



Evaluierung von Techniken zur parallel-synchronen Bedienung einer Web-Applikation auf verschiedenen mobilen Endgeräten

vorgelegt von

Adrian Randhahn

EDV.Nr.:744818

dem Fachbereich VI – Informatik und Medien –
der Beuth Hochschule für Technik Berlin vorgelegte Bachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

im Studiengang

Medieninformatik

Tag der Abgabe 27. Mai 2014

Gutachter

Prof. Dipl.-Inform. Knabe Beuth Hochschule für Technik

Prof. Dr.-Ing. Wambach Beuth Hochschule für Technik

Entwurf



Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Datum

Unterschrift

Entwurf

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne und vertrauliche Informationen der Firma New Image Systems GmbH. Die Weitergabe des Inhalts der Arbeit im Gesamten oder in Teilen sowie das Anfertigen von Kopien oder Abschriften - auch in digitaler Form - sind grundsätzlich untersagt. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma New Image Systems GmbH.

Entwurf

Rechtliches

Alle in dieser Arbeit genannten Unternehmens- und Produktbezeichnungen sind in der Regel geschützte Marken- oder Warenzeichen. Auch ohne besondere Kennzeichnung sind diese nicht frei von Rechten Dritter zu betrachten. Alle erwähnten Marken- oder Warenzeichen unterliegen uneingeschränkt der länderspezifischen Schutzbestimmungen und den Besitzrechten der jeweiligen eingetragenen Eigentümern.

Kurzfassung

Im Zuge dieser Arbeit werden verschiedene Frameworks auf ihre Fähigkeit hin untersucht, eine Webapplikation parallel-synchron, vorzugsweise auf mobilen Endgeräten, zu steuern. Das Ziel der Arbeit ist es, einen Überblick über die sich am Markt befindlichen Werkzeuge zu erbringen, die geeignet scheinen, die Arbeit des Qualitätsmanagements hinsichtlich Zeit und Qualität zu optimieren.

Die Frameworks werden im Allgemeinen daraufhin untersucht, ob sie kompatibel mit Desktop-, sowie Mobilbrowsern sind. Ein Teil der Frameworks wirbt mit einer Vielzahl an Funktionen, wohingegen andere nur Teilfunktionalitäten abdecken. Letztere werden als Einzelframeworks bezeichnet. Des Weiteren werden diese Einzelframeworks dahingehend analysiert, ob sie in kombinierter Form in der Lage sind, ein eigenständiges Framework zu bilden.

Zu Zwecken der Evaluation wurden verschiedene Kriterien wie die Installation und Konfiguration, die Erweiterbarkeit oder der Browsersupport festgelegt und vom Autor gewichtet. Die einzelnen Technologien werden kurz vorgestellt, deren Installationsprozess beschrieben und einer Reihe von Tests unterzogen. Nach erfolgreicher Evaluation werden die Ergebnisse noch einmal zusammengefasst und tabellarisch gegenübergestellt.

Inhaltsverzeichnis

1	Einleitung	2
2	Aufgabenstellung	5
2.1	Problemstellung	5
2.2	Zielsetzung	8
2.3	Abgrenzungskriterien	8
3	Grundlagen	11
3.1	Grundsätzlicher Aufbau	11
3.2	Verwendete Hardware	12
3.3	Verwendete Software	14
4	Auswahl der Frameworks	15
4.1	Frameworks als Komplettlösung	15
4.1.1	Ghostlab	15
4.1.2	Adobe Edge Inspect	16
4.1.3	Remote Preview	16
4.1.4	Browser-Sync	17
4.2	Frameworks als Teilkomponente	18
4.2.1	node.js	18
4.2.2	NPM socket.io	18
4.2.3	Zombie.js	18
4.2.4	Phantom Limb	18
4.2.5	jQuery UI Touch Punch	19
4.2.6	jQuery Touchit	19
5	Evaluation der Frameworks	20
5.1	Auflistung des Evaluationsschlüssels	20
5.2	Ghostlab Version 1.2.3	23
5.2.1	Einrichtung der Testumgebung	23
5.2.2	Testen von Desktopbrowsern	23
5.2.3	Testen von Mobilbrowsern	26
5.2.4	Fazit zu Ghostlab	27
5.2.5	Tabellarische Evaluation	28
5.3	Adobe Edge Inspect CC	29
5.3.1	Einrichtung der Testumgebung	29
5.3.2	Testen von Desktopbrowsern	31

5.3.3	Testen von mobilen Browsern	31
5.3.4	Fazit zu Adobe Edge Inspect	33
5.3.5	Tabellarische Evaluation	33
5.4	Remote Preview	34
5.4.1	Einrichtung der Testumgebung	34
5.4.2	Testen von Desktopseiten	34
5.4.3	Testen von mobilen Browsern	34
5.4.4	Fazit zu Remote Preview	35
5.4.5	Tabellarische Evaluation	35
5.5	Browser-Sync	36
5.5.1	Einrichtung der Testumgebung	36
5.5.2	Testen von Desktopseiten	37
5.5.3	Testen von mobilen Browsern	38
5.5.4	Fazit zu Browser-Sync	38
5.5.5	Tabellarische Evaluation	38
5.6	Eigenes Framework	39
5.6.1	Installation eines node.js Servers	39
5.6.2	Einbinden von socket.io	39
5.6.3	Generierung von Steuerbefehlen über socket.io	39
5.6.4	Implementierung eines einfachen Broadcast	41
5.6.5	Einschätzung zur Umsetzung eines eigenen Frameworks	43
6	Zusammenfassung und Ausblick	44
6.1	Zusammenfassung	44
6.1.1	Ausblick	47
	Glossar	48

Abbildungsverzeichnis

1.1	Entwicklungsprozess	3
2.1	Zeitaufwand pro Testgerät	6
2.2	Qualitätssicherung Testszenario	7
2.3	Darstellung der Lernkurve für Frameworks	9
3.1	Übersicht Netzarchitektur	11
5.1	Startbildschirm Ghostlab	23
5.2	Übersicht Clients	24
5.3	Exemplarisch Weinreansicht	25
5.4	Übersicht mobile Clients Ghostlab	26
5.5	Adobe Edge Inspect Daemon Icon	29
5.6	Adobe Edge Inspect App Client hinzufügen	30
5.7	Adobe Edge Inspect Chrome Extension	30
5.8	Adobe Edge Inspect Gerätemanager	31
5.9	Adobe Edge Inspect App Content Darstellung	32
5.10	Remote Preview Steuerungsmaske	34
5.11	Browser-Sync Installation per Konsole	36
5.12	Browser-Sync Initiierung per Konsole	36
5.13	Browser-Sync Starten per Konsole	36
5.14	Browser-Sync Script-Tag	37
5.15	Browser-Sync verbundener Client	37
5.16	socket.io vereinfachte Darstellung eines Broadcasts	40
5.17	socket.io vereinfachte Darstellung eines Broadcasts mit einem Aktionsraum	41
5.18	socket.io Quellcode Scrollbeispiel Serverseitig	42
5.19	socket.io Quellcode Scrollbeispiel Clientseitig	42
5.20	socket.io Quellcode Scrollbeispiel Clientseitig	42

Tabellenverzeichnis

3.1	verwendete Hardware	12
3.2	Übersicht Nokia Lumia 920	12
3.3	Übersicht LG Nexus 4	12
3.4	Übersicht Apple iPhone4 32 GB	13
3.5	Übersicht Apple iPhone5s 16 GB	13
3.6	Übersicht Apple iPad mini Wi-Fi 32GB	13
3.7	Übersicht Microsoft Surface	14
3.8	verwendete virtuelle Maschine	14
3.9	verwendete Browser	14
4.1	von Ghostlab getestete Browser (Stand 10.03.2014, Version 1.2.3)	15
4.2	von Remote Preview unterstützte Plattformen (stand 19.03.2014, letzter Commit 7dc48caa84)	16
4.3	von Browser-Sync getestete Browser (Stand 21.03.2014, Version 0.7.2)	17
5.1	Kriterienübersicht: Installation	20
5.2	Kriterienübersicht: Konfiguration	20
5.3	Kriterienübersicht: Desktop	21
5.4	Kriterienübersicht: Mobil	21
5.5	Kriterienübersicht: Erweiterbarkeit	22
5.6	Kriterienübersicht: Browser	22
5.7	Kriterienübersicht: Aktivität	22
5.8	Gewichtungstabelle Evaluation von Ghostlab	28
5.9	Gewichtungstabelle Evaluation von Adobe Edge Inspect	33
5.10	Gewichtungstabelle Evaluation von Remote Preview	35
5.11	Gewichtungstabelle Evaluation von Remote Preview	38
6.1	Übersicht der Frameworks: GL(Ghostlab), AEI(Adobe Edge Inspect), RP(Remote Preview), B-S(Browser-Sync)	46

1 Einleitung

In der modernen Webentwicklung durchläuft eine Anwendung verschiedene Etappen eines Entwicklungszyklus. Sie beginnt mit einem Auftrag oder einer Idee, darauf folgt dann die Spezifikation einzelner Usecases¹. Im Anschluss folgt in der Regel die Entwicklung und Implementation² der einzelnen Komponenten. Am Ende der jeweiligen Implementationsphase durchläuft das Produkt³ die Qualitätskontrolle. Sollten in dieser Phase Fehler auftreten wird das Produkt dem Entwickler zur erneuten Bearbeitung vorgelegt.

Dieser Vorgang kann sich beliebig oft wiederholen. Bei großen und komplexen Softwaresystemen ist es trotz zeitgemäßer Implementierung nicht immer ausgeschlossen, dass Kaskadierungsfehler⁴ entstehen. Aus Sicht der Qualitätssicherung ist dies ein lästiges Problem, da sie nach jedem erneuten Modifikationsvorganges eines Softwaresegments einen größeren Segmentblock, wenn nicht sogar das gesamte Softwaresystem, erneut testen muss.

Bei der Entwicklung für mobile Endgeräte⁵ kommt noch ein erschwerender Faktor hinzu, nämlich die diversen, verschiedenen Bildschirmauflösungen. Diese können nicht nur die Darstellung des Inhalts beeinflussen, sondern auch daraus folgend die Interaktionskonformität beeinflussen.

Im Optimalfall wird die Software erst nach vollständiger Homogenität auf allen unterstützten Geräten freigegeben. Dies bedeutet, dass auf allen Endgeräten ein identisches Gesamtergebnis, hinsichtlich der Darstellung und dem Nutzerverhalten, erzielt wurde.

Dieser zyklisch wiederkehrende Prozessablauf ist sehr zeitintensiv und nimmt linear mit der Anzahl der zu testenden Geräte zu.

Das Ergebnis dieser Forschungsarbeit soll zeigen, wie verschiedene Softwareframeworks die Zeit, die in die Qualitätssicherung investiert wird, beeinflussen können. Dies soll durch die parallel-synchrone Steuerung diverser Geräte erzielt werden. Unter dem Wortlaut parallel-synchron wird innerhalb dieser Arbeit, die simulierte synchrone Bedienung mehrerer Endgeräte innerhalb des gleichen Zeit-

¹Szenario oder auch Anwendungsfall

²Einbindung

³hier: einzelne Softwarekomponente

⁴Fehler die nicht im eigentlichen Segment auftreten, sondern eine oder mehr Ebenen weiter unten in der Systemhierarchie

⁵Smartphones, Tablets oder Ähnliche



Abbildung 1.1: Vereinfachte Darstellung eines Softwareentwicklungsprozesses

intervalls verstanden. Die Evaluierung soll feststellen wo die Vorteile und Nachteile der einzelnen Werkzeuge liegen. Weiterhin soll ermittelt werden ob aktuelle Frameworks erweiterbar sind um beispielsweise automatisierte Testunits zu implementieren.

Im Kapitel der 'Aufgabenstellung' befasst sich der Autor sich ausschließlich mit der Ermittlung notwendiger Kriterien für die Durchführung der Evaluation und legt fest, welche Wertigkeit die einzelnen Faktoren in Bezug auf die Gesamtbewertung erhalten. Ebenfalls benennt er in diesem Kapitel die Abgrenzungskriterien, welche dazu dienen die Bearbeitung der Aufgabe auf einen vordefinierten Zeitrahmen einzugrenzen.

In dem darauf folgenden Kapitel 'Grundlagen' klärt der Autor alle allgemeinen sowie auch technischen Grundlagen, die notwendig sind diese Abschlussthesis zu verstehen. Er geht ausführlich auf verwendete Begriffe ein, sowie auf Begriffe die in diesem Umfeld entstanden sind. Ein weiterer Punkt innerhalb dieses Kapitels ist die Erläuterung technischer Versuchsaufbauten die im Rahmen der Thesis notwendig waren um eine Evaluation durchzuführen.

Im Kapitel 'Auswahl der Frameworks' befasst der Autor sich kurz mit diversen Frameworks. Es wird deren Herkunft erläutert, womit sie werben und auf welchen Technologien sie aufbauen. Des Weiteren behandelt er in diesem Abschnitt Technologien, die einzelne funktionelle Komponenten beinhalten, welche in Hinsicht auf die Entwicklung eines eigenen Frameworks zur parallel-synchronen Steuerung von Webapplikationen auf mobilen Endgeräten einen Mehrwert liefern.

Das Kapitel 'Evaluation der Frameworks' umfasst die Auswertung der erlangten Ergebnisse. Hier erläutere ich die Resultate meiner Versuchsreihen und wie man die Ergebnisse nutzen kann, eine optimierte Qualitätssicherung von Webapplikationen, mit Fokus auf mobile Endgeräte, vorzunehmen.

Zum Abschluss fasse ich meine Thesis noch einmal zusammen und kläre Fragen und Probleme die während der Bearbeitungszeit auftraten.

Anmerkung

Aus Gründen der besseren Lesbarkeit wird für alle Personen und Funktionsbezeichnungen durchgängig das generische Maskulinum angewendet und bezieht in gleicher Weise Frauen und Männer ein.

2 Aufgabenstellung

Die Aufgabe dieser Thesis ist die Evaluierung von Techniken zur parallel-synchronen Steuerung von Webapplikationen auf mobilen Endgeräten, um damit die Produktivität der Qualitätssicherung zu steigern.

2.1 Problemstellung

Ein Problem in der aktuellen Softwareentwicklung ist die immer weiter steigende Anzahl an Endgeräten, welche mit verschiedenen Bildschirmauflösungen und eigenen Betriebssystemen in unterschiedlichen Versionen auftreten. Ein Qualitätsprüfer investiert daher linear ansteigend zur Anzahl der zu testenden Geräte Zeit, um vereinzelte Testszenarien durchzuarbeiten. Solch ein Testszenario kann Navigationsabläufe¹, das Ausfüllen und Validieren eines Formulars oder auch das Überprüfen funktionaler² Links sein. Bereits an dieser Stelle ist der zu investierende Aufwand, und dies wiederkehrend, enorm.

Wenn der Qualitätsprüfer innerhalb eines Testszenarios einen schwerwiegenden Fehler bei einem der Geräte entdeckt, muss er den Vorgang beenden. Der Fehler wird den Entwicklern gemeldet, welche diesen dann beheben. Nach korrigierter Quellcodeumsetzung darf der Qualitätsprüfer nicht davon ausgehen, dass bereits kontrollierte Abschnitte immer noch voll funktionsfähig sind. Eventuell treten neue Fehler in bereits kontrollierten Segmenten auf. Daher beginnt an diesem Punkt ein erneuter Durchlauf der Qualitätssicherung.

¹ein Nutzerspezifischer Gang durch die Webseite

²aktive Links und deren Aufruf

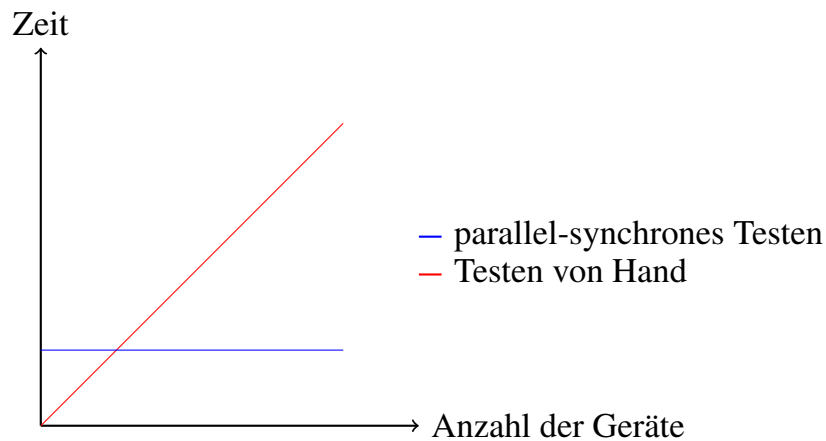


Abbildung 2.1: Zeitaufwand pro Testgerät

Sollte ein Szenario aufgrund eines Fehlers abgebrochen worden sein, wird dem Entwickler das Problem möglichst konkret geschildert. Dessen Aufgabe ist es nun das Problem zu beheben. Ist dies geschehen startet der Prüfer einen erneuten Durchgang des Szenarios. Ein generelles Problem was hier noch zusätzlich besteht, ist der Umstand, dass sich gerade bei nur kleineren Fixes³ und immer wieder auftretenden Testszenarioschleifen eine gewisse Routine einschleicht, worunter die Qualität des Produkts leiden kann.

³Problemlösungen, Codeanpassungen

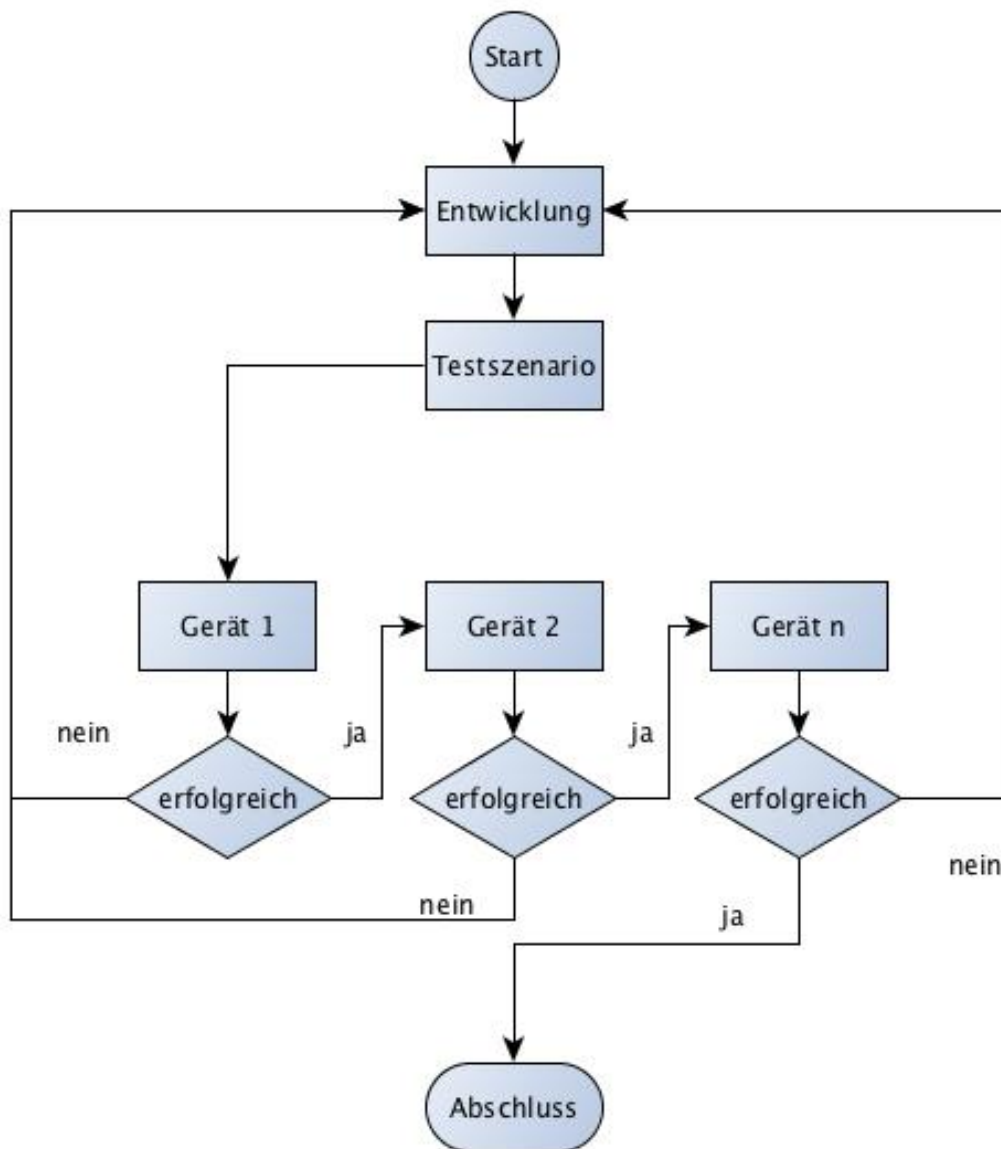


Abbildung 2.2: Darstellung eines Qualitätssicherungsablaufes in der mobilen Anwendungsentwicklung

2.2 Zielsetzung

Das Ziel dieser Arbeit ist es, bestehende Frameworks auf ihre Tauglichkeit in Bezug auf die parallel-synchrone Steuerung von mobilen Endgeräten zur Durchführung von Testszenarien zu evaluieren. Hierzu werden auf mobilen Endgeräten die internen Browser getestet. Hinzu kommen auf Desktopgeräten die aktuellen Versionen von Firefox, Chrome, Safari (nur für Mac-Desktopgeräte) und der Internet Explorer (nur für Windows-Desktops). Diese sind entgegen der Ausgangssituation, nur mobile Browser zu bewerten, wichtig, denn in der derzeitigen Appentwicklung werden immer öfter plattformunabhängige responsive Weblösungen genutzt, welche sich dem jeweiligen Gerät anpassen. Um eine allgemeine Testbarkeit zu gewährleisten werden die Frameworks auch auf Genauigkeit in virtuellen Umgebungen analysiert. Dabei können Abweichungen, seien sie noch so klein, entstehen. Bereits 1 Pixel Abweichung kann bereits ausschlaggebend sein einen Umbruch zu erzeugen und damit das Layout negativ zu verändern.

2.3 Abgrenzungskriterien

Zeit

Als eines der wichtigsten Abgrenzungskriterien gilt es die Einarbeitungszeit zu bewerten. Hier bedeutet je kürzer desto besser, immer gesehen in Relation zum Umfang des Frameworks. Diese setzt sich aus verschiedenen Aspekten des Evaluationsschlüssels zusammen. So wirken sich etwa eine gut dokumentierte API, Installationsanleitungen oder ein guter Support positiv auf die Einarbeitungszeit aus. Ein Framework ist als qualitativ hochwertiger zu bewerten, wenn es eine exponentielle Lernkurve in Relation zur Zeit vorweist.

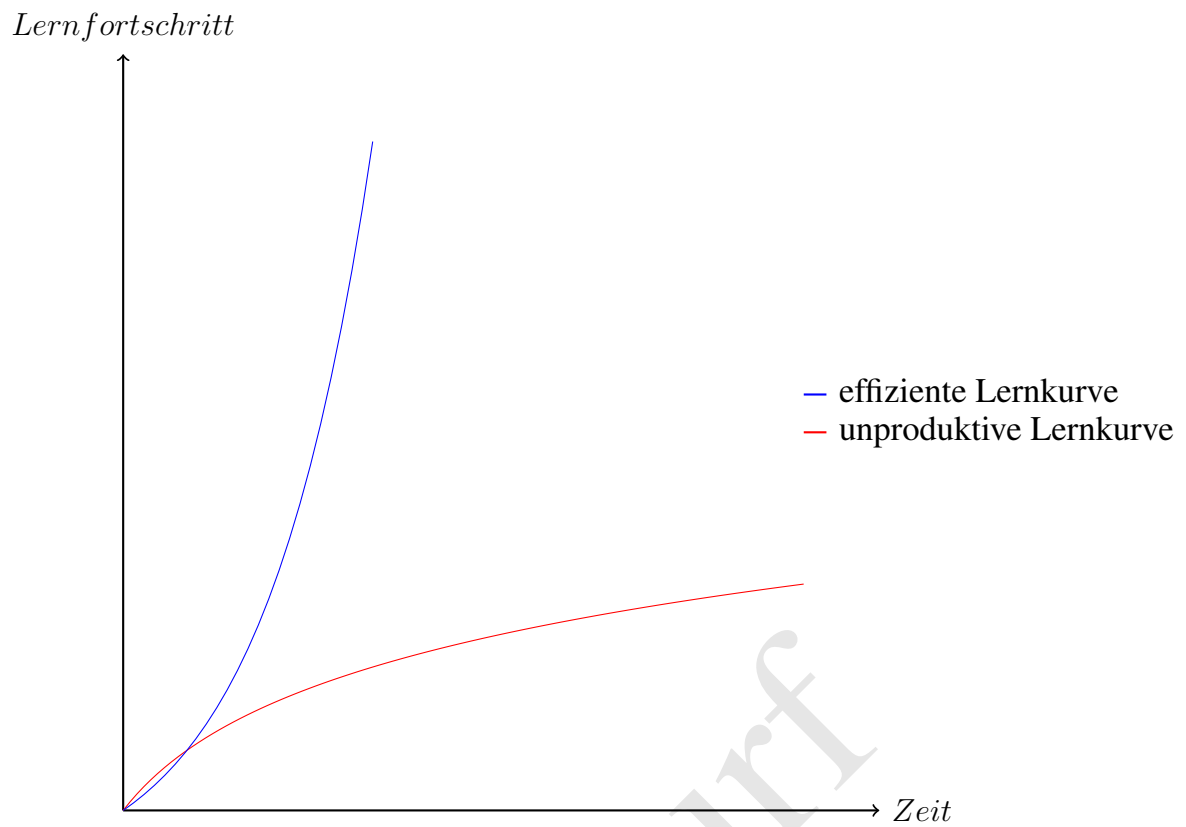


Abbildung 2.3: Lernkurve für Frameworks

Erweiterbarkeit

In Hinsicht auf mehrere Endgeräte gibt es folgende Aspekte zu analysieren. Zum einen ob das Framework auf verschiedene Arten an das System gebunden ist, wie etwa Android oder iOS. Zum anderen ob es eine Limitierung der Anzahl der anzuschließenden Geräte gibt.

Unter den Aspekt der Erweiterbarkeit fällt auch die Möglichkeit das Framework um eigene Funktionalitäten zu erweitern. So sollte im Idealfall ein Framework ein Grundgerüst liefern, auf das der Entwickler mit eigenen Erweiterungen aufbauen kann um das gewünschte Ziel zu erreichen.

Steuerbefehle

Ein wichtiger Aspekt dieser Arbeit ist die Steuerung, beziehungsweise die Generierung von Steuerbefehlen und deren Verteilung auf alle verbundenen Klienten. Insbesondere wurde in dieser Arbeit auf das Scrollen innerhalb einer Web-Applikation, das Ausfüllen von Formularen mit den dazugehörigen Eingabefeldern, sowie das Ausführen eventgesteuerter Aktionen Wert gelegt.

Unterstützte Browser

Ein wichtiger Aspekt der durchzuführenden Tests beinhaltet die Unterstützung möglichst verschiedener Browser. Dies hat den Grund, dass lediglich ein Framework, welches eine große Spanne an Endgeräten abdeckt die Chance hat effektiv genutzt werden zu können.

Virtuelle Umgebung

Ein positiv in die Validierung einfließender Aspekt ist die Einbindung oder Verwendung des Frameworks innerhalb einer virtuellen Umgebung. Das wird durch den Fakt begründet, dass der Tester nicht immer im Besitz aller Testgeräte oder Umgebungen ist. So ist es ohne eine virtuelle Maschine zum Beispiel nicht möglich eine Seite im Internet Explorer innerhalb einer MacOS-Umgebung zu testen, da diese ihn nicht unterstützt.

3 Grundlagen

In diesem Abschnitt werden allgemeine technische Abläufe, die notwendig sind um diese Arbeit und die darin verwendeten Techniken zu verstehen, behandelt. Des Weiteren werden die verwendeten Hardwarekomponenten, sowie genutzte Browser und Software aufgeführt.

3.1 Grundsätzlicher Aufbau

Alle in dieser Arbeit behandelten Softwareprodukte und Frameworks benötigen entweder einen lokal installierten Webserver, wie z. B. Apache, um ihre Daten zu übermitteln, oder sie bringen einen softwareinternen Server mit sich. Als Voraussetzung für eine funktionierende Kommunikation müssen die Testgeräte sich alle innerhalb eines gemeinsamen Subnetzes, befinden. Zum Arbeiten empfiehlt sich ausserdem eine durchgängige Stromzufuhr der verbundenen Geräte.

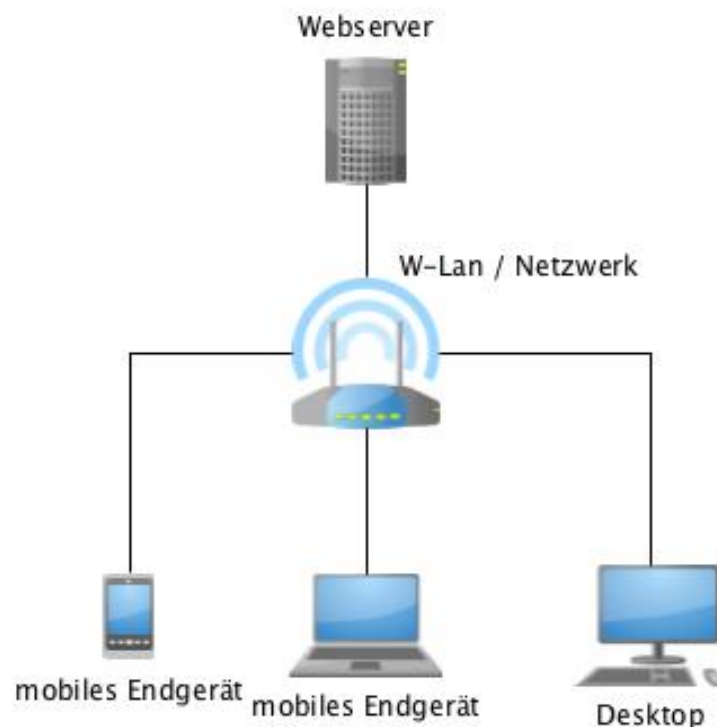


Abbildung 3.1: Übersicht Netzarchitektur

3.2 Verwendete Hardware

Alle in dieser Thesis aufgeführten Tests werden mit den nachfolgenden Geräten und Umgebungen ausgeführt und validiert.

Apple iMac 27"

Zur Durchführung dieser Arbeit und der darin enthaltenen Evaluationsverfahren wird ein Apple iMac mit folgenden Spezifikationen genutzt.

Prozessor	3,4GHz Intel Core i7
Speicher	8GB 1600Mhz DDR3
Grafikkarte	NVIDIA GeForce GTX 675MX 1024 MB
Betriebssystem	OS X 10.8.5 (12F45)

Tabelle 3.1: verwendete Hardware

Mobile Endgeräte

Die in dieser Arbeit durchgeführten Tests nutzen folgende Endgeräte.

Nokia Lumia 920

Komponente	
Betriebssystem	Windows Phone
Versionsnummer	8.0
Bildschirmdiagonale	11,4 cm (4,5 Zoll)
Auflösung	768x1280
primäre Ausrichtung	Porträt

Tabelle 3.2: Übersicht Nokia Lumia 920

LG Nexus 4

Komponente	
Betriebssystem	Android
Versionsnummer	4.4.2 (KitKat)
Bildschirmdiagonale	11,9 cm (4,7 Zoll)
Auflösung	768x1280
primäre Ausrichtung	Porträt

Tabelle 3.3: Übersicht LG Nexus 4

Apple iPhone4 32 GB

Komponente	
Betriebssystem	iOS
Versionsnummer	6.1.3 (10B329)
Bildschirmdiagonale	8,9 cm (3,5 Zoll)
Auflösung	640 x 960
primäre Ausrichtung	Porträt

Tabelle 3.4: Übersicht Apple iPhone4 32 GB

Apple iPhone5s 16 GB

Komponente	
Betriebssystem	iOS
Versionsnummer	7.0.6 (11B651)
Bildschirmdiagonale	10,2 cm (4,0 Zoll)
Auflösung	640 x 1136
primäre Ausrichtung	Porträt

Tabelle 3.5: Übersicht Apple iPhone5s 16 GB

Apple iPad mini Wi-Fi 32GB

Komponente	
Betriebssystem	iOS
Versionsnummer	7.0.4 (11B554a)
Bildschirmdiagonale	20,1 cm (7,9 Zoll)
Auflösung	1024 x 768
primäre Ausrichtung	Landschaft

Tabelle 3.6: Übersicht Apple iPad mini Wi-Fi 32GB

Microsoft Surface

Komponente	
Betriebssystem	Windows
Versionsnummer	8.1 Pro
Bildschirmdiagonale	26,9 cm (10,6 Zoll)
Auflösung	1920 x 1080
primäre Ausrichtung	Landschaft

Tabelle 3.7: Übersicht Microsoft Surface

3.3 Verwendete Software

Virtuelle Maschine

Hersteller	Oracle VM
Product	VirtualBox
Version	4.2.16 r86992
Image	Windows Vista
virtueller Speicher	1024 MB

Tabelle 3.8: verwendete virtuelle Maschine

Verwendete Browser

Browser	Version
Google Chrome	34.0.1847.116
Google Chrome (virtuell)	33.0.1750.149 m
Mozilla Firefox	28.0
Mozilla Firefox (virtuell)	25.0.1
Opera	20.0
Safari	6.1.3 (8537.75.14)
Internet Explorer (virtuell)	9.0.8112.16421

Tabelle 3.9: verwendete Browser

4 Auswahl der Frameworks

4.1 Frameworks als Komplettlösung

Unter diesem Punkt werden alle Frameworks gelistet, welche damit werben das parallel-synchrone Testen von mobilen Seiten zu ermöglichen und somit den Entwicklungsprozess zu optimieren. Im Gegensatz zu den Einzelkomponenten sollten diese im Idealfall ohne weitere Technologien genutzt werden können.

4.1.1 Ghostlab

Ghostlab ist ein Framework des Schweizer Unternehmens Vanamco. Es verspricht das synchrone Testen von Webseiten in Echtzeit. Weiterhin wirbt das Unternehmen mit einem umfangreichen Repertoire an nützlichen Fähigkeiten. Der Funktionsumfang umfasst das Scrollen innerhalb einer Seite, das Ausfüllen von Formularen, das Wahrnehmen und Reproduzieren von Click-Events sowie das Neuladen einer Seite. Ghostlab besitzt ebenso einen Inspektor, welcher die Analyse des DOM, der 'on the fly'¹ Bearbeitung von CSS und der Analyse und Bearbeitung von Javascriptdateien ermöglichen soll. Das Framework gibt an für alle folgenden Browser zu funktionieren, ohne diese konfigurieren zu müssen:

Browser	Version
Firefox	latest
Chrome	latest
Safari	latest
Internet Explorer	8/9/10
Opera	11
Opera Mobile	supportet
Firefox Mobile	supportet
Blackberry	supportet
Windows Phone	supportet
Safari mobile	supportet
Android	2.3 - 4.2

Tabelle 4.1: von Ghostlab getestete Browser (Stand 10.03.2014, Version 1.2.3)

¹ Anpassungen in Echtzeit, ohne Neuladen der Seite

Der Kostenpunkt der Lizenz liegt bei Erstellung dieser Arbeit bei 49\$ (entspricht in etwa 35,30 € Stand: 20.3.2014). Zur Erstellung dieser Thesis wurde die 7-Tage-Testvollversion genutzt.

4.1.2 Adobe Edge Inspect

Die Anwendung Edge Inspect stammt von Adobe und wird derzeit in der CC² Version vertrieben. Um Adobe Edge Inspect nutzen zu können, bedarf es drei separater Komponenten. Adobe wirbt mit synchronem Aufrufen und Auffrischen von Websites, sowie deren Inspizierung per *weinre*³. Besonders angepriesen wird von Adobe die Nutzung und Verwendung der Adobe Edge Inspect API, welche auf GitHub zur Verfügung gestellt wird. Des Weiteren kann Adobe Edge Inspect in andere Edge Produkte⁴ integriert werden.

Adobe Edge Inspect CC steht 30 Tage kostenlos zum Testen bereit. Danach fallen ab 24,59 / Monat für die Nutzung des Einzelprodukt-Abos an. Die Anwendung läuft nur auf mobilen Endgeräten mit iOS oder Android Betriebssystem.

4.1.3 Remote Preview

Remote Preview ist ein kleines Javascriptframework von dem Webdesigner Viljami Salminen aus Helsinki, Finnland. Es überprüft alle 1100ms per AJAX-Request, ob sich die Quell-URL geändert hat und teilt dies dann den verbundenen Testgeräten mit. Es wirbt mit dem synchronen Aufruf von Webseiten auf einer Vielzahl von Plattformen:

Plattform
Android OS 2.1 - 4.1.2 (Default browser + Chrome)
Blackberry OS 7.0 (Default browser)
iOS 4.2.1 - 6 (Default browser)
Mac OS X (Safari, Chrome, Firefox, Opera)
Maemo 5.0 (Default browser)
Meego 1.2 (Default browser)
Symbian 3 (Default browser)
Symbian Belle (Default browser) WebOS 3.0.5 (Default browser)
Windows Phone 7.5 (Default browser)
Windows 7 (IE9)

Tabelle 4.2: von Remote Preview unterstützte Plattformen (stand 19.03.2014, letzter Commit 7dc48caa84)

²Creative Cloud

³Remotedebugger, zur Inspizierung von Ressourcen einer Website <http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html>

⁴zum Beispiel Edge Reflow CC und Edge Code CC

Das Framework ist kostenlos erhältlich und läuft unter der MIT Lizenz. Zum Zeitpunkt dieser Arbeit scheint das Projekt nicht weiter entwickelt zu werden, da seit 5 Monaten auf der Projektseite keinerlei Aktualisierungen vorgenommen wurden.

4.1.4 Browser-Sync

Browser-Sync wurde von Shane Osbourne entwickelt und soll im Zuge dieser Arbeit zeigen, dass es zur Verbesserung der Qualität während des Entwicklungsprozesses einer Webapplikation beiträgt. Es wirbt mit synchronisierter Steuerung, dem Entwickeln von CSS Styles und anderen Projektdateien in Echtzeit, der Installation unter Windows, MacOS und Linux und einer umfangreichen Palette an unterstützten Plattformen. Jedoch unterstützt das Framework im Gegensatz zu Ghostlab oder Adobe Edge Inspect keine Remoteinspection des DOM und der Netzwerkaktivitäten.

Browser	Version
Firefox	latest
Chrome	latest
Safari	latest
Internet Explorer	7/8/9/10
Opera	latest
Opera Mobile	supportet
Firefox Mobile	supportet
Blackberry	supportet
Windows Phone	supportet
Safari mobile	supportet
Android	supportet
iOS	supportet

Tabelle 4.3: von Browser-Sync getestete Browser (Stand 21.03.2014, Version 0.7.2)

Das Framework basiert auf dem node.js Framework und besitzt dadurch ein hohes Erweiterungspotential. Eine parallel zu Browser-Sync entwickelte Erweiterung kombiniert Browser-Sync mit Grunt, was automatisierte Abläufe ermöglicht. Diese fördert die Produktivität durch das Einbinden des Frameworks in bestehende Arbeitsabläufe. Die Software ist kostenlos erhältlich und steht unter der MIT Lizenz. Das Projekt befindet sich zum Zeitpunkt dieser Arbeit in der Version 0.7.2 und wird täglich weiterentwickelt.

4.2 Frameworks als Teilkomponente

Als Teilkomponenten werden hier Frameworks spezifiziert, welche dazu beitragen, das in der Thesis geforderte Werkzeug selbst zu entwickeln. Diese decken verschiedene spezielle Funktionen ab, wie zum Beispiel die Clientverwaltung, Steuerbefehle oder setzen die Voraussetzung für eigene Testszenarien.

4.2.1 node.js

node.js Aufgabe besteht darin, anstelle von beispielsweise Apache, einen Webserver zur Verfügung zu stellen, welcher komplett auf Javascript basiert. Alle notwendigen serverseitigen Anfragen und Funktionen erfolgen in Javascript. Entwickelt wird node.js von der kalifornischen Firma Joyent und befindet sich derzeit in Version 0.10.26. Geführt wird node.js unter der MIT Lizenz und steht kostenlos auf nodejs.org oder unter GitHub zum Download bereit.

4.2.2 NPM socket.io

socket.io ist ein Framework welches die WebSocket Technologie aktueller Browser auf Javascript Ebene abbildet. WebSocket verfolgt den Ansatz nicht in regelmäßigen Abständen Anfragen an den Server zu stellen und damit unnötig viel Datenvolumen zu generieren, sondern eine permanente Verbindung zum Server aufrecht zu halten um auf Statusänderungen am Server zu reagieren. socket.io wurde von Guillermo Rauch unter der MIT Lizenz entwickelt und steht derzeit in der Version 0.9.16 auf GitHub oder per NPM zur Verfügung.

4.2.3 Zombie.js

Das Framework Zombie.js ist ein Open-Source Projekt einer ganzen Gruppe von Entwicklern⁵, welches von dem Kalifornier Assaf Arkin ins Leben gerufen wurde. Zombie.js wirbt mit seiner Einfachheit Tests zu erstellen und in Testsuiten zu integrieren. Zombie.js emuliert einen sogenannten headless⁶ Browser. Dies hat zur Folge, dass natürlich nur nonvisuelle Aspekte in Tests integriert werden können, wie etwa das Ausfüllen von Formularen, das Navigieren durch den Navigationsbaum oder das Testen von Links.

4.2.4 Phantom Limb

Phantom Limb ist ein von Brian Carstensen entwickeltes Werkzeug, welches es ermöglichen soll, die Computermouse generierten Bewegungen in äquivalente Touchevents umzuwandeln. Das Framework läuft unter der Apache Lizenz 2 und kann kostenlos

⁵<https://github.com/assaf/zombie/graphs/contributors>

⁶Kopflos - ohne Gerüst das ihn umschließt, oder auch virtuell

verwendet werden. Es kam in die Auswahl der Frameworks, da seine Funktionalität zur Generierung von Steuerbefehlen geeignet ist.

4.2.5 jQuery UI Touch Punch

jQuery UI Touch Punch ist eine Erweiterung zu der UI Bibliothek von jQuery die David Furfero entwickelt hat. jQuery UI Touch Punch erlaubt von Hause aus nicht die Nutzung von Touchevents auf mobilen Endgeräten, darum hebt die Erweiterung diese Restriktion auf, ohne weiter konfiguriert werden zu müssen. An Quellcode kommen lediglich weitere 584 Bytes hinzu. Die Frameworkerweiterung läuft unter der MIT und der GNU General Public License, wodurch es dem Endnutzer frei steht die Bibliothek unter den Lizenzen des eigenen Projektes zu verwenden.

4.2.6 jQuery Touchit

Das von Daniel Glyde entwickelte Framework jQuery Touchit, wandelt Berührungen in äquivalente Mousevents um und ermittelt deren relative Position in Bezug auf den Viewport. Des Weiteren löst es das Problem bei bereits bestehenden jQuery Anwendungen und deren Darstellung auf mobilen Endgeräten, wo verschiedene Funktionalitäten, wie zum Beispiel die Verwendung von Slidern, nicht nutzbar sind.

5 Evaluation der Frameworks

5.1 Auflistung des Evaluationsschlüssels

Zur Evaluierung der einzelnen Frameworks, wurde ein Schlüssel erstellt, welcher messbare Aspekte abdeckt, die im Vorfeld der Arbeit bereits als wichtig erachtet wurden, um die aufgestellte Thesis in Zahlen darzustellen. Ergänzend kamen Punkte hinzu, die bei der Installation und der Nutzung auffielen und sich als wichtig erwiesen.

Der Schlüssel wurde in sieben Hauptkategorien aufgeteilt. Die Abschnitte 'Installation' und 'Konfiguration' befassen sich in erster Linie mit der Verfügbarkeit, dem Zugang zu dem Framework, dessen Installation und Dokumentation sowie der Voraussetzung anderer Technologien, um es zu nutzen.

Installation

Kriterium	Punktezahl
Notwendigkeit von anderen Technologien	4
Nutzbar direkt nach der Installation	2
Installationsanleitung vorhanden	2
FAQ vorhanden	2

Tabelle 5.1: Kriterienübersicht: Installation

Konfiguration

Kriterium	Punktezahl
Nutzbar ohne Konfiguration	4
Konfigurierbarkeit (IP und Ports)	1
Konfigurierbare Workspaces	2
Support vorhanden (Wiki, Helpdesk, EMail, Forum)	2
Intuitive Benutzeroberfläche	1

Tabelle 5.2: Kriterienübersicht: Konfiguration

Der Teilschlüssel 'Funktion' wurde dupliziert und in einen Mobilteil und einen Desktopteil aufgeteilt, da das Testen von mobilen Seiten andere Schwerpunkte der Bewertung haben sollte, als das von Desktopseiten. So ist eine funktionierende synchrone Gestenkontrolle beim Verwenden von mobilen Seiten zum Beispiel unerlässlich, wohingegen sie auf Desktops durch den Einsatz einer Maus tendenziell eher unhandlich in der Nutzung ist.

Funktion: Desktop

Kriterium	Punktezahl
Darstellung : normale Seiten	2
Darstellung : gesicherte Seiten	2
Darstellung: normale Reaktionsgeschwindigkeit (< 1 Sekunde)	2
Funktion: Seitensteuerung	3
Funktion: Javascript	1

Tabelle 5.3: Kriterienübersicht: Desktop

Funktion: Mobil

Kriterium	Punktezahl
Darstellung : normale Seiten	1
Darstellung : gesicherte Seiten	1
Darstellung: normale Reaktionsgeschwindigkeit (< 1 Sekunde)	2
Funktion: Seitensteuerung	4
Funktion: Javascript	1
Funktion: Gestenkontrolle	1

Tabelle 5.4: Kriterienübersicht: Mobil

Da es in der Praxis kein Framework gab, welches zum Zeitpunkt dieser Arbeit in der Lage war alle gewünschten Aspekte abzudecken, war es wichtig das Werkzeug um eigene Funktionen oder externe Frameworks zu ergänzen. Dies wurde unter Berücksichtigung der API, der Lizenz des Frameworks und dessen Dokumentation bewertet.

Erweiterbarkeit

Kriterium	Punktezahl
API Zugang	5
API lizenztechnisch gesichert	2
API Dokumentation	3

Tabelle 5.5: Kriterienübersicht: Erweiterbarkeit

Ein weiterer wichtiger Aspekt, ist die Unterstützung möglichst vieler verschiedener Browser auf dem Desktop, auf dem Mobilgerät und in der virtuellen Umgebung.

Browser Support (aktuelle Versionen)

Kriterium	Punktezahl
mobile Plattformen (iOS, Android, Windows)	3
Virtuelle Browser	2
Chrome	1
Opera	1
Firefox	1
Safari	1
Internet Explorer	1

Tabelle 5.6: Kriterienübersicht: Browser

Die Aktivität einer Software lässt darauf schließen, ob und gegebenenfalls wie, diese sich in Zukunft entwickeln kann. So sind von einem inaktiven Entwicklungsstand, von über einem halben Jahr, keine neuen Ergebnisse mehr zu erwarten und man muss davon ausgehen, dass die Software um keine neuen Features erweitert werden wird.

Aktivität

Kriterium	Punktezahl
Noch in der Entwicklung (letztes Release, Commit Häufigkeit)	5
Aktives Forum	5

Tabelle 5.7: Kriterienübersicht: Aktivität

5.2 Ghostlab Version 1.2.3

5.2.1 Einrichtung der Testumgebung

Ghostlab kommt von Hause aus mit einer 7-Tage-Testversion. Die Installation verlief einfach und ereignislos. Nachdem das Tool installiert wurde, erfolgte die Zuweisung einer Website zu dem Ghostlabserver. Es wurden in diesem Fall sowohl eine Seite auf einem lokalen Apache Server getestet, als auch die mitgelieferte Demoseite von Ghostlab. Nach dem Start des Ghostlabservers ist dieser über den localhost¹ auf Port 8005 (Default) von allen zu testenden Geräten erreichbar.

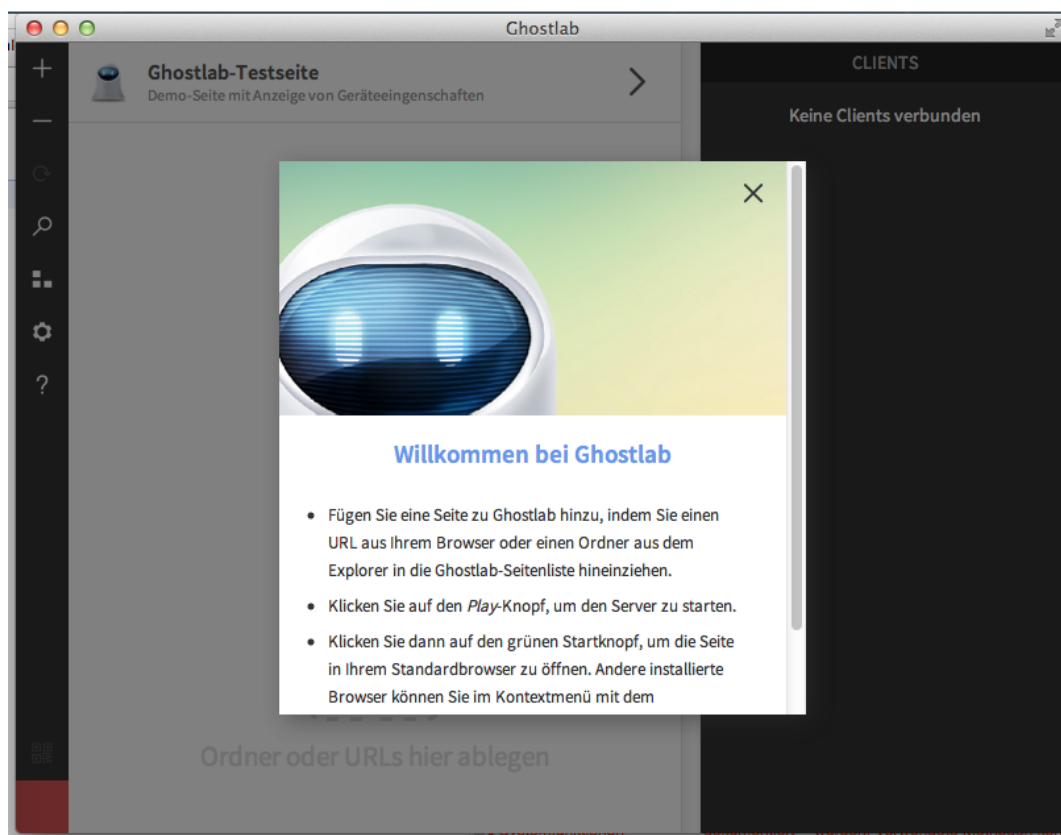


Abbildung 5.1: Startbildschirm von Ghostlab nach der Installation

5.2.2 Testen von Desktopbrowsern

Durch Aufrufen der IP-Adresse des Rechners auf dem der Ghostlabserver läuft, verbindet sich der Browser als Client und wird fortan durch gesendete Signale

¹IP-Adresse des lokalen Rechners

beeinflusst. Hierzu zählen auch virtuelle Browser. Jeder Client wird nun gleichzeitig Sender und Empfänger für Signale, das bedeutet, dass jede Aktion parallel-synchron auf allen anderen Clients gespiegelt wird. Hierzu zählen Javascriptevents, das Ausfüllen eines Formulars oder das Neuladen der gesamten Seite.

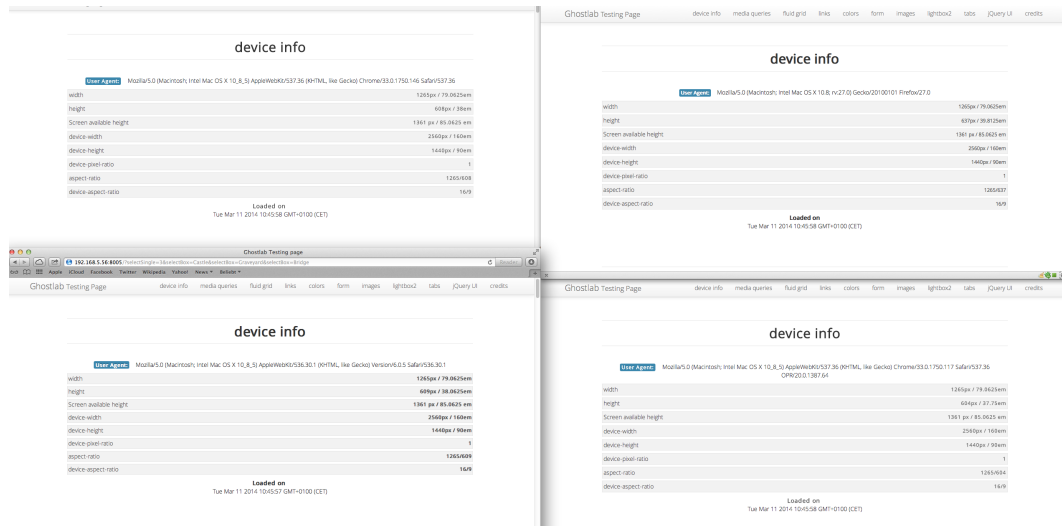


Abbildung 5.2: Darstellung von 4 verschiedenen Clients

Über den Übersichts Bildschirm kann jeder verbundene Client einzeln inspiziert werden. Hier ist der Nutzer in der Lage sich durch das DOM zu navigieren oder temporäre CSS Anpassungen vorzunehmen. Die Handhabung ist intuitiv, was jedoch an dem verwendeten Framework weinre liegt.

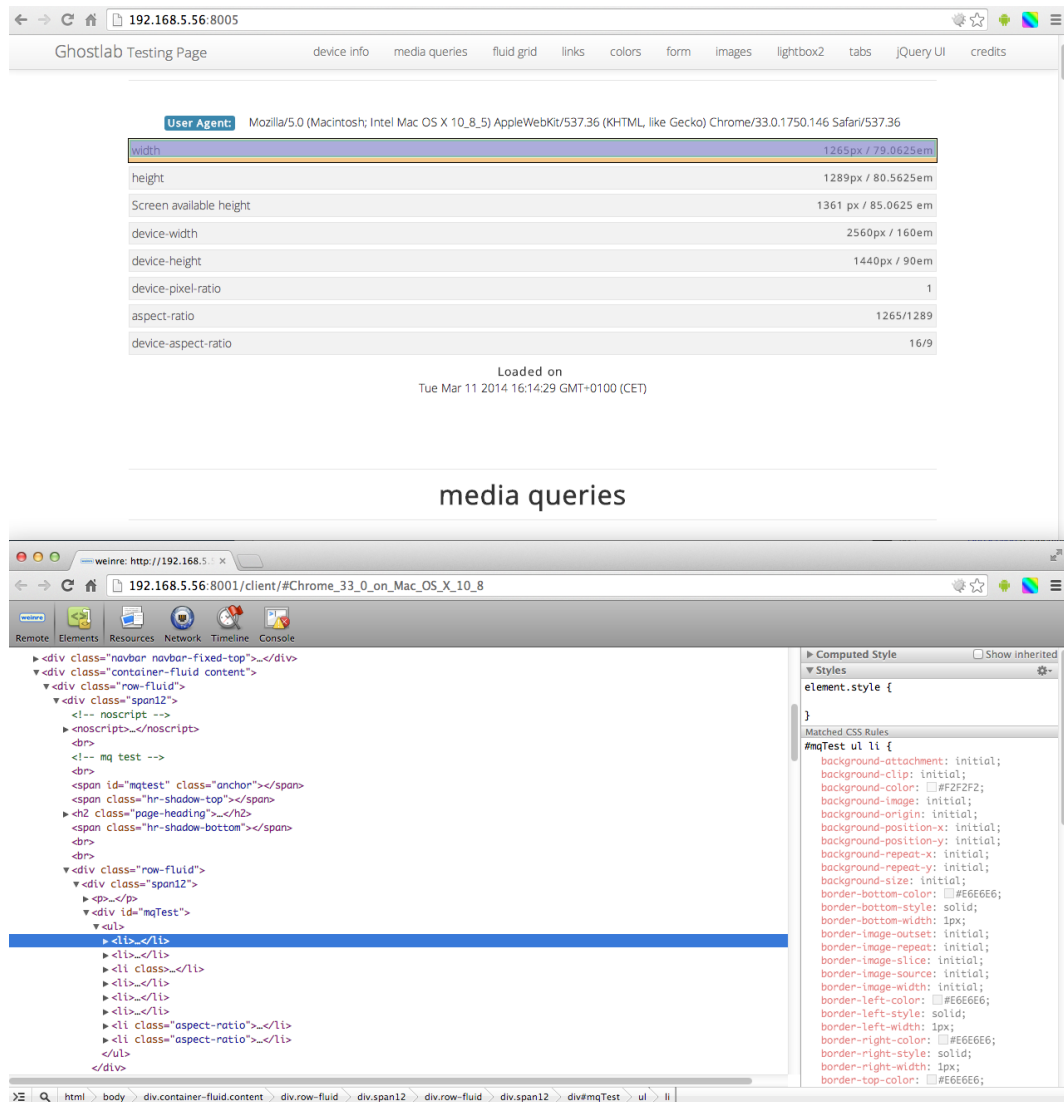


Abbildung 5.3: ausgewähltes DOM-Element in weinre

5.2.3 Testen von Mobilbrowsern

Das Einrichten zum Testen auf mobilen Endgeräten verläuft synchron zu den Desktopbrowsern. Man ruft innerhalb des Browsers die IP-Adresse des Ghostlabrechners auf und ist schon nach wenigen Sekunden² in die Clientliste aufgenommen.

Bei dem Testen auf mobilen Browsern ist es bei Ghostlab³ notwendig ausreichend Zeit zwischen den Eingaben zu lassen, da es sonst bei unterschiedlich schnellen Geräten zu einem Effekt kommt, bei dem die langsameren Geräte beim Ausführen des letzten Signals gleichzeitig wieder zum Sender für alle anderen Geräte werden.

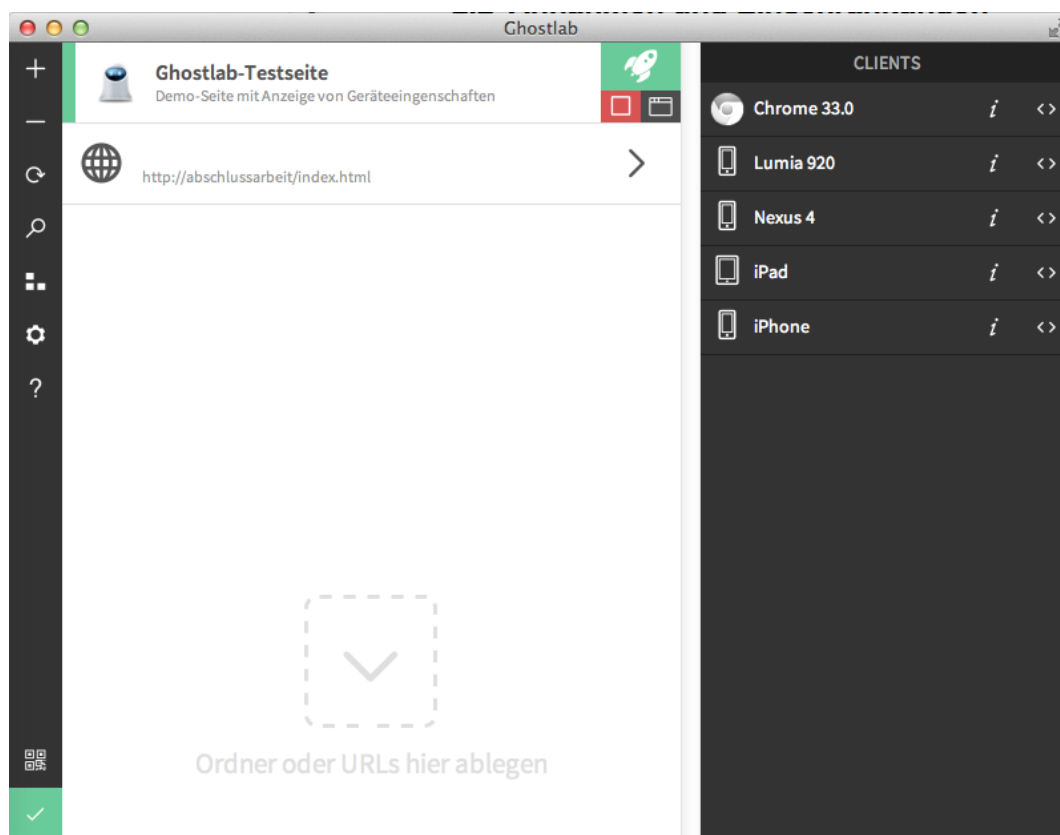


Abbildung 5.4: Ghostlabübersicht der verbundenen Clients

²abhängig von der Geschwindigkeit des Testgerätes

³Version 1.2.3

5.2.4 Fazit zu Ghostlab

Für diese Arbeit wurde Version 1.2.3 von Ghostlab genutzt. Zu diesem Zeitpunkt verfügte die Software noch über keinen Master/Slave-Modus⁴, dadurch kam es bei meinen Testgeräten bereits nach wenigen Minuten zu dem Problem, dass die Geräte sich in einer Endlosschleife von Senden und Empfangen der Steuerbefehle befanden. Für kommende Versionen ist ein solcher Modus laut den Entwicklern aber geplant. Das Problem rührt daher, dass einige Geräte schneller auf die übermittelten Befehle reagieren, als andere. Das führt dazu, dass die langsam ladenden Geräte in dem Augenblick wo sie das Signal umsetzen, für die schnelleren Geräte bereits wieder als Sender fungieren. Dieses Problem tritt bereits bei einer kleinen Anzahl von Geräten auf und wird deshalb als kritisch eingestuft.

Das Testen in mehreren Browsern auf einem Rechner lief hingegen sehr gut. Das Ausführen von Javascript läuft einwandfrei. Das Ausfüllen von Inputs, Checkboxes, Radioboxen und das Absenden des Formulars funktionierte bis auf die Kalenderauswahl im Firefox Browsers anstandslos. Ein Problem scheint das Werkzeug mit passwortgeschützten Seiten zu haben. Diese lassen sich erst nach mehrfacher, abhängig vom jeweiligen Browser, Eingabe des Passwortes aufrufen. Diese Prozedur wiederholt sich für jede weitere Unterseite.

Das Arbeiten in einer virtuellen Umgebung⁵ wird problemlos unterstützt. Das einzige Problem, was ich analysieren konnte, war dass sich virtuelle Browser nicht in einen Workspace integrieren lassen.

Ghostlab unterstützt die Funktion von Workspaces⁶, welche die Position und Größe der verschiedenen Browserfenster speichern. Per Knopfdruck lassen diese sich dann im Kollektiv öffnen, sofern in den Browsereinstellungen die Popups aktiviert sind, für die zu testende Seite. Dieses Feature⁷ bewerte ich als positiv gegenüber dem Zeitaufwand, diesen Vorgang immer wieder manuell auszuführen.

Als Kritikpunkt bewerte ich die nicht existente Möglichkeit, die Anwendung um eigene Funktionalität zu erweitern.

⁴ein Gerät dient als Steuergerät, alle anderen folgen ihm

⁵es wurde VirtualBox von Oracle genutzt

⁶Arbeitsumgebung oder auch Arbeitsumfeld

⁷Funktion, welche ein Teil der Anwendung ist

5.2.5 Tabellarische Evaluation

Komponente	Punkte	Wertigkeit
Installation	10	10 %
Konfiguration	10	10 %
Funktion: Desktop	8	25 %
Funktion: Mobil	5	25 %
Erweiterbarkeit	0	10 %
unterstützte Browser	10	10 %
Aktivität	5	10 %
Gesamt	67.5	100 %

Tabelle 5.8: Gewichtungstabelle Evaluation von Ghostlab

Entwurf

5.3 Adobe Edge Inspect CC

5.3.1 Einrichtung der Testumgebung

Es sind drei Schritte notwendig Adobe Edge Inspect zum Einsatz vorzubereiten. Als erstes benötigen wir den Client aus der Adobe Creative Cloud (CC) Kollektion. Diese gibt es zum Zeitpunkt dieser Arbeit in verschiedenen Modellen und beginnt bei der kostenlosen 30-Tage Testversion, geht über die Einzellizenz, für ausschliesslich Adobe Edge Inspect, von 24,59 € / Monat bis hin zum Komplett-Abo, was dann mit 61,49 € / Monat zu Buche schlägt. Nach Starten des Clients läuft dieser als Daemon im Hintergrund.



Abbildung 5.5: Der laufende Daemon von Adobe Edge Inspect

Als zweiten Schritt benötigen wir die zugehörige Chrome Extension von Adobe Edge Inspect. Diese wird über den Chrome Appstore installiert und kann nach einem Browserneustart aktiviert werden.

Als letztes benötigen wir noch die kostenlos erhältliche App aus dem jeweiligen Shop. Hier gilt für Android der Play Store und für iOS Geräte der AppStore. Windowsgeräte werden derzeit nicht unterstützt.

Sind diese drei Schritte erfolgreich durchgeführt worden, müssen nun die Geräte mit dem Server verbunden werden. Hierzu wird die App gestartet (der folgende Prozess verläuft unter Android wie auch unter iOS identisch) und per IP-Adresse mit der Adobe Edge Inspect Chrome Extension verbunden. Diese verlangt im Gegenzug einen Identifikationscode, welcher auf dem jeweiligen Gerät generiert wurde. Nach erfolgreicher Synchronisation wird das Gerät im Gerätemanager angezeigt.

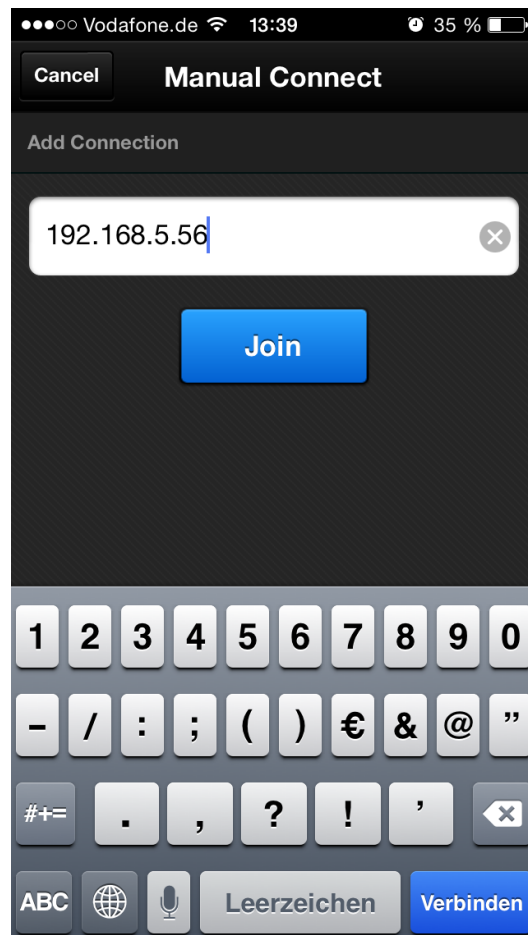


Abbildung 5.6: Eingabe der IP-Adresse zum Edge Inspect Rechner

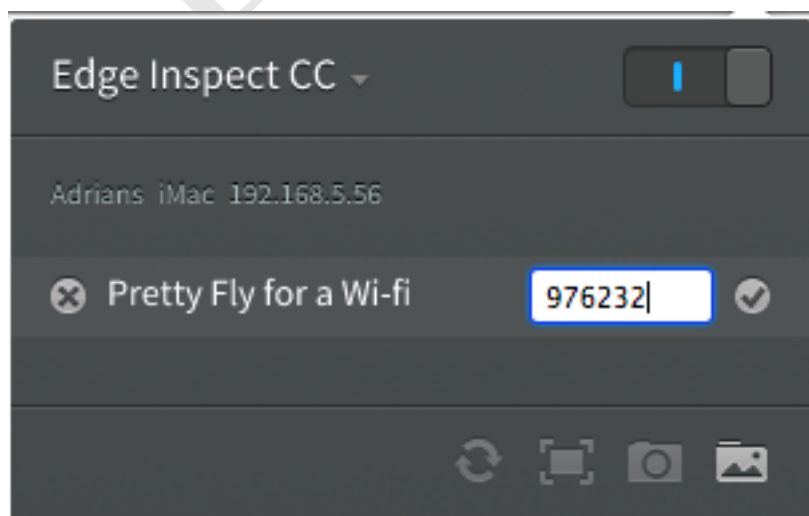


Abbildung 5.7: Eingabe des Sicherheitscodes in die Chrome Extension

Dieser hat mehrere Funktionen. Er liefert eine Übersicht aller verbundenen Clients und ermöglicht das Aufrufen von weinre um z. B. das DOM oder verwendete Ressourcen zu inspizieren oder Javascript auszuführen. Über den Gerätemanager lassen sich auch verbundene Geräte wieder durch einen Klick entfernen. Des Weiteren kann man über dieses Interface Screenshot von allen verbundenen Geräten im aktuellen Zustand aufnehmen und anzeigen lassen. Weiterhin besteht die Möglichkeit den Darstellungsmodus auf den verbundenen Clients von der Appdarstellung auf Vollbild zu ändern.

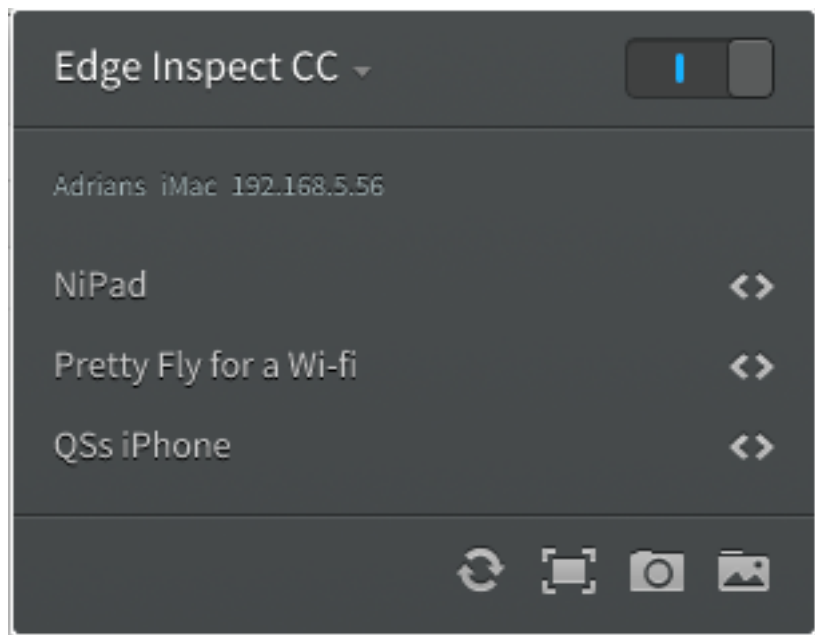


Abbildung 5.8: Übersicht der verbundenen Clients

5.3.2 Testen von Desktopbrowsern

Es gibt zum Zeitpunkt der Erstellung dieser Arbeit keine Möglichkeiten Desktopseiten mit Adobe Edge Inspect zu testen.

5.3.3 Testen von mobilen Browsern

Die Funktionalität zum Testen von mobilen Seiten beschränkt sich derzeit auf den synchronen Aufruf von Seiten über den Chromebrowser, mit installierter Extension als Steuergerät. Die verbundenen Geräte erkennen den Aufruf von Links und das Wechseln von Tabs innerhalb des Browsers. Es besteht wie bereits beschrieben die Option die einzelnen Clients per weinre zu untersuchen.

Die Simulation eines Scrollevents oder das Ausfüllen eines Formulars ist nicht möglich. Es werden lediglich die Informationen dargestellt, die am Steuergerät

aufgerufen wurden. Jedoch wird der Client, sofern vorhanden, auf die mobile Seite weitergeleitet. Während des Testens in der App, wird das Display aktiv gehalten, wodurch es sich nicht von selbst abschaltet. Ein gutes Feature von Adobe Edge Inspect ist die Möglichkeit aus dem Gerätemanager des Browsers Screenshots der verbundenen Geräte anzufordern. Diese werden zusammen mit einer Beschreibung des Gerätes, dessen Modellbezeichnung, der Auflösung sowie Pixeldichte, dem Betriebssystem, der aufgerufenen URL, sowie der aktuellen Ausrichtung des Bildschirms ausgeliefert.

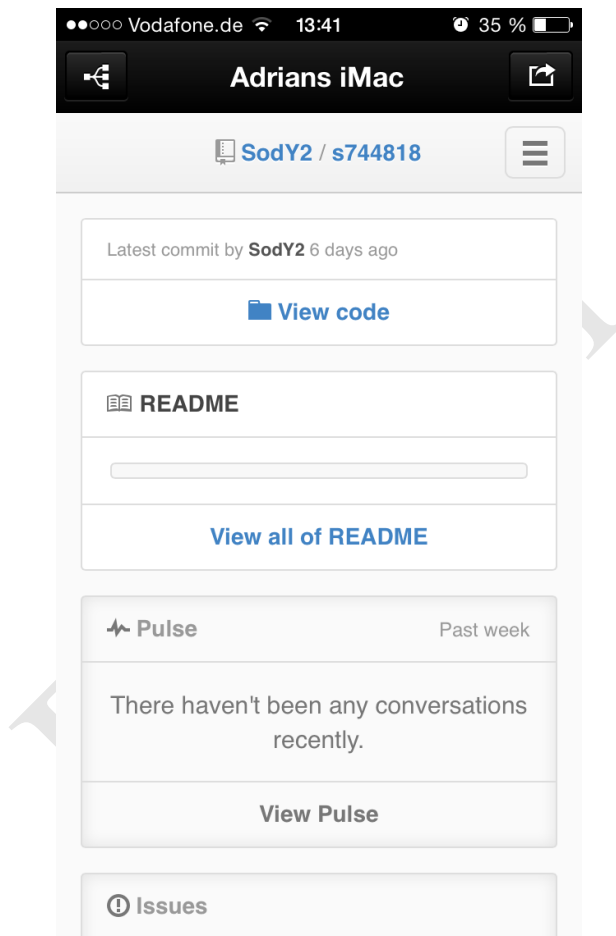


Abbildung 5.9: Darstellung von Content in der Adobe Edge Inspect App unter iOS

Während meiner Versuche ist mir aufgefallen, dass Adobe Edge Inspect unter iOS 6.1.3, Seiten die durch htaccess gesichert sind, nicht darstellen kann. Auf den anderen Testgeräten verlief der Prozess der Authentifizierung problemlos.

5.3.4 Fazit zu Adobe Edge Inspect

Adobe Edge Inspect bedarf viel Aufwand für ein unzureichendes Ergebnis. Man muss an drei verschiedenen Punkten Installationen vornehmen, die dann jedoch ohne Probleme miteinander harmonisieren. Als besonders positiv möchte ich die Screenshotfunktion bewerten. In Zusammenspiel mit der öffentlich zugänglichen API lassen sich hierüber Screenshots im Landschafts-, als auch im Portraitmodus anfordern und durch eine externe Applikationen auswerten.

Der Nutzen des Werkzeuges liegt am ehesten bei One-Page-Sites⁸ oder für Fehler-suche innerhalb des DOM oder CSS Anpassungen mit weinre. Unter dem Aspekt des parallel-synchronen Testens ist Adobe Edge Inspect nicht sinnvoll zu verwenden, da weder Steuerbefehle oder andere Gesten umgesetzt werden, noch werden die Nutzereingaben in Eingabefeldern mit anderen verbundenen Clients geteilt. Alle verbundenen Clients sind nur Empfänger und besitzen keine Möglichkeit als Sender zu fungieren. Folglich gehen alle Steuerbefehle vom Edge-Server aus.

5.3.5 Tabellarische Evaluation

Komponente	Punkte	Wertigkeit
Installation	10	10 %
Konfiguration	7	10 %
Funktion: Desktop	0	25 %
Funktion: Mobil	4	25 %
Erweiterbarkeit	8	10 %
unterstützte Browser	3	10 %
Aktivität	10	10 %
Gesamt	48	100 %

Tabelle 5.9: Gewichtungstabelle Evaluation von Adobe Edge Inspect

⁸Webseiten dessen Inhalt sich füllend auf die gesamte Seite erstrecken

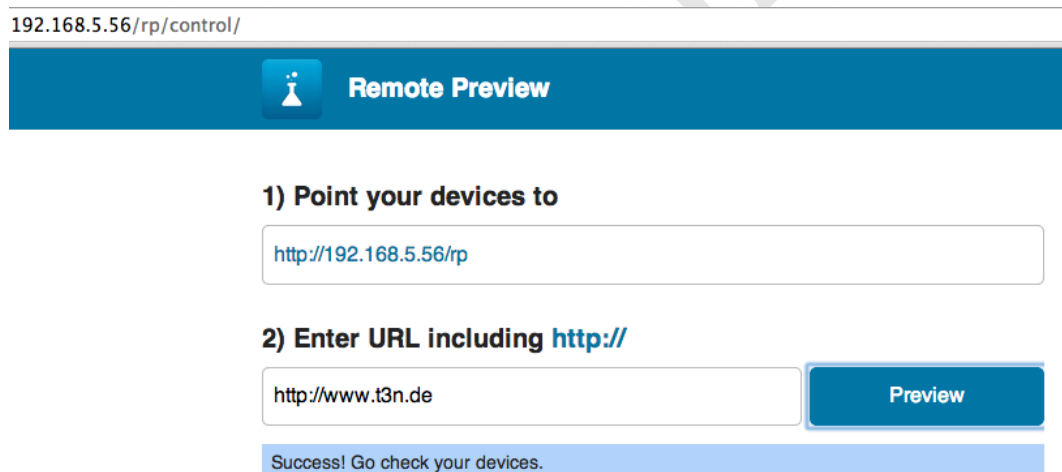
5.4 Remote Preview

5.4.1 Einrichtung der Testumgebung

Es gibt zwei Möglichkeiten dieses Werkzeug zu nutzen. Die eine ist die Installation auf einem lokalen Apache-Server mit PHP. Die andere ist die Installation auf einem Cloud-Dienst wie z. B. Dropbox. Die Ergebnisse dieser Arbeit wurden mit einer lokalen Apache Installation erzielt. Die Installation sieht lediglich vor, das Framework in einen lokalen Entwicklungszweig zu entpacken.

5.4.2 Testen von Desktopseiten

Alle Clients, die in die Testumgebung eingebunden werden sollen, müssen lediglich die IP-Adresse des Servers eingeben. Die Steuerung der Seiten erfolgt sowohl für Desktopseiten, als auch für die mobilen Vertreter über die Browsermaske des Frameworks. In das untere der beiden Eingabefelder, gibt man die aufzurufende URL inklusive Präfix⁹ ein. Diese wird dann auf allen verbundenen Clients innerhalb eines iframes dargestellt.



192.168.5.56/rp/control/

Remote Preview

1) Point your devices to

2) Enter URL including http://

Preview

Success! Go check your devices.

Abbildung 5.10: Steuerungsmaske zur Eingabe der aufzurufenden URL

5.4.3 Testen von mobilen Browsern

Das Testen der mobilen Browser funktioniert parallel zum Testen von Desktopseiten. Positiv möchte ich hier erwähnen, dass das Framework auch wenn es dafür nicht ausgelegt ist, unter aktuellen Windowsgeräten funktioniert.

⁹http://

5.4.4 Fazit zu Remote Preview

Ein positiver Punkt, ist die Möglichkeit, letztendlich jeden Browser unabhängig von dessen Betriebssystem in die Testumgebung zu integrieren, da diese einfach nur auf den Apacheserver oder die Dropbox zugreifen müssen. Als negativ führe ich hier die Tatsache auf, dass es ähnlich Adobe Edge Inspect lediglich dem Aufrufen von Seiten dient, jedoch nicht deren Bedienung. So ist es nicht möglich, weiteren Verlinkungen zu folgen, ohne diese von Hand in die Eingabemaske einzutragen oder Formulare auszufüllen. Das Darstellen von Seiten mit Ankern, funktioniert nur bedingt. Das Aufrufen von gesicherten Seiten gelang nicht. Ebenfalls war es nicht möglich, zertifizierte Webseiten aufzurufen, was den Nutzungsgrad des Frameworks stark einschränkt. Gut finde ich die Tatsache, dass Quellcode komplett zugänglich ist, da er unter der MIT Lizenz steht und jederzeit in eigene Projekte eingebunden oder um eigene Funktionalität erweitert werden kann. Das Projekt scheint zum Zeitpunkt dieser Arbeit nicht weiterentwickelt zu werden. Für den Aufruf einer einfachen Seite auf n-Geräten ist dieses Projekt eine kostenlose Alternative zu Adobe Edge Inspect mit geringerem Funktionsumfang.

5.4.5 Tabellarische Evaluation

Komponente	Punkte	Wertigkeit
Installation	8	10 %
Konfiguration	6	10 %
Funktion: Desktop	3	25 %
Funktion: Mobil	3	25 %
Erweiterbarkeit	8	10 %
unterstützte Browser	10	10 %
Aktivität	0	10 %
Gesamt	47	100 %

Tabelle 5.10: Gewichtungstabelle Evaluation von Remote Preview

5.5 Browser-Sync

5.5.1 Einrichtung der Testumgebung

Als Voraussetzung um Remote-Sync nutzen zu können, wird zu Beginn erst einmal eine node.js Implementation benötigt. Diese kann entweder über die Konsole installiert werden oder per Installationstool von der node.js Homepage.

Nach der node.js Installation wird per NPM das Paket von Browser-Sync per Konsole einmalig installiert:

```
npm install -g browser-sync
```

Abbildung 5.11: Konsolenbefehl um Browser-Sync zu installieren

Nun muss für jedes neue oder bestehende Projekt einmalig im Projektordner Browser-Sync initialisiert werden. Browser-Sync legt in dem aktuellen Verzeichnis eine Konfigurationsdatei ab, in welcher man einzelne Optionen, wie z. B. die zu beobachtenden Dateien oder Einstellungen zum Synchronisationsverhalten, festlegen kann. Dieser Aufruf erfolgt ebenfalls über die Konsole.

```
[3839-adrian@Adrians-iMac:/Library/WebServer/Documents/abschlussarbeit]$ browser-sync init  
[BS] Config file created (bs-config.js)  
[BS] To use it, in the same directory run: browser-sync
```

Abbildung 5.12: Konsolenbefehl um Browser-Sync zu initiieren

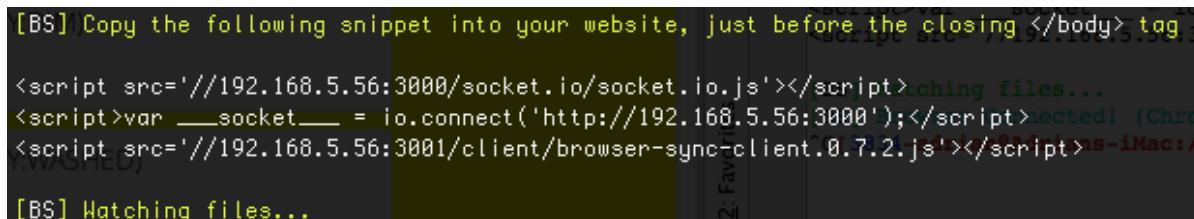
Nach der Initiierung des Servers startet man diesen mit dem Befehl :

```
browser-sync start
```

Abbildung 5.13: Konsolenbefehl um Browser-Sync zu starten

Um nun die Kommunikation zwischen Server und Projekt zu gewährleisten, muss vor dem Ende des Body Elements der Indexdatei zusätzlicher Scriptcode eingefügt

werden, welcher jedoch zum Release entfernt werden sollte. Der einzufügende Code wird anhand der Konfigurationsdatei und der IP-Adresse des Servers generiert und per Konsole dem Nutzer mitgeteilt.



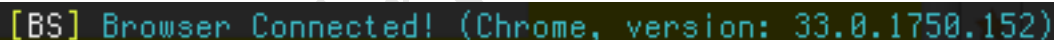
```
[BS] Copy the following snippet into your website, just before the closing </body> tag
<script src='//192.168.5.56:3000/socket.io/socket.io.js'></script><script>watching files...
<script>var ___socket___ = io.connect('http://192.168.5.56:3000');</script><script>connected! (Chrome)
<script src='//192.168.5.56:3001/client/browser-sync-client.0.7.2.js'></script>as-iMac:/...
[BS] Watching files...
```

Abbildung 5.14: Konsolenausgabe mit einzufügendem Quellcode

In künftigen Versionen wird es laut dem Entwickler nicht mehr notwendig sein die Versionsnummer mitanzugeben.

5.5.2 Testen von Desktopseiten

Das Testen erfolgt durch Aufruf der Seite, in die der Steuercode eingetragen wurde, über den Browser. Die parallele Steuerung erfolgt direkt und synchron. Ist ein Browser erfolgreich verbunden, wird dies in der Konsole des Servers angezeigt.



```
[BS] Browser Connected! (Chrome, version: 33.0.1750.152)
```

Abbildung 5.15: Konsolenausgabe bei erfolgreich verbundenem Client

Das Folgen interner Links funktioniert nur unidirektional, sofern der Steuercode nicht mittels Framework oder von Hand in die verlinkten Dateien eingefügt wurde. Das Folgen externer Links erfolgt ebenfalls nur unidirektional. Auch das Aufrufen zertifizierter oder gesicherter Seiten mit Passwordeingabe, funktioniert problemlos. Beim Nutzen von Steuerbefehlen, traten nur bedingt Probleme auf. So gibt es zum Zeitpunkt dieser Arbeit Defizite im Umgang mit dem Javascriptframework jQuery. So lassen sich z. B. Lightboxen öffnen, jedoch werden dann Befehle zum Schließen des Fensters nicht mehr erkannt und übermittelt. Das Ausfüllen von Formularen verlief bis auf eine Mehrfachauswahl fehlerfrei. Das Erkennen von Hoverevents funktionierte in der Version 0.7.2 noch nicht. Auch das parallele Verwenden von Sliderelementen war zu diesem Zeitpunkt noch nicht implementiert.

5.5.3 Testen von mobilen Browsern

Das Testen von mobilen Browsern verläuft parallel zu Desktopseiten. Ein Aufruf über den internen Browser genügt um den Client am Server zu registrieren. Als zusätzliches Problem trat bei den mobilen Geräten ein Verziehen der Elemente auf. Die Geräte richten sich anhand der gescrollten Entfernung aus und nicht der Ausrichtung an der HTML-Struktur. Zum Zeitpunkt dieser Thesis bietet das Framework nicht die Möglichkeit der Ausrichtung an DOM-Elementen der Internetseite. So kommt es bei den Testgeräten zu Unstimmigkeiten in der Darstellung des Inhalts, welche durch die unterschiedlichen Auflösungen und Ausrichtungen der Geräte zu Stande kommen.

5.5.4 Fazit zu Browser-Sync

Browser-Sync ist ein vielversprechendes Framework, welches die zu untersuchenden Aspekte vollkommen abdeckt. Es bestehen noch relativ viele unausgereifte Komponenten, jedoch werden diese bei Auftreten zeitnah von den Entwicklern behoben. Generell scheint das Framework zum Zeitpunkt dieser Arbeit eine hohe Entwicklungsgeschwindigkeit zu besitzen. Es trat gelegentlich ein Fehler auf, bei dem ein verbundener Client, selbst nach mehrfacher Neuverbindung, nicht mehr auf die Steuersignale reagierte. Dieser Fehler trat meistens bei mehr als sechs verbundenen Clients auf. Das Framework ist zum Validieren von Websites gedacht, die sich noch in der Entwicklung befinden. Das Testen ist, aufgrund der notwendigen Testumgebung, nur als lokales Arbeiten angedacht. Als Pluspunkt wird das Injizieren von geänderten Code zur Entwicklungszeit gewertet. So ist es zum Beispiel möglich, vorgenommene Änderungen am Styling oder dem DOM, ohne weitere Handgriffe direkt auf allen Testgeräten zu begutachten.

5.5.5 Tabellarische Evaluation

Komponente	Punkte	Wertigkeit
Installation	5	10 %
Konfiguration	2	10 %
Funktion: Desktop	9	25 %
Funktion: Mobil	7	25 %
Erweiterbarkeit	8	10 %
unterstützte Browser	10	10 %
Aktivität	10	10 %
Gesamt	75	100 %

Tabelle 5.11: Gewichtungstabelle Evaluation von Remote Preview

5.6 Eigenes Framework

Der ursprüngliche Gedanke dieser Arbeit verfolgte den Ansatz ein eigenes Framework zu entwickeln, was die parallel-synchrone Steuerung auf mehreren Endgeräten insbesondere auf mobilen Geräten ermöglicht. Diesen Gedanken berücksichtigend, erfolgte eine Validierung verschiedener Einzeltechnologien, die nur gewisse Aspekte abdecken. Untersucht wurden diese hinsichtlich auf ihre tatsächliche Funktionalität, ihrer Installation und Kombinierbarkeit mit anderen verwendeten Frameworks.

Die Bibliotheken werden insbesondere auf ihre Fähigkeit untersucht, sie in einen node.js Server zu implementieren.

5.6.1 Installation eines node.js Servers

Die Installation des node.js Servers erfolgt einfach über die Konsole unter Mac oder den Installer¹⁰. Alleinstehend erfüllt dieser Server keinerlei der gewünschten Funktionen, jedoch dient dieser als Grundlage für einige nachfolgende Frameworks. node.js ist eine gute Wahl aufgrund der hohen Verarbeitungsgeschwindigkeit, sowohl Client-, als auch Server-seitig. Weitere Pluspunkte sind die rasche Entwicklungsgeschwindigkeit, die hohe Vielfalt an Erweiterungen und Plugins, sowie eine sehr große aktive Entwicklergemeinde.

5.6.2 Einbinden von socket.io

socket.io lässt sich einfach über den NPM installieren. Es ermöglicht das Herstellen einer permanenten Verbindung mit dem Server über einen Socket. Der Vorteil liegt hierbei darin, dass keine zyklischen Anfragen an den Server gesendet werden. Stattdessen wird hier das Observer-Pattern umgesetzt und alle verbundenen Clients werden vom Server informiert, sobald eine Änderung des Status stattgefunden hat.

5.6.3 Generierung von Steuerbefehlen über socket.io

Der generelle Aufbau von socket.io sieht vor, dass der Client sich mit dem Server verbindet und eine permanente Verbindung mit diesem aufrecht erhält. Identifizierbar bleibt diese über eine generierte, einzigartige, alphanumerische Session ID. socket.io funktioniert nach den Observer-Pattern, was bedeutet, dass der Client nicht in zyklischen Abständen Anfragen an den Server sendet, sondern bei einer Änderung der Modelle oder zum Beispiel einem Funktionsaufruf vom Server mittels eines Broadcasts informiert wird. So entsteht das Problem, dass wenn ein Client eine Nachricht an den Server sendet, dieser allen Clients (auch dem Auslöser) diese Nachricht sendet.

¹⁰erhältlich unter Nodejs.org

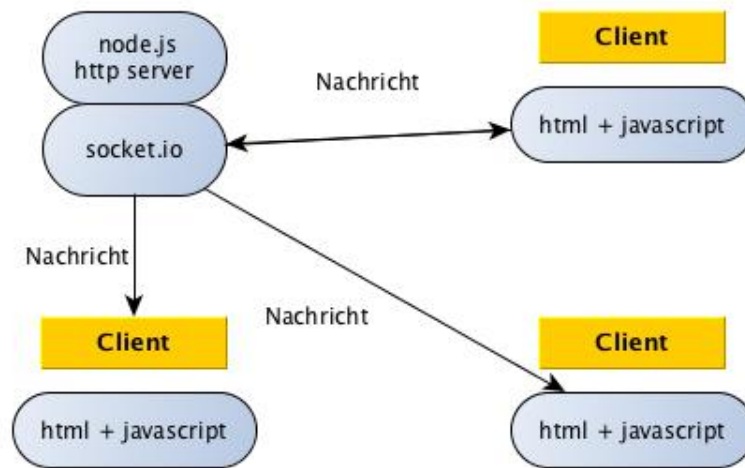


Abbildung 5.16: socket.io Broadcast [ScVi]

Für die Entwicklung eines eigenen Frameworks wirft dies einige Probleme auf.

Ein Beispiel: Ein Nutzer klickt auf einer Seite auf einen Button. Das 'Senden'-Event wird an den Server gesendet und an alle per Socket verbundenen Clients dupliziert. Somit wird der ursprüngliche Sender des Signals, erneut das gleiche Event erhalten. Das Resultat ist, dass dieser den Button zweimalig drückt. Das kann zu Problemen führen, weil beispielsweise eine clientseitige Aktion mehrfach ausgeführt wird. Ein weiteres Problem kann durch rekursives Aufrufen einer Methode einen Dead-Lock erzeugen. Clients, die die empfangenen Signale langsamer als andere verarbeiten, können in dem Moment vom Empfänger direkt wieder zum Sender werden.

Daher ist der Ansatz ein Master-Slave-Pattern umzusetzen denkbar sinnvoll. Es wird ein Steuergerät definiert, welches seine Aktionen dem Server mitteilt und dieser die Events dann an alle verbundenen Clients innerhalb eines Aktionsraumes sendet.

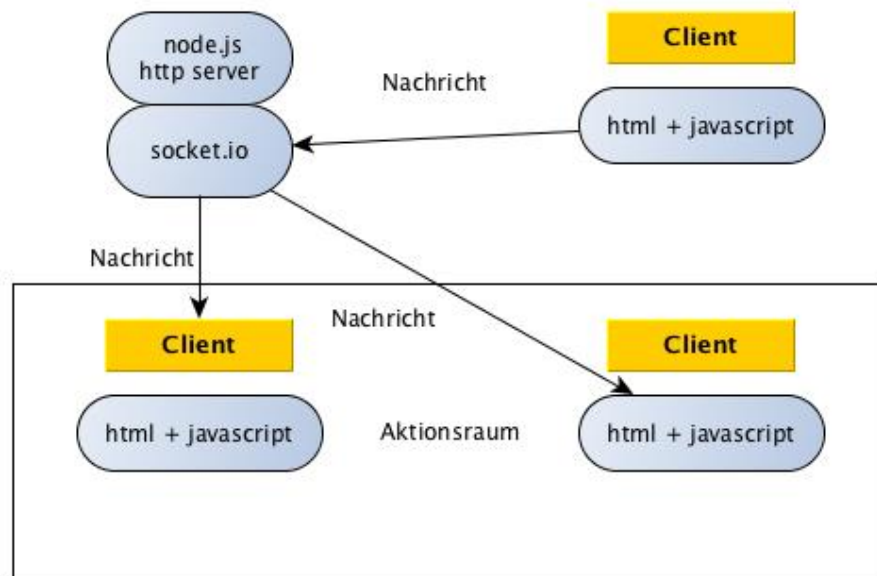


Abbildung 5.17: socket.io Broadcast mit Aktionsraum

5.6.4 Implementierung eines einfachen Broadcast

Das Beispiel soll veranschaulichen, wie ein einfacher Broadcast ohne Aktionsraum implementiert werden kann. Das Beispiel soll das Scrollereignis des Clients abfangen und auf allen verbundenen Clients zur selben Position auf dem Bildschirm scrollen.

Serverseitig

Zu Beginn werden die notwendigen Bibliotheken eingebunden, um einen node.js Server starten zu können (Zeilen: 1-2). Im Anschluss wird eine Serverinstanz von node.js erstellt (Zeile: 4) und gestartet auf Port 8001 (Zeile: 6). Dieser wird nun mit der socket.io Bibliothek verknüpft (Zeile: 7). Sofern nun ein Client sich mit dem Server verbindet, wird das 'connection'-Event gefeuert (Zeile: 9) und der Client wartet auf einen selbst-definierten Methodenaufruf vom 'scroll' (Zeile: 11).

Wenn am Server ein Scrollereignis eingegangen ist, sendet dieser dies per Broadcast an alle verbundenen Clients (Zeile: 12).

```
1  var http = require("http");
2  var io = require('socket.io');
3
4  var server = http.createServer(ph);
5
6  server.listen(8001);
7  var serv_io= io.listen(server);
8
9  serv_io.sockets.on('connection', function(socket){
10     console.log("conntected")
11     socket.on('scroll', function (data) {
12         socket.broadcast.emit('scroll', data);
13     });
14 });
```

Abbildung 5.18: minifizierter Quellcode Serverseitig

Clientseitig

Auf der Seite des Clients müssen zwei Methoden implementiert werden. Zum einen die Methode, die das Scrollevent des Browsers, was hier über jQuery erfolgt, abfängt und über die Socketverbindung die Servermethode 'scroll' aufruft und die aktuelle Scrollposition zum oberen Bildschirmrand übergibt.

```
$(document).on('scroll', function(event){
    socket.emit('scroll', $(document).scrollTop());
});
```

Abbildung 5.19: minifizierter Quellcode Clientseitig

Zum anderen muss die Methode implementiert werden, welche vom Server gesendete Events abfängt und verarbeitet. In diesem Beispiel wartet der Client auf ein Event vom Typ 'scroll'. Dieses bekommt einen Datensatz, die Scrollposition, mitgeliefert. Nach erfolgreichem Eventaufruf wird per jQuery die Position des Bildausschnitts an den des mitgelieferten Datensatzes angepasst.

```
socket.on('scroll', function(data){
    $(document).scrollTop(data)
});
```

Abbildung 5.20: minifizierter Quellcode Clientseitig

5.6.5 Einschätzung zur Umsetzung eines eigenen Frameworks

Der Realisierung eines eigenen Frameworks zur parallel-synchronen Steuerung von Webseiten steht nichts im Wege. Ein positiver Aspekt ist die sehr schnelle Datenübertragung in nahezu Echtzeit mittels node.js. Voraussetzung ist hierfür, dass die Geräte sich im gleichen lokalen Netz befinden. Ein weiterer positiver Aspekt ist die einfache Implementierung von Steuersignalen über socket.io. Die modulare Grundstruktur der Frameworks ermöglicht einen einfachen Einstieg in den Umgang mit dem Framework. So bietet es die Optionen die Standardfunktionen zu nutzen oder aber um eigene Funktionalität zu erweitern.

Die Struktur ermöglicht es sämtliche Events abzufangen, egal ob mit jQuery oder anderen Frameworks zur Eventermittlung, um diese dann in entsprechende Funktionen umzuwandeln und an alle Clients weiterzugeben. Der Einsatz eines Mastergerätes und das Nutzen von Aktionsräumen verhindern die irreführenden Rückkopplungen innerhalb des Nachrichtenzyklus.

Der Einsatz in einer virtuellen Umgebung erfolgt problemlos, da keine weitere Software installiert werden muss. Die Verwendung von socket.io ermöglicht die Unterstützung aller alten Browser Plattformen, da das Framework mit einer Reihe von Fallbacks sich gegen Funktionsverlust absichert. Sollte keine WebSocket-Technologie verfügbar sein, greift das Framework zuerst auf Adobe Flash Sockets zurück und sollten diese auch nicht verfügbar sein, auf eine Reihe verschiedener Long-Polling-Ansätze um die Kommunikation weiterhin zu gewährleisten.

6 Zusammenfassung und Ausblick

Zum Abschluss der Arbeit, werde ich in einem Fazit auf gesetzte Ziele eingehen und die Vorgehensweise der Evaluierung schildern. Im Anschluss wird ein kleiner Ausblick gewährt, was basierend auf dem aktuellen Stand der Framework eventuell verbessert oder noch entwickelt werden könnte.

6.1 Zusammenfassung

Das Ziel dieser Arbeit war die Evaluierung von Techniken zur parallel-synchronen Bedienung einer Web-Applikation auf verschiedenen mobilen Endgeräten. Zu Beginn wurden erst einmal Frameworks ermittelt, welche die gesetzten Kriterien versprechen ganz oder zu großen Teilen abzudecken. Derzeit gibt es nur sehr wenige Anbieter von Produkten für parallele Webseitentests und das Angebot wird durch den Wunsch diese Tests auf mobile Geräte zu erweitern eingeschränkt. Auf Grund dessen wurden auch Segmentframeworks analysiert, welche in Kombination miteinander die Möglichkeit bieten, die geforderten Kriterien zu erfüllen. Im nächsten Schritt wurden diese kurz vorgestellt.

Als nachfolgender Schritt wurde ein Evaluationsschlüssel festgelegt, welcher zum einen Teil aus geforderten Kriterien abgeleitet wurde und zum anderen im Laufe dieser Arbeit um Kriterien erweitert, welche beim Einrichten und Verwenden der einzelnen Werkzeuge als für die Evaluierung wichtig empfunden wurden. Anhand des Schlüssels konnten die Komplettframeworks miteinander, in ihren Stärken und Schwächen, bewertet werden.

Im Anschluss wurden die Frameworks installiert, konfiguriert und mit dem Desktopbrowser sowie mit einer Vielzahl von mobilen Endgeräten getestet, wobei auftretende Fehler oder Lob für eine gute problemlösende Funktion dokumentiert wurden. Am Ende jedes Frameworktestes erfolgte eine tabellarische Evaluierung in welcher Pro und Contra ersichtlich wurde. Außerdem wurde ein Wert anhand der einzelnen Unterkategorien errechnet, welcher in der Gesamtwertung einen Vergleich der Frameworks untereinander ermöglichte.

Das Ziel der Arbeit ist es, sich einen Überblick über die sich am Markt befindlichen Werkzeuge zu verschaffen, die geeignet scheinen, die Arbeit des Qualitätsmanagements hinsichtlich Zeit und Qualität zu optimieren. Dieses Ziel erreichte leider keins der getesteten Frameworks vollends, da sie nie alle Kriterien abdeckten und

somit nach wie vor von Hand nachgetestet werden musste. Die einzelnen Frameworks haben in der Regel jeweils einen Schwerpunkt gut abgedeckt, jedoch dafür andere Aspekte vernachlässigt. Positiv ist das open-source Projekt Browser-Sync aufgefallen. Es erfüllte am besten die gesetzten Kriterien und ist zusätzlich um eigene Funktionalitäten erweiterbar. Zusätzlich basiert Browser-Sync auf node.js was es dem Entwickler ermöglicht auf die umfassende Paketdatenbank von NPM zuzugreifen und sie in das Framework zu implementieren.

Auch das kommerzielle Ghostlab Produkt hat ein sehr solides Grundgerüst, jedoch gibt es gerade im Bereich des mobilen Testens eine Schwachstelle die das Arbeiten, zumindest mit älteren Generationen von Mobilgeräten, fast unmöglich macht. Da es leider keine Möglichkeit bietet das Programm um eigenen Code zu erweitern und dieser derzeit noch über keinen Master-Slave-Modus verfügt, verfängt es sich sehr schnell in einem zyklischen Deadloop.

Abschließend ist zu sagen, dass nach den gesetzten Kriterien, zum Zeitpunkt der Erstellung der Arbeit, keines der Komplettframeworks in der Lage ist den Qualitätssicherungsprozess effizient zu optimieren.

Ich persönlich habe während dieser Arbeit gelernt wie wichtig eine gut strukturierte Planung ist. Diese um Zeitfenster für Eventualitäten zu erweitern und dennoch im Zeitrahmen zu bleiben, war eine große Herausforderung, da Teile dieser Arbeit experimentell waren und daher schwer in Zahlen zu fassen. Das Evaluieren der einzelnen Frameworks lief hingegen überwiegend innerhalb des gesetzten Rahmens. Eine weitere Herausforderung war es die getesteten Frameworks zu vergleichen und dies in Zahlen darzustellen, ohne dabei willkürlich zu wirken. Letzendlich hatte ich auch erhofft ein Framework zu finden, was alle meine Wünsche nach einem verbesserten Workflow erfüllt, jedoch stecken viele der Technologien noch in den Kinderschuhen und einige davon sind aber auf dem richtigen Weg und entwickeln mit Nachdruck daran dieses Ziel auch zu erreichen.

	GL	AEI	RP	B-S
Installation				
Notwendigkeit von anderen Technologien	4	4	4	2
Nutzbar direkt nach der Installation	2	2	2	2
Installationsanleitung vorhanden	2	2	2	1
FAQ vorhanden	2	2	0	0
Konfiguration				
Nutzbar ohne Konfiguration	4	4	4	0
Konfigurierbarkeit (IP und Ports)	1	0	1	1
Konfigurierbare Workspaces	2	0	0	0
Support vorhanden (Wiki, EMail, Forum)	2	2	0	1
Intuitive Benutzeroberfläche	1	1	1	0
Funktion: Desktop				
Darstellung : normale Seiten	2	0	1	2
Darstellung : gesicherte Seiten	2	0	0	2
Darstellung: Reaktionsgeschwindigkeit	2	0	2	2
Funktion: Seitensteuerung	3	0	0	3
Funktion: Javascript	1	0	0	0
Funktion: Mobil				
Darstellung : normale Seiten	1	1	1	1
Darstellung : gesicherte Seiten	0	1	0	1
Darstellung: Reaktionsgeschwindigkeit	2	2	2	2
Funktion: Seitensteuerung	2	0	0	3
Funktion: Javascript	0	0	0	0
Funktion: Gestenkontrolle	0	0	0	0
Erweiterbarkeit				
API Zugang	0	5	5	5
API lizenztechnisch gesichert	0	0	2	2
API Dokumentation	0	3	1	1
Browser Support (aktuelle Versionen)				
mobile Plattformen	3	2	3	3
virtuelle Browser	2	0	2	2
Chrome	1	1	1	1
Opera	1	0	1	1
Firefox	1	0	1	1
Safari	1	0	1	1
Internet Explorer	1	0	1	1
Aktivität				
Noch in der Entwicklung	5	5	0	5
Aktives Forum	0	5	0	5

Tabelle 6.1: Übersicht der Frameworks: GL(Ghostlab), AEI(Adobe Edge Inspect), RP(Remote Preview), B-S(Browser-Sync)

6.1.1 Ausblick

Derzeit besitzt Browser-Sync wohl das höchste Potential, ein Produkt zu liefern, welches die Produktivität in der Web-Applikationsentwicklung steigert. Die lebendige Open-Source-Community ist aktiv und engagiert ein hochwertiges Framework zu erschaffen, welches leichtgewichtig und flexibel einsetzbar ist.

Ein weiterer Kandidat mit hohem Potenzial ist Ghostlab, welches zwar schon gute Allroundansätze derzeit vorweisen kann, jedoch leider im Detail nicht ausgereift ist. Die Nachfrage nach einem qualitativ hochwertigen Produkt ist vorhanden, wie es die große Anzahl¹ an Devicelabs weltweit zeigt.

Auch dem Ansatz ein eigenes Produkt für die spezifischen Anforderungen eines Betriebs zu entwickeln, steht in der Zukunft nichts im Wege, da bereits zu diesem Zeitpunkt node.js, Socket.io und darauf aufbauende Technologien ein fundiertes Grundgerüst liefern. Dieses erfordert zwar eine leicht erhöhte Einarbeitungszeit, jedoch lassen sich hiermit die eigenen Spezifikationen verfolgen und umsetzen. Weiterhin förderlich, ist auch die sehr große und aktive Community rund um die genannten Frameworks, welche auch in Zukunft voraussichtlich viele Einzelaspekte verfolgen, die sich dann einfach über das node.js-eigene Paketverwaltungstool implementieren lassen.

¹Eine große Auswahl findet sich z. B. unter <http://opendevicelab.com/>

Glossar

AJAX

" Ajax (Apronym von engl. Asynchronous JavaScript and XML) bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server. Dieses ermöglicht es, HTTP-Anfragen durchzuführen, während eine HTML-Seite angezeigt wird und die Seite zu verändern, ohne sie komplett neu zu laden " [Wiki01]

. 16

Anker

" Ein Anker bezeichnet eine Sprungmarke innerhalb eines HTML-Dokuments. " [Wiki06]

Diese wird in der Regel durch ein Hashtag (#) gekennzeichnet. 35

Apache

" Der Apache HTTP Server ist ein quelloffenes und freies Produkt der Apache Software Foundation und der meistbenutzte Webserver im Internet. " [Wiki05]

. 11, 18, 23, 34, 35

Apache Lizenz 2

" Die Apache-Lizenz wird von der Free Software Foundation als Lizenz für freie Software anerkannt und ist zu der GNU General Public License Version 3, nicht jedoch zur Version 2, kompatibel. " [Wiki10]

. 18

App

Eine Applikation bezeichnet eine Anwendung oder auch ein Computerprogramm, welche gestellte Probleme beheben soll. vii, 29, 31–33, 50, 51

Auflösung

" Die Bildauflösung ist ein umgangssprachliches Maß für die Bildgröße einer Rastergrafik. Sie wird durch die Gesamtzahl der Bildpunkte oder durch die Anzahl der Spalten (Breite) und Zeilen (Höhe) einer Rastergrafik angegeben. " [Wiki03]

. 32, 38

Browser

Computerprogramm zur Darstellung von Inhalten des World Wide Web. i, vi, 8, 10, 11, 14, 15, 17, 18, 22–24, 26, 27, 29, 31, 32, 34, 35, 46

Checkbox

Eingabefeld zum Anhaken. Besitzt zwei Zustände, wahr und falsch. 27, 51

Cloud

Umgangssprachliche Bezeichnung für eine verteilte IT- bzw. Computerstruktur. 16, 29, 34

Commit

Ein Begriff aus der Versionsverwaltung, welcher eine bestätigte Änderung an einem (Software)projekt bezeichnet. i, 16, 22

Computer

In dieser Arbeit werden gängige Modelle von Personal Computern oder Macs mit einem festen Arbeitsumfeld als Computer bezeichnet. Hierzu zählen auch tragbare Modelle und Laptops. Im Sinne der Thesis umschließe ich nachfolgend mit dem Begriff Desktop oben genannte Komponenten. Dies dient später der Differenzierung ob es sich um ein mobiles Endgerät handelt oder einem Computer. 18, 48, 49, 51, 52

Daemon

Ein Daemon ist ein im Hintergrund laufendes Programm, welches verschiedene Funktionalitäten bereitstellt. vii, 29

Devicelab

Ein physikalischer Ort, an dem es eine Vielzahl an miteinander verbundenen Mobilgeräten und Computern gibt. 47

DOM

" Document Object Model (DOM) ist eine Spezifikation einer Schnittstelle für den Zugriff auf HTML- oder XML-Dokumente. Sie wird vom World Wide Web Consortium definiert. " [Wiki04]

. 15, 17, 24, 25, 31, 33, 38

Endgerät

Komponenten mit primärer mobiler Nutzung werden umfassend als mobile Endgeräte gruppiert. Hierzu zählen Smartphones und Tablets. 2, 4, 5, 8, 10, 12, 16, 19, 26, 39, 44, 52

Event

Übersetzbar als Ereignis. In der Ereignisgesteuerten Programmierung löst ein Event einen Funktionsaufruf aus. 40, 42, 43

Framework

" Ein Framework ist eine semi-vollständige Applikation. Es stellt für Applikationen eine wiederverwendbare, gemeinsame Struktur zur Verfügung. Die Entwickler bauen das Framework in ihre eigene Applikation ein, und erweitern es derart, dass es ihren spezifischen Anforderungen entspricht. Frameworks unterscheiden sich von Toolkits dahingehend, dass sie eine kohärente Struktur zur Verfügung stellen, anstatt einer einfachen Menge von Hilfsklassen. " [RJB88]

Der Einfachheit halber wurden Sammlungen die nach dieser Definition eventuell unter den Begriff eines Toolkits fallen, ebenfalls als Framework betitelt . i, iv–vii, 4, 8–11, 15, 17–21, 24, 34, 35, 37–40, 43–47

gesichert

" Ein digitales Zertifikat ist ein digitaler Datensatz, der bestimmte Eigenschaften von Personen oder Objekten bestätigt und dessen Authentizität und Integrität durch kryptografische Verfahren geprüft werden kann. Das digitale Zertifikat enthält insbesondere die zu seiner Prüfung erforderlichen Daten. " [Wiki07]

. 21, 35, 37, 46

GNU General Public License

" Die GNU General Public License (auch GPL oder GNU GPL) ist die am weitesten verbreitete Software-Lizenz, welche den Endnutzern (Privatpersonen, Organisationen, Firmen) die Freiheiten garantiert, die mit ihr lizenzierte Software nutzen, studieren, verbreiten (kopieren) und ändern zu dürfen. Software, die diese Freiheitsrechte gewährt, wird Freie Software genannt. Die Lizenz wurde ursprünglich von Richard Stallman der Free Software Foundation (FSF) für das GNU-Projekt geschrieben. " [Wiki08]

. 19

Grunt

Ein Javascript-Softwareframework zur automatisierten Job-Verwaltung. Die Software wrd genutzt um zum Beispiel nach jedem erfolgreichen Build, eine Reihe an Test absolvieren zu lassen. 17

htaccess

" .htaccess-Dateien sind Server-Konfigurationsdateien für Verzeichnisse, die zu Ihrem Web-Angebot gehören. So ist die .htaccess-Technik beispielsweise der übliche Weg, um nur bestimmten Benutzern den Zugriff auf bestimmte Daten zu erlauben. " [SeHt01]

. 32

HTML

Die Hypertext Markup Language ist eine Auszeichnungssprache zur Beschreibung von Inhalten. Sie dient der Strukturierung von Texten, Links², Listen und Bildern eines Dokumentes. Eine HTML Seite wird von einem Webbrowser interpretiert und anschließend dargestellt. Die Entwicklung von HTML geschieht durch das World Wide Web Consortium (W3C) und den Web Hypertext Application Technology Working Group (WHATWG) . 38

iframe

HTML-Element welches den Inhalt einer externen Seite anzeigen kann, auch Inlineframe. 34

Input

Verschiedene Eingabefelder in einem HTML-Dokument, zum Beispiel Texteingabe oder Checkboxes. 27

Javascript

" JavaScript (kurz JS) ist eine Skriptsprache, die ursprünglich für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern. Heute findet JavaScript auch außerhalb von Browsern Anwendung, so etwa auf Servern und in Microcontrollern " [Wiki02]

. 15, 16, 18, 21, 24, 27, 31, 37, 46

MIT Lizenz

" Die MIT-Lizenz, auch X-Lizenz oder X11-Lizenz genannt, ist eine aus dem Massachusetts Institute of Technology stammende Lizenz für die Benutzung verschiedener Arten von Computersoftware. Sie erlaubt die Wiederverwendung der unter ihr stehenden Software sowohl für Software, deren Quelltext frei einsehbar ist, als auch für Software, deren Quelltext nicht frei einsehbar ist. " [Wiki09]

²Verweise zu anderen Inhalten

. 17, 18

node.js

Plattform um serverseitige eventgesteuerte Javascriptanwendungen zu entwickeln. v, vi, 17, 18, 36, 39, 41, 43, 45, 47

NPM

Node Packaged Modules, eine Software zur Installation von NodeJS Bibliotheken. 18, 36, 39, 45

parallel-synchron

Simulierte synchrone Bedienung mehrerer Endgeräte innerhalb des gleichen Zeitintervalls. iv, 2, 4–6, 8, 15, 24, 33, 39, 43

PHP

Eine an C und Perl angelehnte Scriptsprache für dynamische Webseiten. 34

Pixel

Auch bekannt als Bildpunkt. Farbwert einer digitalen Rastergrafik. 8, 32

Portraitmodus

Vertikalausrichtung des Bildschirms eines mobilen Endgerätes. 33

Qualitätssicherung

Station, welche ein Produkt (hier die Anwendung) durchlaufen und bestehen muss, um eine gewisse Qualität zu gewährleisten. 2, 4, 5, 7, 45

Radiobox

Eingabefeld für Einfachauswahl von mehreren Möglichkeiten. Besitzt zwei Zustände, wahr und falsch. 27

Subnetz

Simulierte synchrone Bedienung mehrerer Endgeräte innerhalb des gleichen Zeitintervalls. 11

Tablet

Tragbarer flacher Computer mit einem Touchscreen. 2, 49

Test

Ein Test wird im Idealfall vor der Implementierung einer Funktion geschrieben und soll gewährleisten, dass nach jeder Änderung im Programmcode diese Funktion ihren Zweck noch erfüllt. Eine Testsuite ist eine Ansammlung von Tests die nacheinander absolviert werden. 18

Viewport

Der Viewport bezeichnet hier den Sichtbereich eines Bildes. 19

VirtualBox

Virtuelle Desktopumgebung von Oracle. Wird genutzt um zum Beispiel ein anderes Betriebssystem, als das eigentlich genutzte zu emulieren. 27

Workspace

Als Workspace wird eine Arbeitsumgebung im physischen aber auch im virtuellen Bereich bezeichnet. 20, 27, 46

Entwurf

Literaturverzeichnis

[RJBF88] Ralph E. Johnson, Brian Foote: "Designing Reusable Classes" im "Journal of Object-Oriented Programming"(1988)

[Wiki01] [http://de.wikipedia.org/w/index.php?title=Ajax_\(Programmierung\)&oldid=129355492](http://de.wikipedia.org/w/index.php?title=Ajax_(Programmierung)&oldid=129355492)

[ScVi] <http://irlnathan.github.io/sailscasts/blog/2013/10/10/building-a-sails-application-ep21-integrating-socket-dot-io-and-sails-with-custom-controller-actions-using-real-time-model-events/>

[Wiki02] <http://de.wikipedia.org/w/index.php?title=JavaScript&oldid=129548016>

[Wiki03] <http://de.wikipedia.org/w/index.php?title=Bildauf>

[Wiki04] http://de.wikipedia.org/w/index.php?title=Document_Object_Model&oldid=127850631

[Wiki05] http://de.wikipedia.org/w/index.php?title=Apache_HTTP_Server&oldid=129948624

[Wiki06] [http://de.wikipedia.org/w/index.php?title=Anker_\(HTML\)&oldid=124884439](http://de.wikipedia.org/w/index.php?title=Anker_(HTML)&oldid=124884439)

[Wiki07] http://de.wikipedia.org/w/index.php?title=Digitales_Zertifikat&oldid=129208084

[SeHt01] <http://de.selfhtml.org/servercgi/server/htaccess.htm>

[Wiki08] http://de.wikipedia.org/w/index.php?title=GNU_General_Public_License&oldid=1302393

[Wiki09] <http://de.wikipedia.org/w/index.php?title=MIT-Lizenz&oldid=125762366>

[Wiki10] <http://de.wikipedia.org/w/index.php?title=Apache-Lizenz&oldid=130033063>
