ Kurzübersicht der Ergebnisse 7](#_Toc9086351)

[3 Vergleich von Protractor und Webdirver.IO 8](#_Toc9086352)

[3.1 Implementierungsgrundlage und Anwendungsfällen 9](#_Toc9086353)

[3.2 Protractor Details 9](#_Toc9086354)

[3.2.1 Browserunterstützung 9](#_Toc9086355)

[3.2.2 Page-Object-Pattern 10](#_Toc9086356)

[3.2.3 Zugriffsmöglichkeiten 10](#_Toc9086357)

[3.2.4 Bedingungen – warten auf Ereignisse 11](#_Toc9086358)

[3.2.5 Ergebnisse der Implementierung 12](#_Toc9086359)

[3.2.6 Weitere Ergebnisse 14](#_Toc9086360)

[3.3 WebdriverIO Details 15](#_Toc9086361)

[3.3.1 Browserunterstützung 15](#_Toc9086362)

[3.3.2 Page-Object-Pattern 15](#_Toc9086363)

[3.3.3 Zugriffsmöglichkeiten 16](#_Toc9086364)

[3.3.4 Bedingungen –warten auf Ereignisse 16](#_Toc9086365)

[3.3.5 Ergebnisse der Implementierung 17](#_Toc9086366)

[3.4 Direkter Vergleich 18](#_Toc9086367)

[4 Fazit 19](#_Toc9086368)

[Literaturverzeichnis 22](#_Toc9086369)

# Abbildungsverzeichnis

[Abbildung 1: Vereinfachter Ablauf 2](#_Toc9685769)

[Abbildung 2: Ablauf mit E2E-Test-Werkzeug 2](#_Toc9685770)

[Abbildung 3: Kurzübersicht des Schnelltests 7](#_Toc9685771)

[Abbildung 4: Trendanalyse 7](#_Toc9685772)

[Abbildung 5: Protractor Page-Object-Pattern 9](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\web-application\Seminararbeit_SQS_Ampletzer.docx#_Toc9685773)

[Abbildung 6: Protractor - Ausschnitt 1 - Logische Operatoren 11](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\web-application\Seminararbeit_SQS_Ampletzer.docx#_Toc9685774)

[Abbildung 7: Jasmine Testaufbau 12](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\web-application\Seminararbeit_SQS_Ampletzer.docx#_Toc9685775)

# Glossar / Abkürzungsverzeichnis

|  |  |
| --- | --- |
|  |  |
| UI | User-Interface – Benutzeroberflächen |
| HTTP | Hypertext Transfer Protocol – Datenübertragungsprotokoll welches u.a. dazu genutzt wird um Web-Seiten / Web-Anwendungen zu transportieren |
| E2E | End-To-End – End zu End, dahinter versteckt sich der komplette Ablauf in einem System durch alle Schichten. Von der Oberfläche über den Server zur Datenbank und zurück |
| Shadowing | Überschatten von Variablennamen durch mehrmaliges verwenden innerhalb eines Funktionsbereiches |
|  |  |
|  |  |
| GPL | Gnu Public Licence – weit verbreitete offene Lizenz im Open-Source-Bereich |

# Einleitung

In dieser Arbeit geht es um die Analyse und den Vergleich von drei End-to-End-Test-Werkzeugen (E2E-Test). Die Werkzeuge simulieren den Anwender und überprüfen die Reaktionen bzw. das Ergebnis der Interaktion mit einer Web-Anwendungen. Die Werkzeuge sind für das Testen von Web-Anwendungen und Web-Seiten geeignet. Aus diesem Grund wird als Testfallgebende Anwendung eine Single-Page-Anwendung verwenden, welche mit dem Framework Angular entwickelt wurde.

## Problemstellung

Da es im Web-Umfeld sehr viele Test-Werkzeuge und dies für E2E-Tests ähnlich sind, stellt sich natürlich die Frage, welches Werkzeug eignet sich am besten. Da viele Web-Anwendungen als Single-Page-Anwendung entwickelt werden. Geht es speziell um die Frage, welches Werkzeug ist für Single-Page-Anwendungen am besten geeignet.

## Vorgehen & Ziel der Arbeit

Diese Arbeit gibt einen kurzen Überblick über Protractor, Nightwatch.js und WebDriverIO. Mittels einer Kurzüberprüfung werden die Werkzeuge sowohl analysiert als auch das schlechterste Werkzeug verworfen um die besseren beide Werkzeuge anhand einer Beispielsanwendung welche mit dem Framework Angular entwickelt wurde zu testen. Hierfür werden sowohl die Implementierung als auch verschiedene Testfälle betrachtet.

Ziel der Arbeit ist es unter Berücksichtigung der Kurzüberprüfung und des Werkzeug-Tests die Aussage treffen zu können, welches Werkzeug für E2E-Tests für Angular-Anwendungen im Speziellen und Single-Page-Anwendungen im Allgemeinen zu verwenden ist.

## Rahmenbedingungen

Die Werkzeuge, welche in dieser Arbeit betrachtet werden und ausgewählt wurden, sind folgenden Rahmenbedingungen unterworfen:

* Programmiersprache JavaScript und/oder TypeScript
* Außerhalb der Beta-Phase
* Node.js basierend
* WebDriver API basierend
* Interessant für Autor ☺

# E2E-Test-Werkzeuge

Einführend zu den Test-Werkzeugen ist zu erwähnen, dass die Begriffe UI-Test-Werkzeug und E2E-Test-Werkzeuge als Synonym betrachtet werden.

In Abbildung 1: Vereinfachter Ablauf ist zu sehen, wie die Interaktion mit einer Web-Anwendung vonstattengeht. Der Anwender interagiert mit der Benutzeroberfläche (UI) welche ein Teil der Web-Anwendung ist. Die Web-Anwendung stellt Anfragen bei einem Server, welche Antwortet. Diese Antworten werden in welcher Form auch immer auf der UI angezeigt, was der Anwender wiederum wahrnimmt.

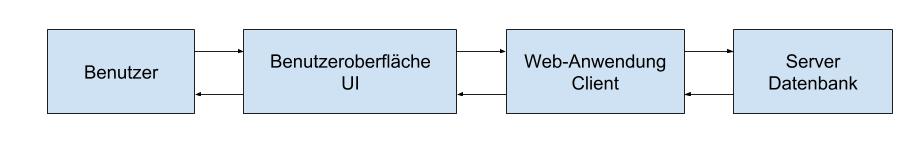


Abbildung : Vereinfachter Ablauf

Bei einem E2E-Test-Werkzeug wird der Benutzer durch das Werkzeug ersetzt. Was zur Folge hat, dass die Interaktion mit der UI durch das Werkzeug passiert. Dadurch kann geprüft werden, ob die UI so reagiert wie gewünscht, der Ablauf durch das komplette System und zurück so funktioniert wie erwartet, sowie eine Automatisierung stattfinden kann. Siehe Abbildung 2: Ablauf mit E2E-Werkzeug

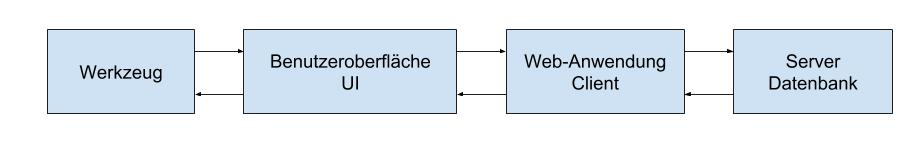


Abbildung : Ablauf mit E2E-Test-Werkzeug

## Kurzüberprüfung / Schnelltest

Für die Kurzüberprüfung wurde ein Katalog mit folgenden Kriterien zusammengestellt:

* Lizenz: Art der Lizenz, wobei die Einstufung MIT/Apache – GPL – Proprietär
* Reifegrad: Existenzzeit und stabile Veröffentlichung, Versionsnummer
* Support: Öffentliche Unterstützung und Professionelle Unterstützung
* Dokumentation: Qualität der Dokumentation, Tutorials, Starthilfen, Alter der Dokumentation
* Qualität des Projekts: Testabdeckung / Bugs durch statische Code Analyse
* Aktivitäten: Stetigkeit von Commits, letzte Veröffentlichung, Anzahl aktiver Entwicklern
* Bekanntheitsgrad: öffentliches Interesse an Projekt anhand von Trendanalyse, Foren Aktivitäten
* Unterstützer des Projekts: Firma oder Personen die hinter dem Projekt stehen, Rückschluss auf Langlebigkeit

Diese Kriterien werden als Schwerwiegend betrachtet und können durchaus dazu führen, dass bei schlechter Beurteilung selbst schon dazu führen, dass die Werkzeuge nicht zu empfehlen sind. So ist Beispielsweise ein Werkzeug, welches keine Dokumentation hat, oder seit Jahren nicht mehr gepflegt wird nicht für produktiven Einsatz zu empfehlen.

## Protractor[[1]](#footnote-1)

Protractor ist ein Werkzeug welches 2013 in der Version 0.2 durch das JavaScript-Framework AngularJS zur Verfügung steht. Es wurde für AngularJS[[2]](#footnote-2) entwickelt kann mittlerweile jedoch auch für andere Web-Anwendungen verwendet werden. Dennoch liegt der Schwerpunkt von Protractor in der Verwendung mit Angular was der Nachfolger von AngularJS ist. Hinter dem JavaScript-Framework Angular wiederum steht Google.

Protractor wurde demnach für das Angular Ökosystem entwickelt kann aber ohne weiteres mit anderen Frameworks wie React verwendet werden. Hierfür gibt es eine *waitForAngular*-Funktion welche deaktiviert werden kann. Auch das der meist in Angular verwendete Test-Framework Jasmine kann durch andere Test-Frameworks wie Mocha oder Cucumber ersetzt werden.

Neben JavaScript können Protractor Test mit TypeScript geschrieben werden. Was besonders in Angular- Anwendungen, welche in TypeScript geschrieben sind, den Vorteil mit sich bringt, sich nicht an andere Syntax gewöhnen zu müssen.

### Lizenz

MIT

### Reifegrad

Ist als Hoch anzusehen, da das Projekt seit 2013 existiert und in der Version 6.0.0 angekommen ist. Des Weiteren haben seit 2013 ca. 100 Veröffentlichungen stattgefunden. Aktuell gib es 13 Bugs in der Offiziellen Issue-Liste und bie aktuelle Version ist stabil

### Support

Leider kein Professioneller Support. Der Community-Support[[3]](#footnote-3) hingegen ist mit StackOverflow (ca. 10.000 Fragen), Gitter (ca. 1.250 Nutzer) und Angular-Disussion-Group (ca. 19.000 Themen) sehr gut.

### Dokumentation

Auf der Protractor Web-Seite findet sich neben einem Tutorial und einem Schnellstart einige Beschreibungen mit Beispielen für die Verschiedenen Bereiche von Protractor vorhanden. Die Dokumentation ist leicht veraltet jedoch sehr ausführlich und besonders in den unterschiedlichen Bereichen und der Api-Referenz sehr übersichtlich und hilfreich. Da diese sich auf Version 5.4.1 bezieht. Mittlerweile ist jedoch Version 6 aktuell, was zwei Veröffentlichungen nach Version 5.4.1 ist.

### Qualität des Projekts

Die interne Struktur von Protractor ist sehr übersichtlich und klar. Zusätzlich zum eigentlichen Werkzeug sind im Repository noch Beispiele und die Website enthalten. Durch statische Codeanalyse konnte auf den ca. 14.000 Zeilen Code von Protractor 6 Bugs gefunden werden, welche bis auf einen potentiell falschen Rückgabewert vernachlässigt werden können. Negativ sind jedoch 102 „unschöne“ Stellen zu werden, bei denen es sich um viel Shadowing und mangelndes aufräumen des Codes handelt. Aber in einem Ausmaß, dass durchaus noch vertretbar ist.

### Aktivitäten, Bekanntheit und Unterstützung

Der letzte Release war im ersten Quartal 2019. Seit dem ist die Commit Rate relativ gering. Was jedoch auch auf den Reifegrad des Werkzeuges zurückzuführen ist. Was aber auch am Reifegrad des Frameworks liegt.

Der Bekanntheitsgrad von Protractor als hoch betrachtet werden. Dies ruht auch daher, dass Protractor standardmäßig in Angular-Anwendungen integriert ist. Zusehen ist die Verbreitung durch ca. 500 Beobachtungen, ca. 8000 Sternen und ca. 2000 Forks des Git-Repositories sowie den ca. 3.500 Followern auf Twitter.

Der Hauptunterstützung von Protractor ist Google sowie mehr als 250 Beitragende Entwickler. Durch die Unterstützung von Google kann prognostiziert werden, dass Protractor noch längere Zeit existieren und weiterentwickelt wird.

## Nightwatch.js

Nightwatch.js ist ebenfalls ein E2E-Test-Werkzeug. Es stammt vom Norwegischen Entwickler PineView Software. Neben der Test-Automatisierung legt Nightwatch.js Wert auf die Erweiterbarkeit und Konfigurierbarkeit für eigene Projekte. Außerdem können mit Nightwatch.js Unit-Tests geschrieben werden und mit Mochas Export Schnittstelle verbunden werden.

### Lizenz:

MIT

### Reifegrad

Eher als mittel bis gering anzusehen. Das Werkzeug ist ab 2014 in der Version 0.2.8 verfügbar. Seitdem gab es über 150 Veröffentlichungen. Diese Stammten allerdings immer von einer Person. Version 1 wurde im April 2018 veröffentlich und befindet sich somit ca. 1 Jahr außerhalb der Beta-Phase. Jedoch ist die aktuelle Version stabil und beinhaltet in der Issue-Liste 7 Bugs.

### Support

Professionellen Support gibt es indirekt durch Pine View Software. Das Unternehmen bietet Consulting und Trainings für IT-Projekte an. Einen klassischen Maintenance-Vertrag gibt es jedoch nicht.

Nightwatch.js wird in erster Linie durch die Community supported[[4]](#footnote-4). Es gibt eine Google Gruppe, welche ca. 1.500 Themen umfasst, welche zeitnah beantwortet werden und StackOverflow mit ca. 1.000 Fragen, welche jedoch zu Großen teilen nicht beantwortet wurden.

### Dokumentation

Die Dokumentation macht einen sehr guten Eindruck. Es finden sich Schnelleinstiege, Hilfestellungen wie das Werkzeug zu verwenden ist. Die Dokumentation ist auf den neuesten Stand. So entspricht diese zum Stand 22.04.19 bereits der Version 1.1.2 welche es aktuell noch als Pre-Release gibt. Api-Referenzen sind ebenfalls sehr ausführlich und leicht verständlich.

### Qualität des Projekts

Der Nightwatch.js Quellcode ist überschaubar und klar strukturiert. Die Testabdeckung ist mit 85% auf einem guten Niveau. Statische Code-Analyse deckt vier Bugs auf, welche sich jedoch nur auf zwei unnötige Strings und zweimal auf einen Objektvergleich mit einem Objekt, welches mit sich selbst verglichen wird. Diese Fehler sind jedoch nicht gravierend. Was außerdem positiv zu werten ist, dass bei den 27.000 Zeilen Code des Projekts nur 33 „unschöne“ stellen zu finden waren.

### Aktivitäten, Bekanntheit und Unterstützung

Die Release-Häufigkeit ist sehr hoch. So gab es im Q4 2018 und Q1 2019 zusammen 8 Veröffentlichungen. Passend dazu ist auch die Commit-Rate sehr hoch. Unschön ist hingegen, dass das Projekt zwar 74 Beteiligte Entwickler besitzt jedoch nur einen, welcher ernsthaft Entwickelt. Der Rest der Beteiligten Entwickler machte in der Vergangenheit nur sehr kleine Änderungen.

Eine große Bekannt ist bei Nightwatch.js ebenfalls nicht festzustellen. So ist der erste Eindruck durch das Git-Repository mit 282 Beobachtungen, ca.9000 Sternen und 883 Forks sowie ca. 3000 Twitter-Followern sehr positiv. In Berücksichtigung des Community-Supports auf StackOverflow und der Google-Trend-Analyse wie in Abbildung 4: Trendanalyse zu sehen, kann der Bekanntheitsgrad auf einen mittleren Bereich eingeschränkt werden.

Der Hauptunterstützer Pine View Software aus Oslo ist ebenfalls kaum Bekannt, was zu einer Unsicherheit in der Lebensdauer von Nightwatch.js führt.

## WebdriverIO

WebdriverIO ist eine weitere WebDriver API Implementierung, welche bereits im Namen verrät, dass es sich um eine solche handelt. WebdriverIO kann sowohl mit TypeScript als auch JavaScript verwendet werden und unterstützt die großen Test-Frameworks: Jasmine, Cucumber und Mocha.

### Lizenz:

MIT

### Reifegrad

Der Reifegrad von WebdriverIO ist als hoch anzusehen. Es wurde in der Version bereits 2013 veröffentlicht. Seitdem gab es ca. 200 Veröffentlichungen. Was einen sehr kleinen Release-Zyklus kennzeichnet. Die Aktuelle Version 5.7.16 handelt es sich um eine stabile Version, welche in der Issue-Liste 8 Bugs führt.

### Support

Community-Support[[5]](#footnote-5) ist in erster Linie durch einen aktiven Gitter-Chat mit ca. 4.000 Nutzer gegeben. Zusäzlich gibt es Support durch StackOverflow (ca. 800 Fragen).

### Dokumentation

Die Dokumentation macht einen sehr guten Eindruck. Es finden sich Schnelleinstiege, Code-Beispiel und Hilfestellungen wie das Werkzeug zu verwenden ist. Zusätzlich werden noch Online-Kurse angeboten. Diese sind jedoch Kostenpflichtig. Bei der Dokumentation bedient die Aktuelle Version 5 jedoch kann die genauer Version nicht ermittelt werden.

### Qualität des Projekts

Durch statische Code-Analyse wurden drei Bugs aufgeworfen, zwei hiervon können potenziell zu ernsthaften Fehlern führen. So gibt es eine *if*-Bedingung welche immer *true* ist und einen fehlende *null*-Überprüfung.

Die Restlichen 22.000 Zeilen Code sind mit 61 „unschönen“ Stellen sehr sauber und mit 98% Testabdeckung auch gut getestet.

Der Quellcode ist in sehr viele kleine Pakete unterteilt, nach kurzer Zeit sehr klar, sauber strukturiert und gleich aufgebaut sind.

### Aktivitäten, Bekanntheitsgrad und Unterstützung

Die Release-Häufigkeit ist sehr hoch. Mit mehreren Veröffentlichungen teilweise innerhalb eines Monats. Die Commit-Rate ist dazu passen ebenfalls hoch.

Die Bekanntheit vorn WebdriverIO kann mit mittel-hoch betrachtet werden. So kann dem Git-Repository entnommen werden, dass es 215 Beobachtungen, 4800 Sterne und 1370 Forks gibt.

WebdriverIO wird durch 116 Beteiligten Entwickler unterstützt, wobei vier von diesen als Hauptentwickler bezeichnet werden können. Hinter WebdriverIO steht die JS.Foundation als starker Unterstützer was auf eine länger zeitige Unterstützung und dadurch Lebenserwartung schließen lässt.

## Zusammenfassung/ Kurzübersicht des Schnelltests

|  |  |  |  |
| --- | --- | --- | --- |
| Kriterium | Protractor | Nightwatch.js | Webdriver.IO |
| Lizenz | MIT | MIT | MIT |
| Reifegrad | Hoch | Mittel - Niedrig | Mittel – Hoch |
| Stabile-Version-Veröffentlicht | Ja | Ja | Ja |
| Support | Gut durch Community | Gemäßigt durch Community; Unsichere durch Pine View Software (Kostenpflichtig) | Gemäßigt – gut durch Community. Zusätzliche Tutorials gegen Gebühr |
| Dokumentation | Gut | Gut | Sehr Gut |
| Qualität des Projekts | Gut | Gut | Gut |
| Aktivitäten | Ja, nach Release Ruhig | Sehr viel Aktivität | Viel Aktivität |
| Bekanntheit | Hoch | Gering - Mittel | Mittel-Hoch |
| Unterstützung | Google | PineView Software | JS.Foundation |

Abbildung 3: Kurzübersicht des Schnelltests

Zur Bekanntheit sollte noch Abbildung 4: Trendanalyse betrachtet werden. Dort ist zu sehen wie die drei Werkzeuge im Google-Trend zueinander abschneiden. Als Kritik an der Grafik ist anzumerken, dass die Begriffe Protractor, Nightwatch und Webdriver nicht nur für die Werkezeuge benutzt werden. Somit ist die Abbildung nur als Indikator zu betrachten. Zusätzlich Normiert sich die Trendanalyse immer auf 100 und zeigt keine Aussagekräfte Skala. Weitere Kritikpunkt an diesem Kapitel ist die Tatsache, dass Werte wie Veröffentlichungen, StackOverflow-Themen und Contributor-Zahlen nur Momentaufnahmen darstellen und leider keine Schnittstellen angeboten werden um diese Zahlen für Wissenschaftliche Zwecke zu erheben.

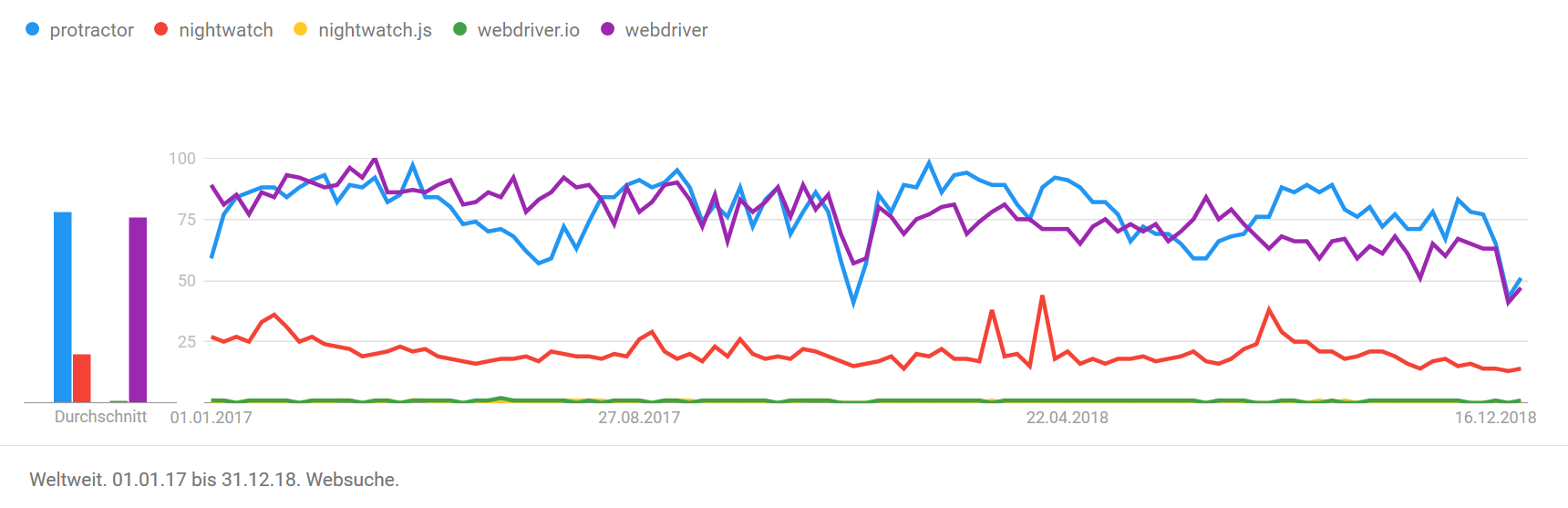


Abbildung : Trendanalyse

# Vergleich von Protractor und Webdirver.IO

Nach der Schnellüberprüfung wird für den detaillierten Vergleich nur noch die zwei Werkzeuge Protractor und WebdriverIO betrachtet. Dies liegt daran, dass schlechte Support-Möglichkeiten, einen geringen Reifegrad und nur einen Hauptentwickler Nightwatch.js einen negativen Eindruck hinterlässt. Die Unterstützung von Pine View Software im Gegensatz zu Google und JS.Foundation ist ebenfalls als eher klein/gering anzusehen, was sich außerdem schlecht auf die Lebenserwartung bzw. Langlebigkeit des Werkzeugs auswirken könnte.

Verglichen wird in diesem Kapitel der Funktionsumfang in Form von gelösten Anwendungsfällen, der Aufwand für Integration und das Schreiben von Tests. Außerdem die Einfachheit in Form von Code Zeilen (Lines-of-Code = LoC) für die Tests und auch deren Lesbarkeit.

## Implementierungsgrundlage und Anwendungsfällen

Als Implementierungsgrundlage der Werkzeuge dient eine einfache Angular-Anwendung. Diese besteht neben Boilerplate-Code, aus dem Testframework Jasmine, wenigen Komponenten sowie wenigen Modulen um Grundfunktionalität abbilden zu können. Es wurde bei der Anwendung kein Wert auf Design oder Feature-Umfang gelegt. Vielmehr sollen Anwendungsfälle abgedeckt werden, wie sie im Produktivsystem gewöhnlich sind.

Anwendungsfälle:

1. Navigation in Anwendung mit Nutzer Eingaben und Ergebnissen
2. Navigation auf Asynchron geladenes Modul mit dynamischen geladenen von Inhalten. Während des Ladens werden Animationen gezeigt.
3. Öffnen eines Dialogs mit dynamischen geladenen Inhalten. Diese öffnen weiteren Dialog mit identischen Element-Ids.
4. Neuen Browser-Tab öffnen in diesen Eingaben vornehmen und Tab zurück wechseln.
5. Daten-Upload (kein Drag & Drop) mittels HTML-File-Input
6. Nutzung eines i-Frame (Webseite der Technischen Hochschule Rosenheim)

## Protractor Details

Protractor kann auch in nicht Angular-Anwendungen verwendet werden. Dafür muss das Warten auf Angular-Ereignisse mittels *browser.waitForAngularEnabled(false)* deaktiviert werden.

### Browserunterstützung

Browserunterstützung findet für alle „großen“ Browsern statt. Somit wird Chrome, Firefox und Safari Out-Of-The-Box unterstützt. Für Microsoft Edge und Internet Explorer muss der Microsoft Web Driver installiert werden.

Des Weiteren wird von PhantomJS / GhostDriver abgeraten, da diese im Zusammenspiel mit Protractor Fehler verursachen können und sich zu sehr von echten Browsern unterscheiden. Dies bedeutet allerdings nicht, dass man auf Oberflächenlose Browser verzichten muss. Einige Browser wie Chrome und Firefox bieten einen oberflächenlosen Modus wodurch Werkzeuge wie PhantomJS abgelöst werden können. Zusätzlich haben der oberflächenlose Modus den Vorteil, dass der Ziel-Browser in einer CI/CD-Pipeline auf einfache Weise getestet werden kann.

Neben klassischen Web-Browsern werden Mobilen Browser durch Appium unterstützt.

### Page-Object-Pattern

Protractor ermöglicht die einfache Nutzung das Page-Object-Patterns hierbei wird eine Objekt-Repräsentation der zu Testenden Seite erzeugt. Dadurch kann auf die Elemente der Seite, wie aus der Objektorientierten Programmierung bekannt bei Attribute, zugegriffen werden. Hierfür wird den Objekten wie in Abbildung 5: Protractor Page-Object-Pattern zu sehen dem Attribut ein Element-Finder übergeben, welcher bei Zugriff das Element auf der Seite sucht bzw. findet. Des Weiteren können komplexere Funktionen hinterlegt werden um z.B. Elemente zu finden, welche nicht eindeutig identifiziert werden können, oder nur den Text eines Elements und nicht das komplette Element zurückzuliefern.

Das Page-Object-Pattern dient außerdem dazu bei Seitenänderungen einfacher die Tests anzupassen. So wird z.B. beim Austausch eines Elements nicht der Test angepasst, sondern der Element-Finder.

Abbildung 5: Protractor Page-Object-Pattern

**export class** SomePage **extends** AppPage {  
 **public dialogOpenButton**: ElementFinder= element(by.id(**'openDialogButton'**));  
 **public iFrameButton**: ElementFinder = element(by.partialButtonText(**'iFr'**));  
 **public intranetButton**: ElementFinder = element(by.partialLinkText(**'ntrane'**));  
 **public iFrame**: ElementFinder = element(by.id(**'thRosenheim'**));  
 **public subTitleFromIntranet**: ElementFinder = element(by.xpath(**'//\*[@id="c55958"]/div/h2'**));  
   
 getRow(index?: **number**): ElementFinder {  
 **return** element(by.id(**'customItem'** + index || **'0'**));  
 }  
  
 getSecondSpinner(): ElementFinder {  
 **return** element.**all**(by.id(**'spinner'**)).get(1);  
 }  
  
 getSecondListRowText(index: **number**): promise.Promise<**string**> {  
 **return** element.**all**(by.id(**'customItem'** + index)).get(1).getText();  
 }  
}

### Zugriffsmöglichkeiten

Wie in Abbildung 5: Protractor Page-Object-Pattern zu sehen bedient sich das Page-Object der *element*-Funktion um Elementfinder zu erzeugt. Diese Elementfinder können nach unterschiedliche Attribute der Elemente suchen. Hierfür bietet Protractor viele Lokalisatoren/Selektoren (Locators/Selectors) an. Neben den geerbten durch die Webdriver-Api:

* className
* css
* id
* linkText
* js
* name
* tagName
* partialLinkText
* xpath

noch zusätzliche welche teilweise nur auf Angular bezogen sind wie:

* binding
* exactBinding
* model
* buttonText
* partialButtonText
* repeater
* exactRepeater
* cssConainigText
* options
* deepCss

Diese Lokalisatoren werden mittels der *by*-Klasse erzeug und ermöglichen es so, sehr leserlich den gewünschten Lokalisator der *element*-Funktion zu übergeben.

### Bedingungen – warten auf Ereignisse

Als Grundlegendes Konzept um mit E2E-Tests beginnen zu können müssen noch Bedingungen bzw. das Warten auf Ereignisse verstanden werden. Auf Grund von Wartezeiten z.B. durch Netzwerkkommunikation oder Rendern (Visualisieren) müssen Tests gelegentlich warten. Dies kann naiv durch die *sleep/wait*-Funktion mit einer Zeitangabe des Browsers gemacht werden oder durch das Warten auf Bedingungen. Für das erstellen und warten auf Bedingungen bietet Protractor sehr viele Möglichkeiten einige davon sind in Abbildung 6: Protractor - Ausschnitt 1 - Logische Operatoren zu sehen. Die gesamte Liste umfasst:

* [alertIsPresent](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.alertIsPresent)
* [elementToBeClickable](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.elementToBeClickable)
* [textToBePresentInElement](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.textToBePresentInElement)
* [textToBePresentInElementValue](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.textToBePresentInElementValue)
* [titleContains](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.titleContains)
* [titleIs](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.titleIs)
* [urlContains](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.urlContains)
* [urlIs](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.urlIs)
* [presenceOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.presenceOf)
* [stalenessOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.stalenessOf)
* [visibilityOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.visibilityOf)
* [invisibilityOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.invisibilityOf)
* [elementToBeSelected](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.elementToBeSelected)

Hierdurch können bestimmte zustände sehr granular abgepasst werden und auf diese reagiert werden. So ist der Unterschied für viele Anwendungen zwischen [elementToBeClickable](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.elementToBeClickable), [visibilityOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.visibilityOf) und [presenceOf](http://www.protractortest.org/#/api?view=ProtractorExpectedConditions.prototype.presenceOf) nicht wirklich gegeben. Sollte der Fall jedoch auftreten, dass unterschieden werden muss, bietet Protractor diese Möglichkeit.

Zusätzlich können diese Bedingungen noch mit den Logischen Operatoren NOT, AND und OR in Form von Funktionen verknüpft werden. Dies ist ebenfalls in Abbildung 6: Protractor - Ausschnitt 1 - Logische Operatoren unter Alternative 1 zu sehen. Für die Leserlichkeit wird hier jedoch Alternative 2 empfohlen.

Abbildung 6: Protractor - Ausschnitt 1 - Logische Operatoren

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1 – Logische Operatoren für Bedingungen *it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

Code : WebdriverIO Test-Aufbau*it*(**'Should wait for Data'**, () => {  
 page.navigateTo();  
 page.**lazyModuleButton**.click();

*// Alternative 1*  
 browser.wait(conditions.and(  
 conditions.visibilityOf(page.**title**),  
 conditions.invisibilityOf(page.**spinner**),  
 conditions.visibilityOf(page.**list**))  
 , 5000);

*// Alternative 2* browser.wait(conditions.visibilityOf(page.**title**), 1000);  
 browser.wait(conditions.invisibilityOf(page.**spinner**), 5000);  
 browser.wait(conditions.visibilityOf(page.**list**), 1000);

*expect*(page.getRowText(0))

.toBe(**'Item Nr 1ein total nutzloses Objekt42'**);  
 *expect*(page.getRowText(1))

.toBe(**'Item Nr 2ein total nutzloses Objekt3'**);  
 *expect*(page.getRowText(2))

.toBe(**'Item Nr 3ein total nutzloses Teil17'**);  
});

### Ergebnisse der Implementierung

Bei der Verwendung mit Angular darauf hinzuweisen, dass Protractor komplett mitgeliefert wird. Wodurch sich alle Konfigurationen und Einstellungen als sehr einfach und schnell erweisen. Dennoch ist es möglich eigene Strukturen zu konfigurieren. Leider stößt man bei neuen Angular Project, welche mit der Angular-CLI in Version 7.1.4 erzeugt wurde, auf ein Problem. Durch die Angular-CLI kann Protactor nicht gestartet werden, wenn die Bibliothek „@angular-devkit/build-angular" nicht aktualisiert wird. Sollte Protractor nicht durch die Angular-CLI gestartet werden, fällt dieser Umstand nicht auf.

Wie erwähnt, kann in einem Angular-Projekt Protractor über die Angular-CLI gestartet werden, was sich für Build-Prozesse sehr gut eignet, oder auch direkt, was sich gerade während der Entwicklung durch das schnelle starten von Testen ohne einem Build-Prozess von Vorteil sei kann.

Die Konfigurations-Datei von Protractor beträgt ca. 30-Zeilen und ist auf den ersten Blick übersichtlich und leicht verständlich. In ihr werden in einfachen Fällen neben Testdateien, Timeouts, Test-Framework, Browser und die zu testende URL nichts weiter beschrieben. Was alles durch die Konfigurations-Datei eingestellt werden kann, findet sich in der Dokumentation sehr gut beschrieben.

Der Testaufbau ist abhängig vom Test-Framework. In Abbildung 7: Jasmine Testaufbau ist der Aufbau zu sehen wie er durch Jasmine vorgeschreiben ist. So wird der Test mit der Beschreibung der Test-Datei begonnen gefolgt von der Variablen Deklarationen. Es gibt in jedem Test eine *beforeEach*-Funktion welche wie der Name sagt vor jedem Test ausgeführt wird, und dazu genutzt wird neue Objekte zu erzeugen und unabhängige Tests zu ermöglichen. Nach jedem Test wird eine *afterEach*-Funktion ausgefüht, welche die Tests bereinigen kann. Die Tests selbst werden sprechend mit *it* gefolgt von der „Should ..“ und der Test-Beschreibung eingeleitet.

*describe*(**'Suite Description'**, () => {  
 **let** page: SomePageObject;  
 **const** conditions: ProtractorExpectedConditions = protractor.**ExpectedConditions**;  
  
 *beforeEach*(() => {  
 page = **new** SomePageObject();  
 });  
  
 *afterEach*(() => {  
 ...  
 });  
  
 *it*(**'Should ...'**, () => {  
 ...  
 });  
}

*describe*(**'Suite Description'**, () => {  
 **let** page: SomePageObject;  
 **const** conditions: ProtractorExpectedConditions = protractor.**ExpectedConditions**;  
  
 *beforeEach*(() => {  
 page = **new** SomePageObject();  
 });  
  
 *afterEach*(() => {  
 ...  
 });  
  
 *it*(**'Should ...'**, () => {  
 ...  
 });  
}

Abbildung 7: Jasmine Testaufbau

Betrachtet man die ausprogrammierten Anwendungsfälle, so stellt man wie in Abbildung 8: Protractor - Ausschnitt 2 - Anwendungsfall 1 fest, dass der Code durch das Page-Object-Patterns fast schon für nicht Informatiker leserlich ist. Die Kompletten Anwendungsfälle sind im Anhang oder unter zu finden.

Abbildung : Protractor - Ausschnitt 2 - Anwendungsfall 1

*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Code : Protractor – Ausschnitt aus Anwendungsfall 1*it*(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.sendKeys(**'Ampletzer'**);  
 page.**formFirstName**.sendKeys(**'Dominik'**);  
  
 page.**formSex**.click();  
 **const** sexOption = element(by.id(**'sexOption1'**));  
 browser.wait(conditions.elementToBeClickable(sexOption), 1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
  
 browser.wait(conditions.visibilityOf(page.**sentName**), 5000);  
 *expect*(page.**sentName**.getText()).toBe(**'Ampletzer'**);  
 *expect*(page.**sentFirstName**.getText()).toBe(**'Dominik'**);  
 *expect*(page.**sentSex**.getText()).toBe(**'Männlich'**);  
});

Protracotr hatte mit keinen der 6 Anwendungsfälle schiwerigkeiten. So war weder das asynchrone laden von Modulen und Daten ein Problem, noch andere Anwendungsflälle. Zwei hürden müssen hier jedoch erwähnt werden. So war beim Anwendungsfall 4 welche einer neuen Tab erzeugt als stolperstein die Tatsache, dass der Kontext des Browsers gewechselt werden muss und bei Anwenungsfall 6 (iFrame) die schiwerigkeit, dass ebenfalls der Kontext gewechselt werden muss kompiniert damit, dass wir die Angular-Anwendung verlassen, was Protractor mitgeteilt werden muss. Sihe .

### Weitere Ergebnisse

Neben Anwendungsfall 1 & 2, welche Protractor gewälltigen konnte, ist noch zu erwähnen, dass die Anderen Anwendungsfälle ebenfalls gelöst werden konnten.

So bestand die schwierigkeit bei Anwendungsfall 3 darin, dass mehrere Elemente die gleiche ID!! hatten, was sehr ungewöhnlich ist, Portractor konnte dennoch damit umgehen. Näheres dazu in gitXXX.

Anwendungsfall 4 zielt darauf ab, dass es für Frameworks schwer sein könnte, mit verschiedenen Tabs und Fenstern zu interagieren. Dies kann bei Protractor nicht gesagt wrden. Es fällt sehr leicht, wenn man weiß, dass man den Tab/Window-Kontext mit *browser.switchTo* wechseln muss, was sich sowohl durch die Dokumentation als auch durch kurze Internet-Recherche herausfinden lässt.

Anwendungsfall 5 – Datei-Upload kann nur in sofern getestet werden, dass gesagt werden muss, dass Protractor keinen Zugriff auf das Betreibssystem eigene Fenster, welches die Dateiauswahl ermöglicht. Was hingegen funktioniert ist den Pfad zur Datei an ein File-Input-Feld zu übergeben. Dieses bekommt darauf die Datei und kann auch mit dieser agieren.

Anwendungsfall 6 – iFrame ist ähnlich Anwendungfall 4 so muss hier ebenfalls der Kontext auf den iFrame gewechselt werden. Auf diesen kann ebenfalls wie gewohnt interagiert werden. Dies bedeutet alle Selektoren und alle Bedingungen sind auf den iFrame anwendbar. Hierfür muss, da es sich beim iFrame nicht um eine Angular-Anwendung handelt im Test browser.waitForAngularEnabled(false) aufgerufen werden. Dies liegt daran, dass Protractor auf Angular-Events warten kann, welche nicht durch den iFrame abgedeckt werden.

Abschließend sollte noch erwähnt werdne, dass die Implementierung der Testfälle mit Änderung der Standardkonfigurationen ca. 6 Stunden in Anspruch genommen hat.

## WebdriverIO Details

WebdriverIO setzt seit v5 stark auf das Page ObjectPattern. Es beschreibt ebenfalls sehr gut, welche vorteile es bietet, und wie es verwendet wird. Natürlich ist es wie alle Pattern nur eine Option. Kein Muss.

### Browserunterstützung

Webdriver unterstützt dir Browser Firefox, Chrome, Opera, Safari, Microsoft Internet Explorer und Edge sowie Mobile Browser durch Appium. In dieser Hinsicht unterscheidet sich WebdriverIO nicht von Protractor. Allerdings findet man in der WebdriverIO Dokumentation nur indirekt welche Browser unterstütz werden. So gibt es keinen Bereich „Unterstützte Browser“. Die Browserunterstützung muss aus den Optionen der Konfiguration und diversen Blogbeiträgen erarbeitet werden.

### Page-Object-Pattern

WebdriverIO setzt seit v5 ebenfalls auf das Page-Object-Pattern. Die Integration ist ebenfalls sehr einfach. WebdriverIO in den Beispielen des Pattern den Zugriff auf die Elementfinder mittels der AUF XY zu sehenden GET-Notation an, was jedoch als Äquivalent zu der Attributsinitailisierung im falle von Protractor zu betrachtetn ist. Desweitern können komplexere Aufrufe und Funktionen wieder im Page-Objekt entwickelt werden und bei bedarf genutzt werden

**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

Code : WebdriverIO Test-Aufbau

Code : WebdriverIO Test-Aufbau**const AppPage** = *require*(**'../app.po'**);  
  
**class** SomePage **extends AppPage** {  
 **get** iFrameButton() {  
 **return $**(**"#iFrame"**);  
 }  
  
 **get** intranetButton() {  
 **return $**(**'\*=ntrane'**);  
 }  
  
 **get** iFrame() {  
 **return $**(**'#thRosenheim'**);  
 }  
  
 **get** subTitleFromIntranet() {  
 **return $**(**'//\*[@id="c55958"]/div/h2'**);  
 }

getSecondTitle() {  
 **return $$**(**'#dialogTitle'**)[1];  
 }

}  
  
module.**exports** = UseCase6Po;

### Zugriffsmöglichkeiten

Zugriffsmöglichkeiten welche durch WebdriverIO angeboten werden sind in einer Kurzschreibweise gehalten, was die Tests sehr kompakt macht, allerdings zu kosten der Leserlichkeit geht. So werden Elementfinders durch die $-Funktion erzeugt. Um zu bestimmen, welcher Elementfinder erzeugt werden soll, wird ebenfalls eine Kurzschreibweise verwendet. Dafür stehen verschiedene Sonderzeichen für die verschiedenen Typen. Diese sind:

* CSS-Query = $(„h2“)
* Link-Text = $(„=someString“)
* Partial-Link-Text = $(„\*=someString“)
* CSS-Klasse = $(„.somestring“)
* Id = $(„#someString“)
* Tag = $(„<my-tag/>“)
* xPath = $(„x-PATH as String“)

Diese Elementfinder können ebenfalls kombiniert werden um Elemente zu finden, welche mehrere Kriterien erfüllen müssen. Z.B.: CSS-Query mit Text = $(„h1=SomeText“)

Zusätzlich gibt es noch die Möglichkeit für Mobile-Anwendungen spezielle Elementfinder zu verwendet, sowie die Möglichkeit mittel JavaScript-Funktionen Elemente zu finden.

Erschwerend kommt noch die $$-Funktion hinzu. Diese verhält sich wie die $-Funktion nur mit dem Unterschied, dass mehrere Elemente gefunden werden können.

### Bedingungen –warten auf Ereignisse

Wie nicht anders zu erwarten, benötigt WebdriverIO ebenfalls Bedingungen bzw. muss auf Ereignisse warten. Diese Bedingungen sind gleich in die Elemente bzw in die Elementfinder integriert, was es sehr einfach macht, diese zu verwenden. So sieht man in CODE XC, dass die *waitForDisplayed*-Funktion direkt aufgerufen werden kann. Webdriver bietet neben der naiven Variante der Browser.wait mit einer Zeitangabe vier weitere möglichkeiten:

* waitForDisplayed
* waitForEnabled
* waitForExist
* waitUntil

Die Kurzschreibweise von WEbdriverIO setzt sich auch hier durch, so wurde z.B. auf eine waitForInvisible verzichtet. Statt dessen wurde die waitForDisplayed mit einem Reverse-Parameter versehen, welche für die Unsichtbarkeit des Elements verwendet wird.

Diese Möglichkeiten sind bis auf waitUntil absolut klar, was diese machen. WaitUntil nimmt als Parameter eine Benutzerdefinierte Funktion und wartet bis die Bedingungen dieser erfüllt sind.

### Ergebnisse der Implementierung

Einführend ist zu erwähnen, dass das Aufsetzten von WebdriverIO sich mühsam gestaltet, selbst wenn man sich an das bereitgestellte Tutorial hält. In diesem wird der Einfachheit halber auf Dienste, welche durch WebdriverIO mit installiert werden können verzichtet. Leider ist gerade der Silenium-Dienst unverzichtbar um die Tests ohne größeren Aufwand auf einen Geräte durchführen zu können.

Die Konfigurationsdatei von WebdriverIO ist sehr übersichtlich und ebenfalls leicht verständlich. Was einen Anfangs leicht überfordern kann, allerdings nach kurzem sehr zur hilfe gereicht ist die Code-Dokumentation welche sich in der Konfigdatei befindet. So ist die Datei ohne Kommentare nur ca. 30 Zeilen lang. Mit Kommentaren jedoch ca. 250 was es einfach macht, herauszufinden, welche einstellungsmöglichkeiten gemacht werden können.

Der Testaufabau wird, wie bei Protractor durch das Test-Framework bestimmt und besteht wie in Code 3: Testaufbau WebdriverIO zu sehen ebenfalls aus der Beschreibung der Test-Datei gefolgt von der Deklaration von Variablen, einer *beforeEach*- und einer *afterEach*-Funktion, welche vor bzw. nach den eigentlichen tests durchgeführt werden. Tests werden mit dem sprechen *it* und Beschreibung erzeugt. Worin hier unterschiede zu Protractor zu finden sind, ist der Programmiersprache geschuldet. Protractor in Angular wird i.d.R. mittels TypeScript entwickelt, welche Typdefinitionen beinhaltet. Die WebdriverIO Test in hier mittels JavaScript geschreiben, welche ohne Typdefinitionen auskommt.

**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Code : Anwendungsfall 1 - WebdriverIO**const** assert = *require*(**'assert'**);  
**const** SomePageObject = *require*(**'./some-page-object.po.js'**);  
  
describe(**'Suite Description'**, () => {  
 **let** page;  
 beforeEach(() => {  
 page = **new** SomePageObject();  
 });  
 afterEach(() => {...});  
  
 it(**'Should ...'**, () => {  
 });  
});

Das Betrachtet von CODE XY zeigt ebenfalls wie klar leserlich der Code mittels des Page-Object-Patterns wird. Die durchgeführten Anwendungsfälle können im Anhang näher betrachtet werden oder unter LINK.

Fünf der sechs Anwendungsfälle konnte mit kleineren Hürden bewälltigt werden. Ebenfalls ist bei Anwendungsfall 6 (iFrame) zu beachtetn, dass der Kontext des Fensters auf dem Frame gewechselt werden musss.

it(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Code : Protractor - Page-Object-Patternit(**'Should insert Data into Form'**, () => {  
 page.navigateTo();  
 page.**formButton**.click();  
  
 page.**formName**.setValue(**'Ampletzer'**);  
 page.**formFirstName**.setValue(**'Dominik'**);  
  
 page.**formSex**.click();  
 **let** sexOption = **$**(**'#sexOption1'**);  
 sexOption.waitForDisplayed(1000);  
 sexOption.click();  
  
 page.**saveButton**.click();  
 page.**sentName**.waitForDisplayed(5000);  
 assert.*equal*(page.**sentName**.getText(), **'Ampletzer'**);  
 assert.*equal*(page.**sentFirstName**.getText(), **'Dominik'**);  
 assert.*equal*(page.**sentSex**.getText(), **'Männlich'**);  
});

Anwendungsfall 4 wird hier explizit erwähnt, da dieser große Probleme mit sich brahteDie Restlichen Anwendungsfälle gestalteten sich aus machbar mit kleineren Hürden. Mit ausnahme des Anwendungfall 4, welcher Folgende Schritte beinhaltet:

* einen neuen Tab öffnen
* in diesem ein Formular aussfüllen und absenden
* den Tab schließen
* zurückkehren auf den Ursprünglichen Tab

Hierfür gibt es in der Dokumentation möglichkeiten mit switchWindow, welche immer zum abbruch der Tests geführt hat! Die switchToWindow-Funktion, welche in diversen Foren und Beiträgen verwendet wurde, welche den Kontextwechsel auf der Tab ermöglichte für nicht zum ewünschten Ziel, da spätestens beim Tab schließen der Test abbricht. Dieser Anwendugnfall wurde nach ca 1,5 Stündigen der Implmenetierung abgebrochen, da dies bereits eine gewisse Aussagekraft besitzt.

Als persönliche Anmerkung muss noch erwähnt werden, dass das Konfigurieren und nur das adaptieren (diese wurden bereits durch die Protractor-Tests druchdacht und entwickelt, es mussten nur die Selektoren und bedingungen angepasst werden) hat sehr viel zeit in Anspruch genommen. So wurden ca. 8-9 Stunden benötigt, und tortzdem konnte Anwendungsfall 4 nicht gelöst werden.

## Direkter Vergleich

Als Vergleichskriterien für den direkten Vergleich werden

* Benötigte Codemenge (Lines-of-Code = LoC) um die Komplexität des Codes darzustellen
* Benötigte Zeit für die Implementierung um die Komplexität und „Schwierigkeit“ der Implementierung auszudrücken
* Anzahl der Gelösten Anwendungsfälle: ebenfalls ein Kriterium für „Schwierikeit“ der Implementierung und um den Funktionsumfang darzustellen

|  |  |  |
| --- | --- | --- |
|  | Protractor | Webdriverio |
| Loc | 458 | 494 |
| Benötigte Zeit | 5-6h | 8-9h |
| Gelöste Anwendungsfälle | 6 | 5 |
| Anmerkungen | Für Nicht-Angular-Anwendung muss waitForAngular deaktiviert werden | Wechseln des Fensters/Tabs funktioniert in Testimplementierung nicht mit akzeptablen aufwand |

Der LoC unterschied von ca. 10% komm in erster Linie von der unterschiedlichen Page-Object-Pattern Implementierung. Betrachtet man die Tests selbst, ställt man kaum einen Unterschied in der Länge fest.

Der gewaltige Zeitunterschied ist zu großen Teilen der Komplexeren und kürzeren Elementfinder Funktionen von WebdriverIO und der Konfiguration von WebdriverIO zu verschulden.

# Fazit

Aus dieser Arbeit ergibt sich, dass Nightwatch das wenigsten zu empfehlende Werkzeug ist. Dies liegt daran, dass Nightwacht nur einen Haupt-Entwickler besitzt. Vermutlich auch daraus fehlt es Nightwatch an einem gewissen Reifegrad, welcher von Protractor und WebdriverIO abgebildet wird. Als letzter Grund, der gegen Nightwatch spricht, ist der Bekanntheitsgrad. Dieser ist nicht sonderlich verbreitet, was zur Folge hat, dass es schwierig werden kann gute Unterstützung zu finden.

WebdriverIO und Protractor setzten beide auf das Page-Object-Pattern als Option und ermöglichen dadurch ein einfaches Testen. Beide Werkzeuge ermöglichen das Testen mittels JavaScript als auch mit TypeScript. Außerdem nutzen beide Werkzeuge „Bedingungen“ um auf zustände des Browsers zu warten. Hierdurch kann asynchronität und dynamsichesladen sehr gut abgebildet werden.

WebdriverIO ist ein sehr gutes Werkzeug, welches kleinere Schwächen aufweist. So gabt es in der Testimplementierung das Problem, dass Test welche einen Tabwechsel einschließen nicht lauffähig gemacht werden konnten, obwohl dies bei WebdriverIOs Testbeispiel funktioniert. WebdriverIO bietet nicht ganz so viele Möglichkeiten Elemente zu finden wie Protractor, dafür aber innerhalb einer Elementsuche die Verschiedenen Möglichkeiten zu kombinieren. Was nicht unbedingt als Vorteil bezeichnet werden kann ist die durchgehen spürbare Kurzschreibweise, welche WEbdriverIO immer wieder verwendet. So ist diese natürlich kurz und Kompakt, geht aber auf kosten der Lesbarkeit, da ganz explizites Wissen benötigt wird, welche Funktionen und Elementfinder benutzt werden sollen, oder welche Bedingung negiert werden muss um eine andere Bedingung zu erhalten.

Sowohl für Angular-Anwendungen im Speziellen als auch im Allgemeinen kann Protractor empfohlen werden. Dieses Werkzeug bietet einen Hohen Reifegrad, starke Unsterütztung durhc die Community und Google und Konnte im Test sowohl durch die geringe Einarbeitung als auch durch das lösen aller Anwendungfälle überzeugen. Als großer Vorteil wird die große anzahl an Bedingungen und Elementfindern betrachtet. Diese ermöglichen es auf unterschiedlichste weisen Elemente zu erhalten und die Bedingungne sehr granular zu bestimmen.

# Literaturverzeichnis

Gitter. (2019). *Gitter - Homepage.* www.gitter.im - gefunden 22.04.2019.

Google. (2019). *Google Gruppen - Homepage.* https://groups.google.com - gefunden 20.04.2019.

Nightwatch.js. (2019). *Nightwatch.js - Homepage.* www.nightwatchjs.org - gefunden 21.04.2019.

Protractor. (2019). *Protractor - Homepage.* www.protractortest.org - gefunden 20.04.2019.

StackOverflow. (2019). *Stack Overflow - Homepage.* https://stackoverflow.com - gefunden 20.04.2019.

Webdriver.IO. (2019). *WebdriverIO - Homepage.* www.webdriver.io - gefunden 20.04.2019.

1. https://www.protractortest.org [↑](#footnote-ref-1)
2. AngularJS und Angular unterscheiden sich stark voneinander und sind somit nicht das gleiche Framework [↑](#footnote-ref-2)
3. Stand 20.04.19 [↑](#footnote-ref-3)
4. Stand 20.04.19 [↑](#footnote-ref-4)
5. Stand 20.04.19 [↑](#footnote-ref-5)