
Master Projektarbeit

Titel: Evaluation von Frameworks und Methodiken zur hybriden, bzw.
Cross-Platform Anwendungsentwicklung (?)

Erstellt von:

B. Sc. Dominique Latza

Studiengang: Medizinische Informatik

Matrikelnummer: 7077438

Fachbereich Informatik

Fachhochschule Dortmund

Betreuung durch:

Prof. Dr. Vollmer (FH Dortmund)

Lutz Mathea (Firma Comline AG)

Dortmund, 15. April 2017

Inhaltsverzeichnis

1 Problematik und Lösungsansätze	5
2 Grundlagen	11
2.1 Das LG Nexus 5	11
2.2 Android	11
2.3 iOS	12
2.4 Entwicklungsumgebung	12
2.5 Native Anwendungs-Entwicklung	13
2.6 Hybride / Cross-Platform Entwicklung	13
2.7 Sensoren und andere Hardware in Smartphones	14
2.7.1 Accelerometer und Gyroskop	14
2.7.2 GPS	15
2.7.3 Näherungssensor und Umgebungslichtsensor	16
2.7.4 Vibration	16
2.7.5 Magnetometer	16
3 Anforderungsanalyse	17
3.1 Problemanalyse	17
3.2 Funktionale und nichtfunktionale Anforderungen	18
3.2.1 FA: Nutzung von Hardware-Komponenten	18
3.2.2 FA: Die grafische Benutzeroberfläche	19
3.2.3 FA: Betriebssysteme	20
3.2.4 FA: Sicherheit	20
3.2.5 FA: Interoperabilität/Erweiterbarkeit	20
3.2.6 FA: Tests und Performance	20
3.2.7 NFA: Hersteller	21
3.2.8 NFA: Support und Community	21
3.2.9 NFA: Entwicklung	21

Inhaltsverzeichnis

3.2.10 NFA: Lizenz und Kosten	22
3.3 Abgrenzungskriterien	22
4 Marktanalyse	23
4.1 Xamarin	29
4.2 Cordova	30
4.3 Ionic	30
4.4 React Native	30
5 Aufstellen der Bewertungskriterien für die Analyse	31
5.1 Kategorie Kosten und Lizenz	32
5.2 Kategorie Support und Community	32
5.3 Kategorie Entwicklung	32
5.4 Kategorie Hersteller	33
5.5 Kategorie OS-Versionen	33
5.6 Funktionsumfang	34
5.7 GUI-Design	35
5.8 Interoperabilität und Erweiterbarkeit	35
5.9 Tests	37
5.10 Performance	37
5.11 Sicherheit	37
6 Planung der Funktionstest-Anwendung	38
6.1 GUI Design	38
6.2 Hardwarezugriffe	41
6.3 Speicher	42
7 Objektorientierte Analyse und Design	44
7.1 Dynamisches Modell (OOA)	44
7.2 Statisches Modell (OOA)	46
7.3 Datenhaltung (OOD)	48
7.4 Die grafische Benutzeroberfläche	48
8 Implementierung	55
8.1 Native Implementierung	55
8.1.1 Grafische Benutzeroberfläche	55
8.1.2 Sensoren	58

Inhaltsverzeichnis

8.1.3 Kameras	60
8.1.4 Speicherzugriffe	61
8.2 Implementierung mit Xamarin	62
8.2.1 Grafische Benutzeroberfläche	62
8.2.2 Sensoren	65
8.2.3 Kameras	66
8.2.4 Speicherzugriffe	67
8.3 Implementierung mit Ionic/Cordova	67
8.3.1 Grafische Benutzeroberfläche	68
8.3.2 Sensoren	71
8.3.3 Kameras	72
8.3.4 Speicherzugriffe	73
8.4 Implementierung mit React Native	74
8.4.1 Grafische Benutzeroberfläche	74
8.4.2 Sensoren	78
8.4.3 Kameras	79
8.4.4 Speicherzugriffe	83
9 Auswertung	85
9.1 Kosten und Lizenz	85
9.2 Support und Community	86
9.3 Entwicklung	86
9.4 Hersteller	90
9.5 OS-Versionen	90
9.6 Funktionsumfang	90
9.7 GUI-Design	90
9.8 Interoperabilität	90
9.9 Tests	90
9.10 Performance	90
9.11 Programmiersprache	90
9.12 Sicherheit	90
Literatur	91

1 Problematik und Lösungsansätze

Über die vergangenen Jahre haben es Smartphones unter Endverbrauchern zu enormer Beliebtheit geschafft und sind mittlerweile überall im Alltag in Gebrauch(Real Challanges in Mobile App Development (X)). Mit dem Smartphone werden Urlaube geplant, Kinokarten gekauft, Rechnungen bezahlt. Wir benutzen das Smartphone und seine Anwendungen um mit Dienstleistern zu kommunizieren(Challenges and Opportunities of mob app dev (X)). Diese Allgegenwart von Smartphones hat entsprechend die Aufmerksamkeit von Software Entwicklern auf sich gezogen. Stand 2013 befanden sich etwa 800.000 mobile Anwendungen in Apple's AppStore, 650.000 im Android Store, 120.000 im Windows Marketplace und 100.000 in Blackberry's AppWorld(Real Challanges in Mobile App Development (X)). Die rasante Entwicklung sieht man, wenn man diese Zahlen mit dem Stand 2016 vergleicht: etwa 2.000.000 Anwendungen befinden sich in Apple's AppStore und in Androids Google Play sogar mehr als 2.560.000¹. Wie in oben genannter Statistik bereits erkennbar, gibt es mehrere Handelsplattformen mit jeweils einem eigenen Betriebssystem, Entwicklerwerkzeuge und Bibliotheken, wie zum Beispiel Android, iOS, Windows Phone oder BlackBerry. Fast jährlich erscheinen für diese Betriebssysteme Major Releases(Challenges and Opportunities of mob app dev (X)). Diese Fragmentierung der Plattformen und Standards führen auf der einen Seite zu verstärktem Wettkampf, was sich positiv auf Fortschritt und Weiterentwicklung auswirkt. Auf der anderen Seite ist dies eine Barriere in der Entwicklung von Inhalten und Services, was die Benutzer an eine spezifische Technologie bindet oder Entwicklerunternehmen einen Mehraufwand bereitet, um ihre Services auf mehreren Plattformen zu anzubieten(Challenges for Mobile Application Dev (X)).

Die Charakteristiken von mobilen Anwendungen und die Erwartungen der Benutzer führen zu folgender Basisherausforderung in der mobilen Anwendungsentwicklung: 'Liefere schnell aus und reagiere schnell auf Feedback', und dies in einem nicht en-

¹Statista Anzahl mobiler Anwendungen 2017.

Inhaltsverzeichnis

denden Zyklus. Jedes Unternehmen, dessen mobile Anwendungen 1-Stern Bewertungen und Kommentare mit Adjektiven wie 'verwirrend', 'unvollständig' oder 'langsam' bekommt, wird auf dem Markt verlieren, wenn diese Kritiken nicht ernst genommen und schnell Verbesserungen implementiert werden(Challenges and Opportunities of mob app dev (X)).

Die 3 großen Kategorien in der mobilen Anwendungsentwicklung sind nativ, Web-basiert und hybrid. Native Anwendungen sind für ein spezifisches Betriebssystem entwickelt und nutzen dessen Schnittstellen und Bibliotheken direkt. Sie müssen für jedes Betriebssystem auf dem sie laufen sollen separat entwickelt werden. Web-basierte Anwendungen laufen in einem Web Browser, der auf dem Smartphone installiert sein muss. Hybride Anwendungen sind 'native-wrapped' Web-Anwendungen. Bei nativen Anwendungen können, im Gegensatz zu Web-basierten Anwendungen, alle nativen Funktionen des Endgeräts genutzt werden, wie zum Beispiel Kameras und Sensoren(Challenges and Opportunities of mob app dev (X)).

Hybride und Cross-Plattform Anwendungen versuchen den Vorteil der Plattformunabhängigkeit Web-basierter Anwendungen mit dem Vorteil des Nutzens nativer Funktionen von nativen Anwendungen zu vereinen. Diese Form von Anwendungsentwicklung erzielt weltweit immer mehr Popularität aufgrund der Tatsache, dass sich ihr Code auf mehreren Plattformen kompilieren lässt. Entwicklungswerkzeuge für hybride und Cross-Plattform Anwendungsentwicklung basieren meist auf Programmiersprachen aus der Webentwicklung wie zum Beispiel HyperTest Markup Language (HTML), JavaScript oder Cascading Style Sheets (CSS). Dazu kommt dann eine Art Wrapper-Code für die Zugriffe auf die nativen Schnittstellen (APIs). Auf diese Weise können dann auch die nativen Funktionen wie Kameras und Sensoren angesprochen werden. Diese Form von Anwendungsentwicklung verspricht eine Reduzierung der Entwicklungskosten bei neuen Anwendungen, was sie für Entwicklerunternehmen attraktiv macht(Comparison of Cross-Platform Mobile Development (X)). Die Arbeit von X und Y (Comparison of Cross-Platform Mobile Development(X)) hat allerdings auch ergeben, dass ein gravierender Nachteil hybrider Anwendungen im Gegensatz zu nativen Anwendungen die Performance ist. Aus diesem Grund beschäftigt sich diese Arbeit mit dem Vergleich verschiedener Frameworks, die für Entwicklung hybrider und Cross-Plattform Anwendungen angeboten werden. Ein besonderes Augenmerk wird dabei auch auf die Performance bei der Nutzung nati-

Inhaltsverzeichnis

ver Funktionen im Vergleich zur nativen Anwendung gelegt.

In den letzten Jahren wurden einige Arbeiten über mobile Anwendungsentwicklung und auch Cross-Plattform, beziehungsweise hybride Anwendungsentwicklung veröffentlicht. In der Arbeit von X und Y (Comparison of Cross-Platform Mobile Development(X)) von 2012 wurden die 4 Frameworks Rhodes, PhoneGap, DragonRad and MoSync miteinander verglichen. Bei ihrer Arbeit konnten sie folgende Vorteile bei der hybriden Anwendungsentwicklung ausmachen:

- Reduzierung der benötigten Skills: Hybride Frameworks verwenden meist gängige Programmiersprachen wie HTML oder JavaScript.
- Weniger Code: Da plattformübergreifender Code produziert wird, muss nicht mehr für jede Plattform eine eigene komplette Anwendung mit separater Codebase entwickelt werden.
- Reduzierung der Entwicklungszeit und Wartungskosten, da nur eine Anwendung entwickelt und gewartet werden muss im Vergleich zu mehreren nativen Anwendungen.
- Die Entwickler müssen sich in weniger APIs einarbeiten und sich auskennen, da nicht direkt mit den APIs der Plattformen gearbeitet wird, sondern mit der API des jeweiligen Framework.
- Wachsende Marktanteile für das Business hinter der Anwendung.

Zusammengefasst lässt sich sagen, dass die hybride Anwendungsentwicklung einerseits verspricht die Investitionskosten zu senken und andererseits den Verkauf der Anwendungen auf mehrere Märkte ausweitet. Die Bewertungskriterien, anhand derer X und Y die Frameworks in ihrer Arbeit evaluiert haben, waren folgende:

- Die Betriebssysteme, die von den Frameworks unterstützt werden
- Lizenzen für die Frameworks zur Evaluierung der Geschäftsbedingungen
- Verfügbare Programmiersprachen für die Entwicklung der Anwendungen
- Verfügbarkeit der APIs hinsichtlich der Fragestellungen welche nativen Funktionen genutzt werden können

Inhaltsverzeichnis

- die Architektur, die für den Entwicklungsprozess der Anwendung zur Verfügung steht
- Integrated Development Environments (IDEs) die verfügbar und nutzbar sind

Für die API-Tests hat sich für die 4 von X und Y ausgewählten Frameworks folgendes Bild ergeben(Abbildung 1.2):

API Name	Rhodes JavaScript	PhoneGap JavaScript	MoSync JavaScript	MoSync C, C++	DragonRad
Accelerometer		✓	✓		
Barcode	✓	✓			✓
Bluetooth	✓	✓		✓	
Calender	✓	✓	✓	✓	✓
Camera	✓	✓		✓	
Capture		✓	✓	✓	✓
Compass		✓	✓		
Connection		✓	✓	✓	
Contacts	✓	✓			✓
Device	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	
Geolocation	✓	✓	✓	✓	✓
Menu	✓				✓
NFC	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✓	
Screen Rotation	✓	✓		✓	
Storage	✓	✓	✓	✓	✓

Abbildung 1.1: Ergebnis API-Tests aus X und Y Arbeit: Comparison of Cross-Platform Mobile Development aus 2012

Wie man in Abbildung 1.2 erkennen kann, schafft es vor allem PhoneGAP alle bis auf eine getestete native Funktionalität umzusetzen. Allerdings fanden X und Y heraus, dass vor allem die Frameworks, welche auf JavaScript basieren, deutliche Performance-Probleme haben. Aus diesem Grund raten die Autoren davon ab JavaScript-basierte Frameworks für Anwendungen mit komplexen Funktionalitäten oder im Hintergrund laufende Services zu nutzen. Auch sei die Unterstützung von High-End Grafiken und 3D-Technologien unzureichend.

Eine weitere Arbeit, die sich mit dem Vergleich von hybriden Frameworks beschäftigt hat ist die von X und Y: Cross-Platform Mobile Development: A Study on Apps with Animations(X) aus dem Jahr 2014. Die Frameworks, die in dieser Arbeit evaluiert wurden sind MoSync, Titanium, JQuery & JQuery Mobile und PhoneGap. Die Bewertungskriterien, die die Autoren aufstellten sind folgende:

Inhaltsverzeichnis

- Lizenzmodell und Kosten
- Die Vielfältigkeit und Qualität der verfügbaren APIs
- Das Angebot an Tutorials
- Die Größe der Community
- Die Komplexität des Codes um die vorgegebene Anwendung zu implementieren
- Die Benutzerfreundlichkeit der IDE des Frameworks
- Die Liste der unterstützten Geräte
- Inwieweit die Frameworks das Erstellen einer nativen Benutzeroberfläche unterstützen
- Das geforderte Basiswissen im Bereich Programmierung und Technologiekenntnis für jedes Framework

Wie man an den Bewertungskriterien oben schon erkennen kann, gab die Autoren eine Beispiel-Anwendung vor, die mit Hilfe der 4 ausgewählten Frameworks implementiert wurde, um diese gegeneinander zu evaluieren. Um einem Bias entgegenzuwirken wurden für die einzelnen Entwicklungen Entwickler ausgewählt, die mit der jeweiligen Programmiersprache bereits vertraut waren. Die Frameworks wurden anschließend in den oben aufgeführten Kategorien mit Noten von 0 (schlecht) bis 5 (sehr gut) bewertet. Unten stehende Tabelle (Abbildung (X)) zeigt die Ergebnisse:

	MoSync	Titanium	jQuery	Phonegap
Licenses	5	4	5	5
API	3	4.5	2	2
Community	3	5	5	5
Tutorials	4	5	4	5
Complexity	2	5	4	4
IDE	4	5	-	-
Devices	2	4	5	4
GUI	4.5	5	3	-
Knowledge	5	5	5	5

Abbildung 1.2: Ergebnis Framework Evaluation von X und Y: Cross-Platform Mobile Development: A Study on Apps with Animations aus 2014

Inhaltsverzeichnis

Auf Basis oben (Abbildung (X)) dargestellten Ergebnisses bewerten die Autoren das Framework Titanium als das Bestes unter den getesteten für Anwendungen mit Animationen. Begründung hierfür ist, dass Titanium Animationen und Übergangseffekte nativ unterstützt und die Performance gut ist und vermuten lässt, dass auch bei komplexeren Anwendungen die Performance noch ausreichend sein wird.

In dieser Arbeit sollen ebenfalls, wie bei oben vorgestellten Publikationen, hybride Frameworks direkt miteinander verglichen werden. Um die zu evaluierenden Frameworks auszuwählen, wird zunächst eine Marktanalyse durchgeführt, welche anzeigen soll, welche Frameworks die aktuell attraktivsten sind. Für die Evaluation wird eine Test-Anwendung entwickelt, die mit den ausgewählten Frameworks implementiert werden soll. Diese Anwendung wird zuvor noch als Referenz für zum Beispiel die Performance noch nativ implementiert. Die Anzahl der Frameworks, die mit dieser Anwendung getestet werden ist dabei auf 5 limitiert. Allerdings werden noch weitere Frameworks mit in die Evaluation aufgenommen. Diese können aber nur nach Bewertungskriterien verglichen werden, die nicht direkt mit dem Entwicklungsprozess zusammenhängen. Es wird versucht die Kriterienkataloge der oben vorgestellten Publikationen mittels einer Umfrage unter Anwendungsentwicklern noch zu erweitern. Da aufgrund bisheriger Publikationen davon auszugehen ist, dass es nicht 'das eine' Framework für alle Problemstellungen gibt, soll das Ergebnis dieser Arbeit ein Entscheidungsbaum werden, anhand dessen das ideale Framework je Problematik ermittelt werden kann.

2 Grundlagen

Zunächst werden für ein besseres Verständnis der Arbeit die eingesetzten Materialien und Grundlagen vorgestellt. Zu diesen zählen diverse Hardware, wie das verwendete Smartphone, aber auch Software wie unter anderem die Entwicklungsumgebung oder das verwendete SDK (Software Development Kit) für die native Evaluationsanwendung. Zudem wird eine Einführung in die Begrifflichkeiten wie native und hybride Anwendungsentwicklung gegeben.

2.1 Das LG Nexus 5

Für die Evaluation anhand einer zu entwickelnden Anwendung, welche unter anderem Hardware-Zugriffe auf Sensorik etc. testen soll, steht ein LG Nexus 5 als Gerät zur Verfügung. Das LG Nexus 5 erschien am 31. Oktober 2013 mit der Android-Version 4.4 KitKat. Es verfügt über ein Full HD IPS Display mit einer Bildschirmdiagonalen von 12,70 cm. Die Auflösung beträgt 1920 x 1080 Pixel. Das Smartphone besitzt eine 8 MP Kamera hinten und eine 1,3 MP Front-Kamera. Als Prozessor ist ein Qualcomm Snapdragon 800 mit einer Taktrate von 2,26GHz und 4 Kernen verbaut. Zudem ist das LG Nexus 5 mit folgenden Sensoren ausgestattet: Accelerometer, Gyroskop, Näherungssensor, Magnetometer und Barometer.

2.2 Android

Android¹ ist ein Betriebssystem für mobile Geräte wie Smartphones oder Tablets, ursprünglich entwickelt von der Firma Android, Inc., welche 2005 von Google aufgekauft wurde². Mobile Anwendungen für Android Systeme werden in der Programmiersprache Java geschrieben und anschließend in Androids eigenes Format DEX

¹Android 2016.

²Manjoo o.D.

(Dalvik Executable Format) konvertiert³. Die Android UI Guidelines⁴ geben Richtlinien für das Design der Benutzeroberfläche von Android Anwendungen vor.

Des Weiteren sind für Android Anwendungen unter anderem noch folgende Grundbedingungen formuliert, an welche sich bei der Planung der Anwendung gehalten wurde: Die Anwendung sollte einfach zu installieren, zu entfernen und zu updaten sein. Sie sollte ansprechend sein und die Anforderungen elegant umsetzen um auch bei vielen Features leicht bedienbar zu bleiben. Wichtig ist, dass die Anwendung stabil, skalierbar, bedienbar ist und angemessen auf Benutzereingaben reagiert.⁵

2.3 iOS

iOS ist das native Betriebssystem von Apple für alle Apple Geräte, wie zum Beispiel das iPhone, iPad oder iMacs . Mobile Anwendungen für iOS werden in einer Programmiersprache namens Swift entwickelt. Apple stellt zum Entwickeln eine eigene IDE zur Verfügung: XCode. Zum Entwickeln von nativen iOS-Anwendungen wird immer ein Apple Computer benötigt⁶.

Die Richtlinien für das Design grafischer Benutzeroberflächen sind in den 'iOS Human Interface Guidelines' definiert. Die drei primären Grundsätze dabei lauten: Klarheit, Fügsamkeit und Tiefe (Clarity, Deference and Depth)⁷.

2.4 Entwicklungsumgebung

Das Android SDK ist eine Entwicklungsumgebung für das Android Betriebssystem, welches sich an Entwickler zur Erstellung von Android-Anwendungen wendet und ist für Windows, Linux und Mac OS verfügbar. Es benötigt für viele Hauptfunktionen ein JDK (Java Development Kit)⁸. Das SDK beinhaltet einen Emulator, der es möglich macht die Anwendung auch ohne angeschlossenes Smartphone zu testen. Als IDE (Integrated Development Environment) wurde Android Studio genutzt, wel-

³Darwin o.D.

⁴Android UI Guidelines 2016.

⁵Darwin o.D.

⁶iOS 2017.

⁷iOS GUI Guidelines 2017.

⁸Android SDK 2016.

ches 2014 von Google veröffentlicht wurde und so Eclipse als primäre Entwicklungs-umgebung für Android Anwendungen ablöste. Android Studio basiert auf der IntelliJ IDEA Community Edition von JetBrains.⁹ Es beinhaltet intuitive Tools, die die Erstellung einer grafischen Benutzeroberfläche nach den Android UI Guidelines¹⁰ erleichtern.

2.5 Native Anwendungs-Entwicklung

Eine native Anwendung ist eine Anwendung, die für eine spezifische Plattform entwickelt wurde und die Schnittstellen dieser Plattform direkt benutzt¹¹. Dadurch kann die Anwendung mit dem Betriebssystem und andere Software, welche auf dem Gerät installiert ist, interagieren. So kann die native Anwendung auch die gerätespezifische Hardware und Software, wie zum Beispiel das GPS oder Kameras verwenden¹². Ein weiterer Vorteil der nativen Anwendungen ist, dass sie die Ressourcen des Geräts durch die einheitliche Nutzung der Schnittstellen zur Hardware optimal nutzen können¹³. Die plattformspezifischen Bibliotheken bieten zudem alle nötigen Elemente für den Aufbau einer der spezifischen Guidelines entsprechenden Benutzeroberfläche. Eine native Anwendung bringt das maximale Ergebnis bezüglich Look-And-Feel und Performance¹⁴.

2.6 Hybride / Cross-Platform Entwicklung

Eine hybride Anwendung ist prinzipiell wie eine Web Anwendung, nur mit einem leichtgewichtigen nativen Container, welcher es erlaubt, auf native plattformspezifische Funktionalitäten und Hardware zuzugreifen. Wie auch Web Anwendungen werden hybride Anwendungen oft in Skript-Sprachen wie JavaScript, HTML5 und CSS programmiert¹⁵. Auf diese Weise entsteht Code, der gemeinsam auf allen Plattformen genutzt werden kann. Dies hat den Vorteil, dass nur eine Anwendung gewartet werden muss und nicht eine pro Plattform, für die die Anwendung herausge-

⁹Android Operating System 2016.

¹⁰Android UI Guidelines 2016.

¹¹App-Entwicklung Varianten 2016.

¹²native Anwendungen 2017.

¹³Ebd.

¹⁴App-Entwicklung Varianten 2016.

¹⁵Ebd.

2.7 Sensoren und andere Hardware in Smartphones

geben werden soll. Als ein Nachteil wird allerdings die Performance genannt, welche unter Umständen, wie zum Beispiel einer erhöhten Nutzung von Hardware-Komponenten, deutlich schlechter als bei der nativen Variante sein soll¹⁶. Nachfolgende Abbildung 2.1 beschreibt noch einmal bildhaft den Unterschied zwischen den drei Entwicklungsvarianten: Nativ, Hybrid und Web:

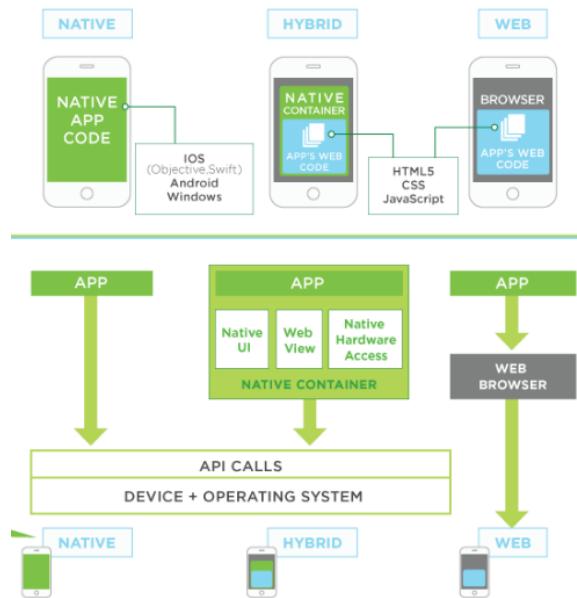


Abbildung 2.1: Vergleich: native, hybride und Web Anwendungsentwicklung

2.7 Sensoren und andere Hardware in Smartphones

Im folgenden werden die gängigen Sensoren, welche in Smartphones eingebaut sind vorgestellt. Diese Sensoren werden in Anwendungen verwendet, um zum Beispiel die aktuelle Position zu bestimmen oder Feedback in Form von Vibrationen zu geben.

2.7.1 Accelerometer und Gyroskop

Das Accelerometer, oder auch der Beschleunigungssensor, misst die Beschleunigung in X-, Y- und Z-Achsenrichtung. Unter Beschleunigung wird die Änderung der Geschwindigkeit zur Zeit verstanden. Im physikalischen Sinne ist jede Form von Änderung einer Bewegung, auch eine Abnahme der Geschwindigkeit oder Schwenken in eine

¹⁶App-Entwicklung Varianten 2016.

2.7 Sensoren und andere Hardware in Smartphones

andere Richtung, eine Beschleunigung. Das Gyroskop misst hingegen die rotatorische Geschwindigkeit, also Drehbewegungen. Zusammen mit dem Accelerometer erkennt das Smartphone so Bewegungsänderungen und kann darauf reagieren¹⁷. Die Ausrichtung der 3 Achsen von einem Smartphone ist in folgender Abbildung 2.2 dargestellt:

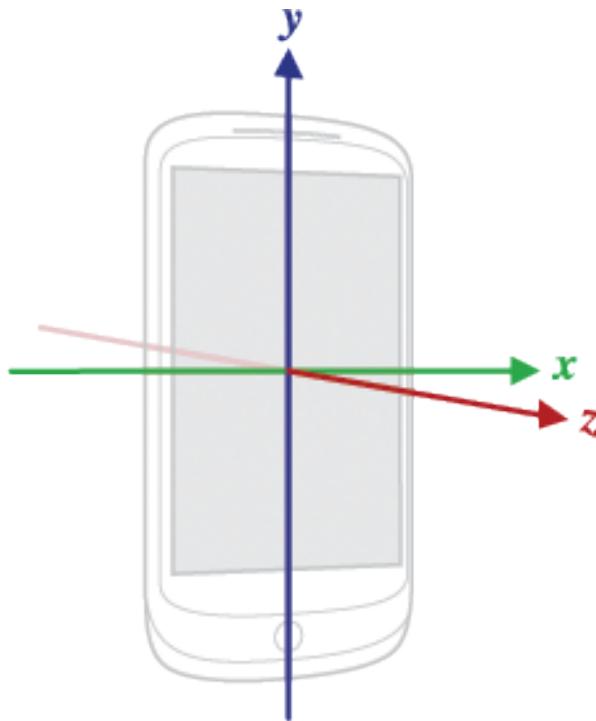


Abbildung 2.2: Schematisches Achsenmodell; Entnommen von: www.droidwiki.de/wiki/G-Sensor

2.7.2 GPS

GPS ist die Abkürzung für Global Positioning System und beschreibt ein System zur globalen Positionsbestimmung mit Hilfe von Satelliten. Smartphones mit GPS besitzen einen GPS-Chip, mit denen sie ihre Position auf 5-10 Meter genau bestimmen können¹⁸. Der GPS-Sensor ist relevant für zum Beispiel Navigations-Anwendungen.

¹⁷ Android Wiki 2017.

¹⁸ Ebd.

2.7.3 Näherungssensor und Umgebungslichtsensor

Der Näherungssensor bei Smartphones besteht üblicherweise aus einem Fotowiderstand und er dient zusammen mit dem Umgebungslichtsensor zur Messung der Helligkeit des Umgebungslichtes. Die Aufgabe des Näherungssensors ist es in der Regel, schnelle Änderungen der Umgebungshelligkeit festzustellen. Der Umgebungslichtsensor misst anhand des photoelektrischen Effekts das Umgebungslicht und passt daraufhin die beispielsweise automatisch die Display-Helligkeit an¹⁹.

2.7.4 Vibration

Um auf Benutzereingaben eine Rückmeldung in Form eines Vibrationsfeedbacks zu geben, sind Vibrationsmotoren in Smartphones verbaut. Diese sind über die API des jeweiligen Betriebssystems ansprechbar. Gerade bei Geräten mit Touchscreen und ohne mechanische Tastatur kann ein solches Feedback es dem Benutzer vereinfachen, seine Eingabe als erfolgt zu erkennen²⁰.

2.7.5 Magnetometer

Ein Magnetometer misst die magnetische Flussdichte und kann so bestimmen, wo sich der magnetische Norden befindet. Im Inneren des magnetometers befindet sich eine metallische Platte, welche unter Strom gesetzt wird. Durch das Wirken der Lorentzkraft auf diese Platte, deren Stärke abhängig von der Stärke des vorhandenen Magnetfeldes ist, wird diese verschoben. Das Magnetometer misst optisch oder elektrisch wie weit sich die Platte verschoben hat. Mit Hilfe des Magnetometers im Smartphone lassen sich Kompass-Anwendungen und Anzeigen der Blickrichtung auf Karten-Anwendungen realisieren²¹.

¹⁹Android Wiki 2017.

²⁰Ebd.

²¹Android Magazine 2017.

3 Anforderungsanalyse

blablabla

3.1 Problemanalyse

Zur Evaluation der ausgewählten Frameworks wird eine Anwendung designed, welche diverse Hardware- und Software-Funktionalitäten eines Smartphones nutzt und testet. Diese Anwendung wird zunächst als Referenz nativ für die Android Plattform(X) in Android Studio entwickelt um anschließend mit 5 ausgewählten Frameworks neu umgesetzt zu werden. Auf diese Weise wird überprüft und evaluiert, welche Funktionalitäten mit welchen Frameworks nutzbar sind und welcher eventuelle Mehr-Aufwand entsteht. Zudem wird bei der Neumsetzung mit den ausgewählten Frameworks überprüft, inwieweit sich die GUI der nativen Anwendung im Material Design(X) nachstellen lässt.

Als weitere Grundlage für die Evaluation der Frameworks dient eine Umfrage, die an Studierende der Fachhochschule Dortmund mit Erfahrung im Bereich Anwendungsentwicklung gestellt wurde. Mit Hilfe dieser Umfrage werden weitere Kriterien ermittelt, die Entwicklern wichtig bei der Arbeit mit einem Framework sind.

Aus oben genannten Hardware- und Software-Funktionalitäten und den durch die Umfrage ermittelten Kriterien wird eine Evaluations-Matrix erstellt, welche während der Umsetzung der Referenz-Anwendung mit den ausgewählten Frameworks gefüllt wird. Aus dieser Matrix sollen anschließend die Möglichkeiten, sowie die Vor- und Nachteile, die ein Framework bietet, abgelesen werden können.

3.2 Funktionale und nichtfunktionale Anforderungen

Zunächst werden die funktionalen Anforderungen, also Anforderungen, die sich direkt auf die Funktionalitäten der Referenzanwendung beziehen, und die nichtfunktionalen Anforderungen definiert. Die funktionalen Anforderungen werden in folgende Bereiche unterteilt: Zugriffsmöglichkeiten und Nutzung von Hardware-Komponenten wie zum Beispiel Sensoren und Kamera, die Möglichkeiten bei der Gestaltung der grafischen Benutzeroberfläche, ... Danach werden noch die nichtfunktionalen Anforderungen beschrieben, welche in folgende Kategorien unterteilt sind: ...

3.2.1 FA: Nutzung von Hardware-Komponenten

- Accelerometer

Es muss möglich sein, die Accelerometer-API anzusprechen und so Daten vom Accelerometer abzugreifen und nutzen zu können.

- Vibration

Es muss möglich sein die Vibrationsfunktion des Smartphones anzusprechen und nutzen zu können.

- Kamera

Sowohl die Front- als auch die Rückkamera des Smartphones müssen über das Framework ansprechbar und benutzbar sein.

- GPS

Das GPS des Smartphones muss nutzbar sein, um die aktuelle Position des Gerätes ermitteln und anzeigen lassen zu können.

- Speicher

Es muss möglich sein Dateien in den lokalen Speicher des Smartphones zu schreiben, lesen und auch wieder zu löschen.

- Netzwerknutzung

Es muss möglich sein in einer mit dem zu testenden Framework gebauten Anwendung Zugriff zum Internet zu bekommen und gegebenenfalls Dateien herunterladen zu können.

3.2 Funktionale und nichtfunktionale Anforderungen

- Lagesensor/Gyroskop

Es soll geprüft werden, ob der Lagesensor von einer Anwendung, welche mit dem zu testenden Framework entwickelt wurde, ansprechbar ist um die Lage des Smartphones ermitteln zu können.

- Näherungssensor

Es soll geprüft werden, ob der Näherungssensor eines Smartphones muss weiterhin nutzbar ist.

- Notifications

Es soll überprüft werden, ob Notifications über eine in dem zu testenden Framework entwickelten Anwendung ausgegeben werden können.

- Kommunikation

Es soll getestet werden, ob Kommunikationen via Bluetooth und WiFi für Gerät-zu-Gerät- und Gerät-zu-Netzwerk-Kommunikation möglich sind.

- Speicher

Es soll geprüft werden, ob Dateien in den lokalen Speicher des Smartphones geschrieben, gelesen und auch wieder gelöscht werden können. Hierbei soll zusätzlich überprüft werden, welche Datenhaltungsformate unterstützt werden.

3.2.2 FA: Die grafische Benutzeroberfläche

Ein natives Look-And-Feel einer mobilen Anwendung bedeutet, dass die grafische Benutzeroberfläche dem Benutzer das Gefühl einer nativen Anwendung vermittelt. Die ausgewählten Frameworks müssen hierzu die Möglichkeit bieten, native Bedienelemente bei der Entwicklung von Anwendungen zu verwenden. Es muss möglich sein die Oberfläche einer Anwendung nach dem Design-Standard des entsprechenden Betriebssystems (Android, iOS) aufzubauen. Hier wird im Falle des Betriebssystems Android der aktuelle Design-Standard ‚Material Design‘ (X) gefordert. Um dies zu testen ist es notwendig in die Test-Anwendung gängige Bedienelemente wie zum Beispiel Action-Bars, Floating Buttons und Navigationselemente zu integrieren.

3.2.3 FA: Betriebssysteme

Für jedes zu untersuchende Framework soll ermittelt werden, für welche Plattformen sich Anwendungen damit entwickeln lassen. Der Fokus liegt hierbei auf den mobilen Betriebssystemen Android und iOS, da diese den Großteil des Marktes ausmachen (vgl. Kapitel X). Hierbei soll auch berücksichtigt werden, wie viele und welche ältere Versionen der Betriebssysteme vom entsprechenden Framework unterstützt werden und inwieweit eine Anwendung abwärtskompatibel ist. Auch soll untersucht werden wie viel Zeit die Hersteller der Frameworks durchschnittlich benötigen um auf ein Betriebssystemupdate seitens iOS und Android zu reagieren.

Zudem wird ermittelt auf welchen Betriebssystemen die Frameworks für die Entwicklung installiert werden können.

3.2.4 FA: Sicherheit

Bezüglich der Sicherheitsaspekte von mobilen Anwendungen soll überprüft werden, ob und inwieweit das Rechtemanagement der jeweiligen Plattform (Android, iOS) nativ unterstützt wird. Dies beinhaltet unter anderem die Regelung der Zugriffe auf Gerätetfunktionen wie Speicher oder die Kameras. Auch wird untersucht, ob es Möglichkeiten zur Hinterlegung von Sicherheitszertifikaten gibt und ob ein eigener privater Zertifikatspeicher eingerichtet werden kann.

3.2.5 FA: Interoperabilität/Erweiterbarkeit

Unter dem Sammelpunkt Interoperabilität und Erweiterbarkeit finden sich Prüfungen, die sich auf eine mögliche Anpassbarkeit der Frameworks und die Nutzung von Bibliotheken von Fremdanbietern beziehen. In diesem Zuge wird auch ermittelt, ob sich die Frameworks in gängige IDEs, wie zum Beispiel Visual Studio oder Eclipse integrieren lassen und ob es eine Auswahl an Programmiersprachen gibt, mit denen entwickelt werden kann.

3.2.6 FA: Tests und Performance

Bei der Evaluierung von Frameworks für Crossplatform- beziehungsweise hybrider Anwendungs-entwicklung wird auch untersucht inwiefern die einzelnen Frameworks

3.2 Funktionale und nichtfunktionale Anforderungen

Werkzeuge zum Schreiben und Durchführen von (automatisierten) Tests selbst anbieten oder unterstützen. Die Anwendung, die als Evaluationshilfe im Rahmen dieser Arbeit entwickelt wird, soll zudem Möglichkeiten bieten die Performance der Cross-Platform-Anwendungen mit einer nativen Anwendung zu vergleichen. Hierzu muss die zu entwickelte Test-Anwendung Messungen der Reaktionszeiten der Anwendung ermöglichen.

3.2.7 NFA: Hersteller

Als eine der nichtfunktionalen Anforderungen wird die Bedeutsamkeit der Hersteller/Anbieter der einzelnen Frameworks recherchiert. Hierzu zählen Attribute wie Unternehmensgröße, Bekanntheitsgrad (Score), Produktpalette und Einsätze. Zudem werden in diesem Rahmen auch die Entwicklungsstadien der Frameworks näher durchleuchtet und Fragen hinsichtlich der aktuellen Version und der Anzahl vergangener Updates beantwortet. Auch wird untersucht, in welchem Umfang versprochene Funktionalitäten bereits umgesetzt wurden und welche Erweiterungen und Verbesserungen in Planung sind.

3.2.8 NFA: Support und Community

Die Untersuchung des Umfangs des Hersteller-Supports findet unter Berücksichtigung folgender Gesichtspunkte statt: Werden Tutorials und Beispielelösungen für verschiedene Problemstellungen geboten? Werden Online-Schulungen und/oder Schulungen mit Anwesenheit angeboten? Gibt es eine Dokumentation, die regelmäßig gepflegt wird? Gibt es ein Forum, in dem Fragen gestellt werden können, die von einem Support-Team des Herstellers beantwortet werden?

Zusätzlich zum Hersteller-Support soll auch die Größe der jeweiligen Community betrachtet werden. Die Größe wird dabei anhand der Internetpräsenz bemessen.

3.2.9 NFA: Entwicklung

Ein wichtiger Faktor bei der Evaluation von Frameworks für hybride und Cross-Platform-Anwendungsentwicklung ist der Entwicklungsprozess mit den entsprechenden Frameworks. Unter diesem Aspekt werden unter anderem die Leichtigkeit der Installation und der Benutzung der Frameworks verglichen. Gemessen wird dies an

3.3 Abgrenzungskriterien

der notwendigen Einarbeitungszeit und der Zeit die insgesamt für Installation und Einrichtung aller für die Entwicklung notwendigen Werkzeuge benötigt wird. Hier fließt auch der Umfang der Bibliotheken, welche die Frameworks mit sich bringen, in Form folgender Fragestellungen ein: Welche Bausteine für die Entwicklung werden angeboten und was muss alles selbst implementiert werden? Zentrales Kriterium ist auch der Anteil des plattformübergreifend nutzbaren Codes am Gesamtcode, hier am Beispiel der Funktionstest-Anwendung, welche als Werkzeug zur Evaluation dient.

3.2.10 NFA: Lizenz und Kosten

Bei der Evaluation werden auch etwaige Kosten und Lizenzmodelle der zu untersuchenden Frameworks betrachtet. Hier werden die einzelnen Frameworks hinsichtlich etwaiger Kaufpreise, Monats- oder Jahreslizenzpreise und Kosten für Support sowie Minor und Major Upgrades untersucht.

3.3 Abgrenzungskriterien

Im Rahmen dieser Projektarbeit wird eine Marktanalyse durchgeführt, um die 5 aktuell relevantesten Frameworks aus dem Bereich hybrider und Cross-Platform-Anwendungsentwicklung auszuwählen. Die als Evaluationswerkzeug dienende Funktionstest-Anwendung wird einmal nativ als Referenz und anschließend nur mit diesen 5 ausgewählten Frameworks implementiert. Weitere Frameworks werden nur aufgrund ihrer Spezifikationen und Literatur untersucht. Aufgrund der zur Verfügung stehenden Hardware und Software wird die native Variante der mobilen Anwendung, welche als Evaluationswerkzeug entwickelt wird, ausschließlich in Android für ein Android Smartphone entwickelt. Somit werden sich sämtliche Teile der Evaluation, welche mit Hilfe der Funktionstest-Anwendung erarbeitet werden, nur auf die Android-spezifischen Bereiche der Frameworks beziehen können. Bei der Evaluation des nativen Look-And-Feel anhand des GUI-Designs wird sich an den aktuellen Richtlinien für das Material Design von Android orientiert.

4 Marktanalyse

Wie schon in der Anforderungsanalyse (Kapitel (X)) beschrieben, werden 3 Frameworks für hybride, bzw. Cross-Platform-Anwendungsentwicklung ausgewählt, um diese mit einer Funktionstest-Anwendung zu evaluieren.

Bevor eine allgemeine Liste aktuell verfügbarer Frameworks aufgestellt wurde, wurden folgende Kriterien festgelegt, welche auf jeden Fall von den Frameworks erfüllt werden müssen, um bei dieser Evaluation Berücksichtigung zu finden: Die Frameworks müssen mehr als eine Plattform bedienen, und 2 dieser Plattformen müssen Android und iOS sein, da diese den größten Marktanteil besitzen. Wobei Android einen durchschnittlichen Marktanteil von ca. 75% und iOS von ca. 18% über die vergangenen 4 Jahre besaß¹. Folgende Frameworks konnten nach oben genannten Kriterien ausfindig gemacht werden:

- React Native
- CodenameOne
- Xamarin
- Ionic
- Intel XDK
- Onsen UI
- Kendo UI
- Mobile Angular UI
- Cordova

¹Statista.

3.3 Abgrenzungskriterien

- PhoneGap
- J2ObjC
- AppGyver
- ViziApps
- Monocross
- Marmalade
- Corona
- Agate
- Apache Flex
- AppConKit

Um die 3 aktuell relevantesten Frameworks zu ermitteln, wurden folgende Medien genutzt: Mit dem Google AdWords Keyword Planer² wurden die durchschnittlichen Suchanfragen nach dem Namen des jeweiligen Frameworks pro Monat über den Zeitraum des letzten Jahres ermittelt. Hierbei ergab sich bei den oben aufgeführten Frameworks folgende Verteilung:

²Google Keyword Planer 2017.

3.3 Abgrenzungskriterien

Anzeigengruppen-Ideen	Keyword-Ideen	Spalten ▾	L	Herunterladen	Alle hinzufügen (19)
Keyword (nach Relevanz)	Durchschnittl. Suchanfragen pro Monat <small>?</small>	Wettbewerb <small>?</small>	Vorgeschlagenes Gebot <small>?</small>	Anteil an Anz.im	Zu Plan hinzufügen
Corona	450.000	Niedrig	0,04 €		»
Xamarin	201.000	Niedrig	0,82 €		»
Ionic	165.000	Niedrig	1,86 €		»
Cordova	135.000	Niedrig	0,65 €		»
PhoneGap	135.000	Niedrig	1,96 €		»
Agate	135.000	Niedrig	0,42 €		»
React Native	90.500	Niedrig	4,13 €		»
Marmalade	90.500	Niedrig	1,31 €		»
Kendo UI	40.500	Niedrig	1,68 €		»
Intel XDK	22.200	Niedrig	0,71 €		»

Zellen anzeigen: 10 ▾ 1 - 10 von 19 Keywords | < < > >|

Anzeigengruppen-Ideen	Keyword-Ideen	Spalten ▾	L	Herunterladen	Alle hinzufügen (19)
Keyword (nach Relevanz)	Durchschnittl. Suchanfragen pro Monat <small>?</small>	Wettbewerb <small>?</small>	Vorgeschlagenes Gebot <small>?</small>	Anteil an Anz.im	Zu Plan hinzufügen
Onsen UI	9.900	Niedrig	–		»
AppGyver	3.600	Niedrig	–		»
Apache Flex	2.900	Niedrig	2,89 €		»
Mobile Angular UI	1.900	Niedrig	3,70 €		»
J2ObjC	1.600	Niedrig	–		»
CodenameOne	880	Niedrig	–		»
Monocross	590	Niedrig	–		»
ViziApps	390	Niedrig	–		»
AppConKit	40	Niedrig	–		»

Zellen anzeigen: 10 ▾ 11 - 19 von 19 Keywords | < < > >|

Abbildung 4.1: Durchschnittliche Suchanfragen bei Google, ermittelt mit Google Ad-words Keyword Planer

3.3 Abgrenzungskriterien

Zusätzlich wurden mit Hilfe von Google Trends³ die Popularität der einzelnen Frameworks im Zeitverlauf visualisiert, um zu verdeutlichen welche Frameworks in letzter Zeit an Interessierten dazugewonnen haben. Die Ergebnisse werden bei Google Trends in Relation zum totalen Suchaufkommen gesetzt. Die Trends sind für einzelne Regionen sowie für die ganze Welt verfügbar. In diesem Fall wurden die weltweiten Trends für die Frameworks ausgewertet. Da sich nur 5 Suchbegriffe gleichzeitig in einem Graphen gegenüberstellen lassen, wurden immer 4 Frameworks "Corona", welches in der vorherigen Untersuchung mit dem Keyword Planer⁴ die meisten monatlichen Suchanfragen aufwies, als Referenz gegenübergestellt. Da einige Frameworks Namen haben, die auch in völlig anderen Zusammenhängen gesucht werden können, wie zum Beispiel "Marmalade", wurde als Suchkategorie "Computer und Elektronikäusgewählt.



Abbildung 4.2: Google Trends: React Native, CodenameOne, Xamarin und Ionic im Vergleich zu Corona

³Google Trends 2017.

⁴Google Keyword Planer 2017.

3.3 Abgrenzungskriterien



Abbildung 4.3: Google Trends: Intel XDK, Onsen UI, Kendo UI und Mobile Angular UI im Vergleich zu Corona



Abbildung 4.4: Google Trends: Cordova, PhoneGap, J2ObjC und AppGyver im Vergleich zu Corona

3.3 Abgrenzungskriterien



Abbildung 4.5: Google Trends: ViziApps, Monocross, Marmalade und Agate im Vergleich zu Corona



Abbildung 4.6: Google Trends: Apache Flex und AppConKit im Vergleich zu Corona

4.1 Xamarin

Aus oben beschriebenen und dargestellten Auswertungen ergeben sich zunächst folgende 5 Frameworks in der engeren Auswahl, welche durch die Implementierung einer Funktionstest-Anwendung evaluiert werden sollen:

- Corona
- Xamarin
- Cordova
- PhoneGap
- Ionic

Nach weiteren Recherchen über diese 5 Frameworks stellte sich allerdings heraus, dass es sich bei Cordova um eine Weiterentwicklung des Frameworks PhoneGap handelt. Dies schlägt sich auch in den Google Trends nieder, wie man in Abbildung 4.4 erkennen kann. Aus diesem Grund wird ausschließlich Cordova evaluiert, der Vorgänger PhoneGap bleibt außen vor. Für die Liste der 5 Frameworks, welche praktisch evaluiert werden, ergibt sich daraus, dass an die Stelle von PhoneGap nun "React Native" tritt.

Das Framework Cordova bietet keine Mittel für die Realisierung einer grafischen Benutzeroberfläche und für das Framework Ionic ist für sämtliche Hardware-Nutzung das Einbinden von Cordova-Plugins notwendig. Aus diesen Gründen werden diese beiden Frameworks in dieser Arbeit kombiniert betrachtet, als 2 Bausteine eines Frameworks sozusagen. So ergeben sich für die Evaluation in dieser Arbeit die Frameworks Xamarin, React Native und Cordova mit Ionic. Diese Frameworks werden nachfolgend etwas näher vorgestellt.

4.1 Xamarin

Das Xamarin Framework verspricht native Anwendungen für Windows Phone, iOS und Android entwickeln zu können basierend auf einer C# Codebase. Konkret gesprochen verspricht es eine native UI, nativen API Zugang und native Performance. Xamarin kommt mit der eigenen IDE 'Xamarin Studio', es ist allerdings auch möglich Xamarin Anwendungen mit der IDE 'Visual Studio' zu entwickeln. Die Benutzeroberfläche muss bei Xamarin Anwendungen allerdings für jede Plattform separat implementiert werden(Zeitschrift(X)).

4.2 Cordova

Cordova von Apache ist ein Open Source Framework für Cross-Platform mobile Anwendungsentwicklung. Das Framework ist kostenlos nutzbar und unterstützt Standard Web Technologien wie HTML5, CSS3 und JavaScript für die Entwicklung. Die Anwendungen werden in sogenannten Wrappern ausgeführt, die die API der jeweiligen Plattform ansprechen um so Zugang zu unter anderem Sensoren zu bekommen. Die Plattformen, die von Apache Cordova unterstützt werden sind: Android, BlackBerry 10, iOS, OS X, Ubuntu, Windows und Windows Phone 8⁵.

4.3 Ionic

Das Ionic Framework basiert auf AngularJS, was wiederum ein clientseitiges JavaScript-Webframework ist⁶. Entsprechend werden Ionic Anwendungen ebenfalls in JavaScript, HTML5 und CSS3 geschrieben. Auf diese Weise wird bei diesem Framework die gemeinsame Code-Basis geschaffen. Ionic bietet eine nativ wirkende GUI, für Hardware-Anbindungen, wie zum Beispiel das Nutzen von Sensoren, müssen allerdings Cordova (4.2) PlugIns genutzt werden⁷.

4.4 React Native

React Native nutzt die Bibliothek React von Facebook zur Entwicklung für die Plattformen Android, iOS und die Universal Windows Platform. Programmiert werden React Native Anwendungen in JavaScript und HTML. Der Code wird anschließend in native Quelldateien übersetzt, weswegen React Native verspricht native Komponenten ohne großen Aufwand einbinden zu können und flüssige Animationen zu ermöglichen(Zeitschrift(X)).

⁵Apache Cordova 2017.

⁶AngularJS: Ionic 2017.

⁷Ünlü 2016.

5 Aufstellen der Bewertungskriterien für die Analyse

Für die Evaluierung der Frameworks wird zunächst ein Katalog mit Bewertungskriterien angefertigt, anhand dessen die einzelnen Frameworks bewertet werden. Der Katalog besteht dabei aus Kriterien, nach denen alle in Kapitel X vorgestellten Frameworks allein durch Recherche untersucht werden können. Zusätzlich werden noch Kriterien aufgenommen, die sich auf die praktische Anwendung der in Kapitel X ausgewählten 5 Frameworks beziehen, mit welchen die in Kapitel X vorgestellte Funktionstest-Anwendung testweise implementiert wird.

Um oben beschriebenen Kriterienkatalog aufzustellen, wurde zunächst eine Umfrage durchgeführt. Die Umfrage richtete sich an (ehemalige) Studierende der Fachhochschule Dortmund, welche angaben, bereits Erfahrungen in der mobilen Anwendungsentwicklung gesammelt zu haben. Die Fragestellung der Umfrage lautete: „Welche Kriterien halten Sie für wichtig in der Entwicklung mit Frameworks für hybride, bzw. Cross-Platform-Anwendungen?“ (E-Mail siehe Anhang) Die Befragten antworteten hierauf mit einem Freitext, indem sie die für sich wichtigen Kriterien zur Bewertung von Frameworks aufzeigten. Diese Kriterien wurden in einem Dokument (Anhang) gesammelt und Mehrfachnennungen entsprechend gekennzeichnet. Ein Kriterium, welches von mehreren Befragten als wichtig beschrieben wurde, bekommt bei der Evaluation eine höhere Gewichtung zugeschrieben, als ein Kriterium, welches nur von einem Befragten genannt wurde. Aus dieser Zusammenfassung der ermittelten Bewertungskriterien wurde anschließend eine Matrix (siehe Abbildung X-X) erstellt. Der Grad der Erfüllung eines Kriteriums wird an der Anzahl an Punkten zu erkennen sein, die vergeben werden. Hier ist eine Skala von 1 bis 5 (gar nicht bis voll erfüllt) vorgesehen. Wobei hier zu jedem Kriterium zunächst definiert werden muss, was „voll erfüllt“ bedeutet. Neben diesen Kriterien wird es auch Kriterien geben, welche nur mit „vorhanden“ und „nicht vorhanden“ bewertet werden können.

5.1 Kategorie Kosten und Lizenz

5.1 Kategorie Kosten und Lizenz

In der Kategorie Kosten und Lizenz gibt es einen Sonderfall bei dem Punkt „Lizenzmodell“. Es ist hier nicht möglich eine konkrete Abstufung von „gut“ nach „schlecht“ in 5 Kategorien zu schaffen, da verschiedene Lizenzmodelle verschiedene Vor- und Nachteile aufweisen können. So wird sich darauf beschränkt, lediglich eine kurze Beschreibung des jeweiligen Lizenzmodells, sofern es eins gibt, zu formulieren.

Die Punkte „Fixkosten“, „Lizenzkosten“ und „Supportkosten“ werden in 5 Abstufungen von „gratis“ bis „teuer“ unterteilt. Die exakten Geldwerte für die Grenzen der Abstufungen werden sich aus dem günstigsten und teuersten Preis, welche sich zu den Frameworks ermitteln lassen, ergeben.

5.2 Kategorie Support und Community

Bezüglich des Punktes 2)a) „Support/Wartung vom Hersteller“ (siehe Abbildung X) wird der Umfang des Supports pro Framework ermittelt und anhand dessen die Abstufung vorgenommen. Dies umfasst zum Beispiel telefonischen Support, ein Ticket-System oder ein moderiertes Forum. Die Qualität von Punkt 2)b) „Dokumentation“ wird unter anderem am Umfang der Dokumentation selber und am Update-Zyklus bemessen, d.h. wie schnell wird die Dokumentation entsprechend Änderungen am Framework aktualisiert? Bei der Untersuchung der Größe der Community werden, falls vorhanden, das Framework eigene Forum und weitere Entwicklerforen nach Anzahl Threads mit Themen, die sich auf das jeweilige Framework beziehen, durchleuchtet. Das Schulungsangebot wird sowohl nach Qualität als auch nach Quantität beurteilt: Das bedeutet, dass unterschieden wird, ob nur Online- oder auch Anwesenheitsschulungen angeboten werden. Hierbei spielt es auch eine Rolle an wie vielen unterschiedlichen Standorten und wie häufig Schulungen angeboten werden. Zudem wird der Angebotsumfang an Schulungen hinsichtlich einer Aufteilung von Anfänger- bis Spezialschulungen bewertet.

5.3 Kategorie Entwicklung

Die Kategorie „Entwicklung“ (siehe Abbildung X) betrifft nur die 5 ausgewählten Frameworks, mit denen die Funktionstest-Anwendung entwickelt wird. Für die Leich-

5.4 Kategorie Hersteller

tigkeit der Installation und Benutzung des Frameworks werden Punkte von 1 bis 5 vergeben: 1 bedeutet kompliziert und aufwändig und 5 bedeutet automatisiert und intuitiv. Als „Einarbeitungszeit“ wird in diesem Falle die Zeit von der Fertigstellung der Installation bis nach dem Schreiben der ersten funktionierenden Seite der Anwendung angesehen. Hier gilt je kürzer, desto besser. Bei Punkt 3)c) „plattformübergreifend nutzbarer Code“ inwieweit der mit dem jeweiligen Framework erzeugte Code plattformübergreifend nutzbar ist, beziehungsweise wie hoch der Anteil des plattformspezifischen Codes ist. Hier gilt entsprechend, je mehr Code plattformübergreifend nutzbar ist, desto besser. Unter „Umfang Bibliotheken/Templates“ wird verstanden, wie viele Bausteine mit dem Framework sozusagen mitgeliefert werden, die einzelne Funktionalitäten kapseln und einfach in den Code zu integrieren sind.

5.4 Kategorie Hersteller

Die „Bekanntheit/Größe des Herstellerunternehmens“ des jeweiligen Frameworks wird anhand der Unternehmensgröße, des Bekanntheitsgrads, der Produktpalette und Einsätze der Produkte gewertet. Bei dem Punkt „Entwicklungsstadium des Frameworks“ (siehe Abbildung X) wird in die Wertung mit einbezogen, ob sich das Framework noch in einer Beta-Version befindet, oder es schon ein Release gibt. Zudem wird betrachtet, wie viele Updates und Bugfixes bereits vorgenommen wurden und wie viel von dem was das Framework laut Hersteller leisten soll bereits umgesetzt oder noch in Planung ist. Eine „Ja“-„Nein“-Unterscheidung gibt es bei dem Punkt „Weiterentwicklung des Frameworks“. Hierbei ist positiv zu bewerten, wenn das Framework nachweislich noch weiterentwickelt werden soll, um mit den Änderungen und der Weiterentwicklung der einzelnen Plattformen mithalten zu können.

5.5 Kategorie OS-Versionen

„Reaktionszeit auf OS-Update“ (siehe Abbildung X) beschreibt die Zeit, die benötigt wurde um nach einem Update eines Betriebssystems, wie zum Beispiel Android, das Framework entsprechend anzupassen oder zu erweitern. Mit dem Punkt „Abwärtskompatibilität“ wird bewertet, bis zu welcher OS-Version das Framework anwendbar ist. Hier ist zum Beispiel für Android nicht ohne Bedeutung, dass weiterhin Version 4.x unterstützt wird, die unter den Android Versionen den größten Marktanteil annimmt (QUEL-

5.6 Funktionsumfang

LE!). Desweiteren werden bewertet, auf wievielen unterschiedlichen Plattformen das Framework installiert und entwickelt werden kann und für wieviele mobile Plattformen entwickelt werden kann.

Kategorie			
2)	Kosten, Lizenz		
a)	Fixkosten Framework		
b)	Subscription		
c)	Supportkosten		
d)	Lizenzmodell (informative Angabe)		
3)	Support, Community		
a)	Support/Wartung vom Hersteller		
b)	Dokumentation		
c)	Größe der Community/Hilfe im Netz		
d)	Schulungen		
4)	Entwicklung		
a)	Leichtigkeit Installation/Benutzung		
b)	Einarbeitungszeit		
c)	Anteil des plattformübergreifend nutzbaren Code vom Gesamtcode		
d)	Umfang der Bibliotheken des Frameworks (was muss alles selbst implementiert werden?)		
5)	Hersteller		
a)	Bekanntheit/Größe des Hersteller(-unternehmens)		
b)	Entwicklungsstadium des Frameworks		
c)	Wird das Framework weiterentwickelt?		
6)	OS-Versionen		
a)	Reaktionszeit auf OS-Update?		
b)	Abwärtskompatibilität		
c)	Auf welchen Systemen läuft das Framework?		
d)	Unterstützte OS (mobil)		

Abbildung 5.1: Matrix der Bewertungskriterien, Teil 1

5.6 Funktionsumfang

In der Kategorie Funktionsumfang wird eine "Ja"NeinUnterscheidung bei der Fragestellung, welche Sensoren angesprochen und genutzt werden können, vorgenommen. Informationen darüber, welche Sensoren mit dem jeweiligen Framework genutzt werden können, bieten die jeweiligen Spezifikationen. Zusätzlich wird der Funktionsumfang der in Kapitel (X) ausgewählten 5 Frameworks mit der Funktionstest-Anwendung praktisch getestet. So kann bei diesen 5 Frameworks zusätzlich eine Bewertung über die Anwendbarkeit vergeben werden. Die Sensoren, die getestet werden sind: Der Beschleunigungssensor, der Lagesensor mit dem Gyroskop, das GPS, der Näherungssensor, das Magnetometer und das Barometer (siehe Abbildung X).

Neben den oben beschriebenen Sensoren werden auch Funktionen wie die Benutzung der Front- und Rückkamera, der Zugriff auf das Dateisystem des Smartphones

5.7 GUI-Design

und Notifications getestet. Hier bekommen die in Kapitel (X) ausgewählten Frameworks wie bei den Sensoren eine erweiterte Bewertung. Bei Punkt 6)6) "Kommunikation" fließt in die Bewertung mit ein, welche und wieviele unterschiedliche Kommunikationswege genutzt werden können, wie zum Beispiel Bluetooth oder WiFi.

5.7 GUI-Design

Nur einen "Ja"NeinUnterschied gibt es bei den Punkten 7)1) "GUI-Designer" und 7)3) "integrierter Emulator", welcher schlicht besagt, ob diese Werkzeuge in dem jeweiligen Framework enthalten sind oder nicht (siehe Abbildung X). Punkt 7)2) "Umfang der verfügbaren (nativen) GUI-Elemente" wird hauptsächlich bei den in Kapitel (X) ausgewählten Frameworks Anwendung finden. Hier kann getestet werden, welche unterschiedlichen Elemente des Android Material Designs einfach einsetzbar bar sind und für welche Elemente aufwändige oder weniger aufwändige Imports von Bibliotheken notwendig sind. Zudem wird der Aufwand bewertet entsprechende GUI-Elemente in die Anwendung zu integrieren.

5.8 Interoperabilität und Erweiterbarkeit

Punkte 8)1) "Webserviceaufrufe" und 8)2) "Bibliotheken von Fremdanbietern" werden mit einer "Ja"NeinUnterscheidung bewertet (siehe Abbildung X), d.h. es wird überprüft, ob Webserviceaufrufe und Einbindung fremder Bibliotheken mit dem jeweiligen Framework möglich sind. Mit Punkt 8)4) "Support-Tools" wird bewertet, ob das Framework Support-Tools zur Verfügung stellt. Bei Punkt 8)3) "Integration in IDEs" wird ermittelt, wieviele unterschiedliche IDEs für die Entwicklung mit dem jeweiligen Framework einsetzbar sind, d.h. in wieviele IDEs das Framework integriert werden kann.

5.8 Interoperabilität und Erweiterbarkeit

Kategorie	
7)	Funktionsumfang
a)	Beschleunigungssensor
b)	Lagesensor
c)	GPS
d)	Näherungssensor
e)	Magnetometer
f)	Gyroskop
g)	Barometer
h)	Umgebungslichtsensor
i)	Kamera vorn
j)	Kamera hinten
k)	Zugriff auf Dateisystem
l)	Speichern von Einstellungen/Konfigurationen
m)	Kommunikation (Bluetooth, WiFi, etc.)
n)	Caching
o)	Notifications
8)	GUI-Design
a)	GUI-Designer?
b)	Umfang der verfügbaren (nativen) GUI-Elemente
c)	Integrierter Emulator?
d)	Wie nah an antiver GUI?
9)	Interoperabilität/Erweiterbarkeit
a)	Lassen sich Webserviceaufrufe programmieren?
b)	Nutzen von Bibliotheken von Fremdanbietern
c)	Integration in vorhandene IDE?
d)	Support-Tools vorhanden?

Abbildung 5.2: Matrix der Bewertungskriterien, Teil 1

5.9 Tests

Bei den beiden Punkten 9)1) "Automatische Tests" und 9)2) "Test-Tools" wird allein mit einer "Ja"/"Nein"-Unterscheidung bewertet, ob automatische Tests möglich und Test-Tools mit dem Framework ausgeliefert werden.

5.10 Performance

Bei dem Bewertungskriterium "Performance" wird eine Abstufung entsprechend der Größe des Performance-Unterschieds zur nativen Anwendung vorgenommen. Hier werden nur die in Kapitel (X) ausgewählten 5 Frameworks bewertet werden können, mit denen die Funktionstest-Anwendung entwickelt wird.

5.11 Sicherheit

In dieser Kategorie werden folgende Fragestellungen beantwortet: 11)1): Wird das RechteManagement der jeweiligen Plattform nativ unterstützt? 11)3): Kann man Sicherheitszertifikate hinterlegen? 11)4): Kann ein eigener Zertifikatspeicher eingerichtet werden? 11)5): Werden VPN Verbindungen nativ umgesetzt (siehe Abbildung X)? Zudem werden informative Angaben zu der Realisierung des Zugriffs auf Gerätefunktionen wie Speicher oder Kamera gemacht.

Kategorie	
10)	Tests
a)	Automatische Tests möglich?
b)	Existieren Test-Tools?
c)	Speichergröße der Anwendung
11)	Performance
a)	Performance einzelner Funktionen im Vergleich zur nativen App
12)	Programmiersprache
a)	Mit welchen (wievielen) Programmiersprachen kann entwickelt werden?
13)	Sicherheit
a)	Wird das RechteManagement der jwl. Plattform nativ unterstützt?
b)	Wie ist der Zugriff auf Gerätefunktionen wie Speicher, Kamera realisiert?
c)	Kann man Sicherheitszertifikate hinterlegen?
d)	Kann ein eigener privater Zertifikatspeicher eingerichtet werden?
e)	Werden VPN Verbindungen nativ umgesetzt?

Abbildung 5.3: Matrix der Bewertungskriterien, Teil 1

6 Planung der Funktionstest-Anwendung

Anhand des durch die Anforderungsanalyse festgelegten Rahmens müssen nun Entscheidungen bezüglich Tests der Datenhaltung, der Sensoren und des Designs der Benutzeroberfläche und Benutzerführung getroffen werden. In den folgenden Abschnitten werden diese Entscheidungen behandelt. Wie schon in Kapitel (X) erläutert, wird die Funktionstest-Anwendung zunächst als Referenz für die Plattform Android nativ entwickelt. Entsprechend wird sich unter anderem bei der graphischen Benutzeroberfläche an den Android Guidelines orientiert, welche im Folgenden näher erläutert werden.

6.1 GUI Design

Die Android GUI-Guidelines beschreiben das Design-Prinzip von Android Anwendungen als 'Material Design'. Das Design, die Oberfläche soll materiell, greifbar wirken. Der Gebrauch von gewohnten, greifbaren Attributen soll dem Benutzer helfen schnell Funktionen und Bedienelemente zu verstehen. Licht, Oberflächenbeschaffenheiten und Bewegung beschreiben wie einzelne Objekte der GUI miteinander interagieren und in welcher Relation sie zueinander stehen. Die fundamentalen Elemente printbasierten Designs wie die Typographie, Raum, Farbe und Metaphorik dienen nicht nur optischen Highlights sondern sie formen Hierarchien, Bedeutung und Fokus. Im Material Design soll so eine Oberfläche geschaffen werden, die als 'bold and graphic' bezeichnet wird. Diese zeichnen gezielte Farbkombinationen, bewusst eingesetzte Leerräume und eine groß-skalierte Typographie aus. Die Bewegungen einzelner Bedienelemente sollen bedeutungsvoll sein und dazu dienen den Fokus auf sich zu ziehen um den Benutzer auf intuitive Art und Weise durch Anwendungen navigieren zu lassen. Auf diese Weise soll auch klar verständliches Feedback auf Benut-

6.1 GUI Design

zugegeben gegeben werden¹.

Die Android Design-Guidelines betonen folgende 3 Prinzipien:

- Enchant Me

Mit 'Enchant Me' ist gemeint, dass zum Beispiel vorsichtig platzierte Animationen und gut gesetzte Soundeffekte dafür sorgen, dem Benutzer ein Gefühl von Mühelosigkeit vermitteln sollen. Dem Benutzer soll die Möglichkeit gegeben werden anstelle von Buttons und Menüs direkt mit einzelnen Objekten in der Anwendung interagieren zu können. Dies soll die kognitive Anstrengung gering halten und die Anwendung ansprechender gestalten. Mit der Möglichkeit optionaler Anpassungen soll der Benutzer der Anwendung eine persönliche Note geben können. Zudem sollte eine Anwendung lernen, was die Präferenzen des Benutzers sind und entsprechende häufig verwendete Funktionen in den Vordergrund heben und leicht erreichbar anbieten².

- Simplify My Life

Unter 'Simplify My Life' zählt unter anderem, dass darauf geachtet werden sollte, kurze Sätze mit einfachen Wörtern zu verwenden, da lange Sätze eher eine abschreckende Wirkung haben. Zudem sollten, wo es nur möglich ist, Bilder und Symbole anstelle von Text genutzt werden, um Ideen zu erklären. Einstellbare, bzw. konfigurable Funktionen sollten voreingestellt sein, um den Benutzer vor der ersten Nutzung der Anwendung einen Einstellungsmarathon zu ersparen. Es sollte dem Benutzer aber die Möglichkeit gegeben werden die Konfigurationen einfach nach seinem Geschmack anzupassen. Für eine bessere Übersichtlichkeit sollte auch darauf geachtet werden nicht zu viele Optionen und Informationen gleichzeitig anzuzeigen und so die Anzeige zu überladen und den Benutzer zu überfordern. Verschiedene Bereiche in einer Anwendung sollen verschiedenen aussehen und es sollen Übergänge zwischen Bereichen geschaffen werden, die deren Beziehungen untereinander verdeutlichen. Auf diese Weise soll der Benutzer die Orientierung in der Anwendung nicht verlieren.

¹Android Guidelines 2017.

²Android Design Principles 2017.

6.1 GUI Design

Des weiteren soll darauf geachtet werden, dass Formen, hinter denen bestimmte Funktionen stecken, ausschließlich für diese Funktionen genutzt werden, frei nach dem Motto 'Was gleich aussieht, sollte gleich funktionieren'. Als letztes wird unter dem Punkt 'Simplify My Life' aufgeführt, dass der Benutzer nur dann bei der Bedienung unterbrochen werden soll, wenn diese Unterbrechung dazu dient wirklich kritische Informationen zu übermitteln. Wegen unnötiger Details sollte der Benutzer nicht gestört werden³.

- Make Me Amazing

'Make Me Amazing' besagt unter anderem, dass man versuchen sollte visuelle und haptische Muster und Gesten aus anderen Android Anwendungen in die eigene Anwendung mit einfließen zu lassen. So hat es der Benutzer nochmals leichter sich in der neuen Anwendung zurecht zu finden. Werden von dem Benutzer Korrekturen verlangt, so sollten diese ihm freundlich vermittelt werden. Wenn etwas schief läuft, sollen klare verständliche Wiederherstellungsanweisungen gegeben werden, auf technische Details sollte verzichtet werden. Alles was im Hintergrund ohne den Benutzer ablaufen kann, sollte dort geschehen. Auf sämtlichen Benutzereingaben sollte es ein Feedback geben, auch wenn es nur ein leiches leuchten ist. Können mehrere komplizierte Arbeitsschritte zusammengefasst werden, sollte man dies tun und es dem Benutzer als Bündel zur Verfügung stellen. Ein Beispiel hierfür wären einige kombinierte Filter für die Fotonachbereitung. Zu guter Letzt wird darauf hingewiesen, dass darauf geachtet werden soll, dass die wichtigsten Funktionen der Anwendung leicht zu finden und zu benutzen sein sollen. Als gutes Beispiel dient hier der Pause-Button bei Musik-Anwendungen oder der Auslöser bei der Kamera⁴.

Für die geplante Funktionstest-Anwendung bedeutet dies, dass zumindest die Standard 'Material Design' Bedien- und Navigationselemente integriert werden sollten. Das Android Studio bietet einige Kompositionen an Elementen für die Seitenerstellung an, an diese sollte sich gehalten werden, da sie dem 'Material Design'-Prinzip entsprechen. Zu diesen Elementen zählen zum Beispiel der 'Navigation Drawer' für die Navigation zwischen einzelnen Seiten der Anwendung oder der 'Floating Action

³ *Android Design Principles* 2017.

⁴Ebd.

6.2 Hardwarezugriffe

Button', ein runder Button, welcher im Vordergrund über dem Inhalt der Anwendung schwebt. Er verschwindet nicht, wenn hinter ihm in der Seite gescrollt wird und steht so immer zur Verfügung. Auch die 'Bottom Navigation Bar' darf in der Anwendung nicht fehlen, sie ist eine einheitliche globale Navigation durch das Android Betriebssystem. Auch eine 'App Bar' soll am oberen Rand des Displays integriert werden. Diese stellt Shortcuts und weitere Menüverwaltungsmöglichkeiten zur Verfügung. In unten stehender Abbildung (X) ist eine leere Beispieleseite einer Android Anwendung im 'Material Design' Abgebildet. Man erkennt deutlich die 'Bottom Navigation Bar' am unteren Rand der Seite, die 'App Bar' am oberen Rand und den 'Floating Action Button' in der rechten unteren Ecke.

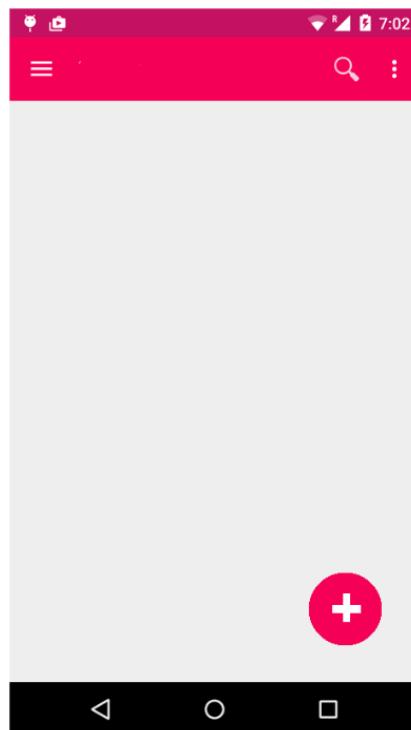


Abbildung 6.1: Beispiel einer leeren Seite einer Android Anwendung im 'Material Design'

6.2 Hardwarezugriffe

Android bietet für den Zugriff auf Sensoren auf der API-Ebene im Paket android.hardware Unterstützung für eine Vielzahl von Sensoren an. In der Sensor-Klasse ist für jeden Sensortyp eine Konstante definiert, wie zum Beispiel

6.3 Speicher

Sensor.TYPE_ACCELEROMETER für den Beschleunigungssensor. Einige dieser Sensoren repräsentieren dabei tatsächliche Hardware-Sensoren, andere wiederum sind Software-basierte Sensoren, deren Messungen auf einer Kombination von Hardware-Sensoren basieren(Buch Android5 Kapitel 15 (X)). Da es Smartphones gibt die pro Sensortyp mit mehreren Sensoren ausgestattet sein können, gibt es pro Typ in Andro- id einen sogenannten Default-Sensor und eine Sensorliste, in welcher alle verfügbaren Sensoren eines Typs enthalten sind. Über den Reporting-Mode kann eingestellt wer- den, wann der Sensor Daten an die Anwendung liefern soll. Hier gibt es unter ande- rem die Möglichkeit kontinuierlich Daten übermitteln zu lassen oder nur bei Änderungen in der Messung(Buch Andorid-Quick-API-References Kapitel 8 (X)).

Für die Standortbestimmung bietet Android eine eigene, separate API. Aufgrund datenschutzrechtlicher Bestimmungen müssen bei der Standortbestimmung Berech- tigungen direkt vom Benutzer eingeholt werden um diese nutzen zu können. Die eigentliche Bestimmung des Standortes kann auf 3 unterschiedliche Weisen gesche- hen: einmal via GPS (X), via WiFi Zugangspunkten oder Sendemasten oder als dritte Möglichkeit passiv über die Standortanfragen anderer auf dem Smartphone befind- licher Anwendungen(Buch Andorid-Quick-API-References Kapitel 8 (X)).

Für die Interaktion mit den Kameras des Smartphones bietet Android ein reichhalti- ges Set an APIs. Ähnlich zur Standortbestimmung werden auch bei der Nutzung der Kameras Berechtigungen gefordert. Der Entwickler kann im Manifest seiner Android Anwendung einstellen, ob die Kameranutzung für die Anwendung notwendig oder optional ist. Eine Anwendung, für die die Kameranutzung notwendig ist, kann über Google Play nicht auf Geräten ohne Kamera installiert werden. Android stellt zudem noch einige Konfigurationsparameter bereit, über die sich zum Beispiel die Qualität des JPEG-Bildes oder Blitzfunktionen voreinstellen lassen(Buch Andorid-Quick-API- References Kapitel 9 (X)).

6.3 Speicher

Für das Speichern von Daten gibt es je nach Zweck und Datengröße unterschiedliche Möglichkeiten bei Android. So können zu jeder Activity über sogenannte Shared Preferences Konfigurationen gespeichert werden. Die auf diese Weise gespeicherten Daten sind Privateigentum der jeweiligen Activity, keine andere Activity oder gar

6.3 Speicher

Anwendung hat Zugriff darauf(Buch Android5 Kapitel 12 (X)).

Bei größeren Datenmengen und für direktes Lesen und Schreiben in Dateien ist der Shared Preferences Ansatz eher ungeeignet. Hier werden Java-IO-Klassen wie zum Beispiel der FileInputStream oder der FileOutputStream verwendet. Als Speicherorte gibt es den internen Speicher (Internal Storage) und den externen Speicher (External Storage)(Buch Android5 Kapitel 12 (X)):

- Interner Speicher:

Der interne Speicher ist immer vorhanden, die enthaltenen Daten sind Anwendungs-spezifisch. Dies bedeutet, dass nur die Anwendung, der die Daten gehören Zugriff auf diese besitzt. Andere Anwendungen können diese Daten nicht nutzen. Wird die Anwendung deinstalliert, so werden in diesem Zuge auch alle Daten in ihrem internen Speicher gelöscht(Buch Android5 Kapitel 12 (X)).

- Externer Speicher:

Daten im externen Speicher können von mehreren Anwendungen genutzt werden. Der externe Speicher kann in Form einer einsteckbaren SD-Karte vorliegen, aber auch der Speicher des Smartphones hat einen Bereich, der als 'External Storage' angesprochen werden kann. Je nach Android-Version gibt es allerdings Unterschiede bezüglich der Zugriffsrechte und der Unterteilung des Speichers(Buch Android5 Kapitel 12 (X)).

In der Funktionstest-Anwendung soll die Verwendung des externen Speichers innerhalb des Smartphones, also ohne extra SD-Karte integriert werden. Hierbei soll sowohl das Lesen als auch das Schreiben von Dateien umgesetzt werden.

7 Objektorientierte Analyse und Design

Im Folgenden wird die objektorientierte Analyse, kurz OOA, beschrieben. In der Analyse geht es darum, die Anforderungen an die Anwendung unter anderem grafisch in Diagrammen zu erfassen und zu beschreiben, mit dem Ziel, ein objektorientiertes Analyse-Modell als Ergebnis zu erhalten. Anschließend wird im objektorientierten Design, kurz OOD, das in der Analyse erstellte Modell weiterentwickelt und in eine konkrete Softwarearchitektur überführt, die dann als direkte Implementierungsvorlage dient¹.

Zunächst werden in der objektorientierten Analyse das statische Modell und das dynamische Modell beschrieben. In der Regel wird das statische Modell anhand eines Klassendiagramms beschrieben, in dem alle notwendigen Klassen und Beziehungen dargestellt werden, um die in der Anforderungsanalyse definierten Anforderungen an die Anwendung umzusetzen. Die dynamischen Modelle beschreiben hingegen Abläufe bei der Nutzung der Anwendung.

7.1 Dynamisches Modell (OOA)

Die Interaktionen des Benutzers mit der Funktionstest-Anwendung werden in folgendem Use-Case-Diagramm (Abbildung 7.1) dargestellt. Links ist der allgemeine Benutzer dargestellt. Der Benutzer der Funktionstest-Anwendung muss eine Auflistung der Sensoren angezeigt bekommen, die er testen kann. Er muss die Kamera des Smartphones benutzen können, das heißt, dass er Fotos schießen und diese auch speichern können soll. Entscheidet sich der Benutzer dazu Sensoren zu testen, so kann er folgende Aktivitäten durchführen: Der Benutzer kann durch Bewegungen seines Smartphones den Beschleunigungssensor und mit diesem zusammen den Lagesen-

¹Balzert 1996.

7.1 Dynamisches Modell (OOA)

sor und das Gyroskop testen. Er kann sich auf einer Karte seinen aktuellen Standort anzeigen lassen. Mit Hilfe des im Smartphone integrierten Magnetometers soll sich der Benutzer den magnetischen Nordpol anzeigen lassen können. Als letzten Sensor soll der Benutzer auch den Näherungssensor testen können. Die Fotos, die der Benutzer mit der Kamerafunktion aufgenommen hat, soll er sich auch später ansehen und auch wieder löschen können. Als letztes gibt es noch eine Funktion, mit der sich der Benutzer das Logging der Anwendung in einem Textdokument anzeigen zu lassen kann und eine Funktion, die es dem Benutzer ermöglicht das Farbschema des Designs der Anwendung zu ändern.

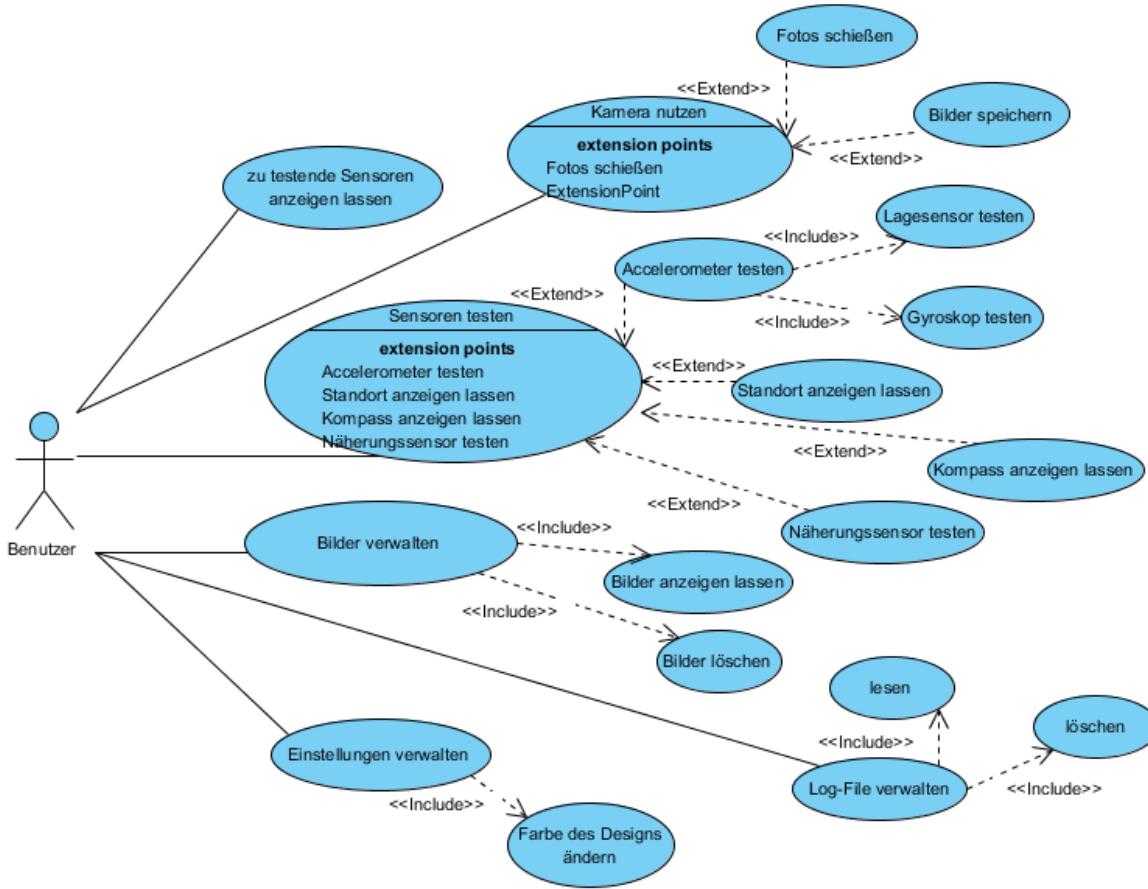


Abbildung 7.1: Funktionstest-Anwendung: Use-Case Diagramm

7.2 Statisches Modell (OOA)

Die Funktionstest-Anwendung wird einmal nativ und anschließend mit 5 unterschiedlichen Frameworks entwickelt, was bedeutet, dass sie mit unterschiedlichen Programmiersprachen implementiert werden muss (siehe Kapitel (X)). Dabei sind nicht alle dieser Programmiersprachen objektorientiert und die vorgegebene Architektur der einzelnen Frameworks ist zu diesem Zeitpunkt noch unbekannt. Aus diesem Grund empfiehlt sich an dieser Stelle kein UML Klassendiagramm. Anstelle dessen werden in einem Diagramm die einzelnen Funktionen und die Navigation zu diesen dargestellt (siehe Abbildung 7.2). So können Hierarchien und Tiefe der Anwendung auch ohne Klassendiagramm analysiert und gezeigt werden.



Abbildung 7.2: Funktionstest-Anwendung: Navigation Hauptmenü und Beschreibung der einzelnen Funktionen

7.2 Statisches Modell (OOA)

Wie in Abbildung 7.2 zu sehen ist, soll die Anwendung von der 'MainPage' aus starten. Die 'MainPage' ist ein Navigationsmenü, von dem aus der Benutzer zu folgenden Seiten navigieren kann: Zur Sensorliste, zur Kamerafunktion, zu der Seite, welche die vom Benutzer gespeicherten Daten verwaltet, zum Logging-Output und zu einer Einstellungsseite. Die Seite mit der Sensorliste stellt dabei selbst wieder ein Navigationsmenü dar. Navigiert der Benutzer vom Hauptmenü zur Kamerafunktion, so soll er da die Möglichkeit haben, zwischen der Front- und der Rückkamera des Gerätes auszuwählen und Fotos zu schießen. Geschossene Fotos können gespeichert werden. Ein weiterer Menüpunkt des Hauptmenüs führt den Benutzer zur Verwaltung seiner gespeicherten Fotos. Dort soll er durch klicken auf ein Foto dieses vergrößert angezeigt bekommen. Zudem hat er dort die Möglichkeit ausgewählte Bilder wieder zu löschen. Im Menüpunkt 'Logging' wird dem Benutzer das von der Anwendung generierte Logfile angezeigt und im Menüpunkt 'Einstellungen' kann der Benutzer zwischen vorkonfigurierten Farbdesigns für die Anwendung auswählen.

Im Navigationsmenü der Sensoren kann der Benutzer zu weiteren Funktionen der Anwendung navigieren, wie in Abbildung 7.3 zu sehen ist:

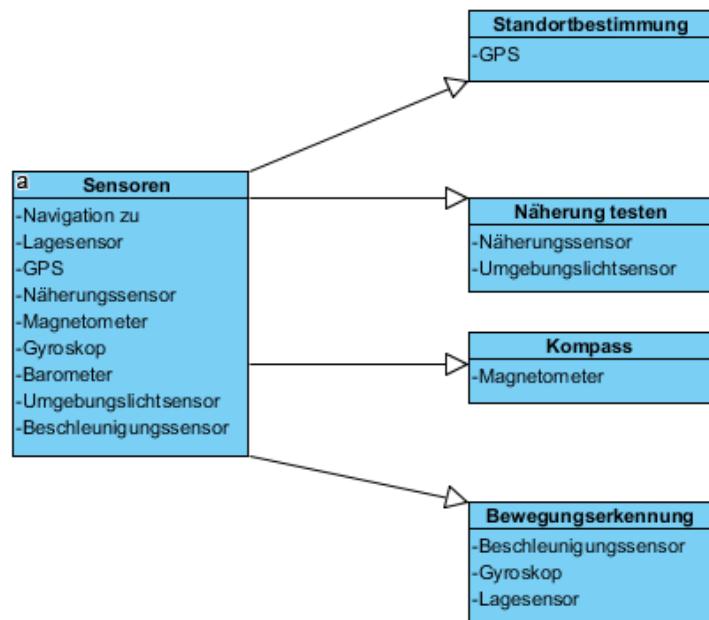


Abbildung 7.3: Funktionstest-Anwendung: Navigation Sensortests und Beschreibung der einzelnen Funktionen

7.3 Datenhaltung (OOD)

Die 4 Funktionstestseiten, zu denen der Benutzer navigieren kann sind einmal die Standortbestimmung, der Näherungstest, die Kompassanzeige und die Bewegungserkennung (siehe Abbildung 7.3). Bei der Standortbestimmung wird der GPS-Sensor des Smartphones angesprochen und getestet. Auf dem Display soll eine Weltkarte angezeigt werden mit einem Marker an der Stelle an der der Benutzer sich mit seinem Smartphone befindet. Es soll in die Karte hineingezoomt werden können, damit Ermittlung des Standorts durch das GPS besser überprüft werden kann. Für den Näherungstest werden der Näherungssensor und der Umgebungslichtsensor angesprochen. Hier soll auf dem Display des Smartphones Feedback gegeben werden, sobald sich ein Gegenstand nähert. Das Magnetometer wird bei der Kompassanzeige getestet. Hier bekommt der Benutzer eine Kompassnadel zu Gesicht, welche zum magnetischen Nordpol zeigt. Unter dem Menüpunkt 'Bewegungserkennung' werden der Beschleunigungssensor, das Gyroskop und gegebenenfalls der Lagesensor getestet, sollte es einen separaten geben.

7.3 Datenhaltung (OOD)

Entsprechend der in Kapitel (X) definierten Anforderungen und der in Kapitel (X) ausgearbeiteten Funktionen der Funktionstest-Anwendung sollen die Fotos, die der Benutzer mit der Kamerafunktion schießt auf dem Smartphone gespeichert werden. Sie sollen im 'External Storage' abgelegt werden. So soll es auch möglich sein aus anderen Anwendungen heraus auf diese Bilder zuzugreifen und sie auf andere Medien zu übertragen. Als Speicherformat für die Fotos wird 'JPG' verwendet. Für die Fotos soll die Anwendung vorher einen eigenen Ordner mit dem Namen 'Feature_Test_App' anlegen.

7.4 Die grafische Benutzeroberfläche

Bei der Erstellung des Prototyps für die grafische Benutzeroberfläche wurde darauf geachtet möglichst einige der Standardelemente des 'Material Designs' zu verwenden, welche auch für die native Entwicklung im Android Studio zur Verfügung gestellt werden.

Die Startseite der Anwendung ist das Hauptmenü (Siehe Kapitel(X)). Hier wird der in Kapitel (X) erwähnte 'Navigation Drawer' für die Navigation zu den weiteren Seiten

7.4 Die grafische Benutzeroberfläche

verwendet. Abbildung 7.4 zeigt das Hauptmenü mit geöffneten 'Navigation Drawer', in dem die Navigationsmöglichkeiten zur Kamerafunktion, zu den SensorTests, zur Galerie, dem Logging und den Einstellungen angezeigt werden. Der 'Navigation Drawer' lässt sich einerseits durch Wischen und andererseits über die 'App Bar' öffnen und schließen. Die Sensorauswahl wird ebenfalls mit einem 'Navigation Drawer' realisiert.



Abbildung 7.4: Funktionstest-Anwendung: Startseite mit Hauptmenü

Wählt der Benutzer im Hauptmenü die Kamerafunktion aus, so wechselt die Anwendung direkt in den Aufnahmemodus. In diesem Modus erscheint das Bild, welches die Kamera aufnimmt im Hintergrund. Vordergründig gibt es einen Auslöse-Button und einen Button für das Wechseln der Kameras im unteren Bildbereich. Ganz oben rechts sind die Blitzfunktionen, durch welche durch klicken navigiert werden kann.

BILD

In der Galerie (siehe Abbildung 7.5) werden dem Benutzer die aufgenommenen und

7.4 Die grafische Benutzeroberfläche

gespeicherten Bilder angezeigt. Durch diese Anzeige kann gescrollt werden. Einzelne Bilder können durch Klicken auf diese ausgewählt werden. Wird ein Bild angeklickt, so bekommt der Benutzer dieses in einem neuen Fenster groß angezeigt. Über den 'Floating Action Button' kann er nun dieses Bild löschen.

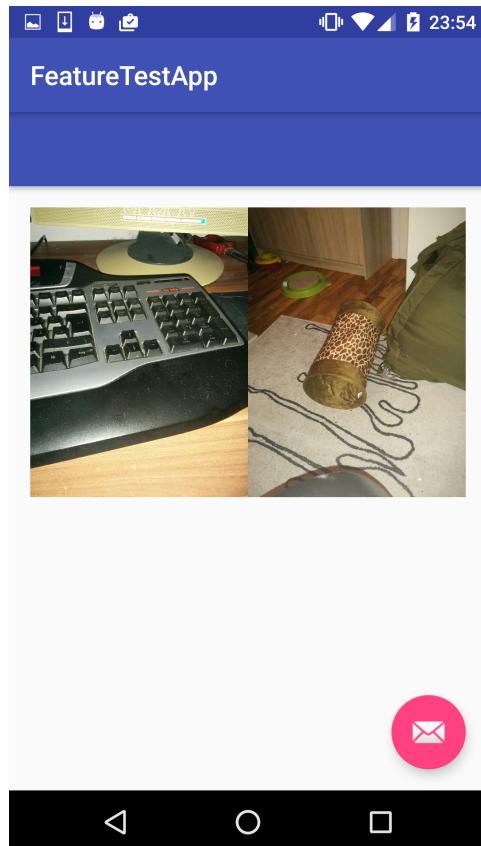


Abbildung 7.5: Funktionstest-Anwendung: Galerie

Wählt der Benutzer die Standortbestimmung aus, so erscheint im Hintergrund der Seite eine Weltkarte. Ist der Standort vom GPS-Sensor ermittelt worden, so erscheint ein MArker an diesem Ort auf der Weltkarte und die Kartenansicht schwenkt dorthin (siehe Abbildung 7.6). Der Benutzer hat nun die Möglichkeit weiter in die Karte hinein zu zoomen, um den genauen Standort angezeigt zu bekommen.

7.4 Die grafische Benutzeroberfläche

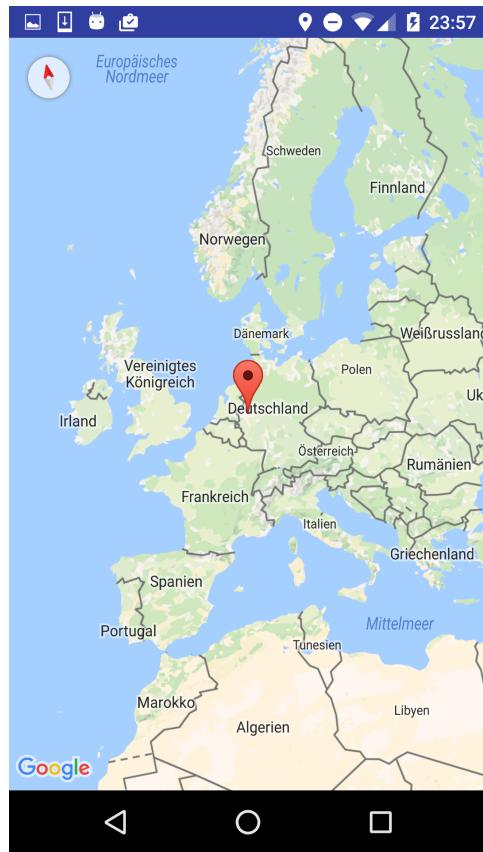


Abbildung 7.6: Funktionstest-Anwendung: Standortbestimmung

Wird die Kompassfunktion gestartet, wird automatisch vom im Smartphone verbauten Magnetometer der magnetische Nordpol ermittelt. Auf der Anzeige wird dem Benutzer eine Kompassrose angezeigt, welche mit der 'N'-Spitze in Richtung magnetischen Nordpol zeigt (siehe Abbildung 7.7).

7.4 Die grafische Benutzeroberfläche

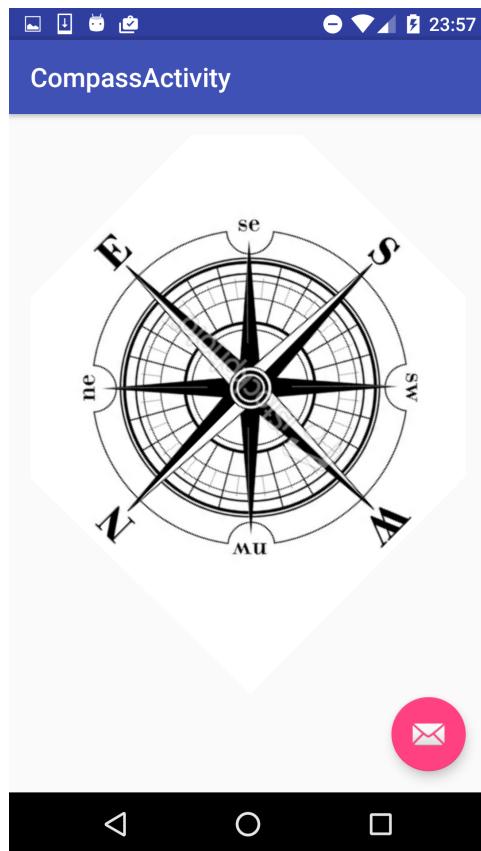


Abbildung 7.7: Funktionstest-Anwendung: Kompassfunktion

Auf der Seite des Bewegungstests werden die vom Beschleunigungssensor ausgelesenen Daten angezeigt. Die Beschleunigungen werden dabei, wie in Kapitel (X) beschrieben, je Achse angegeben. Zusätzlich zur angezeigten momentanen Beschleunigung werden noch die Werte der maximalen Beschleunigung angezeigt und aktuell gehalten (siehe Abbildung 7.8). Ein separater Lagesensor wird an dieser Stelle nicht getestet, der Sensortyp 'ORIENTATION' ist veraltet. Anhand des Achsenmodells des Beschleunigungssensors kann die Lage des Smartphones erkannt werden.

7.4 Die grafische Benutzeroberfläche

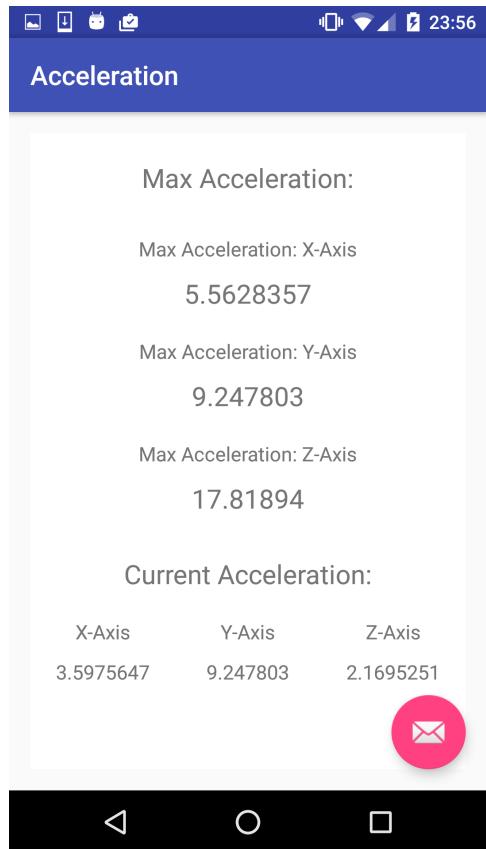


Abbildung 7.8: Funktionstest-Anwendung: Bewegungstest

Bei dem Test des Näherungssensors werden die ausgewerteten Sensordaten an ein 'Toast'-Objekt übergeben. Dieser 'Toast' zeigt 'far' an, wenn sich kein Objekt nahe vor dem Display des Smartphones befindet, und 'near' wenn sich eines genähert hat (siehe Abbildung 7.9).

7.4 Die grafische Benutzeroberfläche

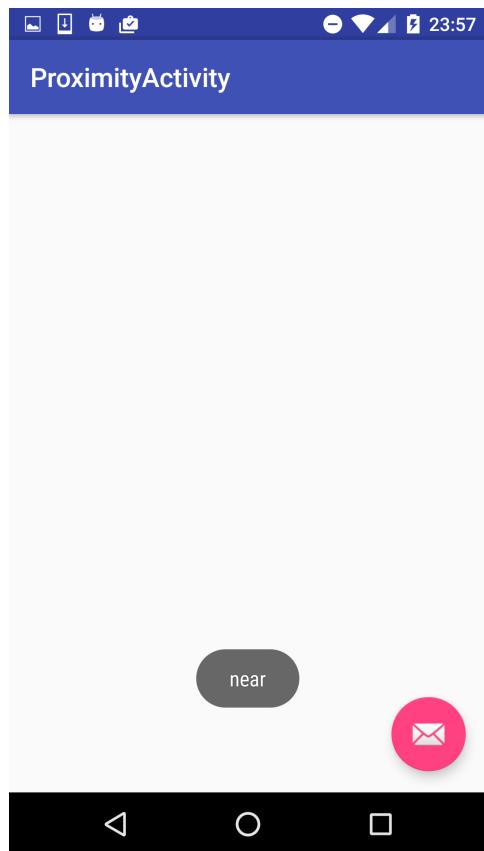


Abbildung 7.9: Funktionstest-Anwendung: Näherungstest

8 Implementierung

In diesem Kapitel werden die Implementierungen der Funktionstest-Anwendung gemäß der in Kapitel (X) erarbeiteten Analyse und Entwurf vorgestellt. Die Anwendung wird insgesamt 6 Mal implementiert: einmal nativ als referenz und anschließend mit den in Kapitel (X) vorgestellten 5 Frameworks. Das Hauptaugenmerk liegt hierbei auf den Bereichen, die der Anwendung ein natives Look-And-Feel verleihen sollen. Dies sind entsprechend die Umsetzung der in Kapitel (X) vorgestellten grafischen Benutzeroberfläche und die Umsetzung der Hardware-Zugriffe für die Nutzung der Sensoren, Kameras und Speicher. So ergeben sich für jede Implementierung 4 Bereiche.

8.1 Native Implementierung

Die native Implementierung der Funktionstest-Anwendung wird mit der IDE 'Android Studio' umgesetzt. Wie schon in Kapitel (X) beschrieben, ist die Programmiersprache nativer Android Anwendungen Java. Die Anwendung wird gegen die API 23, das bedeutet gegen die Major Version 6 (Marshmallow), kompiliert, da diese Version auch auf dem Testgerät installiert ist.

8.1.1 Grafische Benutzeroberfläche

In Android wird die Struktur der grafischen Benutzeroberfläche über Layouts definiert. Es gibt 2 Wege diese Layouts und ihre Elemente zu deklarieren: Einmal über eine XML-Datei und einmal programmatisch innerhalb der Anwendung über die Nutzung der API. Es wird empfohlen, das Layout über XML-Dateien zu gestalten¹, da dies auch eine klarere Trennung der Schicht der grafischen Benutzeroberfläche von der Fachlogikschicht der 3-Schichten-Architektur² begünstigt.

¹Cinar 2015.

²Balzert 1996.

8.1 Native Implementierung

Für das Hauptmenü und die Auswahl der Sensortests wird, wie in Kapitel (X) festgelegt wurde, ein 'Navigation Drawer' genutzt. Das Layout dieser Menüs wird in jeweils 3 XML-Dateien beschrieben. Beispielhaft wird an dieser Stelle nur der Code des Hauptmenüs besprochen. Die einzelnen Menüpunkte, die im 'Navigation Drawer' zur Verfügung gestellt werden sollen, werden in der Datei *activity_main_drawer.xml* definiert. Wie in unten stehendem Listing (X) dargestellt, wird pro Eintrag im Drawer-Menü ein Item-Objekt definiert. Diesen Item-Objekten werden jeweils eine eindeutige ID, ein Icon und eine Bezeichnung, welche dann im Menü angezeigt wird, übergeben.

Listing 8.1: Definition der Menüpunkte im 'Navigation Drawer' der Hauptseite in der Datei *activity_main_drawer.xml*

```
1 <group android:checkableBehavior="single">
2     ...
3     <item
4         android:id="@+id/nav_camera"
5         android:icon="@drawable/ic_menu_camera"
6         android:title="Kamera" />
7     <item
8         android:id="@+id/nav_gallery"
9         android:icon="@drawable/ic_menu_gallery"
10        android:title="Gallery" />
11        ...
12    </group>
```

Jeder 'Navigation Drawer' hat einen Header. Das Design und der Aufbau dessen wird in der Datei *nav_header_main.xml* definiert (siehe Listing (X)). Hier wird bei der Funktionstest-Anwendung der Default-Header von Android Studio genutzt. Für den farbigen Hintergrund, sowie das Android-Icon werden Vorlagen im *drawable*-Ordner mitgeliefert. Diese werden in der Layout-Datei referenziert, wie in Zeile 4 und 11 im unten stehenden Listing (X) zu sehen ist.

8.1 Native Implementierung

Listing 8.2: Definition des Headers des 'Navigation Drawers' in der Datei *nav_header_main.xml*

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="@dimen/nav_header_height"
4     android:background="@drawable/side_nav_bar"
5     ...
6     <ImageView
7         android:id="@+id/imageView"
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:paddingTop="@dimen/nav_header_vertical_spacing"
11        android:src="@android:drawable/sym_def_app_icon" />
12
13    <TextView
14        android:layout_width="match_parent"
15        android:layout_height="wrap_content"
16        android:paddingTop="@dimen/nav_header_vertical_spacing"
17        android:text="Android Studio"
18        android:textAppearance="@style/TextAppearance.AppCompat.Body1" />
19    ...
20 </LinearLayout>
```

Das Layout und der Inhalt des 'Navigation Drawers' werden nun in der Layout-Datei des Hauptmenüs, der *MainActivity* referenziert (siehe Listing (X) Zeile 13 und 14). Hier wird noch über *include* die *App Bar*, die am oberen Bildrand positioniert ist, importiert (siehe Listing (X) Zeile 2 bis 5). Das *App Bar* Design, welches in der Datei *app_bar_main.xml* definiert ist, bringt noch einen *FloatingActionButton* mit sich, der unten rechts im Bildschirm positioniert wird.

Listing 8.3: Layout des Hauptmenüs in der Datei *activity_main.xml*

```
1 ...
2 <include
3     layout="@layout/app_bar_main"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent" />
6
7     <android.support.design.widget.NavigationView
8         android:id="@+id/nav_view"
9         android:layout_width="wrap_content"
10        android:layout_height="match_parent"
11        android:layout_gravity="start"
12        android:fitsSystemWindows="true"
13        app:headerLayout="@layout/nav_header_main"
14        app:menu="@menu/activity_main_drawer" />
15 ...
```

8.1 Native Implementierung

Für die Kamerafunktion musste keine eigene Benutzeroberfläche gestaltet werden, da über die API die in Android integrierte Kameraanwendung direkt angesprochen und gestartet wird. Diese Anwendung stellt entsprechend der Anforderungen aus Kapitel (X) sämtliche Benutzereingabemöglichkeiten bezüglich Kamerawechsel oder Blitzeinstellungen zur Verfügung.

Für die Darstellung der Werte des Beschleunigungssensors wurden *LinearLayout*-Objekte in 3 Ebenen verwendet, welche mit *TextView*-Objekten gefüllt sind, über die die Werte dem Benutzer präsentiert werden. Zur allgemeinen Aufteilung der Layout-Elementen lässt sich sagen, dass die globalen Elemente einer Seite, wie die *AppBar* oder der *FloatingActionButton* in der hierarchisch obersten Layout-Datei *activity-[name].xml* definiert werden.

8.1.2 Sensoren

Da die Zugriffe auf die Sensoren und das Auslesen ihrer Daten äquivalent sind werden diese Vorgänge hier am Beispiel des Näherungssensors erläutert. Damit Die *Activity* auf Sensordaten zugreifen kann, braucht sie ein *SensorManager*-Objekt und ein *Sensor*-Objekt (siehe Listing (X)).

Listing 8.4: Definition und Initialisierung von Sensor und SensorManager

```
1 private SensorManager mSensorManager;
2 private Sensor mSensor;
3 ...
4 mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
5 mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

Beide Objekte werden direkt beim initialen Start der *Activity* erzeugt. Der *SensorManager* wird benötigt um die *Listener*, welche auf den Sensor horchen, zu registrieren und abzumelden. Dies geschieht in den Methoden *onResume()* und *onPause()* (siehe Listing (X)).

8.1 Native Implementierung

Listing 8.5: Die Methoden *onResume()* und *onPause()*

```
1 protected void onResume() {
2     super.onResume();
3     mSensorManager.registerListener(this, mSensor,
4                                     SensorManager.SENSOR_DELAY_NORMAL);
5 }
6 protected void onPause() {
7     super.onPause();
8     mSensorManager.unregisterListener(this);
9 }
```

Die vom Sensor übermittelten Daten werden in einem *SensorEvent*-Objekt der Methode *onSensorChanged()* übergeben. Dort können diese Daten weiterverarbeitet werden. Für die Verarbeitung der Daten des Näherungssensors bedeutet dies, dass dort die Textausgaben 'near' und 'far' generiert werden (siehe Listing (X)).

Listing 8.6: Die Methode *onSensorChanged()*

```
1 public void onSensorChanged(SensorEvent event) {
2     if (event.values[0] == 0) {
3         Toast.makeText(getApplicationContext(), "near", Toast.LENGTH_SHORT).show();
4     } else {
5         Toast.makeText(getApplicationContext(), "far", Toast.LENGTH_SHORT).show();
6     }
7 }
```

Für den Zugriff auf manche Sensoren und Dienste benötigt die Anwendung eine besondere Erlaubnis. Diese Erlaubnis wird der Anwendung im *AndroidManifest* gegeben. Reicht dies nicht aus, muss eine Anfrage an den Benutzer gestellt werden, welcher dann mit seiner Zustimmung die Erlaubnis erteilt. Im *AndroidManifest* wird diese Erlaubnis über den *uses-permission*-Parameter eingetragen (siehe Listing (X)).

Listing 8.7: Erlaubnis für die Nutzung eines Dienstes im *AndroidManifest*

```
1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Die Anfrage einer Erlaubnis direkt an den Benutzer wird im Code der jeweiligen *Activity* realisiert, wie in Listing (X) zu sehen ist:

8.1 Native Implementierung

Listing 8.8: Generierung der Anfrage für die Erlaubnis einen Dienst zu nutzen an den Benutzer

```
1 ...
2 if (ContextCompat.checkSelfPermission(this, android.Manifest.permission.←
3     ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
4     ActivityCompat.requestPermissions(this,
5         new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
6         MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
7 }
```

8.1.3 Kameras

Wie bereits in Kapitel (X) erwähnt, wird für die Kameranutzung die interne Kameraanwendung von Android aufgerufen. Dies geschieht direkt, nachdem vom Hauptmenü aus die Kamerafunktion (*CameraActivity*) aufgerufen wird, also in der *onCreate()*-Methode der *CameraActivity*. Hier ist dieser Aufruf, wie in Listing (X) zu sehen ist in der Methode *takeAPicture()* gekapselt:

Listing 8.9: Aufruf der Android-Kamerafunktion in der *onCreate()*-Methode der Klasse *CameraActivity*

```
1 @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity_camera);
5
6         //dispatchTakePictureIntent();
7         _imageView = (ImageView) findViewById(R.id.imageView1);
8         takeAPicture();
9     }
```

Eine *Activity* wird in Android über einen *Intent* aufgerufen. Ein *Intent* kann auf verschiedene Weisen erzeugt werden. In diesem Fall übergeben wir dem Konstruktor eine *Action* in Form einer **String**-Variable, nämlich die *ACTION_IMAGE_CAPTURE* (siehe Listing (X)).

8.1 Native Implementierung

Listing 8.10: Methode `takeAPicture()`: Aufruf der Android-Kamerafunktion über einen Intent

```
1 private void takeAPicture() {
2     Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
3     try {
4         _file = createImageFile();
5     } catch (IOException e) {
6         // ToDo
7     }
8     intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(_file));
9     startActivityForResult(intent, REQUEST_IMAGE_CAPTURE);
10 }
```

Die Methode `createImageFile()`, welche oben im Listing (X) in Zeile 4 aufgerufen wird, generiert Namen für die Bilder, die mit der Kamerafunktion aufgenommen und gespeichert werden. Sollen noch weitere Funktionen ausgeführt werden, sobald ein Bild gespeichert wird, so müssen diese in der Methode `onActivityResult()` definiert werden. Diese Methode wird automatisch immer dann ausgeführt, wenn der Benutzer ein Bild abspeichert. Als Beispiel können hier noch Bildbearbeitungen, wie Formatänderungen, vorgenommen werden, oder das Bild kann, wie Listing (X) zeigt, in der Android *Gallery* zur Verfügung gestellt werden.

Listing 8.11: Bereitstellen des gespeicherten Bildes für die Android *Gallery*

```
1 ...
2 Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
3 Uri contentUri = Uri.fromFile(this._file);
4 mediaScanIntent.setData(contentUri);
5 this.sendBroadcast(mediaScanIntent);
6 ...
```

8.1.4 Speicherzugriffe

Der in Kapitel (X) beschriebene 'External Storage' wird über die Methode `getExternalFilesDir()` adressiert. Da wir in diesem Fall dort Bilder speichern wollen, übergeben wir der Methode dies als Information in Form von `Environment.DIRECTORY_PICTURES`. In diesem Ordner wird nun der Ordner 'Feature_Test_App' erzeugt, wie in Listing (X) zu sehen ist.

8.2 Implementierung mit Xamarin

Listing 8.12: Adressieren des 'External Storage' und Anlegen eines Ordners für die Fotos der Funktionstest-Anwendung

```
1 public void CreateDirectoryForPictures() {
2     File pubPictures = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
3     //this._dir = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), "FeatureTestApp");
4     CameraActivity._dir = new File(pubPictures, "Feature_Test_App");
5     if (!CameraActivity._dir.exists())
6     {
7         CameraActivity._dir.mkdirs();
8     }
9 }
```

8.2 Implementierung mit Xamarin

Für die Entwicklung von Xamarin Anwendungen auf einem Windows PC wird die IDE 'Visual Studio' verwendet. Die verwendete Programmiersprache ist, wie schon in Kapitel (X) erwähnt, C#. Für den optimalen Vergleich zur nativen Anwendung wird auch die Xamarin-Version gegen die API 23, das bedeutet gegen die Major Version 6 (Marshmallow), kompiliert.

8.2.1 Grafische Benutzeroberfläche

Wenn man sich dafür entscheidet in Xamarin eine Anwendung für Android mit den bekannten Android-Bedienelementen zu schreiben, anstelle einer Anwendung mit Xamarin.Forms, so ergibt sich die gleiche Dateistruktur im Projekt wie auch bei der nativen Anwendung. Die Layout-Dateien enden allerdings hier nicht auf .xml, sondern auf .axml. Der interne Aufbau der Layout-Dateien ist identisch mit dem der nativen Layout-Dateien. Möchte man also eine native Anwendung in Xamarin migrieren, ließe sich das Layout ganz einfach übernehmen.

Um unnötigen doppelten Code auszusparen, wurde ab dieser Implementierung nur noch mit einem *Navigation Drawer* gearbeitet, anstelle von zwei. Da die Inhalte der Layout-Dateien des *Drawers*, wie oben schon erwähnt, identisch mit der nativen Version sind, wird an dieser Stelle auf Ausschnitte dieser verzichtet. Der Zuordnung der einzelnen Seiten der Anwendung zu den Navigationselementen erfolgt ebenfalls ähnlich der nativen Version über *Intents*. Der Code ist bei der Xamarin-Anwendung zwar in C# geschrieben, die aufgerufene Klasse *Intent* stammt dabei aus dem Android

8.2 Implementierung mit Xamarin

SDK (siehe Listing (X)).

Listing 8.13: Zuweisung der einzelnen Seiten der Anwendung zu den Navigations-elementen im Navigation Drawer

```
1 void NavigationView_NavigationItemSelected(object sender, NavigationView.NavigationItemSelectedEventArgs e)
2 {
3     switch (e.MenuItem.ItemId)
4     {
5         case (Resource.Id.nav_camera):
6             Intent intentCamera = new Intent(this, typeof(CameraActivity));
7             StartActivity(intentCamera);
8             break;
9         case (Resource.Id.nav_gallery):
10            Intent intentGallery = new Intent(this, typeof(PictureGalleryActivity));
11            ;
12            StartActivity(intentGallery);
13            break;
14            ...
15    }
16 }
```

In nachfolgender Abbildung (X) ist der *NavigationDrawer* mit den einzelnen Menüpunkten abgebildet. Wie bei der nativen Anwendung ist der *NavigationDrawer* auch in der Xamarin-Anwendung über Wischen und über den Hamburger-Button oben links in der Appbar aus- und wieder einklappbar.

8.2 Implementierung mit Xamarin



Abbildung 8.1: Menü in der Xamarin Anwendung

In der Xamarin-Anwendung wird wie in der nativen Anwendung auf der Seite mit der Kamerafunktionalität über die API die in Android integrierte Kameraanwendung direkt angesprochen und gestartet. So verfügt die Anwendung auch über die Möglichkeit des Kamerawechsels und des Einstellen des Blitzes.

Der *FloatingActionButton* wird wie auch die anderen GUI-Elemente wie der *NavigationDrawer* in einer Layout-Datei deklariert. Da der *FloatingActionButton* auf allen Seiten zur Verfügung stehen soll, wird er wie in der nativen Anwendung in der Layout-Datei *app_bar.axml* deklariert, welche dann in der Haupt-Layout-Datei importiert wird. Die Deklaration des *FloatingActionButton* ist unten im Listing (X) dargestellt. Bei der Deklaration können Farbe, Größe und Positionierung mit angegeben werden. Wie in Kapitel (X) dargestellt, wird in dieser Anwendung der *FloatingActionButton* rechts unten positioniert, was über die Angabe 'bottom—right' geschieht (siehe Listing (X)).

8.2 Implementierung mit Xamarin

Listing 8.14: Deklaration des *FloatingActionButton* in der Datei *app_bar.axml*

```
1 ...
2 <com.refractored.fab.FloatingActionButton
3     android:id="@+id/fab"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:layout_gravity="bottom|right"
7     android:layout_margin="16dp"
8     android:src="@drawable/ic_action_content_new"
9     fab:fab_colorNormal="@color/primary"
10    fab:fab_colorPressed="@color/primary_pressed"
11    fab:fab_colorRipple="@color/ripple" />
12 ...
```

8.2.2 Sensoren

Wie auch bei der nativen Anwendung müssen Berechtigungen im *AndroidManifest* erteilt werden. Hierfür bietet Xamarin in Visual Studio eine grafische Benutzeroberfläche, in der die benötigten Berechtigungen einfach ausgewählt werden können und anschließend automatisch in die Datei *AndroidManifest.xml* eingetragen werden (siehe Abbildung (X)).

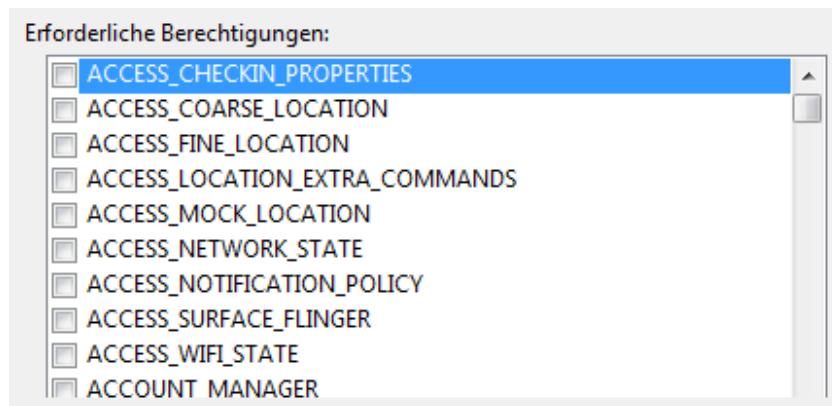


Abbildung 8.2: Auswahlfenster für benötigte Berechtigungen in Xamarin (Visual Studio)

Der generierte Eintrag in das *AndroidManifest* ist nachfolgend in Listing (X) dargestellt:

Listing 8.15: Erlaubnis für die Nutzung eines Dienstes im *AndroidManifest* (Xamarin)

```
1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

8.2 Implementierung mit Xamarin

Auch das Auslesen von Sensordaten ist nahezu identisch mit der nativen Variante. Für eine gute Vergleichbarkeit bleiben wir auch in diesem Kapitel bei der Vorstellung der Nutzung des Näherungssensors. Wie man in unten stehendem Listing (X) erkennen kann, ist der Code zum Auslesen der Daten des Näherungssensors praktisch identisch mit dem aus Listing (X) (nativ). Auch hier wird die Bearbeitung der Daten in der Methode *onSensorChanged()* der Klasse *SensorEventListener* vorgenommen.

Listing 8.16: Auslesen der Daten des Näherungssensors (Xamarin)

```
1 public void onSensorChanged(SensorEvent event) {  
2     if (event.values[0] == 0) {  
3         Toast.MakeText(this, "near", ToastLength.Short).Show();  
4     } else {  
5         Toast.MakeText(this, "far", ToastLength.Short).Show();  
6     }  
7 }
```

8.2.3 Kameras

Wie bereits oben erwähnt, wird auch bei der Xamarin-Anwendung die im Android SDK integrierte Kameraanwendung aufgerufen und für das Schießen von Fotos verwendet. Ähnlich wie in der nativen Anwendung wurde auch hier eine Methode *TakeAPicture()* geschrieben, in der die Kameraanwendung als *Intent* aufgerufen wird. Dies ist nachfolgend in Listing (X) dargestellt. Wie auch in der nativen Version wird hier über die Klasse *MediaStore* der Standard *Intent ActionImageCapture* aufgerufen, um die Kameraanwendung von Android zu starten. Der Code der Xamarin-Anwendung ähnelt auch an dieser Stelle sehr dem nativen Code, da direkt die Android API für verschiedene Funktionalitäten (in diesem Fall die Kameraanwendung) angesprochen wird.

8.3 Implementierung mit Ionic/Cordova

Listing 8.17: Methode *TakeAPicture()*: Aufruf der Android-Kamerafunktion über einen *Intent* in Xamarin

```
1 private void TakeAPicture()
2 {
3     Intent intent = new Intent(MediaStore.ActionImageCapture);
4     try
5     {
6         _file = createImageFile();
7     } catch (IOException e)
8     {
9     }
}
```

8.2.4 Speicherzugriffe

Auch bei der Nutzung des *External Storage* ist die Umsetzung mit Xamarin sehr ähnlich der nativen Weise. Auch hier wird die Methode *GetExternalFilesDir* aus der Android API für den Zugriff auf den externen Speicher verwendet. Die Berechtigung für die Nutzung des externen Speichers muss dafür zuvor in das *Android Manifest* eingetragen worden sein (siehe Kapitel (X)). Nachfolgendes Listing (X) zeigt die Implementierung der Methode *GetExternalFilesDir* mit dem Xamarin Framework:

Listing 8.18: Methode *GetExternalFilesDir()*: Adressieren des 'External Storage' und Anlegen eines Ordners für die Fotos der Funktionstest-Anwendung

```
1 private void CreateDirectoryForPictures()
2 {
3     File pubPictures = GetExternalFilesDir(Environment.DirectoryPictures);
4     CameraActivity._dir = new File(pubPictures, "FeatureTestAppXamarin");
5     if (!CameraActivity._dir.Exists())
6     {
7         CameraActivity._dir.Mkdirs();
8     }
9 }
```

8.3 Implementierung mit Ionic/Cordova

Für die Entwicklung mit dem Ionic und Cordova Framework auf einem Windows PC musste zunächst Node.js installiert werden, anschließend wurden sowohl die Ionic und Cordova Kommandozeilen-Tools installiert. Entwickelt wird die Ionic-Anwendung mit JavaScript, hierfür kann ein beliebiger Editor genutzt werden. Auch die Ionic-Anwendung wird für den optimalen Vergleich gegen die API 23, die Major Version 6

8.3 Implementierung mit Ionic/Cordova

(Marshmallow), kompiliert. Hier ist allerdings noch zu erwähnen, dass in der Version Ionic 1 direkt mit JavaScript entwickelt wurde und bei Ionic 2 die von Microsoft entwickelte Programmiersprache TypeScript verwendet wird³, welche auch objektorientierte Prinzipien wie Klassen mit sich bringt. Da jeder JavaScript-Code allerdings auch gültiger TypeScript-Code ist, können weiterhin gängige Javascript-Bibliotheken in TypeScript und somit in Ionic 2 verwendet werden⁴. Für diese Arbeit wird Ionic 2 genutzt, da dies die aktuelle Version vom Ionic Framework ist, welche auch weiterentwickelt wird.

8.3.1 Grafische Benutzeroberfläche

Die Dateistruktur einer Ionic-Anwendung unterscheidet sich sehr stark von einer nativen Android-Anwendung. Dies ist allein schon dadurch bedingt, dass eine Ionic-Anwendung nicht mit einer objektorientierten Programmiersprache, sondern mit der Skriptsprache JavaScript, beziehungsweise TypeScript entwickelt wird.

Ionic unterstützt verschiedenste UI-Elemente von iOS, Android und Windows, die je nachdem für welche Plattform die Anwendung gebaut wird, automatisch angepasst werden. Das Layout mit seinen einzelnen Elementen wird pro Seite der Anwendung in einer zugehörigen HTML-Datei deklariert. Das Menü mit seinen einzelnen Elementen, in diesem Fall Seiten, wird in der Datei *app.component.ts* deklariert. Diese Datei liegt unter dem Root-Verzeichnis der Anwendung unter *src/app/app.component.ts*. Hier wird nicht das Design des Menüs festgelegt, sondern nur sein Inhalt mit seinen Bezeichnern. Hierzu müssen die einzelnen Seiten, zu denen man navigieren können soll, importiert werden (siehe Listing (X)).

Listing 8.19: Import der einzelnen Seiten für die Navigation in der Datei *app.component.ts*

```
1 import { Page1 } from './pages/page1/page1';
2 import { Page2 } from './pages/page2/page2';
3 import { Page3 } from './pages/page3/page3';
4 import { Page4 } from './pages/page4/page4';
5 import { Page5 } from './pages/page5/page5';
6 import { Page_main } from './pages/page_main/page_main';
```

³Ünlü 2016.

⁴TypeScript 2017.

8.3 Implementierung mit Ionic/Cordova

Anschließend werden in dieser Datei noch die Startseite der Anwendung und die Bezeichnungen der Seiten, zu denen man navigieren können soll, gesetzt. Das Benennen der Bezeichnungen und die Zuordnung derer zu den einzelnen Seiten geschieht im Konstruktor (siehe Listing (X)).

Listing 8.20: Setzen der Bezeichnungen der einzelnen Seiten für die Navigation

```
1 constructor(public platform: Platform) {
2     this.initializeApp();
3
4     this.pages = [
5         { title: 'Kamera', component: Page1 },
6         { title: 'Galerie', component: Page2 },
7         { title: 'Sensoren', component: Page3 },
8         { title: 'Logging', component: Page4 },
9         { title: 'Manage', component: Page5 }
10    ];
11
12 }
```

Das Äquivalent zum *NavigationDrawer* aus Android ist das *Side Menu*. In der Datei *app.html*, welche neben der Datei *app.component.ts* ebenfalls im Ordner *src* liegt, wird dieses deklariert. Eine Toolbar ist hierbei auch mit integriert. Diese bietet unter anderem den Hamburger-Button, über den sich das *Side Menu* nebst Swipe öffnen lässt. Wird eine Seite ausgewählt, schließt sich das Menü und die neue Seite wird über die in der Datei *app.component.ts* implementierte Methode *openPage()* geöffnet (siehe Listing (X)).

Listing 8.21: Deklaration des *Side Menus* in der Datei *app.html*

```
1 <ion-menu [content]="content">
2     <ion-header>
3         <ion-toolbar>
4             <ion-title>Menu</ion-title>
5         </ion-toolbar>
6     </ion-header>
7
8     <ion-content>
9         <ion-list>
10            <button menuClose ion-item *ngFor="let p of pages" (click)="openPage(p)">
11                {{p.title}}
12            </button>
13        </ion-list>
14    </ion-content>
15
16 </ion-menu>
```

8.3 Implementierung mit Ionic/Cordova

Nachfolgende Abbildung (X) zeigt das Menü in der Ionic 2 Anwendung. Man kann es wie in der nativen Android Anwendung über Wischen öffnen und schließen. Im Gegensatz zum nativen *NavigationDrawer* und dem von Xamarin, hat das Menü in Ionic nur einen kleinen integrierten Header, der in diesem Fall mit 'Menü' beschriftet ist.



Abbildung 8.3: Menü in der Ionic 2 Anwendung

In den HTML-Dateien der einzelnen Seiten, in denen deren Layout definiert wird, muss nur noch seitenspezifisches Design implementiert werden, das Menü wird hier nicht weiter bearbeitet. Das Ionic Framework bietet hier auch direkt die Möglichkeit einen *FloatingActionButton* zu benutzen, was dem nativen GUI-Design entgegenkommt (siehe Listing (X)).

Listing 8.22: Deklaration eines *FloatingActionButton*

```
1 <ion-fab right bottom>
2     <button ion-fab color="red" (click)="presentActionSheet()"></button>
3 </ion-fab>
```

8.3 Implementierung mit Ionic/Cordova

Im Gegensatz zur nativen Android Entwicklung müssen UI-Elemente, die in der Layout-Datei, hier in einer HTML-Datei, deklariert wurden, nicht mehr im Konstruktor der zugehörigen Source-Datei initialisiert werden. Bei Buttons, die durch Klicken eine Aktion ausführen sollen, wird die auszuführende Methode in der HTML-Datei dem Button zugewiesen, wie oben in Listing (X) zu erkennen ist.

8.3.2 Sensoren

Ionic 2 bringt ein *native* Paket mit sich, in dem Klassen definiert sind, welche Cordova-Plugins kapseln und so in TypeScript genutzt werden können. Es existieren bisher nicht für alle Sensoren kapselnde Klassen, sondern nur für den Beschleunigungssensor, das Magnetometer und GPS. Möchte man andere Sensoren nutzen, so ist man darauf angewiesen, ein den Anforderungen entsprechenden Cordova-Plugin zu finden oder ein eben solches selber zu schreiben. Bei der Einbindung und Nutzung eines solchen Cordova-Plugins ist man allerdings wieder dazu gezwungen JavaScript zu nutzen. Dies wird in dieser Arbeit nicht umgesetzt, sondern es wird sich auf das beschränkt, was das Ionic 2 Framework direkt mitliefert.

Um beispielsweise mit Ionic 2 auf das Magnetometer zugreifen zu können, muss zunächst ein bestimmtes Cordova-Plugin installiert werden. Auf der Homepage des Ionic Frameworks sind diese Informationen in der Dokumentation des *native* Pakets zu finden. Im Fall des Magnetometers muss das Plugin *Device Orientation* heruntergeladen und installiert werden. Dies erfolgt über 2 Kommandos (Siehe Listing (X)), welche in einer Kommando-Shell ausgeführt werden müssen, in welcher zunächst in den Root-Ordner der Anwendung navigiert wurde.

Listing 8.23: Installation des Cordova-Plugins für das Magnetometer

```
1 ionic plugin add cordova-plugin-device-orientation  
2 npm install --save @ionic-native/device-orientation
```

Anders als bei der nativen Android Entwicklung oder auch bei der Entwicklung mit dem Framework Xamarin, müssen bei der Verwendung von Hardware-Komponenten mit dem Ionic Framework keine Berechtigungen manuell der Anwendung hinzugefügt werden. Die nötigen Berechtigungen bringt das Cordova-Plugin mit sich. Um das installierte Cordova-Plugin in der Anwendung zu nutzen, müssen die dieses Plugin kapselnden Klassen aus dem *native* Paket importiert werden (siehe Listing (X)).

8.3 Implementierung mit Ionic/Cordova

Listing 8.24: Import der Klassen für die Nutzung des Magnetometers aus dem *native Paket von Ionic 2*

```
1 import {DeviceOrientation, DeviceOrientationCompassHeading} from 'ionic-native';
```

Anschließend können die importierten Klassen und ihre Methoden dazu genutzt werden, Daten des Sensors auszulesen und anzuzeigen. Um einmalig den aktuellen Wert des Magnetometers auszulesen, kann die Methode *getCurrentHeading()* der Klasse *DeviceOrientation* verwendet werden. Der Rückgabewert dieser Methode ist vom Typ *DeviceOrientationCompassHeading*. Mithilfe der Methode *magneticHeading()* kann nun der aktuelle Sensorwert aus dem *DeviceOrientationCompassHeading*-Objekt gelesen und weiter genutzt werden. Möchte man den Sensor in einem regelmäßigen Intervall auslesen, so wird die Methode *watchHeading()* benötigt. Da es bei einer Kompass-Anwendung sinnvoll ist, immer den aktuellen Sensorwert anzeigen zu lassen ohne manuell aktualisieren zu müssen, werden die Sensordaten hier mittels der Methode *watchHeading()* ausgelesen und angezeigt (siehe Listing(X)).

Listing 8.25: Auslesen der Daten des Magnetometers mithilfe der Methode *watchHeading()*

```
1 platform.ready().then(() => {
2     DeviceOrientation.watchHeading().subscribe(
3         (data: DeviceOrientationCompassHeading) => this.orientation2 = data.↔
4             magneticHeading.toString()
5     );
5});
```

8.3.3 Kameras

Für die Nutzung der Kamera gibt es ähnlich wie oben bei der Sensornutzung beschrieben eine kapselnde Klasse im *native* Paket des Ionic 2 Frameworks. Der Name dieser Klasse ist simpel *Camera*. Um diese Klasse zu nutzen muss ebenfalls wie bei der Sensornutzung vorab ein Cordova-Plugin installiert werden. In diesem Fall ist es das Plugin *Camera*. Die Installation erfolgt über die Kommando-Shell.

Ruft man die Methode *getPicture()* der Klasse *Camera* auf, so wird über das installierte Cordova-Plugin die native Android Kamera-Anwendung gestartet. Der Methode *getPicture()* können hierbei noch weitere Optionen, zum Beispiel bezüglich der Bildqualität, mitgegeben werden (siehe Listing(X)).

8.3 Implementierung mit Ionic/Cordova

Listing 8.26: Aufruf der Android Kamera-Anwendung über die Methode `getPicture()`

```
1 public takePicture(sourceType) {
2
3     var options = {
4         quality: 100,
5         sourceType: sourceType,
6         saveToPhotoAlbum: true,
7         correctOrientation: true
8     };
9     Camera.getPicture(options).then((imagePath) => {
10
11         var currentName = imagePath.substr(imagePath.lastIndexOf('/') + 1);
12         var correctPath = imagePath.substr(0, imagePath.lastIndexOf('/') + 1);
13         this.copyFileToLocalDir(correctPath, currentName, this.createFileName());
14
15     }
16 }
```

Wie man oben in Listing (X) sehen kann, wird, nachdem mit der Android Kamera-Anwendung ein Foto geschossen wurde, ein Dateiname mit der Methode `createFileName()` für dieses generiert und die Methode `copyFileToLocalDir` aufgerufen, in der das Speichern dieses Bildes implementiert ist. Die Methode `getPicture()` liefert hier unter anderem den Pfad, unter dem die Android Kamera-Anwendung das Bild zwischenspeichert, zurück.

8.3.4 Speicherzugriffe

Die Speicherzugriffe zum Speichern und laden geschossener Fotos werden ebenfalls über ein Cordova-Plugin (`File` und `File Transfer`) abgehandelt. Die benötigten Klassen aus dem *native* Paket, welche importiert werden müssen sind `File` und `FilePath`. Wie oben in Listing (X) schon zu sehen war, wird das Speichern in der Methode `copyFileToLocalDir()` vorgenommen. Der Methode werden dafür der aktuelle Pfad, der Name des Bildes und ein neuer Name übergeben. Mit der Methode `copyFile()` wird das Bild nun in dem angegebenen Ordner abgelegt. In diesem Fall ist das `cordova.file.externalDataDirectory`. Das `externalDataDirectory` des Cordova-Plugins entspricht dem *External Storage* von Android. Der eben beschriebene Speichervorgang ist unten in Listing (X) zu sehen.

8.4 Implementierung mit React Native

Listing 8.27: Methode `copyFileToLocalDir()` zum Speichern der aufgenommenen Bilder

```
1 private copyFileToLocalDir(namePath, currentName, newFileName) {
2     File.copyFile(namePath, currentName, cordova.file.externalDataDirectory, newFileName). $\leftarrow$ 
3         then(success => {
4             this.lastImage = newFileName;
5         }, error => {
6             this.presentToast('Error while storing file.');
7         });
}
```

8.4 Implementierung mit React Native

Wie schon zuvor bei der Entwicklung mit dem Ionic 2 Framework und Cordova muss auch für die Entwicklung mit dem React Native Framework Node.js installiert sein. Anschließend wird das React Native Kommandozeilen Interface installiert. Eine React Native Anwendung basiert auf React. React, oder auch ReactJS ist ein JavaScript Framework zum Designen von Benutzeroberflächen (Quelle (X)). Der Quellcode ist teilweise in JavaScript, teils in HTML geschrieben. Zum Entwickeln mit React Native kann ein beliebiger Editor verwendet werden. Zum optimalen Vergleich wird auch die React Native Anwendung gegen die API 23, die Major Version 6 (Marshmallow), kompiliert.

8.4.1 Grafische Benutzeroberfläche

Im Gegensatz zur Entwicklung mit dem Ionic Framework und auch mit Xamarin, steht die Implementierung der grafischen Benutzeroberfläche und die der Fachlogik nicht in getrennten Dateien. Die grafischen Elemente werden zwar wie bei dem Ionic Framework in HTML implementiert, dieser HTML-Code ist allerdings im JavaScript-Quellcode eingebettet.

React Native unterstützt bereits einige gängige UI-Elemente, wie zum Beispiel den *FloatingActionButton*. Sollte man allerdings etwas Spezielles benötigen, was nicht in den React Native Bibliotheken enthalten ist, so können auch manuell weitere native Komponenten der Anwendung hinzugefügt werden. Die Basis-Quellcode-Dateien für die beiden Systeme Android und iOS liegen direkt unter dem Root-Verzeichnis der Anwendung. Möchte man ausschließlich gemeinsamen Code verwenden, so kann

8.4 Implementierung mit React Native

man eine neue gemeinsame Basis-Quelldatei anlegen und in den plattformspezifischen Dateien diese importieren.

Für ein wie in Kapitel (X) gefordertes Menü bietet das React Native Framework den sogenannten *DrawerNavigator*. Um diesen zu nutzen muss er zunächst oben in der Basis-Quelldatei importiert werden (siehe Listing (X)).

Listing 8.28: Import der Klasse *DrawerNavigator* für das Menü in React Native

```
1 import { DrawerNavigator } from 'react-navigation';
```

Eine Seite, hier ein Screen, wird durch eine eigene Klasse dargestellt, welche von der Klasse *React.Component* erben muss. Der Titel der Seite wird nebst anderen Einstellungen wie ein Icon in den statischen *navigationOptions* festgelegt (siehe Listing (X)).

Listing 8.29: Festlegen des Seitentitels

```
1 class MyHomeScreen extends React.Component {
2   static navigationOptions = {
3     drawer: () => ({
4       label: 'Home',
5       icon: ({ tintColor }) => (
6         <Image source={require('./images/ic_home_black_24dp.png')} style={[styles.icon, {←
7           tintColor: tintColor}]}/>
8       ),
9     })
10   ...
11 }
```

Zuletzt müssen die einzelnen Seiten, zu denen man navigieren können soll einem *DrawerNavigator*-Objekt zugeordnet werden. Dies geschieht, wie man in Listing (X) sehen kann in der Konstanten *DrawerApp*.

8.4 Implementierung mit React Native

Listing 8.30: Konfiguration des *DrawerNavigators*

```
1 const DrawerApp = DrawerNavigator({
2   Home: {
3     screen: MyHomeScreen,
4   },
5   Kamera: {
6     screen: CameraRollGallery,
7   },
8   Senoren: {
9     screen: CameraRollGallery,
10  },
11 ...
12});
```

Nachfolgend ist in Abbildung (X) das Menü der React Native Anwendung dargestellt. Es lässt sich wie gewohnt über Wischen öffnen und schließen. Anders als die Menüs der anderen Frameworks und das native Menü hat das Menü der React Native Anwendung keinen integrierten Header.



Abbildung 8.4: Menü in der React Native Anwendung

8.4 Implementierung mit React Native

Einen *FloatingActionButton* bringt das React Native Framework nicht direkt mit. Ein *TouchableHighlight*-Objekt lässt sich allerdings so modifizieren, dass es aussieht und sich verhält wie ein *FloatingActionButton* von Android. Das *TouchableHighlight*-Objekt wird in der *render()*-Funktion der Klasse der Seite deklariert, auf dem der FAB zu sehen sein soll (siehe Listing (X)).

Listing 8.31: Deklaration eines *TouchableHighlight*-Objekts als *FloatingActionButton*

```
1 render() {
2     return (
3         <TouchableHighlight style={styles.addButton} underlayColor="#ff7043" onPress={() => ←
4             this.props.navigation.navigate('Camera')}
5             <Text style={{fontSize: 40, color: 'white'}}>+</Text>
6         </TouchableHighlight>
7     );
8 }
```

Über *style=styles.addButton* wird dem Objekt sein Design zugeordnet. Das *FloatingActionButton*-Design steckt dabei in dem Attribut *addButton*. In diesem wurden Größe, Form, Position, Verhalten etc. definiert, wie in Listing (X) zu sehen ist.

Listing 8.32: Das *FloatingActionButton*-Design

```
1 addButton: {
2     backgroundColor: '#ff5722',
3     borderColor: '#ff5722',
4     borderWidth: 1,
5     height: 75,
6     width: 75,
7     borderRadius: 50,
8     alignItems: 'center',
9     justifyContent: 'center',
10    position: 'absolute',
11    bottom: 20,
12    right: 20,
13    shadowColor: "#000000",
14    shadowOpacity: 0.8,
15    shadowRadius: 2,
16    shadowOffset: {
17        height: 1,
18        width: 0
19    }
20},
```

8.4.2 Sensoren

Möchte man in einer React Native Anwendungen auf die Sensoren des Smartphones zugreifen, so bietet sich hierfür die *react-native-sensor-manager*-Bibliothek an. Sensoren, auf die über den Sensor Manager zugegriffen werden können sind: der Beschleunigungssensor, das Gyroskop, das Magnetometer, der Lagesensor, das Thermometer, der Lichtsensor und der Näherungssensor. Das GPS ist direkt über die API des React Native Frameworks ansprechbar.

Um den Sensor Manager zu nutzen muss dieser zunächst über die Kommando-Shell installiert via *npm* installiert und die Bibliothek verlinkt werden (siehe Listing (X)).

Listing 8.33: Installation und Verlinkung der Bibliothek *react-native-sensor-manager*

```
1 npm i react-native-sensor-manager --save  
2 react-native link react-native-sensor-manager
```

Fehlen Berechtigungen, so können diese manuell im *AndroidManifest* unter */android/app/src/main* hinzugefügt werden. Anschließend muss die Klasse *SensorManager* aus dem Paket *NativeModules* in der Quelldatei importiert werden (siehe Listing (X)).

Listing 8.34: Import der *SensorManager*-Klasse

```
1 import { SensorManager } from 'NativeModules';
```

Das Auslesen und Präsentieren von Sensordaten wird in diesem Fall wieder anhand des Näherungssensors gezeigt. Der Sensor wird im Konstruktor der Seite gestartet, auf der die Daten des Sensors präsentiert werden sollen. Über die Methode *startProximity()* wird der Näherungssensor gestartet (siehe Listing (X)). Die übergebene Zahl '100' bedeutet eine Verzögerung von mindestens 100ms zwischen den Events. Mithilfe der Klasse *DeviceEventEmitter* aus der *react-native*-Bibliothek wird ein *Listener* hinzugefügt.

8.4 Implementierung mit React Native

Listing 8.35: Auslesen und Anzeigen der Daten des Näherungssensors

```
1 constructor(props) {
2     super(props);
3     SensorManager.startProximity(100);
4     DeviceEventEmitter.addListener('Proximity', function (data) {
5         if (data.isNear === true) {
6             ToastAndroid.show('Near',ToastAndroid.SHORT);
7         } else {
8             ToastAndroid.show('Far',ToastAndroid.SHORT);
9         }
10    });
11 }
```

8.4.3 Kameras

Um die Kamerafunktionen des Smartphones in einer React Native Anwendung nutzen zu können, muss zunächst die Bibliothek *react-native-camera* via *npm* installiert werden. Anschließend muss die Bibliothek noch mit dem Befehl *react-native link react-native-camera* verknüpft und in der Quelldatei importiert werden. Gegebenenfalls müssen noch Berechtigungen im *AndroidManifest*, welches unter */android/app/src/main* liegt, hinzugefügt werden.

Anders als beim Ionic und Xamarin Framework gibt es im React Native Framework direkte Möglichkeit die Android Kameraanwendung innerhalb der React Native Anwendung zu starten. Aus diesem Grund wurden Elemente wie der Auslösebutton und die Blitzeinstellungen manuell der Seite mit der Kamerafunktion hinzugefügt. Die Designs der einzelnen UI-Elemente wurden wie das Design des *FloatingActionButton* unter *const styles* hinzugefügt. Listing (X) zeigt beispielhaft das Design des Auslösebuttons.

Listing 8.36: Das Design des Auslösebuttons für die Kamerafunktion

```
1 const styles = StyleSheet.create({
2 ...
3 captureButton: {
4     padding: 15,
5     backgroundColor: 'white',
6     borderRadius: 40,
7 },
8 ...
9 })
```

8.4 Implementierung mit React Native

In der `render()`-Funktion werden die einzelnen Elemente der Benutzeroberfläche deklariert. Das oben (Listing(X)) gezeigte Design wird hier einem `TouchableOpacity`-Objekt zugeordnet, welches dann den Auslösebutton darstellt (siehe Listing (X)).

Listing 8.37: Deklaration eines `TouchableOpacity`-UI-Objekts für die Darstellung des Auslösebuttons der Kamera

```
1 render() {
2     return (
3         ...
4         <TouchableOpacity
5             style={styles.captureButton}
6             onPress={this.takePicture}
7         >
8             <Image source={require('./images/ic_photo_camera_36pt.png')} />
9         </TouchableOpacity>
10    ...
11 );
12 }
```

Wird der Button gedrückt, so wird die Methode `takePicture()` aufgerufen, welche nachfolgend in Listing (X) dargestellt ist. Innerhalb dieser Methode wird die Methode `capture()` des Kameraobjekts `camera` aufgerufen. Dies ist die eigentliche Auslösefunktion der Kamera. Das geschossene Bild wird automatisch mit Timestamp im Namen im öffentlichen Bilder-Verzeichnis des Android Smartphones gespeichert.

Listing 8.38: Die Methode `takePicture()` zum Auslösen der Kamera

```
1 takePicture = () => {
2     if (this.camera) {
3         this.camera.capture()
4             .then((data) => console.log(data))
5             .catch(err => console.error(err));
6     }
7 }
```

Das oben in Listing (X) verwendete Kameraobjekt `camera` wird im Konstruktor der Klasse initialisiert (siehe Listing (X)). Hier werden dem Objekt Voreinstellungen mitgegeben. Beim Start der Kameraanwendung ist in diesem Fall die Rückkamera des Smartphones aktiv und der Blitz ist auf automatisch eingestellt.

8.4 Implementierung mit React Native

Listing 8.39: Initialisierung des Kameraobjekts im Konstruktor

```
1 ...
2 camera: {
3     aspect: Camera.constants.Aspect.fill,
4     captureTarget: Camera.constants.CaptureTarget.cameraRoll,
5     type: Camera.constants.Type.back,
6     orientation: Camera.constants.Orientation.auto,
7     flashMode: Camera.constants.FlashMode.auto,
8 },
9 isRecording: false
10 ...
```

Für das Ändern des Blitzmodus wird ein UI-Element, welches selbst den aktuell ausgewählten Blitzmodus anzeigt, oben rechts im Bild gesetzt. Durch tippen auf dieses Element wird durch die 3 verschiedenen Blitzoptionen *on*, *off* und *auto* rotiert. Listing (X) zeigt die Initialisierung dieses Elements.

Listing 8.40: Initialisierung des UI-Elements zum Wechseln des Blitzmodus

```
1 ...
2 <TouchableOpacity
3     style={styles.flashButton}
4     onPress={this.switchFlash}
5 >
6     <Image source={this.flashIcon}/>
7 </TouchableOpacity>
8 ...
```

Die Methode *switchFlash()* übernimmt das Rotieren durch die Blitzoptionen. Wie man in Listing (X) erkennen kann, besitzt das Kameraobjekt ein Attribut *flashMode*, welches entsprechend dem momentan ausgewählten Modus bei Aufruf der Methode *switchFlash()* einen anderen Modus zugewiesen bekommt. Die Methode *flashIcon()* ist wie die Methode *switchFlash()* aufgebaut, nur das in ihr das entsprechende Icon des UI-Elements für die Blitzoptionen gesetzt wird.

8.4 Implementierung mit React Native

Listing 8.41: Die Methode `switchFlash()` für das Ändern des Blitzmodus

```
1 switchFlash = () => {
2     let newFlashMode;
3     const { auto, on, off } = Camera.constants.FlashMode;
4
5     if (this.state.camera.flashMode === auto) {
6         newFlashMode = on;
7     } else if (this.state.camera.flashMode === on) {
8         newFlashMode = off;
9     } else if (this.state.camera.flashMode === off) {
10        newFlashMode = auto;
11    }
12
13    this.setState({
14        camera: {
15            ...this.state.camera,
16            flashMode: newFlashMode,
17        },
18    });
19 }
```

Die manuell erstellte Benutzeroberfläche für die Bedienung der Kamerafunktionen in der React Native Anwendung ist nachfolgend in Abbildung (X) dargestellt. In der linken oberen Ecke ist das Element zum Wechseln zwischen Rück- und Frontkamera. Die Funktionsweise dieses Elements entspricht der der Blitzoptionen.

8.4 Implementierung mit React Native



Abbildung 8.5: Kamerafunktion in der React Native Anwendung

8.4.4 Speicherzugriffe

Um die geschossenen Fotos, oder bei Bedarf auch weitere Bilder, die auf dem Smartphone gespeichert sind, zu laden wird die Methode `getPhotos()` der Klasse `CameraRoll` aufgerufen (siehe Listing (X)). Die Klasse `CameraRoll` ist aus dem Plugin `rn-camera-roll`, welches zuvor über `npm` installiert und anschließend in der Quelldatei importiert werden musste. Der Methode `getPhotos()` können hierbei verschiedene Optionen mitgegeben werden, die unter anderem angeben, welche Bilddateien geladen werden sollen. Ein paar Möglichkeiten sind hier `Library`, `All` oder `Album`. Per Default wird `SavedPhotos` mitgegeben.

8.4 Implementierung mit React Native

Listing 8.42: Aufruf der Methode `getPhotos()` für die Anzeige gespeicherter Bilder in einer Galerie

```
1 fetchPhotos(count = PHOTOS_COUNT_BY_FETCH, after) {
2     CameraRoll.getPhotos({
3         first: count,
4         after,
5     }, this.onPhotosFetchedSuccess.bind(this), this.onPhotosFetchError.bind(this));
6 }
```

9 Auswertung

Im folgenden Kapitel wird die Auswertung anhand der in Kapitel (X) aufgestellten Bewertungskriterien beschrieben. Zu den einzelnen Kategorien (siehe Kapitel (X)) wird der jeweilige ausgefüllte Abschnitt der Bewertungsmatrix dargestellt und erläutert.

9.1 Kosten und Lizenz

Nachfolgende Abbildung (X) zeigt den Abschnitt 'Kosten und Lizenz' der ausgefüllten Bewertungsmatrix.

Kosten, Lizenz	Xamarin	React Native	Cordova/Ionic 2
Fixkosten Framework	499\$ Kauflizenz für Professional	Keine Kaufoption	Keine Kaufoption
Subscription	Professional: 50\$ pro Monat, Enterprise 250\$ pM	Keine	Business: 99\$ pro Monat, Enterprise individuell
Supportkosten	ab 50\$ pro Monat	Kein bezahlter Support	ab 99\$ pro Monat

Abbildung 9.1: Bewertungsmatrix Kategorie Kosten und Lizenz

Das Xamarin Framework gibt es in einer kostenlosen Community-Version und in den kostenpflichtigen Versionen 'Professional' und 'Enterprise'. Die kostenlose Community-Version gibt es für Windows inklusive einer Community-Version von Visual Studio und für Apple mit der IDE 'Xamarin Studio'. Die Preise der kostenpflichtigen Versionen richten sich nach den Preisen der entsprechenden Version der IDE Visual Studio. Die Professional-Version kann für 499\$ ohne Subscription oder für 1199\$ mit Subscription erworben werden. Eine Subscription hält 2 Jahre, ein Auffrischen danach kostet 799\$. Die Enterprise-Version kann nur mit Subscription erworben werden. Sie kostet 5999\$ und ein Auffrischen nach Ablauf der 2 Jahre kostet 2569\$. Einen technischen Support erhält man mit jeder Subscription, das bedeutet ab umgerechnet 50\$ pro Monat. E-Mail-Support erhalten alle Business- und Enterprise-Kunden. Zusätzlich zu allen Modellen kann die Xamarin Test-Cloud ab einen monatlichen Preis von 99\$ genutzt werden.

9.2 Support und Community

Für das Framework React Native gibt es keine kostenpflichtigen Versionen.

Für das Framework Cordova gibt es ebenfalls keine kostenpflichtigen Modelle. Das Framework Ionic bietet dagegen folgende Varianten an: eine kostenlose Community-Version und die kostenpflichtigen Versionen 'Indie', 'Business' und 'Enterprise'. Im Gegensatz zu den kostenpflichtigen Modellen bei Xamarin, fallen bei den Modellen von Ionic monatliche Subscription-Gebühren an. Die Indie-Variante kostet 25\$ pro Monat und die Business-Variante kostet 99\$ pro Monat. Für die Enterprise-Version gibt es einen individuellen Preis auf Anfrage. Ab der Business-Variante erhält man E-Mail-Support, das bedeutet ab 99\$ pro Monat. Möchte man die Ionic Test Cloud nutzen, so kostet dies 20\$ pro Monat und Anwendung.

9.2 Support und Community

9.3 Entwicklung

Die ausgefüllte Bewertungsmatrix für die Kategorie 'Entwicklung' ist in nachfolgender Abbildung (X) dargestellt.

Entwicklung	Xamarin	React Native	Cordova/Ionic 2
Installation/Benutzbarkeit	X		
Einarbeitungszeit/Implementierung		X	
Anteil plattformübergreifender Code	X		
Umfang der Bibliotheken		X	X

Abbildung 9.2: Bewertungsmatrix Kategorie Entwicklung

Hat man bereits ein Visual Studio installiert und möchte Xamarin integrieren, so ist dies nicht ohne Mehraufwand möglich. Einfacher ist es, Visual Studio direkt zusammen mit Xamarin zu installieren. Hierzu muss nur der Installer von der Xamarin Homepage heruntergeladen und ausgeführt werden. Alles Notwendige für die Entwicklung mit Xamarin ist bei der Installation von Visual Studio bereits vorausgewählt. Der Xamarin Installer installiert zusätzlich noch weitere benötigte Komponenten, wie ein Android SDK und NDK. Es muss keine weitere Software separat beschafft werden. Startet man nach der erfolgreichen Installation die IDE Visual Studio, so kann direkt mit der Entwicklung mit Xamarin gestartet werden, indem ein neues Projekt 'Android App' angelegt wird. Anders als bei der nativen Entwicklung mit Android Studio gibt es hier allerdings keine Design-Vorauswahl bei der Projekterstellung wie zum Beispiel einen *NavigationDrawer* für die Navigation. Dafür gibt

9.3 Entwicklung

es sogenannte 'Pre Built Apps', das sind beispielhafte fertige Anwendungen, wie unter anderem Shopping- oder CRM-Anwendungen.

Auf der Xamarin Homepage findet sich einiges an Beispielcode zu verschiedenen Funktionalitäten. Viele dieser Beispielprojekte sind allerdings nicht direkt kompilierbar. Es ist immer aufwendiges recherchieren notwendig, welche Verweise, Components oder Einstellungen im Projekt angepasst werden müssen, da diese Informationen nicht mit angegeben sind. Für die Verwendung von zum Beispiel nativen UI-Elementen müssen sogenannte 'Components' installiert und dem Projekt hinzugefügt werden. Welche 'Components' für ein gewünschtes Feature benötigt werden muss selbst recherchiert werden. Bei der Auswahl dieser 'Components' gibt es oft Schwierigkeiten, da manche 'Components' zwingend in einer übereinstimmenden Version installiert sein müssen, um Versionskonflikte zu vermeiden. Welche 'Components' übereinstimmende Versionen haben müssen, kann nur durch Recherche oder Probieren ermittelt werden. Hierbei muss zusätzlich darauf geachtet werden, dass eventuell nicht in allen Versionen das gewünschte Feature enthalten ist.

Xamarin Anwendungen werden, wie schon in Kapitel (X) erwähnt, mit der objekt-orientierten Programmiersprache C# entwickelt. Da C# vom Aufbau her Java sehr ähnlich ist, ist es für einen Android- oder Java-Entwickler keine große Umstellung. Die Projektstruktur einer Android-Anwendung mit Xamarin ist identisch mit der nativen Projektstruktur. Die Layout-Dateien können mit wenigen Anpassungen von einer nativen Android-Anwendung übernommen werden. Mit diesen Voraussetzungen ist das Portieren einer nativen Android-Anwendung nach Xamarin unkompliziert. Die API für Zugriffe auf Hardware wie Sensoren und Speicher wirkt wie eine 1 zu 1 Umsetzung der Android API. Viele Klassen heißen gleich oder sehr ähnlich. Auch die Verwendung deckt sich mit der nativen. Dies macht es zwar jedem Android-Entwickler leicht, sich in Xamarin einzuarbeiten, jedoch bedingt dies zugleich mit sich, dass eben diese Code-Abschnitte nicht plattformübergreifend nutzbar sind. Ein höherer Anteil an plattformübergreifenden Code kann mit der Nutzung von Xamarin.Forms erreicht werden, wobei dann auf native UI-Elemente verzichtet werden muss. Zum Umfang der von Xamarin zur Verfügung gestellten Bibliotheken ist zu sagen, das im Rahmen dieser Arbeit keine Android-Funktionalität gefunden wurden, die nicht mit Xamarin umsetzbar waren.

Auf der Homepage von Cordova fand ich leider keine schrittweise Installationsanleitung, man musste sich von mehreren Stellen die Informationen, darüber, was an

9.3 Entwicklung

Software installiert werden muss, zusammensuchen. Auch die Installation selbst verlief nicht ganz reibungslos, da es immer wieder zu Problemen bezüglich Pfadangaben wie zum Beispiel das JAVA_HOME-Verzeichnis kam. Unter dem 'Get Startet'-Abschnitt findet sich erst keine direkte Anleitung zum Aufbau und der Entwicklung von Anwendungen und Plugins mit Cordova. Es wird lediglich die Installation und das Erstellen eines Projektes erklärt. Die Projektstruktur wird nicht näher erläutert und es bleibt unklar an welcher Stelle sich Source-Dateien zu befinden haben. In den Tutorials wird nur aufgezeigt, welche Berechtigungen und Methoden für verschiedene Hardware-Nutzungen wie Beschleunigungssensor oder Kamera benötigt werden. Es gibt aber kein 'Rezept' für eine erste lauffähige Anwendung. Da für diese Arbeit allerdings nur fertige Cordova Plugins in einer Ionic Anwendung genutzt werden, fällt dieser Aspekt bei der Bewertung des Zusammenspiels von Cordova und Ionic nicht groß ins Gewicht.

Ist Cordova bereits installiert, gestaltet sich die Installation von Ionic denkbar einfach: es muss nur ein Befehl in einer Kommando-Shell ausgeführt werden. Eine Ionic Anwendung kann mit jedem beliebigen Editor entwickelt werden. Dies bringt einerseits Freiheiten, andererseits gibt es nur bedingt eine Auto vervollständigung und keine Code-Generierungsmöglichkeiten. Ionic bietet beim Erstellen einer neuen Anwendung direkt 2 Navigationselemente mit an: eine Tableiste und einen Navigation-Drawer. Ionic bietet für sämtliche UI-Elemente eine Übersicht mit Beispielcode. Für Zugriffe auf Hardware-Elemente wie Sensoren oder Kamera werden Cordova-Plugins installiert und der Anwendung hinzugefügt. Welches Plugin für welche Funktionalität benötigt wird ist auf der Homepage von Ionic beschrieben. Benötigte Berechtigungen müssen nicht manuell hinzugefügt werden. Wie schon beim Xamarin Framework sind leider auch bei Ionic einige Beispiele nicht direkt kompilierbar.

Da das Ionic Framework in der Version 2 erst Januar 2017 erschienen ist, gibt es noch nicht für alle Funktionalitäten Beispiele. Viele Beispiele beziehen sich noch auf das Ionic 1 Framework. Da in Ionic 2 nicht direkt in JavaScript, sondern in TypeScript entwickelt wird, sind viele Ionic 1 Beispiele nicht 1 zu 1 in Ionic 2 umsetzbar. TypeScript bringt im Gegensatz zu 'reinem' JavaScript auch objektorientierte Strukturen mit sich¹. Die 'ionic-native'-Bibliothek stellt Klassen zur Verfügung, die die Funktionalitäten der Cordova-Plugins kapseln und so für eine Implementierung mit TypeScript nutzbar machen. Leider fehlen in dieser Bibliothek noch einige Sensoren, wie zum Beispiel der Näherungssensor. Man muss allerdings auch bei einer Ionic 2

¹ TypeScript 2017.

9.3 Entwicklung

Anwendung dadurch nicht gänzlich auf die Verwendung dieser Sensoren verzichten. Externe oder selbstgeschriebene Cordova-Plugins können weiterhin in die Anwendung integriert und genutzt werden. Auf diese Weise muss dann aber meist auf die Nutzung von TypeScript in diesen Teilen der Anwendung verzichtet und JavaScript verwenden werden. Die Benutzeroberfläche einzelner Seiten einer Anwendung wird in separaten HTML-Dateien erstellt. Anwendungen werden direkt für alle 3 Plattformen, Android, iOS und Windows Phone, implementiert. Je nachdem für welche Plattform die Anwendung am Schluss gebaut wird, werden die einzelnen UI-Elemente optisch automatisch angepasst. Der Anteil an plattformübergreifenden Code ist dadurch sehr hoch. Zum Umfang der existierenden Bibliotheken lässt sich sagen, dass leider vieles, besonders im Bereich der Sensor-Nutzung, noch nicht in der Ionic 2 API integriert ist. Dies macht die Verwendung einiger Funktionalitäten momentan noch umständlich.

Die 'Get Startet'-Anleitung auf der React Native Homepage führt einen Schritt für Schritt durch die Installation über die Kommando-Shell bis zum Start einer ersten leeren Anwendung auf dem Testgerät. Wie auch bei dem Ionic Framework kann der Code einer React Native-Anwendung mit jedem beliebigen Editor geschrieben werden. Erstellt werden die Anwendungen über eine Kommando-Shell. Auf viele Vorteile einer Entwicklung in einer IDE wie Android Studio oder Visual Studio muss auch hier verzichtet werden. Bei der Erstellung einer neuen Anwendung gibt es bei React Native leider keine vorgefertigten Navigationselemente. Dafür bietet die Ionic Homepage ausführliche und hilfreiche Guides für die Integration und Verwendung verschiedener Navigationselemente.

9.4 Hersteller

9.5 OS-Versionen

9.6 Funktionsumfang

9.7 GUI-Design

9.8 Interoperabilität

9.9 Tests

9.10 Performance

9.11 Programmiersprache

9.12 Sicherheit

Literatur

- Android* (2016). URL: https://www.android.com/intl/de_de/ (besucht am 16.02.2016).
- Android Design Principles* (2017). URL: <https://developer.android.com/design/get-started/principles.html> (besucht am 27.01.2017).
- Android Guidelines* (2017). URL: <https://developer.android.com/design/index.html> (besucht am 27.01.2017).
- Android Magazine* (2017). URL: <http://androidmag.de/report/ein-wahres-sensibelchen-handy-sensoren/> (besucht am 16.01.2017).
- Android Operating System* (2016). URL: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system)) (besucht am 16.02.2016).
- Android SDK* (2016). URL: http://developer.android.com/guide/practices/ui_guidelines/index.html (besucht am 16.02.2016).
- Android UI Guidelines* (2016). URL: http://developer.android.com/guide/practices/ui_guidelines/index.html (besucht am 16.02.2016).
- Android Wiki* (2017). URL: <https://www.droidwiki.de/wiki/> (besucht am 16.01.2017).
- AngularJS: Ionic* (2017). URL: <https://angularjs.de/artikel/ionic-tutorial-deutsch> (besucht am 27.01.2017).
- Apache Cordova* (2017). URL: <https://cordova.apache.org/docs/en/latest/guide/overview/index.html> (besucht am 27.01.2017).
- App-Entwicklung Varianten* (2016). URL: <https://www.upwork.com/hiring/mobile/should-you-build-a-hybrid-mobile-app/> (besucht am 01.07.2016).
- Balzert, H. (1996). *Lehrbuch der Softwaretechnik*. Springer Verlag.
- Cinar, O. (2015). *Android Quick APIs Reference*. Apress.
- Darwin, Ian F. *Android Cookbook*. O'Reilly Media.
- Google Keyword Planer* (2017). URL: https://adwords.google.com/ko/KeywordPlanner/Home?__c=6792367252&__u=2274549778&authuser=0&__o=cues#search (besucht am 16.01.2017).
- Google Trends* (2017). URL: <https://www.google.de/trends/?hl=de> (besucht am 16.01.2017).

Literatur

- iOS* (2017). URL: <https://developer.apple.com/> (besucht am 16.01.2017).
- iOS GUI Guidelines* (2017). URL: <https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/> (besucht am 16.01.2017).
- Manjoo, Farhad. "A Murky Road Ahead for Android, Despite Market Dominance". In:
- native Anwendungen* (2017). URL: <http://searchsoftwarequality.techtarget.com/definition/native-application-native-app> (besucht am 16.01.2017).
- Statista Anzahl mobiler Anwendungen* (2017). URL: <https://de.statista.com/statistik/daten/studie/20150/umfrage/anzahl-der-im-store-verfuegbaren-applikationen-fuer-das-apple-iphone/> (besucht am 11.02.2017).
- TypeScript* (2017). URL: <http://www.typescriptlang.org/> (besucht am 07.04.2017).
- Ünlü, S. (2016). *Ionic – Grundlagen*. video2brain.