ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Evaluation des Smartphone Betriebssystems FirefoxOS anhand der Konzeption und Realisierung einer mobilen Anwendung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Julian Frank
julian-1.frank@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2013

Fassung 9. Januar 2014

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

In unserer heutigen Smartphone-Generation gibt es eine Vielzahl von verschiedenen Geräten und mobilen Betriebssystemen, allen voran Google's Android, Apple's iOS und Microsoft's Windows Phone. Nun veröffentlicht die Mozilla Corporation mit *FirefoxOS* ein open-source Betriebssystem, dessen Benutzeroberfläche und Apps ausschließlich mit offenen Webstandards realisiert sind. Es ist die erste Plattform speziell für diese sogennanten *WebApps*.
Anhand der Konzeption und der Realisierung einer solchen Anwendung untersucht diese Arbeit das Betriebssystem *FirefoxOS*, dessen Aufbau und Funktionsweise. Des Weiteren überprüft sie in wieweit das Betriebssystem mit den marktführenden Konkurrenten mithalten kann. Insbesondere werden *WebApps* näher beleuchtet und Besonderheiten sowie Vor- und Nachteile aufgelistet, um festzustellen, wie Apps mit HTML, CSS und Javascript realisiert werden, die gleichwertig mit Apps proprietäre Systeme sind.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen Bedanken, die mich bei der Fertigstellung dieser Arbeit unterstützt und motiviert haben.

Zunächst möchte ich mich bei Herrn Prof. Dr. Manfred Reichert (Universität Ulm, Institut für Datenbanken und Informationssysteme) für die Begutachtung dieser Bachelorarbeit bedanken.

Mein besonderer Dank gilt meinem Betreuer Herr Marc Schickler, der mir zunächst dieses auf meine Interessen zugeschnittene Thema gefunden, mich auf dem ganzen Weg unterstützt und nicht zuletzt sehr viel Zeit in die Korrektur dieser Arbeit gesteckt hat. Durch den ein oder anderen Anstoß habe ich letztendlich doch immer noch die Lösung auf ein Problem selbst gefunden wo ich bereits aufgegeben hatte.

Zu guter Letzt möchte ich meinen Freunden und Kommilitonen danken, die mich während des Studiums motiviert und unterstützt haben, wenn es mal etwas trocken wurde. Unter diesen gilt mein größter Dank Frau Sultana Chatzivasiliadou, auf deren Hilfe ich jederzeit zählen konnte.

Inhaltsverzeichnis

1

Einleitung

In unserer heutigen Smartphone-Generation ist der blühende Markt der mobilen Betriebssysteme hart umkämpft. Android (Google), iOS (Apple), RIM (BlackBerry), Windows Phone (Microsoft) und Asha (Nokia) sind die wohl bekanntesten, wenn auch längst nicht alle derzeit auf dem Markt befindlichen Betriebssysteme für mobile Geräte.
Und dennoch drängen bereits wieder neue Produkte auf den Markt, wie z.B. das Cloud-basierte Aliyun OS [Ali] oder Ubuntu Touch OS [Ltd]. Im Folgenden soll aber vor allem das von der Mozilla Corporation momentan entwickelte *FirefoxOS* näher begutachtet werden.
Braucht man bei dieser Vielfalt ein weiteres Betriebssystem? Was unterscheidet *FirefoxOS* von anderen mobilen Betriebssystemen? Welche Vorteile bietet *FirefoxOS* gegenüber bereits existierenden Betriebssystemen?

Das bescheidene Ziel von Mozilla ist es mit *FirefoxOS* das mobile Betriebssystem zu *revolutionieren*. Entwickler sowie Benutzer sollen frei von proprietären Plattformen großer Konzerne sein. Dazu setzt *FirefoxOS* vollständig auf offene Webstandards. Obwohl es auf demselben Linux-Kernel wie Android basiert, ist die Benutzeroberfläche sowie sämtliche Apps komplett in HTML5, CSS und Javascript geschrieben.
Dadurch kann praktisch jeder Webentwickler eine App für *FirefoxOS* schreiben, zudem soll es größtmögliche Offenheit und Kompatibilität ermöglichen. Da die Mozilla Foundation zudem eine non-Profit Organisation ist, ist die *FirefoxOS*-Plattform frei und für jedermann zugänglich wie das Web. Und genau das ist es, was sich Mozilla auf die Fahne schreibt. Wie schon beim Firefox Browser soll beim mobilen Betriebssystem der Anwender an erster Stelle stehen und vor allem frei sein von den Beschränkungen und Regeln proprietärer Systeme, wie z.B. iOS. Dabei soll *FirefoxOS* das volle Potential des open Web veranschaulichen. Es ist also nicht einfach nur ein weiteres Betriebssystem für Smartphones, sondern zum einen eine Alternative zu proprietären System wie iOS und Windows Phone und zum anderen eine erste Realisierung der Idee eines webbasierten Handys, bei dem die gesamte Benutzeroberfläche eine WebApp ist, die andere WebApps starten kann.

1.1 Motivation

Mittlerweile ist das erste Smartphone mit *FirefoxOS* auf dem deutschen Markt verfügbar, daher soll in dieser Arbeit das neue Betriebssystem genauer betrachtet werden. Hält es was es verspricht? Wozu ist es tatsächlich fähig? Abgesehen davon wird das Ganze natürlich hauptsächlich aus der Entwickler-Perspektive betrachtet. Wie man mit bekannten Mitteln wie HTML5, CSS und Javascript Apps baut, was eine WebApp von einer normalen Website unterscheidet und wie man sofort selbst Apps für FirefoxOS entwickeln, testen und veröffentlichen kann.
Um dies veranschaulichen und evaluieren zu können, soll dies am Beispiel einer App gezeigt werden. Dazu soll die für iOS bereits existierende DBIS-App [Her] für *FirefoxOS* neu konzipiert und realisiert werden und die daraus resultierenden Erfahrungen als Grundstein für die fundierte Evaluierung des Betriebssystems dienen.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es durch die Konzeption und Realisierung der App sowie durch Vergleiche bereits existierender Systeme das mobile Betriebssystem *FirefoxOS* zu evaluieren und zu bewerten. Dabei sollen Aufbau und Funktionsweise des Betriebssystems, bzw. der Apps deutlich werden. Danach sollten die dadurch gewonnen Ergebnisse sowohl zu einer objektiven Aussage über die Vor-und Nachteile als auch über die Möglichkeiten des Betriebssystems führen. Des Weiteren wird auch das Vorgehen und die Besonderheiten bei der Entwicklung von WebApps für *FirefoxOS* erläutert, sodass anschließend ein Überblick über alle wesentlichen Bestandteile des Betriebssystems existiert.
Insbesondere wird auf den Aufbau und die Funktionsweise der im Laufe dieser Arbeit entwickelten App eingegangen. Diese zeigt nicht nur Besonderheiten des Betriebssystems auf, sondern geht vor allem auf generelle Aspekte bei der Entwicklung mobiler Anwendungen ein. Dazu gehören Design für kleine Displays, responsive Design für verschiedene Displaygrößen und Caching-Mechanismen aufgrund der beschränkten Bandbreite sowie für den Offline-Betrieb.
Dabei wird allerdings nicht auf der Grundlagen der verwendeten Technologien wie HTML5 und CSS eingegangen, dafür aber auf nützliche (und z.T. verwendete) Frameworks und Services sowie weitere Tools.

1.3 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgebaut: Zuerst wird in Kapitel 2 - *Grundlagen* nach einem kurzen geschichtlichen Abriss über *FirefoxOS* dessen Aufbau und Architektur veranschaulicht und erläutert. Daraufhin werden die zur App-Entwicklung spezifischen Grundkenntnisse vermittelt und Hilfsmittel aufgelistet und erklärt. Dazu gehören unter anderem Grundlagen über WebApps, hilfreiche UI-Frameworks für mobile Anwendungen sowie nützliche Entwicklertools wie den Simulator.
Die folgenden drei Kapitel beziehen sich auf die Entwicklung der Anwendung, die aus der Anforderungsanalyse, der Konzeption und Entwicklung sowie dem Anforderungsab-

gleich besteht. Bei der Anforderungsanalyse stehen hierbei für mobile Anwendungen spezifische Anforderungen im Mittelpunkt, deren Umsetzung dann in der Entwicklung beschrieben werden, wobei hier vor allem auf die plattformspezifischen Besonderheiten eingegangen wird. Außerdem wird der Aufbau der App in absteigender Granularität dargestellt und beschrieben. Anschließend werden die zu Beginn definierten Anforderungen mit der Anwendung abgeglichen.

Die Konzeption und Implementierung der Anwendung schließt eine Zusammenfassung ab, welche die bei der Entwicklung entstandenen Ergebnisse nochmals kurz beschreibt. Darauf basierend werden Vorteile sowie Probleme von *FirefoxOS* diskutiert.
In einem Fazit werden daraufhin die daraus resultierenden Schlüsse gezogen. Zuletzt wird ein Ausblick auf (mögliche) zukünftige Entwicklungen gegeben.

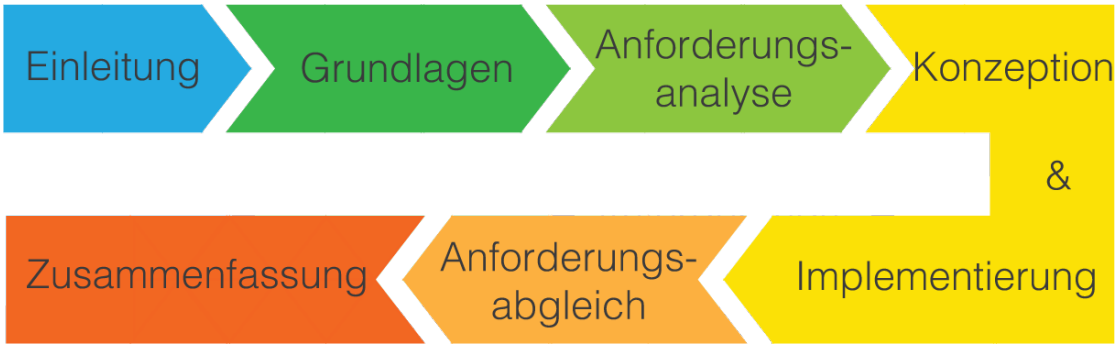

Abbildung 1.1: Aufbau der Arbeit

2

Grundlagen

Um das Betriebssystem und die Entwicklung von Anwendungen für dieses diskutieren zu können, ist ein gewisses Grundwissen vonnöten. Dieses Kapitel beschäftigt sich daher mit der Architektur und dem Sicherheitskonzept von *FirefoxOS*. Außerdem werden die Eigenschaften und Besonderheiten der Anwendungen für das Betriebssystem, den sogenannten *WebApps*, erläutert. Zunächst folgt jedoch ein kurzer geschichtlicher Hintergrund über das noch junge *FirefoxOS*.

2.1 Geschichte

Am 25. Juli 2011 wurde von *Andreas Gal*, dem Leiter der Forschungsabteilung von Mozilla, über eine Mailingliste ein Projekt angekündigt, welches das Ziel hatte ein vollständiges Betriebssystem für das open Web zu entwickeln. Jeder kann somit Anwendungen mit

freien Web-Standards entwickeln und veröffentlichen. Es sollte einen gleichwertigen Ersatz für gängigen mobilen Betriebssysteme, wie Android, iOS und Windows Phone, schaffen, wobei es den Benutzern sowie den Entwicklern größtmögliche Freiheit durch die Unabhängigkeit von proprietären Systemen gibt.

Das Projekt erhielt den Namen *Boot to Gecko* (kurz: B2G), was auf den bereits in Mozilla's Firefox Browser verwendeten HTML-Rendering-Engine verweist. Ein Jahr später (Juli 2012) wurden erste Screenshots veröffentlicht, außerdem gab Mozilla bekannt, dass erste Geräte mit dem neuen Betriebssystem von ZTE und TCL Communication Technology (Alcatel) entwickelt werden sollten. Außerdem sicherten globale Netzanbieter, wie die Deutsche Telekom und Telefónica (O_2), ihre Unterstützung zu [Rel].

Im September 2012 veröffentlichte Mozilla eine erste Vorschau des inzwischen in *FirefoxOS* umbenannten Systems in Form eines Videos, das einige Funktionen, wie Telefonieren, Kontaktverwaltung oder den Browser, zeigt [cha]. Am 3. Dezember wurde die erste Version des *FirefoxOS Simulators* als Erweiterung für den Firefox Browser herausgebracht, um Entwicklern die Möglichkeit zu geben, Apps zu entwickeln und zu testen, ohne ein entsprechendes Gerät zu besitzen (welches zu diesem Zeitpunkt auch noch nicht öffentlich verfügbar war).
Anfang 2013 brachte der spanische Hersteller *Geeksphone* zwei Developer-Preview Modelle mit *FirefoxOS* auf den Markt, welche damit auch die ersten in Deutschland erhältlichen Smartphones mit diesem Betriebssystem waren. Das erste für Endbenutzer gedachte Gerät wurde in Deutschland erst am 15. Oktober veröffentlicht. Das Alcatel One Touch Fire wird derzeit nur von der Telekom Tochter Congstar vertrieben. In Spanien vertreibt die Telefónica als erster Mobilfunkanbieter das ZTE Open bereits seit 2. Juli.

Im Gegensatz zu den Konkurrenten, wie Android und iOS, zielen diese Geräte allerdings bislang auf eine Zielgruppe von Smartphone-Einsteigern ab. Das Alcatel One Touch Fire beispielsweise ist technisch nicht aktuellen Smartphones von z.B. Samsung oder Apple zu vergleichen. So hat dises beispielsweise eine 3,2 Megapixel Kamera, 512MB internen Speicher und ein Display mit einer Auflösung von 480x320 Pixeln. Im

Vergleich dazu hat z.B. Apples iPhone 4S, das vor 2 Jahren erschien, damals schon eine 8 Megapixel Kamera, eine Auflösung von 960x460 Pixeln und war mit einem internen Speicher bis zu 64GB erhältlich. Der größte Unterschied dürfte aber im Preis liegen. Mit 90€ [Con] findet man wohl nicht viele andere Smartphones, die dennoch sämtliche Smartphone-typischen Funktionen mit sich bringen.

2.2 Architektur

Dieses Kapitel beschäftigt sich mit den einzelnen Bestandteilen des Betriebssystems und deren Zusammenspiel ohne dabei detailliert auf spezielle Prozesse und Implementierungen einzugehen. Es wird deutlich gemacht, wie die einzelnen Schichten aufgebaut sind und wie deren Kommunikation untereinander funktioniert.

2.2.1 Terminologie und Übersicht

Folgende Begriffe werden im Verlauf der nachfolgenden Kapitel häufiger auftauchen und sollten deshalb bekannt sein:

Boot to Gecko

Oft nur mit *B2G* referenziert. Der Codename unter dem das *FirefoxOS* - Projekt gestartet wurde, referenziert im Folgenden nicht nur auf das Projekt an sich, sondern vor allem auf das Betriebssystem selbst.

Gaia

Die Benutzeroberfläche von *FirefoxOS*. Alles was der Benutzer sieht, kommt von der Gaia-Schicht. Gaia implementiert den Sperr-und Homebildschirm sowie sämtliche vorinstallierten Apps. Gaia ist vollständig in HTML, CSS und Javascript implementiert und kommuniziert mit der darunterliegenden Schicht ausschließlich über WebAPIs, die wiederum in der Gecko-Schicht implementiert sind.

Gecko

Gecko ist die Runtime von *FirefoxOS*. Es ist diesselbe Rendering-Engine, die schon beim Firefox-Browser verwendet wird. Gecko verarbeitet HTML, CSS und JS-Dateien und stellt diese als formatierten Text, Grafiken und Animationen auf dem Bildschirm dar. Gecko implemetiert außerdem die sogenannten WebAPIs.

WebAPI

WebAPIs ermöglichen die Kommunikation der Anwendungsebene mit der Geräte-Hardware (z.B. Batterie-Status, GPS oder Kamera) sowie Geräte-Daten (z.B. Kontakte oder Kalender). Viele dieser APIs sind noch nicht standardisiert, weshalb Mozilla die Entwicklung in diesem Bereich sehr vorantreibt.

Gonk

Gonk ist die unterste Schicht des Betriebssystems und besteht aus einem Linux-basiertem Kernel und der Hardwareabstraktionsschicht (HAL = Hardware Abstraction Layer).

2.2.2 Anwendungsebene (Gaia)

Die Benutzeroberfläche von B2G ist Gaia. Gaia ist eine Sammlung von vorinstallierten Apps, wie Kalender, Musik, E-Mail, Kontakte usw., sowie der Sperr-und Homebilschirm. Insgesamt orientiert sich die Oberfläche stark an Vorbildern wie iOS und Android. Es gibt einen Sperrbildschirm, nach dem (nach PIN-Eingabe) der Home-Bildschirm erscheint. Hier wird durch Wisch-Gesten zwischen verschiedenen Screens geblättert, auf denen die Apps als Icons dargestellt sind. Die vorinstallierten Apps sind Telefon, Nachrichten, Kontakte, Browser, Kamera, Gallerie, FM Radio, Einstellungen, Marketplace (der Mozilla *App Store*), Maps (Nokias *HERE Maps*), Kalender, Uhrzeit, E-Mail, Musik und Video. Außerdem befinden sich in dieser Schicht sämtliche vom Benutzer installierten Apps, sowie die JS Bibliotheken für Entwickler [gai].

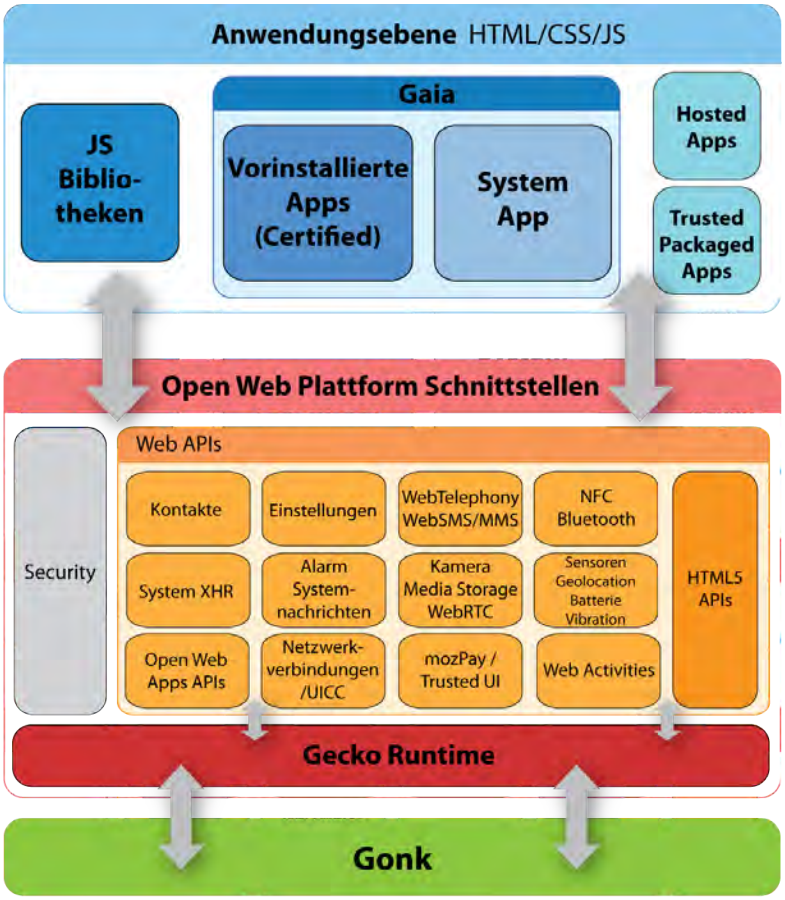

Abbildung 2.1: B2G Architektur

2.2.3 Open Web Plattform Schnittstellen (Gecko)

Gecko ist die Layout-Engine, die auch im Firefox Browser verwendet wird. Außerdem implementiert Gecko die WebAPIs, welche die Schnittstelle zu Gaia darstellen. Sie ermöglichen den Zugriff auf Systemfunktionen, wie Kamera, Bluetooth, Vibration etc., aber auch der Zugriff auf Daten, wie Kontakte, Batteriestatus und Standortbestimmung. Durch die Web Activity API können Aufgaben an andere Apps delegiert werden oder über die Media Storage API z.B. Bilder in die Gallery gespeichert werden.

Wichtigster Bestandteil dieser Schicht ist Laufzeitumgebung von Gecko. Hier werden die Frameworks für die Ausführung der Apps bereitstellt. Außerdem ist Gecko die “Sicherheitsschranke”, welche die darunterliegende Schicht und vor allem das Gerät selbst vor Missbrauch schützt [gecb][geca].

2.2.4 Security

Da FirefoxOS *free open-source software (FOSS)* ist, legt Mozilla besonderen Wert auf Sicherheit bei der Entwicklung. Um das Sicherheitskonzept zu veranschaulichen, werden im Folgenden die verschiedenen Schichten nochmals vereinfacht dargestellt und betrachtet (siehe Abbildung 2.2).

Abbildung 2.2: B2G Schichten

Die entscheidende Rolle in diesem Sicherheitskonzept spielt Gecko. Die Gonk-Ebene hat direkten Zugriff auf die Gerätehardware und liefert deren Funktionen direkt an die Gecko-Schicht. Apps können auf das Gerät nur durch die WebAPIs zugreifen und auch nur dann, wenn Gecko den Zugriff erlaubt. Gecko erzwingt eine Erlaubnis und schützt so vor unautorisiertem Zugriff.
Die Sicherheit fängt schon ganz am Anfang an. *FirefoxOS* wird bereits auf dem Gerät vorinstalliert geliefert. Das original Image kommt von einer vertrauenswürdigen Quelle, üblicherweise vom Erstausrüster des Geräts (OEM = Original Equipment Manufacturer), welcher das Package digital signiert. Weitere Sicherheitsaspekte sind die Einhaltung von

Dateisystem-Privilegien durch Linux's ACLs oder einer read-only Partition für System-Apps (außer während Updates). Außerdem plant Mozilla regelmäßig (alle 3 Monate) Sicherheitsupdates zu veröffentlichen. Diese werden ebenfalls vom OEM getestet und digital signiert.
Aber am wichtigsten für Entwickler dürfte die Sicherheitspolitik der App sein. Je nach Typ hat eine App verschiedene Rechte. Die höchste Vertrauensstufe haben zertifizierte System-Apps (`type = "certified"`) bzw. Services. Dieser Typ ist nicht gedacht für Apps von Drittanbietern. Diese Apps wurden vom OEM oder Systemadmin überprüft. Beispiele für kritische Funktionen, auf die nur diese Apps Zugriff haben, sind SMS, Bluetooth oder Telefon. Die darunterliegende Vertrauensstufe sind privilegierte Apps (`type = "priviliged"`), die als solche von einem autorisiertem Marketplace überprüft und digital signiert wurden. Dies können demnach nur Packaged Apps sein, da sie vom Marketplace signiert und von dort runtergeladen werden müssen, (siehe Kapitel 2.3 WebApps). Alle anderen Apps bzw. Webinhalte, ob installiert oder gehostet, gelten als nicht vertrauenswürdig und haben daher Zugriff auf die wenigsten WebAPIs.
Der Typ der App sowie deren Rechte müssen im Manifest festgelegt werden (siehe Kapitel 2.3 WebApps). Dieses wird vom Marketplace heruntergeladen und auf dem Gerät gespeichert. In Kapitel 2.3.1 WebApps $\rightarrow$ Manifests werden weitere sicherheitsbezogene Felder beschrieben, die hier gesetzt werden können (bzw. müssen).
Weiterhin wird durch *Sandboxing* (siehe Abbildung 2.3) sichergestellt, dass eine App während ihrer Laufzeit nur Zugriff auf die ihr zugesicherten WebAPIs und Daten, wie IndexedDB Datenbanken, Cookies, usw., hat [sec].
Da die Apps so isoliert sind, können diese nicht in Konflikt mit anderen Apps oder deren Daten treten. Außerdem kann eine Anwendung nur über die entsprechende WebAPI auf die Geräte-Hardware zugreifen. Es gibt keine andere Möglichkeit oder Hintertürchen. Bei jedem WebAPI Aufruf wird überprüft, ob die Anwendung die entsprechenden Berechtigungen, also eventuell auch einen privilegierten Status, hat. Apps können nur indirekt über den zentralen B2G-Prozess miteinander Kommunizieren.

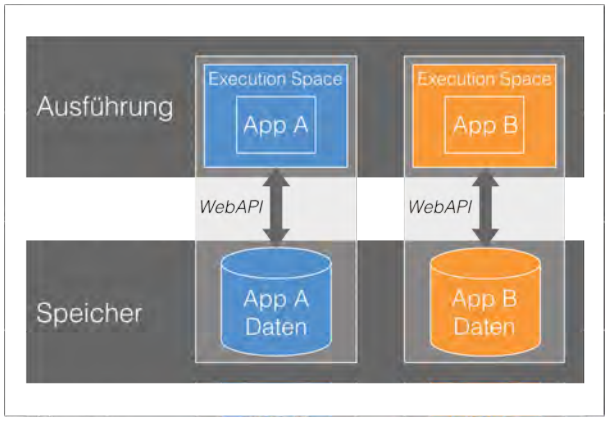

Abbildung 2.3: Sandboxing

2.2.5 Gonk

Die unterste Schicht der *FirefoxOS*-Architektur ist Gonk. Gonk besteht zum einen aus einem Linux Kernel sowie der Hardware Abstraction Layer (HAL). Einige Teile der HAL sind open-source Projekte (siehe Abbildung 2.4), andere, wie Kamera oder GPS, werden von Android übernommen. B2G ist also nicht völlig eigenständig und unabhängig.

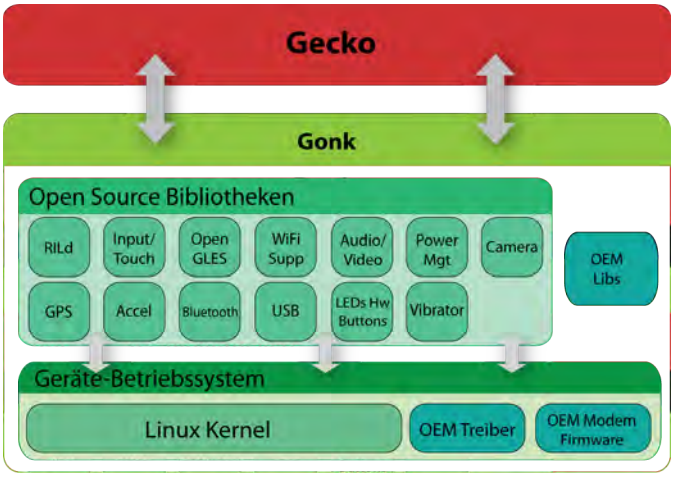

Abbildung 2.4: Bestandteile von Gonk

Gonk ist im Prinzip eine einfache Linux-Distribution und bildet die Verbindung zwischen dem Gerät und Gecko. Gonk stellt die Schnittstellen zur Gerätehardware für Gecko (und nur für Gecko) zur Verfügung, auf die Gecko unter anderen Betriebssystemen keinen Zugriff hat. Im Firefox Browser auf einem PC zum Beispiel, der ebenfalls Gecko verwendet, machen die dort implementierten APIs für z.B. Telefonie keinen Sinn, da der PC nicht telefonieren kann. Die darunterliegende Schicht bietet demnach Gecko auch nicht die entsprechenden Schnittstellen an, wie Gonk es tut [gon].

2.3 WebApps

WebApps sind Apps, die mit Webtechnologien, wie z.B. HTML, CSS und Javascript, entwickelt werden. Sie können demnach von jedem modernen Browser ausgeführt werden und mit gängigen Webdevelopment-Tools (oder auch Texteditoren) erstellt werden. Der wesentliche Unterschied zu normalen Webseiten ist, dass WebApps vom Benutzer installiert werden und auch so implementiert werden können, dass sie offline funktionsfähig bleiben. WebApps sind daher plattformunabhängig und können mit einfachsten Mitteln entwickelt werden. *FirefoxOS* ist die erste Plattform speziell für WebApps. Sie sind allerdings auch die einzige Art von Apps, die *FirefoxOS* hat. Ein wichtiges Ziel ist es daher, mit WebApps das *Look & Feel* nativer Apps zu simulieren. Mit nativen Apps sind dabei plattformspezifische Anwendungen für z.B. iOS gemeint, die speziell für dieses Betriebssystem entwickelt wurden und auch nur auf diesem installiert und ausgeführt werden können. Diese haben bei den bereits etablierten mobilen Betriebssystemen, wie Android oder iOS, im Gegensatz zu WebApps uneingeschränkten Zugriff auf die Hardware, insbesondere auf die Sensorik [SSP+13]. Moderne, sensorlastige Anwendungen, wie z.B. Augmented Reality Apps [GPSR13], sind somit mit Hilfe von WebApps nicht realisierbar. WebApps basieren dagegen auf offenen Web-Standards, sind plattformunabhängig und bieten auch keine nativen FirefoxOS-spezifischen Widgets oder Ähnliches an. Um ein einheitliches *Look & Feel* zu erreichen, empfiehlt Mozilla die Einhaltung eines *FirefoxOS*-Styleguide [des].

2.3.1 Manifests

Grundlegender Bestandteil jeder WebApp ist das Manifest, welches sie zugleich von normalen Webseiten abgrenzt. Das Manifest ist eine JSON-Datei, die Informationen enthält, welche ein Webbrowser (oder hier *FirefoxOS*) braucht, um mit der App zu interagieren. Das App Manifest muss den Namen `manifest.webapp` tragen und muss folgende Felder haben [man]:

`name`	Name der App
`description`	Eine kurze Beschreibung der App
`launch_path`	relativer Pfad zum Start der App (i.d.R. /index.html)
`icons`	Pfade zu den Icons der App (in versch. Größen)
`developer`	Informationen über den Entwickler (Name und URL zur Homepage) *(eigentlich optional aber vom Marketplace verlangt)*

Außerdem existieren noch eine ganze Reihe an optionalen Feldern, wovon hier nur die wichtigsten aufgezählt werden. Eine vollständige Liste ist unter https://developer.mozilla.org/en-US/Apps/Developing/Manifest zu finden.

`activities`	Eine Liste von Web Activities, die diese App unterstützt
`appcache_path`	Pfad zum AppCache manifest
`orientation`	erlaubte Displayausrichtungen (“`portrait`”, “`landscape`” oder beide)
`origin`	nur für *privileged* und *certified* Apps. Gibt die Internet-Herkunft an, da packaged Apps keine besitzen. Ermöglicht den gebraucht von Authentisierungsprotokollen oder XHRs.
`permissions`	Eine Liste von WebAPIs, die die App benutzen darf. Manche davon benötigen *privileged* oder sogar *certified* Status.
`type`	Typ der App (der ihr auch die entsprechenden Rechte gewährt). *web, privileged* oder *certified*
`version`	Version des Manifests

Listing 2.1 zeigt, wie ein solches Manifest aussehen könnte. In diesem Fall enthält es die wichtigsten Felder des Manifests der DBIS App. Neben den obligatorischen ersten fünf werden hier unter anderem die Berechtigungen für den Speicherzugriff und für XMLHttpRequests eingeholt, wofür wiederum der Typ der Anwendung auf `privileged` gesetzt werden muss.

Listing 2.1: App Manifest

```
{
        "name": "DBIS Experience",
        "description": "Information platform of the Institute
                        of Databases...",
        "launch_path": "/index.html",
        "icons": {
                "128": "/img/icon2x.png",
                "64": "/img/icon.png"
        },
        "developer": {
        "name": "Julian Frank",
        "url": "http://your-homepage-here.org"
        },
        "default_locale": "en",
        "orientation": ["portrait"],
        "origin": "app://uni-ulm.de",
        "installs_allowed_from": ["*"],
        "permissions": {
                "storage": {},
                "systemXHR": {}
        },
        "type": "privileged",
        "version": "1.0"
}
```

2.3.2 Packaged vs. Hosted Apps

Es gibt grundsätzlich zwei Arten von WebApps. *Packaged Apps* sind in einer .zip-Datei verpackt, die alle essentiellen App-Bestandteile, wie HTML-/CSS- und JavaScript-Dateien, sowie Bilder und das Manifest enthält. *Hosted Apps* hingegen werden, wie der Name schon sagt, auf einem Server gehostet und von dort aus ausgeführt. In diesem Fall befindet sich nur das Manifest auf dem Gerät. Wird also eine App aus dem Marketplace geladen, erhält man entweder eine .zip-Datei mit der App oder eine URL zu einer Hosted App [pck].

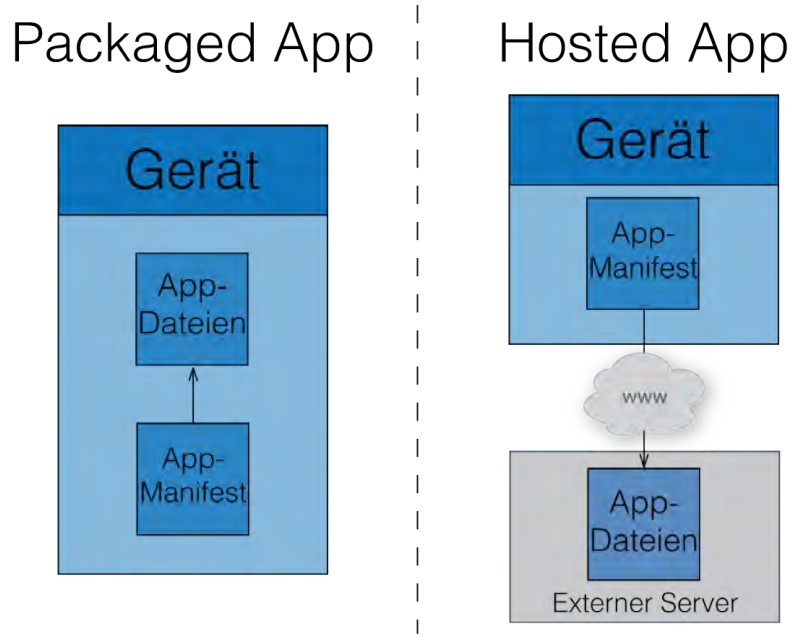

Abbildung 2.5: Packaged vs. Hosted Apps

Da gehostete Apps die wenigsten Berechtigungen haben und als nicht vertrauenswürdig gelten, ist es oft nötig eine Packaged App zu verwenden, bei denen der Typ auf `privileged` gesetzt werden kann. Diese haben dazu den Vorteil, dass sie vom Marketplace überprüft, verifiziert und anschließend digital signiert werden, sodass der Benutzer sich über die Sicherheit, Privatsphäre und Datennutzung der App bewusst ist.
Es ist auch mäglich, eine reguläre App ohne spezielle Berechtigungen als Packaged App veröffentlichen ohne die Rechte über den Typ anzufordern, d.h. das `type`-Feld wird

weggelassen oder auf `web` gesetzt, was der Default-Wert ist. Diese wird dann ebenfalls vom Marketplace signiert, hat allerdings dieselben Berechtigungen wie eine Hosted App. Dies kann nützlich sein, wenn alle Daten zur Nutzung der App sich bereits auf dem Gerät befinden sollen, ohne Inhalte vorher herunterzuladen.

2.3.3 WebAPIs

WebAPIs sind die Schnittstellen zur Geräte-Hardware, wie Batterie-Status oder Virbrationsalarm, sowie zu auf dem Gerät gespeicherte Daten, wie Kontakte, Bilder, etc. Es gibt verschiedenste APIs, die unterschiedlichen Zwecken dienen, wie z.B. die Communication APIs für Bluetooth, SMS, Telefon, WLAN oder mobiles Internet. Außerdem existieren Hardware APIs für Batteriestatus, Geolocation, Vibration, Gyrosensor oder Kamera, Datenmanagement APIs für den Zugriff auf Kontakte oder Bilder auf dem Gerät, sowie für Datenbanksysteme wie, IndexedDB.

Listing 2.2 zeigt den Gebrauch der WebActivities API. Mithilfe von Activities lassen sich Aufgaben an andere Apps delegieren. Hier wird Beispielsweise die Telefon-App aufgerufen, um eine bestimme Nummer anzurufen.

Listing 2.2: WebActivities

```
1 function makeCall() {
2         var call = new MozActivity({
3                 name: "dial",
4                 data: {number: "+497315024131"}
5         });
6 }
```

Manche APIs benötigen bestimmte Berechtigungen damit sie benutzt werden können. Diese müssen einerseits im `permissions`-Feld im Manifest angegeben werden und erfordern andererseits einen bestimmten Typ, der das Vertrauens-Level der App festlegt. Apps vom Typ `web` haben Zugriff auf die wenigsten WebAPIs. Dies sind alle Hosted Apps, sowie Packaged Apps ohne Typ. Apps vom Typ `privileged` haben Zugriff auf einen

größeren Umfang an Funktionen, wie z.B. Zugriff auf Kontakte oder den Geräte-Speicher. Kritische System-Apps haben den Typ `certified` und vollen Zugriff auf alle WebAPIs. Dieser Typ ist allerdings nur für diese speziellen Anwendungen gedacht und nicht für Drittanbieter-Apps.
Die folgende Auflistung zeigt die möglichen Berechtigungen für die drei Typen, wobei `privileged` alle Berechtigungen von `web` und `certified` alle Berechtigungen von `privileged` einschließt. Die speziellen Berechtigungen von `certified` Apps sind implizit und benötigen keine Bestätigung vom Benutzer [per].

- **web:** Alarm, Mikrofon, Audio von Musik, Video, Web und Radio, Benachrichtigungen, Radio, Standort, Speichernutzung (Appcache oder IndexedDB)
- **privileged:** Audio von Wecker, Kalender, E-Mail oder SMS, iFrames, Kontakte, Geräte-Speicher (Musik, Bilder, SD-Karte, Videos), Tastatur, mobile Daten, XHRs (cross-origin), TCP Sockets
- **certified:** "Attention-Screen" (öffnet sich über allen anderen offenen Fenstern), Audio von Telefon, Background-Apps, Kamera, eingebettete Apps, Benutzerinaktivität, Verbindungsstatus, Netzwerk Events und Statistiken, Berechtigungen anderer Apps, Power Management (Bildschirm/Gerät an-/ausschalten, CPU Kontrolle), Geräte-Einstellungen, SMS, Telefon, Uhrzeit einstellen, Mailbox, andere Apps öffnen, WLAN-Management, WAP Push-Nachrichten

2.3.4 Layout & Design

Bei Apps für mobile Geräte spielt das Layout und Design eine besonders große Rolle, da der Bildschirm wesentlich kleiner wie bei einem Laptop oder Desktop-PC ist. Außerdem variieren die Bildschirmauflösungen von Gerät zu Gerät teils stark, worauf sich die App (bzw. der Entwickler) anpassen muss. Neben einem für kleine Bildschirme optimiertem Layout ist deshalb vor allem ein *responsive Design* gefragt, welches sich auf die Bildschirmauflösung einstellt. Dafür haben sich die *CSS Media Queries* bewährt gemacht, um für verschiedene Bildschirmauflösungen verschiedene Styles zu definierten. Ein Beispiel hierfür zeigt Listing 2.3.

Listing 2.3: CSS Media Queries

```
@media only screen and (device-width: 320px) {
        /* Displays, die genau 320 Pixel breit sind  */
}
@media only screen and (min-width: 350px) {
        /* Displays mit einer Mindestbreite von 350 Pixeln */
}
```

Wesentlich einfacher und schneller geht es allerdings mithilfe von UI-Frameworks. Gerade für Smartphone-Apps eignen sich Frameworks, wie *jQuery mobile* [Fou], *bootstrap* [MO] oder *Sencha Touch* [Incb]. Alle passen sich der Displaygröße an.
Twitter's Framework *bootstrap* ist nicht speziell für mobile Anwendungen ausgelegt. Es hat standardmäßig ein 940 Pixel breites Grid-System, kann aber auf jede Displaygröße angepasst werden. Es bietet Templates für Typografie, Formulare, Buttons, Navigationselemente sowie viele weitere Komponenten und ist in HTML, CSS, LESS und Javascript geschrieben.
Sencha Touch dagegen ist speziell für mobile Anwendungen und Webseiten entwickelt. Es ist im Gegensatz zu *bootstrap* eine reine Javascript Bibliothek. *Sencha Touch* zielt vor allem darauf ab, das Look & Feel nativer Apps zu simulieren. Dazu bietet es sämtliche grafischen Elemente, wie sie von Apps für iOS, Android oder Windows Phone bekannt sind. *Sencha Touch* ist kommerziell, aber für open-source Apps gibt es auch eine entsprechende Lizenz kostenlos.
Ein weiteres Beispiel für ein UI-Framework für mobile Anwendungen ist *jQuery mobile*, welches hier etwas ausführlicher beschrieben wird, da es auch für die im Rahmen dieser Arbeit entwickelten App verwendet wurde.

jQuery mobile

jQuery mobile ist ein vom jQuery Project entwickeltes Framework. Es hat den Vorteil, dass es auf jQuery aufbaut und damit für jeden, der sich mit jQuery bereits auskennt, leicht zu erlernen ist. Abgesehen davon, dass wahrscheinlich jeder Web-Entwickler

schon mit jQuery gearbeitet hat, ist es nicht schwierig mit Hilfe von jQuery mobile in kurzer Zeit eine gut aussehende App zu bauen, die außerdem über ein responsive Design verfügt.
Das grundlegende Layout kann vollständig in HTML5 realisiert werden, ohne den Einsatz von JavaScript. Dazu bietet jQuery mobile mehrere Widgets, die über *Data*-Attribute spezifiziert werden. Als primäre Einheit zur Gruppierung logischer Ansichten dienen die *Pages*. Diese organisieren den gesamten Inhalt. Über gewöhnliche Links, mit dem `id`-Attribut der jeweiligen Seite, kann zwischen diesen navigiert (Navigation erfolgt via Ajax) werden. Der Aufbau einer typischen Seite wird in Listing 2.4 gezeigt, wobei header und footer optional sind.

Listing 2.4: Anatomie einer jQuery mobile Page

```
1  <div data-role="page" id="page1">
2          <div data-role="header">
3                  <h1>Titel</h1>
4          </div>
5          <div data-role="content">
6                  <p>Inhalt...</p>
7          </div>
8          <div data-role="footer">
9                  <h4>Fusszeile</h4>
10         </div>
11 </div>
```

Bei einer Seiten-basierten App können so alle Seiten untereinander definiert werden. Beim Start wird nur die erste angezeigt. Über einen Link wie in Listing 2.5 wird z.B. zu obiger Seite navigiert. Außerdem wird dieser Link als Button dargestellt, wie über das *data-role*-Attribut definiert ist.

Listing 2.5: jQuery Navigation

```
1  <a href="#page1" data-role="button">Link</a>
```

Ähnlich den Pages, bietet *jQuery mobile* viele weitere Widgets an, die jeweils über ein *data*-Attribut beschrieben werden. So gibt es z.B. verschiedenste Listenansichten, Formulare, Dialoge und Popups sowie Buttons, Sliders, Toggles und Checkboxen. Diese können alle über ein mächtiges Theming Framework individuell gestaltet werden. Es gibt vier vorgefertigte Themes, die aber alle überschrieben werden können. Am einfachsten geht das mit dem speziell dafür ausgelegten Tool *ThemeRoller*.
Abgesehen von den Widgets bietet *jQuery mobile* einige zusätzliche Events, gerade z.B. für Touch-Eingaben, sowie virtuelle Mouse-Events. Damit können normale MouseEventListener registriert werden, die auch auf Mauseingaben reagieren. Auf einem Touch-Gerät sorgt *jQuery mobile* dafür, dass die Touch-Events an die korrespondierenden MouseEvent-Handler delegiert werden. Es gibt auch Events für die Orientierung (portrait/landscape), sowie load/change Events für die Page-Widgets.

Mit *jQuery mobile* kann daher schnell und ohne große Vorkenntnisse schnell ein Grundgerüst für WebApps entwickelt werden, das sich an verschiedene Displayauflösungen anpasst. Dazu gibt eine übersichtliche Dokumentation und viele Demos auf der Herstellerseite [Fou].

2.3.5 Simulator

Neben den gängigen Webdevelopment-Tools bieten sich bei der Entwicklung von *FirefoxOS* WebApps vor allem der *Firefox* Browser sowie den dafür als Add-On erhältlichen *FirefoxOS* Simulator an. Der Browser bietet viele, gerade für die Entwicklung von *FirefoxOS*-Apps nützliche Entwickler-Tools. Der *Responsive Design Mode* simuliert einerseits verschiedene Displaygrößen und -orientierungen sowie Touch-Events. Konsole, Inspector und Debugger sind eine große Hilfe beim debuggen. Außerdem lasssen sich über die Konsole direkt Javascript-Funktionen aufrufen. Es empfiehlt sich dabei immer den neuesten Nightly-Build mit der aktuellen Gecko-Version zu verwenden. Der Browser eignet sich zum schnellen testen und debuggen, es lassen sich aber nicht alle Funktionen damit testen.
Deshalb ist bei der Entwicklung von *FirefoxOS* Apps der *FirefoxOS Simulator* ein sehr

wichtiges Tool. Er ist eine vollständige, lauffähige Version von *FirefoxOS*. Er ist momentan nur in einer Alpha-Version verfügbar und kann selbstverständlich auch nicht sämtliche Aspekte eines mobilen Geräts mit *FirefoxOS* simulieren. So wird z.B. nur eine voreingestellte Displaygröße angeboten, CPU-Leistung und Speicherplatz eines solchen Gerätes werden ebenfalls nicht berücksichtigt. Dennoch vermittelt der Simulator einen Eindruck vom Look & Feel der App, und es lassen sich einige Funktionen testen, die der Browser nicht beherrscht (z.B. WebActivities).
Des Weiteren wird das App Manifest verifiziert und die entsprechenden Spezifikationen eingehalten. Der Simulator dient außerdem dazu, Apps auf ein Gerät mit *FirefoxOS* zu pushen, um sie dort zu testen. Ist ein Gerät mit *FirefoxOS* an den Computer angeschlossen, so wird dies auf der linken Seite des Simulator-Dashboards (siehe Abbildung 2.6) angezeigt, und neben den Apps wird außer den *Refresh*- und *Connect*-Buttons auch noch ein *Push*-Button angezeigt, welcher die entsprechende Anwendung auf dem Gerät installiert [Pus].

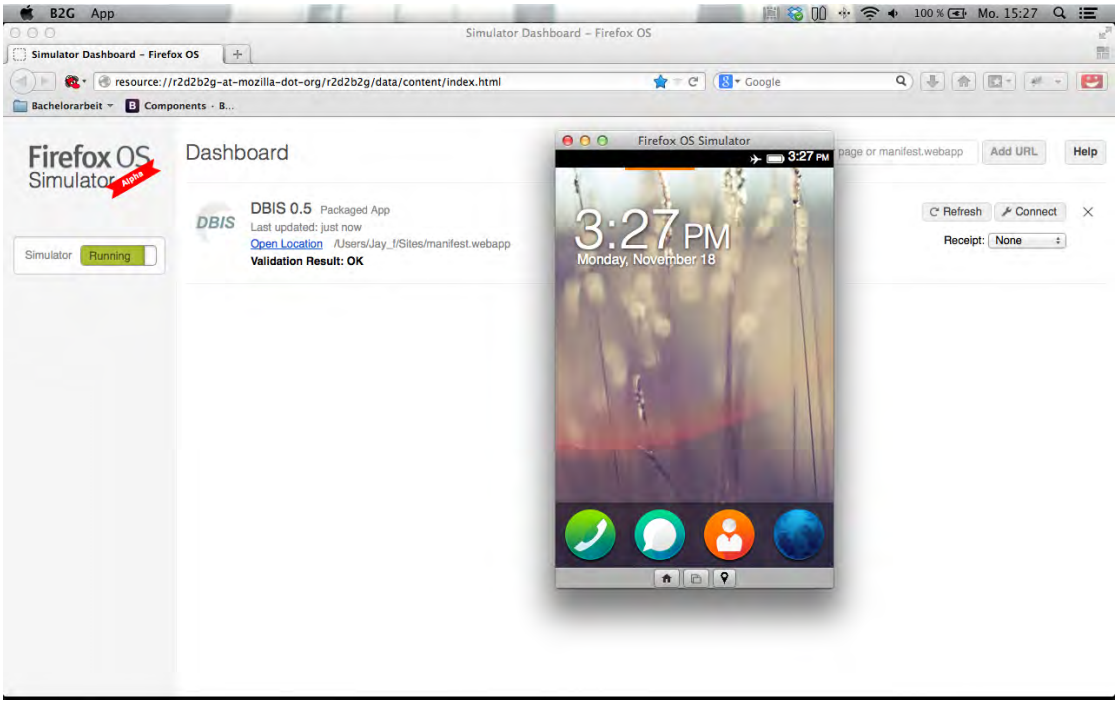

Abbildung 2.6: FirefoxOS Simulator

3

Anforderungsanalyse

Zur Veranschaulichung und zum tieferen Verständnis bestimmter Aspekte von *FirefoxOS*, wurde im Zuge dieser Arbeit die *DBIS uulm* App neu konzipiert und entwickelt. Anhand dieser zeigt sich sowohl der Entwicklungsprozess einer Anwendung für dieses Betriebssystem, als auch die dafür notwendigen Anforderungen und Vorgehensweisen. Im Folgenden werden die funktionalen und nicht-funktionalen Anforderungen an diese App definiert. Dabei gilt es zu beachten, dass die DBIS uulm App eine reine Informationsplattform ist, also keine Benutzereingaben erhält und lediglich Informationen abruft, speichert, aufbereitet und darstellt. Die im Folgenden definierten Anforderungen, insbesondere die Funktionalen, beziehen sich daher hauptsächlich darauf, wie der Benutzer an gesuchte Informationen gelangt und wie diese geladen, gespeichert, aufbereitet und dargestellt werden.

3.1 Funktionale Anforderungen

Die Anwendung lässt sich in acht Kategorien unterteilen. Dies sind die acht Punkte der Liste auf der Startseite. Anhand dieser werden die funktionalen Anforderungen gegliedert und definiert. Diese Kategorien sind:

1. News
2. Events
3. Staff
4. Publications
5. Projects
6. DBIS-Website
7. Contact
8. About

Jede Kategorie zeigt dem Benutzer die entsprechenden Informationen, die ggf. noch detaillierter betrachtet werden können. Im Folgenden werden die funktionalen Anforderungen der Reihe nach entsprechend der zugehörigen Kategorie spezifiziert. Die Nummerierung (FA#1) dient dabei der späteren Referenzierung der einzelnen Anforderungen.

3.1.1 News

FA#1.1 News Anzeigen: Wenn der Benutzer “News” auswählt, wird geprüft, ob die Daten in der Datenbank auf dem aktuellen Stand sind, und die DB ggf. mit Daten aus dem RSS-Feed aktualisiert. Anschließend werden die Einträge aus der DB als Liste angezeigt.

FA#1.2 News Artikel anzeigen: Wählt der Benutzer einen Listeneintrag in “News” aus, so wird geprüft, ob eventuell in dem Artikel enthaltene Bilder schon in der DB sind.

Ist dies nicht der Fall, so werden diese geladen und in der DB gespeichert. Dann werden die Bilder und der Artikel aus der DB geladen und angezeigt.

FA#1.3 Bildergallerie anzeigen: Wählt der Benutzer ein Thumbnail in einem News-Artikel aus, so wird eine Gallerie mit allen Bildern aus dem Artikel erstellt und das gewählte Bild im Vollbild-Modus angezeigt.

FA#1.4 Gallerie Wisch-Gesten: Bei einer Wisch-Geste auf einem Bild in der Gallerie wird das nächste (Wisch-Geste nach links), bzw. voherige (Wisch-Geste nach rechts) Bild angezeigt, falls vorhanden.

3.1.2 Events

FA#2.1 Events anzeigen: Analog zu FA#1.1. Die Einträge werden gruppiert nach Datum angezeigt.

FA#2.2 Event-Artikel anzeigen: Analog zu FA#1.2.

3.1.3 Staff

FA#3.1 Staff anzeigen: Analog zu FA#1.1. Die Einträge werden als Thumbnail-Liste angezeigt.

FA#3.2 Staff Personeninformationen anzeigen: Wählt der Benutzer einen Eintrag aus der Staff-Liste aus, so wird eine neue Seite mit den entsprechend formatierten Daten aus der DB erstellt und angezeigt. Dabei werden das Mitarbeiter-Foto und die Biographie angezeigt, weitere Felder, wie z.B. Kontaktinformationen und Publikationen, werden minimiert und ausklappbar dargestellt.

FA#3.3 Personenkontaktinformationen/-publikationen anzeigen: Wird eines dieser Felder ausgewählt, so werden die entsprechenden Informationen auf der selben Seite eingeblendet (bzw. ausgeklappt).

FA#3.4 Staff E-Mail: Wählt der Benutzer die E-Mail-Adresse unter den Kontaktinformationen aus, so wird die E-Mail App geöffnet und eine neue, leere E-Mail mit der entprechenden E-Mail-Adresse als Absender erstellt.

FA#3.5 Staff anrufen: Wenn der Benutzer unter den Kontaktinformationen die Telefonnummer auswählt, so wird die Telefon App geöffnet und die Nummer gewählt.

3.1.4 Publications

FA#4.1 Publications anzeigen: Wählt der Benutzer “Publications” aus, so wird eine Liste mit Jahreszahlen von 1990 bis heute angezeigt.

FA#4.2 Publications aus Jahr X anzeigen: Wird aus der Jahreszahlenliste ein Eintrag ausgewählt, so wird geprüft, ob die DB auf dem neuesten Stand ist und diese ggf. mit Daten aus dem entsprechenden Dienst aktualisiert und anschließend alle Publications-Einträge aus dem gewählten Jahr angezeigt.

FA#4.3 Publications Einzelansicht: Wenn der Benutzer einen Eintrag aus den Publications auswählt, wird entweder die verlinkte Webseite in einem eingebetteten Browser angezeigt, oder, falls es sich um einen Link handelt, der direkt zu einem PDF-Dokument führt, dieses direkt angezeigt.

3.1.5 Projects

FA#5.1 Projects anzeigen: Analog zu FA#1.1.

FA#5.2 Project Kurzbeschreibung: Wählt der Benutzer einen Eintrag aus der Projects-Liste aus, so wird eine Kurzbeschreibung zu dem Projekt auf derselben Seite ausgeklappt. Diese Kurzbeschreibung enthält außerdem einen Link zu weiteren Informationen ($\rightarrow$ Project Einzelansicht).

FA#5.3 Project Einzelansicht: Wählt der Benutzer den Link zu weiteren Informationen zu einem Projekt aus, so wird überprüft, ob eventuell enthaltene Bilder bereits in der DB und diese ggf. geladen und in der DB gespeichert. Anschließend wird eine neue Seite mit Beschreibung und Kontakt zum Projekt erstellt und angezeigt. Die Bilder (falls vorhanden) werden am Ende als Thumbnails, mit Links zu einer Bilder-Gallerie (siehe FA#1.3 und FA#1.4) dargestellt. Falls der Dienst zusätzliche

Informationen zu z.B. Partner, Sponsoren, Dauer des Projekts, etc. zu Verfügung stellt, werden hierfür entsprechende Absätze im Artikel eingefügt.

FA#5.4 Project Links: Wenn der Benutzer einen Link auf einer Projektseite auswählt, so wird die verknüpfte Webseite in einem eingebetteten Browser angezeigt.

3.1.6 DBIS Website

FA#6 DBIS-Webseite anzeigen: Wählt der Benutzer den Eintrag “DBIS-Website” aus, so wird ein eingebetteter Browser mit Navigationselementen (Vor/Zurück/Aktualisieren) erstellt und die DBIS-Hompage in diesem angezeigt.

3.1.7 Contact

FA#7.1 Kontakt anzeigen: Wählt der Benutzer “Contact” aus, werden die Kontaktinformationen des DBIS, Buttons für Anrufen und SMS schicken, sowie ein Link zu einer Google Maps Ansicht angezeigt.

FA#7.2 DBIS anrufen: Wählt der Benutzer den “Call”-Button auf der Contact-Seite, so wird die Telefon App geöffnet, und die Telefonnummer des DBIS gewählt.

FA#7.3 SMS an DBIS schicken: Wählt der Benutzer den “Send Message”-Button auf der Contact-Seite, so wird die Nachrichten App geöffnet, und eine neue, leere SMS mit der DBIS Nummer als Empfänger erstellt.

FA#7.4 Google Maps Ansicht: Wählt der Benutzer den “Show in Google Maps”-Button auf der Contact-Seite, so wird eine Karte mit dem Google Maps Plugin angezeigt, und auf die Koordinaten der Uni Ulm zentriert und gezoomt.

3.1.8 About

FA#8.1 Institutsinformationen anzeigen: Wenn der Benutzer “About” auswählt, werden Informationen über das DBIS Institut und die App angezeigt. Außerdem wird ein Button “Send Message”, sowie ein Link zum Impressum eingefügt.

FA#8.2 About SMS: Siehe FA#7.3.

FA#8.3 Impressum anzeigen: Wählt der Benutzer den Link “Imprint” auf der About-Seite aus, so wird das Impressum der Uni Ulm auf einer neuen Seite angezeigt.

3.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen spezifizieren die Qualität und Leistung, die von der App erwartet werden, das Verhalten bei Fehlern sowie eine nachvollziehbare Beschreibung der Implementierung der Anforderungen und einfache Wart- und Änderbarkeit.

NFA#1 - Qualität

Die Anwendung sollte *korrekt* sein, d.h. keine Fehlinformationen zeigen (externe Inhalte ausgeschlossen). Alle Funktionen müssen korrekt und erwartungsgemäß ausgeführt werden. Sie sollte *verlässlich* sein und auch bei Nichterreichen externer Dienste funktionieren. Eine große Rolle bei der Qualität spielt auch die *Anwenderfreundlichkeit*. Dazu gehört vor allem *Konsistenz*, bzgl. des Designs, sowie der Sprache und der Navigation. Hier empfehlen sich bereits bekannte Navigations- und Design-Konzepte, wie sie bei anderen mobilen Anwendungen verwendet werden. Da die Anwendung Daten von externen Diensten abruft und dazu entsprechende Netzwerkverbindungen herstellt, muss auch der Aspekt der *Sicherheit* beachtet werden. Das betrifft die Privatsphäre des Benutzers sowie den Schutz vor Missbrauch des Gerätes. Als letztes Qualitätskriterium sollte die App *robust* sein, d.h. auch bei unerwarteten oder falschen Eingaben nicht abstürzen. Da die App keine Eingaben vom Benutzer erhält, betrifft dies vor allem die externen Dienste.

NFA#2 - Leistung

Als wesentliche Leistungsmerkmale gelten *Anwortzeitverhalten*, *Rechenaufwand* und *Datendurchsatz*. Dies bedeutet die Anwendung sollte schnell auf Benutzeranfragen reagieren und nicht einfrieren. Rechenaufwändige Operationen sollten vermieden werden,

gerade auch wegen der verminderten Leistungsfähigkeit von Smartphone-CPUs und deren negativen Einfluss auf die Akkulaufzeit bei hoher Beanspruchung. Der Datendurchsatz ist so gering wie möglich zu halten. Dies betrifft zum Einen den Datenabruf von externen Diensten. Unnötige Datentransfers sind zu vermeiden und und unbedingt nötige Datentransfers so gering wie möglich zu halten. Dies trägt nicht nur zur Leistung der Anwendung bei, denn tarifliche Datenvolumengrenzen des Mobilfunkanbieters sollten nicht unnötig belastet werden. Außerdem variiert die Übertragungsgeschwindigkeit in mobilen Netzen sehr stark, sodass es unter Umständen zu verlängerten Antwortzeiten der Anwendung führt. Dem kann durch Caching entgegengewirkt werden. Des weiteren sind auch anwendungsinterne Datentransfers, wie Datenbankzugriffe ebenfalls zu minimieren.

NFA#3 - Fehlerverhalten

Im Falle eines Fehlers soll die App dem Nutzer entsprechende, verständliche und nützliche Rückmeldungen geben, z.B. durch einen Dialog. Fehlerquellen sind z.B. Fehler beim Datenbankzugriff oder das Fehlschlagen einer Internetverbindung, welche bei einem mobilen Gerät schnell vorkommen kann.

NFA#4 - Dokumentation & Änderungsmanagement

Zu gutem Softwaredesign gehört auch, dass die Anwendung ein entsprechendes Maß an Flexibilität, Erweiterbarkeit und nicht zuletzt Wartbarkeit aufweist. Dies wird durch einen modularisierten Entwurf gewährleistet, wobei die einzelnen Teile voneinander abgegrenzte Aufgaben erfüllen (Kohäsion) und dabei nur so wenig wie möglich voneinander abhängen (Kopplung) und vor allem auch getrennt voneinander bearbeitet werden können.
Zuletzt sollte die Einhaltung der spezifizierten Anforderungen in der Implementierung nachverfolgbar dokumentiert werden.

4

Konzeption & Implementierung

Dieses Kapitel behandelt den Aufbau der Anwendung und deren Bestandteile, deren Kommunikation untereinander und mit externen Quellen. Dazu sollen entsprechende schematische Darstellungen den Aufbau in absteigender Granularität veranschaulichen. Dabei wird sowohl auf die verwendeten externen Dienste und APIs eingegangen, als auch besondere wiederverwendbare Module der Anwendung. Im Mittelpunkt steht der Caching-Mechanismus, der nicht nur den Datendurchsatz minimiert, sondern auch die Verwendung der Datenbank demonstriert. Anschließend wird beschrieben, wie das Gallery Modul eine Foto-Galerie realisiert, wie sie von etablierten mobilen Betriebssystemen bekannt ist. Die folgende Abbildung zeigt den groben Aufbau der Anwendung sowie deren Anbindung an externe Quellen und Dienste. Die Anwendung besteht teils aus dynamischen und teils aus statischen Inhalten. Die dynamischen Inhalte beziehen ihre Daten von externen Diensten, auf die im Folgenden näher eingegangen wird.

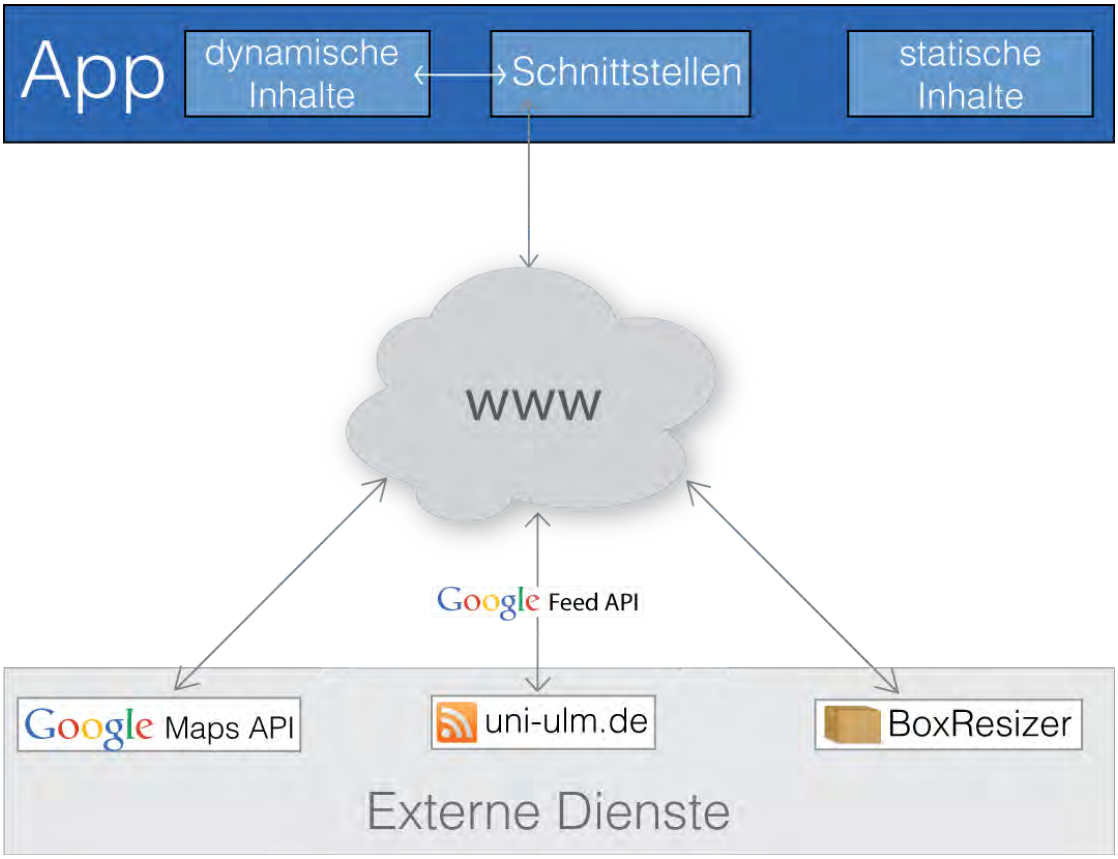

Abbildung 4.1: Datenaustausch Szenario

4.1 Externe Dienste

Abbildung 4.1 zeigt alle externen Quellen und Dienste, mit denen die App kommuniziert. Die eigentlichen Daten, welche die App aufbereitet und darstellt, kommen von entsprechenden Diensten der Uni Ulm. Diese RSS Feeds werden mithilfe der Google Feed API geparst, welche die einzelnen Items eines Feeds als Array zurückliefert. Über die Google Maps API werden die Karteninformationen für die Contact-Seite bezogen. BoxResizer ist ein HTTP-basierter Service, der Bilder über einen Proxy-Server lädt und skaliert zurückliefert. Im Folgenden sollen diese Komponenten genauer beschrieben werden.

4.1.1 RSS Feeds

Die eigentlichen Informationen, welche die App darstellt, werden von den Services der Uni Ulm bereitgestellt. Leider standen zum Zeitpunkt der Erstellung der Arbeit nur die RSS Feeds für News und Events zur Verfügung. Die anderen Services für Projects, Publications und Staff waren zu diesem Zeitpunkt unter Bearbeitung oder aus sonstigen Gründen nicht erreichbar. So stellen momentan leider nur die ersten zwei Kategorien der App dynamische Daten dar.
Die Feeds haben einen typischen Aufbau und enthalten jeweils immer die 10 aktuellsten Einträge. Diese `items` bestehen aus Titel, Link zum Artikel, Beschreibung, Inhalt (HTML Code mit Bildern und Links) sowie dem Veröffentlichungsdatum.

Listing 4.1: Feed Aufbau

```
<item>
        <title>Titel</title>
        <link>https://www.uni-ulm.de/...</link>
        <description>Beschreibung...</description>
        <content:encoded>
                <![CDATA[...Inhalt mit HTML-Tags...]]>
        </content:encoded>
        <pubDate>Mon, 23 Sep 2013 08:30:00 +0200</pubDate>
</item>
```

Die Feeds werden mit Hilfe der Google Feed API [Inca] geparst. Diese liefert ein Array von Item-Elementen zurück, über welche auf die einzelnen Attribute zugegriffen werden kann.

4.1.2 BoxResizer

BoxResizer [MDP] ist ein HTTP-basierter Service, der Bilder durch einen Proxy-Server skaliert. Damit können Bilder auch gespiegelt, gedreht, in ein anderes Format konvertiert oder die Qualität geändert werden. Für die App ist allerdings nur die Skalierfunktion

von Interesse, da zum einen Bandbreite gespart werden soll und zum anderen keine hochauflösenden Bilder für Smartphone-Displays benötigt werden. Um die Bilder skaliert abzurufen, wird das src-Attribute im Image-Element ersetzt.

Listing 4.2: BoxResizer

```
1 var bildURL = "http://www.uni-ulm.de/beispiel.jpg";
2 //BoxResizer Proxy
3 var boxresizer = "http://proxy.boxresizer.com/convert?"
4 //Bild auf 50x50 skalieren
5 var optionen = "resize=50x50&source="+bildURL;
6 var bild = document.getElementByID('bild');
7 //src Attribut ersetzen
8 bild.src = boxresizer+optionen;
```

4.1.3 Google Maps

Für die Kartenansicht auf der Contact-Seite wird das jquery.ui.map Plugin eingebunden [map] (siehe Abbildung 4.7). Dieses nutzt jQuery Events und Methoden, um mit der Google Maps API zu interagieren und ist speziell für den Einsatz mit jQuery mobile optimiert. Um damit eine Kartenansicht darzustellen, werden nur wenige Zeilen Code benötigt. Das Ganze wird an das *pageshow*- bzw. *pageinit*-Event der entsprechenden Seite gebunden. Ein div-Element dient als Platzhalter für die Karte, auf diesem wird die *gmap*()-Funktion gerufen, die eine neue Karte erstellt. Die *gmap*()-Funktion ist ein Konstruktor, dem verschiedene Optionen übergeben werden können.
Listing 4.3 zeigt, wie die Karte aktualisiert wird, sobald die Seite angezeigt wird. Dies ist nötig, da die Seitenübergänge mithilfe von Ajax erfolgen und die Karte deswegen i.d.R. falsch gerendert wird. Bereits wenn die Karten-Seite initialisiert wird, also bei Anwendungsstart, wird auf die Koordinaten der Uni Ulm zentriert und entsprechend herangezoomt. Zu den grundlegenden Funktionen gehört noch das Setzen und Beschriften von Markern, wovon hier aber kein Gebrauch gemacht wird.

Listing 4.3: jQuery Map-Plugin

```
/*Neue Karte auf der 'map'-Seite erstellen,
nachdem diese angezeigt wird*/
$('#map').on("pageshow", function() {
        $('#map_canvas').gmap('refresh');
});
//Karte auf Uni-Ulm zentrieren und zoomen
$('#map').on("pageinit", function() {
        $('#map_canvas').gmap({
                'center': '48.422935, 9.95327',
                'zoom': 14
        });
});
```

4.2 App-Module

Abbildung 4.2 zeigt einen detaillierteren Aufbau der Anwendung und macht deutlich, dass nur die entsprechenden Module mit den externen Quellen kommunizieren und nicht die UI-Klassen.
Die UI-Klassen, welche dynamisch erzeugte Daten darstellen, holen sich diese von einem RSS Feed über das Feed-Modul. Dieses liefert die einzelnen Feed-Items in einem Array zurück. Diese Items werden in der Datenbank gespeichert, auf welche über das DB-Modul zugegriffen wird. Sämtliche DB-Anfragen werden ausschließlich in diesem Modul gestellt. Anschließend müssen die in den Feeds verlinkten Bilder geladen werden. Dies geschieht ebenfalls vom DB-Modul aus, da dieses die Bilder direkt in die DB speichert. Bilder die größer sind als die Auflösung des Displays werden vorher mithilfe des BoxResizer-Services skaliert und dann geladen. Dabei wird ausgenutzt, dass die Größe des Bilds ist immer im img-Element angegeben ist und das Bild daher nicht vorher geladen werden muss.

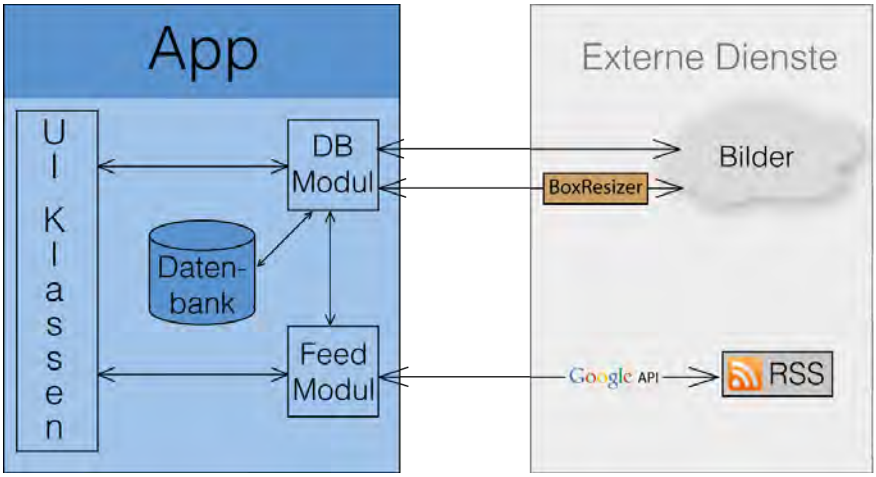

Abbildung 4.2: App-Module

4.2.1 Feed Modul

Das Modul feed.js vereinfacht die Verwendung der Google Feed API. Es ist nicht unbedingt notwendig, aber nützlich, da es nur die für die App relevanten Funktionen der Google Feed API zur Verfügung stellt, und eventuelle Exceptions abfängt, falls der Service nicht erreichbar ist. Das Modul bietet eine einzige Funktion nach außen hin an:

Listing 4.4: Funktion um Feeds zu Laden

```
1 function loadFeed (url, entries, callback) {
2         //Hole Feed und rufe callback mit dem Ergebnis-Array
3 }
```

Dabei ist `url` die URL des Feeds und `entries` die Anzahl der Einträge, die geladen werden soll. Dies ist dann nützlich, wenn nur der erste Eintrag geladen werden soll, um zu überprüfen, ob der gespeicherte Feed noch aktuell ist und somit Bandbreite zu sparen. Anschließend wird die übergebene Callback-Funktion gerufen, welcher ein Array mit den Einträgen übergeben wird. Konnte keine Verbindung zum Feed hergestellt werden, wird eine Fehlermeldung angezeigt.

4.2.2 DB Module

Wie die Überschrift schon andeutet, gibt es mehrere Module, die mit einer Datenbank interagieren, genauer gesagt zwei. Zum einen *feedDB*, welches Feed-Items speichert, und zum anderen *imgDB*, welches Bilder speichert. Beide benutzen indexedDB, eine client-seitige NoSQL-Datenbank, welche dazu dient, Daten im Browser zu speichern oder in diesem Fall auf dem Gerät bzw. in der App. IndexedDB-Zugriffe sind asynchron, d.h. der laufende Thread wird nicht geblockt und es werden nach Ende eines Zugriffs oder einer Transaktion entsprechende Events gefeuert. IndexedDB speichert ausschließlich Key-Value Paare. Der Wert kann dabei ein beliebiges, komplexes Objekt sein, allerdings kann eine einzige Datenbank nur eine Art von Objekten speichern. Der Schlüssel ist ein identifizierender Wert, mit dem das Objekt wieder gefunden werden kann. Dies kann ein automatisch inkrementierter Zähler sein oder ein einzigartiges Attribut des zu speichernden Objekts. Da dies vorher festgelegt werden muss, und eine indexedDB-Datenbank nur eine Art von Objekten speichern kann, werden für Feeds und Bilder getrennte Datenbanken und daher auch zwei verschiedene Module benötigt, die mit diesen interagieren.
So speichert feedDB das gesamte Item-Objekt, mit dem publishedDate, also der Veröffentlichungszeit des Artikels, als Schlüssel. Da dieser kein Objekt sein darf, sondern ein einfacher Wert sein muss, wird dieses Datum mit Hilfe der `getTime()`-Funktion in Millisekunden seit dem 1. Januar 1970 um Mitternacht umgerechnet. Bei den Bildern dient die URL des Bildes als Schlüssel. Das Bild selbst wird als BLOB gespeichert.
Die beiden Module sind im Prinzip analog zueinander aufgebaut. Sie bieten diesselben Funktionen nach außen hin an:

- **open(dbName)** Öffnet eine neue DB und einen dazugeörigen ObjectStore
- **save(URL)** Speichert ein Objekt in die DB
- **get(key)** Holt ein Objekt mit dem gegebenen Schlüssel aus der DB
- **clear()** Entfernt alle Objekte aus der DB

In dieser vereinfachten Übersicht ist nicht dargestellt, dass sämtlichen Funktionen zwei weitere Parameter übergeben werden. Zum einen ein Name für einen objectStore. Eine

indexedDB kann verschiedene objectStores enthalten, allerdings nicht mit verschiednen Objekten, da alle objectStores wiederum zum selben DB-Objekt gehören. ObjectStore-Objekte speichern die eigentlichen Daten. Zum anderen wird allen Funktionen auch noch eine callback-Funktion übergeben. Da sämtliche DB-Anfragen asynchron sind, kann mit dem entsprechenden Ergebnis erst nach erfolgreichem Zugriff weitergearbeitet werden. Die callback-Funktion wird daher in einem Event-Handler für `onsuccess` bzw. `oncomplete` gerufen. Das feedDB-Modul bietet außerdem noch eine weitere, die `getAll()`-Funktion an. Mit dieser können alle Feed-Einträge auf einmal aus der DB geholt werden, was meistens der Fall ist. Die get()-Funktion dient daher nur zum Überprüfen, ob ein entsprechender Eintrag (insbesondere der aktuellste des Feeds) schon in der DB ist.

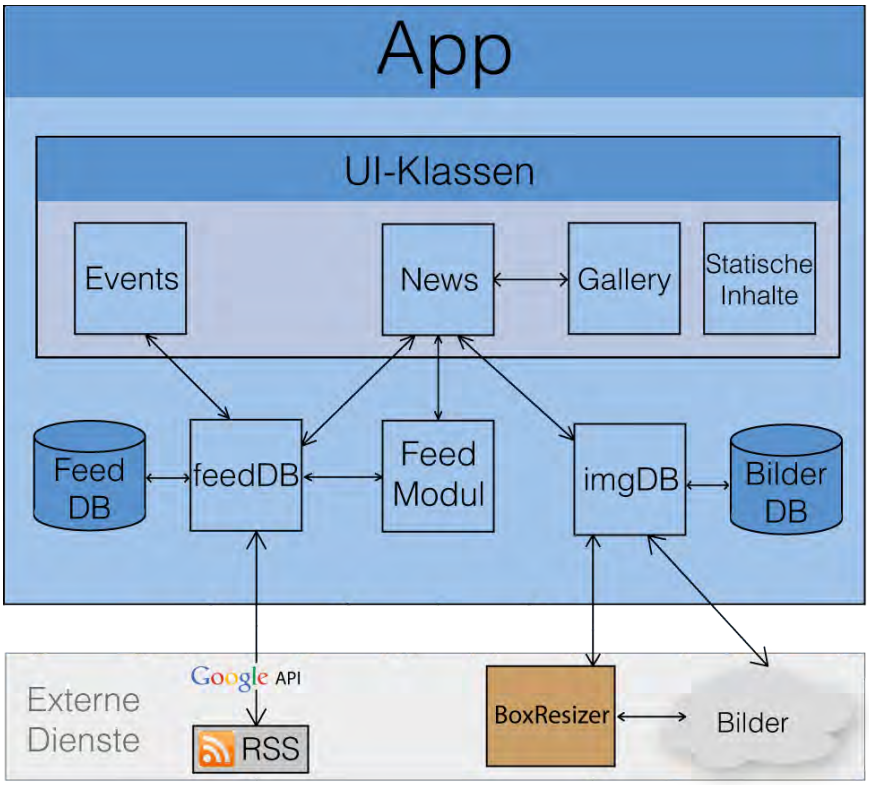

Abbildung 4.3: Detailaufbau

Abbildung 4.3 zeigt die Gesamtarchitektur der Anwendung. Insbesondere ist die Aufteilung des DB-Moduls in die beiden Module für Feeds und Bilder, sowie den jeweils zugehörigen Datenbanken zu beachten. Außerdem ist zu erkennen, dass das Feed-Modul nicht direkt, sonderen ebenfalls über das feedDB-Modul die Feeds abruft. Des Weiteren sind die bis jetzt noch nicht behandelten statischen Inhalte (siehe Kapitel 4.6), sowie das Gallery-Modul (siehe Kapitel 4.4) dargestellt.

4.3 Caching-Mechanismus

Zur Minimierung des Datenverkehrs und vor allem auch zur Offline-Nutzung der App, werden dynamisch geladene Daten in der Datenbank gespeichert. Die Datenbank soll dabei möglichst immer auf dem aktuellen Stand bleiben, wobei Daten aus dem Internet nur wenn unbedingt nötig geladen werden sollten. Zum Zeitpunkt der Erstellung dieser Arbeit waren nur die RSS-Feeds für News und Events verfügbar, andere Services und Dienste für Publications, Projects und Staff konnten nicht eingebunden und diese Inhalte sind daher nur statisch oder nicht vorhanden. Außerdem ist der News-Feed der Einzige, der Bilder enthält.

Der Caching-Mechanismus soll am Beispiel der News-Seite (siehe Abbildung 4.4) beschrieben werden, da diese auch Bilder enthält und somit alle Bereiche abdeckt. Bei der Seiteninitialisierung werden die beiden Datenbanken für Feed und Bilder geöffnet. Anschließend wird nur das erste und neueste Element aus dem Feed geladen und das auch nur, wenn eine Internetverbindung besteht. Ist dies nicht der Fall, wird direkt der Feed aus der DB angezeigt. Dieses erste Element wird nun mit Hilfe seines Veröffentlichunsdatums in der DB gesucht. Wird erfolgreich ein Ergebnis zurückgeliefert, so ist die Datenbank auf dem aktuellen Stand, d.h. alle Feed-Items können aus der Datenbank geladen werden. Wird kein Element in der DB mit dem entsprechenden Schlüssel gefunden, so muss die DB aktualisiert werden. Dazu wird diese zuerst mit `clear()` geleert und dann der gesamte Feed geladen und gespeichert. Dies ist deshalb notwendig, da inzwischen mehr als nur ein neuer Artikel im Feed sein kann und außerdem der Feed immer nur die 10 aktuellsten Einträge liefert und die DB daher auch nicht mehr

speichern sollte. Außerdem spart es Speicherplatz. Der Feed wird außerdem in einem Array zwischengespeichert, so dass während des Lebenszyklus der App dieser nicht noch einmal aus der DB geladen werden muss, um z.B. Detailinformationen zu einem Artikel anzuzeigen.

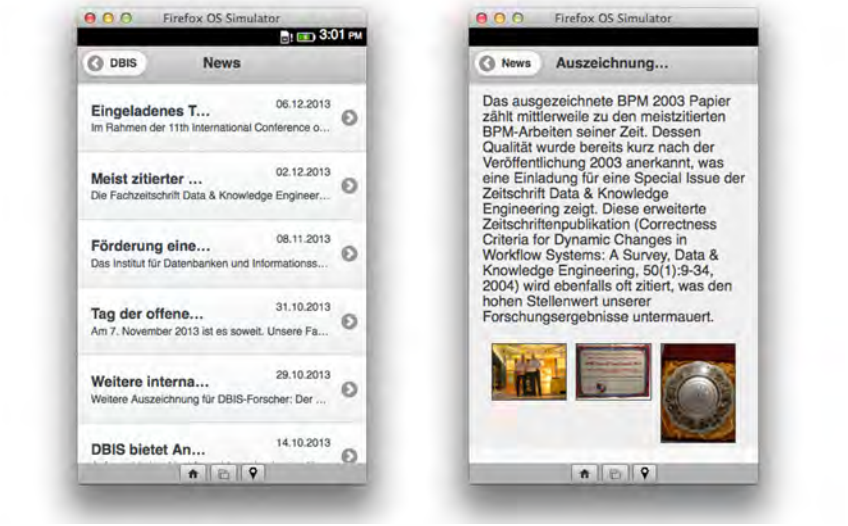

Abbildung 4.4: Die Feed-Items in der Liste und ein Artikel, mit Thumbnails

Bei den Bildern ist es etwas komplizierter. Der Inhalt eines Feed-Items besteht aus HTML-Code, in dem die Bilder in img-Elementen eingebettet sind. Wird dieser Inhalt nun in ein jQuery-Objekt gespeichert, so wird der HTML-Code bereits geparst und GET-Requests zu den Bildern gesendet, auch wenn noch nichts davon zu sehen ist, und der Inhalt noch nicht in das DOM eingefügt wurde. Dieser Inhalt muss zunächst als String behandelt geparst werden, und die src-Attribute der img-Elemente, welche die Bilder-URLs enthalten, entfernt bzw. durch einen leeren String ersetzt werden. Damit die URL nicht verloren geht und weiterhin in Verbindung mit dem entsprechenden img-Element steht, wird diese als id-Attribut im img-Element gesetzt. Dies dient außerdem dazu, das entsprechende img-Element später zu referenzieren.
Der so bearbeitete Inhalt kann nun in das DOM eingefügt werden, ohne dass Bilder geladen werden. Nun wird versucht, die Bilder aus der DB zu laden. Wird das Bild mit dem gesuchten Schlüssel in der DB gefunden, so wird eine objectURL, also eine

Adresse zu dem eigentlichen Bild-Objekt, welches als BLOB in der DB vorliegt, erstellt und diese objectURL als src-Attribut eingesetzt. Wird das Bild nicht in der DB gefunden, so wird zunächst die Größe des Bildes mit der des Displays verglichen (width- und height-Attribute sind im img-Element angegeben) und das Bild dann entweder direkt von den Uni-Servern oder skaliert über den BoxResizer-Service geladen und gespeichert. Dies geschieht mithilfe von Cross-Origin-XMLHttpRequests (XHR), deren Einsatz vorher im Manifest erlaubt werden muss. XHRs sind wegen der Same-Origin-Policy eigentlich nicht möglich, da eine Packaged App aber an sich keine Origin hat, kann diese im Manifest gesetzt werden, was in diesem Fall *app://uni-ulm.de* ist (Packaged Apps nutzen ein spezielles app-Protokoll). Dazu ist wiederum ein privilegierter Status nötig wobei dies bei der Zertifizierung im Marketplace kontrolliert wird, sodass es nicht möglich ist sich als jemand anderes auszugeben.

4.4 Gallery Modul

Die Ansicht der in News-Artikeln enthaltenen Bildern wurde für mobile Geräte optimiert. So werden sämtliche in einem Artikel enthaltenen Bilder unter diesem als Thumbnails angezeigt. Wird eines dieser Thumbnails ausgewählt, so wird das Bild in seiner vollen Größe, jedoch maximal so groß wie die Auflösung des Displays, angezeigt. Genauer gesagt, öffnet sich eine Galerie, in der durch alle Bilder des Artikels “gewischt” werden kann. Dies wird alles von *gallery.js* erledigt. Dieses Modul bietet nur eine öffentliche Funktion an.

Listing 4.5: Gallery Methode

```
1 function init (images,origin,startIndex) {
2         //neue Seite erstellen, Bilder einsetzen, ...
3 }
```

`images` ist dabei ein Array mit den IDs der anzuzeigenden Bilder welche gleichzeitig der Schlüssel zu den entsprechenden BLOBs in der DB sind. `origin` ist die ID der vorangegangenen Seite von welcher der Titel übernommen wird und `startIndex` das initial anzuzeigende Bild im Array, d.h. das Thumbnail, das der Benutzer ausgewählt hat).
Alle Bilder werden horizontal nebeneinander in einem div-Element eingefügt, welches dann mittels CSS-Transitions verschoben wird. Diese wiederum werden von Swipe-Events ausgelöst. Dabei werden nicht die von jQuery mobile bereitgestellten Swipe-Events verwendet. Diese sind zu eingeschränkt, da ein Swipe über min. 30px gehen muss um ausgelöst zu werden. Gleichzeitig sollen die Bilder sich bereits mit der Touch-Bewegung mit verschieben und dann mit Vollendung des Swipe endgültig zum nächsten Bild übergehen. Ist man beim letzten bzw. ersten Bild angekommen und versucht weiter- bzw. zurückzuwischen, so bewegt sich das Bild zwar mit dem Finger mit, springt danach aber wieder zurück, wie man es von der nativen Gallery von anderen mobilen OS kennt. Gallery.js implementiert also eine Bildergallerie, wie sie evtl. schon von z.B. iOS gewohnt ist, mit Hilfe von HTML5 Touch-Events (touchstart, touchmove, touchend) und CSS-Transitions.

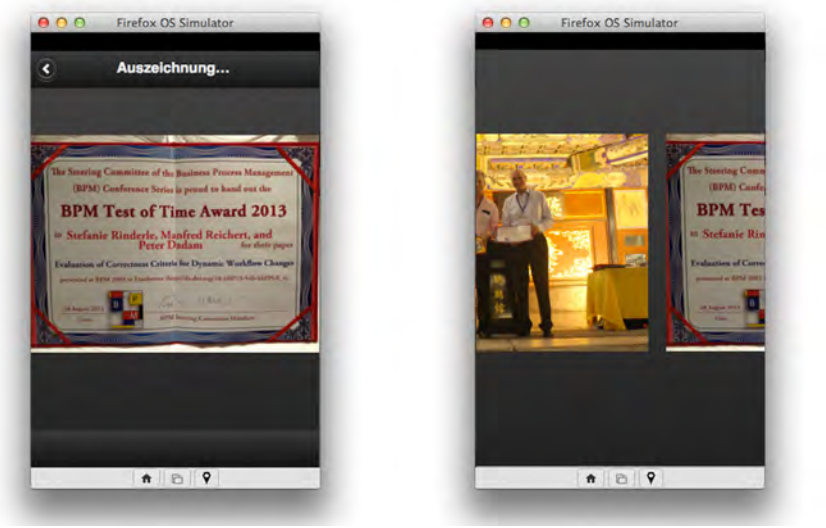

Abbildung 4.5: Galerie-Ansicht vor bzw. nach und während einer Wisch-Geste

4.5 iFrames

Bei der Einbindung der DBIS-Webseite (oder generell externen Seiten), wird eine große Schwäche von *FirefoxOS* sichtbar. Das Einbinden an sich geschieht ganz unkompliziert mit Hilfe eines iFrame. Allerdings zeigt dieser die Seite gezoomt an, es ist also nur eine Ausschnitt sichtbar. Da dies die Regel bei iFrames ist, ist das keine große Überraschung, aber selbst der native Browser von *FirefoxOS* zeigt dieselbe Seite genauso an, während die mitgelieferten Browser von z.B. Android oder iOS die Seite dem Display anpassen, so dass alles sichtbar ist.
Ein möglicher Grund dafür ist, dass *FirefoxOS* an sich quasi ein Browser ist und alle Apps in iFrames laufen. Der Browser ist also ein iFrame, der innerhalb des System-iFrames läuft. Im Browser-iFrame wird wiederum eine Webseite in wieder einem iFrame angezeigt. Deshalb überrascht es nicht, dass ein iFrame innerhalb der App, die wieder in einem iFrame ist, die Webseite genauso anzeigt, wie der iFrame im Browser. Ein Problem, bei dem *FirefoxOS* weit hinterher hinkt, denn so ziemlich jeder mobile Browser kann heutzutage Webseiten entsprechend skaliert darstellen [Cam].

4.6 Statische Inhalte

Die meisten Inhalte der Anwendung sind statisch. Das liegt vor allem daran, dass bereits erwähnte Dienste und Services zur Zeit der Erstellung dieser Arbeit nicht verfügbar waren. Andere Teile der Anwendung sind wiederum an sich statisch (z.B. das Impressum). Deshalb sollen im Folgenden die bis jetzt nicht behandelten Kategorien der App erläutert werden. Dies betrifft alle bis auf News und Events, die ihre Inhalte von RSS Feeds beziehen. Für die Kategorien, die ebenfalls ihre Daten dynamisch beziehen würden, werden Lösungsvorschläge gegeben. Diese beziehen sich hauptsächlich auf das Layout, da deren Realisierung immer von den zur Verfügung gestellten Daten der Dienste abhängig ist.

4.6.1 Staff

Hier werden alle Mitarbeiter des Instituts aufgeführt, zusammen mit ihren Kontaktdaten, Sprechzeiten und evtl. dem Lebenslauf und ihre Publikationen. Da der hierfür zuständige Service zum Zeitpunkt der Implementierung nicht verfügbar wahr, wurde die aktuelle Liste der Mitarbeiter (inklusive Bilder) statisch programmiert.
Sämtliche Daten und Bilder stammen von *http://www.uni-ulm.de/in/iui-dbis/team.html* und unterliegen dem Copyright der Uni Ulm. Die Links dieser Liste führen dann auf die entsprechende Mitarbeiter der Uni Ulm. Der erste Eintrag in der Liste (Prof. Dr. Dadam, siehe Abbildung 4.6) zeigt, wie die Daten vom Dienst gelayoutet werden könnten.
Insbesondere werden die Kontaktinformationen in ein ausklappbares Element eingefügt und die Publikationen, nach Jahr sortiert, in einer Liste aus Links bereitgestellt.
Ein eventueller Lebenslauf (hier nicht vorhanden) wird als formatierter Text dazwischen platziert. Für weitere optionale Kategorien, wie Forschung, Interessen oder Projekte, würde man am Ende der Seite zusätzliche Sektionen einfügen. Dies setzt aber eine Strukturierte Form der zugrundeliegenden Daten voraus. Die zusätzlichen Sektionen könnten wie die Kontaktdaten ausklappbar sein, sodass man beim Aufruf einer Personenseite auf einen Blick sieht, welche Informationen vorliegen um dann die gewünschte Sektion auszuklappen.

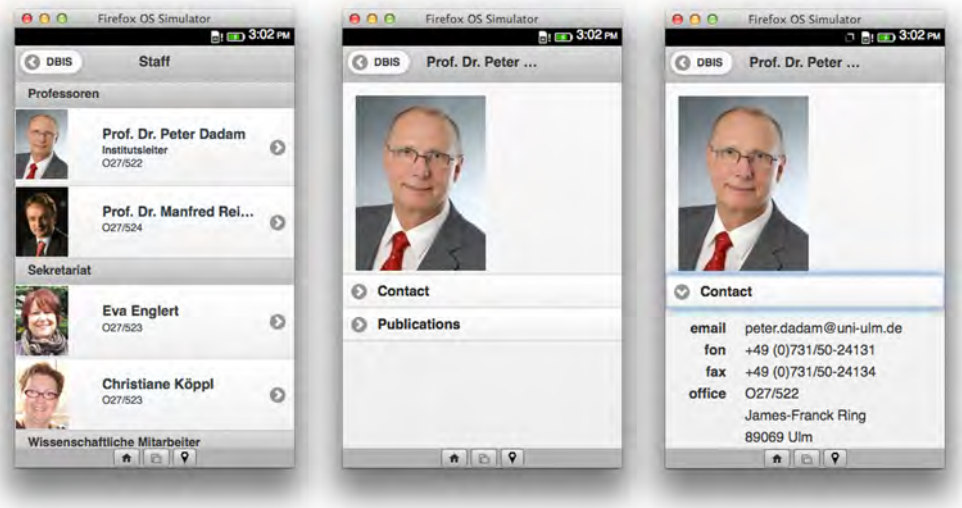

Abbildung 4.6: Staff Liste und Einzelansicht

4.6.2 Publications & Projects

Die Veröffentlichungen des Instituts werden ebenfalls in einer formatierten Liste dargestellt. Durch Auswählen eines Listeneintrags wird eine neue Seite mit einer Kurzzusammenfassung angezeigt, die einen Link zur eigentlichen Webseite mit Detailinformationen enthält. Die Projects-Seite wurde nicht implementiert, diese würde höchstwahrscheinlich analog zu Publications ausfallen, je nachdem, wie der jeweilige Dienst, der die Daten bereitstellt, aussieht.

4.6.3 Contact

Die Kontakt-Seite, mit der Adresse des DBIS sowie der Telefon- und Faxnummer, ist an sich statisch und enthält drei Buttons. “Call” und “Send Message” delegieren die Aufgabe mittels WebActivities an die Telefon- bzw. Nachrichten-App, wobei die jeweilige Nummer mit übergeben wird. “Show in Google Maps” öffnet eine bereits beschriebene Kartenansicht mithilfe des jquery.ui.map-Plugins.

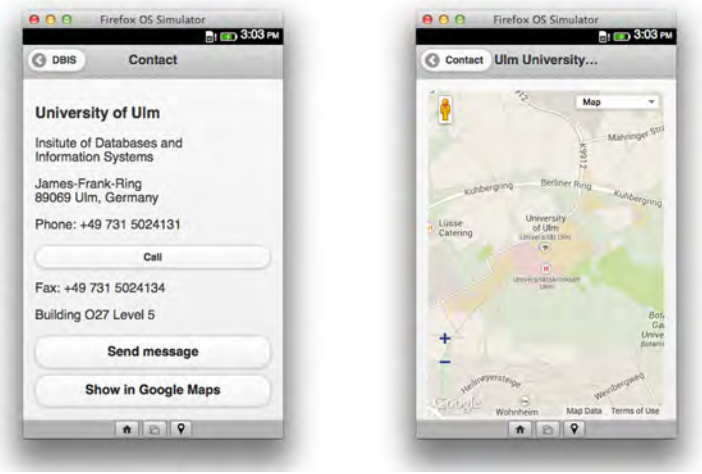

Abbildung 4.7: Contact Seite und Google Maps Kartenansicht

4.6.4 About

Diese ebenfalls statische Seite enthält neben einem weiteren “Send Message” Button (siehe Contact) nur noch einen Link auf das Impressum. Dieses ist ebenfalls statisch in der App programmiert und wird nicht dynamisch von der Uni-Ulm-Webseite geladen.

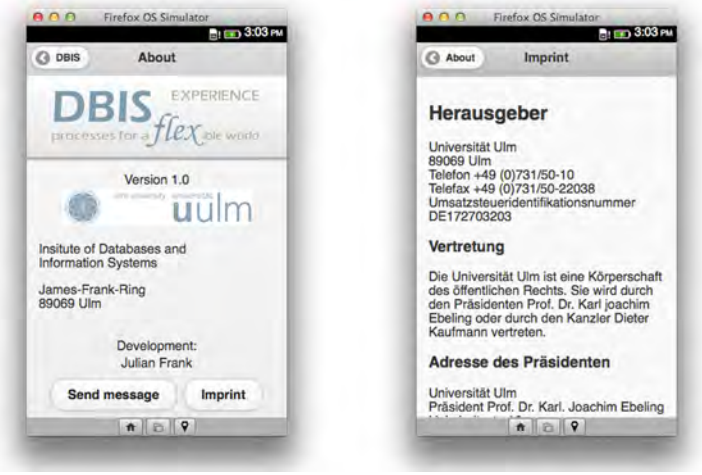

Abbildung 4.8: About Seite und Impressum

5

Anforderungsabgleich

In diesem Kapitel werden die in der Anforderungsanalyse gestellten Anforderungen an das System überprüft. Dabei wird nicht nur festgestellt, ob bestimmte Anforderungen erfüllt sind oder nicht, sondern auch, warum manche nicht erfüllt werden konnten, wie sie hätten erfüllt werden können und was an erbrachten Lösungen noch verbesserungswürdig wäre. Für nicht erfüllte Anforderungen werden Lösungskonzepte aufgezeigt.
Der Anforderungsabgleich ist analog zur Anforderungsanalyse in funktionale und nichtfunktionale Anforderungen gegliedert. Die Anforderungen werden mit ihrer in Kapitel 3 angegebenen Nummerierung referenziert.

5.1 Funktionale Anforderungen

Da es viele analoge funktionale Anforderungen gibt, werden diese zusammengefasst abgeglichen. Dabei wird nur kurz auf die verwendeten Lösungstechniken eingegangen, um größere Überschneidungen mit dem vorangegangenen Kapitel zu vermeiden.

5.1.1 Navigation

Dies betrifft implizite Vorraussetzungen der Anforderungen FA#x.1, also das Anzeigen der verschiedenen Seiten und die Navigation zwischen diesen, wobei diese Anforderungen allein damit noch nicht erfüllt sind. Dies wird im Großen und Ganzen von jQuery mobile erledigt. Alle Seiten sind in Container gegliedert, mit einem data-attribute “page”. Das Navigationskonzept von jQuery mobile lädt diese in das DOM und zeigt die erste davon an. So sind in der *index.html* bereits alle Hauptseiten der Anwendung untereinender definiert, als erstes jedoch die Liste mit den Unterkategorien der App, so dass diese als Startseite angezeigt wird. Weitere Seiten werden per Ajax nachgeladen und in das DOM eingefügt. Dies alles erledigt allerdings das Framework. Es wurden nur für die entsprechende Verlinkung gesorgt und die Übergange zwischen den Seiten (Transitions) modifiziert.

5.1.2 News

Wird *News* ausgewählt, wird überprüft, ob die Datenbank auf dem aktuellsten Stand ist (Vergleich mit 1. Element des Feeds), bei Bedarf aktualisiert (siehe Kapitel 4.3 Caching-Mechanismus) und anschließend eine Liste mit den Feed-Einträgen erzeugt, welche nach Datum sortiert ist (neuester Eintrag zuerst). Damit ist FA#1.1 vollständig erfüllt.
Durch Auswahl eines Listeneintrags wird eine neue Seite erzeugt, der Artikel-Text darin eingefügt und eventuell enthaltene Bilder als Thumbnails daruntergesetzt. Diese werden auch, wie im Caching-Mechanismus beschrieben, in der DB gespeichert. Dies erfüllt FA#1.2.

Wird ein Thumbnail ausgewählt, wird eine neue, dynamisch erzeugte Seite gezeigt, in der die Bilder auf dem ganzen Display betrachtet werden können. Sind mehrere Bilder in einem Artikel enthalten, so kann man durch eine Galerie “wischen”, wie beim Gallery-Modul beschrieben. Dies erfüllt FA#1.3 und FA#1.4.

5.1.3 Events

Analog zu *News*, wird die Datenbank auf Aktualität überprüft und aus den Einträgen eine nach Datum sortierte Liste generiert. Zusätzlich werden Events, die am gleichen Tag stattfinden, unter einem Listenabschnitt gruppiert (FA#2.1).
Wählt der Benutzer einen Listeneintrag aus, so wird auf einer neu angelegten Seite der Artikel angezeigt, welcher aus dem zwischengespeicherten Array mit den Event-Artikeln geladen wird. Somit ist auch FA#2.2 erfüllt.

5.1.4 Staff

Wählt der Benutzer *Staff*, so wird eine Liste der Mitarbeiter, gruppiert nach Dozenten, Sekretariat, HiWis usw. gruppiert angezeigt. Allerdings kommen diese Daten und Bilder nicht aus der Datenbank und werden bei eventuellen Personaländerungen auch nicht aktualisiert, da alle Daten hier statisch sind. Der zuständige Service, der diese Daten bereitstellen sollte, war zum Zeitpunkt der Implementierung nicht verfügbar. Demnach funktionierte diese Funktion auch nicht bei der bereits bestehenden iOS App, die denselben Service in Anspruch nimmt und daher konnte diese Ansicht dem Original nicht nachempfunden werden. Diese Ansicht soll aber zeigen, wie man die Mitarbeiter darstellen könnte, z.B. als Liste mit Thumbnails. Anforderung FA#3.1 ist daher nur bedingt erfüllt. Ansonsten würde man die Daten und Bilder abrufen und in der DB speichern, die dann evtl. nur wöchentlich oder noch seltener aktualisiert wird. Da Personaländerungen bei Weitem nicht so häufig sind wie Neuigkeiten aus dem Institut, ist dies ausreichend und spart zudem Bandbreite. Ansonsten könnten die Bilder wieder über das bestehende *imgDB.js* Modul gespeichert und abgerufen werden. Bei den restlichen Daten hängt dies wiederum von dem Uni Ulm Dienst ab und wie dieser die Daten bereitstellt.

Dieser Umstand betrifft natürlich auch die anderen FA#3.x Anforderungen. So konnten FA#3.2 - FA#3.5 nicht erfüllt werden, allerdings zeigt der erste Eintrag in der Liste (Prof. Dr. Peter Dadam) ein Beispiel, wie diese erfüllt werden können (nur eben, dass die Daten wieder statisch sind). So können Kontaktinformationen eingeblendet werden und durch tippen auf die Telefonnummer oder E-Mail-Adresse öffnet sich die Telefon- bzw. E-Mail-App. Ähnlich dazu können die Publikationen ausgeklappt werden und ein Jahr gewählt werden. Dies zeigt, wie FA#3.2 - FA#3.5 hätten erfüllt werden können. Bei den restlichen Mitarbeitern wird allerdings der Einfachheit halber auf deren Uni-Seite verlinkt.

5.1.5 Publications

Da der Service für die Veröffentlichungen ebenfalls zum Zeitpunkt der Erstellung dieser Arbeit nicht funktionierte, ist auch hier nur ein demonstrativer, statischer Lösungsansatz implementiert. Über eine Liste von Jahreszahlen wird zu einer Übersicht der Veröffentlichungen aus dem gewünschten Jahr navigiert. Diese sind wiederum unter Monaten gruppiert. Dies demonstriert wie FA#4.1 und FA#4.2 gelöst werden könnten. Hier wird lediglich gezeigt, wie die Liste aussehen könnte, die Links führen allerdings nicht zu Detailansichten, d.h. FA#4.3 wurde nicht erfüllt, da bei der Publications-Liste auch nur Platzhalter vorhanden sind. Man könnte aber beispielsweise direkt auf die entsprechende Seite verlinken.

5.1.6 Projects

Da für diesen Bereich ebenfalls kein Service zur Verfügung stand, der Daten bereitstellte, konnte dieser Teil ebenfalls nicht erfüllt werden. Es wurde auch kein Platzhalter oder eine Beispielansicht implementiert. FA#5.1 - FA#5.4 sind also nicht erfüllt. Eine denkbare Möglichkeit dies darzustellen wäre z.B. wieder eine Listenansicht mit ausklappbaren Einträgen (siehe die Kontaktinformationen bei Staff) mit einer Kurzbeschreibung des Projekts. Durch einen dort enthaltenen Link würde zu einer neuen Seite mit detaillierteren Informationen führen, je nachdem, welche Daten der Service bereitstellt. Sämtliche

Daten sowie eventuelle Bilder würden natürlich wieder über einen Caching-Mechanismus in der DB gespeichert.

5.1.7 DBIS Webseite

FA#6 ist bedingt erfüllt. Die Webseite wird in einem iFrame angezeigt, allerdings besteht das im vorigen Kapitel unter *iFrames* beschriebene Problem, dass die Webseite gezoomt dargestellt wird und man daher nur einen Ausschnitt sieht, was sehr unübersichtlich ist. Zu diesem Zeitpunkt, existiert für dieses Problem noch keine Lösung.

5.1.8 Contact

Wählt der Benutzer *Contact*, so wird die Adresse, sowie die Telefon- und Faxnummer des DBIS angezeigt. Mit den drei dargestellten Buttons kann angerufen oder eine Nachricht geschickt werden, worauf sich die jeweils passende App öffnet, mit dem entsprechenden Empfänger. FA#7.1 - FA#7.3 sind somit erfüllt. Da Kartenansicht für FA#7.4 wurde mit Hilfe des jquery.map-Plugins gelöst, welches die Google Maps API nutzt. So wird die Karte innerhalb der App angezeigt. Alternativ könnte man auch die native Map App benutzen (Nokias HERE Maps), was aber weniger komfortabel wäre, da zwischen Apps hin- und hergewechselt werden müsste. Zum Anrufen bzw. zum Verschicken von Nachrichten ist das in jeden Fall erforderlich, da nur zertifizierte Apps, wie gerade eben Telefon und Nachrichten, die Berechtigungen dafür besitzen.

5.1.9 About

Dieser Inhalt ist komplett statisch. Der Button verhält sich genauso, wie der auf der Contact-Seite. Der Link zum Impressum ist ebenfalls, wie in der Original iOS-App als Button dargestellt. Das Impressum ist statisch, muss also nicht heruntergeladen werden. Sollte sich hier allerdings etwas ändern, so ließe sich dies nur über ein App-Update korrigieren. Da dies aber unwahrscheinlich ist und im Rahmen dieser Arbeit auch nicht von Bedeutung ist, schien diese Lösung als ausreichend.

5.2 Nicht-funktionale Anforderungen

Während der gesamten Implementierungsphase wurde auf die Einhaltung der nicht-funktionalen Anforderungen geachtet. Der Abgleich mit den definierten Anforderungen ist zum Teil subjektiv, es werden aber weitestgehend konkrete Beispiele und Situationen genannt.

5.2.1 NFA#1 - Qualität

Die Qualitätsanforderungen sind, so weit es geht, erfüllt. Dass heißt, auf Grund der fehlenden Dienste konnten nicht sämtliche Funktionen implementiert werden. Die die umgesetzt wurden, funktionieren allerdings, wie es von ihnen erwarten wird. Durch den Caching-Mechanismus wird gewährleistet, dass die App auch offline oder bei Nichterreichen eines Dienstes funktioniert (bis auf die Kartenansicht). Die App ist konsistent gehalten, was des Design sowie die Sprache angeht. Letztere ist durchgängig Englisch, bis auf die Informationen, die von den Feeds kommen. Das Navigationskonzept entspricht einem bekannten, seitenbasierten Konzept, wie es von vielen anderen mobilen Anwendungen bekannt ist. Bis auf die Bilder-Galerie ist alles der Original-App nachempfunden, in der allerdings gar keine Bilder angezeigt werden. Allerdings ist es ein auf Smartphones gängiges Prinzip, durch Bilder mittels Wisch-Gesten zu blättern.
Die Sicherheit wird allein schon dadurch gewährleistet, dass alle Berechtigungen im Manifest angegeben werden müssen und privilegierte Berechtigungen einen entsprechenden App-Typ voraussetzen, der dann wiederum vom Marketplace vor Veröffentlichung überprüft wird. In diesem Fall sind dies lediglich die XHRs, die aber nur dazu dienen, Bilder von den Uni-Servern abzurufen. Die Anwendung greift nicht auf persönliche Daten zu und erlaubt auch keine Benutzereingaben, was einen Missbrauch des Gerätes auch ausschließt.
Was die Robustheit der Anwendung betrifft, so wird eine gewisse Verlässlichkeit und Vertrauenswürdigkeit der genutzten Dienste vorausgesetzt. Falls die Uni-Server nicht erreichbar sein sollten, so werden die Daten aus der DB angezeigt, dies trifft aber nicht auf das Kartenmaterial der Contact-Seite zu. Eine weitere Stelle für Fehler stellt der

BoxResizer Service dar, der bei Überlastung evtl. lange für eine Antwort braucht oder auch gar nicht verfügbar ist. Der Service läuft auf dem Amazon Elastic Compute Cloud Service (Amazon EC2), der selbst eine Verfügbarkeit von 99,95% angibt. Der BoxResizer Service war in der Testphase zwar stehts verfügbar, jedoch kam es zuweilen zu geringen Verzögerungen gegenüber den direkt von den Uni-Servern geladenen Bildern.

5.2.2 NFA#2 - Leistung

Die Leistung der Anwendung soll anhand der Kriterien Antwortzeitverhalten, Rechenaufwand und Datendurchsatz gemessen werden. Das Antwortzeitverhalten stellt kein großes Problem dar. Der Benutzer kann flüssig von einer Seite zur nächsten wechseln, auch bei Datenbankzugriffen ist kaum eine Verzögerung wahrnehmbar. Der einzig erwähnenswerte Aspekt wäre der BoxResizer Service, bei dem es teilweise zu kurzen Verzögerungen kommen kann. Dies ist aber nicht weiter gravierend, da sich die Bilder, meistens ist es nur ein Bild pro Artikel, am Ende des Artikels befinden, wozu der Benutzer i.d.R. erst ganz nach unten scrollen muss. Sind die Bilder einmal in der Datenbank gespeichert, tritt diese Verzögerung natürlich nicht wieder auf.
Ein großer Rechenaufwand besteht bei einer Anwendung, die lediglich Daten aufbereitet und darstellt nicht. Der einzig rechenintensive Prozess, das Skalieren der Bilder, übernimmt ein externer Prozess, belastet demnach die Smartphone CPU nicht.
Der Datendurchsatz wird so gering wie möglich gehalten. Das betrifft nicht nur den durch den Caching-Mechanismus minimierten Datenaustausch mit externen Quellen, sondern auch die Anzahl der Datenbankzugriffe. So werden die Feeds beispielsweise in einem temporären Array gespeichert, so dass während der Laufzeit der App dieser nur einmal aus der DB geholt werden muss, und nicht jedes mal, wenn ein neuer Artikel angezeigt werden soll. Bei den Bildern ist das allerdings nicht ohne Weiteres möglich. So können die erzeugten objectURLs der Thumbnails nicht für die Galerie wiederverwendet werden, daher müssen hier neue DB-Anfragen gestellt werden.

5.2.3 NFA#3 - Fehlerverhalten

Die einzigen möglichen Fehlerquellen der Anwendung sind die externen Dienste, da die Anwendung keine Benutzeranfragen erhält und der Rest statisch ist. In diesem Fall wird eine Fehlermeldung angezeigt, dass bei Nichterreichen des Feeds, oder der dazu nötigen Google API, womöglich ein Problem mit der Internetverbindung besteht. Dies ist nur dann der Fall, wenn z.B. ein Datennetz verfügbar ist (`navigator.onLine` ist in diesem Fall `true`), der Verbindungsaufbau schlägt aber fehl. Besteht erst gar keine Internetverbindung, so werden die Daten aus der DB geladen, ohne dass dem Benutzer eine Meldung angezeigt wird, was in diesem Fall auch nicht hilfreich wäre.

5.2.4 NFA#4 - Dokumentation & Änderungsmanagement

Wiederverwendbare Teile der Anwendung, wie z.B. die DB-Module oder die Galerie, sind unabhängig vom Rest und können demnach auch getrennt von diesem verändert werden. Ebenso sind die einzelnen Seiten und deren Anwendungslogik, insbesondere News und Events, zwar analog, dennoch vollständig voneinander unabhängig. Dies sorgt zusammen mit dem durchgehend kommentierten Code sowie den Darstellungen und Erläuterungen der vorigen Kapitel für eine einfache Änderbarkeit und nicht zuletzt Erweiterbarkeit. So könnten die fehlenden Dienste anhand der bereits bestehenden Module leicht eingebunden werden.

6

Zusammenfassung & Ausblick

In diesem Kapitel werden als Abschluss dieser Arbeit die behandelten Inhalte noch einmal aufgearbeitet und zusammengefasst. Dabei spielt weniger die Entwicklung der Anwendung eine Rolle, als die während dieser Arbeit gewonnenen Erkenntnisse über das Betriebssystem. Die daraus resultierenden Ergebnisse werden in einem Fazit ausgewertet und anschließend in einem Ausblick mögliche zukünftige Entwicklungen diskutiert.

6.1 Zusammenfassung

Ziel dieser Arbeit war es durch die Konzeption und Realisierung einer Anwendung für *FirefoxOS* das Betriebssystem näher zu beleuchten und um seine Besonderheiten, Stärken und Schwächen aufzuzeigen. Dadurch sollte das noch nicht besonders weit verbreitete System zugänglicher gemacht werden, die Entwicklung und Veröffentlichung

von Apps dargelegt werden und nicht zuletzt die Möglichkeit gegeben werden, sich fundierend auf den dargelegten Fakten eine eigene Meinung über *FirefoxOS* zu bilden.
In Kapitel 2 wurden einige Grundlagen, die zum Verständnis des Betriebssystems sowie zur Entwicklung von Apps notwendig sind, erläutert. Dazu gehören neben der Betriebssystems-Architektur und den Sicherheitsaspekten wichtige Grundbegriffe wie WebApps, Manifests, Packaged bzw. Hosted Apps und WebAPIs. Auch wichtige Hilfsmittel zur App-Entwicklung, wie UI-Frameworks und den *FirefoxOS Simulator* wurden vorgestellt.

In einer Anforderungsanalyse (Kapitel 3) wurden die Anforderungen an die zu entwickelnde Anwendung gestellt. Diese orientieren sich natürlich stark an der bereits für iOS existierenden DBIS-App, es wurden aber neue Aspekte hinzugebracht, gerade um bestimmte Besonderheiten an *FirefoxOS* aufzuzeigen.

In Kapitel 4 wurde die Entwicklung der Anwendung dokumentiert und deren Aufbau erläutert. Dabei ergaben sich wichtige Erkenntnisse über das Betriebssystem.
Da *FirefoxOS*-Apps in den Webstandards HTML, CSS und Javascript geschrieben sind, ist der Quellcode ganz klar nach Struktur (HTML), Styles (CSS) und Anwendungslogik (JS) gegliedert, wobei letztere eindeutig den größten Teil einnimmt. Die Anwendungslogik wiederum ist in Module aufgeteilt, um eine bessere Wartbarkeit und Erweiterbarkeit zu gewährleisten.
Da für Javascript viele externe APIs existieren, macht das die Arbeit damit wesentlich einfacher. Mit Hilfe des UI-Frameworks jQuery mobile wurde das Grundgerüst der Anwendung schnell erstellt. Alternativ dazu gibt es auch noch einige weitere nützliche Frameworks. Ein weiteres nützliches API war die Javascript-Bibliothek von Google, die gerade zum parsen von Feeds nützliche Funktionen liefert, die somit nicht selbst implementiert werden müssen.
Die Speicherung von Daten auf dem Gerät funktioniert über die NoSQL-Datenbank *indexedDB*, welche nach einer kurzen Einarbeitungszeit einfach zu verwenden ist. Damit wurde ein Caching-Mechanismus realisiert, um den Datentransfer zu gering wie möglich zu halten.

Als kleines “Experiment” wurde versucht, eine Bilder-Galerie, bei der man mittels Wisch-Gesten durch die Bilder blättern kann, zu realisieren. Dafür gibt es bei *FirefoxOS* keine vorgefertigten Widgets die das erledigen. Mit Hilfe von HTML5 Touch-Events und CSS3 Transitions wurde eine solche Galerie nachgeahmt.
Ein Problem stellte sich bei der Einbindung von externen Webseiten mittels iFrames heraus, da diese die Seiten nicht wie ein moderner mobiler Browser skalieren und entsprechend zoomen. Selbst der native Browser von *FirefoxOS* kann das nicht.
Ein Großteil der Informationen musste wegen der inaktiver Dienste statisch implementiert werden, wobei hier das Layouting-Potenzial des UI-Frameworks noch einmal demonstriert wurde.

Abschließend wurden in Kapitel 5 die in Kapitel 3 definierten Anforderungen mit den tatsächlich erfüllten verglichen. Aufgrund bereits genannter Gründe konnten leider viele Anforderungen nicht erfüllt werden.

6.2 Fazit

Die während der Entwicklung gewonnen Erkenntnisse zeigen klare Vor- und Nachteile von *FirefoxOS* auf. Jeder Webentwickler kann, nachdem er sich mit dem Konzept der Manifests und WebAPIs auseinandergesetzt hat loslegen und Anwendungen für *FirefoxOS* programmieren. Da sehr viele Entwickler mit Webtechnologien vertraut sind, könnte man eine große Vielzahl von Apps erwarten.
Dennoch wirkt das gesamte Betriebssystem wie eine alte Version von Android oder iOS. *FirefoxOS* wirkt zunächst wie die meisten mobilen Betriebssysteme mit einem oder mehreren Home-Bildschirmen, auf denen die Apps als Icons dargestellt sind. Das Design ist zwar etwas einfacher ausgefallen als bei den erfolgreichen Konkurrenten, ist aber zweckmäßig.

Am Wichtigsten aber ist, das *FirefoxOS* keinerlei Konkurrenz zu den marktführenden Produkten von Google, Apple oder Microsoft ist und auch nicht darauf abzielt. Mozilla’s Betriebssystem ist eine Plattform für das Open Web, die erste Plattform allein für We-

bApps. Es geht vor allem darum frei von proprietären Systemen zu sein. Jeder kann Apps mit freien Webstandards entwickeln und veröffentlichen.
In der Realität sieht es allerdings so aus, dass es zum Zeitpunkt dieser Arbeit auf dem deutschen Markt genau ein Gerät mit *FirefoxOS* gibt. Dieses hat dazu noch eine veraltete Hardware und steht in keinem Vergleich zu modernen Smarphones. Es ist aber sehr günstig, wodurch wohl eine Zielgruppe von Smarphone-Neulingen angepeilt wird.

6.3 Ausblick

FirefoxOS ist ein noch sehr junges Betriebssystem mit noch vielen “Kindheitsleiden”. Es bleibt abzuwarten, wie schnell Mozilla diese Fehler behebt und sich *FirefoxOS*-Handys in der Nische der Smartphone-Neulinge etabliert. In jedem Fall müssen die Geräte dann von mehr als nur einem Anbieter vertrieben werden und vor allem auch auf das neue Betriebssystem und die günstigen Geräte aufmerksam gemacht werden. Viele Leute, und gerade die, die sich nicht viel mit Smartphones und deren Betriebssystemen auseinandersetzen, haben noch nie etwas von *FirefoxOS* gehört. Da gerade dies die Zielgruppe der Billig-Smartphones ist, ist noch viel Arbeit und Werbung nötig, um sich damit einen Namen zu machen.

Abbildungsverzeichnis

Literaturverzeichnis

[Ali] ALICLOUD: *Aliyun OS Homepage*. http://www.yunos.com/. – zuletzt besucht am: 27.10.2013

[Cam] CAMPBELL, Zac: *UI Testing on FirefoxOS. Working with iFrames*. http://blog.mozilla.org/webqa/2013/02/13/part-2-ui-testing-on-firefox-os-working-with-iframes/. – zuletzt besucht am: 19.12.2013

[cha] CHANNEL firefox y.: *FirefoxOs Demo 09-06-12*. http://www.youtube.com/watch?v=5MzuGWFIfio. – zuletzt besucht am: 17.07.2013

[Con] *Handyauswahl: Alcatal One Touch FIRE weiss orange*. http://www.congstar.de/handy/alcatel-one-touch-fire-weiss-orange/. – zuletzt besucht am: 02.12.2013

[des] *Design Principles*. https://developer.mozilla.org/en-US/Apps/Quickstart/Design/Design_Principles. – zuletzt besucht am: 08.10.2013

[Fou] FOUNDATION, The jQuery: *jQuery mobile*. http://jquerymobile.com/. – zuletzt besucht am: 04.11.2013

[gai] *FirefoxOS Platform - Gaia*. https://developer.mozilla.org/en-US/Firefox_OS/Platform/Gaia. – zuletzt besucht am: 08.10.2013

[geca] *FirefoxOS Architecture Overview - Gecko*. https://developer.mozilla.org/en-US/Firefox_OS/Platform/Architecture#Gecko. – zuletzt besucht am: 14.11.2013

[gecb] *Gecko*. https://developer.mozilla.org/en-US/docs/Mozilla/Gecko. – zuletzt besucht am: 08.10.2013

[gon] *FirefoxOS Platform - Gonk*. https://developer.mozilla.org/en-US/Firefox_OS/Platform/Gonk. – zuletzt besucht am: 08.10.2013

[GPSR13] GEIGER, Philip ; PRYSS, Rüdiger ; SCHICKLER, Marc ; REICHERT, Manfred: Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices / University of Ulm. Version: October 2013. http://dbis.eprints.uni-ulm.de/972/. Ulm : University of Ulm, October 2013 (UIB-2013-09). – Technical Report

[Her] HERRMANN, Jochen: *DBIS uulm auf iTunes*. https://itunes.apple.com/de/app/dbis-universitat-ulm/id448153332?mt=8. – zuletzt besucht am: 21.09.2013

[Inca] INC., Google: *Google JS-Library - Feeds*. https://developers.google.com/feed/. – zuletzt besucht am: 20.11.2013

[Incb] INC., Sencha: *Sencha Touch*. http://www.sencha.com/products/touch. – zuletzt besucht am: 27.10.2013

[Ltd] LTD., Canonical: *Ubuntu for phones*. http://www.ubuntu.com/phone. – zuletzt besucht am: 27.10.2013

[man] *App manifest*. https://developer.mozilla.org/en-US/Apps/Developing/Manifest. – zuletzt besucht am: 08.10.2013

[map] *Google map v3 plugin for jQuery and jQuery Mobile*. code.google.com/p/jquery-ui-map/. – zuletzt besucht am: 04.11.2013

[MDP] MARCEL DU PREEZ, Andrew R. Simon Smith S. Simon Smith: *Box Resizer*. http://www.boxresizer.com. – zuletzt besucht am: 04.11.2013

[MO] MARK OTTO, Chris Rebert Julian T. Jacob Thornton T. Jacob Thornton: *Bootstrap*. http://getbootstrap.com/. – zuletzt besucht am: 27.10.2013

[pck] *Packaged Apps*. https://developer.mozilla.org/en-US/Apps/Publishing/Packaged_Apps. – zuletzt besucht am: 08.10.2013

[per] *App permissions*. https://developer.mozilla.org/en-US/Apps/Developing/App_permissions. – zuletzt besucht am: 04.11.2013

[Pus] *Mozilla Developer Network - FirefoxOS Simulator*. https://developer.mozilla.org/en-US/docs/Tools/Firefox_OS_Simulator#Push_to_device. – zuletzt besucht am: 27.10.2013

[Rel] RELEASES, Mozilla P.: *Mozilla Gains Global Support for a Firefox mobile OS*. <http://blog.mozilla.org/blog/2012/07/02/firefox-mobile-os/. – zuletzt besucht am: 15.10.2013

[sec] *Security Model*. https://developer.mozilla.org/en-US/Firefox_OS/Security/Security_model. – zuletzt besucht am: 08.10.2013

[SSP+13] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; NIENHAUS, Hans ; REICHERT, Manfred: 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*, 2013, 509–518

Name: Julian Frank Matrikelnummer: 711148

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Julian Frank