Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Konzeption und Realisierung einer hybriden App auf Basis moderner Webtechnologien

Bachelorarbeit an der HAW Hamburg

Vorgelegt von:
Richard Holzner
2079761
richard.holzner@gmail.com

Erstkorrektor:
Prof. Dr. Andreas Plaß

Zweitkorrektor:
Prof. Dr. Nils Martini

Fassung vom 29.08.2015

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Media Systems
am Department Medientechnik
der Fakultät DMI
der Hochschule für Angewandte Wissenschaften Hamburg

Kurzfassung

Moderne mobile Browser bieten dank HTML5 und CSS3 neue Möglichkeiten: Mit sogenannten WebApps lassen sich Anwendungen realisieren, welche plattformunabhängig laufen und somit einen größeren Anteil an Betriebssystemen erreichen können. Mithilfe verschiedener Hybrid Frameworks wird eine WebApp zu einer nativen App portiert. Diese Arbeit beschäftigt sich mit der Konzeption und Realisierung einer hybriden App. Dazu wird erst eine WebApp auf Basis moderner Webtechnologien wie HTML5, CSS3 und JavaScript bzw. diversen Frameworks entwickelt, anschließend wird diese App zu einer nativen App konvertiert. Dieser Vorgang durchläuft eine Anforderungsphase, eine Konzeptionsphase sowie eine Realisierungsphase.

Inhalt

Abkürzungsverzeichnis

URL	*Uniform Research Locator*
HTML	*Hypertext Markup Language*
CSS	*Cascading StyleSheets*
JS	*JavaScript*
API	*Application Programming Interface*
XML	*Extensible Markup Language*
Ajax	*Asynchronous JavaScript and XML*
HTTP	*Hypertext Transfer Protocol*
MVC	*Model View Controller*
MVVM	*Model View ViewModel*
PHP	*PHP HyperText Preprocessor*
SQL	*Structured Query Language*
RWD	*Responsive WebDesign*

1 Einleitung

1.1 Motivation

Mobile Geräte wie Smartphones oder Tablets sind inzwischen kaum noch wegzudenken aus dem Alltag der heutigen Gesellschaft, gewinnen sie doch immer mehr an Bedeutung. Mittlerweile nutzen bereits rund 45 Millionen Deutsche ein Smartphone, im Vergleich dazu lag diese Zahl im Jahr 2011 noch bei 18 Millionen (vgl. Statista [1]). Bis zum Jahr 2018 soll dieser Anteil an Nutzern auf 70% ansteigen (vgl. Statista 2]). Mit dem rasanten Wachstum der Anzahl an Geräten steigt auch das Angebot der verfügbaren mobilen Applikationen (Apps). Diese Apps sind ein wesentlicher Bestandteil der mobilen Nutzung geworden. Auf den zwei Marktplätzen der größten Plattformen, nämlich Android und iOS, stehen inzwischen jeweils über 1,4 Millionen Applikationen (vgl. Statista [3]) zum Download bereit. Bei der Entwicklung einer App sollte allerdings beachtet werden dass diese aufgrund der Inkompatibilität für jedes Betriebssystem separat entwickelt werden muss. Dies verursacht oft hohe Entwicklungskosten da Konzeption und Implementierung mehrfach durchgeführt werden müssen. Zudem unterliegt der Markt einem starken, sich ständig im Wandel befindenden Wettbewerb. Um diese Herausforderungen zu bewältigen, liegt die Entwicklung von plattformunabhängigen, sogenannten hybriden Apps nahe. Entwicklungskosten können durch die einmalig durchzuführende Konzeption und Realisierung gesenkt werden, auch das Risiko der Abhängigkeit von einzelnen Plattformen wird dadurch verringert. Eine Möglichkeit, plattformunabhängige Apps zu entwickeln, besteht in der Nutzung von Webtechnologien. Hierbei wird der Code nur einmal geschrieben, ausgeführt wird eine solche App in einem leistungsstarken Browser, welcher die gemeinsame Schnittstelle der verschiedenen Betriebssysteme bildet.

1.2 Ziel

Das Ziel dieser Arbeit ist die Konzeption und Realisierung einer hybriden App auf Basis moderner Webtechnologien wie HTML5, CSS3 und JavaScript, welche mit einmaliger Entwicklung eine plattformübergreifende Nutzung ermöglichen soll. Bei der App handelt es sich um einen Kurznachrichtendienst, vergleichbar beispielsweise mit Twitter. Im Rahmen dieser Arbeit wird der komplette Entstehungsprozess einmal durchlaufen, am Ende des Prozesses soll ein lauffähiger Prototyp stehen. Dieser Prototyp soll auf verschiedene Plattformen portierbar sein. Dargestellt wird der Verlauf mittels Anforderungsphase, Konzeption- bzw. Designphase und Realisierungsphase. Insbesondere werden in der Realisierung der Aufbau sowie die Funktionsweise der App verdeutlicht. Außerdem kommen verschiedene Webtechnologien wie das JavaScript Framework AngularJS zum Einsatz.

1.3 Aufbau

Die vorliegende Arbeit gliedert sich in sieben Kapitel:

Kapitel eins (*Einleitung*) gibt einen groben Überblick sowie einen Einstieg in das behandelte Thema.

In Kapitel zwei (*Theorie und Grundlagen*) werden die Grundlagen behandelt. Hier werden Grundbegriffe erläutert, die für das Verständnis der App-Entwicklung wie auch dieser Arbeit notwendig sind. Zudem beschäftigt sich dieser Teil mit den aktuellen Webtechnologien und den verwendeten Frameworks.

Kapitel drei (*Anforderungen*) beinhaltet die genauen Anforderungen der App, welche im Rahmen dieser Arbeit erstellt wird. Unterteilt werden diese in funktionale und nicht funktionale Anforderungen.

Das Kapitel vier (*Konzeption*) beschäftigt sich mit der Planung und dem Entwurf dieser App. Hierbei werden Mockups entwickelt, welche die zukünftige Bedienoberfläche

zeigen. Außerdem wird ein Styleguide entworfen, um designtechnische Aspekte festzulegen.

Kapitel fünf (*Realisierung*) zeigt den Vorgang der Implementierung. Hier entsteht – unter Beachtung des entwickelten Konzepts – ein erster Prototyp der App. Dieser Vorgang wird gezeigt am Beispiel einzelner Aspekte der Implementierung, welche hierfür näher beleuchtet werden.

Kapitel sechs (*Anforderungsabgleich*) gleicht die anfangs aufgestellten Anforderungen mit den realisierten Ergebnissen ab.

Im siebten und letzten Kapitel (*Schluss*) folgt eine Zusammenfassung der Arbeit. Außerdem wird ein Fazit gezogen sowie ein Ausblick auf eine mögliche Erweiterung des Prototypen gegeben.

2 Theorie und Grundlagen

Dieses Kapitel beschreibt die Technologien, die zur Konzeption und Realisierung einer hybriden App notwendig sind.
Die behandelten theoretischen Inhalte dienen als Grundwissen für das weitere Verständnis der nachfolgenden Kapitel dieser Arbeit.

2.1 Unterschiedliche Arten mobiler Anwendungen

Die nachfolgenden Kapitel stellen die unterschiedlichen Arten von mobilen Anwendungen dar. Hierbei werden kurz die Vor- und Nachteile erläutert, im Anschluss daran wird die Wahl der verwendeten Anwendung begründet.

2.1.1 Native App

Eine native App wird für ein bestimmtes Betriebssystem konzipiert und erstellt. Je nach Betriebssystem werden unterschiedliche Technologien verwendet. Auf Android laufende Apps sind in der Programmiersprache Java entwickelt, iOS Apps hingegen mittels Objective-C. Das Aussehen einer solchen App lässt sich leicht produzieren und an das standardmäßige Design, dem sogenannten „Look and Feel“ des jeweiligen Betriebssystems anpassen. Ein weiterer Vorteil liegt im schnellen Zugriff auf Systemkomponenten wie Kamera, Sensoren oder der GPS-Position. Der Zugriff auf diese gerätespezifische Hardware erfolgt in der Regel schneller als bei WebApps, da die erforderlichen systemnahen Schnittstellen bereits gegeben sind. Die Installation erfolgt nach einem Download aus dem jeweiligen Store. Die Entwicklungskosten sind

allerdings vergleichsweise hoch, da für jedes Betriebssystem eine komplett neue Programmierung erfolgen muss.

2.1.2 WebApp

Im Gegensatz zu einer nativen App benötigt eine WebApp keine Installation, da diese direkt im Browser über eine URL aufgerufen wird und sich der jeweiligen Nutzeroberfläche des Endgerätes anpasst. Programmiert wird diese mittels Webtechnologien wie HTML, CSS und JavaScript. Der Vorteil liegt hierbei in der betriebssystemunabhängigen Entwicklung, welche nur ein einziges Mal erfolgen muss, um anschließend auf allen Geräten zu laufen. Um das standardisierte, dem Betriebssystem entsprechendes Aussehen zu erhalten, sind aufwendigere Maßnahmen notwendig, da keine Bibliotheken vorhanden sind. Der Zugriff auf Hardwarekomponenten ist jedoch kaum möglich, da keine notwendigen Schnittstellen vorhanden sind, um auf native Plattform-Funktionen zugreifen zu können.

2.1.3 Hybride App

Eine hybride App vereint die Vorteile einer nativen App mit denen einer WebApp. Entwickelt wird diese ebenfalls mit HTML, CSS und JavaScript, kann allerdings mithilfe von Frameworks in einen nativen App-Container eingebettet werden. Dies ermöglicht den Zugriff auf Hard- und Softwarekomponenten des Endgerätes, da diese Frameworks die erforderlichen Schnittstellen bereitstellen. Die Entwicklung einer WebApp bzw. hybriden App ist zwar günstiger als die einer nativen App, jedoch sollte im Einzelfall immer zuerst geprüft werden, welche Anforderungen die App erfüllen soll. Sind beispielsweise der schnelle Zugriff auf Hardwarekomponenten oder allgemeine performancekritische Operationen erforderlich, bietet sich eine native App an. Beschränkt sich die Verwendung lediglich auf das relativ simple Abrufen, Speichern und Darstellen von Daten, kann eine WebApp die richtige Wahl sein.

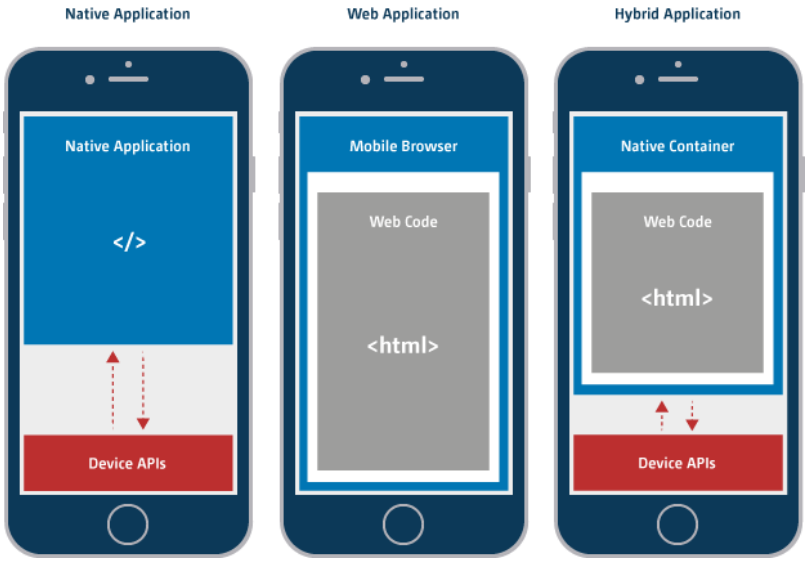

Abbildung 2-1 Unterscheidung der App Arten
Quelle: Innotix [22]

Abbildung 2-1 stellt noch einmal die verschiedenen Arten auf grafische Weise dar. In dieser Arbeit wird die Umsetzung einer hybriden App behandelt. Diese Version wird verwendet, da kein schneller Zugriff auf etwaige Hardware gefordert ist, die Funktionen beschränken sich auf die Abfrage und Speicherung von Daten über das Internet. Zur einfachen Portierbarkeit auf die verschiedenen Plattformen wird das Framework „Phonegap“ verwendet, welches im späteren Verlaufe dieses Kapitels noch vorgestellt wird.

2.2 Verwendete Technologien

In den nachfolgenden Absätzen werden die verwendeten Webtechnologien kurz vorgestellt.

2.2.1 HTML

Bei der Hypertext Markup Language handelt es sich um eine Auszeichnungssprache für Webdokumente. Sie dient dazu, ein Dokument semantisch zu strukturieren bzw. bildet das Grundgerüst einer Website.

Die neueste Version HTML5 bringt eine Reihe moderner Features mit sich. So ist es beispielsweise möglich, multimediale Inhalte auf mobilen Geräten abzubilden, den aktuellen Standort über die Geolocation API zu ermitteln oder Inhalte einer Website in einer Offline-Datenbank zu speichern.
Diese Eigenschaften bilden oftmals die essentiellen Grundfunktionen einer WebApp. Das hat mittlerweile dazu geführt, dass sich die WebApp zu einem bedeutenden Konkurrenten der nativen App entwickelt hat.

2.2.2 CSS

Cascading StyleSheets bezeichnet eine Formatierungssprache für HTML- und XML-Dokumente. Zusammen mit HTML bildet sie die Grundlage für das World Wide Web. Sinn dieser Sprache ist die Trennung von Inhalt und Darstellung. Im Gegensatz zu HTML dient CSS daher der reinen optischen Strukturierung eines Dokuments, also dem Gestalten des Layouts. Mit der Einführung der neuesten Version CSS3 hielten auch die sogenannten „Media Queries“ Einzug in die Welt der mobilen Anwendungsprogrammierung. Mit ihnen ist es möglich, genau definierten Bildschirmbreiten unterschiedliche Darstellungen zuzuweisen. Die erforderlichen Breiten erstrecken sich von Smartphones über Tablets bis hin zu großen Desktop-Monitoren. Je mehr Platz zur Verfügung steht, desto größer kann auch die Darstellung von Inhaltselementen erfolgen. Das Listing 2-1 zeigt ein Container Element, welches zwischen 480 Pixel und 767 Pixel eine Breite von 100% besitzt, ab 768 Pixel ändert sich diese Breite auf 90%. Media Queries sind ein wichtiger Teil des Responsiven Webdesigns.

```
@media screen and (min-width:480px;) and (max-width:767px){
        // Displays mit 480px Mindestbreite
                #container{
                        width:100%;
                }
}

@media screen and (min-width:768px;) {
        // Displays mit 768px Mindestbreite
                #container{
                        width:90%;
```

```
            }
}
```

Listing 2-1 CSS Media Queries

2.2.3 Responsive Webdesign

Responsive Webdesign – im übertragenen Sinne „reagierendes Webdesign“ - ist eine bestimmte Art von Webdesign, bei der flexibel auf die Größe und Eigenschaften des jeweiligen Endgerätes reagiert wird. Im Gegensatz zu vielen älteren Webseiten, deren Layout und Struktur starr und unflexibel programmiert sind, wird beim RWD ein anderer Ansatz verfolgt. Der Inhalt sowie dessen Struktur als auch das daraus resultierende Layout passen sich der Bildschirmauflösung des mobilen Endgerätes an – es reagiert individuell auf den Nutzer. Auch bei der Programmierung einer WebApp sollte dies beachtet werden, da auch hier bereits die Auflösung stark variieren kann, von älteren, noch gering auflösenden Smartphones bis hin zu moderneren Tablets. In Abbildung 2-2 ist die Verhaltensweise eines flexiblen Layouts bei verschiedenen Bildschirmbreiten dargestellt.

Abbildung 2-2 Responsive Design auf verschiedenen Devices
Quelle: Comvatix [23]

2.2.4 JavaScript

Bei JavaScript handelt es sich um eine clientseitige Skriptsprache, sie wird also im Browser des Benutzers ausgeführt. Sie dient dazu, eine Website dynamischer zu gestalten bzw. auf das Verhalten des Nutzers, wie z.B. das Klicken eines Buttons, oder

die Eingabe in ein Formularfeld, zu reagieren und entsprechende Aktionen auszulösen. Die dadurch bedingten Änderungen können sowohl den Inhalt als auch die Darstellung betreffen. Geregelt wird dieser Zugriff über das sogenannte „Document Object Model (DOM)“. Insbesondere bei der Erstellung einer WebApp ist JavaScript unverzichtbar, denn damit wird die komplette Anwendungslogik gesteuert. Sämtliche Interaktionen des Users mit der App, beispielsweise das Speichern von eingegebenen Daten in einem Formularfeld sowie das Abrufen von Daten aus einer Datenbank, erfolgen dadurch.

2.2.5 DOM

Das Document Object Model ist eine Schnittstelle zu HTML oder XML-Dokumenten, es bildet die logische Struktur von Dokumenten und bietet Zugriff auf alle Elemente einer HTML-Seite. Der Aufbau entspricht einer Baumstruktur, die Elemente werden in Beziehung zueinander gesetzt. In der Regel wird mit JavaScript auf einen bestimmten Teil des Baumes zugegriffen, Inhalt oder Layout verändert und dem User sichtbar gemacht; man spricht vom sogenannten „Dom-Scripting“ (vgl. W3C DOM [20]). Bei einer WebApp ist dies ein wichtiger Bestandteil, da die App mit dem User interagiert bzw. auf seine Aktionen reagiert und diese auch entsprechend visualisiert.

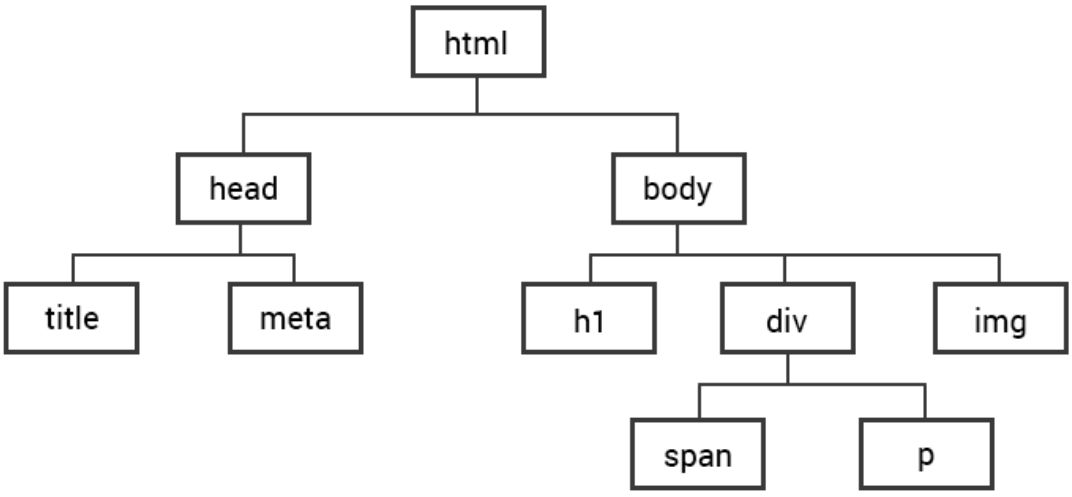

Abbildung 2-3 DOM Baumstruktur eines HTML Dokuments
Quelle: Eigene Abbiildung

2.3 Verwendete Frameworks

2.3.1 jQuery

jQuery ist eine schlanke, umfangreiche Open-Source-JavaScript-Bibliothek. Vorteilhaft ist hierbei, dass viele häufig genutzte JavaScript-Funktionen dem Programmierer bereits zur Verfügung stehen und nicht mehr aufwendig mit nativem JavaScript-Code geschrieben werden müssen. Mit nur wenigen Codezeilen kann eine Vielzahl an Funktionen eingebunden werden, dies spart wiederrum Zeit und Ressourcen.
jQuery bietet umfangreiche Möglichkeiten zur Navigation und Manipulation von DOM Elementen. Des Weiteren hält es Funktionen zum Event-Handling und Ajax-Funktionalitäten bereit, die der einfachen Kommunikation mit dem Server dienen.

2.3.2 AngularJS

Bei AngularJS (vgl. W3 School AngularJS [12]) handelt es sich um ein JavaScript-Framework beruhend auf dem MVC bzw. MVVM-Entwurfsmuster. Es findet vor allem Verwendung für die Erstellung von dynamischen browserbasierten Webapplikationen bzw. Single-Page-Applikationen.
Eine Anwendung mit AngularJS wird meist von Anfang an basierend auf dessen Einsatz konzipiert und entwickelt, es ist jedoch auch ohne Probleme möglich – ähnlich wie bei jQuery – es nachträglich zu integrieren und nur teilweise, aber gezielt einzusetzen.
In den nächsten Abschnitten werden die wichtigsten Merkmale von AngularJS kurz vorgestellt.

2.3.2.1 Anlegen eines Projekts

Zuerst muss die App definiert werden, indem ein Modul deklariert und Angular zugeordnet wird. Anschließend wird dieser Modulname im HTML angelegt, wodurch der Zugriffsbereich für Angular festgelegt ist.

```
//Anlegen des Moduls
var app = angular.module('myApp');

// NG Deklaration
<div id=“container“ ng-app=“myApp“>
        //HTML
</div>
```

Listing 2-2 Anlegen eines AngularJS Moduls

2.3.3 Controller

Jeder benötigten View muss ein Controller zugewiesen werden, damit dieser das Model aktualisieren und synchron halten kann. Der Controller nimmt beispielsweise Input-Daten vom User auf, übergibt diese an einen Service, welcher diese wiederum an einen Server sendet und Daten zurück erhält. Die erhaltenen Daten werden dann vom Controller an die View übermittelt. Das Dokument sollte idealerweise keine hard-codierten Texte enthalten, um die Anwendung dynamisch zu nutzen.

```
app.controller(‚NaviCtrl‘,[‚$scope‘,function ($scope) {…}]};
app.controller(‚MainCtrl‘,[‚$scope‘,function ($scope) {…}]};
app.controller(‚ContentCtrl‘,[‚$scope‘,function ($scope) {…}]};
```

Listing 2-3 Initialisierung verschiedener Controller

Neue Dateien, egal ob Controller oder Service, werden an diese Variable bzw. das Modul gebunden und sind Angular somit bekannt gemacht.

2.3.2.3 Services

Ein Controller ist zuständig für die Geschäftslogik, also die Kommunikation zwischen der View und dem Model, beispielsweise Daten in die View zu bringen sowie Daten

entgegenzunehmen. Daher gibt es zusätzlich Services, welche die erforderliche Datenlogik enthalten.
Wiederverwendbare und komplexere Logik, die bei einer Anwendung mehrfach benötigt wird, sich jedoch nicht ändert, kann in einen Service ausgelagert werden und gewissermaßen vom Controller konsumiert werden.
AngularJS liefert zudem eine Menge bereits vordefinierter Services mit, z.B. den $http Service, um mit einem Server zu kommunizieren (vgl. AngularJS Docs [16]), gekennzeichnet sind diese mit einem Dollar-Zeichen.

```
// Service Aufruf
app.service('DataService', function ($http){
   this.getConnection = function () {
       var request = $http.get('data.json');
         request.success(function (data, status, headers, config){
         // do something
       });
request.error(function (data, status, headers, config){
           // do something
       });
}
});
```

Listing 2-4 Service Aufruf

2.3.2.4 Dependency Injection

Konventionell kümmert sich eine Funktion selbst um das Erstellen neuer Objekte.
Aufgrund der modularen Bauweise von AngularJS jedoch kann ein Service - gleich ob es sich um einen vordefinierten oder einen eigens erstellten handelt – in einem Controller oder einem anderen Service „injiziert“ werden, um von diesem verwendet werden zu können. Hierbei spricht man von der sogenannten Dependency Injection. Der Code wird dadurch wiederverwendbarer und wartbarer.

```
// Checkconnection  Service in DataService
app.service('DataService', function ($http,CheckconnectionService){
    this.getConnection = function () {
        // HTTP Request
```

```
    }
});
```

Listing 2-5 Checkconnection Service zur Prüfung einer vorhandenen Verbindung

2.3.2.5 Directiven

Unter einer Directive versteht man eine „Markierung“ im DOM, an der Angular einen bestimmten Programmcode aufruft und die View an dieser spezifischen Stelle beeinflusst bzw. mit der Logik verbindet. Ähnlich wie bei den Services bietet Angular auch hier bereits eine Reihe an vorgefertigen Directiven. Diese werden hauptsächlich verwendet, um die View zu aktualisieren bzw. um DOM-Manipulationen vorzunehmen. Alternativ kann eine Directive auch selbst erstellt werden, falls keine der mitgelieferten Directive dem gewünschten Zweck entspricht.

```
// ng-repeat Directive
// Anzeige mehrerer Elemente in einer Liste
<div controller=“MainCtrl“>
    <ul>
        <li ng-repeat="item in myData></li>
    </ul>
</div>
```

Listing 2-6 Directive zum Anzeigen mehrerer Listenelemente

2.3.2.6 Scopes

Jeder Controller besitzt ein Scope-Objekt, welches dazu dient, die Funktionen und Variablen im Gültigkeitsbereich des Controllers bereitzuhalten. Dieses Scope-Objekt ist eine Art Vermittlungsschicht zwischen Controller und der View.

```
<div controller=“MainCtrl“>
    {{ message }}
</div>
```

```
app.controller(MainCtrl, function($scope,) {
    $scope.message = "Hello World“;
});
});
```

Listing 2-7 Anzeige einer Scope Variable

2.3.2.7 One Way / Two Way Data Binding

Einfache Datenbindung (One-Way) bezeichnet das Bereitstellen von Daten, welche mit einem Template verbunden und anschließend in die View übergeben werden.
Das hierbei entstehende Problem ist, dass sich anschließende Änderungen im Modell nicht mehr auf die View auswirken.
Anders verhält es sich bei der doppelten Datenbindung (Two-Way): Hier wird eine Verbindung zwischen dem Modell und den zugehörigen Daten in der View hergestellt. Gibt der User beispielsweise in einem Input-Feld einen neuen Wert an, wird dies erkannt und im Modell aktualisiert. Andersrum wird bemerkt, wenn sich ein Wert im Modell verändert oder entfällt und wird dementsprechend in der View aktualisiert.

```
//Input Feld wird an ng-model Directive gebunden
<div controller=“MainCtrl“>
   <input type=“text“ ng-model=“message“ />
   <p>Your Message: {{ message }}<p>
</div>
```

Listing 2-8 Inut Feld mit Directive

2.3.4 Phonegap

Phonegap ist ein Cross-Plattform-Framework für mobile Anwendungen bzw. WebApps, mithilfe dessen hybride Apps erstellt werden können. Phonegap bietet die Möglichkeit, eine WebApp als native App zur Verfügung zu stellen, indem sie als Wrapper fungiert und die WebApp in eine Art Container einbettet (vgl. Phonegap [7]). Zudem besteht dadurch Zugriff auf native Plattform-Funktionen, was bei einer reinen WebApp hingegen nicht realisierbar ist. So ist es beispielsweise möglich, per JavaScript Hardwarekomponenten wie die Kamera oder den Beschleunigungssensor anzusprechen. Die Entwicklung erfolgt mittels HTML, CSS und JavaScript, im Anschluss daran wird diese in Phonegap importiert und steht als eigenständige, native App zur Verfügung. Die App kann dann im jeweiligen Plattform-Container exportiert und über die üblichen Marktplätze angeboten werden, in der Art der Installation besteht kein Unterschied mehr zur nativen Version.

2.4 Single-Page-Application Prinzip

2.4.1 Allgemeines

Eine „Single Page Application“ (SPA) ist eine App bestehend aus nur einer einzigen Webseite, deren sich ändernde Inhalte von JavaScript dynamisch bestimmt werden. Die Serverkommunikation zum Laden neuer Daten findet im Hintergrund statt, ohne die gesamte Seite neu anzufordern. Diese Art von Webseiten steht im Gegensatz zu einer konventionellen Webseite.
Abbildung 2-4 stellt den Unterschied zwischen den zwei Arten dar:
Bei einer konventionellen Webseite wird ein Request an den Server bzw. das Backend gestartet, dieser verarbeitet daraufhin z.B. eventuell vorhandene Usereingaben und generiert dazu eine fertige HTML-Seite für den Client. Die Programmlogik findet hier größtenteils serverseitig statt, mittels Skriptsprachen wie PHP. Bei einer Single Page Application hingegen beruht die Programmlogik auf der Client bzw. Frontend-Seite. Der komplette HTML Code wird nur ein einziges Mal angefordert, danach ist dieser auf der Client-Seite vorhanden.

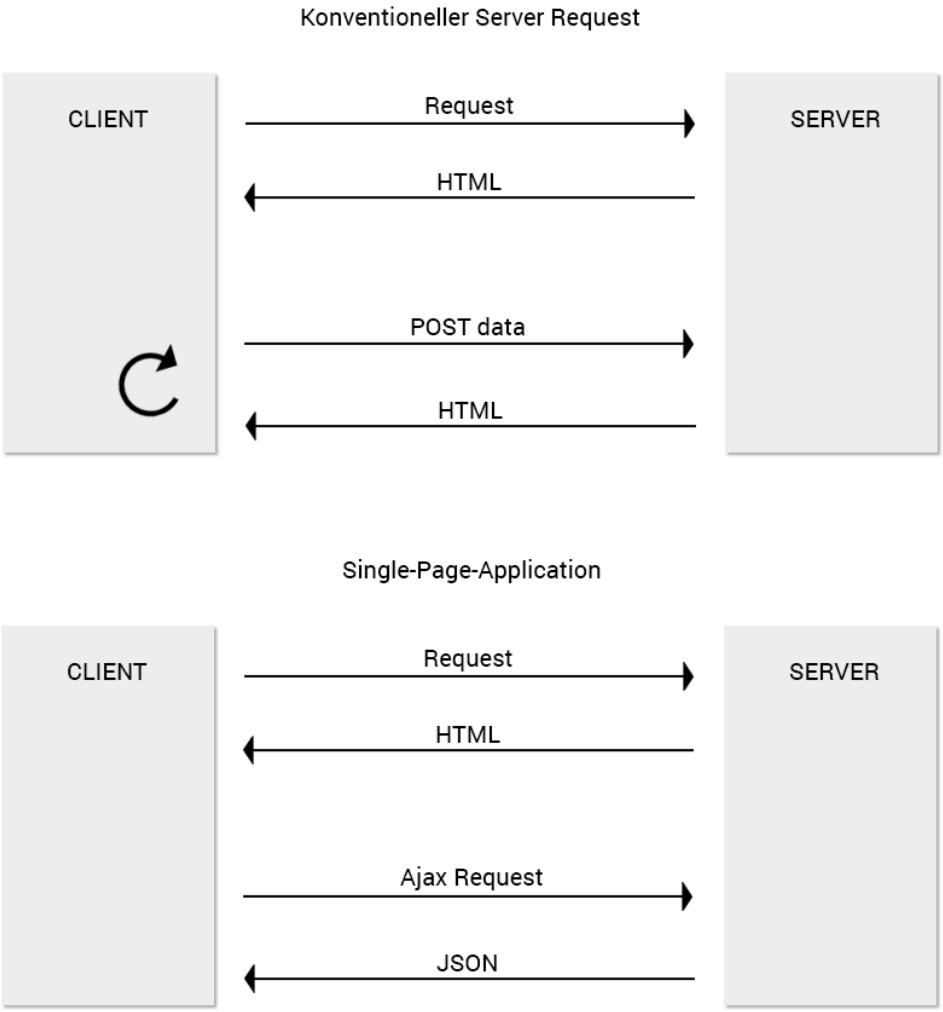

Abbildung 2-4 Konventioneller Server Request im Vergleich mit SPA Request
Quelle: Eigene Abbildung

Die anschließende Kommunikation zwischen Client und Server findet über asynchrone HTTP-Anfragen statt. Bevorzugtes Format zum Übermitteln von Daten ist hierbei JSON, ein von Programmiersprachen unabhängiges und einfach zu lesendes JavaScript-Textformat. Single Page Applications sind also, verglichen mit konventionellen Webseiten, um einiges schneller. Nachteilig wirkt sich die Verwendung allerdings beim Thema Suchmaschinenoptimierung aus. Nachgeladene Inhalte werden von Suchmaschinen nicht mehr erfasst, mögliche wichtige Inhalte sind also nur schwer auffindbar.

Es werden folglich keine kompletten HTML-Seiten übertragen, sondern nur noch die nötigen, angeforderten Daten, die von JavaScript – ohne die Seite neu zu laden - in die Webseite mit eingebaut werden. Die Anzahl der page reloads wird minimiert, es werden keine Daten mehr unnötigerweise mehrfach übertragen. Damit die Webseite auch die notwendige logische Struktur erhält, kommen hier verschiedene Entwurfsmuster wie MVC oder MVVM zum Einsatz.

2.4.2 MVC

Bei MVC handelt es sich um das Model-View-Controller-Entwurfsmuster, das in vielen Programmiersprachen zur Architektur von Software verwendet wird.
Ziel hierbei ist die Trennung von Layout und Programmcode bzw. die logische Strukturierung in verschiedene, wiederverwendbare Bereiche (vgl. Erich Gamma, Ralph Johnson [19]). Bei einer gut strukturierten Anwendung kann beispielsweise das Modell für andere Anwendungen wiederverwendet werden, nur noch Controller und View müssen neu implementiert werden.

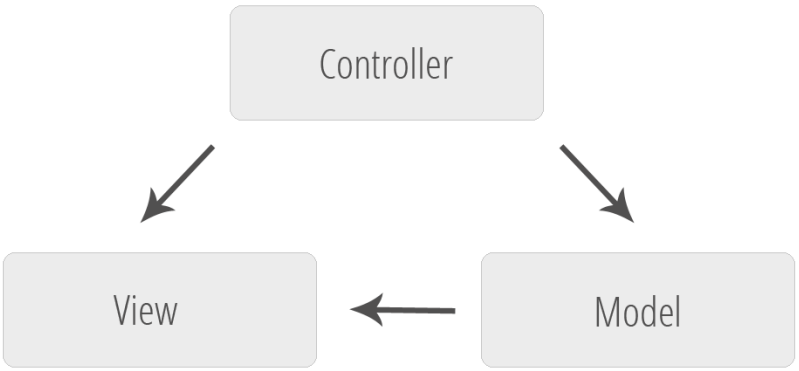

Abbildung 2-5 MVC Pattern Skizze
Quelle: Eigene Abbildung

Model: Das Model stellt das Anwendungsobjekt dar, also die Datenstruktur die zwischen Client und Server ausgetauscht wird. Das Model benachrichtigt die abhängigen Views, wenn sich Änderungen ergeben, diese werden daraufhin aktualisiert, um die Änderungen synchron zu halten.

View: Die View ist zuständig für die Bildschirmpräsentation, sie visualisiert die Daten, die das Model der Anwendung aufweist.

Controller: Der Controller beinhaltet die Programmlogik und reagiert auf Eingaben des Nutzers. Er dient als Vermittler zwischen Model und View, Interaktionen des Users werden vom Controller registriert und an das Model weitergeleitet.
Auch die Kommunikation mit dem Server wird vom Controller geregelt, erhaltene Daten werden wieder an das Model übergeben.

2.4.3 MVVM

Als MVVM wird das Entwurfsmuster Model-View-ViewModel bezeichnet, hierbei handelt es sich um eine Variante des MVC-Musters.
Model und View ähneln dem MVC-Muster, der Unterschied liegt beim ViewModel.
Zwischen dem Model und der View befindet sich das ViewModel, es fungiert als Bindeglied und beinhaltet die Logik.
Im Vergleich zum MVC, wo der Controller sowohl die View als auch das Model steuert, hat das ViewModel in diesem Falle keinerlei Kenntnis von der Existenz der View, es werden lediglich Daten für diese bereitgestellt.
Dies bietet wiederum den Vorteil, dass das ViewModel flexibel einsetzbar bleibt.

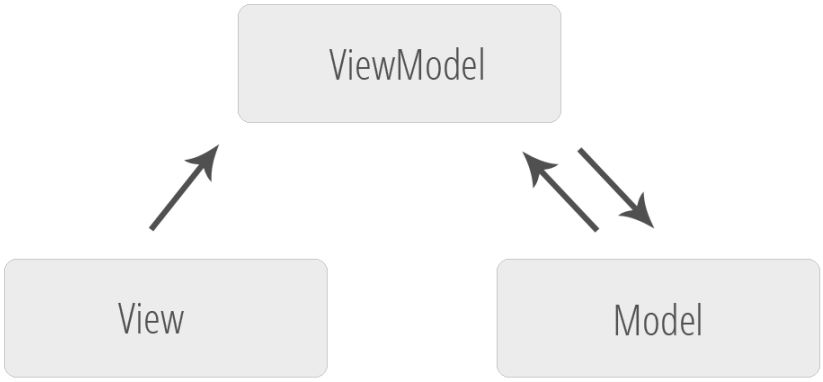

Abbildung 2-6 MVVM Pattern Skizze
Quelle: Eigene Abbildung

Bei Single Page Applications handelt es sich somit nicht – auch wenn es so scheint – um eine neuartige Technologie, sondern eine Zusammensetzung verschiedener Einzeltechnologien zu einem neuen Gesamtkonzept.

2.5 Sonstige Technologien

2.5.1 PHP

PHP Hypertext Preprocessor ist eine Skriptsprache, die, anders als JavaScript beispielsweise, serverseitig ausgeführt wird. PHP ist plattformübergreifend und wurde speziell für das Web entwickelt, um dynamische Webseiten oder Anwendungen erstellen zu können.
Im Falle einer Anfrage vom Client wird erst ein Skript auf dem Server ausgeführt, anschließend wird das fertige HTML-Dokument an den Client übergeben.
PHP findet häufig Verwendung in der Validierung und Auswertung von Formularen oder um Inhalte aus einer Datenbank lesen bzw. speichern zu können.

2.5.2 SQL

SQL ist eine Datenbanksprache zum Erstellen von Datenbankstrukturen in relationalen Datenbanken. Die am häufigsten verwendeten Datenbanken sind MySQL oder PostgreSQL. Diese werden häufig in Verbindung mit PHP verwendet, um Daten von web-basierten Anwendung zu verarbeiten, sogenannte CRUD (Create, Read, Update, Delete) Operationen.

2.5.3 JSON

JSON ist ein auf JavaScript basierendes leichtgewichtiges Format zum Austausch von Daten in einer sowohl für Mensch als auch Maschinen lesbaren Form.
Dieses Format hat den Vorteil, nur Daten zu übertragen, die gebraucht bzw. angefordert werden, der Overhead wird dabei mit jeder Anfrage auf ein Minimum reduziert.
JSON selbst ist bereits gültiges JavaScript, kann also leicht in ein JavaScript-Objekt

überführt werden. Die meisten Programmiersprachen bieten mittlerweile eine Schnittstelle, um JSON ohne Probleme zu implementieren. JSON wird auch oft in Verbindung mit asynchronen Requests, beispielsweise in Verbindung mit Ajax bei mobilen Apps, genutzt.

```
{
   "ID": "1",
   "username": "Richard",
   "gender": „male“,
   "age": 27
},
{
   "ID": "2",
   "username": "Marie",
   "gender": „female“,
  "age": 21
}
```

Listing 2-9 Inhalt eines JSON Object

2.5.4 Ajax

Bei Asynchronous JavaScript and XML (Ajax) handelt es sich um eine Technologie, die es ermöglicht Daten dynamisch von einem Server abzurufen, ohne die komplette Seite neu laden zu müssen.
Somit kann beispielsweise schneller auf Interaktionen des Nutzers reagiert werden.
Ein Ajax Request läuft asynchron ab, d.h. eine Webseite muss nicht auf eine Antwort warten da die Anfrage im Hintergrund ausgeführt wird, die Hauptanwendung läuft somit weiter. Bei einer klassischen Web Anwendung hingegen hat eine Anfrage ein komplettes Neuladen der Seite zur Folge.
XML – eine Auszeichnungssprache zum Darstellen von Daten in Textform – ist in vielen Fällen bereits überholt, mittlerweile werden asynchrone Requests mit JSON durchgeführt.

3 Anforderungen

Das Ziel dieser Arbeit ist es, den Entwicklungsverlauf einer hybriden App zu durchleuchten. Dieser Vorgang wird gezeigt am Beispiel einer Kurznachrichtendienst-App. Im folgenden Kapitel werden die funktionalen Anforderungen in Form von Use Cases sowie die nicht-funktionalen Anforderungen definiert. Die Nummerierung der einzelnen Anforderungen dient der späteren Referenzierung im Anforderungsabgleich.

3.1 Funktionale Anforderungen

Unter funktionalen Anforderungen versteht man konkrete Funktionalitäten, welche die Anwendung leisten soll.
Die Anforderungen lassen sich in mehrere Hauptanwendungen aufteilen, die dem User zur Verfügung stehen bzw. in Anwendungen, die keinen Nutzerzugriff erfordern und sich im Hintergrund ereignen.

1. Authentifizierung
2. News-Anzeige
3. Einstellungen
4. Suche
5. Verfassen eines neuen Posts
6. Profilseite
7. Hinweis auf nicht vorhandene Netzwerkverbindung

3.1.1 Authentifizierung

Nr.	Bezeichnung	Beschreibung
FA#1.1	Anmeldung	Der Benutzer kann sich mit Username und Passwort anmelden.
FA#1.2	Registrierung	Der Benutzer kann sich mit Username und Passwort registrieren.

Tabelle 3-1 Funktionale Anforderungen Authentifizierung

3.1.2 News Anzeige

Nr.	Bezeichnung	Beschreibung
FA#2.1	Anzeige aller News	Der Benutzer kann sich alle Posts mit folgenden Details anzeigen lassen: • Name des Autors • Datum • Post • NEW Marker • Vergangene Zeit seit dem Posten
FA#2.2	Anzeige der abonnierten News	Der Benutzer kann sich alle Posts seiner abonnierten Kontakte anzeigen lassen: • Name des Autors • Datum • Post • NEW Marker • Vergangene Zeit seit dem Posten
FA#2.3	Wisch Geste für News	Der Benutzer kann zwischen den zwei Newsansichten per Wischgeste hin- und her wechseln.

FA#2.4	Nachladen älterer Einträge	Der Benutzer kann durch Scrollen ältere Einträge nachladen lassen. (immer 10 Stück)
FA#2.5	Bewertungssystem	Der Benutzer kann Posts anderer Benutzer bewerten.

Tabelle 3-2 Funktionale Anforderungen Newsanzeige

3.1.3 Einstellungen

Nr.	Bezeichnung	Beschreibung
FA#3.1	Anzeige eigenes Profil	Der Benutzer kann sein eigenes Profil einsehen.
FA#3.2	Account löschen	Der Benutzer kann seinen eigenen Account löschen.
FA#3.3	Logout	Der Benutzer kann sich ausloggen.

Tabelle 3-3 Funktionale Anforderungen Einstellungen

3.1.4 Suche

Nr.	Bezeichnung	Beschreibung
FA#4.1	Personensuche	Der Benutzer kann per Suchbegriff nach anderen Personen suchen.
FA#4.2	Postsuche	Der Benutzer kann per Suchbegriff einen Eintrag gezielt suchen.

Tabelle 3-4 Funktionale Anforderungen Suche

3.1.5 Verfassen eines neuen Eintrags

FA#5

Der Benutzer kann einen eigenen, neuen Eintrag verfassen. Die Länge ist beschränkt auf maximal 150 Zeichen, was dem Benutzer angezeigt werden soll.

3.1.6 Profilseite

Nr.	Bezeichnung	Beschreibung
FA#6.1	Anzeige verschiedener Details	Der Benutzer kann sich anzeigen lassen: • Follower • Following • Anzahl eigener Posts
FA#6.2	Eigene Posts löschen	Der Benutzer kann eigene Posts löschen.
FA#6.3	Anderen Benutzern folgen	Der Benutzer kann per Button anderen Personen folgen oder entfolgen.

Tabelle 3-5 Funktionale Anforderungen Profilseite

3.1.7 Hinweis auf nicht vorhandene Netzwerkverbindung

FA#7

Bei jeder Seitenanfrage bzw. einer Interaktion durch den Nutzer wird abgefragt, ob die Netzwerkverbindung vorhanden ist. Ist dies nicht der Fall, soll dies dem Benutzer angezeigt werden und die Möglichkeit zum sofortigen Reload bieten. Ist die Verbindung wieder vorhanden, wird der Inhalt geladen.

3.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Anforderungen an die Qualitätseigenschaften der Anwendung. Hierbei handelt es sich oft um Funktionen, die dem Benutzer nur im Falle einer Fehlfunktion negativ auffallen.

NFA#1: Benutzbarkeit, Usability, Aussehen

Das Aussehen der Anwendung soll möglichst klar und intuitiv für den Benutzer sein. Im Idealfall sollte sich der Benutzer bereits nach wenigen Augenblicken zurechtfinden, navigieren und die Funktionen der App anwenden können.
Neue, eventuell dem Nutzer unbekannte Funktionen sollen möglichst schnell und einfach zu erlernen sein. Die Anwendung sollte die üblichen, bekannten Benutzungsmechanismen enthalten, die der Benutzer eines mobilen Gerätes gewohnt ist.
Die dargestellten Daten bzw. Informationen sollten so strukturiert und abgebildet werden, dass es trotz verhältnismäßig kleinem Display kein Problem darstellt, sie zu lesen. Die Bedienbarkeit muss möglichst einfach und effizient sein, jede Funktion sollte durch wenige Klicks erreichbar sein. Die Konsistenz der Optik, Funktionen und Sprache muss in der gesamten Anwendung gewährleistet sein, damit sich der Benutzer an eine einheitliche Bedienung gewöhnt.

NFA#2: Performance, Leistung, Latenz

Die Anwendung soll möglichst performant auf jedem System laufen. Jede Aktion des Benutzers muss mit geringstmöglicher Verzögerung beantwortet werden.
Dies gilt insbesondere für externe Anfragen an einen Server. Da dies unter Umständen – abhängig vom verfügbaren Netz – einige wenige Sekunden dauern kann, muss dies dem Benutzer dargestellt werden. Des Weiteren sollte die Menge an extern geladenen Daten sowie die Anzahl der Anfragen auf ein Minimum begrenzt werden, um den Datentransfer und somit auch die Wartezeit so gering wie möglich zu halten.

NFA#3: Zuverlässigkeit, Verfügbarkeit, Reife, Fehlerverhalten

Die Anwendung sollte zuverlässig sein, das heißt sowohl vom Benutzer ausgeführte Aktionen als auch im Hintergrund laufende Aktionen sollen erwartungsgemäß und korrekt ausgeführt werden. Zudem sollte die Anwendung möglichst ausgereift sein und keine unbehandelten Fehler auslösen, da dies oftmals schnell die Abneigung des Benutzers gegenüber der Anwendung zur Folge hat.
Mögliche, auftretende Fehler sollten erkannt und behandelt werden bzw. die Anwendung sollte in einen konkreten Zustand gebracht werden, sodass die Benutzung weiterhin möglich ist. In diesem Falle kann der Nutzer eine Rückmeldung erwarten und wird somit nicht im Ungewissen gelassen. Keinesfalls sollte die Anwendung unerwartet abstürzen und den Benutzer dadurch unzufrieden stimmen. Die Anwendung sollte in einer dem Benutzer zumutbaren Zeit geladen werden können.

NFA#4: Wartbarkeit

Aus Entwicklersicht sollte die Architektur der Anwendung möglichst strukturiert und modular aufgebaut sein, um eine Weiterentwicklung oder Wartung des bestehenden Systems zu erlauben. Durch die Verwendung ausgereifter Entwurfsmuster soll dies ermöglicht werden. Neue oder bestehende Funktionen zu warten sollte ohne unnötig großen Aufwand erfolgen können.

NFA#5: Portierbarkeit

Die Anwendung soll ohne große Änderungen auf mehreren Plattformen einsetzbar sein. Aufgrund der Verwendung des Frameworks Phonegap ist dies gewährleistet.

4 Konzeption

Dieses Kapitel beschreibt den Styleguide, also die einheitliche Verwendung von Logo, Farben und Typografie. Außerdem wird der grundsätzliche Aufbau der zu entwickelnden mobilen Anwendung mithilfe von Mockups dargestellt.
Im Anschluss daran wird die im Backend liegende Datenbank-Architektur erläutert.

> *„Gutes Design ist so wenig Design wie möglich:*
> *Weniger Design ist mehr, konzentriert es sich doch auf das Wesentliche, statt die Produkte mit Überflüssigem zu befrachten. Zurück zum Puren, zum Einfachen!“*
>
> - Dieter Rams (vgl. 10 Thesen von Dieter Rams über gutes Produktdesign [19])

4.1 Styleguide

Um der Anwendung ein einheitliches, rundes Design zu geben ist es notwendig, feste Layoutvorgaben zu definieren. Diese sorgen für eine schnelle, einfach zu erlernende Bedienung durch den Benutzer, da viele optisch identische Elemente auch die gleiche Funktion erfüllen. Dieses Layout wird konsequent auf alle Seiten angewendet, um die App ausgereift und professionell darzustellen.

4.2 Farben

Um die Anwendung nicht zu bunt und dadurch möglicherweise zu unübersichtlich zu gestalten, wird auf die Verwendung mehrerer Farben verzichtet, es wird lediglich eine Basis-Farbe verwendet, welche jedoch in verschiedenen Intensitätsstufen zum Einsatz kommt.

Die Gestaltung von Buttons und Text wird bewusst schlicht gehalten, damit die Anwendung nicht überladen wirkt.

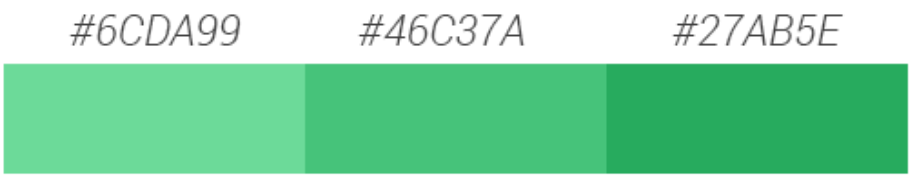

Abbildung 4-1 HEX Code der Konzeptionsfarben
Quelle: Eigene Abbildung

4.3 Typografie

Auch bei der Verwendung der Schriftart sowie Schriftgröße wird darauf geachtet, ein einheitliches Erscheinungsbild zu erhalten. Deshalb wird eine Hauptschriftart verwendet, allerdings in mehreren Schriftdicken sowie Schriftgrößen. Hierbei handelt es sich um die Schriftart „Roboto“. Diese bietet in allen Schriftgrößen bzw. Schriftdicken eine gute Lesbarkeit sowie ein modernes Aussehen.

Abbildung 4-2 : Ausgewählte Schriftarten in verschiedenen Größen und Stärken
Quelle: Eigene Abbildung

4.3.1 Icons

Für bestimmte Darstellungen werden Icons statt Texte verwendet, beispielsweise in der Navigationsleiste oder für Input-Felder die der Anmeldung oder Registrierung dienen. Hierbei handelt es sich um gängige, bei mobilen Anwendungen bekannte Icons. Diese sind so gewählt, dass der Benutzer die zugrunde liegende Funktion intuitiv erkennt und sich somit schnell zurechtfindet. Verwendet werden allerdinge keine Grafiken, sondern moderne sogenannte Icon Fonts, welche ressourcenschonend geladen werden können (vgl. Ionicons [11]).

Abbildung 4-3 Iconpack
Quelle: Eigene Abbildung

4.4 UI Konzept

In diesem Kapitel werden die erstellten Mockups, also die einzelnen Benutzeroberflächen der verschiedenen Seiten, dargestellt und erläutert. Diese sollen einheitlich gestaltet werden, wesentlich ist hierbei die Einhaltung des definierten Styleguides.

4.4.1 Grundlayout

Das Grundlayout der App teilt sich auf in zwei Bereiche, den Header und den Content-Bereich.

- Im Header befindet sich die Navigationsleiste, um zwischen den einzelnen Seiten zu wechseln

- Der Content Bereich enthält den eigentlichen Inhalt, der je nach gewähltem Navigationspunkt unterschiedlich befüllt wird

Der Benutzer soll in der Navigationsleiste zudem die Möglichkeit erhalten zur jeweiligen Überseite zurück zu navigieren, falls er sich auf einer Unterseite befindet.

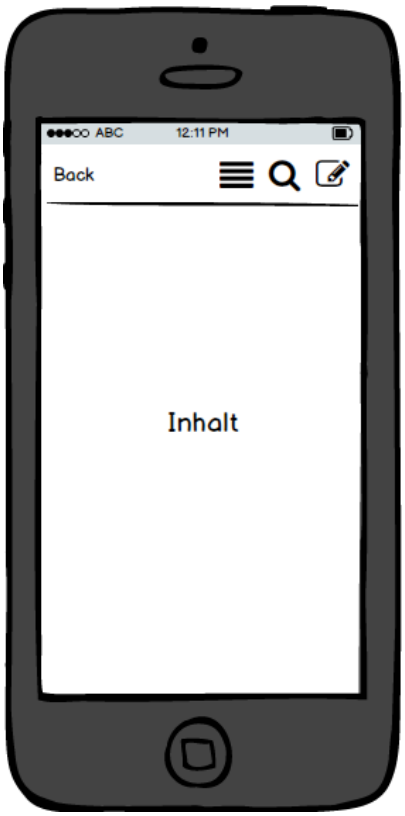

Abbildung 4-4 Mockup des Grundlayouts
Quelle: Eigene Abbildung

4.4.2 Mockups der Views

4.4.2.1 Login / Register View

Die View für den Login bzw. Registrierungsvorgang weicht vom Grundlayout ab, da der Benutzer zu diesem Zeitpunkt noch nicht authentifiziert ist.
Es soll stattdessen ein Formular zur Eingabe der Benutzerdaten im Fullscreen-Format darstellen.

Abbildung 4-5 Mockup für Anmeldung
Quelle: Eigene Abbildung

Abbildung 4-6 Mockup für Registrierung
Quelle: Eigene Abbildung

4.4.2.2 Eigene News / Latest News

Der Startbildschirm repräsentiert nach erfolgreichem Einloggen des Benutzers den personalisierten, eigenen Newsstream.
Falls dieser noch keine News enthält – da der Benutzer noch niemandem folgt – wird eine Alternativdarstellung gewählt. Diese fordert den Benutzer dazu auf, nach Freunden zu suchen oder die gesamten News durchzusehen.

Per Swipe (kurze Erklärung, quelle, link?) Geste lässt sich zwischen der erwähnten View und der Latest News View hin und her wischen.
Die Latest News View zeigt alle News an, geordnet nach dem Zeitpunkt der Erstellung in absteigender Reihenfolge.

Abbildung 4-8 Mockup für die Anzeige der personalisierten News
Quelle: Eigene Abbildung

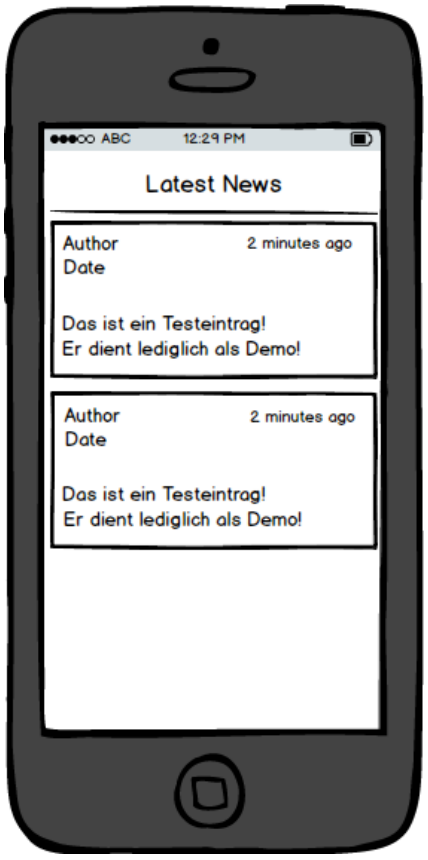

Abbildung 4-7 Mockup für die Anzeige aller News
Quelle: Eigene Abbildung

4.4.2.3 Einstellungen

In den Einstellungen hat der Benutzer Zugriff auf verschiedene personalisierte Funktionen wie z.B. das eigene Profil einzusehen oder den Account zu löschen.

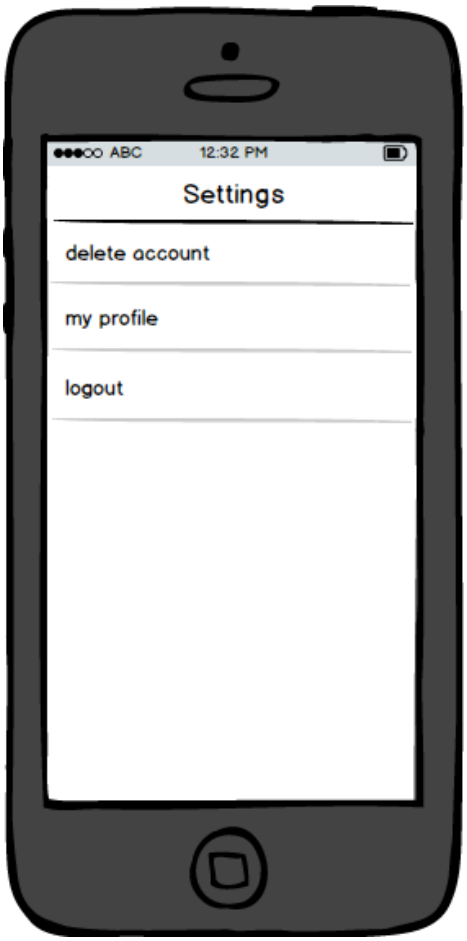

Abbildung 4-9 Mockup für das Settingsmenü
Quelle: Eigene Abbildung

4.4.2.4 Suche

Die Suche bietet die Möglichkeit, gezielt Benutzer oder einzelne Posts zu suchen. Erreicht wird dies über ein Suchfeld, per Dropdown-Menü wird eine Suchoption gewählt.

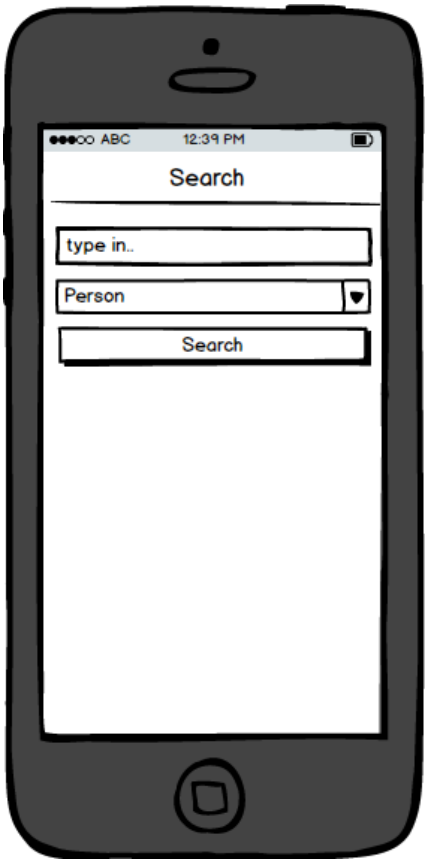

Abbildung 4-11 Mockup für die Suche
Quelle: Eigene Abbildung

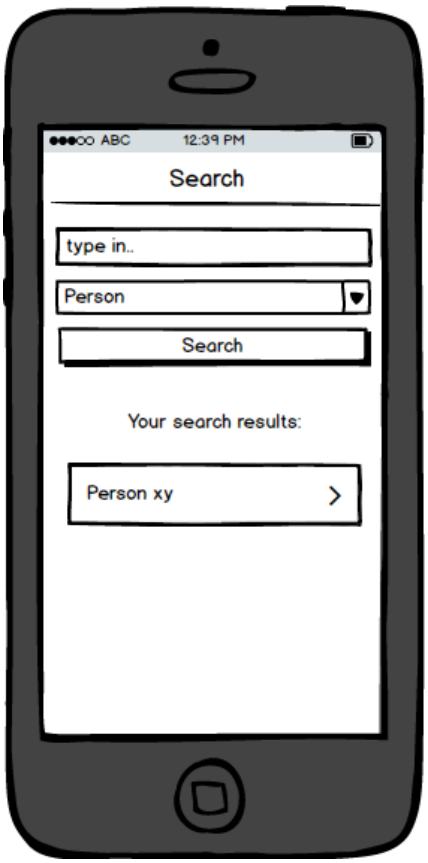

Abbildung 4-10 Mockup für gefundene Suchergebnisse
Quelle: Eigene Abbildung

4.4.2.5 Verfassen eines neuen Posts

Die View zum Verfassen eines neuen Posts enthält ein Textfeld, welches bei der Texteingabe automatisch mitwächst, um dem Benutzer seinen eingetippten Text komplett anzuzeigen. Zusätzlich soll die maximale Zeichenlänge bzw. die restlichen verfügbaren Zeichen angezeigt werden.

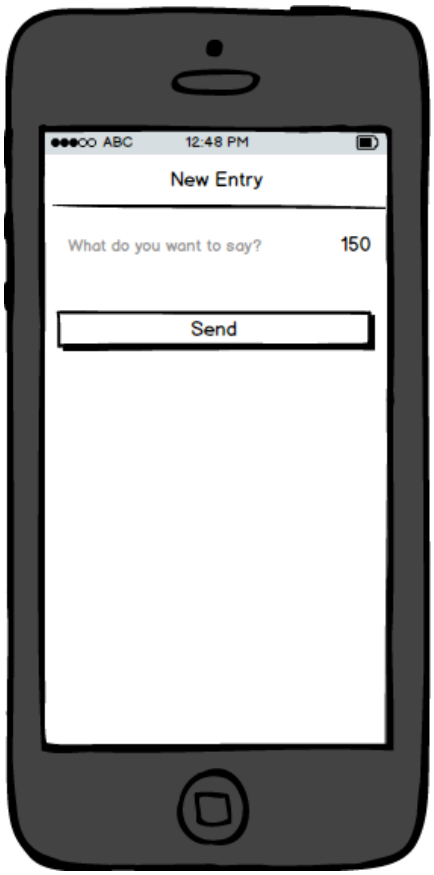

Abbildung 4-13 Mockup für leeres Textfeld
Quelle: Eigene Abbildung

Abbildung 4-12 Mockup für Textfeld mit Inhalt
Quelle: Eigene Abbildung

4.4.2.6 Profilseite

Die Profilseite zeigt eine Übersicht über:

- Follower (Personen, die einem selbst folgen)
- Following (Personen, denen man folgt)
- geschriebene Posts

Zudem besteht die Möglichkeit, dem angezeigten Benutzer per Klick auf einen Button zu folgen und somit seine zukünftigen Posts lesen zu können.

Abbildung 4-14 Mockup für Profilseite
Quelle: Eigene Abbildung

4.4.2.7 Hinweis auf nicht vorhandene Netzwerkverbindung

Falls keine Netzwerkverbindung besteht um Daten nachzuladen, soll dies dem Benutzer signalisiert werden. Ermöglicht wird dies auf zweierlei Arten:

Abbildung 4.15.) Fullscreen Reload Meldung:

News zeigt einen Button zum erneuten Laden im Fullscreen Modus an

Abbildung 4.16.) Einblendung einer Informationsleiste als Hinweis

Abbildung 4-15 Mockup für No Connection Fullscreen Meldung
Quelle: Eigene Abbildung

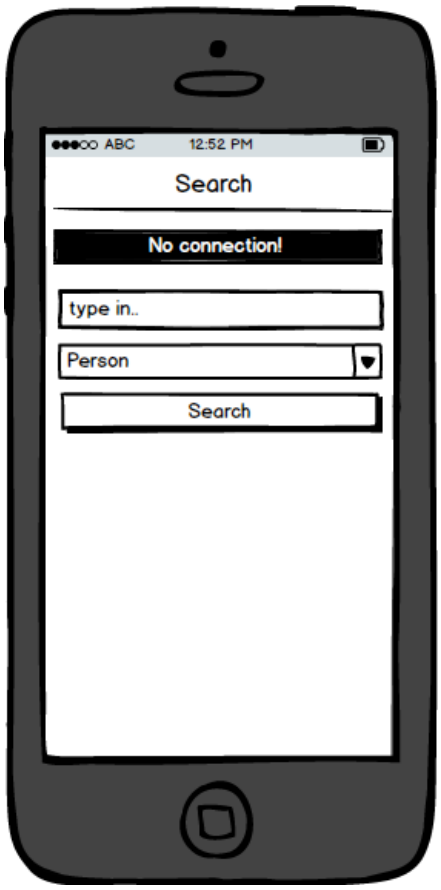

Abbildung 4-16 Mockup für No connection Leiste
Quelle: Eigene Abbildung

5 Realisierung

Das Kapitel der Realisierung beschreibt die praktische Umsetzung der mobilen Anwendung anhand der ermittelten Anforderungen unter Berücksichtigung des Konzeptes. Zunächst werden noch einmal die Grundlagen aufgegriffen, also die verwendeten Technologien im Front- und Backend-Bereich sowie verschiedene Aspekte zur Entwicklungsumgebung. Anschließend werden ausgewählte Funktionen der Anwendung aufgezeigt und deren genauer Ablauf sowie Implementierung auszugsweise dargestellt. Schließlich wird die Portierung der WebApp zu einer hybriden App mittels Phonegap geschildert.

5.1 Grundlagen zur Implementierung

5.1.1 Technologieauswahl

Folgende Technologien wurden in Kapitel zwei bereits erläutert und finden für die Entwicklung der App Anwendung.

5.1.1.1 Frontend

Bei der Implementierung der Frontend-Seite finden HTML5, CSS3 und JavaScript Verwendung. HTML5 bildet die Grundstruktur und den Inhalt, CSS3 wird für die optische Darstellung eingesetzt und mit JavaScript wird die Programmlogik, also das Verhalten der App umgesetzt. Statt nativem JavaScript findet das Framework AngularJS Anwendung, welches vor allem im Bereich der Webapplikationen vielen Funktionen vereinfacht.

5.1.1.2 Backend

Im Backend kommt die Skriptsprache PHP in Zusammenspiel mit der Datenbanksprache SQL zum Einsatz. Die einzelnen PHP-Skripte werden je nach Bedarf aufgerufen, stellen die entsprechenden Datenbankabfragen und geben einen JSON Datensatz zurück, der im Frontend dargestellt wird.

5.1.2 Versionskontrolle

Um Änderungen am Projekt zu protokollieren und einen zentralen Zugriffspunkt hierfür zu schaffen, wird ein Versionsverwaltungssystem genutzt. In diesem Falle handelt es sich um Git in Verbund mit Bitbucket (vgl. Bitbucket [4],[5]), einem webbasierten Filehosting-Dienst für Softwareprojekte. Dies ermöglicht einen genauen Überblick über den Fortschritt des Projekts, da die einzelnen Commits protokolliert werden. Es ist zudem möglich, mehrere verschiedene Stände des Projekts parallel, aber unabhängig voneinander zu entwickeln. Auch das Laden einer älteren Version im Nachhinein ist möglich.

5.2 Verschiedene Aspekte der technischen Umsetzung

In diesem Kapitel werden die wesentlichen Bestandteile der App aus technischer Sicht erläutert, um deren Funktionsweise darzulegen.
Die Beschreibungen der Hauptfunktionen zeigen, wie die App letztlich funktioniert.

5.2.1 Allgemeiner Aufbau

Bei der zentralen Datei des Projekts handelt es sich um die index.html, in dieser wird das komplette Grundgerüst der Anwendung definiert.
Dort werden auch sämtliche notwendige Dateien für die Darstellung und Programmlogik mit eingebunden.

Alle Dateien, die zur Steuerung der Programmlogik benötigt werden, also AngularJS Controller, Services und Directiven, befinden sich im Verzeichnis js.
Diese wiederum beziehen sich auf die Libraries, in denen Angular Module gelagert sind. Hier befinden sich Module wie

- angular.min.js: minimales AngularJS Paket, notwendig für die Verwendung von AngularJS
- angular-animate.js: Modul zum Animieren verschiedener Elemente, z.B. Ein- und Ausblendungen
- angular-touch.js: notwendig für vordefinierte Touch-Gesten, wie die Wisch-Geste
- angular-route.js: Modul zum dynamischen Laden des Inhalts mittels Navigation

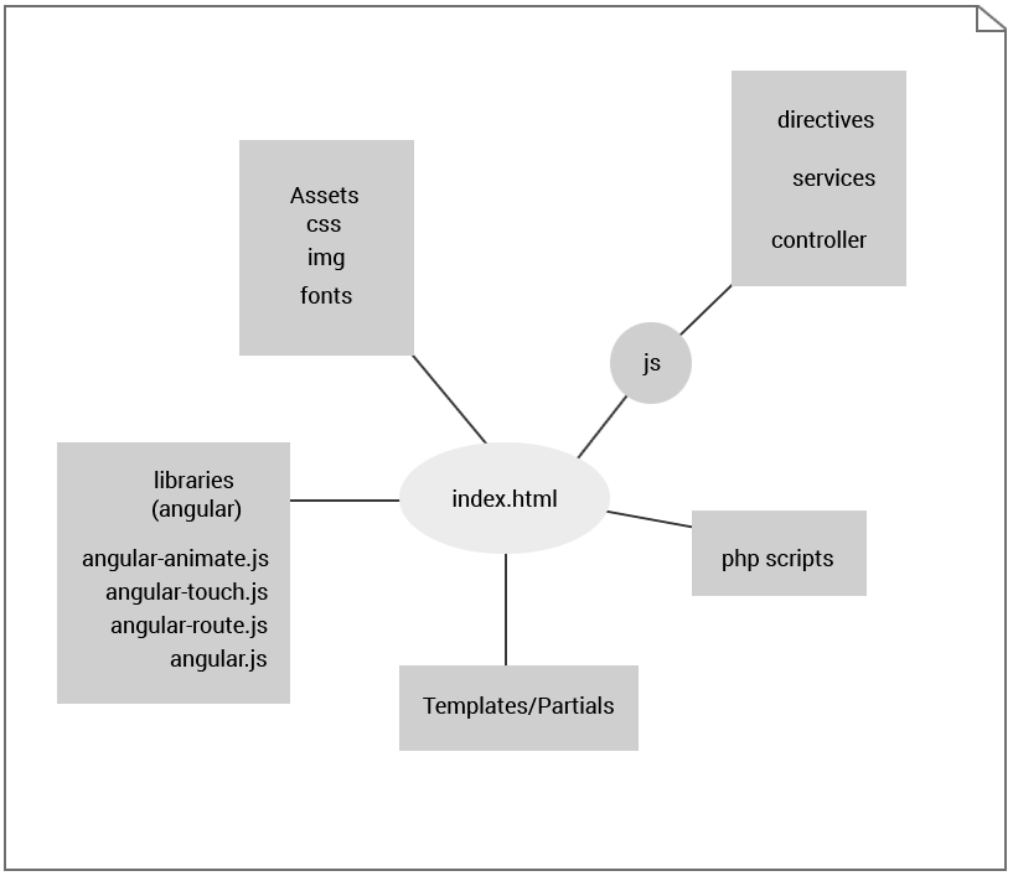

Abbildung 5-1 Grundstruktur des Projekts
Quelle: Eigene Abbildung

In der index.html wird die AngularJS Directive ng-view verbaut, hier handelt es sich um eine Art Platzhalter für den Inhalt, der über die Navigation dynamisch geladen wird. Der Inhalt für diese Directive wird im Template Verzeichnis ausgelagert.

```
<!doctype html>
<html>
<head>
<title></title>
</head>
        <body ng-app="myApp">
                <div ng-view>
                </div>
        </body>
</html>
```

Listing 5-1 Allgemeiner Aufbau AngularJS Directive

5.2.2 Initialisierung und Ablauf

Der Programmablaufplan aus Abbildung 5-2 erläutert die verschiedenen Abzweigungen bzw. den Funktionsablauf innerhalb der Anwendung, beginnend vom Start der App.

Die Authentifizierung bildet den Startpunkt der Anwendung. Hier wird geprüft, ob der Benutzer über eine gültige Session-ID verfügt. Ist das nicht der Fall, wird er zum Login- Screen weitergeleitet. An dieser Stelle kann der Benutzer entscheiden, ob er sich einloggen oder neu registrieren will. Nach Eingabe der Login-Daten wird überprüft, ob der User vorhanden ist. Falls dieser noch nicht vorhanden ist, folgt eine Weiterleitung zum Register-Screen. Hier muss der Benutzer einen Nutzernamen sowie ein Passwort wählen, anschließend wird er im Falle einer erfolgreichen Registrierung zum Login-Screen zurückgeleitet, um sich erstmals mit der richtigen Benutzername/Passwort- Kombination anzumelden. Mit der erfolgreichen Anmeldung wird eine Session-ID zugewiesen, die den Nutzer eindeutig identifiziert.
Ist die Abfrage nach einer Session-ID anfangs erfolgreich, wird der Benutzer direkt

zum Home Screen weitergeleitet. Dort besteht Zugriff auf alle Funktionen der Anwendung.

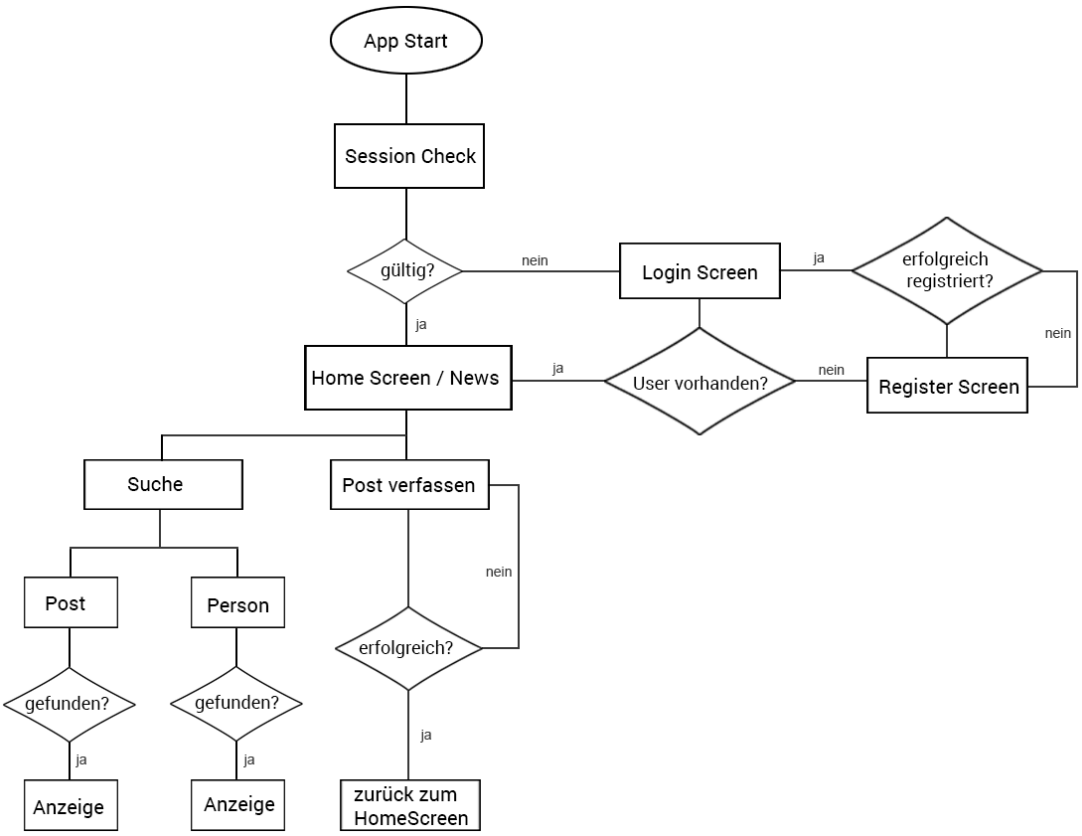

Abbildung 5-2 Ablaufschema der App
Quelle: Eigene Abbildung

Nach erfolgreicher Anmeldung besteht Zugriff auf die Anzeige der News sowie auf die Einstellungen und die Suche nach Post oder Person sowie die Möglichkeit, einen eigenen Eintrag zu verfassen. Bei der Eingabe eines Suchbegriffs wird eine Suchanfrage gestartet. Wird ein passender Begriff oder eine Person gefunden, wird der Post oder das Profil der entsprechenden Person angezeigt.
Nach der Eingabe eines eigenen Posts bzw. des Absendens mittels Bestätigungsbutton erfolgt eine Weiterleitung zurück zum Home Screen. Hier wird der neue Eintrag unmittelbar angezeigt und steht auch anderen Benutzern zur Verfügung.

5.2.3 Kommunikation zwischen App und Server

Die Kommunikation zwischen der App und der auf dem Server befindlichen Datenbank wird realisiert über eine REST Schnittstelle, welche die angeforderten Daten im JSON Format bereitstellt. Die Implementierung einer REST Schnittstelle ermöglicht die Verwendung verschiedener Datenbanktypen, d.h. falls sich der Datenbanktyp ändert, muss lediglich der Zugriff angepasst werden.

Ein Zugriff auf Daten aus der Datenbank läuft wie folgt ab:
Im Frontend wird aus AngularJS eine HTTP POST Anfrage initiiert, optional auch mit verschiedenen Parametern zur Spezifizierung der Datenbankanfrage versehen. Diese Parameter werden von dem aufgerufenen PHP-Script entgegengenommen, für jede Anfrage steht eine explizite PHP-Datei zur Verfügung. Das Script prüft die Variablen und startet anschließend einen Query an die SQL Datenbank, welche den gewünschten Datensatz zurückliefert. Dieser wird daraufhin in ein gültiges JSON Objekt umgewandelt und an das Frontend zur Weiterverarbeitung zurückgegeben.

```
<?php

$postdata = file_get_contents("php://input");
$myData = json_decode($postdata);
$userName = $myData->userName;

$json_data = array();

$sql = "DELETE FROM userTable WHERE userName = :userName";
$row = $db->prepare($sql);
$row->execute(array('userName'=>$userName));

        if($row){
                $json_array["deleteUserSuccess"] = true;
        }
        else{
                $json_array["deleteUserSuccess"] = false;
        }

        array_push($json_data,$json_array);
```

```
return json_encode($json_data);

?>
```

Listing 5-2 Kommunikation zwischen Client und Server

In Listing 5-2 ist das Löschen eines Users aus der User Tabelle dargestellt. Das Skript bekommt den zu löschenden User als Parameter übergeben, welcher im JSON Format vorliegt. Dieser wird per json_decode in ein Objekt umgewandelt. Anschließend wird die Datenbankabfrage in Form eines PDO Statements ausgeführt. Je nachdem welche Bedingung erfüllt ist, wird das JSON Array unterschiedlich befüllt, in ein gültiges Objekt umgewandelt und an die Clientseite zurückgegeben.

5.2.4 Ajax Funktion am Beispiel der Überprüfung des Nutzernamens

Für die Datenübertragung zwischen Client und Server wird „Asynchronous JavaScript and XML“ verwendet. Diese Technik ermöglicht das asynchrone Übermitteln von Daten. Der Vorteil liegt darin, dass eine Anfrage an den Server nicht auf dessen Antwort warten muss, vorhandener Programmcode kann weiterhin ausgeführt werden. Trifft eine Antwort vom Server ein, wird diese weiterverarbeitet.

In diesem Beispiel wird eine häufig verwendete Art des Ajax Requests erläutert, wie man sie typischerweise beim Registrierungsvorgang einer App findet. Der Registrierungsvorgang erfordert die Eingabe des Benutzernamens sowie des Passworts.
Nach Eingabe des Benutzernamens wird ein Ajax Request gestartet, noch während der Benutzer mit der Eingabe des Passworts beschäftigt ist. In dieser Zeit wird mittels einer Datenbankabfrage geprüft, ob der Username bereits existiert.
Abhängig vom Rückgabewert wird anschließend die Verfügbarkeit des Benutzernamens auf der Clientseite visuell dargestellt. Der Vorteil besteht hier in einer klaren Usability-Verbesserung, der Benutzer wird nicht erst nach Absenden seiner Daten auf die Verfügbarkeit des Namens hingewiesen, sondern bereits zuvor.

```
app.directive("checkUserDirective", function($q,RegisterService) {
	return {
		link: function (scope,element,attrs) {
		 element.bind('blur', function (e) {
		var keyProperty = scope.$eval(attrs.checkUserDirective);
		var currentValue = element.val();
		RegisterService.checkUserRegistered(keyProperty.property, currentValue)
				.then(function (unique) {
					if(unique.registerSuccessCheck == true){
						scope.showCheckingSuccess = false;
					}
					else if(unique.registerSuccessCheck == false){
						scope.showCheckingSuccess = true;
					}
		});
});
```

Listing 5-3 checkUserDirective

Das Listing 5-3 zeigt die AngularJS Directive, die „on blur“ ausgelöst wird, d.h. die Directive wird dann ausgelöst, wenn der Benutzer das Input-Feld für den Nutzernamen verlässt. Daraufhin wird die Methode checkUserRegistered des RegisterService aufgerufen. Abhängig von Rückgabewert dieser Methode wird die Variable showCheckingSuccess gesetzt und zeigt in der View an, ob der Nutzername vergeben oder frei ist.

```
checkUserRegistered:function(id, property, value,$scope){

$request = $http.post('php/register_user_check.php',data);

$request.success(function (data, status, headers, config){
		return status;
});
$request.error(function (data, status, headers, config){
		return status;
});
```

Listing 5-4 RegisterService

Der RegisterService in Listing 5-4 startet eine http-Anfrage an den Server bzw. das PHP-Skript register_user_check.php, welches die Verfügbarkeit des Namens in der Datenbank prüft.
Dieser Request wird überwacht von Angular Promise, welcher auf success oder error reagiert und eine entsprechende Statusmeldung an die Directive zurückliefert.

5.2.5 Single-Page-Application Navigation mittels Routing

Eine Single-Page-Application muss lediglich einmal auf der Client-Seite geladen werden, der unterschiedliche Inhalt wird anschließend dynamisch per Ajax nachgeladen. Die App besteht aus einer festen Navigationsleiste, der darunter angeordnete Inhalt wird durch Wählen des jeweiligen Navigationspunktes geladen. Dieser Wechsel der einzelnen Seiten erfolgt über den Route Provider Service von AngularJS (vgl. Chris Sevilleja [14]).

```
<ul>
        <li><a href="#/settings"><i class="ion-navicon-settings"></i></a></li>
        <li><a href="#/search"><i class="ion-navicon"></i></a></li>
</ul>
```

Listing 5-5 Liste für Navigation inkl. Pfaden

Im Listing 5-5 ist die Navigationsstruktur im index.html-Dokument zu sehen. Die jeweilige URL für die zu ladende Seite wird im href-Attribut des Links definiert.

```
app.config(['$routeProvider',
   function($routeProvider) {
      $routeProvider
         .when('/settings', {
            templateUrl: 'partials/settings.html',
            controller: 'settingsCtrl'
         })
```

```
        .when('/search', {
        templateUrl: 'partials/search.html',
        controller: 'searchCtrl'
        })
        .otherwise({
        redirectTo: '/login'
        });
}]);
```

Listing 5-6 Konfiguration des AngularJS RouteProviders

Das Listing 5-6 zeigt die Konfiguration des Route Providers. Hierbei wird jede mögliche Route mit einem „when“ Parameter abgefragt, die URL wird bei jedem Schritt verglichen. Sobald eine dieser Bedingungen zutrifft, wird das jeweilige verknüpfte Template der View zugeordnet, inklusive der Angabe des Controllers.
Trifft keine der Bedingungen zu, erfolgt die Weiterleitung mittels otherwise zu einer default Route.

```
app.run(function($rootScope, $location, loginService){
        var routespermission=['/write_entry','/search','/login];
        $rootScope.$on('$routeChangeStart', function(){
        if( routespermission.indexOf($location.path()) !=-1){
                var connected=loginService.islogged();
                connected.then(function(msg){
                        if(!msg.data){
                                $location.path('/login');
                        }
                });
        }
        });
});
```

Listing 5-7 Definition der erlaubten Routen

Listing 5-7 stellt die Abfrage der autorisierten Routen dar.
In der Variable routespermission werden die verfügbaren Routen gespeichert.
Will der Benutzer per Klick in der Navigation eine andere Seite wählen, wird zuerst

geprüft, ob die gewählte Route vorhanden ist. Daraufhin wird über den LoginService eine Anfrage an den Server geschickt, welcher prüft, ob die Session des Benutzers noch gültig ist. Ist dies der Fall, wird er zur gewünschten Route weitergeleitet. Falls die Session-ID abgelaufen ist, folgt die Weiterleitung zum Login-Bereich.

5.2.6 Nachladen des Inhalts am Beispiel von Infinite Scrolling

Bei Infinite Scrolling handelt es sich um eine Technik, die den Inhalt Stück für Stück nachlädt. Dies geschieht entweder automatisch per Scrolling oder per Buttonklick. Dem Benutzer wird beim Laden der App bzw. der Newsansicht nur eine bestimmte Anzahl an Artikeln/Inhalt angezeigt. Um weiteren Inhalt nachzuladen, muss ans Seitenende gescrollt werden.

```
scroll.directive('whenScrolledNews', function ($document,$timeout) {
        return {
        link: function (scope, element, attrs) {
                $(document).scroll(function() {
                if($(window).scrollTop() + $(window).height() == $(docu-
ment).height()) {
                        scope.$apply(attrs.whenScrolledNews);
                }
        });
};
```

Listing 5-8 whenScrollnedNews Directive für Infinite Scrolling

Zu Anfang werden lediglich 10 Einträge geladen, welche in einem Array gespeichert und angezeigt werden. Bei jedem Scrollvorgang wird eine Directive, wie sie in Listing 5-8 zu sehen ist, aufgerufen. Diese überprüft via jQuery, ob der Benutzer das Ende der Seite erreicht hat. Ist das der Fall, wird die Methode whenScrolledNews aktiviert, welche einen http-Request zur Datenbank via Ajax startet. Hierbei wird auch der timestamp des letzten Eintrags als Parameter mit übergeben. Mithilfe dieses Parameters kann der SQL Query genauer spezifiziert werden. In Listing 5-9 wird die Anzahl der Einträge mittels LIMIT auf eine feste Anzahl beschränkt.

```
<?php
    …
        $sql = "SELECT * FROM news WHERE timestamp < '$query_time' OR-
DER BY        DESC LIMIT 10";
        $row = $db->prepare($sql);
        $row->execute();
    …
?>
```

Listing 5-9 DB Limit Query für Infinite Scrolling

Die gefundenen Datensätze werden anschließend wieder an die Client Seite übergeben und in das Array, welches bereits die ersten 10 Einträge enthält, gepusht.

5.2.7 Authentifizierung des Benutzers mittels PHP Session

Die Authentifizierung des Benutzers ist erforderlich, da der Benutzer personalisierte Daten aus der Datenbank benötigt. Realisiert wird diese Authentifizierung beim Server mittels PHP-Sessions (vgl. Jason Watmore [15]). Erfolgt die Anmeldung eines Benutzers mit den richtigen Zugangsdaten, wird für diesen eine Session auf dem Server gestartet bzw. eine Session-ID an den Benutzer zurückgeliefert.

5.2.7.1 Verwendung des HTML5 SessionStorage

Diese Session-ID wird dann clientseitig im WebStorage, genauer dem SessionStorage, gespeichert, und zwar in Form eines Schlüssel-Wert-Paares.
Zugehörige Methoden sind:

- sessionService.set: Speichern des Schlüssel-Wert-Paares
- sessionService.get: Abrufen des gespeicherten Schlüssel-Wert-Paares

Abgerufen wird dieses über eine entsprechende get Methode. Der Gültigkeitsbereich bezieht sich hierbei lediglich auf die geöffnete Browserinstanz, andere Tabs oder Fenster haben keinen Zugriff auf den SessionStorage. Die gespeicherten Daten bleiben nur solange erhalten, bis das aktive Fenster geschlossen wird. Ein Neustart des Browsers ist gleichzusetzen mit dem Neustart der App, ein erneutes Anfordern einer Session-ID ist die Folge.

5.2.7.2 Ablauf des Login/Register Vorgangs

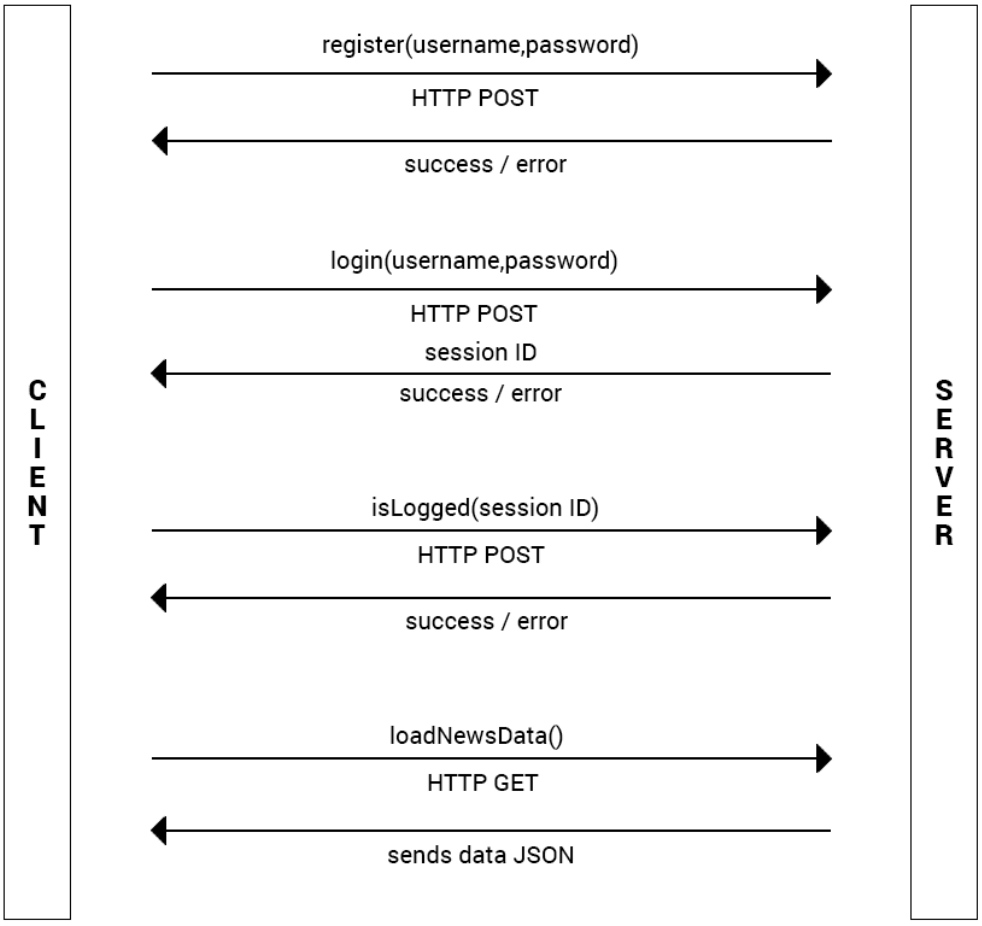

Abbildung 5-3 Ablaufschema des Anmeldung/Registrierung Vorgangs
Quelle: Eigene Abbildung

In Abbildung 5-4 ist der Ablauf des Anmeldungs- und Registrierungsvorgangs zwischen Client und Server zu sehen. Die register() Methode sendet die Registrierungsdaten per http post an den Server, dieser prüft ob der Benutzer vorhanden ist und gibt eine entsprechende Erfolgs-/Fehlermeldung an den Client zurück. Bei erfolgreicher Registrierung erfolgt der Login mittels login() Methode, der Server startet daraufhin eine Session und gibt die zugehörige Session-ID zurück, welche dann im SessionStorage gespeichert wird. Erfolgt beispielsweise ein Wechsel des Inhalts per Navigation, überprüft die Methode isLogged() zuerst beim Server, ob es sich bei der ID um eine noch gültige Session handelt. Beantwortet wird diese Anfrage wiederum mit einer Erfolgs-/Fehlermeldung. Fordert der Benutzer weitere Einträge an, beispielsweise per Infinite Scrolling, wird die Methode loadNewsData() aufgerufen, der Server liefert dann Daten im JSON Format.

5.3 Implementierung der hybriden App

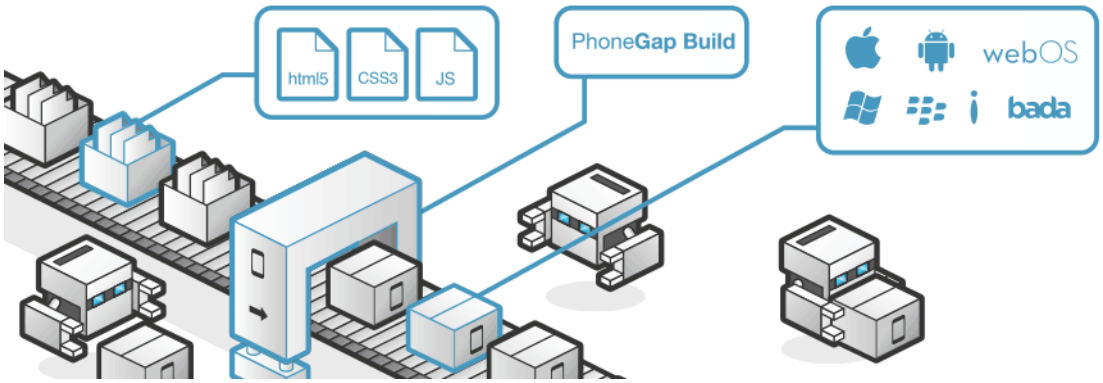

Abbildung 5-4 Phonegap Portierungsdarstellung
Quelle: Phonegap Build [8]

5.3.1 Allgemein

Phonegap ist ein Open-Source-Framework für mobile Anwendungen bzw. WebApps, mit dessen Hilfe hybride Apps erstellt werden können.
Phonegap bietet die Möglichkeit, eine WebApp als native App zur Verfügung zu stellen, indem sie als Wrapper fungiert und die WebApp in eine Art Container einbettet. Zudem besteht dadurch Zugriff auf native Plattform-Funktionen, was bei einer reinen WebApp hingegen nicht realisierbar ist. So ist es beispielsweise möglich per JavaScript Hardwarekomponenten die Kamera oder den Beschleunigungssensor anzusprechen. Die Entwicklung erfolgt mittels HTML, CSS und JavaScript oftmals in Kombination mit Frameworks. im Anschluss daran wird die WebApp mittels Phonegap portiert und steht als eigenständige, native App zur Verfügung.
Die App kann dann im jeweiligen Plattform--Container exportiert und über die üblichen Marktplätze angeboten werden, in der Art der Installation besteht dann kein Unterschied mehr zur nativen Version.

5.3.2 Funktionsweise

Die portierte WebApp ist in einer sogenannten WebView eingebettet, welche eine Schnittstelle zur Phonegap API beinhaltet (vgl. Android Developer [21]). Die WebView bietet ähnliche Funktionen wie ein Browser und bedient sich der Browserengine Webkit, um die WebApp aufzurufen. Browserspezifische Anpassungen sind nicht notwendig, der HTML, CSS und JS Code wird von der WebView interpretiert und zur Laufzeit ausgeführt.

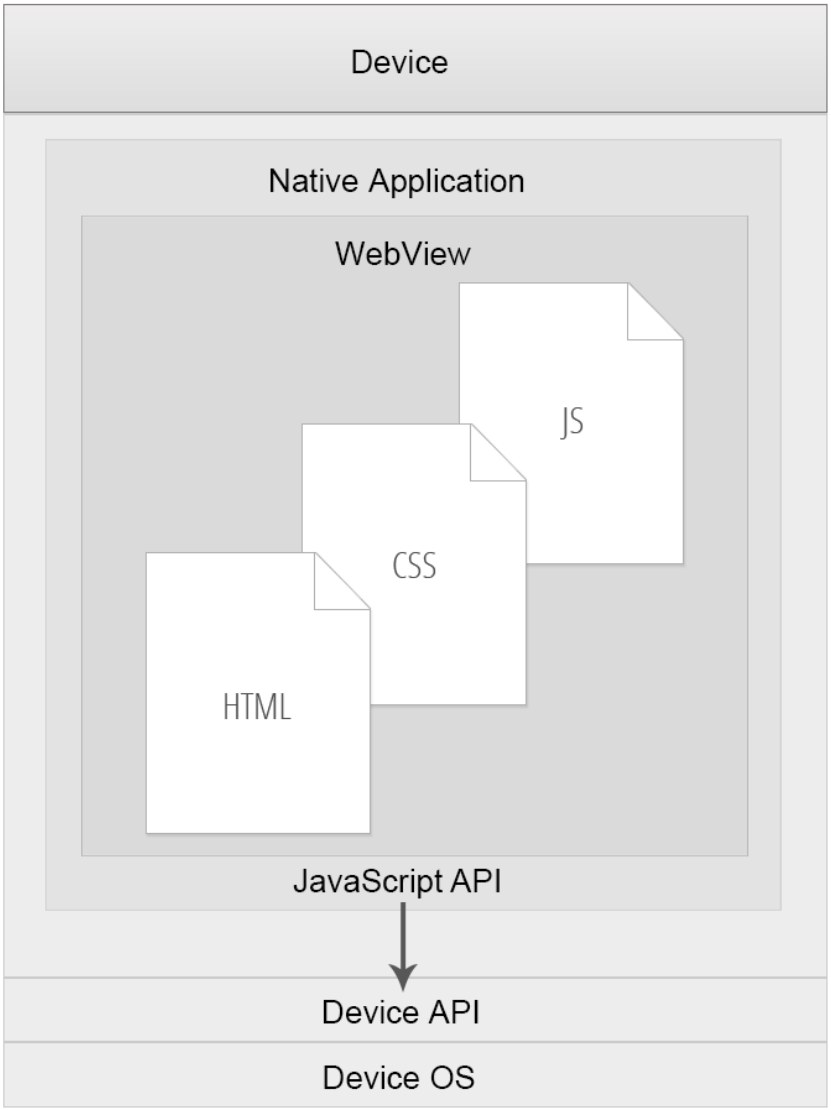

Abbildung 5-5 Phonegap Funktionsweise
Quelle: Eigene Abbildung

Um gerätespezifische Funktionen anzusprechen, bietet Phonegap Plugins, welche aus der WebApp heraus per JavaScript definiert werden können. Mit Phonegap wird die Brücke zur API der nativen Plugins geschlagen, somit ist der Zugriff auf die notwendige Geräte-Hardware möglich.

5.3.3 Integration der WebApp

Zur Portierung der WebApp in eine hybride App benötigt man zuerst das Phonegap-Framework. Nach erfolgreichem Anlegen eines neuen Projekts erhält man eine vorgefertigte Projektstruktur, folgende Ordner/Dateien sollten gesondert betrachtet werden:

- www Verzeichnis
- config.xml
- phonegap.js

Das www-Verzeichnis beinhaltet die eigentliche WebApp. Dort werden sämtliche für die WebApp notwendige Dateien eingebunden.

Die Datei config.xml, auszugsweise in Listing 5-19 dargestellt, beinhaltet wichtige Konfigurationen, die je nach Einsatzzweck der App von Bedeutung sein können. Beispielsweise werden hier die Plugins der App, die beim Start der App initial aufgerufene Datei – im Standardfall handelt es sich hierbei um index.html – und die nötigen Berechtigungen, welche die App erfordert, definiert. Bei den Berechtigungen stehen zwei Möglichkeiten zur Auswahl. Man kann entweder jede Berechtigung einzeln festlegen, oder aber darauf verzichten, was die automatische Erkennung der geforderten Berechtigungen zur Folge hat.

```
<?xml version="1.0"?>
        <content src="index.html"/>
        <preference value="none" name="permissions"/>
        <gap:plugin name="org.apache.cordova.vibration"/>
        <gap:plugin name="org.apache.cordova.camera"/>
```

```
        <gap:plugin name="org.apache.cordova.geolocation"/>
        preference value="default" name="orientation"/>
```

Listing 5-10 Konfiguration der Phonegap config.xml

Bei der phonegap.js Datei handelt es sich um die Phonegap-Bibliothek. Diese wird im Head-Bereich der index.html mit angegeben. Beim Initialisieren der App wird ein onDeviceReady() Event gefeuert, sobald Phonegap fertig geladen hat. Danach ist es möglich, mit verschiedenen Plugins auf Gerätehardware wie Kamera oder Beschleunigungssensor zuzugreifen (vgl. Peter Traeg [9]).

```
onDeviceReady: function() {
    app.run();
}
```

Listing 5-11 onDeviceReady() Funktion

6 Anforderungsabgleich

Nach der Realisierung des Prototyps soll in diesem Kapitel ein Abgleich der erreichten Funktionen mit den in Kapitel 3 definierten Anforderungen erfolgen.
Anhand der vergebenen Nummerierungen für die funktionalen und nicht funktionalen Anforderungen lässt sich diese Überprüfung leicht realisieren.
Die Anforderungen FA#1, FA#2, FA#3, FA#4, FA#5, FA#6 und FA#7 wurden alle erfüllt und werden nachfolgend einzeln betrachtet.

6.1 Funktionale Anforderungen

FA#1.1 Anmeldung

Die Funktion der Anmeldung wurde vollständig umgesetzt. Die Validierung der Login-Daten wurde mithilfe des Frameworks AngularJS realisiert. Dieses prüft die Eingaben des Users und vergleicht sie per Ajax mit den in der Datenbank gespeicherten Anmeldedaten. Bei einer erfolgreichen Anmeldung erhält der Benutzer Zugang mittels einer zugewiesenen Session-ID.

FA#1.2 Registrierung

Analog zur Anmeldung wurde auch hier die Formularvalidierung umfangreich umgesetzt. Dieses Formular verknüpft gewisse Bedingungen mit den Eingaben des Benutzers wie beispielsweise eine Mindestlänge bei der Passworteingabe oder den Vergleich beider Passwörter. Zusätzlich wurde die asynchrone Überprüfung des Nutzernamens, die noch während der Dateneingabe stattfindet, realisiert.

FA#2.1 Anzeige aller News

Wird die Anzeige aller News ausgewählt, werden die geschriebenen Einträge ungefiltert und nach Datum in absteigender Reihenfolge geordnet angezeigt. Jeder Eintrag enthält hierbei die geforderten Details wie die Anzeige des Datums oder einen Stempel, der jeden neuen Eintrag speziell kennzeichnet.

FA#2.2 Anzeige der abonnierten News

Ähnlich zu FA#2.1 werden auch hier die News angezeigt, jedoch gefiltert: Der Benutzer bekommt nur die Einträge von Personen, denen er folgt, zu sehen. Somit erhält man einen eigenen, personalisierten Newsstream.

FA#2.3 Wisch Geste für News

Die Wischgeste, welche das Hin- und Herwischen der zwei verschiedenen Newsarten auf dem Touchscreen ermöglicht, konnte mithilfe des AngularJS Moduls angular-swipe.js realisiert werden.

FA#2.4 Nachladen älterer Einträge

Die geforderte Funktion zum Nachladen älterer Einträge konnte mithilfe der Infinite Scrolling-Technik erfüllt werden. Per Scrolling zum unteren Ende der App wird immer wieder ein Datensatz mit mehreren Einträgen geladen. So lange, bis keine weiteren Einträge mehr in der Datenbank vorhanden sind.

FA#3 Einstellungen

Wählt der Benutzer das Einstellungsmenü, besteht die Wahl der Anzeige des eigenen Profils, dem Löschen des Accounts oder dem Abmelden. Beim Vorgang des Abmeldens wird die aktuelle Session aus dem sessionStorage gelöscht.

FA#4 Suche

Wählt der Benutzer die Suche aus, so kann er per Dropdown-Menü wählen, ob eine gezielte Person oder ein Eintrag gesucht werden soll. Nach erfolgreicher Suche werden die gefundenen Ergebnisse angezeigt, bei erfolgloser Suche wird der Benutzer

darüber in Kenntnis gesetzt.

FA#5 Verfassen eines neuen Posts

Die geforderten Features, also die Anzeige der verbleibenden Zeichen sowie das mitwachsende Textfeld, konnten umgesetzt werden. Dies ist von Vorteil, um dem Benutzer vor dem Absenden die komplette Nachricht anzuzeigen. Als zusätzliches Feedback für den Benutzer wird das Textfeld bei erfolgreichem Absenden gelöscht.

FA#6 Profilseite

Die geforderte Profilseite konnte realisiert werden. Dem Betrachter werden die Anzahl der Einträge und Follower/Following angezeigt. Außerdem besteht die Möglichkeit, eigene Beiträge wieder zu löschen sowie dem betrachteten Benutzer per Buttonklick zu folgen.

FA#7 Hinweis auf nicht vorhandene Netzwerkverbindung

Für den Fall einer nicht vorhandenen Netzwerkverbindung sollte eine Anzeige eingebaut werden, die den Benutzer darüber in Kenntnis setzt. Diese Anforderung wurde mithilfe eines Services erfüllt, der vor jedem Serverrequest startet und überprüft, ob eine Verbindung besteht. Wenn keine Verbindung besteht, wird dies angezeigt und eine Möglichkeit zum erneuten Laden geboten. Dadurch wird ungewünschtes Verhalten, wie beispielsweise das browserseitige Anzeigen einer 404 Fehlerseite, verhindert.

6.2 Nicht-funktionale Anforderungen

NFA#1: Benutzbarkeit, Usability, Aussehen

Zur Vereinheitlichung des Aussehens wurde in Kapitel 3 Konzeption ein Styleguide geschaffen, welcher in der Realisierung zu einer durchgehend konsistenten Optik geführt hat. So wurden beispielsweise die Icons so gewählt, dass deren Funktion intuitiv erkennbar ist. Auch die Schriftart ist trotz kleineren Displaygrößen problemlos lesbar. Zu einer gründlichen Erforschung der Benutzbarkeit wäre jedoch mindestens eine Testrunde zur Qualitätssicherung in Form eines Usability-Tests notwendig, bei der zufällig ausgewählte Benutzer die Anwendung ausgiebig testen.

NFA#2: Performance, Leistung, Latenz

Die Performance der mobilen Anwendung konnte zufriedenstellend getestet werden. Der wichtigste Performancefaktor, nämlich der Zugriff auf das Netzwerk per WLAN wie auch über das mobile Netz, erzielte zufriedenstellende, dem Benutzer gegenüber vertretbare Verzögerungen. Im Mobilfunknetz kommt es – abhängig von der verfügbaren Netzstärke – zu einer geringfügig längeren Wartezeit bei Serveranfragen als per WLAN, dies wird aber jederzeit durch einen Ladeindikator visualisiert, um den Benutzer nicht im Unklaren zu lassen. Möglich wurde dies unter anderem durch die Implementierung einer REST Schnittstelle, bei welcher die angeforderten Daten im JSON Format übertragen werden um so ein Minimum an transferierter Dateigröße zu erreichen. Auch die Verwendung des Single-Page-Application Prinzips führt zu einer Optimierung der Performance, da die App nur noch einmal geladen wird. Alle nachfolgend notwendigen Daten werden per Ajax nachgeladen, sodass keine Daten unnötigerweise doppelt übertragen werden.

NFA#3: Zuverlässigkeit, Verfügbarkeit, Reife, Fehlerverhalten

Die Verfügbarkeit der Daten ist gewährleistet, solange eine Verbindung zum Netzwerk besteht. Vor dem Laden neuer Daten wird zuerst ein Verbindungscheck durchgeführt, um möglicherweise auftretende Verbindungsfehler zwischen Client und Server abzufangen, welcher einen entsprechenden Statuscode liefert. Abhängig von diesem Statuscode erfolgt ein Request an den Server, bzw. ein Hinweis (zu sehen in

Mockup 4.15 / 4.16) an den Benutzer, dass keine Verbindung besteht. Dies erfolgt entweder per Hinweisleiste oder einer Aufforderung zum erneuten Laden im Full-screen-Modus, ermöglicht wird dies durch FA#9. Um die korrekte Anzeige der angeforderten Daten zu gewährleisten, werden diverse Statusvariablen im JSON Objekt übermittelt, welche der Client ausliest und weiterverarbeitet.

NFA#4: Wartbarkeit

Der Punkt der Wartbarkeit der App wurde erfüllt, die Verwendung des MVC Frameworks bietet bereits eine gewisse Aufteilung der Strukturen. Zudem liefert das Framework AngularJS einen guten Ansatz zur modularen Entwicklung. So ist es beispielsweise möglich, verschiedene Teile der Anwendung in Services auszulagern, um diese mehrfach zu verwenden. Im Falle einer Erweiterung oder eines Umbaus können die geforderten Änderungen an einer Stelle einfach zentral geändert werden.

NFA#5: Portierbarkeit

Die Möglichkeit der Portierung auf die gewünschte Plattform ist erfüllt durch Verwendung es Frameworks Phonegap.

7 Schluss

In diesem Kapitel werden abschließend die erarbeiteten Inhalte der vorangegangenen Kapitel zusammengefasst. Im Anschluss daran folgt ein Ausblick auf verschiedene Möglichkeiten, die App zu erweitern.

7.1 Zusammenfassung

Das Ziel dieser Arbeit bestand darin, durch Konzeption und Realisierung eines Prototyps einer hybriden App dessen Entwicklungsvorgang – angefangen von der Idee bis hin zur Implementierung - näher zu beleuchten.

Die funktionalen und nicht funktionalen Anforderungen aus Kapitel drei bildeten hierbei die notwendigen Rahmenbedingungen. Diese waren ausschlaggebend, um eine erfolgreiche Basis zur weiteren Entwicklung zu schaffen.

Nach der Festlegung der Anforderungen konnte in Kapitel vier mit der Konzeption begonnen werden. Zuerst wurden Designregeln wie die Verwendung von Farben oder der passenden Schriftart erarbeitet, danach wurde die mobile Anwendung anhand der Gestaltung von Mockups entworfen.

Im darauffolgenden Kapitel fünf Realisierung wurde mit der Implementierung der WebApp begonnen. Dafür wurden zuerst verschiedene, technische, für das Verständnis des nachfolgenden Inhalts erforderliche Punkte erläutert. Des Weiteren wurden ausgewählte Aspekte einer genaueren Betrachtung unterzogen und dessen Implementierung anhand verschiedener Beispiele näher beleuchtet. Anschließend folgte die Vorgehensweise zur Portierung der WebApp zu einer hybriden App.

Die Auswertung der Anforderungen erfolgte in Kapitel sechs (Anforderungsabgleich). Die festgelegten Anforderungen aus Kapitel drei wurden mit dem tatsächlich erreichten Stand abgeglichen. Dadurch konnte gezeigt werden, dass der Prototyp die geforderten Funktionen erfüllt.

7.2 Ausblick

Ein großer Teil der App konnte in Form eines Prototyps realisiert werden, so dass dieser bereits einsatzfähig ist. Ungeachtet dessen ist dieser Prototyp noch an verschiedenen Stellen erweiterungsfähig. Diverse, sinnvolle Funktionalitäten könnten noch verbessert bzw. hinzugefügt werden, um der App die nötige Marktreife zu verleihen, d.h. damit diese auch tatsächlich auf den entsprechenden Marktplätzen wie dem Google Play Store oder dem Appstore angeboten werden kann.
Nachfolgend werden einige Möglichkeiten aufgegriffen und kurz erörtert.

Für eine Distribution auf den Marktplätzen wäre die Durchführung eines oder mehrerer ausgiebiger Usability Tests zur Qualitätssicherung sinnvoll bzw. notwendig. Hierzu zählt einerseits der Test durch mehrere, unabhängige Testpersonen aus verschiedenen Altersschichten sowie unterschiedlicher technischer Erfahrung. Andererseits muss auch die technische Ausgereiftheit auf die Probe gestellt werden, um eventuell vorhandene Schwachstellen aufzudecken. Hierzu gehören beispielsweise das Beobachten der Performance – abhängig vom verfügbaren Netz – oder auch die Überprüfung auf den verschiedensten mobilen Geräten um die Darstellung zu testen.

Das in der Konzeption bereits geplante Bewertungssystem – FA#2.5 – konnte nicht im Prototypen realisiert werden. Dieses Feature ermöglicht dem Benutzer, die Einträge anderer Nutzer zu bewerten, beispielsweise negativ und positiv. Diese erhalten dadurch ein Feedback auf ihre Einträge.

Nicht in der Anforderung und Konzeption berücksichtigte, aber dennoch sehr sinnvolle bzw. bereichernde Verbesserungen wären die Implementierung eines Push Notification Systems und die Verwendung einer lokalen Datenbank.

7
Schluss

Push Benachrichtigungen finden heutzutage bei vielen Apps Anwendung, diese schicken dem Benutzer eine Nachricht über neue Ereignisse, ohne dass dieser die App aktiv bedient. Dadurch könnte der Benutzer informiert werden, sobald ein von abonnierter Nutzer einen neuen Eintrag verfasst hat und direkt darauf reagieren.
Die Verwendung einer Offline Datenbank – mittlerweile realisierbar mittels HTML5 – würde es dem Benutzer ermöglichen, auch Einträge zu lesen obwohl keine Verbindung zum Netzwerk besteht da die zuletzt geladenen Einträge in der Datenbank gespeichert werden. Sobald wieder eine Verbindung verfügbar ist, synchronisiert sich die App automatisch und zeigt, falls vorhanden, neuere Einträge an.

Anhang

Auf der beigelegten CD befindet sich der komplette Quelltext der entwickelten App zur Einsicht.

Für den tatsächlichen Betrieb der App muss diese auf einen Webserver geladen werden, außerdem muss die Datenbank eingerichtet werden.

Zu Demonstrationszwecken ist die WebApp Version unter folgender URL zu erreichen:

- http://www.richardholzner.de/ba

Literaturverzeichnis

[1] Statista – URL: http://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/ – Zugriffsdatum: 16.08.2015

[2] Statista – URL: http://de.statista.com/statistik/daten/studie/321935/umfrage/prognose-zum-anteil-der-smartphone-nutzer-in-deutschland/ – Zugriffsdatum: 16.7.2015

[3] Statista – URL: http://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/– Zugriffsdatum: 17.08.2015

[4] Bitbucket – URL: https://bitbucket.org/ – Zugriffsdatum: 02.06.2015

[5] Atlassian – URL: https://www.atlassian.com/git/ – Zugriffsdatum: 02.06.2015

[6] Briant Ford – URL: http://briantford.com/blog/angular-phonegap – Zugriffsdatum: 02.08.2015

[7] Phonegap – URL: http://phonegap.com/ – Zugriffsdatum: 06.08.2015

[8] Phonegap Build – URL: http://phonegap.com/about/ – Zugriffsdatum: 05.08.2015

[9] Peter Traeg – URL: http://www.smashingmagazine.com/2014/02/four-ways-to-build-a-mobile-app-part3-phonegap/ – Zugriffsdatum: 30.7.2015

[10] Infinite Scroll – URL: https://github.com/sroze/ngInfiniteScroll/tree/master/build – Zugriffsdatum: 01.07.2015

[11] Ionicons – URL: http://ionicons.com/ – Zugriffsdatum: 15.07.2015

[12] W3 School AngularJS – URL: http://www.w3schools.com/angular/default.asp – Zugriffsdatum: 25.06.2015

[13] Learn Angular – URL: http://www.ng-newsletter.com/posts/how-to-learn-angular.html – Zugriffsdatum: 25.06.2015

[14] Chris Sevilleja, Single Page Apps with AngularJS Routing – URL: https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templating – Zugriffsdatum: 22.06.2015

[15] Jason Watmore, AngularJS Basic HTTP Authentication Example – URL: http://jasonwatmore.com/post/2014/05/26/AngularJS-Basic-HTTP-Authentication-Example.aspx – Zugriffsdatum: 13.07.2015

[16] AngularJS Docs – URL: https://docs.angularjs.org/api/ng/service/$http – Zugriffsdatum: 10.06.2015

[17] Paul Lunow, Das Zusammenspiel von Directives,Services und Templates – URL: http://www.interaktionsdesigner.de/2013/das-zusammenspiel-von-directives-services-und-templates-in-angularjs/ – Zugriffsdatum: 10.06.2015

[18] 10 Thesen von Dieter Rams über gutes Produktdesign – URL: http://www.designwissen.net/seiten/10-thesen-von-dieter-rams-ueber-gutes-produktdesign – Zugriffsdatum: 14.07.2015

[19] Erich Gamma, Ralph Johnson - Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley ,2011

[20] W3C DOM – URL: http://www.w3.org/DOM/– Zugriffsdatum: 04.07.2015

[21] Android Developer – URL: http://developer.android.com/guide/webapps/webview.html – Zugriffsdatum: 04.08.2015

[22] Innotix– URL: http://www.innotix.com/blog – Zugriffsdatum: 09.08.2015

[23] Comvatix– URL: https://www.comvation.com/de/home– Zugriffsdatum: 11.08.2015

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel:

---

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Name Ort, Datum