Anwendungsorientierte App-Entwicklung zur Geolokalisierung von Fotos unter Einbindung von Social Media Technologien

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Jan-Hendrik Kruse

2054410

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Andreas Plaß

Zweitprüfer: B. Sc. Helge Morgenstern

Hamburg, 18.07.2016

Inhaltsverzeichnis

Abstract

This thesis deals with the conceptual design and realisation of a client-server application. By using this application, it will be possible to locate images. For this matter social media technologies such as liking and sharing of images are applied. Furthermore, this thesis enlarges on a frontend site connected to a server via HTTP protocol according to the REST paradigm. The realisation of the backend including database connection is part of the Bachelor's thesis "Konzeption und Entwicklung einer Web-API am Beispiel des Geotagging-Services Grafflr" by Levin Mauritz. This thesis aims at analysing which technologies work properly in order to develop an application that can be presented to users quickly and that offers a coherent surface. After realising the application the use of the surface will be conducted and evaluated by a test. Furthermore, the test will be analysed by a survey among users.

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Konzeption und Implementierung einer Client-Server-Applikation über die Fotos geolokalisiert werden können. Dabei werden Technologien aus dem Social Media Bereich wie zum Beispiel das Liken bzw. Teilen der Fotos eingesetzt. In dieser Arbeit wird die Frontend Seite mit Anbindung an einen Server über das HTTP Protokoll nach dem REST Paradigma behandelt. Die Umsetzung des Backends inklusive Anbindung an Datenbanken ist Teil der Bachelorarbeit "Konzeption und Entwicklung einer Web-API am Beispiel des Geotagging-Services Grafflr" von Levin Mauritz. Ziel ist es zu untersuchen welche Technologien derzeit gut geeignet sind, um eine Applikation zu entwickeln, die schnell implementiert sowie vielen Nutzern zugänglich gemacht werden kann und eine Benutzeroberfläche besitzt, die sich dem Nutzer innerhalb kürzester Zeit erschließt. Nach der Umsetzung dieser Applikation wird die Bedienung der Benutzeroberfläche im Praxis-Test evaluiert und über eine Nutzerumfrage ausgewertet.

1 Einleitung

1.1 Motivation

Apps sind mittlerweile allgegenwärtig und werden für fast alle Plattformen bzw. Endgeräte angeboten. Im deutschen Sprachraum hat sich das Wort App für Applikation mit dem Erscheinen des App-Stores im Jahr 2008 eingebürgert und wurde in erster Linie mit mobilen Anwendungen für Smartphones und Tablets gleichgesetzt. Heute sind App-Stores ebenfalls für Betriebssysteme wie Apples Mac OS X oder Microsoft Windows sowie Browser wie Google Chrome oder Safari vorhanden. App-Stores sind nicht nur bei Smart TVs, TV-Boxen wie der Apple TV, Amazon Fire Box oder Spielekonsolen wie der Playstation oder der X-Box zu finden, sondern auch bei den so genannten Smartwatches gängig.

Neben den nativen Apps werden mittlerweile auch Webanwendungen (Web-Apps) immer häufiger in Unternehmen sowie im privaten Bereich eingesetzt und machen es möglich auf Desktopanwendungen zu verzichten. Viele große Unternehmen wie Google, Microsoft oder Apple bieten mittlerweile einen Teil ihrer Software nicht nur als Desktopanwendung an, sondern stellen diese ebenfalls als Web-App im Internet zur Verfügung. Google stellt über Google Docs ein komplettes Office Paket zur Erstellung von Dokumenten, Tabellen und Präsentationen ausschließlich als Web-App im Internet zur Verfügung. Apples Office Paket (Pages, Numbers, Keynote) ist neben den nativen Apps ebenfalls als Web-App vorhanden. Microsoft ist diesem Trend ebenfalls gefolgt und stellt sein Office 365 Paket zusätzlich als Web-App zur Verfügung. Bei all den genannten Unternehmen ist eine Bedienung ihrer Mailclients, sowie das Verwalten von Kalender und Kontakten im Internet über eine Web-App möglich.

Google geht sogar einen Schritt weiter und bietet mit Chrome OS ein Betriebssystem an, dass zwar auf einem Linux-Kern basiert aber als Betriebssystemoberfläche lediglich auf den Webbrowser Chrome setzt. Im Gegensatz zu Microsoft Windows oder Mac OS, die auf native Anwendungen setzen, muss unter Chrome OS keine Anwendung installiert oder aktualisiert werden[1].

Apps bieten seinem Nutzer eine einfache Möglichkeit der Installation bzw. Aktualisierung über einen App-Store an und sind in hohem Maße benutzerfreundlich in ihrer Bedienung.

Einen aus meiner Sicht sehr großen Nutzen bieten Apps, die in ihre Funktionali-

[1] Chrome OS bietet seit Ende 2014 die Möglichkeit, ebenfalls Android Apps unter Chrome OS laufen zu lassen. Dies ermöglicht die Native Client (NaCl) Sandbox Technik, die kompilierten C oder C++ Code im Browser ausführt.

tät GPS unterstützen und dem Nutzer darüber geolokalisierte Dienste anbieten. So können neben der Möglichkeit des Routings zu beliebigen Positionen auf der Karte, ebenfalls interessante Orte wie zum Beispiel Restaurants in der Umgebung angezeigt werden.

Neben Google Maps, auf deren Karte mittlerweile so ziemlich alles von Einkaufsläden bis hin zu Ärzten zu finden ist, wäre zum Beispiel Yelp[2] solch eine App. Diese App bezieht neben der Möglichkeit zur Geolokalisierung von Restaurants ebenfalls die soziale Komponente mit ein und bietet die Möglichkeit zur Bewertung der Restaurants. Zusätzlich können Kommentare zu den Restaurants hinterlegt werden.

Die Idee eine App zur Geolokalisierung von Fotos zu entwickeln, ist aus der Idee heraus entstanden einem Nutzer die Möglichkeit zu bieten Streetart und Graffiti Fotos über eine App zu sammeln und diese darüber anderen Nutzern zugänglich zu machen bzw. selber deren Position auf einer Karte anzeigen zu lassen, falls diese erneut aufgesucht werden sollen. Es sollen dafür Technologien aus dem Social Media Bereich zum Einsatz kommen, durch die der Nutzer die Möglichkeit haben soll Fotos zu liken, teilen, melden oder Kommentare zu hinterlassen.

Der Zeitpunkt für eine solche App scheint günstig, da Streetart und Graffiti eine immer höhere Akzeptanz in der breiten Bevölkerung findet und sich mittlerweile als Kunstform etabliert hat. In manchen Städten wie zum Beispiel Hamburg oder Berlin werden bereits Graffiti und Streetart Touren angeboten.

[2] https://www.yelp.de, zuletzt abgerufen am 03.06.2016

1.2 Aufgabenstellung und Zielsetzung

Ziel dieser Bachelorarbeit ist es eine Applikation zu entwickeln über die es möglich ist, Fotos inklusive ihrer GPS Position auf einen Server zu laden und diese innerhalb der Applikation auf einer Karte darstellen zu lassen. Neben der Karte sollen die Fotos zusätzlich als Liste dargestellt werden. Nutzer dieser Applikation sollen sich registrieren können, um Fotos auf den Server zu laden. Dem Nutzer sollen zusätzlich verschiedene Möglichkeiten aus dem Social Media Bereich zur Verfügung gestellt werden. So soll er zum Beispiel Fotos liken können oder sich gelikte bzw. auf den Server geladene Fotos eines anderen Nutzers anzeigen lassen können. Die Benutzerfreundlichkeit soll dabei einen Schwerpunkt bilden. Der Nutzer soll sich sehr schnell und ohne vorherige Einweisung in der Bedienung dieser App zurechtfinden.

Dabei soll der Fokus auf das Frontend gelegt werden. Das Backend, sowie dessen Anbindung mittels REST API wird in dieser Bachelorarbeit nicht berücksichtigt und ist Teil der Bachelorarbeit “Konzeption und Entwicklung einer Web-API am Beispiel des Geotagging-Services Grafflr” von Levin Mauritz.

Das Ziel lässt sich in folgende Punkte gliedern:

1. Es gilt zu entscheiden welche Technologien derzeit sinnvoll sind um eine erweiterungsfähige Applikation zu entwickeln, die schnell implementiert werden kann.
2. Es soll eine einsatzfähige Applikation implementiert werden, die vielen Nutzern zugänglich gemacht werden kann sowie benutzerfreundlich ist.
3. In einer Evaluation soll mittels Umfrage ermittelt werden, inwiefern die Applikation bereits benutzerfreundlich ist und was noch verbessert werden kann.

1.3 Aufbau der Arbeit

In dieser Arbeit werden zunächst unter Grundlagen (siehe Kapitel [2]) die verschiedenen App-Technologien vorgestellt, um einen Überblick über den aktuellen Stand der App-Technologien sowie deren Vor- bzw. Nachteile zu schaffen. In der Zusammenfassung dieses Kapitels findet ein Vergleich zwischen diesen Technologien statt. Die weiteren Kapitel beschreiben die notwendigen Schritte zur erfolgreichen Umsetzung einer einsatzfähigen Applikation und welche Technologien für dessen Umsetzung genutzt werden und warum:

1. In der Anforderungsanalyse (siehe Kapitel [3]) soll mittels Personas, UML Use Case Diagramm und User Stories ein möglichst klares Bild der Nutzer und deren Anforderungen an die Applikation geschaffen werden.
2. Unter Konzeption und Design (siehe Kapitel [4]) soll daraufhin, anhand der beiden vorherigen Kapitel, entschieden werden welche App Technologie am sinnvollsten erscheint um eine Applikation zu implementieren, die schnell implementiert und vielen Nutzer zugänglich gemacht werden kann sowie die Anforderungen aus der Anforderungsanalyse erfüllt. Ebenfalls sollen hier die Techniken und Routen für die Anbindung an den Server beschrieben, sowie Mock-ups für die Umsetzung des Designs der Benutzeroberfläche erstellt werden.
3. Unter Umsetzung und Implementierung (siehe Kapitel [5]), soll nun auf Basis des Konzeptes aus dem vorherigen Kapitel die Applikation implementiert und dessen Implementierung beschrieben werden. Dadurch soll der Leser einen Überblick über die derzeit nutzbaren Technologien bekommen, mit denen diese Applikation gemäß der Anforderungen umgesetzt werden kann und warum diese Technologien genutzt wurden.
4. Während der Evaluation (siehe Kapitel [6]) sollen die Probanden anhand eines Praxis-Tests, der in direktem Bezug zu den User Stories steht, ermitteln, inwieweit die Anforderungen hinsichtlich der Benutzeroberfläche dieser Applikation erfüllt wurden. Dafür sollen die Nutzer eine Umfrage ausfüllen, die anschließend ausgewertet werden soll.
5. Im Fazit (siehe Kapitel [7]) sollen die Ergebnisse dieser Bachelorarbeit zusammengefasst und ein Ausblick auf die Weiterentwicklung dieser Applikation gegeben werden.

2 Grundlagen

In diesem Kapitel soll zunächst der Begriff App erklärt werden. Im Anschluss sollen die verschiedenen App Typen vorgestellt, um sie anschließend in der Zusammenfassung miteinander zu vergleichen.

2.1 App

Der Begriff App ist laut Duden eine

„[...] zusätzliche Applikation, die auf bestimmte Mobiltelefone heruntergeladen werden kann.“ (Duden - App, die oder das 2016).

Im deutschen Sprachraum wird das Wort App seit Einführung des iOS App-Stores im Jahr 2008 in erster Linie für Mobilgeräte wie Smartphones bzw. Tablets verwendet. So stellt neben Apple mittlerweile jeder große Hersteller von mobilen Geräten wie Google, Amazon oder Microsoft seinen eigenen App-Store zur Verfügung. Dabei ist klar zu sehen, dass Apple mit seinem App-Store[1] und Google mit dem Google Play Store[2] mit Abstand die meisten Apps anbietet (siehe Abb. 2.1).

Der Begriff App wird mittlerweile jedoch auch im Browser-Umfeld genutzt. Dort kommen die so genannten Web-Apps zum Einsatz, die zwar im Browser laufen aber von ihrer Benutzeroberfläche und deren Bedienung her eher an native Apps als an Webseiten erinnern.

Seit Frühjahr 2012 gibt es das Facebook-App Center. Eine Facebook-App wird von einem externen Server geladen und direkt innerhalb der Facebook-Seite in einem iFrame angezeigt oder kann separat als Webseite im Facebook-Kontext geladen werden. Neben den Web SDKs für die Programmiersprachen PHP oder JavaScript sind ebenfalls SDKs für weitere Umgebungen wie zum Beispiel Android oder iOS vorhanden, mit denen über eine API verschiedene Facebook Funktionen wie zum Beispiel der Facebook-Login oder das Teilen eines Beitrages aus der App heraus möglich ist. Ebenfalls können nach Genehmigung durch den Nutzer seine Daten wie Name, E-Mail-Adresse, Freunde, Fotos etc. abgerufen werden[3].

Neben Erweiterungen für den Google Chrome Browser, gibt es Apps im Chrome Web Store. Diese werden wie die Web-Apps mit Web-Technologien HTML, JavaScript

[1] https://itunes.apple.com/de/genre/ios/id36?mt=8, zuletzt abgerufen am 04.06.2016
[2] https://play.google.com/store/apps?hl=de, zuletzt abgerufen am 04.06.2016
[3] Dokumentation des Facebook-SDKs https://developers.facebook.com/docs/, zuletzt abgerufen am 25.05.2016

und CSS implementiert. Sie laufen jedoch nicht direkt im Browser, sondern werden wie native Apps in den Desktop integriert. So besitzen sie zum Beispiel keine Adressleiste und können direkt aus dem Desktopumfeld über den Chrome-App Launcher gestartet werden. Unter Mac OS X werden diese Apps auch über die Spotlight Suche gefunden und können direkt darüber gestartet werden. Darüber hinaus ist es aus Chrome-Apps heraus möglich Zugriff auf das Dateisystem des Rechners zu erhalten. Ebenfalls kann die Hardware wie zum Beispiel USB oder Bluetooth angesprochen werden.

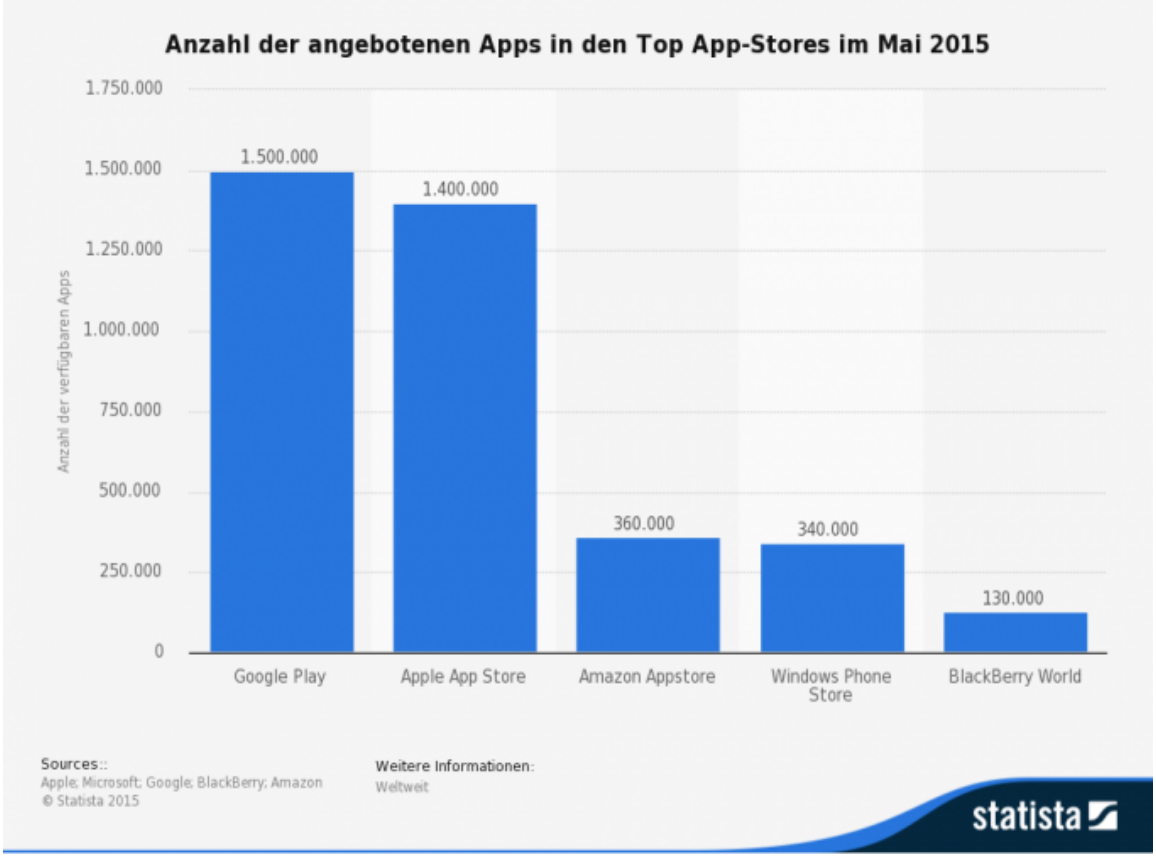

Abbildung 2.1: Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 (Statista - Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 2015)

Auch im Desktopumfeld wird von Apps gesprochen. So existieren zum Beispiel der Mac OS X App-Store oder die Windows Apps[4]. Seit Einführung von Microsoft Windows 10 im Juli 2015 bietet Microsoft einen einheitlichen App-Store für alle Geräte an (siehe Unterabschnitt [2.1.1]). Apps unterscheiden sich von traditioneller

[4]Microsoft Windows 2008 wurde auf Desktop PCs, sowie Tablets eingesetzt und ist für die Bedienung per Touch Screen optimiert ist.

Software, da sie über eine Internetverbindung und einen zentralisierten App-Store vertrieben werden.

„One of the key differences between an app and a traditional piece of software is that apps are delivered over an Internet connection. Apps are also distributed via centralised portals rather than through traditional retail channels, with portal operators taking a percentage of the sale as a commission.“ (OECD - The App Economy 2013: 8)

Ebenfalls lassen sich Apps von Systemsoftware abgrenzen.

„Anwendungssoftware steht (nach ISO/IEC 2382) im Gegensatz zu Systemsoftware und systemnaher Unterstützungssoftware. Dazu „zählen die Programme, die für den korrekten Ablauf einer Rechenanlage erforderlich sind, sowie alle Programme, die die Programmerstellung unterstützen, z. B. Übersetzer und Testwerkzeuge und allgemeine Dienstleistungen bereitstellen [... Formatierung, Dateiverwaltung, Datentransfer ...]“,[1] die aber keinen Anwender-bezogenen 'Nutzen' bringen. Beispiele sind das Betriebssystem, Compiler für verschiedene Programmiersprachen oder Programme zur Datensicherung.“ (Wikipedia - Anwendungssoftware 2016).

2.1.1 native App

Im mobilen Bereich sind besonders der Apple App-Store, Google Play Store, Amazon App-Store[5] und der Microsoft Windows Store[6] zu nennen. Mit Microsoft Windows 10 hat Microsoft seine App-Stores zusammengelegt. Es gibt nun einen einheitlichen App-Store für mobile bzw. Desktop Geräte, sowie für die Spielekonsole Xbox. Diese nativen Apps, die speziell für ein Betriebssystem wie iOS, Microsoft Windows oder Android entwickelt werden, bieten die größtmögliche Anbindung an die Hardware-Schnittestellen des jeweiligen Systems wie zum Beispiel Zugriff auf Kamera, Mikrofon, Lautsprecher, Bluetooth, Wlan, 3G, Touch ID, Barometer, 3-Achsen-Gyrosensor oder Beschleunigungssensor. Es ist über native Apps auch möglich mittels Bluetooth oder WLAN eine Kommunikation mit anderen Hardwareelementen herzustellen. So können native Apps zum Beispiel mit Drohnen, Smartwatches, Fittness-Armbändern, Waagen, Steckdosen, Heizungen, Speakern und vielen weiteren Geräten kommunizieren. Ebenfalls wird die systemübergreifende Kommunikation zwischen nativen Apps unterstützt. So ist es zum Beispiel möglich aus einer nativen App heraus mit den Kontakten, dem Kalender oder den Fotos des jeweiligen Systems zu interagieren. Ebenfalls können Dateien zum Beispiel aus Dropbox heraus in anderen Apps geöffnet werden. Das Installieren und Aktualisieren der Apps erfolgt einfach und direkt über den jeweiligen App-Store. Apple setzt für die Entwicklung seiner iOS Apps auf die Programmiersprache Swift, Google mit dem Android Betriebssystem auf die

[5] https://www.amazon.de/mobile-apps/b?node=1661648031, zuletzt abgerufen am 04.06.2016
[6] https://www.microsoft.com/de-de/store/apps, zuletzt abgerufen am 04.06.2016

Programmiersprache Java und Amazon nutzt auf seinem Betriebssystem Fire OS[7], welches auf Android basiert, ebenfalls auf die Programmiersprache Java. Microsoft setzt mit seiner Universal Windows Plattform[8] (UWP) auf die Programmiersprachen C#, Visual Basic .NET, Visual C++ und JavaScript.

Im Desktopbereich ist Microsoft, wie oben bereits erwähnt, mit seinem Windows Store vertreten. Den Mac App-Store[9] gibt es seit Mac OS X 10.6.6 und dem Jahr 2011[10]. Diese Apps können ebenfalls direkt über den jeweiligen App-Store gekauft, installiert und aktualisiert werden.

2.1.2 Web-App

Eine Webapplikation (Web-App) läuft in einem Webbrowser und funktioniert nach dem Client-Server-Modell (siehe Abb. 2.2). In der Regel erfolgt die Kommunikation über das HTTP Protokoll. Über eine Internet- oder Intranet-Verbindung wird mittels URL die Web-App auf dem Server aufgerufen. Im Browser wird nun der WebClient angezeigt. Dieser zeigt in der Regel die Benutzeroberfläche für die Bedienung der Web-App an. Für eine Web-App wird die Programmiersprache JavaScript zusammen mit der Hypertext-Auszeichnungssprache HTML und der Stylesheet Sprache CSS eingesetzt.

Es können zwei verschiedene Architekturen für eine Web-App realisiert werden. Bei der **Thin Client Architektur** werden über Hyperlinks URLs auf dem Server aufgerufen. Der Server liefert daraufhin statisch oder dynamisch erstellte Webseiten an den Client aus. Dies führt zu einem Seitenwechsel, bei dem der Browser die neue Webseite lädt und damit den HTML DOM der vorherigen Seite überschreibt. Die Funktionen zur Erstellung des neuen HTML DOM werden dabei komplett serverseitig durchgeführt.

Dem gegenüber steht die **Fat Client Architektur**, die zum Beispiel bei Single-page Webanwendungen genutzt wird.

„Als Single-page-Webanwendung (englisch Single-page Web-Application, kurz SPA) oder Einzelseiten-Webanwendung wird eine Webanwendung bezeichnet, die aus einem einzigen HTML-Dokument besteht und deren Inhalte dynamisch nachgeladen werden. “ (Wikipedia - Single-page-Webanwendung 2016)

Hierbei wird zunächst ebenfalls die Web-App vom Server geladen und im Browser des Clients interpretiert. Im Unterschied zu der Thin Client Architektur wird nun jedoch, bei Änderungen an der Benutzeroberfläche des Web Clients, nicht mehr der komplett neue HTML DOM vom Server angefordert, sondern lediglich Daten per

[7] https://developer.amazon.com/public/solutions/platforms/android-fireos, zuletzt abgerufen am 03.06.2016
[8] https://de.wikipedia.org/wiki/Universal_Windows_Platform, zuletzt abgerufen am 04.06.2016
[9] http://www.apple.com/de/osx/apps/app-store, zuletzt abgerufen am 04.06.2016
[10] https://de.wikipedia.org/wiki/Mac_App_Store, zuletzt abgerufen am 04.06.2016

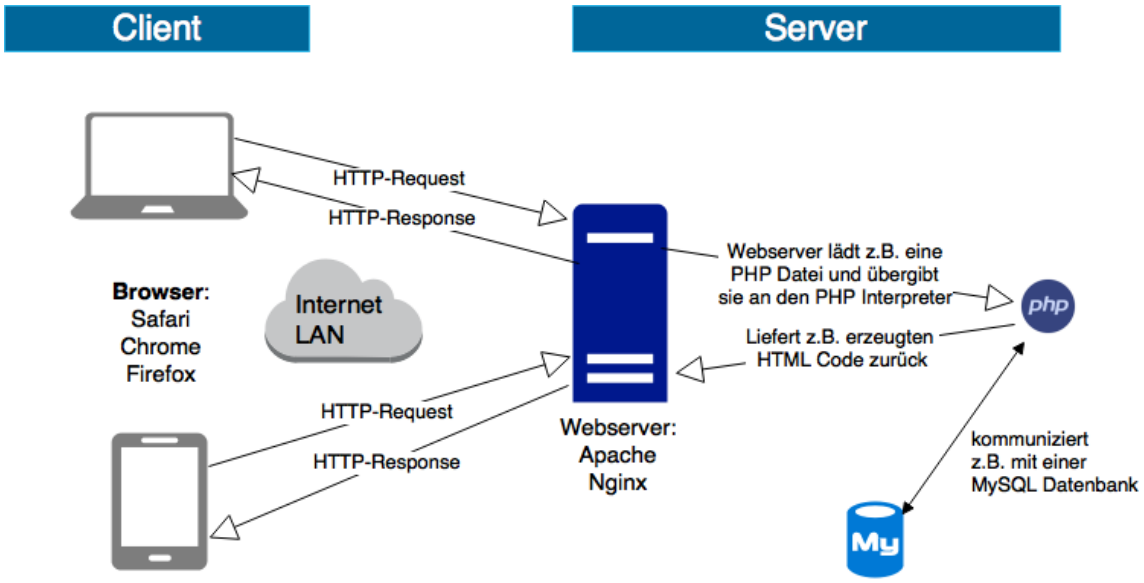

Abbildung 2.2: Client-Server Kommunikation über HTTP-Request und HTTP-Response (eigene Darstellung)

AJAX[11] nachgeladen. Mittels JavaScript werden nun Teilbereiche des HTML DOMs manipuliert, ohne einen Seitenwechsel durchführen zu müssen.

Per AJAX ist es so zum Beispiel möglich Daten im JSON- oder XML-Format asynchron von dem Server zu laden und an den Client zu übergeben. Neben SOAP[12], WSDL[13] bzw. RPC[14] kommt mittlerweile häufig das REST Paradigma zum Einsatz. Für die Client-Server Kommunikation mittels REST Paradigma wird auf das HTTP Protokoll mit seinen Methoden GET, POST, PUT und DELETE gesetzt. Mit diesen Methoden werden alle wichtigen Funktionen wie das Abrufen, Erstellen, Aktualisieren und Löschen von Daten gewährleistet. Es können sämtliche Dokument Typen wie PDF, Bilder oder HTML übertragen werden. Da eine REST Nachricht alle Informationen für die Kommunikation zwischen dem Client und dem Server enthält, wird dieses Protokoll auch als zustandslos bezeichnet. Das bedeutet, dass alle Anfragen als voneinander unabhängige Transaktionen behandelt werden, ohne in Bezug zu

[11] Über AJAX ist eine asynchrone Datenübertragung zwischen Browser und Server möglich. https://de.wikipedia.org/wiki/Ajax_(Programmierung), zuletzt abgerufen am 04.06.2016

[12] SOAP ist ein Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht werden können. https://de.wikipedia.org/wiki/SOAP, zuletzt abgerufen am 14.06.2016

[13] WSDL ist eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Netzwerkdienste (Webservices) zum Austausch von Nachrichten auf Basis von XML. https://de.wikipedia.org/wiki/Web_Services_Description_Language, zuletzt abgerufen am 14.06.2016

[14] RPC ist eine Technik zur Realisierung von Interprozesskommunikation. https://de.wikipedia.org/wiki/Remote_Procedure_Call, zuletzt abgerufen am 14.06.2016

vorherigen Anfragen zu stehen[15].

Durch eine Fat Client Architektur und einem responsive Webdesign, bei dem darauf geachtet wird, dass der Client unabhängig von der Displaygröße gut bedient werden kann und der Inhalt immer gut lesbar ist, ist es möglich Web-Apps zu entwickeln, die auch auf mobilen Endgeräten wie native Apps bedienbar sind. Im Unterschied zu nativen Apps haben Web-Apps jedoch den Nachteil immer von einer Internetverbindung abhängig zu sein. Dadurch kann es bei schlechtem Empfang oder langsamen Internetverbindungen zu Verzögerungen kommen, sowie im Ausland zu Roaming Gebühren führen. Eine Nutzung ohne Internetverbindung ist nur eingeschränkt möglich. So bieten Browser die Möglichkeit Daten als Cookie, im Web-Storage, in der IndexedDB oder einer Web SQL Database zu speichern. Dabei werden lediglich Cookies und der Web-Storage (siehe Abb. 2.3) von allen wichtigen Browsern und Versionen unterstützt. Für IndexedDB (siehe Abb. 8.1) und Web SQL Database (siehe Abb. 8.2) ist noch keine Browser übergreifende Funktionalität gewährleistet.

Abbildung 2.3: Browser Support für HTML5 Web-Storage (Can I Use - Web Storage 2016)

Die Speicherkapazität des Web-Storage ist je nach Browser auf eine Datenmenge von 5-10 MB[16] begrenzt. Ebenfalls bietet eine Web-App nur einen eingeschränkten

[15] Beispiel eines REST Web Services. http://www.oio.de/public/xml/rest-webservices.htm, zuletzt abgerufen am 30.05.2016

[16] Quelle: https://de.wikipedia.org/wiki/Web_Storage#Speichern_von_JSON-Objekten, zuletzt

Zugriff auf die Hardware Komponenten des Systems. Lediglich der Beschleunigungssensor[17], die Audio Hardware[18], Kamera[19] sowie die GPS Position[20] ist mit Einverständnis des Nutzers seit Einführung von HTML5 möglich. Dabei wird lediglich die Abfrage der GPS Position (siehe Abb. 2.4) browserübergreifend unterstützt und wird ebenfalls von vielen älteren Versionen der jeweiligen Browser unterstützt. Für den Beschleunigungssensor (siehe Abb. 8.3), die Audio Hardware (siehe Abb. 8.4) sowie die Kamera (siehe Abb. 8.5) ist nur eine eingeschränkte Browserkompatibilität gegeben.

Abbildung 2.4: Browser Support für HTML5 Geolocation (Can I Use - Geolocation 2016)

Webseiten unterscheiden sich dahingehend von Web-Apps, da Webseiten dem Nutzer ihren Inhalt zur Darstellung präsentieren so wie zum Beispiel ein Blog oder eine Nachrichtenseite dies tut, während der Nutzer mit Web-Apps direkt interagieren soll. Dieser Unterschied ist zum Beispiel bei dem Web-App Player[21] von Spotify ersichtlich, wenn dieser mit der Nachrichtenseite von die Zeit[22] verglichen wird. Die Benutzeroberfläche einer Web-App erinnert dabei eher an ein native App als an die normale Bedienung in einem Browser mit dem vor bzw. zurück Button. Der Autor Ciprian Borodescu fasst in seinem Text über „Web Sites vs. Web-Apps: What the experts

abgerufen am 05.06.2016

[17] Quelle: http://www.w3.org/TR/orientation-event/, zuletzt abgerufen am 15.06.2016
[18] Quelle: http://www.w3.org/TR/webaudio/, zuletzt abgerufen am 15.06.2016
[19] Quelle: http://www.w3.org/TR/mediacapture-streams/, zuletzt abgerufen am 15.06.2016
[20] Quelle: https://dev.w3.org/geo/api/spec-source.html, zuletzt abgerufen am 14.06.2016
[21] https://play.spotify.com/browse, zuletzt abgerufen am 03.06.2016
[22] http://www.zeit.de/index, zuletzt abgerufen am 03.06.2016

think“ folgende Punkte zusammen, die eine Web-App definieren:

„[...]

- *self-contained*
- *rich/interactive user interface, possibly mimicking the native UI of the device*
- *using advanced device capabilities – like geolocation, camera integration, or other technologies that the W3C Device Apis and Policy Working Group is developing*
- *action-oriented rather than information oriented*
- *not relying heavily on (or hiding when possible) the browser chrome (back button, reload button, address bar)*
- *working off-line, for example using HTML5 ApplicationCache, localStorage, or indexed database*

“ (Borodescu - Web Sites vs. Web Apps: What the experts think 2013)

2.1.3 Hybrid-App

Mittels bestimmter Frameworks ist es möglich aus einer Web-App eine „native App“ zu erstellen. Diese wird dann als Hybrid-App bezeichnet. Eine Hybrid-App wird wie native Apps über die jeweiligen App-Stores vertrieben und aktualisiert. Dafür wird zum Beispiel bei dem von Adobe entwickelten open source Framework PhoneGap[23] die, mit den Web-Technologien (JavaScript, HTML und CSS) entwickelten, Web-Apps durch das Framework in dem nativen Browser der jeweiligen mobilen Plattform wie zum Beispiel Android, Blackberry, iOS oder Windows Phone aufgerufen. Das Framework unterdrückt dabei die Menüleiste des jeweiligen Browsers und stellt eine Schnittstelle zur Verfügung, mittels der es möglich ist über JavaScript eine Kommunikation mit den Hardware- und Software Komponenten der jeweiligen Plattform herzustellen. Auf den mobilen Endgeräten kann so zum Beispiel auf Kontakte, Kamera, Bewegungssensor oder GPS zugegriffen werden. PhoneGap stellt native Apps für dessen Entwicklung bereit[24]. Diese nativen Apps stehen für die Betriebssysteme Microsoft Windows und Mac OS X zur Verfügung. Ebenfalls gibt es bereits verschiedene Boilerplates für die Entwicklung mittels PhoneGap, sowie npm[25] Pakete[26] für Node.js[27]. Der Nachteil einer Hybrid-App die mit PhoneGap erstellt wurde ist, dass Hybrid-Apps nicht so betriebssystemnah wie native Apps agieren und ein Browser immer als Zwischenschicht dient. Dies kann bei rechenintensiven Anwendungen wie

[23] http://phonegap.com, zuletzt abgerufen am 03.06.2016
[24] http://phonegap.com/products/#desktop-app-section, zuletzt abgerufen am 03.06.2016
[25] Npm ist ein Paketmanager für JavaScript. https://www.npmjs.com, zuletzt abgerufen am 03.06.2016
[26] https://www.npmjs.com/package/grunt-phonegap, zuletzt abgerufen am 03.06.2016
[27] Node.js ist eine JavaScript Laufzeitumgebung, die auf Chrome's V8 JavaScript Maschine aufbaut. https://nodejs.org, zuletzt abgerufen am 03.06.2016

zum Beispiel Spielen zu Geschwindigkeitsproblemen führen. Ebenfalls ist es möglich, dass im Vergleich zu nativen Apps längere Reaktionszeiten während der Bedienung der Benutzeroberfläche entstehen können. Diese sind aber nur geringfügig wahrzunehmen.

Das von GitHub entwickelte open source Framework Electron[28] welches zum Beispiel für deren Texteditor Atom[29] eingesetzt wird, basiert auf Chromium[30] und Node.js . So können ebenfalls die Web-Technologien HTML, CSS und JavaScript für die Entwicklung Hybrider-Apps im Desktopbereich genutzt werden. Die mittels Framework erstellten Hybrid-Apps sind Cross-Plattform fähig und laufen unter Mac OS X, Microsoft Windows und Linux. Dieses Framework stellt Funktionen zur Verfügung um Dialoge, Desktop Benachrichtigungen oder Menüs innerhalb des Frameworks zu erstellen, die den nativen Apps der jeweiligen Plattform gleichen. Ebenfalls lassen sich mit Electron Hybrid-Apps erstellen, die unter Microsoft Windows und unter Mac OS X in der Menüleiste neben der Uhr angezeigt werden sowie Desktop Benachrichtigungen erstellen können. Das Electron Framework lässt sich durch weitere npm Pakete erweitern, indem es nach der Installation des Paketes neu gebaut wird[31]. Dadurch ist es möglich das Framework Electron beliebig zu erweitern und Pakete zur Speicherung von Daten oder den Zugriff auf Hardware Komponenten wie zum Beispiel den USB bzw. Bluetooth Port zu installieren. Ein Nachteil von Electron ist sicherlich, dass beim Erstellen der Installationspakete[32] Chromium immer mit in die Hybrid-App übernommen werden muss. So ist eine Hybrid-App, erstellt mit dem Framework Electron, selbst bei minimal selbst geschriebenem Quellcode wie zum Beispiel für die Mac OS X Menüleiste neben der Uhr um die 40MB groß.

„[...] you are bundling Chromium with every update, compressed app size is around 40MB [...] “ (Pracucci - Electron? It works for us, and makes desktop fun and fast 2015)

[28] http://electron.atom.io, zuletzt abgerufen am 03.06.2016

[29] https://atom.io, zuletzt abgerufen am 03.06.2016

[30] Chromium ist ein open source Browser. http://www.chromium.org, zuletzt abgerufen am 04.06.2016

[31] Suz Hinton - Using node-serialport in an electron app (2015) http://meow.noopkat.com/using-node-serialport-in-an-electron-app/, zuletzt abgerufen am 03.06.2016

[32] Es lassen sich für alle unterstützten Betriebssysteme (Mac OS X, Microsoft Windows und Linux) Installationspakete mit nativem Installer generieren.

2.2 Zusammenfassung

Eine native App bietet neben der größtmöglichen Anbindung an die Hardwareschnittstellen des jeweiligen Betriebssystems eine systemübergreifende Kommunikation der nativen Apps untereinander. Es werden direkt Komponenten der Benutzeroberfläche des jeweiligen Betriebssystems genutzt, wie zum Beispiel Menüleisten oder Dialoge. Die Apps können direkt aus dem jeweiligen App-Store installiert und aktualisiert werden. Für jedes Betriebssystem wie zum Beispiel Microsoft Windows, Mac OS X, Android oder iOS müssen jeweils eigene Apps in verschiedenen Programmiersprachen implementiert werden. Dadurch entsteht ein hoher Programmieraufwand, der zu mehr Personenstunden, längerer Entwicklungszeit und somit zu höheren Kosten führt. Web-Apps sind plattformunabhängig, so lange auf dem jeweiligen Betriebssystem ein Browser installiert ist. Falls sie nach dem responsive Webdesign entwickelt wurden, sind sie auf allen Endgeräten wie Handys, Tablets oder Desktop Rechnern bedienbar. Es braucht also nur eine App entwickelt werden, um sämtliche Betriebssysteme und Endgeräte zu bedienen. Ebenfalls ist es möglich eine Web-App mit geringem Aufwand in eine Facebook-App bzw. Google Chrome-App zu konvertieren und diese im jeweiligen App-Store anzubieten. Die Vorteile einer Facebook bzw. Google Chrome-Apps wurden im Abschnitt [2.1] beschrieben. Es besteht nicht die Möglichkeit aus der Web-App heraus mit nativen Apps zu kommunizieren. Ebenfalls ist nur eine eingeschränkte Kommunikation mit den Hardware Komponenten des jeweiligen Systems möglich. Lediglich die GPS Position wird von fast allen Browsern und deren Versionen unterstützt. Hybrid-Apps schließen mittels Framework diese Lücke. So ist es einer Web-App möglich über die API eines Hybrid-App Frameworks eine Kommunikation mit den Hardwareschnittstellen des jeweiligen Systems herzustellen, die von einem Browser nicht direkt unterstützt werden. Ebenfalls ist darüber eine Kommunikation mit den nativen Apps des jeweiligen Systems möglich. Dadurch, dass Hybrid-Apps einen Browser zum Anzeigen der Web-App und die Framework Funktionen der API mit in die erstellte Hybrid-App übernehmen, benötigen Hybrid-Apps deutlich mehr Speicherplatz als die eigentliche Web-App. Ebenfalls ist es möglich, dass es während der Bedienung der Hybrid-App zu Latenzen kommt, da ein Browser immer eine Zwischenschicht zwischen der Web-App und dem Betriebssystem bildet.

3 Anforderungsanalyse

In diesem Kapitel sollen die Anforderungen an die zu erstellende Applikation definiert werden. Dafür sollen zunächst Personen definiert werden, die exemplarisch für die Zielgruppe dieser Applikation stehen. So genannte Personas. Mit einem Use Case Diagramm sollen daraufhin die Anforderungen an die Applikation visualisiert werden. Mittels User Stories sollen diese Anforderungen konkretisiert werden, um diese dann in funktionale und nicht - funktionale Anforderungen zu gliedern. Schließlich sollen ähnliche Projekte aus dem Bereich Streetart und Graffiti Fotos untersucht werden, um die in diesem Kapitel definierten Anforderungen mit den Funktionen von bestehenden Projekten zu vergleichen und gegebenenfalls Ideen von deren Design zu übernehmen.

3.1 Personas

In diesem Abschnitt sollen zwei Personas definiert werden, die exemplarisch für die Zielgruppe dieser Applikation stehen.

3.1.1 Mathis Schulz - 37 Jahre - Hamburg

„Früher habe ich selber Graffiti gemalt. Heute ist mein Interesse an Streetart und Graffiti noch immer vorhanden, aber meine Aktivität hat sich zu dem fotografischen Bereich hin verlagert.“

Abbildung 3.1: Mathis Schulz (37 - Hamburg)

Wer ich bin

- Angestellter Ingenieur seit 6 Jahren
- Feste Lebenspartnerin (Inga 35), seit 4 Jahren zusammenlebend
- keine Kinder
- doppeltes Einkommen

Meine Interessen

- Graffiti, Streetart, Fotografie, Musik (Jazz, Rap, Klassik, Metal, Reggae, Punk, Electro), Konzerte, Reisen, Filme, Serien, Fußball, Playstation, Schallplatten

- kaufen bzw. sammeln von Graffiti und Streetart Bildern aus aller Welt

Meine Persönlichkeit

- Intellektuell, entspannt, offen
- Arbeiten um Reisen zu finanzieren
- Arbeit macht Spaß aber Karriere wird nicht angestrebt
- Schaut kaum Fernsehen und wenn dann nur öffentlich-rechtliche Sender wie ARTE oder Phoenix

Meine Technik

- Vestax PMC-05 Mischpult, zwei Technics Plattenspieler, Maschine Mikro
- iPhone 6, Samsung Galaxy TAB S2 Tablet, Acer Aspire Notebook

Meine Zeitschriften, Blogs und Facebook-Likes

- Zeitschriften: Backspin, Stylefile, Zugzwang, Stern, die Zeit
- Blogs: Vice, RBNSHIT, Fatcap, Zeit Online, Der Postillon, ZDFHeute, Netzpolitik, iFun
- Facebook-Likes: StreetArtGlobe, Street Art School, urbanshit, Ableton Live, Streetart in Germany, iFun, Native Instruments, HHV.DE, FC St. Pauli, MenscH-Hamburg e.V., OZ Hamburg, Hamburg – meine Stadt, I fucking love science

Mein soziales Netzwerk

- Internet: Facebook, Instagram
- Familie: Eltern leben in einer kleinen Stadt. Älterer Bruder und jüngere Schwester. Leben ebenfalls in Hamburg. Beide haben bereits kleine Kinder
- Freunde: aus Studentenzeit, aus Schulzeit und Arbeitswelt, gemeinsamer Freundeskreis mit Lebenspartner, aus dem Fußballverein

Meine Reiseziele

- Deutschland, Jamaika, USA, Italien, Spanien, Frankreich, Peru, Japan

3.1.2 Tim Meier - 27 Jahre - Berlin

„Früher habe ich mich kaum für Graffiti interessiert aber die Streetart Kunst finde ich sehr spannend.“

Abbildung 3.2: Tim Meier (27 - Berlin)

Wer ich bin

- Student, arbeitet als Werkstudent in der Grafikabteilung einer Werbeagentur
- Single
- keine Kinder
- Eltern sind wohlhabend und finanzieren das Studium

Meine Interessen

- besucht gerne Foto- und Streetart Ausstellungen
- Konzerte, hochwertige Mode (Markenbewusstsein), Reisen, Kite-Surfen, Snowboard, Longboard, Musik (Electro, Hip Hop), Serien, Literatur, Clubs & Bars, gutes Essen und hochwertige Spirituosen

Meine Persönlichkeit

- Intellektuell, nett aber wirkt auf andere schnell arrogant durch Zurückhaltung, großer Freundeskreis, kommunikativ, macht gerne was für sich selbst
- Studiert um sich selbst zu verwirklichen
- Investiert viel Geld in seine Hobbys

Meine Technik

- iPhone 6S, iPad Pro Tablet, MacBook Air Notebook

Meine Zeitschriften, Blogs und Facebook-Likes

- Zeitschriften: brand eins, Neon, Horizont, Spiegel, Stern, die Zeit
- Blogs: brand eins, RBNSHIT, Zeit Online, ZDFHeute, Apfelpage, MacRumors, Grafikdesign
- Facebook-Likes: About U, Thomas Henry, Werben und Verkaufen, K17, Berghain / Panorama Bar, Clarks Shoes, Joko & Claas, Marteria, Vans, Longboard

Mein soziales Netzwerk

- Internet: Tinder, Facebook, Instagram, Pinterest, Yelp
- Familie: Einzelkind, Eltern leben in Potsdam
- Freunde: aus Studentenzeit, aus Schulzeit und Arbeitswelt, vom Sport, aus Clubs & Bars

Meine Reiseziele

- USA, Italien, Frankreich, Portugal, England, Schweden, Norwegen, Deutschland

3.2 UML Use Case

Für die zuvor definierten Personen können nun folgende Fälle in dem UML Use Case Diagramm dargestellt werden.

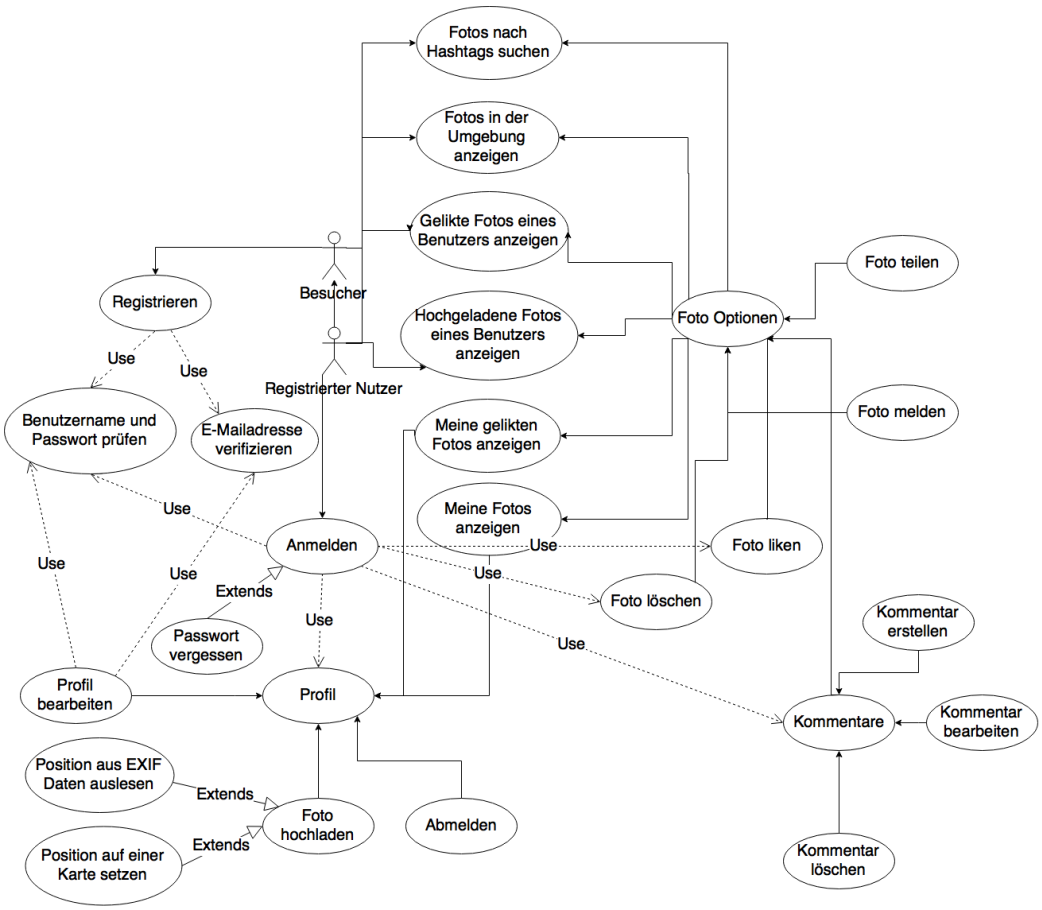

Abbildung 3.3: UML Use Case Diagramm der Applikation (eigene Darstellung)

3.3 User Stories

Anhand des UML Use Case Diagramms können nun folgende User Stories definiert werden.

- **Registrierung:** Als Nutzer möchte ich mich registrieren können, um zusätzliche Funktionalitäten wie den Upload von Fotos nutzen zu können.
  Akzeptanz-Kriterien:
  - Teste die Eingabe auf einen gültigen Benutzernamen (Mindestlänge 2, darf noch nicht registriert sein)
  - Teste die Eingabe auf ein gültiges Passwort (Mindestlänge 6)
  - Teste die Eingabe auf eine gültige E-Mail-Adresse (darf noch nicht registriert sein)
  - Überprüfe ob die E-Mail-Adresse dem Nutzer gehört
- **Anmeldung:** Als registrierter Nutzer möchte ich mich anmelden, um zusätzliche Funktionen wie den Upload von Fotos nutzen zu können.
  Akzeptanz-Kriterien:
  - Teste die Eingabe auf einen gültigen Benutzernamen (Mindestlänge 2)
  - Test die Eingabe auf ein gültiges Passwort (Mindestlänge 6)
- **Abmeldung:** Als registrierter Nutzer möchte ich mich von der Applikation abmelden können, damit kein anderer Nutzer meines Rechners auf die Applikation zugreifen kann.
- **Passwort zurücksetzen:** Als registrierter Nutzer möchte ich mein Passwort zurücksetzen können, falls ich dieses vergessen haben sollte.
  Akzeptanz-Kriterien:
  - Teste die Eingabe auf eine gültige E-Mail-Adresse (muss am Nutzer hinterlegt sein)
- **Suche nach Hashtags:** Als Nutzer möchte ich nach Fotos zu bestimmten Hashtags suchen und mir diese auf einer Karte und in einer Liste anzeigen lassen, um dorthin zu navigieren zu können bzw. einer Übersicht über die Fotos zu erhalten.
- **Fotos in der Umgebung:** Als Nutzer möchte ich mir auf einer Karte und in einer Liste Fotos in meiner Umgebung anzeigen lassen, um dorthin navigieren zu können bzw. um eine Übersicht über die Fotos zu erhalten.
- **Fotos eines Nutzers:** Als Nutzer möchte ich mir auf einer Karte und in einer Liste Fotos eines bestimmten Nutzers anzeigen lassen, um dorthin zu navigieren bzw. eine Übersicht über die Fotos zu erhalten.
- **Gelikte Fotos eines Nutzers:** Als Nutzer möchte ich mir auf einer Karte und in einer Liste die gelikten Fotos eines bestimmten Nutzers anzeigen lassen, um dorthin navigieren zu können bzw. eine Übersicht über die Fotos zu erhalten.

- **Merken der letzten Ansicht:** Als Nutzer möchte ich, dass meine letzte Ansicht wie Fotos in der Umgebung, Fotos eines bestimmten Nutzers sowie die gelikten Fotos eines bestimmten Nutzers gespeichert wird, damit ich mit der zuletzt gewählten Ansicht weitermachen kann, wenn ich die Applikation erneut öffne.
- **Carousel Ansicht:** Als Nutzer möchte ich mir die Fotos in ihrer kompletten Größe als Carousel Ansicht anzeigen lassen können, um durch die Fotos wechseln zu können.
- **Ein- bzw. Ausblenden der Seitenleiste:** Als Nutzer möchte ich die Seitenleiste mit den Fotos ein- bzw. ausblenden können, damit ich eine größere Ansicht der Karte erhalte.
- **Merken ob die Seitenleiste ein- bzw. ausgeblendet ist:** Als Nutzer möchte ich, dass gespeichert wird ob ich die Fotoübersicht ein- bzw. ausgeblendet habe damit ich mit der zuletzt gewählten Ansicht weitermachen kann, wenn ich die Applikation erneut öffne.
- **Zoomen:** Als Nutzer möchte in die Karte hinein- bzw. hinauszoomen können, um eine genauere Ansicht bzw. einen besseren Überblick über die Fotos in einem bestimmten Bereich zu bekommen.
- **Heatmap:** Als Nutzer möchte ich mir eine Heatmap anzeigen lassen können, um eine bessere Übersicht über die Verteilung der Fotos zu erhalten.
- **Kartentyp:** Als Nutzer möchte ich den Kartentypen wechseln können, um mir die Karte als ROADMAP, SATELLITE oder HYBRID anzeigen zu lassen.
- **Nutzer Position anzeigen:** Als Nutzer möchte ich meine aktuelle Position auf der Karte sehen können, damit ich weiß wo ich mich derzeit befinde.
- **Merken der letzten Position:** Als Nutzer möchte ich, dass meine letzte Position gespeichert wird, damit bei erneuten Öffnen der Applikation auch ohne GPS Verbindung eine Positionierung möglich ist.
- **Nutzer Position zentrieren:** Als Nutzer möchte ich meine aktuelle Position auf der Karte zentrieren können damit ich jederzeit wieder zu meiner aktuellen Position zurückfinden kann, wenn ich den Kartenausschnitt verschiebe.
- **Foto aus der Seitenleiste auf der Karte zentrieren:** Als Nutzer möchte ich ein Foto aus der Seitenleiste auswählen können um dieses auf der Karte zu zentrieren, damit ich weiß wo sich das Foto aus der Seitenleiste auf der Karte befindet.
- **Fotos melden:** Als Nutzer möchte ich Fotos melden können, falls diese Fotos nicht Streetart oder Graffiti zeigen bzw. doppelt vorhanden sind.
- **Fotos teilen:** Als Nutzer möchte ich Fotos in meinen sozialen Medien wie zum Beispiel Facebook oder Twitter teilen können.

- **Profil bearbeiten:** Als angemeldeter Nutzer möchte ich mein Profil bearbeiten können, um im Nachhinein meine E-Mail-Adresse, meinen Benutzernamen oder mein Passwort ändern zu können.
  Akzeptanz-Kriterien:
  - Teste die Eingabe auf einen gültigen Benutzernamen (Mindestlänge 2, darf noch nicht registriert sein)
  - Teste die Eingabe auf ein gültiges Passwort (Mindestlänge 6)
  - Teste die Eingabe auf eine gültige E-Mail-Adresse (darf noch nicht registriert sein)
  - Überprüfe ob die E-Mail-Adresse dem Nutzer gehört
- **Fotos sammeln:** Als angemeldeter Nutzer möchte ich meine Fotos auf den Server laden können, um diese anderen Nutzern zugänglich zu machen bzw. diese selber wiederfinden zu können.
  Akzeptanz-Kriterien:
  - Teste die Eingabe auf ein vorhandenes Foto
  - Teste die Eingabe auf vorhandene Koordinaten
- **Fotos bearbeiten:** Als angemeldeter Nutzer möchte ich meine Fotos bearbeiten können, um im Nachhinein die Koordinaten oder die Beschreibung ändern zu können.
- **Fotos löschen:** Als angemeldeter Nutzer möchte ich meine Fotos löschen können, um sie anderen Nutzern nicht mehr zur Verfügung zu stellen.
- **Gesammelte Fotos anzeigen:** Als angemeldeter Nutzer möchte ich mir auf einer Karte und in der Seitenleiste meine gesammelten Fotos anzeigen lassen können, um dorthin zu navigieren bzw. eine Übersicht über die Fotos zu erhalten.
- **Fotos liken:** Als angemeldeter Nutzer möchte ich Fotos Nutzern liken und unliken können.
- **Gelikte Fotos anzeigen:** Als angemeldeter Nutzer möchte ich mir auf einer Karte und in der Seitenleiste meine gelikten Fotos anzeigen lassen können, um dorthin zu navigieren bzw. eine Übersicht über diese Fotos zu erhalten.
- **Kommentar schreiben:** Als angemeldeter Nutzer möchte ich einen Kommentar zu einem Foto schreiben können.
- **Kommentar bearbeiten:** Als angemeldeter Nutzer möchte ich einen Kommentar zu einem Foto bearbeiten können.
- **Kommentar löschen:** Als angemeldeter Nutzer möchte ich einen Kommentar zu einem Foto löschen können.

3.4 Anforderungen an die Applikation

Aus dem UML Use Case Diagramm und den User Stories können nun die funktionalen und nicht - funktionalen Anforderungen definiert werden.

3.4.1 Funktionale Anforderungen

- Der Nutzer soll sich registrieren können. Dafür sollen die Eingabefelder vor dem Absenden auf ihre Gültigkeit überprüft werden. Die Felder dürfen nicht leer, die E-Mail-Adresse muss gültig, der Benutzername darf noch nicht vorhanden und das Passwort muss zwei Mal eingegeben werden und identisch sein. Ebenfalls soll das Passwort MD5 verschlüsselt an den Server übermittelt werden. Der Benutzername muss mindestens die Länge 2 und das Passwort die Länge 6 besitzen. Nach erfolgreicher Registrierung soll der Nutzer eine E-Mail an seine eingegebene E-Mail-Adresse erhalten, um über den Link in der E-Mail seine E-Mail-Adresse zu bestätigen. Der Nutzer soll sich mit seinen registrierten Daten anmelden, um weitere Funktionen nutzen zu können. Ein angemeldeter Nutzer soll seine Profildaten wie Benutzername, E-Mail-Adresse und Passwort ändern können. Wie bei der Registrierung sollen die Felder auf ihre Gültigkeit hin geprüft werden. Seine Profildaten samt Token sollen in der Applikation gespeichert werden, damit der Nutzer bei erneutem Start der Applikation ohne wiederholte Anmeldung bereits angemeldet ist. Der Token soll bei jedem Start der Applikation auf seine Gültigkeit geprüft werden. Sollte der Token nicht mehr gültig sein, soll der Nutzer automatisch abgemeldet werden. Falls ein Nutzer sein Passwort vergessen hat, soll ein Link mit einem Token an seine E-Mail-Adresse gesendet werden, über den der Nutzer sein Passwort ändern kann. Dafür muss die E-Mail-Adresse gültig und im System hinterlegt worden sein. Des Weiteren soll sich ein angemeldeter Nutzer vom System abmelden können.
- Die aktuelle Position eines Nutzers soll nach dessen Zustimmung über den Browser ausgelesen auf einer Karte dargestellt und in der Applikation gespeichert werden. Sollte die aktuelle Position nicht verfügbar sein, soll zunächst geschaut werden ob bereits eine ältere Position des Nutzers in der Applikation gespeichert wurde. Falls dies nicht der Fall ist, soll eine grobe Positionsbestimmung über IP geolocation[1] vorgenommen werden.
- Die Hauptansicht der Applikation soll in eine Karte und eine Seitenleiste mit einer Übersicht der, im aktuellen Kartenausschnitt vorhandenen, Fotos unterteilt werden. Die Seitenleiste soll ein- bzw. ausgeblendet werden können, damit der Nutzer sich eine vergrößerte Karte darstellen lassen kann. Die Marker sollen auf der Karte zunächst als Marker-Clusterer dargestellt werden. Der Nutzer soll die Möglichkeit besitzen, sich auf der Karte eine heatmap Schicht anzeigen zu

[1] https://www.iplocation.net, zuletzt abgerufen am 18.06.2016

lassen. Ebenfalls soll der Nutzer zwischen den Kartentypen ROADMAP, SATELLITE oder HYBRID wechseln zu können. Des Weiteren soll der Nutzer die Möglichkeit besitzen in die Karte hinein- und hinauszoomen zu können. Die ausgewählte Schicht sowie der Kartentyp, sollenvgl in der Applikation gespeichert werden, damit der Nutzer bei erneutem Öffnen der Applikation mit diesen Einstellungen fortfahren kann.

- Die Applikation soll dem Nutzer verschiedene Ansichten zur Verfügung stellen. Dieser soll die Möglichkeit besitzen sich entweder die Fotos in seiner Umgebung, die auf den Server geladenen Fotos eines bestimmten Nutzers, die gelikten Fotos eines bestimmten Nutzers, seine eigenen auf den Server geladenen Fotos oder die von ihm selbst gelikten Fotos anzuzeigen. Dabei soll für die letzteren beiden Ansichten eine vorherige Anmeldung vorausgesetzt werden. Die Ansichten der Fotos die ein bestimmter Nutzer gelikt oder auf den Server geladen hat sollen direkt über URLs via **/:username** und **/:username/likes** erreichbar sein, damit diese besser verlinkt werden können. Die zuletzt ausgewählte Ansicht soll in der Applikation gespeichert werden, damit der Nutzer nach erneutem Öffnen der Applikation zu dieser Ansicht zurückkehren kann. Ebenfalls sollen die Fotos in einem Dialogfenster als Carousel dargestellt werden. Hier soll sich der Nutzer durch alle Fotos klicken können, die bereits als Marker auf der Karte vorhanden sind. Die Bedienung des Carousels soll neben der Maus ebenfalls über die Tastatur möglich sein (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 40]). So soll es mit den Pfeiltasten links und rechts möglich sein, zwischen den Slides zu wechseln. Weiterhin soll der Nutzer die Möglichkeit besitzen über eine Suche die Fotos nach bestimmten Hashtags zu durchsuchen, wenn in diesen vorher eine Beschreibung hinterlegt worden ist.
- Ein Nutzer soll die Möglichkeit besitzen ein Foto zu melden, falls dieses nicht in die Kategorie Streetart oder Graffiti passt oder bereits vorhanden ist. Ein angemeldeter Nutzer soll zusätzlich ein Foto liken bzw. unlinken können. Ebenfalls soll ein Nutzer beliebige Fotos in seinen sozialen Medien wie zum Beispiel Facebook oder Twitter teilen können.
- Ein angemeldeter Nutzer soll ein Foto auf den Server laden können. Die Koordinaten und Blickrichtung des Fotos sollen aus den EXIF[2] Daten ausgelesen werden. Falls das Foto keine Koordinaten in den EXIF Daten gespeichert hat, soll die Position des Fotos auf einer Karte händisch gesetzt werden können. Zusätzlich soll der Nutzer eine Beschreibung dem Foto hinzufügen können. Die Beschreibung, sowie die Koordinaten des Fotos sollen im Nachhinein geändert werden können. Ebenfalls soll der Nutzer seine Fotos löschen können. Über einen Confirm-Dialog soll überprüft werden ob der Nutzer sein Foto tatsächlich löschen möchte.

[2] https://de.wikipedia.org/wiki/Exchangeable_Image_File_Format, zuletzt abgerufen am 18.06.2016

- Ein angemeldeter Nutzer soll Kommentare zu einem Foto hinterlegen können. Diese soll er im Nachhinein editieren und löschen können.

3.4.2 Nicht-funktionale Anforderungen

- Die Übertragung der Daten soll korrekt und ohne größere Verzögerungen erfolgen.
- Die Bedienung der Karte soll performant sein. Das Nachladen der Marker, das Wechseln zwischen den Schichten sowie die Bedienung der Karte wie das Hinein- und Hinauszoomen oder das Verschieben des Kartenausschnittes, soll ohne größere Verzögerungen möglich sein.
- Das Laden der Fotos und das Scrollen durch die Fotos in der Seitenleiste soll performant sein. Dazu sollen die Fotos über eine Paginierung nachgeladen werden können.
- Die Applikation soll ein modernes und ansprechendes Design besitzen, welches hinsichtlich der Benutzeroberfläche eine hohe Benutzerfreundlichkeit bietet und deren Bedienung der Nutzer auf Anhieb versteht.
- Die Applikation soll modular aufgebaut werden, damit sie ohne Probleme erweiterbar ist.
- Die Applikation soll auf möglichst allen Endgeräten bedienbar sein. Dazu zählen mobile Endgeräte ebenso wie Tablets bzw. Desktop Rechner. Das bedingt, dass die Applikation unabhängig der Bildschirmgröße gut bedienbar ist.
- Die Applikation soll stabil laufen, auch wenn der Nutzer unvorhergesehene oder falsche Eingaben tätigt.
- Die Dateien dieser Applikation sollen über eine Versionsverwaltung geteilt werden können.
- Es soll ein automatisiertes Deployment implementiert werden.

3.5 Vorstellung bestehender Projekte

Im Folgenden sollen nun Webseiten und Applikationen vorgestellt werden, die ebenfalls den Bereich Streetart und Graffiti Fotos behandeln. Dafür sollen das Konzept und die funktionalen Möglichkeiten der jeweiligen Webseite bzw. Applikation beschrieben werden. Dies bietet zum einen die Möglichkeit, die zuvor definierten Anforderungen an diese Applikation, mit den Funktionen bestehender Lösungen zu vergleichen und ebenfalls eine gute Möglichkeit um Ideen für die Planung des Designs der Benutzeroberfläche zu sammeln.

3.5.1 Geo Street Art

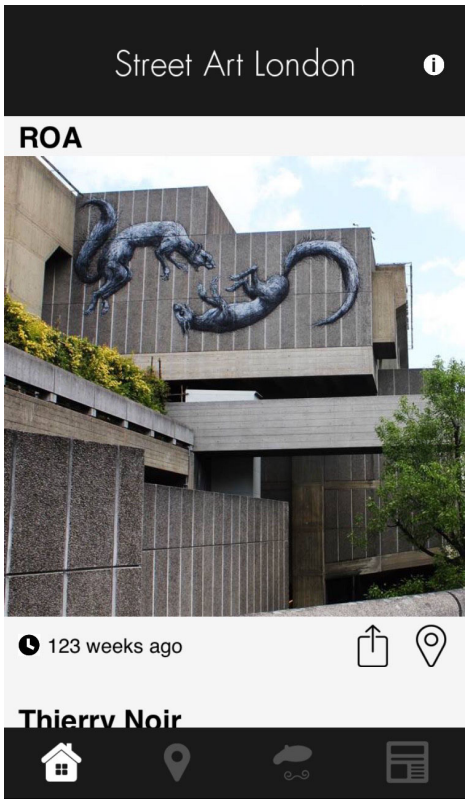

Abbildung 3.4: Screenshot der Geo Street Art iOS App

Geo Street Art[3] bietet seine Apps lediglich im App-Store an (siehe Abb. 3.4). Sie verfolgen dabei das Prinzip, für jede Stadt eine eigene App zu entwickeln. Bislang stehen Apps für die Städte London (kostenlos) und NYC (2,99€) zur Verfügung. Deren Konzept zielt darauf ab nur mit Autoren und Fotografen zu arbeiten, die sich sehr leidenschaftlich mit Streetart beschäftigen und sich in ihrer lokalen Szene sehr gut auskennen.

„We truly care about the street art scene and only work with authors and photographers who are extremely passionate about street art and truly embedded within their respective local scenes.“(Geo Street Art - Mission 2016).

Diese Autoren und Fotografen arbeiten direkt mit Geo Street Art zusammen und sollen neben den Fotos auch Hintergrundwissen zu den Künstlern und deren Kunst liefern. Der Nutzer der App ist also komplett abhängig von dem gelieferten Inhalt und hat keine Möglichkeit selber Einfluss auf den Inhalt zu nehmen bzw. eigenen Inhalt und Fotos beizusteuern.

Die App bietet neben der Möglichkeit eine Liste der neusten Fotos anzuzeigen, ebenfalls die Möglichkeit Fotos und deren Position auf einer Karte darstellen zu lassen. Des Weiteren bietet die App eine Liste der lokalen Künstler an, über die sich nach Auswahl eines Künstlers dessen Fotos bzw. die Positionen seiner Fotos auf einer Karte darstellen lassen. Zusätzlich ist es möglich dort Hintergrundinformationen zu diesem Künstler zu erhalten. Ein Blog mit aktuellen Veranstaltungen zu Streetart und Interviews mit Künstlern ist ebenfalls vorhanden. An jeder Stelle in der App ist es möglich die Fotos auf Twitter, Facebook oder per E-Mail zu teilen.

[3] http://www.geostreetart.com, zuletzt abgerufen am 27.05.2016

3.5.2 RedBull Streetart

Das Konzept von RedBull[4] ist es, jedem Nutzer die Möglichkeit zu bieten auf einer Google Maps-Karte die Position eines Streetart Fotos zu setzen. Dafür muss der Nutzer zunächst zu der Position auf der Karte zoomen, um dann das Google Streetview Icon auf der Karte zu platzieren und so in die Google Streetview Ansicht zu wechseln. Danach soll er in dieser Ansicht den besten Betrachtungswinkel auf das Streetart Foto auswählen, den Künstlernamen, den Bildtypen (zum Beispiel Graffiti oder Stencil) und eine Beschreibung eintragen um diese Ansicht dann zu speichern. Es werden also Bilder markiert, die zu einem bestimmten Zeitpunkt von den Google Streetview Autos aufgenommen wurden. Da die Lebensdauer von Graffiti und Streetart Bildern meist sehr kurz ist, da diese in hoher Frequenz übermalt werden, ist es also fraglich ob diese Bilder tatsächlich noch existieren. Des Weiteren werden keine mobilen Apps angeboten.

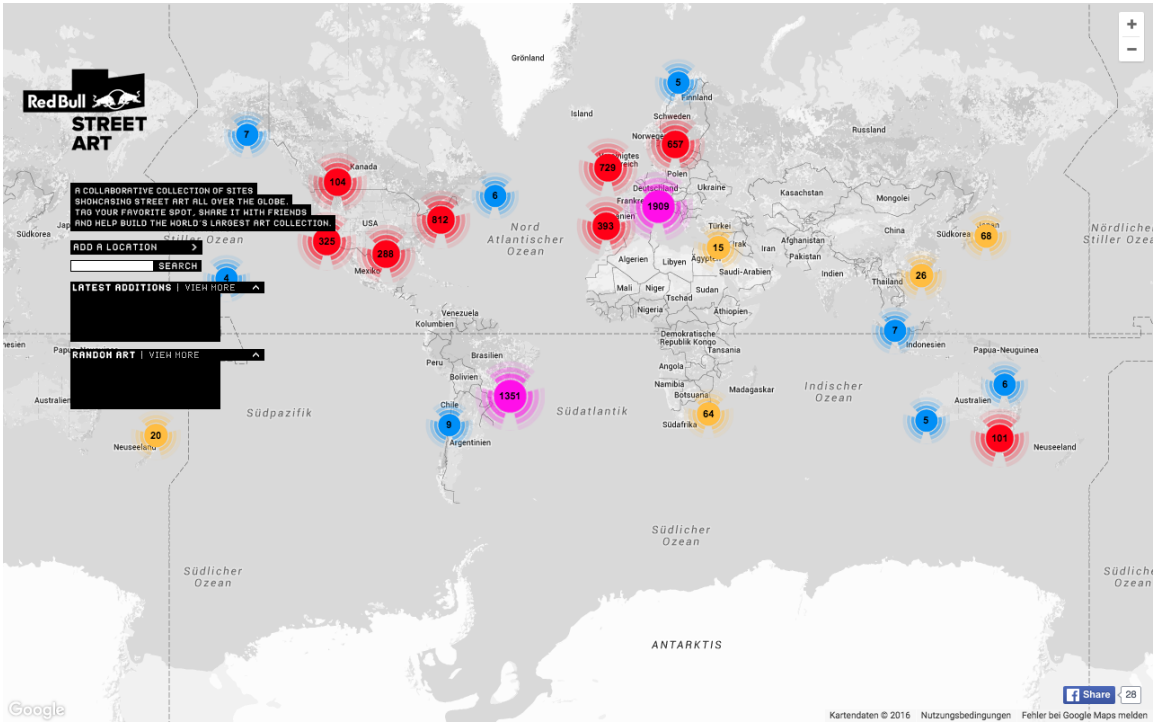

Abbildung 3.5: Screenshot der RedBull Street Art Web-App

Die Bilder werden zunächst als Marker-Clusterer auf der Karte angezeigt (siehe Abb. 3.5). Wird in die Karte hinein gezoomt, dann werden die Bilder als Marker dargestellt. Mit einem Klick auf den Marker wird in die Google Streetview Ansicht gewechselt, auf der neben der Ansicht des Künstlernamens, des Bild-Typs und der

[4] http://www.redbullstreetart.com, zuletzt abgerufen am 28.05.2016

Beschreibung ebenfalls die Möglichkeit besteht die aktuelle Ansicht direkt auf Twitter oder Facebook zu teilen. Neben der Möglichkeit auf der Karte nach einem bestimmten Künstler zu suchen, sollten ebenfalls die zuletzt hinzugefügten sowie zufällig ausgewählte Bilder angezeigt werden. Diese Darstellung war jedoch unter Mac OS X im Safari bzw. Google Chrome leer.

3.5.3 Streetartfinder

Streetartfinder[5] war ein Projekt der beiden Medieninformatik Studenten Patrick Schneider und Christopher Bogatzki für ein Seminar mit dem Thema "Digital Humanities" der Uni Regensburg [6]. Da das Feedback zu diesem Projekt in der Uni sehr positiv ausgefallen ist, haben die beiden Studenten beschlossen das Projekt privat weiterzuführen [7].

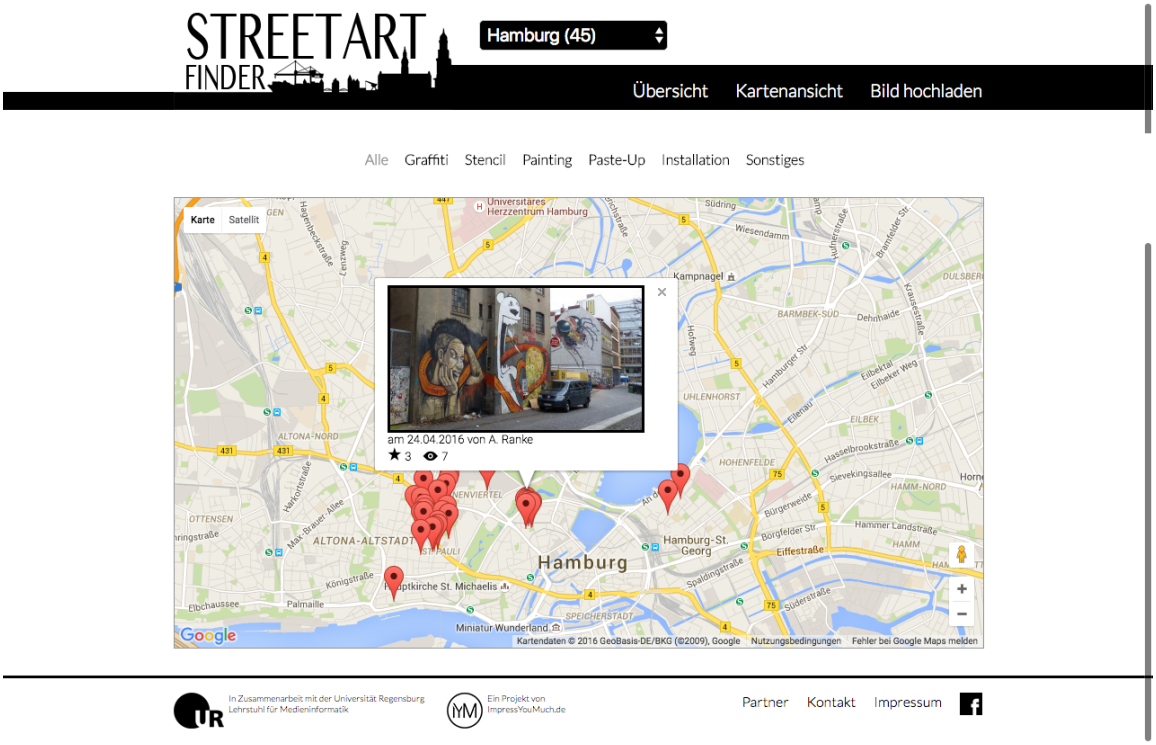

Abbildung 3.6: Screenshot der Streetartfinder Webseite

Auf der Webseite ist es möglich über eine Auswahlbox die Stadt auszuwählen.

[5] http://www.streetartfinder.de, zuletzt abgerufen am 28.05.2016
[6] Quelle: http://www.impressyoumuch.de/img/IYM_koelner-stadtanzeiger_27.08.14.jpg, zuletzt abgerufen am 28.05.2016
[7] Quelle: http://www.impressyoumuch.de/img/IYM_kult_12.03.14.jpg, zuletzt abgerufen am 28.05.2016

In einer Übersicht werden die Fotos in einer Liste nach verschiedenen Kategorien angezeigt. So kann zwischen den Kategorien Graffiti, Stencil, Painting, Paste Up, Installation und Sonstiges ausgewählt werden. Ebenfalls lassen sich die Fotos nach verschiedenen Kriterien sortieren. Es lässt sich nach den neusten Fotos, den Fotos mit den meisten Aufrufen und den Fotos, die am besten bewertet wurden sortieren. Neben der Übersicht dieser Fotos, lassen sich Fotos ebenfalls auf einer Google Maps-Karte (siehe Abb. 3.6) darstellen. In dieser Ansicht ist es ebenfalls möglich verschiedene Städte, sowie zwischen den Kategorien zu wählen.

Das Hochladen eines Fotos ist ohne vorherige Anmeldung möglich. Der Nutzer muss dazu ein Foto auswählen, seinen Namen eingeben, mit Komma getrennte Tags hinterlegen, ein oder mehrere Kategorien auswählen und kann zusätzlich eine Beschreibung hinterlegen. Für die Positionierung des Fotos muss auf einer Google Maps-Karte eine Stecknadel auf die Position des Fotos gesetzt werden. Die Seite ist responsive gehalten und kann auch auf mobilen Endgeräten genutzt werden. Eine Android sowie iOS App gibt es nicht.

3.5.4 Urbacolors

Die in Frankreich entwickelte Webseite Urbacolors[8] gibt es in der ersten Version seit 2011. Im Jahre 2014 hatte die Webseite bereits 7000 Nutzer und 17.000 mit Geotags versehene Fotos zu verzeichnen. Nutzer und Fotos waren in erster Linie in Frankreich zu finden, jedoch auch in New York, San Francisco, Frankfurt sowie Aserbaidschan. Die Webseite wurde daraufhin komplett überarbeitet, um ein intuitiveres Arbeiten zu ermöglichen und den Fokus mehr auf den Künstler zu legen. Fotos konnten nun einfacher kommentiert und geteilt werden[9].

Die Webseite bietet auf der Startseite zunächst eine Übersicht über die neusten, meist betrachteten und die am häufigsten gelikten Fotos. Weiterhin ist es möglich eine Übersicht der Künstler anzeigen zu lassen. Auf der Detailseite des Künstlers werden neben der Übersicht der Fotos dieses Künstlers ebenfalls Informationen zu der Anzahl seiner Fotos angezeigt, sowie in wie vielen Städten bzw. Ländern es Fotos von diesem Künstler gibt, wie oft seine Fotos betrachtet wurden, wie viele Kommentare bereits zu den Fotos hinterlegt und wie oft bereits Fotos dieses Künstlers gelikt wurden. Auf den Detailseiten jedes Fotos wird neben einer großen Ansicht des Fotos, die Möglichkeit geboten das Foto zu speichern, es zu liken, die Position des Fotos auf einer Google Maps-Karte anzuzeigen zu lassen oder eine Übersicht zu dem Nutzer zu erhalten. Neben dem Profilfoto wird angezeigt, wie viele Fotos der Nutzer bereits auf den Server geladen hat, aus wie vielen Städten bzw. Ländern diese Fotos kommen, wie oft die Fotos des Nutzers betrachtet, wie viele Kommentare zu seinen Fotos hinterlegt und wie oft die Fotos des Nutzers gelikt wurden. Auf der Profilseite eines Nutzers werden, neben den zuvor genannten Details, eine Übersicht all seiner Fotos angezeigt

[8] www.urbacolors.com, zuletzt abgerufen am 29.05.2016
[9] Siehe about auf der Webseite www.urbacolors.com/en/about, zuletzt abgerufen am 29.05.2016

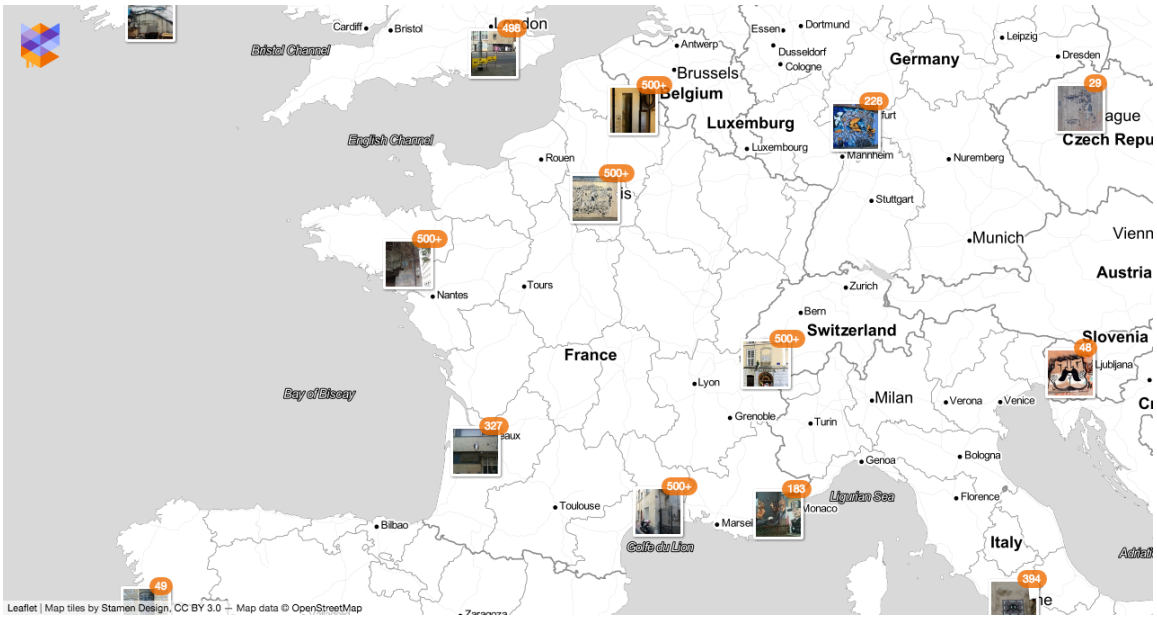

Abbildung 3.7: Screenshot der Urbacolors Web-App

und die Anzahl der Nutzer denen er folgt bzw. die ihm folgen.

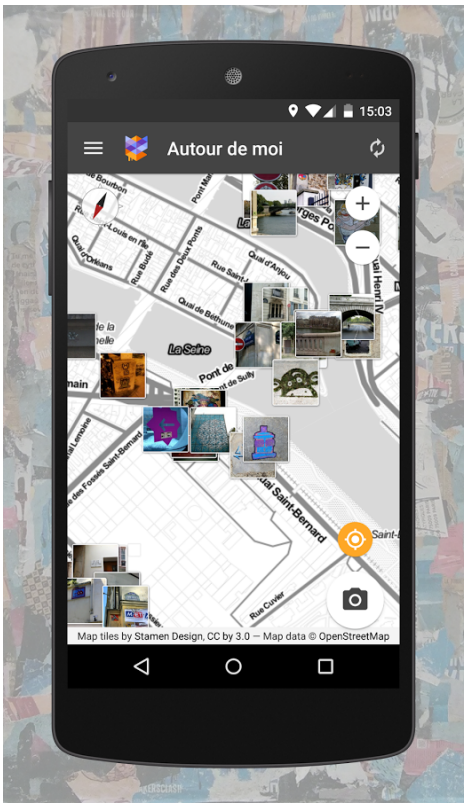

Abbildung 3.8: Screenshot der Urbacolors Android App

Als angemeldeter Nutzer ist es möglich einen Nutzer unter seine Favoriten aufzunehmen und ihm zu folgen. Ebenfalls ist es dann möglich Fotos auf den Server zu laden, Fotos zu liken oder anderen Nutzern zu folgen. Dafür muss der Nutzer zuvor mit Name, E-Mail-Adresse und Passwort registriert worden sein.

Urbacolors bietet zusätzlich die Möglichkeit die Position der Fotos auf einer Karte darstellen zu lassen (siehe Abb. 3.7). Daneben gibt es einen Blog in Französischer Sprache. Vernetzt ist Urbacolors mit Twitter, Facebook, Instagram, Pinterest und Vimeo.

Neben einer Google Chrome-App ist ebenfalls eine Android (siehe Abb. 3.8) und iPhone App erhältlich. Diese Apps sind in Französischer Sprache und nur über die französischen App-Stores zu beziehen.

4 Konzeption und Design

Aus den zuvor definierten Anforderungen an die Applikation ist es nun möglich, das Konzept und Design zu definieren.

Wie unter der Aufgabenstellung und Zielsetzung (siehe Abschnitt [1.2]) gefordert, soll die Applikation möglichst vielen Nutzern zugänglich gemacht werden. Aufgrund der Zusammenfassung der Grundlagen (siehe Abschnitt [2.2]) und der Anforderungsanalyse (siehe Kapitel [3]), habe ich mich deshalb für eine Umsetzung dieser Applikation als Web-App entschieden. Eine Web-App ist plattformunabhängig und kann somit vielen Nutzern zugänglich gemacht werden. Voraussetzung ist lediglich ein Browser. Dieser steht meist jeder Plattform und jedem Endgerät zur Verfügung. Eine Web-App bietet ebenfalls eine gute Grundlage, um sie in Zukunft in eine Google Chrome bzw. Facebook App zu konvertieren. Dadurch ist es möglich die Web-App einem breiteren Publikum bekannt zu machen. Nicht zuletzt ist eine hohe Zeitersparnis zu erwarten, da eine Web-App lediglich einmalig mit Web Technologien implementiert werden und nicht, wie im Falle von nativen Apps, in vielen verschiedenen Programmiersprachen umgesetzt werden muss. Dies erleichtert ebenfalls die Pflege und Erweiterung dieser Applikation. Durch HTML5 Geolocation (siehe Abb. 2.4) und HTML5 Web-Storage (siehe Abb. 2.3) sind die notwendigen Anbindungen an die Schnittstellen wie GPS, um die aktuelle Position des Nutzers auslesen zu können, und das lokale Speichern der Nutzerdaten gegeben und bieten eine Unterstützung für fast alle Browser und deren Versionen. Lediglich der Opera Mini 8 unterstützt kein HTML5 Web-Storage, was jedoch bei einem Marktanteil von 0.04 %[1] vernachlässigt werden kann. Als Einschränkung ist jedoch zu sagen, dass bei mobilen Endgeräten sowie Tablets ein direkter Zugriff auf die Kamera nicht möglich ist (siehe Abb. 8.5). Die Fotos können jedoch vom Dateisystem auf den Server geladen werden. Ein direkter Zugriff auf die Kamera ist jedoch nicht ausgeschlossen, da die Web-App zukünftig in eine Hybrid-App (siehe Unterabschnitt [2.1.3]) zum Beispiel mittels PhoneGap umgewandelt werden kann. Durch die PhoneGap API wäre dann ein direkter Zugriff auf die Kamera möglich. Mittels Electron wäre es ebenfalls möglich eine Hybrid-App für den Desktop Bereich zu erstellen.

[1] Quelle: http://www.webmasterpro.de/portal/webanalyse-aktuell.html, zuletzt abgerufen am 21.06.2016

4.1 Technische Umgebung

Die Web-App soll in einem Browser laufen und über das Internet mittels HTTP-Protokoll mit einem Server kommunizieren. Dafür stehen verschiedene Routen zur Verfügung, die über die Methoden GET, POST, PUT und DELETE angesprochen werden (siehe Abschnitt [4.2]). Das Routing wird serverseitig von Symfony 2[2] übernommen, welches in einem Docker[3] Container läuft. In dem Docker Container läuft nginx[4] in Kombination mit PHP-FPM[5]. Symfony 2 wiederum speichert die Daten in einer MongoDB[6], die ihrerseits wiederum in einem Docker Container läuft (siehe Abb. 4.1).

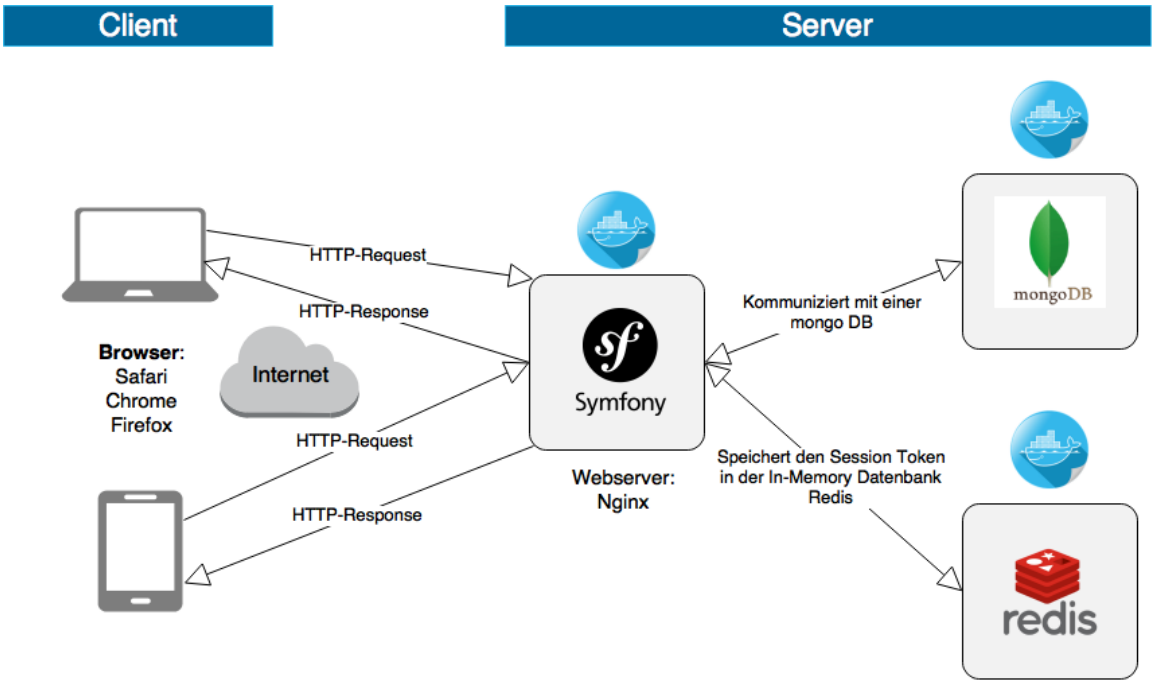

Abbildung 4.1: Client-Server Modell der Web-App (eigene Darstellung)

Einige Routen, wie zum Beispiel der Upload eines Fotos auf den Server benötigen

[2]Symfony 2 ist ein open source PHP Framework für die Entwicklung von Web Projekten. https://symfony.com, zuletzt abgerufen am 26.06.2016

[3]Über Docker können Anwendungen in einem isoliert laufenden virtuellen Container ausgeführt werden. Docker basiert auf einem Linux kernel und ist open source. https://www.docker.com, zuletzt abgerufen am 26.06.2016

[4]nginx ist ein HTTP und reverse proxy Server. http://nginx.org, zuletzt abgerufen am 26.06.2016

[5]PHP-FPM (FastCGI Process Manager) ist eine Alternative zu PHP FastCGI mit einigen zusätzlichen Features (besonders) für Websites mit hoher Last. http://php-fpm.org, zuletzt abgerufen am 26.06.2016

[6]MongoDB 2 ist eine open source NoSQL-Datenbank. https://www.mongodb.com, zuletzt abgerufen am 26.06.2016

eine Registrierung mit anschließender Anmeldung. Nach der Anmeldung erzeugt der Server einen Session Token, der an den Client übergeben wird und dort im Browser in einem Cookie gespeichert werden soll (siehe Abb. 4.3). Serverseitig wird der Session Token in einer Redis[7] Datenbank gespeichert, die ebenfalls in einem Docker Container läuft.

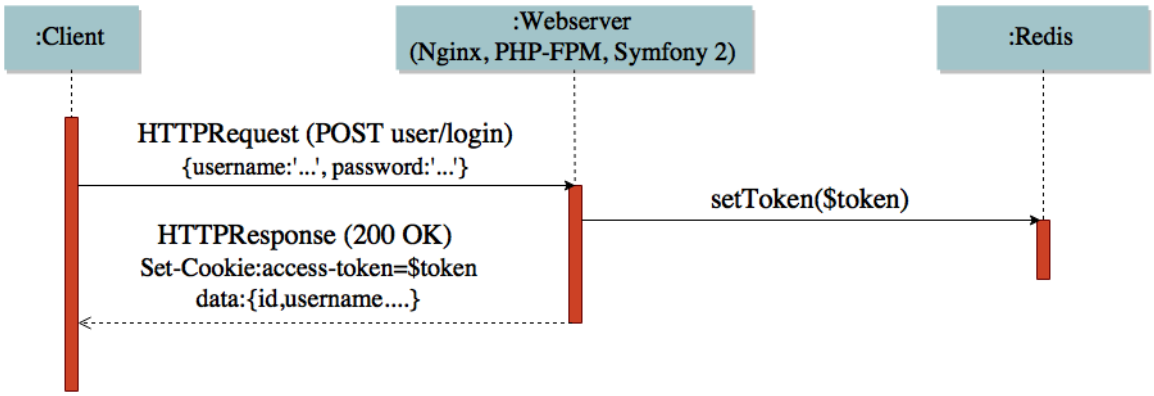

Abbildung 4.2: UML Sequenzdiagramm für den Anmeldeprozess dieser Web-App (eigene Darstellung)

Es stehen zwei verschiedene Umgebungen auf dem Server zur Verfügung. Zum einen die Live Umgebung, die später für alle Nutzer im Internet zugänglich sein wird und zum anderen die Development Umgebung, die lediglich für die Entwicklung genutzt wird. Jede dieser Umgebungen besitzt einen eigenen Docker Container für Symfony 2, MongoDB und Redis. Für das Routing stehen zwei Domains zur Verfügung:

1. Live: https://api.grafflr.com
2. Development: http://api-dev.grafflr.com

Für das Deployment (siehe Abschnitt [5.9]) der Web-App stehen ebenfalls zwei Domains zur Verfügung, über die die Web-App im Internet erreichbar sein soll:

1. Live: https://grafflr.com/app/#
2. Development: http://dev.grafflr.com/app/#

Die Domain der Development Umgebung ist durch eine .htaccess Datei mit einem Passwort geschützt.

[7] Redis ist eine open source In-Memory Datenbank. https://www.mongodb.com, zuletzt abgerufen am 26.06.2016

4.2 Routing

Das Routing erfolgt nach dem REST Paradigma. Auf der Abbildung (siehe Abb. 4.3) ist das Sequenzdiagramm exemplarisch für die Route **GET /images** zu finden. Ebenfalls sind dort alle derzeit verfügbaren Routen mit ihrer Funktion beschrieben. Die dort aufgezeigten Routen stehen derzeit zur Verfügung. Eine HTML Datei mit ausführlicher Beschreibung der Routen ist auf der CD unter **rest_api_doku/** zu finden (siehe Anhang [8.B]). Die unter User Stories (siehe Abschnitt [3.3]) definierten Anforderungen zum Schreiben, Bearbeiten und Löschen von Kommentaren sowie das Suchen nach Hashtags werden in dieser Version der Applikation zunächst nicht berücksichtigt, da die dafür benötigten Routen fehlen.

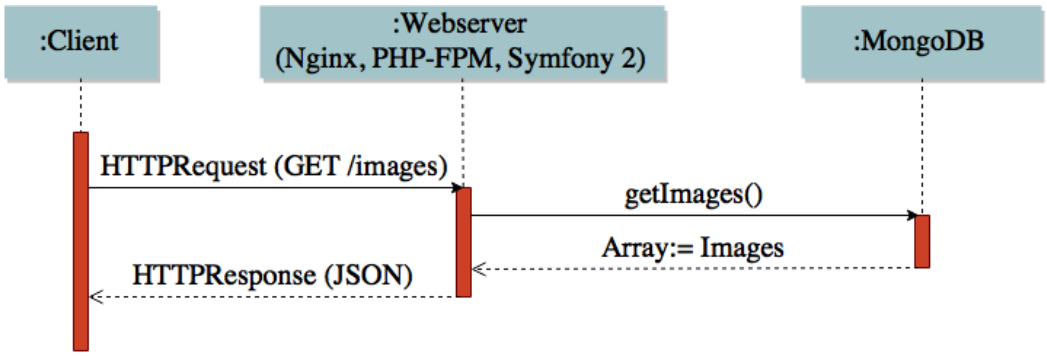

Image Routen

- **GET /images:** Holt alle Fotos in einer paginierten Struktur.
- **GET /images/geobox:** Holt alle Fotos in einer paginierten Struktur, die sich innerhalb eines Rechteckes definiert aus 2 Koordinaten befinden.
- **GET /images/{id}:** Holt ein einzelnes Foto anhand seiner ID.
- **POST /images:** Speichert ein Foto auf dem Server.
- **PUT /images/{id}:** Aktualisiert ein Foto anhand seiner ID.
- **DELETE /images/{id}:** Löscht ein Foto anhand seiner ID.
- **PUT /images/like/{id}:** LIKE/UNLIKE ein Foto anhand seiner ID.
- **POST /images/report/{id}:** Meldet ein Foto anhand seiner ID.

User Routen

- **GET /user:** Holt sich die Nutzerdaten anhand des Tokens, der in einem Cookie gespeichert ist. Der Cookie mit dem Token muss vorher durch die Anmeldung erstellt werden.
- **POST /user:** Erstellt einen Nutzer und sendet ihm per Mail einen Token über den er seine E-Mail-Adresse bestätigen kann.
- **POST /user/login:** Authentifiziert den Nutzer anhand seiner Anmeldedaten und gibt einen Cookie mit einem Token zurück.
- **GET /user/logout:** Meldet den Nutzer ab und löscht den Cookie mit dem Token.
- **POST /user/confirm/registration/{token}:** Bestätigt die E-Mail-Adresse des Nutzers anhand eines Tokens, der ihm bei der Registrierung per Mail zugesandt wurde.
- **POST /user/password/forgot:** Sendet dem Nutzer einen Token mit dem er sein Passwort zurücksetzen kann.
- **POST /user/password/reset/{token}:** Setzt das Passwort des Nutzers mittels Token zurück.
- **GET /user/confirm/email/{token}:** Bestätigt die neue E-Mail-Adresse eines Nutzers.
- **GET /user/profile/{username}:** Holt alle öffentlichen Profildaten eines Nutzers.
- **GET /user/profile/{username}/likes:** Holt alle gelikten Fotos eines Nutzers anhand seines Nutzernamen.
- **GET /user/profile/{username}/uploads:** Holt alle Fotos eines Nutzers anhand seines Nutzernamen.
- **POST /user/feedback:** Sendet ein Feedback Text an den Server.

Abbildung 4.3: Sequenzdiagramm des Routings anhand der Route GET /images. Auflistung aller Routen. (eigene Darstellung)

4.3 Design

Für die Umsetzung des Designs der Benutzeroberfläche sollen Vorschläge aus den Büchern *DON'T MAKE ME THINK* von Steve Krug [(Krug 2006)], *About Face: The Essentials of Interaction Design* von Alan Cooper [(Cooper 2014), sowie der Webseite GoodUI [(GoodUI - 75 GoodUI ideas 2016)] für die Entwicklung der Web-App berücksichtigt werden, sofern diese als sinnvoll erscheinen. Das Buch von Steve Krug wurde zwar bereits vor 10 Jahren geschrieben aber wird noch heute häufig zitiert, da sich bestimmte Paradigmen in der Erstellung von Benutzeroberflächen für Webseiten als dauerhaft für sinnvoll erwiesen haben und noch immer zum Einsatz kommen. Das Buch von Alan Cooper wurde im Jahr 2014 in der 4.Auflage veröffentlicht und beschäftigt sich mit der Gestaltung des Designs von Benutzeroberflächen für die Interaktion mit einem Nutzer. Neben den Benutzeroberflächen von nativen Applikationen auf Desktop Endgeräten, werden ebenfalls die Touch-basierten Oberflächen von mobilen Endgeräten, sowie von Webseiten untersucht. Die Webseite GoodUI liefert unter anderem nützliche Hinweise auf die Gestaltung von Elementen der Benutzeroberfläche wie Fomular-Elemente, Buttons oder Dialoge.

Da eine Implementierung als Web-App angestrebt wird, sollen die Design Paradigmen aus beiden Welten, einer Webseite und einer Applikation, berücksichtigt werden. Das Design der Web-App soll so gestaltet werden, dass der Nutzer die Benutzeroberfläche ohne viel nachzudenken sofort bedienen kann. Dabei soll die Bedienung selbsterklärend, nicht überladen und auf das wesentliche reduziert werden (vgl. [(Krug 2006), Kapitel 1]). Sollte es doch nötig sein an bestimmten Stellen der Oberfläche Hinweise auf die Bedienung zu geben, dann soll der Text auf ein absolutes Minimum reduziert werden (vgl. [(Krug 2006), Kapitel 3]). Auch andere Texte innerhalb der Web-App sollen auf ein Minimum reduziert werden (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 69]). Es soll ein klarer Aufbau der Benutzeroberfläche entwickelt werden, so dass der Nutzer sofort weiß, welcher Bereich der Oberfläche der Web-App welchen Zweck erfüllt. Dies ist dahingehend notwendig, da ein Nutzer die Oberfläche einer Applikation nicht liest, sondern lediglich überfliegt und bei der Bedienung der Benutzeroberfläche nicht die beste, sondern die schnellste Auswahl trifft, die für ihn am sinnvollsten erscheint (vgl. [(Krug 2006), Kapitel 3]).

Es soll dem Nutzer sofort ersichtlich sein, welche Teile der Benutzeroberfläche klickbar sind. Dabei sollen sich die Positionen der Menüs nicht verändern, damit der Nutzer sich merken kann, an welcher Stelle er welche Aktionen durchführen kann. Ebenfalls sollen die Wege zu den Untermenüs so gering wie möglich gehalten werden (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 62]). So soll die Auswahl der Nutzerfunktionen wie zum Beispiel das Ändern der Profildaten, das Laden eines Fotos auf den Server oder das Abmelden direkt als Untermenü bei Klick auf den Nutzernamen erscheinen. Ebenfalls soll für das Teilen eines Fotos auf Instagram, Twitter, Facebook etc. als Untermenü bei Klick auf den Teilen Button erscheinen. Generell soll viel mit Icons gearbeitet werden, da diese einen hohen Wiedererkennungswert besitzen und durch ihre Bilder generell bestimmten Funktionen zugeordnet werden können.

Der Nutzer soll bei der Eingabe von Daten nicht überfordert werden.

„The point is, we face choices all the time on the Web and making the choices mindless is one of the main things that make a site easy to use. “ [(Krug 2006), Seite 43]

So soll bei der Eingabe der Beschreibung, hinsichtlich des Fotos, bei dem Laden von Fotos auf den Server darauf verzichtet werden, dem Nutzer eine Klapp-Box zur Verfügung zu stellen, über die er eine Kategorie wie zum Beispiel Streetart, Graffiti oder Stencil auswählen kann. Dem Nutzer soll frei überlassen werden mit welchen Hashtags er das Foto versieht. Da die Suche später anhand der Beschreibung von statten geht, ist ein Filter oder ähnliches nicht vorgesehen. Eine Klappbox ermöglicht es dem Nutzer nicht auf Anhieb deren Auswahlmöglichkeiten zu überfliegen, ohne diese vorher zu öffnen (vgl. [(Krug 2006), Kapitel 7]).

Welche Ansicht der Nutzer gerade ausgewählt hat, zum Beispiel Fotos in der Umgebung oder auf den Server geladene Fotos eines Nutzers, soll im Header neben dem Logo angezeigt werden. Dies ist dahingehend sinnvoll, da Nutzer dies bereits von Breadcrumbs auf Internetseiten her kennen (vgl. [(Krug 2006), Kapitel 6]).

Um den Nutzer in seiner Aufmerksamkeit nicht zu überfordern, soll so wenig wie möglich mit Begrenzungen gearbeitet werden (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 23]). Dies soll lediglich erfolgen, wenn der Fokus eines Nutzers auf bestimmte Bereiche gelenkt werden soll, die von großer Bedeutung sind. So zum Beispiel auf den Inhalt einer Card, sowie als Begrenzung zwischen den Kommentaren. In diesem Fall jedoch nur eine sehr dezente, kaum sichtbare Begrenzung. Zu viele und zu stark ausgeprägte Begrenzungen können bei dem Nutzer ein Gefühl von Rauschen erzeugen, so dass er nicht weiß worauf er seinen Fokus legen soll (vgl. [(Krug 2006), Kapitel 3]). Ebenfalls soll mit ausreichend Abstand zwischen den Elementen gearbeitet werden, da dies den Inhalt und die Daten besser lesbar macht (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 63]).

Bestimmte Funktionen soll der Nutzer bereits ohne Registrierung und Anmeldung nutzen können, damit er einen Überblick über die Web-App erhält und deren Mehrwert für sich erkennen soll (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 22]). Falls ein Nutzer noch kein Foto auf den Server geladen hat oder noch keine Fotos gelikt hat und sich in einer seiner Ansichten dazu befindet, dann soll er ermutigt werden ein Foto auf den Server zu laden oder eines zu liken (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 25]). Da die Karte den wichtigsten Bestandteil der Web-App einnimmt, soll dieser der meiste Platz in der Benutzeroberfläche eingeräumt werden. Es soll ein responsive Design implementiert werden, damit die Web-App auch auf mobilen Endgeräten gut bedienbar ist(vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 70]).

4.3.1 Formulare

- Es sollen so wenig Eingabefelder wie nötig genutzt werden. So soll der Nutzer bei der Registrierung lediglich E-Mail-Adresse, Nutzername und Passwort eingeben (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 13]) müssen.
- Dem Nutzer sollen zu den Eingabefeldern Hinweise angezeigt werden. So zum Beispiel warum er ein bestimmtes Eingabefeld ausfüllen bzw. was er dabei beachten sollte (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 68]).
- Es sollen keine Klappboxen verwendet werden (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 14]).
- Die Formularfelder sollen in ihrer Größe so definiert werden, dass sie auf mobilen Endgeräten und Desktop Rechnern gut bedient werden können.
- Die Formulare sollen klar strukturiert werden, ohne Zwischenlinks die den Nutzer ablenken könnten (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 16]).
- Für die Anordnung der Buttons innerhalb der Formulare, sollen immer die gleichen Konventionen eingehalten werden. So soll sich zum Beispiel die Anordnung der Buttons wie OK oder CANCEL nicht in ihrer Anordnung und Position verändern (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 29]).
- Die Hinweise zu der Überprüfung auf eine korrekte Eingabe innerhalb der Eingabefelder, soll direkt am Eingabefeld selber erfolgen. So soll der Hinweis auf eine Falscheingabe nicht oben im Formular, sondern direkt am Eingabefeld angezeigt werden (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 33]).

4.3.2 Buttons und Tooltips

- Auch wenn GoodUI empfiehlt Icon Buttons mit Labels zu versehen (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 47]), so soll dennoch zunächst darauf verzichtet und mit Tooltips gearbeitet werden. Für erfahrene Nutzer sind Icon Buttons meist völlig ausreichend und haben einen hohen Wiedererkennungswert. Weniger erfahrene bzw. neue Nutzer bekommen anhand der Tooltips die nötigen Hinweise über die Funktion der Icon Buttons (vgl. [(Cooper 2014), Kapitel 18]). Dies muss jedoch bei der mobilen Ansicht überdacht und soll ebenfalls in der Evaluation abgefragt werden.
- Auf den Buttons die nicht mittels Icon gekennzeichnet sind, soll die Bezeichnung möglichst kurzgehalten werden, so dass der Nutzer über deren Funktion nicht nachdenken muss.
- Der Nutzer soll gleich erkennen, dass es sich um einen Button handelt.
- Alle Aktionen sollen mit maximal zwei Klicks erreichbar sein.
- Die Buttons für die Aktionen, die an einem Foto durchgeführt werden können, sollen sich direkt an diesem befinden. Dies soll ebenfalls für die Kommentare gelten (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 19]).

- Die Buttons sollen in ihrer Größe, Form und Farbe konsistent gehalten werden, damit sich bei dem Nutzer ein Lerneffekt einstellt (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 27]).
- Zusammengehörige Buttons wie zum Beispiel die Bedienung der Karte, sollen sich in demselben Bereich der Benutzeroberfläche befinden und sich nicht über diese verteilen (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 32]).
- Um die Benutzeroberfläche nicht zu überladen, sollen einige Buttons mehrere Funktionen erfüllen. So soll es lediglich einen Button geben, um zwischen den verschiedenen Kartentypen zu wechseln oder die Heatmap zu aktivieren bzw. zu deaktivieren (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 46]).

4.3.3 Dialoge

- Dialoge sollen für die Registrierung, die Anmeldung, den Login, das Laden von Fotos auf den Server und die Carousel Detailansicht des Fotos eingesetzt werden.
- Bestätigungsdialoge sollen nur dann angezeigt werden, wenn eine Entscheidung des Nutzers in diesem Moment notwendig ist (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 45]).

4.3.4 Notifikationen

- Der Nutzer soll ein Feedback bekommen, wenn er eine bestimmte Aktion durchgeführt hat. Dies soll in Form von Notifikationen passieren (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 61]).
- Statt den Nutzer lediglich zu informieren, dass er eine Aktion erfolgreich durchgeführt hat, soll ihm dafür gedankt werden, dass er sich zum Beispiel registriert oder ein Foto auf den Server geladen hat. (vgl. [(GoodUI - 75 GoodUI ideas 2016), idea 52]).

4.3.5 Cards und infinite scrolling

Cards werden zum Beispiel von Facebook sowie LinkedIn eingesetzt, um Inhalt und Medien wie Fotos in Kombination mit sozialen Aktionen wie zum Beispiel Liken oder Teilen darzustellen. Google Now setzt ebenfalls auf Cards aber eher um kontextabhängige Inhalte wie Wetter oder Karten darzustellen. Cards werden häufig in Kombination mit vertikalem Scrollen bzw. in der Grid Darstellung eingesetzt und bieten ein gute Möglichkeit in sich geschlossenen Inhalt darzustellen (vgl. [(Cooper 2014), Kapitel 19]). Aus diesen Gründen sollen Cards in der Sidebar der Web-App in Kombination mit infinite scrolling eingesetzt werden. Jedoch auch in dem Info-Window über dem Marker und in der Carousel Ansicht zum Einsatz kommen. Infinite scrolling sollte mit Bedacht eingesetzt werden, da ein Nutzer niemals bis zum Ende scrollen

wird. Ebenfalls kann es für einen Nutzer frustrierend sein ein Foto erneut zu suchen, wenn er vorher bereits länger scrollen musste (vgl. [(Cooper 2014), Kapitel 20]) um dies zu finden. Aus meiner Sicht scheint dies jedoch vertretbar, da der Nutzer ebenfalls die Möglichkeit erhält sich die Fotos auf der Karte oder im Carousel anzeigen zu lassen.

4.3.6 Mock-ups

Die Mock-ups wurden mit Draw.io[8] erstellt und befinden sich im Anhang (siehe Abb. 8.6, 8.7, 8.8)

[8] draw.io (ehemals Diagramly) ist eine kostenlose Online-Diagramm-Software https://www.draw.io, zuletzt abgerufen am 01.07.2016

5 Umsetzung und Implementierung

Anhand der Vorgaben aus dem vorherigen Kapitel ist es nun möglich die Web-App zu implementieren. In diesem Kapitel werden die Technologien beschrieben, die für die Entwicklung dieser Applikation genutzt wurden und warum. Ebenfalls werden einige Prozesse vorgestellt, die für die Entwicklung und das Deployment wichtig sind.

Da es sich bei der Entwicklung dieser Applikation um eine Web-App gehandelt hat, standen zunächst **HTML**, **CSS** und **JavaScript** zur Verfügung. Für die Entwicklungsumgebung wurden über den **Node.js** Paketmanager **npm** (siehe Abschnitt [5.1]) weitere Pakete wie zum Beispiel **Grunt** (siehe Abschnitt [5.2]) installiert. Grunt sowie dessen Erweiterungen übernehmen verschiedene Aufgaben, wie zum Beispiel das Bauen der Web-App und dessen Deployment auf den Server. Der Paketmanager **Bower** (siehe Abschnitt [5.3]), wurde eingesetzt um verschiedene Pakete für die Webentwicklung zu installieren. So wurde darüber zum Beispiel das **AngularJS** (siehe Abschnitt [5.4]) und **Angular Material** (siehe Abschnitt [5.5]) Framework installiert. Statt die Services von AngularJS zu nutzen, wurden diese als Klassen in **ECMA Script 6**[1] umgesetzt. Das Klassendiagramm befindet sich auf der CD unter **klassendiagramm.png** (siehe Anhang [8.B]). Davon befinden sich die Klassen Dialog, File, Image, Marker und User in dem Repository dieser Web-App und die Klassen DjGmap, DjLocalstorage, DjLog, DjResource, MdDialog und MdToast in einem separaten Repository, dass dem Repository dieser Web-App als Submodul hinzugefügt wurde. Es gibt lediglich einen Controller **MainCtrl** (zu finden auf der CD in der Datei **webapp/app/scripts/controller/main.js** - siehe Anhang [8.B]), der bei Start der Web-App zusammen mit dem **Main Template** (zu finden auf der CD in der Datei **webapp/app/view/main.html** - siehe Anhang [8.B]) aufgerufen wird (siehe Abb. [5.1]).

Dieser Controller erstellt Objekte von allen relevanten ECMA Script 6 Klassen und stellt die Schnittstelle zwischen dem Angular.js Framework und diesen Klassen dar. So werden alle Module des Angular.js Frameworks und seiner Erweiterungen über den Controller an die ECMA Script 6 Klassen übergeben. ECMA Script 6 wurde eingesetzt, da dadurch eine weitgehende Unabhängigkeit von Angular.js bestehen bleibt. So ist es zum Beispiel möglich, Angular.js durch ReactJS[2] zu ersetzen oder gänzlich auf Frameworks zu verzichten. Das Definieren von eigenen Styles, sowie das

[1] ECMA Script 6 ist eine Programmierspezifikation, die unter anderem von JavaScript implementiert wird. https://en.wikipedia.org/wiki/ECMAScript, zuletzt abgerufen am 08.07.2016

[2] ReactJS ist eine von Facebook entwickelte JavaScript-Bibliothek. https://facebook.github.io/react/, zuletzt abgerufen am 11.07.2016

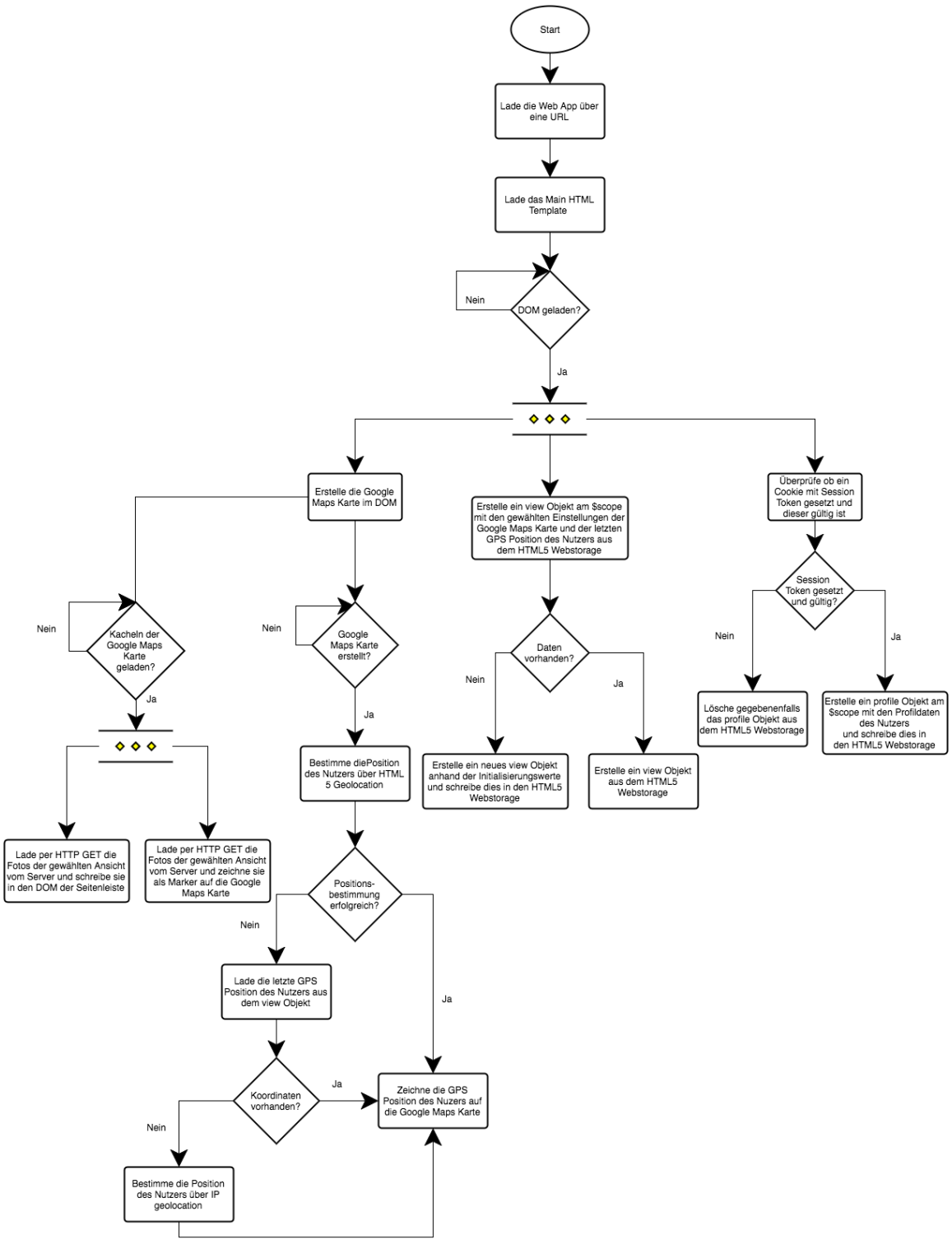

Abbildung 5.1: Der Programmablaufplan des MainCtrl bei Start der Web-App (eigene Darstellung)

teilweise Überschreiben der Styles des Angular Material Designs, wurde über **SASS**[3] umgesetzt. Die Kompilierung von SASS nach CSS erfolgt über das npm Grunt Modul **Compass**[4]. Die Entwicklung der Kartendarstellung und deren Bedienung wurde über die **Google Maps API** (siehe Abschnitt [5.6]) umgesetzt. Als Erweiterungen für die Google Maps-Karte wurden die **Geolocation** Komponente zur Darstellung der Nutzerposition und die **Marker-Clusterer** Komponente zur gruppierten Ansicht der Marker installiert. Für die Icons wurde das CSS Toolkit **Font Awesome**[5] über Bower installiert. Die Dateien der Schriftarten befinden sich auf der CD (siehe Anhang [8.B]) im Ordner **webapp/app/fonts/**. Ebenfalls über Bower wurden die JavaScript-Bibliotheken **jQuery**[6] und **Underscore**[7] installiert, um bestimmte Funktionen wie zum Beispiel **$.each()** oder **__.size()** nutzen zu können. Die Angular.js Module **MD5**[8] und **Moment.js**[9] wurden ebenfalls über Bower installiert, um zum einen das Passwort MD5 verschlüsseln zu können (siehe Unterabschnitt [3.4.1]) und zum anderen Zugriff auf verschiedene Funktionen zum Bearbeiten eines Datums zu erhalten. Um das Auslesen der EXIF Daten eines Fotos zu gewährleisten (siehe Unterabschnitt [3.4.1]), wurde die Bibliothek **exifjs**[10] installiert. Für die Versionskontrolle wurde **Git** eingesetzt (siehe Abschnitt [5.8]). Die Konfiguration der Web-App befindet sich auf der CD (siehe Anhang [8.B]) im Ordner **webapp/app/scripts/app.js**. Dort sind die REST API Routen, die Konfiguration der Google Maps-Karte und eine allgemeine Konfigurationen definiert. Die Bilder der Web-App wie zum Beispiel das Marker Icon oder das Logo befinden sich auf der CD (siehe Anhang [8.B]) im Ordner **webapp/app/images/**.

[3] SASS ermöglicht eine strukturierte Schreibweise von CSS Code und bietet zum Beispiel die Möglichkeit Variablen zu nutzen. http://sass-lang.com, zuletzt abgerufen am 08.07.2016

[4] Compass ist ein SASS zu CSS Compiler für Grunt. https://github.com/gruntjs/grunt-contrib-compass, zuletzt abgerufen am 08.07.2016

[5] Font Awesome ist ein CSS Toolkit um Icons über eine Schriftart darzustellen. http://fontawesome.io, zuletzt abgerufen am 08.07.2016

[6] jQuery ist eine JavaScript-Bibliothek. https://jquery.com, zuletzt abgerufen am 08.07.2016

[7] Underscore ist eine JavaScript-Bibliothek. http://underscorejs.org, zuletzt abgerufen am 08.07.2016

[8] angular-md5 ist ein MD5 Konverter für Angular.js. https://github.com/gdi2290/angular-md5, zuletzt abgerufen am 08.07.2016

[9] angular-momentjs ist eine Bibliothek für Angular.js, über die verschiedene Datumsfunktionen von Moment.js genutzt werden können. https://github.com/urish/angular-moment, zuletzt abgerufen am 08.07.2016

[10] exif-js ist eine JavaScript-Bibliothek, die verschiedene Funktionen zum Auslesen von EXIF Daten wie zum Beispiel Koordinaten bietet. https://github.com/exif-js/exif-js, zuletzt abgerufen am 08.07.2016

5.1 Node.js und npm

Node.js ist eine JavaScript Laufzeitumgebung, in die bereits der Paketmanager npm vorinstalliert ist (siehe Unterabschnitt [2.1.3]). Über den Befehl

```
npm install <package> --save
```

können Pakete installiert und durch das –**save** der Name des Paketes inklusive seiner Versionsnummer zusätzlich in eine **package.json**[11] geschrieben werden. Die package.json wird mit in das Git Repository übernommen. So ist es anderen Entwicklern möglich über den Befehl

```
npm install
```

alle für die Entwicklung nötigen Pakete zu installieren. Die installierten Pakete können über

```
npm update <package>
npm update
```

aktualisiert werden. Mit dem Befehl in Zeile eins werden alle Pakete aktualisiert und mit dem Befehl in Zeile zwei wird das jeweilige Paket aktualisiert. Dieser Paketmanager wurde genutzt, um verschiedene Pakete für die Entwicklung wie zum Beispiel Grunt (siehe Abschnitt [5.2]), sowie dessen Erweiterungen zu installieren. Die package.json Datei für dieses Projekt ist auf der CD (siehe Anhang [8.B]) unter **webapp/package.json** zu finden.

[11] Struktur einer package.json. https://docs.npmjs.com/files/package.json, zuletzt abgerufen am 05.07.2016

5.2 Grunt

Grunt[12] bietet mit seiner **Gruntfile.js** eine Datei, in der verschiedene JavaScript Tasks definiert und über **grunt «task»** aufgerufen werden können. Die Gruntfile.js befindet sich auf der CD unter **webapp/Gruntfile.js** (siehe Anhang [8.B]). Grunt wurde für die Automatisierung des Deployments, das Livereloading während der Entwicklung und weitere Prozesse eingesetzt. Im Folgenden ist beispielhaft der **build** Task (siehe Listing [1]) dargestellt. Dieser Task übernimmt das Bauen der Web-App für das spätere Deployment und teilt sich in einige Unteraufgaben auf, die im Einzelnen erläutert werden.

```
1  grunt.registerTask('build', [
2          'clean:dist',
3          'wiredep',
4          'useminPrepare',
5          'concurrent:dist',
6          'autoprefixer',
7          'concat',
8          'browserify',
9          'ngAnnotate',
10         'copy:dist',
11         'cssmin',
12         'uglify',
13         'filerev',
14         'usemin',
15         'htmlmin'
16 ]);
```

Listing 1: Der Grunt build Task

- **clean:dist** löscht den .tmp Ordner
- **wiredep** fügt die JavaScript Bower Pakete in die index.html ein.
- **useminPrepare** ist die Konfiguration für usemin. Liest aus der index.html HTML-Blöcke mit den Pfaden zu JavaScript sowie CSS-Dateien aus. Die ausgelesenen Pfade werden in die Optionen der in useminPrepare definierten Prozesse wie **concat**, **uglifyjs** und **cssmin** eingefügt. So brauchen diese Prozesse nicht zusätzlich konfiguriert werden.
- **concurrent:dist** führt 3 Tasks parallel aus (compass:dist, imagemin, svgmin). **compass: dist** kompiliert die SASS in CSS Dateien. **imagemin** reduziert die Dateigrößen der .jpg und .png Bilder. **svgmin** reduziert die Dateigrößen der .svg Bilder.

[12] Grunt ist ein JavaScript Task-Runner. http://gruntjs.com, zuletzt abgerufen am 10.07.2016

- **autoprefixer** kompiliert die CSS-Dateien und fügt automatisch browserspezifische Styles hinzu, wenn diese in der jeweiligen CSS Klasse fehlen.
- **concat** fügt alle JavaScript Dateien, die in useminPrepare gefunden wurden, zu einer Datei zusammen.
- **browserify** nutzt das Grunt Packet **babelify** um die in ECMA Script 6 geschriebenen Klassen nach ECMA Script 2015 zu kompilieren. Dies ist nötig, damit auch ältere Browser wie der Internet Explorer 9 den JavaScript Code interpretieren können.
- **ngAnnotate** kümmert sich um die dependency injection von Angular.js
- **copy:dist** kopiert alle relevanten Dateien in den **webapp/dist/** Ordner.
- **cssmin** verkleinert die CSS-Dateien.
- **uglify** optimiert und verkleinert den JavaScript Code.
- **filerev** ändert die Dateinamen der Bild, CSS und JavaScript Dateien in einzigartige Namen, um dadurch zu verhindern dass alte Dateien aus dem Cache des Browsers geladen werden.
- **usemin** - ändert in den HTML Dateien den, durch filerev neu erzeugten, Dateinamen der JavaScript und CSS-Dateien. In den JavaScript und CSS-Dateien werden zusätzlich die Dateinamen der Bilder ersetzt.
- **htmlmin** - verkleinert die HTML Dateien.

Für die Entwicklung dieser Web-App wurden verschiedene Grunt-Tasks genutzt, die wie folgt über die Konsole aufgerufen werden können (siehe Listing [2]).

```
1 grunt serve
2 grunt deploy-app
3 grunt deploy-app-dev
```

Listing 2: Grunt-Tasks, die für die Entwicklung und das Deployment genutzt wurden

In der Reihenfolge ihrer Zeilen erfüllen diese Tasks folgende Funktionen:

1. Führt zunächst den oben beschriebenen **build-Task** aus und erstellt einen Server unter **http://localhost:8000** mit Zugriff auf den **webapp/dist/** Ordner. Ebenfalls wird ein **watch-Task** ausgeführt, der die HTML, SASS und JavaScript Dateien auf Änderungen hin beobachtet und bei Änderungen in einer der jeweiligen Dateien die notwendigen Grunt **build** Tasks durchführt, um die jeweiligen Dateien im **webapp/dist/** Ordner neu zu erstellen. Danach wird die Web-App unter **http://localhost:8000** neu geladen.
2. Führt zunächst den oben beschriebenen **build-Task** aus und deployt die fertig gebaute Web-App auf den Server in den Ordner für die Live Umgebung. (siehe Abschnitt [5.9]) .
3. Führt zunächst den oben beschriebenen **build-Task** aus und deployt die fertig gebaute Web-App auf den Server in den Ordner für die Development Umgebung. (siehe Abschnitt [5.9]) .

5.3 Bower

Bower[13] benötigt Node.js, npm sowie Git und wird über

```
npm install -g bower
```

installiert. Das **-g** bedeutet in diesem Fall, dass Bower global installiert wird. Somit ist Bower systemweit und nicht nur in dem Projektordner einsetzbar. Neue Pakete werden dann über

```
bower install <package> --save
```

installiert. Über das –**save** wird der Name des Paketes inklusive seiner Versionsnummer zusätzlich in eine **bower.json** hinzugefügt, die ähnlich aufgebaut ist wie die package.json (siehe Abschnitt [5.1]). Diese wird zu dem Git-Repository hinzugefügt und kann so von anderen Entwicklern genutzt werden. Diese können dann alle notwendigen JavaScript Pakete über

[13]Bower ist ein Paketmanager für die Webentwicklung. https://bower.io, zuletzt abgerufen am 10.07.2016

```
bower install
```

installieren. Die installierten Pakete können über

```
1 bower update <package>
2 bower update
```

aktualisiert werden. Mit dem Befehl in Zeile eins werden alle Pakete aktualisiert und mit dem Befehl in Zeile zwei wird das jeweilige Paket aktualisiert. Die bower.json für dieses Projekt ist auf der CD (siehe Anhang [8.B]) unter **webapp/bower.json** zu finden. Neben Angular.js wurden ebenfalls Pakete wie Angular Material, jQuery, Font Awesome, Underscore.js und exif-js installiert. Als Erweiterungen für Angular.js wurden unter anderem angular-resource, angular-local-storage, angular-md5 und angular-socialshare installiert. Bower übernimmt die Installation der Pakete und fügt automatisch den Pfad zu der JavaScript Datei des jeweiligen Paketes in die index.html ein. Der Name des installierten Moduls muss jetzt nur noch in die Modulliste der Web-App eingetragen werden (siehe Listing [3]).

```
1  var app = angular
2      .module('grafflrWebApp', [
3              'ngResource',
4              'ngRoute',
5              'ngMaterial',
6              'angular-md5',
7              'picardy.fontawesome',
8              'underscore',
9              'LocalStorageModule',
10             '720kb.socialshare',
11             'angular-momentjs'
12     ]).[...]
13 }
```

Listing 3: Die aktuelle Modulliste des Angular.js Frameworks dieser Web-App

5.4 Angular.js

Das von Google entwickelte JavaScript- Webframework Angular.js[14] und dessen Erweiterungen wurden vor allem für das Routing der Clientseite und die REST Anbindung an den Server genutzt.

Angular.js bietet eine gute Möglichkeit, um schnell Web-Apps zu realisieren. Dafür lässt sich das Angular.js Framework sehr einfach über Bower um weitere Module erweitern. Diese Module stellen Funktionen für das Routing, REST Anbindung an den Server, Datenspeicherung im HTML5 Web-Storage etc. zur Verfügung.

Wie unter den funktionalen Anforderungen gefordert (siehe Unterabschnitt [3.4.1]), sollen bestimmte nutzerbezogen Daten in der Applikation gespeichert werden. Dazu wurde das **LocalStorageModule**[15] für Angular.js über Bower installiert. Die Daten werden über die Methoden set und get in den HTML5 Web-Storage geschrieben und aus diesem gelesen (siehe Listing [4] und Bild [5.2]).

```
1 localStorageService.set(key, val);
2 localStorageService.get(key);
```

Listing 4: Angular LocalStorageModule set und get Methode

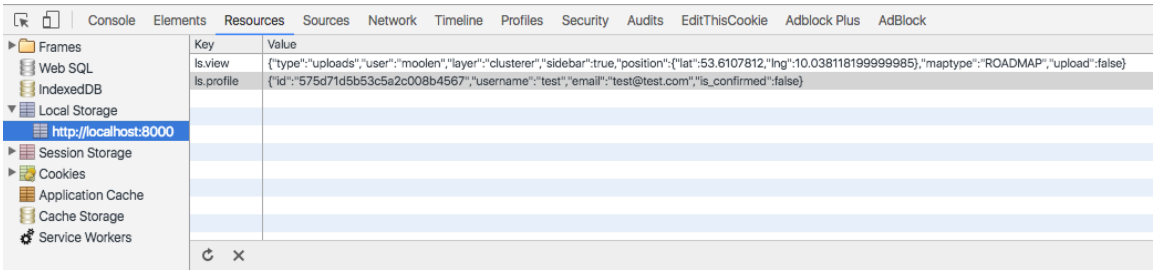

Abbildung 5.2: Die HTML5 Web-Storage Daten dieser Web-App, angesehen in der Google Chrome Entwicklerkonsole (eigene Darstellung)

Für die REST API Anbindung (siehe Kapitel [4.2]) wurde das Angular.js Modul **ngResource**[16] über Bower installiert. Über dessen Funktion **$resource** ist es möglich HTTP Request Anfragen an den Server zu stellen (siehe Listing [5]). Alle für das REST Paradigma notwendigen Methoden wie GET, POST, PUT und DELETE stehen zur Verfügung.

[14] Angular.js ist ein JavaScript - Webframework für Web-Apps https://angularjs.org, zuletzt abgerufen am 10.07.2016

[15] Dokumentation: https://github.com/grevory/angular-local-storage, zuletzt abgerufen am 10.07.2016

[16] Dokumentation: https://docs.angularjs.org/api/ngResource/service/$resource, zuletzt abgerufen am 10.07.2016

```
return $resource(route, params, {
    'get': {
        method : 'GET',
        headers : this.headers,
        isArray : false,
        cache : true,
        withCredentials : true
        transformRequest: function(data, headers){
            log.info('Header Request: ' + route, headers());
            log.info('Data Request: ' + route, data);
            return data;
        },
        transformResponse: function(data, headers){
            log.info('Header Response: ' + route, headers());
            log.info('Data Response: ' + route, data);
            return data;
        }
    },[...]
```

Listing 5: Beispiel des ngResource Moduls anhand der GET Methode

Das über Bower installierte Angular.js Modul **ngRoute**[17] wurde eingesetzt um zu gewährleisten, dass für die Routen **/:username** und **/:username/likes** zum einen bei dem Laden der Web-App und zum anderen bei Ändern der Route über JavaScript, die korrekten Daten geladen werden (siehe Unterabschnitt [3.4.1]). Dies erfolgt beim Laden der Web-App im Controller **MainCtrl** (zu finden auf der CD in der Datei **webapp/app/scripts/controller/main.js** - (siehe Anhang [8.B])) über die **$route** Komponente. Die URL wird über diese Funktion ausgelesen und auf bestimmte Parameter geprüft. Dementsprechend werden die Parameter für die Ansicht gesetzt (siehe Listing [6]).

```
[...]
var view = $route.current.params && $route.current.params.username &&
        $route.current.params.username.length ?
        angular.isDefined($route.current.originalPath.split("/")[2]) ?
        { type: $route.current.originalPath.split("/")[2], user :
                  $route.current.params.username } :
        { type : 'uploads', user : $route.current.params.username } :
        { type : 'around', user : '' };
[...]
```

Listing 6: Einsatz der $route Funktion zum Auslesen der aktuellen Route

[17]Dokumentation: https://docs.angularjs.org/api/ngRoute, zuletzt abgerufen am 10.07.2016

Bei Ändern der Route über JavaScript wird die Komponente **$routeProvider**[18] in der Datei **webapp/app/scripts/app.js** (zu finden auf der CD - (siehe Anhang [8.B])) aufgerufen, woraufhin die Parameter für die korrekte Ansicht gesetzt werden, bevor die neue Ansicht im Hintergrund geladen wird (siehe Listing [7]).

```
$routeProvider.when('/:username', {
    resolve: {
        func: ['$route', 'object', function ($route, object) {
            [...]
            user.view = { type : 'uploads', user : $route.current.params.username };
            [...]
        }]
    }
}), [...]
```

Listing 7: Einsatz der $routeProvider Funktion bei Routenänderung über JavaScript

Das **$scope**[19] Objekt von Angular.js bietet ein gute Möglichkeit sehr einfach Daten via JavaScript „On-the-fly“ in der HTML Datei zu ändern. Dies können einzelne Strings aber auch Arrays sein. Dieses Objekt wurde für die Manipulation aller Daten in den HTML Templates genutzt. So zum Beispiel für die Formulare oder das Nachladen der Fotos in der Seitenleiste.

```
$scope.showCards = true;
$scope.cardlist.data = [
    {'username' : 'John',  'likes' : '15'},
    {'username' : 'Doe',  'likes' : '20'}
]
```

Listing 8: Beispiel von $scope Variablen in Angular.js

```
<div ng-show="showCards" ng-repeat="card in cardlist.data">
    {{card.username}},{{card.likes}},
</div>
```

Listing 9: Die HTML Code Darstellung zu Listing 8

[18] Dokumentation: https://docs.angularjs.org/api/ngRoute/provider/$routeProvider, zuletzt abgerufen am 10.07.2016

[19] Dokumentation: https://docs.angularjs.org/guide/scope, zuletzt abgerufen am 10.07.2016

5.5 Angular Material

Das ebenfalls von Google entwickelte Framework für Benutzeroberflächen, Angular Material[20] bringt neben Funktionen für Komponenten wie zum Beispiel Dialoge auch das Google Material Design[21] mit. Dadurch lassen sich Oberflächen entwickeln, die von Googles Android System und Googles Weboberflächen her bekannt sind.

Für die Eingabefelder der Formulare (siehe Unterabschnitt [4.3.1]) wurde das Material Design genutzt. Die inline Validierung dieser Felder erfolgt über Angular.js (siehe Listing [10]).

```
<md-input-container class="md-icon-float md-block">
    <label>Username</label>
    <md-icon class="fa fa-user-secret"
        ng-class="{'fa-color-success': login.username.$valid }"></md-icon>
    <input ng-minlength="2" type="text" name="username" ng-model="username" required>
    <div ng-messages="login.username.$invalid" role="alert" class="position-absolute">
        <ng-message ng-show="login.username.$error.minlength">
            must be at least 2 characters long
        </ng-message>
        <ng-message ng-show="login.username.$error.required">required</ng-message>
    </div>
</md-input-container>
```

Listing 10: Beispiel eines Formular Eingabefeldes in Angular Material und Angular.js

Für die Buttons[22], sowie dessen Tooltips[23] (siehe Unterabschnitt [4.3.2]) wurden ebenfalls Direktiven von Angular Material genutzt. Die Tooltips können direkt im HTML Quellcode als Direktive eingebunden werden (siehe Listing [11]).

[20] Angular Material ist ein Framework, dass Komponenten für eine Benutzeroberfläche und ein eigenes Layout (Material Design) zur Verfügung stellt. https://material.angularjs.org, zuletzt abgerufen am 10.07.2016

[21] Quelle: https://design.google.com, zuletzt abgerufen am 10.07.2016

[22] Dokumentation: https://material.angularjs.org/latest/api/directive/mdButton, zuletzt abgerufen am 10.07.2016

[23] Dokumentation: https://material.angularjs.org/latest/api/directive/mdTooltip, zuletzt abgerufen am 10.07.2016

```
<md-button class="hide-xs" ng-click="click.openLoginDialog()"
    aria-label="Log in to see your photos an write comments" ng-if="!profile">
    <md-tooltip class="hide-xs hide-sm" md-direction="bottom">
        Log in to see your photos an write comments
    </md-tooltip>
    <md-icon class="fa fa-user"></md-icon>Log In
</md-button>
```

Listing 11: Beispiel eines Buttons mit Tooltip in Angular Material

Die Dialoge (siehe Unterabschnitt [4.3.3]) wurden mit der Angular Material Direktive **$mdDialog**[24] umgesetzt. Dazu zählen auch Confirm Dialoge, die eingesetzt wurden um den Nutzer bestätigen zu lassen, ob er zum Beispiel ein Foto tatsächlich löschen möchte (siehe Listing 12). Dies wurde in den Anforderungen (siehe Unterabschnitt 3.4.1) definiert. Die Templates der Dialoge befinden sich auf der CD (siehe Anhang [8.B]) im Ordner **webapp/app/view/dialog/**.

```
var confirm = $mdDialog.confirm()
    .title('title')
    .theme('theme')
    .htmlContent('htmlContent')
    .ariaLabel('ariaLabel')
    .targetEvent(ev)
    .ok('okTxt')
    .cancel('cancelTxt');

$mdDialog.show(confirm).then(function() {
    okCb ? okCb() : '';
}, function() { cancelCb ? cancelCb() : cancelDialog(); });
```

Listing 12: Beispiel eines Confirm Dialogs in Angular Material

Um dem Nutzer Notifikationen zur Verfügung zu stellen (siehe Unterabschnitt [4.3.4]), wurde der Angular Material Service **$mdToast**[25] genutzt. Mittels dieses Services lassen sich Benachrichtigungen an einer beliebigen Seite der Web-App darstellen (siehe Listing 13). Das Template der Notifikationen befindet sich auf der CD (siehe Anhang [8.B]) in der Datei **/webapp/app/view/toast/message.html**.

[24] Dokumentation: https://material.angularjs.org/latest/api/directive/mdDialog, zuletzt abgerufen am 10.07.2016

[25] Dokumentation: https://material.angularjs.org/latest/api/service/$mdToast, zuletzt abgerufen am 10.07.2016

```
$mdToast.show(
    $mdToast.simple()
        .content('content')
        .position('position')
        .hideDelay(hideDelay)
        .theme('theme')
);
```

Listing 13: Beispiel des $mdToast Service in Angular Material

Für den Einsatz von Cards (siehe Unterabschnitt [4.3.5]) wurde die Angular Material Direktive **md-card** genutzt (siehe Listing 14). Das Template der Card befindet sich auf der CD (siehe Anhang [8.B]) in der Datei **/webapp/app/view/-card/img.html**.

```
<md-card>
    <md-card-header>
        <md-card-avatar></md-card-avatar>
        <md-card-header-text>HEADER</md-card-header-text>
    </md-card-header>
    <md-card-title><md-card-title-text>TITLE</md-card-title-text></md-card-title>
    <md-card-actions layout="row" layout-align="start center">
        <md-button></md-button>
        <md-card-icon-actions>
            <md-button class="md-icon-button" aria-label="icon">
                <md-icon md-svg-icon="icon"></md-icon>
            </md-button>
        </md-card-icon-actions>
    </md-card-actions>
    <md-card-content>TEXT</md-card-content>
</md-card>
```

Listing 14: Beispiel des HTML Gerüstes der Card Direktive in Angular Material

Um zu gewährleisten, dass die Web-App unabhängig von der Bildschirmgröße gut bedienbar ist (siehe Unterabschnitt [3.4.2]) wurde die **Flex** Direktive[26] von Angular Material genutzt, die auf flex[27] von CSS3 basiert. Über diese Direktive ist es möglich mittels Attributen verschiedene flex Breiten für unterschiedliche Auflösungen zu definieren. Ebenfalls ist es über inline Styles möglich bestimmte Container für bestimmte Auflösungen auszublenden (siehe Listing 15).

[26] Dokumentation: https://material.angularjs.org/latest/layout/children, zuletzt abgerufen am 08.07.2016

[27] Quelle: http://www.w3schools.com/cssref/css3_pr_flex.asp, zuletzt abgerufen am 08.07.2016

```
<div layout-wrap layout="row" layout-align="start start">
    <div class="hide-xs" layout="column" flex="50" flex-gt-sm="30"
        layout-align="start start"</div>
</div>
```

Listing 15: Beispiel der Flex Direktive in Angular Material

5.6 Google Maps

Google Maps bietet eine einfache Bedienung seiner Karte über eine JavaScript API[28] an. Darüber können verschiedene Methoden und Events angesteuert werden, wie zum Beispiel der Zoom, die Anzeige der HeatMapLayer oder das Ändern des Kartentypen. Neben der Google Maps API wurden ebenfalls die Google Maps Bibliotheken für den Marker Clusterer[29] und den Geolocation-Marker[30] installiert. Somit werden alle Anforderungen (siehe Unterabschnitt [3.4.2]) an die Karte erfüllt.

```
var map = new google.maps.Map(document.getElementById('mapID'), config);
map.setZoom(value);
map.setCenter(position);
map.setMapTypeId(google.maps.MapTypeId.ROADMAP);
var markerClusterer = new MarkerClusterer(map, markers, options);
var heatMap = new google.maps.visualization.HeatmapLayer({ data: positions });
var geolocationMarker = new GeolocationMarker(map, { icon : 'icon' });
var clientLocation = google.loader.ClientLocation;
```

Listing 16: Beispiele verschiedener JavaScript-Funktionen der Google Maps API und der zusätzlich installierten Bibliotheken

[28] Die Google Maps JavaScript v3 API Dokumentation: https://developers.google.com/maps/documentation/javascript/3.exp/reference?hl=de, zuletzt abgerufen am 08.07.2016

[29] Der Marker Clusterer ist eine JavaScript-Bibliothek für die Google Maps-Karte https://github.com/googlemaps/js-marker-clusterer, zuletzt abgerufen am 08.07.2016

[30] Der Geolocation-Marker ist eine JavaScript-Bibliothek für die Google Maps-Karte https://github.com/ChadKillingsworth/geolocation-marker, zuletzt abgerufen am 08.07.2016

Die Funktionen (siehe Listing 16) erfüllen je nach Zeile folgende Funktion:

1. Erstellen eines neuen Google Maps-Objektes
2. Setzen des Zooms der Google Maps-Karte auf eine bestimmte Zoomstufe
3. Zentrieren der Google Maps-Karte auf eine bestimmte Koordinate
4. Setzen des Kartentyps (ROADMAP, SATELLITE, HYBRID)
5. Erstellen eines neuen Marker-Clusterer-Objektes
6. Erstellen einer neuen Heatmap-Layer
7. Erstellen eines neuen Geolocation-Marker-Objektes (Auslesen der aktuellen Position des Nutzers über HTML5 Geolocation)
8. Positionsbestimmung über IP Geolocation

5.7 SASS

SASS[31] ist eine Stylesheet-Sprache, die es zum Beispiel ermöglicht CSS Regeln verschachtelt zu schreiben und mit Variablen zu arbeiten. Um SASS Dateien in CSS zu kompilieren wird das Grunt-Modul Compass genutzt (siehe Kapitel [5.2]). Die SASS Dateien befinden sich auf der CD (siehe Anhang [8.B]) im Ordner **webapp/app/styles/**.

```
1 $rgba-primary-5: rgba(54, 66, 77, 1);
2 #ng-app {
3     .header {
4         width: 100%;
5         background : $rgba-primary-5;
6     }
7 }
```

Listing 17: Beispiel von SASS Merkmalen

```
1 #ng-app .header {
2     width: 100%;
3     background : rgba(54, 66, 77, 1);
4 }
```

Listing 18: Der aus (siehe Listing [17]) kompilierte CSS Code

[31] Dokumentation: http://sass-lang.com/documentation/file.SASS_REFERENCE.html, zuletzt abgerufen am 10.07.2016

5.8 Versionskontrolle

Die Versionskontrolle wurde über Git[32] durchgeführt. Dafür wurde serverseitig Stash[33] für die Verwaltung der Git-Repositorys eingesetzt. Stash wurde von Atlassian entwickelt und stellt eine in Java geschriebene Weboberfläche für die Verwaltung der Repositorys zur Verfügung. Der Zugriff auf das Repository für dieses Projekt erfolgt via SSH Schlüssel, der zuvor in Stash hinterlegt werden muss. Neben dem Git Repository dieser Web-App wurde im Ordner **webapp/app/scripts/dotjs/** (siehe Anhang [8.B]) ein weiteres Git Repository mit helper-Klassen genutzt und als Submodul dem Repository dieser Web-App hinzugefügt. Für dieses Repository muss ebenfalls ein SSH Schlüssel in Stash hinterlegt werden.

5.9 Deployment

Das Deployment der App wird über Grunt (siehe Bereich [5.2]) durchgeführt. Dazu wird zunächst der Grunt **build-Task** durchgeführt, um die Web-App zu bauen. Danach wird der Inhalt des **webapp/dist/** Ordners über den Grunt-Task **sftp-deploy** via SFTP auf den Server kopiert (siehe Listing [19]). Es gibt zwei sftp-deploy Tasks. Einen für die development und einen für die live-Umgebung.

[32] Git ist ein open source System für die Versionskontrolle. https://git-scm.com, zuletzt abgerufen am 09.07.2016

[33] Stash wurde von Atlassian in Bitbucket Server umbenannt und bietet eine Weboberfläche für die Verwaltung von Git-Repositorys an. https://de.atlassian.com/software/bitbucket/server, zuletzt abgerufen am 09.07.2016

```
'sftp-deploy': {
    [...]
    app_dev: {
        auth: {
            host: '87.106.26.76',
            port: 22,
            authKey: 'privateKey'
        },
        cache: false,
        src: 'dist',
        dest: 'dev.grafflr.com/current/app',
        exclusions: ['dist/**/.DS_Store', 'dist/**/Thumbs.db', 'dist/tmp'],
        serverSep: '/',
        concurrency: 4,
        progress: true
    },[...]
}
```

Listing 19: Der Grunt Task sftp-deploy: app_dev

Eine Beispiel Datei für die Einstellungen des privaten SSH Schlüssels zu diesem Task ist auf der CD (siehe Anhang [8.B]) unter **webapp/.ftppass_example** zu finden. Diese Datei muss in **webapp/.ftppass** umbenannt und die korrekten Daten zu dem privaten SSH Schlüssel müssen eingetragen werden (siehe Listing [20]).

```
{
  "privateKey": {
      "username"      : "",
      "passphrase"    : "",
      "keyLocation"   : ""
    }
}
```

Listing 20: Die .ftppass_example Datei für die Einstellungen des privaten SSH Schlüssels

Der öffentliche SSH Schlüssel für diesen Prozess muss vorher auf dem Server hinterlegt werden.

5.10 Design

Hier ist das aktuelle Design der Web-App zum Stand der Abgabe dieser Bachelorarbeit zu sehen.

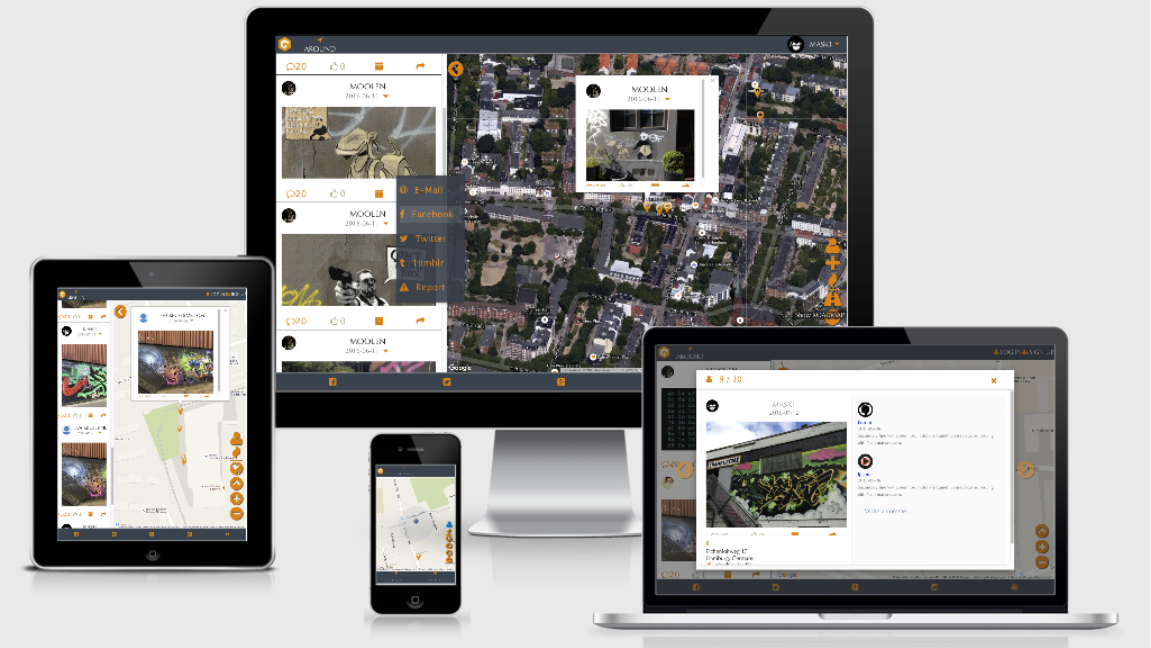

Abbildung 5.3: Das Design dieser Web-App auf unterschiedlichen Geräten (eigene Darstellung)

5.11 Einschränkungen und Fehler

Während der Entwicklung, bei Testläufen und während des Praxis-Tests (siehe Abschnitt [6.1]) traten Fehler auf, die im Rahmen dieser Bachelorarbeit nicht mehr gelöst werden konnten.

- Da nicht alle Endgeräte zur Verfügung standen, konnte die Web-App noch nicht auf allen Endgeräten und in allen Browsern getestet werden.
- Da serverseitig nicht das Routing für die Kommentarfunktionen und die Suche implementiert waren, konnten diese Funktionalitäten noch nicht umgesetzt werden.
- Trotz HTML 5 Geolocation konnte die Nutzerposition in einigen wenigen Fällen nicht immer exakt bestimmt werden.
- In der Bedienung der Google Maps-Karte kam es stellenweise im Internet Explorer von Microsoft zu Verzögerungen.
- Es wurden noch keine Unit Tests implementiert.
- In einem Fall wurde die Google Maps-Karte nicht auf die Position eines Probanden zentriert, nachdem er den zugehörigen Button gedrückt hatte.
- Wenn ein Nutzer ein Foto löscht und direkt danach eine weitere POST, PUT oder DELETE Anfrage an den Server schickt, erhält der Browser keine Antwort von dem Server und bleibt in dem Prozess stehen.
- Die Tooltips blieben in wenigen Fällen geöffnet, auch wenn der Proband den jeweiligen Button mit der Maus verlassen hat. Dabei handelt es sich um einen Bug von Angular Material[34]
- Auf dem mobilen Endgerät iPhone 6 wird der Ladebalken nicht wieder ausgeblendet, nachdem das Foto erfolgreich auf den Server geladen wurde.

[34]Quelle: https://github.com/angular/material/issues/4249, zuletzt abgerufen am 14.07.2016

6 Evaluation

6.1 Praxis-Test

Dieser Praxis-Test ist dahingehend notwendig, da unvoreingenommene Probanden diese Web-App anders bedienen, als dies von dem Entwickler zu erwarten ist. Alan Cooper fasst dies in einem Absatz zusammen.

„If you want a great site, you've got to test. After you've worked on a site for even a few weeks, you can't see it freshly anymore. You know too much. The only way to find out if it really works is to test it. " (Cooper 2014), Seite 133

Dadurch soll ermittelt werden, inwiefern die Oberfläche dieser Web-App bereits benutzerfreundlich ist und was noch verbessert werden kann. Ebenfalls soll überprüft werden, ob der Proband bestimmte Funktionen wünscht, die derzeit in dieser Version der Web-App noch nicht implementiert wurden.

Bereits während der Entwicklungsphase wurde die Web-App immer wieder in verschiedenen Browsern auf Fehler hin getestet. Voraussichtlich werden die Probanden dennoch auf Betriebs- und Bedienungsfehler stoßen, die in der Entwicklung übersehen wurden.

Für diesen Praxis-Test wurde die aktuelle Version dieser Web-App in der Development Umgebung (siehe Abschnitt [4.1]) auf dem Server zur Verfügung gestellt. Ebenfalls wurde eine Umfrage mittels SurveyMonkey[1] erstellt, die der Proband über eine URL erreichen kann. Für die Umfrage soll der Proband verschiedene Aufgaben durchführen, die direkten Bezug zu den User Stories besitzen (siehe Abschnitt [3.3]). Der komplette Fragebogen inklusive der Aufgabenstellungen für den Probanden ist auf der CD (siehe Anhang [8.B]) in der Datei **umfrage/SurveyMonkey.pdf** zu finden.

Mittels der Umfrage soll folgendes untersucht werden:

1. Welches Betriebssystem bzw. welche Browser am häufigsten genutzt werden.
2. Wie der Proband die Geschwindigkeit des Ladens der Bilder in der Seitenleiste und der Karte, je nach Browser empfindet. Ebenfalls soll nach Browser untersucht werden, ob und wo die meisten Fehler aufgetreten sind.

[1] SurveyMonkey ist eine Webseite über die es möglich ist Online Umfragen zu erstellen. https://de.surveymonkey.com, zuletzt abgerufen am 01.07.2016

3. Ob der Proband die Buttons für bestimmte Aktionen auf Anhieb findet.
4. Ob eher Probanden, die sich mit Graffiti und Streetart auskennen, oder Probanden, die sich nicht damit auskennen, nicht vorstellen können die Applikation in Zukunft zu nutzen.
5. Ob Probanden es eher präferieren, wenn nach dem Schließen bestimmter Dialoge auf das zuletzt angezeigte Bild bzw. auf die Position des Probanden zentriert wird oder der Dialog geschlossen wird, ohne dass eine Aktion ausgeführt wird.
6. Ob der Proband es als sinnvoll erachtet, wenn die Buttonleiste auf der Card einmal in der Seitenleiste sowie im Info-Window über einem Marker angezeigt wird oder ob es bei geöffneter Seitenleiste ausreicht, wenn die Buttonleiste nur einmal angezeigt wird.
7. Ob dem Probanden klar ist, dass er die Slides im Carousel ebenfalls per Tastatur vor- und zurück navigieren kann.
8. Wie aufwendig der Proband den Ablauf der Registrierung findet.
9. Ob der Proband zum Zoomen der Google Maps-Karte ebenfalls auf das Scroll-Rad der Maus zurückgreifen möchte.
10. Ob der Proband zu einem Foto geroutet werden möchte.
11. Ob der Proband einem anderen Nutzer folgen und diesen zu seinen Favoriten hinzufügen möchte.
12. Ob der Proband die Applikation eher als Webseite oder native App empfindet.
13. Ob der Proband einen Facebook bzw. Google-Login wünscht.
14. Ob dem Probanden die Tooltips über den Buttons ausgereicht haben oder ob diese zusätzlich mit Labels versehen werden sollen.
15. Ob dem Probanden zu jeder Zeit klar war, in welcher Ansicht er sich befunden hat (Fotos in der Umgebung, auf den Server geladene Fotos eines Nutzers, gelikte Fotos eines Nutzer, seine auf den Server geladenen Fotos, seine gelikten Fotos).

In zukünftigen Entwicklungszyklen soll das Feedback zu dieser Umfrage in die Entwicklung mit einfließen und für zukünftige Versionen berücksichtigt werden. Dafür soll diese Umfrage erweitert und vor jeder Einführung einer neuen Version dieser Web-App durchgeführt werden.

6.2 Auswertung der Nutzerumfrage

Diese Umfrage wurde **18-mal** durchgeführt und mindestens **10-mal** alle Fragen beantwortet. Vermutlich haben einige Probanden die Umfrage nicht bis zum Ende durchgeführt oder die Session der Umfrage ist von Seite des SurveyMonkey Dienstes abgelaufen, da der Nutzer zwischenzeitlich die Web-App bedienen musste. Die kompletten Ergebnisse dieser Umfrage in Form von Diagrammen sind auf der CD (siehe Anhang [8.B]) in dem Ordner **umfrage/ergebnisse/** zu finden. In Bezug auf die Untersuchungen aus dem vorherigen Abschnitt (siehe Abschnitt [6.1]) ergeben sich in dieser Reihenfolge folgende Ergebnisse:

1. Diese Frage wurde von **18 Probanden** beantwortet. Micorsoft Windows wurde zu **50 %** genutzt und MacOS ebenfalls zu **50 %**. Der Safari wurde zu **16,67 %** genutzt, der Google Chrome zu **61,11 %** und der Mozilla Firefox zu **22,22 %**. Der Internet Explorer und andere Browser wurden auf Anraten hin zu **0 %** genutzt.
2. Mit Google Chrome wurde die Geschwindigkeit zum Laden von Fotos in der Seitenleiste zu **50 %** mit **gut** und zu **50 %** mit **ok** bewertet. Die Geschwindigkeit der Google Maps-Karte zu **87,5 %** mit **gut** und zu **12,5 %** mit **könnte schneller sein** bewertet. Es wurden zu **50 %** Fehler gefunden und zu **50 %** keine Fehler gefunden. Es haben **8** Probanden mit Google Chrome Browser diese Fragen beantwortet. Diese Fragen wurden hinsichtlich der anderen Browser nur jeweils von einem Probanden durchgeführt und sind daher nicht repräsentativ.
3. Die Fragen hinsichtlich der Buttons wurden von **10-12 Probanden** beantwortet. Es hat sich gezeigt, dass fast alle Buttons sofort gefunden wurden. Abweichungen gibt es bei dem Button zum Öffnen und Schließen der Seitenleiste. **50 %** haben diesen Button **sofort gefunden** und **50 %** haben diesen Button **nicht auf Anhieb gefunden**. Den Button um sich alle Fotos eines Nutzers anzuzeigen, die er auf den Server geladen bzw. gelikt hat, haben **58,33 % sofort gefunden**, **25 % nicht auf Anhieb gefunden** und **16,67 % gar nicht gefunden**. Den Button um wieder zurück in die Ansicht zu Bildern in der Umgebung zu kommen, haben **58,33 % sofort gefunden**, **33,33 % nicht auf Anhieb gefunden** und **8,33 % gar nicht gefunden**. Den Button zum Öffnen des Carousels haben **25 % sofort gefunden**, **50 % nicht auf Anhieb gefunden** und **25 % gar nicht gefunden**[2].
4. In der Umfrage wurde gefragt ob der Proband weiß was ein Stencil ist oder ob er dies Googlen müsste. Dadurch konnte ermittelt werden, ob sich der Proband mit Graffiti bzw. Streetart auskennt. Ein Stencil ist in der Regel ein Begriff für jemanden, der sich in diesem Bereich auskennt. Von **4 Probanden** die sich mit Graffiti auskennen, können sich **100 %** vorstellen diese Web-App zu nutzen um

[2] An dieser Steller müsste in Zukunft überprüft werden ob dieser Button tatsächlich nicht gefunden wurde oder ob nicht klar war was ein Carousel ist.

ihre Graffiti und Streetart Bilder zu verwalten. Von **6 Probanden** die sich nicht damit auskennen, können sich **83,33 %** vorstellen diese Web-App zu nutzen.

5. Diese Frage wurde von **10-12 Probanden** beantwortet. Für das Schließen der Dialoge Carousel, Upload und den Dialog zum Bearbeiten eines Fotos hat sich ergeben, dass sich in allen Fällen eine Mehrheit zwischen **40-60 %** weiterhin wünschen auf der Google Maps-Karte auf das zuletzt angezeigte Foto zentriert zu werden. **20-30 %** der Probanden wünschen sich, dass entweder auf die Nutzerposition zentriert oder der Dialog einfach nur geschlossen wird.
6. Diese Frage wurde von **12 Probanden** beantwortet. Davon finden es **58,33 %** sinnvoll, dass die Buttonleiste in der Sidebar und im Info-Window über dem Marker jeweils auf der Card angezeigt wird. **33,33 %** finden dies nur sinnvoll, wenn die Seitenleiste ausgeblendet ist und **8,33 %** würden sich wünschen, wenn die Buttonleiste nicht mehr auf der Card in Seitenleiste angezeigt wird.
7. Diese Frage wurde von **12 Probanden** beantwortet. Davon war es **41,67 %** der Probanden klar, dass sie sich auch über die Tastatur durch die Slides des Carousels bewegen können. **58,33 %** war dies nicht klar.
8. Von **10 Probanden** fanden **90 %** den Ablauf der Registrierung **einfach** und **10 % ok**.
9. Es wünschen sich **60 %** von **10 Probanden**, neben dem Zoom der Google Maps-Karte über Buttons ebenfalls über das Scroll-Rad der Maus zoomen zu können. Für **40 %** der Probanden waren die Buttons ausreichend.
10. **90 %** von **10 Probanden** wünschen es sich zu der Position eines Fotos geroutet werden zu können. **10 %** der Probanden wünschen dies nicht.
11. Es haben **10 Probanden** diese Frage beantwortet, von denen sich **80 %** wünschen einem Nutzer folgen und diesen zu seinen Favoriten hinzufügen zu können.
12. Alle der **10 Probanden**, die diese Frage beantwortet haben, empfanden die Web-App als native Applikation, wenn diese in einem nativen Fenster des Betriebssystems gelaufen wäre und niemand empfand sie als Webseite.
13. Von **10 Probanden**, die diese Frage beantwortet haben, wünschen sich **10 %** einen Google bzw. Facebook-Login. **70 %** der Probanden wünschen keine Zuordnung ihres Google bzw. Facebook-Accounts zu dieser Web-App und **20 %** besitzen keinen Facebook-Account.
14. Es hat sich gezeigt, dass sich von **10 Probanden**, die diese Frage beantwortet haben, **20 %** zu dem Tooltip ein Label unter oder neben dem Button wünschen, während **80 %** der Probanden die Tooltips gereicht haben.
15. Es war **70 %** von **10 Probanden** zu jeder Zeit klar, in welcher Ansicht sie sich befanden (Fotos in der Umgebung, auf den Server geladene Fotos eines Nutzers, gelikte Fotos eines Nutzers, seine auf den Server geladenen Fotos, seine gelikten Fotos).

7 Fazit

Ziel dieser Arbeit war es eine Applikation zu entwickeln, über die es möglich ist Fotos inklusive ihrer GPS Position auf einen Server zu laden und diese auf einer Karte und als Liste in einer Seitenleiste darzustellen. Dem Nutzer sollten ebenfalls verschiedene Technologien aus dem Social Media Bereich zur Verfügung gestellt werden.

Zunächst sollte entschieden werden, welche Technologien es derzeit ermöglichen eine solche Applikation schnell zu implementieren, vielen Nutzern zugänglich zu machen und eine Benutzeroberfläche zu bieten, die sich dem Nutzer auf Anhieb erschließt. Dazu wurden die verschiedenen App-Technologien untersucht, die sich in native Apps, Web-Apps und Hybrid-Apps aufteilen. Darauf folgend wurden die Anforderungen an die zu entwickelnde Applikation definiert. Dazu wurden Personas, ein UML Use Case Diagramm und User Stories definiert, um daraus die funktionalen und nicht-funktionalen Anforderungen zu bestimmen. Auf Basis dieser Anforderungen konnte nun ein Konzept und Design entwickelt werden. An dieser Stelle wurde entschieden, die Applikation mit Web-Technologien als Web-App zu realisieren, da sie über einen Browser betriebssystemunabhängig vielen Nutzern zugänglich gemacht werden kann. So bot die Webtechnologie CSS die Möglichkeit ein Design zu entwickeln, dass geräteübergreifend bedienbar ist. Ebenfalls konnte mittels Web-Technologien ein Zugriff auf die notwendige Hardware wie GPS und den lokalen Speicher implementiert werden. Während der Implementierung wurden verschiedene Paketmanager, Frameworks, Module und Bibliotheken etc. wie zum Beispiel Node.js, Grunt, Bower, Angular.js, Angular Material, SASS und Git eingesetzt, um eine erweiterungsfähige Applikation zu implementieren und das Deployment auf den Server zu gewährleisten.

In dem darauffolgenden Praxis-Test sollten die Probanden verschiedene Aufgaben, die direkten Bezug zu den User Stories besaßen, in der Web-App durchführen und eine online Umfrage zu diesen Aufgaben beantworten. In der Auswertung der Nutzerumfrage hat sich gezeigt, dass die Web-App bereits benutzerfreundlich ist und nur an wenigen Stellen nachgebessert werden muss. So wurden von den Probanden fast alle Buttons auf Anhieb und nur wenige Buttons nicht auf Anhieb (16,67 %-50 %) oder gar nicht (8,33 %-25 %) gefunden. Ebenfalls hat sich gezeigt, dass Icon Buttons mit ihren Symbolen in Kombination mit Tooltips ausreichen, um dem Nutzer ein Verständnis von deren Funktion zu vermitteln. So haben 80 % der Probanden angegeben, dass die Tooltips für das Verständnis der Funktion des Buttons ausgereicht haben und kein Label zu dem Button erforderlich war. Die Geschwindigkeit der Google Maps-Karte wurde von fast allen Probanden als gut und die des Nachladens der Fotos in der Seitenleiste als gut oder ok bewertet. Des Weiteren wussten 70

% der Probanden jederzeit in welcher Ansicht sie sich befunden haben. Es hat sich ebenfalls ergeben, dass sich nur 10 % der Probanden einen Google bzw. Facebook Login wünschen. Alle Probanden haben angegeben, dass sie diese Web-App eher als Applikation empfinden und nicht als Webseite, wenn sie in einem nativen Fenster des jeweiligen Betriebssystems laufen würde.

Es hat sich jedoch auch gezeigt, dass nur 58,33 % der Probanden klar war, dass sie das Carousel auch über die Tastatur hätten bedienen können. Ebenfalls wünschen sich die Probanden folgende Funktionen, die noch implementiert werden sollen:

- 60 % der Probanden wünschen sich, die Google Maps-Karte neben den Buttons auch über ein Scroll-Rad bedienen zu können.
- 90 % der Probanden wünschen sich ein Routing zu den Fotos.
- 80 % der Probanden wünschen es sich, einem Nutzer folgen und diesen zu seinen Favoriten hinzufügen zu können.

Es hat sich gezeigt, dass mit Web-Technologien die Umsetzung einer benutzerfreundlichen Oberfläche für eine Applikation zur Geolokalisierung von Fotos unter Einbindung von Social Media Technologien möglich ist und dabei schnell implementiert und vielen Nutzer zugänglich gemacht werden kann. Es können sich alle Probanden, die sich mit Streetart und Graffiti auskennen, und 83,33 % der Probanden, die sich nicht damit auskennen, vorstellen diese Web-App zu nutzen, um ihre Fotos aus diesem Bereich darüber zu verwalten. Aus diesem Grund soll angestrebt werden, diese Web-App zukünftig zu erweitern und zu verbessern.

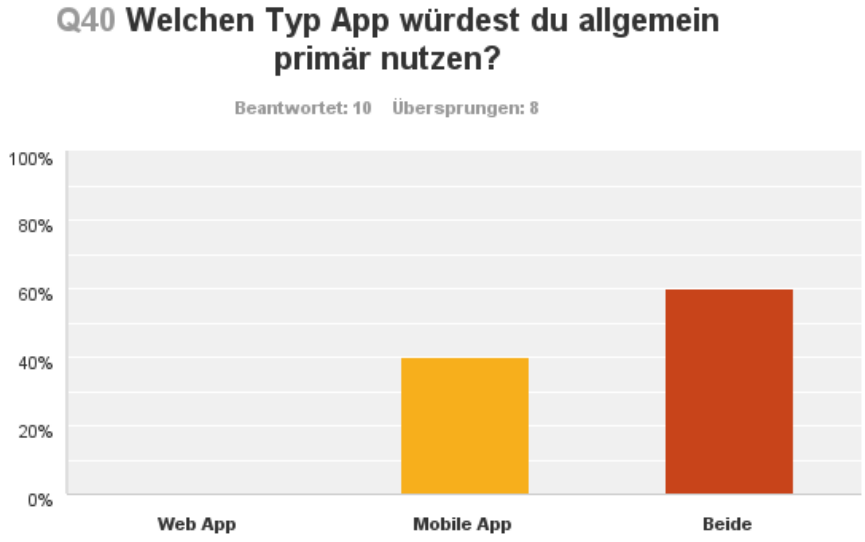

Abbildung 7.1: Das Diagramm aus der Auswertung der Nutzerumfrage zu der Frage welchen Typ App der Proband primär nutzt (eigene Darstellung)

Neben der Beseitigung von Fehlern und Weiterentwicklung dieser Web-App soll zukünftig in Betracht gezogen werden, aus dieser Web-App eine Hybrid-App mittels den Frameworks PhoneGap für mobile Endgeräte bzw. Electron für Desktop Rechner zu erstellen, da alle Probanden aus der Nutzerumfrage diese Web-App bereits als native App empfunden haben, wenn diese nicht in einem Browser sondern einem nativen Fenster des jeweiligen Betriebssystem gelaufen wäre und sich gezeigt hat, dass 40 % der Probanden primär eine mobile App bevorzugen würden, während 60 % der Probanden eine mobile sowie die Web-App nutzen würden (siehe Abb. 7.1). Ebenfalls soll geschaut werden, dass die Web-App noch unabhängiger von Frameworks wie Angular.js oder Angular Material wird bzw. irgendwann gänzlich auf Frameworks dieser Art verzichtet werden kann. Solche Frameworks eignen sich gut dazu schnell Web-Apps zu implementieren, jedoch liefern zusätzlichen JavaScript Code von dem nur ein Teil für die Implementierung genutzt wurde. Dadurch muss eine höhere Datenmenge über das Internet an den Client ausgeliefert werden[1]. Ebenfalls muss der Browser dadurch zusätzlichen JavaScript Code interpretieren, was zu einem höheren Leistungsanspruch an das jeweilige Endgerät führt. De Weiteren muss berücksichtigt werden, dass sich Web-Technologien gerade im JavaScript Bereich unheimlich schnell weiterentwickeln und dadurch bestehende Frameworks schnell durch Neuere abgelöst werden.

[1]Diese Web-App hat derzeit eine Gesamtgröße von ca 2.3MB

8 Anhang

8.A Grafiken

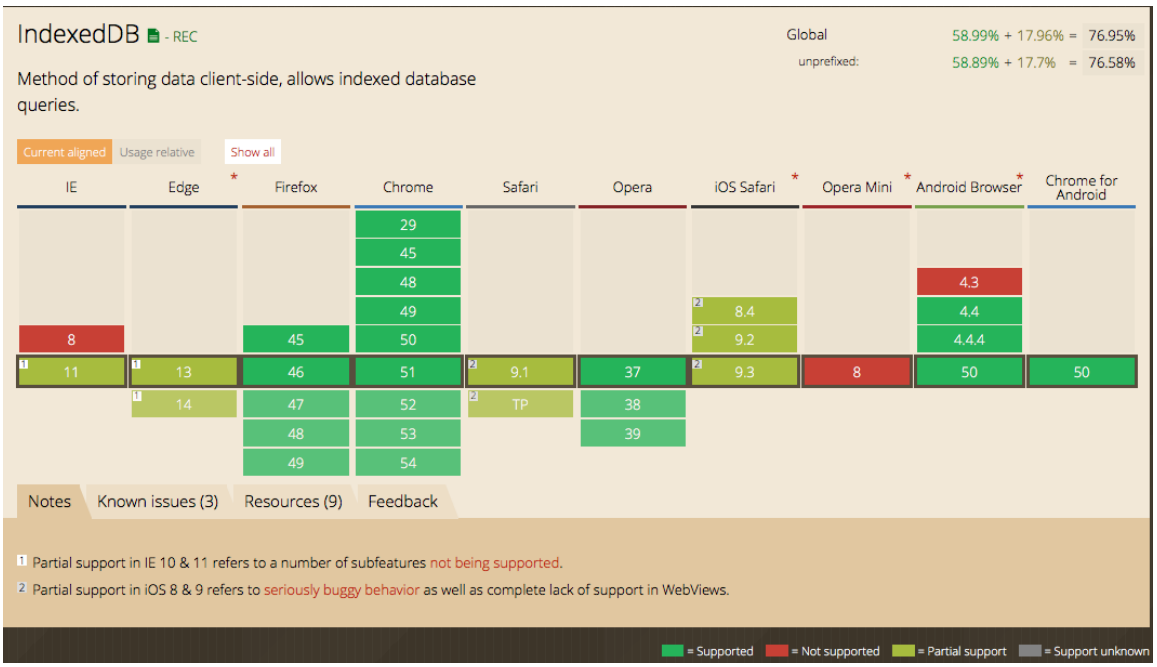

Abbildung 8.1: Browser Support für IndexedDB
(Can I Use - IndexedDB 2016)

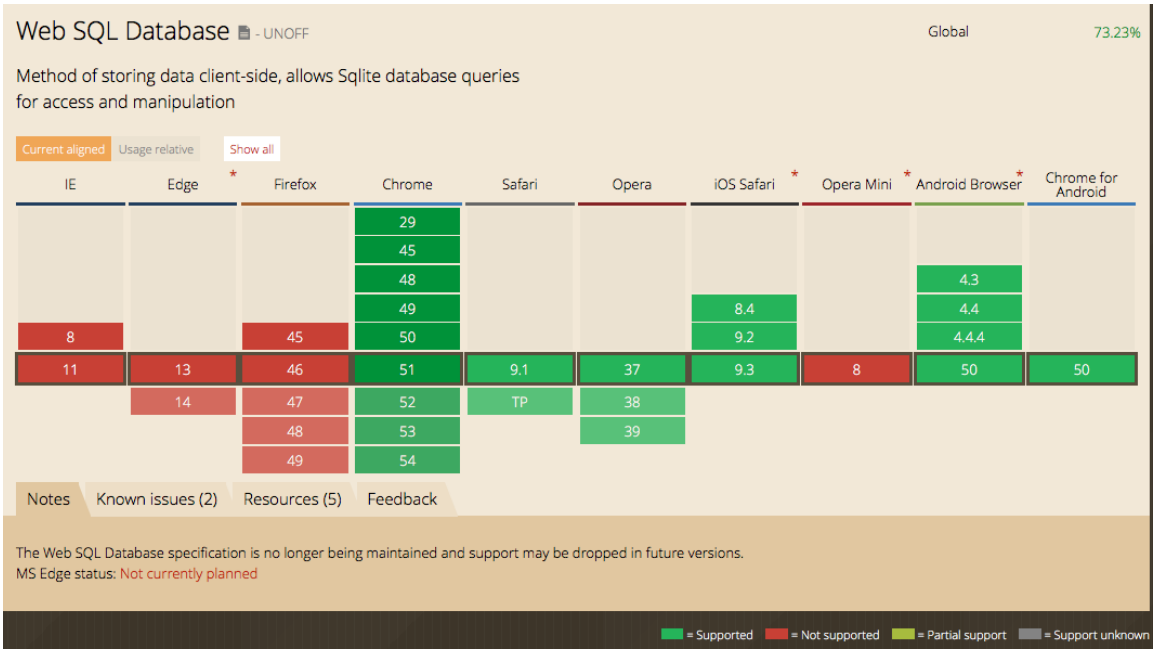

Abbildung 8.2: Browser Support für Web SQL Database
(Can I Use - Web SQL Database 2016)

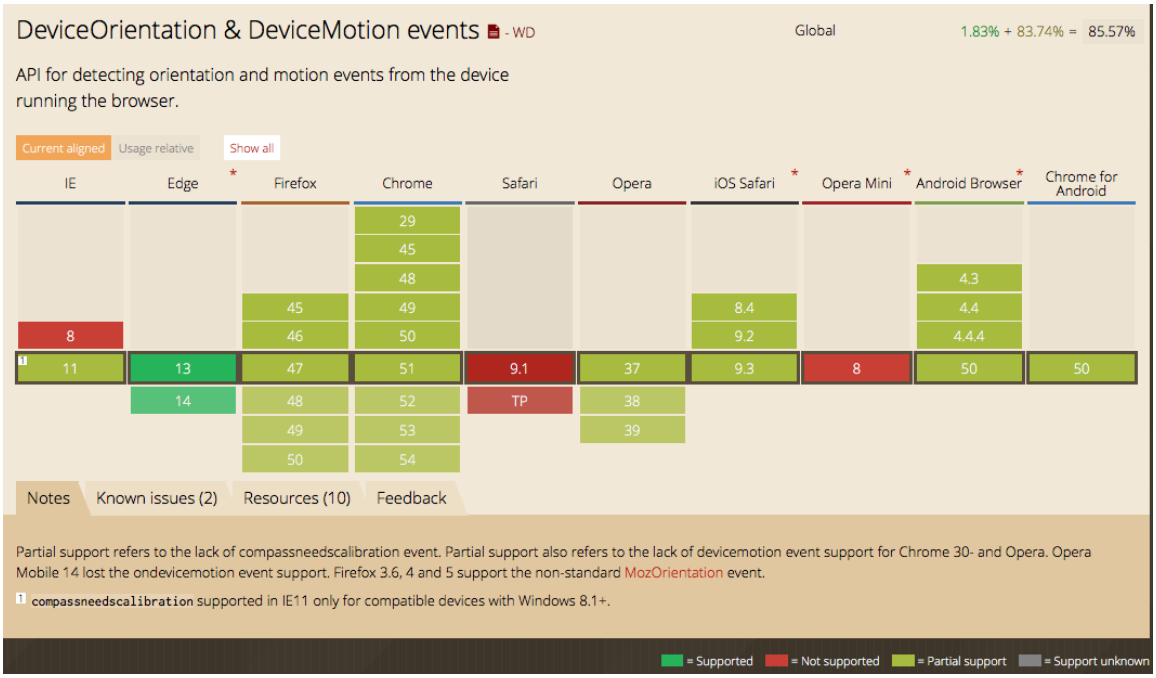

Abbildung 8.3: Browser Support für HTML5 DeviceOrientation
(Can I Use - DeviceOrientation & DeviceMotion events 2016)

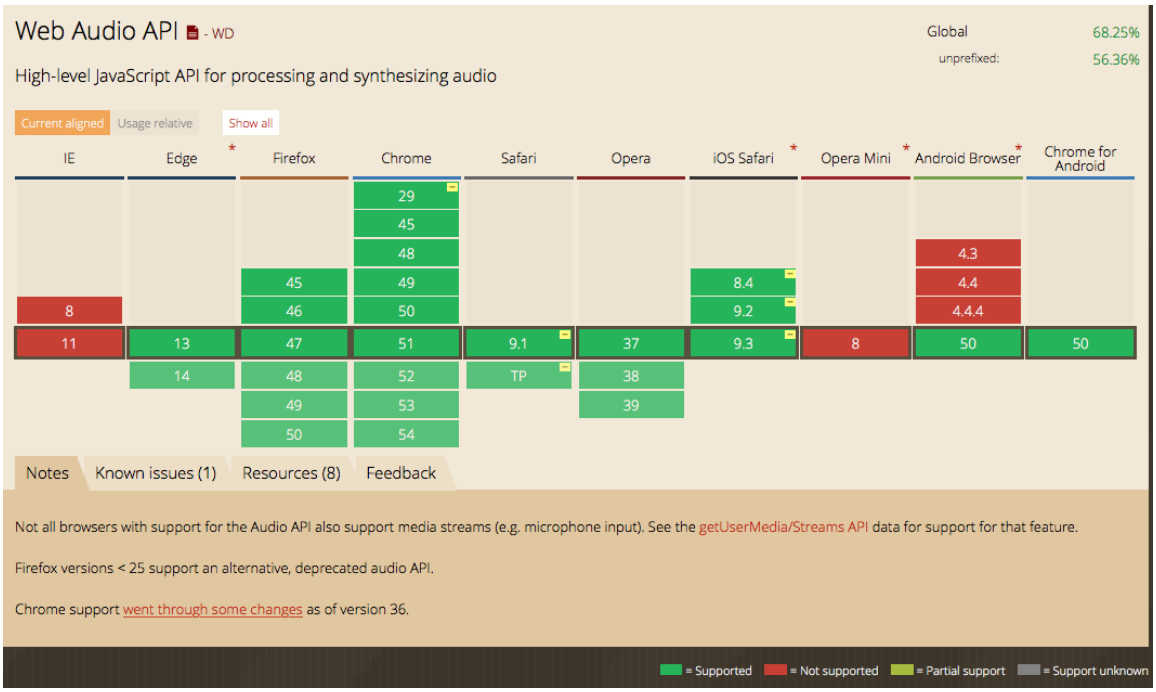

Abbildung 8.4: Browser Support für HTML5 Web Audio API (Can I Use - Web Audio API 2016)

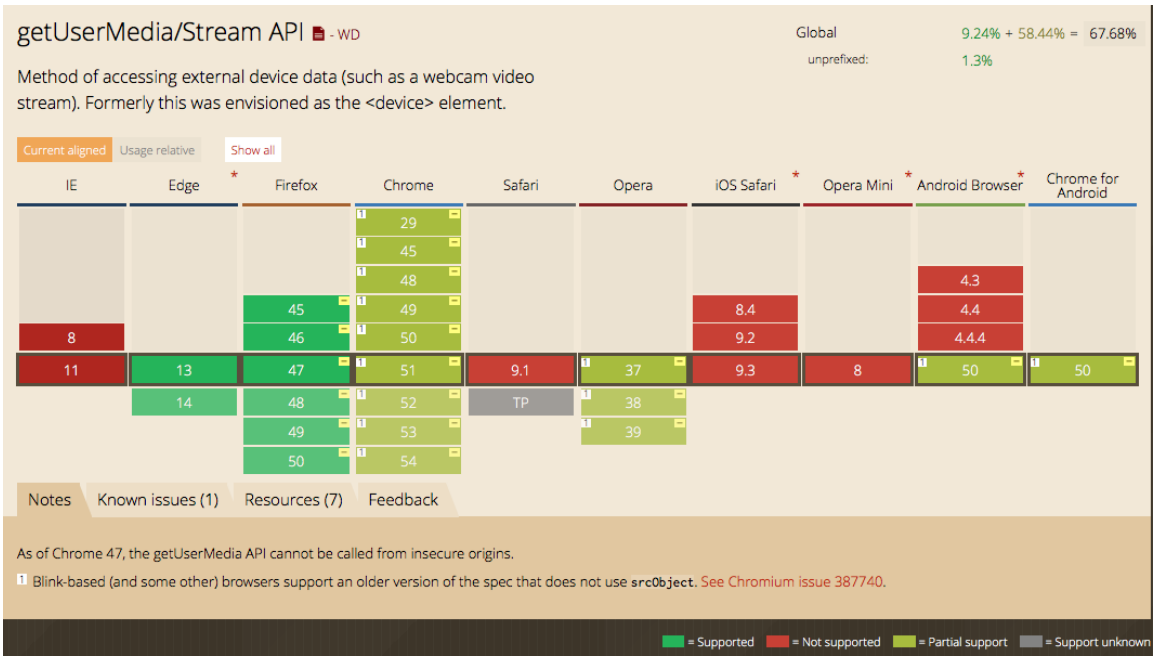

Abbildung 8.5: Browser Support für HTML5 getUserMedia/Stream API (Can I Use - getUserMedia/Stream API 2016)

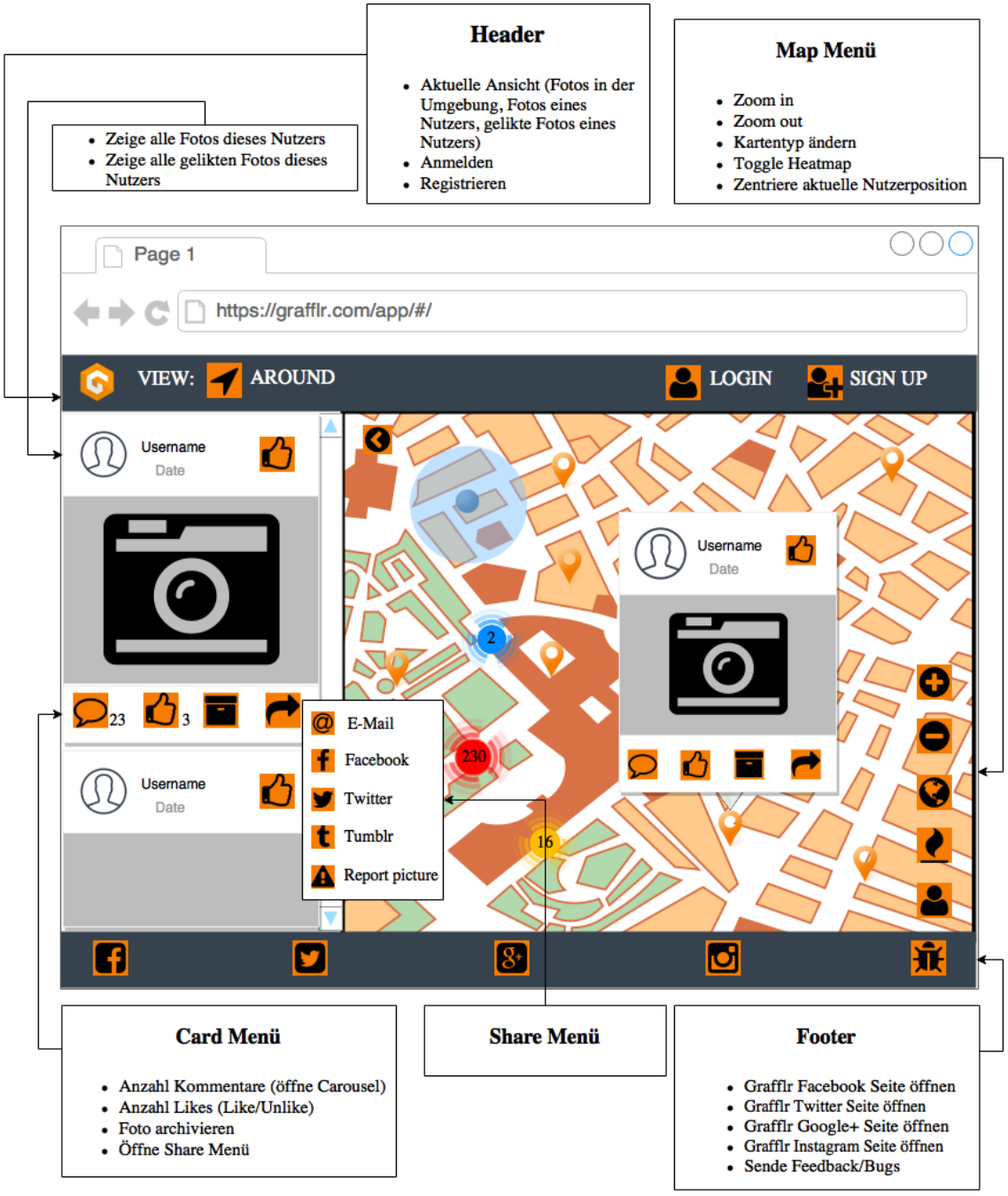

Abbildung 8.6: Mockup der Besucher Ansicht (eigene Darstellung)

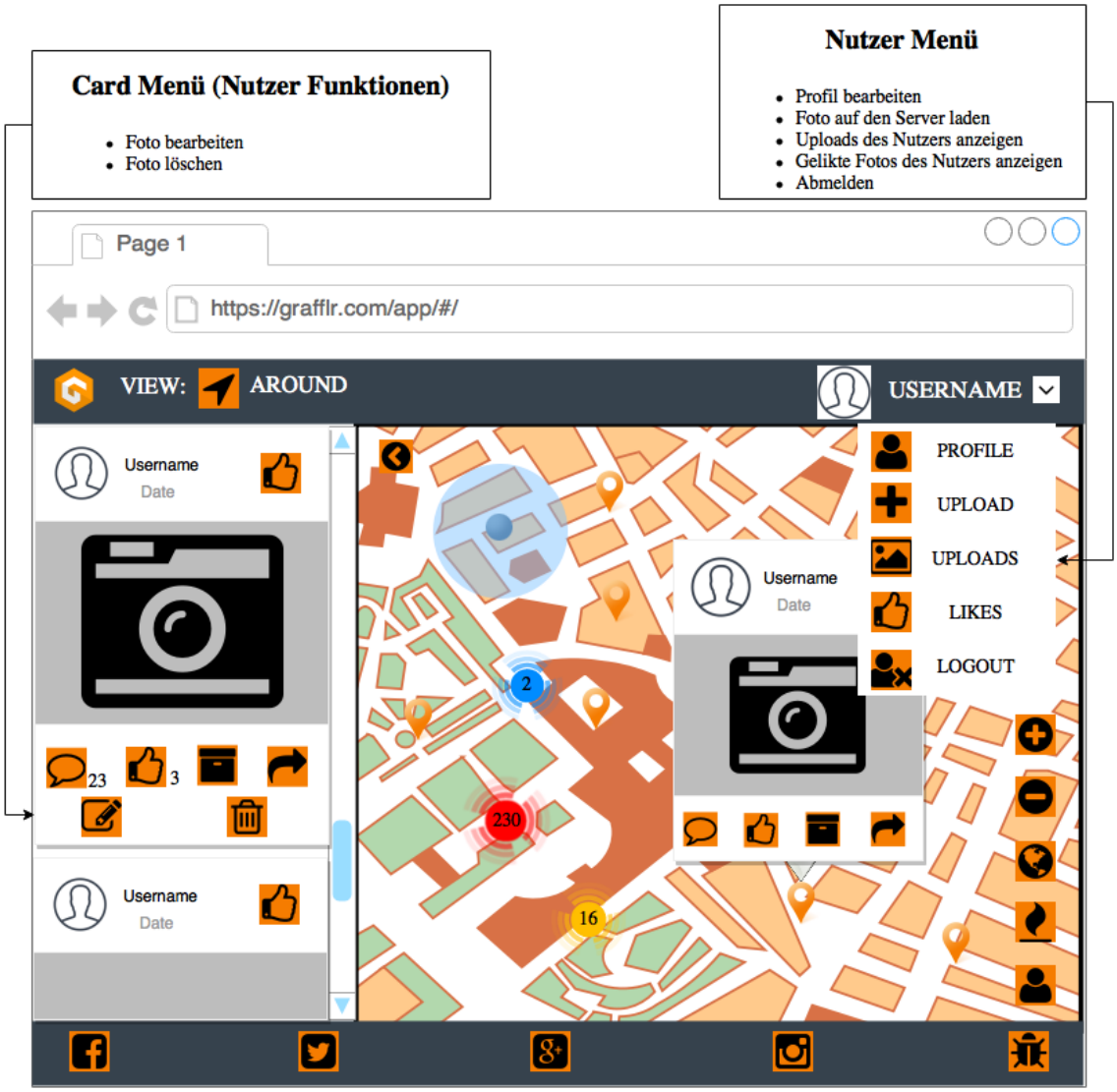

Abbildung 8.7: Mockup der Ansicht eines registrierten Nutzers (eigene Darstellung)

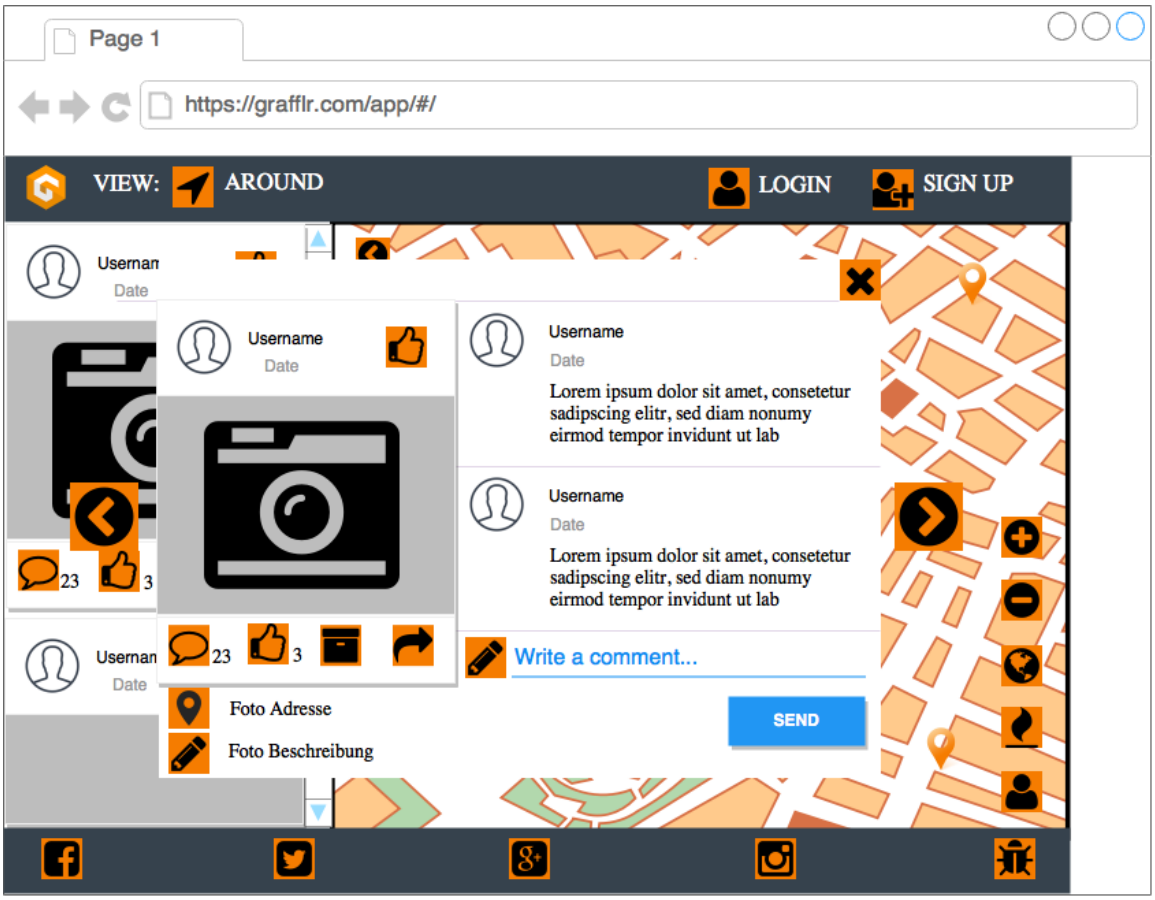

Abbildung 8.8: Mockup der Carousel Ansicht (eigene Darstellung)

8.B Inhalt der CD

- **classes/**
  In diesem Ordner befinden sich die Bilder zu dem Klassendiagramm.
- **klassendiagram.png**
  Das Klassendiagramm zu den ECMA Script 6 Klassen
- **kruse_jan_hendrik_160718.pdf**
  Das PDF dieser Bachelorarbeit.
- **latex/**
  In diesem Ordner befindet sich der LaTeXQuellcode inklusive der Bilder zum Erstellen dieses PDF Dokumentes.
- **rest_api_doku/**
  Der Order mit der HTML Datei der detaillierten REST API Dokumentation für den Zugriff auf das Backend.
- **umfrage/ergebnisse/**
  Der Ordner mit den PDF Dateien der Diagramme zu der Auswertung Nutzerumfrage für die Evaluation (siehe Abschnitt [6.2]).
- **umfrage/SurveyMonkey.pdf**
  Die PDF Datei mit den Fragen zu dem Praxis-Test der Evaluation (siehe Abschnitt [6.1]).
- **webapp/app/**
  In diesem Ordner befindet sich der Quellcode für die, in dieser Bachelorarbeit entwickelte, Web-App inklusive einer README Datei für dessen Installation und Deployment.

Quellcodeverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
APP	Applikation
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DOM	Document Object Model
EXIF	Exchangeable Image File Format
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MB	Megabyte
OECD	The Organisation for Economic Co-operation and Development
SDK	Software Development Kit
SSH	Secure Shell
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
UWP	Universal Windows Platform
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protoco
WSDL	Web Services Description Language
XML	Extensible Markup Language

Literaturverzeichnis

Borodescu, Ciprian: *Web Sites vs. Web Apps: What the experts think*, http://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think/, 2013, letzter Zugriff: 03.06. 2016

Can I Use: *DeviceOrientation & DeviceMotion events*, http://caniuse.com/#search=DeviceOrientation, 2016, letzter Zugriff: 15.06.2016

Can I Use: *Geolocation*, http://caniuse.com/#search=geolocation, 2016, letzter Zugriff: 15.06.2016

Can I Use: *getUserMedia/Stream API*, http://caniuse.com/#search=getUserMedia, 2016, letzter Zugriff: 15.06.2016

Can I Use: *IndexedDB*, http://caniuse.com/#search=indexeddb, 2016, letzter Zugriff: 05.06.2016

Can I Use: *Web Audio API*, http://caniuse.com/#search=Web%20Audio%20Api, 2016, letzter Zugriff: 15.06.2016

Can I Use: *Web Storage*, http://caniuse.com/#search=namevalue-storage, 2016, letzter Zugriff: 05.06.2016

Can I Use: *Web SQL Database*, http://caniuse.com/#search=sql-storage, 2016, letzter Zugriff: 05.06.2016

Cooper, Alan (Hrsg.): *About Face: The Essentials of Interaction Design*, 4. Aufl., John Wiley & Sons, Inc. 2014

Duden: *App, die oder das*, http://www.duden.de/rechtschreibung/App, 2016, letzter Zugriff: 30.05.2016

Geostreetart: *Mission*, http://geostreetart.com/mission/, 2016, letzter Zugriff: 31.05.2016

GoodUI: *The 75 GoodUI ideas*, http://www.goodui.org, 2016, letzter Zugriff: 25.06.2016

Krug, Steve: *DON'T MAKE ME THINK*, 2. Aufl., New Riders 2006

OECD: *The App Economy*, http://www.oecd.org/officialdocuments/publicdisplaydocumentpdf/?cote=DSTI/ICCP/IE(2012)1/FINAL&docLanguage=En, 2013, letzter Zugriff: 30.05.2016

Pracucci, Marco: *Electron? It works for us, and makes desktop fun and fast*, http://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think/, 2013, letzter Zugriff: 03.06. 2016

Statista: *Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015*, http://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/, 2015, letzter Zugriff: 05.06.2016

Wikipedia: *Anwendungssoftware*, https://de.wikipedia.org/wiki/Anwendungssoftware#cite_note-DudenInf-1, 2016, letzter Zugriff: 31.05.2016

Wikipedia: *Single-page-Webanwendung*, https://de.wikipedia.org/wiki/Single-page-Webanwendung, 2016, letzter Zugriff: 14.06.2016

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Jan-Hendrik Kruse