



# 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 8. Mai 2014

## **Gutachter**

Prof. Knabe

Beuth Hochschule für Technik

Prof. Dr. 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

Es sollen bestehende Technologien analysiert werden in Bezug auf die Prozessoptimierung innerhalb der Qualitätssicherung im Entstehungszyklus von Webapplikationen, Desweiteren sollen Alleinstehende Frameworks dahingehend untersucht werden auf ihren Nutzfaktor zur Erstellung einer Software die eben diese Anforderung erfüllt.

## Abstract

TOREMOVE→  
english  
<-TOREMOVE

translation

Entwurf

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Aufgabenstellung</b>	<b>5</b>
2.1	Problemstellung . . . . .	5
2.2	Zielsetzung . . . . .	7
2.3	Abgrenzungskriterien . . . . .	7
<b>3</b>	<b>Grundlagen</b>	<b>9</b>
3.1	Begriffsklärung . . . . .	9
3.2	verwendete Hardware . . . . .	10
3.3	verwendete Software . . . . .	12
<b>4</b>	<b>Auswahl der Frameworks</b>	<b>13</b>
4.1	Frameworks als Komplettlösung . . . . .	13
4.1.1	Ghostlab . . . . .	13
4.1.2	Adobe Edge Inspect . . . . .	14
4.1.3	Remote Preview . . . . .	14
4.1.4	Browser-Sync . . . . .	15
4.2	Frameworks als Einzelkomponente . . . . .	16
4.2.1	NodeJS . . . . .	16
4.2.2	NPM socket.io . . . . .	16
4.2.3	Zombie.js . . . . .	16
4.2.4	Phantom Limb . . . . .	16
4.2.5	jQuery UI Touch Punch . . . . .	17
4.2.6	jQuery Touchit . . . . .	17
<b>5</b>	<b>Evaluation der Frameworks</b>	<b>18</b>
5.1	Auflistung des Evaluationsschlüssels . . . . .	18
5.2	Ghostlab Version 1.2.3 . . . . .	21
5.2.1	Einrichtung der Testumgebung . . . . .	21
5.2.2	Testen von Desktopbrowsern . . . . .	21
5.2.3	Testen von mobilen Browsern . . . . .	24
5.2.4	Fazit zu Ghostlab . . . . .	25
5.2.5	Tabellarische Evaluation . . . . .	26
5.3	Adobe Edge Inspect CC . . . . .	27
5.3.1	Einrichtung der Testumgebung . . . . .	27
5.3.2	Testen von Desktopbrowsern . . . . .	29

---

5.3.3	Testen von mobilen Browsern . . . . .	29
5.3.4	Fazit zu Adobe Edge Inspect . . . . .	31
5.3.5	Tabellarische Evaluation . . . . .	31
5.4	Remote Preview . . . . .	32
5.4.1	Einrichtung der Testumgebung . . . . .	32
5.4.2	Testen von Desktopseiten . . . . .	32
5.4.3	Testen von mobilen Browsern . . . . .	32
5.4.4	Fazit zu Remote Preview . . . . .	33
5.4.5	Tabellarische Evaluation . . . . .	33
5.5	Browser-Sync . . . . .	34
5.5.1	Einrichtung der Testumgebung . . . . .	34
5.5.2	Testen von Desktopseiten . . . . .	35
5.5.3	Testen von mobilen Browsern . . . . .	36
5.5.4	Fazit zu Browser-Sync . . . . .	36
5.5.5	Tabellarische Evaluation . . . . .	36
5.6	Eigenes Framework . . . . .	37
5.6.1	Installation eines Node.JS Servers . . . . .	37
5.6.2	Einbinden von socket.io . . . . .	37
5.6.3	Generierung von Steuerbefehlen über socket.io . . . . .	37
5.6.4	Implementierung eines einfachen Broadcast . . . . .	39
5.6.5	Einschätzung zur Umsetzung eines eigenen Frameworks . . . . .	41
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>42</b>
6.1	Zusammenfassung . . . . .	42
6.1.1	Ausblick . . . . .	43
	<b>Glossar</b>	<b>45</b>

# Abbildungsverzeichnis

1.1	Entwicklungsprozess . . . . .	3
2.1	Qualitätssicherung Testszenario . . . . .	6
2.2	Darstellung der Lernkurve für Frameworks . . . . .	7
5.1	Startbildschirm Ghostlab . . . . .	21
5.2	Übersicht Clients . . . . .	22
5.3	Exemplarisch Weinreansicht . . . . .	23
5.4	Übersicht mobile Clients Ghostlab . . . . .	24
5.5	Adobe Edge Inspect Deamon Icon . . . . .	27
5.6	Adobe Edge Inspect App Client hinzufügen . . . . .	28
5.7	Adobe Edge Inspect Chrome Extension . . . . .	28
5.8	Adobe Edge Inspect Gerätemanager . . . . .	29
5.9	Adobe Edge Inspect App Content Darstellung . . . . .	30
5.10	Remote Preview Steuerungsmaske . . . . .	32
5.11	Browser-Sync Installation per Konsole . . . . .	34
5.12	Browser-Sync Initiierung per Konsole . . . . .	34
5.13	Browser-Sync Starten per Konsole . . . . .	34
5.14	Browser-Sync Script-Tag . . . . .	35
5.15	Browser-Sync verbundener Client . . . . .	35
5.16	socket.io vereinfachte Darstellung eines Broadcasts . . . . .	38
5.17	socket.io vereinfachte Darstellung eines Broadcasts mit einem Aktionraum . . . . .	39
5.18	socket.io Quellcode Scrollbeispiel Serverseitig . . . . .	40
5.19	socket.io Quellcode Scrollbeispiel Clientseitig . . . . .	40
5.20	socket.io Quellcode Scrollbeispiel Clientseitig . . . . .	40

---



# Tabellenverzeichnis

3.1	verwendete Hardware . . . . .	10
3.2	Übersicht Nokia Lumina 920 . . . . .	10
3.3	Übersicht LG Nexus 4 . . . . .	10
3.4	Übersicht Apple iPhone4 32 GB . . . . .	11
3.5	Übersicht Apple iPhone5s 16 GB . . . . .	11
3.6	Übersicht Apple iPad mini Wi-Fi 32GB . . . . .	11
3.7	Übersicht Microsoft Surface . . . . .	12
3.8	verwendete virtuelle Maschine . . . . .	12
3.9	verwendete Browser . . . . .	12
4.1	von Ghostlab getestete Browser (Stand 10.03.2014, Version 1.2.3)	13
4.2	von Remote Preview unterstützte Plattformen (stand 19.03.2014, letzter Commit 7dc48caa84) . . . . .	14
4.3	von Browser-Sync getestete Browser (Stand 21.03.2014, Version 0.7.2) . . . . .	15
5.1	Kriterienübersicht: Installation . . . . .	18
5.2	Kriterienübersicht: Konfiguration . . . . .	18
5.3	Kriterienübersicht: Desktop . . . . .	19
5.4	Kriterienübersicht: Mobil . . . . .	19
5.5	Kriterienübersicht: Erweiterbarkeit . . . . .	20
5.6	Kriterienübersicht: Browser . . . . .	20
5.7	Kriterienübersicht: Aktivität . . . . .	20
5.8	Gewichtungstabelle Evaluation von Ghostlab . . . . .	26
5.9	Gewichtungstabelle Evaluation von Adobe Edge Inspect . . . . .	31
5.10	Gewichtungstabelle Evaluation von Remote Preview . . . . .	33
5.11	Gewichtungstabelle Evaluation von Remote Preview . . . . .	36

---

# 1 Einleitung

In der modernen Webentwicklung durchläuft eine Anwendung verschiedene Etappen eines Entwicklungszykluses. Er beginnt bei einem Auftrag oder einer Idee, darauf folgt dann die Spezifikation einzelner Usecases<sup>1</sup>. Im Anschluss folgt in der Regel die Entwicklung und Implementation<sup>2</sup> der einzelnen Komponenten. Am Ende der jeweiligen Implementationsphase durchläuft das Produkt<sup>3</sup> die Qualitätskontrolle. Sollten in diesem Abschnitt 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 Kaskadierungsfehler<sup>4</sup> entstehen. Aus Sicht der Qualitätssicherung ist dies ein lästiges Problem, da diese nach jedem erneuten Modifikationsvorgang eines Softwaresegments einen größeren Segmentblock, wenn nicht sogar das gesamte Softwaresystem erneut testen muss.

Bei der Entwicklung auf und für mobile Endgeräte<sup>5</sup> kommt noch ein erschwerender Faktor hinzu, nämlich die diversen, verschiedenen Bildschirmauflösungen. Diese können nicht nur die Darstellung des Inhaltes beeinflussen, sondern auch daraus folgend die Interaktionskonformität beeinflussen.

Im Optimalfall wird die Software erst nach vollständiger Homogenität auf allen unterstützten Geräten freigegeben.

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, indem sie die Steuerung diverser Geräte parallel-synchron steuern. Die Evaluierung soll zeigen wo die Vorteile und Nachteile der einzelnen Werkzeuge liegen. Weiterhin soll gezeigt werden ob aktuelle Frameworks erweiterbar sind um beispielsweise automatisierte Testunits zu implementieren.

---

<sup>1</sup>Szenario oder auch Anwendungsfall

<sup>2</sup>Einbindung

<sup>3</sup>hier: einzelne Softwarekomponente

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

<sup>5</sup>Smartphones, Tablets oder Ähnliche

---



Abbildung 1.1: Vereinfachte Darstellung eines Softwareentwicklungsprozesses

Im Kapitel der Aufgabenstellung befasse ich mich ausschließlich mit der Ausformulierung der Aufgabenstellung. Ich ermittle welche Kriterien Notwendig sind für die Durchführung der Evaluation und lege feste welche Wertigkeit die einzelnen Faktoren in Bezug auf die Gesamtbewertung erhalten. Ebenfalls lege ich in diesem Kapitel die Abgrenzungskriterien fest, welche dazu dienen die Bearbeitung der Aufgabe innerhalb eines vordefinierten Rahmens zu halten.

In dem darauf folgenden Kapitel kläre ich alle allgemeinen sowie auch technischen Grundlagen, die Notwendig sind diese Abschlussthesis zu verstehen. Ich werde ausführlich auf verwendete Begriffe eingehen, sowie Begriffe die in dessen 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 der Technologien werde ich mich kurz mit den einzelnen Frameworks befassen. Ich erläutere dessen Herkunft, womit sie werben und auf welchen Technologien sie aufbauen. Desweiteren behandle ich in diesem Abschnitt Technologien die einzelne Funktionelle Komponenten sind, welche ich in Hinsicht auf die Entwicklung eines eigenen Frameworks zur parallel-synchronen Steuerung von Webapplikationen auf mobilen Endgeräten auf einen Mehrwert untersuchen werde.

Das Kapitel der Evaluation der Techniken umfasst die Auswertung der erlangten Ergebnisse. Hier werde ich die Resultate meiner Versuchsreihen erläutern und wie man die Ergebnisse nutzen kann, eine optimierte Qualitätssicherung von Webapplikationen, mit dem Fokus auf mobilen Endgeräten, vorzunehmen .

Zum Abschluss werde ich meine Thesis noch einmal zusammenfassen und Fragen klären die während der Bearbeitungszeit auftraten. Probleme die entstanden werden hier erörtert.

## **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 Aufgaben dieser Thesis ist die Evaluierung von Techniken zur parallel-synchronen Steuerung von Webapplikationen auf mobilen Endgeräten, um damit die Produktivität der Qualitätssicherung zu optimieren.

### 2.1 Problemstellung

Ein Problem in der aktuellen Softwareentwicklung ist die immer mehr wachsende Anzahl an Endgeräten, welche mit verschiedenen Bildschirmauflösungen und eigenen Betriebssystemen in unterschiedlichen Versionen auftreten. Ein Qualitätsprüfer der einen hohen Qualitätsstandard hat investiert daher linear zu der Anzahl der zu testenden Geräte ansteigend Zeit, lediglich um vereinzelte Testszenarien durchzuarbeiten. Solch ein Testszenario kann Navigationsabläufe<sup>1</sup>, das ausfüllen und validieren eines Formular oder auch das überprüfen funktionaler<sup>2</sup> Links sein. Bereits an dieser Stelle ist die zu investierende Zeit, und dies wiederholt, enorm. Wenn der

Qualitätsprüfer innerhalb eines Testszenarios einen schwerwiegenden Fehler bei einem der Geräte entdeckt, muss dieser den Vorgang beenden. Abgebrochen werden muss deshalb, da bei korrigierter Implementierung der Qualitätsprüfer nicht davon ausgehen darf, das bereits kontrollierte Abschnitte immernoch voll funktionsfähig sind, da eventuell neue Fehler in bereits Kontrollierten Segmenten auftreten können. Sollte ein Szenario aufgrund eines Fehler 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 entstehen kann, ist der Umstand, dass sich grade bei nur kleineren fixes<sup>3</sup> und immer wieder auftretenden Testszenarioschleifen eine gewisse Routine einschleichen kann, worunter die Qualität des Produkts leidet.

---

<sup>1</sup>ein Nutzerspezifischer Gang durch die Webseite

<sup>2</sup>aktive Links und deren Aufruf

<sup>3</sup>Problemlösungen, Codeanpassungen

---

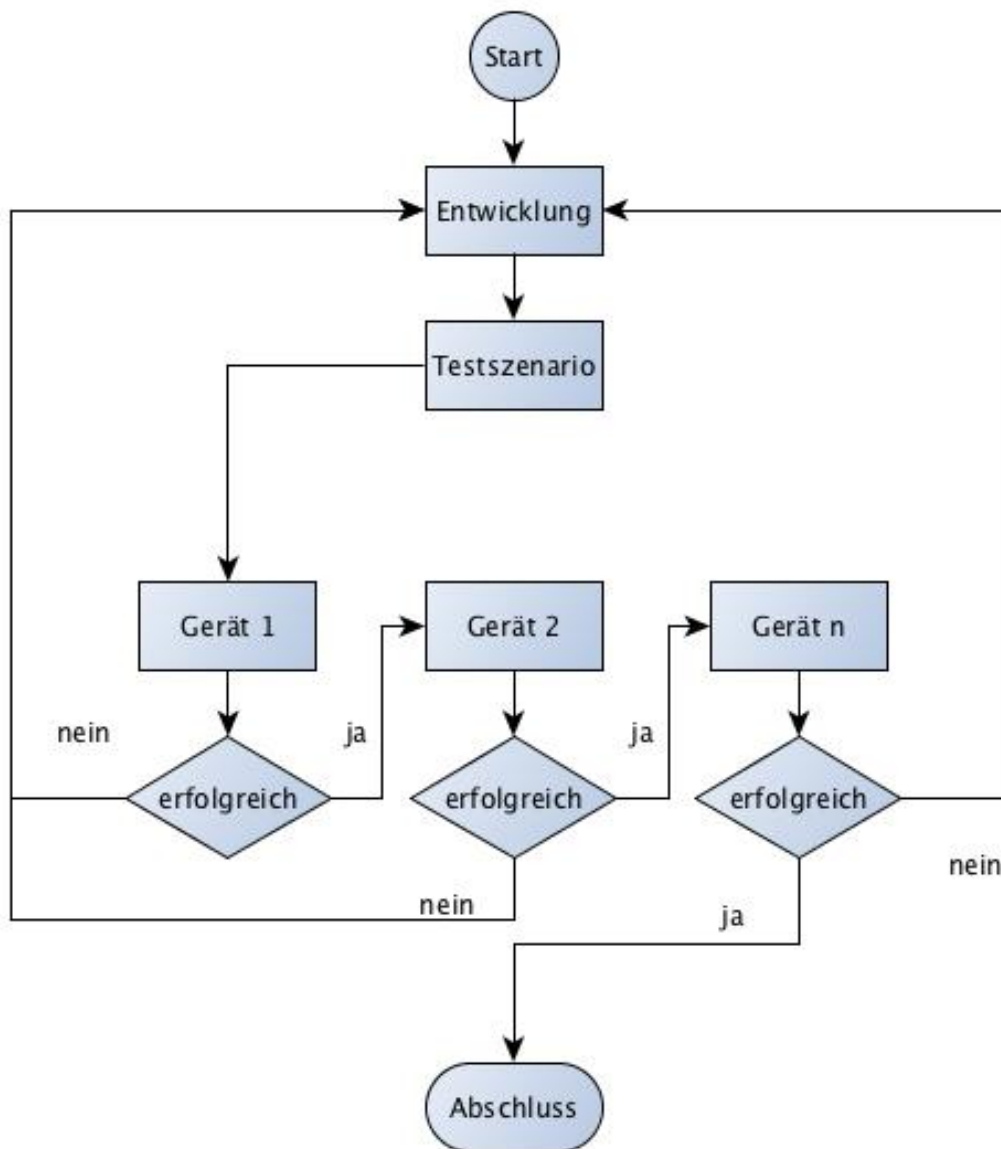


Abbildung 2.1: Darstellung eines Qualitätssicherungsablaufes in der mobilen Anwendungsentwicklung

## 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). 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 eins der wichtigsten Abgrenzungskriterien gilt es die Einarbeitungszeit zu bewerten. Hier gilt je kürzer desto besser, immer gesehen in Relation zu dem Umfang des Frameworks. So ist ein Framework qualitativer zu bewerten, wenn es exponentielle Lernkurve in Relation zur Zeit vorweist.

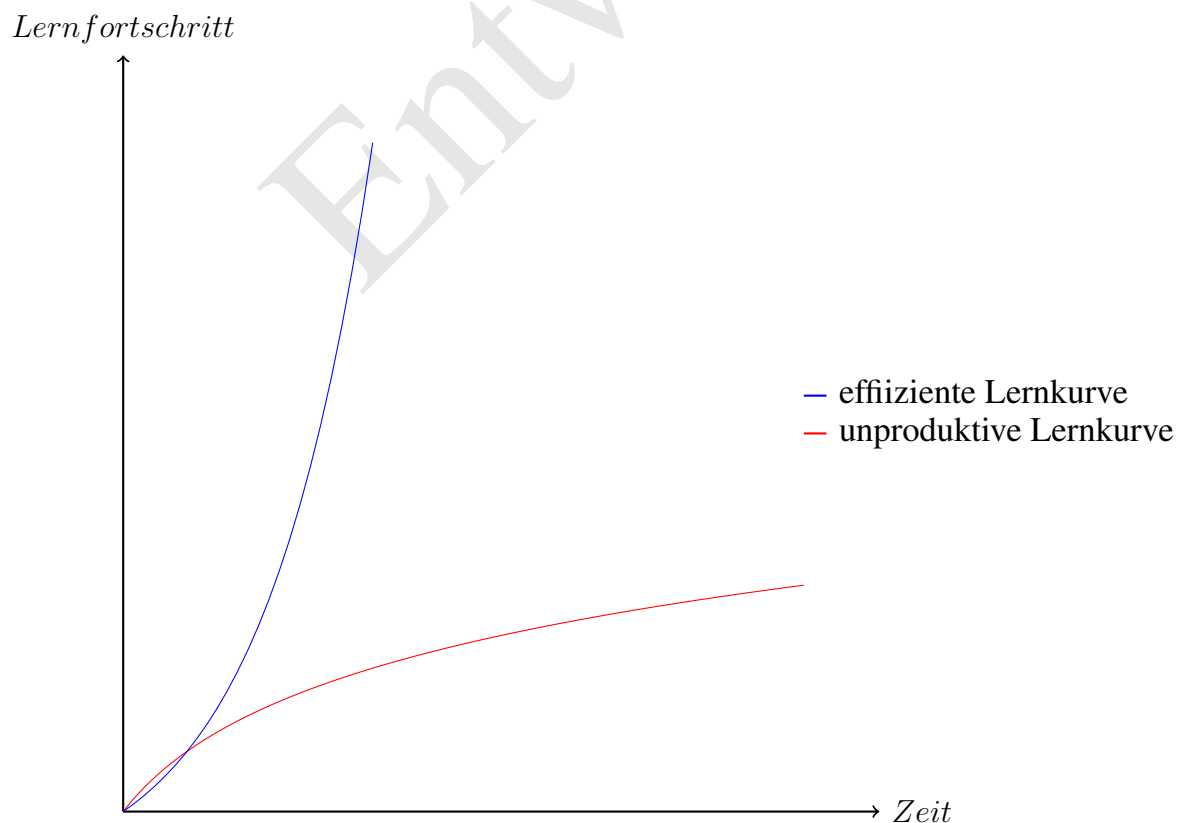


Abbildung 2.2: Lernkurve für Frameworks

### **Erweiterbarkeit**

in Hinsicht auf mehrere Endgeräte gibt es mehrere Aspekte zum analysieren. Zum einen ob das Framework auf verschiedene Arten an System gebunden sind, 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 gewünscht 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.

### **Unterstütze Browser**

Ein wichtiger Aspekt der durchzuführenden Tests wird die unterstützung möglichst verschiedener Browser beinhalten. Dies hat den Grund, dass lediglich ein Framework was eine große Spanne an Endgeräten abdeckt in der Lage ist effektiv genutzt werden zu können.

### **Virtuelle Umgebung**

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

---



# 3 Grundlagen

In diesem Abschnitt behandle ich spezifische Definitionen wie zum Beispiel verwendetes Fachvokabular, allgemeine technische Abläufe die Notwendig sind um diese Arbeit und die darin verwendetet Techniken zu verstehen, sowie verwendete Hardwarekomponenten.

## 3.1 Begriffsklärung

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 .

Ajax Mobiles Endgerät Framework Webbrowser HTML Javascript  
NodeJS PHP NPM Qualitätssicherung VirtualBox  
Smartphone Tablet Portrait View Panorama View Pixel Bildauflösung

---

## 3.2 verwendete Hardware

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

### Apple iMac 27"

Zur Durchführung dieser Arbeit und der darin enthaltenen Evaluationsverfahren wurde 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 Lumina 920

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

Tabelle 3.2: Übersicht Nokia Lumina 920

### LG Nexus 4

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

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
priemäre Ausrichtung	Portrait

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
priemäre Ausrichtung	Portrait

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
priemäre Ausrichtung	Landschaft

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

**Microsoft Surfcae**

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

Tabelle 3.7: Übersicht Microsoft Surfcae

**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 keine weiteren Technologien benötigen um genutzt zu werden.

### 4.1.1 Ghostlab

Ghostlab ist ein Framework des Schweizer Unternehmens Vanamco. Es verspricht das synchrone Testen von Websites in Echtzeit. Weiterhin wirbt das Unternehmen mit einem umfangreichen Repertoire an nützlichen Fähigkeiten. Der Funktionsumfang umschliesst das Scrollen innerhalb einer Seite, das ausfüllen von Formularen, das wahrnehmen und reproduzieren von Click-Events sowie dem neuladen einer Seite. Ghostlab soll ebenso einen Inspektor besitzen, welcher die Analyse des DOMs, der on the fly Bearbeitung der CSS und der Analyse und Bearbeitung von Javascriptdateien. 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)

Der Kostenpunkt der Lizenz liegt zur Erstellung dieser Arbeit bei 49\$ (entspricht

---

35,30€ beim aktuellen Umrechnungswert). 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 CC<sup>1</sup> Version vertrieben. Um Adobe Edge Inspect nutzen zu können bedarf es 3 separaten Komponenten. Adobe wirbt mit synchronem aufrufen und auffrischen von Websites, sowie deren Inspektion per Weinre. 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 Produkte<sup>2</sup> 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 Javascript Framework von dem Web Designer Viljami Salminen aus Helsinki, Finnland. Es überprüft alle 1100ms per AJAX-Request ob sich die Quellurl geändert hat und teilt dies dann den verbundenen Testgeräten mit. Er wirbt mit dem synchronen Aufruf von Websites 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)

---

<sup>1</sup>Creative Cloud

<sup>2</sup>zum 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 den Ansprüchen des Themas gerecht werden. Es wirbt mit synchronisierter Steuerung, dem Entwickeln an Stiles 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 den 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 NodeJS 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 befindend sich zum Zeitpunkt dieser Arbeit in der Version 0.7.2 und wird täglich weiterentwickelt.

## 4.2 Frameworks als Einzelkomponente

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

### 4.2.1 NodeJS

Node.JS Aufgabe besteht darin, anstelle von zum Beispiel Apache, einen Webserver zur Verfügung zu stellen, welcher nur auf Javascript basiert. Alle Notwendigen serverseitigen Anfragen und Funktionen erfolgen in Javascript. Entwickelt wird Node.JS von der Californischen 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](http://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. Der Gedanke der Technologie dahinter verfolgt den Gedanken 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 Entwicklern<sup>3</sup>, welches von dem in Californien sitzenden 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 headless<sup>4</sup> Browser. Dies hat zur Folge das natürlich nur non- visuelle 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 umwandelt. Das Framework läuft unter der Apache Lizenz 2 und kann

---

<sup>3</sup><https://github.com/assaf/zombie/graphs/contributors>

<sup>4</sup>Kopflos - ohne Gerüst das ihn umschliesst, oder auch virtuell

---



kostenlos verwendet werden. Es kam in die Auswahl der Frameworks, da es von Nutzen ist das Steuergerät für die Tests an einem Computer zu verwenden.

### 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. Diese erlaubt von Hause aus nicht die Nutzung von Touch Events auf mobilen Endgeräten. Die Erweiterung hebt 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 GPL Lizenz, wodurch es dem Endnutzer freisteht 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. Desweiteren 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

Um die einzelnen Frameworks zu Evaluieren habe ich einen Schlüssel aufgelistet, welcher messbare Aspekte abdeckt, die ich im Vorfeld der Arbeit bereits als wichtig erachtet habe um die aufgestellte Thesis in Zahlen darzustellen. Ergänzend kamen Punkte hinzu die bei der Installation und der Nutzung auffielen und sich als wichtig erwiesen.

Ich habe den Schlüssel in 7 Hauptkategorien aufgeteilt. Die Abschnitte Installation und Konfiguration befassen sich in erster Linie mit der Verfügbarkeit, dem Zugang zu dem Framework, deren Installation und Dokumentation sowie der Voraussetzung andere 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. Das hat den Grund, dass das testen von mobilen Seiten andere Schwerpunkte der Bewertung haben sollte, als das testen von Desktopseiten. So ist eine funktionierende synchrone Gestenkontrolle beim testen 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, dessen Lizenz und Dokumentation bewertet.

**Erweiterbarkeit**

Kriterium	Punktezahl
API Zugang	5
API lizenzteschnich 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 das 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 localhost<sup>1</sup> auf Port 8005 (Default) von allen zu testenden Geräten erreichbar.

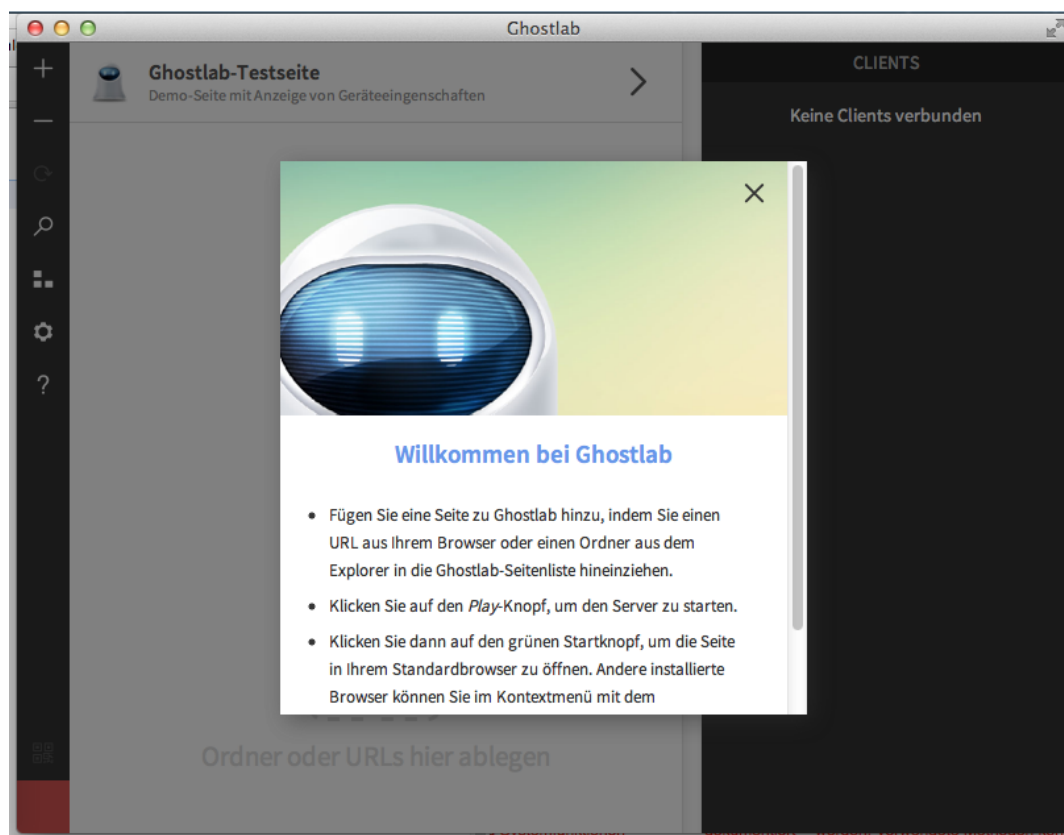


Abbildung 5.1: Startbildschirm von Ghostlab nach der Installation

### 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

---

<sup>1</sup>IP-Adresse des lokalen Rechners

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

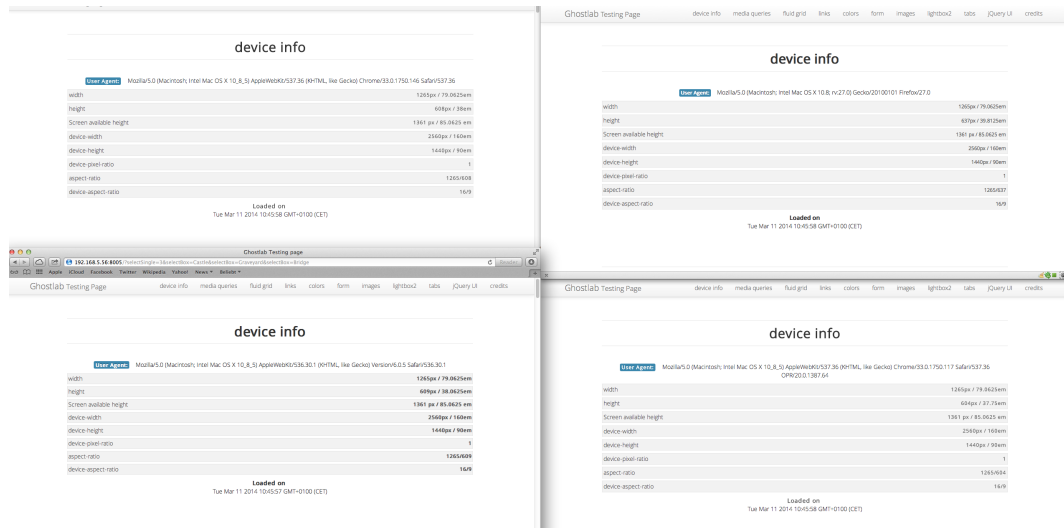


Abbildung 5.2: Darstellung von 4 verschiedenen Clients

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.

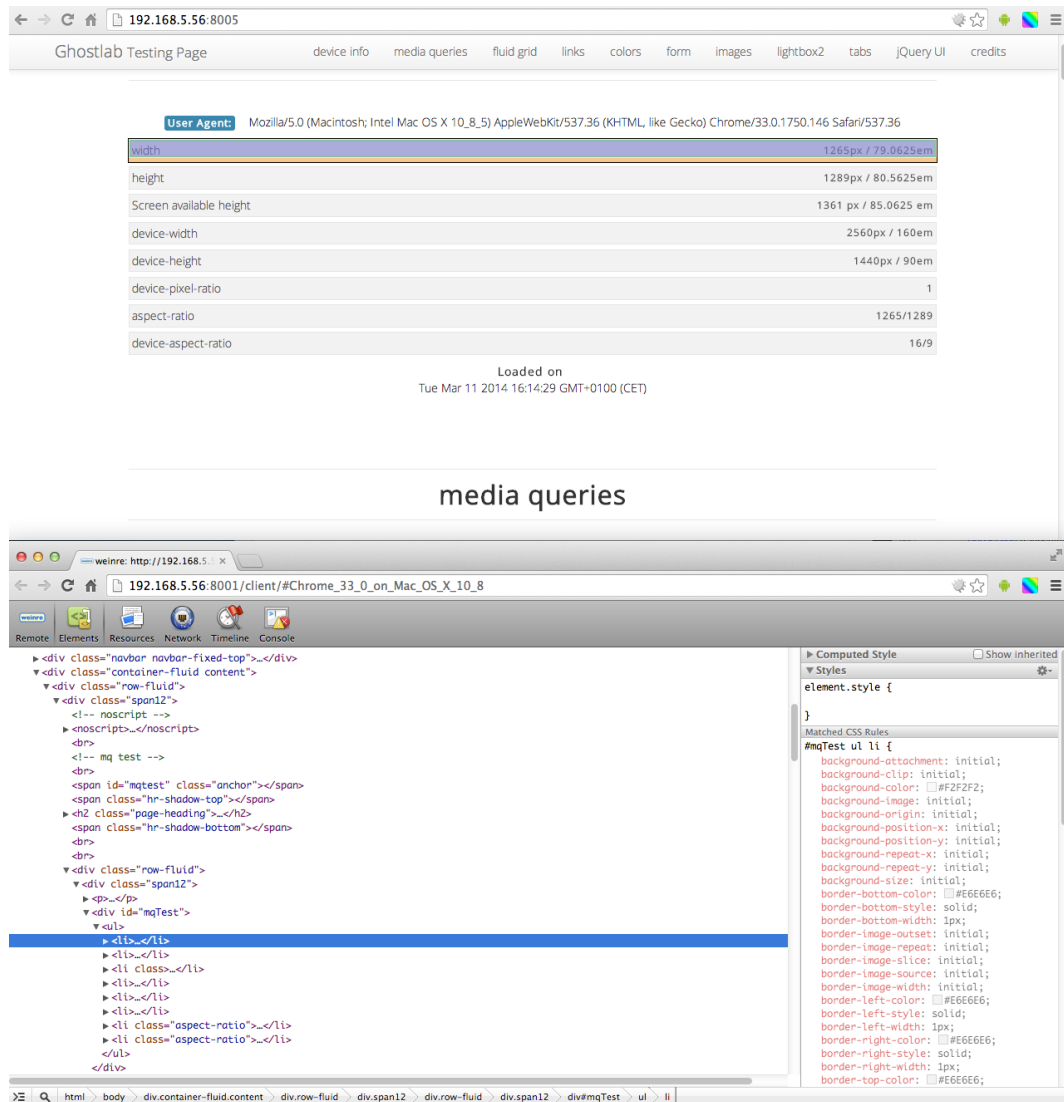


Abbildung 5.3: ausgewähltes DOM-Element in Weinre

### 5.2.3 Testen von mobilen Browsern

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 Sekunden<sup>2</sup> in der Clientliste aufgenommen.

Bei dem Testen auf mobilen Browsern ist es bei Ghostlab<sup>3</sup> 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 Letzen Signals gleichzeitig wieder zum Sender für alle anderen Geräte wird.

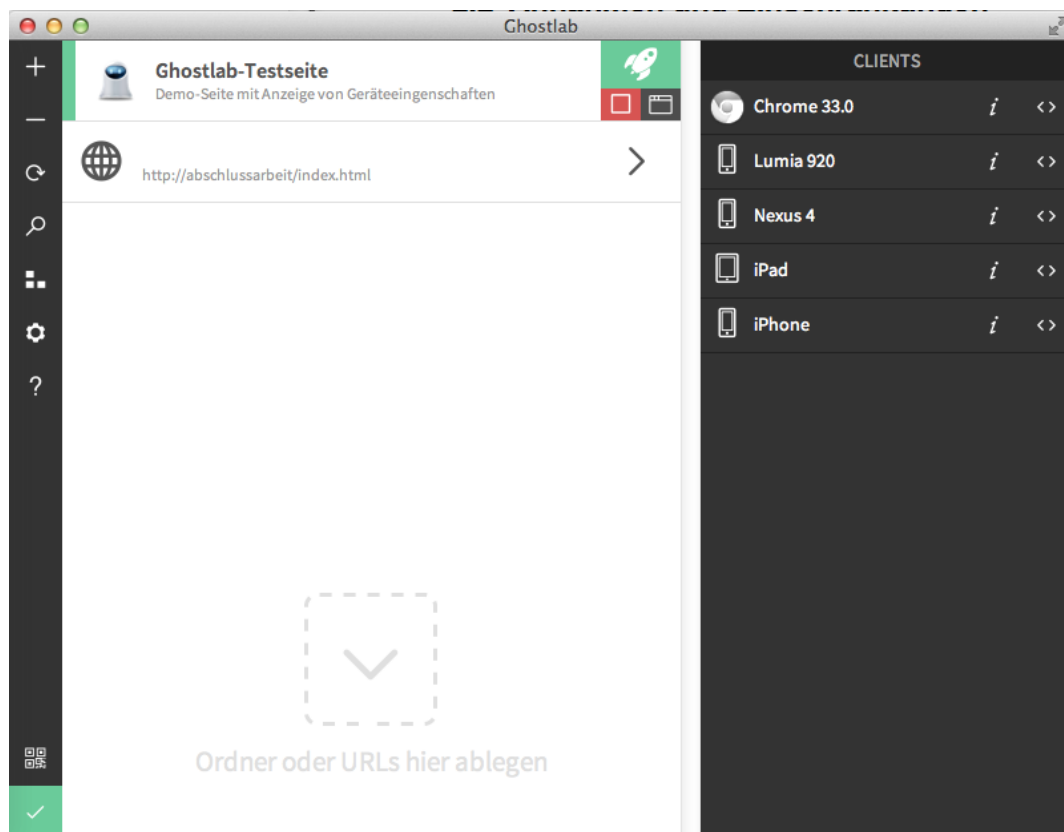


Abbildung 5.4: Ghostlabübersicht der verbundenen Clients

<sup>2</sup>abhängig von der Geschwindigkeit des Testgerätes

<sup>3</sup>Version 1.2.3



### 5.2.4 Fazit zu Ghostlab

Zum Stand dieser Arbeit wurde Version 1.2.3 von Ghostlab genutzt. Zu diesem Zeitpunkt verfügte die Software noch über keinen Master/Slave-Modus<sup>4</sup>, 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 sehe ich bei einer bereits kleinen Anzahl von Geräten als kritisch an.

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 erneut.

Das arbeiten in einer Virtuellen Umgebung<sup>5</sup> 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 Workspaces<sup>6</sup>, welche sich die Position und Größe der verschiedenen Browserfenster speichert. Per Knopfdruck lassen diese sich dann im Kollektiv öffnen sofern in den Browsereinstellungen die Popups aktiviert sind für die zu testende Seite. Dieses Feature<sup>7</sup> bewerte ich als Positiv in Hinsicht der Zeitersparnis, diesen Vorgang immer wieder von Hand auszuführen.

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

---

<sup>4</sup>ein Gerät dient als Steuergerät, alle anderen folgen ihm

<sup>5</sup>es wurde VirtualBox von Oracle genutzt

<sup>6</sup>Arbeitsumgebung oder auch Arbeitsumfeld

<sup>7</sup>Funktion 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 3 Schritte notwendig Adobe Edge Inspect zum Einsatz bereit zu machen. 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 kostenlose 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. Dieser wird gestartet und läuft ab diesem Zeitpunkt 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, für iOS Geräte der AppStore. Windowsgeräte werden derzeit nicht unterstützt.

Sind diese 3 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 werden. Diese verlangt im Gegenzug einen Identifikationscode, welcher auf dem jeweiligen Gerät generiert wurde. Nach erfolgreicher Synchronisation wird das Gerät im Gerätemanager angezeigt.

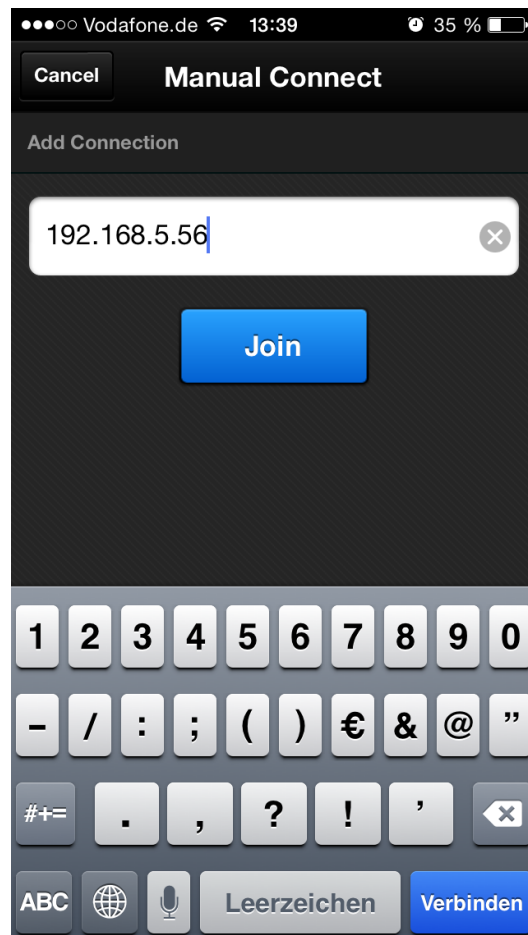


Abbildung 5.6: Eingabe der IP-Adresse zum Edge Inspect Rechner

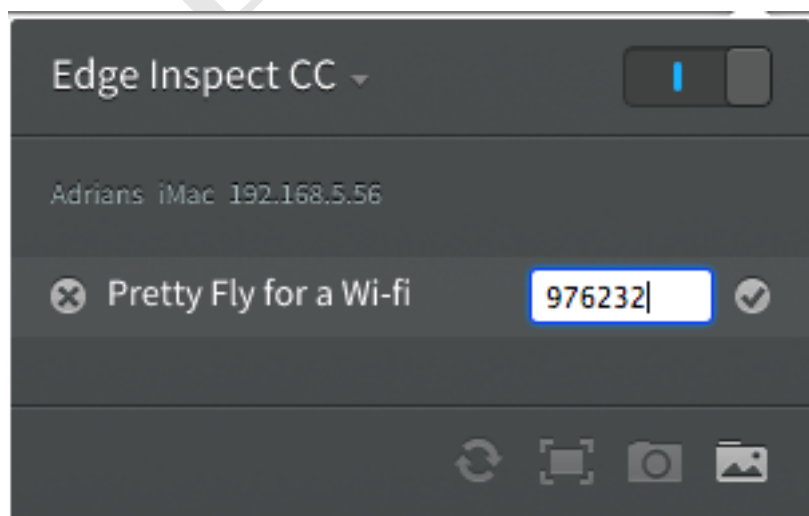


Abbildung 5.7: Eingabe des Sicherheitscodes in die Chrome Extension

Dieser hat mehrere Funktionen. Er liefert eine Übersicht aller verbundenen Clients und ermöglicht das aufrufen von Weinre um z.B. das DOM zu inspizieren, verwendete Ressourcen zu inspizieren oder Javascript auszuführen. Über den Gerätemanager lassen sich auch verbundene Geräte wieder durch einen Klick entfernen. Desweiteren 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 in den Vollbildmodus zu wechseln.

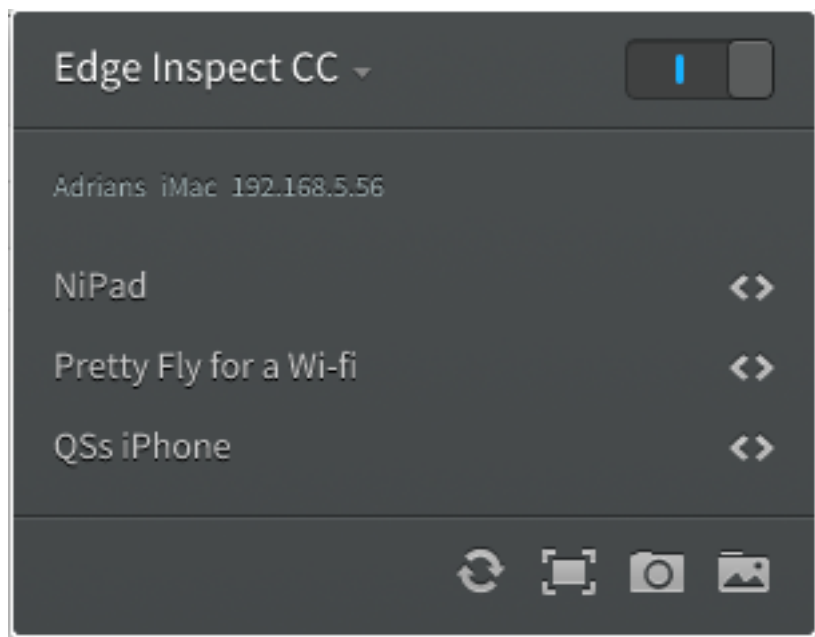


Abbildung 5.8: Übersicht der verbundenen Clients

### 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 nur 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 simulierung 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 Geräts, dessen Modellbezeichnung, die Auflösung sowie Pixeldichte, dem Betriebssystem, der aufgerufenen URL sowie der aktuellen Ausrichtung des Bildschirms ausgeliefert.

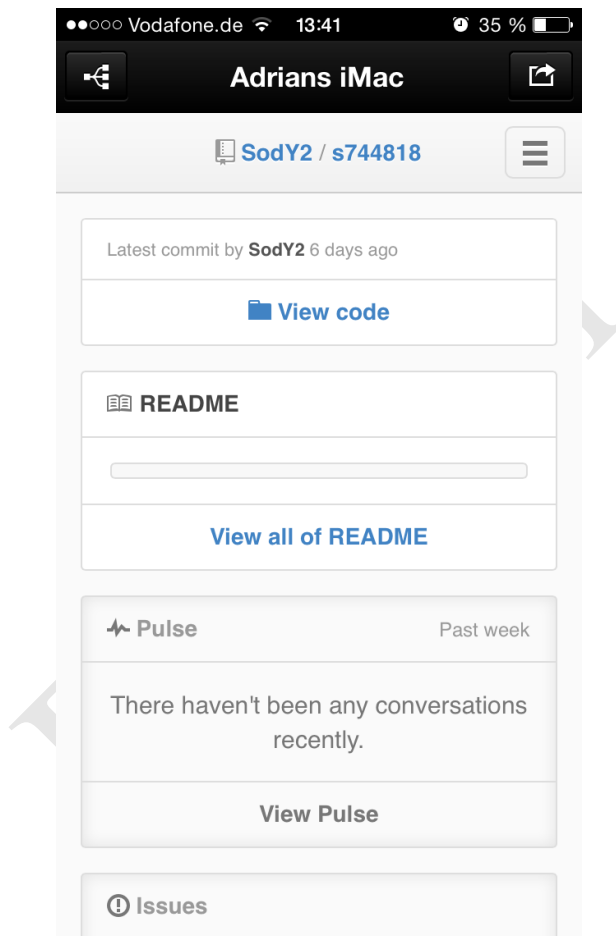


Abbildung 5.9: Darstellung von Content in der Adobe Edge Inspect App unter iOS

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 relativ geringes Ergebnis. Man muss an 3 verschiedenen Punkten Installationen vornehmen, die dann jedoch ohne Probleme miteinander harmonisiert haben. 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 Werkzeugs liegt am ehesten bei One-Page-Sites<sup>8</sup> oder für Fehlersuche innerhalb des DOM oder CSS Anpassungen mit Weinre. Unter dem Aspekt des parallel-synchronen Testens ist Adobe Edge Inspect leider 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

<sup>8</sup>Webseiten dessen Inhalt sich füllend auf die gesamte Seite erstrecken

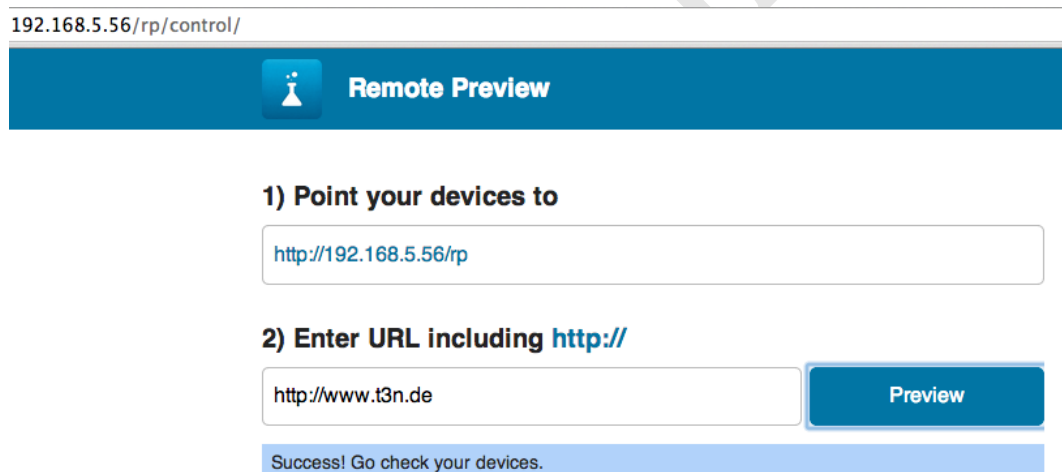
## 5.4 Remote Preview

### 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 hab ich mit der 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 Präfix<sup>9</sup> 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, dennoch unter aktuellen Windowsgeräten funktioniert.

---

<sup>9</sup>http://



### 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 das es ähnlich Adobe Edge Inspect lediglich dem Aufruf von Seiten dient, jedoch nicht dessen Bedienung. So ist es nicht möglich weiteren Verlinkungen zu folgen ohne diese von Hand in der Eingabemaske einzutragen oder Formulare auszufüllen. Bedingt funktioniert das Darstellen von Seiten mit Ankern. Das Aufrufen von gesicherten Seiten gelang mir nicht. Ebenfalls war es mir nicht möglich zertifizierte Webseiten aufzurufen, was den Nutzungsgrad des Frameworks stark einschränkt. Gut finde ich die Tatsache das Quellcode komplett zugänglich ist, da er unter der MIT Lizenz steht und jederzeit in eigene Projekte eingebunden oder um eigene Funktionalität erweitert werden kann. Somit kann man Remote Preview nutzen und es um erweiterte Funktionalität erweitern kann. Das Projekt scheint zum Zeitpunkt dieser Arbeit nicht weiter entwickelt 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

Um Remote-Sync nutzen zu können wird zu Beginn erst einmal eine NodeJS Implementation benötigt. Diese kann entweder über die Konsole installiert werden oder per Installationstool von der NodeJS Homepage.

Nach der NodeJS 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 die zu beobachtenden Dateien oder Einstellungen zum Synchronisationsverhalten. Dies geschieht 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 :

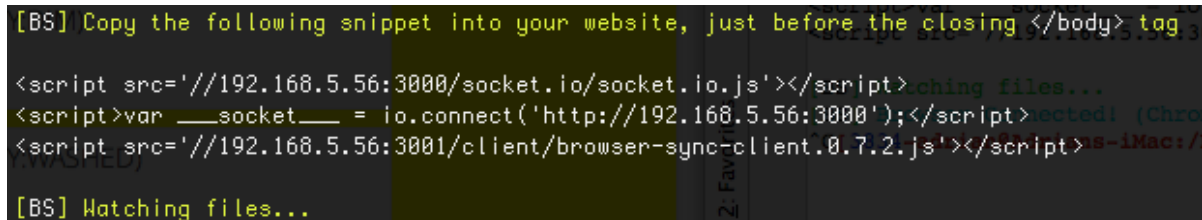
```
browser-sync start
```

Abbildung 5.13: Konsolenbefehl um Browser-Sync zu starten

Um nun die Kommunikation zwischen dem Server und dem 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>var ___socket___ = io.connect('http://192.168.5.56:3000');</script>
<script src='//192.168.5.56:3001/client/browser-sync-client.0.7.2.js'></script>

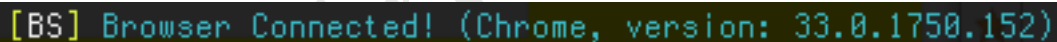
[BS] Watching files...
```

Abbildung 5.14: Konsolenausgabe mit einzufügendem Quellcode

In künftigen Versionen wird es laut dem Entwickler nicht mehr Notwendig sein die Versionsnummer mit anzugeben.

### 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 von internen Links funktioniert nur Unidirektional, sofern der Steuercode nicht mittels Framework oder von Hand in die verlinkten Dateien eingefügt wurde. Das folgen externer Links erfolgt 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 Verzug an Elementen auf. Die Geräte richten sich anhand der gescrollten Entfernung aus und bieten zum Zeitpunkt dieser Thesis nicht die Möglichkeit der Ausrichtung an Elementen der Internetseite. So kommt es bei den Testgeräten zu Unstimmigkeiten im dargestellten Inhalt, welche durch die Unterschiedlichen Auflösungen und Ausrichtungen der Geräte zu Stande kommen.

### 5.5.4 Fazit zu Browser-Sync

Browser-Sync ist ein viel Versprechendes Framework, welches die zu untersuchen- den 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 bei meistens bei mehr als 6 verbundenen Klienten auf. Das Framework ist zum validieren von Websites gedacht, die sich noch in der Entwicklung befinden. Das testen ist aufgrund der notwendigen Testumgebung nur zum lokalen Arbeiten vorgesehen. Als Pluspunkt wird das injizieren von geänderten Code zur Entwicklungszeit gewertet. So ist es Möglich zum Beispiel 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 %
unterstütze 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 in Hinsicht auf ihre tatsächliche Funktionalität, ihre Installation und Kombinierbarkeit mit anderen verwendeten Frameworks.

Die Bibliotheken werden insbesondere auf ihre Implementation in einen Node.JS Server überprüft.

### 5.6.1 Installation eines Node.JS Servers

Die Installation des Node.JS Servers erfolgt einfach über die Konsole unter Mac oder den Installer<sup>10</sup>. 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 dem Observer-Pattern, das bedeutet das 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.

---

<sup>10</sup>erhältlich unter [Nodejs.org](http://Nodejs.org)

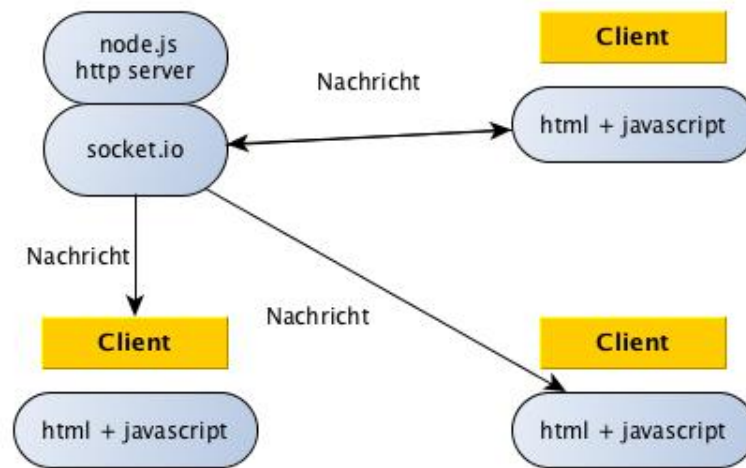


Abbildung 5.16: socket.io Broadcast [ScVi]

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

Ein Beispiel: Ein Nutzer klickt auf einer Seite auf einen Button. Das sende-Event wird an den Server gesendet und an alle per socket verbundenen Clients dupliziert. Somit würde der ursprüngliche Sender des Signals, erneut das selbe Event erhalten. Das Resultat wäre, 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 empfangene 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.

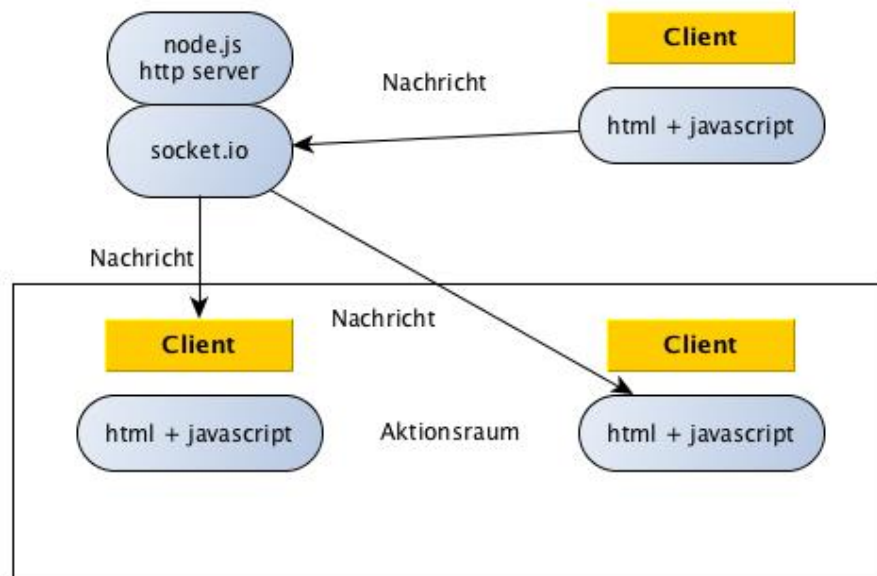


Abbildung 5.17: socket.io Broadcast mit Aktionsraum

### 5.6.4 Implementierung eines einfachen Broadcast

Das Beispiel soll veranschaulichen wie ein einfacher Broadcast ohne Aktionsraum implementiert werden kann. Das Beispiel soll das Scrollevent des Clients abfangen und auf allen verbundenen Clients an die selbe Position auf dem Bildschirm scrollen.

#### Serverseitig

Zu Beginn werden die notwendigen Bibliotheken eingebunden um einen NodeJS Server starten zu können (Zeilen: 1-2). Im Anschluss wird eine Serverinstanz von NodeJS 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 ein selbstdefinierten Methodenaufruf vom »scroll« (Zeile: 11).

Wenn am Server ein Scrollevent 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 Bildausschnittes 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. Die sehr schnelle Datenübertragung in nahezu Echtzeit mittels Node.js, Voraussetzung ist hier, dass die Geräte sich im gleichen Lokalen Netz befinden, die einfache Implementierung von Steuersignalen über socket.io und die modulare Grundstruktur der Frameworks ermöglichen einen einfachen Einstieg in die Materie.

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 irreführende 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 sollte dies auch nicht verfügbar sein auf eine Reihe verschiedener Long-Polling-Ansätzen um die Kommunikation weiterhin zu gewährleisten.

Entwurf

# 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 Webseitentest und das Angebot wird durch den Wunsch diese Test 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, gewertet werden.

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

Das Ziel der Arbeit war es durch das effiziente Testen von Web-Applikationen mehr Qualität zu erreichen und dies in weniger Zeit als im herkömmlichen Sinne notwendig ist. 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 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 grade im Bereich des mobilen Testen 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 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 erfassen. 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 arbeiten mit Nachdruck daran dieses Ziel auch zu erreichen.

### 6.1.1 Ausblick

Derzeit besitzt Browser-Sync wohl das höchste Potential, ein Produkt zu erschaffen, welches die Produktivität in der Web-Applikationsentwicklung stark optimiert. 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 Potential ist Ghostlab, welche zwar schon gute Allroundansätze derzeit vorweisen können, jedoch leider im Detail nicht ausgereift sind. Die Nachfrage nach einem qualitativ hochwertigen Produkt ist vorhanden, wie es diverse Rezensionen von verschiedenen Fraktionen der Webgemeinde belegen.

Auch der 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 vorraussichtlich viele Einzelaspekte verfolgen, die sich dann einfach über das Node.JS-eigene Paketverwaltungstool Implementieren lassen.

Entwurf

# 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]

. 9

## Bildauflösung

" 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]

. 9

## 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. " [RJBF88]

Der Einfachheit halber wurden Sammlungen die nach dieser Definition eventuell unter den Begriff eines Toolkits fallen, ebenfalls als Framework betitelt  
. 9, 41

## HTML

Die Hypertext Markup Language ist eine Auszeichnungssprache zur Beschreibung von Inhalten. Sie dient der Strukturierung von Texten, Links<sup>1</sup>, Listen und Bildern eines Dokumentes. Eine HTML Seite wird von einem Webbrowser interpretiert und anschließend dargestellt. Die Entwicklung von HTML

---

<sup>1</sup>Verweise zu anderen Inhalten

---

geschieht durch das World Wide Web Consortium(W3C) und den Web Hypertext Application Technology Working Group (WHATWG) . 9

## **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]

. 9

## **mobiles Endgerät**

Komponenten mit primärer mobiler Nutzung werden umfassend als mobile Endgeräte gruppiert. Hierzu zählen Smartphones, Tablets, sowie das Microsoft Surface. 9

## **NodeJS**

Plattform um serverseitige eventgesteuerte Javascriptanwendungen zu entwickeln. 9

## **NPM**

Node Packaged Modules, eine Software zur Installation von NodeJS Bibliotheken. 9

## **Panorama View**

Horizontalausrichtung des Bildschirms eines mobilen Endgerätes. 9

## **PHP**

Eine an C und Perl angelehnte Skriptsprache für dynamische Webseiten. 9

## **Pixel**

Auch bekannt als Bildpunkt. Farbwert einer digitalen Rastergrafik. 9

## **Portrait View**

Vertikalausrichtung des Bildschirms eines mobilen Endgerätes. 9

## **Qualitätssicherung**

Station, welche ein Produkt (hier die Anwendung) durchlaufen und bestehen muss um eine gewisse Qualität zu gewährleisten. 9

---

**Smartphone**

Mobiltelefon mit Computerähnlicher Struktur, meist mit Touchdisplay ausgestattet. 9

**Tablet**

tragbarer flacher Computer mit einem Touchscreen. 9

**VirtualBox**

Virtuelle Desktopumgebung von Oracle. Wird genutzt um zum Beispiel ein anderes Betriebssystem als das eigentlich genutzte zu Emulieren. 9

**Webbrowser**

Computerprogramm zur Darstellung von Inhalten des World Wide Web. 9

Entwurf

# Index

Abgrenzungskriterien, 4  
Abschlussthesis, 4  
Adobe Edge Inspect, 16, 27  
  
Browser-Sync, 17, 33  
  
Evaluation, 4  
  
Framework, 35  
Frameworks, 3, 4, 6  
  
Ghostlab, 15, 23  
  
jQuery Touchit, 19  
jQuery UI Touch Punch, 19  
  
Kaskadierungsfehler, 2  
  
NodeJS, 18  
  
Phantom Limb, 18  
  
Remote Preview, 16, 31  
  
socket.io, 18  
Softwareframeworks, 3  
  
Testunits, 3  
  
Usecases, 2  
  
Weinre, 23, 28, 30  
  
Zombie.js, 18

---



# 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>

---