Version 0.1α
letzte Änderung: 18. März 2014

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

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 18. März 2014

Gutachter

Prof. Knabe	Beuth Hochschule für Technik
Prof. Dr. Wambach	Beuth Hochschule für Technik

[1.5ex]

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

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.

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 translation
<–TOREMOVE

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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[1]. Im Anschluss folgt in der Regel die Entwicklung und Implementation[2] der einzelnen Komponenten. Am Ende der jeweiligen Implementationsphase durchläuft das Produkt[3] 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[4] entstehen. Aus Sicht der Qualitätssicherung ist dies ein lästiges Problem, da diese nach jedem erneuten Modifikationsvorganges eines Softwaresegments einen größeren Segmentblock, wenn nicht sogar das gesamte Softwareystem erneut testen muss. Bei der Entwicklung auf und für mobile Endgeräte[5] kommt noch ein erschwe-

render 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 unterstützen 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.

[1] Szenario oder auch Anwendungsfall

[2] Einbindung

[3] hier: einzelne Softwarekomponente

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

[5] Smartphones, Tabletts oder Ähnliche

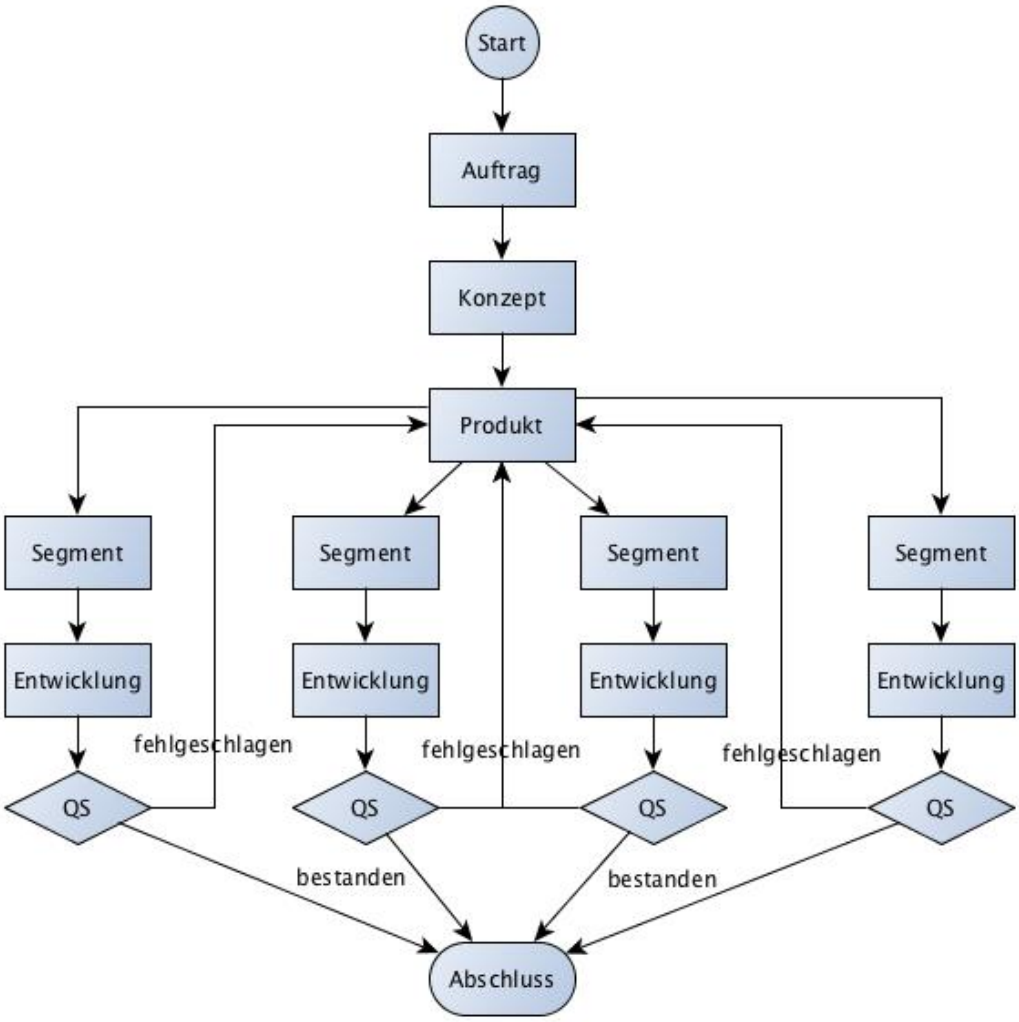

Abbildung 1.1: Vereinfachte Darstellung eines Softwareentwicklungsprozesses

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 Betriebsystemen 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 Testszenraio kann Navigationsabläufe[1], das ausfüllen und validieren eines Formular oder auch das überprüfen funktionaler[2] 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[3] und immer wieder auftretenden Testszenarioschleifen eine gewisse Routine einschleichen kann, worunter die Qualität des Produkts leidet.

[1] ein Nutzerspezifischer Gang durch die Webseite

[2] aktive Links und deren Aufruf

[3] Problemlösungen, Codeanpassungen

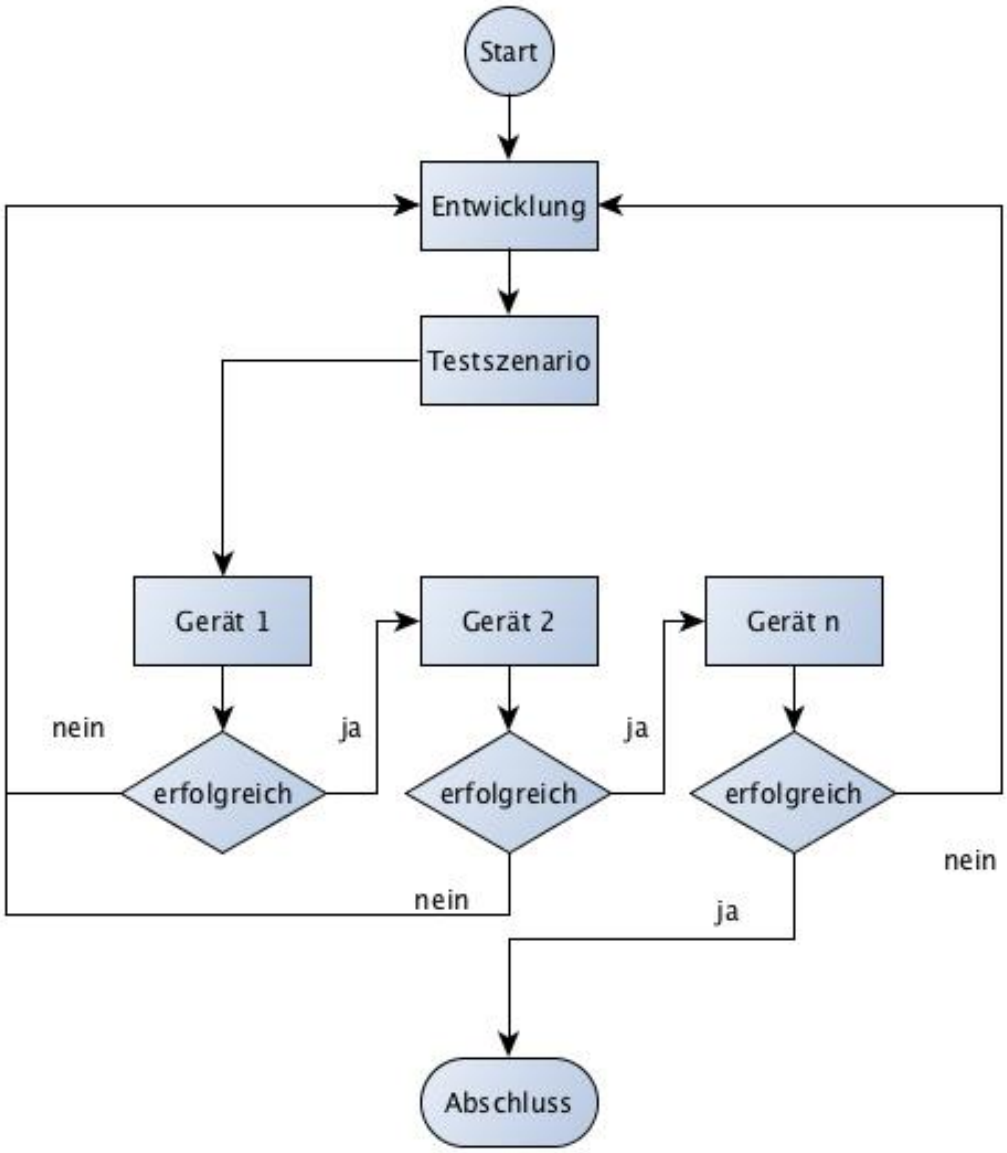

Abbildung 2.1: Darstellung eines Qualitätssicherungsablaufes in der mobilen Anwendungsentwicklung

2.2 Annahmen und Einschränkungen

2.3 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 TOREMOVE–>
Versionsnummern
<–TOREMOVE 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.

2.4 Abgrenzungskriterien

- Einarbeitungszeit
- Erweiterbarkeit in Hinsicht auf mehrerer Geräte
- Erweiterbarkeit des verwendeten Frameworks durch eigene Funktionen
- Browsersupport
- Virtuelle Umgebung - ...

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

3.1.1 parallel-synchron

3.1.2 Web-Applikation

3.1.3 HTML

Die Hypertext Markup Language ist eine Auszeichnungsprache zur Beschreibung von Inhalten. Sie dient der Strukturierung von Texten, Links[1], 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).

3.1.4 Webbrowser

3.1.5 Desktopcomputer / Desktops

In dieser Arbeit werden gängige Modelle von Personal Computern oder Macs mit einem festen Arbeitsumfeld als Desktops 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 .

3.1.6 Mobiles Endgerät

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

[1] Verweise zu anderen Inhalten

3.1.7 Javascript

3.1.8 Framework

3.1.9 Nodejs

3.1.10 PHP

3.1.11 NPM

3.1.12 Qualitätssicherung

3.1.13 VirtualBox / virtuelle Umgebung

3.1.14 Smartphone

3.1.15 Tablet

3.1.16 Panorama / Portrait View

3.1.17 Pixel

3.1.18 Auflösung

3.1.19 Event

3.1.20 DOM

3.1.21 Apache

3.1.22 Form, Checkbox, Radiobox, Inputs

3.1.23 Workspace

3.1.24 Commit

3.1.25 Deamon

3.1.26 App

3.2 verwendete Hardware

3.2.1 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)

3.2.2 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 3.1: 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 3.2: 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 3.3: 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 3.4: 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 3.5: 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 3.6: Übersicht Microsoft Surfcae

3.3 verwendete Browser

3.3.1 Raspberry Pi

3.3.2 Hardware

4 Technologien

4.1 Ghostlab

Ghostlab ist ein Framework des Schweizer Unternehmens Vanamco. Es verspricht das synchrone Testen von Websiten 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 Mobile	supportet
Opera	11
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.

4.2 NodeJS

4.3 Zombie.js

4.4 W3C Touch Events Extensions

4.5 Phantom Limb

4.6 jQuery UI Touch Punch

4.7 jQuery Touchit

4.8 NPM touchit

4.9 Adobe Edge Inspect

4.10 Remote Preview

4.11 Browser-Sync

4.12 Eigenes Framework

5 Evaluation der Techniken

5.1 Auflistung des Evaluationsschlüssels

1. Installation
   a) sind zusatzinstallationen notwendig (basiert das Werkzeug auf anderen Technologien) 4Pt
   b) kann nach der Installation die Software direkt genutzt werden ? 2Pt
   c) gibt es eine zur Version passende installationsanleitung? 2Pt
   d) gibt es eine FAQ? 2Pt
2. Konfiguration
   a) mann die Software out-of-the-Box[1] genutzt werden ? 4Pt
   b) ist die Software Konfigurierbar in Hinsicht auf IP-Adressen und Ports? 1Pt
   c) kann man verschiedene Konfigurationen abspeichern ? (für z.B. verschiedene Arbeitsumgebungen) 2Pt
   d) gibt es Support (Wiki, Helpdesk, EMail, Forum)? 2Pt
   e) Intuitive Benutzeroberfläche 1Pt
3. Funktion: Desktop
   a) Darstellung : normale Seiten 2Pt
   b) Darstellung : gesicherte Seiten 2Pt
   c) Darstellung: normale(< 1 Sekunde) Reaktionsgeschwindigkeit 2Pt
   d) Funktion: Seitensteuerung 3Pt
   e) Funktion: Javascript 1Pt
4. Funktion: Mobil
   a) Darstellung : normale Seiten 1Pt
   b) Darstellung : gesicherte Seiten 1Pt
   c) Darstellung: normale(< 1 Sekunde) Reaktionsgeschwindigkeit 2Pt
   d) Funktion: Seitensteuerung 4Pt

[1] ohne weitere Konfiguration nach der Installation

e) Funktion: Gestenkontrolle 1Pt

f) Funktion: Javascript 1Pt

5. Erweiterbarkeit
   a) gibt es eine API zur Implementierung in eigenen Entwicklungen ? 5Pt
   b) Ist die API rechtlich durch Lizenzen geschützt? 2Pt
   c) ist die API Dokumentiert? 3Pt
6. unterstützte Browser (aktuelle Version zum Zeitpunkt der Erstellung dieser Thesis)
   a) mobile Plattformen (iOS, Android, Windows) 3Pt
   b) Unterstützung von Browser innerhalb einer virtuellen Umgebung 2Pt
   c) Chrome 1Pt
   d) Opera 1Pt
   e) Firefox 1Pt
   f) Safari 1Pt
   g) Internet Explorer 1Pt
7. Aktivität
   a) wird die Software noch entwickelt? (letzes Release, Commit Häufigkeit) 5Pt
   b) gibt es ein aktives Forum (letzter Beitrag jünger 14 Tage) 5Pt

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[2] auf Port 8005 (Default) von allen zu testenden Geräten erreichbar.

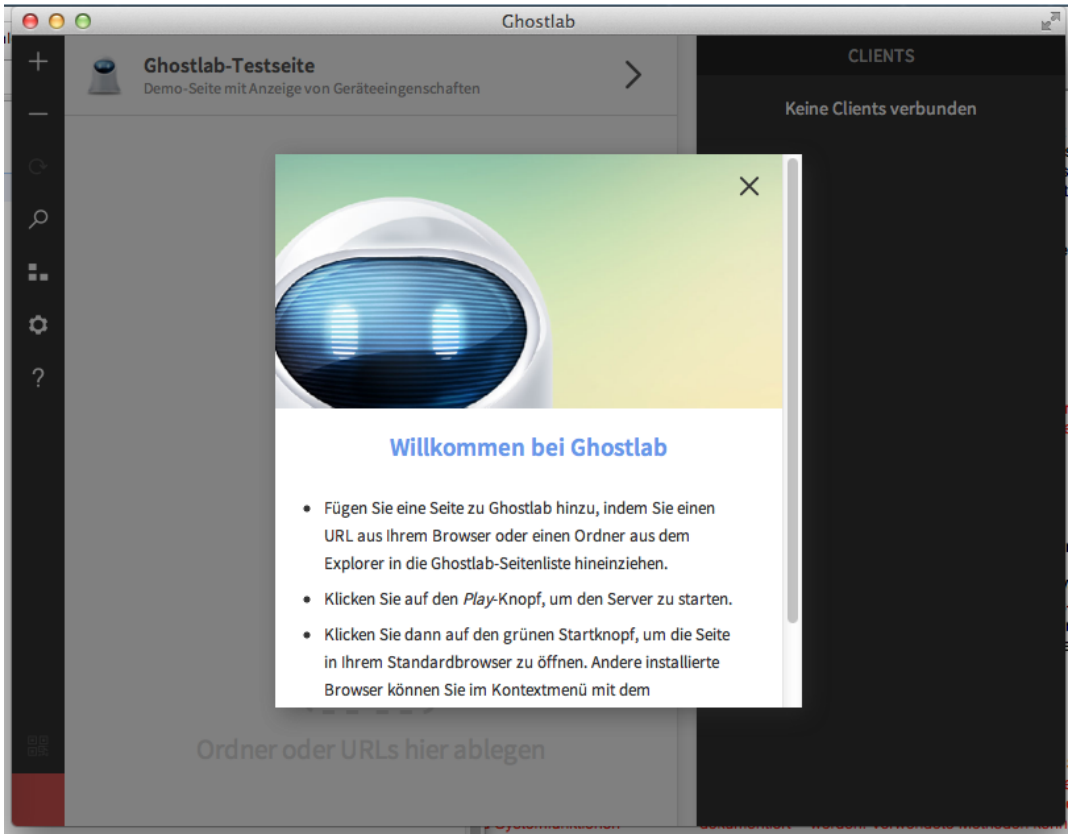

Abbildung 5.1: Startbildschirm von Ghostlab nach der Installation

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

[2]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 parallelsynchron auf allen anderen Clients gespiegelt wird. Hierzu zählen Javascriptevents, das ausfüllen eines Formulars oder das neuladen der gesamten Seite.

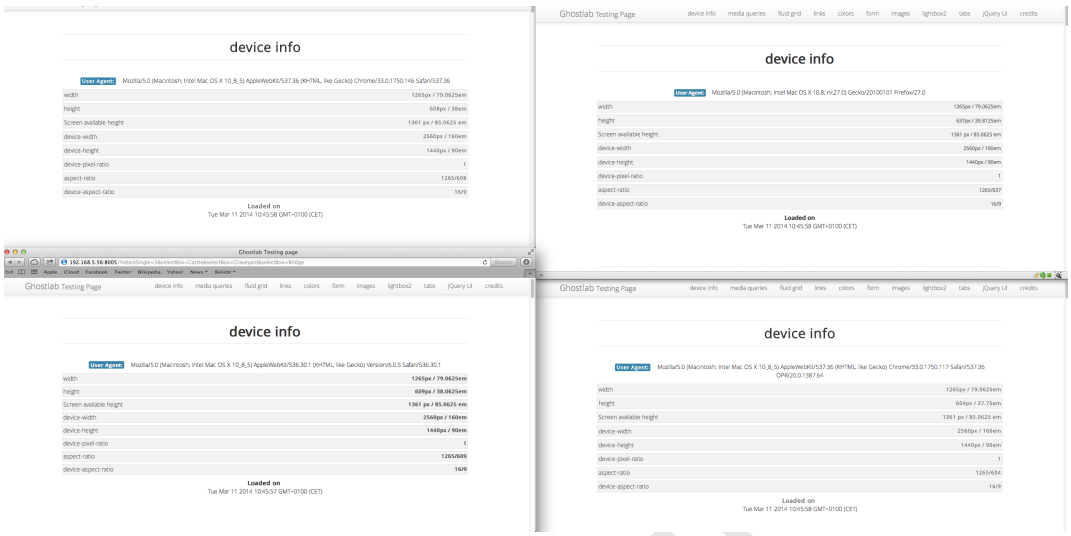

Abbildung 5.2: Darstellung von 4 verschiedenen Clients

Über den Übersichtsbildschirm 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.

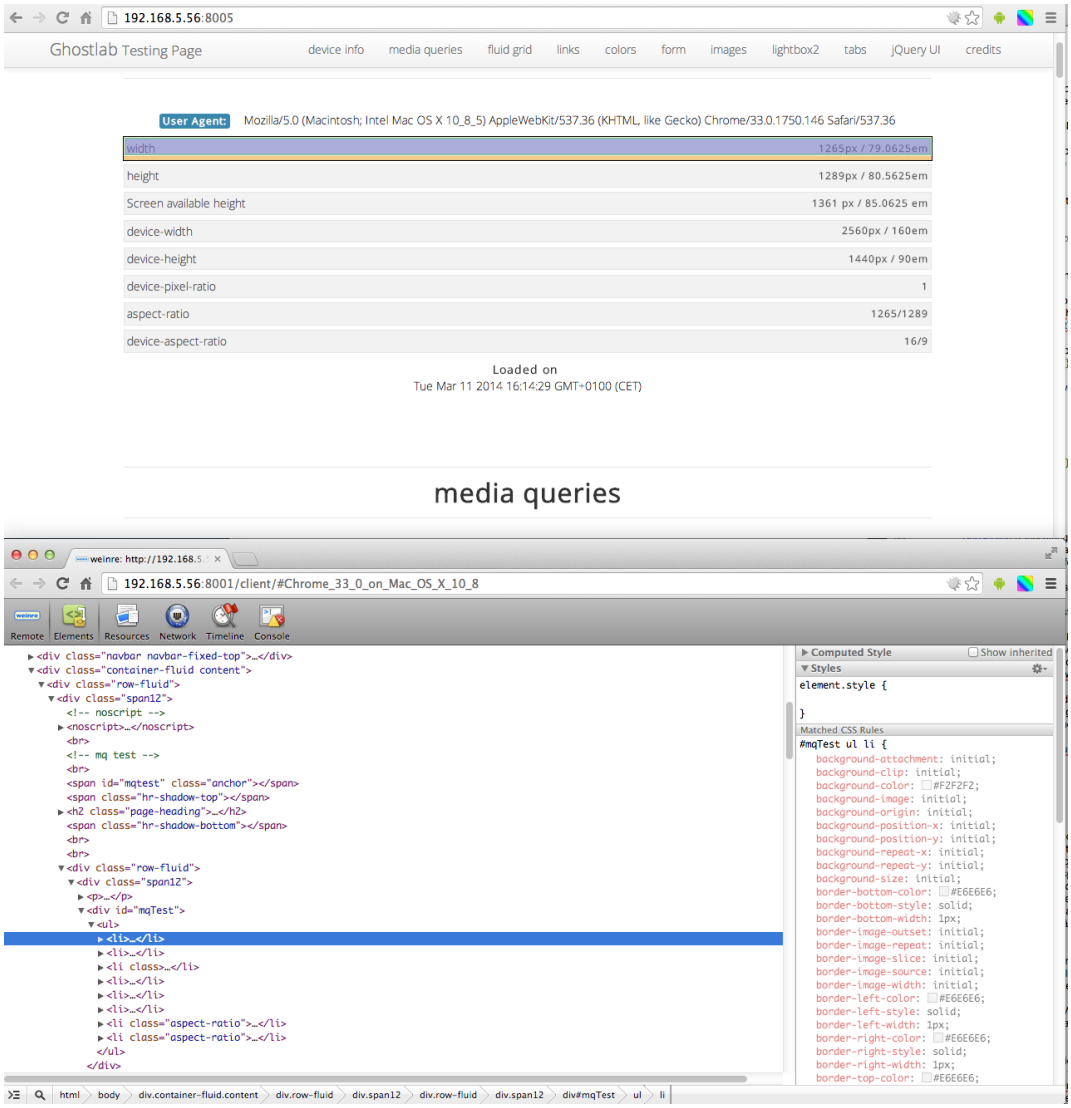

Abbildung 5.3: ausgewähltes DOM-Element in Weinre

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[3] in der Clientliste aufgenommen.

Bei dem Testen auf mobilen Browserns ist es bei Ghostlab[4] 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.

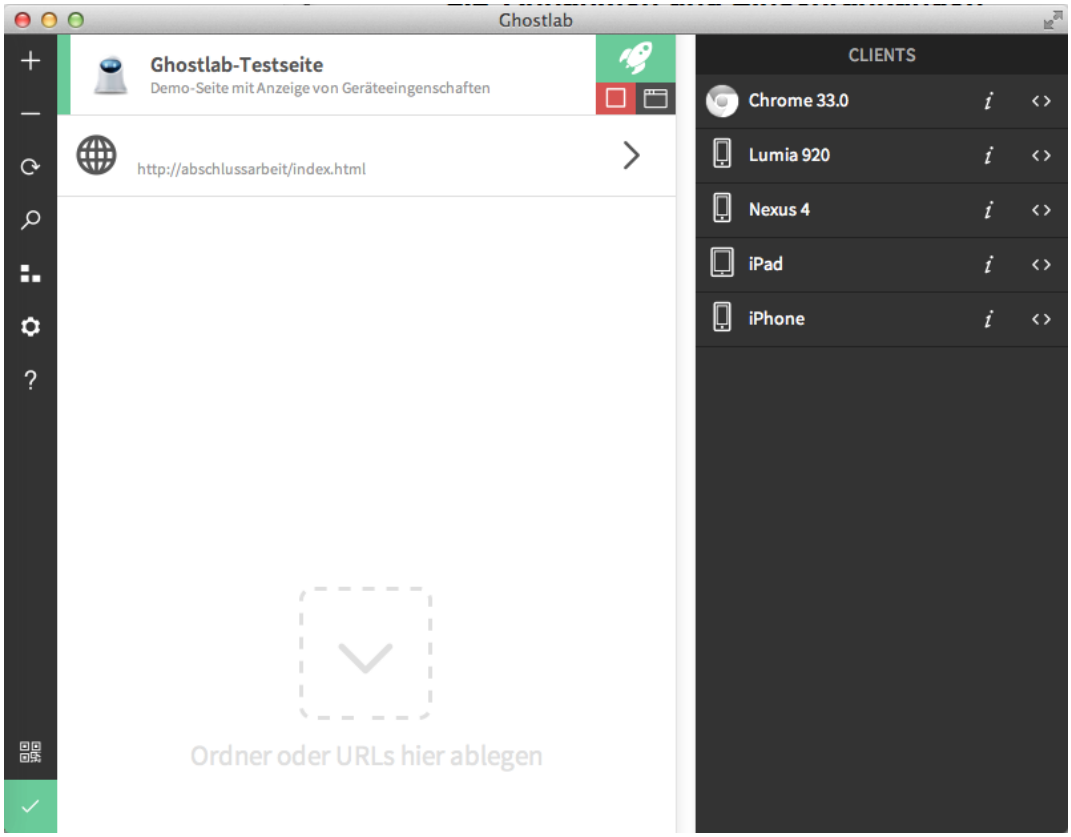

Abbildung 5.4: Ghostlabübersicht der verbundenen Clients

[3] abhängig von der Geschwindigkeit des Testgerätes

[4] 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[5], dadurch kam es bei meinen Testgeräten bereits nach wenigen Minuten zu dem Problem, dass die Geräte sich in einer Endlosschelife 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, Checkboxen, 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[6] 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[7], 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[8] 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.

[5] ein Gerät dient als Steuergerät, alle anderen folgen ihm

[6] es wurde VirtualBox von Oracle genutzt

[7] Arbeitsumgebung oder auch Arbeitsumfeld

[8] 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 %
unterstütze Browser	10	10 %
Aktivität	5	10 %
Gesamt	67.5	100 %

Tabelle 5.1: Gewichtungstabelle Evaluation von Ghostlab

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 Deamon im Hintergrund.

Abbildung 5.5: Der laufende Deamon 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 letzes 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.

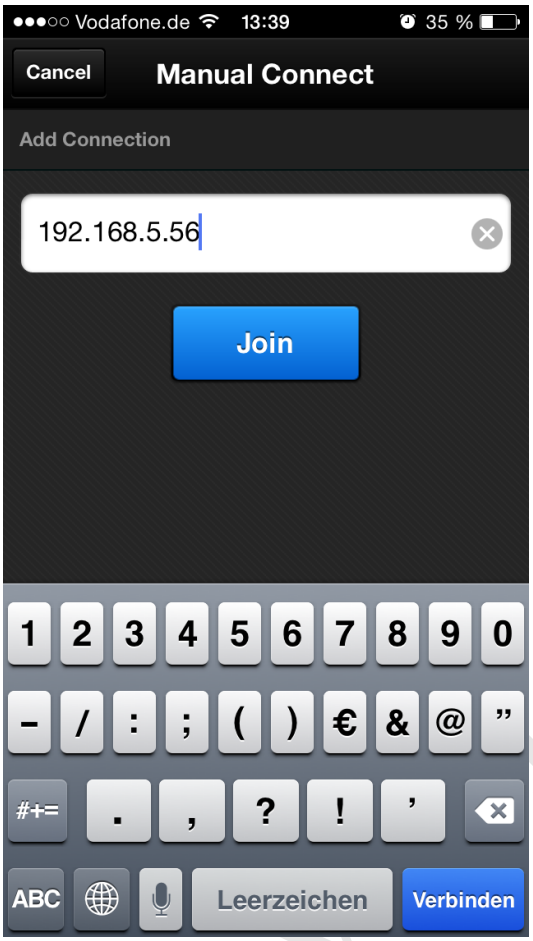

Abbildung 5.6: Eingabe der IP-Adresse zum Edge Inspect Rechner

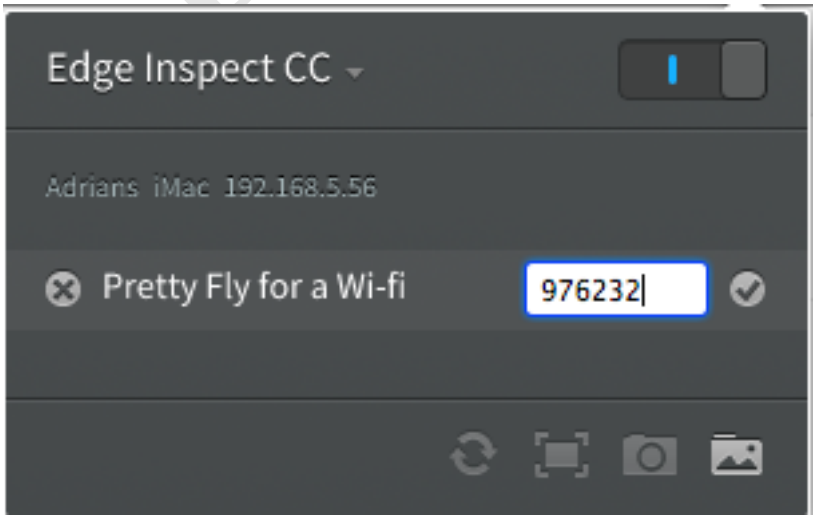

Abbildung 5.7: Eingabe des Sicherheitscodes in die Chrome Extension

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.

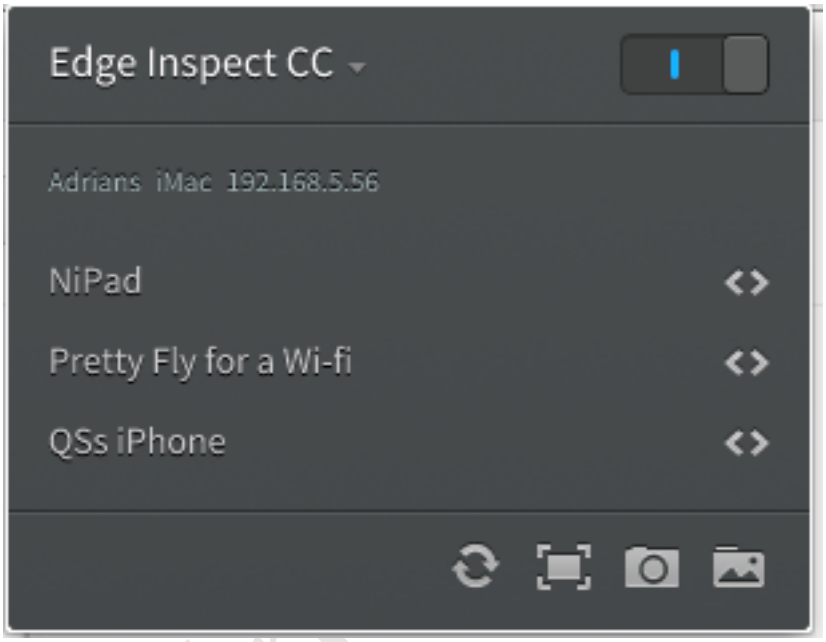

Abbildung 5.8: Übersicht der verbundenen Clients

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 Betriebssystems, der aufgerufenen URL sowie der aktuellen Ausrichtung des Bildschirms ausgeliefert.

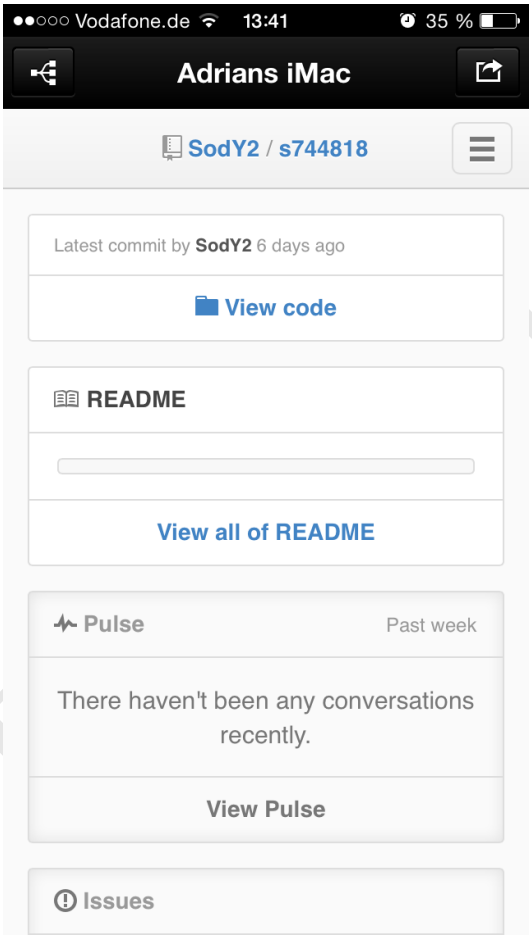

Abbildung 5.9: Darstellung von Content in der Adobe Edge Inspect App

TOREMOVE–> iOS 6 gesicherte Seiten <–TOREMOVE

5.3.4 Fazit zu Adobe Edge Inspect

Adobe Edge Inspect bedarf viel Aufwand für relativ geringes Ergebnis. Man muss an 3 verschiedenen Punkten Installationen vornehmen, die dann jedoch ohne Pro-

bleme miteinander harmoniert 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 externe Applikationen auswerten lassen.

Der Nutzen des Werkzeugs liegt am ehesten bei One-Page-Sites[9] oder für Fehlersuche innerhalbs des DOM oder CSS Anpassungen. Unter dem Aspekt des parrallelsynchronen Testens ist Adobe Edge Inspect leider nicht sinnvoll zu verwenden, da weder Scrollevents oder andere Gesten umgesetzt werden, noch werden die Nutzereingaben in Eingabefeldern mit anderen verbundenen Clients geteilt. Alle verbunden 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 %
unterstütze Browser	3	10 %
Aktivität	10	10 %
Gesamt	48	100 %

Tabelle 5.2: Gewichtungstabelle Evaluation von Adobe Edge Inspect

5.4 Remote Preview

5.5 Browser-Sync

5.6 Eigenes Framework

5.6.1 Systementwurf

Ablaufdiagram

Klassendiagramm

[9] Webseiten dessen Inhalt sich füllend auf die gesamte Seite erstreckt

6 Ausblick

Index