

Fakultät für Informatik

Studiengang Wirtschaftsinformatik

Vergleich von JavaScript-UI-Test Werkzeugen für Single-Page-Applikationen am Bespiel einer Angular-Applikation

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 UI/E2E-Test-Werkzeuge welche auf JavaScript basieren und für den Einsatz mit Web-Applikationen entwickelt wurden. Bei diesen Werkzeugen handelt es sich um: Protractor, Nightwatch.js und Webdriver.IO. Diese wurdem 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-Applikation an verschiedenen Anwendungsfällen verprobt um über Integrationsfähigkeit, Aufwand und Leistungs bzw. Fähigkeitsumfang der Werkzeuge eine Aussage geben zu können. Aus dem Kurzcheck und der Verprobung soll ersichtlich werden, welches Werkzeug im Speziallfall von Angular-Applikationen aber auch im Allgmeinen das Bessere ist und demnach verwendet werden soll.

Es wurde ersichtlich, dass Protractor im Speziallfall von Angular-Applikationen im Grunde Alternativlos ist. Desweitern kann man im Allgemeinen die Ausage getroffen werden, das Protractor dem anderen Werkzeugen vorraus ist, und demnach eine klare Empfehlung darstellt.

Schlagworte

E2E-Werkzeuge, UI-Testing, Angular, Protractor, Nightwatch.js, WebDriver.IO

**Inhaltsverzeichnis**

[Abbildungsverzeichnis iii](#_Toc7421330)

[Tabellenverzeichnis iv](#_Toc7421331)

[Glossar / Abkürzungsverzeichnis vi](#_Toc7421332)

[1 Einleitung 1](#_Toc7421333)

[1.1 Problemstellung 1](#_Toc7421334)

[1.2 Vorgehen & Ziel der Arbeit 1](#_Toc7421335)

[1.3 Rahmenbedingungen 1](#_Toc7421336)

[2 E2E-Test-Werkzeuge 1](#_Toc7421337)

[2.1 Kurzüberprüfung / Schnelltest 3](#_Toc7421338)

[2.2 Protractor 3](#_Toc7421339)

[2.2.1 Lizenz: 3](#_Toc7421340)

[2.2.2 Reifegrad 3](#_Toc7421341)

[2.2.3 Stabile-Version-Veröffentlicht 4](#_Toc7421342)

[2.2.4 Support 4](#_Toc7421343)

[2.2.5 Dokumentation 4](#_Toc7421344)

[2.2.6 Bekannte Mängel/Fehler/Sicherheitslöcher 4](#_Toc7421345)

[2.2.7 Qualität des Projekts 4](#_Toc7421346)

[2.2.8 Aktivitäten 4](#_Toc7421347)

[2.2.9 Bekanntheitsgrad 4](#_Toc7421348)

[2.2.10 Unterstützer des Projekts 5](#_Toc7421349)

[2.3 Nightwatch.js 5](#_Toc7421350)

[2.3.1 Lizenz: 5](#_Toc7421351)

[2.3.2 Reifegrad 5](#_Toc7421352)

[2.3.3 Stabile-Version-Veröffentlicht 5](#_Toc7421353)

[2.3.4 Support 5](#_Toc7421354)

[2.3.5 Dokumentation 5](#_Toc7421355)

[2.3.6 Bekannte Mängel/Fehler/Sicherheitslöcher 5](#_Toc7421356)

[2.3.7 Qualität des Projekts 5](#_Toc7421357)

[2.3.8 Aktivitäten 6](#_Toc7421358)

[2.3.9 Bekanntheitsgrad 6](#_Toc7421359)

[2.3.10 Unterstützer des Projekts 6](#_Toc7421360)

[2.4 WebDriver.IO 6](#_Toc7421361)

[2.4.1 Lizenz: 6](#_Toc7421362)

[2.4.2 Reifegrad 6](#_Toc7421363)

[2.4.3 Stabile-Version-Veröffentlicht 6](#_Toc7421364)

[2.4.4 Support 6](#_Toc7421365)

[2.4.5 Dokumentation 6](#_Toc7421366)

[2.4.6 Bekannte Mängel/Fehler/Sicherheitslöcher 7](#_Toc7421367)

[2.4.7 Qualität des Projekts 7](#_Toc7421368)

[2.4.8 Aktivitäten 7](#_Toc7421369)

[2.4.9 Bekanntheitsgrad 7](#_Toc7421370)

[2.4.10 Unterstützer des Projekts 7](#_Toc7421371)

[2.5 Zusammenfassung/ Kurzübersicht der Ergebnisse 7](#_Toc7421372)

[2.6 Vergleich von Protractor und Webdirver.IO 8](#_Toc7421373)

[2.6.1 Implementierungsgrundlage und Anwendungsfällen 9](#_Toc7421374)

[2.6.2 Ergebnisse des Vergleichs 9](#_Toc7421375)

[2.6.3 Auf Vergleich der Implementierungen 9](#_Toc7421376)

[3 Fazit 9](#_Toc7421377)

[Literaturverzeichnis 11](#_Toc7421378)

# Abbildungsverzeichnis

[Abbildung 1: Vereinfachter Ablauf 2](#_Toc8729479)

[Abbildung 2: Ablauf mit E2E-Test-Werkzeug 2](#_Toc8729480)

[Abbildung 3: Ablauf mit UI-Test-Werkzeug 2](#_Toc8729481)

[Abbildung 4: Trendanalyse 8](#_Toc8729482)

# Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Code-Verzeichnis

[Code 1: Protractor Test-Aufbau 11](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\Seminararbeit_SQS_Ampletzer.docx#_Toc8729533)

[Code 2: Protractor – Ausschnitt aus Anwendungsfall 1 11](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\Seminararbeit_SQS_Ampletzer.docx#_Toc8729534)

[Code 3: WebdriverIO Test-Aufbau 14](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\Seminararbeit_SQS_Ampletzer.docx#_Toc8729535)

[Code 4: Anwendungsfall 1 - WebdriverIO 15](file:///C:\Users\Privat\Studium\Master%202\Software%20Qualitätssicherung\Seminararbeit_SQS_Ampletzer.docx#_Toc8729536)

# 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 |
|  |  |
|  |  |
|  |  |
| 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 Benutzeroberflächen-Test-Werkzeugen (UI-Test) die ebenfalls dazu genutzt werden können End-zu-End-Tests (E2E-Tests) durchzuführen. Diese Werkzeuge sind nur 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/UI-Tests ähnlich ist, stellt sich natürlich die Frage, welches Werkzeug eignet sich am besten. Da sich viele Web-Anwendungen als Single-Page-Applikation entwickelt werden. Geht es speziell um die Frage, welches Werkzeug ist für Single-Page-Applikationen am besten geeignet.

## Vorgehen & Ziel der Arbeit

Diese Arbeit gibt einen kurzen Überblick über Protractor, Nightwatch und WebDriver.IO. Mittels einer Kurzüberprüfung werden die Werkzeuge sowohl analysiert als auch das schlechterste Werkzeug verworfen um die besseren 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-Applikationen im Speziellen und Single-Page-Applikationen 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. Diese Aussage wird ersichtlich, wenn man die die vereinfachten Abläufe der Interaktion mit dem System betrachtet. In Abbildung 1: Vereinfachter Ablauf ist zu sehen, dass der Benutzer mit der Benutzeroberfläche (UI) welche natürlich ein Teil der Web-Anwendung ist interagiert. Die Web-Anwendung stellt Anfragen beim Server, welche Antwortet. Diese Antworten werden in welcher Form auch immer auf dem UI angezeigt, was der Benutzer wiederum wahrnimmt.

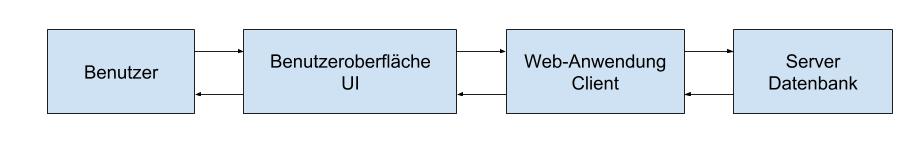


Abbildung 1: Vereinfachter Ablauf

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

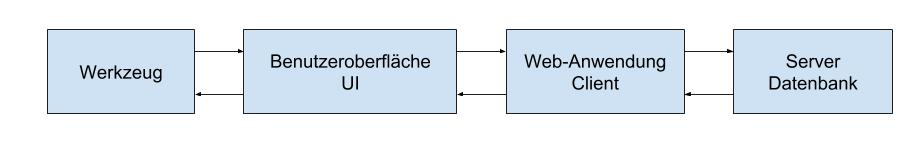


Abbildung 2: Ablauf mit E2E-Test-Werkzeug

Ein UI-Test-Werkzeug ist, vom Prinzip her dasselbe wie ein E2E-Test-Werkzeug, wenn es sich nur auf die Benutzeroberfläche beschränke identisch. Da durch das Interagieren mit der Benutzeroberfläche durch das Werkzeug wird eine Änderung erwartet. Was geprüft werden kann. Allerdings soll dies beim UI-Test isoliert vom Server geschehen, was wie in Abbildung 3: Ablauf mit UI-Test-Werkzeug zu sehen dadurch erzielt wird, dass ein Unechter Server (Mock/Stub) verwendet wird, welcher fest vorgegebene Ergebnisse zurückgibt. Dadurch kann die Oberfläche unabhängig vom Server getestet werden.

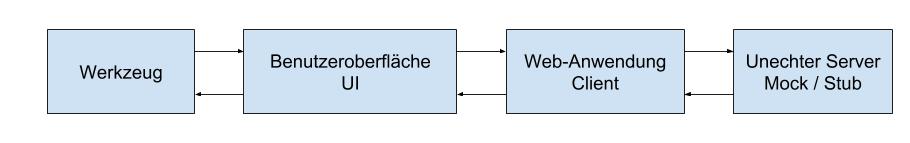


Abbildung 3: Ablauf mit UI-Test-Werkzeug

Somit ist der Unterschied des UI-Test-Werkzeugs und des E2E-Test-Werkzeugs der, dass der angesprochene Server ausgetauscht wird. Was nichts mit dem Test-Werkzeug zu tun hat, sondern mit der Modularität und Unabhängigkeit des Systems. Daraus resultiert, dass UI-Test-Werkzeuge und E2E-Test-Werkzeuge für diese Arbeit als gleichbedeutet betrachtet werden könne. Aus Gründen der Klarheit, wird möglichst nur der Begriff E2E-Test-Werkzeug verwendet.

## Kurzüberprüfung / Schnelltest

Für die Kurzüberprüfung wurde ein Katalog aus Folgenden Kriterien zusammengestellt:

* Lizenz: Art der Lizenz, wobei die Einstufung MIT/Apache – GPL – Propritär ist
* Reifegrad: Existenzzeit und stabile Veröffentlichung, Versionsnummer
* Support: Öffentliche Unterstützung und Porfessionelle Unterstützung
* Dokumentation: Qualität der Dokumentation, Tutorials, Starthilfen, Alter der Dokumentation
* Bekannte Mängel/Fehler/Sicherheitslöcher
* Qualität des Projekts: Testabdeckung / Bugs durch statische CodeAnalyse
* 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 den Projekt stehen => Rückschluss auf langlebigkeit

Diese Kriterien werden als Schwerwiegend betrachtet werden 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 im Produktiven Einsatz zu gebrauchen.

## 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. Auch ist der meist in Angular verwendete Test-Runner Karma kann durch andere Werkzeuge wie Mocha oder Cucumber ersetzt werden.

Neben JavaScript können Protractor Test mit TypeScript geschrieben werden. Was besonders in Angular-Applikationen, welche in TypeScript geschrieben sind, den Vorteil mit sich bringt, sich nicht an leicht 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. Desweiteren haben seit 2013 ca 100 Veröffentlichungen stattgefunden

### Stabile-Version-Veröffentlicht

Ja

### Support

Community-Support[[3]](#footnote-3) auf StackOverflow (ca. 10.000 Fragen), Gitter (ca. 1.250 Nutzer) und Angular-Discussion-Group (ca. 19.000 Themen).   
Issue-Liste enthält nur 13 Bugs

### Dokumentation

Auf der Protractor Seite ist neben einem Tutorial und einem Schnellstart Beschreibungen mit Beispielen für die Verschiedenen Bereiche von Protractor vorhanden. Dokumnetation ist leicht veraltet jedoch sehr ausführlich und besonders in den Hilfen der Bereiche und Api-Referenz sehr übersichtlich und hilfreich. Da diese sich auf Version 5.4.1 bezieht. Mittlerweile ist man jedoch bei Version 6, was zwei Veröffentlichungen nach V 5.4.1 ist.

### Bekannte Mängel/Fehler/Sicherheitslöcher

Sicherheitslöcher sind nicht bekannt und eigentlich auc nicht relevant. Dies ist allerdings auch nicht verwunderlich, da die Werkzeuge nicht veröffentlich werden, sondern für das Testen Local oder in der CI pipline dienen.

### Qualität des Projekts

Innere Betrachtung Fehlt in allesn 3 wie sind diese interne STRUKTURIERT!!!

Durch statische Code-Analyse wurden 6 Bugs aufgeworfen, welche sich auf:

* Auf frische Fehler wie 3x const statt let benutzt
* Einen nicht gesetzten DocType im HTML
* Einen CSS fehler
* Und einen falschen Rerturn-Wert handelt

Unschön hingegen sind 102 stellen im Code wleche sich allerdings meist auf Shadowing und fehlendes Clean-Up beruht  
Testabdeckung beträgt: XX%

Mit Ca 14K Zeilen Code eher klein. Diese Enthalten ebenfalls die WebSite und den BesipielsCode

### Aktivitäten

Der letzte Release war im Q1 2019. Seit dem ist die Commit Rate relativ gering. Was aber auch am Reifegrad des Frameworks liegt. Das Projekt habt ebenfalls über 250 Contributors.

### Bekanntheitsgrad

Der bekanntheitsgrad hängt stark an Angular, da es standardmäßig mit Angular-Projekten mit installiert wird. Außerdem hat es 500 beobachtungen 8000 Starts und über 2000 Forks.

Ca 3,5K Follower auf Twitter

### Unterstützer des Projekts

Genau Wie Angular, wird protractor von Google mentained und hat augrund dessen bereits einen Starken partner.

## 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 auf eigene Projekte. Außerdem knnen mit Nightwatch.js Unit-Tests geschrieben werden.

### Lizenz:

MIT

### Reifegrad

Eher als mittel – gering anzusehen. Das Werkzeug ist ab 2014 in der Version 0.2.8 verfügbar. Seit dem geb es über 150 Veröffentlichungen. Diese Stammten allerdings immer von einer Person. V 1 wurde im April 2018 veröffentlich und befindet sich somit ca 1 Jahr auserhalb der Beta-Phase

### Stabile-Version-Veröffentlicht

Ja

### Support

Community-Support[[4]](#footnote-4) auf StackOverflow (ca. 1.000 Fragen), welche zum Großen teil nicht beantwrotet wurden,sind eher negativ zu werden. Ähnlich ist es mit der Google Grup. Besser sieht es da in der Google Gruppe aus. Diese umfasst ca 1500 Themen, welche weitgehendst zeitnah 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 neuest Stand. So entspricht diese zum Stand 22.04.19 bereits der Version 1.1.2 welche es aktuell noch als Pre-Realese gibt. Api-Referenzen sind ebenfalls sehr ausführlich und leicht verständlich

### Bekannte Mängel/Fehler/Sicherheitslöcher

Sicherheitslöcher sind nicht bekannt. Allerdins sind in der Issue-Liste nur 10 Bugs offen, welche wie es scheint in bearbeitung oder zumindest bekannt sind

### Qualität des Projekts

Durch statische Code-Analyse wurden 4 Bugs aufgeworfen, welche sich auf:

* 2 Unnötige strings beziehen
* 2 x auf die Überprüfung ein und des selben objects

Mit 33 Schlecht geschreiben Codestellen ebenfalls sehr sauber ist. Insbesondere wenn man die größe von 27K LoC bedenkt.   
Testabdeckung beträgt: 85%

### Aktivitäten

Die Release-Häufigkeit ist sehr hoch. So gab es im Q4 2018 und Q1 2019 zusammen 8 Veröffentlichungen. Demnach ist auch die Commut-Rate sehr hoch. Unschön ist hingegen, dass das Projekt zwar 74 Contributor hat. Allerdings nur einen, welcher ernsthaft Entwickelt.

### Bekanntheitsgrad

Der bekanntheitsgrad ist eher mittel. Allerdings hat es bereits über 9000 Stars auf github, sowei 282 beobachtungen und 883 Forks. Twitter 5 Folower ☹

### Unterstützer des Projekts

Ist im Grunde nur PineView Software mit einem Entwickler.

## WebDriver.IO

Weitere WebDriver API Implementierung, welche von Namen her sich sehr an dem Anlehnt, was sie tut. WebDriver.io kann ebenfalls mit TypeScript betrieben werden und unterstützt 3 große test-frameworks. Jasmine, Cucumber und Mocha.

### Lizenz:

MIT

### Reifegrad

Sehr hoch – gering anzusehen. 2013 wurde V1 Released seit dem ca 200 Releases. Mai 2012 bereits als Beta bzw. v0.06.6 zur verfügung. Sehr kleiner Releasezyklus. Teilweise mehrmals Monatlich!!! Aktuell V5.7.16

### Stabile-Version-Veröffentlicht

Ja

### Support

Community-Support[[5]](#footnote-5) auf StackOverflow (ca. 800 Fragen), welche zum Großen teil nicht beantwrotet wurden,sind eher negativ zu werden. Desweiteren gibt es einen Gitter Chat, welcher mit ca 4000 Usern neben dem, dass er gut besucht ist, auch aktiv und dadurch hilfreich ist.

### Dokumentation

Die Dokumentation macht einen sehr guten eindruck. Es finden sich Schnelleinstiege, Boilderplate-Projecte, Hilfestellungen wie das Werkzeug zu verwenden ist. Zusätzlich werden noch UOnlinecurse angeboten. Diese sind jedoch Kostenfplichgti. Die Dokumentation scheint aktuell zu sein. Leider findet man in keine Versionsnummer. Allerdings ist sie definitiv nicht veraltet, da Seit dem V5 Release im Dezember 2018 bereits Poilerplate-Projecte für v5 integriert wurden

### Bekannte Mängel/Fehler/Sicherheitslöcher

Sicherheitslöcher sind nicht bekannt. Allerdins sind in der Issue-Liste nur 8 Bugs offen, allerdings handelt es sich bei allen um bugs, welche nicht gravierend sind und alle in bearbeitung bzw. auf der suche nach einen contributor sind, um sie zu lösen

### Qualität des Projekts

Durch statische Code-Analyse wurden 3 Bugs aufgeworfen, 2 davon können zumindest die mgölichkeit eines ernsthaften bugs führen. So gibt es eien if abfrage, welche immer true wird und einen notnull check, welcher nicht durgeführt wurde.

Mit 61 Schlecht geschreiben Codestellen ebenfalls sehr sauber ist. Insbesondere wenn man die größe von 22K LoC bedenkt.   
Testabdeckung beträgt: 98%

### Aktivitäten

Die Release-Häufigkeit ist sehr sehr hoch. Mit mehreren Realeases teilweise innerhalb eines Monats!!. Demnach ist auch die Commut-Rate sehr hoch. Aktuell sind auch 4 Haupt Constributoren beteiligt.

### Bekanntheitsgrad

Der bekanntheitsgrad ist eher mittel. Allerdings hat es bereits über 4800 Stars auf github, sowei 215 beobachtungen und 1370 Forks.

Was das finden und suchen nach dem Projekt erschwert ist der Name. Der Begriff Webdriver ist durchaus verbreitet und bekannt. Dies führt dazu, dass explizit nach Webdriver.io gesucht werden muss um das Projekt zu finden.

### Unterstützer des Projekts

Hinter Webdriver.io steht die JS.Foundation und somit ist sie durch einen Starken partner vertreten.

## Zusammenfassung/ Kurzübersicht der Ergebnisse

|  |  |  |  |
| --- | --- | --- | --- |
| Kriterium | Protractor | Nightwatch.js | Webdriver.IO |
| Lizenz | MIT | MIT | MIT |
| Reifegrad | Hoch | Niedrig | Mittel – Hoch |
| Stabile-Version-Veröffentlicht | Ja | Ja | Ja |
| Support | Gut durch Community | Gemäßigt durch Community | Gemäßigt durch Community. Zusätzliche Tutorials gegen Gebühr |
| Dokumentation | Gut | Gut | Sehr Gut |
| Mängel | Keine Bekannt | Keine Bekannt | Keine Bekannt |
| Qaulität 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 |

Zur Bekanntheit kann zusätzlich Abbildung 4: Trendanalyse betrachtet werden. Dort ist zu sehen wie die drei Werkzeuge im Google-Trend zueinander abschneiden. Hier ist allerdings 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.

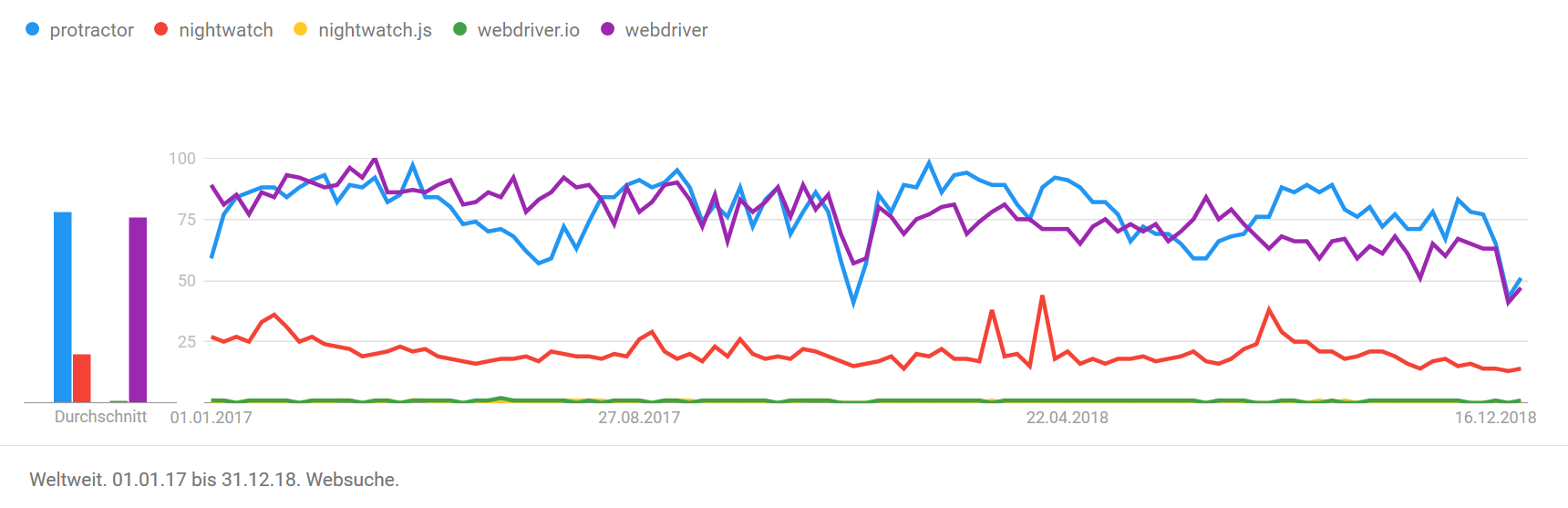


Abbildung 4: Trendanalyse

# Vergleich von Protractor und Webdirver.IO

Nach der Schnellüberprüfung wird für den detailierten Vergleich nur noch die zwei Werkzeuge Protractor und Webdriver.IO betrachtet. Dies liegt daran, dass sich Nightwatch neben eher Mäßiger Community unterstützund einen Geringen Reifegrad und nur einen Haupt-Contributor hat. Die Unterstützung von PineView Software im Gegensatz zu Google und JS.Foundation ist ebenfalls als eher klein/gering anzusehen, was für ein Produktivsystem unter Umständen den entschiedensten Ausschlag gibt.

Verglichen wird in diesem Kapitel der Funktionsumfang, aufwand für Integration in die Test-Anwendung (Hierbei wird berücksichtigt, dass die Angular-CLI Protractor mit Integriert). Einfachheit in Form von LoC für die Testfälle und auch deren Lesbarkeit. Sowie möglichkeiten der Zugriffe auf Elemente und insbesondere das Warten auf Bedingungen für die Elemente.

## Implementierungsgrundlage und Anwendungsfällen

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

Als Test-Framework wird Jasmine benutzt, welches in Standard Angular-CLI Projekten mitgeliefert wird.

Anwendungsfälle:

1. Navigation auf Applikation mit Nutzer Eingaben und Ergebnissen
2. Navigation auf Asynchron geladenes Modul mit Dynamischen laden von Inhalten. Während des Ladens werden Animationen gezeigt
3. Öffnen eines Dialogs mit dynamischen laden der Inhalte. Diese öffnen durch Betätigen einen weiteren Dialog
4. Neuen Browser-Tab öffnen in diesen Eingaben vornehmen und Tab-Wechseln
5. Daten-Upload (kein Drag & Drop)
6. Nutzung eines i-Frame

## Protractor Details

Bevor Protractor anprogrammiert wird, ist es nötig einen genaueren Bilck auf Protractor zu werfen.

Protractor ermöglicht die einfache Nutzung das Page-Object-Patterns (SIEHE CODEBEISPIEL). Bei diesem wird eine Objektrepräsentation der zu testenden Seite erzeuge. Dadurch kann wie gewohnt mit Objekten interagiert werden und die Logik der Seite wird hinter dem Page-Object versteckt.

### Browserunterstützung

Browserunterstützung findet bei den „großen Browsern“ statt. Somit wird Chrome, Firefox, Safari und IE out-of-the-box unterstützt. Für Microsoft Edge muss der Microsoft Web Driver installiert werden. Dadurch kann auch auf Edge zugegriffen 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. So können sowohl Chrome als auch Firefox im „Headless-Mode“ also ohne Oberfläche gestartet werden. Was das nutzen in einer CI/CD-Pipeline auf einfache weise ermöglicht.

Protractor bietet neben dem Klassischen Web-Browsern noch Mobilen Browser Support. Es unterstützt Appium für IOS/Safari und Android/Chrome und Selendroid

### 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 auf Attribute zugegriffen werden. Hierfür wird den Objecten wie in Abbildung XY zu sehen dem Attribut ein Element-Finder übergeben, welcher bei Zugriff das Element auf der Seite sucht bzw. findet. Desweiteren 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 dazu, wenn sich die Oberfläche ändert nicht den Test neu schreiben zu müssen. Statt dessen genügt es oft das Page-Object anzupassen.

**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 Abbilung XY zu sehen bedient sich das Page-Object unterschiedlicher Lokalisoren/Selektoren (Locators/Selectors) auch Elementfinder(Element-Finder) genannt.

Protractor bedient sich hierbei sehr vielen Elementfinder. Neben geerbten Elementfeinder wie: className, css, id, linkText, js,name, tagName, partialLinkText und xpath. Noch zusätzliche wie: binding, exactBinding, model, buttonText, partialButtonText, repeater, exactRepeater, cssConainigText, options und deepCss. Die viele dieser zusätzlichen Elementfindern sind Angular bezogen, und somit nicht für alle Frameworks nutzbar. Als sehr hilfreich für Legacy-Systeme oder Systeme, welche nicht verändert werden sollen sind sicherlich: buttonText, partialButtonText, cssConainigText, und deepCss. Mit anderen Worten Elementfinder, welche es nicht die ID oder den xPath benötigen.

In Protractor werden Elementfinder mit der *element*-Function erzeugt. Diese erhält einen Locator, welcher aus dem Basierenden Webdriver stammen kann oder mittels Protractors sehr leserlicher *by*-Klasse. Diese by-Klasse erzeugt mittels den wiederum leserlichen Funktionen den Locator. So wird:

* Id: element(by.id(„string“))
* CSS-Klasse: element(by.css(„string“)) oder element(by.class(„string“))
* Button-Text: element(by.partialButtonText(„string“))
* …

erzeugt. Was durchaus schreibaufwand bedeutet, aber der Leserlichkeit und der Übersicht dient. Was wiederum die Wartbarkeit und Austauschbarkeit unterstützt.

### Ergebnisse der Implementierung

Zuerst ist mit 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 bereits 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 die leicht Gewichtigkeit 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.

In der Regel sind Protractor-Tests wie in Code 1: Protractor Test-Aufbau zu sehen so aufgebaut, dass sie mit der Beschreibung der Test-Datei beginnen. Gefolgt von Variablen Deklarationen. Es gibt in jedem Protractor-Test eine *beforeEach*-Funktion welche wie der Name sagt vor jedem Test ausgeführt wird, und dazu genutzt wird neue Objekte zu erzeugen um unabhängige Tests zu ermöglichen. Nach jedem Test wird eine *afterEach*-Funktion ausgefüht, welche die Tests bereinigen kann.

Die Tests selbst werden mit Sprechend mit *it* und der Test-Beschreibung eingeleitet

Code 1: Protractor Test-Aufbau

Code 1: Protractor Test-Aufbau

*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 ...'**, () => {  
 ...  
 });  
}

#### Anwendungsfall 1 – Navigation und Formular

Code 2: Protractor - Anwendungsfall 1 ist mit Hilfe des Page-Object-Patterns fast schon für nicht Informatiker leserlich. Was neben der leserlichkeit erwähnung finden muss, ist, dass Protractor den Test synchron ausführt so wird beim browser.wait auf die erfüllung einer Bedingung ein bestimmte zeit gewartet. Sollte diese Bedigung nicht eintretten würde ein Fehler entstehen.

Bedinungen an sich sind bereits ebenfalls zu erwähnen. Protractor bietet hier viele verschiedene an, so unterscheidet es sogar zwischen sichtbar, clickbar und existiert. Diese Bedinungen könne zusätzlich noch mit Logischen Operatoren verbunden werden, wie im nächsten Anwendungsfall zu sehen.

*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 2: 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 2: Protractor – Ausschnitt aus Anwendungsfall 1

Code 2: Protractor – Ausschnitt aus Anwendungsfall 1

#### Anwendungsfall 2 – Asynchrones Nachladen von Modulen mit Dynamischen Inhalten

Der Anwendungfall 2 zeigt, dass Protractor selbsständig darauf wartet, dass Module Asynchron im Borwser nachgeladen werden. Außerdem kann man gut erkennen, wie über Bedingungen darauf gewartet werden kann, bis Daten geladen sind. In Abbildung 3 wird es dadurch deutlich, dass man eine Spinner-Animation hat, welche nach einer gewissen Zeit verschwinden muss und die Liste der Daten angezeigt werden muss. In Abbildung 3 sind zwei Möglichkeiten des Warten gezeigt. So sieht man drei Bedingungen, welche mit einem Logischen Operator verknüpft wurden. Was jedoch der Leserlichkeit förderlicher ist, idt jede Bedinungen einzeln in eine Zeile zu schreiben.

*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 3: 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'**);  
});

### 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-Applikation 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.

### Zugriffsmöglichkeiten

WebdriverIO bietet ebenfalls eine ganze Reihe von Selektoren bzw Lokalisatoren. eine ganze menge an selectoren: css query selector (TAGS!!); Link text (Shortcut mit ‚=‘); partial Link-Text mit (\*=); Element mit Text z.B $(h1=‘TEXT‘); mit partila Text( \*=); id mittel # und cssClass mittels (.) ; kombinationen mittels .someClass\*=SOMETEXT; tagName $(‚<my-element />‘); xPath und diverse MobileSelectores;

Kein button by partialText – alles was nicht einfach zu erreichen ist, ist extrem mühsam

### 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.

Code 3: WebdriverIO Test-Aufbau

Code 3: WebdriverIO Test-Aufbau

**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 4: 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 ...'**, () => {  
 });  
});

#### Anwendungfall 1 – Navigation und Formular (evlt nur Bedingungen

Das Betrachtet von CODE XY zeigt ebenfalls wie klar leserlich der Code mittels des Page-Object-Patterns wird. Dadurch wird ebenfalls kein Unterschied zu Protractor realisiert. Was hingegeng schnell aufffällt ist, dass der Code kompakter und in den Zeilen weniger steht als im Protractor fall. Dies liegt an der Kurzschreibewiese des Selektor/Lokalisators mittel der $-Notation. Hierbei handelt es sich um ein Funktion welche wiederum gekürzt zugriff au die Elemente nimmt.

Zusätzlich haben wir in alle Elemente bestimmte Bedingungen eingebaut. Diese sind leider nicht ganz so zahlreicht wie in Protractor, ermöglichen allerdings eine kompakte scheibeweise. Etwas hinderlich ist hierbei wie in der Testanwendung zu sehen, wenn man darauf warten will, bis ein Element nicht mehr sichtbar ist, kann keine *waitForInvisible*-Funktion aufgerufen werden. Statt dessen wird die *waitForDisplayed*-Funktion mit einem True-Flag.

Die Restlichen Anwendungsfälle gestalteten sich aus machbar mit kleineren Hürden. Mit ausnahme des Anwendungfall 4, welcher Folgende Schritte beinhaltet:

Code 4: Anwendungsfall 1 - WebdriverIO

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'**);  
});

* 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! Und eine switchToWindow-Funktion, welche in diversen Foren und Beiträgen verwendet wurde, welche zwar das agieren auf der Tab ermöglichte. Allerdings brechen die test jedesmal,wenn der Tab Geschlossen werden soll. Dieser Anwendugnfall wurde nach ca 1,5 Stündigen versuchen abgebrochen, da dies bereits eine 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.

<http://www.webdriverjs.com/protractor-vs-webdriverio-vs-nightwatch/>

### Auf Vergleich der Implementierungen

Dieses Kapitel beschäftigt sich mit der „Tauglichkeit“ und dem Vergleich der unterschiedlichen Algorithmen bzw. Implementierungen. Es werden bereits erhobene Daten miteinander verglichen sowie Stärken und Schwächen, die aus diesen erhobenen Daten auszulesen sind, aufgezeigt.

# Fazit

Eine der wichtigsten Kriterien, ob die Implementierungen überhaupt sinnvoll verwendet werden Fazit

# 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)