ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Bachelorarbeit
im Studiengang Medieninformatik

Konzeption und Realisierung einer mobilen Schnittstelle gegen ein aktuelles Enterprise Content-Management-System

vorgelegt von

Michael Stach

Juni 2014

1. Gutachter	Prof. Dr. Manfred Reichert
Betreuer:	Rüdiger Pryss
Matrikelnummer	724231
Arbeit vorgelegt am:	12.06.2014

Kurzfassung

Informationen jeglicher Form werden heutzutage fast ausschließlich über das Internet abgerufen. Im Regelfall werden diese im World Wide Web über *Websites* angeboten, welche von Content-Management-Systemen erstellt und verwaltet werden. Die wachsende Verbreitung von mobilen Endgeräten erlaubt es dem Benutzer auch von unterwegs auf die gesuchten Informationen zuzugreifen. Existierende Informationssysteme müssen nun, im Hinblick auf die ansteigende Nutzung mobiler Endgeräte, ihre Inhalte auch für neue Plattformen anbieten können. Diese besitzen jedoch im Gegensatz zu klassischen Plattformen, wie beispielsweise der Desktop-Computer, hardwarebedingte Einschränkungen und deshalb eigene Formen der Informationsdarstellung, die man bei der Aufbereitung von Inhalten berücksichtigen muss.

Im Rahmen dieser Bachelorarbeit wird eine Schnittstelle, unter Verwendung der *REST*-Architektur, für ein existierendes Enterprise Content-Management-System entwickelt, dass die Basis für eine Applikation für mobile Betriebssysteme bildet. Die Applikation soll dabei über die Schnittstelle eine bidirektionale Kommunikation zum Enterprise Content-Management-System aufbauen, um eine konsistente Datenhaltung zu gewährleisten. Die Bereitstellung der Inhalte soll, im Zuge der Applikationsentwicklung, auf die plattformspezifischen Konventionen und Darstellungsformen von mobilen Endgeräten angepasst werden.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 12.06.2014 Michael Stach

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Zu einem guten Ende gehört auch ein guter Beginn.

Konfuzius, (551 - 479 v. Chr.)

1 Einleitung

Das öffentliche Anbieten eigener Informationen, über Produkte, Ziele und Dienstleistungen von Unternehmen oder Vereinigungen ist heutzutage ein zentraler Standteil der Öffentlichkeitsarbeit und des Marketing. Durch die steigende Verbreitung von mobilen Endgeräten, vor allem von Smartphones und Tablets, müssen Informationen zu eigenen Leistungen immer und überall abrufbar sein. Dieser Veränderung in der Informationsbeschaffung (Abbildung 1.1) müssen Unternehmen oder Vereinigungen gerecht werden, in dem sie Informationen über das World Wide Web bereitstellen. Diese Problematik wird detailliert durch die Referenzen [PMLR14], [PLRH12] und [RPR11] verdeutlicht, die eine wissenschaftliche Aufarbeitung von mobilen Problemstellungen durchgeführt haben.

Viele Unternehmen oder Vereinigungen nutzen für die Informationsbereitstellung sogenannte Enterprise Content-Management-Systeme (*E-CMS*), welche die Verwaltung von Informationen und anderen Inhalten im World Wide Web übernehmen. Die typischen Content-Management-Systeme (*CMS*) bereiten die verfügbaren Inhalte jedoch für stationäre Computer auf, welche in der Regel über eine hohe Auflösung auf einem größerem Display verfügen. Dies steht im Kontrast zu den Veränderungen, die Smartphones und Tablets bewirken. Smartphones und Tablets haben eigene Formen der Informationsdarstellung entwickelt, die auf kleine Displays mit niedrigen Auflösungen und Bedienung mit dem Finger optimiert sind. Aus Gründen der Konsistenz ist es von Vorteil, dass Informationen, in Form von Daten, nur von einem System zentral verwaltet werden. Deshalb müssen Informationen, die in der Regel nicht an Darstellungsbeschränkungen gebunden sind, auf den verschiedenen Plattformen selbst angepasst werden. Dafür werden spezielle Application Programming Interfaces (*APIs*) benötigt, auf die alle Plattformen zugreifen können, um Informationen abzurufen.

In dieser Arbeit wird eine solche Schnittstelle für ein bestehendes *CMS* entworfen und realisiert. Zusätzlich wird eine Applikation für mobile Betriebssysteme (*mobile App*) entwickelt, die auf die entstehende Schnittstelle zugreift, um Informationen aus dem *CMS* anzeigen und verwalten

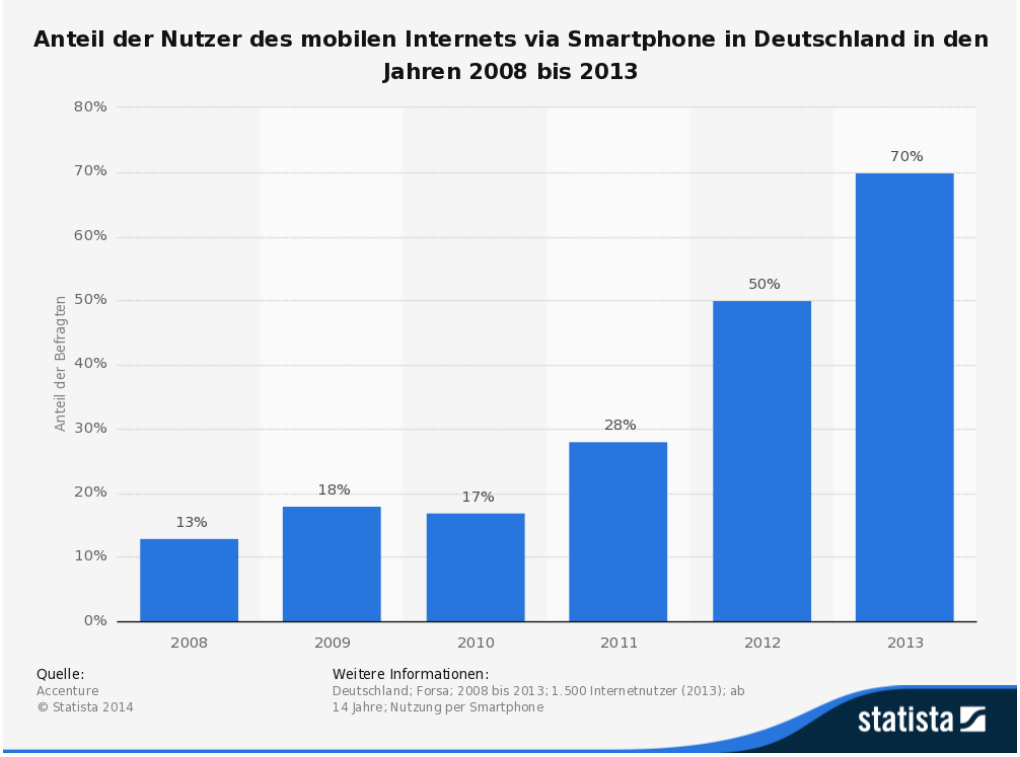


Abbildung 1.1.: Anteil der mobilen Internetnutzung via Smartphone zwischen 2008 und 2013, Quelle:[Gmba]

zu können. Die Verbindung soll bidirektional entwickelt werden, sodass Änderungen, aus der *App*, auch im System verarbeitet werden. Ein besonderer Blickwinkel gilt der Erweiterbarkeit, um Änderungen einfach in die entstehende Schnittstelle und die entstehende Applikation für mobile Betriebssysteme (*App*) zu integrieren und der Robustheit, damit Fehler nicht über Systemgrenzen weitergetragen werden.

1.1. Motivation

Die Arbeitsgemeinschaft niedergelassener Chirurgen Baden-Württemberg Nord (*ANC-BWN*), präsentiert sich bisher mit einer klassischen Internetplattform (*Website*), die von einem *E-CMS* namens "typo3" verwaltet wird. Mit dem stetig wachsenden Fortschritt und Verbreitung der mobilen Technologie, ändert sich das Nutzungsverhalten der Menschen und somit auch der Zugriff auf Informationen im World Wide Web. Um diese Anforderungen zu erfüllen, benötigt die *ANC-BWN* eine Applikation, welche ein breites Spektrum der aktuell verfügbaren mobilen Betriebssysteme abdecken soll.

1.2. Problemstellung

Betrachtet man die bisherige *Website*, so ist diese auf einen Zugriff mit einem stationären Computer, welcher über ein großes Display mit hoher Auflösung verfügt, ausgerichtet. Heutige *Websites*, welche von einem *CMS* verwaltet werden, müssen jedoch eine Vielzahl von Plattformen abdecken. Dadurch entstehen mehrere Problemstellungen (Tabelle 1.1).

Art	Kommentar
Flexibilität	Die Vielzahl von internetfähigen Geräten, welche unterschiedliche Auflösungen, Displaygrößen und Betriebssysteme besitzen, erhöht den Zwang einer universellen und stufenloser Darstellung.
Mobilität	Daten müssen an jedem Ort sicher abgerufen, verändert oder gelöscht werden können.
Modularität	Es müssen unterschiedliche Funktionen bereit gestellt werden, welche einfach erweitert und verändert werden können.
Funktionsumfang	Der Funktionsumfang muss der speziellen Plattform entsprechen.
Sicherheit	Auf bestimmte Inhalte sollen nur berechtigte Nutzer Zugriff erlangen.

Tabelle 1.1.: Problemstellungen heutiger Internetplattformen

Um diese Problemstellungen zu lösen, muss eine *API* für mobile Endgeräte errichtet werden, welche mit der Datenhaltung des *CMS* kommuniziert. Gleichzeitig wird eine *mobile App* entwickelt, welche mit der *API* bidirektional kommunizieren und ein möglichst großes Spektrum an Betriebssystemen abdecken soll. Da jederzeit neue Daten bereitgestellt werden können, soll die Erweiterbarkeit und Austauschbarkeit von einzelnen Komponenten des Systems ermöglicht werden. Dabei muss sichergestellt sein, dass nur bestimmte Nutzergruppen den Zugriff auf entsprechende Inhalte erlangen können.

1.3. Aufbau der Arbeit

Nachdem sich dieses Kapitel der Motivation und der Problemstellung gewidmet hat, wird in **Kapitel 2** nach verwandten Arbeiten und Lösungen zur aktuellen Problemstellung recherchiert. Dabei wird ein Ausschnitt einer exemplarisch ausgewählten Applikationen vorgestellt.

Im Folgenden **Kapitel 3** werden die Anforderungen an das zu entwickelnde System analysiert. Die Vorgehensweise der Analyse orientiert sich dabei an dem Referenzmodel für Usability Engineering [Off].

Mit Bezug auf die Anforderungsanalyse wird in **Kapitel 4** die Architektur des Systems erstellt. Hierbei werden die zu entwickelnden Anwendungsfälle spezifiziert und damit die einzelnen Be-

standteile der Systemarchitektur erstellt. Anschließend werden die Architekturbestandteile in eine Gesamtarchitektur zusammengesetzt.

Das **Kapitel 5** widmet sich anschließend der Implementierung des Systems. Zuerst werden die verwendeten Technologien vorgestellt und eine Struktur für die Kommunikation der Systembestandteile spezifiziert. Anschließend wird die Schnittstelle zum Content-Management-System und ihre Funktionsweise detailliert vorgestellt. Es folgt die Vorstellung der Funktionsweise der mobilen Applikation und eines ausgewählten Aspektes der Implementierung.

Die Vorstellung der fertigen Applikation für mobile Betriebssysteme folgt in **Kapitel 6**. Zu Beginn werden die Grundbausteine der *App* vorgestellt, die ein Grundgerüst bilden. Anschließend werden verwendete Dialog- und Benachrichtigungsformen beschrieben, die für die Umsetzung der Usability Ziele benötigt werden. Dem Aufbau des Anwendungsfalldiagramm folgend wird danach der öffentliche Bereich, die Basis für alle Nutzergruppen, vorgestellt, um mit der Vorstellung des zugriffsbeschränkten Bereichs ergänzt zu werden.

Um die Vollständigkeit der entwickelten Software zu überprüfen, wird in **Kapitel 7** das Entwicklungsergebnis mit den Spezifikationen aus Kapitel 3 abgeglichen und gleichzeitig bewertet.

Das **Kapitel 8** enthält eine Zusammenfassung des behandelten Themas und wird mit einem Ausblick auf die Zukunft der mobilen Entwicklung ergänzt.

2
Verwandte Arbeiten

In diesem Kapitel werden verwandte Systeme, respektive verwandte Arbeiten, gesucht, um Erkenntnisse über mögliche Anforderungen zu sammeln. Dies ist im Allgemeinen ein Teil der Anforderungsanalyse, beziehungsweise ein Teil der Analyse Ist-Stand (Kapitel 3). Da kein vergleichbares System für das verwendete *CMS* oder die verwendeten Hosting-Plattformen [Incf] existiert, trifft dieser Umstand zu der aufgeworfenen Problemstellung jedoch nicht zu. *APIs* für bestimmte *CMS* werden wegen der Kombination aus anwendungs- und systemspezifischer Problemstellungen selten generalisiert entwickelt. Dies würde den Anwendungskontext beeinträchtigt oder die Entwicklungszeit über ein bestimmtes Maß erhöhen, weshalb Eigenentwicklungen zu bevorzugen sind. Aufgrund der noch zu verfassenden Anforderungen, werden jedoch in Kapitel 2.1 einige Entwicklungen analysiert, um Erkenntnisse aus der Funktionsweise und dem Aufbau zu erlangen. Dabei werden verschiedene *APIs* vorgestellt, welche exemplarisch, aufgrund der universellen Einsetzbarkeit, ausgesucht wurden.

2.1. Exemplarische Schnittstellen

Die *APIs*, welche in diesem Kapitel vorgestellt werden, wurden exemplarisch aufgrund ihrer Funktionalität ausgewählt. Sie bieten ihre Funktionen dabei über Representational State Transfer (*REST*) konforme Schnittstellen an, über welche die Ausgabe von Informationen in JavaScript Object Notation (*JSON*) erfolgt.
Da es sich, zum Teil, um kommerzielle Produkte handelt, hebt der Verfasser hervor, dass keinerlei Verbindung zu den Produkten und dadurch kein Vorteil durch Erwähnung existiert.

2.1.1. Sandman

Sandman [Knu] ist eine in Python [Foub] programmierte *API*, basierend auf der *REST* Architektur (*RESTful API*). Der Schwerpunkt dieser Software liegt in der Einfachheit der Installation und Anwendung. Es unterstützt eine Vielzahl von Datenbanken und alle gängigen Betriebssysteme. Außerdem ist die Software in der Lage, die Datenbank nach einem Verbindungsaufbau zu scannen und kann dadurch ein übersichtliches Admin-Interface erstellen. Mittels diesem Admin-Interface ist der Nutzer in der Lage, Entities [Rei] zu suchen, Tabellen zu filtern und zu editieren. Um den Service für eine Tabelle anzubieten, muss der Entwickler dabei eine Klasse erstellen, welche von der existierenden Klasse *Model* erbt. Nachdem die Klasse erstellt wurde, muss diese im System registriert werden und wird damit automatisch überwacht. Sandman unterstützt die *HTTP*-Methoden [Web] *GET*, *POST*, *PATCH*, *PUT* und *DELETE*. Die Entities werden in *JSON* ausgegeben und sind über ein einfaches *URL*-Routing erreichbar.

Routing-Schema der *API*[1]: `<API-URL>/<modelname>/<primary-key>`

2.1.2. DreamFactory

DreamFactory [Inca] ist ein Softwarepaket für alle gängigen Betriebssysteme, Server und Cloud-Services, welches eine *RESTful API* für die Softwareentwicklung von Anwendersoftware für Unternehmen anbietet. Über die App-Bereitstellung lassen sich mehrere, in ihrer Konfiguration verschiedene, *APIs* direkt in DreamFactory erzeugen, wobei jede *API* eine eigene Route im *URL*-Dispatcher [Ltda] besitzt. Dadurch können verschiedene Tabellen und Felder unter unterschiedlichen Routen angezeigt werden. Der Schwerpunkt dieser Software liegt in dem umfangreichen Funktionsportfolio, wozu User-Management, Datenzugriff, App-Bereitstellung, *URL*-Dispatching und Sicherheit gehören. Die Ausgabe erfolgt in *JSON* oder in *XML*.

Routing-Schema der *API*[1]: `<API-URL>/<resource>/<tablename>/<primary-key>`

2.1.3. Apify

Apify [Ltdb] ist eine Bibliothek für ein PHP-Projekt (Kapitel 5.1), um die Erstellung von *RESTful APIs* zu beschleunigen. Der Schwerpunkt liegt bei Apify auf der Entwicklung von *APIs* gegen ein existierendes System. Dabei wurden Web *APIs* auf wiederkehrende Entwurfsmuster untersucht, um diese zu abstrahieren. Dadurch soll in der Projekt-Entwicklung Zeit und Aufwand eingespart werden. Um Apify zu verwenden, muss die Bibliothek auf dem Server abgelegt werden. Es unterstützt *URL*-Dispatching, Input-Validation, Content-Type-Negotiation und verwendet die *HTTP*-Response-Codes [Web]. Es werden die *HTTP*-Methoden *GET*, *POST*, *PUT* und *DELETE* unterstützt. Mithilfe des *URL*-Dispatchers können Requests an bestimmte Controller übergeben werden. Dabei können bestimmte Aktionen definiert werden, die im Allgemeinen die *CRUD*-Operationen abbilden. Über die Content-Type-Negotiation können die von Apify unterstützten Ausgabeformate (*JSON*, *XML*, *HTML*, *RSS*) zurückgegeben werden.

Routing-Schema der *API*[1]: `<API-URL>/<tablename>/<primary-key>/<operation>`

[1] Schema für den Entwicklungskontext dieser Bachelorarbeit vereinfacht.

3 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an das System, bestehend aus *API* und *mobile App*, analysiert und dargestellt. In Kapitel 3.1 wird zunächst, in der Benutzerprofilanalyse, auf die eigentlichen Benutzer des Systems eingegangen. Das folgende Kapitel 3.2 wird die Aufgaben und Funktionen im System analysieren. Dabei wird zwischen existierenden und durch die Entwicklung auf einer mobilen Plattform entstandenen Aufgaben getrennt. Im letzten Kapitel der Anforderungsanalyse (Kapitel 3.3) werden anschließend die Anforderungen, die ein mobiles Endgerät von einer robusten *API* fordert, spezifiziert. Die gesamten Ergebnisse der Anforderungsanalyse werden im Kapitel 4 für die Erstellung der Architektur verwendet.

3.1. Benutzerprofilanalyse

In der Benutzerprofilanalyse werden die Eigenschaften von Nutzern analysiert, um die Benutzerfreundlichkeit des Systems zu erhöhen und die Erwartungen, die ein Nutzer an ein solches System hat, zu erfüllen [Off]. Dafür werden in Kapitel 3.1.1 und 3.1.2 zuerst die persönlichen Eigenschaften und das Vorwissen der Nutzer analysiert, um Rückschlüsse auf die maximale Komplexität der Aufgaben und die Gestaltung des Endsystems zu ziehen. In Kapitel 3.1.3 werden anschließend die Nutzerrollen identifiziert, die im System berücksichtigt werden müssen. Bei der Entwicklung eines symbiotischen Systems für die *ANC-BWN* ist zu beachten, dass beide System-Komponenten getrennt voneinander betrachtet werden müssen. Für die *API*, die mit dem *CMS* kommuniziert, muss jedoch keine spezifische Benutzerprofilanalyse durchgeführt werden, da diese über keine grafische Oberfläche verfügt. Dies impliziert, dass das Systems als Experten-System konzipiert wird, da Programmierkenntnisse gefordert werden, und im weiteren nur Anforderungen an die *App* analysiert werden.

3.1.1. Persönliche Eigenschaften

Die persönlichen Eigenschaften der Nutzer werden benötigt, um Rückschlüsse auf die existierenden und gegebenenfalls fehlenden Fähigkeiten der Nutzer zu ziehen. Die Ergebnisse haben Einfluss auf die Gestaltung des Systems, um eine möglichst hohe Benutzerfreundlichkeit zu gewährleisten. Für die Analyse steht hierbei zunächst das Alter der Nutzer im Fokus. Dies ist ein wichtiger Implikator, um herauszufinden, ob es sich um ein Laien-System oder ein Experten-System handelt. Bei der *App* für die *ANC-BWN* liegt der Schwerpunkt im Bereich des mittleren, bis späten mittleren, Alters. Es entstehen dadurch keine Einschränkungen, da auf mobilen Plattformen die verwendeten Bedien- und Steuerelemente der grafischen Benutzeroberfläche (*Widgets*) eine ausreichende Grundgröße und Selbsterklärung besitzen.

3.1.2. Vorwissen

Um benutzerfreundliche Systeme zu erstellen, ist es wichtig, dass das Vorwissen und die Erfahrung der Benutzer analysiert wird. Dadurch können Unsicherheit im Umgang mit dem System vermieden werden. Außerdem ist es elementar, wenn das entstehende System in den Alltag des Benutzers einfließen soll. Bei der *App* für den *ANC-BWN* müssen allgemeine Computer-Kenntnisse, sowie die die Erfahrung mit mobilen Anwendungen berücksichtigt werden. Da keine detaillierten Informationen über Nutzer vorliegen, dient die in Kapitel 3.1.1 analysierte Altersgruppe als Grundlage für die Analyse des Vorwissen. Aufgrund dessen und dass nur wenige Nutzer beim bisherigem System angemeldet sind, kann man schließen, dass zumindest ein Teil der Benutzer nur rudimentäre Grundkenntnisse im Umgang mit *Apps* besitzt. Es muss zusätzlich berücksichtigt werden, dass bei einer Veröffentlichung der *App* eine neue Altersschicht eine Rolle spielen kann. Diese kann, im Allgemeinen, jedoch als jünger oder gleichen Alters und erfahrener oder zumindest als gleich erfahren eingeschätzt werden. Da es sich bei diesen Nutzern um Gäste handelt, kann man davon ausgehen, dass sie die *App* deutlich seltener nutzen.

3.1.3. Rollen

Aufgrund der bisher analysierten Benutzer und der bisher vorherrschenden Trennung von Gast und Mitglied im *CMS*, ergeben sich zwei Benutzerrollen. Diese werden in Tabelle 3.1 weiter spezifiziert.

3.2. Aufgabenanalyse

In diesem Kapitel werden alle nötigen Aufgaben analysiert, die das entstehende System erfüllen soll. Die *API* und die *mobile App* sind getrennt zu betrachten, da sie im Realeinsatz autonom agieren und eine extrem lose Kopplung [VAC+08] besitzen. Es muss zusätzlich unterschieden werden, welche Aufgaben im vorhandenen System schon existieren und welche mit dem zu entwickelnden System neu entstehen. Im Kapitel 3.2.1 werden zuerst die existierenden

Rolle	Kommentar
Gast	Der Gast ist ein *ungeübter und sporadischer* Benutzer [Off] und soll keine erweiterten Interaktionsfähigkeiten mit dem Inhalt der *App* besitzen. Die *App* ist für diese Benutzerrolle reine Informationsplattform.
Mitglieder	Das Mitglied ist ein *ungeübter und regelmäßiger* Benutzer [Off] und soll über erweiterte Interaktionsfähigkeiten mit der *App* verfügen. Diese soll als möglicher Ersatz für das bestehende User Interface (*UI*) der klassischen Website dienen und deren Funktionen abbilden.

Tabelle 3.1.: Entstandene Rollen aus der Benutzerprofilanalyse

Aufgaben analysiert und kommentiert. Das Kapitel 3.2.2 widmet sich anschließend den durch die Entwicklung der mobilen Applikation entstandenen Aufgaben.

3.2.1. Existierende Anforderungen

Um die existierenden Aufgaben zu ermitteln, wird das vorhandene System auf sein Funktionsspektrum und seine Charakteristiken analysiert (Tabelle 3.2). Dies ist notwendig, damit keine vom Nutzer erwarteten Funktionen verloren gehen. Ein Verlust würde sich negativ auf die Erwartungskonformität des neuen Systems [Off] auswirken. Die in Abbildung 3.1 dargestellte Navigation dient als Grundlage für das momentane Aufgabenspektrum eines Mitglieds des *ANC-BWN*. Dabei werden die Aufgaben in funktionale und nicht funktionale Anforderungen [Par] getrennt.

Abbildung 3.1.: Ausschnitt aus der Website der *ANC-BWN* und deren Mitgliedernavigation, Quelle: [BN]

Anforderung	Art	Kommentar
Ein Gast soll öffentliche Neuigkeiten lesen können.	funktional	Für das Bereitstellen öffentlich Neuigkeiten muss ein Unterschied zu internen Neuigkeiten in der Datenhaltung gefunden werden. Die Trennung zwischen öffentlicher und interner Neuigkeit ist eine Kernfunktion.
Ein Gast soll Standorte auf einer Karte einsehen können.	funktional	Es muss ein Anbieter für Kartenmaterial gefunden werden, der die Markierung von Standorten unterstützt. Die Karte muss über JavaScript eingebunden und konfiguriert werden können. Es müssen Standorte auch ohne Markierung auf der Karte, beziehungsweise nur für interne Sichtbarkeit, angezeigt werden können.
Ein Gast soll Kontakt aufnehmen und rechtliche Informationen einholen können.	funktional	Diese Informationen werden statisch in der *App* gespeichert, da sie in der Regel nicht geändert werden und mit störenden Markup in der Datenbank vorliegen.
Ein Benutzer soll sich über das System anmelden und abmelden können.	funktional	Die *App* muss sich bei der *API* autorisieren können. Dies wird bei jedem zugriffsbeschränkten Aufruf nötig, um ein Session-Timeout zu verhindern. Deshalb ist der Abmeldevorgang nur lokal möglich.
Das System soll zwischen Benutzergruppen unterscheiden können.	funktional	Damit Mitglieder geschützt Informationen austauschen können, soll das System Gästen nur eine eingeschränkte Funktionalität bieten. Diese sollen dabei nicht in ihrer User-Experience [Inck] gestört werden.
Ein Mitglied soll öffentliche und interne Neuigkeiten einsehen und verwalten können.	funktional	Die bidirektionale Verbindung muss von der *API* unterstützt werden. Änderungen sollen als atomare Transaktion [Rei] ausgeführt werden.
Das System soll ein möglichst großer Querschnitt an mobilen Betriebssystemen abdecken.	nicht funktional	Das System wird als Web-Projekt entwickelt und mittels der jeweiligen Web-View-Komponente des mobilen Betriebssystems angezeigt. Die Projekt-Binaries werden mit dem Cordova-Framework [Foua] (Kapitel 5.1) erstellt.
Das System soll äußerst zuverlässig sein.	nicht funktional	Durch die Verwendung von Skriptsprachen ist das Ergebnis eines Aufrufs vom vorherigen Aufruf in der Regel unabhängig. Dies garantiert für jeden einzeln Aufruf eine höhere Zuverlässigkeit.
Das System soll dem Nutzer bei jeder Handlung ein Feedback geben.	nicht funktional	Es wird für jede Aktion eine entsprechende Rückmeldung angezeigt. [Off]

Tabelle 3.2.: Aufgabenanalyse des bereits existierenden Systems

3.2.2. Entstandene Anforderungen

Bei der Entwicklung von neuen Systemen entstehen gleichzeitig neue Anforderungen, die im Allgemeinen spezifischer an einen bestimmten Nutzungskontext gebunden sind. In der Tabelle 3.3 und 3.4 werden diese neuen Anforderungen, für das entstehende System, aufgelistet und in funktional und nicht funktional [Par] getrennt.

3.3. Randbedingungen

Bevor das System entwickelt werden kann, müssen zuerst die Randbedingungen, in der das System zum Einsatz kommt, analysiert werden. In diesem Kapitel wird zuerst auf die Umgebungsbedingungen (Kapitel 3.3.1) als Bestandteil der Anforderungsanalyse [Off] eingegangen. Anschließend werden die Hardware- und Software-Bedingungen (Kapitel 3.3.2) analysiert.

3.3.1. Umgebungsbedingungen

Bei der Analyse der Umgebungsbedingungen spielt die physikalische Arbeitsumgebung des Benutzers und des Systems eine wesentliche Rolle. Diese Aspekte sind wesentliche Indikatoren für Störeinflüsse, denen das System ausgesetzt ist. Bei der Entwicklung des Systems wird sich das Einsatzgebiet voraussichtlich nicht auf den Arbeitsplatz beschränken. Bei einer mobilen Anwendung muss davon ausgegangen werden, dass das System von überall aufgerufen wird. Dies bedeutet, dass nicht immer eine Verbindung zum Internet gewährleistet werden kann. Zusätzlich muss berücksichtigt werden, dass bestehende Verbindungen äußerst instabil sein können und deshalb eine äußerst robuste und fehlertolerante Kommunikation zwischen *mobile App* und *API* entwickelt werden muss. Weitere Einflüsse, wie Licht, Klima, Schmutz oder Lärm [Off] spielen bei diesem System keine Rolle.

3.3.2. Hard- und Software

Vor dem Erstellen einer Architektur müssen Hardware- und Software-Randbedingungen erhoben werden, um beispielsweise die Möglichkeiten der Gestaltung, auf einer bestimmten Zielplattform, einzugrenzen. Bei der Erstellung von *mobilen Apps* besitzt jedes Betriebssystem eigene Styleguides und Konventionen, die bestimmte grafische Objekte, Stilrichtungen und Anordnungen vorgeben. Außerdem müssen spezifische Eigenheiten der Hardware, wie beispielsweise geringe Auflösung, und Eigenheiten der Software, wie Kompatibilität, in der Analyse berücksichtigt werden. Bei der entstehenden *App* handelt es sich um ein Web-Projekt, weshalb die Bindung an bestimmte grafische Objekte und Konventionen entfällt. Eine Kompilierung für ein bestimmtes Betriebssystem erfolgt über das Cordova-Framework [Foua] (Kapitel 5.1), welches alle aktuellen Betriebssysteme (Abbildung 3.2) unterstützt.

Es muss zusätzlich berücksichtigt werden, dass Smartphones und Tablets in der Regel über kleinere Displays verfügen und die Informationen kompakter dargestellt werden müssen. Wie

Anforderung	Art	Kommentar
Ein Benutzer soll sich einen Eindruck der Praxis verschaffen können.	funktional	Mithilfe von *AJAX* wird eine *JSON*-Datei [Inch] eingelesen, in der die Praxiszugehörigkeit, ein Bild und ein Beschreibungstext steht. Es kann, der Reihenfolge des Vorkommens in der *JSON*-Datei folgend, ein bestimmtes Element mittels Vor- und Zurück-*Button* ausgewählt werden.
Das System soll mittels einer *API* Kontakt zur Datenbank aufnehmen.	funktional	Es wird ein Datenbank-Controller mit den jeweiligen *CRUD*-Operationen [Ince] implementiert.
Die *API* soll Requests auf Syntax und Autorisierung validieren.	funktional	Eine Validierungs-Klasse prüft jeden Request auf korrekte Syntax. Falls ein zugriffsbeschränkter Request eingeht, wird zusätzlich auf Autorisierung geprüft.
Das System soll auf wichtige Hardware- und Software-Events reagieren.	funktional	Jedes Modul soll bei Modulinitialisierung alle für sich interessanten Events behandeln. [Wen10]
Ein Mitglied soll einfach öffentliche und interne Standort-Daten und -Einstellungen, sowie Rundschreiben, einsehen und verwalten können.	funktional	Es wird innerhalb eines Aktionsmanagers eine Zugriffsbeschränkung für bestimmte Aktionen auf der Datenbank geben. Diese können nur mit der entsprechenden Autorisierung ausgeführt werden.
Die Praxisdarstellungen sollen leicht erweiterbar sein.	funktional	Die *JSON*-Datei liegt auf dem Server und kann, aufgrund der einfachen Syntax, leicht erweitert werden.
Das System soll modular aufgebaut und leicht erweiterbar sein.	nicht funktional	Modularisierung der *API* wird mittels einzelner Klassen und Aktionen realisiert. Diese Aktionen können jederzeit geändert oder gelöscht, beziehungsweise hinzugefügt werden. In der *App* werden Funktionen und Einstellungen innerhalb einzelner Objekte gekapselt und können einzeln als Modul initialisiert werden.
Das System soll als benutzerfreundliches Laiensystem gestaltet werden.	nicht funktional	Das Look-and-Feel orientiert sich an existierenden Darstellungskonventionen für mobile Endgeräte und wird selbsterklärend gestaltet.

Tabelle 3.3.: Aufgabenanalyse des zu entwickelnden Systems

Anforderung	Art	Kommentar
Das System soll eine optimale Darstellung in jeglicher Auflösung bieten.	nicht funktional	Mittels Stil-Angaben in Prozent-Werten wird ein Responsive Design [Incj] sichergestellt.
Das System soll sehr robust gegenüber Fehleingaben, struktureller Mängel und Verbindungsfehlern sein.	nicht funktional	Mittels extrem loser Kopplung [VAC+08] werden Abhängigkeiten von Methoden vermieden. Die *API* agiert unabhängig von der *App* und liefert nur das Ergebnis. Wird ein fehlerhaftes Ergebnis geliefert, wird die Bearbeitung innerhalb der *App* gestoppt und ein entsprechendes Feedback ausgegeben.
Das System soll, vor allem für Mitglieder, selbsterklärend sein.	nicht funktional	Durch Beachtung von Gestaltungsprinzipien [Off] werden Analogien zu existierenden Systemen erstellt.

Tabelle 3.4.: Aufgabenanalyse des zu entwickelnden Systems (Forts.)

bereits in Kapitel 3.2.2 erläutert, müssen bei der Gestaltung der Oberfläche auch verschiedene Auflösungen berücksichtigt werden, weshalb eine Entwicklung im Responsive Design [Incj] angestrebt wird.

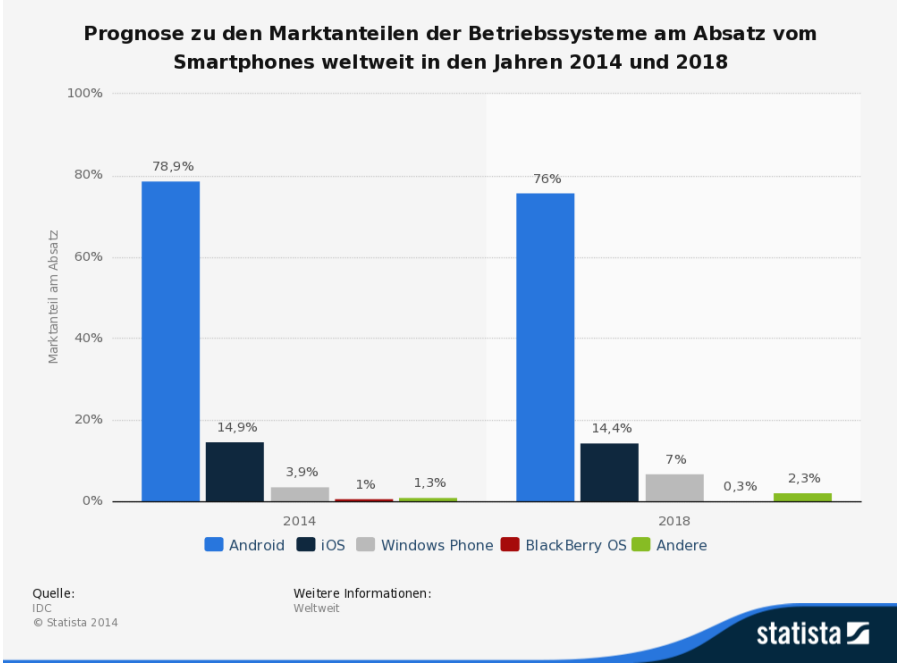


Abbildung 3.2.: Prognose der Marktanteile von Smartphone-Betriebssystemen weltweit, Quelle: [Gmbb]

4 Architektur

In diesem Kapitel wird aus der Anforderungsanalyse eine Systemarchitektur entworfen. In Kapitel 4.1 wird ein Anwendungskontext generiert, der sich auf die Aufgabenanalyse in Kapitel 3.2 bezieht. Anschließend wird in Kapitel 4.2 das System nach dem "Teile und Herrsche"-Prinzip in zwei Teilarchitekturen aufgeteilt. Zuletzt werden in Kapitel 4.3 die Teilarchitekturen zu einer Gesamtarchitektur zusammengesetzt.

4.1. Anwendungskontext

Das Anwendungsfalldiagramm kommt aus der Gruppe der Verhaltensdiagramme der Unified Modeling Language (*UML*). Es gibt einen Überblick über die wesentlichen Systemfunktionen und deren Beziehungen aus der Benutzersicht [Par]. Es besteht eine bidirektionale Kommunikation zwischen der *RESTful API* [Inci] und der *App*, wobei diese auf dem "Request-Response"-Schema des *HTTP*-Protokolls [Web] basiert. Die Kommunikation wird von der *App* mit einem Requests ausgelöst, der, für den Benutzer nicht sichtbar, im Hintergrund abgeschickt wird. Die *API* wird so implementiert, dass sie zu jeder Zeit auf ein Request antwortet, selbst wenn der Request für sie unbekannt oder fehlerhaft ist. Eine Response des Server erfolgt stets in der JavaScript Object Notation [Inch].

Aus dem Anwendungsfalldiagramm (Abbildung 4.1) wird deutlich, dass es zwei Benutzerrollen (Tabelle 3.1) im System geben wird. Dabei erbt das Mitglied das komplette Funktionsspektrum des Gastes. Parallel zur Vererbung in Java, werden Funktionen eines Gastes von den Funktionen eines Mitgliedes überschrieben. Dies hat zur Folge, dass beispielsweise die Ansicht der Neuigkeiten, aus der Rolle eines Mitgliedes, erweiterte Funktionen aufweist. Diese sollen direkt in die bestehende *UI* eingebettet werden und den, in Kapitel 3.2 definierten, nicht funktionalen Anforderungen entsprechen.

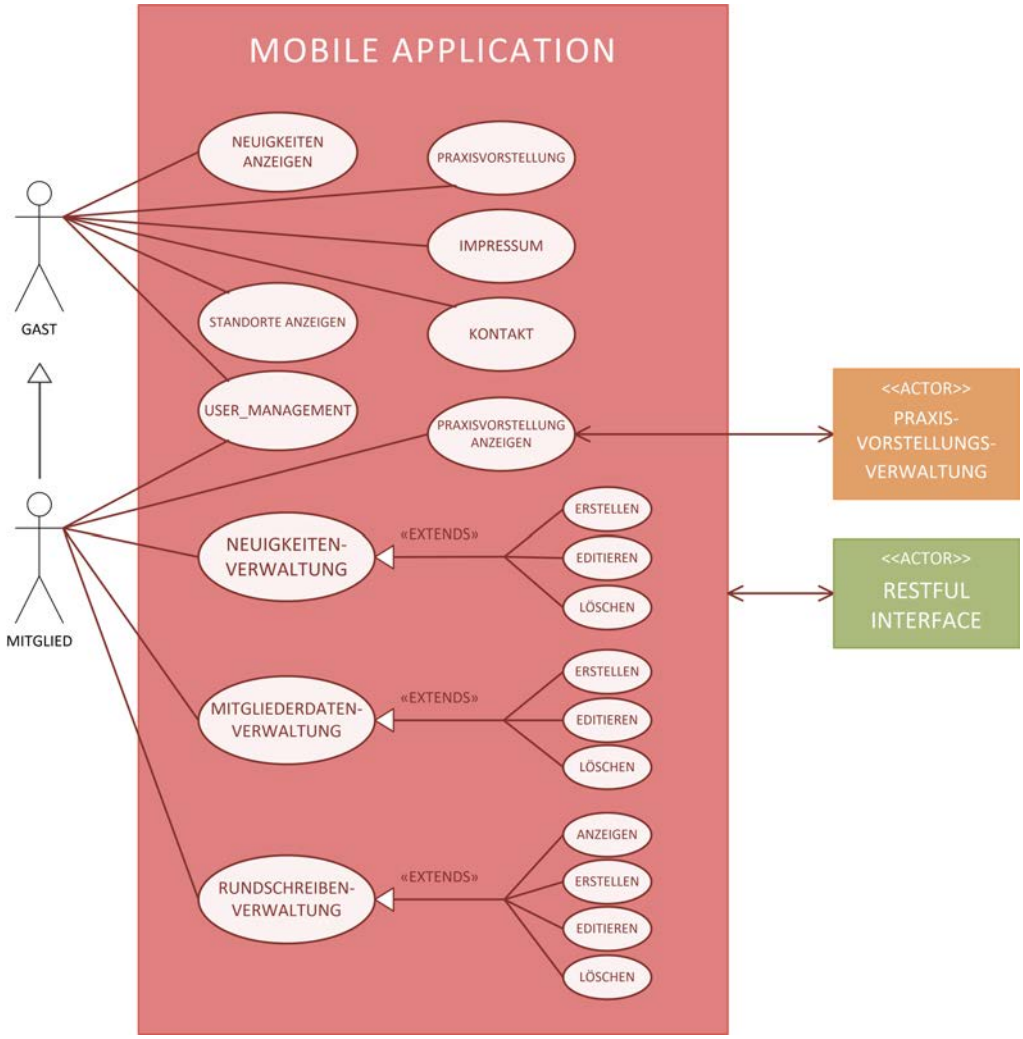


Abbildung 4.1.: Anwendungsfälle des Systems

Zusätzlich besitzen Mitglieder eigene Anwendungsfälle, die hauptsächlich zur Verwaltung des Inhaltes dienen. Dazu gehört die Neuigkeitenverwaltung, die Standort- und Mitgliederdatenverwaltung und die Verwaltung von Rundschreiben, welche Änderungen, mittels der *RESTful API*, im *CMS* abgleichen. Eine Besonderheit ist der Anwendungsfall "Praxisvorstellung anzeigen", welcher die benötigten Daten nicht über die *RESTful API* erhält, sondern über ein weiteres, nicht näher spezifiziertes, System, welches die Informationen in *JSON* ausgibt. Dieses System wurde ausgelagert, da die Verwaltung von Bildern auf einem Dateisystem und nicht in einer Datenbank erfolgen soll und es Gästen, über eine bekanntes *UI* ermöglicht, Bilder und Beschreibungstexte zu verfassen und mittels *FTP* auf den Server zu laden.

4.2. Architekturbestandteile

Die Architektur des Systems für die *ANC-BWN* ist in zwei autonome Bestandteile getrennt. Im ersten Kapitel 4.2.1 wird eine *RESTful API* anhand der Anforderungen aus Kapitel 3 dargestellt, die auf *HTTP*-Requests reagiert und mit einer entsprechenden *HTTP*-Response antwortet. Den Aufbau einer Anfrage, beziehungsweise einer Antwort, wird in Kapitel 5.2 definiert. In Kapitel 5.4 wird anschließend die Architektur der *mobile App* beschrieben, welche mit der *RESTful API* bidirektional kommuniziert.

4.2.1. RESTful API

Die *RESTful API* (Abbildung 4.2) verbindet die Datenhaltung des *CMS* mit externen Anwendungen und kann damit die Konsistenz des verwalteten Inhalts sicherstellen. Damit die *RESTful API* die Anforderungen der Aufgabenanalyse (Kapitel 3.2) erfüllt, muss sie in verschiedene gekapselte Aufgabenbereiche eingeteilt werden. Diese Modularisierung der Aufgabenbereiche soll mithilfe von Objektorientierung entstehen. Es sollen fünf Module entwickelt werden, welche den Request entgegen nehmen, die Validation und Autorisierung der Anfrage übernehmen, die Anbindung an die Datenbank sicherstellen, vordefinierte Aktionen prüfen und laden und welche die Query für eine geladene Aktion schreiben und ausführen.

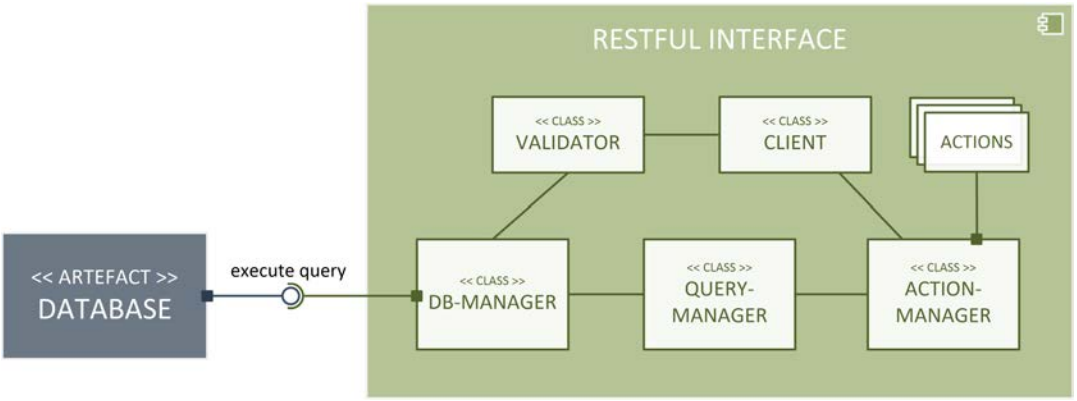


Abbildung 4.2.: Architektur der Schnittstelle zum Content-Management-System

Ein *HTTP*-Request wird zuerst im *Client* entgegengenommen und analysiert. Diese Analyse ist notwendig, da es verschiedene Nutzergruppen (Kapitel 3.1.3) und damit unterschiedliche Sichtbarkeiten auf Ressourcen im System gibt. Dabei werden auch Nutzerdaten mit der Anfrage gespeichert, um Fehler durch Anfragen besser ausfindig machen zu können. Nach der Analyse wird die Anfrage zur Validation übergeben, welche die vorgeschriebene Syntax (Kapitel 5.2) auf Fehler überprüft. Falls eine Anfrage eine Autorisierung verlangt, so wird in der Validation, bei erfolgreicher Syntaxüberprüfung, eine Anfrage mit den Nutzerdaten an die Datenbank gesendet. Dadurch wird, falls die Nutzerdaten korrekt sind, eine ID zurückgegeben, welche für weitere Verarbeitungen im *QueryManager* benötigt wird. Sind die Nutzerdaten unvollständig oder nicht korrekt, so wird eine passende Antwort im *QueryManager* erstellt

und dem *Client* übergeben. Dieser bricht die Bearbeitung ab, wenn eine Antwort generiert wurde und leitet diese als Systemantwort weiter. Falls er keine Antwort vom *Validator* erhält, so kann er die Anfrage an den *ActionManager* weiterleiten. Dieser prüft im System hinterlegte Aktionen, welche eine vollständige Datenbanktransaktion [Rei] beschreiben, und liest diese bei Anfrage ein. Ist eine *Action* valide, so wird er diese an den *QueryManager* weiterleiten, welcher daraus die endgültige Query erstellt. Ist diese invalide, so wird der *ActionManager* die Bearbeitung abbrechen und eine entsprechende Response an den Client weiterleiten. Der *QueryManager* prüft dabei, ob alle in der Aktion geforderten Daten in dem *HTTP*-Request vorhanden sind und übergibt diese danach dem *DB-Manager*. Aus dem *DB-Manager* verlässt die Query anschließend die *API* und wird in der Datenbank ausgeführt. Das Ergebnis, ob erfolgreich oder fehlerhaft, wird danach an den *QueryManager* zurückgegeben, worauf eine entsprechende Antwort generiert wird. Diese wird anschließend, über den *ActionManager*, an den *Client* weitergereicht und als Systemantwort ausgegeben.

4.2.2. Mobile Applikation

Die Architektur der *mobile App* (Abbildung 4.3) baut auf dem Cordova-Framework (Kapitel 5.1) auf und kann dadurch auf mehrere mobile Betriebssysteme portiert werden. Innerhalb des Cordova-Frameworks wird ein Web-Projekt entstehen, welches Hypertext Markup Language für die Beschreibung des Inhaltes und JavaScript für den Zugriff auf Inhalte verwendet. Diese JavaScript-Dateien bilden den *Application Code*, welcher in vier einzelne Container unterteilt ist. Diese Container verwalten und initiieren das Projekt (*INDEX*), greifen auf Hardware-Informationen (*DEVICE*), mithilfe des Cordova-Frameworks, zu, beschreiben die Kernfunktionen der Applikation (*APP*) und bieten Funktionen für eine interaktive Steuerung (*UI*) an. Jede Aufgabe wird dabei in einem Container als Modul gekapselt und zur Verfügung gestellt.

Es existieren, zusätzlich zum *Application Code*, auch weitere Frameworks (Kapitel 5.1) innerhalb des Web-Projektes. Diese agieren über bidirektionale, gegebenenfalls auch unidirektionale, Schnittstellen mit dem *Application Code*. Eine große Rolle in der Entwicklung der *mobile App* nimmt das jQueryMobile-Framework ein. Dieses Framework bietet *Widgets* und interne Mechanismen, um die Darstellungsformen von *Apps*, mithilfe von statischem Hypertext Markup Language (*HTML*)-Markup, zu simulieren. Um den Inhalt, in Abhängigkeit der Server-Antwort, zu erstellen, kommt das Handlebars-Framework zum Einsatz. Dieses Framework kann *JSON*-Dateien parsen und mittels eines vorgefertigten Templates zu *HTML*-Markup kompilieren. Das Kompilieren des Markup soll dabei während der Ladephase einer *HTML*-Datei ausgeführt werden, um das Template vor dem Nutzer zu verbergen. Ist die Kompilierung abgeschlossen, wird das entstandene Markup dem jQueryMobile-Framework übergeben, welches die Ausgabe übernimmt. Zusätzlich werden zwei weitere Frameworks eingebunden, welche nach der Initialisierung nicht mehr kontaktiert werden. Das *fastclick*-Framework verringert nach der Initialisierung die Zeit, in der vom Betriebssystem sichergestellt wird, dass ein Nutzer nicht versehentlich mit dem System interagiert. Das *normalizer*-Framework wird benötigt, um die verschiedenen Standard-Werte der Stildefinition von *HTML*-Elementen zu vereinheitlichen und ist bei einer Entwicklung für mehrere Plattformen unerlässlich, wenn eine konsistente Darstellung garantiert werden soll.

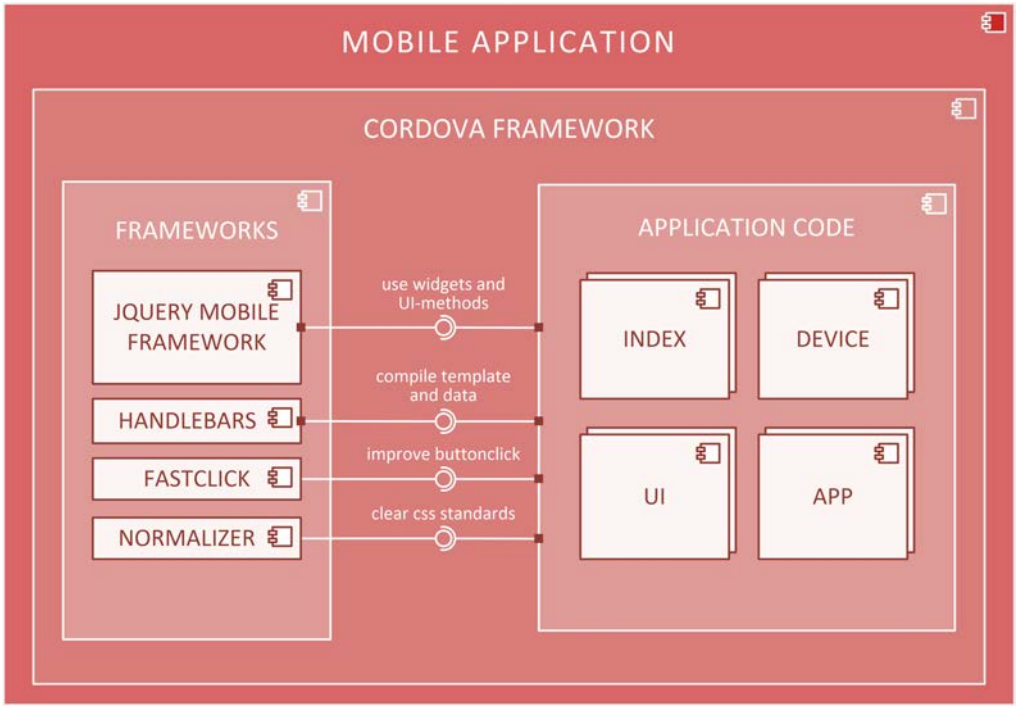


Abbildung 4.3.: Architektur der mobilen Applikation

4.3. Gesamtarchitektur

Die Gesamtarchitektur (Abbildung 4.4) stellt alle Komponenten mit den kommunizierenden Bestandteilen und dessen Verbindungen untereinander dar. Dadurch können Abhängigkeiten zwischen den, bisher autonom betrachteten, Komponenten sichtbar gemacht werden, um, in den einzelnen Komponenten, das Verhalten nach außen (Kapitel 5.2) abzustimmen. In Abbildung 4.4 werden Verbindungen mit der entsprechenden Farbe, über die Systeme miteinander kommunizieren, gekennzeichnet. Die Enden mit einem kompletten Kreis, auch *Provided Interfaces* genannt, stellen dabei Funktionen zur Verfügung, welche mittels eines Requests von außerhalb ausgelöst werden können. Die Enden mit einem halben Kreis, auch *Required Interfaces* oder *Sockets* genannt, rufen Funktionen bei anderen Interfaces auf und können selbst nicht von außen kontaktiert werden. Die quadratischen Kästchen, an den Enden der Verbindungen, sind sogenannte *Ports*, die eine Öffnung in das Systems verdeutlichen.

Die *mobile App*, in Abbildung 4.4 innerhalb der roten Komponente, besitzt einen Port, über den Sie mit dem *RESTful API* und der Verwaltung von Praxisvorstellungen verbunden ist. Dies ist die einzige Schnittstelle, über die sich das System mit Inhalt versorgt und ist deshalb generalisiert. Diese einzelne Möglichkeit zur Kommunikation erhöht die Sicherheit des Systems, da dadurch weniger Möglichkeiten zur Infiltrierung des Systems entstehen. Außerdem ist in der Abbildung 4.4 zu erkennen, dass die *RESTful API* nur über den *Client* Anfragen erhalten kann. Ein Zugriff auf Funktionen außerhalb des Systems ist nur dem *DB-Manager* gestattet, um Operationen auf der Datenbank auszuführen.

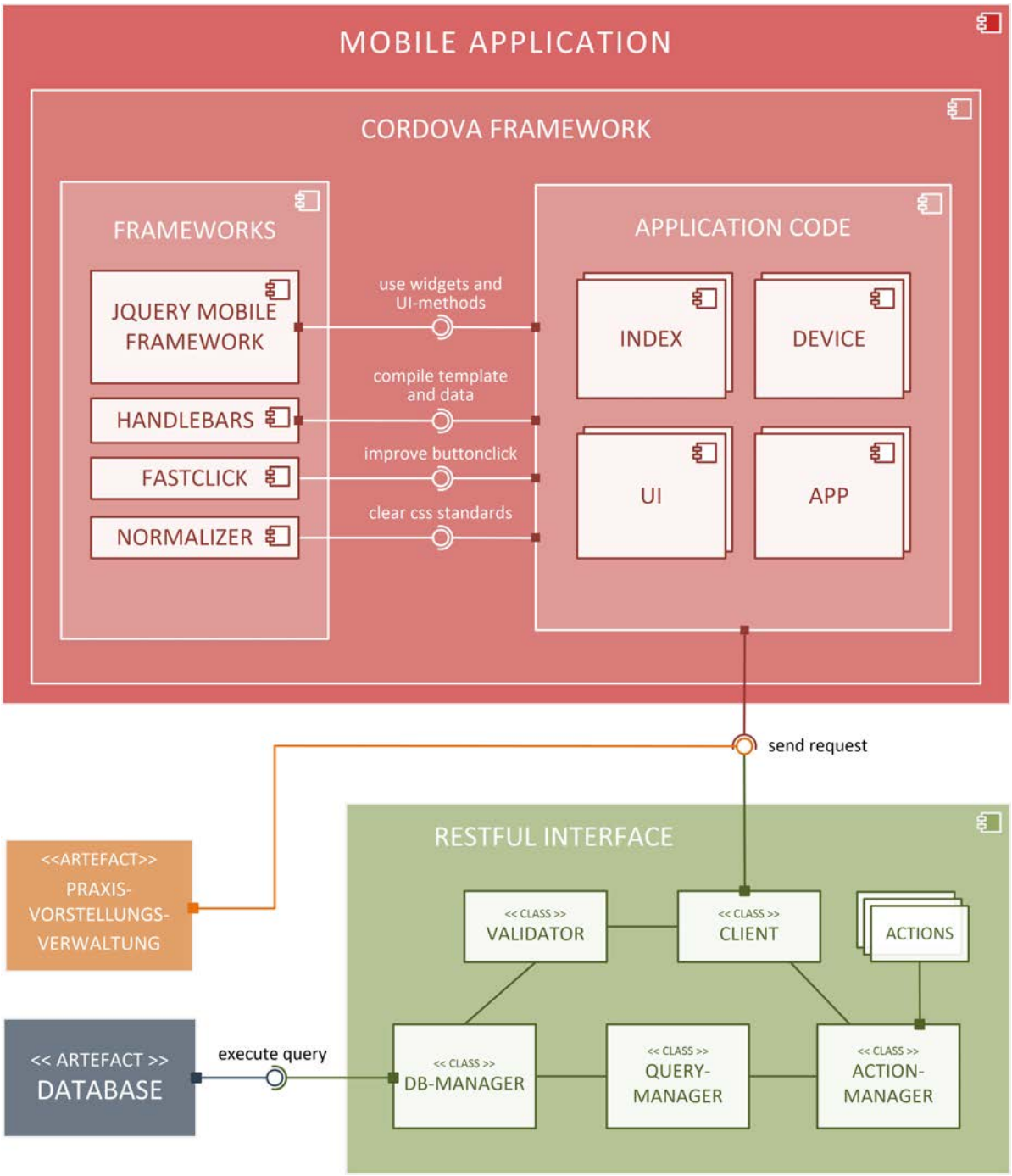


Abbildung 4.4.: Gesamtarchitektur des zu entwickelnden Systems

5 Implementierung

Im Kapitel 4 wurde die vollständige Architektur des Systems ausgearbeitet. In diesem Kapitel wird, auf der Basis der Architektur, ein funktionsfähiges System entwickelt und erläutert. Dabei werden in Kapitel 5.1 die verwendeten Technologien und ihre Notwendigkeit skizziert. Anschließend werden in Kapitel 5.2 strukturelle Vereinbarungen getroffen, die für eine Kommunikation der *API* und der *App* erforderlich sind. Auf Grundlage der Architektur und der Vereinbarungen wird darauf folgend die *RESTful API* in Kapitel 5.3 beschrieben, die in Kapitel 5.4 durch die Mobile Applikation ergänzt wird. Zuletzt werden im letzten Kapitel einzelne Aspekte der Implementierung beider Teilsysteme genauer betrachtet.

5.1. Verwendete Technologien

In diesem Kapitel werden die verwendeten Technologien kurz vorgestellt. Des weiteren wird erläutert, warum die Notwendigkeit bestand, die Technologie für das Projekt zu verwenden. Dabei werden zusätzlich Problemfälle im System erläutert und weshalb die entwickelte Lösung sich letztendlich als Vorteil erweist.

PHP

Die klassische Skriptsprache *PHP* ist, mit 82% (Stand: 31. Mai 2014) weltweit, die am häufigsten verwendete serverseitige Programmiersprache [W3T]. Seit der Version 5 unterstützt *PHP* zusätzlich die Objektorientierung, mittels der die *RESTful API* entwickelt wurde. Die Notwendigkeit der Verwendung von *PHP* basiert auf dem existierenden *CMS* der *ANC-BWN*, welches mit *PHP* entwickelt und veröffentlicht wurde. Daraus resultierte die Anforderung *PHP* als Programmiersprache, für die *RESTful API* zu verwenden, um die vorhandenen Leistungen des Servers aufzugreifen. Die Funktionsweise von *PHP* (Abbildung 5.1) unterstützt zudem den

Entwurf der *API*, da jede Anfrage erneut den *PHP*-Interpreter aufruft und dies die Zustandslosigkeit und Unabhängigkeit der Anfragen unterstreicht.

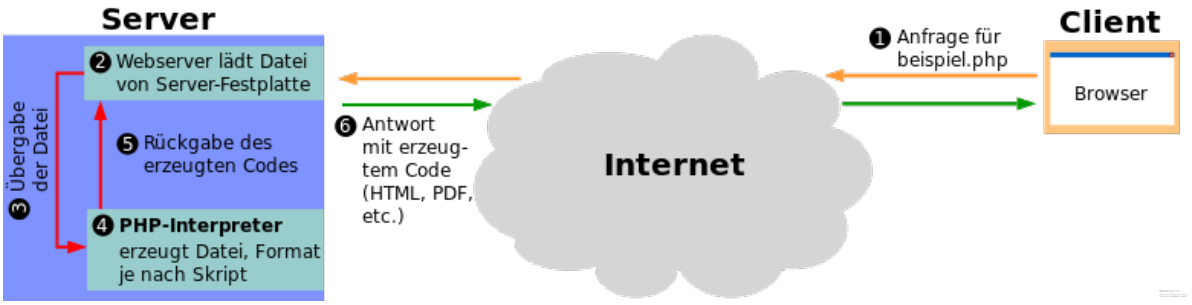


Abbildung 5.1.: Funktionsweise von *PHP*, Quelle:[Incb]

Cordova

Cordova, ehemals PhoneGap, ist ein Framework von Apache [Foua], um native Applikationen für mobile Betriebssysteme mit Web-Technologien, wie *HTML*, *CSS* oder JavaScript, zu entwickeln. Die entstandenen Web-Projekte kann Cordova für alle gängigen mobilen Betriebssysteme (Abbildung 3.2) kompilieren, worin der große Vorteil des Frameworks liegt. Bei der Entwicklung der *App* für die *ANC-BWN* wurde Cordova verwendet, um diesen großen Vorteil auszunutzen und eine getrennte Entwicklung der *App* zu verhindern. Cordova kann auf die Hardware des Smartphones mittels JavaScript zugreifen, wofür jedoch die Cordova-Bibliothek eingebunden werden muss. Cordova, als sogenanntes Cross-Framework, eignet sich jedoch nicht für alle Anwendungen, da bei komplexen Szenarien ([PMR13], [PTR10], [PTKR10]) native Applikationen vorzuziehen sind.

jQueryMobile

jQueryMobile ist ein Web-Framework, dass Elemente der Hypertext Markup Language für Touch-Eingaben optimiert und Inhalte für kleine Auflösungen optimiert. Es ermöglicht mobilen Endgeräten Webseiten, die mit jQueryMobile umgesetzt wurden, ohne Einschränkung der eigenen Darstellungsformen zu betrachten. jQueryMobile baut auf den Web-Technologien *HTML*, *CSS* und JavaScript auf und bietet eigene Events und Methoden für die Entwicklung. Die Verwendung dieses Frameworks basiert auf der Entscheidung, Cordova als Applikations-Framework zu verwenden. Der Vorteil der Verwendung entsteht, da jQueryMobile *Widgets* eine Eigenentwicklung der grafischen Grundstruktur für mobile Applikationen übernimmt, die nach Kapitel 3.2.2 gefordert wird.

AJAX

AJAX ist eine hybride Form der Programmierung, um Inhalte im World Wide Web dynamisch und asynchron nachzuladen. Dafür führt es Requests mithilfe des *HTTP*-Protokolls durch und kann das Ergebnis mittels JavaScript innerhalb eines *HTML*-Dokumentes platzieren, ohne das Dokument dafür neu zu laden. Diese Technologie kommt in der *mobile App* zum Einsatz, da Inhalte dynamisch über die *RESTful API* abgefragt und in bestehende Seiten eingefügt werden sollen. Dies unterstützt die Benutzerfreundlichkeit und hilft die Anforderung in Kapitel 3.2.2, die entstehende *App* einer nativen Applikation für mobile Betriebssysteme anzugleichen, zu erfüllen.

Handlebars.JS

Handlebars.JS ist ein in JavaScript geschriebener Kompiler für Mustache- und Handlebar-Templates. Eine Besonderheit an Handlebars.JS ist, dass das Framework semantische Logik, wie *if*, *unless* oder *each*, interpretieren kann und zusätzlich, mittels Helfer-Funktionen, Daten vor dem logischen Auswerten manipulieren kann. Innerhalb von logischen Blöcken kann *HTML*-Markup platziert werden. Ein großer Vorteil bei der Anwendung sind die logischen Operatoren, die ohne zusätzlichen JavaScript-Code eine flexible Ausgabe ermöglichen. So kann ein einziges Template, aufgrund von Nutzerattributen, mehrere Sichtbarkeiten generieren, ohne eine Zeile JavaScript-Code auszuführen. Des Weiteren bleibt mittels dieser Methodik das *HTML*-Markup und somit die Benutzeroberfläche, stets konstant und kann, für alle Sichtbarkeiten, an nur einer Stelle entwickelt werden.

fastclick

Die Bibliothek fastclick verkürzt die Zeit eines Browsers, die er wartet, bis er eine Berührung des Touch-Displays als gewollte Eingabe erkennt. Eine Verwendung ist sinnvoll, wenn die gewünschte Interaktionszeit verkürzt werden soll, so wie es bei nativen *Apps* der Fall ist. Nach Kapitel 3.2.2 soll die *mobile App* für die *ANC-BWN* eine native *App* so perfekt wie möglich simulieren, um den Benutzer die Eingewöhnungszeit zu verringern.

normalizer

Das Framework normalizer ist ein Cascading Style Sheet (*CSS*), das zu Beginn des Projektes eingebunden wird. Dort führt sie für alle *HTML*-Elemente eine Standard-Stildefinition ein. Dies ist elementar, da jeder Browser, so auch die Webview-Komponente innerhalb der mit Cordova erzeugten *App*, eine Standard-Stildefinition für *HTML*-Elemente besitzt und dies, bei späterer Erweiterung der Stildefinitionen, zu Inkonsistenzen der Anzeige zwischen den mobilen Betriebssystemen führt.

5.2. Strukturelle Vereinbarungen

In diesem Kapitel werden syntaktische und semantische Vereinbarungen getroffen, welche den zwei autonomen Systemen die Kommunikation ermöglicht. Dabei ist zu unterscheiden, ob öffentlicher oder nicht öffentlicher Inhalt abgefragt wird, sowie ob Änderungen an bestehenden Inhalten vorgenommen werden.

5.2.1. Codierung des Inhaltes

Das Ergebnis der Datenbank, das *HTML*-Markup innerhalb der mobilen Applikation und die Antwort des Servers ist stets in *UTF-8* codiert.

5.2.2. Identifikationsnummern für Aktionen

Um die Teilung öffentlicher und nicht öffentlicher Bereiche zu konkretisieren, werden beide Bereiche durch natürliche Zahlen eines unterschiedlichen Intervalls identifiziert. Diese eindeutigen Zahlen sind die Identifikationsnummer einer Action, welche eindeutig und gleichzeitg ihr Dateiname ist. Diese ID wird als *URL*-Parameter *id* dem Skript übergeben.

Intervall	Beschreibung
01 bis 99	Eine Nummer aus diesem Intervall identifiziert eine öffentliche Anfrage.
100	Dieser Sonderfall für nicht öffentliche Anfragen erlaubt es dem Nutzer, mittels Username-/Passwort-Kombination, seine eigene UID im System zu erfahren.
101 bis 299	Nicht öffentliche Anfragen werden mithilfe von Nummern dieses Intervalls identifiziert. Für gewöhnlich fallen hier mehrere Anfragen an, weshalb dieser Intervall größer ist!

Tabelle 5.1.: Zuteilung von Intervallen für Identifikationsnummern von Aktionen

5.2.3. Öffentliche Anfragen

Der Nutzer versendet öffentliche Anfragen mithilfe der *HTTP*-Methode *GET* (Listing 5.1). Es ist anzumerken, dass diese Anfrage ausdrücklich nur für öffentlich sichtbare Bereiche ausgewertet wird, da damit eine sichere Übertragung von Nutzerdaten nicht möglich ist. Anhand des *URL*-Parameters *id* kann die *API* entscheiden, welche *Action* ausgeführt, sprich welche Daten abgefragt werden soll. Falls mit der aufgerufenen *Action* ein bestimmter Inhalt abgefragt werden soll, so muss zusätzlich der *URL*-Parameter *uid* mit einer natürlichen Zahl angehängt werden. Ist dieser nicht gesetzt, so werden alle Einzelabfragen nacheinander in einem Array ausgegeben.

```
1 GET /client.php?id=1&uid=13 HTTP/1.1
2 Host: www.anc-bwn.de
```

Listing 5.1: Beispiel einer parametrisierten und öffentlichen Anfrage

5.2.4. Zugriffsbeschränkte Anfragen

Der Nutzer versendet eine zugriffsbeschränkte Anfrage mithilfe der *HTTP*-Methode *POST* (Listing 5.2). Dabei sendet er seine Nutzerdaten und gegebenenfalls aktionsspezifische Daten mit dem Parameter *data* des *HTTP-POST*-Requests mit. Bei der Abfrage eines bestimmten Inhaltes muss beispielsweise eine *uid* im *data*-Paramenter enthalten sein. Die Daten werden dabei als *JSON*-String enkodiert. Die benötige *Action-ID* wird mittels *URL*-Parameter *id* abgeschickt. Damit kann eine eindeutige Zuordnung der Aktion durchgeführt werden. Die benötigten Daten werden in den *Action*-Skripts im Ordner „actions" definiert. Werden benötigte Daten nicht vollständig mitgeliefert, wird ein Fehler an die Applikation gesendet. Daten werden grundsätzlich im *JSON*-Format übertragen und nur in diesem Format von der *API* decodiert. Dadurch ist die *API* in der Lage, der gewünschten Aktion ein Ziel zuzuordnen, eine Action zu laden sowie auszuführen und den Zugriff gleichzeitig zu autorisieren. Der Benutzer muss sich bei jeder Anfrage mit seinen Nutzerdaten autorisieren, da auf eine Session, aus Gründen der Konvertierung in eine native Applikation, verzichtet wird.

```
1 POST /client.php?id=101 HTTP/1.1
2 Host: www.anc-bwn.de
3 data={"username":"hans123","password":"passwort","uid":13,
4       "hidden":1,"title":"Dies ist der Titel!","image":"",
5       "short":"Das ist ein kurzes ein Beispiel.",
6       "bodytext":"Dies ist ist die Langversion eines Beispieltextes."}
```

Listing 5.2: Beispiel einer parametrisierten und nicht öffentlichen Anfrage

5.2.5. Antwort der API

Die Antwort der *API* an den Client ist stets ein valider *JSON*-String, welcher die Antwort beinhaltet. Eine Antwort kann nur als Erfolg (*success*) oder Fehler (*fail*) ausgegeben werden. Dies wird im *JSON*-String mit dem Attribut *response* übergeben, um den Client zu informieren, ob die Anfrage erfolgreich war. Die Vielzahl der möglichen Antwortszenarien, welche mit oder ohne Daten erfolgen können, benötigen eine Kodierung, mit welcher der Client die Antwort verstehen kann. Dazu wird eine Auswahl der *HTTP*-Statuscodes (Tabelle 5.2) verwendet. Allein der Statuscode *200* liefert eine, mit Inhalt erweiterte, Antwort, welche nach dem *response*-Attribut ein einzelnes Object *oder* einen *numerischen* Array beinhaltet. Es ist ein numerischer Array, falls mehrere Datensätze zurückgegeben werden müssen.

Status	Beschreibung
200	Die Anfrage wurde erfolgreich bearbeitet. Es werden zusätzliche Daten mitgeliefert.
204	Die Anfrage wurde erfolgreich bearbeitet. Keine Daten mitgeliefert.
400	Die Anfrage war wegen struktureller Mängel fehlerhaft. Es fehlen Parameter.
401	Der Nutzer ist nicht autorisiert für diese Anfrage.
404	Die notwendigen Daten konnten nicht abgefragt werden.
423	Die Anfrage konnte nicht bearbeitet werden, da die Ressource gesperrt ist.
500	Interner Server Fehler für alle nicht definierten Fehler.
501	Die Anfrage ruft eine nicht implementierte Funktion auf.

Tabelle 5.2.: Für den Entwicklungskontext verwendete Statuscodes [Incg]

Beispiel eines fehlerhaften Antwortszenarios (fail)

Ein fehlerhaftes Antwortszenario (Listing 5.3) wäre in diesem Beispiel eine nicht autorisierte Anfrage an das System. Das bedeutet, dass entweder die Nutzerdaten nicht übereinstimmen oder gar keine mitgeliefert wurden. Außerdem tritt dieser Fehler auf, wenn eine nicht öffentliche Anfrage mittels der *HTTP*-Methode *GET* angefragt wird.

```
1 {"response": 401}
```

Listing 5.3: Antwort der API auf eine fehlerhafte Anfrage

Beispiel eines erfolgreichen Antwortszenarios (success)

Ein erfolgreiches Antwortszenario (Listing 5.4) wurde durch eine korrekt gestellte Anfrage erstellt. Dabei wurde der Statuscode *200* ausgegeben und deshalb auch zusätzliche Daten. Hierbei handelt es sich als um die Abfrage mehrerer Datensätze, welche in dem Array *data* nacheinander aufgereiht werden.

```
{
  "response": 200,
  "data": [
    {
      "uid": "1",
      "crdate": "1251577053",
      "cruser_id": "1",
      "title": "Die neue Webseite ist fertig",
      "datetime": "1307045760",
      "short": "Die neue Website ist ab jetzt online",
      "author": "Michael",
      "category": "1"
    },
    {
      "uid": "6",
      "crdate": "1319551826",
      "cruser_id": "1",
      "title": "Neuigkeiten per RSS-Feed abrufbar",
      "datetime": "1335276600",
      "short": "News-Feed ist online",
      "author": "Sebastian",
      "category": "0"
    }
  ]
}
```

Listing 5.4: Antwort der API auf eine korrekte Anfrage

5.3. RESTful API

In diesem Kapitel wird die *RESTful API*, welche anhand der spezifizierten Anforderungen aus Kapitel 3 und der Architektur aus Kapitel 4.2.1 entwickelt wurde, erläutert. Um einen Einblick in die Implementierung der Architektur zu erhalten, wird in Kapitel 5.3.1 das Datenmodell der *API* vorgestellt. Die Beschreibung der einzelnen Klassen und deren Abhängigkeiten innerhalb der *API* wird zusätzlich in der Tabelle 5.3 dargestellt, die auf das Datenmodell folgt. Das Kapitel 5.3.2 beschreibt daraufhin den Ablauf für verschiedenen Anwendungsfälle auf dem vorgestellten Datenmodell und visualisiert dabei den Datenfluss bei einer eingehenden Anfrage.

5.3.1. Datenmodell

Die Architektur der *RESTful API* wurde objektorientiert konzipiert. Dadurch können Module gekapselt und die Sichtbarkeit von Attributen eingeschränkt werden. Aus dem Entwurf der Architektur in Kapitel 4.2.1 kann man bereits die fünf wichtigen Klassen der *API* erkennen. Zusätzlich wird bei der Implementierung eine Klasse für die Aktionen, welche der *ActionManager* verwaltet, eingeführt (Abbildung 5.2). Bei der Entwicklung wurde darauf geachtet, dass eine Aktion der Klasse *Action* für alle Datenbankoperationen geeignet ist und dass sie die Anforderungen einer robusten Architektur erfüllt. Deshalb besitzt sie Felder, in der sie nicht übertragene Datenfelder automatisch füllen kann, um die Fehlertoleranz der *API* zu erhöhen. Des Weiteren wurden Klassen für die Antwort des Systems gebildet. Die Klasse *Response* speichert nur einen Statuscode (Kapitel 5.2.5), der dem Request-Steller Auskunft über die Verarbeitung geben soll. Für Anfragen, die mehr als einen Statuscode als Antwort besitzen, erbt eine weitere Klasse *DataResponse* von der Klasse *Response*, um diese mit einem Datenfeld zu erweitern. Dieses Datenfeld ist mit den angefragten Informationen aus der Datenbank gefüllt und kann nur mit dem Statuscode *200* auftreten. Eine Antwort kann jederzeit ausgeben werden, um die Robustheit im System zu erhöhen. Dadurch wird sichergestellt, dass zu jeder Zeit eine valide Antwort ausgegeben werden kann.

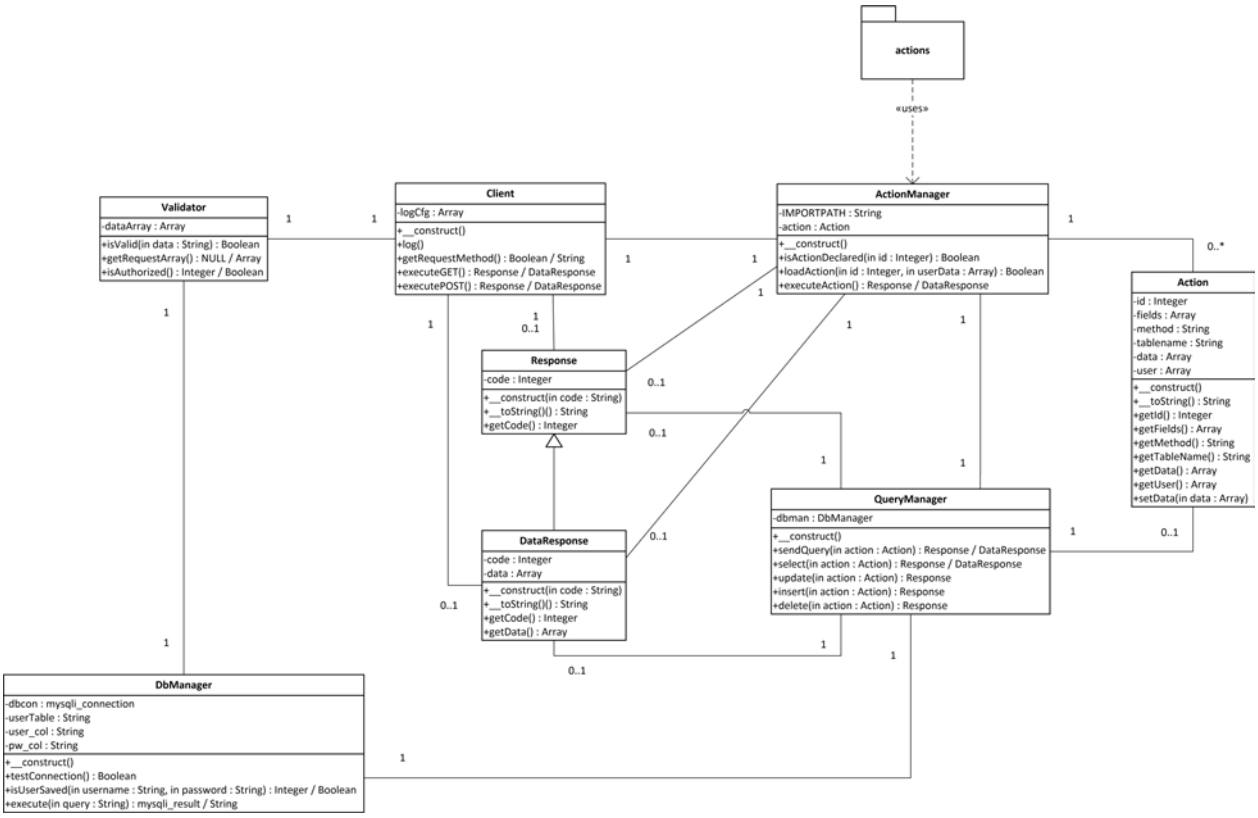


Abbildung 5.2.: Klassendiagramm der *RESTful API*

Klasse	Beschreibung
Client	Ist der Ansprechpartner und das Bindeglied zwischen *App* und Server. Diese Klasse loggt jeden Zugriff, unterscheidet auf *GET*- und *POST*-Anfragen und übermittelt den dadurch involvierten Klassen die nötigen Daten. Zuerst leitet die Klasse den *JSON*-String an den *Validator* und gibt entweder eine *Response* aus oder leitet den validierten Array an den *ActionManager* weiter. Danach wird die erstelle *Action* an den *QueryManager* ausgeliefert und von diesem bekommt er letztendlich auch die *Response*, respektive die *DataResponse* und gibt sie als Systemantwort aus.
Action	In dieser Klasse werden Attribute und Methoden bereitgestellt, aus denen ein *Action*-Objekt erstellt werden kann. Ein *Action*-Objekt beinhaltet im besten Fall alle benötigt Attribute, um eine Datenbankoperation ausführen zu können. Eine *ID* dient zu eindeutigen Identifizierung und Namensgebung für alle *Action*-Objekte. Es werden im Ordner „actions" alle definierten *Action*-Objekte als Skripte gespeichert und bei Anfrage vom *ActionManager* gelesen.
Response	Dies ist die Klasse zur Generierung von Antworten, welche nur einen Fehlercode enthält. Eine *Response* gibt sich selbst in JavaScript Object Notation aus.
DataResponse	Die *DataResponse*-Klasse erbt von der *Response*-Klasse und kann zusätzlich Daten aufnehmen und wiederum in JavaScript Object Notation ausgeben.
Validator	Diese Klasse prüft den Request auf Korrektheit, Vollständigkeit und gegebenenfalls Autorisierung. Falls der Request valide ist, so kann ein *Validator*-Objekt den *JSON*-String als assoziatives Array zurückgeben. Ansonsten wird eine *Response* zurückgeben.
ActionManager	Diese Klasse stellt Methoden bereit, um eine *Action* aus dem *Action*-Ordner zu laden - falls definiert - und diese auszuführen. Dabei wird ein *QueryManager* erstellt und versucht, die *Action* mit den bereitgestellten Daten auszuführen. Gelingt dies nicht, so wird eine *Response* zurückgegeben.
QueryManager	Dies ist die Klasse zur Prüfung einer *Action* auf Vollständigkeit und gegebenfalls Erstellung einer passenden Query. Danach wird der *QueryManager* diese Query an den *DbManager* senden und aus der Antwort eine *Response*/*DataResponse* verfassen.
DbManager	Diese Klasse beinhaltet die Verbindung zur Datenbank, sowie Methoden zur Bestimmung eines Vorkommens von Usern und ausführung von Queries. Falls die Verbindung nicht hergestellt werden kann, so beendet der *DbManager* die Ausführung und gibt eine passende *Response* aus.

Tabelle 5.3.: Beschreibung von Aufgaben einzelner Klassen innerhalb der *RESTful API*

5.3.2. Kommunikationsmodell

Um eine korrekte und robuste Abarbeitung eines Requests zu ermöglichen, werden die Abläufe innerhalb der *RESTful API* für jeden Anwendungsfall, der sich aus der Rollenverteilung (Kapitel 3.1.3) ergibt, geplant. Öffentliche Anfragen werden grundsätzlich mit der *HTTP*-Methode *GET* empfangen. Zugriffsbeschränkte Anfragen hingegen nutzen die *HTTP*-Methode *POST*. Die unterschiedlichen Methoden werden in Kapitel 5.2.3 und Kapitel 5.2.4 detailliert dargestellt. Bei der Auswertung einer Anfrage können mehrere Probleme auftreten, welche mit einer *Response* - ohne mitgelieferte Daten - beantwortet werden. In den folgenden Anwendungsfällen werden die hauptsächlichen Fehlerquellen dargestellt. Dabei ist zu beachten, dass eine syntaktisch falsch gestellte Anfrage nicht berücksichtigt wird, da diese standardmäßig durch den *Validator* abgelehnt wird.

Anwendungsfall: öffentliche und erfolgreiche Anfrage

Bei einer öffentlichen und erfolgreich ausgeführen Anfrage wird zuerst die Anfrage vom *Validator* geprüft. Danach wird der *JSON*-String in ein assoziatives Array umgewandelt, welches an den *ActionManager* weitergereicht wird. Dieser prüft, ob eine passende vorgefertige Aktion existiert und erzeugt diese gegebenenfalls. Danach wird im *QueryManager* aus der *Action* ein Query-String erzeugt, welcher im *DbManager* ausgeführt wird. Ein entsprechendes Antwort-Objekt wird als *DataResponse*, mit Daten angereicherte Antwort, zur Ausgabe zurückgereicht.

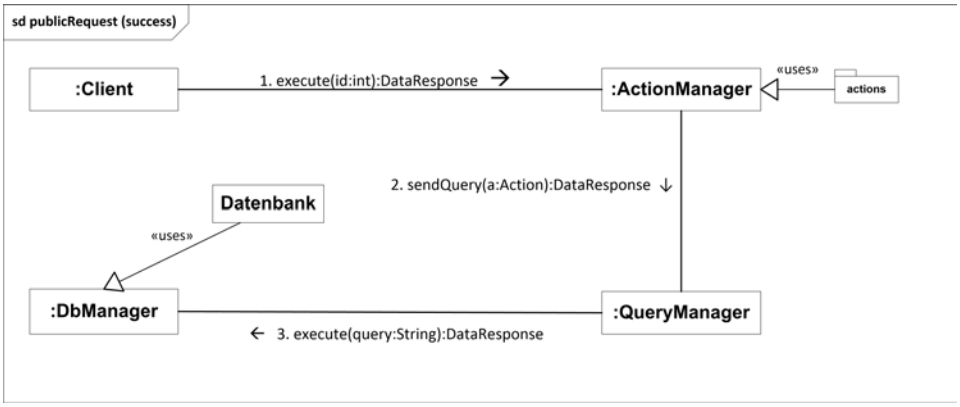


Abbildung 5.3.: Öffentliche und erfolgreiche Anfrage

Anwendungsfall: zugriffsbeschränkte und erfolgreiche Anfrage

Bei einer zugriffsbeschränkten und erfolgreich ausgeführten Anfrage wird zuerst die Anfrage vom *Validator* geprüft und die mitgelieferten Nutzerdaten in der Datenbank abgefragt. Falls diese existieren, wird ein Array aus den mitgelieferten Daten und der *ID* des Nutzers erstellt.

Ab diesem Punkt ist der weitere Ablauf identisch zu der öffentlichen und erfolgreichen Anfrage.

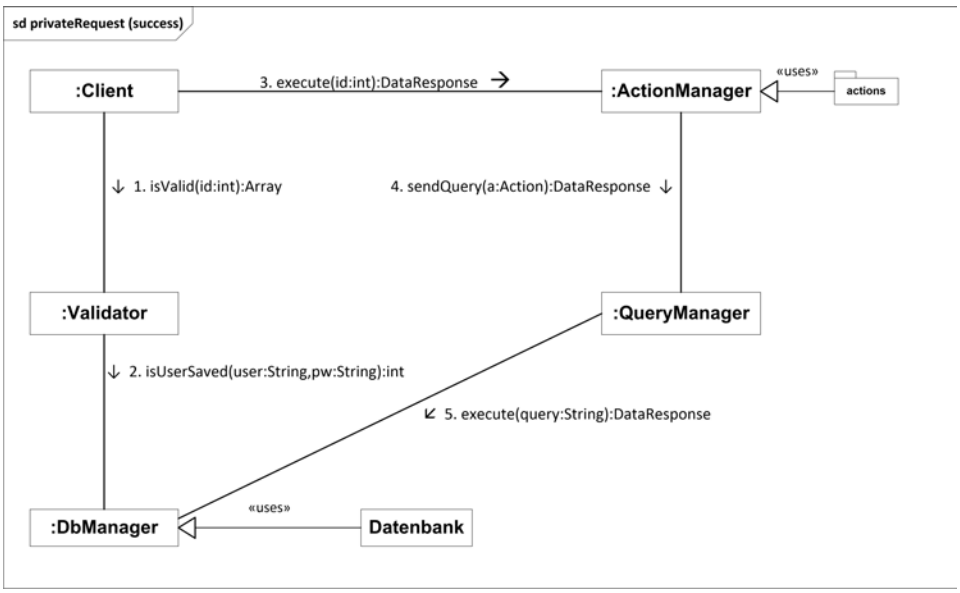


Abbildung 5.4.: Zugriffsbeschränkte und erfolgreiche Anfrage

Anwendungsfall: öffentliche und fehlgeschlagene Anfragen

Öffentliche Anfragen werden in der Regel an zwei Orten einen Fehler produzieren und dadurch als fehlgeschlagen gelten. Die erste Fehlerquelle kann bei einer fehlerhaften Anfrage der *ActionManager* sein. Falls die Anfrage eine nicht definierte *Action* aufrufen möchte, so wird dieses Objekt einen Fehler werfen (Abbildung 5.6). Die andere Fehlerquelle kann der *QueryManager* sein. Falls die *Action* falsch deklariert wurde, wird hier ein Fehler geworfen (Abbildung 5.5).

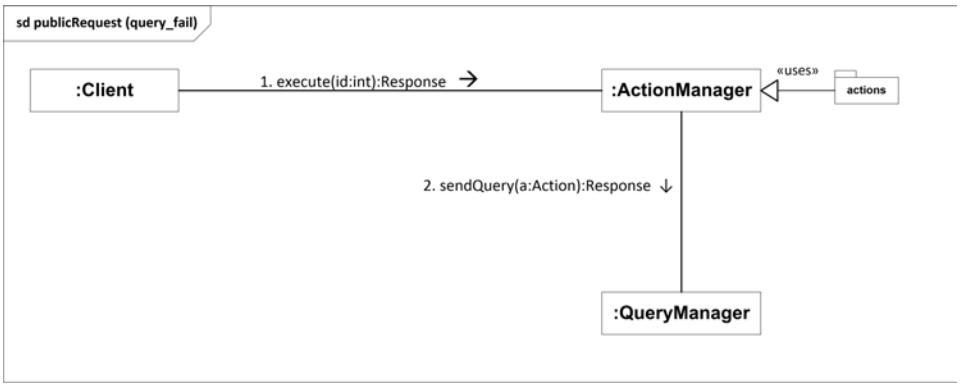


Abbildung 5.5.: Öffentliche und wegen falscher Deklaration fehlgeschlagene Anfrage

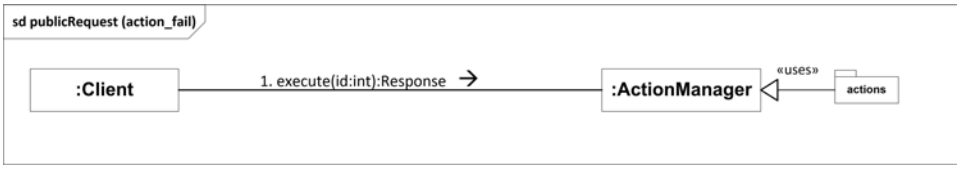


Abbildung 5.6.: Öffentliche und wegen nicht definiertem Aufruf fehlgeschlagene Anfrage

Anwendungsfall: zugriffsbeschränkte und fehlgeschlagene Anfragen

Bei zugriffsbeschränkten Anfragen können mehrere Probleme auftreten, wodurch die Anfrage fehlschlägt. Es kann dabei zu gleichen Problemen kommen wie in Abbildung 5.6, da auch im validiertem Array eine nicht definierte *Action* übergeben werden kann. Zusätzlich gibt es noch die Möglichkeit, dass ein User eine definierte *Action* aufruft, für die er keine Berechtigung hat. In diesem Fall wird der *ActionManager* ebenfalls einen Fehlercode als *Response* ausliefern.

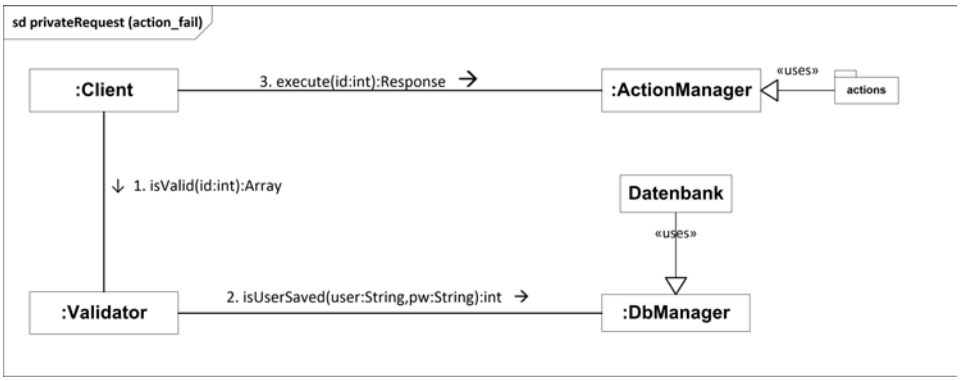


Abbildung 5.7.: Zugriffsbeschränkte und fehlgeschlagene Anfrage

Einen größeren Unterschied gibt es jedoch bei der Bearbeitung einer *Action* und deren Abfrage im *QueryManager*. Dieser kann zusätzlich zu bekannten Fehlern (Abbildung 5.5) auch noch einen weiteren Fehler abfangen. Falls die Anfrage nicht alle geforderten Parameter mitliefert, wird der *QueryManager* den dazu passenden Fehlercode ausliefern. Dazu prüft der *QueryManager* die, in der *Action* geforderten Pflichtfelder, mit den Parametern der Anfrage ab.

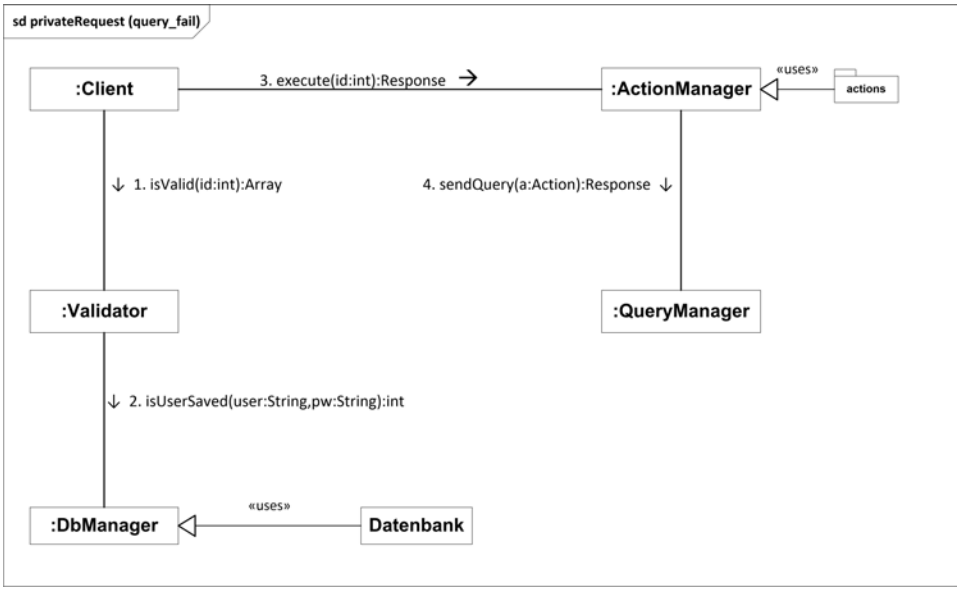


Abbildung 5.8.: Zugriffsbeschränkte und falsch parametrisierte, fehlgeschlagene Anfrage

Ein weiterer Fehler kann bei der zugriffsbeschränkten Anfrage die Lieferung falscher Benutzerdaten sein. Diese werden schon im *Validator* abgeglichen und so kann frühzeitig ein Fehler geworfen werden.

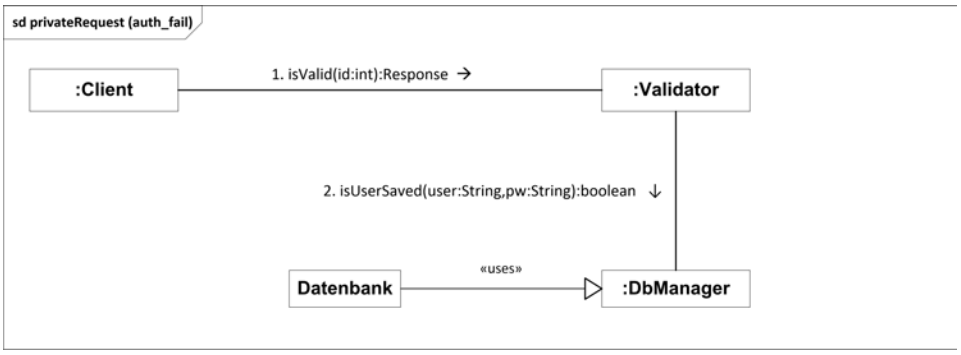


Abbildung 5.9.: Zugriffsbeschränkte und nicht authorisierte, fehlgeschlagene Anfrage

5.4. Mobile Applikation

Dieses Kapitel widmet sich der Entwicklung der mobilen Applikation. Als Basis dienen die Anforderungen aus Kapitel 3 und die entwickelte Architektur aus Kapitel 4.2.2. Als Mantel-Framework, zur Erstellung nativer Apps mittels Webtechnologien, wurde das Cordova-Framework (Kapitel 5.1) gewählt. Dabei liegt der Schwerpunkt auf der Veröffentlichung für Android und iOS, welche aktuell und in naher Zukunft die verbreitetsten mobilen Betriebssysteme sind (Abbildung 3.2). Der Aufbau einer Applikation mit dem Cordova-Framework

wird im Kapitel 5.4.1 detailliert beschrieben. Um das Look-And-Feel einer Applikation mit Webtechnologien zu simulieren, wurde das jQueryMobile-Framework (Kapitel 5.1) verwendet, welches eigene *Widgets* besitzt und im Responsive Design aufgebaut ist. Das Schema einer jQueryMobile-Seite wird im Kapitel 5.4.2 vorgestellt. Um variable Inhalte aus dem *CMS* in eine vordefinierte Umgebung einzubetten, wurde das Templating-Framework Handlebars.JS (Kapitel 5.1) verwendet. Wie das Einbinden von Handlebars-Templates in der *App* der *ANC-BWN* realisiert wurde, wird im Kapitel 5.4.3 beschrieben.

5.4.1. Aufbau einer Applikation mit Cordova

Um eine native Applikation mit dem Cordova-Framework zu erstellen, muss die Cordova-Bibliothek in eine jQueryMobile-Seite (Kapitel 5.4.2) eingebunden werden. Danach kann auf die implementierten Plattform-Plugins (Abbildung 5.10) zugegriffen werden. Aufgrund eines Paradigmenwechsels in der Cordova-Plugin-Strategie, der mit der Version 3.0 vollzogen wurde, werden als Plattform-Plugins nur noch die Plugins mit der wichtigsten Hardwareunterstützung eingeschlossen. Möchte man weitere Handware verwenden, so kann man Cordova mit Plugins erweitern (Abbildung 5.10). Somit bleiben Ressourcen, die von nicht genutzten Plugins verwendet würden, frei.

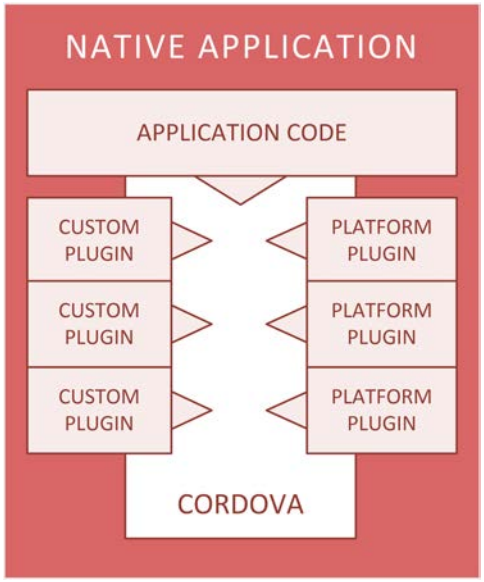


Abbildung 5.10.: Aufbau einer nativen Applikation mit dem Cordova-Framework

Zusätzlich kann über eigenen Programmcode auf Plugins zugegriffen werden. Dies geschieht, für erweiternde oder Plattform-Plugins gleichermaßen, über das Cordova-Objekt, welches über die Cordova-Bibliothek erzeugt wurde. Cordova bietet somit eine Schnittstelle zwischen Plugins, die den Zugriff auf die Hardware bieten und eigenem Programmcode, mit dem Benutzeraktionen erfasst und Inhalte geändert werden.

5.4.2. Aufbau einer Seite mit jQueryMobile

Eine jQueryMobile-Seite (Abbildung 5.11) wird mittels Hypertext Markup Language beschrieben. Zu Beginn stehen Meta-Deklarationen, die benötigt werden, um die Seite an den Viewport des mobilen Endgerätes anzupassen. Dabei wird unter anderem die Breite und Höhe eines mobilen Endgerätes verwendet, um die Maximallänge von *100%* in Pixel umzurechnen. Darauf folgen die Einbindungen der Cascading Style Sheets, die das Layout der *App* bestimmen. Falls man Symbole und Layoutstrukturen von jQueryMobile verwenden möchte, so muss man das mitgelieferte Stylesheet einbinden. Es ist darauf zu achten, dass eigene Stylesheets nach dem jQueryMobile-Stylesheet eingebunden werden. Anschließend können die JavaScript-Importe eingefügt werden. Anders als bei dem Stylesheet-Import, muss die jQueryMobile-Bibliothek explizit nach den eigenen Script-Dateien eingefügt werden.

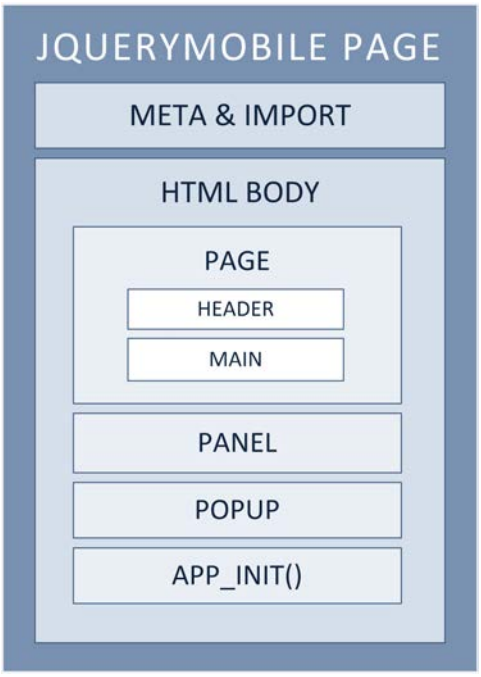


Abbildung 5.11.: Aufbau einer Seite mit dem jQueryMobile-Framework

Falls die JavaScript-Bibliothek und das Cascading Style Sheet von jQuery-Mobile eingebunden wurde, so kann nun um *Body* des *HTML*-Dokumentes ein *Div*-Container erstellt werden, der das *data-role*-Attribut *Page* trägt. Damit erkennt die jQueryMobile-Bibliothek, dass nun ein vordefiniertes *Widget* beschrieben wird. Ein Aufruf der Seite rastert nun ein vorgegebenes Layout für eine Seite. In einer Seite können weitere *Div*-Container platziert werden, die beispielsweise das *data-role*-Attribute *Header* oder *Main* tragen können (Abbildung 5.11). Der *Header* erstellt dabei eine Menüleiste am oberen Bildschirmrand, während in der *Main* der Seiteninhalt beschrieben wird. Zusätzlich zur Page können weitere *Widgets* auf die selbe Art beschrieben werden, wie beispielsweise ein *Panel* oder ein *Popup*. In der Dokumentation ist weiterhin beschrieben, dass eine Applikationsinitialisierung zum Schluss, nachdem alle *Widgets* in den *DOM* geladen wurden, empfohlen wird.

5.4.3. Einbinden von Templates

Um vordefinierte Felder mit Inhalten aus dem *CMS* zu füllen, wird das Templating-Framework Handlebars.JS verwendet. Dadurch können Informationen eines *JSON*-Objektes mit einem vordefinierte Template kompiliert werden. Eine Besonderheit von Handlebars.JS ist, dass auch semantische Operatoren eingefügt wurden. Im Rahmen der Entscheidung für die Verwendung von Handlebars.JS in dieser Bachelorarbeit wurde ein Einblick in die Thematik mit einer beispielhaften Anwendung für den Betreuer verfasst. Die Definition der wichtigsten Ausdrücke, eine Einführung in das Erzeugen von Templates mit Handlebars.JS und ein Beispielprojekt können aus dem Anhang, beziehungsweise dem beigelegten Datenträger, entnommen werden.

5.5. Aspekte der Implementierung

Nachdem beide Systembestandteile vorgestellt wurden, werden in diesem Kapitel besondere Aspekte der Implementierung vertiefend dargestellt. In Kapitel 5.5.1 wird der Seiteninitialisierungsprozess für Templates mit Online- oder Offline-Inhalten vorgestellt, der die Funktionen und Syntax des jQueryMobile-Frameworks erweitert.

5.5.1. Erweiterte Seiteninitialisierung

Um Seiten, die als Container für Inhalte dienen, zu wechseln, wird in jQueryMobile die Funktion *changePage* aufgerufen. Diese bekommt bei Aufruf eines internen Verweises vom Framework mehrere Parameter mitgeliefert, um das Ziel, die Herkunft des Aufrufes und erweiternde Aktionsoptionen zu erfahren. Ziel- und Herkunftsangaben werden mit einem absoluten Pfad zur Datei angegeben, da jQueryMobile von einem statischen Aufruf der Inhalte ausgeht. Möchte man nun eine template-generierte Seite aufrufen, so müsste jedes Template den kompletten Aufbau einer jQueryMobile-Seite (Abbildung 5.11) besitzen. Da einzelne Grundbausteine der *App* (Kapitel 6.1), wie beispielsweise der *Header*, nicht vom Inhalt des Templates abhängig und eigenständig verwaltet werden sollen, muss die *changePage*-Methode um eine Injektion von Template-Inhalten in existierende Seiten erweitert werden. Um den *Header* dabei automatisch neu zu erstellen, wird auf ein Event gehorcht, dass bei der Erstellung der Seite geworfen wird. Wird das Event geworfen, injiziert eine Funktion den zur Nutzergruppe passenden *Header* in die entstehende Seite.

Aufgrund der Konstruktion eines Templates, kann es gleichzeitig einen bestimmten oder beliebig viele Datensätze aus der *RESTful API* nach demselben Schema erstellen. Deshalb muss diese Information beim Aufruf mitgeliefert werden, sodass ein Template die richtige Anzahl an Datensätze erhält. Zusätzlich soll es eine Erweiterungsfähigkeit geben, die über das Ziel der Template-Injizierung bestimmen kann. Deshalb wurde ein eigenes Schema für die erweiterte Seiteninitialisierung (Abbildung 5.12) entwickelt, mit dem Template-Aufrufe spezifiziert und behandelt werden können.

Syntax der Zielangabe: `templateName?uid#container`

Die erste Angabe der Syntax ist die Pflichtangabe und verlangt den Name des Templates, welcher in einer externen *JSON*-Datei mit den benötigten *URLs* der *RESTful API* verknüpft ist. Die Datei *MAP.json* (Listing 5.5) ist nach den existierenden Nutzergruppen strukturiert und regelt somit zusätzlich die Autorisierung auf Template-Zugriffe und ob diese Online-Inhalte abrufen dürfen. Mit der optionalen Syntax-Angabe der *uid* kann ein bestimmter Datensatz ausgewählt werden. Wird die *uid* nicht gesetzt, so werden alle Datensätze zurückgegeben. Dabei wird die Struktur der *RESTful API* genutzt, die bei fehlender *uid*-Angabe alle Datensätze als Array ausgibt. Bei Aufruf der *changePage*-Methode wird nun, statt des absoluten Pfades, die Zielangabe mit der eigens entwickelten Syntax übergeben. Da dabei Aktionsoptionen verloren gehen können, werden diese durch explizites Angeben, beziehungsweise einer globalen Default-Einstellung, ersetzt. Es muss jedoch für interne Framework-Aufrufe möglich sein, die alte originale *changePage*-Methode mit absoluter Pfadangabe zu verwenden. Deshalb wurde diese zwischengespeichert, um sie bei der Angabe eines absoluten Pfades zu verwenden.

```
1  "member": {
2
3      "news": {
4
5          "name": "Neuigkeiten",
6          "online": true,
7          "url": {
8              "get": "http://www.anc-bwn.de/api/Client.php?id=103"
9          }
10     },
11
12     "news-single": {
13
14         "name": "Neuigkeit",
15         "online": true,
16         "url": {
17             "get": "http://www.anc-bwn.de/api/Client.php?id=104",
18             "insert": "http://www.anc-bwn.de/api/Client.php?id=105",
19             "update": {
20                 "editlock": "http://www.anc-bwn.de/api/Client.php?id=106",
21                 "data": "http://www.anc-bwn.de/api/Client.php?id=107",
22                 "delete": "http://www.anc-bwn.de/api/Client.php?id=108"
23             }
24         }
25     }
26
27 }
```

Listing 5.5: Ausschnitt der Zugriffsverwaltung MAP.json

Der Ablauf einer Seiteninitialisierung (Abbildung 5.12) beginnt mit dem Zwischenspeichern der originalen und Überschreiben der eigenen *changePage*-Methode. Anschließend wird die Pfad-Angabe überprüft und diese an die passende *changePage*-Methode übergeben. Ist der Seitenaufruf in eigener Syntax verfasst, so wird diese in einer eigenen Funktion in die Bestandteile zerlegt und als Array zurückgegeben. Mit den Informationen aus dem Array und der aktuellen Nutzergruppe kann nun der *MAP.json*, falls dieser dafür autorisiert ist, die *URL* für den Request auf die *RESTful API* entnommen werden. Ist der Benutzer autorisiert, so wird im System nach dem entsprechendem Template gesucht. Existiert dieses, so erfolgt eine weitere Überprüfung, die bestimmt, ob das Template Datensätze aus der *API* zum Erstellen des Inhaltes benötigt oder ohne dargestellt werden kann. Dies ist bei statisch hinterlegtem Inhalt, wie dem *Impressum* (Kapitel 6.3.6), der Fall. Werden Datensätze aus der *API* benötigt, so wird eine Anfrage mittels *AJAX* abgesetzt. Wurden die Datensätze erfolgreich erhalten, so kompiliert das Handlebars-Framework diese mit dem Template-Inhalt. Das Ergebnis wird daraufhin in eine bestehende Seite injiziert und an die originale jQueryMobile-Funktion übergeben, die nun über einen absoluten Pfad auf den erzeugten Inhalt zugreifen kann. Es ist zu beachten, dass jegliche Abweichungen, wie beispielsweise fehlerhafte *AJAX*-Requests, den Abbruch der Seiteninitialisierung und ein *Error*-Feedback (Kapitel 6.2.1) zur Folge haben.

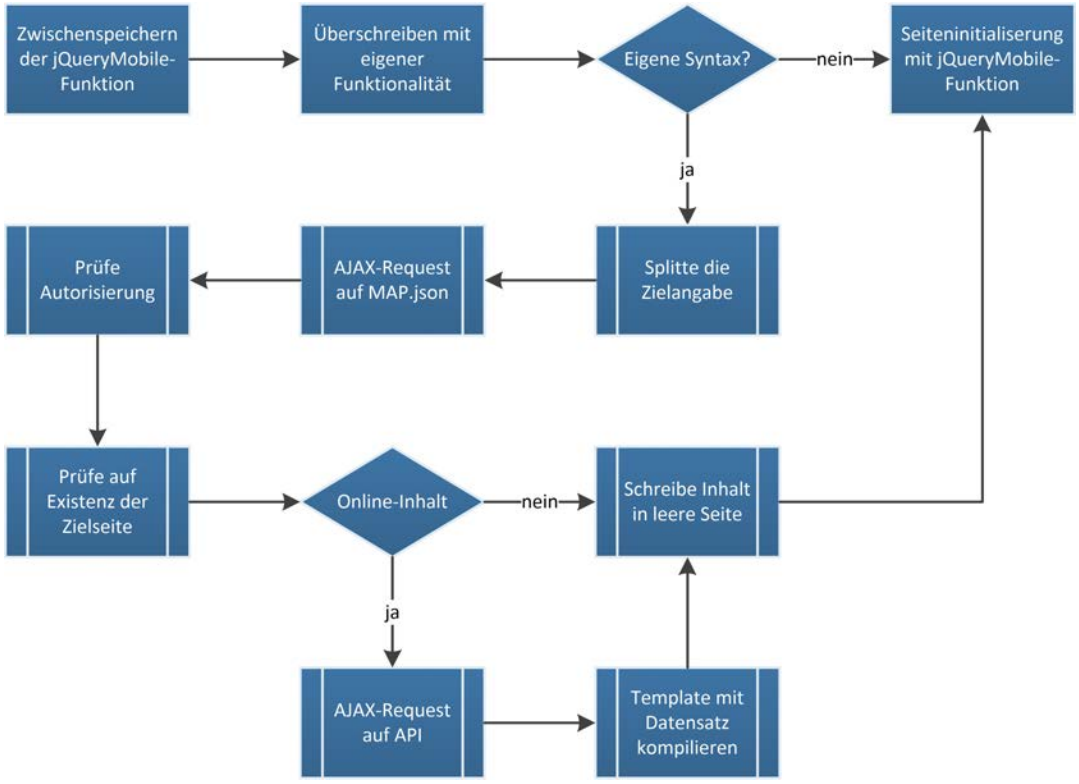


Abbildung 5.12.: Schematischer Ablauf einer Seiteninitialisierung in der mobilen Applikation

6 Walkthrough

In diesem Kapitel wird die *mobile App*, im Hinblick auf Gestaltung und Anforderungsumsetzung, vorgestellt und beschrieben. Die Beschreibung der *API* ist nicht vonnöten, da diese über kein User Interface verfügt. Das Kapitel 6.1 beginnt mit der Beschreibung der *UI*-Grundbausteinen, in der alle weiteren Inhalte eingebettet werden. Anschließend wird in Kapitel 6.2 beschrieben, wie die *mobile App* dem Nutzer Statusmeldungen und Feedback anzeigt. Der öffentliche Bereich der *App* wird darauf folgend in Kapitel 6.3 vorgestellt. Die beinhalteten Funktionen sollen den *ANC-BWN*, gemäß den in Kapitel 3.2 definierten Anforderungen, in der Öffentlichkeit präsentieren. Anschließend wird in Kapitel 6.4 der zugriffsbeschränkte Bereich vorgestellt, welcher registrierten Mitgliedern vorenthalten ist. Dabei verzahnen sich die Funktionen des zugriffsbeschränkten Bereichs in die des öffentlichen Bereichs, um eine konsistente Linie des Gesamtsystems, unabhängig von der Nutzergruppe, zu erreichen.

6.1. Grundbausteine

Die sichtbare *UI* der *mobile App* besteht aus drei Grundbausteinen, welche getrennt voneinander erstellt werden können. Der oberste Abschnitt (Abbildung 6.1, Markierung 1) der *App* bildet der *Header*. Dieser wird bei der *App*- und Seiten-Initialisierung, der Nutzergruppe (Kapitel 3.1.3) entsprechend, automatisch erzeugt und erstreckt sich über die Breite des Bildschirmes. Falls sich die Nutzergruppe, beispielsweise durch Anmeldung beim System, ändert, so wird der *Header* automatisch ausgetauscht. Die auszutauschenden Elemente werden in einer Template-Datei gespeichert und können zur Laufzeit erweitert werden. In jeder Nutzergruppe beinhaltet der *Header* einen *Menü-Button* an der linken oberen Ecke, der durch mehrere Kästchen symbolisiert wird. Dieser öffnet die, der Nutzergruppe entsprechende, Navigation (Abbildung 6.1, Markierung 3). Es wird zusätzlich ein Platzhalter in der linken oberen Ecke erstellt, der für systembeeinträchtigende Benachrichtigungen, beispielsweise einem Hinweis auf fehlende Netzwerkverbindung, reserviert ist.

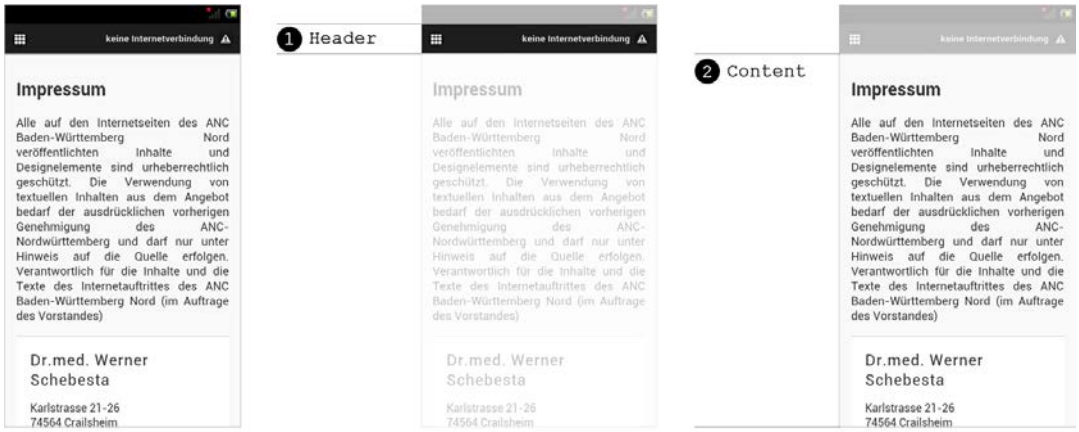


Abbildung 6.1.: Grundstruktur der Applikation

Der zweite Grundbaustein, der *Content*, schließt sich direkt an den *Header* an (Abbildung 6.1, Markierung 2). Dies ist der Platzhalter für alle Benutzerinhalte und Hauptdarstellungsbereich des dynamisch generierten Inhaltes. Der *Content*-Platzhalter besteht aus einer Bezeichnung des aufgerufenen Navigations-Eintrages und einem Bereich für die Darstellung der abgerufenen, respektive definierten, Informationen. Der *Content* erstreckt sich, bis zu einer bestimmten Maximalbreite, über die ganze Bildschirmbreite. Mittels Wisch, respektive Scroll, nach oben kann der nicht sichtbare Inhalt in den Bildschirmbereich bewegt werden. Dabei überlagert der *Header* stets den *Content*-Platzhalter. Zusammen mit dem *Header* bildet der *Content* eine *Page*, die als sichtbare Einheit im jQueryMobile-Framework (Kapitel 5.1) gilt.

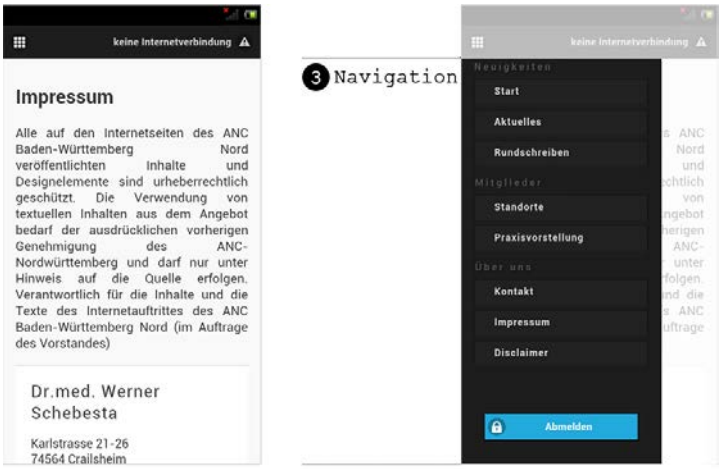


Abbildung 6.2.: Grundstruktur der Applikation (Forts.)

Der dritte Grundbaustein (Abbildung 6.1, Markierung 3) ist die *Navigation*, die mittels *Menü-Button* oder Swipe nach links eingeblendet werden kann. Die *Navigation* wird, in der Z-Ebene, optisch über dem *Content* dargestellt und erstreckt sich dabei nicht über die gesamte Bildschirmbreite. Dies unterstützt die Orientierung der Nutzer, in dem es einen Teil des vom Nutzer zuletzt aufgerufenen *Content* zeigt. Die *Navigation* wird, der Nutzergruppe

entsprechend, initialisiert und unterscheidet sich dabei in den Einträgen. Die Einträge werden in der Navigation in Rubriken eingeteilt, die dem Nutzer bei der Suche von Einträge hilft. Ein Austausch, der in einer Template-Datei gespeicherten Menüeinträge, erfolgt beim Wechsel der Nutzergruppe, welcher durch *anmelden* oder *abmelden* beim System ausgelöst wird. Die *Navigation* ist von der variablen Einheit *Page* stets getrennt und wird auch bei einer Seiteninitiierung nicht geändert.

6.2. Dialoge und Benachrichtigungen

Eine wichtige Regel, zur Verbesserung der Usability, besagt, dass dem Nutzer zu jeder Aktion ein zeitnahes Feedback zurückgegeben werden muss, damit er zu keiner Zeit den Eindruck des Kontrollverlustes erlebt. Die Umsetzung dieser Regel wird deshalb in diesem Kapitel eine Rolle spielen. Im Folgenden werden die wichtigsten Konzepte, wie das System den Nutzer informiert, vorgestellt. Die Überlegung, weniger Feedbackvarianten zu verwenden, lag darin, Varianten zu finden, die für mehrere Anwendungsfälle eingesetzt werden können. Eine große Anwendungsvielfalt einzelner Varianten ermöglicht eine Konsistenz in der Ausgabe und damit höhere Erwartungskonformität beim Nutzer.

6.2.1. Error-Dialog

Der Error-Dialog (Abbildung 6.3) ist eine Ausgabemöglichkeit von jQueryMobile, um Fehlermeldungen auszugeben. Diese Dialoge werden geworfen, falls eine Funktion nicht erwartungsgemäß ausgeführt werden konnte. Die Anzeigezeit der Dialoge ist sehr gering, um die Zuordnung einer Benutzeraktion zu einer Benutzeraktion zeitnah zu ermöglichen. Der Dialog verblasst deshalb nach einigen Sekunden von selbst. Ausgelöst werden diese Fehler bei jeglicher Art von unerwartetem Verhalten, wie beispielsweise eines nicht erfolgreichen *AJAX*-Requests auf die *API*.

6.2.2. Lade-Dialog

Der Lade-Dialog (Abbildung 6.4) ist ein Feedback des Systems, um den Nutzer zu versichern, dass seine Aktion ausgelöst wurde und gerade verarbeitet wird. Sie wird vor dem eigentlichen Ausführen der Aktion aufgerufen und wird als, als letzter Befehl, vor Beendigung geschlossen.

6.2.3. Benachrichtigungen

Um wichtige Nachrichten an den Benutzer zu visualisieren, muss die Aufmerksamkeit des Benutzers auf die entsprechende Meldung gelenkt werden. Diese Benachrichtigungen werden in der *App* mittels nativer, das bedeutet vom Betriebssystem abhängigen, *Alert*-Dialoge visualisiert. Die Entscheidung, auf native Informationsmethodiken zurückzugreifen, basiert

Abbildung 6.3.: Error-Dialog

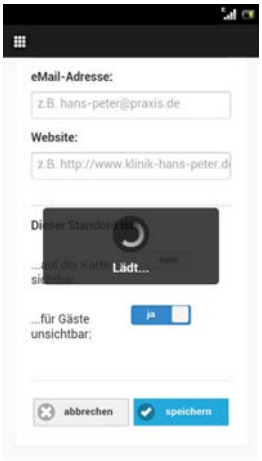


Abbildung 6.4.: Lade-Dialog

auf der Vertrautheit des Nutzers im Umgang mit diesen *UI*-Elementen. Benachrichtigungen der *App* werden automatisch aufgerufen, falls der Nutzer beispielsweise keine Verbindung zu einem Netzwerk besitzt (Abbildung 6.5), falsche oder ungenügende Daten eingibt oder auf eine zusätzliche Freischaltung (Abbildung 6.5) aufmerksam gemacht werden soll.

6.2.4. Entscheidungen

Indem der Nutzer Eingaben oder Aktionen, die ein ungewolltes Verhalten auslösen können, bestätigen muss, wird die Sicherheit gegen Fehleingaben und die Usability des Systems erhöht. Diese Entscheidungen, ob Aktionen ausgeführt werden sollen, werden in der *App* mithilfe nativer *Confirm*-Dialoge realisiert. Dies ist die Fortführung der Designentscheidungen, die auch bei den Benachrichtigungen getroffen wurde. Nutzer müssen mithilfe der Ja-/Nein-Struktur eine Aktion bestätigen, um ihnen den Eindruck der Kontrolle über das System zu gewähren. Dies ist notwendig, falls beispielsweise Neuigkeiten geändert werden müssen (Abbildung 6.6) oder sich der Nutzer bei dem System abmelden will.

6.3. Öffentlicher Bereich

Der öffentliche Bereich übernimmt, die in Kapitel 3.2.1 definierten, Anforderungen der *Website* und optimiert diese für mobile Endgeräte. Dazu wurde die Navigation (Abbildung 6.8) in Rubriken unterteilt, die als semantische Trennung einzelner Aufgaben dienen. In der ersten Rubik *Neuigkeiten* wird der Startpunkt der *App* und die aktuellen Neuigkeiten zusammengefasst. Die zweite Rubrik *Mitglieder* widmet sich den einzelnen Mitgliederstandorten und -praxen, in der diese vorgestellt werden. Die letzte Rubrik *Über uns* umfasst rechtliche Informationen zu Inhalten und eine Möglichkeit zum Kontakt mit dem offiziellen Vertreter der *ANC-BWN*.

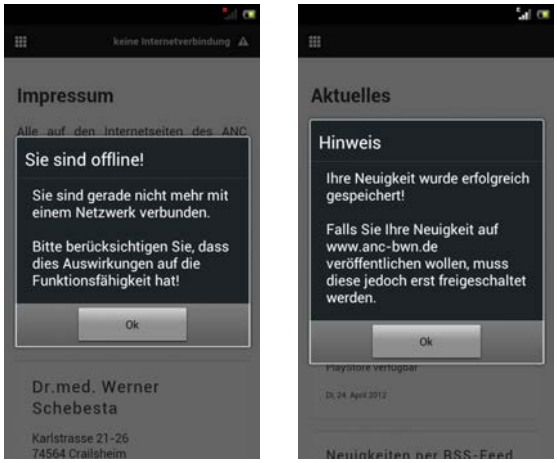


Abbildung 6.5.: Benachrichtigungen

Abbildung 6.6.: Entscheidungsabfrage

6.3.1. Start

Ruft ein Nutzer die *App* auf, so wird zuerst der *Splashscreen* (Abbildung 6.7, links) angezeigt. Das bildschirmfüllende Logo soll den Benutzer informieren, dass er die *App* der *ANC-BWN* gestartet hat und über den Fehlaufruf informieren. Des Weiteren kann darüber die gesamte Aufmerksamkeit des Nutzer auf das Logo und die *URL* konzentriert werden, um ihn damit vertraut zu machen. Falls der Benutzer über eine Verbindung zum World Wide Web verfügt, so wird er automatisch auf ein Navigationspunkt *Start* geleitet. Dieser begrüßt den Nutzer mit einer Willkommensnachricht (Abbildung 6.7, rechts) und informiert ihn über den Inhalt der *App*. Dazu ruft die *App* den, der Nutzergruppe entsprechenden, Text bei der *RESTful API* ab und umschließt damit das Logo der *ANC-BWN*.

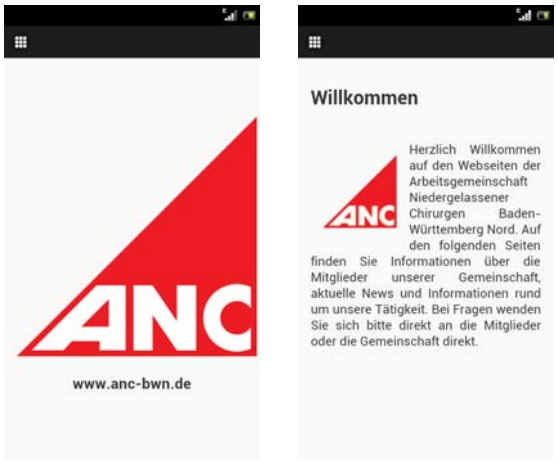


Abbildung 6.7.: Startpunkt der Applikation

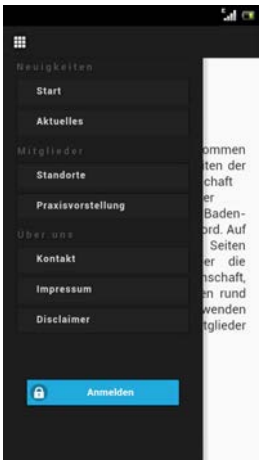


Abbildung 6.8.: Öffentliche Navigation

6.3.2. Aktuelles

Aktuelle Neuigkeiten können in der *mobile App* unter dem Navigationspunkt *Aktuelles* eingesehen werden. Hier werden Neuigkeiten angezeigt, die von öffentlichen Interesse sind. Der Benutzer der *App* wird nach dem Aufruf zuerst auf die Übersicht der aktuellen Neuigkeiten (Abbildung 6.9), absteigend nach dem Erstellungsdatum sortiert, geleitet. Einzelne Neuigkeiten werden als Einheit dargestellt, die aus Titel, Kurzbeschreibung, Erstelldatum und gegebenenfalls Autor besteht. Nach einem Touch- oder Klick-Event auf eine Neuigkeit wird eine Einzelansicht (Abbildung 6.9) mit dessen Inhalt aufgerufen. Dabei wird die Übersicht nach links aus dem Bildschirm geschoben, um eine Vertiefung des Inhaltes zu simulieren. Die Einzelansicht beinhaltet den Titel, die vollständige Neuigkeit, Erstelldatum und gegebenenfalls Autor. Eine Neuigkeit in der Einzelansicht unterscheidet sich in ihrer Stildefinition nicht von einer Neuigkeit in der Übersicht, um dem Nutzer eine bekannte Struktur zu bieten. Bei einem *Zurück*-Event wird die Einzelansicht nach rechts verschoben, um der von links einschiebenden Übersicht zu weichen.

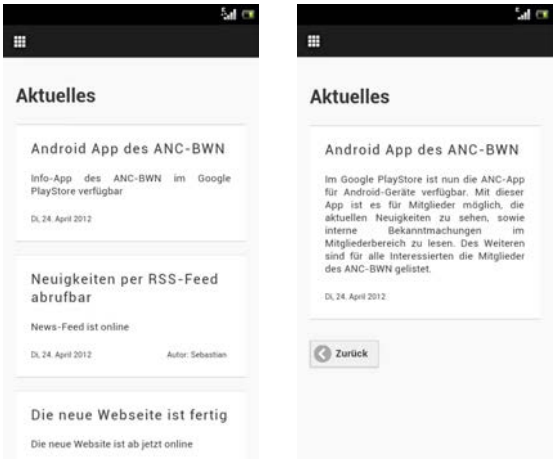


Abbildung 6.9.: Übersicht (links) und Einzelansicht (rechts) öffentlicher Neuigkeiten

6.3.3. Standorte

Über den Navigationspunkt *Standorte* können einzelne Mitglieder Informationen über ihre Praxis veröffentlichen. Dabei werden die Standorte der Mitglieder, falls sie dies wünschen, auf einer Karte markiert. Bei Aufruf der *Standorte* wird zunächst eine Übersichtskarte und eine Auflistung aller öffentlich sichtbaren Mitglieder (Abbildung 6.10) angezeigt. Einzelne Standorte, markiert mit dem *ANC-BWN*-Logo, können bei Touch- oder Klick-Event auf der Karte eine Infobox einblenden, in der die wichtigsten Informationen über den Standort gelistet sind. Visualisiert mit einem Pfeil nach rechts, kann eine Einzelansicht eines gelisteten Mitglieds aufgerufen werden, um detailliertere Informationen zu erhalten. Um eine Vertiefung des Inhaltes zu visualisieren, wird die Einzelansicht aus der Pfeilrichtung (rechts) in den sichtbaren Bereich eingeschoben. Bei der Einzelansicht eines Mitglieds (Abbildung 6.10) wird, in Abhängigkeit der getroffenen Einstellung, eine Karte mit dem Standort und die

Kontaktinformationen, in Form einer Visitenkarte, dargestellt. Da nur noch ein einzelner Standort visualisiert werden muss, wurde der entsprechende Kartenausschnitt vergrößert.

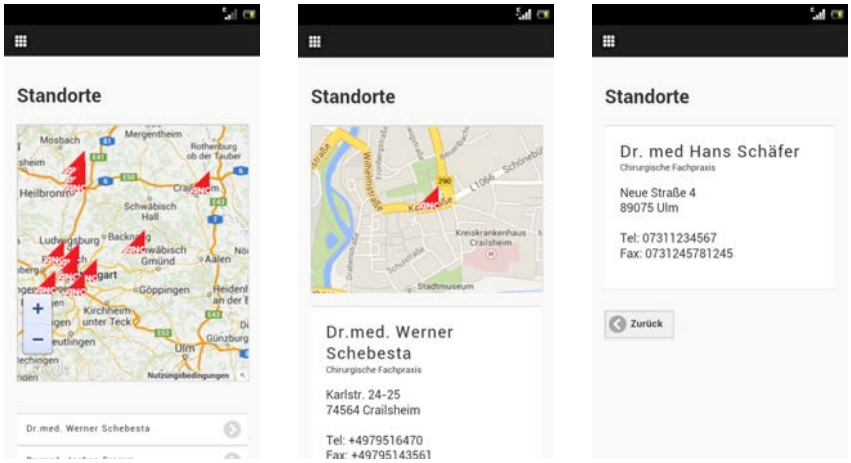


Abbildung 6.10.: Übersicht (links), Einzelansicht mit Karte (mittig) und Einzelansicht ohne Karte (rechts) öffentlicher Standorte

6.3.4. Praxisvorstellung

In der *Praxisvorstellung* werden Impressionen eines Mitgliederstandortes veröffentlicht und beschrieben. Das Prinzip ist ähnlich einer sogenannten *Slideshow*, da einzelne Impressionen aneinander gereiht werden und mittels Vor- und Zurück-Navigation ein- und ausgeblendet werden können. Eine Impression (Abbildung 6.11) besteht aus einem Bild, einem Mitgliedsname und einem Beschreibungstext. Damit können, beispielhaft in Abbildung 6.11, Wartezimmer, Praxismitarbeiter oder Behandlungsräume vorgestellt werden.

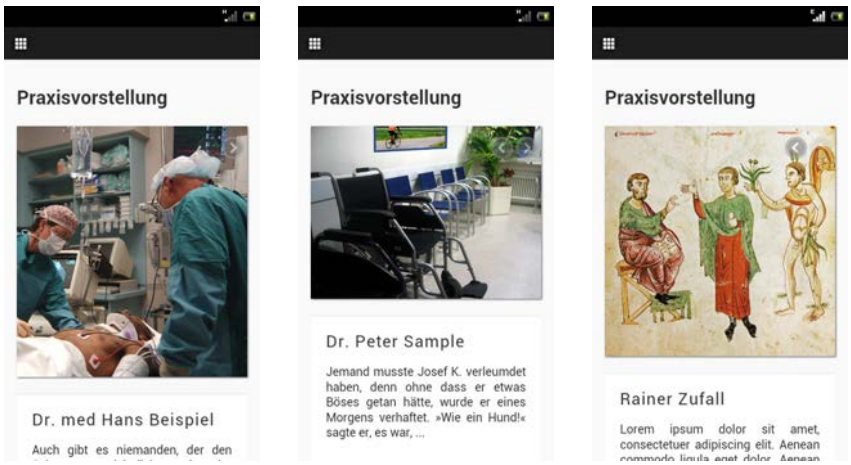


Abbildung 6.11.: Beispielhafte Verwendung der Praxisvorstellung, [Incc] [Pix] [Incd]

6.3.5. Kontakt

Über den Navigationspunkt *Kontakt* (Abbildung 6.12, links) werden die Standort- und Adressdaten des offiziellen Vorsitzenden der *ANC-BWN* angezeigt, um Benutzern die Möglichkeit einer Kontaktaufnahme zu geben. Dazu gehören, vergleichbar mit der Einzelansicht eines Standortes, eine vergrößerte Kartenansicht und die Kontaktdaten des Vorsitzenden der *ANC-BWN*.

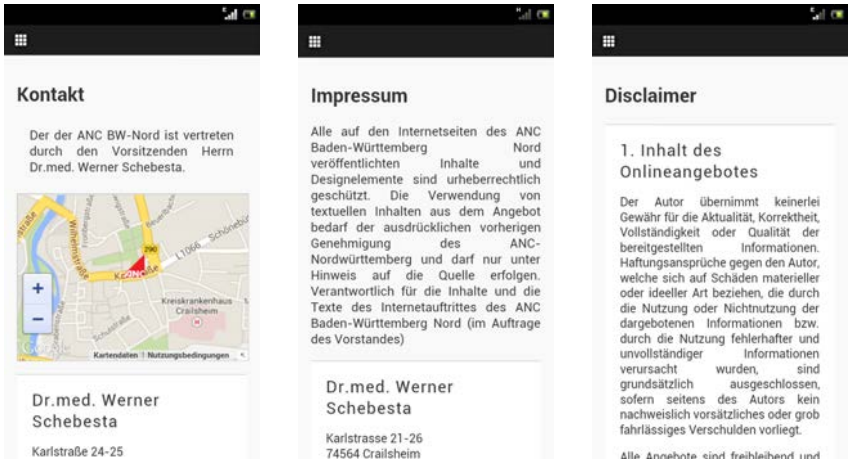


Abbildung 6.12.: Navigationsrubrik *Über uns*

6.3.6. Impressum

Das *Impressum* (Abbildung 6.12, mittig) ist für die Kontaktaufnahme zwecks rechtlichen Anspruchs eingerichtet, der unter dem Navigationspunkt *Disclaimer* detailliert erläutert wird. Es werden die Adressdaten einer Kontaktperson ausgewiesen, die für den Inhalt, beziehungsweise die Umsetzung der *mobile App*, verantwortlich ist, da dies nach der Impressumspflicht verlangt ist.

6.3.7. Disclaimer

Der *Disclaimer* (Abbildung 6.12, rechts) ist der Haftungsausschluss für verwendete Inhalte. Dieser ergänzt das *Impressum* um die haftungsausschließenden Erklärungen des *ANC-BWN* und wird benötigt, da Mitglieder öffentliche Einträge im Namen der *ANC-BWN* verfassen und damit beispielsweise Urheberrecht verletzen könnten. Einzelne Erklärungen werden in der *App* als Block dargestellt, um eine Zusammengehörigkeit von Text und Thema zu verdeutlichen.

6.3.8. Login

Der *Login* ist über den *Anmelden-Button* in der Navigation zu erreichen. Der *Button* hebt sich optisch von den restlichen Navigationspunkten ab, um eine Abgrenzung des Inhaltes zu verdeutlichen. Diese Abgrenzung ist im *Login* (Abbildung 6.13, links) weiter verwendet, um den Nutzer bei der Wegfindung zu unterstützen. Der *Login* umfasst zwei Eingabefelder für Nutzername und Passwort. Benutzer müssen vor der Benutzung der *App* im *CMS* registriert sein, damit sie sich anmelden können. Eine Registrierung in der *App* ist nicht vorgesehen. Die eingegebenen Nutzerdaten werden bei Touch- oder Klick-Event auf den hervorgehobenen *Button*, mit der Beschriftung *"Anmelden!"*, zuerst validiert. Dabei werden die Daten an die *RESTful API* gesendet, um den Benutzer zu authentifizieren. Ist die Authentifizierung erfolgreich, so werden die eingegebenen Daten gespeichert und der Nutzer wird als Mitglied in den zugriffsbeschränkten Bereich (Kapitel 6.4) weitergeleitet. Ist die Authentifizierung fehlgeschlagen, so wird eine native Benachrichtigung (Abbildung 6.13, mittig) ausgegeben, um den Benutzer darüber zu informieren. Ist eine Validierung fehlgeschlagen, werden die Eingabefelder für den Nutzername und das Passwort rot umrandet (Abbildung 6.13, rechts), um den Nutzer die Fehleingabe zu visualisieren.

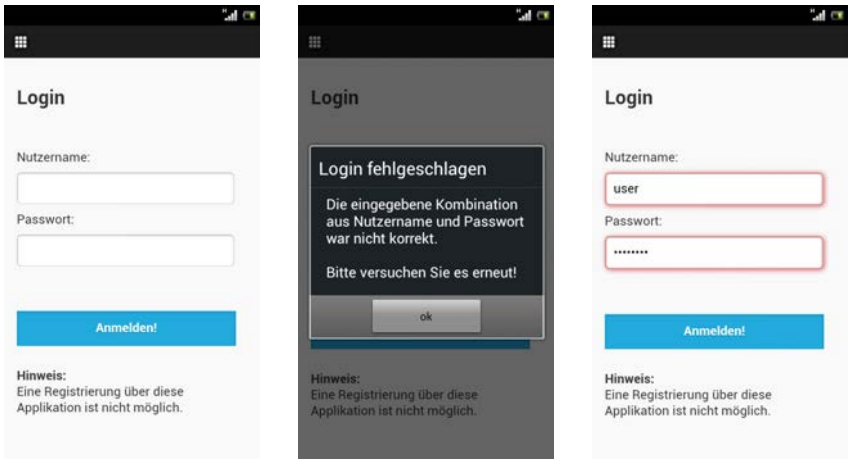


Abbildung 6.13.: Anmeldeformular mit Validierungsbenachrichtigungen

6.4. Zugriffsbeschränkter Bereich

Der zugriffsbeschränkte Bereich erweitert das Funktionsspektrum des öffentlichen Bereichs (Kapitel 6.3) um autorisierungspflichtige Funktionen (Abbildung 4.1), die den Inhalt über die *RESTful API* beeinflussen können. Die Erweiterungen wurden dabei nahtlos in die bestehende *UI* eingefügt, um eine einheitliche Darstellung des Inhaltes in beiden Nutzergruppen zu gewährleisten. Die Navigation wurde zusätzlich um die Rubrik *Rundschreiben* erweitert (Abbildung 6.14, links), über welche Informationen nur für Mitglieder des *ANC-BWN* verwaltet werden können. Außerdem wurde der *Button* für das *Anmelden* am System in ein *Abmelden* ausgetauscht. Dieser verlangt bei Aktivierung des *Button* eine Bestätigung der Benutzeraktion

(Abbildung 6.14, rechts) und verhindert somit Fehleingaben.

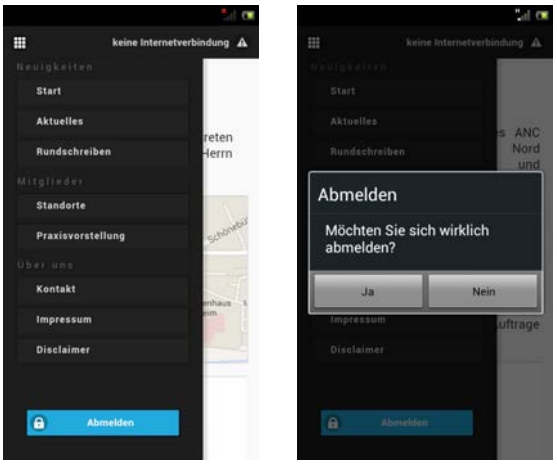


Abbildung 6.14.: Navigation mit Logout-Funktion im zugriffsbeschränkten Bereich

Abbildung 6.15.: Startpunkt und Willkommensnachricht für Mitglieder

6.4.1. Start

Bei *App*-Aufruf wird der Benutzer vom, in Kapitel 6.3 vorgestellten, *Splashscreen* ebenfalls automatisch auf den Navigationspunkt *Start* und dessen Willkommensnachricht weitergeleitet. Dabei ruft die *App* jedoch die im *CMS* gespeicherte Nachricht für Mitglieder (Abbildung 6.15) auf, um dem Benutzer nach dem Aufruf der *App* zu signalisieren, dass er die Rechte eines Mitglieds besitzt und sich somit im zugriffsbeschränkten Bereich befindet.

6.4.2. Aktuelles

Im zugriffsbeschränkten Bereich wurde der Navigationspunkt *Aktuelles* mit verwaltungstechnischen Funktionen, wie Bearbeiten, Erstellen oder Löschen einer Neuigkeit, ausgestattet. Diese Funktionen sollen die gewohnte *UI* erweitern, um keine sichtbare Trennung zwischen dem öffentlichen und zugriffsbeschränkten Bereich zu erzeugen. Dazu wurden in die Übersicht und die Einzelansicht der Neuigkeiten (Abbildung 6.16) Interaktionselemente eingebunden.

Erstellen einer Neuigkeit

Das Interaktionselement zum *Erstellen* einer Neuigkeit wird in der Übersicht (Abbildung 6.16), jeweils vor der ersten und nach der letzten Neuigkeit, eingefügt. Diese Redundanz erhöht die Verfügbarkeit der Aktion und dadurch die Usability des Systems. Mittels eines Plus-Zeichen als Icon im *Button* wird versucht die Selbsterklärungsfähigkeit der Aktion zu verdeutlichen.

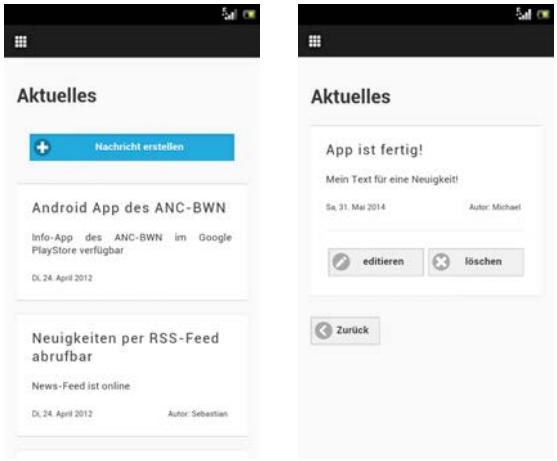


Abbildung 6.16.: Übersicht (links) und Einzelansicht (rechts) von Neuigkeiten im zugriffsbeschränktem Bereich

Wird die Funktion zum *Erstellen* einer Neuigkeit aufgerufen, so erscheint ein Eingabeformular (Abbildung 6.17) für die einzugebenden Daten. Das Formular ist in einer Einheit, die von Struktur und Aufbau der einer einzelnen Neuigkeit (Abbildung 6.9, rechts) gleichen soll, eingebettet. Der Titel wurde optisch hervorgehoben, um die Analogie einer größeren Schrift für Überschriften zu verdeutlichen. Nach den Eingabefeldern für textbasierte Daten gibt es eine Auswahlmöglichkeit, ob die Neuigkeit nur im zugriffsbeschränkten Bereich angezeigt werden soll. Dies ermöglicht es den Mitgliedern interne Neuigkeiten, die anders als ein *Rundschreiben* auch auf der *Website* sichtbar sind, zu erzeugen. Dazu muss der Benutzer mittels Touch- oder Klick-Event den sogenannten "Flipswitch" (Abbildung 6.17, rechts) auf die passende Einstellung setzen. Des Weiteren wird ein Erstelldatum der Nachricht gesetzt, welches die Systemzeit auswertet und daraus einen lesbaren String erzeugt. Dies soll dem Benutzer die Eingabezeit für das Datum nehmen und seine Konzentration auf der eigentlichen Neuigkeit belassen. Um die Selbsterklärungsfähigkeit zu erhöhen, wurde eine *Button*-Gruppe mit Kontrollfunktionen, getrennt durch eine horizontale Linie, an das Ende der Neuigkeit, jedoch innerhalb der einzelnen Einheit, platziert. Dabei wurde, um den Workflow zu erhöhen, der *Speichern-Button* in der selben Weise hervorgehoben, wie der *Erstellen-Button* in der Neuigkeitenübersicht. Diese konstante Linie der Benutzerführung, die in allen weiteren autorisierungspflichtigen Funktionen fortgeführt wird, soll den Benutzer unterbewusst helfen, den korrekten *Button* auszulösen. Ein Auslösen des *Abbrechen-Button* würde in diesem Fall die Neuigkeit ungefragt verwerfen. Es wurde zusätzlich, anders als bei der Einzelansicht einer Neuigkeit, auf den *Zurück-Button* verzichtet, damit das Verwerfen einer Neuigkeit nur über einen *Button* ausgelöst werden kann und der Benutzer in seiner Erwartungskonformität für eindeutige Benutzerführung bestätigt wird.

Um eine Neuigkeit dem *CMS* vollständig zu übergeben, müssen alle Pflichtfelder ausgefüllt werden. Diese werden in der *App* mit einem roten Stern markiert (Abbildung 6.17). Falls der Benutzer eine unvollständige Neuigkeit speichern möchte, so wird er mittels einer nativen Benachrichtigung (Abbildung 6.18, links) hingewiesen, dass leere Pflichtfelder ausgefüllt werden müssen. Dazu werden die ungültigen Pflichtfelder rot umrandet, um den Nutzer auf die Lokalität

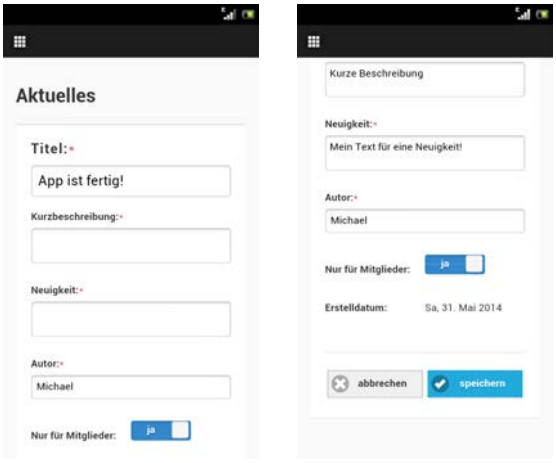


Abbildung 6.17.: Eingabeformular für das Erstellen einer Neuigkeit

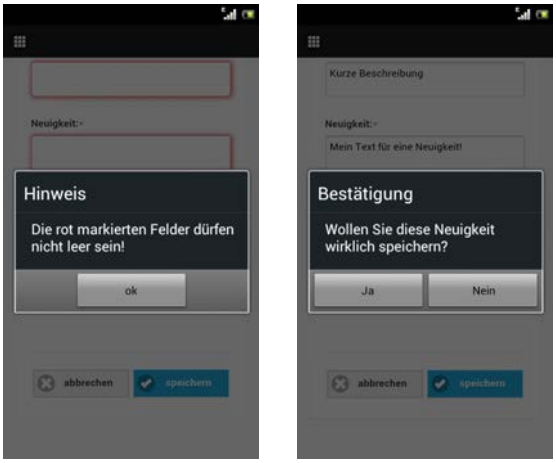


Abbildung 6.18.: Validation und Bestätigung einer Neuigkeit

seines Fehlers hinzuweisen. Ist eine Neuigkeit vollständig, so wird nach Betätigen des *Speichern-Button* der Validierungshinweis mit einer Bestätigung der Speichern-Aktion (Abbildung 6.18, rechts) ausgetauscht. Falls der Benutzer die Aktion bewilligt, so wird die Nachricht über die *RESTful API* im *CMS* gespeichert und der Benutzer auf die Neuigkeitenübersicht weitergeleitet, in der die erstelle Neuigkeit erscheint (Abbildung 6.19, rechts). Dabei wird der Benutzer informiert, dass seine Neuigkeit erst über die Verwaltung des *CMS* freigeschaltet werden muss, um seine Aktion zu bestätigen (Abbildung 6.19, links). Vor einer Freischaltung ist die Neuigkeit nicht auf der *Website* der *ANC-BWN* verfügbar, kann jedoch in der *App* angezeigt werden.

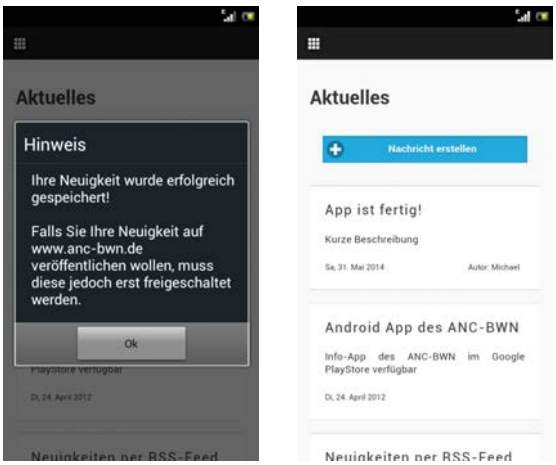


Abbildung 6.19.: Freischalten einer Neuigkeit

Erweiterte Einzelansicht

Wird eine Neuigkeit im zugriffsbeschränkten Bereich aufgerufen, so erscheint die gewohnte Einzelansicht der Neuigkeiten (Abbildung 6.9), jedoch um die Funktionen *editieren* und *löschen* erweitert (Abbildung 6.16, rechts). Die Trennung erfolgt, mittels einer horizontalen Trennlinie, im Element selbst, um den Bezug zwischen Aktionen und betroffenen Objekt zu verdeutlichen. Zur Aktion passende Icons verdeutlichen außerdem die zu erwartende Aktion, um die Selbsterklärungsfähigkeit des Systems zu erhöhen.

Bearbeiten einer Neuigkeit

Wird der *Editieren-Button* in der Einzelansicht betätigt, so erscheint ein mit den Daten der einzelnen Neuigkeit ausgefülltes Eingabeformular (Abbildung 6.20), vergleichbar mit dem Formular der Neuigkeitenerstellung (Abbildung 6.17). Die eindeutige Darstellung bestätigt dabei die Erwartung des Benutzers und erhöht die Usability des Systems. Zusätzlich zum Eingabeforular der Erstellfunktion wird das Datum der letzten Änderung angezeigt, welches bei einem Speichern der bearbeiteten Neuigkeit auf das aktuelle Datum geändert wird. Die Validierung des Inhaltes, die Eindeutigkeit der Benutzerführung und die Bestätigung der Speichern-Aktion ist analog zu der Neuigkeitenerstellung implementiert.

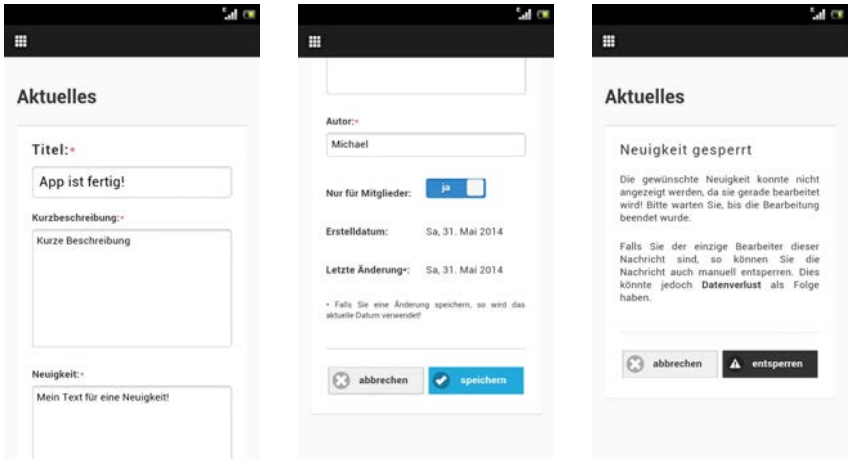


Abbildung 6.20.: Bearbeiten einer Neuigkeit

Ein besonderes Merkmal der Neuigkeitenbearbeitung gilt dem Sperren von Funktionen bei Mehrfachaufruf. Die *mobile App* implementiert eine Sperrfunktion, die fünf Sekunden nach dem Aufrufen des Eingabeformulars ausgelöst wird. Die zeitliche Verzögerung soll den Benutzer bei ungewollten Aufruf das *Abbrechen* des Bearbeitungsvorgangs ermöglichen, ohne dass die Sperrfunktion aktiviert wird. Ist eine Nachricht durch einen Bearbeiter gesperrt, so bekommt ein zweiter Benutzer bei Betätigung des *Editieren-Button* ein Hinweis auf die Sperrung des Inhaltes (Abbildung 6.20, rechts) angezeigt. Eine Entsperrung der Bearbeitungsfunktion für diese Nachricht erfolgt entweder über das erfolgreiche *Speichern* der bearbeiteten Nachricht, das *Abbrechen* des Bearbeitungsvorgangs von dem Benutzer, der die Sperrfunktion aktiviert hat oder über eine manuelle Entsperrung über den Sperrhinweis. Die manuelle Entsperrung

ist notwendig, da ein transaktionelles Verhalten [Rei] nur simuliert werden kann und ein Zurücksetzen des Sperrvorgangs durch Absturz der *App* oder Verlust der Netzwerkverbindung verhindert werden kann.

Löschen einer Neuigkeit

Um eine Neuigkeit zu *löschen*, muss eine Einzelansicht (Abbildung 6.16, rechts) der entsprechenden Neuigkeit aufgerufen werden. Dort können Mitglieder über einen *Löschen-Button* die entsprechende Nachricht entfernen. Bei Betätigung wird ein Bestätigungsdialog (Abbildung 6.6) aufgerufen, in dem der Benutzer die Aktion bestätigen muss. Bei Bestätigung wird eine Anfrage an die *RESTful API* geschickt, die die passende Neuigkeit als gelöscht markiert. Um Fehleingaben der Benutzer zurücksetzen zu können, ist eine Neuigkeit weiterhin im *CMS* gespeichert, wird jedoch nicht mehr angezeigt.

6.4.3. Rundschreiben

Über den Navigationspunkt *Rundschreiben* können interne Nachrichten für Mitglieder und der Verweis auf eine Datei veröffentlicht werden. Nach dem Aufruf wird der Benutzer zuerst eine Übersicht (Abbildung 6.21, links), über erstelle Rundschreiben erhalten. In dieser Ansicht kann er entweder über einen *Button* neue Rundschreiben erstellen oder eine Einzelansicht eines Rundschreibens (Abbildung 6.21, rechts) aufrufen. Anders als in der Neuigkeitenübersicht, wird hier nur der Titel des Rundschreibens, das Erstelldatum und der Autor angezeigt. Wählt der Nutzer ein Rundschreiben für eine Einzelansicht aus, so wird, zusätzlich zu Titel, Erstelldatum und Autor, die Nachricht des Rundschreibens und gegebenenfalls eine *URL* angezeigt. Des Weiteren werden innerhalb der sichtbaren Einheit, über eine horizontale Linie getrennt, die *Editieren-* und die *Löschen*-Funktion platziert.

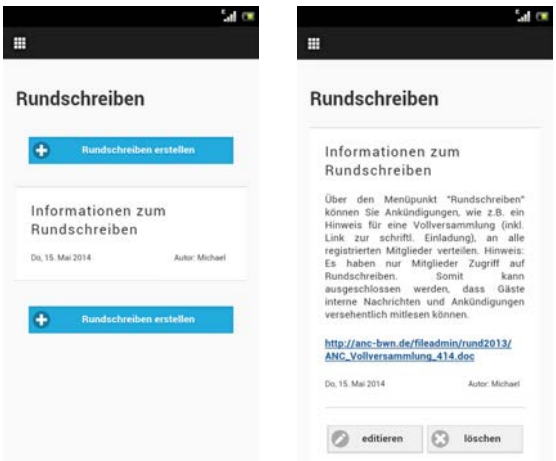


Abbildung 6.21.: Übersicht (links) und Einzelansicht (rechts) eines Rundschreibens

Erstellen eines Rundschreibens

Über die Rundschreibenübersicht gelangt man zum Eingabeformular (Abbildung 6.22, links) der *Erstellen*-Funktion für Rundschreiben. Darin können, bis auf das Erstelldatum, die einzelnen Daten eines Rundschreiben eingegeben werden. Das Erstelldatum wird automatisch bei der Speicherung des Rundschreibens ergänzt. Analog zu der Neuigkeitenvalidierung (Kapitel 6.4.2) werden benötigte Eingabefelder, markiert mit einem roten Stern, rot umrandet (Abbildung 6.22, rechts), falls diese, bei einem Speicherversuch, nicht gefüllt wurden. Da die Form der *URL* nicht selbsterklärend ist, wird diese mit einem Platzhalter in das entsprechende Eingabefeld eingefügt, bis der Nutzer eine Eingabe praktiziert. Bei Erstellung des Rundschreibens wird der Platzhalter nicht berücksichtigt. Nachdem ein Rundschreiben erstellt wurde, wird der Benutzer auf die Übersicht weitergeleitet.

Bearbeiten eines Rundschreibens

Die Bearbeitenfunktion kann der Benutzer über die Einzelansicht des Rundschreibens aufrufen. Dabei wird eine ausgefüllte, mit dem Eingabeformular der Rundschreibenerstellung vergleichbare, Eingabemaske (Abbildung 6.23, links) ausgegeben. Über diese kann der Inhalt, analog zur Neuigkeitenbearbeitung (Kapitel 6.4.2), validiert und gespeichert werden. Vor der Übernahme einer Änderung muss der Benutzer diese über einen nativen Dialog (Abbildung 6.23, rechts) bestätigen.

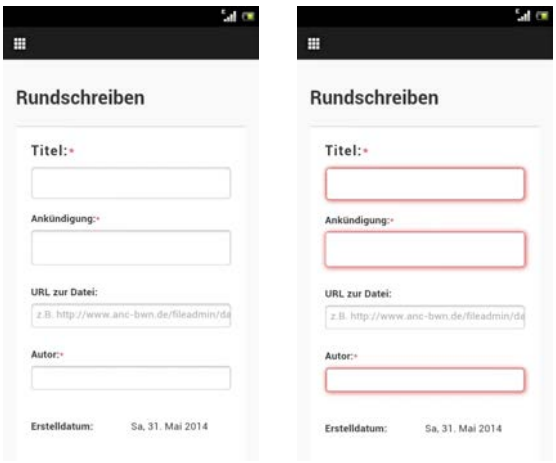


Abbildung 6.22.: Eingabeformular für das Erstellen eines Rundschreibens

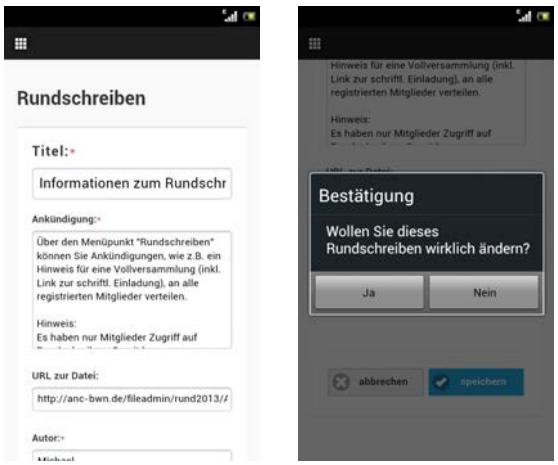


Abbildung 6.23.: Bearbeiten eines Rundschreibens

Löschen eines Rundschreibens

Ein Rundschreiben kann über die Einzelansicht gelöscht werden. Dabei wird ein nativer Dialog erzeugt, um von dem Benutzer eine Bestätigung der Aktion zu verlangen. Nach

Erhalt der Bestätigung wird eine Anfrage, über die *RESTful API* an das *CMS*, versendet und das entsprechende Rundschreiben als gelöscht markiert. Eine komplette Löschung eines Rundschreibens aus der Datenbank ist aus Gründen der Datensicherheit nicht vorgesehen.

6.4.4. Standorte

Unter dem Navigationspunkt *Standorte* können Adress- und Standortdaten von Mitglieder visualisiert und verwaltet werden. Zur Visualisierung können alle Mitglieder, falls sie dies wünschen, in einer Karte markiert und in einer Liste oder einer Einzelansicht (Kapitel 6.3.3) ausgegeben werden. In der Übersicht wird nun, analog zu der Neuigkeitenverwaltung (Kapitel 6.3.2), ein *Button*, jeweils vor und nach der Mitgliederliste, für die Standorterstellung platziert (Abbildung 6.24). Außerdem können im zugriffsbeschränkten Bereich Standorte angezeigt werden, die als "nur intern sichtbar" markiert sind.

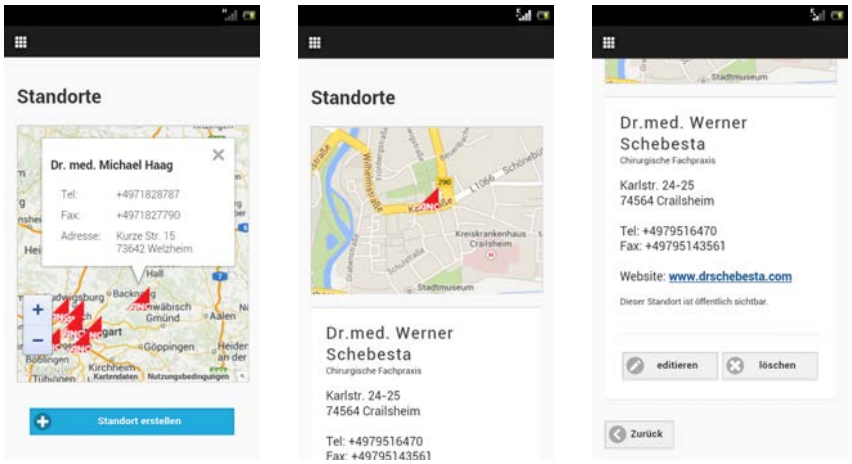


Abbildung 6.24.: Übersicht mit aktiviertem Infofenster (links) und Einzelansicht (mittig und rechts) eines Standortes im zugriffsbeschränktem Bereich

Erweiterte Einzelansicht

Die Einzelansicht eines Standortes im zugriffsbeschränkten Bereich (Abbildung 6.25) wurde um einen Hinweis erweitert, der Auskunft über die Sichtbarkeit des Standortes gibt. Werden Standorte nur im zugriffsbeschränkten Bereich angezeigt, so kann dies über diesen Hinweis erkannt und gegebenenfalls geändert werden. Wie bereits im öffentlichen Bereich (Kapitel 6.10) können einzelne Standorte auch ohne Karte angezeigt werden. Ist dies der Fall, so werden diese auch nicht in der Standortübersicht angezeigt. Mit einer horizontalen Linie getrennt wird die *Editieren*- und *Löschen*-Funktion eingebunden.

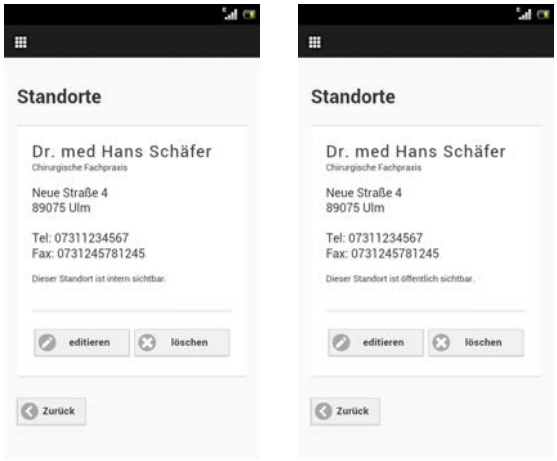


Abbildung 6.25.: Einzelansicht eines internen/öffentlichen Standortes ohne Kartenfunktion im zugriffsbeschränktem Bereich

Erstellen eines Standortes

Über die Standortübersicht kann das Eingabeformular zum Erstellen von Standorten (Abbildung 6.26) aufgerufen werden. Es umfasst mehrere Angaben zur Person, Adresse der Praxis, Kontaktmöglichkeiten und Anzeigeverhalten in der *App*. Aufgrund der Komplexität und Länge der Eingabedaten, wurde für jedes Feld ein beispielhafter Platzhalter eingefügt. Diese Platzhalter werden überschrieben, sobald der Nutzer auf ein Feld zugreift und werden bei einem Speicherversuch als leer angesehen. Um die Übersichtlichkeit des Formulars zu erhöhen, wurden horizontale Trennlinien eingefügt, um semantisch zusammenhängende Felder zu gruppieren. Eine Validierung von Pflichtfeldern und Aktionsbestätigung, wie sie in der Nachrichtenverwaltung (Kapitel 6.4.2) vorgestellt wurde, wurde ebenfalls integriert (Abbildung 6.26, rechts). Nachdem ein Standort erstellt wurde, wird der Benutzer auf die Standortübersicht weitergeleitet. Es erscheint außerdem ein Hinweis, dass der Standort für Ansicht auf der *Website* der *ANC-BWN* erst über das *CMS* freigeschaltet werden muss, um fehlerhafte Einträge vor Veröffentlichung zu prüfen.

Bearbeiten eines Standortes

Über die Einzelansicht eines Standortes kann die Standortbearbeitung aufgerufen werden. Die Standortbearbeitung besteht, je nach Sichtbarkeitseinstellung, aus einer Karte mit dem zuvor angegebenen Standort und einem Eingabeformular, welches mit der Standorterstellung (Abbildung 6.27) vergleichbar ist. Einzelne Felder werden dabei entweder mit den gegebenen Mitgliederdaten gefüllt oder werden leer dargestellt. Auf die Integration von Platzhalten wurde aus Gründen der Verwechselbarkeit mit Inhalt verzichtet. Die letzte semantische Gruppierung innerhalb des Formulars ist für die Sichtbarkeit in der *App* zuständig. Dabei können Standorte auf der Karte oder für Gäste ausgeblendet werden. Es wurde außerdem aus Gründen der Redundanz auf einen *Zurück-Button* verzichtet, um die Erwartungskonformität zu erhöhen. Möchte der Benutzer eingegebene Daten übernehmen, so wird er um Bestätigung der Aktion

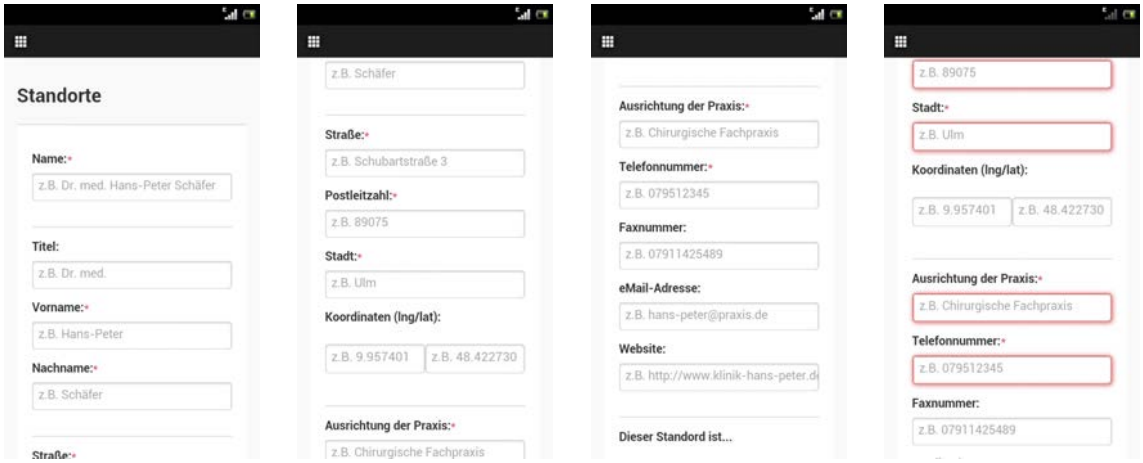


Abbildung 6.26.: Eingabeformular für das Erstellen eines Standortes

gebeten. Falls Eingabefelder nicht ausgefüllt wurden, so werden diese bei der Validierung bemängelt und das Speichern unterbrochen.

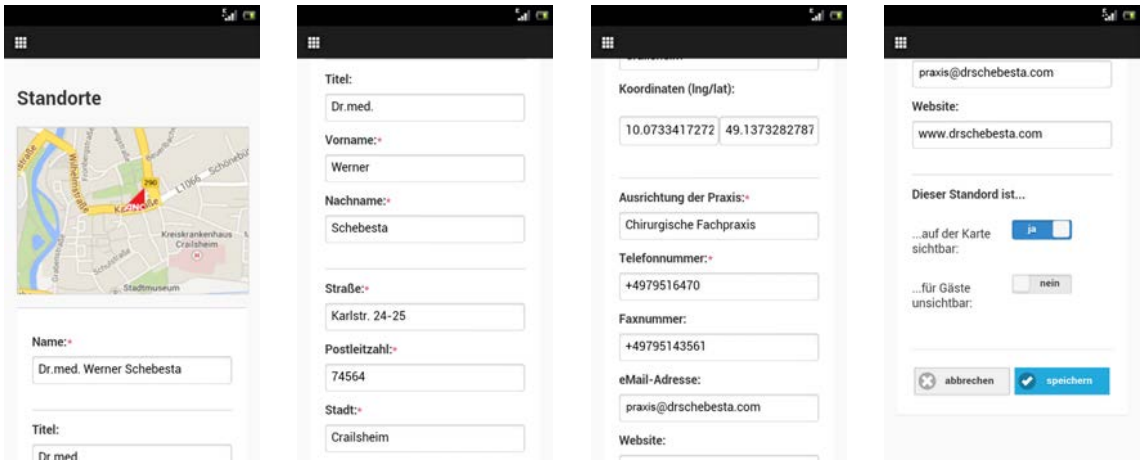


Abbildung 6.27.: Eingabeformular für das Bearbeiten eines Standortes

Löschen eines Standortes

Um einen Standort zu *löschen*, muss dieser in einer Einzelansicht (Abbildung 6.24) ausgewählt werden. Nachdem Aufruf des *Löschen-Button* wird der Benutzer mit einem nativen Dialog die Aktion zu bestätigen. Bestätigt er diese, so wird eine Anfrage über die *RESTful API* an das *CMS* gesendet, welches den Standort als gelöscht markiert. Aus Gründen der Datensicherheit werden Standorte über die *mobile App* nicht endgültig gelöscht.

7 Anforderungsabgleich

In diesem Kapitel werden die analysierten Anforderungen an die *mobile App* und die *RESTful API* (Kapitel 3) mit der Umsetzung, die aus Architektur (Kapitel 4), Implementierung (Kapitel 5) und dem Walkthrough (Kapitel 6) besteht, verglichen. Die einzelnen Anforderungen werden in der nachfolgenden Tabelle aufgelistet und deren Umsetzung bewertet. Ein Kommentar zur Bewertung gibt zusätzlichen Aufschluss über die Bewertungsentscheidung.

Es wird der folgende Bewertungsschlüssel verwendet:

++	Die Anforderung wurde vollständig erfüllt.
+	Die Anforderung wurde zufriedenstellend erfüllt.
0	Die Anforderung wurde nicht zufriedenstellend erfüllt.
-	Die Anforderung wurde nicht erfüllt.

Anforderung	Bewertung	Kommentar
Ein Gast soll öffentliche Neuigkeiten lesen können.	++	Solange der Benutzer eine Verbindung mit dem Internet besitzt, ist die Funktion verfügbar. Es kann bei schlechter Funknetzabdeckung jedoch dazu kommen, dass das sechssekündige *AJAX*-Timeout einer Anfrage an die *RESTful API* überschritten wird. In diesem Fall wird zugunsten einer Systemrückmeldung entschieden und die aktive Anfrage abgebrochen.
Ein Gast soll Standorte auf einer Karte einsehen können.	++	
Ein Gast soll Kontakt aufnehmen und rechtliche Informationen einholen können.	++	
Ein Benutzer soll sich über das System anmelden und abmelden können.	++	
Das System soll zwischen Benutzergruppen unterscheiden können.	++	Die definierten Benutzergruppen werden systemintern getrennt behandelt. Aus der Trennung in Benutzergruppen entstehen zusätzlich keine sichtbaren Beeinträchtigungen des User Interfaces für Benutzer.
Ein Mitglied soll öffentliche und interne Neuigkeiten einsehen und verwalten können.	++	Die Einzelansicht einer Neuigkeit enthält, ohne Nachladen von Daten, alle Funktion und Inhalte für die Verwaltung und eine Ressourcensperrung bei Mehrfachzugriff.
Das System soll ein möglichst großer Querschnitt an mobilen Betriebssystemen abdecken.	++	Durch die Verwendung von Cordova können alle aktuellen Betriebssysteme versorgt werden. Aufgrund der Entwicklung mit Webtechnologien kann die *App* auch als Rich Internet Application verwendet werden und weitere Systeme über den Browser erreichen.
Das System soll äußerst zuverlässig sein.	++	Die Systeme sind robust und zuverlässig entwickelt und geben bei erfolgreichen oder fehlgeschlagenen Aktionen stets ein Feedback.
Das System soll dem Nutzer bei jeder Handlung ein Feedback geben.	+	Es wird stets ein Feedback ausgegeben, jedoch könnte die Vielfalt an Fehlernachrichten erweitert werden, um den Benutzer eine genauere Information der Fehlerquelle zu geben.

Tabelle 7.1.: Anforderungsabgleich existierender Anforderungen

Anforderung	Bewertung	Kommentar
Ein Benutzer soll sich einen Eindruck der Praxis verschaffen können.	+	Der Zugriff ist einfach gestaltet und wird über ein Skript realisiert, das eine *JSON*-Datei weiterleitet. Neue Daten können über eine einfache Struktur in die *JSON*-Datei eingepflegt werden. Die Darstellung in der Applikation könnte jedoch über eine Neugestaltung der Slideshow im Landscape-Modus verbessert werden.
Das System soll mittels einer *API* Kontakt zur Datenbank aufnehmen.	++	Funktioniert serverseitig über die *RESTful API* und clientseitig über *AJAX*.
Die *API* soll Requests auf Syntax und Autorisierung validieren.	++	Bei einer eintreffenden Anfrage wird vor semantischer Auswertung die Syntax im *Validator* überprüft. Nutzt die Anfrage die *HTTP*-Methode *POST*, so prüft der *Valditor* nach der Syntax zusätzlich auf Autorisierung und bricht gegebenenfalls die Bearbeitung direkt ab.
Das System soll auf wichtige Hardware- und Software-Events reagieren.	++	Die *mobile App* wurde eventbasierend aufgebaut und horcht auf Hardware-Events des Cordova-Frameworks, sowie auf Software-Events des jQueryMobile-Frameworks. Zusätzlich hat die *App* eigene Funktionen implementiert, mit denen man einfach Software-Events erstellen kann.
Ein Mitglied soll einfach öffentliche und interne Standort-Daten und -Einstellungen, sowie Rundschreiben, einsehen und verwalten können.	++	Analog zur Nachrichtenverwaltung wurde die Standort- und Rundschreibenverwaltung in das vorhandene *UI* integriert. Eine vollständige Verwaltung der verwendeten Daten ist damit umgesetzt.
Die Praxisdarstellungen sollen leicht erweiterbar sein.	+	Aufgrund des einfachen *JSON*-Struktur können Inhalte ohne großes Einlesen erweitert werden. Um neue Inhalte einzupflegen, benötigt man nur einen *FTP*-Client und einen Texteditor. Als Erweiterung könnte man ein eigenes System zur Verwaltung der dateibasierten Daten entwickeln oder eine neue Tabelle in der Datenbank erstellen, worin die Bilder in *base64* oder als *BLOB* gespeichert werden.

Tabelle 7.2.: Anforderungsabgleich entstandener Anforderungen

Anforderung	Bewertung	Kommentar
Das System soll modular aufgebaut und leicht erweiterbar sein.	++	Aufgrund der internen Paket-Struktur und der Verwendung von Templates kann das System einfach erweitert werden. Für eine Erweiterung reicht es eine Template-Datei hinzuzufügen und den einen Eintrag in der *map.json* zu erstellen. Danach kann die neue Seite aufgerufen werden.
Das System soll als benutzerfreundliches Laiensystem gestaltet werden.	++	Die erstellte *UI* ist an vielen Stellen selbsterklärend gehalten und ermöglicht, dank Analogien zu anderen Applikationen, selbst unerfahrenen Nutzern die Bedienung.
Das System soll eine optimale Darstellung in jeglicher Auflösung bieten.	++	Durch die Verwendung von jQueryMobile konnt ein Responsive Design entwickelt werden, welches eine optimale Darstellung in jeder Auflösung garantiert.
Das System soll sehr robust gegenüber Fehleingaben, struktureller Mängel und Verbindungsfehlern sein.	++	Das System ist durch den modularen Aufbau und der losen Kopplung vor Fehleingaben und ähnlichen Mängeln geschützt, da es zu jeder Zeit versucht für sich als autonome Einheit zu reagieren. Deshalb werden alle Aufrufe vor der Ausführung auf die notwendige Form überprüft, um die Robustheit zu erhöhen.
Das System soll, vor allem für Mitglieder, selbsterklärend sein.	++	Es wurde verstärkt bei den Verwaltungsfunktionen darauf geachtet, ein selbsterklärendes System zu entwickeln. Außerdem wurden Analogien zu existierenden Systemen oder Konventionen eingebettet, um Mitgliedern die Verwendung zu vereinfachen.

Tabelle 7.3.: Anforderungsabgleich entstandener Anforderungen (Forts.)

8 Zusammenfassung

Abschließend erfolgt eine Rekapitulation der vorgestellen Themen, Zielsetzungen und Gedanken dieser Arbeit. Das Kapitel 8.1 widmet sich dabei der Zusammenfassung der vorgestellten Inhalte und Problemstellungen. Im anschließenden Kapitel 8.2 folgt ein Ausblick über die Thematik der Applikationserstellung für existierende Plattformen und der Entwicklung mobiler Anforderungen.

8.1. Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde eine Schnittstelle für ein existierendes Enterprise Content-Management-System konzipiert und damit eine mobile Applikation realisiert. Dazu wurde im **Kapitel 1** die Problemstellung der plattformspezifischen Ausgabe von Inhalten mittels Content-Management-Systemen erarbeitet. Die Einarbeitung in die Problematik der Aufgabenstellung wurde in **Kapitel 2** mit der Recherche von existierenden Lösungen und verwandten Arbeiten komplettiert.

Im Folgenden **Kapitel 3** wurde nach dem Referenzmodell des Usability Engineering [Off] eine Anforderungsanalyse durchgeführt. Dabei mussten existierende Aufgaben der *Website* der *ANC-BWN* ebenso analysiert werden, wie die durch die Entwicklung für mobile Endgeräte entstandenen Aufgaben und die Randbedingungen.

Das **Kapitel 4** widmete sich anschließend der Erstellung einer Systemarchitektur. Dafür wurde ein Anwendungskontext erstellt, der die Anforderungen aus Kapitel 3 in Anwendungsfälle transferiert hat. Die entstandenen Anwendungsfälle bildeten dabei die Basis für die verteilten Architekturbestandteile, die autonom existieren sollten. Bei der Erstellung der Architekturbestandteile, ebenso wie der Entwicklung der gesamten Systemarchitektur, wurde dabei besonders auf die Robustheit der einzelnen Komponenten geachtet. Dies war wegen der losen Kopplung einzelner Bestandteile besonders notwendig. Außerdem wurde bei der Konzipierung der mobilen

Applikation für die Verwendung von Webtechnologien entschieden, um das Cordova-Framework verwenden zu können. Dabei wurden weitere Frameworks eingebunden, die für das Look-And-Feel einer nativen App und die Template-Kompilierung zuständig sind.

Bei der Implementierung in **Kapitel 5** wurden sodann die verwendeten Technologien vorgestellt und die darauf basierend strukturelle Vereinbarung getroffen, um die Kommunikation von Schnittstelle und mobiler Applikation zu gewährleisten. Die Struktur der Antwort des Servers, die in öffentliche und zugriffsbeschränkte Bereiche geteilt wurde, war dabei besonders zu spezifizieren. Darauf aufbauend wurde die *RESTful API* erstellt, deren interner Aufbau wegen der explizit geforderten Robustheit und Benutzergruppentrennung genauer betrachtet wurde. Es folgte der Aufbau der mobilen Applikation, wobei die wichtigsten Mechanismen der eingebundenen Frameworks erläutert wurden. Mit diesem wissen wurde daraufhin ein elementarer Aspekt der Implementierung der Applikation genauer betrachtet, bei dem es sich um die Einbettung von template-basierenden Inhalten in einen statisch orientierten Seitenaufbau handelte.

Um das Resultat der Entwicklung vorzustellen, widmete sich das **Kapitel 6** dem Walkthrough durch die mobile Applikation. Dabei wurde zu Beginn das Grundgerüst des User Interface vorgestellt, gefolgt von den unterschiedlichen Dialogen und Benachrichtigungen im System. Anschließend wurden die differenziert behandelten Bereiche der Applikation vorgestellt und mit Bildschirmausschnitten mit einem Smartphone visualisiert.

Im letzten Kapitel wurden die Anforderungen aus Kapitel 3 mit dem entstandenen Entwicklungsergebnis abgeglichen. Der Abgleich wurde dabei mit einem einfachen Bewertungsschlüssel und einem Kommentar realisiert.

8.2. Ausblick

Während der Erarbeitung der Problemstellung wurde deutlich, dass sich die Verwendung des Cordova-Frameworks für einfache Informationssysteme, die mit existierenden Plattformen im World Wide Web kommunizieren, ideal eignet. Die steigende Mobilität wird von klassischen Content-Management-Systemen in Zukunft fordern, Lösungen oder Schnittstellen für mobile Anwendungen anzubieten. Dieser Umbruch, der durch die Verbreitung mobiler Endgeräte ausgelöst wurde, spiegelt sich auch in weiteren Anwendungskontexten [GPSR13] [GSP+14] [SSP+13] wieder. Aufgrund der großen Konkurrenz am Markt der mobilen Endgeräte und der aktuellen Marktdominanz von Android und iOS (Abbildung 3.2), werden in den kommenden Jahren weitere mobile Betriebssysteme entstehen und eine größere Vielfalt auf dem Markt hervorbringen. Daher ist die Entwicklung einer Applikation mit einem Cross-Framework wie Cordova die geeignete Wahl, um mobile Inhalte für ein breites Anwenderspektrum zu veröffentlichen. Die Tatsache, dass jedes Betriebssystem Webtechnologien unterstützt und bestehende Content-Management-Systeme auf Webtechnologien basieren, unterstreicht diese Entwicklung. Es ist jedoch zu hervorzuheben, dass sich Cross-Frameworks auch weiterhin nicht für komplexere Szenarien eignen (Kapitel 5.1). Im Allgemeinen werden Web Services eine zunehmende Rolle im Alltag von Privatanwendern, als auch von Arbeitgebern, eine Rolle spielen, weshalb Infrastrukturen für Service-orientierte Architekturen [BBP09] immer notwendiger werden.

Abkürzungsverzeichnis

AJAX Asynchronous JavaScript and XML

ANC-BWN Arbeitsgemeinschaft niedergelassener Chirurgen Baden-Württemberg Nord 2

API Application Programming Interface 1

App Applikation für mobile Betriebssysteme 2

Button Interaktionselement

CMS Content-Management-System 1

CRUD Create, Read, Update and Delete

CSS Cascading Style Sheet 23

E-CMS Enterprise Content-Management-System 1

FTP File Transfer Protocol

HTML Hypertext Markup Language 18

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation 5

mobile App Applikation für mobile Betriebssysteme 1

PHP Personal Home Page Tools

REST Representational State Transfer 5

RESTful API *API*, basierend auf der *REST* Architektur 6

RSS Rich Site Summary

UI User Interface 9

UML Unified Modeling Language 15

URL Uniform Resource Locator

Widget Bedien- und Steuerelement der grafischen Benutzeroberfläche 8

WWW World Wide Web

Website Internetplattform 2

XML Extensible Markup Language

Literaturverzeichnis

[BBP09] Stephan Buchwald, Thomas Bauer, and Rüdiger Pryss. IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung. In *13. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW'09)*, number P-144 in Lecture Notes in Informatics (LNI), pages 524–543. Koellen-Verlag, March 2009.

[BN] Arbeitsgemeinschaft Niedergelassener Chirurgen BW-Nord. Internetauftritt des ANCBWN. `https://www.anc-bwn.de/`. Letzter Abruf: 25.05.2014.

[Foua] The Apache Software Foundation. Apache Cordova. `http://cordova.apache.org/`. Letzter Abruf: 20.05.2014.

[Foub] The Python Software Foundation. Welcome to Python.org. `https://www.python.org/`. Letzter Abruf: 25.05.2014.

[Gmba] Statista GmbH. Anteil der Nutzer des mobilen Internets via Smartphone in Deutschland in den Jahren 2008 bis 2013. `http://de.statista.com/statistik/daten/studie/197383/umfrage/mobile-internetnutzung-ueber-handy-in-deutschland/`. Letzter Abruf: 20.05.2014.

[Gmbb] Statista GmbH. Prognose zu den Marktanteilen der Betriebssysteme von Smartphones weltweit von 2014 und 2018. `http://de.statista.com/graphic/1/182363/prognostizierte-marktanteile-bei-smartphone-betriebssystemen.jpg`. Letzter Abruf: 20.05.2014.

[GPSR13] Philip Geiger, Rüdiger Pryss, Marc Schickler, and Manfred Reichert. Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices. Technical Report UIB-2013-09, University of Ulm, Ulm, October 2013.

[GSP+14] Philip Geiger, Marc Schickler, Rüdiger Pryss, Johannes Schobel, and Manfred Reichert. Location-based Mobile Augmented Reality Applications: Challenges, Examples, Lessons Learned. In *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, pages 383–394, April 2014.

[Inca] DreamFactory Software Inc. DreamFactory Services Platform. `http://www.dreamfactory.com/`. Letzter Abruf: 25.05.2014.

[Incb] Wikimedia Foundation Inc. Abbildung: Funktionsweise von PHP. `http://upload.wikimedia.org/wikipedia/commons/6/67/PHP_funktionsweise.svg`. Letzter Abruf: 30.05.2014.

[Incc] Wikimedia Foundation Inc. Abbildung ohne Titel. `http://upload.wikimedia.org/wikipedia/commons/b/b9/USArmyTrauma.jpg`. Letzter Abruf: 31.05.2014.

[Incd] Wikimedia Foundation Inc. Abbildung ohne Titel. http://upload.wikimedia.org/wikipedia/commons/1/16/Homer,_Arzt,_Merkur.jpg. Letzter Abruf: 31.05.2014.

[Ince] Wikimedia Foundation Inc. Artikel: CRUD. http://de.wikipedia.org/wiki/CRUD. Letzter Abruf: 20.05.2014.

[Incf] Wikimedia Foundation Inc. Artikel: Hosting. http://de.wikipedia.org/wiki/Hosting. Letzter Abruf: 25.05.2014.

[Incg] Wikimedia Foundation Inc. Artikel: HTTP-Statuscode. http://de.wikipedia.org/wiki/HTTP-Statuscode. Letzter Abruf: 25.05.2014.

[Inch] Wikimedia Foundation Inc. Artikel: JavaScript Object Notation. http://de.wikipedia.org/wiki/JavaScript_Object_Notation. Letzter Abruf: 20.05.2014.

[Inci] Wikimedia Foundation Inc. Artikel: Representational State Transfer. http://de.wikipedia.org/wiki/Representational_State_Transfer. Letzter Abruf: 20.05.2014.

[Incj] Wikimedia Foundation Inc. Artikel: Responsive Webdesign. http://de.wikipedia.org/wiki/Responsive_Webdesign. Letzter Abruf: 20.05.2014.

[Inck] Wikimedia Foundation Inc. Artikel: User Experince - Ziele und nähere Definition. http://de.wikipedia.org/wiki/User_Experience. Letzter Abruf: 20.05.2014.

[Knu] Jeff Knupp. Sandman - Your Database In Your Browser. http://www.sandman.io/. Letzter Abruf: 25.05.2014.

[Ltda] Canonical Ltd. Artikel: URLDispatcher. https://wiki.ubuntu.com/URLDispatcher. Letzter Abruf: 28.05.2014.

[Ltdb] Kewnode Ltd. Apify on GitHub. https://github.com/fedecarg/apify-library. Letzter Abruf: 25.05.2014.

[Off] Michael Offergeld. Lehrveranstaltung: Usability Engineering, Universität Ulm. http://www.uni-ulm.de/in/mi/mi-lehre/mi-vorlesungsarchiv/2012ws/usability-engineering.html. Letzter Abruf: 17.05.2014.

[Par] Helmuth Partsch. Lehrveranstaltung: Softwaretechnik 1, Universität Ulm. https://www.uni-ulm.de/in/pm/lehre/ws2011/swt1.html. Letzter Abruf: 19.05.2014.

[Pix] Hans Braxmeier und Simon Steinberger Pixabay.com. Abbildung ohne Titel. http://pixabay.com/de/arzt-arztpraxis-praxis-wartezimmer-73117/. Letzter Abruf: 31.05.2014.

[PLRH12] Rüdiger Pryss, David Langer, Manfred Reichert, and Alena Hallerbach. Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, number 132 in LNBIP, pages 43–54. Springer, September 2012.

[PMLR14] Rüdiger Pryss, Nicolas Mundbrod, David Langer, and Manfred Reichert. Supporting medical ward rounds through mobile task and process management. *Information Systems and e-Business Management*, pages 1–40, 2014.

[PMR13] Rüdiger Pryss, Steffen Musiol, and Manfred Reichert. Collaboration Support Through Mobile Processes and Entailment Constraints. In *9th IEEE Int'l Conference*

on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'13). IEEE Computer Society Press, October 2013.

[PTKR10] Rüdiger Pryss, Julian Tiedeken, Ulrich Kreher, and Manfred Reichert. Towards Flexible Process Support on Mobile Devices. In *Proc. CAiSE'10 Forum - Information Systems Evolution*, number 72 in LNBIP, pages 150–165. Springer, 2010.

[PTR10] Rüdiger Pryss, Julian Tiedeken, and Manfred Reichert. Managing Processes on Mobile Devices: The MARPLE Approach. In *CAiSE'10 Demos*, June 2010.

[Rei] Manfred Reichert. Lehrveranstaltung: Programmieren von Systemen, Universität Ulm. `http://www.uni-ulm.de/in/iui-dbis/lehre/archiv/ss-11/pvs-ss11.html`. Letzter Abruf: 20.05.2014.

[RPR11] Andreas Robecke, Rüdiger Pryss, and Manfred Reichert. DBIScholar: An iPhone Application for Performing Citation Analyses. In *CAiSE Forum-2011*, number Vol-73 in Proceedings of the CAiSE'11 Forum at the 23rd International Conference on Advanced Information Systems Engineering. CEUR Workshop Proceedings, June 2011.

[SSP+13] Johannes Schobel, Marc Schickler, Rüdiger Pryss, Hans Nienhaus, and Manfred Reichert. Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, pages 509–518, May 2013.

[VAC+08] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, and U. Zdun. *Software-Architektur: Grundlagen - Konzepte - Praxis*. Spektrum Akademischer Verlag, 2008.

[W3T] W3Techs: Q-Success DI Gelbmann GmbH (Co. Ltd.). Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2014. `http://w3techs.com/technologies/overview/programming_language/all`. Letzter Abruf: 31.05.2014.

[Web] Michael Weber. Lehrveranstaltung: Web Engineering, Universität Ulm. `https://www.uni-ulm.de/in/mi/mi-lehre/mi-vorlesungsarchiv/2012ws/web-engineering.html`. Letzter Abruf: 21.05.2014.

[Wen10] Christian Wenz. *JavaScript - das umfassende Handbuch ; [inkl. jQuery ; Einstieg, Praxis, Referenz ; Web 2.0: DOM, CSS, XML, Web Services ; für Einsteiger, Fortgeschrittene und Profis ; DVD-ROM Code-Beispiele, JavaScript-Bibliotheken u.v.m.]*. Galileo Press, Bonn, 2010.

A
Anhang

1. Analyse der Datenbank und Spezifikation der Inhalte

2. Einführung in die Template-Erstellung mit Handlebars.JS an einer beispielhaften Single-Page-Anwendung

3. Struktur der Navigation und User-Interface-Mockups

4. Ausschnitt der Test-Seite für die Schnittstellenüberprüfung

5. Skizze des ersten Implementierungsversuchs

1. Analyse der Datenbank und Spezifikation der Inhalte

Tabellen für die ANC-BWN-App

Stand: 15.05.2014

Welche Tabelle beinhaltet Inhalt eines Menüpunktes?

Menüpunkt	Speichert	Tabelle	UID [1]	PID [2]	HTML-Markup
Home	Text	tt_content	53	31	Nein
Neuigkeit	Neuigkeit	tt_news	1…*	27	Nein
	Kategorien	tt_news_cat	1	27	Nein
Mitglieder	Mitglied	tt_address	1…*	28	Nein
Rundschreiben	Rundschreiben	api_rundschreiben	1…*	-	Nein
Kontakt	Text	tt_content	54	32	Ja
Praxisdarstellungen	Text	tt_content	42	6	Ja
Disclaimer	Text	tt_content	52	30	Ja
Impressum	Text	tt_content	45	3	Ja
Mitgliederbereich	Text	tt_content	47	24	Nein
	User	fe_users	1…*	23	Nein

[1] **UID = Unique ID** – eindeutig vergebene Identifikationsnummer für jedes Element und PrimaryKey

[2] **PID = Page ID** – referenziert die Seite, auf der Elemente (eindeutig mit UID) stehen

Für Applikation interessante Spaltennamen

Menüpunkt	Tabelle	Spalten [1]
Home	tt_content	***uid***, pid, tstamp, ***bodytext***, ***image***
Neuigkeiten	tt_news	***uid***, pid, tstamp, crdate, cruser_id, deleted, hidden, ***title***, datetime, ***short***, author, category
Neuigkeit	tt_news_cat	***uid***, pid, tstamp, ***title***
Neuigkeit	tt_news	***uid***, pid, tstamp, crdate, cruser_id, editlock, deleted, hidden, ***title***, datetime, ***image***, ***short***, ***bodytext***, author, category, ***news_files***
Mitglieder	tt_address	***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, ***uid***, pid, hidden, ***name***, gender, ***first_name***, ***middle_name***, ***last_name***, title, email, phone, mobile, www, ***address***, company, city, zip, region, country, ***image***, fax, deleted, tstamp, ***description***
Rundschreiben	api_rundschreiben	**uid**, **title**, crdate, cruser_id, author, deleted, **bodytext**, file_url
Mitgliederbereich	tt_content	***uid***, pid, tstamp, ***bodytext***
	fe_users	***uid***, pid, tstamp, username, password, ***usergroup***, crdate, cruser_id

[1] Felder, die keinen Default-Wert besitzen, sind ***fett und kursiv***

Spalten ohne DEFAULT-Flag

Tabelle	Spalten
tt_content	uid, bodytext, image, imagecaption, media, records, pages, image_link, altText, titleText, longdescURL, multimedia, pi_flexform, l18n_diffsource
tt_news	uid, title, image, imagecaption, imagealttext, imagetitletext, short, bodytext, news_files, links, keywords
tt_address	uid, tx_rggooglemap_lng, tx_rggooglemap_lat, tx_rggooglemap_cat2, tx_rggooglemap_ce, name, first_name, middle_name, last_name, address, image, description
api_rundschreiben	**uid**, **title**, **bodytext**
fe_users	uid, usergroup, uc, image, TSconfig

Tabellen für die ANC-BWN-App

Stand: 15.05.2014

Benötigte Spalten für Tabellenoperationen

(ohne username/password Berücksichtigung für den Mitgliederbereich)

Tabelle	Operation	Spalten (Daten) [1]	Spalten (Zuordnung) [1]
tt_content	**SELECT**	***bodytext***, ***image***	***uid*** = 53
	SELECT ⇨ private	***bodytext***	***uid*** = 47
tt_news_cat	**SELECT**	***title***	***uid*** = 1...*
tt_news	**SELECT** (Alle) ⇨ public	***uid***, crdate, cruser_id, ***title***, datetime, ***short***, author, category	pid = 27 hidden = 0 deleted = 0
	SELECT (Alle) ⇨ private	***uid***, crdate, cruser_id, deleted, hidden, ***title***, datetime, ***short***, author, category	pid = 27
	SELECT (Einzel) ⇨ public	crdate, cruser_id, ***title***, datetime, ***image***, ***short***, ***bodytext***, author, category, ~~***news_files***~~	***uid*** = 1...* pid = 27 hidden = 0 deleted = 0
	SELECT (Einzel) ⇨ private	crdate, cruser_id, editlock, deleted, hidden, ***title***, datetime, ***image***, ***short***, ***bodytext***, author, category, ~~***news_files***~~	***uid*** = 1...* pid = 27
	INSERT	pid, tstamp, crdate, cruser_id, editlock, hidden, ***title***, datetime, ***image***, ***short***, ***bodytext***, author, category, ~~***news_files***~~	---
	UPDATE (Pre)	editlock	***uid*** = 1...* pid = 27
	UPDATE (Data)	editlock, deleted, hidden, ***title***, datetime, ***image***, ***short***, ***bodytext***, author, category, ~~***news_files***~~	***uid*** = 1...* pid = 27
	UPDATE (Delete)	editlock, deleted, hidden	***uid*** = 1...* pid = 27
	DELETE	---	***uid*** = 1...* pid = 27
tt_address	**SELECT** (Alle) ⇨ public	***uid***, ***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, ***name***, ***first_name***, ***middle_name***, ***last_name***, phone, www, ***address***, city, zip, fax	pid = 28 hidden = 0 deleted = 0
	SELECT (Alle) ⇨ private	***uid***, ***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, hidden, ***name***, ***first_name***, ***middle_name***, ***last_name***, phone, www, ***address***, city, zip, fax, deleted	pid = 28
	SELECT (Einzel) ⇨ public	***uid***, ***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, ***name***, ***first_name***, ***middle_name***, ***last_name***, title, email, phone, www, ***address***, company, city, zip, region, country, ***image***, fax, ***description***	***uid*** = 1...* pid = 28 hidden = 0 deleted = 0
	SELECT (Einzel) ⇨ private	***uid***, ***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, hidden, ***name***, ***first_name***, ***middle_name***, ***last_name***, title, email, phone, www, ***address***, company, city, zip, region, country, ***image***, fax, deleted, ***description***	***uid*** = 1...* pid = 28
	INSERT	***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, pid, hidden, ***name***, ***first_name***, ***last_name***, title, email, phone, www, ***address***, company, city, zip, region, country, ***image***, fax, deleted, tstamp, ***description***	---

Tabellen für die ANC-BWN-App

Stand: 15.05.2014

Tabelle	Operation	Spalten (Daten) [1]	Spalten (Zuordnung) [1]
	UPDATE (Data)	***tx_rggooglemap_lng***, ***tx_rggooglemap_lat***, tx_rggooglemap_display, hidden, ***name***, ***first_name***, ***last_name***, title, email, phone, www, ***address***, company, city, zip, region, country, ***image***, fax, deleted, ***description***	***uid*** = 1…* pid = 28
	UPDATE (Delete)	tx_rggooglemap_display, hidden, deleted	***uid*** = 1…* pid = 28
	DELETE	**---**	***uid*** = 1…* pid = 28
fe_users	**SELECT** (User)	***uid***	username = ??? password = ???
	SELECT (Einzel)	username, password, crdate	***uid*** = 1…* pid = 23 username = ??? password = ???
	SELECT (Alle)	***uid***, username, password, ***usergroup***, crdate, cruser_id	pid = 23
	INSERT	***uid***, pid, tstamp, username, password, ***usergroup***, crdate, cruser_id	---
	UPDATE	username, password, ***usergroup***, crdate, cruser_id	***uid*** = 1…* pid = 23
	DELETE	---	***uid*** = 1…* pid = 23
api_rundschreiben	**SELECT** (Alle) ⇨ private	**uid**, **title**, crdate, cruser_id, author, deleted, **bodytext**, file_url	---
	INSERT	**title**, crdate, cruser_id, author, deleted, **bodytext**, file_url	**---**
	UPDATE (Data)	**title**, crdate, cruser_id, author, deleted, **bodytext**, file_url	***uid*** = 1...*
	UPDATE (Delete)	deleted	***uid*** = 1...*
	DELETE	**---**	***uid*** = 1…*

[1] Felder, die keinen Default-Wert besitzen sind ***fett und kursiv***

Felder, die eine besondere Kennzeichnung in der App-Ansicht benötigen, sind unterstrichen

Felder, die evtl. nachgerüstet werden, sind ~~durchgestrichen~~

2. Einführung in die Template-Erstellung mit Handlebars.JS an einer beispielhaften Single-Page-Anwendung

Beispielhafte Single-Page-Anwendung unter Einsatz der HandleBars.js- und jQuery-Bibliothek

Michael Stach

25.02.2014

Inhaltsverzeichnis

1 Was ist HandleBars.js?

„Handlebars provides the power necessary to let you build semantic templates effectively with no frustration."

Quelle: http://handlebarsjs.com/

„Handlebars.s is a compiler written in JavaScript. So, you can include it into your HTML documents like any other JavaScript library like JQuery. Basically, it compiles and processes any HTML with Handlebar's expressions you give it and outputs dynamic content."

Quelle: http://jaskokoyn.com/handlebars-js-tutorial-series/

2 Anforderungen an die Beispielanwendung

2.1 Funktionale Anforderungen

- Anzeige einer Nachrichtenübersicht
- Auswahl und Anzeige einzelner Nachrichten
- AJAX-getriebene Datenhaltung unter Verwendung von JSON

2.2 Nicht funktionale Anforderungen

- Anpassung der Darstellung an die Umgebungsauflösung
- minimaler und darstellungsunabhängiger Code
- automatisierte GUI-Erstellung durch Templates
- semantische Oberflächen-Generierung
- Modularer und übersichtlicher Aufbau der Anwendung

2.3 Plattformabhängige Einschränkungen

Es ist anzumerken, dass der Browser, durch die Cross-Site-Skripting (XSS) Einstellungen, das Nachladen von Daten einer anderen Domain als der eigenen, verhindert. Deshalb wurde in diesem Beispiel die Ausgabe des Servers (ein json-String) in dem lokalen Verzeichnis `/json` gespeichert. Der Aufruf auf einen anderen Server wäre jedoch, bis auf die geänderte URL, fast identisch. Unter Verwendung des `PhoneGap-Frameworks` ist es möglich alle Domains, auf die man in der Applikation zugreifen möchte, auf eine `Domain Whitelist` zu setzen. Dadruch wird für diese Domains die XSS-Einstellung verworfen und ein AJAX-Aufruf möglich!

2.3.1 Hinweis zu den verwendeten Daten

Zur Verdeutlichung verschiedener Problemstellungen der Ergebnis-Aufbereitung durch `HandleBars.js`, wurden die nicht-öffentliche Nachrichten (Mitglieder-intern) des ANC-BWN verwendet, da diese eine größere Breite an Fehlermöglichkeiten bietet. Beispielsweise wurden einzelne Felder wie `author` oder `short` nicht ausgefüllt!

3 Inhalt der Beispielanwendung

Die Anwendung kann in 3 modulare Bestandteile gegliedert werden:

Umgebung Dies ist die Hülle, in der die Daten aufbereitet werden. Dies umfasst die Hauptseite (`index.html`), welches das Bindeglied dieser Single-Page-Anwendung ist und den Style (`style.css` und `normalize.css`), welcher zur allgemeinen Darstellung von Markup dient.

Templates Sie sind Vorlagen für einzelne datenabhängige Elemente, welche in die Umgebung eingebettet werden.

Steuerung Die Steuerung holt sich angeforderte Daten und füllt damit das Template. Danach bettet sie den kompilierten Inhalt in die Umgebung ein.

3.1 Die Umgebung

Die Umgebung der Anwendung ist eine html5-basierte Hülle, welche alle benötigten Styles, Skripte und Templates einfügt. Es werden darin alle üblichen und notwendigen Meta-Angaben gemacht. Damit mobile Endgeräte die Anwendung korrekt darstellen, muss der Viewport auf die Gerätebreite gesetzt werden und das Skalieren verboten, denn das soll die eigene Style-Definition übernehmen.

Des weiteren muss in der Umgebung der Style normalisiert werden, damit die plattform-spezifische Startkonfiguration vereinheitlicht wird. In der `style.css` werden alle genutzten Elemente der Website konfiguriert. Dabei wird mittels `Media-Queries` das *responsive Design* realisiert, welches das Layout an bestimmten `Breakpoints` umschaltet.

Algorithmus 1 Umgebung der Anwendung

```
<!doctype html>
<html>
   <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, height=device-height,
          initial-scale=1.0, maximum-scale=1.0, target-densityDpi=device-dpi" />

      <title>ANC-BWN: Nachrichten mit Ajax und HandleBarsJS</title>

      <!-- STYLE -->
      <link rel="stylesheet" href="lib/css/normalize.css" media="screen" />
      <link rel="stylesheet" href="lib/css/style.css" media="screen" />

      <!-- JAVASCRIPT --> [...]
      <!-- TEMPLATES --> [...]

   </head>
   <body>
      <div id="wrapper">
         <div id="template-content">
               <header>Nachrichtenübersicht</header>
               <section>
                  <ul id="news">
                     <!-- TEMPLATE HIER EINFÜGEN -->
                  </ul>
               </section>
         </div>
         <footer>Michael Stach - 19.02.2014</footer>
      </div>
   </body>
</html>
```

3.2 Die Templates

Die Templates sind semantik-abhänige Vorlagen, welche Daten im JSON-Format benötigen, um Inhalte zu rendern. Dabei nutzen Sie `Expressions`, welche durch die *HandleBars.JS-Bibliotek* geparsed werden und durch die abgefragten Daten ersetzt werden. In dieser Beispiel-Applikation müssen mehrere Nachrichten dargestellt werden. Mehrere gleich aussehende Nachrichten sollen in einer Vorschau untereinander aufgelistet und durch einen Click als Einzelnachricht dargestellt werden. Überlicherweise wird dies in der *Steuerung* mittels Schleifen realisiert. Dadurch wird jedoch GUI-Markup in der *Steuerung* erzeugt, welches eine **Unabhängigkeit** von View und Controller nicht mehr sicher stellt!

Algorithmus 2 Beispiel Template der Anwendung

```
<header>{{title}}</header>
<section>
   <ul id="news">
      <h4>Datum: {{datum crdate}}</h4>
      {{#if author}}
      <br/>
      <h4>Von: {{author}}</h4>
      {{/if}}
      <p>{{bodytext}}</p>
      <p><a target="_self" onclick="init()">Zurück zur Übersicht</a></p>
   </ul>
</section>
```

3.3 Die Steuerung

Die Steuerung setzt alle einzelnen Komponenten zusammen und führt Aktionen aus. Direkt beim Seitenaufbau fängt die Steuerung das `ready-Event` ab, dass von der jQuery-Bibliothek geworfen wird und führt dann eine initiale Methode aus. Des weiteren werden Datenaufrufe und Änderungen an der GUI hier definiert. Durch die Einführung der Templates kann die Steuerung inhaltunabhänig agieren. Somit kann ohne Änderungen der Steuerung die Oberfläche modifiziert werden. Zusätzlich werden in der Steuerungen Methoden registriert, die bei der Verarbeitung der Templates helfen sollen, sogenannte `Helper` oder auch `Helfer-Funktionen`.

3.4 Problemlösung durch Einsatz von Templates

Durch den Einsatz von Templates können mehrere Problemstellungen in der Verarbeitung von Daten elegant gelöst werden. Im Folgenden werden die, in der Anwendung anfallenden, Problemstellungen mittels `Expressions` der *HandleBars.js-Bibliothek* und eigenen registrierten `Helpers` gelöst. Dabei ist anzumerken, dass vorgefertige `Expressions` mit einer `#` vor dem Namen ausgewiesen werden.

Algorithmus 3 Schema des Arrays

```
"data": [
   {
      "id":8,
      "author": "Sebastian"
   }, {
      "id":8,
      "author": "Gerhard",
      "zahl": 3
   }
]
```

3.4.1 Listendarstellung

Die vom Server gelieferten Daten enthalten einen Array an JavaScript-Objekten. Mittels `{{#each data}} [...] {{/each}}` wird nun für jedes Objekt im Array `data` das Markup zwischen Anfang und Ende des each-Blocks ausgegeben.

Algorithmus 4 Beispiel zu {{#each data}}

```
{{#each data}}
[...]
<h4>{{datum crdate}}</h4>
<h1>{{title}}</h1>
[...]
{{/each}}
```

3.4.2 Fehlerdarstellung

Falls die Daten des Servers keinen `data`-Array enthalten kann mittels `{{else}}` innerhalb des each-Blocks ein Fehler ausgegeben werden.

Algorithmus 5 Beispiel zu {{else}}

```
{{#each data}}
[...]
<h4>{{datum crdate}}</h4>
<h1>{{title}}</h1>
[...]
{{else}}
<h4>Neuigkeiten konnten nicht geladen werden...</h4>
<br/><br/>
<h1>:-(</h1>
{{/each}}
```

3.4.3 Existenzgebundene Darstellung

Werden Daten durch die angegebene `Expression` nicht gefunden, so wird diese einfach nicht gefüllt. Damit keine Lücken in der Darstellung entstehen, ist es in Abhänigkeit von der Existenz des Elements möglich, mittels `{{#if data-element}} [...] {{else}} [...] {{/if}}` Daten auszugeben. Damit können vom Inhalt abhänige Ausgaben erzeugt werden, die auch beliebig geschachtelt werden können. Optional kann der else-Block auch weggelassen werden!

Algorithmus 6 Beispiel zu {{#if}}

```
{{#each data}}
[...]
{{#if author}}
<h4>Von: {{author}}</h4>
{{/if}}
[...]
{{#if short}}
<p>{{short}}</p>
{{else}}
<p>Leider ist kein Inhalt verfügbar!</p>
{{/if}}
[...]
{{/each}}
```

3.4.4 Helfer-Funktionen

Die Bibliothek `HandleBar.js` bietet eine Funktion an, mit welcher man Helfer-Funktionen definieren kann. Diese Helfer-Funktionen können die mitgelieferten `Expressions` erweitern und eine spezifische Ausgabe erzeugen.

Algorithmus 7 Beispielhafte Helfer-Funktion

```
Handlebars.registerHelper('datum', function(date) {
   var datum = new Date(date * 1000);
   return datum.getDay() + "." + datum.getMonth() + "." + datum.getFullYear();
});
```

Das Template ist nun in der Lage auf die Helfer-Funktion mit dem Namen `datum` zuzugreifen. Diese verändert das Datumsformat auf das in Deutschland übliche und wird durch seinen Namen aufgerufen. Zusätzlich benötigt es einen Übergabeparameter, welche danach angegeben werden muss. Die Ausgabe ist der `return`-String der registrierten Helfer-Funktion!

Algorithmus 8 Anwendung einer Helfer-Funktion

```
{{#each data}}
[...]
<h4>{{datum crdate}}</h4>
[...]
{{/each}}
```

3.4.5 Dynamische Parametierung

`HandleBar.js` sucht beim Kompilieren von Templates nach `Expressions` in solchen `{{ }}` Klammern. Dabei ist es egal, ob diese als *Übergabeparameter* von JavaScript-Aufrufen stehen oder sogar in Anführungszeichen! Dies erleichtert die Erstellung von Links ungemein, da man diese Funktion natürlich auch in den each-Blocks verwenden kann und somit für jedes Element aus einem Daten-Array einen passenden Parameter übergeben kann.

Algorithmus 9 Beispiel zur dynamischen Parametisierung

```
{{#each data}}
[...]
<h4>{{datum crdate}}</h4>
<h1>{{title}}</h1>
<p><a onclick="openNews({{uid}})">Spezifische Nachricht anzeigen!</a></p>
{{/each}}
```

Anmerkung: Die Erstellung von Links ist auch mit einer Helfer-Funktion möglich!

3.4.6 Kompilieren und Einfügen von Templates

Die beiden Bibliotheken `jQuery` und `HandleBars.js` ermöglichen mit sehr wenig Zeilen Code den Inhalt zu ändern, in dem ein Template mittels Daten kompiliert wird.

Algorithmus 10 Kompilierung und Einfügen einer Template-Datei

```
// fülle das News-Template und schreibe es ins Document
function writeNewsTemplate(data) {
   var source = $("#news-template").html();
   var template = Handlebars.compile(source);
   $("#news").html(template(data));
}
```

3.4.7 Auswahl der Templates mittels Callback

Dank der `jQuery-Bibliothek` ist es sehr einfach möglich Callbacks beispielsweise für Erfolg- oder Fehler-Fälle zu registrieren. Dies ermöglicht es Methoden, welche zur Kompilierung einzelnen Templates angelegt wurden, in mehreren Abfragen zu verwenden. Ein Fehler-Callback wäre hier ein Beispiel.

Algorithmus 11 Callbacks als Templateauswahl

```
// hole alle News mittels Ajax
ajaxGetLocal(103, writeNewsTemplate, errorNewsTemplate);
```

3.4.8 Ajax-getriebenes Nachladen von Templates

In dieser Single-Page-Anwendung werden die Template Dateien in einem script-Tag in der Umgebungs-Seite gespeichert. Dazu benötigen diese eine eindeutige ID und den TYPE „text/x-handlebars-template".

Algorithmus 12 Deklaration von Templates

```
<script id="news-template" type="text/x-handlebars-template">
[...]
</script>
```

Es ist allerdings auch möglich diese Templates mittels Ajax nachzuladen und wird bei großen Projekten auch empfohlen. Dazu wird lediglich der Pfad und der Dateiname benötigt.

4 Ausblick auf Applikationen mittels jQuery-Mobile

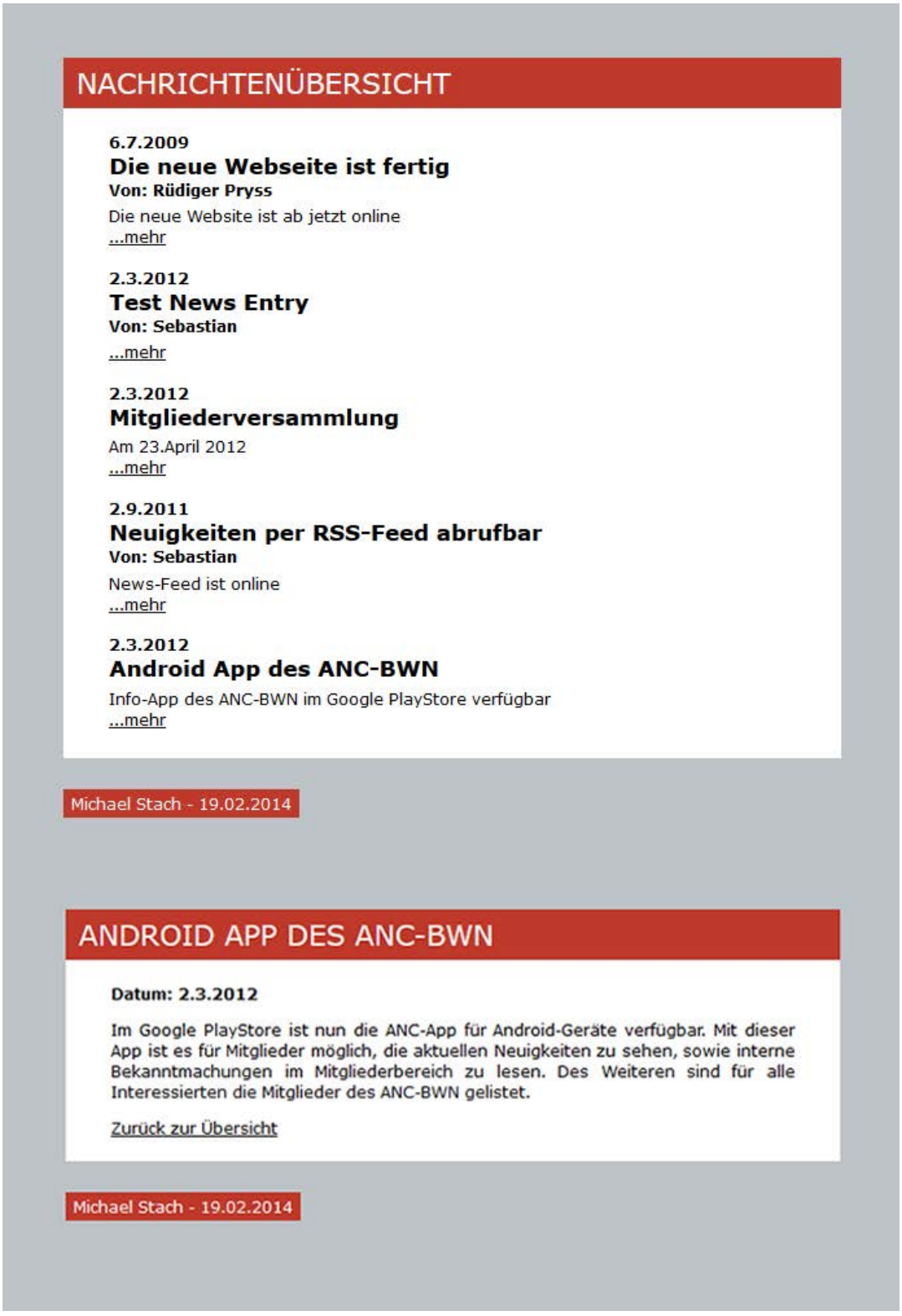
Die Bibliothek `jQuery-Mobile` lädt Daten auf eine ähnliche Art nach, wie es mit `HandleBar.js` realisiert wird. Der Unterschied liegt jedoch darin, dass in `jQuery-Mobile` nur statisches Markup mittels AJAX nachgeladen wird. Es ist aber konzeptionell zu dieser Anwendung ähnlich, da auch dort nur Teile der Umgebungs-Seite geladen werden, nicht jedoch die komplette Seite neu geladen wird. Dies ermöglicht eine effiziente Verzahnung beider Bibliotheken.

Wird auf eine neue `jQuery-Mobile` Seite referenziert, so kann nun, statt eines statischen Markups, ein Template dynamisch ausgewertet und das Ergebnis als statisches Markup nachgeladen werden. Dadurch können interne Methoden der `jQuery-Mobile`-Bibliothek genutzt werden, wie beispielsweise den partiellen Lademechanismus (AJAX) und die Erstellung einer History. Das letztere ist besonders interessant, da AJAX-Aufrufe eigentlich keine neuen Einträge in der Browser-History erstellen!

Weitere Vorteile:

- durchgängige Erscheinung dank vorgefertigter Stylesheets
- Trennung von Inhalt und Steuerung
- `Datenhaltung entfällt` größtenteils, da alle Daten vom Server abgefragt und dynamisch aufbereitet werden

Ausschnitt aus der Beispielanwendung



3. Struktur der Navigation und User-Interface-Mockups

Struktur der Navigation und User-Interface-Mockups

Michael Stach

24.03.2014

Inhaltsverzeichnis

1 Struktur der Navigation

1.1 Identifizierung der Menüpunkte zur passenden Nutzergruppe

Nutzergruppe	Menüpunkte	Inhalt
Gast	Einloggen	Der Nutzer kann sich hier anmelden. Zusätzlich werden Informationen zur Registration und zum Verlust des Passwortes bereitgestellt.
	Aktuelles (öffentlich)	Hier werden alle öffentlichen Neuigkeiten in einer Übersicht aufgelistet. Man kann einzelne Neuigkeiten mittels Auswahl aufrufen.
Gast / Mitglied	Standorte	Hier werden alle Standorte der Mitglieder in einer Karte dargestellt. Mittels Auswahl eines Standortes kann man nähere Informationen erfahren.
	Praxisvorstellung	Dieser Menüpunkt übergibt eine Übersicht aller Mitglieder, die eine Praxisvorstellung/-ansicht zur Verfügung stellen. Bei Auswahl wird eine nähere Ansicht aufgerufen.
	Kontakt	Hier kann der Nutzer alle wichtigen Information über einen Ansprechpartner und Verantwortlichen finden.
	Disclaimer	Darlegung der Rechte.
Mitglied	Aktuelles (geschlossen)	Hier werden alle verfügbaren, auch nicht öffentliche, Neuigkeiten in einer Übersicht angezeigt. Bei Auswahl können einzelne Neuigkeiten komplett angezeigt werden.
	Rundschreiben	Diese Rubrik bietet eine Auswahl an Rundschreiben für die Mitglieder an. Bei Auswahl werden diese angezeigt.
	Mein Profil*	Hier können Mitglieder ihre Profildaten, welche öffentlich in der Karte angezeigt werden, ändern. Nutzerdaten wie Username oder Passwort können nicht geändert werden (Hinweis).
	Mitglieder*	In dieser Rubrik werden alle Mitglieder in einer Liste angezeigt. Bei Auswahl wird ähnlich zum Menüpunkt „Mein Profil" ein Formlar angezeigt, mit welchem man Profildaten aller Nutzer ändern kann.
	Ausloggen	Mit Klick auf diesen Button wird ein Mitglied wieder zum Gast.

* Die Authorisierung wird vom serverseitigem Skript übernommen.

1.2 Gruppierung der Menüpunkte

Nutzergruppe	Gruppenname	Menüpunkte
Gast	Neuigkeiten	Start
		Aktuelles
	Mitglieder	Standorte
		Praxisvorstellung
	Verwaltung	Einloggen
	Impressum	Kontakt
		Disclaimer
Mitglied	Neuigkeiten	Start
		Aktuelles
		Rundschreiben
	Mitglieder	Standorte
		Praxisvorstellung
	Verwaltung	Mein Profil
		Mitglieder
		Ausloggen
	Impressum	Kontakt
		Disclaimer

2 User-Interface-Mockups

Die folgenden Mockups haben die Menüstruktur der Nutzergruppe „Mitglieder". Diese zeigt alle möglichen Menüpunkte, welche in dieser Applikation integriert werden. Die Nutzergruppe „Gast" enthält, bis auf den Menüpunkt „Einloggen", keine eigenen Menüpunkte und ist somit in den folgenden Mockups integriert.

2.1 Klassische Tab-Navigation mit Fullscreen-Funktion

Dieses UI-Konzept zeigt den Versuch, die klassische Tab-Navigation, die der Nutzer aus Webauftritten bereits kennt, für die Applikation umzusetzen. Dabei bilden die 4 Gruppen, mit den dazu passenden Icons, als Hauptnavigation für den Nutzer. Diese Bereiche sind eindeutig abgetrennte Bereiche und helfen dem Nutzer bei der Navigation zur Ziel-Seite.

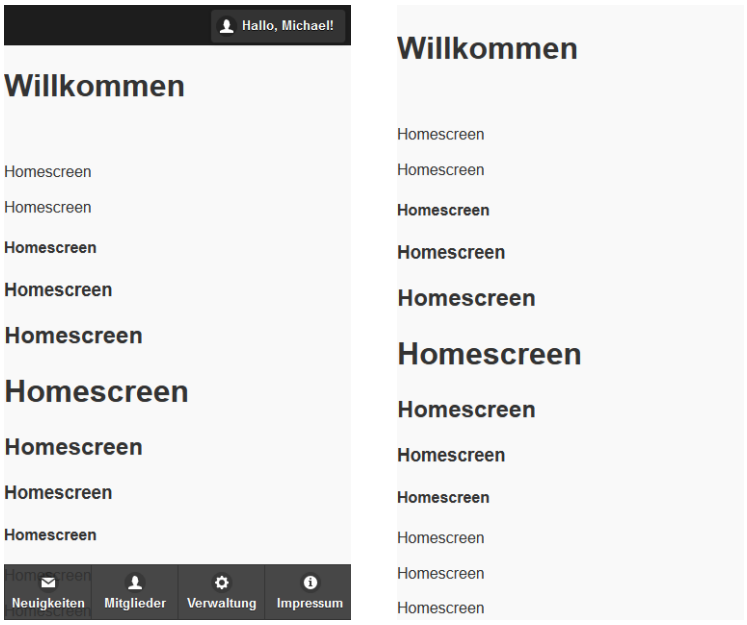


Abbildung 2.1: Homescreen mit Tab-Navigation und im Fullscreen-Modus

Zusätzlich hat der Nutzer eine Kopfleiste im oberen Teil der Seite, in welcher weitere Menüpunkte angezeigt werden. Diese variieren mit der aktuell geöffneten Seite. Im Allgemeinen wird das eigene Profil und ein Zurück-Button angezeigt.

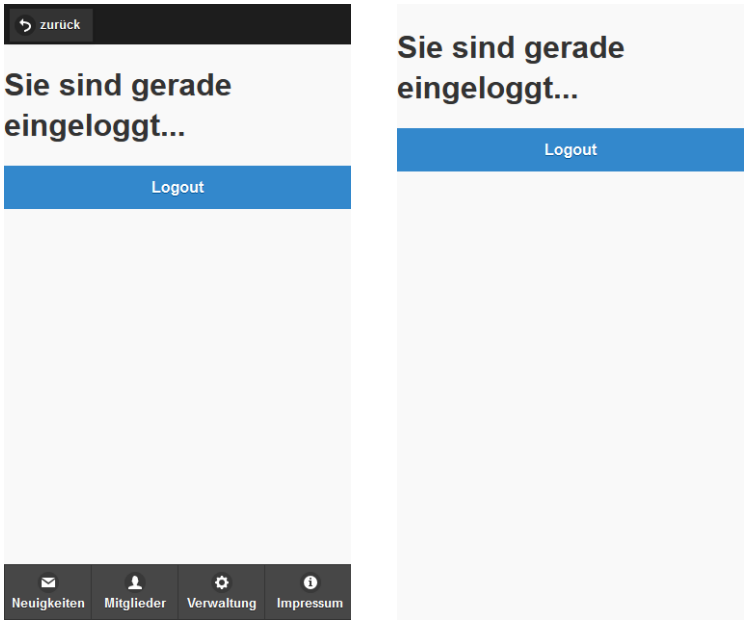


Abbildung 2.2: Logout mit variierender Kopfleiste und im Fullscreen-Modus

Der Homescreen ist der Startpunkt der Applikation und hat keinen eigenen Menüpunkt. Dieser kann in diesem UI-Konzept mittels „Zurück"-Button jederzeit wieder erreicht werden. Falls eine Seite sehr ausgiebigen Inhalt bietet, so kann man mittels „Klick" auf die Seite die Navigation ausblenden. Dieser Fullscreen-Modus ermöglicht es, die Einschränkung der platzintensiven Navigation auszugleichen. Dies ist besonders wichtig, da bei Auswahl einer Gruppe (z.b. „Mitglieder") ein weiteres Menü eingeblendet werden. Dieses Untermenü enthält alle einzelnen Menüpunkte der Gruppe. Bei Auswahl einer Gruppe wird der erste Menüpunkt (links) ausgewählt. Ausgewählte Elemente werden als „aktiv" markiert, um den Nutzer die aktuelle Position darzustellen.

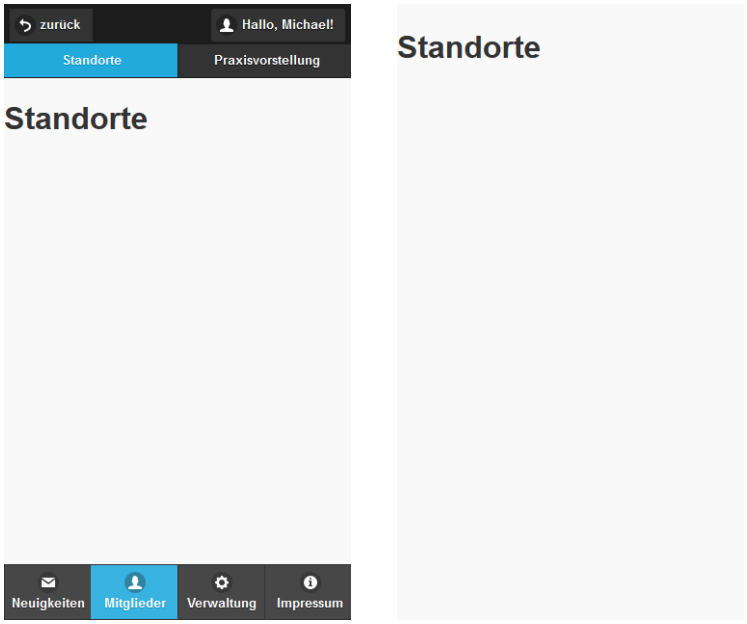


Abbildung 2.3: Mitglieder mit Tab-Navigation und im Fullscreen-Modus

Vorteile	Nachteile
• Nutzer benötigt zur Navigation keine Vorkenntnisse • Fullscreen-Modus für mehr Inhalt auf kleinem Display • Zurück-Funktion • Schnellzugriff auf Profil	• Navigation nimmt sehr viel Platz in Anspruch • Zurück-Funktion ohne History-Darstellung • Möglichkeit des Fullscreen-Modus kann auch verwirren • Fullscreen-Modus muss erst beschrieben werden • kein Platz für weitere (Unter-) Menüpunkte • schlecht erweiterbar

2.2 Verstecktes Menü mit Einschub von links

In diesem UI-Konzept wird der aktuelle Trend des Navigationsaufbau in mobilen Applikationen berücksichtigt. Durch die hohe Auflösung können mehrere Menüpunkte untereinander aufgelistet werden und durch Javascript können selbige ein- und ausgeblendet werden. Das Menü kann durch einen Klick auf den Menü-Button oder durch einen „wisch" (bzw. swipe) nach rechts geöffnen werden. Der Menü-Button befindet sich in der Kopfleiste, in der sich auch eine Schaltfläche für „Einloggen", beziehungsweise das eigene Profil, befindet. Diese ist redundant zu der im Menü gelisteteten Schaltfläche.

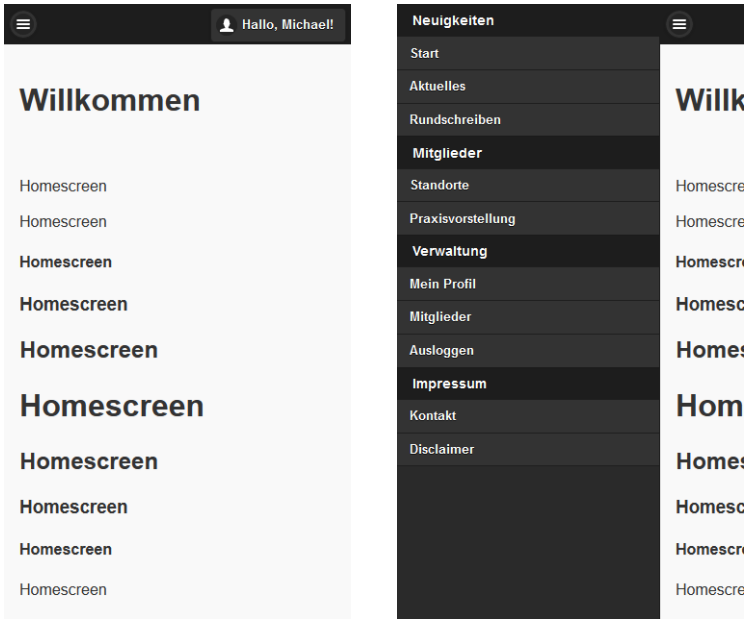


Abbildung 2.4: Homescreen ohne / mit eingeblendetem Menü

Das Menü blendet sich von links ein und schiebt den Inhalt etwas nach rechts. Dadurch behält der Nutzer die Übersicht zum aktuellen Inhalt, hat aber gleichzeitig das, in dieser Auflösung komplette, Menü vor sich. Das Menü ist in Gruppen aufgeteilt, bei denen die Gruppennamen als Überschrift dient. Das Menü ist stets in jeder Seite gleich, somit kann der Nutzer nach initialer Sichtung aller Menüpunkte treffsicher navigieren.

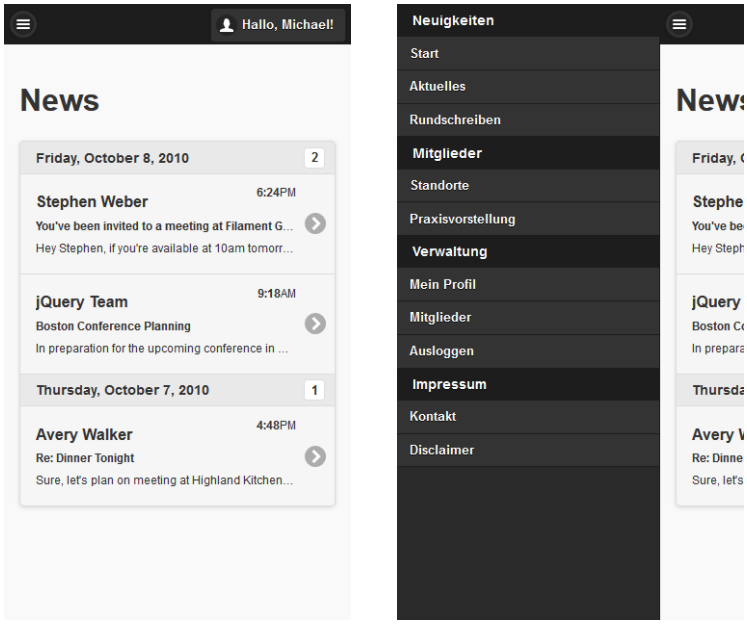


Abbildung 2.5: News ohne / mit eingeblendetem Menü

Vorteile	Nachteile
• Nutzer mobiler Apps mit Navigation vertraut (vorallem Android) • Swipe Bewegung zum Öffnen des Menüs für schnelles Zugriff • Schnellzugriff auf Menü • Leicht mit neuen Menüpunkten erweiterbar • Trennung der Menüpunkte durch Gruppierung • Platz für Inhalt	• Keine Zurück-Funktion in der Kopfleiste (Untermenüpunkte) • initiales Sichten der Liste • aktuelle Position in der Navigation • zurück nur mit „Zurück"-Befehl des Systems System

2.3 Verstecktes Fullscreen-Menü mit Einschub von oben

Dieses UI-Konzept ähnelt dem vorherigem. Dabei wurde aber die Ansicht des Menüs verändert. Das Menü kann wiederum mittels Button-Klick, aber auch mit einem „wisch" (bzw. swipe) nach links oder rechts geöffnet werden. Es umfasst dabei den kompletten Bildschirm und überlegt die aktuelle Ansicht dabei vollständig. Dies bietet Vorteile für Geräte mit geringer Auflösung und ist nicht sehr an den Styleguide einzelner Betriebssysteme gebunden. Eine weitere Neuerung ist, dass der redundante Button für die Profil-Auswahl in das Menü versetzt wurde. Dabei wurde er aber größer und prominenter platziert. Damit bekommt die Trennung zwischen Gast und Mitglied eine stärkere Bedeutung, da sie sichtbarer für die Nutzer wird.

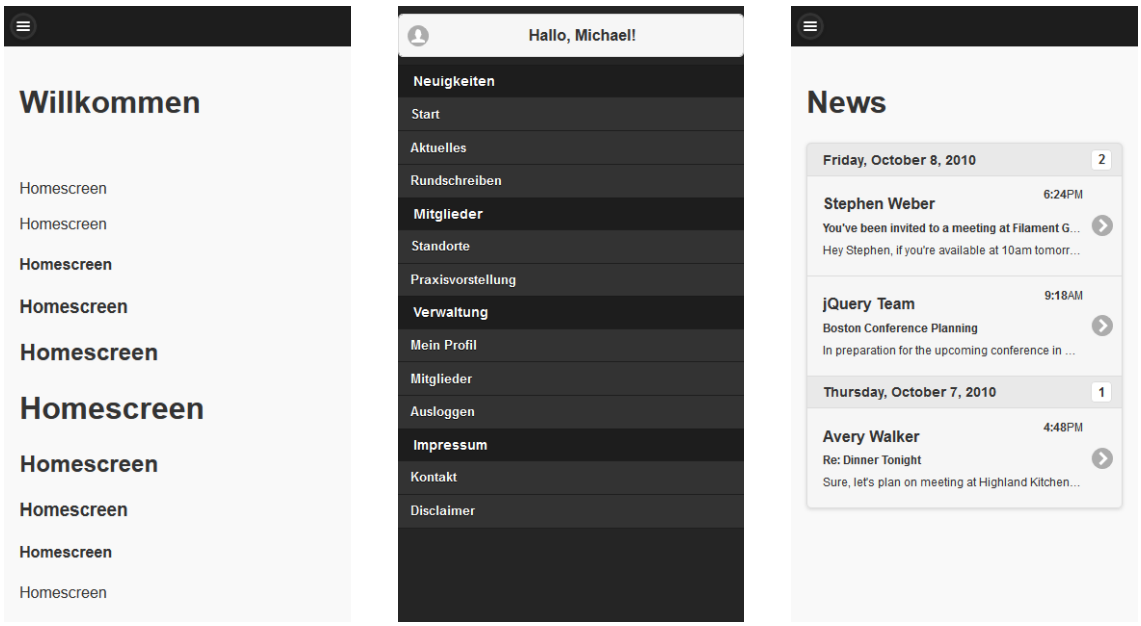


Abbildung 2.6: Homescreen ohne / mit eingeblendetem Menü und News-Seite

Vorteile	Nachteile
• Nutzer mobiler Apps mit Navigation vertraut (vorallem Android) • Swipe Bewegung zum Öffnen des Menüs für schnelles Zugriff • Schnellzugriff auf Menü • Leicht mit neuen Menüpunkten erweiterbar • Trennung der Menüpunkte durch Gruppierung • Platz für Inhalt • bessere Darstellung auf niedrig aufgelösten Displays	• Keine Zurück-Funktion in der Kopfleiste (Untermenüpunkte) • initiales Sichten der Liste • aktuelle Position in der Navigation • zurück nur mit „Zurück"-Befehl des Systems System • keine Sicht auf die geöffnete Seite • starke Einblendung der Nutzerfunktion (für Gäste der Einloggen-Funktion)

4. Ausschnitt der Test-Seite für die Schnittstellenüberprüfung

!!!! Actions werden direkt auf ANC-BWN.DE ausgeführt !!!!

SQL-Methode: SELECT

UID Abfrage zu bestimmter User/PW-Kombination: Action-ID: 100

abschicken

Homescreen geschlossener Bereich: Action-ID: 101

abschicken

Neuigkeiten kurz: Action-ID: 103

abschicken

Neuigkeit lang: Action-ID: 104

abschicken

Mitglieder kurz: Action-ID: 110

abschicken

Mitglieder lang: Action-ID: 111

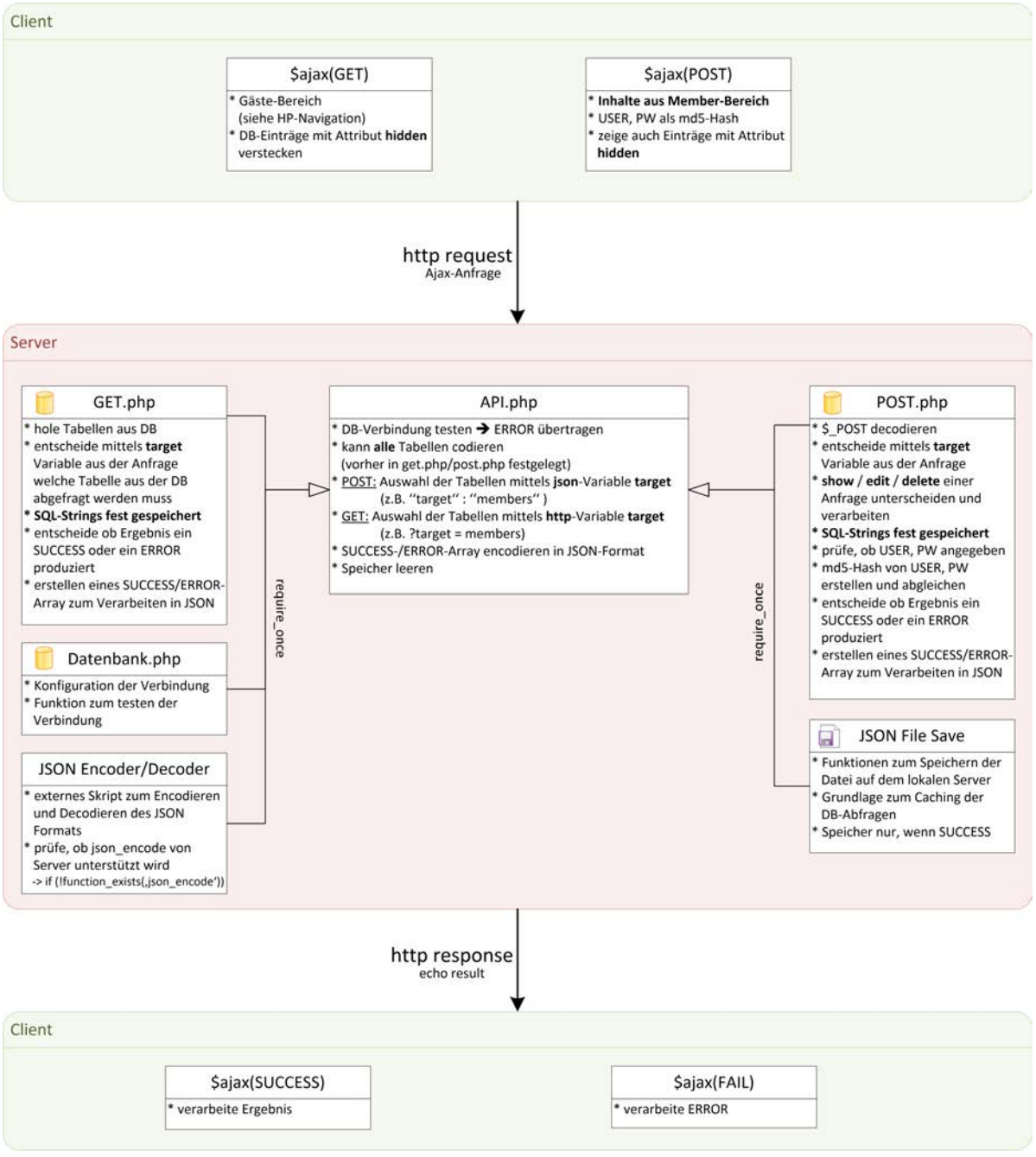
abschicken

Öffentliche Anfragen mit ID: 1 2 3 4 10 11

Ausgaben:

{"response":200, "data":[{"uid":"1","crdate":"1251577053","cruser_id":"1","title":"Die neue Webseite ist fertig","datetime":"1307045760","image":"20001_anc_02.gif","short":"Die neue Website ist ab jetzt online","bodytext":"Bei Fragen, Problemen und W\u00fcnschen wenden Sie sich bitte an den Administrator der Webseite unter ancbwn@elitec.info<\/link> . Herzlichst Ihr Vorstand Dr.med Werner Schebesta.","author":"R\u00fcdiger Pryss","category":"1"},
{"uid":"6","crdate":"1319551826","cruser_id":"1","title":"Neuigkeiten per RSS-Feed abrufbar","datetime":"1335276600","image":"","short":"News-Feed ist online","bodytext":"Ab sofort k\u00f6nnen alle Neuigkeiten auch per RSS-News-Feed empfangen werden.","author":"Sebastian","category":"0"},
{"uid":"9","crdate":"1335280783","cruser_id":"1","title":"Android App des ANC-BWN","datetime":"1335280740","image":"","short":"Info-App des ANC-BWN im Google PlayStore verf\u00fcgbar","bodytext":"Im Google PlayStore ist nun die ANC-App f\u00fcr Android-Ger\u00e4te verf\u00fcgbar. \r\nMit dieser App ist es f\u00fcr Mitglieder m\u00f6glich, die aktuellen Neuigkeiten zu sehen, sowie interne Bekanntmachungen im Mitgliederbereich zu lesen. Des Weiteren sind f\u00fcr alle Interessierten die Mitglieder des ANC-BWN gelistet.","author":"","category":"0"}]}

5. Skizze des ersten Implementierungsversuchs


Client
$ajax(GET)
* Gäste-Bereich (siehe HP-Navigation)
* DB-Einträge mit Attribut hidden verstecken
$ajax(POST)
* Inhalte aus Member-Bereich
* USER, PW als md5-Hash
* zeige auch Einträge mit Attribut hidden
http request
Ajax-Anfrage
Server
GET.php
* hole Tabellen aus DB
* entscheide mittels target Variable aus der Anfrage welche Tabelle aus der DB abgefragt werden muss
* SQL-Strings fest gespeichert
* entscheide ob Ergebnis ein SUCCESS oder ein ERROR produziert
* erstellen eines SUCCESS/ERROR-Array zum Verarbeiten in JSON
Datenbank.php
* Konfiguration der Verbindung
* Funktion zum testen der Verbindung
JSON Encoder/Decoder
* externes Skript zum Encodieren und Decodieren des JSON Formats
* prüfe, ob json_encode von Server unterstützt wird
-> if (!function_exists('json_encode'))
require_once
API.php
* DB-Verbindung testen → ERROR übertragen
* kann alle Tabellen codieren (vorher in get.php/post.php festgelegt)
* POST: Auswahl der Tabellen mittels json-Variable target (z.B. "target" : "members")
* GET: Auswahl der Tabellen mittels http-Variable target (z.B. ?target = members)
* SUCCESS-/ERROR-Array encodieren in JSON-Format
* Speicher leeren
require_once
POST.php
* $_POST decodieren
* entscheide mittels target Variable aus der Anfrage
* show / edit / delete einer Anfrage unterscheiden und verarbeiten
* SQL-Strings fest gespeichert
* prüfe, ob USER, PW angegeben
* md5-Hash von USER, PW erstellen und abgleichen
* entscheide ob Ergebnis ein SUCCESS oder ein ERROR produziert
* erstellen eines SUCCESS/ERROR-Array zum Verarbeiten in JSON
JSON File Save
* Funktionen zum Speichern der Datei auf dem lokalen Server
* Grundlage zum Caching der DB-Abfragen
* Speicher nur, wenn SUCCESS
http response
echo result
Client
$ajax(SUCCESS)
* verarbeite Ergebnis
$ajax(FAIL)
* verarbeite ERROR