

QuiXilver Mobile Client

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühlingssemester 2014

Autoren:	Flavio Reinhard & Ken Steiner
Betreuer:	Prof. Dr. Olaf Zimmermann & Sebastian Hunkeler
Projektpartner:	Vigience AG, Horgen
Experte:	Roman Schlegel, ABB
Gegenleser:	Prof. Dr. Eduard Glatz

Abstract

Die Software QuiXilver der Firma Vigience AG bietet Cloud-basiertes Taskmanagement für Unternehmen an. Aktuell stehen aus Benutzersicht ein Desktop- und ein Browser-Client zur Verfügung. Der Browser-Client kann auch auf Smartphones benutzt werden. Er erfüllt jedoch die folgenden Anforderungen nicht: Nachrichten in Echtzeit erhalten, nach einem Neustart der App angemeldet bleiben und Benutzeroberfläche durchgehend auf mobile Geräte optimieren. Deshalb plant der Industriepartner, das Produkt QuiXilver um eine mobile App zu erweitern. Ergänzend zu den beschriebenen Anforderungen ist die Plattformunabhängigkeit eine zentrale nichtfunktionale Anforderung; Die App soll den Kunden auf iOS, Android und Windows Phone zur Verfügung stehen, plattformspezifische Entwicklungsaufwände dabei aber minimiert werden. Die Bachelorarbeit hat zum Ziel, den Industriepartner bei der Technologiewahl zu unterstützen.

Zunächst werden ein Kriterienkatalog präsentiert und geeignete Cross-Plattform-Lösungen vorgestellt. Anschliessend werden sechs Technologien unter Verwendung des Kriterienkatalogs qualitativ auf ihre Vor- und Nachteile evaluiert und PhoneGap als geeignetste Technologie identifiziert. Danach wird der Prototyp der QuiXilver Mobile App vorgestellt, welcher die Technologieempfehlung legitimiert. Der Prototyp realisiert die besonders architekturelevanten Use Cases aus der Anforderungsanalyse. Mit dem vorliegenden Prototypen können Mitteilungen/Aufgaben gelesen und mittels Push-Benachrichtigungen in Echtzeit empfangen werden.

Die Technologie-Evaluation zeigt, dass Push-Benachrichtigungen und Windows Phone 8 Kompatibilität die Auswahl an möglichen Technologien stark einschränken. Die beiden Anforderungen können nur mit PhoneGap und Xamarin umgesetzt werden. Bei der Entwicklung mit Xamarin kann lediglich die Logik für die verschiedenen Plattformen wiederverwendet werden, was die geforderte Plattformunabhängigkeit beeinträchtigt. Der entwickelte Prototyp zeigt, dass die Anforderungen der QuiXilver Mobile App mit PhoneGap erfüllt werden können. Für die Abdeckung aller geforderten Plattformen waren lediglich 30 Zeilen plattformspezifischen Codes nötig. Nachteile sind die limitierte Verfügbarkeit von Systemaufrufen sowie minimale Zeitverzögerungen bei der Bedienung der Benutzeroberfläche.

Management Summary

Ausgangslage

QuiXilver Die Software QuiXilver der Firma Vigience AG bietet Cloud-basiertes Taskmanagement für Unternehmen an. Aktuell stehen aus Benutzersicht ein Desktop- und ein Browser-Client zur Verfügung. Der Browser-Client kann auch auf Smartphones benutzt werden. Er erfüllt jedoch die folgenden Anforderungen nicht:

- Nachrichten in Echtzeit erhalten
- Nach einem Neustart der App angemeldet bleiben
- Benutzeroberfläche auf mobile Geräte optimieren

Vorhaben Aus diesen Gründen plant der Industriepartner, das Produkt QuiXilver um eine mobile App zu erweitern. Ergänzend zu den beschriebenen Anforderungen ist die Plattformunabhängigkeit eine zentrale nichtfunktionale Anforderung; Die App soll den Kunden auf iOS, Android und Windows Phone zur Verfügung stehen, plattformspezifische Entwicklungsaufwände dabei aber minimiert werden. Die Bachelorarbeit hat zum Ziel, den Industriepartner bei der Technologiewahl zu unterstützen.

Vorgehen, Technologien

Technologie-Evaluation Zunächst werden ein Kriterienkatalog präsentiert und geeignete Cross-Plattform-Lösungen vorgestellt. Anschliessend werden die folgenden sechs Technologien unter Verwendung des Kriterienkatalogs qualitativ auf ihre Vor- und Nachteile evaluiert:

- Adobe PhoneGap
- AppGyver Steroids
- Xamarin
- jQuery Mobile
- OpenUI5

Adobe PhoneGap wird als geeignetste Technologie für den vorliegenden Anwendungsfall identifiziert. Grund dafür sind die folgenden Eigenschaften:

- Unterstützt Push Benachrichtigungen (Empfangen von Daten in Echtzeit)
- Unterstützt die geforderten Plattformen
- Sehr tiefe Lizenzkosten
- Bestehende Teile von QuiXilver Online können wiederverwendet werden, da die selbe Entwicklungstechnologie zum Einsatz kommt
- Entwicklerteam des Industriepartners besitzt Know-How in den von PhoneGap verwendeten Technologien
- Grosse Entwicklergemeinde

QuiXilver Mobile Prototyp Im Anschluss an die Technologie-Evaluation wird der Prototyp der QuiXilver Mobile App vorgestellt, welcher die Technologieempfehlung legitimiert. Der Prototyp realisiert die besonders architekturelevanten Use Cases aus der Anforderungsanalyse. Mit dem vorliegenden Prototypen können Mitteilungen/Aufgaben gelesen und mittels Push-Benachrichtigungen in Echtzeit empfangen werden. Untenstehende Abbildung zeigt den Prototypen auf iOS, Android und Windows Phone.

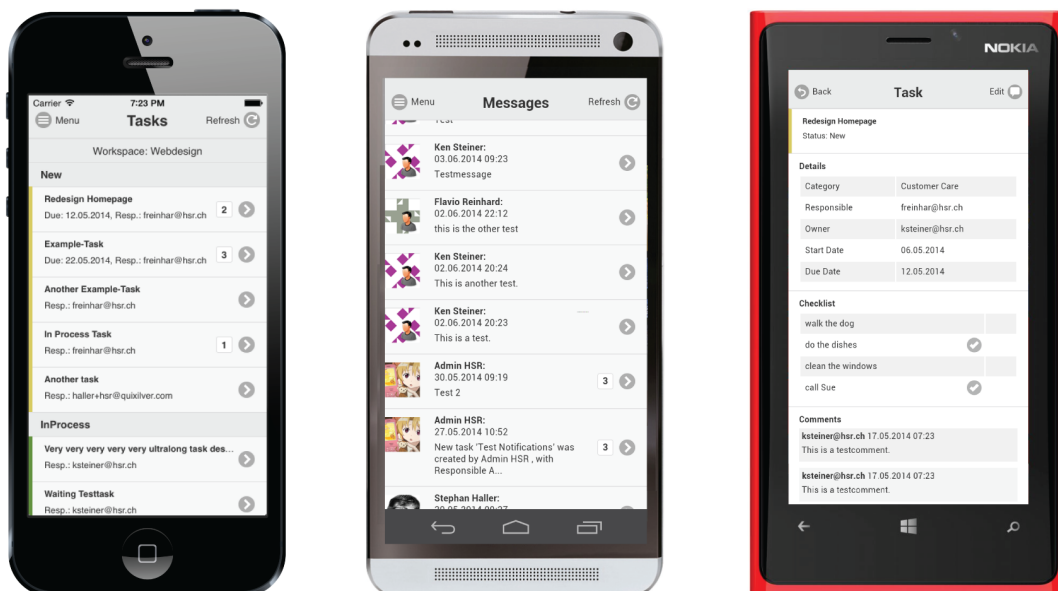


Abbildung 1: QuiXilver Mobile Prototyp auf geforderten Plattformen

Ergebnisse

Technologie-Evaluation Die Technologie-Evaluation zeigt, dass der Datenempfang in Echtzeit (Push Benachrichtigungen) und Windows Phone 8 Kompatibilität die Auswahl an möglichen Technologien stark einschränken. Die beiden Anforderungen können nur mit PhoneGap und Xamarin umgesetzt werden.

Bei der Entwicklung mit Xamarin können lediglich Teile der Applikation für die verschiedenen Plattformen wiederverwendet werden, was die geforderte Plattformunabhängigkeit beeinträchtigt und den plattformspezifischen Entwicklungsaufwand erhöht.

QuiXilver Mobile Prototyp Der entwickelte Prototyp zeigt, dass die Anforderungen der QuiXilver Mobile App mit PhoneGap erfüllt werden können. Für die Abdeckung aller geforderten Plattformen waren lediglich 30 Zeilen plattformspezifischen Codes nötig. Der Einsatz von PhoneGap hat die folgenden Nachteile mit sich gebracht:

- Minimale Zeitverzögerung bei der Bedienung der Benutzeroberfläche
- Limitierte Verfügbarkeit von Systemaufrufen

Ausblick

Die Anwendung der PhoneGap-Technologie hat diverse Vor- und Nachteile mit sich gebracht. Für zukünftige App-Projekte kann PhoneGap eingesetzt werden, sofern die untenstehenden Bedingungen erfüllt sind:

- Hohe Anforderungen an die Plattformunabhängigkeit
- Performanceeinbussen bei der Bedienung der Benutzeroberfläche sind verkraftbar
- Know-How in Web-Technologien beim Entwicklungsteam vorhanden
- Die App soll hauptsächlich Listen darstellen. Keine 3D-Animationen oder ähnliches notwendig.
- Keine native Darstellung der Benutzeroberfläche notwendig

Inhaltsverzeichnis

I	Technischer Bericht	11
1	Ausgangslage	12
1.1	Kurzeinführung QuiXilver	12
1.1.1	Hauptaufgaben	12
1.1.2	Systemübersicht	13
1.1.3	Allgemeine Begriffe	14
1.1.4	Apps	15
1.1.5	Grafische Oberfläche	16
1.2	Marktsituation Mobile	19
2	Problembeschreibung	20
2.1	Probleme des aktuellen QuiXilver Online Client	20
2.2	Problem: Technologiewahl für App	21
2.3	Anforderungen an die QuiXilver Mobile App	21
2.3.1	Funktionale Anforderungen	21
2.3.2	Nicht-funktionale Anforderungen	24
3	Aufgabenstellung	25
4	Technologie Evaluation Cross Plattform Entwicklung	26
4.1	Beschreibung des Evaluationsprozesses	26
4.2	Definition der Miniprototyp-Anforderungen	27
4.3	Definition der Testkriterien	28
4.3.1	Portabilität	29
4.3.2	Security	30
4.3.3	Community	30
4.3.4	Erlernbarkeit und Dokumentation	30
4.3.5	Lizenzbedingungen	31
4.3.6	Wartbarkeit	31
4.3.7	Entwicklungsumgebung	31
4.3.8	Installationsdauer	31
4.3.9	Effizienz	32
4.3.10	Fluidität der Benutzerschnittstelle	32
4.3.11	Performance	32
4.4	Technologie-Recherche	33
4.4.1	Entwicklungskonzepte mobile Apps	33
4.4.2	Konzepte Push-Benachrichtigungen	35

4.4.3	Konkurrenzanalyse	40
4.4.4	Produktübersicht	42
4.4.5	PhoneGap	43
4.4.6	Steroids	45
4.4.7	Titanium	47
4.4.8	Xamarin	48
4.4.9	jQuery Mobile	50
4.4.10	OpenUI5	51
4.5	Übersicht der Testresultate	53
4.5.1	Adobe PhoneGap	53
4.5.2	AppGyver Steroids	54
4.5.3	Xamarin	55
4.5.4	jQuery Mobile	56
4.5.5	OpenUI5	57
4.6	Detaillierte Testresultate	57
4.7	Technologie-Entscheidung	58
4.7.1	Empfehlung des Projektteams	58
4.7.2	Entscheidung des Industriepartners	58
5	Lösungskonzept QuiXilver Mobile Prototyp	59
5.1	Systemübersicht	59
5.2	Schichtenarchitektur QuiXilver Mobile	60
5.2.1	View	60
5.2.2	Presenter	61
5.2.3	DataAccess	61
5.2.4	Model	62
5.2.5	Services	64
5.2.6	Util	64
5.2.7	Externe Komponenten	65
5.2.8	Zusammenarbeit der Schichten	67
5.3	Architekturentscheidungen	69
5.3.1	Persistierungskonzept	69
5.3.2	Single-page App	72
5.3.3	Wahl eines Benutzeroberflächen-Frameworks	76
5.3.4	Einsatz eines Architektur-Frameworks	79
5.3.5	Wahl einer Template-Library	82
5.4	Spezifikation Push Service	84
5.5	User Interface	85
5.5.1	Anmeldung	85
5.5.2	Messages	86
5.5.3	Tasks	87
5.5.4	Navigation	88
6	Umsetzung QuiXilver Mobile	90
6.1	Struktur des Projektordners	90
6.2	Login-Prozess	91
6.2.1	Konzeptioneller Ablauf	91

6.2.2	Einsatz des InAppBrowser-Plugin	93
6.3	Implementation des Push Services	94
6.3.1	Registrierung beim plattformspezifischen Push Server	95
6.3.2	Registrierung beim QuiXilver Server	96
6.3.3	Erhalten einer Push Benachrichtigung	96
7	Technologieerkenntnisse PhoneGap	97
7.1	Stärken	97
7.2	Schwächen	97
7.3	Empfohlene Einsatzgebiete	99
8	Ausblick QuiXilver Mobile	100
8.1	Use Cases	100
8.2	Verschiedenene Verbesserungen	100
	Quellenverzeichnis	101
	Glossar	105
	Abbildungsverzeichnis	106
	Tabellenverzeichnis	108
II	Weitere Projektliefergegenstände	109
9	Analyse der funktionalen Anforderungen	110
9.1	Aktorenbeschreibung	110
9.1.1	Benutzercharakteristik	110
9.2	Use Case Diagramm	111
9.2.1	UC 01: App öffnen	112
9.2.2	UC 02: Anmelden	113
9.2.3	UC 03: Abmelden	114
9.2.4	UC 04: Workspace wechseln	115
9.2.5	UC 05: Aufgabensfortschritt kontrollieren	116
9.2.6	UC 06: Neue Mitteilung erfassen	117
9.2.7	UC 07: Mitteilungen ansehen	118
9.2.8	UC 08: Push-Benachrichtigung erhalten	119
9.2.9	UC 09: Aufgabenstatus ändern	120
10	Analyse der nicht-funktionalen Anforderungen	121
10.1	Portabilität	121
10.2	Interoperabilität	121
10.3	Benutzerschnittstellen	121
10.3.1	Erlernbarkeit	122
10.4	Wiederherstellbarkeit	122
10.5	Sicherheit	123
10.6	Anpassbarkeit	123

11	Detailresultate Technologie Evaluation	124
11.0.1	PhoneGap	124
11.0.2	AppGyver Steroids	129
11.0.3	Xamarin	134
11.0.4	jQuery Mobile	138
11.0.5	openUI5	141
12	Benutzerschnittstelle und Usability	145
12.1	Entwurf eines Prototyps	145
12.1.1	Anforderungen	145
12.1.2	Paperprototyping	146
12.1.3	Benutzerschnittstelle: Erster Entwurf	150
12.2	Usability Testing	154
12.2.1	Analyse mit Stakeholdern	154
12.3	Cognitive Walkthrough	155
12.3.1	Definition der Handlungssequenzen	155
12.3.2	Untersuchung der Handlungssequenzen	155
12.3.3	Testplan	157
12.3.4	Aufgaben	158
12.3.5	Testlog	159
12.3.6	Erkenntnisse	162
12.4	Korrekturen am Prototyp	162
13	API-Beschreibungen	166
13.1	Request Token	166
13.1.1	Request	166
13.1.2	Response	166
13.2	Push Registrierung	167
13.2.1	Request	167
13.2.2	Response	167
13.3	Datenzugriff Messages	168
13.3.1	Request	168
13.3.2	Response	168
13.4	Datenzugriff Tasks	168
III	Technologie Tutorium PhoneGap	169
13.5	Requirements	170
13.6	Deploy QuiXilver Mobile Prototype	170
13.6.1	Preperation	170
13.6.2	Deploy on iOS	170
13.6.3	Deploy on Android	172
13.6.4	Deploy Windows Phone	173
13.7	Architectural Overview	175
13.7.1	View	175
13.7.2	Presenters	176
13.7.3	DataAccess	176

13.7.4 Model	177
13.7.5 Services	179
13.7.6 Util	179
13.7.7 Externe Komponenten	180
 IV Spezifikation Push Service	 182

Teil I

Technischer Bericht

1 Ausgangslage

1.1 Kurzeinführung QuiXilver

Dieses Kapitel beschreibt das Produkt QuiXilver, für welches eine Mobile-App entwickelt werden soll.

1.1.1 Hauptaufgaben

QuiXilver ist ein Service, welcher die Software-Plattform in der Cloud zur Verfügung stellt (engl. platform as a service oder PaaS). Dies bedeutet folgendes: Der Kunde entwickelt eine Software mit der Unterstützung durch Werkzeuge, Bibliotheken und Datenbanken, welche durch den Serviceprovider zur Verfügung gestellt werden. Der Anbieter stellt das Netzwerk, Server und Speicherplatz zur Verfügung, dass die Kundensoftware gehostet werden kann.

Um dem Kunden die Einstiegshürde zu verkleinern, bietet die QuiXilver Plattform bereits ein Grundgerüst an verfügbaren Services an. Die wichtigsten Funktionen dieses Grundgerüsts sind:

- Synchronisierte Aufgabenlisten
- Strukturierung in Workspaces
- Benutzerverwaltung
- Teamzusammenarbeit
- Messages
- Datenverwaltung und -verteilung

Zusätzlich zum Grundgerüst können individuelle Apps entwickelt oder im eigenen App Store eingekauft werden, um die Funktionalität der Cloudplattform zu erweitern. Das Angebot an verfügbaren Apps im eigenen App Store wird laufend erweitert.

1.1.2 Systemübersicht

Untenstehende Abbildung stellt den Aufbau des QuiXilver-Systems dar. Anschließend werden alle wichtigen Komponenten beschrieben.

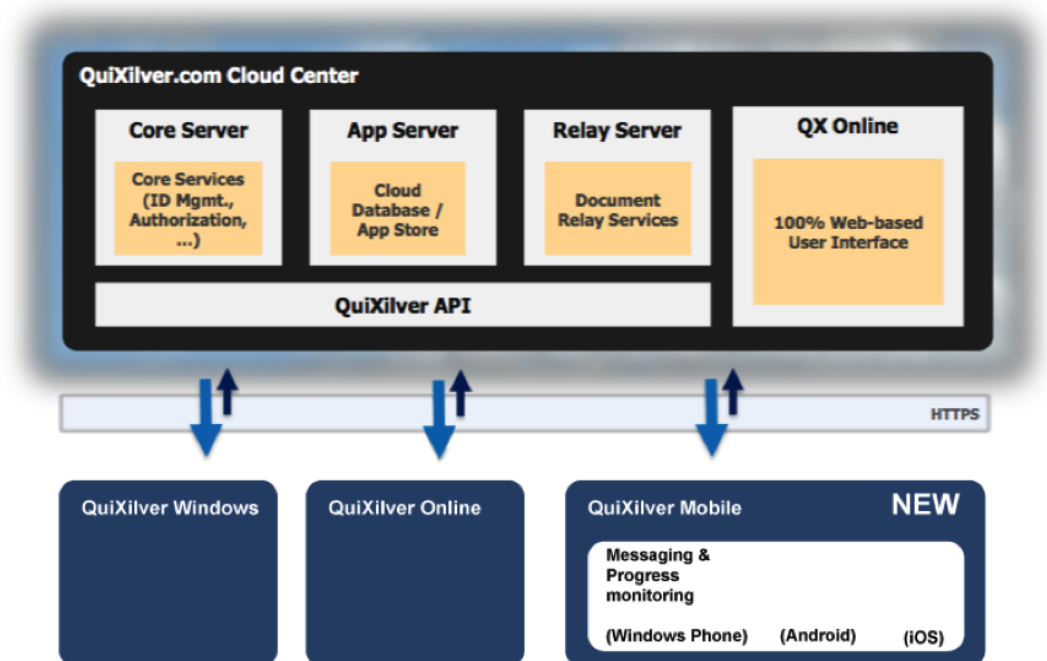


Abbildung 1.1: Systemübersicht QuiXilver

Die aktuelle QuiXilver Umgebung basiert auf einem Client-Server-Modell mit den folgenden Komponenten:

QuiXilver Cloud Center Der Server bietet folgende Services an:

- Database-access (App Server): Zuständig für Bereitstellung der Daten über RESTful-Schnittstelle.
- Document Relay Service (Relay Server): Zuständig für Speicherung und Synchronisierung von Dokumenten auf die Desktop-Clients.
- Management Services (Core Server): Zuständig für die Benutzerverwaltung und Sicherheit.

QuiXilver Windows C-Sharp .NET basierter nativer Client für Windows.

QuiXilver Online HTML/JavaScript/CSS basierter Client für Browser (Internet Explorer, Chrome, Firefox). Nur die wichtigsten Seiten sind optimiert für mobile.

QuiXilver Mobile Im Rahmen dieser Bachelorarbeit zu entwickelnde Mobile-App. Installierbar auf Windows Phone, Android und iOS.

1.1.3 Allgemeine Begriffe

Dieses Kapitel beschreibt die bei QuiXilver verwendete Terminologie. Als Beginn gibt die nachfolgenden Grafik eine Übersicht der Elemente. Anschliessend folgt die Erklärung der wichtigsten Elemente.

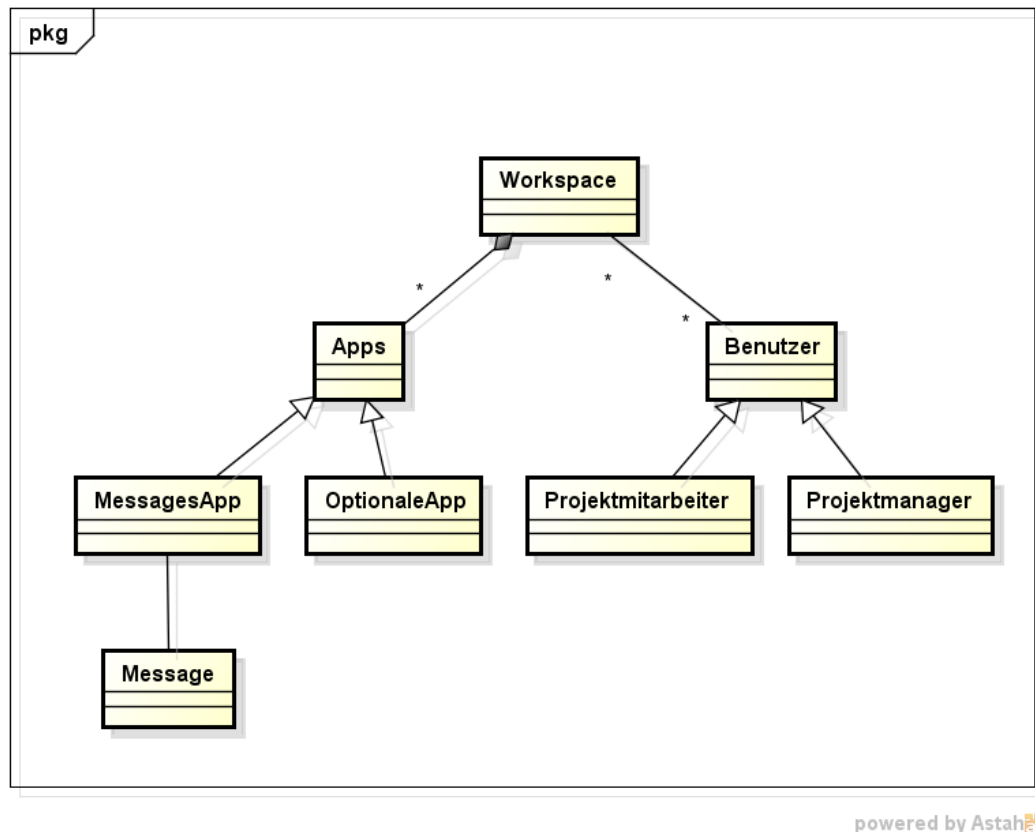


Abbildung 1.2: Domain-Übersicht QuiXilver

Ein Benutzer stellt einen Menschen dar, der QuiXilver benutzt. Auf der Plattform können benutzerspezifische Rechte gesetzt werden.

Workspace bezeichnet einen isolierten Arbeitsordner. Er enthält die Apps *Messages* und *Dateimanager* und ist einer relationalen Datenbank zugewiesen. Ein Workspace wird als Arbeitsumgebung für ein Projekt oder ein Programm verwendet. Diese beiden Verwendungstypen sind wie folgt definiert:

- **Projekte** sind typischerweise einmalige Vorhaben mit einem fixen Start- und Enddatum. Häufig wird in Projekten ausserhalb der normalen Organisationsstruktur gearbeitet.
- **Programme** sind (auf unbestimmte Zeit) wiederkehrende, stets neu abzuarbeitende Arbeitspakete.

1.1.4 Apps

Apps stehen für Webapplikationen, die innerhalb eines Workspace installiert werden können. Ein QuiXilver-Kunde hat die Möglichkeit, eigene Apps zu entwickeln und diese auf seinem Workspace zu installieren. Im App Store von QuiXilver (nicht zu verwechseln mit dem App-Store von Apple) können auch Apps bezogen werden, welche von Vigience und Partnern entwickelt wurden. Diese Apps können auf mehreren Workspaces installiert werden. Jeder Workspace erhält dabei eine eigene Instanz der App. Alle Apps haben Zugriff auf die relationale Datenbank und alle Dateien des Workspaces. Das Angebot des App Stores gliedert sich in die folgenden Kategorien.

Vorinstallierte Apps

Vorinstallierte Apps sind in jedem Workspace vorhanden. Sie können grundsätzlich auch jederzeit vom Benutzer entfernt werden. Zu den vorinstallierten Apps gehören:

Die Dateiverwaltung zeigt dem Benutzer die Dateien des aktuellen Workspaces an.

Messages bietet den Benutzern des Workspaces die Möglichkeit, Nachrichten zu verschicken. Diese Nachrichten haben die folgenden Eigenschaften:

- Messages können an alle Benutzer des Workspaces (Broadcast) oder an einzelne (Unicast) gesendet werden.
- Messages können Referenzen zu Dateien enthalten.
- Messages können auch von anderen Apps erfasst und versandt werden.
- Zu jeder Message können Kommentare hinzugefügt werden.

Optionale Apps

Im Gegensatz zu den vorinstallierten Apps müssen die optionalen Apps dem Workspace manuell hinzugefügt werden. Sie können im App Store zugekauft oder selbst entwickelt werden. Ein häufig genutztes Beispiel für eine optionale App:

Die Aufgabenverwaltung enthält Aufgaben in Form von Tickets.

1.1.5 Grafische Oberfläche

Nachdem die Umgebung und Begrifflichkeiten von QuiXilver in den vorherigen Abschnitten beschrieben wurde, zeigt dieser Unterabschnitt, wie die beschriebenen Elemente im aktuellen Desktop-Client aussehen. Dies stellt keine detaillierte Einführung in die grafische Oberfläche von QuiXilver dar, sondern trägt lediglich für ein besseres Verständnis des Produkts bei.

Workspace-Auswahl und deren Management befindet sich in der Kopfzeile der Seite. Anhand eines Drop-Down-Menus kann der Benutzer den aktuell geladenen Workspace ansehen oder eine Alternative auswählen.

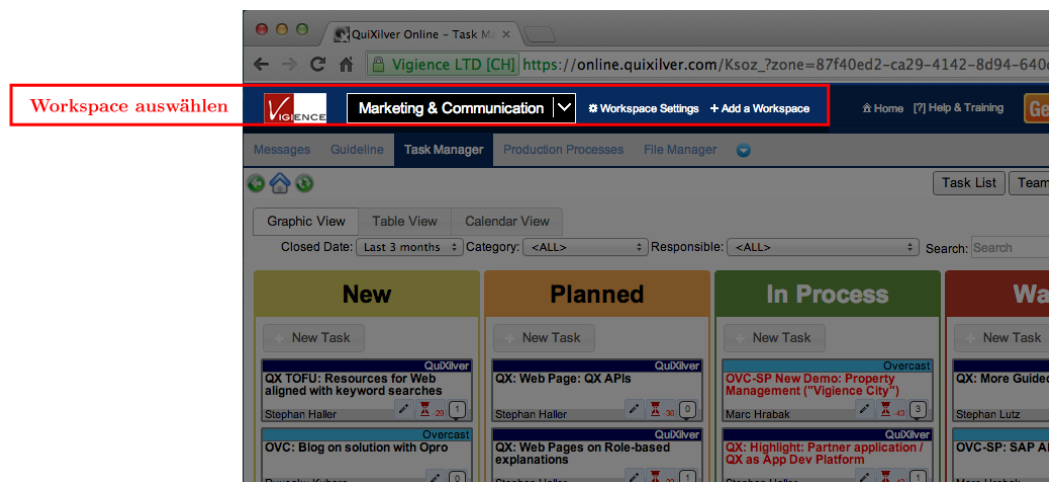


Abbildung 1.3: QuiXilver Online: Workspace-Auswahl

Apps können durch das Selektieren in der App-Navigation geöffnet werden. Nach der Selektion werden sie im App-Bereich geöffnet. Untenstehendes Beispiel zeigt die Task-Manager-App.

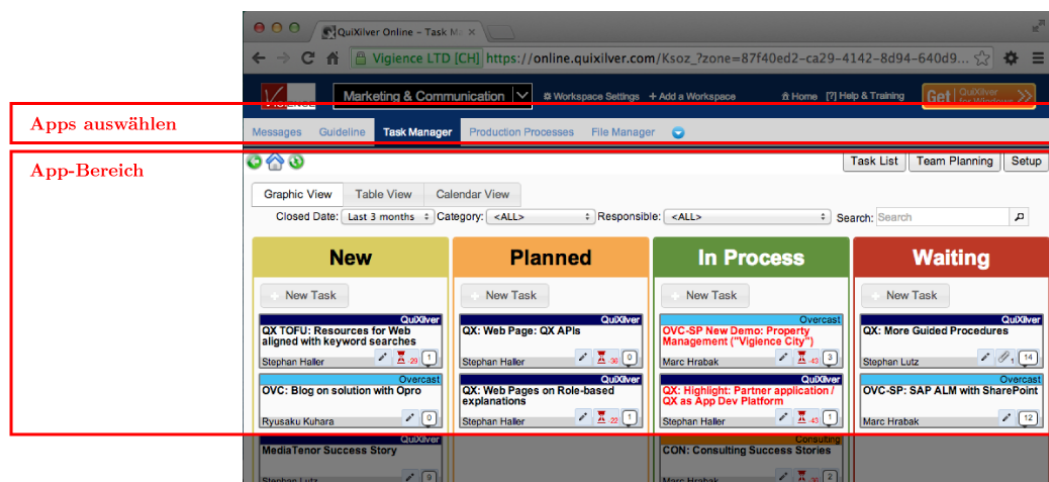


Abbildung 1.4: QuiXilver Online: App-Auswahl und Darstellung

1.1 Kurzeinführung QuiXilver

Messages besteht aus zwei Hauptbereichen. Im oberen Bereich können neue Messages erfasst werden. Im unteren Bereich können die gesendeten Messages gelesen werden. Diese Auflistung von alten Messages wird als **Wall** bezeichnet.

Attach Files

Select Users ▾

Post a new message here

Post



Stephan Haller
Stephan Haller updated QTM: Task Manager.
September 10 at 4:35: PM - [Comment](#)

view all 16 comments

**Stephan Haller**
Stephan Haller updated Task Manager.
9 hours ago

**Stephan Haller**
Stephan Haller updated Task Manager.
9 hours ago

Comment



Task Manager
New task "'Project collaboration" feature page on QX' was created by **Conference Room Switzerland** , with Responsible **Stephan Haller** and Owner **Markus Stierli**.
Tuesday at 3:25 PM - [Comment](#)



Task Manager
Comment 'Describe how QX helps project managers:
1) QX helps to keep all content in one place: files, tasks, messages.
Across company boundaries and remote locations
- file sharing and replication is great. task management is great.
messaging is great.
Beispiel: Nintendo Projekt (we keep all our project info for NEC and Nintendo in one place)

2) For IT project managers and product managers, we offer an
optimized task manager with the "Agile Product Management" which
allows to manage the tasks of a product development unit - for

Abbildung 1.5: QuiXilver Online: Messages

Die **Aufgabenverwaltung** besteht aus mehreren Ansichten. Folgend wird die wichtigste Ansicht Aufgabenübersicht aufgeführt. Für jede Aufgabe sind Informationen zum aktuellen Zustand ersichtlich.

Table View

1-20 / 20 rows per page: [10] [50] [100] [All]

Search:

Save all New Delete Duplicate

			Task	Category	Responsible	Status	Owner	Due Date	Prior...
			OVC-SP New Demo: Property Management ("Vigence City")	Overcast	Marc Hrabak	In Process	Stephan Haller	2014/01/17	1.1
			QX: Highlight: Partner application / QX as App Dev Platform	QuiXilver	Stephan Haller	In Process	Stephan Haller	2014/01/17	1.6
			CON: Consulting Success Stories	Consulting	Marc Hrabak	In Process	Stephan Haller	2014/01/24	1.65
			QX: Web Page: QX APIs	QuiXilver	Stephan Haller	Planned	Stephan Haller	2014/01/24	1.7
			QX TOFU: Resources for Web aligned with keyword searches	QuiXilver	Stephan Haller	New	Stephan Haller	2014/01/31	1.9
			OVC-SP: Use as Order Management Hub (ABB)	Overcast	Andrew McLeod	In Process	Stephan Haller	2014/01/31	1.95
			OVC: Vigence in Gartner's iPaaS quadrant	Overcast	Stephan Haller	In Process	Stephan Haller	2014/01/31	1.98
			OVC-SP: How to develop a user-friendly UI	Overcast	Stephan Haller	In Process	Stephan Haller	2014/02/07	2
			QX: Web Pages on Role-based explanations	QuiXilver	Stephan Haller	Planned	Stephan Haller	2014/02/07	13
			OVC: Blog on solution with Opro	Overcast	Ryusaku Kuhara	New	Stephan Haller		19
			QX: More Guided Procedures	QuiXilver	Stephan Lutz	Waiting	Stephan Haller		35
			MediaTenor Success Story	QuiXilver	Stephan Lutz	New	Madhus Stiedl		43
				None	Ken Steine	New	Ken Steine		58

Abbildung 1.6: QuiXilver Online: Aufgabenverwaltung

1.2 Marktsituation Mobile

Mit den zunehmenden Marktanteilen der mobilen Betriebssystemen wird es für Unternehmen immer wichtiger, ihre Dienstleistungen auch auf mobilen Plattformen anzubieten. Die Entwicklung solcher mobiler Software hat sich in der Vergangenheit als sehr kostenintensive Angelegenheit erwiesen, da unterschiedliche Betriebssysteme auf dem Markt sind. Nachfolgende Tabelle zeigt die weltweiten Verkaufszahlen von Smartphones (in Tausend Stück) von 2012 und 2013, gruppiert nach Betriebssystemen.

OS	2013 Units	2013 Market Share (%)	2012 Units	2012 Market Share (%)
Android	758,719.9	78.4	451,621.0	66.4
iOS	150,785.9	15.6	130,133.2	19.1
Microsoft	30,842.9	3.2	16,940.7	2.5
BlackBerry	18,605.9	1.9	34,210.3	5
Other OS	8,821.2	0.9	47,203.0	6.9
Total	967,775.8	100	680,108.2	100

Tabelle 1.1: Weltweite Verkaufszahlen Smartphones

Diese Zahlen ¹ stellen die Unternehmen vor die folgende Situation:

Voraussetzung des Marktanteils Um einen sinnvollen Anteil an Nutzern zu erreichen, darf die Software nicht nur auf der Android-Plattform zur Verfügung stehen. Weitere Plattformen wie iOS und Microsoft Windows Phone sollten ebenfalls berücksichtigt werden. Da jede Plattform eine eigene Programmiersprache benutzt, muss die selbe Anwendung jedoch mehrfach programmiert werden. Die dadurch verursachten Kosten stellen eine grosse Eintrittshürde für Unternehmen in das mobile Softwaresegment dar.

Gewinn des Marktanteils Um die Softwareentwicklung zu beschleunigen und kostengünstiger zu gestalten, gibt es diverse Ansätze, um das Problem der mehrfachen Programmierung zu beheben. Die Lösungen haben alle gemeinsam, dass die Software in einer Sprache geschrieben wird und das resultierende Programm auf mehreren Plattformen lauffähig ist. Dies wird plattformunabhängige Programmierung genannt.

¹<http://www.gartner.com/newsroom/id/2665715>, zuletzt abgerufen am 19.03.2014, 18:14 Uhr

2 Problembeschreibung

2.1 Probleme des aktuellen QuiXilver Online Client

Die aktuelle Version des QuiXilver Online Clients ist über einen mobilen Browser abrufbar und sieht wie folgt aus:

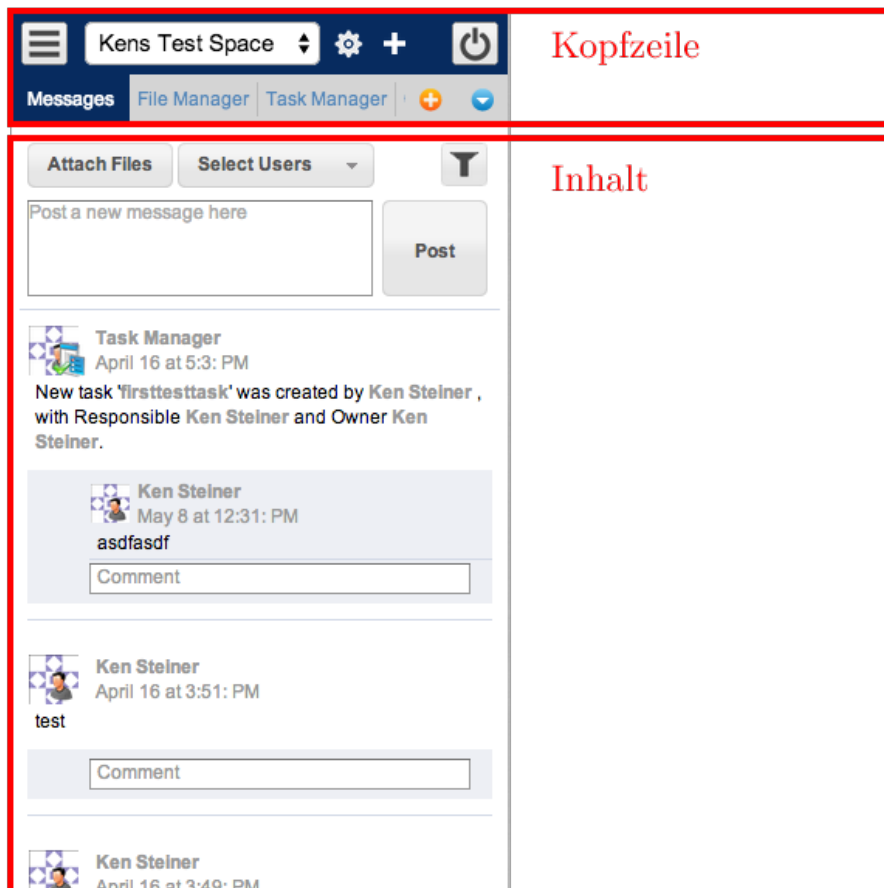


Abbildung 2.1: Mobile Ansicht: QuiXilver Online

Gemäss Industriepartner werden folgende Anforderungen nicht erfüllt:

- Push Benachrichtigungen bei neuen Mitteilungen und Aufgaben erhalten
- Angemeldet bleiben über eine längere Dauer als eine Session (Sessiondauer: 30 Minuten)
- Nicht alle Seiten sind für mobile Displays optimiert. Bei nicht-optimierten Seiten wird die Benutzeroberfläche der Desktop-Version geladen
- Design der Benutzeroberfläche wirkt sehr technisch und unfreundlich

Um die oben erwähnten Probleme zu beheben, hat sich die Vigience AG entschieden, die QuiXilver-Produktpalette um eine App zu erweitern.

2.2 Problem: Technologiewahl für App

Aus wirtschaftlichen Gründen interessiert sich Vigience AG für die plattformunabhängige Programmierung. Mit diesem Ansatz muss die mobile App nur einmal entwickelt werden. Anschliessend kann sie direkt oder mit minimalen Aufwand auf den verschiedenen Plattformen zum Laufen gebracht werden. Hierzu kommen folgende Technologien in Frage:

- Cross-Plattform-Technologien
- Web-Technologien

Aktuell sind auf dem Markt sehr viele Angebote für plattformunabhängige Entwicklungstechnologien im App-Bereich verfügbar.

2.3 Anforderungen an die QuiXilver Mobile App

In diesem Abschnitt werden die gemeinsam mit dem Industriepartner Vigience AG ermittelten, wichtigsten Anforderungen beschrieben. Die detaillierte, umfassende Beschreibung der Anforderungen sind im Anhang in Kapitel 9 zu finden.

2.3.1 Funktionale Anforderungen

In diesem Unterabschnitt werden die wichtigsten funktionalen Anforderungen in der Form von Akteuren und deren Use Cases beschrieben.

Akteuren

Die App wird hauptsächlich von zwei Akteuren bedient: Projektmanager und Projektmitarbeiter.

Aktor: Projektmanager Der Projektmanager hat das Hauptinteresse, den Fortschritt der Arbeitspakete schnell und effizient zu überwachen. Dabei ist ihm wichtig, dass der aktuelle Stand der Arbeitspakete effizient ersichtlich ist.

Aktor: Projektmitarbeiter Der Projektmitarbeiter hat das Hauptinteresse, die anstehenden Arbeitspakete schnell und effizient abzuarbeiten. Dabei ist ihm wichtig, dass zugewiesene Arbeitspakete schnell ersichtlich sind und die Kommunikation mit dem Team möglichst simpel gestaltet ist.

Use Cases

Folgend werden die Anwendungsfälle der Software anhand von Use Cases beschrieben. Hierzu wird zuerst ein Use Case Diagram abgebildet. Im Anschluss folgt eine tabellarische Beschreibung aller Use Cases.

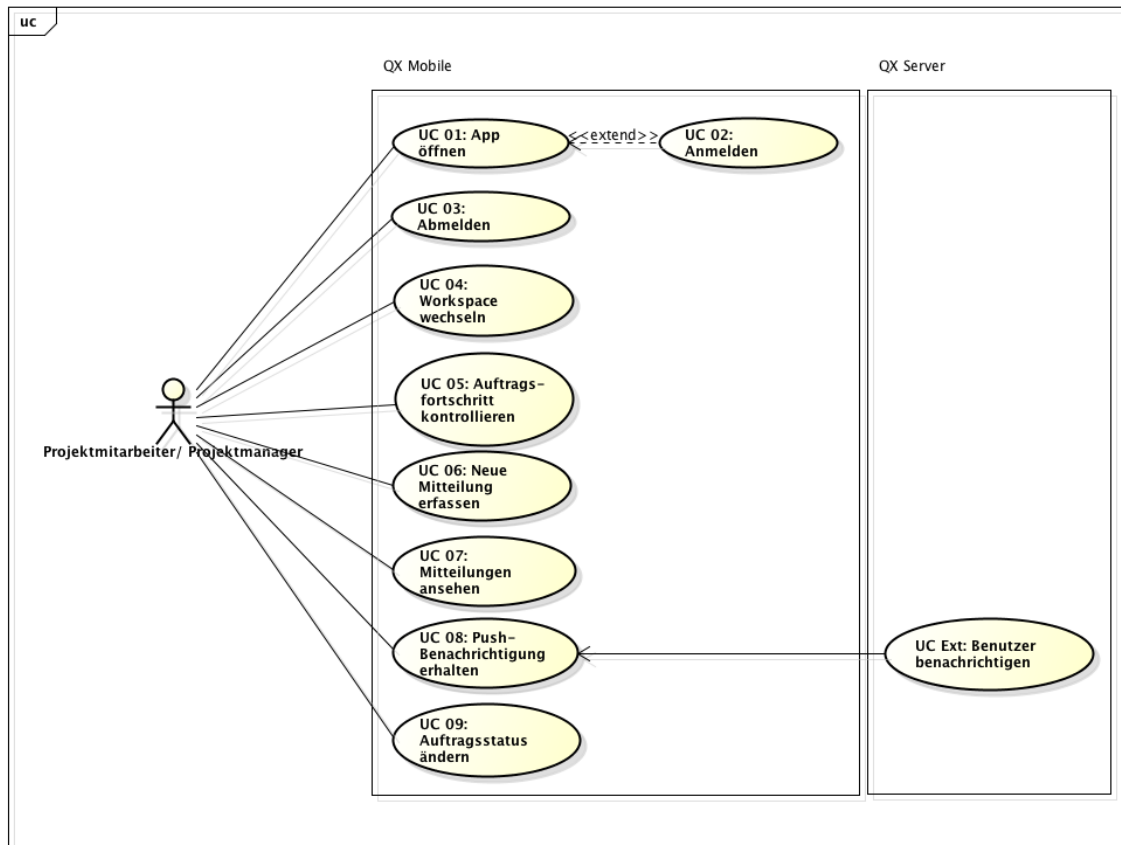


Abbildung 2.2: Use Case Diagram

In der obigen Grafik ist ersichtlich, dass die Benutzerkategorien (Projektmitarbeiter/Projektmanager) in einem einzigen Akteur untergebracht sind. Damit wird gezeigt, dass nicht nur ein einziger Akteur existiert. Tatsächlich sind es zwei unterschiedliche Akteure, denen zu diesem Zeitpunkt jedoch die identischen Use Cases angeboten werden. In einem späteren Entwicklungsstadium könnten den beiden Akteuren unterschiedliche Use Cases zugewiesen werden.

2.3 Anforderungen an die QuiXilver Mobile App

In der folgenden Tabelle werden die wichtigsten Use Cases kurz beschrieben:

ID	Titel	Beschreibung
01	App öffnen	Der Benutzer öffnet über den Homescreen des Betriebssystems die Applikation
02	Anmelden	Der Benutzer meldet sich mit gültigem Benutzername und Passwort bei der Anwendung an
03	Abmelden	Der Benutzer meldet sich ab
04	Workspace wechseln	Der Benutzer wechselt den Workspace. Die Ansichten zu Mitteilungen und Aufgaben enthalten danach die Daten des neu gewählten Workspaces.
05	Aufgabenfortschritt kontrollieren	Benutzer lässt sich den Aufgabenfortschritt des aktuell geladenen Workspaces anzeigen
06	Neue Mitteilung erfassen	Benutzer postet eine Mitteilung an die Wall oder sendet sie an ausgewählte Workspace-Beteiligte
07	Mitteilung ansehen	Benutzer liest Mitteilungen und deren Kommentare
08	Push-Benachrichtigung erhalten	Benutzer erhalten bei relevanten Neuigkeiten (z.B. Erledigung eines Arbeitsschrittes) in Echtzeit Benachrichtigungen. Sobald die gewünschten Ereignisse eintreten, wird der Benutzer direkt auf dem Smartphone informiert.
09	Aufgabenstatus ändern	Benutzer ändert den Status einer Aufgabe

Siehe Kapitel 9 im Anhang für Use Cases Fully Dressed. Die Gestaltung der Software hat sich an obige Use Cases zu richten.

2.3.2 Nicht-funktionale Anforderungen

In diesem Unterabschnitt werden die wichtigsten nicht-funktionalen Anforderungen vorgestellt. Die Begriffe der nicht-funktionalen Anforderungen richten sich an der ISO/IEC 9126 ¹.

Portabilität ist die zentrale Anforderung an die Applikation. Die Applikation muss auf folgenden Plattformen lauffähig sein:

- **Android** (Version 4.2.2 oder neuer)
- **Windows Phone** (Version 7.8 oder neuer)
- **iOS** (Version 8 oder neuer)

Bei der Abdeckung dieser Plattformen soll der plattformspezifische Entwicklungsaufwand minimiert werden.

Benutzerschnittstelle Die Benutzerschnittstelle soll eine schnelle, effiziente Bedienung der App ermöglichen. Um dies zu gewährleisten, sollen Benutzer die folgenden Use Cases in den vorgegeben Zeiten erlernen und durchführen:

Use Case	Manager-Zeit	Mitarbeiter-Zeit
UC 01: App öffnen	<10s	<10s
UC 02: Anmelden	<60s	<60s
UC 03: Abmelden	<15s	<15s
UC 04: Workspace wechseln	<15s	<15s
UC 05: Aufgabenfortschritt kontrollieren	<180s	<180s
UC 06: Neue Mitteilung erfassen	<60s	<60s
UC 07: Mitteilungen ansehen	<120s	<120s

Tabelle 2.1: Erlernbarkeit von Use Cases

UC 08: Push-Benachrichtigung erhalten wurde in obenstehender Liste nicht behandelt, da Push-Benachrichtigungen je nach Mobile-OS über externe Server verschickt werden und demzufolge zeitlich **nicht beeinflussbar** sind.

¹http://en.wikipedia.org/wiki/ISO/IEC_9126, zuletzt aufgerufen am 11.06.14, 14:18 Uhr

3 Aufgabenstellung

Die Aufgabe der Bachelorarbeit ist die Erstellung der folgenden Liefergegenstände:

- **Technologie-Empfehlung** Es wird evaluiert, welche Cross-Plattform Technologien auf dem Markt für den vorliegenden Anwendungsfall von QuiXilver Mobile in Frage kommen. Daraus wird eine konkrete Technologie-Empfehlung abgeleitet und begründet.
- **QuiXilver Mobile Prototyp** Für die Legitimation der Technologie-Empfehlung wird ein Prototyp des QuiXilver Mobile Client implementiert, welcher die architekturelevanten Use Cases aus der Anforderungsanalyse umsetzt.
- **Tutorium für Entwickler** Für den leichten Einstieg der zukünftigen App-Entwickler von QuiXilver Mobile wird ein Tutorium erstellt, welches das Installieren der App auf mobilen Geräten beschreibt und eine Architekturübersicht über den implementierten Prototyp gibt.

4 Technologie Evaluation Cross Plattform Entwicklung

Dieses Kapitel beschreibt die Evaluation von geeigneten Technologien zur Implementierung von Cross-Plattform Mobile Apps.

4.1 Beschreibung des Evaluationsprozesses

Bei der Technologie-Evaluation geht das Projektteam in den folgenden Schritten vor:

1. **Definition der Miniprototyp-Anforderungen** Zu Beginn werden die Anforderungen an einen Miniprototypen definiert. Dieser Miniprototyp soll im Rahmen der Evaluation mit allen Technologien implementiert werden.
2. **Definition der Testkriterien** Es wird definiert, nach welchen Kriterien der Implementierungsprozess und das Endresultat des Miniprototypen beurteilt werden.
3. **Technologie-Recherche** In diesem Schritt wird recherchiert, welche Technologien und Konzepte für Cross-Plattform-Entwicklung aktuell auf dem Markt angeboten werden.
4. **Durchführung der definierten Tests** In diesem Schritt wird mit jeder Technologie ein Miniprototyp entwickelt. Diese Miniprototypen werden im Anschluss auf einem einzigen Gerät (Android) diversen Tests unterzogen. Die Tests zeigen, wie performant die verschiedenen Technologien sind.
5. **Technologie-Entscheidung** Zum Schluss wird aus den Resultaten vom vierten Schritt eine Technologie-Entscheidung für den konkreten Anwendungsfall des QuiXilver Mobile Client getroffen und begründet.

Die Struktur der nachfolgenden Kapitel entspricht dem obigen Vorgehen.

4.2 Definition der Miniprototyp-Anforderungen

Das Projektteam entwickelt mit jeder infrage kommenden Technologie denselben Miniprototypen. Diese Miniprototypen werden im Anschluss getestet. Das Projektteam verfolgt bei diesem Vorgehen das Ziel, die Unterschiede (in verschiedenen Disziplinen) der Technologien aufzeigen zu können. Dieser Abschnitt definiert, aus welchen Elementen die Miniprototypen bestehen werden.

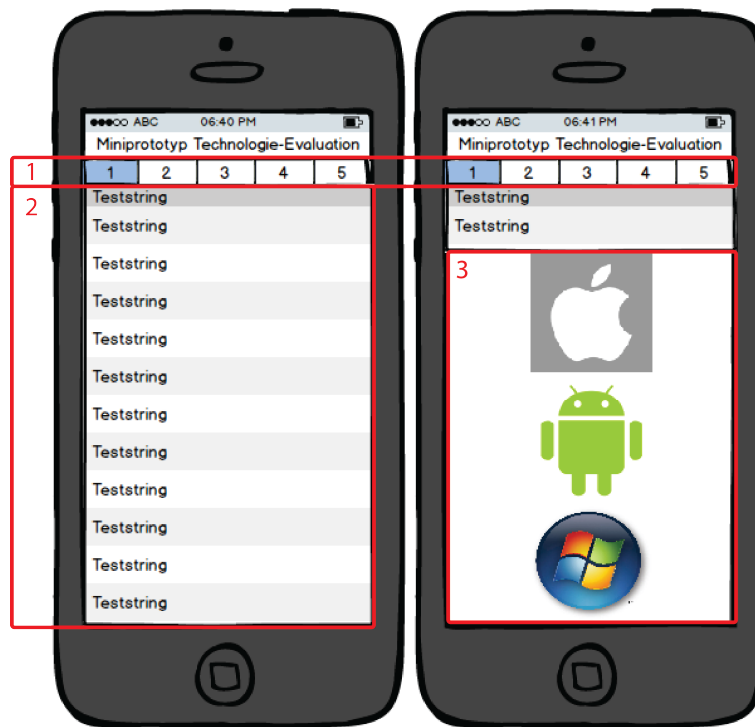


Abbildung 4.1: Benutzeroberfläche Miniprototyp

Obenstehende Grafik skizziert die Benutzeroberfläche der Miniprototypen. Sie bestehen je aus 5 identischen Seiten. Die einzelnen Seiten sind mit einer Tab-Navigation (Bildmarkierung 1) erreichbar. Das Projektteam hat sich für eine Tab-Navigation entschieden, da sie mit hoher Wahrscheinlichkeit auch im späteren QuiXilver Mobile Client zum Einsatz kommen wird. Diese These wird auf Gespräche mit dem Industriepartner und Erfahrungen mit der existierenden QuiXilver-Oberfläche gestützt.

Alle 5 Seiten sind identisch. Sie bestehen aus dem folgenden Inhalt:

- **Eine Tabelle** (Bildmarkierung 2), die aus 50 identischen Einträgen besteht. Das Projektteam hat sich für diese Tabellengröße entschieden, da sie nicht auf einer normalen Displaygröße Platz findet und somit Scrollingverhalten der unterschiedlichen Technologien getestet werden kann. Aus Erfahrungswerten mit dem bestehenden QuiXilver-Produkt geht das Projektteam nicht davon aus, dass Tabellen im späteren im QuiXilver Mobile Client markant mehr Einträge haben werden. Die Tabelleneinträge bestehen alle aus einem String mit dem Inhalt *Teststring*.

- **Drei Bilder** (Bildmarkierung 3), die am Ende jeder Seite nach der Tabelle angezeigt werden. Sie sollen zeigen, wie performant sich das Scrolling verhält, wenn auch Bilder dargestellt werden müssen. Die Details der Bilder (Dateityp und Grösse) sind nicht relevant. Wichtig ist lediglich, dass bei allen Miniprototypen die gleichen Bilddateien verwendet werden.

Zu den Inhalten der Tabelle und den Bildern ist zu sagen, dass alle für die Darstellung des Miniprototypen notwendigen Dateien bereits in der Installation der App enthalten sind. Das Projektteam hat sich für diesen Weg entschieden, um die äusserlichen Einflüsse auf die Performance möglichst tief zu halten.

Deployment Nachdem der Miniprototyp in jeder zu evaluierenden Technologie entwickelt wurde, wird er für Android exportiert. Als Grundlage für die Tests (siehe nachfolgende Abschnitte) werden alle Miniprototypen auf einem einzigen Android-Gerät installiert. Das Projektteam hat entschieden, aufgrund von exponentiellem Aufwand nicht alle Tests mit allen Miniprototypen auf jeder Plattformen durchzuführen. Die Performance-Tests werden auf einem Android-Gerät durchgeführt.

4.3 Definition der Testkriterien

Es folgt eine Auflistung aller definierten Testkriterien. Im Anschluss werden alle Kriterien und ihre Beurteilung beschrieben.

Kategorie	Kriterium
Implementationsprozess	Portabilität Security Community Erlernbarkeit und Dokumentation Lizenzbedingungen Maintainability Entwicklungsumgebung Installationsdauer
Produktqualität	Speichereffizienz Fluidität der Benutzeroberfläche Performance

Tabelle 4.1: Kriterientabelle für Technologie-Evaluation

Obenstehende Tabelle zeigt die definierten Kriterien. Für alle Kriterien gelten die folgenden Grundsätze:

Bewertung Im Rahmen der Evaluation werden die Technologien anhand obiger Kriterien allgemein bewertet. Die Vorgehensweisen zur Bewertungen der einzelnen Kriterien sind in den nachfolgenden Abschnitten festgehalten. Grundsätzlich wird zwischen *qualitativer* und *quantitativer* Bewertung unterschieden. Die qualitative Bewertung beruht auf subjektiven Beobachtungen des Projektteams. Sie werden für Kriterien ausgewählt, die zwar wichtig für die Gesamtbeurteilung der Technologien

4.3 Definition der Testkriterien

sind, jedoch nicht eindeutig gemessen werden können. Die quantitativen Kriterien sind messbare Kriterien. So werden beispielsweise Startup-Zeiten (Kriterium *Performance*) der Miniprototypen zeitlich gemessen.

Gewichtung Für die Entscheidung, welche Technologie im QuiXilver Mobile Client eingesetzt werden soll, werden die gesammelten Bewertungen für den konkreten Anwendungsfall gewichtet. Aus der resultierenden, für QuiXilver gewichteten Gesamtbeurteilung wird die Technologie-Entscheidung abgeleitet.

4.3.1 Portabilität

Bestimmt den erforderlichen Grad an Betriebssystem-spezifischen Anpassungen, um die Software auf allen gewünschten Plattformen lauffähig zu machen. Die folgenden Punkte sollten mit möglichst wenigen Anpassungen möglich sein:

- Kompatibilität mit den folgenden Plattformen:
 - iOS (Version 7 oder neuer)
 - Android (Version 4.2.2 oder neuer)
 - Windows Phone (Version 8 oder neuer)
- Zugriff auf die folgenden Betriebssystem-Funktionalitäten:
 - Push-Benachrichtigungen** ist gemäss Kundenwunsch die wichtigste Betriebssystem-Funktionalität.
 - GPS Lokalisierung** wird für Bewertung miteinbezogen. Könnte für QuiXilver Mobile Client von Interesse sein, wird jedoch nicht im ersten Prototyp erwartet.
 - Volumenregler** wird für Bewertung miteinbezogen. Könnte für QuiXilver Mobile Client von Interesse sein, wird jedoch nicht im ersten Prototyp erwartet.
 - 3-Achsen-Gyrosensor** wird für Bewertung miteinbezogen. Könnte für QuiXilver Mobile Client von Interesse sein, wird jedoch nicht im ersten Prototyp erwartet.
 - Gespeicherte Kontakte** wird für Bewertung miteinbezogen. Könnte für QuiXilver Mobile Client von Interesse sein, wird jedoch nicht im ersten Prototyp erwartet.
- Darstellung der grafischen Benutzeroberfläche in nativem Design
- Aufwand für Plattformwechsel (Vendor Lock-in)

Bewertung Die obenstehenden Zugriffe auf Betriebssystem-Funktionalitäten werden mit einer Checkliste bewertet. Die Erfahrungen mit der Darstellung in nativem Design und der Aufwand für Plattformwechsel wird in einer qualitativen Pro- und Kontra-Liste beurteilt.

4.3.2 Security

Dieses Kriterium überprüft, ob die von Vigience geforderten Security-Standards von den evaluierten Technologien unterstützt werden. Hierfür wird folgendes überprüft:

- Wird HTTPS unterstützt?

Bewertung Ja/Nein

4.3.3 Community

Wichtiges Kriterium einer Technologie ist die Anzahl ihrer Anwender. Mit zunehmender Grösse des anwendenden Publikums verringern sich tendenziell die folgenden Risiken:

- Verfügbare Updates
- Hilfestellungen in Foren
- Verfügbarkeit von technologiespezifischen Mitarbeitern (aus Firmensicht)

Bewertung Die Hilfestellungen in Foren werden qualitativ nach Erfahrungen des Projektteams beurteilt.

4.3.4 Erlernbarkeit und Dokumentation

Dieses Kriterium bewertet, wie gut sich ein Software-Entwickler in die entsprechende Technologie einarbeiten kann. Die Erlernbarkeit und Dokumentation wird folgendermassen bewertet:

- **Erlernbarkeit:** Funktionsweise, Konzepte und Abläufe der Technologie sind einfach verständlich.
- **Dokumentation:** Verständliche, Kernprobleme abdeckende und aktuell gehaltene Nachschlagewerke sind vorhanden.

Für die Einschätzung wird vorausgesetzt, dass der Entwickler in groben Zügen bereits die folgenden Technologien beherrscht:

- JavaScript, HTML und CSS
- Java

Erklärungen zu diesen Technologien werden nicht im Rahmen der Dokumentation erwartet.

Bewertung Dieses Kriterium wird qualitativ in einer Pro- und Kontra-Liste bewertet.

4.3.5 Lizenzbedingungen

Für jede Technologie werden die folgenden Lizenzeigenschaften erfasst:

- Welcher Lizenz untersteht die Benutzung der Technologie?
- Wie hoch sind die Entwicklungsgebühren?

Bewertung Die gewonnen Erkenntnisse werden zusammengefasst.

4.3.6 Wartbarkeit

Bei diesem Kriterium werden die folgenden Fragen beantwortet:

- Qualität von erhaltenen Fehlermeldungen
- Debugging Möglichkeiten

Bewertung Die Bewertung erfolgt qualitativ in der Beantwortung der oben genannten Fragen.

4.3.7 Entwicklungsumgebung

Dieses Kriterium bewertet die Qualität der Technologie-spezifischen Entwicklungsumgebung. Hierbei werden die folgenden Themen bewertet:

- Umfang an Hilfsfunktionen
- Bedienbarkeit

Bewertung Für die Bewertung obestehender Attribute werden die Entwicklungsumgebungen der verschiedenen Technologien qualitativ in einer Pro- und Kontra-Liste bewertet.

4.3.8 Installationsdauer

Dieses Kriterium bewertet, wie schnell die Entwicklungsumgebung mit allen benötigten Zusatzprogrammen installiert werden kann. Hierzu werden konkret die folgenden Werte ermittelt:

- Installationsdauer der entsprechenden Entwicklungsumgebung
- Installationsdauer aller weiteren benötigten Zusatzprogrammen für die Entwicklung auf den geforderten Plattformen
- Dauer zur Erstellung einer lauffähigen HelloWorld-Applikation.

Bewertung Es wird gemessen, wie lange es dauert, die erforderlichen Komponenten lauffähig zu installieren.

4.3.9 Effizienz

Bei diesem Test wird analysiert, wie effizient die verschiedenen Technologien den Speicher verwalten. Auch für diesen Test werden die entwickelten Miniprototypen verwendet. Es werden die folgenden zwei Verhalten verglichen:

- Dateigrösse der installierbaren Android-App.
- Speicherplatzbelegung (in MB) im Random Access Memory (RAM) nach dem Starten des Miniprototypen (spezifisch auf Android, da die App wie beschrieben nur auf Android installiert wird).

Bewertung Quantitativ: Die gemessenen Werte werden tabellarisch festgehalten.

4.3.10 Fluidität der Benutzerschnittstelle

Die Fluidität gibt an, wie performant Animationen und Seiteninhalte dargestellt werden. Eine hohe Fluidität wird dann erzielt, wenn der Benutzer mit blossen Auge kein Ruckeln oder Stocken bei der Darstellung von Animationen erkennen kann. Mit jedem Miniprototypen wird getestet, ob die folgenden Punkte zutreffen:

- Lange Listen (50 Einträge) werden beim Scrollen vom Seitenanfang zum Ende komplett dargestellt, ohne dass zwischenzeitlich nachgeladen werden muss.
- Animationen beim Wechseln eines Tabs werden angezeigt, ohne dass der Benutzer ein Ruckeln oder Stocken feststellen kann. Animationen werden hierbei nicht zwingend für den Miniprototypen vorgeschrieben. Sie werden jedoch bewertet, sofern sie von der Technologie angeboten werden.

Bewertung Die Fluidität wird qualitativ durch das Projektteam beurteilt.

4.3.11 Performance

Die erstellten Miniprototypen werden Performance-Tests unterzogen. Dabei kommen die Miniprototypen jeder Technologie zum Einsatz. Für die Tests werden diese auf einem einzigen Android-Gerät installiert. Das Android Gerät soll die Performance-Unterschiede exemplarisch darstellen. Die folgenden Zeiten werden gemessen:

- **Startzeit des Miniprototypen** Vom Android Homescreen aus wird die App gestartet. Es wird die Zeit gemessen, die zwischen Klicken des Benutzers und kompletter Darstellung des ersten Tabs verstreicht. Zur Messung wird das bei Android mitgelieferte Monitoring System DDMS (Dalvik Debug Monitor Server)¹ verwendet. Es kann mit einem Eclipse-Plugin angesteuert werden. Die damit einsehbaren Android Log-Meldungen beinhalten einen Zeitstempel. Anhand dieser Zeitstempel wird berechnet, wie lange der Boot-Vorgang der einzelnen Miniprototypen dauert.

Bewertung Die gemessenen Zeiten werden tabellarisch aufgeführt.

¹<http://developer.android.com/tools/debugging/ddms.html>, zuletzt aufgerufen am 10. April 2014, 19:25 Uhr

4.4 Technologie-Recherche

Dieser Abschnitt beschreibt den aktuellen Stand der Technik in der App-Entwicklung. Hierfür werden zuerst grundsätzliche, für die Unterscheidung der Technologien notwendige Begriffe definiert. Im Anschluss daran werden die zu evaluierenden Produkte beschrieben.

4.4.1 Entwicklungskonzepte mobile Apps

Aktuell sind zwei verschiedene Entwicklungskonzepte für die Entwicklung von mobilen Apps zu unterscheiden.

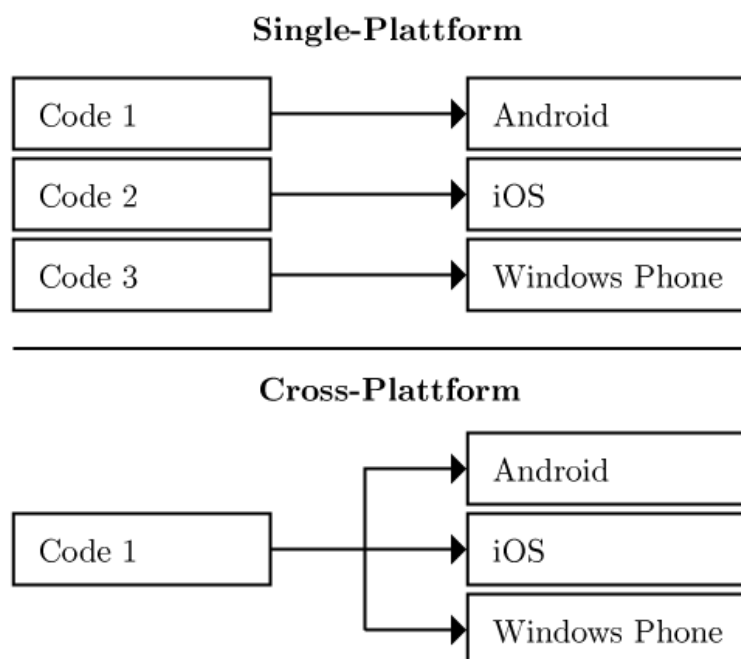


Abbildung 4.2: Unterscheidung Single Plattform und Cross Plattform

Single Plattform Bei diesem Entwicklungskonzept wird die App für jede zu unterstützende Plattform separat in der plattformspezifischen Entwicklungssprache geschrieben.

Cross Plattform Bei Cross-Plattform Entwicklung wird die Idee verfolgt, einen einzigen Code auf mehrere Plattformen zu installieren.

Die App-Resultate von Cross-Plattform-Entwicklungslösungen sind in die folgenden drei Kategorien zu unterteilen:

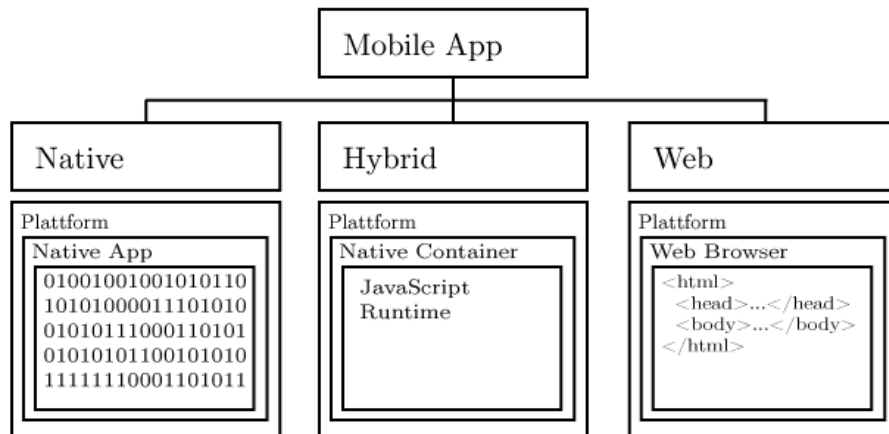


Abbildung 4.3: Entwicklungskonzepte für Mobile Apps

Native Bei nativen, mobilen Apps ist das ganze ausführbare Programm kompiliert auf dem Gerät verfügbar. Native Apps sind der oben definierten Kategorien Single-Plattform sowie Cross-Plattform zuzuweisen.

Hybrid Bei hybrider Entwicklung wird die App in JavaScript entwickelt. Auf dem Gerät werden die JavaScript-Dateien in einem nativen Container ausgeführt. Der native Container ist eine App, welche folgende Komponenten enthält:

- WebView. In einer WebView ist es möglich, Webinhalte darzustellen. Sie wird als JavaScript-Runtime für die App verwendet.
- Betriebssystemfunktionen (z.B. GPS-Standort ermitteln) der WebView zur Verfügung stellen, um der JavaScript-App Zugriffe auf das System zu ermöglichen.

Die JavaScript-Dateien werden nicht kompiliert, sondern zur Laufzeit von der WebView interpretiert. Hybride Entwicklung werden Cross-Plattform realisiert.

Web Bei diesem Entwicklungskonzept werden herkömmliche Webseiten entwickelt, die auf dem mobilen Gerät im Browser ausgeführt werden. Der Hauptunterschied zum hybriden Entwicklungskonzept liegt darin, dass aktuell nur sehr bedingt Betriebssystemfunktionen aufrufbar sind.

Für dieses Projekt ist vor allem die Funktionalität von Push Benachrichtigungen wichtig. Im Anschluss wird detailliert beschrieben, wie gut Push Benachrichtigungen mit diesen drei Entwicklungskonzepten unterstützt werden können.

4.4.2 Konzepte Push-Benachrichtigungen

Die im letzten Unterabschnitt beschriebenen Konzepte zur App-Entwicklung unterscheiden sich nicht nur in der Entwicklung, sondern auch in der Verfügbarkeit von Betriebssystemressourcen. Die wichtigste Betriebssystemressource für den QuiXilver Mobile Client ist der Dienst für Push Benachrichtigungen. Dieser Unterabschnitt beschreibt die verschiedenen Konzepte für Push Benachrichtigungen. Es wird auch beschrieben, welche Möglichkeiten für Push Benachrichtigungen mit den besprochenen Entwicklungs-Konzepten (Nativ, Hybrid und Web) möglich sind.

Push Benachrichtigungen auf Betriebssystem-Ebene

Push Benachrichtigungen im mobilen OS zeichnen sich dadurch aus, dass sie im Benachrichtigungszentrum eines Smartphones dargestellt werden.

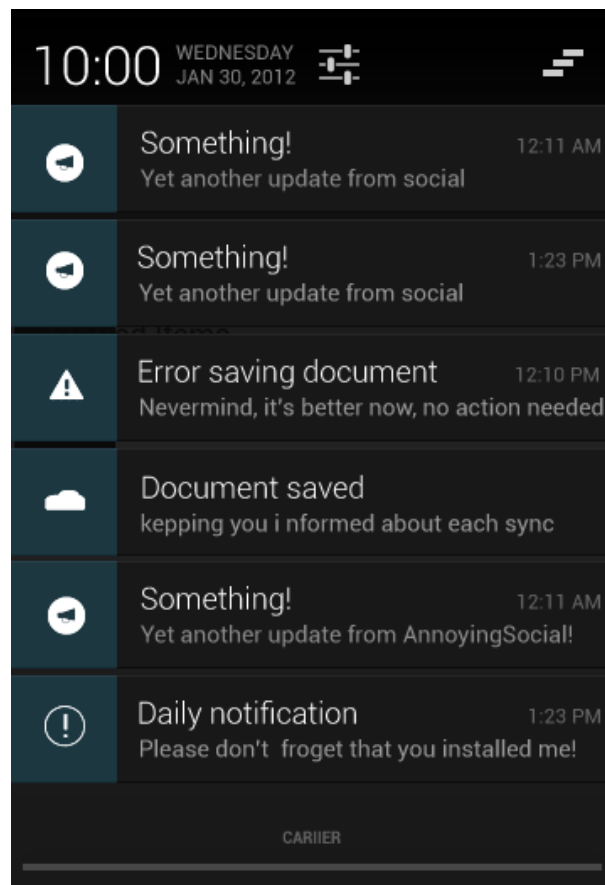


Abbildung 4.4: Push Benachrichtigungen bei einem Android-Gerät ²

Obenstehende Abbildung zeigt, wie Push Benachrichtigungen im Android-Betriebssystem dargestellt werden. Jede Push Benachrichtigung erzeugt einen eigenen Eintrag im Benachrichtigungszentrum.

Dieser Typ von Push Benachrichtigung wird mit untenstehender Grafik realisiert.

²http://developer.android.com/design/media/notifications_pattern_social_fail.png, zuletzt aufgerufen am 12. April 2014, 15:05 Uhr

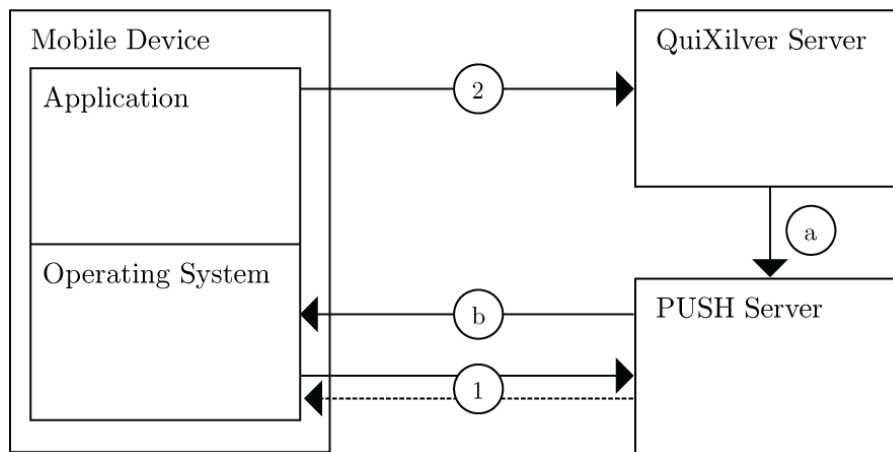


Abbildung 4.5: Paketverkehr: Push Benachrichtigungen auf OS-Ebene

Der Prozess für die Auslieferung einer Push Benachrichtigung geschieht grundsätzlich in zwei Teilen. Er ist bei allen in diesem Projekt geforderten Plattformen (Android, iOS, Windows Phone) vergleichbar umgesetzt.

Registrierung für den Push Dienst:

1. Zuerst muss sich das mobile Gerät am plattform-spezifischen Push-Server anmelden (In obenstehender Grafik mit PUSH Server beschriftet). Diese Registrierung übernimmt das Betriebssystem des Smartphones, wird jedoch von einer installierten Applikation (App) gestartet. Bei erfolgreicher Registrierung erhält das Betriebssystem des Smartphones ein Token. Dies ist ein einzigartiger, für eine gewisse Zeit gültiger String. Die genaue Definition des Strings ist bei jeder Plattform unterschiedlich. Das Betriebssystem leitet diesen Token an die Applikation weiter.
2. Die Applikation leitet diesen Token an einen Server weiter. Beim Projekt des QuiXilver Mobile Clients erhält der QuiXilver Server diesen Token. Er wird ihn in der nächsten Phase benötigen, um eine Push Benachrichtigung abzusenden.

Nach erfolgreicher Durchführung dieser zwei Schritte ist die Registrierung des Smartphones für den Push-Dienst abgeschlossen.

Versenden einer Push Benachrichtigung: Dieser Abschnitt kann nur durchlaufen werden, wenn vorher eine erfolgreich Registrierung für den Push Dienst erfolgt ist. Er startet, sobald auf dem QuiXilver Server ein Event auftritt, der eine Push Benachrichtigung nach sich ziehen soll. Dies ist beispielsweise beim Erhalten einer Nachricht der Fall.

- A Der Server hält die Tokens, die er im Registrierungsprozess erhält, bei sich gespeichert. Soll nun eine Push Benachrichtigung abgesendet werden, sendet er ein Paket an den Plattform-spezifischen Push Server. Jede Plattform (Android, iOS, Windows Phone) besitzt ihre eigene Definition dieses Pakets. Sie

alle haben jedoch gemeinsam, dass das Token und eine Nachricht mitgeliefert werden müssen. Mit diesem Token wird sichergestellt, dass nur der QuiXilver Server Push Benachrichtigungen an das Smartphone versenden kann.

B Erhält der Push Server ein solches Push-Paket, überprüft er, ob das darin enthaltene Token mit einem registrierten Gerät übereinstimmt. Trifft dies zu, wird eine Push Benachrichtigung an das Betriebssystem des Smartphones weitergeleitet. Das Betriebssystem des Smartphones ist dann dafür zuständig, die Push Benachrichtigung im Benachrichtigungszentrum darzustellen, wie das in obiger Abbildung exemplarisch für Android dargestellt wurde.

Umsetzbarkeit Diese Art von Push Benachrichtigungen kann auf den geforderten Plattformen zurzeit nur von Apps umgesetzt werden, die nativ oder hybrid entwickelt wurden. Aktuell (Stand 12. April 2014) ist es nicht möglich, aus reinen Web-Apps, die im mobilen Browser ausgeführt werden, für mobile Geräten auf diese Weise Push Benachrichtigungen zu versenden. Erste Ansätze für eine solche Lösung bietet jedoch die am 22. Oktober 2013 erschienene Safari-Version 7. Welche Features dort genau hinzugekommen sind, wird im Unterabschnitt Push Benachrichtigungen durch Browser auf Betriebssystem-Ebene beschrieben.

Push Benachrichtigungen auf Browser-Ebene

Grundsätzlich ist mit Push Benachrichtigungen auf Browser-Ebene gemeint, dass der Webserver dem Client jederzeit Nachrichten senden kann. Untenstehende Grafik stellt die Architektur dieser Lösung dar.

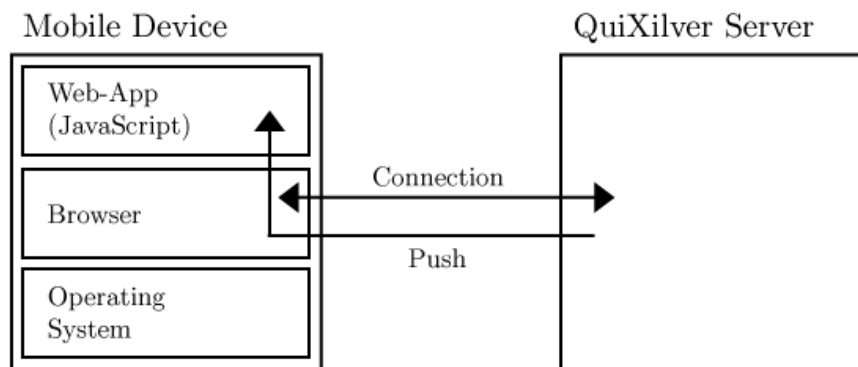


Abbildung 4.6: Schema für Push Benachrichtigungen auf Browser-Ebene

Die Abbildung illustriert, wie diese Neuerungen mitgeteilt werden. Zwischen Client und Server wird eine bidirektionale Verbindung aufgebaut und aktiv gehalten. Die Verbindung wird erst beendet, sobald die Webseite im Browser des Clients geschlossen wird. Der Server hat mit dieser Verbindung jederzeit die Möglichkeit, dem Client Nachrichten zu schicken. Dies bietet dem Entwickler diverse Möglichkeiten. Häufig wird sie für die folgenden Zwecke verwendet:

- **Aktualisierung des Seiteninhalts** Der bestehende Seiteninhalt wird automatisch verändert, wenn es auf dem Server eine Neuerung gegeben hat. Dies

wird beispielsweise bei News-Webseiten verwendet, wo ein sogenannter Live-Ticker automatisch über aktuelle Ereignisse informieren soll. Ein weiteres Beispiel hierfür ist Facebook. Neue Seiteninhalte werden automatisch nachgeladen, ohne dass der Benutzer die Seite neu öffnen muss.

- **Weiterleitung an Browser-Benachrichtigung** Die Neuerungen können an den Browser-internen Benachrichtigungsdienst weitergeleitet werden. Diese Benachrichtigungen werden angezeigt, selbst wenn der Benutzer gerade einen anderen Tab geöffnet hat oder der Browser minimiert ist. Aktuell (Stand 12. April 2014) sind diese Browser Benachrichtigungs-Dienste jedoch nicht auf mobilen Geräten erhältlich.

Umsetzbarkeit Grundsätzlich ist diese Architektur in Web-Apps umsetzbar. Hierfür kann beispielsweise mit **HTML5 Websockets** gearbeitet werden. Mit dem HTML5-Standard ist die JavaScript-Klasse WebSockets verfügbar, die eine solche Verbindung anbietet. Diese Technologie wird von den folgenden Plattformen unterstützt (unvollständige Liste) ³:

- iOS Safari
- Android Browser
- Chrome for Android
- Firefox for Android
- IE Mobile

³<http://caniuse.com/websockets>, zuletzt aufgerufen am 12. April 2014, 18:04 Uhr

Push Benachrichtigungen durch Browser auf Betriebssystem-Ebene

Zum Zeitpunkt dieser Bachelorarbeit sind neue Technologien auf dem Markt erschienen, die eine Kombination der oben beschriebenen Push Benachrichtigungs-Dienste ermöglichen. Beispiel hierfür sei Apple. Mit Safari Version 7, die am 22. Oktober 2013 erschienen ist ⁴, und dem gleichzeitig neu erschienenen Mac OS X Mavericks ist es möglich, über eine Webseite Push Benachrichtigungen zu versenden, die im offiziellen Benachrichtigungszentrum des Clients aufgelistet werden. Grösste Neuerung hierbei ist es, dass diese Benachrichtigungen selbst dann versendet und dargestellt werden, wenn Safari bereits wieder beendet wurde ⁵. Untenstehende Abbildung ⁶ illustriert die hierfür notwendige Architektur:

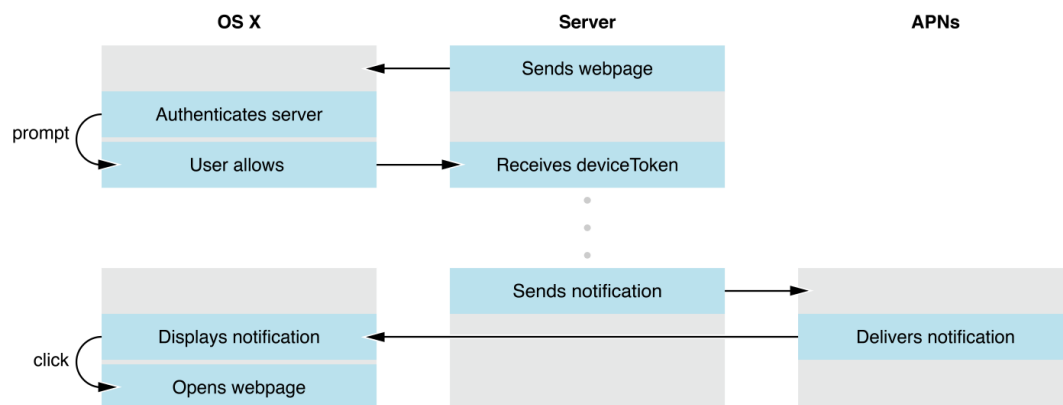


Abbildung 4.7: Schema für Push Benachrichtigungen durch Browser auf Betriebssystem-Ebene

Obenstehende Abbildung illustriert den Aufbau dieses Push-Dienstes. Er besteht aus den folgenden Schritten:

- **Sends Webpage** Der Webserver sendet dem Client die angeforderte Webseite. Sie beinhaltet das Angebot für eine Push-Dienst Anmeldung.
- **Authenticates server** Der Benutzer wird gefragt, ob er den Push Dienst dieser Webseite abonnieren will.
- **User allows** Der Benutzer willigt ein.
- **Receives deviceToken** Der Webserver erhält ein deviceToken. Dieses Token wird, analog zum Push-Konzept Push Benachrichtigungen auf Betriebssystem-Ebene, später für das Versenden einer Benachrichtigung verwendet.

⁴[http://de.wikipedia.org/wiki/Safari_\(Browser\)](http://de.wikipedia.org/wiki/Safari_(Browser)), zuletzt aufgerufen am 12. April 2014, 18:30 Uhr

⁵<https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/NotificationProgrammingGuideForWebsites/Introduction/Introduction.html>, zuletzt aufgerufen am 12. April 2014, 18:33 Uhr

⁶https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/NotificationProgrammingGuideForWebsites/Art/safari_notifs_rev_2x.png, zuletzt aufgerufen am 13. April 2014, 14:24 Uhr

- **Sends Notification** Zu einem späteren Zeitpunkt, wenn eine Neuerung auf der Webseite stattgefunden hat, die dem Benutzer mitgeteilt werden soll, sendet der Webserver eine Anfrage an den APNs (Apple Push Notification Server). Diese Server sind dafür zuständig, Push Benachrichtigungen an das Endgerät zu versenden.
- **Delivers notification** Der APNs versendet die gewünschte Push Benachrichtigung an das Endgerät.
- **Displays notification** Die Benachrichtigung wird im Benachrichtigungszentrum des Betriebssystems dargestellt. Hierbei spielt es keine Rolle mehr, ob Safari noch geöffnet ist.
- **Opens webpage** Klickt der Benutzer auf die dargestellte Benachrichtigung, gelangt er direkt auf die Webseite, von welcher eine Push Benachrichtigung erhalten wurde.

Umsetzbarkeit Leider ist dieser Dienst zum Zeitpunkt der Bachelorarbeit lediglich auf Macs (nicht auf iPhones) erhältlich ⁷. Aus diesem Grund kommt diese Push-Technologie nicht weiter für den QuiXilver Mobile Client in Frage. Sollen Push Benachrichtigungen direkt im Benachrichtigungszentrum des Betriebssystems dargestellt werden, so kann dies weiterhin nur mit einer nativ oder hybrid entwickelten mobile-App erreicht werden.

4.4.3 Konkurrenzanalyse

In diesem Kapitel wird untersucht, auf welche Techniken (Native, Hybrid oder Web) die Konkurrenz setzt. Der Industriepartner gab dem Projektteam an, welche Konkurrenten analysiert werden sollen.

Facebook ⁸ Die App für den mobilen Zugriff auf das soziale Netzwerk Facebook ist zum Zeitpunkt der Entwicklung dieser Bachelorarbeit nativ entwickelt. Lange Zeit verfolgte Facebook jedoch die Strategie der Plattformunabhängigkeit. So wurde die Facebook-App ursprünglich mit HTML5 entwickelt. Damals hat man sich aus den folgenden Gründen für diese Technologie entschieden:

- **Plattformunabhängigkeit** Mit dem Einsatz von HTML5 war Facebook auf jeder Plattform erhältlich
- **Wiederverwendbarkeit des Codes** Grosse Teile des Codes konnten in den Versionen für Android, iOS und mobile Web mehrfach eingesetzt werden.

Nach einiger Zeit wurde die iOS-Version jedoch von Grund auf neu entwickelt. Dabei wurde neu der native Ansatz verfolgt. Die folgenden Gründe trugen wesentlich zu dieser Entscheidung bei:

⁷<http://mac.appstorm.net/general/opinion/the-good-the-bad-and-the-terrible-of-safari-push-notifications/>, zuletzt aufgerufen am 12. April 2014, 18:38 Uhr

⁸<https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920> zuletzt abgerufen am 10. April 2014, 10:58 Uhr

- **Geschwindigkeit** Die nativ implementierte Facebook-Version wies eine höhere Performanz auf. Konkret sei das Scrollen durch eine lange Liste mit Facebook-News performanter geworden.
- **Offline-Caching** Die native Implementation bot mehr Möglichkeiten zu Offline-Caching (Speichern von Daten auf dem Gerät für späteren Offline-Zugriff). Wird heute die native App gestartet, werden sofort alte Facebook News angezeigt. Im Hintergrund werden jedoch bereits die aktuellen Einträge geladen.

Die nativ implementierte iOS-App ist seit dem **23. August 2012** verfügbar⁹. Wenige Monate später, am **13. Dezember 2012**, veröffentlichte Facebook auch eine neue Version der Android-App. Sie wurde ebenfalls nativ entwickelt. Die Begründung für diese Entscheidung blieb dieselbe wie für die native iOS-App.¹⁰

Trello Trello ist eine kostenlos benutzbare, webbasierte Projektmanagement-Software der Firma Fog Creek Software¹¹. Die mobile App ist für die folgenden Plattformen verfügbar¹²:

- iOS (Version 7.0 oder neuer)
- Android (Version 4.0 oder neuer)
- Kindle Fire HD (2. Generation oder neuer)

Gemäss den Aussagen eines angeblichen Trello-Entwicklers¹³ wurde die Trello iOS-App mit Ausnahme des Attachment Viewers (Darstellung von Anhängen) komplett nativ implementiert. Der Attachment Viewer sei mit einer reinen WebView (Darstellung von Webseiten in nativen Apps) umgesetzt worden. Für die Kommunikation mit dem Server werde RestKit (ein iOS Objective-C Framework für die Kommunikation mit RESTful Web Services¹⁴) verwendet. Angaben zur Implementation der Versionen für Android und Kindle Fire sind nicht öffentlich verfügbar.

Podio Polio ist ein cloudbasierter Collaboration-Service. Er wurde 2009 gegründet und 2012 von Citrix übernommen¹⁵. Die mobile App für Polio ist auf den folgenden Plattformen erhältlich:

- Android
- iOS

Das Implementationskonzept von Podio ist nicht öffentlich verfügbar.

⁹<http://www.theverge.com/2012/8/23/3262782/facebook-for-ios-native-app> zuletzt aufgerufen am 10. April 2014, 13:00 Uhr

¹⁰<http://www.theverge.com/2012/12/13/3763196/facebook-for-android-native-app>, zuletzt aufgerufen am 10. April 2014, 13:05 Uhr

¹¹<http://en.wikipedia.org/wiki/Trello>, zuletzt aufgerufen am 10. April 2014, 14:00 Uhr

¹²<http://help.trello.com/customer/portal/articles/940690-what-browsers-and-mobile-platforms-does-trello-support->, zuletzt aufgerufen am 10. April 2014, 14:16 Uhr

¹³<http://stackoverflow.com/questions/10700460/how-is-trello-iphone-app-developed>, zuletzt aufgerufen am 10. April 2014, 13:51 Uhr

¹⁴<http://www.restkit.org>, zuletzt aufgerufen am 10. April 2014, 13:56 Uhr

¹⁵<http://en.wikipedia.org/wiki/Podio>, zuletzt aufgerufen am 10. April 2014, 15:59 Uhr

4.4.4 Produktübersicht

Dieser Unterabschnitt enthält eine Liste an Beispielen von aktuell erhältlichen Technologien für die App-Entwicklung.

Name	Sprache	Konzept	Resultat
Google, Android	Java	Single Platform	Native
Apple, iOS	Objective-C	Single Platform	Native
Microsoft, Windows Phone	C#	Single Platform	Native
Xamarin Inc., Xamarin	C#	Cross Platform	Native
Adobe, PhoneGap	JavaScript, HTML, CSS	Cross Platform	Hybrid
AppGyver, Steroids	JavaScript, HTML, CSS	Cross Platform	Hybrid
Appcelerator Titanium	JavaScript, HTML, CSS	Cross Platform	Hybrid
RemObjects	C#	Cross Platform	Native
jQuery Foundation, jQuery Mobile	JavaScript, HTML, CSS	Cross Platform	Web
SAP, OpenUI5	JavaScript, HTML, CSS	Cross Platform	Web
Sencha, Sencha Touch	JavaScript, HTML, CSS	Cross Platform	Web

Tabelle 4.2: Verbreitete Produkte

Die in obenstehender Tabelle fett markierten Technologien wurden zur Evaluation ausgewählt. Sie werden in den nachfolgenden Abschnitten eingeführt. Dabei wird auch beschrieben, weshalb das Projektteam gerade diese Technologien für die Evaluation ausgewählt hat.

4.4.5 PhoneGap

Anbieter Ursprünglich wurde Phonegap durch die Firma Nitobi Software entwickelt. Im Jahr 2011 übernahm der Softwaregigant Adobe Systems Inc das Unternehmen und gleichzeitig ihre Produktpalette. Gleichzeitig mit der Übernahme hat Adobe die Softwarebasis von PhoneGap an die Apache Foundation übergeben, um das Open Source Projekt namens Apache Cordova zu gründen ¹⁶. Das Projekt Phonegap wird von Adobe weiterentwickelt und Apache Cordova von der Apache Foundation. Beide Produkte profitieren gegenseitig voneinander und unterstützen sich in der Weiterentwicklung. Der Vorteil Phonegaps zeigt sich darin, dass durch den Softwaregiganten und die Open Source Community eine grosse, stabile Benutzerbasis existiert. Die Technologie kann als bewährte Technologie eingestuft werden und bringt die Sicherheit mit sich, dass sie in absehbarer Zukunft unterhalten und weiterentwickelt wird.

Grund für Evaluation PhoneGap wurde für die Evaluation ausgewählt, weil es gemäss der Marktanalyse *How can HTML5 compete with Native?* der Firma Vision Mobile ¹⁷ den Markt für Hybride Web-Wrapper-Technologien anführt.

Technologiestack Der Grundsatz von Phonegap liegt darin, dass Entwickler eine Applikation mit den Sprachen JavaScript, HTML und CSS einmal schreiben und anschliessend auf allen gängigen Plattformen portieren können. Aktuell werden die folgenden Plattformen angeboten ¹⁸:

- iOS
- Android
- Blackberry OS
- WebOS
- Windows Phone
- Symbian
- Bada

In der folgenden Grafik wird der Best Practice Stack für die Entwicklung mit Phonegap dargestellt:

Die Basis des Stacks ist der Phonegap Native Wrapper. Er packt die ganze Applikation in eine nativ aussehende Applikation und übernimmt die Kommunikation mit dem Betriebssystem. Dieser Layer muss betriebssystemspezifisch sein und wird durch Phonegap zur Verfügung gestellt. Aufbauend folgt die Phonegap System Library, welche System Level Calls (z.B. Foto aufnehmen, GPS Koordinaten

¹⁶<http://en.wikipedia.org/wiki/PhoneGap> aufgerufen am 6. April, 2014, 9:31 Uhr

¹⁷<http://www.visionmobile.com/product/how-can-html5-compete-with-native/>, zuletzt aufgerufen am 12. April 2014, 18:58 Uhr

¹⁸<http://phonegap.com/about/feature/>, aufgerufen am 6. April 2014, 11:13 Uhr

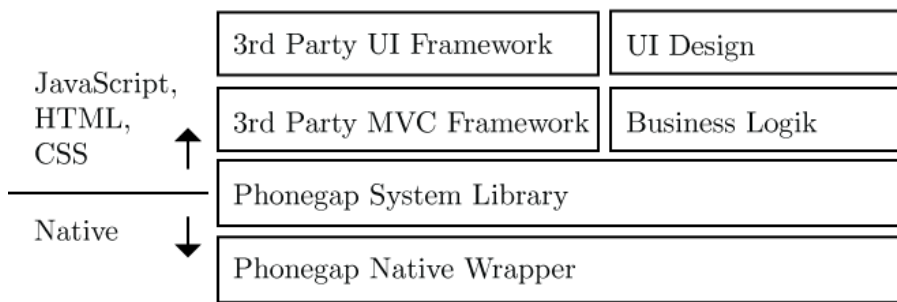


Abbildung 4.8: Stack für Phonegap

ermitteln) den höheren Schichten ermöglicht. Die oberen zwei Schichten bilden die Logik und View der Applikation. Die Entwicklung dieser Schichten erfolgt nur in den Sprachen JavaScript, HTML und CSS. Dadurch kann die Logik und View beliebig portiert werden, sofern die unteren PhoneGap-Layers in anderen Betriebssystemen angeboten werden. Weiter erwähnenswert ist, dass eigene User Interface (UI)- und Architektur-Frameworks eingesetzt werden können.

Besonderheiten Die zentrale Besonderheit der Phonegap-Technologie ist die Portabilität. Im Vergleich zur Konkurrenz fällt auf, dass keine Rücksicht auf betriebs-systemspezifische Eigenschaften genommen werden muss. Hinzu kommt, dass die Applikation in einem normalen Webbrowser vollständig lauffähig ist, sofern keine architekturelevanten System Level Calls verwendet werden.

Weiter erwähnenswert ist die Technologiefreiheit der obersten zwei Layers. Sie kennzeichnet sich dadurch, dass keine Zwänge zur Verwendung von Phonegap-Komponenten bestehen. Bei Konkurrenzprodukten kann es vorkommen, dass bestimmte UI-Frameworks oder Architektur-Frameworks vorgegeben werden. Dadurch, dass der Entwickler kein UI-Framework vorgeschrieben bekommt, kann er vom nativen Look-and-Feel komplett abweichen.

Als zusätzlicher, kostenpflichtiger Dienst bietet Adobe einen Build Service an, um die fertige JavaScript-Applikation auf allen verfügbaren Plattformen zu portieren. Die sonstige Verwendung der Technologie ist für private und kommerzielle Verwendung kostenlos.

Zukunftsauussichten Phonegap ist eine der bewährten Technologien im Bereich Cross-Plattform-Entwicklung. Gestärkt durch die zusätzlichen Argumente, dass sie von einem Softwaregiganten und einer Open Source Bewegung gepflegt und weiterentwickelt wird, sind die Zukunftsauussichten der Technologie als sehr gut einzustufen.

4.4.6 Steroids

Anbieter AppGyver ist ein in Finnland basiertes Technology-Start-Up mit innovativen Lösungen im Bereich der mobilen Applikationsentwicklung. Mit ca. 15 Mitarbeitern¹⁹ ist das Unternehmen eher klein im Vergleich zu den anderen erwähnten Unternehmen dieser Evaluation. Das Unternehmen unterscheidet sich massgeblich von der Konkurrenz durch ihr innovatives Angebot. Besonders herausstechend sind: Vereinfachtes Testing über alle Plattformen (ein Klick für das Deployment auf mehrere Geräte) und native Benutzerschnittstellen.

Grund für Evaluation Der Industriepartner wünschte, dass AppGyver Steroids im Rahmen der Bachelorarbeit evaluiert wird.

Technologiestack Das Produkt Steroids verwendet grundsätzlich Phonegap als Basis und erweitert die Technologie mit weiteren Features. In der folgenden Grafik ist ersichtlich, welche Komponenten von Phonegap übernommen werden und welche neue Komponenten durch Steroids hinzukommen:

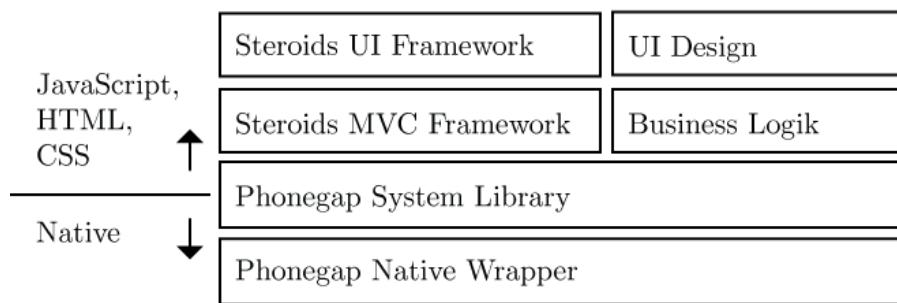


Abbildung 4.9: Steroids Stack

In der obigen Grafik ist ersichtlich, dass Steroids die unteren zwei Layers von PhoneGap (System Library und Native Wrapper) direkt wiederverwendet.

In den oberen beiden Layers wird bei Steroids die komplette Freiheit der Entwickler beschränkt und das Steroids eigene UI- und Architektur-Framework vorgeschrieben.

Besonderheiten Eine wichtige Besonderheit sind die folgenden nativen User-Interface-Elemente²⁰:

- Multiple WebViews
- Tabs

¹⁹<http://www.appgyver.com/>, zuletzt aufgerufen am 6. April 2014, 13:01 Uhr abgerufen am 6. April, 2014

²⁰http://www.appgyver.com/steroids_game_change, zuletzt aufgerufen am 12. April 2014, 18:48 Uhr

- Modal windows
- Navigation bar
- Native animations
- Drawer

Diese Objekte können bei der Entwicklung von Steroids-Apps verwendet werden. Bei der Erstellung eines Plattform-spezifischen Builds werden die nativen UI-Elemente der entsprechenden Plattformen eingesetzt ²¹.

Eine weitere wichtige Besonderheit von Steroids ist die Entwicklungsumgebung. Der Deploymentprozess und die Testumgebung sind vollautomatisiert. Während der Entwicklung der App wird das Resultat laufend automatisiert auf das Testgerät portiert. Somit können inkrementelle Veränderungen sofort vom Entwickler getestet werden. Weiter ist es möglich, mehrere Geräte an die automatisierte Portierung anzuschliessen. Damit wird das geräteübergreifende Testing sehr praktisch und effizient gestaltet.

Zukunftsauksichten AppGyer fällt mit der einfachen Implementation der nativen Benutzerschnittstelle und ihren innovativen Produkten positiv auf. Steroids hinterlässt beim Projektteam jedoch auch den Eindruck von Instabilität. So war während sehr langer Zeit keine Android Unterstützung von Android geboten. In dieser Zeit konnten mit Steroids lediglich iOS-Apps erstellt werden ²². Als Grund für diese lange Bearbeitungszeit sieht das Projektteam mitunter auch die kleine Firmengrösse.

²¹http://www.reddit.com/r/webdev/comments/1ksp6l/steroidsjs_bridges_the_performance_gap_between/, zuletzt aufgerufen am 12. April 2014, 18:47 Uhr

²²<https://github.com/AppGyver/scanner/issues/35>, zuletzt aufgerufen am 13. April 2014, 16:37 Uhr

4.4.7 Titanium

Anbieter Appcelerator ist ein Anbieter von mobilen Technologien, gegründet im Jahr 2006 mit Sitz in Mountain View, Kalifornien. Das Unternehmen beschäftigt aktuell ca. 160 Mitarbeiter und entwickelt unter anderem ihr Produkt Titanium²³.

Grund für Evaluation Titanium wird im Rahmen dieser Bachelorarbeit aufgrund seiner hohen Präsenz in den Medien und des eigenen Cross-Plattform-Ansatzes evaluiert.

Technologiestack Der Grundsatz von Titanium ist, dass Entwickler mobile Apps mit JavaScript entwickeln können. Dabei soll das native Look-And-Feel nicht verlorengehen.

In der folgenden Grafik werden die Schichten einer Titanium-App dargestellt:

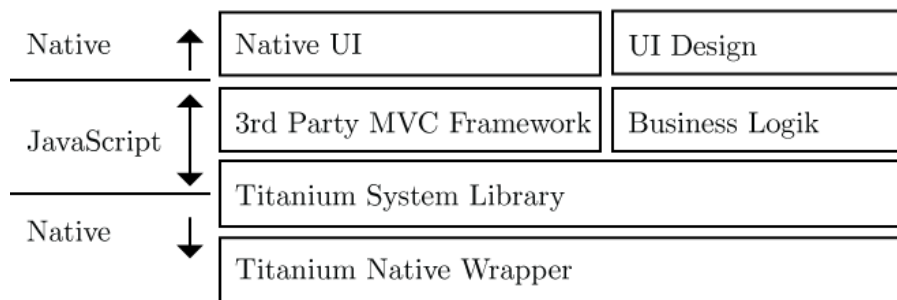


Abbildung 4.10: Titanium Stack

Analog zu PhoneGap bietet Titanium ebenfalls einen Native Wrapper. Weiter bietet die Schicht eine JavaScript Runtime, welche die Ausführung von JavaScript Code ermöglicht. Bei der Wahl des MVC-Frameworks überlässt die Technologie dem Entwickler die Freiheit ein eigenes auszuwählen, jedoch empfiehlt der Anbieter das eigene namens Alloy zu verwenden. Im höchsten Layer wird die Benutzerschnittstelle in JavaScript mit Unterstützung der Titanium System Library geschrieben. Dies führt dazu, dass beim Kompilierungsprozess die UI-Elemente durch nativen Code ersetzt werden.

Besonderheiten Eine wichtige Besonderheit Titaniums sind die nativen User Interface Elemente. Sie werden in JavaScript geschrieben, beim Buildprozess jedoch zu nativen Elemente übersetzt. Diese Eigenschaft von Titanium führt wiederum dazu, dass sich die Portabilität des Codes verringert. Bei einer Migration von Titanium auf eine andere Plattform (z.B PhoneGap) müsste ein Grossteil des Codes neu geschrieben werden. Einzig die JavaScript-Logik kann wiederverwendet werden.

²³<http://en.wikipedia.org/wiki/Appcelerator>, zuletzt aufgerufen am 12. April, 2014, 17:53 Uhr

Die Plattform Windows Phone wird nicht unterstützt. Damit erfüllt Titanium eines der notwendigen Kriterien nicht. Die Technologie wird somit in der Technologieevaluation nicht mehr weiter berücksichtigt .

Zukunftsaussichten Das Projektteam schätzt die Zukunftsaussichten von Appcelerator Titanium als mittelmässig ein, da der ein ausgeprägter Vendor-Lockin besteht.

4.4.8 Xamarin

Firma Xamarin ist eine in San Francisco basierte, im Jahre 2011 gegründete, Softwareherstellerfirma. Die Gründungsmitglieder sind Softwareentwickler, welche massgeblich bei der Entwicklung des Vorgängermodells Mono - eine Cross-Plattform Entwicklungstechnologie für Applikationen - beteiligt waren ²⁴. Das Unternehmen beschäftigt derzeit 112 Mitarbeiter²⁵ und ist somit als Unternehmen mittlerer Grösse einzustufen. Des weiteren geben sie auf der Firmenhomepage bekannt, dass über 450'000 Entwickler mit Xamarin arbeiten²⁶. Der Ruf des Unternehmens wird zusätzlich durch die Publikation gestärkt, offizieller Partner von Microsoft zu sein²⁷.

Grund für Evaluation Xamarin wird in die Evaluation miteinbezogen, weil es zusammen mit PhoneGap zu den Marktführern im Bereich Cross-Plattform-Entwicklung gehört. Dies geht aus einer Marktanalyse der Firma VisionMobile hervor ²⁸. Hierbei ist besonders interessant, dass Xamarin einen grundlegend unterschiedlichen Entwicklungsansatz als PhoneGap verfolgt. Die genauen Unterschiede werden in den nachfolgenden Absätzen aufgezeigt.

Technologiestack Xamarin beruht auf dem oben beschriebenen Cross-Plattform Compilation Grundsatzes. Native iOS, Android oder Windows Phone-Programme können in C# geschrieben werden. Die Übersetzung von C# zu den Plattformspezifischen Apps erfolgt individuell pro Plattform. Für iOS wird der Source-Code ahead-of-time kompiliert zur ARM Assembly Language. Für Android wird der Source-Code in die Intermedia Language übersetzt und MonoVM und JIT'ing verpackt. Die Applikation wird parallel zu Java/Dalvik ausgeführt. Für Windows Phone wird der Source-Code in Intermedia Language übersetzt und mit der integrierten Runtime ausgeführt²⁹. Zu beachten ist, dass die Benutzerschnittstelle in der jeweiligen Native Markup Language geschrieben werden muss.

²⁴<http://en.wikipedia.org/wiki/Xamarin>, abgerufen am 6. April, 2014, 14:05 Uhr

²⁵<http://xamarin.com/about> abgerufen am 6. April, 2014, 14:30 Uhr

²⁶<http://xamarin.com/> abgerufen am 6. April, 2014, 15:05 Uhr

²⁷http://www.microsoft.com/germany/visualstudio/aktuell/news/show.aspx?id=msdn_de_51428 abgerufen am 6. April, 2014, 15:20 Uhr

²⁸<http://www.visionmobile.com/product/developer-economics-q1-2014-state-developer-nation/>, zuletzt aufgerufen am 13. April 2014, 16:52 Uhr

²⁹http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/, abgerufen am 7. Juni, 2014, 15:10 Uhr

4.4 Technologie-Recherche

In folgender Grafik wird die Vorgehensweise bei der Entwicklung mit Xamarin illustriert:

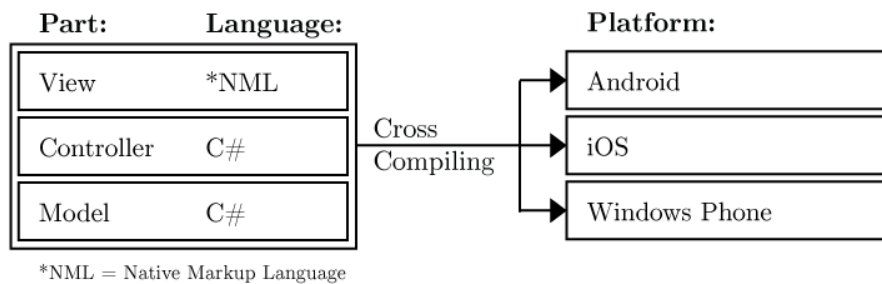


Abbildung 4.11: Xamarin Stack

Gemäss vorgehender Grafik ist ersichtlich, dass C# als Programmiersprache für alle Plattformen angewendet werden kann. Die View wird üblicherweise in den Plattform-spezifischen Designern für Benutzerschnittstellen erstellt und in das Xamarin-Projekt kopiert. Sobald alle notwendigen Komponenten der Software erstellt sind, kann Xamarin die mobile Software kompilieren. Der Kompilierungsprozess übersetzt die Software für die gewünschte Plattform.

Besonderheiten Ein grosser Vorteil im Vergleich zur Cross-Plattform Konkurrenz ist, dass die lauffähige Applikation rein nativ ausgeführt wird. Es bestehen keine Übersetzungsschichten dazwischen und führt dadurch zu einer effizienteren Programmdurchführung.

Ein grosser Nachteil im Vergleich zur Konkurrenz ist, dass die Portabilität nicht im selben Ausmass vorhanden ist. Nur die Business Logic kann wiederverwendet werden. Alle anderen Bereiche (Benutzerschnittstelle, System Level Calls, Event Binding) müssen für jede Plattform neu geschrieben werden. Der Vorteil in dieser Situation ist, dass für jede Plattform die Applikation in der Sprache C# geschrieben werden kann.

Zukunftsauussichten Durch die offizielle Partnerschaft mit Microsoft und der grossen Entwicklerbasis ist Xamarin einer der Schwergewichte bei den Cross-Plattform-Anbietern. Somit ist die Weiterentwicklung und Wartung in der vorhersehbaren Zukunft gewährleistet. Zusätzlich sind weitere Projekte in Planung, um die Portabilität des Codes zu erhöhen (z.B. MVVMCross).

4.4.9 jQuery Mobile

Firma Die jQuery Foundation ist eine Non Profit Organisation. Gegründet durch John Riesig im Jahr 2006 und nach wie vor finanziert durch ihre Mitglieder³⁰. Die Foundation wurde ursprünglich bekannt mit dem Projekt jQuery - die populärste Javascript-Library. jQuery wird von über 80% der 10'000 meist besuchten Webseiten verwendet³¹.

Im Laufe der Jahre wurden weitere Libraries innerhalb der Foundation entwickelt. Eine resultierende Library ist das jQuery Mobile.

Grund für Evaluation jQuery Mobile wird in die Evaluation miteinbezogen, weil es zu den Markführern im Bereich Cross-Plattform kompatiblen UI-Frameworks gehört. Interessant ist, dass jQuery Mobile einen ganz anderen Ansatz zu den verglichenen marktführenden Technologien wählt. Die genauen Unterschiede werden in den nachfolgenden Absätzen aufgezeigt.

Technologiestack Applikationen, welche die Technologie jQuery Mobile anwenden sind in die Kategorie der Web Applikationen einzuordnen. Der Unterschied zu hybriden und nativen Lösungen besteht darin, dass jQuery mobile-Apps im Browser aufgerufen werden.

Folgender Stack wird bei der Entwicklung mit jQuery Mobile verwendet:

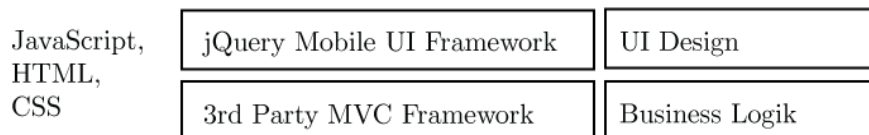


Abbildung 4.12: jQuery Mobile Stack

Gemäss vorgehender Grafik ist ersichtlich, dass jQuery Mobile als User Interface Framework eingesetzt wird. Das Framework ist entwickelt worden, um die Applikationsdarstellung im mobilen Bereich zu standardisieren und dem Benutzer die benutzerfreundliche Interaktion mit der Applikation zu ermöglichen. Das Framework ist in den Sprachen JavaScript, HTML und CSS geschrieben.

Besonderheiten Als grosser Vorteil des jQuery Mobiles ist der Wiedererkennungseffekt der Applikationsanwender. Durch die Popularität des Frameworks sind viele Anwender bestens vertraut mit den User Interface Komponenten (z.B. Knöpfe oder Navigationsleisten) und finden sich innerhalb des Programms schnell zurecht.

³⁰<http://en.wikipedia.org/wiki/JQuery> abgerufen am 12. April, 2014, 10:43 Uhr

³¹<http://en.wikipedia.org/wiki/JQuery> abgerufen am 12. April, 2014, 10:56 Uhr

4.4 Technologie-Recherche

Eine weitere Besonderheit ist die Freiheit für den Entwickler. jQuery Mobile wird nur in der Darstellung verwendet und gibt somit keine Vorschriften und Abhängigkeiten in der Gestaltung anderer Bereiche der Applikation. Da die Applikation im mobilen Webbrowser ausgeführt wird, muss der Entwickler lediglich die Cross Browser Kompatibilität prüfen.

jQuery Mobile bietet keine OS Level Calls an. Deshalb müssen andere Technologie zugezogen werden, um eine Applikation mit Fähigkeiten (z.B. Push Benachrichtigungen) zu realisieren.

Zukunftsauussichten Als die älteste und meist angewandte Technologie im Rahmen dieser Untersuchung gilt sie als sehr sichere Technologie.

4.4.10 OpenUI5

Firma SAP ist ein weltbekanntes Softwareunternehmen, welches im Jahr 1972³² gegründet wurde. Der internationale Erfolg gelang dem Unternehmen mit der proprietären Enterprise Resource Planning Software und ist heute eines der bedeutendsten Schwergewichte in der Softwarebranche.

Grund für Evaluation OpenUI5 wird in die Evaluation auf expliziten Kundenwunsch miteinbezogen.

Technologiestack Applikationen, welche die Technologie OpenUI5 anwenden, sind der Kategorie der Web Applikationen einzuordnen. Der Unterschied zu hybriden und nativen Lösungen liegt für den Applikationsanwender darin, dass die Web-App im Browser aufgerufen wird.

Folgender Stack wird bei der Entwicklung mit OpenUI5 verwendet:

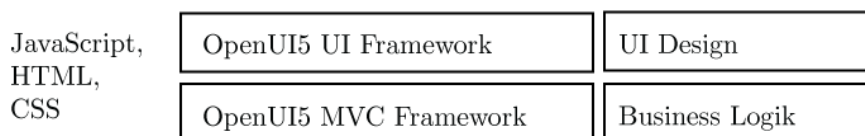


Abbildung 4.13: OpenUI5 Stack

Gemäss vorgehender Grafik ist ersichtlich, dass OpenUI5 die Rollen als User Interface Framework und Model-View-Controller Framework einnimmt. Das umfangreiche Framework ermöglicht dem Entwickler schnell und effizient clientseitige Webapplikationen zu entwickeln. Das Framework ist in den Sprachen JavaScript, HTML und CSS geschrieben.

³²http://en.wikipedia.org/wiki/SAP_AG abgerufen am 12. April, 2014, 12:51 Uhr

Besonderheiten Das OpenUI5 ist mit Abstand das umfangreichste Framework unter den verglichenen webbasierten Frameworks. Einige besonders nennenswerte Features sind:

- Standardmässig eingebauter Schutz gegen Cross-Site-Scripting
- Verfügbarkeit von verschiedenen Themes
- Automatisches Laden von sprachspezifischen Ressourcen

Für eine effiziente Entwicklung wird grosses Wissen zu den angebotenen Features vorausgesetzt. Zudem ist zu vermerken, dass durch die vielen Features die Freiheit der Entwickler eingeschränkt ist.

Zukunftsansichten Mit Produktentwicklung und Produktwartung durchgeführt von SAP ist OpenUI5 als sicher bezüglich der nahegelegenen Zukunft zu kategorisieren.

4.5 Übersicht der Testresultate

Dieser Abschnitt stellt die wichtigsten Eigenschaften der evaluierten Technologien in SWOT-Tabellen dar.

4.5.1 Adobe PhoneGap

Stärken • Bestehende Code-Base kann gut in PhoneGap integriert werden, da wenige Vorgaben • Sehr tiefe Lizenzkosten • Code-Base für Entwicklung vorhanden (in JavaScript) • Entwicklerteam hat Erfahrungen mit JavaScript • Grosse Entwicklergemeinde	Schwächen • JavaScript ermöglicht wenige Debugging-Möglichkeiten • Darstellung von nativer UI-Darstellung nur mit UI-Framework
Chancen • Entwickler PhoneGap-Code kann für QuiXilver Online eingesetzt werden (mobile Version) • Deploymöglichkeit auf allen benötigten Plattformen (iOS, Android, Windows Phone) • Verwenden von Technologie (JavaScript) mit hoher Zukunftserwartung	Risiken • Abhängigkeit von Third-Party Plugin für Push-Notifications • Spürbare Performanceeinbussen im Vergleich zu nativer Implementierung • Längerfristige Marktverdrängung durch Xamarin, da keine Performanceeinbussen

Tabelle 4.3: SWOT-Analyse PhoneGap

4.5.2 AppGyver Steroids

Stärken • Sehr komfortabler Deployment-Prozess • Rapide Entwicklung von nativen User Interfaces • Code-Base für Entwicklung vorhanden (in JS) • Entwicklerteam hat Erfahrungen mit JS	Schwächen • Keine Unterstützung für Windows Phone • Push-Notification nur für iOS unterstützt • JavaScript ermöglicht wenige Debugging-Möglichkeiten • Kleine Entwicklergemeinschaft
Chancen • Entwickelter Steroids-Code kann (mit Anpassungen) für QuiXilver Online eingesetzt werden (mobile Version) • Verwenden von Technologie (JavaScript) mit hoher Zukunftserwartung • Sehr innovative Herstellerfirma mit vielversprechenden, zukünftig erscheinenden Steroids-Add-Ons.	Risiken • Abhängigkeit von kleiner Firma (AppGyver) • Spürbare Performanceeinbussen im Vergleich zu nativer Implementierung

Tabelle 4.4: SWOT-Analyse Steroids

4.5.3 Xamarin

Stärken • Keine Performanzeinbussen im Vergleich zu komplett nativ implementierten Apps • Zugriff auf alle Systemressourcen aller benötigten Plattformen • Entwicklerteam (Vigience AG) hat Erfahrung mit .NET • Technologie-spezifisches, grafisches SDK	Schwächen • Sehr hohe Lizenzkosten • Nur Business-Logik wiederverwendbar auf allen Plattformen • Tiefe der Dokumentation
Chancen • Positive Entwicklungschancen, da Herstellerfirma neu Microsoft Partner • Innovative Technologie mit vielversprechenden, zukünftig erscheinenden Features	Risiken • Zeitaufwendige Einarbeitung in Plattform-spezifische Architektur • Längerfristige Marktverdrängung durch PhoneGap, da tiefere Entwicklungskosten • Schwierigkeiten bei Personal-Rekrutierung für .NET im Vergleich zu JavaScript

Tabelle 4.5: SWOT-Analyse Xamarin

4.5.4 jQuery Mobile

Stärken • Rasches Erlernen der Technologie durch Entwickler • Keine Lizenzkosten • Entwicklerteam (Projektteam sowie Vigience AG) hat bereits Erfahrung mit jQuery Mobile • Grosse Entwicklergemeinde	Schwächen • Kein Zugriff auf Systemressourcen • Keine MVC Struktur vorhanden, muss daher mit anderen Frameworks kombiniert werden • Keine native Darstellung möglich
Chancen • Verwendung von Technologie (JavaScript) mit hoher Zukunftserwartung • jQuery Mobile-Spezialisten sind einfach zu finden	Risiken • Probleme mit Wachstum und Komplexität der Anforderungen

Tabelle 4.6: SWOT-Analyse jQuery Mobile

4.5.5 OpenUI5

Stärken • Featurereiches Framework • Keine Lizenzkosten • Sicherheit des Frameworks • SAP-Vorwissen der Mitarbeiter bei Vigience AG	Schwächen • Langsames Erlernen des Frameworks durch Entwickler • Kein Zugriff auf Systemressourcen • Keine native Darstellung möglich
Chancen • Keine Probleme bei wachsender Komplexität und Anforderungen an Software • Gute Wartung und Aktualisierung der Technologie durch Anbieter	Risiken • OpenUI5 Experten sind schwierig zu finden • Sehr hoher Grad an Vendor-Lock-In, da alle Layers gemäss Vorgabe gestaltet sein müssen

Tabelle 4.7: SWOT-Analyse OpenUI5

4.6 Detaillierte Testresultate

Die detaillierten Testresultate wurden aus Platzgründen in den Teil *Weitere Projektliefergegenstände* in Kapitel 11 verschoben.

4.7 Technologie-Entscheidung

Bei der Entscheidung für eine Technologie formuliert das Projektteam eine Empfehlung. Getroffen wird die Entscheidung durch den Industriepartner.

4.7.1 Empfehlung des Projektteams

Das Projektteam empfiehlt, **PhoneGap** als Technologie zu verwenden. Die wichtigsten Vor- und Nachteile dieser Wahl lauten wie folgt:

Vorteile

- Bestehender JavaScript-Code von QuiXilver Online kann wiederverwendet werden
- Hybrider Entwicklungsansatz ermöglicht das Verwenden von Push Benachrichtigungen, die im Benachrichtigungszentrum des mobilen Betriebssystems angezeigt werden. Dies ist mit reinen Web-Apps nicht möglich.
- Entwickelter PhoneGap-Code könnte später bei QuiXilver Online eingesetzt werden
- Keine Einarbeitung in die verschiedenen Plattformen nötig
- Sehr tiefe Lizenzkosten (Technologienutzung kostenlos, optionaler cloudbasierter Build-Service mit Kosten verbunden)
- Grosse PhoneGap Entwicklergemeinde

Nachteile

- Bei grösseren Apps können Performanceeinbussen auftreten. Schon die Startup-Zeit der App unterscheidet sich von nativen Apps (Xamarin-Prototyp). Die Performanceeinbussen waren bei der Konkurrenz ein Entscheidungsgrund dafür, auf native Implementierung zu wechseln.
- Für Darstellung von nativen UI-Elementen muss ein weiteres Framework beigezogen werden.

4.7.2 Entscheidung des Industriepartners

Der Industriepartner hat sich in einer Besprechung mit dem Projektteam am 15.04.14 (siehe Anhang Kapitel ??) gemäss Empfehlung für **PhoneGap** entschieden.

5 Lösungskonzept QuiXilver Mobile Prototyp

5.1 Systemübersicht

Untenstehende Abbildung zeigt alle für diese Bachelorarbeit relevanten Systeme und ihren Kernaufgaben.

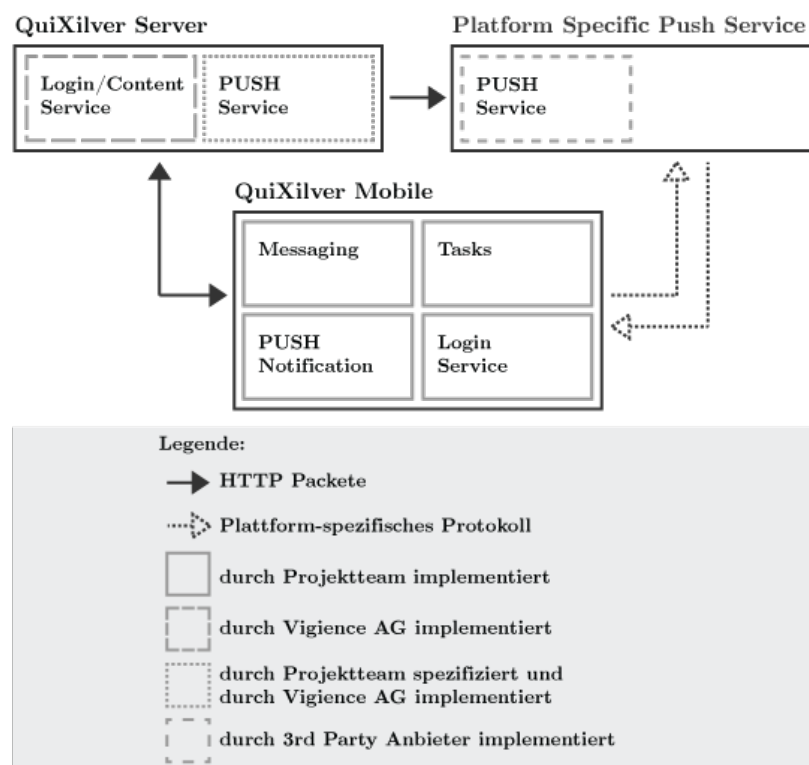


Abbildung 5.1: Systemübersicht QuiXilver

QuiXilver Server stellt einen Login/Content Service sowie einen Push-Service zur Verfügung. Der Login/Content-Service ermöglicht das Anmelden/Abmelden sowie das Abrufen von QuiXilver-Daten. Ein mobiles Gerät kann sich über den Push-Service für Push-Benachrichtigungen registrieren. Der Push-Service ist auch für das Versenden von Push-Benachrichtigungen über den Plattform-spezifischen Push-Service verantwortlich.

Platform Specific Push Service stellt den Push-Service für jede unterstützte mobile Plattform (iOS, Android und Windows Phone) dar. Die genaue Spezifikation der einzelnen Push-Services sowie die Kommunikation zwischen den drei Systemen sind im Kapitel 5.4 zu finden.

QuiXilver Mobile stellt den zu entwickelnden Prototypen der QuiXilver Mobile App dar. Die wichtigsten Kernfunktionalitäten sind das Anmelden beim QuiXilver Server (Login-Service), das Darstellen von Messages und Tasks, sowie das Registrieren und Empfangen von Push-Benachrichtigungen. Die Architektur von QuiXilver Mobile wird in den nachfolgenden Abschnitten beschrieben.

5.2 Schichtenarchitektur QuiXilver Mobile

Dieses Kapitel beschreibt den Schichtenaufbau der QuiXilver Mobile App. Untenstehende Abbildung illustriert den Schichtenaufbau. Im Anschluss wird jede einzelne Schicht detailliert beschrieben.

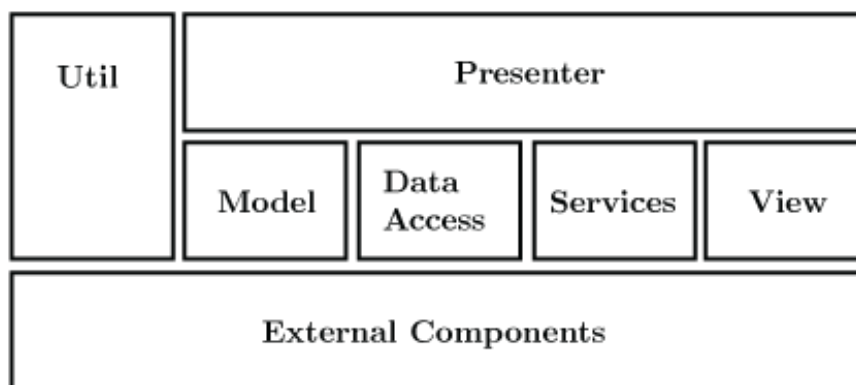


Abbildung 5.2: Schichtenarchitektur QuiXilver Mobile

5.2.1 View

Die View-Schicht ist für die Darstellung der App verantwortlich. Bei Web-Technologien kommen hierfür mehrere HTML- und CSS-Dateien zum Einsatz. Nachfolgend werden die einzelnen Views beschrieben.

Die genaue Struktur der HTML- und CSS-Dateien ist im Kapitel 6.1 beschrieben. Die Architektur der View-Schicht ist nach dem Prinzip von Single-Page App aufgebaut, welches detailliert im Kapitel 5.3.2 beschrieben wird.

- **Index** Hauptseite der Single-Page App. Besteht aus mehreren leeren div-Containern, in welche die nachfolgend beschriebenen Templates beim Seitenaufruf dynamisch nachgeladen werden.
- **Login** Login-Seite
- **Messages** Auflistung aller Messages
- **Message** Detailansicht einer einzelnen Message mit Anzeige der Kommentare
- **Tasks** Auflistung aller Tasks, geordnet nach zugehörigem Status.

5.2 Schichtenarchitektur QuiXilver Mobile

- **Message** Detailansicht eines einzelnen Tasks mit den Details, der Task-Checkliste und den Kommentaren.

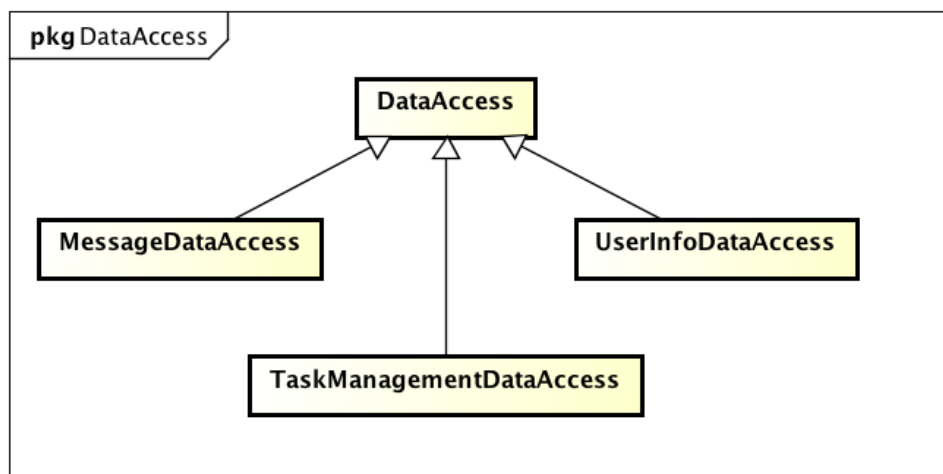
5.2.2 Presenter

In der Presenter-Schicht existiert für jedes oben beschriebene View-Template ein zugehöriger Presenter. Die Presenter haben die folgenden Aufgaben:

- Laden der View und Darstellung in index.html
- Behandeln der Benutzerinteraktionen von Benutzerschnittstelle
- Parsing von erhaltenen Daten von Server

5.2.3 DataAccess

Data Access hat die Aufgabe, die asynchronen HTTP-Calls für den Datenabruf an den QuiXilver Server auszuführen. Hierzu werden asynchrone HTTP-Anfragen an den QuiXilver Server abgesetzt und Daten in den HTTP-Antworten an den Aufrufer zurückgegeben.



powered by Astah

Abbildung 5.3: DataAccess-Schicht Domainmodell

Obenstehende Abbildung illustriert die Klassenstruktur der DataAccess-Schicht. Sie besteht aus den folgenden Klassen:

- **DataAccess** stellt vorgefertigte HTTP-Anfragen den ableiten Klassen zur Verfügung
- **MessageDataAccess** ruft Messages ab
- **TaskManagementDataAccess** ruft Tasks ab
- **UserInfoDataAccess** ruft Informationen des angemeldeten Users ab

5.2.4 Model

Im Model-Package befindet sich die Klassenstruktur, in welcher die vom Server abgerufenen Daten abgelegt werden. Die Daten werden in dieser Struktur persistiert (mehr dazu im Kapitel 5.3.1).

Es werden alle für die Darstellung benötigten Daten von Messages und Tasks in dieser Schicht abgelegt. Zur Veranschaulichung der Struktur ist untenstehend das Domainmodell der Model-Schicht abgebildet. Darauf folgt eine Beschreibung jeder einzelnen Entität und ihrer Kardinalitäten.

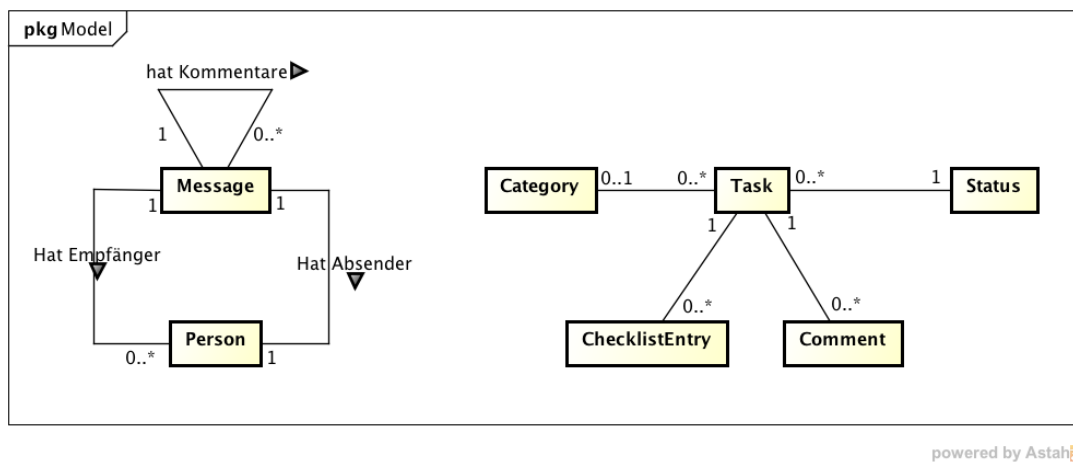


Abbildung 5.4: Model-Schicht Domänenmodell

Message-spezifische Entitäten

- **Message** stellt eine einzige QuiXilver-Message dar. Einer Message kann keine oder beliebig viele weitere Messages als Kommentare hinzugefügt werden. Des weiteren kann eine Message eine beliebig grosse Anzahl an Empfängern beinhalten. Jede Message hat einen einzigen Absender.
- **Person** modelliert einen QuiXilver-Benutzer, der bei einer Message als Absender oder als Empfänger verwendet werden kann.

Folgend wird der Zusammenhang der Entitäten in der JavaScript Object Notation (kurz: JSON) dargestellt:

```
1 {
2   "messages": [
3     {
4       "message": "Hey John. Bist Du heute im Buero?",
5       "sent_by": {
6         "name": "Sarah",
7         "familyName": "Connor"
8       },
9       "receivingPersons" [
10        {
11          "name": "John",
12          "familyName": "Wayne"
13        }
14      ],
15       "comments": [
16         {
17           "message": "Ja. Bis 5 Uhr.",
18           "sent_by": {
19             "name": "John",
20             "familyName": "Wayne"
21           }
22         }
23       ]
24     }
25   ]
26 }
```

Listing 5.1: Vereinfachte JSON-Darstellung: Messages

Task-spezifische Entitäten

- **Task** repräsentiert einen Task des QuiXilver Task Managers. Eine Aufgabe im realen Leben (Bsp. "Papier bestellen") wird auf einen Task projiziert. Jeder Task ist genau einem Status zugeordnet. Weiter kann er einer Kategorie zugewiesen werden und enthält keine oder beliebig viele ChecklistEntries und Comments.
- **Category** repräsentiert eine Task-Kategorie des QuiXilver-Systems.
- **Status** repräsentiert den aktuellen Fortschritt des Tasks. In der QuiXilver-Anwendung sind standardmässig folgende Status definiert: New, In Process, Waiting, Done und Closed. Der Benutzer hat jederzeit die Möglichkeit, weitere Status hinzuzufügen.
- **ChecklistEntry** repräsentiert einen Eintrag in der Checkliste eines Tasks. Der Eintrag kann als offen oder abgeschlossen markiert werden.
- **Comment** repräsentiert einen Kommentar. Jeder Kommentar ist einem Task zugewiesen.

Folgend wird der Zusammenhang der Entitäten in der JSON dargestellt:

```
1 {
2   "tasks": [
3     {
4       "title": "Hotelzimmer buchen",
5       "category": categoryID,
6       "status": statusID,
7       "checklistEntries": [
8         {
9           "title": "Max. 4 Sterne",
10          "state": "checked"
11        }, {
12          "title": "Konferenzraum mit mind. 20 Sitzen",
13          "state": "unchecked"
14        }
15      ],
16      "comments": [
17        {
18          "comment": "Tipp: tripadvisor.com",
19          "sent_by": "John Doe"
20        }
21      ]
22    }
23  ]
24 }
```

Listing 5.2: Vereinfachte JONS-Darstellung: Tasks

5.2.5 Services

Die Services-Schicht stellt die folgenden Funktionalitäten zur Verfügung:

- **Login-Service** führt den Loginprozess auf dem QuiXilver Server durch. Detaillierte Informationen zum Login-Vorgang sind im Kapitel 6.2 zu finden.
- **Push-Service** ist für Push-Benachrichtigungen zuständig. Dazu gehört die Registrierung auf dem Plattform-spezifischen Push Server sowie Push-Registrierung auf dem QuiXilver Server. Erhaltene Push-Benachrichtigungen werden ebenfalls von diesem Service behandelt. Details zur Implementierung sind im Kapitel 6.3

5.2.6 Util

Diese Schicht besteht aus den folgenden zwei Klassen:

- **Util** bietet folgende Funktionalität an: Zeitformatierung, Zeitumrechnung und Pfadbestimmung
- **Constants** definiert alle für die Applikation relevanten Konstanten.

Beide Domänen haben die Eigenschaft, dass sie grundsätzlich von jeder Schicht aufgerufen werden können.

5.2.7 Externe Komponenten

Folgende Grafik illustriert die verwendeten Komponenten von 3rd-Party-Anbieter:

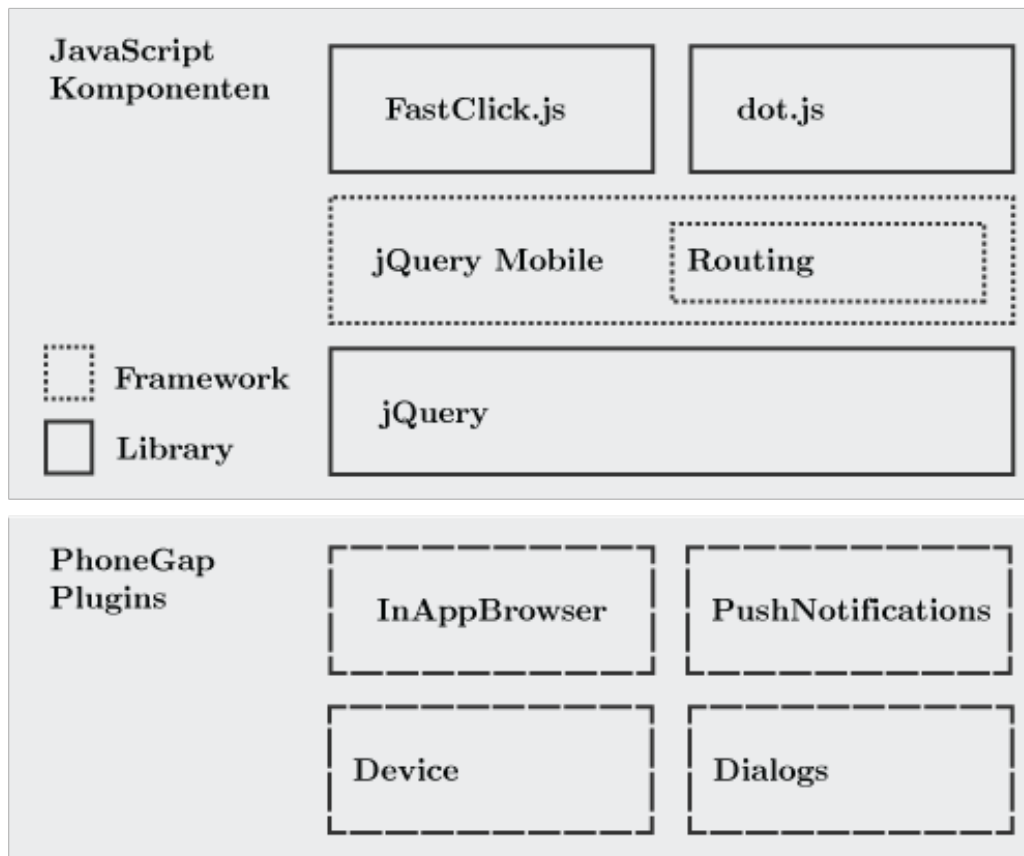


Abbildung 5.5: Verwendete externe Komponenten

FastClick.js ist eine unabhängige JavaScript-Bibliothek, um den 300ms Verzug zwischen physischen Tap auf dem Display und Auslösung des Click-Events auf mobilen Browsern zu beheben. Der 300ms Zeitverzug wurde ursprünglich bei mobilen Browsern eingesetzt, um einen Doppel-Tap abzufangen. FastClick.js vermindert diesen Zeitverzug und steigert damit die Reaktionszeit der Benutzeroberfläche.

jQuery ist eine unabhängige JavaScript-Bibliothek, welche die standardmässige JavaScript-Library erweitert. Im Rahmen dieser Arbeit wird die Library in den folgenden Zwecke verwendet:

- DOM selektieren und modifizieren
- HTTP-Aufrufe absetzen und Antworten abhandeln

jQuery Mobile ist ein Touch-optimiertes Web-Framework, welches auf jQuery aufbaut. Im Rahmen dieser Arbeit werden folgende Elemente des Frameworks eingesetzt:

- Vorgefertigte, Touch-optimierte Bausteine für die Benutzerschnittstelle (Bsp. Knöpfe oder Tabellen), welche grafisch aufgewertet sind
- Vorgefertigte, auf mobile Displays optimierte Layouts, welche grafisch aufgewertet sind
- Rendering und Event-Binding der Seiten
- Routingmechanismus

jQuery Mobile Routing ist ein Plugin, welches den Einsatz von jQuery Mobile voraussetzt. Mit dem jQuery Mobile Routing-Framework wird der Routingmechanismus von jQuery Mobile so verändert, dass ihm eigene Logiken angehängt werden können. Der genaue Mechanismus ist im Unterabschnitt 5.3.2 beschrieben.

5.2.8 Zusammenarbeit der Schichten

In diesem Unterabschnitt wird dargestellt, wie die oben beschriebenen Schichten miteinander kommunizieren. Als konkretes Beispiel wird hierzu das Darstellen von Messages verwendet. Abbildung 5.6 zeigt in Form eines UML-Sequenzdiagramms auf, wie der interne Ablauf bei diesem Prozess aussieht. Bei allen weiteren Views (Message-Detail, Task-Auflistung und Task-Detail) wurde dasselbe Vorgehen umgesetzt. Die UML-Kommentare bei den Klassen geben die Schicht an.

Das Abrufen und Darstellen von neuen Messages ist grundsätzlich in drei Schritte unterteilt.

Abrufen der Messages-Daten von QuiXilver-Server

- A. Der Benutzer klickt auf der Benutzeroberfläche der Messages auf den "Refresh-Button".
- B. Die Funktion `getMessages()` wird auf dem zugehörigen Presenter aufgerufen.
- C. Der Presenter leitet die Anfrage an die `MessageDataAccess`-Klasse weiter. Hierbei werden Callbacks für Success und Error mitgegeben. In der Klasse `MessageDataAccess` wird der URL-String für das Abrufen von Messages aufgebaut.
- D. Die `MessageDataAccess`-Klasse ruft die Funktion `startHTTPCall()` auf ihrer Oberklasse `DataAccess` auf. Dort wird mit JQuery ein asynchroner GET-Request an den Server abgesetzt.
- E. Sobald JQuery eine Antwort vom Server erhalten hat, wird die vom Presenter mitgegebene Callback-Funktion aufgerufen. Die vom Server erhaltenen Daten werden als Parameter mitgegeben.

Parsing der erhaltenen Daten Der `MessagePresenter` ruft die eigene Methode `buildMessageContainer()` auf. Diese Methode hat die Aufgabe, die erhaltene JSON-Datei zu parsen und die darin enthaltenen Daten in Model-Objekten abzulegen. Diese Objekte werden im Anschluss persistiert. Dieser Schritt wurde in obigem Sequenzdiagramm aus Übersichtsgründen bewusst weggelassen. Das Persistierungskonzept ist detailliert im Kapitel 5.3 dokumentiert.

Darstellen der neuen Daten in Messages-View Nachdem die Daten komplett geparkt und persistiert wurden, müssen die Daten in die View gelangen. Hierfür wird die Funktion `loadMessagesView()` aufgerufen. Sie nimmt die Daten aus dem Model und erstellt eine neue Messages-View. Der genaue Vorgang hierbei wird detailliert im Kapitel 5.3 beschrieben.

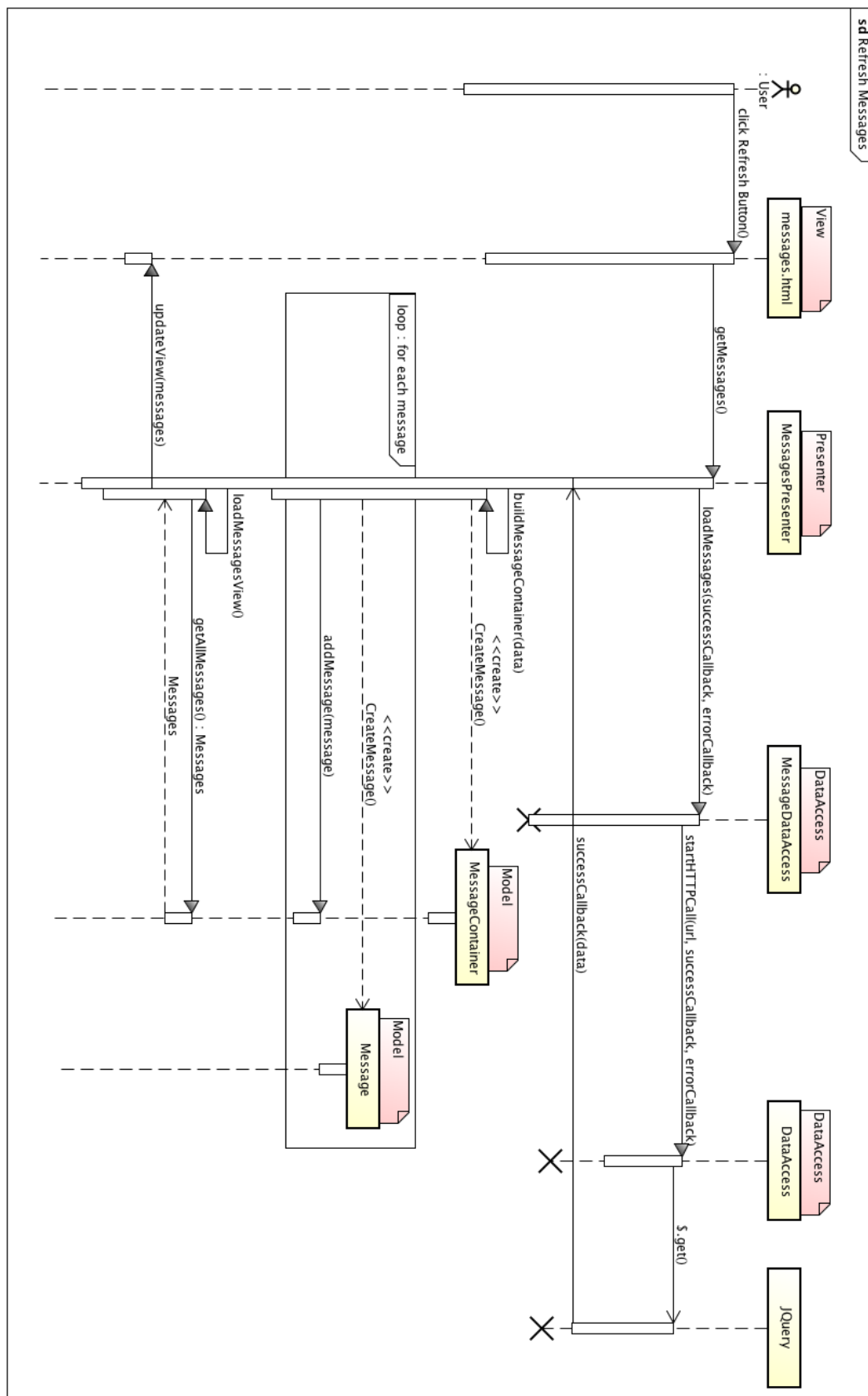


Abbildung 5.6: Sequenzdiagramm Refresh Messages

5.3 Architekturentscheidungen

Dieser Abschnitt beschreibt die wichtigsten Architekturentscheidungen beim Design und der Umsetzung des QuiXilver Mobile Client.

5.3.1 Persistierungskonzept

Anforderungskontext Die Entscheidung, bei QuiXilver Mobile eine Persistierung einzusetzen, leitet sich aus den folgenden Anforderungen ab:

- A. **Benutzeranmeldung** Bei einem erfolgreichen Anmeldeprozess sollen die Tokens, die das Gerät beim Prozess vom Server erhält, persistiert werden, damit der Benutzer sich nicht bei jedem Neustart der App neu anmelden muss. Diese Anforderung wurde im *Use Case 02: Anmelden* im Teil *Weitere Projektliefergegenstände* in Kapitel 9 definiert.
- B. **Offline-Modus** Hat das Gerät des Benutzers keine Verbindung zum Internet, sollten trotzdem die letzten abgerufenen Daten angezeigt werden können. Dies wurde in den nicht-funktionalen Anforderungen im Kapitel 10 des Teils *Weitere Projektliefergegenstände* definiert. Aus diesem Kontext müssen diverse Model-Objekte persistiert werden.

Lösungsbeschreibung Als Lösung für diese Anforderung wurde entschieden, die localStorage Technologie als Persistenzgrundlage zu verwenden. Sie wurde als Teil von Web Storage mit HTML5 eingeführt¹. localStorage bietet im wesentlichen das Speichern von simplen Key-Value-Paaren. Untenstehendes Code-Beispiel zeigt die Verwendung von localStorage aus einer PhoneGap-Applikation. Das Beispiel zeigt, wie ein String mit localStorage persistiert und wieder ausgelesen wird.

```
1 //Get localStorage
2 var localStorage = window.localStorage;
3
4 //define key and value
5 var value = "This is a test";
6 var key = "testKey";
7
8 //save item
9 localStorage.setItem(key, value);
10
11 //get item
12 var retrievedString = localStorage.getItem(key);
13
14 console.log(retrievedString === value); //true
```

Listing 5.3: localStorage Verwendungsbeispiel 1

Als weiterführendes Beispiel wird in nun gezeigt, wie komplette Objekte mit localStorage persistiert werden können. Hierfür muss das zu persistierende Objekt

¹http://www.w3schools.com/html/html5_webstorage.asp, zuletzt aufgerufen am 31.05.2014, 15:13 Uhr

zuerst serialisiert werden, da nur reine Strings mit localStorage gespeichert werden können. Nachdem der Value wieder ausgelesen wurde, muss daraus erneut das Objekt geparkt werden.

```
1 //Get localStorage
2 var localStorage = window.localStorage;
3
4 //define testObject and key
5 var testObject = {
6     testString:"TestString"
7 };
8 var key = "testKey";
9
10 //serialize testObject
11 var value = JSON.stringify(testObject);
12
13 //store value
14 localStorage.setItem(key, value);
15
16 //retrieve string from localStorage
17 var rawObject = localStorage.getItem(key);
18
19 //parse Object
20 var parsedObject = JSON.parse(rawObject);
21
22 //true
23 console.log(parsedObject.testString === testObject.testString);
```

Listing 5.4: localStorage Verwendungsbeispiel 2

Die oben beschriebene Persistierung kommt in den folgenden Architekturschichten zum Einsatz:

- **Presenter** Vor dem Laden der Views werden die persistierten Model-Objekte geladen, damit dem Benutzer auch im Offline-Modus Daten angezeigt werden können.
- **DataAccess** Beim Aufbau der HTTP-Pakete für die Datenanfragen zum QuiXilver Server werden die im localStorage persistierten OpenID Access Tokens verwendet.
- **Services** Beim Durchführen des Anmeldeverfahrens werden die erhaltenen Open-ID-Tokens für den späteren Gebrauch durch die Data Access-Schicht persistiert.

Alternative Lösungen Eine alternative Lösung für das Persistieren von Daten wäre eine direkte Speicherung in einer Datei auf dem Dateisystem des Geräts. PhoneGap bietet für diesen Anwendungsfall eine File-API². Das Projektteam hat sich für die localStorage-Variante entschieden, weil sie eine wesentlich schlankere und besser auf den Anwendungsfall passende API bietet. Die Verwendung von direkten Dateizugriffen hätte keine funktionalen Verbesserungen, sondern lediglich mehr Code-Overhead mit sich gebracht.

²<https://github.com/apache/cordova-plugin-file/blob/master/doc/index.md>, zuletzt aufgerufen am 31.05.2014, 18:28 Uhr

5.3 Architekturentscheidungen

Konsequenzen der Architekturentscheidung Die Stärke der Persistierung mit `localStorage` liegt vor allem in der simplen, aber völlig für den Anwendungsfall ausreichenden API.

Eine Schwäche zeigt sich jedoch in der Security. Bei Webseiten wird klar vom Einsatz von `localStorage` für sensible Daten abgeraten, da verschiedene offene Sicherheitslücken bestehen³. Mit PhoneGap kann `localStorage` besser für sensible Daten eingesetzt werden, da eine Kapselung des `localStorage`-Speichers in der PhoneGap-App eine grundsätzliche Sicherheit bietet. Um vollständige Sicherheit zu erreichen, sollten die Daten jedoch trotzdem verschlüsselt werden⁴.

Bei einer Sitzung mit dem Industriepartner am 05.05.14 (siehe Sitzungsprotokolle im Anhang) wurde beschlossen, dass für den QuiXilver Mobile Prototypen aus Zeitgründen keine Verschlüsselung eingesetzt werden soll. Im Kapitel *Ausblick QuiXilver* (Kapitel 8) wird dokumentiert, dass dies für einen produktiven Release der QuiXilver Mobile App nachgeführt werden müsste.

³<http://stackoverflow.com/questions/3718349/html5-localstorage-security>, zuletzt aufgerufen am 31.05.2014, 18:34 Uhr

⁴<http://stackoverflow.com/questions/18144414/how-secure-is-storing-data-with-localstorage>, zuletzt aufgerufen am 31.05.2014, 18:38 Uhr

5.3.2 Single-page App

Dieser Unterabschnitt beschreibt den gewählten Architekturansatz für den Aufbau der View Schicht und die Zusammenarbeit mit den Klassen aus der Presenter-Schicht.

Anforderungskontext In den nicht-funktionalen Anforderungen an den QuiXilver Mobile Client (siehe *Weitere Projektliefergegenstände*, Kapitel 10) wurde definiert, dass der zu entwickelnde Prototyp eine gute Wart- und Anpassbarkeit aufweisen soll. Dazu gehört, dass die Architektur darauf ausgelegt ist, mit möglichst wenigen Anpassungen zu einem späteren Zeitpunkt weitere Funktionalität hinzuzufügen.

Lösungsbeschreibung Das Projektteam hat sich aus diesem Grund dafür entschieden, Single-Page App-Architektur zu verwenden. Webseiten, die nach diesem Prinzip aufgebaut sind, enthalten die folgenden Elemente:

- **Hauptseite** Die Hauptseite definiert das Gerüst der Webseite. Sie bietet Platzhalter für die später eingefügten Inhalte. Diese Platzhalter wurden mit `div`-Tags umgesetzt.

```

1 <body id="indexBody">
2 <div data-role="page" id="login"></div>
3 <div data-role="page" id="messages"></div>
4 <div data-role="page" id="message"></div>
5 <div data-role="page" id="tasks"></div>
6 <div data-role="page" id="task"></div>
7 ...
8 </body>

```

Listing 5.5: Auszug aus index.html

Obenstehendes Code-Beispiel zeigt einen Auszug aus der Datei `index.html`, die als Hauptseite von QuiXilver Mobile fungiert. Der `body` der Seite besteht lediglich aus `div`-Tags. Die Inhalte der einzelnen Seiten werden in diese `div`-Container geladen, sobald die Seite aufgerufen wird.

- **Template-Seiten** Die sogenannten Template-Seiten enthalten den Aufbau der einzelnen Seiten. Sie werden durch die Template-Library (beschrieben in Unterabschnitt 5.3.5) mit den darzustellenden Daten bestückt.

Dieser Aufbau hat die Eigenschaft, dass die Hauptseite niemals verlassen wird. Führt man die Seite in einem normalen Browser aus, zeigt die URL stets auf `index.html`.

Zusätzlich muss jedoch angegeben werden, welche Unterseite geladen werden soll. Dies wurde mit dem sogenannten Fragment Identifier ⁵ gelöst. Der Fragment Identifier dient dazu, eine Unterressource anzugeben. Der Inhalt wird im Anschluss an

⁵http://en.wikipedia.org/wiki/Fragment_identifier, zuletzt aufgerufen am 4.6.14, 18:28 Uhr

5.3 Architekturentscheidungen

die URL mit einem Doppelkreuz eingeleitet. Danach folgt der Name der gewünschten Ressource. Beim Aufbau der URL kann ebenfalls ein sogenannter Query-String⁶ eingesetzt werden. Dieser wird mit einem Fragezeichen angeführt. Danach folgen Daten, die dem entsprechenden Presenter übergeben werden. Für QuiXilver Mobile wurden die folgenden URLs definiert:

URL	Beschreibung des Aufrufs
<code>index.html#login</code>	Bei diesem Fragment-Identifizier wird automatisch die Login-Seite geladen. Dies ist beim Starten der App der Fall. Sollte der Benutzer bereits eingeloggt sein, wird er jedoch automatisch auf die Messages-Seite weitergeleitet.
<code>index.html#messages</code>	Darstellung der Messages-View, welche alle Messages enthält.
<code>index.html#message?id=1234</code>	Detail-Ansicht einer einzelnen Message. In dieser View werden ebenfalls alle Kommentare angezeigt. Im Query-String der URL wird in der Variable <code>id</code> die Message-ID angegeben.
<code>index.html#tasks</code>	Darstellung der Tasks-View, welche alle Tasks enthält. Die Tasks sind gruppiert nach ihrem Status.
<code>index.html#task?id=1234</code>	Detail-Ansicht eines einzelnen Tasks. Zusätzlich zum Status werden alle Details, Checklisten-Einträge sowie Kommentare angezeigt. Die Task-ID wird wie bei der Detail-Ansicht einer Message mit der Variable <code>id</code> im Query-String übergeben.

Tabelle 5.1: URL-Definitionen QuiXilver Mobile Client

Für die Implementierung der Aufrufe mittels Fragment Identifizier wurde das für JQuery Mobile entwickelte Plugin JQuery Mobile Router verwendet. Es hat die Aufgabe, bei jedem Seitenaufruf von `index.html` den Fragment Identifizier und den Query String ausgelesen. Je nach Fragment Identifizier wird ein unterschiedlicher Presenter aufgerufen, welcher dann wiederum zur Aufgabe hat, die View darzustellen. Der Inhalt des Query Strings wird, sofern vorhanden, ebenfalls an den Presenter weitergeleitet.

⁶http://en.wikipedia.org/wiki/Query_string, zuletzt aufgerufen am 04.06.2014, 19:32 Uhr

```
1 var router = new $.mobile.Router({
2   "#messages": {handler: "messagesHandler", events: "bs"},
3   "#message": {handler: "messageHandler", events: "bs"}
4 }, {
5   messagesHandler: function (type, match, ui) {
6     new MessagesPresenter();
7   },
8   messageHandler: function (type, match, ui) {
9     var params = router.getParams(match[1]);
10    new MessagePresenter(params.id);
11  }
12 });
```

Listing 5.6: Konfigurationsbeispiel des Plugin JQuery Mobile Router

Obenstehendes Code-Beispiel zeigt die abgekürzte Konfiguration des JQuery Mobile Router für QuiXilver Mobile. Das Code-Beispiel zeigt lediglich die Konfiguration für die Ansichten Messages und Message. Die gesamte Definition befindet sich auf der Schicht **Presenter** in der Klasse **MainPresenter**.

Auf **Zeile 1** wird eine neue Instanz des Routers angelegt. Die Parameter dazu sind auf den nachfolgenden Zeilen definiert.

Auf den **Zeilen 2 und 3** werden die Keywords für den Fragment Identifier festgelegt (im Code Beispiel **messages** und **message**). Weiter werden die Methodennamen angegeben, die aufgerufen werden sollen, wenn die Hauptseite mit dem angegebenen Fragment Identifier aufgerufen wird.

Die Implementation der angegebenen Methoden befinden sich auf den **Zeilen 5 bis 11**. Zuletzt werden die Eventcodes festgelegt, bei welchen der entsprechende Handler aufgerufen werden soll. Das Plugin bietet die Möglichkeit, den Handler bei verschiedenen Events aufzurufen. Für QuiXilver Mobile Client wurde lediglich der **bs**-Event gewählt. Er steht für **Before show** und garantiert, dass der Handler aufgerufen wird, bevor die entsprechende View angezeigt wird. Dies ist notwendig, da die View vor der Darstellung mit Daten bestücken werden muss.

In der Plugindokumentation ⁷ des JQuery Mobile Router sind alle angebotenen Eventcodes zu finden.

Auf den **Zeilen 5 bis 11** werden schliesslich die Handler für Messages und Message implementiert. Sie rufen direkt die entsprechenden Presenter auf. Zeile 9 zeigt, wie mit JQuery Mobile Router die Parameter aus dem Query String gelesen werden können. Wie bei den URL-Definitionen bereits beschrieben, muss die ID der entsprechenden Message dem MessagePresenter mitgegeben werden.

⁷<https://github.com/azicchetti/jquerymobile-router>, zuletzt aufgerufen am 04.06.14, 19:58 Uhr

5.3 Architekturentscheidungen

Alternative Lösungen Die Alternative zur gewählten Single-Page Architektur wäre der herkömmliche Aufbau der einzelnen Seiten in getrennten HTML-Dateien. Bei diesem Ansatz wären keine Template-Files im Einsatz, die beim Gebrauch in die vordefinierten Platzhalter geladen werden.

Konsequenzen der Architekturentscheidung Die gewählte Architektur bringt die folgenden Vorteile mit sich:

- **Performance** In der Hauptseite von Single-Page Apps werden alle für die Ausführung der App erforderlichen JavaScript-Dateien geladen. Dadurch, dass die Seite nie verlassen wird, müssen diese Dateien niemals neu geladen werden.
- **Vermeidung von dupliziertem Code** Durch die Hauptseite, die bei Single-Page Apps zum Einsatz kommt, können die Includes von JQuery, JQuery Mobile, doT.js und FastClick.js an einem einzigen Ort durchgeführt werden. Sollte zu einem späteren Zeitpunkt eine weitere externe Komponente verwendet werden, muss der neue Include nur an einem einzigen Ort eingesetzt werden.
- **Zentrale Zuweisung von View zu Presenter** Durch den Einsatz des Routing-Plugins wird die Zuweisung von Presentern zu deren Views an einem einzigen Ort im Code durchgeführt. Bei der Aufteilung der Seite in getrennte HTML Dateien müsste verteilt in jeder HTML-Datei definiert werden, welche Presenter-Klasse geladen wird.

5.3.3 Wahl eines Benutzeroberflächen-Frameworks

Dieser Unterabschnitt beschreibt die Architekturentscheidung der Framework-Wahl für die Benutzeroberfläche.

Anforderungskontext Die folgenden Anforderungen waren relevant:

- **Portabilität** In den nicht-funktionalen Anforderungen an den QuiXilver Mobile Client (siehe Anhang-Kapitel 10) wurde definiert, dass die App auf Android, iOS und Windows Phone 8 zur Verfügung stehen soll.
- **Erlernbarkeit** Zusätzlich wurden in den nicht-funktionalen Anforderungen (siehe Anhang-Kapitel 10) definiert, wie viel Zeit ein neuer Benutzer der App benötigen sollte, um die Use Cases durchzuführen. Die Erlernbarkeit wird maßgeblich durch die verwendeten Oberflächen-Elemente bestimmt. Werden Elemente benutzt, die sich optisch komplett von dem unterscheidet, was der Benutzer in anderen Apps antrifft, kann dies die Erlernbarkeits-Zeit verlängern oder den Benutzer gar überfordern. Aus diesem Kontext ist es wichtig, dass das gewählte Framework Elemente benutzt, deren Funktion für eine breite Benutzergruppe direkt ersichtlich ist.

Lösungsbeschreibung Der Industriepartner hat sich gemeinsam mit dem Projektteam dafür entschieden, jQuery Mobile ⁸ als Framework für die Benutzeroberfläche von QuiXilver Mobile zu verwenden.

Dieser Unterabschnitt erläutert die Funktionsweise von jQuery Mobile an einem kurzen Beispiel. Untenstehende Abbildung illustriert das gewünschte Endresultat.

A Header	
First item	➤
Second item	➤
Third item	➤

Abbildung 5.7: Gewünschtes Endresultat jQuery Mobile-Beispiel

Die Seite besteht aus einem Header und einer Liste mit drei Einträgen. Untenstehender HTML-Code zeigt, wie diese Seite mit jQuery Mobile erstellt werden kann.

⁸<http://jquerymobile.com>, zuletzt aufgerufen am 06.06.14, 11:20 Uhr

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>jQuery Mobile Example</title>
5
6     <meta charset="utf-8" />
7     <meta name="format-detection" content="telephone=no" />
8     <meta name="viewport" content="width=device-width, user-
9         scalable=no">
10
11     <script type="text/javascript" src="jquery.js"></script>
12     <script type="text/javascript" src="jquery.mobile-1.4.2.min.js"
13         ></script>
14
15     <link rel="stylesheet" type="text/css" href="jquery.mobile
16         -1.4.2.css"/>
17 </head>
18 <body>
19     <div data-role="page">
20         <div data-role="header">
21             <h1>A Header</h1>
22         </div>
23
24         <div data-role="content">
25             <ul data-role="listview">
26                 <li><a href="first.html">First item</a></li>
27                 <li><a href="second.html">Second item</a></li>
28                 <li><a href="third.html">Third item</a></li>
29             </ul>
30         </div>
31     </div>
32 </body>
33 </html>
```

Listing 5.7: Auszug aus index.html

Auf den **Zeilen 10 bis 13** werden die für jQuery Mobile benötigten JavaScript- und CSS-Dateien eingefügt. Für die Ausführung von jQuery Mobile muss ebenfalls jQuery importiert werden.

Zeile 16 definiert innerhalb des body-Elements einen div-Container mit dem Attribut `data-role="page"`. Mit diesem Attribut wird eine jQuery Mobile-Seite generiert.

Auf den **Zeilen 17-19** wird der div-Container für den Header definiert. Mit dem `data-role="header"`-Attribut wird mit jQuery Mobile ein Header generiert.

Auf den **Zeilen 22-26** befindet der Seiteninhalt der JQuery Mobile Seite. Das div-Tag mit dem Attribut `data-role="content"` beinhaltet den Inhalt. Darin wird mit `data-role="listview"` eine neue Liste angelegt.

Das Beispiel zeigt, wie Seiten mit jQuery Mobile erstellt werden können. Grundsätzlich kommt normales HTML zum Einsatz. Die HTML-Tags werden jedoch mit `data-role`-Attributen erweitert. Diese Attribute geben an, die das entsprechende Element mit jQuery Mobile gerendert werden soll. Eine vollständige Auflistung aller zur Verfügung stehenden Oberflächenelemente befindet sich auf der jQuery Mobile Webseite ⁹.

Alternative Lösungen Die folgenden Frameworks standen als Alternativen zu jQuery Mobile zur Verfügung:

- **OpenUI5** OpenUI5 wurde im Verlaufe der Technologie-Evaluation durch den Industriepartner vorgeschlagen. Die Funktionsweise der Technologie wird in Unterkapitel 4.4.10 beschrieben. Im direkten Vergleich zu jQuery Mobile unterscheidet sich OpenUI5 darin, dass die Technologie Vorgaben zum Architekturaufbau stellt. Dies hilft bei wachsender Komplexität der App, da auf dokumentierte Architekturvorgaben zurückgegriffen werden können. Jedoch erhöht diese Eigenschaft auch den Grad an Vendor-Lock-In. Sollte zu einem späteren Zeitpunkt ein anderes Framework für die Benutzeroberfläche eingesetzt werden, muss bei OpenUI5 nicht nur der Code der View-Schicht, sondern auch das Model und die Logik angepasst werden.
- **Ionic Framework** Eine weitere Alternative für jQuery Mobile war das Ionic Framework ¹⁰. Ionic ist nicht nur ein reines Framework für die Benutzeroberfläche, sondern arbeitet ebenfalls mit Architekturvorgaben. Als Architekturgrundlage wird Angular.js verwendet. Das Framework baut direkt auf PhoneGap/Cordova auf und stellt somit eine Alternative zur Verwendung von PhoneGap/Cordova dar. Ionic kann nicht als isoliertes Framework für die Benutzeroberfläche verwendet werden. Bei der Evaluation musste Ionic ausgeschlossen werden, da keine Kompatibilität zu Windows Phone 8-Geräten gegeben ist.

Konsequenzen der Architekturentscheidung Der Industriepartner hat sich gemäss einer Empfehlung des Projektteams bei einer Sitzung am 05.05.14 für jQuery Mobile entschieden. Diese Wahl bringt die folgenden Konsequenzen mit sich:

- **Anpassbarkeit** Die geforderte Anpassbarkeit ist mit dem Einsatz von jQuery Mobile aus zwei Gründen sichergestellt. Erstens verfügt jQuery Mobile über eine sehr detaillierte Dokumentation, welche den Einstieg in die Technologie massiv erleichtert. Zweitens wurde jQuery Mobile als isoliertes Framework für die Benutzeroberfläche verwendet. Der tiefe Grad an Vendor-Lock-In erlaubt, das Framework zu einem späteren Zeitpunkt durch ein anderes zu ersetzen, ohne Änderungen am Model vorzunehmen. In der Presenter-Schicht müsste lediglich der Event-Aufruf zum neuen Rendern der Oberfläche nach dem Laden von Views auf das neue Framework angepasst werden.

⁹<http://demos.jquerymobile.com/1.2.1/docs/forms/forms-all.html>, zuletzt aufgerufen am 06.06.14, 11:21 Uhr

¹⁰<http://ionicframework.com>, zuletzt aufgerufen am 06.06.14, 11:29 Uhr

- **Portabilität** Beide evaluierten Frameworks sind kompatibel mit den geforderten Plattformen.
- **Erlernbarkeit** Beide evaluierten Frameworks erfüllen die Anforderung, dass bekannte Oberflächenelemente zum Einsatz kommen.

5.3.4 Einsatz eines Architektur-Frameworks

Anforderungskontext Die folgenden Anforderungen waren für die Entscheidung eines Einsatzes von Architektur-Frameworks relevant:

- **Anpassbarkeit** In den nicht-funktionalen Anforderungen an den QuiXilver Mobile Client (siehe Anhang-Kapitel 10) wurde definiert, dass die erstellte Architektur für spätere Entwickler möglichst anpassbar sein sollte.
- **Kompatibilität mit jQuery Mobile** Zum Zeitpunkt der Entscheidung für einen Einsatz von Architectural-Frameworks hatte der Industriepartner bereits entschieden, dass jQuery Mobile für die Darstellung der Benutzeroberfläche verwendet werden soll. Aus diesem Grund war es ein notwendiges Kriterium für die Architektur-Wahl, dass sie mit jQuery Mobile kompatibel ist.
- **Zuweisung von View zu Presenter** Die technische Anforderung an die Lösung bestand darin, eine Zuweisung von Presentern zu Views herzustellen. Bei jedem Aufruf einer Seite sollte zuerst der Presenter aufgerufen werden, welcher die zugehörige View mit Daten bestückt und anzeigt.

Lösungsbeschreibung Das Projektteam hat sich aus den später in diesem Unterabschnitt genannten Gründen gegen den Einsatz eines Architektur-Frameworks entschieden. Stattdessen wurde die folgende Lösung implementiert:

- **Routing mit jQuery Mobile Plugin** Die Zuweisung von Presentern zu Views wurde mit dem Plugin JQuery Mobile Router gelöst, welches die Routing-Funktionalität von jQuery Mobile um nützliche Funktionen erweitert. Die Funktionsweise des Plugins ist in Unterkapitel 5.3.2 beschrieben.
- **Architektur mit reinem JavaScript** Die Architektur der App wurde ohne den Einsatz von Architectural-Frameworks mit reinem JavaScript aufgebaut.

Alternative Lösungen Eine Alternative zur gewählten Lösung wäre der Einsatz eines Architectural-Frameworks gewesen. Ein Beispiel hierfür ist Angular.js. Das Framework bietet die folgenden Funktionalitäten:

- **Routing** Das Framework bietet eine Routing-Funktionalität. Damit kann angegeben werden, welche Presenter und welche zugehörige View bei einem konkreten Seitenaufruf geladen werden sollen. Der Entwickler muss sich bei einem Einsatz von Angular.js nicht mehr darum kümmern, die entsprechende View zu laden und darzustellen.
- **Module** Mit sogenannten Modulen kann die Businesslogik sowie das Model einer Applikation definiert werden. Beim Routing kann direkt auch angegeben werden, welches Modul dem entsprechenden Presenter übergeben werden soll.

Konsequenzen der Architekturentscheidung Wie bereits beschrieben hat sich das Projektteam gegen den Einsatz eines Architectural-Frameworks entschieden. Die Entscheidung wurde aus den folgenden Gründen gefasst:

- **Inkompatibilität von AngularJS und JQuery Mobile** Der Aufbau von AngularJS als Architekturgrundlage und JQuery Mobile als Framework für die Benutzeroberfläche brachte das Problem mit sich, dass die beschriebene Routing-Funktionalität in beiden Frameworks bereits vorhanden ist. Damit das Routing von AngularJS funktioniert, hätte das Routing von JQuery Mobile manuell im Code des Frameworks deaktiviert werden müssen. Dieser Eingriff hätte der Anpassbarkeit der Software geschadet, da dieselbe Anpassung bei einem Upgrade auf eine neue Version von JQuery Mobile erneut notwendig gewesen wäre.
- **Inkompatibilität von AngularJS und PhoneGap Plugin** Ein weiteres Problem beim Einsatz von Angular.js zeigte sich beim Einsatz des PhoneGap-Plugins InAppBrowser, welches für den Login-Prozess verwendet wurde. Das Plugin öffnet innerhalb der App ein neues Browser-Fenster, in welchem der Login-Prozess durchgeführt wird. Für den Betrieb des Plugins hätten jedoch Anpassungen an Angular.js vorgenommen werden müssen, da das Öffnen von neuen Fenstern von Angular.js separat abgehandelt wird.

Die Entscheidung, kein Architectural-Framework einzusetzen, bringt bei QuiXilver Mobile die folgenden Vorteile mit sich:

- **JQuery Mobile Optimierung** Das für das Routing verwendete Plugin JQuery Mobile Router wurde eigens für JQuery Mobile entwickelt. Es ist kompatibel mit dem standardmässig in JQuery Mobile existierenden Routing und ergänzt die Funktionalität um das Zuweisen von Fragment Identifiern zu Presentern, wie es in Unterkapitel 5.3.2 beschrieben ist.
- **Erfüllt alle Bedingungen** Der gewählte Lösungsansatz erfüllt alle Anforderungen, die durch den Industriepartner und das Projektteam an den Architekturaufbau der App gestellt wurde.
- **Keine redundanten Funktionalitäten** Durch den Verzicht auf Angular.js existiert keine funktionale Redundanz zwischen den Frameworks Angular.js und JQuery Mobile. Damit konnte unbenötigter und duplizierter Code vermieden werden.
- **Kompatibilität mit InAppBrowser** Ohne den Einsatz von Angular.js funktioniert das benötigte PhoneGap-Plugin InAppBrowser problemlos auf allen geforderten Plattformen.

Mit dem Entscheid sind jedoch auch die folgenden Nachteile verbunden:

- **Kein Einsatz eines Architektur-Standards** Der Einsatz von Angular.js hätte dazu geführt, dass die gesamte Architektur des QuiXilver Mobile Client

nach dem Angular-Architekturstandard aufgebaut worden wäre. Dieser Standard ist gut dokumentiert und aufgrund der Popularität von Angular.js vielen Webentwicklern bereits bekannt. Bei dem gewählten Lösungsansatz musste die Architektur der App selbstständig entworfen und mit reinem JavaScript implementiert werden. Zukünftige Entwickler müssen sich zuerst in die entworfene Architektur einarbeiten, um neue Funktionalitäten hinzuzufügen. Das Projektteam begrenzt diese Einarbeitungszeit so gut wie möglich, indem beim Aufbau der App auf dokumentierte Patterns zurückgegriffen wird. Zusätzlich unterstützt die vorliegende Architekturdokumentation, sowie das Technologie-Tutorium.

- **Code-intensive Implementierung** Die Implementation der Architektur mit JavaScript erfordert im Vergleich zu einem Einsatz von Angular.js mehr Code. Wie bereits beschrieben übernimmt Angular die Zuweisung von Presentern zu Views. Bei QuiXilver Mobile mussten das Laden der Views selbstständig implementiert werden. Dies kann die Einarbeitungszeit von zukünftigen Entwicklern ebenfalls verlängern. Damit dieser Effekt verringert werden kann, wurde grosser Wert auf die Dokumentation des Codes mit Kommentaren gelegt.

5.3.5 Wahl einer Template-Library

In diesem Unterabschnitt wird die Architekturentscheidung für die Wahl eines Template-Framework beschrieben.

Anforderungskontext Eine Template-Library ist grundsätzlich nötig für die Implementierung der definierten Use Cases (siehe *Weitere Projektliefergegenstände* Kapitel 9) **05: Auftragsfortschritt kontrollieren** sowie **Use Case 07: Mitteilungen ansehen**. In diesen Use Cases ist gefordert, dass vom QuiXilver Server erhaltene Daten in der App dargestellt werden sollen.

Lösungsbeschreibung Template-Libraries werden als Schnittstellen zwischen erhaltenen Daten und View eingesetzt. Konkret ermöglichen sie, Fragmente des DOM-Baums als sogenannte Templates zu benutzen. Diese Templates werden durch die Library mit Daten befüllt. Innerhalb der Templates wird definiert, welche Daten an welcher Stelle einzusetzen sind.

Das Projektteam hat sich bei der Wahl einer Template-Library für doT.js¹¹ entschieden. In diesem Unterabschnitt wird das Funktionsprinzip von doT.js beschrieben. In einem kleinen Beispiel soll mit der Template-Library ein Titel mit einer kurzen Aufzählung generiert werden. Das gewünschte Endresultat ist in untenstehendem HTML-Beispiel dargestellt.

```

1 <h1>Title of the list</h1>
2 <ul>
3   <li>First item</li>
4   <li>Second item</li>
5   <li>Third item</li>
6 </ul>

```

Listing 5.8: Gewünschtes Resultat des doT.js-Beispiels

Damit dieses Resultat generiert werden kann, müssen zuerst die zugrundeliegenden Daten beschrieben werden.

```

1 var model = {
2   title: "Title of the list",
3   listItems: [
4     "First item",
5     "Second item",
6     "Third item"
7   ]
8 }

```

Listing 5.9: Model für doT.js-Beispiel

Obenstehendes JavaScript definiert das zu verwendende Model-Objekt. Es besteht aus der Variable `title` und dem Array `listItems`. In einem nächsten Schritt muss das Template erstellt werden.

¹¹<http://olado.github.io/doT/index.html>, zuletzt aufgerufen am 04.06.14, 23:17 Uhr

5.3 Architekturentscheidungen

```
1 <h1>{{=it.title}}</h1>
2 <ul>
3   {{~it.listItems :value:index}}
4     <li>{{=value}}</li>
5   {{~}}
6 </ul>
```

Listing 5.10: Template für doT.js-Beispiel

Das obenstehende Beispiel-Template definiert einerseits die HTML-Struktur und gibt andererseits an, wo die Daten des Models eingesetzt werden sollen.

Zeile 1 zeigt, wie ein einzelnes Datenfeld im Template dargestellt werden kann. Die Variable `it` stellt das Objekt dar, welches an die Template-Library als Datengrundlage übergeben wurde. Die Namensgebung `it` ist durch doT.js standardisiert.

Das Statement auf **Zeile 3** startet eine Schleife durch die Variable `listItems`. Der aktuelle Wert und der Index jedes Schlaufendurchgangs werden in die Variablen `value` und `index` geschrieben. Auf **Zeile 4** wird schliesslich für jedes Element des Arrays `listItem` ein Aufzählungspunkt mit dem Wert des Elements angelegt. Auf **Zeile 5** wird die Schleife beendet.

Damit das gewünschte Resultat generiert werden kann, muss zum Schluss muss die Template-Library mit JavaScript aufgerufen werden.

```
1 $.get("template.html", function (templateHTML) {
2   var template = doT.template(templateHTML);
3   var result = template(model);
4   console.log(result); //gewuenshtes resultat
```

Listing 5.11: JavaScript-Code für doT.js-Beispiel

Obenstehender JavaScript-Code zeigt, wie aus Model und Template schliesslich das Resultat generiert werden kann.

Auf **Zeile 1** wird das entsprechende Template-File mit einem JQuery `$.get`-Befehl geladen. Die JQuery-Funktion lädt den gesamten Inhalt der HTML-Datei `template.html` in die Variable `templateHTML`. Diese Variable wird einer anonymen Methode übergeben.

Zeile 2 ist dafür zuständig, aus dem erhaltenen Inhalt der Template-Datei mit doT.js ein Template-Objekt zu erstellen.

Dem erstellten Template-Objekt kann auf **Zeile 3** schliesslich das `model`-Objekt übergeben werden. Die Platzhalter des Template-Files werden mit diesem Befehl durch die Daten aus dem Model ersetzt. Rückgabewert der Funktion ist das mit Daten befüllte Template. Der gewünschte HTML-Code befindet sich in der Variable `result`.

Alternative Lösungen Eine alternative Lösung zum Einsatz einer Template-Library besteht lediglich darin, die gewünschte Funktionalität selber zu entwickeln. In nachfolgendem Absatz sind die Gründe für die Wahl einer Template-Library beschrieben.

Konsequenzen der Architekturentscheidung Das Projektteam hat sich aus den folgenden Gründen für den Einsatz von doT.js als Template-Library entschieden:

- **Gute Erfahrungen** In vorherigen Projekten hat das Projektteam bereits doT.js eingesetzt und gute Erfahrungen damit gemacht.
- **Leserlichkeit** Der Syntax von doT.js ist aus Sicht des Projektteams gut lesbar und übersichtlich.
- **Schlankheit und Performance** Die Library umfasst lediglich 120 Zeilen Code und ist von keiner weiteren Library abhängig. Des Weiteren zeigte eine Internet-Recherche, dass doT.js weitverbreitet als performanteste Template-Library gilt.

5.4 Spezifikation Push Service

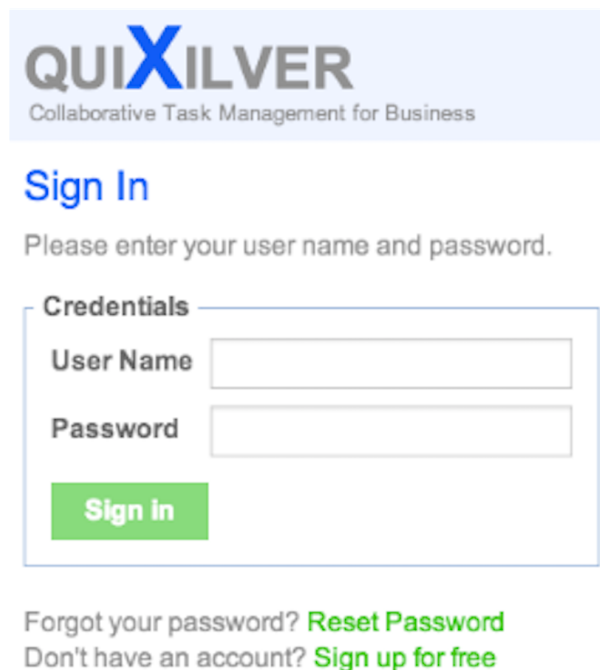
Für die Implementation eines Push Service sind Logiken auf dem QuiXilver Server sowie auf dem QuiXilver Mobile Client notwendig. Da die Logiken auf Serverseite in Japan entwickelt wurden, hat das Projektteam eine Spezifikation des gesamten Push Service auf englisch verfasst. Das gesamte Dokument ist in Teil IV *Spezifikation Push Service* zu finden.

5.5 User Interface

Dieser Abschnitt beschreibt die für den QuiXilver Mobile entworfene grafische Benutzeroberfläche. Die Usability-Analyse, die der Benutzeroberfläche zugrunde liegt, ist im Teil *Weitere Projektliefergegenstände* in Kapitel 12 beschrieben.

5.5.1 Anmeldung

Wie später im Kapitel 6 beschrieben wird, musste aus Sicherheitsgründen das Anmeldeformular von QuiXilver Online verwendet werden. Die Anmeldung wird in einem Popup durchgeführt, wie in untenstehender Abbildung dargestellt wird.



The image shows a login interface for QuiXilver. At the top, the logo 'QUIXILVER' is displayed in blue and grey, with the tagline 'Collaborative Task Management for Business' below it. The main heading is 'Sign In' in blue. Below this, a grey instruction says 'Please enter your user name and password.' The login form is enclosed in a light blue border and contains a 'Credentials' section. This section has two input fields: 'User Name' and 'Password'. Below the input fields is a green 'Sign In' button. At the bottom of the form, there are two links: 'Forgot your password? Reset Password' and 'Don't have an account? Sign up for free', both in green text.

Abbildung 5.8: Login-Formular QuiXilver Mobile

5.5.2 Messages

Nach einem erfolgreichen Login wird der Benutzer direkt auf die Ansicht der Messages weitergeleitet. Untenstehende Abbildung zeigt die Darstellung der Messages.

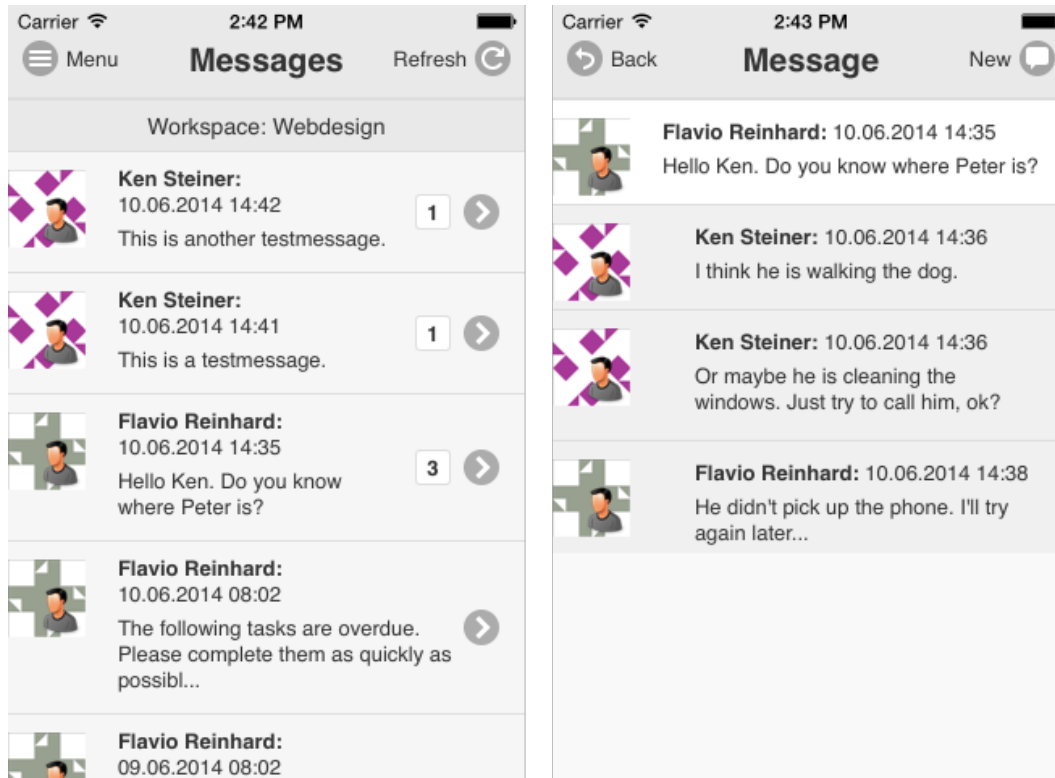


Abbildung 5.9: Messages-Ansicht QuiXilver Mobile

Die linke Abbildung zeigt die Hauptansicht. Die Nachrichten werden absteigend nach Datum dargestellt. Eine kleine Box (in obenstehendem Beispiel mit Inhalt 3) neben der Nachricht zeigt die Anzahl Kommentare an. Über den *Menu*-Button oben links kann die Navigation geöffnet werden. Mit dem *Refresh*-Button oben rechts werden neue Nachrichten vom Server geladen. Der Balken mit dem Text *Workspace: Webdesign* zeigt, in welchem QuiXilver-Workspace sich die dargestellten Nachrichten befinden.

Die rechte Abbildung zeigt die Detail-Ansicht einer einzelnen Message. Sie wird beim Klicken auf eine Nachricht geöffnet. Obenstehendes Beispiel zeigt die Darstellung der Kommentare. Der *Back*-Button oben rechts bringt den Benutzer zurück auf die linke Ansicht. Mit dem *New*-Button oben rechts soll ein neuer Kommentar erfasst werden können. Dieses Feature wurde jedoch nicht im Rahmen der Bachelorarbeit implementiert.

5.5.3 Tasks

Untenstehende Abbildung zeigt die Ansicht der Aufgaben. Die Task-Ansicht kann über die Navigation (wird im Anschluss beschrieben) geöffnet werden.

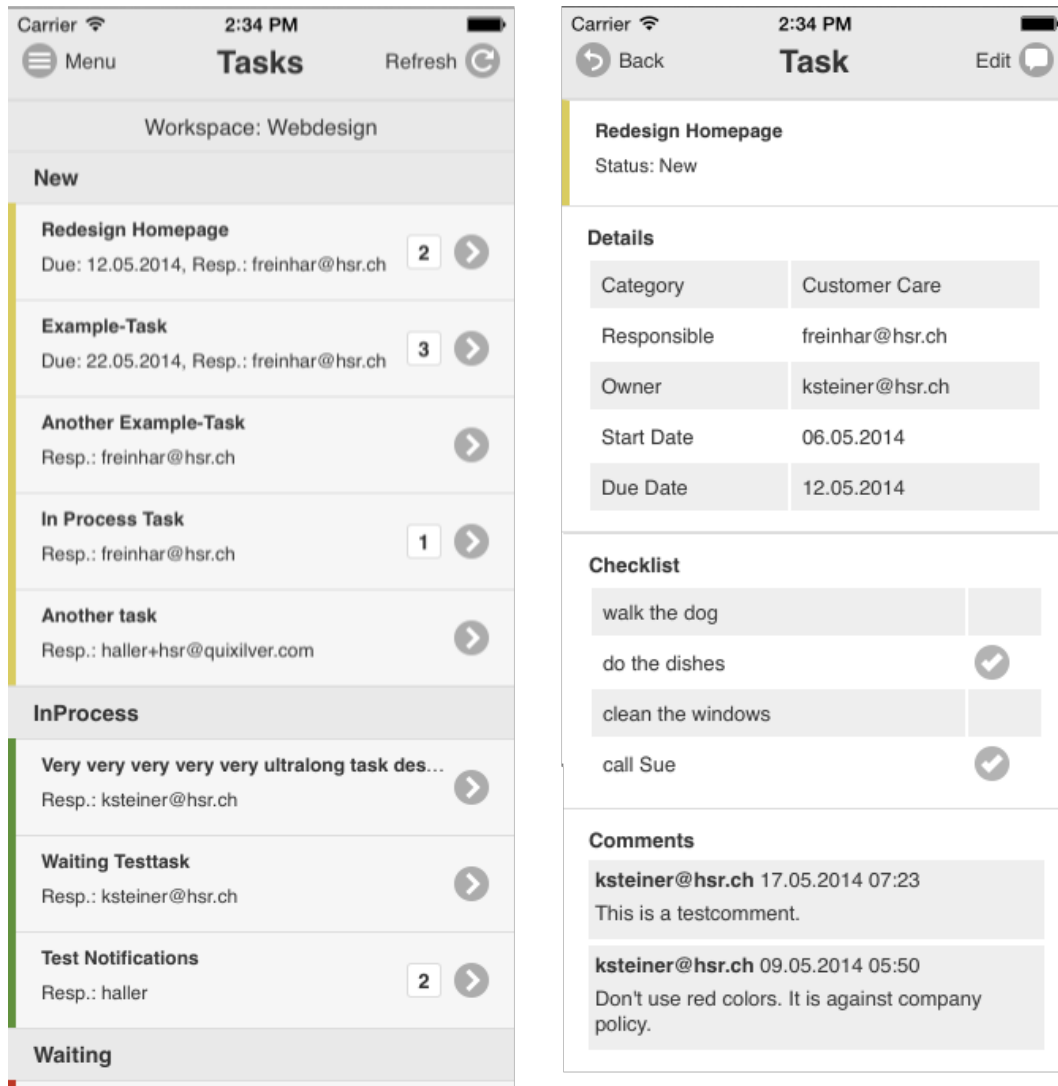


Abbildung 5.10: Tasks-Ansicht QuiXilver Mobile

Die linke Abbildung zeigt die Hauptansicht. Die Tasks werden in Status-Kategorien dargestellt. Die Farbenwahl des schmalen Balkens auf der linken Seite wird automatisch von QuiXilver übernommen. Dies hilft, damit zukünftige Benutzer eine Verbindung zwischen QuiXilver Online und QuiXilver Mobile herstellen können. Auch in dieser Ansicht wird durch eine kleine Box angezeigt, wie viele Kommentare zum entsprechenden Task erfasst wurden. Der *Menu*-Button oben links öffnet das Navigations-Menü. Der *Refresh*-Button oben rechts aktualisiert die Task-Ansicht mit neuen Daten vom Server.

Die **rechte Abbildung** zeigt die Detail-Ansicht eines einzelnen Tasks. Die Ansicht besteht aus den folgenden Elementen:

- **Details** In der Details-Tabelle werden die erfassten Details dargestellt. Sollte ein Detail-Element nicht erfasst sein, wird die entsprechende Zeile automatisch ausgeblendet.
- **Checklist** Die Checklist-Tabelle zeigt alle erfassten Checklist-Elemente. Der Haken zeigt an, ob das entsprechende Element bereits als abgeschlossen markiert wurde.
- **Comments** Der Comments-Teil zeigt alle Kommentare an, die zu diesem Task erfasst wurden. Die Box in der Haupt-Ansicht referenziert auf die Anzahl Kommentare.

5.5.4 Navigation

Wird auf den oben vorgestellten Ansichten der *Menu*-Button angeklickt, öffnet sich das unten abgebildete Navigationsmenü.

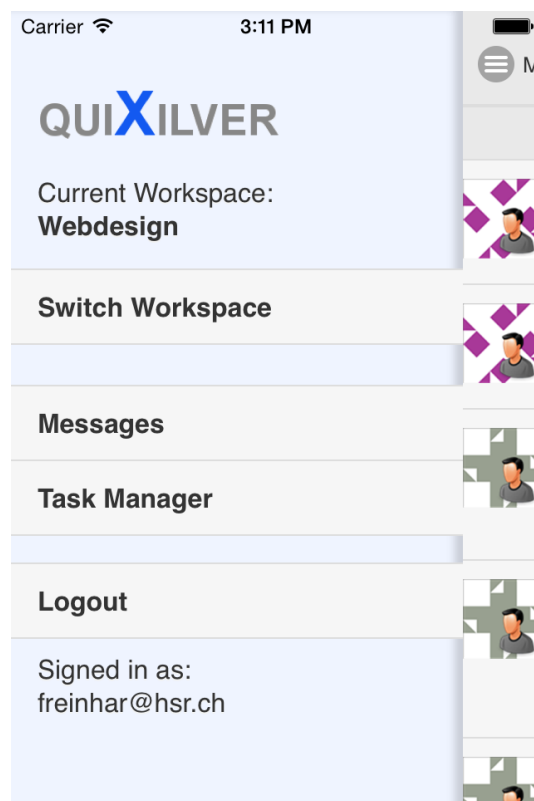


Abbildung 5.11: Navigation QuiXilver Mobile

Das Navigationsmenü besteht aus den folgenden Elementen:

- **Switch Workspace** Über diesen Button hat der Benutzer die Möglichkeit, den Workspace zu wechseln. In unstenstehendem Popup kann der Workspace

5.5 User Interface

gewechselt werden. Für diese Funktionalität wurde nur die Benutzeroberfläche entworfen.

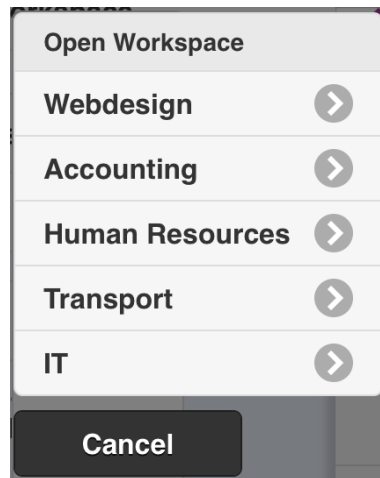


Abbildung 5.12: Switch-Workspace Popup QuiXilver Mobile

- **Messages** Öffnet die Messages-Hauptansicht (wie oben beschrieben).
- **Task Manager** Öffnet die Tasks-Hauptansicht (wie oben beschrieben).
- **Logout** Meldet den Benutzer von QuiXilver ab. Er wird an die Login-Ansicht weitergeleitet.

6 Umsetzung QuiXilver Mobile

In diesem Kapitel werden ausgewählte Themen der Umsetzung des QuiXilver Mobile Clients beschrieben.

6.1 Struktur des Projektordners

Dieser Abschnitt widmet sich der Struktur des Projektordners. Hierbei beschreibt er die durch das Projektteam erstellten Teile der Struktur.

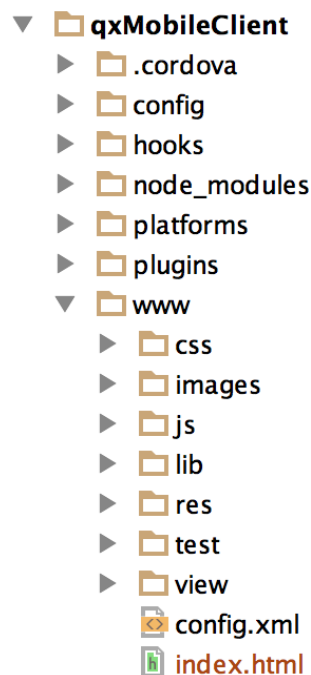


Abbildung 6.1: Ordnerstruktur des Projektordners QuiXilver Mobile Client

Obenstehende Abbildung illustriert die Ordnerstruktur. Die Ordner auf oberster Ebene sind durch PhoneGap vorgegeben und werden beim Erstellen eines neuen Projekts generiert. Der Ordner **www** stellt den Entwicklungsordner der App dar. Die Namensgebung dieses Ordners ist ebenfalls durch PhoneGap vorgegeben. Die Struktur des **www**-Ordners kann jedoch durch die Entwickler definiert werden. Untenstehend folgt eine Auflistung der Ordner und einer Beschreibung ihrer Inhalte.

css Enthält alle CSS-Dateien der QuiXilver Mobile App.

images Enthält alle für die Darstellung benötigten Bilder.

js Enthält die gesamte Applikationslogik. Die verschiedenen JavaScript-Dateien sind gemäss den Applikationsschichten (siehe Abbildung 5.2 in Abschnitt 5.2) in Unterordnern untergebracht.

6.2 Login-Prozess

lib Enthält die Dateien von allen verwendeten Libraries und Frameworks.

res In diesem Ordner sind die Bilddateien für Icons und Splash-Screens der App abgelegt.

test Enthält die Unit-Tests für die Applikationslogik. Hierfür wurde das Test-Framework Jasmine ¹ verwendet. Die für Jasmine benötigten Dateien sind ebenfalls im test-Ordner enthalten. Dies ermöglicht, dass der Ordner entfernt werden kann, damit die Testdateien nicht ungenutzt in eine produktive Version der App eingebaut werden.

view Enthält die View-Templates für die unterschiedlichen Views, wie es im Kapitel 5.3.5 beschrieben wurde.

config.xml Konfigurationsdatei für die PhoneGap-App. Die genauen Aufgaben der Konfigurationsdatei sind im Teil III beschrieben.

index.html Hauptseite der App gemäss Single-Page Prinzip (siehe 5.3.2). PhoneGap gibt den Namen und den Ablageort der index.html-Datei vor.

6.2 Login-Prozess

Für den Login-Vorgang verwendet QuiXilver den OAuth 2.0 Standard ². Dieser Abschnitt beschreibt in einem ersten Schritt den konzeptionellen Ablauf des Prozesses. In einem zweiten Schritt wird beschrieben, wie der Prozess aus Sicht der App implementiert wurde.

6.2.1 Konzeptioneller Ablauf

Untenstehende Grafik illustriert den Ablauf des Login-Prozesses zwischen QuiXilver Mobile und dem QuiXilver Login Service. Im Anschluss an die Grafik werden die einzelnen Schritte beschrieben. Die QuiXilver API ist im Teil *Weitere Projektliefergegenstände* im Kapitel Kapitel 13 dokumentiert.

Request token QuiXilver Mobile initiiert den Login-Prozess, indem eine fix definierte URL aufgerufen wird.

Redirect to Login-Form Der Benutzer wird an das Anmelde-Formular von QuiXilver weitergeleitet.

¹<http://jasmine.github.io/edge/introduction.html>, zuletzt aufgerufen am 07.06.14, 12:18 Uhr

²<http://oauth.net/2/Standard>, zuletzt aufgerufen am 07.06.14, 12:54 Uhr

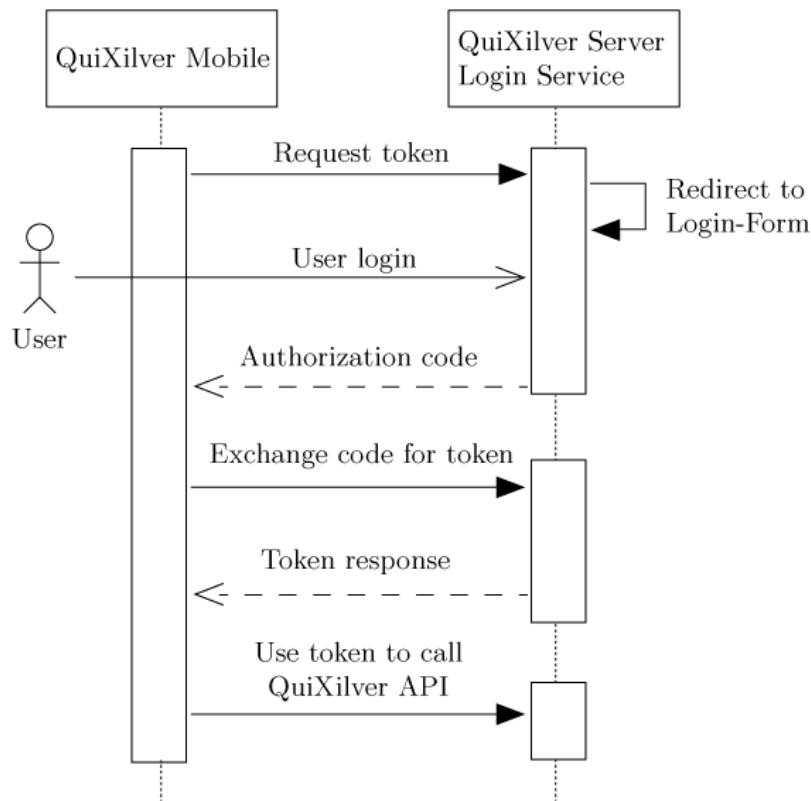


Abbildung 6.2: Konzeptioneller Ablauf des Login-Prozesses für QuiXilver

User Login Im dargestellten Formular gibt der Benutzer seinen Benutzernamen und das Passwort an. Sollte er eine ungültige Kombination eingegeben haben, wird dies direkt im Anmeldeformular dargestellt. Der nächste Schritt im Anmelde-Prozess wird erst ausgeführt, wenn der Benutzer eine gültige Kombination von Benutzernamen und Passwort eingegeben hat.

Authorization Code Der QuiXilver Server Login Service sendet einen Authorization Code an QuiXilver Mobile. Dieser Code wird bei jedem Login-Vorgang neu generiert und kann lediglich ein einziges Mal eingesetzt werden.

Exchange code for token QuiXilver Mobile ruft eine REST-Schnittstelle auf, welche für das Ausstellen von Tokens zuständig ist. Beim Aufruf muss der vorher erhaltene Authorization Code mitgegeben werden.

Token response Sofern ein gültiger Authorization Code angegeben wurde, antwortet der QuiXilver Server Login Service mit den folgenden Tokens:

- **Access Token** Das Access Token autorisiert den Benutzer für das Abrufen von Daten über die QuiXilver API (nächster Schritt). Es hat eine Gültigkeit von 3 Tagen.
- **Refresh Token** Das Refresh Token wird verwendet, um ein neues Access Token zu beantragen. Auch dies geschieht über eine REST-Schnittstelle. Das

6.2 Login-Prozess

Refresh Token hat eine Gültigkeit von einem Jahr. Ist es nicht mehr gültig, muss der komplette Login-Prozess erneut durchgeführt werden.

Access Token und Refresh Token werden durch QuiXilver Mobile persistiert, wie es in Unterabschnitt 5.3.1 beschrieben wurde.

Use token to call QuiXilver API Mit dem erhaltenen Access Token können nun über diverse REST-Schnittstelle Daten abgerufen werden. Die verfügbaren Schnittstellen sind im Teil *Weitere Projektliefergegenstände*, Kapitel 13 zu finden.

6.2.2 Einsatz des InAppBrowser-Plugin

Für den Login-Prozess wurde das PhoneGap-Plugin InAppBrowser³ verwendet. Mit diesem Plugin kann eine externe Webseite in einem Popup geöffnet werden.

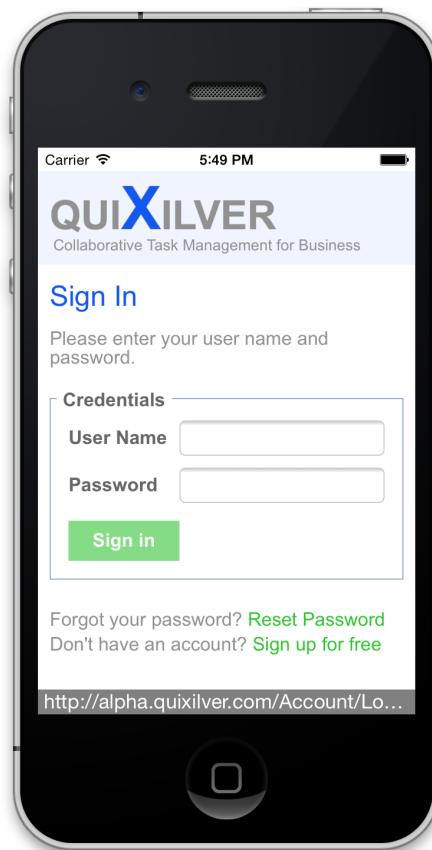


Abbildung 6.3: Darstellung von Login-Maske mit InAppBrowser

Obenstehende Abbildung illustriert die Darstellung der Login-Maske mit dem InAppBrowser-Plugin.

Erster Lösungsansatz Zu Beginn des Projektes strebte das Projektteam einen anderen Lösungsansatz an. Das Loginformular sollte selbstständig als View innerhalb der App dargestellt werden. Die gesamte Kommunikation mit dem QuiXilver

³http://cordova.apache.org/docs/en/3.0.0/cordova_inappbrowser_inappbrowser.md.html, zuletzt aufgerufen am 07.06.14, 17:53 Uhr

Login Service sollte mittels HTTP-Kommunikation im Hintergrund erfolgen. Dieser Lösungsansatz hätte das Prinzip von Single-Page-Apps konsequenter eingehalten. Ein Popup, welches die Loginmaske anzeigt wie es mit dem InAppBrowser der Fall ist, stellt einen Kontextbruch dar, widerspricht dem Single-Page-App-Prinzip und sollte deshalb verhindert werden.

Grund für Einsatz des InAppBrowsers Die Implementation des oben beschriebenen ersten Lösungsansatzes war jedoch nicht möglich. Wie in Abbildung 6.2 ersichtlich, führt der QuiXilver-Server nach dem Erhalten des Request-Token-Aufrufs eine Weiterleitung auf die Login-Maske durch. Dabei liefert der Server im Header-Feld des HTTP-Redirect-Pakets das Cookie `ASP.NET-SessionId` mit. Diese Session-ID muss auf dem Client gespeichert werden und beim Absenden des Anmeldeformulars wieder mitgegeben werden.

Das Problem beim ersten Lösungsansatz bestand darin, dass dieses Cookie nicht ausgelesen werden konnte, wenn die Kommunikation mit jQuery AJAX durchgeführt wird. Erhält die jQuery-Library ein HTTP-Paket mit dem Statuscode 302⁴, welcher einen Redirect initiiert, wird automatisch eine weitere Anfrage an die neue URL abgesetzt. Erst die Antwort auf dieses Paket wird durch jQuery an die höheren Schichten weitergeleitet.

Gemäss Nobuyuki Yokomizu (Software-Engineer von Vigience, Japan) ist es aus Sicherheitsgründen nicht zulässig, das Anmeldeverfahren der QuiXilver Mobile App isoliert mit HTTP-Calls durchzuführen. Aus diesem Grund musste das InAppBrowser-Plugin eingesetzt werden, welches das Anmeldeformular von QuiXilver darstellt und die Persistierung der Cookies übernimmt.

6.3 Implementation des Push Services

Dieser Abschnitt beschreibt die Umsetzung des Push Service in QuiXilver Mobile. Die APIs für die Push Benachrichtigungen auf Serverseite sind im Anhang-Kapitel 13 dokumentiert.

Implementationsort Die Implementation des Push-Service ist der Schicht Services zugeordnet und befindet sich in der Klasse `PushService`.

Identifikation der Plattform Zuerst wird mit dem PhoneGap-Plugin `device`⁵ ausgelesen, auf welcher Plattform die App ausgeführt wird. Je nach Plattform wird der zugehörige Push Service aufgerufen. Jeder dieser Push Services führt die nachfolgenden Schritte durch.

⁴http://en.wikipedia.org/wiki/HTTP_302, zuletzt aufgerufen am 07.06.14, 18:21 Uhr

⁵http://docs.phonegap.com/en/3.0.0/cordova_device_device.md.html, zuletzt aufgerufen am 08.06.14, 13:07 Uhr

6.3.1 Registrierung beim plattformspezifischen Push Server

Die QuiXilver Mobile App registriert sich für Push Benachrichtigungen beim plattformspezifischen Push Service. Hierfür wird das PhoneGap Push Notifications-Plugin⁶ verwendet.

```
1 registerIOS: function () {
2     console.log("Registering iOS push notifications...");
3     var pushNotification = window.plugins.pushNotification;
4     pushNotification.register(
5         iosPushService.tokenHandler,
6         iosPushService.errorHandler, {
7         "badge": "false",
8         "sound": "true",
9         "alert": "true",
10        "ecb": "pushService.onNotificationAPN"
11    });
12 },
13
14 tokenHandler: function (result) {
15     console.log("Successfully registered to APNS. Device token: " +
16         result);
17     pushService.sendRegistrationIDtoQX(result, "apns");
18 },
19
20 errorHandler: function(){
21     console.log("Error occurred while registering to APNS");
22 }
```

Listing 6.1: Registrierung des Push Service

Zeile 4 in obenstehendem Code-Beispiel zeigt die Registrierung beim Apple Push Notification Service (APNS). Der `register()`-Funktion werden die zwei Callback-Funktionen `tokenHandler()` und `errorHandler()` mitgegeben.

Auf den **Zeilen 7-10** wird das Verhalten beim Empfangen einer Push Benachrichtigung konfiguriert. Die Konfigurationsparameter haben die folgenden Auswirkungen:

- **badge** Darstellung der Anzahl von eingetroffenen Push Benachrichtigungen auf dem Icon der App
- **sound** Abspielen eines Tons
- **alert** Darstellung einer Benachrichtigungs-Nachricht im Notification Center des mobilen Betriebssystems
- **ecb** Callback-Funktion, die beim Eintreffen einer Push Benachrichtigung ausgeführt wird, sofern die App zu diesem Zeitpunkt geöffnet ist. Ist die App nicht geöffnet, übernimmt das Betriebssystem die Behandlung.

⁶<https://github.com/phonegap-build/PushPlugin>, zuletzt aufgerufen am 08.06.14, 13:58 Uhr

Zeile 14-17 definiert die Callback-Funktion für die erfolgreiche Registrierung beim APNS. Als Parameter wird das Device Token übergeben, welches für das Versenden von Push Benachrichtigungen benötigt wird.

6.3.2 Registrierung beim QuiXilver Server

Nachdem das Device Token erhalten wurde, muss es an den QuiXilver Server weitergeleitet werden. Wie in der Push-Spezifikation in Teil IV beschrieben, benötigt der QuiXilver Server dieses Token, um eine Push Benachrichtigung über den plattformspezifischen Push Server zu versenden. Die Registrierung beim QuiXilver-Server erfolgt auf **Zeile 16**. Neben dem erhaltenen Device Token muss die Plattform mitgegeben werden.

6.3.3 Erhalten einer Push Benachrichtigung

Nachdem die oben beschriebene Registrierung erfolgreich durchgeführt wurde, können Push Benachrichtigungen empfangen werden. Untenstehendes Code-Beispiel zeigt, wie diese behandelt werden können.

```
1 onNotificationAPN: function (event) {  
2     if (event.alert) {  
3         navigator.notification.alert(event.alert);  
4     }  
5     if (event.sound) {  
6         var snd = new Media(event.sound);  
7         snd.play();  
8     }  
9 }
```

Listing 6.2: Behandlung einer Push Benachrichtigung auf iOS

Wie bereits obenstehend beim Registrierungsprozess beschrieben wurde, wird die Funktion `onNotificationAPN()` aufgerufen, wenn eine Push Benachrichtigung bei geöffneter App empfangen wird. Als Parameter erhält sie ein `event`-Objekt. Darin sind alle Daten des erhaltenen Pakets enthalten.

Ist im erhaltenen Paket eine Alert-Nachricht enthalten, wird sie auf den **Zeilen 2-4** als Message-Popup innerhalb der App dargestellt.

Ist im erhaltenen Paket eine Information zum abzuspielenden Ton enthalten, wird der entsprechende Ton auf den **Zeilen 5-8** abgespielt.

7 Technologieerkenntnisse PhoneGap

Dieses Kapitel zeigt die Stärken und Schwächen von PhoneGap auf. Des weiteren wird beschrieben, für welche Einsatzzwecke in zukünftigen Projekten der Einsatz von PhoneGap Sinn macht.

7.1 Stärken

- **Sehr guter Einstieg** Der Einstieg in die Technologie fällt leicht, da im Internet viele Tutorials verfügbar sind. Ein erstes HelloWorld-Programm kann enorm schnell umgesetzt werden.
- **Plattformunabhängigkeit** PhoneGap bietet eine hohe Plattformunabhängigkeit. Für die Entwicklung des QuiXilver Mobile Prototypen musste lediglich der Push Service plattformspezifisch implementiert werden.
- **Lizenzkosten** PhoneGap unterliegt der Apache License 2.0 ¹ und ist damit im privaten sowie kommerziellen Umfeld mit keinerlei Kosten verbunden.
- **Einsatz von Web-Technologien** PhoneGap stellt keinerlei Architekturvorgaben an den Aufbau der App. Dies bedeutet, dass für die Architektur etablierte Web-Technologien eingesetzt werden können. Diese freie Technologiewahl war bei den anderen evaluierten Technologien nicht der Fall.
- **Wiederverwendung von bestehendem Code** Falls zum Zeitpunkt der Entwicklung einer App bereits eine Webplattform besteht, wie das bei QuiXilver der Fall ist, kann Code wiederverwendet werden, da bei PhoneGap JavaScript zum Einsatz kommt. Dies stellt vor allem einen Vorteil gegenüber nativer App-Implementierung dar.

7.2 Schwächen

- **Performanceeinbussen** Beim Einsatz von PhoneGap treten minimale Zeitverzögerung bei Interaktionen mit der Benutzeroberfläche auf. Der Einsatz der JavaScript-Library FastClick.js vermindert zwar die Zeitverzögerung, sie kann jedoch nicht komplett verhindert werden.
- **Architekturdesign** Bei der Technologie-Evaluation wurde der Fakt, dass PhoneGap keine Architekturvorgaben stellt, als positiv bewertet, da es den Einsatz von unterschiedlichen Web-Technologien ermöglicht. Diese Eigenschaft stellte sich jedoch auch als problematisch heraus. Im Internet sind die unterschiedlichsten Architekturvorschläge zu finden. Eine einheitliche Architekturvorgabe würde dazu führen, dass Begriffe innerhalb der Community einheitlicher benutzt werden. Des weiteren würde eine Architekturvorgabe Neueinstei-

¹<http://phonegap.com/about/>, zuletzt aufgerufen am 08.06.14, 18:17 Uhr

gern einen noch besseren Einstieg ermöglichen, da in jedem PhoneGap-Projekt derselbe Aufbau zu finden wäre.

- **Debugging** Da bei PhoneGap Web-Technologien zum Einsatz kommen, können PhoneGap-Apps theoretisch innerhalb des Browsers ausgeführt werden. Auf diese Weise könnte am effizientesten entwickelt werden, da kein zeitintensiver Deployment-Prozess von Nöten wäre. In der Praxis ist dies jedoch nicht möglich, da die Systemaufrufe an die PhoneGap-API in einem Browser nicht verfügbar sind. Im Gegensatz zur Konkurrenz AppGyver Steroids bietet PhoneGap keine Funktionalitäten für das Debugging wie das Setzen von Haltepunkten.
- **Unterscheidung PhoneGap/Cordova** Das ursprünglich von Adobe gekaufte PhoneGap wurde unter dem Namen Cordova der Apache-License unterstellt. Die Unterscheidung zwischen diesen beiden Technologien ist für Technologieeinsteiger jedoch schwer auszumachen und kann verwirren. Bei der Implementation des QuiXilver Mobile Client hat sich gezeigt, dass keine grossen Unterschiede zwischen den beiden Technologien bestehen. Die Command-Line-Interfaces, welche beispielsweise für das Starten des Buildprozesses verwendet wird, benutzen eine unterschiedliche Befehlssyntax. Die ausgeführten Befehle bewirken jedoch das selbe Resultat.
- **Beschränkte Anzahl Systemaufrufe** PhoneGap bietet diverse Plugins für Systemaufrufe auf das darunterliegende Betriebssystem an. Es ist jedoch festzuhalten, dass nicht alle vom Betriebssystem angebotenen Systemaufrufe mit diesen Plugins umgesetzt werden können.

7.3 Empfohlene Einsatzgebiete

Dieser Abschnitt beschreibt, für welches App-Anforderungsprofil bei zukünftigen Projekten der Einsatz von PhoneGap in Frage kommt. Die Erfahrungen bei der Entwicklung des QuiXilver Mobile Client haben gezeigt, dass PhoneGap im Vergleich zu nativer Implementierung für Apps eingesetzt werden kann, die das folgende Anforderungsprofil beinhalten:

- **Hohe Anforderungen an Plattformunabhängigkeit** Die App soll auf mehreren Plattformen verfügbar sein. Dabei soll möglichst wenig plattformspezifischer Code notwendig sein.
- **Performanceeinbussen verkraftbar** Die Anforderung an wenig plattformspezifischen Code wird höher gewichtet als die Performance der Benutzeroberfläche. Wie oben beschrieben sind minimale Zeitverzögerung bei Benutzerinteraktionen nicht vermeidbar.
- **Know-How in Web-Technologien vorhanden** Im Entwicklungsteam ist Know-How in den Technologien JavaScript, HTML und CSS vorhanden.
- **Listenbasiert** Die App soll hauptsächlich Listen darstellen. 3D-Animationen sind nicht notwendig.
- **Keine nativen Benutzeroberflächen-Elemente** Es ist keine komplett native Benutzeroberfläche gefordert.

8 Ausblick QuiXilver Mobile

Wie in der Aufgabenstellung gefordert, wurde im Rahmen dieser Bachelorarbeit ein Prototyp des QuiXilver Mobile Client umgesetzt. Dieses Kapitel zeigt, welche Schritte für eine produktive Version der App noch umzusetzen sind.

8.1 Use Cases

Die folgenden Use Cases wurden in den Anforderungen an eine produktive Version von QuiXilver Mobile definiert:

- UC04: Workspace wechseln
- UC06: Neue Mitteilung erfassen
- UC09: Auftragsstatus ändern

8.2 Verschiedenene Verbesserungen

Die nachfolgenden Änderungen wurden während des Projektverlaufs als notwendige Verbesserungen für eine produktive Version von QuiXilver Mobile erkannt. Eine Priorisierung durch den Industriepartner ergab, dass diese Verbesserungen jedoch nicht mehr im Rahmen der Bachelorarbeit durchgeführt werden.

- **Push: Öffnen von spezifischen Seiten bei Click** Empfängt ein Mobiltelefon eine Push Benachrichtigung, kann QuiXilver Mobile mit einem Click auf die Benachrichtigung im Notification Center geöffnet werden. Hierbei könnten je nach Push Benachrichtigung spezifische QuiXilver-Seiten geöffnet werden.
- **Sicherheitsverbesserung localStorage** Wie in Kapitel 5.3.1 beschrieben wurde, existiert ein Sicherheitsproblem bei der Verwendung von localStorage. Die persistierten Tokens sind durch die PhoneGap-App gekapselt und nicht direkt sichtbar gegen Aussen. Eine Verschlüsselung der Daten ist trotzdem anzuraten.
- **AJAX-Calls mit HTTPS** Die Sicherheitsrichtlinien von QuiXilver definieren, dass jegliche Kommunikation nur über HTTPS erfolgen darf. Für den Prototypen wurde diese Anforderung nicht umgesetzt, da der zur Verfügung gestellte QuiXilver-Testserver HTTPS nicht unterstützte.
- **Push Service abmelden** Die Abmeldung vom QuiXilver Push Service wurde ebenfalls nicht implementiert, da die API für die serverseitige Abmeldung durch den Industriepartner nicht zur Verfügung gestellt werden konnte.
- **Systemtests** Die Durchführung von Systemtests stellt die Qualität des gesamten QuiXilver-Systems (Server und Mobile Client) sicher.

Quellenverzeichnis

1.1	http://www.gartner.com/newsroom/id/2665715 , zuletzt abgerufen am 19.03.2014, 18:14 Uhr	19
2.1	http://en.wikipedia.org/wiki/ISO/IEC_9126 , zuletzt aufgerufen am 11.06.14, 14:18 Uhr	24
4.1	http://developer.android.com/tools/debugging/ddms.html , zuletzt aufgerufen am 10. April 2014, 19:25 Uhr	32
4.2	http://caniuse.com/websockets , zuletzt aufgerufen am 12. April 2014, 18:04 Uhr	38
4.3	http://de.wikipedia.org/wiki/Safari_(Browser) , zuletzt aufgerufen am 12. April 2014, 18:30 Uhr	39
4.4	https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/NotificationProgrammingGuideForWebsites/Introduction/Introduction.html , zuletzt aufgerufen am 12. April 2014, 18:33 Uhr . . .	39
4.5	https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/NotificationProgrammingGuideForWebsites/Art/safari_notifs_rev_2x.png , zuletzt aufgerufen am 13. April 2014, 14:24 Uhr . . .	39
4.6	http://mac.appstorm.net/general/opinion/the-good-the-bad-and-the-terrible-of-safari-push-notifications/ , zuletzt aufgerufen am 12. April 2014, 18:38 Uhr	40
4.7	https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920 zuletzt abgerufen am 10. April 2014, 10:58 Uhr	40
4.8	http://www.theverge.com/2012/8/23/3262782/facebook-for-ios-native-app zuletzt aufgerufen am 10. April 2014, 13:00 Uhr	41
4.9	http://www.theverge.com/2012/12/13/3763196/facebook-for-android-native-app , zuletzt aufgerufen am 10. April 2014, 13:05 Uhr	41
4.10	http://en.wikipedia.org/wiki/Trello , zuletzt aufgerufen am 10. April 2014, 14:00 Uhr	41
4.11	http://help.trello.com/customer/portal/articles/940690-what-browsers-and-mobile-platforms-does-trello-support- , zuletzt aufgerufen am 10. April 2014, 14:16 Uhr	41
4.12	http://stackoverflow.com/questions/10700460/how-is-trello-iphone-app-developed , zuletzt aufgerufen am 10. April 2014, 13:51 Uhr	41
4.13	http://www.restkit.org , zuletzt aufgerufen am 10. April 2014, 13:56 Uhr	41
4.14	http://en.wikipedia.org/wiki/Podio , zuletzt aufgerufen am 10. April 2014, 15:59 Uhr	41

4.1	http://en.wikipedia.org/wiki/PhoneGap aufgerufen am 6. April, 2014, 9:31 Uhr	43
4.1	http://www.visionmobile.com/product/how-can-html5-compete-with-native/ , zuletzt aufgerufen am 12. April 2014, 18:58 Uhr	43
4.1	http://phonegap.com/about/feature/ , aufgerufen am 6. April 2014, 11:13 Uhr	43
4.1	http://www.appgyver.com/ , zuletzt aufgerufen am 6. April 2014, 13:01 Uhr abgerufen am 6. April, 2014	45
4.1	http://www.appgyver.com/steroids_game_change , zuletzt aufgerufen am 12. April 2014, 18:48 Uhr	45
4.2	http://www.reddit.com/r/webdev/comments/1ksp6l/steroidsjs_bridges_the_performance_gap_between/ , zuletzt aufgerufen am 12. April 2014, 18:47 Uhr	46
4.2	https://github.com/AppGyver/scanner/issues/35 , zuletzt aufgerufen am 13. April 2014, 16:37 Uhr	46
4.2	http://en.wikipedia.org/wiki/Appcelerator , zuletzt aufgerufen am 12. April, 2014, 17:53 Uhr	47
4.2	http://en.wikipedia.org/wiki/Xamarin , abgerufen am 6. April, 2014, 14:05 Uhr	48
4.2	http://xamarin.com/about abgerufen am 6. April, 2014, 14:30 Uhr . . .	48
4.2	http://xamarin.com/ abgerufen am 6. April, 2014, 15:05 Uhr	48
4.2	http://www.microsoft.com/germany/visualstudio/aktuell/news/show.aspx?id=msdn_de_51428 abgerufen am 6. April, 2014, 15:20 Uhr	48
4.2	http://www.visionmobile.com/product/developer-economics-q1-2014-state-developer-nation/ , zuletzt aufgerufen am 13. April 2014, 16:52 Uhr	48
4.2	http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/ , abgerufen am 7. Juni, 2014, 15:10 Uhr .	48
4.2	http://en.wikipedia.org/wiki/JQuery abgerufen am 12. April, 2014, 10:43 Uhr	50
4.3	http://en.wikipedia.org/wiki/JQuery abgerufen am 12. April, 2014, 10:56 Uhr	50
4.3	http://en.wikipedia.org/wiki/SAP_AG abgerufen am 12. April, 2014, 12:51 Uhr	51
5.1	http://www.w3schools.com/html/html5_webstorage.asp , zuletzt aufgerufen am 31.05.2014, 15:13 Uhr	69
5.2	https://github.com/apache/cordova-plugin-file/blob/master/doc/index.md , zuletzt aufgerufen am 31.05.2014, 18:28 Uhr	70
5.3	http://stackoverflow.com/questions/3718349/html5-localstorage-security , zuletzt aufgerufen am 31.05.2014, 18:34 Uhr	71
5.4	http://stackoverflow.com/questions/18144414/how-secure-is-storing-data-with-localstorage , zuletzt aufgerufen am 31.05.2014, 18:38 Uhr . .	71
5.5	http://en.wikipedia.org/wiki/Fragment_identifier , zuletzt aufgerufen am 4.6.14, 18:28 Uhr	72
5.6	http://en.wikipedia.org/wiki/Query_string , zuletzt aufgerufen am 04.06.2014, 19:32 Uhr	73

5.7	https://github.com/azicchetti/jquerymobile-router , zuletzt aufgerufen am 04.06.14, 19:58 Uhr	74
5.8	http://jquerymobile.com , zuletzt aufgerufen am 06.06.14, 11:20 Uhr	76
5.9	http://demos.jquerymobile.com/1.2.1/docs/forms/forms-all.html , zuletzt aufgerufen am 06.06.14, 11:21 Uhr	78
5.10	http://ionicframework.com , zuletzt aufgerufen am 06.06.14, 11:29 Uhr	78
5.11	http://olado.github.io/dot/index.html , zuletzt aufgerufen am 04.06.14, 23:17 Uhr	82
6.1	http://jasmine.github.io/edge/introduction.html , zuletzt aufgerufen am 07.06.14, 12:18 Uhr	91
6.2	http://oauth.net/2/Standard , zuletzt aufgerufen am 07.06.14, 12:54 Uhr	91
6.3	http://cordova.apache.org/docs/en/3.0.0/cordova_inappbrowser_inappbrowser.md.html , zuletzt aufgerufen am 07.06.14, 17:53 Uhr	93
6.4	http://en.wikipedia.org/wiki/HTTP_302 , zuletzt aufgerufen am 07.06.14, 18:21 Uhr	94
6.5	http://docs.phonegap.com/en/3.0.0/cordova_device_device.md.html , zuletzt aufgerufen am 08.06.14, 13:07 Uhr	94
6.6	https://github.com/phonegap-build/PlugIn , zuletzt aufgerufen am 08.06.14, 13:58 Uhr	95
7.1	http://phonegap.com/about/ , zuletzt aufgerufen am 08.06.14, 18:17 Uhr	97
10.	http://martinfowler.com/articles/richardsonMaturityModel.html , aufgerufen am 01.03.2014, 20:07	121
10.2	Vigience AG, Security White Paper, Version 2.0.3	123
13.	http://en.wikipedia.org/wiki/Single-page_application , last visited June 8th, 2014 19:29	175

Glossar

Apps

Webapplikationen innerhalb von QuiXilver, die innerhalb eines Workspace installiert werden können. 6, 15

Dateimanager

QuiXilver-App, die alle Dateien eines Workspace anzeigt. Per Default in jedem Workspace installiert. 14

Messages

QuiXilver-App, mit welcher Nachrichten verschickt werden können. Per Default in jedem Workspace installiert. 14

OS

Operating System. 19, 51

Programme

Auf unbestimmte Zeit wiederkehrende, stets neu abzuarbeitende Arbeitspakete. 14

Projekte

Einmalige Vorhaben mit fixem Start- und Enddatum. Bei Projekten wird häufig ausserhalb der normalen Organisationsstruktur gearbeitet. 14

QuiXilver

Software von Vigience AG, für welche ein Mobile Client entwickelt wird. 12, 14

RAM

Random Access Memory. 32

REST

Representational State Transfer bezeichnet ein Programmierparadigma für Webanwendungen. 13

Single-Page App

Webseite, welche aus einer einzigen HTML-Seite besteht. Die Inhalte der Page werden dynamisch nachgeladen.. 60

UI

User Interface. 44–47, 50, 53, 58

Workspace

Isolierter QuiXilver-Arbeitsordner. Er enthält Apps und ist einer relationalen Datenbank zugewiesen. 14

Abbildungsverzeichnis

1	QuiXilver Mobile Prototyp auf geforderten Plattformen	4
1.1	Systemübersicht QuiXilver	13
1.2	Domain-Übersicht QuiXilver	14
1.3	QuiXilver Online: Workspace-Auswahl	16
1.4	QuiXilver Online: App-Auswahl und Darstellung	16
1.5	QuiXilver Online: Messages	17
1.6	QuiXilver Online: Aufgabenverwaltung	18
2.1	Mobile Ansicht: QuiXilver Online	20
2.2	Use Case Diagram	22
4.1	Benutzeroberfläche Miniprototyp	27
4.2	Unterscheidung Single Plattform und Cross Plattform	33
4.3	Entwicklungskonzepte für Mobile Apps	34
4.4	Push Benachrichtigungen bei einem Android-Gerät ¹	35
4.5	Paketverkehr: Push Benachrichtigungen auf OS-Ebene	36
4.6	Schema für Push Benachrichtigungen auf Browser-Ebene	37
4.7	Schema für Push Benachrichtigungen durch Browser auf Betriebssystem-Ebene	39
4.8	Stack für Phonegap	44
4.9	Steroids Stack	45
4.10	Titanium Stack	47
4.11	Xamarin Stack	49
4.12	jQuery Mobile Stack	50
4.13	OpenUI5 Stack	51
5.1	Systemübersicht QuiXilver	59
5.2	Schichtenarchitektur QuiXilver Mobile	60
5.3	DataAccess-Schicht Domainmodell	61
5.4	Model-Schicht Domänenmodell	62
5.5	Verwendete externe Komponenten	65
5.6	Sequenzdiagramm Refresh Messages	68
5.7	Gewünschtes Endresultat jQuery Mobile-Beispiel	76
5.8	Login-Formular QuiXilver Mobile	85
5.9	Messages-Ansicht QuiXilver Mobile	86

5.10	Tasks-Ansicht QuiXilver Mobile	87
5.11	Navigation QuiXilver Mobile	88
5.12	Switch-Workspace Popup QuiXilver Mobile	89
6.1	Ordnerstruktur des Projektordners QuiXilver Mobile Client	90
6.2	Konzeptioneller Ablauf des Login-Prozesses für QuiXilver	92
6.3	Darstellung von Login-Maske mit InAppBrowser	93
9.1	Use Case Diagram	111
12.1	Skizze: Mitteilungen	146
12.2	Skizze: Aufgaben	147
12.3	Skizze: Navigation	148
12.4	Vorgabe: Loginformular	149
12.5	Erster Entwurf: Messages	151
12.6	Erster Entwurf: Tasks	152
12.7	Erster Entwurf: Navigation	153
12.8	Abgabeverision: Messages	163
12.9	Abgabeverision: Tasks	164
12.10	Abgabeverision: Navigation	165
13.1	Open Project Directory	170
13.2	Build iOS Project	171
13.3	Build iOS Project	171
13.4	Run on iOS device	172
13.5	Settings: USB Debugging	172
13.6	Open Project Directory	173
13.7	Run on Android device	173
13.8	Open Project Directory	174
13.9	Build Windows Phone 8 Project	174
13.10	Open Windows Phone 8 Project	174
13.11	Run on Windows Phone 8 Device	175
13.12	QuiXilver Mobile	175
13.13	DataAccess class hierarchy	176
13.14	Model layer Entities	177
13.15	3rd-Party Components	180

Tabellenverzeichnis

1.1	Weltweite Verkaufszahlen Smartphones	19
2.1	Erlernbarkeit von Use Cases	24
4.1	Kriterientabelle für Technologie-Evaluation	28
4.2	Verbreitete Produkte	42
4.3	SWOT-Analyse PhoneGap	53
4.4	SWOT-Analyse Steroids	54
4.5	SWOT-Analyse Xamarin	55
4.6	SWOT-Analyse jQuery Mobile	56
4.7	SWOT-Analyse OpenUI5	57
5.1	URL-Definitionen QuiXilver Mobile Client	73
10.1	Interoperabilität	121
10.2	Erlernbarkeit von Use Cases	122
12.1	Erlernbarkeit von Use Cases	145
12.2	Handlungssequenzen des Cognitive Walkthrough	156
12.3	Aufgabenstellung an den Testbenutzer	158
12.4	Personendaten: Kandidat Pius Steiner	159
12.5	Testresultate: Kandidat Pius Steiner	159
12.6	Personendaten: Kandidatin Martina Staub	160
12.7	Testresultate: Kandidatin Martina Staub	160
12.8	Personendaten: Kandidat Kristijan Dujmovic	161
12.9	Testresultate: Kandidat Kristijan Dujmovic	161

Teil II

Weitere Projektliefergegenstände

9 Analyse der funktionalen Anforderungen

9.1 Aktorenbeschreibung

In diesem Abschnitt werden die zwei Haupt-Aktoren der Software beschrieben.

9.1.1 Benutzercharakteristik

QuiXilver ist für Benutzer mit grundsätzlichen Kenntnissen in der Bedienung von Computern entwickelt worden. Die Mobile-App ist auf die selbe Zielgruppe ausgerichtet. In den Anwendungsanforderungen der Mobile-App unterteilt sich die Zielgruppe in zwei verschiedene Kategorien: **Manager** und **Mitarbeiter**. Die Erwartungshaltung dieser zwei Benutzerkategorien an die Mobile-App ist wie folgt definiert:

Manager leiten Projekte und Programme. Sie haben die folgende Erwartungshaltung:

- Fortschritt der Tasks in Projekten/Programmen überwachen
- Team koordinieren (primär: Verantwortlichkeiten von Tasks definieren)
- Möglichst schnelle Benachrichtigung bei diversen Neuigkeiten
- Mit dem Team kommunizieren (Nachrichten versenden)

Mitarbeiter bearbeiten Arbeitspakete, die ihnen zugewiesen worden sind. Sie haben die folgende Erwartungshaltung:

- Schnelle Übersicht über **aktuellen Stand** und **Priorität** von ihnen zugewiesenen Arbeitspaketen
- Fortschritte der ihnen zugewiesenen Arbeitspakete erfassen
- Möglichst schnelle Benachrichtigung bei neu zugewiesenen Arbeitspaketen
- Kommunikation mit anderen Mitarbeitern und Vorgesetzten (Nachrichten versenden)

9.2 Use Case Diagramm

In diesem Abschnitt werden die Anwendungsfälle der Software beschrieben. Hierzu wird zuerst ein Use Case Diagramm abgebildet. Im Anschluss folgt eine tabellarische Beschreibung aller Use Cases.

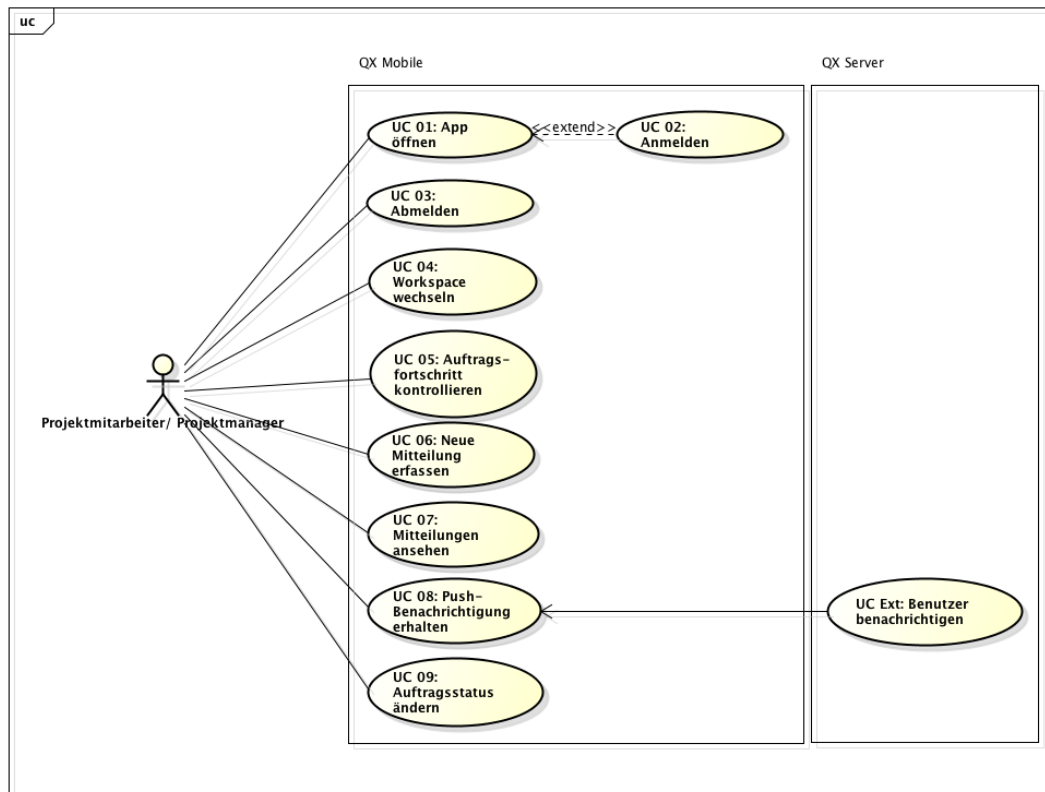


Abbildung 9.1: Use Case Diagram

In der obigen Grafik ist ersichtlich, dass die Benutzerkategorien (Mitarbeiter/-Manager) in einem einzigen Aktor untergebracht sind. Damit wird gezeigt, dass nicht nur ein einziger Aktor existiert. Tatsächlich sind es zwei unterschiedliche Aktoren, denen zu diesem Zeitpunkt jedoch die identischen Use Cases angeboten werden. In einem späteren Entwicklungsstadium könnten den beiden Aktoren unterschiedliche Use Cases zugewiesen werden.

9.2.1 UC 01: App öffnen

Use Case ID	01
Use Case Name	App öffnen
Overview	Der Benutzer öffnet vom Homescreen des Betriebssystems aus die Applikation.
Stakeholder Interests	Benutzer: Der Startvorgang der App soll nicht lange dauern.
Preconditions	QuiXilver Mobile ist erfolgreich installiert und fähig, mit dem Server eine Verbindung herzustellen.
Postconditions	Default Workspace ist geladen.
Format	Fully dressed
Main Success Scenario	1. Benutzer startet Applikation. 2. Der Benutzer wird weitergeleitet in den Workspace, welcher zuletzt verwendet wurde.
Extensions	1a. Falls kein Benutzer angemeldet, dann UC 02: Anmelden durchführen.
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Bei jedem Programmstart.
Open Issues	-
Bemerkungen	-

9.2.2 UC 02: Anmelden

Use Case ID	01
Use Case Name	Anmelden
Overview	Der Benutzer meldet sich mit gültigem Benutzernamen und Passwort bei der Anwendung an.
Stakeholder Interests	Benutzer: Vertraulicher Umgang mit geheimen Daten.
Preconditions	QuiXilver Mobile ist erfolgreich installiert und fähig, mit dem Server eine Verbindung herzustellen. <i>UC 01: App öffnen</i> wurde vorgängig erfolgreich durchgeführt. Gültiger Benutzernamen und Passwort vorhanden.
Postconditions	Sofern der Benutzer eine gültige Kombination von Benutzernamen und Passwort eingegeben hat, wurde er erfolgreich angemeldet. <i>UC 01: App öffnen</i> beenden. Die Anmeldinformationen sind auf dem Gerät gespeichert.
Format	Fully dressed
Main Success Scenario	1. Der Benutzer gibt Benutzernamen und Passwort ein. 2. Der zuletzt geöffnete Workspace wird geladen.
Extensions	1a. Bei fehlerhaften Anmeldungsdaten wird der User benachrichtigt und zur Neueingabe aufgefordert.
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Nach der Installation der Applikation und nach dem Use Case Abmelden
Open Issues	-
Bemerkungen	-

9.2.3 UC 03: Abmelden

Use Case ID	03
Use Case Name	Abmelden
Overview	Der Benutzer meldet sich ab.
Stakeholder Interests	Aus Benutzersicht: Anmeldeinformationen sind nach der Abmeldung auf dem Gerät gelöscht.
Preconditions	Der Benutzer ist erfolgreich angemeldet.
Postconditions	Der Benutzer wurde erfolgreich abgemeldet.
Format	Fully dressed
Main Success Scenario	Der Benutzer meldet sich in der App ab.
Extensions	-
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	-
Open Issues	-
Bemerkungen	-

9.2.4 UC 04: Workspace wechseln

Use Case ID	04
Use Case Name	Workspace wechseln
Overview	Ein Benutzer kann in mehreren Workspaces beteiligt sein. Ist der Benutzer angemeldet, soll er jederzeit den Workspace wechseln können.
Stakeholder-Interests	Benutzer: Gute Ersichtlichkeit des aktuellen Workspaces im Benutzerinterface.
Preconditions	Benutzer ist erfolgreich Angemeldet.
Postconditions	Der vom Benutzer ausgewählte Workspace wurde aktiv.
Format	Fully dressed
Main Success Scenario	1. Der Benutzer lässt sich eine Liste aller ihm zugewiesenen Workspaces anzeigen. 2. Der Benutzer wählt aus, welcher Workspace aktiv werden soll.
Extensions	-
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 6 mal pro Tag
Open Issues	-
Bemerkungen	-

9.2.5 UC 05: Aufgabensfortschritt kontrollieren

Use Case ID	05
Use Case Name	Aufgabensfortschritt kontrollieren
Overview	Manager und Mitarbeiter können sich den Aufgabensfortschritt des aktuell geladenen Workspaces anzeigen lassen.
Stakeholder Interests	Benutzer: Korrekte und aktuelle Darstellung des Fortschritts.
Preconditions	Benutzer ist erfolgreich Angemeldet.
Postconditions	Der Benutzer wurde über den aktuellen Stand der Tasks informiert.
Format	Fully dressed
Main Success Scenario für Projektmanager	1. Der Projektmanager öffnet im aktiven Workspace die <i>Aufgabenmanager</i> (Analoge Terminologie zu QuiXilver Desktop-Client). 2. Der aktuelle Stand der aller Tasks aller Benutzer im Workspace werden angezeigt.
Main Success Scenario für Projektmitarbeiter	1. Der Projektmitarbeiter öffnet im aktiven Workspace die <i>Aufgabenmanager</i> (Analoge Terminologie zu QuiXilver Desktop-Client). 2. Der aktuelle Stand der aller ihm zugewiesenen Tasks im Workspace werden angezeigt.
Extensions	-
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 10 mal pro Tag
Open Issues	-
Bemerkungen	-

9.2.6 UC 06: Neue Mitteilung erfassen

Use Case ID	06
Use Case Name	Mitteilung erfassen
Overview	Manager und Mitarbeiter können in beteiligten Workspaces Mitteilungen an die Wall posten oder an ausgewählte Workspace-Beteiligte senden.
Stakeholder Interests	Benutzer: Einfach und gut ersichtlich.
Preconditions	Benutzer ist erfolgreich Angemeldet.
Postconditions	Die Mitteilung wurde erfolgreich gesendet.
Format	Fully dressed
Main Success Scenario	1. Der Benutzer öffnet Messages-Ansicht. 2. Der Benutzer erfasst die neue Nachricht. 3. Die Message wird vom Benutzer abgeschickt.
Extensions	2a. Dateianhang hinzufügen. 2b. Der Benutzer gibt konkrete Empfänger der Nachricht an, um zu verhindern, dass alle Workspaceteilnehmer benachrichtigt werden.
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 2 mal pro Tag
Open Issues	-
Bemerkungen	-

9.2.7 UC 07: Mitteilungen ansehen

Use Case ID	07
Use Case Name	Mitteilungen ansehen
Overview	Manager und Mitarbeiter können in beteiligten Workspaces Mitteilungen und deren Anhänge ansehen.
Stakeholder Interests	Benutzer: Korrekte und aktuelle Darstellung der Messages.
Preconditions	Benutzer ist erfolgreich Angemeldet.
Postconditions	Der Benutzer sah einen aktuellen Stand der Messages.
Format	Fully dressed
Main Success Scenario	1. Der Benutzer navigiert zu Messages.
Extensions	-
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 2 mal pro Tag
Open Issues	-
Bemerkungen	-

9.2.8 UC 08: Push-Benachrichtigung erhalten

Use Case ID	08
Use Case Name	Push-Benachrichtigung erhalten
Overview	Manager und Mitarbeiter haben die Möglichkeit, bei relevanten Neuigkeiten (z.B. Erledigung eines konkreten Arbeitsschrittes) in Echtzeit informiert zu werden. Sobald die gewünschten Ereignisse eintreten, wird der Benutzer direkt auf dem Smartphone informiert, ohne das Programm QuiXilver Mobile geöffnet zu haben.
Stakeholder Interests	Benutzer: Möglichst schnelle Push-Benachrichtigung von relevanten Statusänderungen der Tasks.
Preconditions	Benutzer ist erfolgreich Angemeldet.
Postconditions	Der Benutzer hat die Push-Benachrichtigung auf seinem Mobiltelefon erhalten.
Format	Fully dressed
Main Success Scenario	1. Die Mobile-App erhält eine Push-Nachricht. 2. Benutzer schaut Push-Nachricht an.
Extensions	2a. Wählt aus, um Messages anzuschauen. Weiter mit UC 07: Mitteilungen ansehen.
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 10 mal pro Tag
Open Issues	-
Bemerkungen	-

9.2.9 UC 09: Aufgabenstatus ändern

Use Case ID	09
Use Case Name	Aufgabenstatus ändern
Overview	Manager und Mitarbeiter haben die Möglichkeit, den Status einer Aufgabe zu ändern.
Stakeholder Interests	Benutzer: Einfach und gut ersichtlich.
Preconditions	Benutzer ist erfolgreich Angemeldet. Benutzer hat mindestens eine zugewiesene Aufgabe.
Postconditions	Aufgabenstatus ist verändert.
Format	Fully dressed
Main Success Scenario	1. Aufgabe selektieren. 2. Status auswählen und speichern.
Extensions	-
Special Requirements	-
Technology and Data Variations	-
Frequency of Occurrence	Schätzung: 5 mal pro Tag
Open Issues	-
Bemerkungen	-

10 Analyse der nicht-funktionalen Anforderungen

Wir orientieren uns bei den nicht-funktionalen Anforderungen an Begriffen der ISO/IEC 9126. Es wurden diejenigen ausgewählt, die für dieses Projekt am relevantesten sind.

10.1 Portabilität

Für den Industriepartner Vigience AG ist die Portabilität die wichtigste nicht-funktionale Anforderung. Sie legt grossen Wert darauf, dass der QuiXilver Mobile Client auf den folgenden Plattformen lauffähig ist:

- **Android** (Version 4.2.2 oder neuer)
- **Windows Phone** (Version 7.8 oder neuer)
- **iOS** (Version 8)

Dabei sollen nicht drei native Apps implementiert werden. Vigience AG wünscht, dass der Client (mit entsprechenden Technologien) so entwickelt wird, dass eine einzige Implementation auf den gewünschten Plattformen installiert werden kann.

10.2 Interoperabilität

QuiXilver Mobile kommuniziert mit den folgenden auf QuiXilver.com Cloud Center angebotenen Services:

Server	Service	Architekturstil
Core Server	ID Verwaltung, Authorisierung	REST
App Server	Cloud Database	REST

Tabelle 10.1: Interoperabilität

Wie in obenstehender Tabelle dargestellt wird, verwenden Core Server und App Server REST als Servicearchitektur. Dabei ist festzuhalten, dass REST in Form des **Richardson Maturity Level 0**¹ zum Einsatz kommt.

10.3 Benutzerschnittstellen

Für die Bedienung der Mobilesoftware werden lediglich die Finger der Benutzer benötigt. Die Interaktionen beschränken sich auf Eingaben über die virtuelle Tastatur und Anwählen von Buttons. Konkrete Benutzerinterface-Designs werden zu einem späteren Zeitpunkt entwickelt.

¹<http://martinfowler.com/articles/richardsonMaturityModel.html>, aufgerufen am 01.03.2014, 20:07

10.3.1 Erlernbarkeit

Messung der Zeit, in welcher ein Benutzer mithilfe der Anleitung selbstständig erlernen kann, eine Aktion in der Software durchzuführen.

Nachfolgende Tabelle definiert die Zeiten, die für die Erlernung und Durchführung eines Use Cases maximal aufgewendet werden soll. Die Zeiten wurden gemeinsam Stephan Haller der Vigience AG erstellt. Weitere Details zu den Benutzertypen können im Abschnitt *Benutzercharakteristik* entnommen werden. Nach der Implementation eines Prototypen wird gemessen, ob diese Zeiten mit echten Benutzern eingehalten werden können.

Use Case	Manager-Zeit	Mitarbeiter-Zeit
UC 01: App öffnen	<10s	<10s
UC 02: Anmelden	<60s	<60s
UC 03: Abmelden	<15s	<15s
UC 04: Workspace wechseln	<15s	<15s
UC 05: Aufgabenfortschritt kontrollieren	<180s	<180s
UC 06: Neue Mitteilung erfassen	<60s	<60s
UC 07: Mitteilungen ansehen	<120s	<120s

Tabelle 10.2: Erlernbarkeit von Use Cases

UC 08: Push-Benachrichtigung erhalten wurde in obenstehender Liste nicht behandelt, da Push-Benachrichtigungen je nach Mobile-OS über externe Server verschickt werden und demzufolge **nicht beeinflussbar** sind.

10.4 Wiederherstellbarkeit

Die Kommunikation mit dem Server erfolgt stateless. Daraus resultiert, dass bei Verbindungsverlust, während die Applikation geöffnet ist, nur die aktuelle Anfrage an den Server terminiert. Anschliessend wird bei der nächsten Aktion, ohne Vorkenntnisse auf den vorhergehenden Fehlversuch, nochmals die Anfrage an den Server gesendet.

Alle abgerufenen Daten werden lokal gespeichert. Werden neuere Daten vom selben Datenstamm geladen, verwirft der Mobile-Client die alten Daten. Sollte die Software mitten in Betrieb abstürzen, dann kann beim nächsten Start die zuletzt heruntergeladenen Daten geladen werden.

10.5 Sicherheit

Gemäss Vigience AG wurden die Sicherheitrichtlinien der QuiXilver-Produktlinie durch externe Stellen geprüft und als sicher eingestuft. Um die Applikationssicherheit auf gleich hohem Niveau zu halten, wird bei der Entwicklung von QuiXilver Mobile Client die Schutzstrategie² von den bestehenden QuiXilver Clients übertragen. Demzufolge kommen die folgenden Sicherheitsmassnahmen zur Verwendung:

- Kommunikation erfolgt ausschliesslich über **HTTPS** (Ausgenommen PUSH-Benachrichtigungen: Da wird der vorgeschlagene Sicherheitsstandard des Betriebssystem verwendet)
- **Cookies** sind exakt **20 Minuten** lang gültig. Sie können erneuert werden oder ablaufen.

Sollte der QuiXilver Mobile-Client zusätzliche, in obiger Liste nicht genannter Security-Standards erfüllen, wird dies zu einem späteren Zeitpunkt mit dem Industriepartner verhandelt.

10.6 Anpassbarkeit

Der QuiXilver Mobile Client, entwickelt im Rahmen der Bachelorarbeit, legt das Fundament zur vollständigen Version des QuiXilver Mobiles. In den Folgejahren soll die App mit weiterer Funktionalität erweitert werden. Eine detaillierte Beschreibung der entwickelten Softwarearchitektur sorgt für eine gute Anpassbarkeit des entwickelten Prototypen.

Ein Nachschlagewerk für Entwickler wird ebenfalls im Rahmen der Arbeit verfasst. Das Dokument soll zukünftigen Entwicklern in folgenden Situationen dienen:

- Strukturkenntnisse über Software übermitteln, um Erweiterung und Anpassungen vereinfacht zu ermöglichen.
- Übersicht der Schnittstellen für die erleichterte Entwicklungen von Apps für QuiXilver Mobile.

²Vigience AG, Security White Paper, Version 2.0.3

11 Detailresultate Technologie Evaluation

11.0.1 PhoneGap

In diesem Unterabschnitt werden die Testresultate von PhoneGap vorgestellt.

Portabilität	Testresultat
Kompatibilität mit den folgenden Plattformen: • iOS (Version 7 oder neuer) • Android (Version 4.2.2 oder neuer) • Windows Phone (Version 8 oder neuer)	PhoneGap unterstützt die geforderten Plattformen.
Zugriff auf die folgenden Betriebssystem-Funktionalitäten: • Push-Benachrichtigungen • GPS Lokalisierung • Volumenregler • 3-Achsen-Gyrosensor • Gespeicherte Kontakte	PhoneGap unterstützt gemäss offizieller API alle genannten Funktionalität mit Ausnahme von Push-Benachrichtigungen. Auf der Homepage befindet sich jedoch ein Verweis auf ein Plugin, welches Push-Funktionalitäten bietet.
Darstellung der grafischen Benutzeroberfläche in nativem Design	Native Darstellung ist nicht in der standardmässigen PhoneGap-Library enthalten. Hierfür müssen User Interface Frameworks hinzugezogen werden. Bei der Implementation des Miniprototypen wurde hierfür Angular UI Mobile verwendet.
Aufwand für Plattformwechsel (Vendor Lock-in)	PhoneGap bietet lediglich den nativen Container und keine UI und MVC Frameworks. Apps werden mit reinem HTML, CSS und JavaScript entwickelt. Für native UI-Darstellung und MVC müssen weitere Third-Party Frameworks hinzugezogen werden. Damit bleibt PhoneGap vergleichsweise plattformunabhängig. Der Aufwand bei einem Plattformwechsel bleibt somit tief.

Security	Testresultat
Wird HTTPS unterstützt?	Ja, HTTPS wird durch PhoneGap unterstützt.

Community	Testresultat
Hilfestellungen in Foren	Das Forum von PhoneGap bietet im Vergleich zu den anderen evaluierten Technologien sehr viele Hilfestellungen.

Erlernbarkeit und Dokumentation	Testresultat
Erlernbarkeit	Auf der Webseite sind Plattform-spezifische Guides für alle angebotenen Plattformen zu finden. Anhand dieser Tutorials erhält man einen schnellen Einstieg in die Technologie.
Dokumentation	Es gibt sehr ausführliche Dokumentation der API. Dabei sind für alle Funktionen auch Codebeispiele zu finden.

Lizenzbedingungen	Testresultat
Welcher Lizenz untersteht die kommerzielle Nutzung der Technologie?	Apache License, Version 2.0 ⁷
Wie hoch sind die Entwicklungsgebühren?	Die Entwicklung von mobile Apps mit PhoneGap ist kostenlos für jeden Verwendungszweck. Zusätzlich bietet PhoneGap einen cloudbasierten Build Service an. Die erste App ist kostenlos, weitere 24 Apps kosten monatlich 9.99 US-Dollar. Dieser Build Service-Dienst ist jedoch fakultativ.

⁷<http://phonegap.com/about/license/>, zuletzt aufgerufen am 5. April 2014, 16:01 Uhr

Wartbarkeit	Testresultat
Qualität von erhaltenen Fehlermeldungen	Fehler, die beim Build-Prozess auftreten, werden in der Konsole angezeigt. Die angezeigten Fehler sind sehr ausführlich und auf den ersten Blick kryptisch. Die Erfahrungen bei der Entwicklung des Miniprototypen haben jedoch gezeigt, dass mit genauerer Betrachtung der Fehlermeldungen viele Fehler behoben werden konnten.
Debugging Möglichkeiten	PhoneGap bietet keine eigenen Debugging-Möglichkeiten an, die über das übliche an JavaScript-Debugging hinausgeht. Es können beispielsweise (im Gegensatz zu Xamarin) keine Breakpoints gesetzt werden.

Entwicklungsumgebung	Testresultat
Umfang an Hilfsfunktionen	Die wichtigsten Hilfsfunktionen: • Automatische Ausführung des Buildprozesses auf allen Plattformen • Automatisches Hochladen der App in den PhoneGap Build Service • Starten von Plattform-spezifischen Simulatoren • Erstellen eines Projektordners gemäss Vorlage
Bedienbarkeit der Entwicklungsumgebung	Die Entwicklungsumgebung ist per Konsole bedienbar. Grundlegende Kenntnisse in der Bedienung von Konsolen reichen aus. Die nötigen Befehle sind kurz und prägnant.

Installationsdauer	Testresultat
Installationsdauer der Entwicklungsumgebung	Resultat der Zeitmessung: 6 Minuten
Installationsdauer aller benötigten Zusatztools für die Entwicklung auf den geforderten Plattformen	Resultat der Zeitmessung: 10 Minuten. Die folgenden Tools sind benötigt: • Node.js • NPM (Node Package Manager) Die Dauer für die Installation der Plattform-spezifischen SDKs ist in der angegeben Zeit nicht enthalten.
Dauer zur Erstellung einer lauffähigen HelloWorld Applikation	Resultat der Zeitmessung: 3 Minuten. Das konsolenbasierte PhoneGap SDK bietet die automatische Generierung einer HelloWorld-Applikation

Speichereffizienz	Testresultat
Dateigrösse des installierbaren APK (Android Package File)	28.3 MB
Speicherplatzbelegung nach dem Starten des Miniprototypen	27.45 MB

Fluidität der Benutzeroberfläche	Testresultat
Werden lange Listen beim Scrollen dargestellt, ohne dass nachgeladen werden muss?	Beim erstmaligen Scrollen nach dem Starten der App ist ein kurzes Flackern erkennbar. Dies tritt jedoch nur beim erstmaligen Scrollen bei hoher Geschwindigkeit auf. Bei weiteren Tests war die Fluidität nicht weiter zu beanstanden.
Werden Animationen beim Wechseln eines Tabs angezeigt, ohne dass ein Ruckeln feststellbar ist?	Der PhoneGap-Prototyp beinhaltet keine Übergangs-Animationen.

Performance	Testresultat
Zeitmessung der Startzeit der App	Resultate der Messungen: • 2.81 Sekunden • 2.89 Sekunden • 2.81 Sekunden Durchschnitt: 2.84 Sekunden

11.0.2 AppGyver Steroids

Portabilität	Testresultat
Plattformkompatibilität: • iOS (Version 7 oder neuer) • Android (Version 4.2.2 oder neuer) • Windows Phone (Version 8 oder neuer)	AppGyver Steroids unterstützt kein Windows Phone, sondern lediglich iOS und Android. Es sei jedoch geplant, in Zukunft weitere Plattformen anzubieten. ⁷
Zugriff auf die folgenden Betriebssystem-Funktionalitäten: • Push-Benachrichtigungen • GPS Lokalisierung • Volumenregler • 3-Achsen-Gyrosensor • Gespeicherte Kontakte	Steroids unterstützt die genannten Betriebssystem-Funktionalitäten mit Ausnahme von Push-Benachrichtigungen. Für diese Funktionalität existiert zwar ein Third-Party Plugin. Damit wird jedoch nur iOS unterstützt. Nach Aussagen von offiziellen Steroids-Entwicklern sollen Push-Benachrichtigungen ohne Plugin in Zukunft möglich werden. Konkrete Zeitangaben sind dazu jedoch nicht zu finden. ⁸
Darstellung der grafischen Benutzeroberfläche in nativem Design	Apps, die mit Steroids entwickelt wurden, weisen eine hohe optische Ähnlichkeit zu nativ entwickelten Apps auf. Bei der Entwicklung des Steroids-Miniprototypen musste kein UI-Framework beigezogen werden. Im Rahmen des Build-Prozesses werden gewisse UI-Elemente wie beschrieben in die nativen Elemente der jeweiligen Plattformen übersetzt.
Aufwand für Plattformwechsel (Vendor Lock-in)	Durch die beschriebenen CSS-Klassen bestehen Steroid-Apps im Vergleich zu PhoneGap-Apps aus viel Steroids-spezifischem Code, was den Aufwand für einen Plattformwechsel erhöht.

⁷http://www.appgyver.com/steroids_game_change, zuletzt aufgerufen am 12. April 2014, 18:56 Uhr

⁸<https://moot.it/i/appgyver/steroids:does-steroids-support-push>, zuletzt aufgerufen am 5. April 2014, 15:38

Security	Testresultat
Wird HTTPS unterstützt?	Ja, HTTPS wird durch Steroids unterstützt.

Community	Testresultat
Hilfestellungen in Foren	Angestellte von AppGyver unterstützen Steroids-Entwickler im Forum. Die Community ist jedoch spürbar kleiner als die Community des direkten Konkurrenzprodukts PhoneGap.

Erlernbarkeit und Dokumentation	Testresultat
Erlernbarkeit	Mit der Steroids-SDK lassen sich lokal verschiedene Beispiel-Apps sowie Tutorials generieren. Mit diesen Hilfen fällt der Einstieg in die Technologie sehr leicht. Auf der Webseite sind zusätzlich diverse Guides zu finden. ⁹
Dokumentation	API-Beschreibungen sind verfügbar und grundsätzlich in guter Qualität mit viel Beispielcode. Die Erfahrungen bei der Entwicklung der Miniprototypen hat gezeigt, dass es leider häufig bei Code-Beispielen bleibt und keine weiterführenden Informationen zu finden sind. ¹⁰

⁹<http://x.appgyver.com/categories>, zuletzt aufgerufen am 5. April 2014, 16:06 Uhr

¹⁰<http://docs.appgyver.com>, zuletzt aufgerufen am 5. April 2014, 16:07 Uhr

Lizenzbedingungen	Testresultat
Welcher Lizenz untersteht die kommerzielle Nutzung der Technologie?	AppGyver Steroids ist ein proprietäres Produkt. Die Nutzung der Software ist wie folgt definiert: AppGyver claims no ownership or control over any Content or Application created by You. You retain copyright and any other rights You already hold in the Content and/or Application, [...] Furthermore, by creating an Application through use of the AppGyver Services, You give AppGyver a worldwide, royalty-free, and non-exclusive license to reproduce, adapt, modify, translate, publish, publicly perform, publicly display and distribute such Application for the sole purpose of enabling AppGyver to provide You with the AppGyver Services. ¹¹
Wie hoch sind die Entwicklungsgebühren?	Es wird damit geworben, dass die Technologie für jeden Verwendungszweck kostenlos ist und auch bleiben wird. Für jährlich 199 Dollar können jedoch Add-ons und weitere Cloud-Dienste beansprucht werden. Zu den Add-ons gehört beispielsweise ein Plugin für die Anbindung von Facebook-Diensten. ¹²

¹¹<http://www.appgyver.com/tos>, zuletzt aufgerufen am 5. April 2014, 16:18 Uhr

¹²<http://www.appgyver.com/steroids/pricing>, zuletzt aufgerufen am 5. April 2014, 16:20 Uhr

Wartbarkeit	Testresultat
Qualität von erhaltenen Fehlermeldungen	Die Fehlermeldungen des Build-Vorgangs sind in der Regel zu kurz, um aussagekräftig zu sein.
Debugging Möglichkeiten	Bei Steroids-Projekten können keine Breakpoints gesetzt werden. Grundsätzlich ist das Debugging durch den komfortablen Deployment-Prozess im Vergleich zur Konkurrenz stark erleichtert.

Entwicklungsumgebung	Testresultat
Umfang an Hilfsfunktionen	Die Entwicklungsumgebung ist vergleichbar mit der PhoneGap-Entwicklungsumgebung. Die wichtigsten sind auch hier die folgenden: • Automatische Ausführung des Build-Prozesses für alle Plattformen • Automatisches Hochladen der App in den PhoneGap Build Service • Starten von Plattformspezifischen Simulatoren • Erstellen eines Projektordners gemäss Vorlage
Bedienbarkeit der Entwicklungsumgebung	Die Entwicklungsumgebung ist per Konsole bedienbar. Grundlegende Kenntnisse in der Bedienung von Konsolen reichen aus. Die nötigen Befehle sind kurz und prägnant.

Installationsdauer	Testresultat
Installationsdauer der Entwicklungsumgebung	Resultat der Zeitmessung: 6 Minuten
Installationsdauer aller benötigten Zusatztools für die Entwicklung auf den geforderten Plattformen	Resultat der Zeitmessung: 16 Minuten. Steroids benötigt zusätzlich zur eigenen SDK die komplette PhoneGap Entwicklungsumgebung.
Dauer zur Erstellung einer lauffähigen HelloWorld Applikation	Resultat der Zeitmessung: 11 Minuten. Das konsolenbasierte Steroids SDK bietet die automatische Generierung einer HelloWorld-Applikation.

Speichereffizienz	Testresultat
Dateigrösse des installierbaren APK (Android Package File)	3.8 MB
Speicherplatzbelegung nach dem Starten des Miniprototypen	8.51 MB

Fluidität der Benutzeroberfläche	Testresultat
Werden lange Listen beim Scrollen dargestellt, ohne dass nachgeladen werden muss?	Das Scrolling funktionierte beim entwickelten Miniprototypen selbst bei langen Listen ohne erkennbares Flackern oder Stocken. Die Fluidität des Steroids-Miniprototypen war in kleinster Weise zu beanstanden.
Werden Animationen beim Wechseln eines Tabs angezeigt, ohne dass ein Ruckeln feststellbar ist?	Bei Steroids werden keine Animationen beim Wechseln eines Tabs angezeigt.

Performance	Testresultat
Zeitmessung der Startzeit der App	Resultate der Messungen: • 3.81 Sekunden • 3.93 Sekunden • 3.95 Sekunden Durchschnitt: 3.9 Sekunden

11.0.3 Xamarin

Dieser Unterabschnitt stellt die Testresultate von Xamarin vor.

Portabilität	Testresultat
Kompatibilität mit den folgenden Plattformen: • iOS (Version 7 oder neuer) • Android (Version 4.2.2 oder neuer) • Windows Phone (Version 7.8 oder neuer)	Xamarin unterstützt die geforderten Plattformen.
Zugriff auf die folgenden Betriebssystem-Funktionalitäten: • Push-Benachrichtigungen • GPS Lokalisierung • Volumenregler • 3-Achsen-Gyrosensor • Gespeicherte Kontakte	Zugriff auf aufgelistete Ressourcen ist ohne weitere Plugins möglich. Für jede Plattform muss der Zugriff auf die Ressourcen einzeln implementiert werden.
Darstellung der grafischen Benutzeroberfläche in nativem Design	Die Benutzeroberfläche wird im nativen GUI-Designer entwickelt. Muss deshalb auch für jede Plattform einzeln implementiert werden.
Aufwand für Plattformwechsel (Vendor Lock-in)	Die Applikationslogik kann in der Windows-Umgebung wiederverwendet werden. Die Views existieren im Xamarin in der jeweiligen nativen Markup Language, somit können sie vereinfacht in eine native Applikation portiert werden.

Security	Testresultat
Wird HTTPS unterstützt?	Ja, HTTPS wird durch Xamarin unterstützt.

¹³<http://www.gnu.org/licenses/lgpl-2.1.html> aufgerufen am 05. April, 2011, 14:02 Uhr

Community	Testresultat
Hilfestellungen in Foren	Sehr aktives und umfassendes Forum angeboten von Xamarin. Xamarin-Angestellte unterstützten Entwickler als Forum-Administratoren bei Problemstellungen.

Erlernbarkeit und Dokumentation	Testresultat
Erlernbarkeit	Auf der Webseite sind Plattform-spezifische Anleitungen für alle angebotenen Plattformen zu finden. Anhand der Tutorials erhält man einen schnellen Einstieg in die Technologie.
Dokumentation	Xamarin stellt für jede Plattform eine umfangreiche Dokumentation zur Verfügung. Jedoch werden nur die wichtigsten Fälle abgedeckt. Während der Entwicklung des Prototyps sind Probleme aufgetaucht, zu welchen noch keine Lösungsansätze über Suchmaschinen auffindbar sind. Abhilfe leistet die Zuhilfenahme der nativen Dokumentation (z.B. Android API Dokumentation), jedoch muss die Lösung auf gut Glück in das Xamarin übertragen werden.

Lizenzbedingungen	Testresultat
Welcher Lizenz untersteht die kommerzielle Nutzung der Technologie?	Die Community Version untersteht der LGPLv2 Lizenz ¹³ . Alternativ können kommerzielle Versionen gekauft werden, um die Nachteile der LGPLv2 zu umgehen.
Wie hoch sind die Entwicklungsgebühren?	Zur kommerziellen Nutzung von Xamarin wird vom Hersteller die Business Lizenz empfohlen. Die Lizenzgebühren betragen pro Entwickler pro Jahr 999 US-Dollar.

Wartbarkeit	Testresultat
Qualität von erhaltenen Fehlermeldungen	Fehler, die beim Build-Prozess auftreten, werden in der Konsole angezeigt. Die angezeigten Fehler sind sehr ausführlich. Die Fehlermeldungen werden ähnlich wie bei den Konkurrenzprodukten (vergleichbar mit Xcode, Eclipse oder Visual Studio) angezeigt.
Debugging Möglichkeiten	Debugging Möglichkeiten sind gleichzusetzen mit den nativen Entwicklungsumgebungen. Es können Break-points gesetzt werden, um während der Laufzeit das System anzuhalten und Untersuchungen durchzuführen.

Entwicklungsumgebung	Testresultat
Umfang an Hilfsfunktionen	Die wichtigsten Hilfsfunktionen: • Automatische Ausführung des Build-Prozesses für alle Plattformen • Starten von Plattformspezifischen Simulatoren • Erstellen eines Projektordners gemäss Vorlage • Integriertes Versionierungstool
Bedienbarkeit der Entwicklungsumgebung	Xamarin wird mit einer eigenen Entwicklungsumgebung Xamarin Studio (vergleichbar mit Xcode, Eclipse oder Visual Studio) geliefert. Die Software ist sehr benutzerfreundlich und enthält alle notwendigen Komponenten (Ausnahme Simulatoren), um entwickeln zu können. Ein besonderer Vorteil zu den anderen Technologien ist, dass der Entwickler ganz auf die Konsoleneingabe für das Konfigurationsmanagement verzichten kann.

Installationsdauer	Testresultat
Installationsdauer der Entwicklungsumgebung	Resultat der Zeitmessung: 35 Minuten.
Installationsdauer aller benötigten Zusatztools für die Entwicklung auf den geforderten Plattformen	Es werden die Plattform-spezifischen Simulatoren benötigt, welche üblicherweise mit den Plattform-spezifischen SDKs installiert werden. Die Installation der SDKs ist in der angegebenen Zeit nicht enthalten.
Dauer zur Erstellung einer lauffähigen HelloWorld Applikation	Resultat der Zeitmessung: 3 Minuten. Das Xamarin Studio bietet die automatische Generierung einer HelloWorld-Applikation.

Speichereffizienz	Testresultat
Dateigröße des installierbaren APK (Android Package File)	7.8 MB
Speicherplatzbelegung (in MB) nach dem Starten des Miniprototypen	7.72 MB

Fluidität der Benutzeroberfläche	Testresultat
Werden lange Listen beim Scrollen dargestellt, ohne dass nachgeladen werden muss?	Die Applikation wirkt sehr performant. Bei den Tests ergaben sich keine Beanstandungen betreffend der Fluidität.
Werden Animationen beim Wechseln eines Tabs angezeigt, ohne dass ein Ruckeln feststellbar ist?	Bei Tabs, die mit Xamarin entwickelt wurden, sind Übergangs-Animationen im Einsatz. Bei der Darstellung der Animationen war kein Ruckeln oder Stocken erkennbar.

Performance	Testresultat
Zeitmessung der Startzeit der App	Resultate der Messungen: • 0.67 Sekunden • 0.68 Sekunden • 0.7 Sekunden Durchschnitt: 0.68 Sekunden

11.0.4 jQuery Mobile

Dieser Unterabschnitt stellt die Testresultate von jQuery Mobile vor.

Portabilität	Testresultat
Kompatibilität mit den folgenden Plattformen: • iOS (Version 7 oder neuer) • Android (Version 4.2.2 oder neuer) • Windows Phone (Version 8 oder neuer)	jQuery Mobile unterstützt die geforderten Plattformen.
Zugriff auf die folgenden Betriebssystem-Funktionalitäten: • Push-Benachrichtigungen • GPS Lokalisierung • Volumenregler • 3-Achsen-Gyrosensor • Gespeicherte Kontakte	jQuery Mobile bietet keinen Zugriff auf die aufgelisteten Ressourcen.
Darstellung der grafischen Benutzeroberfläche in nativem Design	jQuery Mobile hat ein eigenes Design. Es unterstützt somit die native Darstellung nicht.
Aufwand für Plattformwechsel (Vendor Lock-in)	Teile der View und des Controllers müssten bei einem Technologiewechsel angepasst werden. Jedoch können die anderen Teile direkt wiederverwendet werden.

Security	Testresultat
Wird HTTPS unterstützt?	Ja, dadurch dass JQuery Mobile in einem Browser ausgeführt wird, ist die HTTPS Unterstützung gewährleistet.

Community	Testresultat
Hilfestellungen in Foren	Sehr aktives und umfassendes Forum angeboten von jQuery Mobile. Eine starke Open Source Community bietet Hilfe im Forum.

Erlernbarkeit und Dokumentation	Testresultat
Erlernbarkeit	Auf der Webseite sind Anleitungen zur Installation und ersten lauffähigen Applikation zu finden. Anhand der Tutorials erhält man einen sehr schnellen Einstieg in die Technologie.
Dokumentation	jQuery Mobile stellt eine umfangreiche Dokumentation und Anleitungen zur API zur Verfügung.

Lizenzbedingungen	Testresultat
Welcher Lizenz untersteht die kommerzielle Nutzung der Technologie?	jQuery Mobile untersteht der MIT Licence ¹⁴ .
Wie hoch sind die Entwicklungsgebühren?	Es fallen keine lizenzbedingten Gebühren an.

Wartbarkeit	Testresultat
Die Qualität der erhaltenen Fehlermeldungen	Fehlermeldungen hängen massgeblich von der im Entwicklungsprozess verwendeten Browser ab. Die Fehlerbehandlung von jQuery Mobile ist sehr aufschlussreich und meldet dem Entwickler potentielle Problemzonen.
Debugging Möglichkeiten	Debugging Möglichkeiten variieren je nach verwendetem Browser während dem Entwicklungsprozess. Es können Breakpoints gesetzt werden, um während der Laufzeit das System anzuhalten und Untersuchungen durchzuführen.

Entwicklungsumgebung	Testresultat
Umfang an Hilfsfunktionen	jQuery Mobile bietet keine Hilfsfunktionen an. Es wird lediglich das Framework angeboten.
Bedienbarkeit der Entwicklungsumgebung	jQuery Mobile wird ohne Entwicklungsumgebung geliefert.

¹⁴http://en.wikipedia.org/wiki/MIT_License aufgerufen am 11. April, 2014, 14:51 Uhr

Installationsdauer	Testresultat
Installationsdauer der Entwicklungsumgebung	Resultat der Zeitmessung: Messung nicht möglich.
Installationsdauer aller benötigten Zusatztools für die Entwicklung auf den geforderten Plattformen	Es ist nur ein Texteditor und ein Browser zur Entwicklung notwendig. Der Texteditor und der Browser sind üblicherweise auf den Entwicklungsrechner ohne zusätzliche Installation bereits vorhanden.
Dauer zur Erstellung einer lauffähigen HelloWorld Applikation	Resultat der Zeitmessung: 5 Minuten. Vorgefertigte Templates können über die Suchmaschine gefunden werden und kopiert werden.

Speichereffizienz	Testresultat
Dateigrösse der Software	2.8 MB
Speicherplatzbelegung (in MB) nach dem Starten des Miniprototypen	3.53 MB

Fluidität der Benutzeroberfläche	Testresultat
Werden lange Listen beim Scrollen dargestellt, ohne dass nachgeladen werden muss?	Die Applikation wirkt sehr performant. Bei den Tests ergaben sich keine Beanstandungen betreffend der Fluidität.
Werden Animationen beim Wechseln eines Tabs angezeigt, ohne dass ein Ruckeln feststellbar ist?	Bei Tabs, die mit jQuery Mobile entwickelt wurden, sind standardmässig keine Übergangs-Animationen im Einsatz.

Performance	Testresultat
Zeitmessung der Startzeit der App	Eine Zeitmessung ist im Fall jQuery Mobile nicht sinnvoll, da der grösste Zeitverlust bei der Datenübertragung stattfindet.

11.0.5 openUI5

Dieser Unterabschnitt stellt die Testresultate von OpenUI5 vor.

Portabilität	Testresultat
Kompatibilität mit den folgenden Plattformen: • iOS (Version 7 oder neuer) • Android (Version 4.2.2 oder neuer) • Windows Phone (Version 8 oder neuer)	OpenUI5 unterstützt die geforderten Plattformen.
Zugriff auf die folgenden Betriebssystem-Funktionalitäten: • Push-Benachrichtigungen • GPS Lokalisierung • Volumenregler • 3-Achsen-Gyrosensor • Gespeicherte Kontakte	OpenUI5 bieten keinen Zugriff auf die aufgelisteten Ressourcen.
Darstellung der grafischen Benutzeroberfläche in nativem Design	OpenUI5 hat ein eigenes Design. Es unterstützt somit die native Darstellung nicht.
Aufwand für Plattformwechsel (Vendor Lock-in)	Ausschliesslich die Logik kann wiederverwendet werden. Alle anderen Bereiche müssten bei einem Technologiewechsel angepasst werden.

Security	Testresultat
Wird HTTPS unterstützt?	Ja, dadurch dass openUI5 im Browser ausgeführt wird, ist die HTTPS-Unterstützung gewährleistet.

¹⁴<http://www.apache.org/licenses/LICENSE-2.0.html> aufgerufen am 11. April, 2014, 18:37 Uhr

Community	Testresultat
Hilfestellungen in Foren	SAP bietet regelmässig erscheinende Beiträge mit Hilfestellungen in den eigenen Foren an. Während der Entwicklung ist aufgefallen, dass noch lange nicht alle Problemstellen auf den Foren beantwortet sind. Während der Entwicklung sind Fälle aufgetaucht, zu welchen eine Problemlösung fehlte.

Erlernbarkeit und Dokumentation	Testresultat
Erlernbarkeit	Auf der Webseite sind ausführliche Anleitungen für das Framework vorhanden. Jedoch durch die Komplexität und Grösse des Frameworks dauert der Einstieg in die Technologie sehr lange. So nahm beispielsweise die Entwicklung eines Miniprototypen fast einen ganzen Arbeitstag in Anspruch.
Dokumentation	SAP stellt auf der eigenen Homepage ausführliche Dokumentation zur Verfügung. Das OpenUI5 biete verschiedene Varianten (HTMLView, JavaScriptView, JSONView und XMLView) an, um die View zu gestalten. Für Webentwickler ist es sehr naheliegend, die HTMLView zu verwenden, jedoch wird in der Dokumentation die HTMLView nur einführend erklärt. Die restliche Dokumentation verwendet für ihre Beispiele die JavaScriptView und erschwert dem Webentwickler, die Inhalte zu verstehen.

Lizenzbedingungen	Testresultat
Welcher Lizenz untersteht die kommerzielle Nutzung der Technologie?	Das Framework OpenUI5 untersteht der Apache License 2.0 ¹⁴ .
Wie hoch sind die Entwicklungsgebühren?	Es fallen keine Lizenzgebühren an.

Wartbarkeit	Testresultat
Qualität von erhaltenen Fehlermeldungen	Fehlermeldungen hängen massgeblich von den im Entwicklungsprozess verwendeten Browser ab. Die Fehlerbehandlung von OpenUI5 ist sehr aufschlussreich und meldet dem Entwickler potentielle Problemzonen.
Debugging Möglichkeiten	Debugging Möglichkeiten variieren je nach verwendetem Browser. Es können Breakpoints gesetzt werden, um während der Laufzeit das System anzuhalten und Untersuchungen durchzuführen.

Entwicklungsumgebung	Testresultat
Umfang an Hilfsfunktionen	Die wichtigsten Hilfsfunktionen: • Starten von Simulatoren • Erstellen eines Projektordners gemäss Vorlage • Code Completion des OpenUI5-Frameworks
Bedienbarkeit der Entwicklungsumgebung	OpenUI5 wird mit einem Plugin für Eclipse Studio geliefert. Das Plugin erweitert das frei erhältliche Eclipse Studio und sorgt, dass die Entwicklung mit OpenUI5 vereinfacht wird. Durch spezielle Features wie Code Completion in der JavaScript-Umgebung wird das Entwickeln massiv erleichtert.

Installationsdauer	Testresultat
Installationsdauer aller benötigten Zusatztools für die Entwicklung auf den geforderten Plattformen	Zur Entwicklung wird das Eclipse Studio benötigt. Dessen Installation beträgt 5 Minuten.
Dauer zur Erstellung einer lauffähigen HelloWorld Applikation	Resultat der Zeitmessung: 5 Minuten. Das OpenUI5-Plugin für Eclipse Studio bietet die automatische Generierung einer HelloWorld-Applikation.

Speichereffizienz	Testresultat
Dateigrösse der Software	8.9 MB
Speicherplatzbelegung (in MB) nach dem Starten des Miniprototypen	10.2 MB

Fluidität der Benutzeroberfläche	Testresultat
Werden lange Listen beim Scrollen dargestellt, ohne dass nachgeladen werden muss?	Die Applikation wirkt sehr performant. Bei den Tests ergaben sich keine Beanstandungen betreffend der Fluidität.
Werden Animationen beim Wechseln eines Tabs angezeigt, ohne dass ein Ruckeln feststellbar ist?	Bei Tabs, die mit OpenUI5 entwickelt wurden, sind Übergangs-Animationen im Einsatz. Bei der Darstellung der Animationen war kein Ruckeln oder Stocken erkennbar.

Performance	Testresultat
Zeitmessung der Startzeit der App	Eine Zeitmessung ist im Fall OpenUI5 nicht sinnvoll, da der grösste Zeitverlust bei der Datenübertragung stattfindet.

12 Benutzerschnittstelle und Usability

In diesem Kapitel wird der Entwicklungsprozess der grafischen Benutzerschnittstelle näher vorgestellt.

Im ersten Abschnitt wird der Entwurfsprozess des Prototyps vorgestellt. Anschliessend wird der Prototyp getestet. Mit den Erkenntnissen aus dem Testing wird der Prototyp angepasst und vorgestellt.

12.1 Entwurf eines Prototyps

In diesem Abschnitt werden die durchlaufenen Phasen für den ersten Prototyp der Benutzerschnittstelle vorgestellt. Zuerst werden die Anforderungen an die Benutzerschnittstelle erhoben. Zweitens wird das Layout der Applikation in groben Zügen skizziert. Letztens werden die Skizzen mit Layoutcode umgesetzt, um eine konkretere Vorstellung des Endprodukts zu erhalten.

12.1.1 Anforderungen

Der Prototyp berücksichtigt die sieben Use Cases gemäss Kapitel Anforderungen. In der folgenden Tabelle (zu finden im Anhang-Kapitel 10) ist ersichtlich, in welcher Zeit die Use Cases erlernbar sein sollten:

Use Case	Manager-Zeit	Mitarbeiter-Zeit
UC 01: App öffnen	<10s	<10s
UC 02: Anmelden	<60s	<60s
UC 03: Abmelden	<15s	<15s
UC 04: Workspace wechseln	<15s	<15s
UC 05: Aufgabenfortschritt kontrollieren	<180s	<180s
UC 06: Neue Mitteilung erfassen	<60s	<60s
UC 07: Mitteilungen ansehen	<120s	<120s

Tabelle 12.1: Erlernbarkeit von Use Cases

12.1.2 Paperprototyping

In diesem Unterabschnitt wird der Paperprototyp vorgestellt. Anhand der Anforderungen werden einfache Skizzen der einzelnen Seiten der Applikation mittels Stift und Papier erstellt. Jede folgend dargestellte Skizze ist das Resultat eines Workshops des Entwicklerteams.

Messages

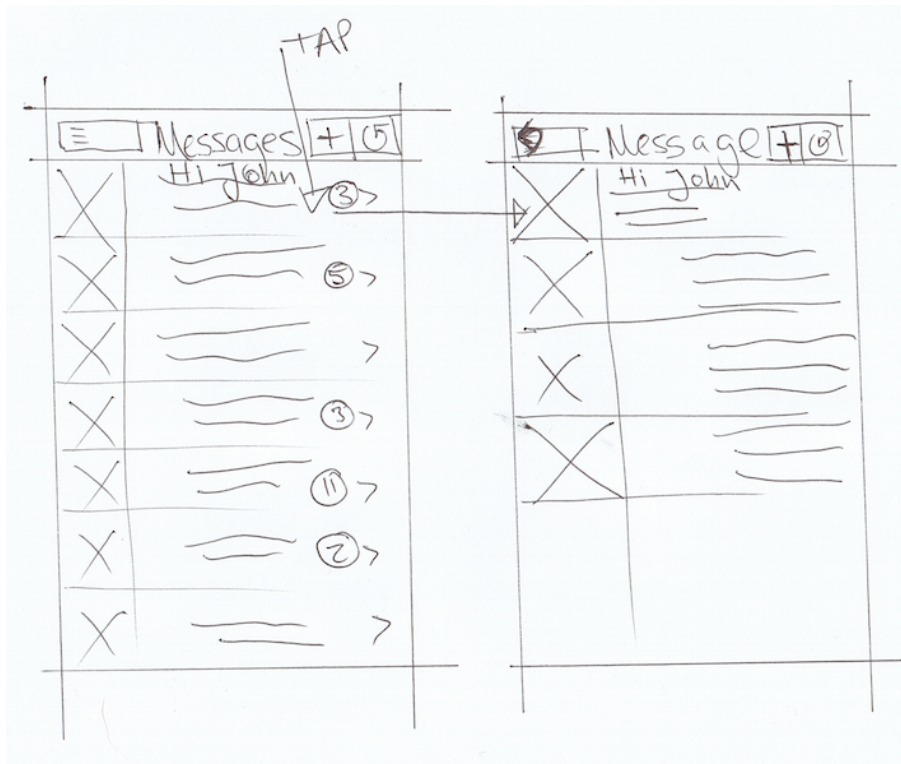


Abbildung 12.1: Skizze: Mitteilungen

Die obige Darstellung berücksichtigt folgende Use Cases:

- UC 06: Neue Mitteilung erfassen
- UC 07: Mitteilungen ansehen

Im linken Teil der Darstellung sieht der Benutzer eine Übersicht aller Mitteilungen. Neue Mitteilungen können über das Plus-Icon erfasst werden. Wählt der Benutzer eine Mitteilung in der List an, so werden die Details zur Mitteilung inklusive Kommentare geladen (rechter Teil).

Tasks

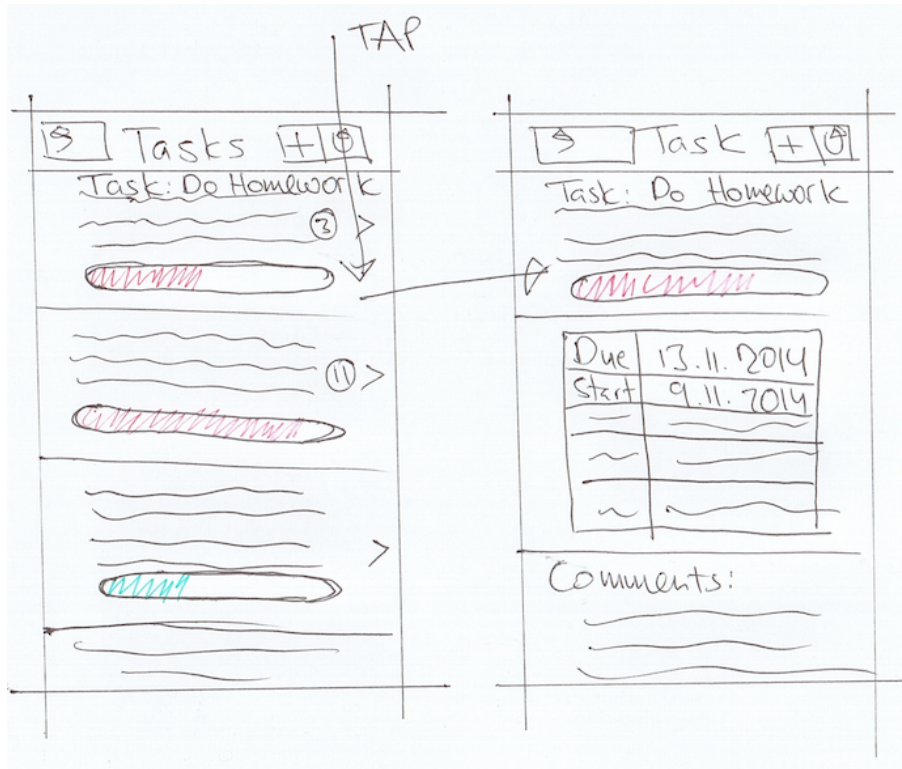


Abbildung 12.2: Skizze: Aufgaben

Die obige Darstellung berücksichtigt folgende Use Cases:

- UC 05: Aufgabenfortschritt kontrollieren

Im linken Teil der Darstellung sieht der Benutzer eine Übersicht aller Aufgaben. Neue Aufgaben können über das Plus-Icon erfasst werden. Fortschritt der einzelnen Aufgaben ist über den Fortschrittsbalken in der Übersicht ersichtlich. Wählt der Benutzer eine Aufgabe in der Liste an, so werden die Details zur Aufgabe inklusive Kommentare geladen (rechter Teil).

Navigation

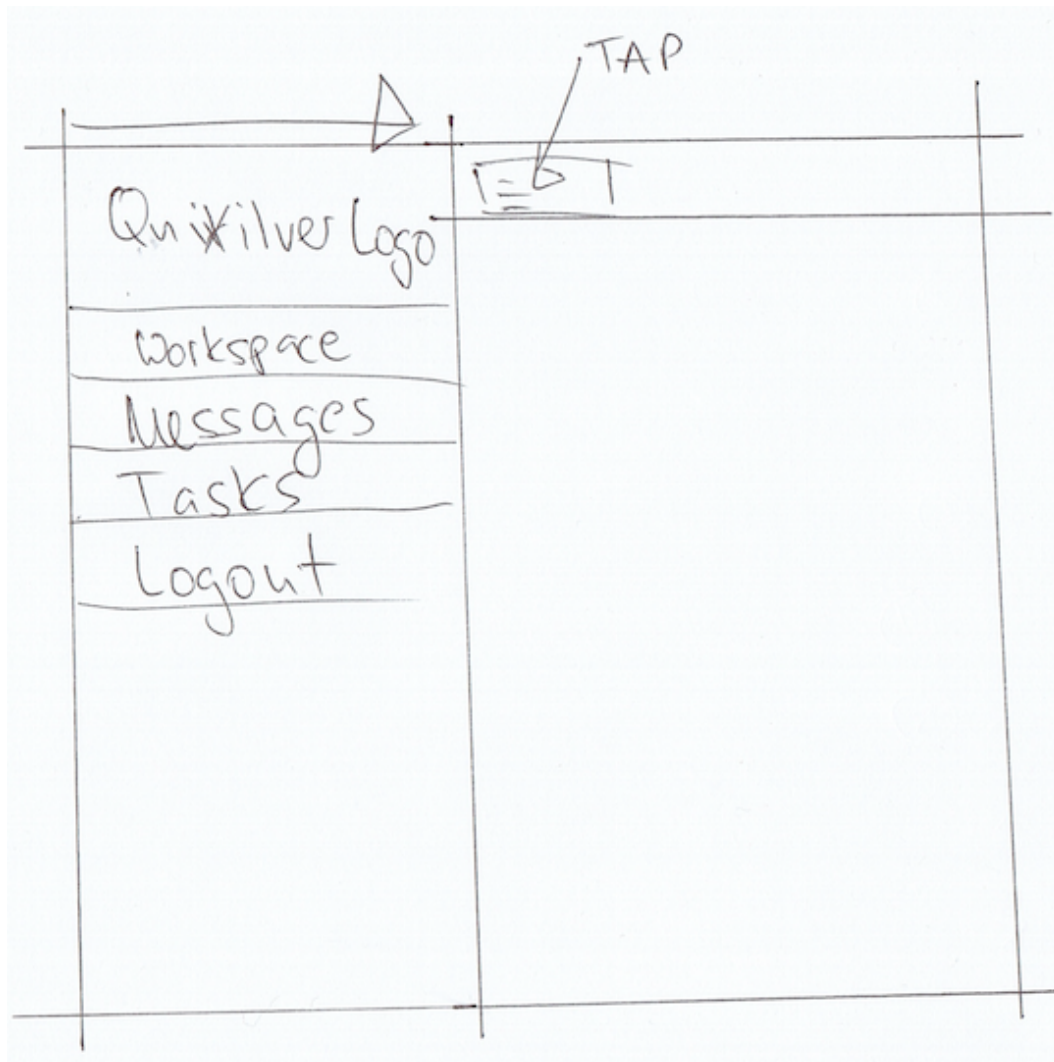


Abbildung 12.3: Skizze: Navigation

Die obige Darstellung berücksichtigt folgende Use Cases:

- UC 03: Abmelden
- UC 04: Workspace wechseln

Von den Seiten Mitteilungen und Aufgaben kann die Navigation mit dem oben links platzierten Menu-Icon geöffnet werden. Das Navigationsmenü öffnet sich von der linken Seite. Um den Workspace zu wechseln, wählt der Benutzer das Listenelement Workspace an. Um sich abzumelden wählt der Benutzer das Listenelement Logout an.

Login



The image shows a login form for 'QUIXILVER'. At the top, the logo 'QUIXILVER' is displayed in blue and grey, with the tagline 'Collaborative Task Management for Business' below it. Below the logo, the text 'Sign In' is written in blue. A prompt 'Please enter your user name and password.' follows. The form itself is a light blue box with a 'Credentials' header. Inside, there are two input fields: 'User Name' and 'Password'. Below these fields is a green 'Sign in' button. At the bottom of the form, there are two links: 'Forgot your password? Reset Password' and 'Don't have an account? Sign up for free', both in green text.

Abbildung 12.4: Vorgabe: Loginformular

Die obige Darstellung berücksichtigt folgende Use Cases:

- UC 02: Anmelden

Auf Grund von Sicherheitseinschränkungen muss die Login-Seite von QuiXilver Browser Client verwendet werden. Dadurch ist das Design und Layout zur Anmeldemaske vorgegeben und kann nicht angepasst werden.

Der Use Case UC 01: App öffnen wird im Rahmen des Paperprototyping nicht dargestellt, da das Layout nicht betroffen ist.

12.1.3 Benutzerschnittstelle: Erster Entwurf

Im Anschluss zum Paperprototype wird das Layout und Design in Layoutcode umgesetzt.

Verwendete Technologien Durch den Architekturentscheid (siehe Unterabschnitt 5.3.3) ist vorgehend festgelegt worden, die aktuellste Ausgabe der Benutzerschnittstellen-Library JQuery Mobile (Version 1.4.2) zu verwenden. Die Library stellt vorgefertigte, grafisch aufgewertete Layoutkomponenten zur Verfügung (z.B. Knöpfe, Navigationsleisten oder Listen).

Umsetzung mit JQuery Mobile Bei der Umsetzung der Skizzen besteht die Arbeit aus folgenden Punkten:

- Passende Layoutkomponenten in Library-Dokumentation suchen
- Fehlende Komponenten mit HTML und CSS erstellen
- Zusammensetzen der Komponenten
- Korrekturen für einheitliches Layout

Resultat: Erster Entwurf Nachfolgende Abbildungen zeigen die Resultate des ersten Entwurfs. Sie dienen als Basis für das weitere Vorgehen.

12.1 Entwurf eines Prototyps

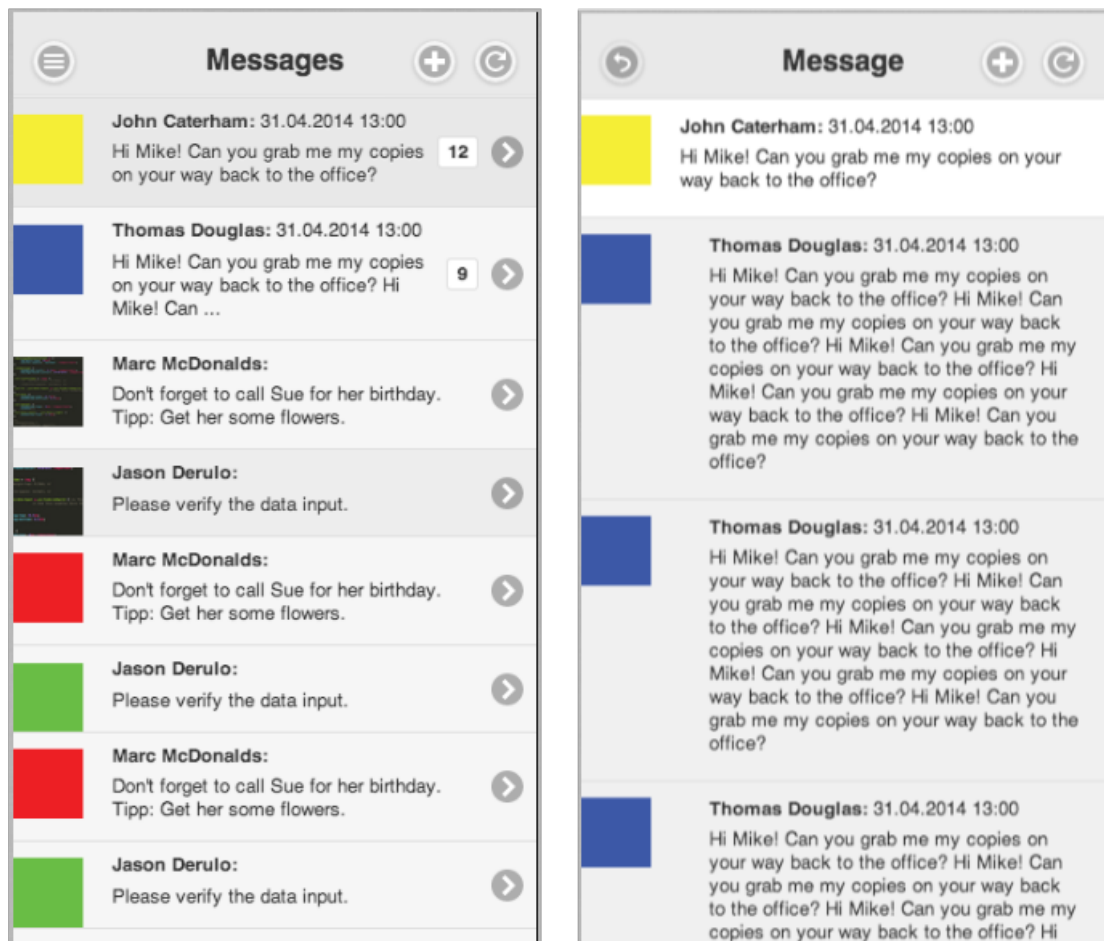


Abbildung 12.5: Erster Entwurf: Messages

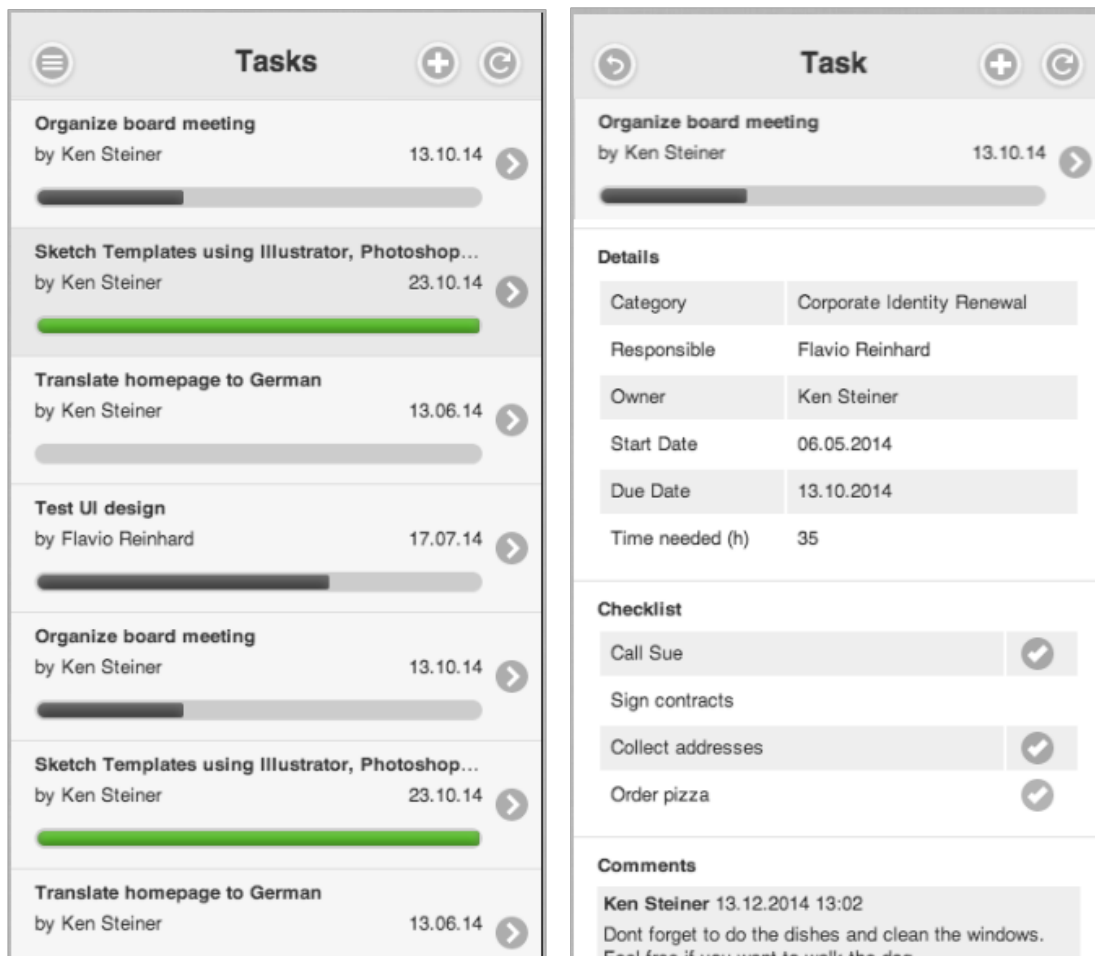


Abbildung 12.6: Erster Entwurf: Tasks

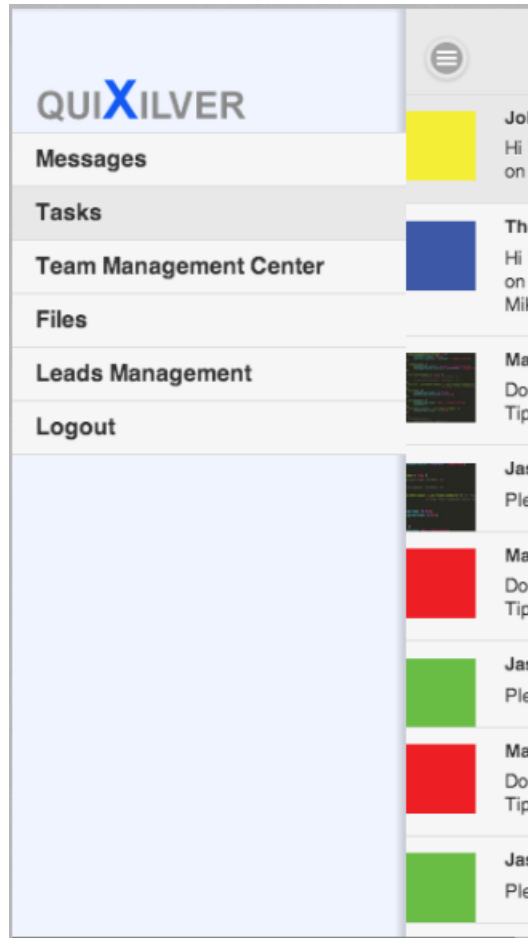


Abbildung 12.7: Erster Entwurf: Navigation

Umsetzungbericht Bei der Umsetzung des Layouts wurden die folgenden Erkenntnisse gewonnen:

- JQuery Mobile Komponenten sind Cross-Plattform kompatibel
- Entwickeln von eigenen Komponenten ist komplex, da viele Tag-Wörter bereits von JQuery Mobile reserviert sind und schnell zu Überschneidungen führen

Bei der Umsetzung des Layouts stachen folgende Usability-Punkte heraus:

- Knöpfe und Texte so gross wie möglich gestalten

12.2 Usability Testing

Um die Benutzerfreundlichkeit des ersten Prototyps zu ermitteln, wird der erste Entwurf gemeinsam mit den Stakeholdern besprochen. Anschliessend zur Besprechung wird ein Cognitive Walkthrough ausgeführt. Danach werden Usability Tests mit projektfremden Personen durchgeführt. Das Ziel dieses Abschnitts ist es, die Problemstellen des Entwurfs zu erkennen, um sie ausbessern zu können.

12.2.1 Analyse mit Stakeholdern

Sitzung mit externem Partner, 10.05.14 Beteiligte:

- Stephan Haller (Vigience AG)
- Flavio Reinhard (Projektteam QuiXilver)
- Ken Steiner (Projektteam QuiXilver)

Folgende Verbesserungsvorschläge sind an der ausserordentlichen Sitzung erfasst worden:

- Gleiche Farben des Aufgaben-Status wie im QuiXilver Online Clients verwenden
- Aufgabenfortschritt nicht in Form eines prozentualen Fortschrittsbalkens darstellen

Folgende Verbesserungsvorschläge sind an der wöchentlichen Sitzung vom 13. Mai 2014 mit den Projektbetreuern erfasst worden:

- Aktueller Workspace soll jederzeit ersichtlich sein
- Eigenes Feld für die Statusfarbe einzelner Aufgaben verschwendet zu viel Platz, deshalb sollten die Aufgaben nach Kategorie sortiert und nur einmal mit dem Farbcode versehen werden
- Begleiticon zu dem Button Workspace wechseln ist irreführend. Die anderen Buttons in der gleichen Auflistung haben keine Icons. Somit muss das Icon für Workspace wechseln entfernt werden

12.3 Cognitive Walkthrough

- Klickbereich der Listenelemente in der Navigation muss vergrößert werden
- Listenelemente in der Navigation nach Navigationselementen und Einstellungselementen visuell trennen
- X-Button oben rechts, um das Workspace-Popup zu schliessen, ist in mobilen Benutzerschnittstellen verboten. Regulärer Button am Fuss des Popups einbauen.
- Darstellungsfehler bezüglich verschiedenen Hintergrundfarben beheben

Beim nächsten Projekt zu berücksichtigen:

- Benutzerschnittstelle muss den OS-spezifischen Eigenheiten angepasst werden. Beispiel: Back-Button bei iOS ist oben links, bei Android ist der native Back-Button zu verwenden

12.3 Cognitive Walkthrough

In diesem Unterabschnitt wird der Entwurf mit Usability-Inspektionsmethode Cognitive Walkthrough getestet. Beim Cognitive Walkthrough versetzt sich der Inspektor in die Rolle eines hypothetischen Benutzers und analysiert konkrete vorgegebene Handlungsabläufe. Dabei geht er davon aus, dass der Anwender den Weg des geringsten kognitiven Aufwands gehen wird.

12.3.1 Definition der Handlungssequenzen

Die Projektmitarbeiter und Projektmanager werden in der Untersuchung als den gleichen Akteur Benutzer gehandhabt. Der Benutzer muss als Handlungssequenzen die sieben Use Cases durchlaufen.

12.3.2 Untersuchung der Handlungssequenzen

Bei der Untersuchung werden folgende Fragen in der aufgelisteten Reihenfolge beantwortet:

- Wird der Benutzer den richtigen Effekt zu erzielen?
- Wird der Benutzer erkennen, wo die gewünschte Aktion zur Verfügung steht?
- Wird der Benutzer eine Verbindung herstellen zwischen der korrekten Aktion und dem gewünschten Effekt?
- Wenn die korrekte Aktion ausgeführt worden ist: Wird der Benutzer den Fortschritt erkennen und Rückmeldung erhalten?

Tabelle 12.2: Handlungssequenzen des Cognitive Walkthrough

Handlungssequenz	Untersuchung
UC 01: App öffnen	Der Benutzer ist mit dem Effekt, eine App zu öffnen, vertraut. Er erkennt, dass Icons auf dem Home-Screen Aktionen sind. Der Benutzer stellt die Verbindung her, dass mit dem Klick auf ein Icon eine App geöffnet wird. Der Benutzer merkt anhand der visuell vollständig geladenen App, dass die Aktion fertig durchgeführt ist.
UC 02: Anmelden	Der Benutzer ist mit dem Effekt, in einer App angemeldet zu sein, vertraut. Der Benutzer erkennt die Aktion, da das Anmeldeformular aus den Feldern Benutzername, Passwort und dem <i>Anmelden</i> -Button besteht. Der Benutzer kann die Verbindung herstellen, dass mit dem Klick auf den <i>Login</i> -Button er angemeldet sein wird. Bei der korrekten Ausführung erkennt der Benutzer, dass seine persönlichen Nachrichten geladen sind.
UC 03: Abmelden	Der Benutzer ist mit dem Effekt, sich von einer App abzumelden, vertraut. Er erkennt die Aktion, da sich <i>Abmelden</i> -Button gut ersichtlich in der <i>Navigation</i> positioniert ist. Der Benutzer stellt die Verbindung her, dass der Klick auf den <i>Abmelden</i> -Button ihn abmelden wird. Indem er anschliessend die <i>Anmelden</i> -Seite zu Sicht bekommt, wird er merken, dass seine Aktion erfolgreich durchgeführt wurde.
UC 04: Workspace wechseln	Der Benutzer ist gefährdet, dass er das Konzept der Workspaces nicht versteht und somit den Effekt den Workspace zu wechseln nicht ganz versteht. Er erkennt die Aktion problemlos, da der <i>Workspace wechseln</i> -Button klar beschriftet ist. Dem Benutzer wird ersichtlich, dass die Aktion erfolgreich war, indem die Inhalte des ausgewählten Workspaces angezeigt werden.
UC 05: Aufgabenfortschritt kontrollieren	Der Benutzer ist mit dem Effekt der Kontrolle vom Aufgabenfortschritt vertraut. Er erkennt, dass die Aktion den <i>Aufgaben</i> -Button zu klicken, eine Aktion ausführt. Er stellt die Verbindung her, dass der Klick auf den <i>Aufgaben</i> -Button die Aufgabenübersicht öffnet und den Status der Aufgaben überprüft werden können. Der Benutzer erkennt den Fortschritt indem die richtigen Aufgaben geladen sind.

12.3 Cognitive Walkthrough

Handlungssequenz	Untersuchung
UC 06: Neue Mitteilung erfassen	Der Benutzer ist mit dem Effekt eine neue Nachricht zu erfassen vertraut. Er erkennt, dass das Klicken auf das <i>Sprechblase</i> -Icon eine Aktion ausführt. Er stellt die Verbindung her, dass er mit dem Klick auf das <i>Sprechblase</i> -Icon eine neue Nachricht erfassen kann. Der Benutzer erkennt den Fortschritt, indem er wieder zurück in der Mitteilungsübersicht ist und seine eben erfasste Nachricht in der Liste ersichtlich ist.
UC 07: Mitteilungen ansehen	Der Benutzer ist mit dem Effekt Mitteilungen anzusehen vertraut. Er erkennt die Aktion problemlos, da in der Navigation ein <i>Mitteilungen</i> -Button ersichtlich ist. Der Benutzer ist damit vertraut, dass der Klick auf den <i>Mitteilungen</i> -Button die Mitteilungsübersicht öffnet. Er erkennt den Fortschritt indem er in der Mitteilungsübersicht eine Liste von Mitteilungen sieht.

12.3.3 Testplan

Instruktion Das nachfolgende Testing mit realen Personen unterstützt die Entwicklung einer effizienten und zielführenden Benutzeroberfläche. Der Test hält die Problemstellen fest, welche Testpersonen bei der Zielverfolgung begegnet sind. Durch die zusammengetragenen Beobachtungen können Problemstellen gezielt eruiert und behoben werden.

Testdurchführung Der Test wird auf einem Android Version 4.2.2, zur Verfügung gestellt durch das Entwicklerteam, durchgeführt. Für eine Testdurchführung ist eine einzige Testperson notwendig. Das Entwicklerteam misst die benötigte Zeit zur Erfüllung jeder Aufgabe. Parallel dazu notiert das Team Auffälligkeiten und Probleme der Testperson.

Testbedingungen Die folgend erwähnten Aufgaben sind der Reihenfolge nach zu lösen. Der Testleiter wird durch die Aufgaben führen und bei allfälligen Verständnisfragen zur Verfügung stehen.

Teilen Sie Ihre Gedankenschritte laufend dem Testleiter mit. Somit können die Gedankenschritte möglichst gut nachvollzogen werden und geben Anhaltspunkte wo Probleme oder Missverständnis auftreten und Handlungsbedarf besteht.

Der Testleiter wird die Gedankenschritte sowie seine Benutzerbeobachtungen notieren. Unsicherheiten oder lange Denkpausen im Ablauf geben Aufschluss über schwer verständliche Teile im Prozess.

12.3.4 Aufgaben

Kontext Sie sind Mitarbeiter der FutureVision GmbH. Sie haben aufgrund ihrer geschäftlichen Reisetätigkeiten soeben ein Smartphone erhalten.

Nun möchte Ihr Arbeitgeber, dass sie auch während den Geschäftsreisen besser in der Firma integriert sind und somit das Tool QuiXilver verwenden.

Aufgaben-Nr.	Use Case	Aufgabenstellung
A1	UC 01: App öffnen	Starten Sie die App <i>QuiXilver</i>
A2	UC 02: Anmelden	Melden Sie sich als <i>ksteiner@hsr.ch</i> an.
A3	UC 03: Abmelden	Melden Sie sich ab.
A4	UC 04: Workspace wechseln	Wechseln Sie in den Workspace <i>Webdesign</i> .
A5	UC 05: Aufgabenfortschritt kontrollieren	Finden Sie den aktuellen Status zum Task <i>Adressen bereinigen</i> heraus.
A6	UC 07: Mitteilungen ansehen	Lesen Sie die als letztes erhaltene Message von Herrn Flavio Reinhard.

Tabelle 12.3: Aufgabenstellung an den Testbenutzer

12.3 Cognitive Walkthrough

12.3.5 Testlog

Dieser Unterabschnitt präsentiert die Testresultate der drei Usability-Tests mit projektfremden Personen.

Testperson: Pius Steiner

Name	Steiner
Vorname	Pius
Alter	62
PC Kenntnisse (Selbsteinstufung)	Täglicher Anwender
Tätigkeit	Sales

Tabelle 12.4: Personendaten: Kandidat Pius Steiner

Aufgabe	SOLL-Dauer [s]	IST-Dauer [s]	Probleme
A1	10	9	keine
A2	60	37	keine
A3	15	132	Hilfe mit Navigationsleiste notwendig
A4	15	99	Erneute Hilfe mit Navigationsleiste notwendig
A5	180	12	keine
A6	60	21	keine
A7	120	3	keine

Tabelle 12.5: Testresultate: Kandidat Pius Steiner

Testperson: Martina Staub

Name	Staub
Vorname	Martina
Alter	24
PC Kenntnisse (Selbsteinstufung)	Informatikstudium
Tätigkeit	Software Entwicklung

Tabelle 12.6: Personendaten: Kandidatin Martina Staub

Aufgabe	SOLL-Dauer [s]	IST-Dauer [s]	Probleme
A1	10	6	keine
A2	60	65	Passwort einmal falsch eingegeben
A3	15	14	keine
A4	15	10	keine
A5	180	25	keine
A6	60	32	keine

Tabelle 12.7: Testresultate: Kandidatin Martina Staub

12.3 Cognitive Walkthrough

Testperson: Kristian Dujmovic

Name	Dujmovic
Vorname	Kristijan
Alter	25
PC Kenntnisse (Selbsteinstufung)	Täglicher Anwender
Tätigkeit	Kalkulation (Offertenerstellung)

Tabelle 12.8: Personendaten: Kandidat Kristijan Dujmovic

Aufgabe	SOLL-Dauer [s]	IST-Dauer [s]	Probleme
A1	10	7	keine
A2	60	24	keine
A3	15	40	Lange Zeit auf dem Bildschirm den korrekten Folgeschritt gesucht. Konnte Menu selbständig öffnen und abmelden
A4	15	35	Nicht gewusst was ein Workspace ist. Hat schnell gelernt, dass das Menu für spezielle Tätigkeiten anzuwenden ist
A5	180	89	keine
A6	60	9	keine

Tabelle 12.9: Testresultate: Kandidat Kristijan Dujmovic

12.3.6 Erkenntnisse

Während den Tests mit den aussenstehenden Personen wurde die vorgegebene Dauer (SOLL-Dauer) bei drei Aufgabentypen überschritten.

Überschreitung A2: Passwort eingeben Die Zeitüberschreitung von der Kandidatin Martina Staub ist zu vernachlässigen, da sie das Passwort falsch eingegeben hat.

Überschreitung A3: Abmelden und A4: Workspace wechseln Die Zeitüberschreitung der beiden Kandidaten Pius Steiner und Kristijan Dujmovic sind auf das Unwissen der Testbenutzer das Menu-Icon (respektive Hamburger-Icon) als Menu-Button zu erkennen. Das Icon hat die Aufgabe verfehlt, den Benutzer zu informieren, dass sich dahinter ein Menu befindet.

12.4 Korrekturen am Prototyp

In diesem Abschnitt werden die eingeflossenen Korrekturen an der Benutzerschnittstelle aufgezeigt und anschliessend wird die abgegebene Version der Benutzerschnittstelle grafisch dargestellt.

Korrekturen der Analyse mit Stakeholdern Alle erwähnten Korrekturvorschläge (siehe Unterabschnitt 12.2.1) sind in der abgegebenen Version des Prototyps enthalten.

Korrekturen vom Testing Mit der Haupteinkenntnis, dass das Menu-Icon nicht klar verständlich ist, wird die Benutzerfläche entsprechend angepasst. Das Menu-Icon wird ergänzt mit dem Text "Menu", um dem Benutzer klar aufzuzeigen, dass sich ein Menu hinter dem Button befindet.

12.4 Korrekturen am Prototyp

Benutzerschnittstelle mit Korrekturen Nachfolgende Abbildungen zeigen die Abgabeverision der grafischen Benutzeroberfläche mit den eingeflossenen Feedbacks der Usability-Tests.

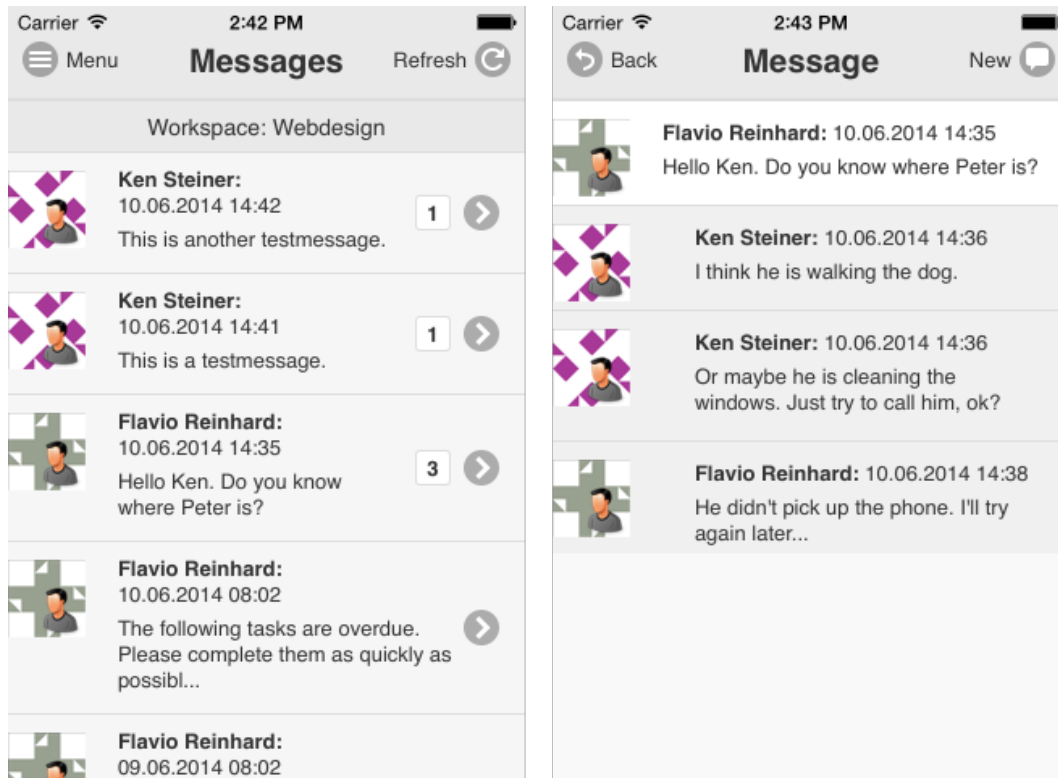


Abbildung 12.8: Abgabeverision: Messages

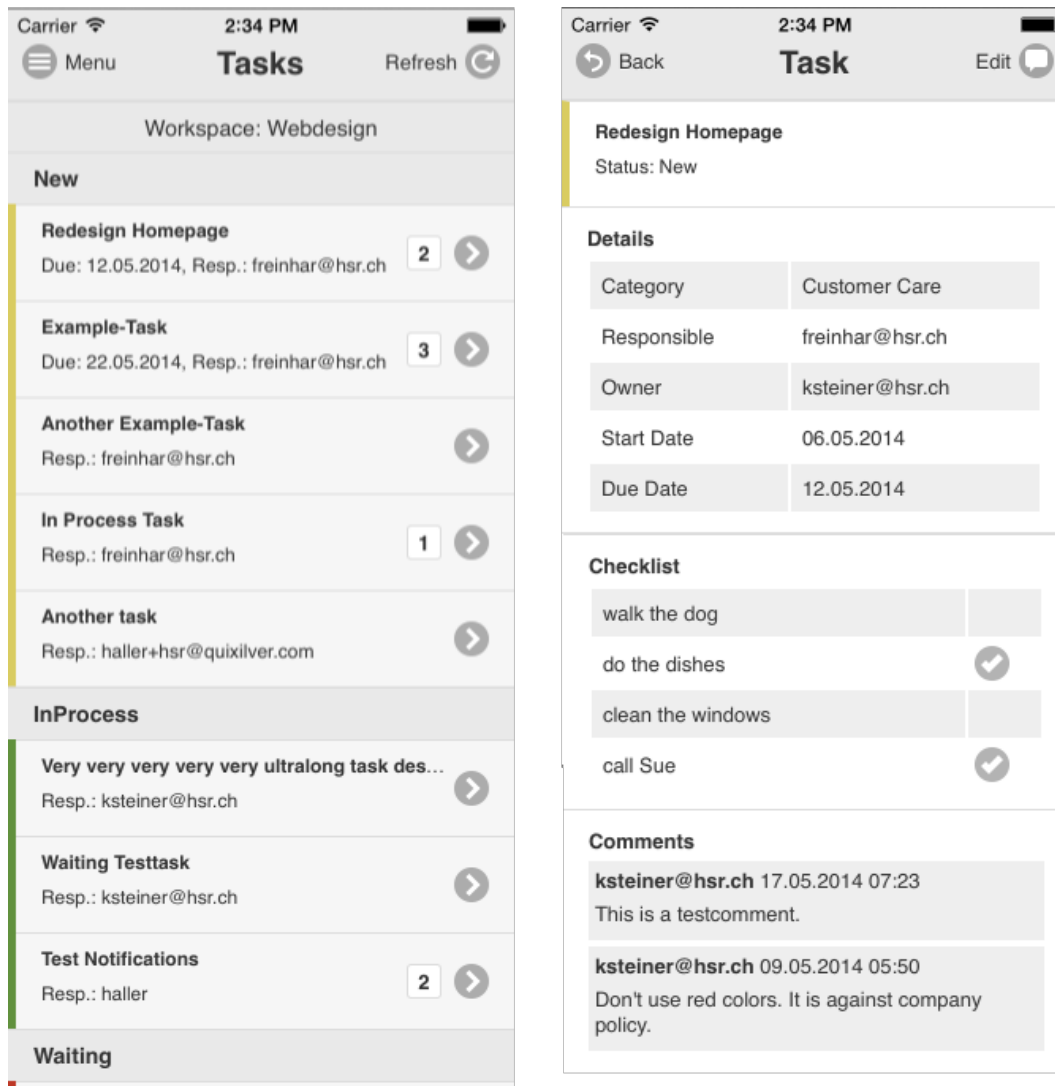


Abbildung 12.9: Abgabeverision: Tasks

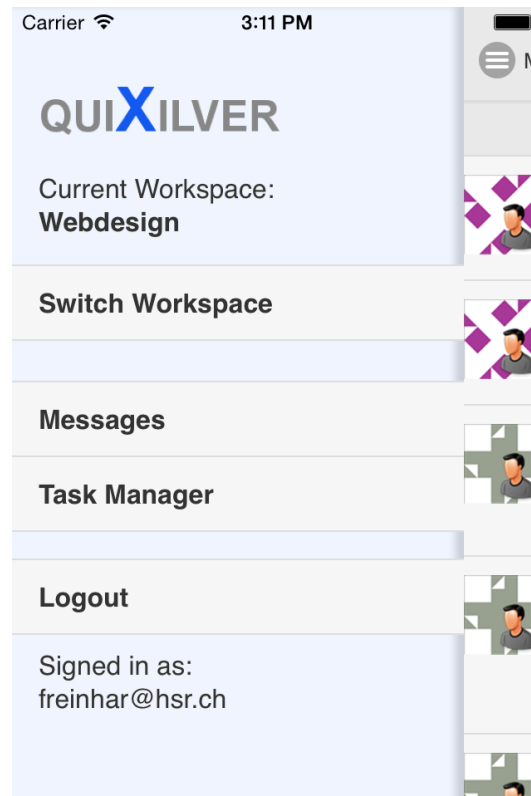


Abbildung 12.10: Abgabeverision: Navigation

13 API-Beschreibungen

In diesem Kapitel werden die durch Vigience zur Verfügung gestellten HTTP-Schnittstellen von QuiXilver beschrieben.

13.1 Request Token

Request Token wird beim Login-Prozess verwendet, um mit dem erhaltenen Authorization Code die benötigten Tokens zu erhalten.

13.1.1 Request

```
1 URL: http://alpha.quixilver.com/oauth/requestToken
2 ----
3 POST /oauth/requestToken HTTP/1.1
4 Content-Type: application/x-www-form-urlencoded
5
6 code=b81d6632-f6db-4d65-b12a-232a849ab0ad&
7 client_id=id&
8 client_secret=secret&
9 redirect_uri=http%3A%2F%2Flocalhost%2Foauth2callback&
10 grant_type=authorization_code
```

Listing 13.1: Request-Paket für Request Token Aufruf

code	Erhaltener Authorization Code
client_id	Fixer Wert (nicht im Bericht enthalten)
client_secret	Fixer Wert (nicht im Bericht enthalten)
redirect_uri	Fixer Wert
grant_type	Fixer Wert

13.1.2 Response

Bei erfolgreichem Aufruf wird ein JSON-Objekt im Body des Response-Pakets mitgegeben. Die folgenden Daten werden bei QuiXilver Mobile verwendet:

access_token	Access Token für Dateizugriff
refresh_token	Refresh Token für Erneuerung des Access Token
expires_in	Gültigkeits-Zeit des Access Token in Sekunden

13.2 Push Registrierung

Bei der Push Registrierung registriert sich der QuiXilver Mobile Client beim QuiXilver Server für Push Benachrichtigungen.

13.2.1 Request

```
1 URL: http://alpha.quixilver.com/api/1/users/me/devices
2 -----
3 POST /api/1/users/me/devices HTTP/1.1
4 Content-Type: application/json
5 Authorization: Bearer 3f5c4a80-3306-4fbc-9432-90c5e9aa6ff8
6
7 {
8     "registrationId": "{34khj34}",
9     "platform": "gcm",
10    "language": "en"
11 }
```

Listing 13.2: Request-Paket für Push Registrierung

registrationId	Die vom plattformspezifischen Push Server erhaltene Registration-ID
platform	Angabe der Push-Plattform. Mögliche Werte: • apns für iOS • gcm für Android • mpns für Windows Phone
language	Sprache für Push Benachrichtigungen

13.2.2 Response

Bei erfolgreichem Aufruf wird ein JSON-Objekt im Body des Response-Pakets mitgegeben. Die folgenden Daten werden bei QuiXilver Mobile verwendet:

registrationId	Beim Request übergebene registrationId
----------------	--

13.3 Datenzugriff Messages

13.3.1 Request

```

1 URL: http://alpha.quixilver.com
2 ----
3 GET /api/1/messages HTTP/1.1
4 Authorization: Bearer 875c3544-f7ba-42c2-abf8-2c14bc04b148

```

Listing 13.3: Request-Paket für Message-Datenzugriff

Das Header-Feld **Authorization** enthält das Access Token

Im Query-String der URL können folgende Filtermöglichkeiten angegeben werden:

zoneId (optional)	Workspace der Messages. Der Alias home referenziert auf Home-Workspace
sortOrder (optional)	Sort-Order der Messages, ASC oder DESC . Case-insensitive.
sinceId (optional)	Nur Messages mit höherer ID als übergebener Wert
maxId (optional)	Nur Messages mit tieferer ID als übergebener Wert
page (optional)	Page-Funktion. Startet bei 1.
pageSize (optional)	Anzahl der Messages. Standard: 100. Maximal: 200.

13.3.2 Response

Bei erfolgreichem Aufruf wird ein JSON-Objekt mit allen Messages im Body des Response-Pakets übermittelt.

13.4 Datenzugriff Tasks

Der Datenzugriff für Tasks funktioniert analog zum Datenzugriff für Messages. Die Task-API ist unter folgendem Namen erreichbar:

Datenzugriff	URL
Tasks	/api/1/cdb/TeamManagementTool_TaskManager_alpha
Task-Categories	/api/1/cdb/TeamManagementTool_Categories_alpha
Task-Status	/api/1/cdb/TeamManagementTool_Status_alpha
Task-Checklist	/api/1/cdb/TeamManagementTool_Checklist_alpha

Teil III

Technologie Tutorium PhoneGap

13.5 Requirements

To be able to deploy and continue developing QuiXilver Mobile Prototype make sure you have following tools installed:

- PhoneGap, Version 3.4.0-0.19.8
- XCode, Version 5.1.1
- Android Development Tools, Version 22.2.1
- Visual Studio Express 2012, Version 11.0.50727.38

13.6 Deploy QuiXilver Mobile Prototype

13.6.1 Preperation

Unzip the File *QuiXilverMobilePrototype.zip* to a directory of your choice.

13.6.2 Deploy on iOS

Verify that the target iOS device properly provisioned to be able to run QuiXilver Mobile Prototype. For instructions how to add a device to the provisioning profile visit: <https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html>. URL was last visited at June 8th, 2014 14:11.

Open the application called *Terminal* and navigate into the *QuiXilverMobile-Prototype* directory using the *cd* command.

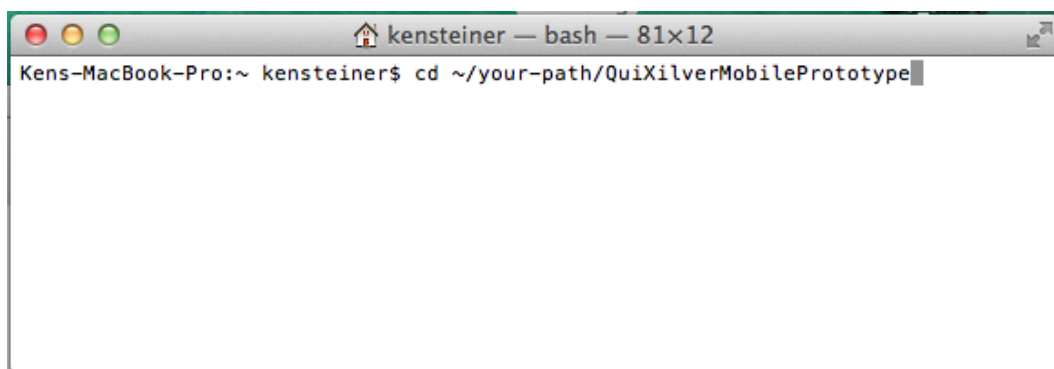


Abbildung 13.1: Open Project Directory

Build the iOS Project using the command *cordova build ios*.

13.6 Deploy QuiXilver Mobile Prototype

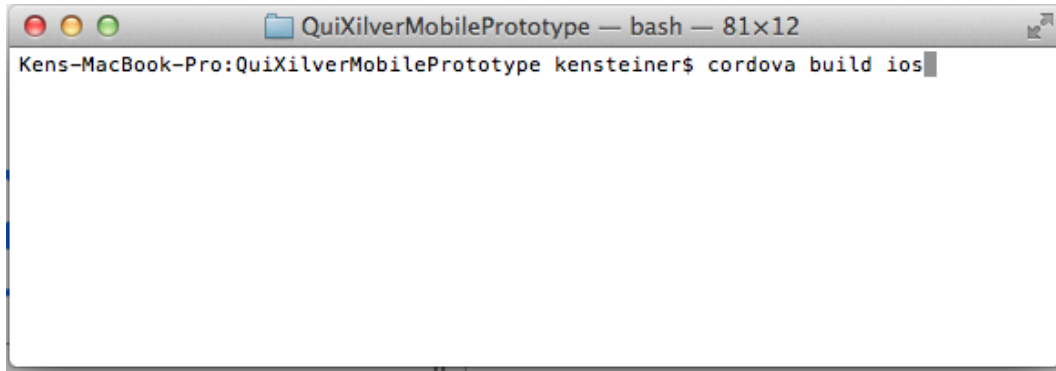


Abbildung 13.2: Build iOS Project

Open generated XCode project file */your-path/QuiXilverMobilePrototype/platforms/ios/QuiXilver Mobile Client.xcodeproj* using *XCode*. Make sure your iOS device is connected to your computer using an USB cable.

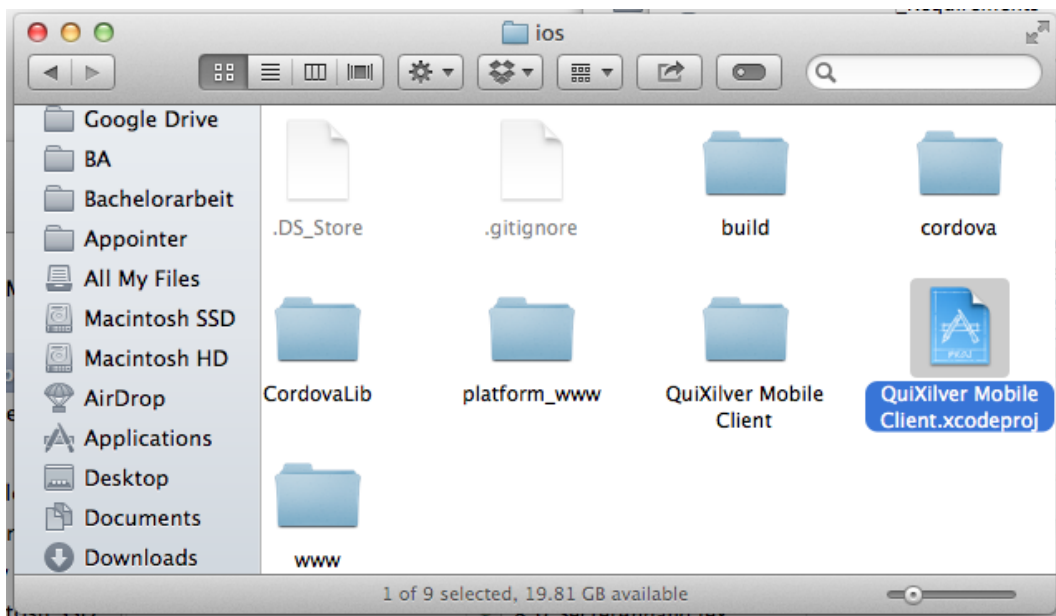


Abbildung 13.3: Build iOS Project

Once the project file is loaded in *XCode* press *PLAY* to run the app. A working copy of QuiXilver Mobile should now be running on your iOS device.

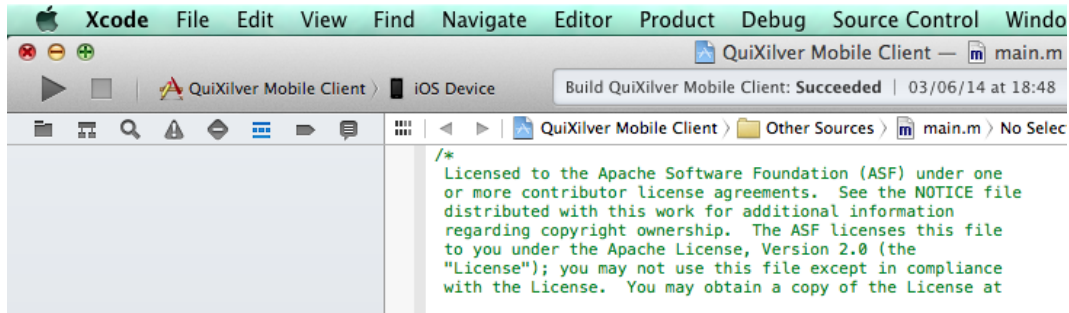


Abbildung 13.4: Run on iOS device

13.6.3 Deploy on Android

To be able to deploy QuiXilver Mobile Prototype to an Android device make sure */Settings/Developer options/USB debugging* is enabled.

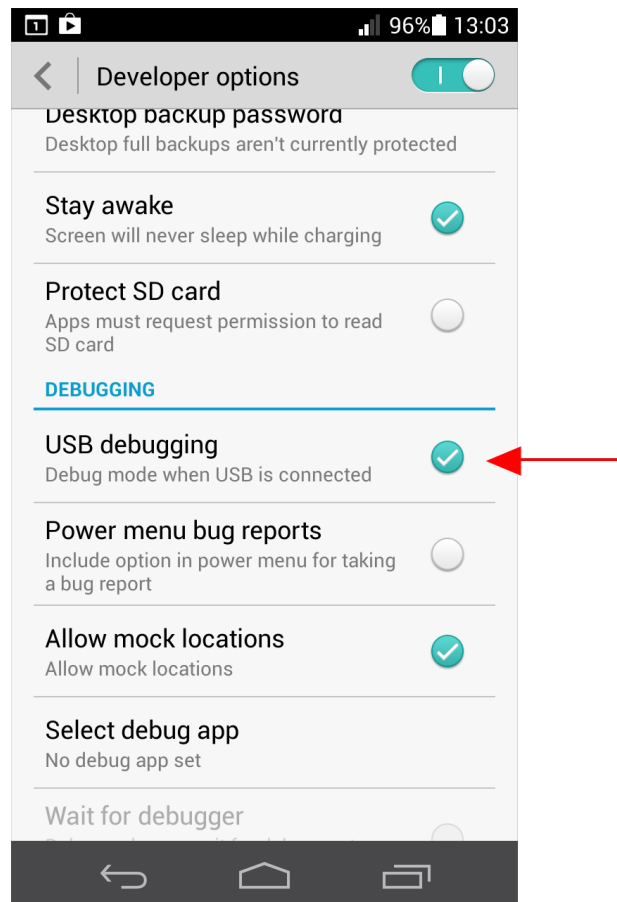


Abbildung 13.5: Settings: USB Debugging

Open the application called *Terminal* and navigate into the *QuiXilverMobile-Prototype* directory using the *cd* command.

13.6 Deploy QuiXilver Mobile Prototype

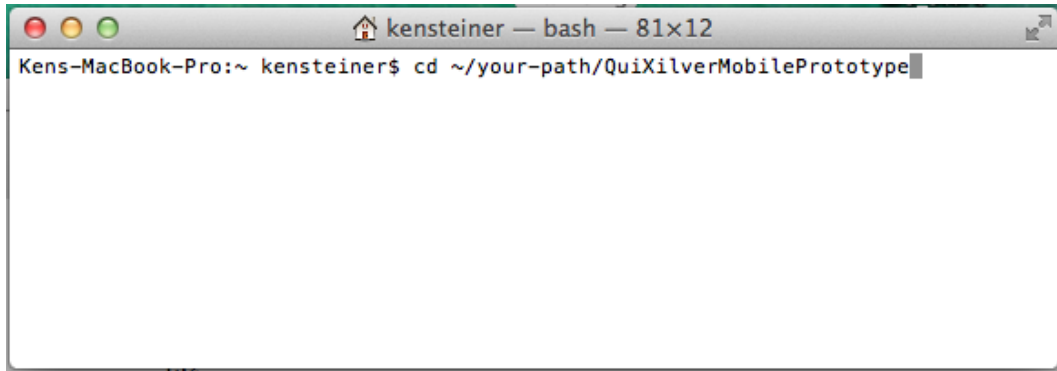


Abbildung 13.6: Open Project Directory

Make sure your Android device is connected to your computer using an USB cable. To run the app on the Android device enter following command in Terminal: *cordova run android*

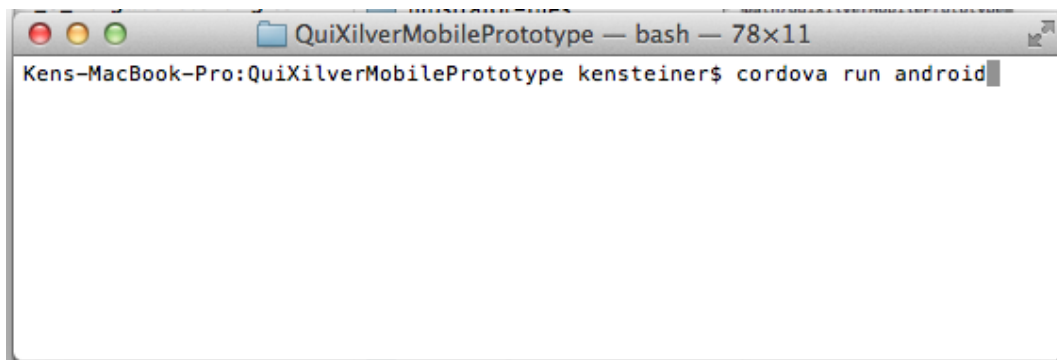


Abbildung 13.7: Run on Android device

13.6.4 Deploy Windows Phone

To be able to deploy following requirements have to be met:

- registered as a developer
- phone must be registered

Instructions on how to meet the requirements: [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402565\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402565(v=vs.105).aspx), last visited June 8th, 2014 19:12.

Open the application called *Command Line* and navigate into the *QuiXilverMobilePrototype* directory using the *cd* command.

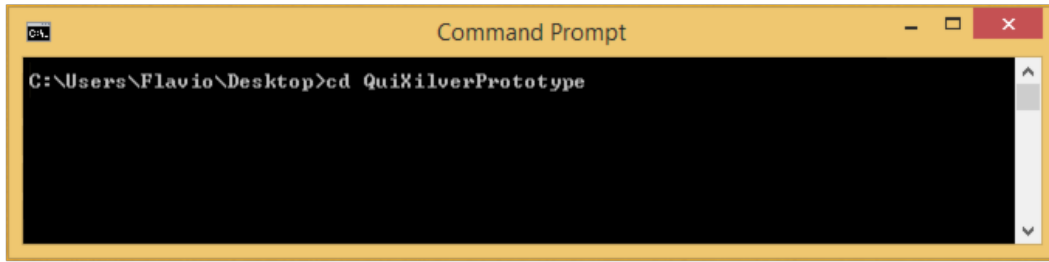


Abbildung 13.8: Open Project Directory

Build the Windows Phone 8 project using the command *cordova build wp8*.

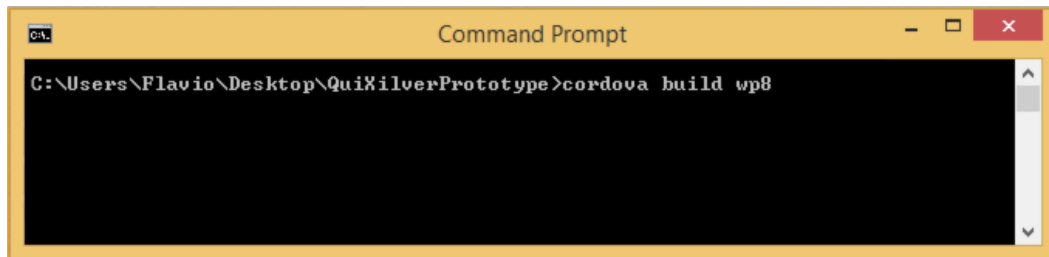


Abbildung 13.9: Build Windows Phone 8 Project

Open the Visual Studio project file */your-path/QuiXilverMobilePrototype/platforms/wp8/QuiXilver_Mobile_Client* using *Visual Studio Express*. Make sure your Windows Phone 8 device is connected to your computer using an USB cable.

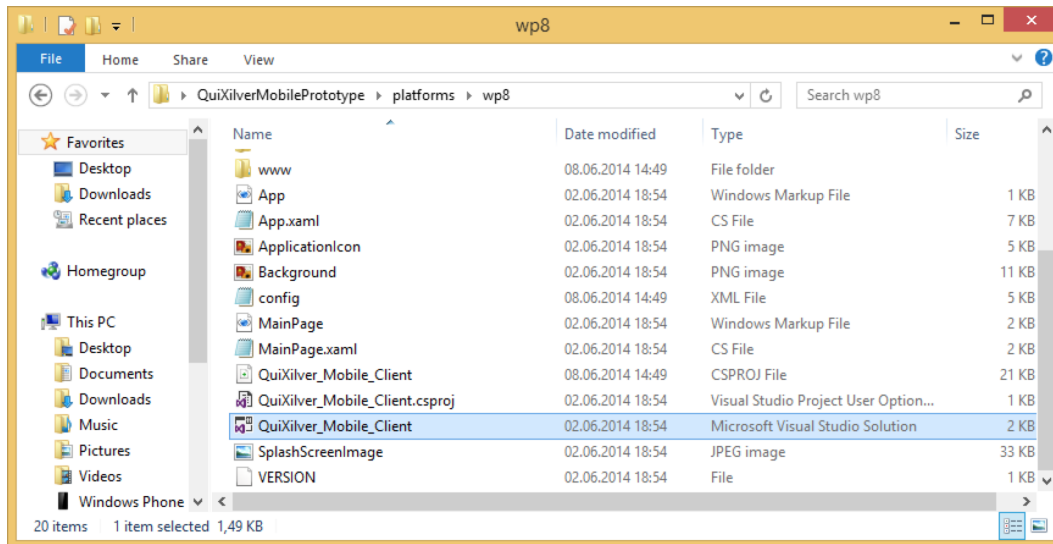


Abbildung 13.10: Open Windows Phone 8 Project

Once the project file is loaded in *Visual Studio Express* press *PLAY* to run the app. A working copy of QuiXilver Mobile should now be running on your Windows Phone 8 device.

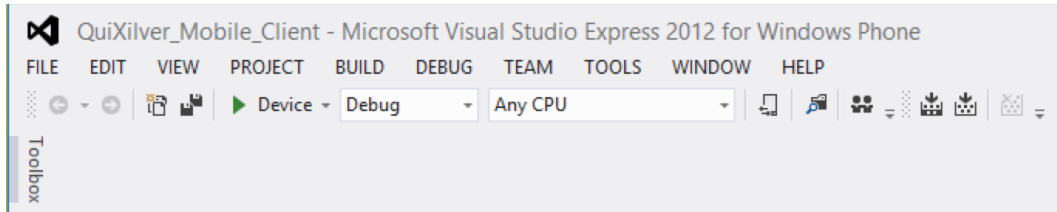


Abbildung 13.11: Run on Windows Phone 8 Device

13.7 Architectural Overview

This chapter explains QuiXilver Mobile Prototype's architectural layers. Following figure illustrates all implemented packages:

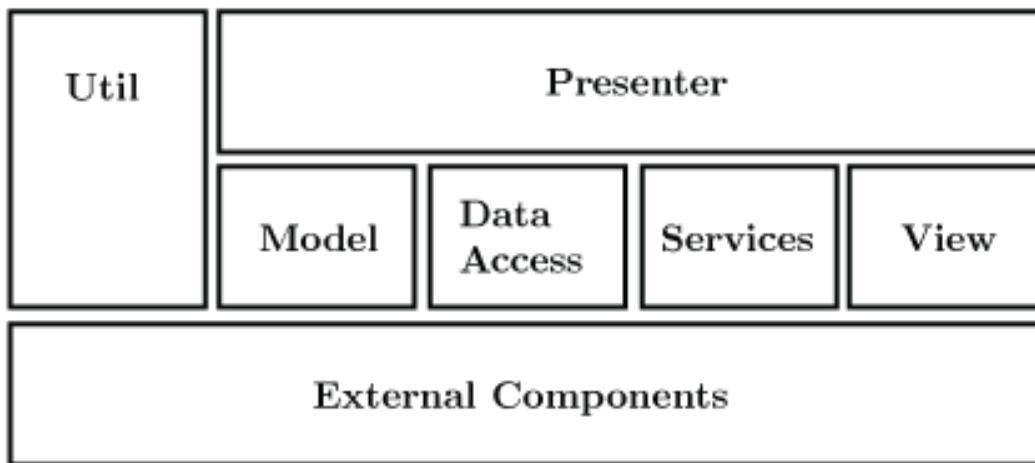


Abbildung 13.12: QuiXilver Mobile

13.7.1 View

The View layer is constructed as a single-page app¹ and is responsible to display the app using HTML and CSS.

The following list contains a brief description of each view, which will be explained in further detail in the next section:

- **Index** is the main page of the single-page app. It contains multiple div containers, into which the following templates will be filled into
- **Login** WebDiv which loads the login page
- **Messages** List of all available messages
- **Message** Detail view of a single message with comments

¹http://en.wikipedia.org/wiki/Single-page_application, last visited June 8th, 2014 19:29

-
- **Tasks** List of all tasks grouped by status
 - **Message** Detail view of a single task with a checklist and comments

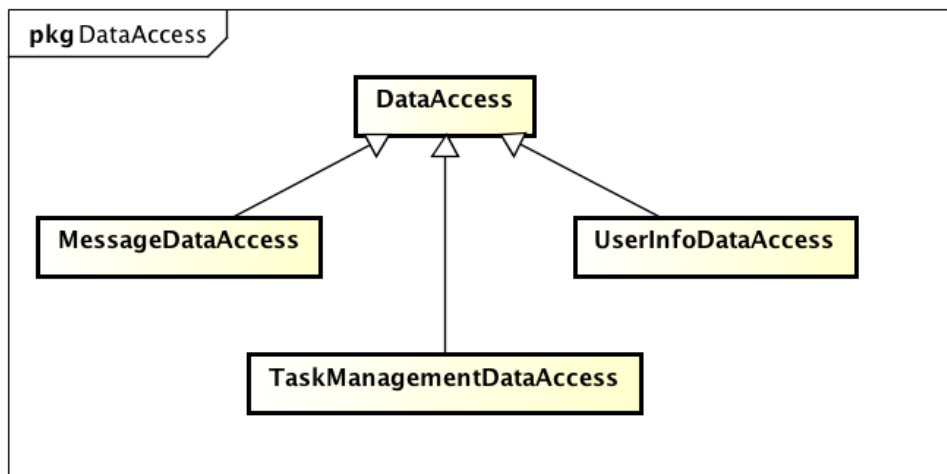
13.7.2 Presenters

The Presenter layer consists of presenters. Each view template has it's own presenter. The presenters are responsible for:

- Load a template, fill content into the template and render the view
- Handle user inputs
- Parse data returned by the server

13.7.3 DataAccess

DataAccess is responsible for handling asynchronous HTTP calls to get data from QuiXilver Server.



powered by Astah

Abbildung 13.13: DataAccess class hierarchy

The figure above shows the DataAccess class structure. It contains the following classes:

- **DataAccess** provides prepared HTTP calls, which are available to all deriving classes
- **MessageDataAccess** retrieves all available messages
- **TaskManagementDataAccess** retrieves all available tasks
- **UserInfoDataAccess** retrieves all user specific information

13.7.4 Model

The Model layer consists of the class structure, where the data received from the server is stored. The persisted data has the exact same structure.

All necessary data, used to display messages and tasks, are saved in this layer. See figure below for a structure overview of the Model layer.

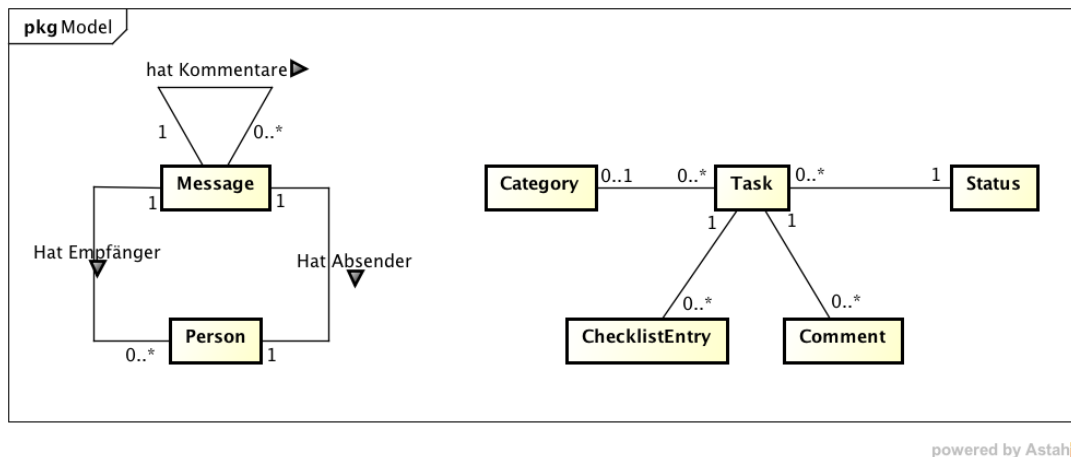


Abbildung 13.14: Model layer Entities

Message-specific Entities

- **Message** represents a single QuiXilver message. A message can have zero or more related messages as comments. Furthermore a message always has one message sender and can have one or more receivers.
- **Person** represents a QuiXilver user. A user is either a message sender or receiver.

The following code sample illustrates the relationship in JavaScript Object Notation (aka: JSON):

```
1 {
2   "messages": [
3     {
4       "message": "Hey John. Bist Du heute im Buero?",
5       "sent_by": {
6         "name": "Sarah",
7         "familyName": "Connor"
8       },
9       "receivingPersons" [
10        {
11          "name": "John",
12          "familyName": "Wayne"
13        }
14      ],
15       "comments": [
16        {
```

```

17         "message": "Ja. Bis 5 Uhr.",
18         "sent_by": {
19             "name": "John",
20             "familyName": "Wayne"
21         }
22     }
23 ]
24 }
25 ]
26 }

```

Listing 13.4: Simplified JSON: Messages

Task-specific Entities

- **Task** represents a task in QuiXilver's Task Managers. Every task has a status. Furthermore can a task be assigned to a category and has zero or more ChecklistEntries and Comments.
- **Category** represents a QuiXilver task category.
- **Status** represents the progress status of a task. Statuses available by default are: New, In Process, Waiting, Done and Closed. Users can add custom statuses.
- **ChecklistEntry** represents an entry in the checklist. An entry is either check or unchecked.
- **Comment** represents a task-specific comment. A comment belongs to one task.

The following code sample illustrates the relationship in JSON:

```

1  {
2    "tasks": [
3      {
4        "title": "Hotelzimmer buchen",
5        "category": categoryID,
6        "status": statusID,
7        "checklistEntries": [
8          {
9            "title": "Max. 4 Sterne",
10           "state": "checked"
11         }, {
12           "title": "Konferenzraum mit mind. 20 Sitzen",
13           "state": "unchecked"
14         }
15       ],
16       "comments": [
17         {
18           "comment": "Tipp: tripadvisor.com",
19           "sent_by": "John Doe"
20         }
21       ]

```

13.7 Architectural Overview

```
22     }  
23   ]  
24 }
```

Listing 13.5: Simplified JSON: Tasks

13.7.5 Services

The Services layer consists of the following functionality:

- **Login-Service** is responsible to handle login with QuiXilver Server.
- **Push-Service** is responsible for push notification. Retrieving the platform-specific registration tokens and sending the registration tokens to QuiXilver Server are it's main responsibilities.

13.7.6 Util

The Util layer consists of the following two classes:

- **Util** is responsible for: time formatting, time conversion and to get base path
- **Constants** contains all settings.

Both classes can be called from any layer.

13.7.7 Externe Komponenten

Following figure illustrates all 3rd-party components used in the app:

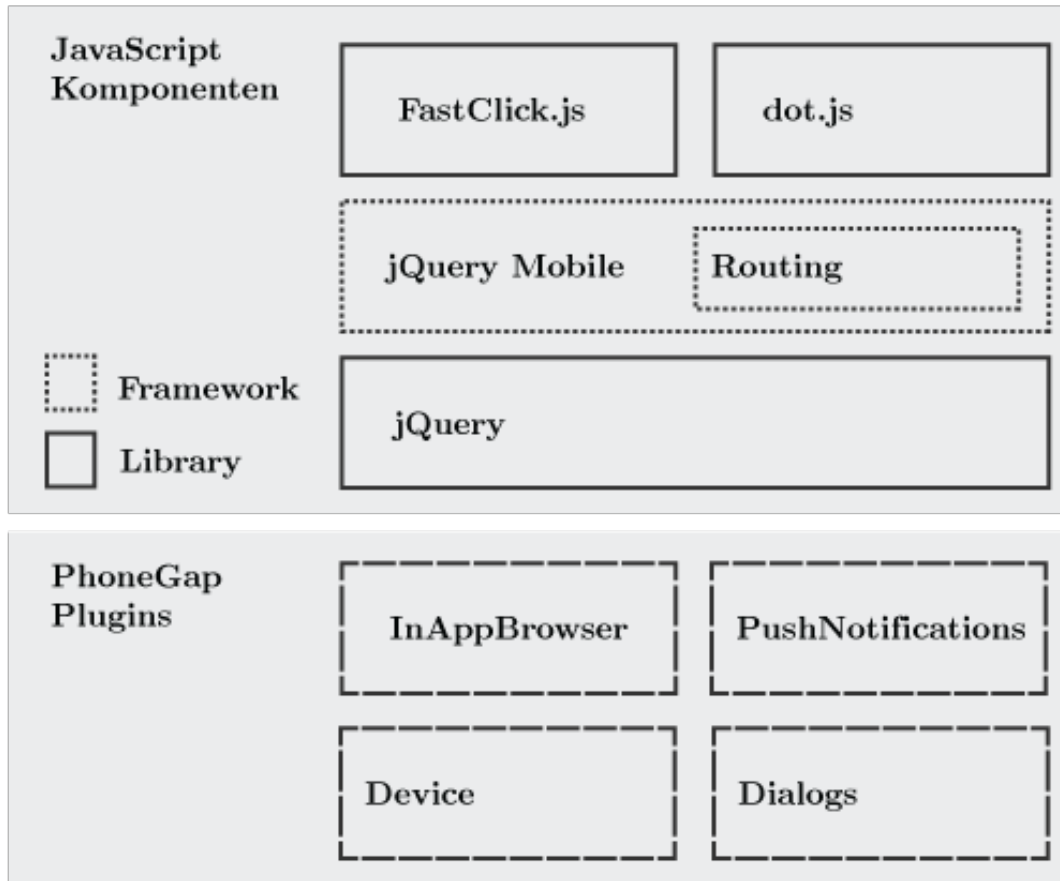


Abbildung 13.15: 3rd-Party Components

FastClick.js is an independent JavaScript library to reduce the 300ms delay in between physical tap and click-event execution. FastClick.js is used to make the app feel more reactive.

jQuery is a independent JavaScript library, which extends functionality of the JavaScript standard library. QuiXilver Mobile Prototype uses following jQuery functionality:

- DOM selection and modification
- handle HTTP calls and handle their replies

jQuery Mobile is a touch-optimised web-framework, which depends on jQuery. QuiXilver Mobile Prototype uses following jQuery Mobile functionality:

- Ready-to-use, touch-optimised user interface components (Exp.: Buttons and Tables)

13.7 Architectural Overview

- Ready-to-use, mobile display-optimised layouts
- Rendering and event-binding of pages
- Routing

jQuery Mobile Routing is a jQuery Mobile dependent plugin. jQuery Mobile Routing enable the developer attach functionality to the routing algorithm provided by jQuery Mobile.

Teil IV

Spezifikation Push Service

QuiXilver Mobile Client

SRS: Serverside Push Service

Department of Computer Science
University of Applied Sciences Rapperswil

Spring Term 2014

Authors: Flavio Reinhard & Ken Steiner

Professor: Prof. Dr. Olaf Zimmermann & Sebastian Hunkeler

Partner: Vigience AG, Horgen

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Validity	3
1.3	Structure of This Document	3
1.4	References	3
2	Platform-Independent Concepts	4
2.1	Push Notifications	4
2.2	Push Architecture	4
3	Platform-Specific Architecture	6
3.1	Android	6
3.2	Windows Phone	6
3.3	iOS	7
4	QuiXilver Push-Service Specification	8
4.1	Responsibilities of The Developing Parties	8
4.1.1	Vigience Development Team	8
4.1.2	University of Applied Sciences Rapperswil (HSR) Development Team	8
4.2	Registration Process	9
4.2.1	Push-Server Registration	9
4.2.2	Vigience-Server Registration	9
4.3	Push process	11
4.3.1	Triggering Events	11
4.3.2	QuiXilver Push Messages Definition	11
4.3.3	Push Message Structure	11
5	Testing	13
5.1	Registration Service	13
5.2	Triggering Events	13
5.3	Message Construction	13
6	Due date	14
6.1	Testing	14
7	Glossary	15

1 Introduction

1.1 Purpose

This document contains the software requirements specifications for the server-side push notification service. This service is needed to develop the mobile clients as defined in the bachelor thesis "QuiXilver Mobile Client" written by Flavio Reinhard and Ken Steiner.

1.2 Validity

This specification is valid for the whole duration of this bachelor thesis project.

1.3 Structure of This Document

This document has the following structure:

1. **Platform-Independent Concepts** gives an insight into how push-notifications are delivered in today's technologies. It specifies the common aspects of the covered platform solutions.
2. **Platform-Specific Architecture** contains references to websites, where the platform specific architectures are specified in detail.
3. **QuiXilver-Push-Service Specification** specifies what needs to be implemented in order to fulfill the requirements of the quixilver mobile client. One major goal of the chapter is to clearly distinct between what the project team will implement and what Vigience AG has to implement.
4. **Testing** describes how the project team (Flavio Reinhard & Ken Steiner) will test the push notification service implemented by Vigience.
5. **Due date** defines the due date for Vigience to complete the requirements described in this document.

1.4 References

- No referenced documents

2 Platform-Independent Concepts

This chapter gives an insight into the commonalities of the platform-specific solutions.

2.1 Push Notifications

Push notifications are defined as follows:

1. Push notifications on a smartphone are messages for an installed app.
2. These messages are sent from a central server.
3. The smartphone always handles the received messages in some way. It doesn't matter if the app is running or not.
4. How these messages are displayed depends on the platform.

2.2 Push Architecture

This section describes both the architecture and the process behind push notifications.

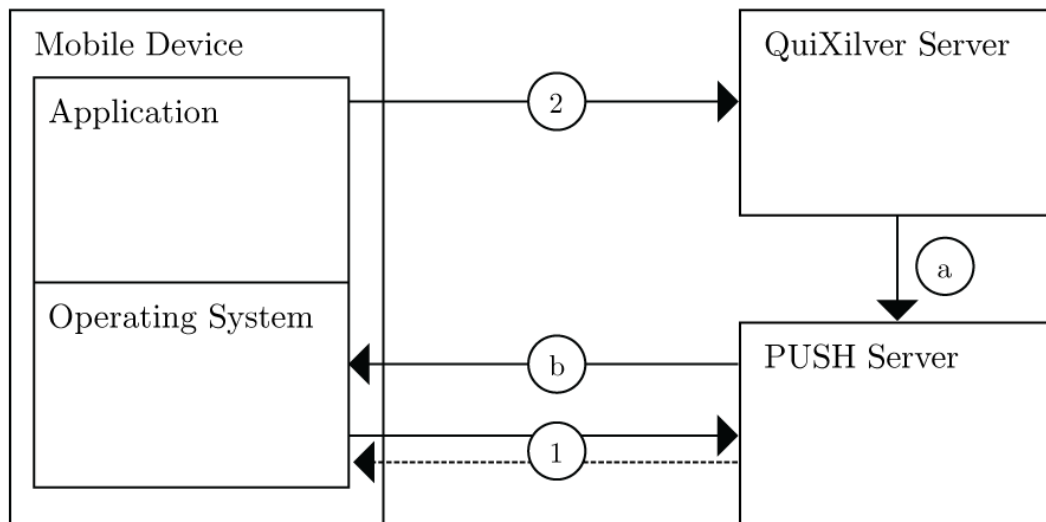


Figure 2.1: Basic Push Architecture

Figure 2.1 illustrates all participants in the push notification process:

- **Mobile device** consists of an **Application**(QuiXilver Mobile Client) and an **Operating System**. Those parts are distinguished since they take a different role in the push process.

2.2 Push Architecture

- **QuiXilver Server** represents the QuiXilver server as it is used in today's QuiXilver software.
- **PUSH server** represents the platform-specific push server. It's role in the process is the delivery of push notifications.

The **push process** itself is divided in two sections:

Registration for Push Notification At first the mobile device has to register itself on the push server and the QuiXilver server.

1. The mobile operating system sends a registration request to the push server. If no errors occur, the request will be replied with a registration id. The mobile operating system delivers the id to the application.
2. The application delivers the received registration id to the QuiXilver server. From now on the server stores the id for later usage (as described in the following paragraphs).

Sending Push Notification

- A An event on the server (defined later in this document) triggers a push notification. For each subscribed mobile client, the QuiXilver server sends a package to the push server. The most important part of that package is the registration id. It guarantees that no one but the original server can initiate a push notification, since he's the only one who has the registration id.
- B The push server delivers the requested push notification to the mobile device. The operating system handles the received push notification.

3 Platform-Specific Architecture

This chapter described the major differences of each platform compared to the platform-independent concept and their specialities. Furthermore it serves as a list of sources for reading on platform-specific architecture. All mentioned URLs of this chapter were available at this date: **March 17th, 2014, 21:04:16**.

3.1 Android

Registration process is nearly equal to the platform-independent concept. The user receives a unique registration id and sends it to the QuiXilver server.

Push process is also very close to the platform-independent concept. A HTTP request is sent to the push server. A JSON message is attached to the HTTP body. The structure of the JSON content follows a scheme defined by google.

Major notable difference is the possibility to define an array of recipients instead of a single recipient in one single HTTP request.

For further information on this topic visit:

<http://developer.android.com/google/gcm/index.html>

3.2 Windows Phone

Registration process is very close to the platform-independent concept. The user receives a unique registration URI. Then sends his registration URI to QuiXilver server.

Push process is also very close to the platform-independent concept. Every packet must address a specific URI and therefore is a singlecast message. The sent HTTP packet contains a JSON in it's body, which contains all the relevant data for pushing out a message.

Major notable difference is the unique identification of devices using URIs instead of JSON encapsulated IDs.

For further information on this topic visit:

[http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558\(v=vs.105\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558(v=vs.105).aspx)

3.3 iOS

Registration process user receives a unique registration token. Then sends his registration token to QuiXilver server.

Push process is also very close to the platform-independent concept. Every message must contain a JSON packet that includes the registration token for identification of the device and the content of the push message itself.

Major notable difference the message sent from QuiXilver server to the push server isn't a HTTP packet. Apple defined their own specific packet format, which is described at the URL below.

For further information on this topic visit:

<https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction.html>

4 QuiXilver Push-Service Specification

4.1 Responsibilities of The Developing Parties

4.1.1 Vigience Development Team

1. Is responsible to offer a RESTful web service so the registration IDs can be sent from mobile devices to the QuiXilver system.
2. Is responsible to send push notifications from QuiXilver push server to the platform-specific push server while conforming to the information of chapter *4.3 Push Process*.

4.1.2 University of Applied Sciences Rapperswil Development Team

1. Is responsible to call the provided registration interface with registration IDs, which were previously received from the platform-specific push servers.
2. Is responsible for providing an interface to handle the notifications from the platform-specific push server.

4.2 Registration Process

The registration process consists of two steps. The first step is to retrieve a registration id from the platform-specific push server. The second step is notifying the QuiXilver server the registration id.

4.2.1 Push-Server Registration

Vigience must provide information for the registration process so the client can successfully retrieve a registration id. The information contains:

Google Cloud Messaging (GCM):

- Sender ID (aka Project ID)
- API key

Microsoft Push Notification Service (MPNS):

- (none)

Apple Push Notification Service (APNS):

- App ID (containing product name und company identifier)

4.2.2 Vigience-Server Registration

Vigience must provide a RESTful service to notify the QuiXilver push server of the registration Id received from GCM, MPNS or APNS. The Hochschule Rapperswil's development team suggests to extend the current public RESTful-service with the following method.

REST API for Vigience server registration:

Method	HTTP request	Description
insert	POST /Api/1/users/me/devices	Creates new push message subscription

Table 4.1: REST API for Vigience server registration

Request body The following properties must be defined within the HTTP packet:

Property name	Value	Description
registrationID	string	Platform specific registration ID.
platform	string	Treat registrationId as: gcm Android client. mpns Windows Phone client. apns iOS client
userID	string	The unique user id

Table 4.2: Vigience server registration, request body scheme

4.3 Push process

4.3.1 Triggering Events

Push notifications are triggered by exactly the same events as in the current implementation of QuiXilver notification mails. Currently there are many different email-triggering events, such as:

- The user (who will later get the email) has received a message in the messaging centre
- Another user has assigned a task to the user

4.3.2 QuiXilver Push Messages Definition

In the notification center of each operating system it is possible to display a very brief text message to the user. The QuiXilver Server shall compose these messages. The already existing messages generated in the messages app (specified in requirements document) shall be used. The maximal message length is 70 characters, so if the message is longer, it must be truncated to 70 characters.

4.3.3 Push Message Structure

The generated message has to be sent to the correct platform specific push Server.

Android packets must conform to the following template:

POST <https://android.googleapis.com/gcm/send> HTTP/1.1

```
Request headers:
project_id: <project_id>
Authorization: key=<api_id>

Payload:
{
  "registration_ids" : [<registration_id>],
  "data" : {
    "message": "<message>"
  }
}
```

The following variables in the template above are to be replaced with computed values:

- <project_id>
- <api_id>
- <registration_id>

- `<message>`

The terms "api_id" and "project_id" are defined in androids developer specification (see section *3.1 Android*).

Windows Phone packets must conform to the following template:

```
<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
  <wp:Toast>
    <wp:Text1>QuiXilver</wp:Text1>
    <wp:Text2><message></wp:Text2>
    <wp:Param>/event=newMessage</wp:Param>
  </wp:Toast>
</wp:Notification>
```

The following variables in the template above are to be replaced with computed values:

- `<message>`

iOS packets must conform to following template:

```
{
  "aps" :
  {
    "alert" : "<message>",
    "sound" : "default"
  }
}
```

The following variables in the template above are to be replaced with computed values:

- `<message>`

5 Testing

The University of Applied Sciences Rapperswil development team will test the registration service, triggering events and message construction. The testing date is defined in the next chapter.

5.1 Registration Service

A single user will register mobile clients on all three implemented platforms and send the registration ids to the RESTful registration service provided by QuiXilver server.

The test expects a status code response with a value 200 for each HTTP request indicating the registration was successful.

5.2 Triggering Events

After successfully testing of *Test: Registration Service* the team will manually write messages in the QuiXilver Online client to each other. The first message will be posted to all workspace users (workspace contains three users) and the second message will be sent to at least two users.

The test expects that only the target people receive push notification for received messages on every platform. The first message is expected to trigger 6 notifications. The second message is expected to trigger 6 notifications.

5.3 Message Construction

After successful testing of *Test: Triggering Evenets* the team will verify the 12 notification messages. All 12 notifications will be compared to the previously defined *Push Message Structure*.

6 Due date

The projected due date for Vigience to complete the requirements in this document is:

April 20th, 2014

6.1 Testing

The project team will run the specified test within a week after delivery by Vigience.

7 Glossary

APNS is a messaging service provided by Apple to send push notifications to devices running iOS. 9

GCM is a messaging service provided by Google to send push notifications to devices running Android. 9

MPNS is a messaging service provided by Microsoft to send push notifications to devices running Windows Phone. 9

platform is a mobile OS. Examples: iOS, Android, Windows Phone. 4, 6, 11, 13

University of Applied Sciences Rapperswil is the University in Applied Sciences with strong focus on technical sciences based in Rapperswil, Canton St. Gallen, Switzerland. 2, 8, 13