ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Strukturierter Vergleich aktueller Frameworks zur Entwicklung mobiler Anwendungen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Dominik Schwer
dominik.schwer@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Johannes Schobel

2017

Fassung 17. Dezember 2017

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Mit dem starken Wachstum des Smartphone-Marktes über die letzten Jahre hat die Entwicklung mobiler Anwendungen zunehmend an Bedeutung gewonnen. Eine der größten Herausforderungen ist dabei die Entwicklung für mehrere Zielplattformen. Möchte man seine Anwendung für Android und iOS parallel entwickeln, um den Großteil des Marktes zu erschließen, bedeutet das faktisch, dass die Anwendung zweimal entwickelt werden muss. Abhilfe schaffen sogenannte Cross-Plattform Frameworks, mit denen parallel für beide großen Plattformen entwickelt werden kann. Aufgrund der großen Auswahl und des sehr unterschiedlichen Funktionsumfangs dieser Frameworks ist es sehr schwierig das *"richtige"* Framework für die eigenen Anforderungen zu finden. Daher stellt sich die Frage welche Frameworks in welchen Anwendungsfällen am besten geeignet sind.

Ziel dieser Arbeit ist einen Überblick der aktuell existierenden Ansätze zu geben und bei der Entscheidung für ein bestimmtes Framework zu helfen. Dafür werden zunächst allgemein die Entwicklungsansätze erklärt und anschließend zu jedem dieser Ansätze, stellvertretend ein Framework vorgestellt. Diese Frameworks werden dann, anhand zuvor definierter Kriterien, verglichen und mit Hilfe einer Methode aus der Entscheidungstheorie im Kontext von drei unterschiedlichen Anwendungsfällen bewertet. Die Ergebnisse dieser Auswertungen zeigen, dass nicht pauschal entschieden werden kann, welches Framework am besten geeignet ist. Hauptgrund dafür ist, dass die Frameworks je nach Anwendungsfall unterschiedlich gut abschneiden. Das in dieser Arbeit beschriebene Vorgehen kann als Vorlage dienen, um eine Entscheidung über das verwendete Framework zu treffen.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Erstellung dieser Bachelor-Arbeit unterstützt haben. Das gilt vor allem für alle, die meine Arbeit Korrektur gelesen und Feedback gegeben haben.

Ein besonderer Dank gilt meinem Betreuer Johannes Schobel, der mir immer mit Rat zu Seite stand und mich mit stets konstruktiver Kritik voran gebracht hat. Vielen Dank für deine Geduld und Mühen.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Problemstellung 2

1.2 Zielsetzung 2

1.3 Struktur der Arbeit 3

2 Grundlagen **5**

2.1 Ansätze zur Entwicklung mobiler Anwendungen 5

2.2 JavaScript 7

2.2.1 ECMAScript 2015 7

2.2.2 TypeScript 10

2.2.3 Angular 13

3 Cross-Plattform Frameworks **17**

3.1 MD^2 17

3.1.1 Architektur 17

3.1.2 MD^2-DSL 18

3.2 React Native 23

3.2.1 Architektur 23

3.2.2 Grundlegende Funktionsweise 26

3.2.3 Native Komponenten 28

3.3 Ionic 30

3.3.1 Architektur 30

3.3.2 Grundlegende Funktionsweise 32

3.3.3 Command Line Interface 34

3.4 Weitere Frameworks und Ausblick 35

4 Evaluation **39**

4.1 Kriterien 39

4.1.1 Harte Kriterien 40

4.1.2 Weiche Kriterien 41

4.2 Vergleich der Frameworks 42
4.2.1 Lizenz und Kosten 42
4.2.2 Unterstützung plattformspezifischer Funktionen 44
4.2.3 Entwicklungsunterstützung 44
4.2.4 Benutzeroberfläche 45
4.2.5 Einstiegsschwierigkeiten 45
4.2.6 Wartbarkeit 46
4.2.7 Performance 47
4.2.8 Langlebigkeit 48
4.3 Auswertung mit dem Analytischen Hierarchieprozess 48
4.3.1 Analytischer Hierarchieprozess 48
4.4 Anwendungsfälle mit Auswertung 51
4.4.1 Portierung einer Webanwendung 53
4.4.2 Geschäftsanwendungen 55
4.4.3 Anwendungen mit Facebook Integration 57

5 Verwandte Arbeiten **61**

6 Zusammenfassung **65**

A AHP-Webanwendung Eingaben **69**

1

Einleitung

Der Smartphone-Markt ist über die letzten Jahre weltweit rapide gewachsen. Gartner schätzt die Absatzzahlen von Smartphones in 2018 auf über 1,6 Milliarden[1]. Aufgrund dieser Entwicklung finden immer mehr mobile Anwendungen ihren Weg in unseren Alltag oder werden vom klassischen Desktop-PC auf das Smartphone verlagert. Die meisten großen Webanwendungen existieren mittlerweile in einer mobilen Version, um mit eingeschränktem Funktionsumfang auch offline verfügbar zu sein. Entsprechend steigen die Anforderungen an Softwareentwickler, die nun nicht mehr nur für eine Zielplattform entwickeln müssen, sondern durch den mobilen Bereich mindestens zwei weitere Plattformen (Android und iOS) bedienen müssen. Damit verbunden existiert ein höherer Entwicklungsaufwand, denn mit jeder weiteren Plattform muss oftmals eine weitere Technologie erlernt werden. Zudem wird die Wartung durch die vielen separaten Projekte ebenfalls deutlich zeitaufwändiger. Das hat, unter Umständen, eine signifikante Kostensteigerung zur Folge. Das betrifft insbesondere kleinere bis mittelgroße Unternehmen, die diese Kosten nicht ohne Weiteres tragen können. Je nach verkauftem Produkt kann eine mobile Anwendung jedoch wichtig sein, um konkurrenzfähig bleiben zu können. Einen Lösungsansatz für diese Problematik bieten eine immer größer werdende Anzahl von sogenannten Cross-Plattform Frameworks. Über die letzten Jahre hinweg sind nicht nur viele solcher Frameworks entstanden, auch die Ansätze, die von diesen Frameworks verfolgt werden, sind vielfältiger geworden.

[1] https://www.gartner.com/newsroom/id/3816763, 12.12.2017

1.1 Problemstellung

Das Angebot von Cross-Plattform Frameworks wurde insbesondere in der jüngeren Vergangenheit immer unübersichtlicher. Die Frameworks unterscheiden sich zum einen in den verwendeten Technologien und zum anderen in ihrem Funktionsumfang im Vergleich zu nativen Anwendungen. Entsprechend ist nicht jedes Framework für jeden Anwendungsfall uneingeschränkt geeignet. Besonders problematisch wird es bei neuen Gerätefunktionen, die von Cross-Plattform Frameworks nicht sofort unterstützt werden können. Bevor man sich für die Verwendung eines Frameworks entscheidet, gilt es daher sehr genau abzuwägen, welches sich im Einzelfall am besten eignet. Eine solche Entscheidung zu treffen ist, aufgrund der vielen Faktoren die beachtet werden müssen, nicht trivial. Aus diesem Grund sollte versucht werden, möglichst viele zukünftige Anwendungsfälle zu definieren, bevor die Frage *"Welches Framework eignet sich für diesen Anwendungsfall am besten?"* beantwortet wird.

1.2 Zielsetzung

In dieser Arbeit werden zwei wesentliche Ziele verfolgt. Zuerst soll ein Überblick über derzeitige Entwicklungsansätze mobiler Cross-Plattform Frameworks gegeben werden. Zu diesem Zweck wird für jeden Ansatz ein stellvertretendes Framework vorgestellt. Darüber hinaus werden diese Frameworks einem strukturierten Vergleich unterzogen, um festzustellen, welches der Frameworks in drei unterschiedlichen Anwendungsfällen am besten geeignet ist. Dafür werden zunächst Kriterien definiert, anhand dieser die Frameworks strukturiert verglichen werden. Die abschließende Bewertung, im Kontext des jeweiligen Anwendungsfalls, erfolgt mit Hilfe einer mathematischen Methode aus der Entscheidungstheorie. Dieses Vorgehen hat zum Ziel, möglichst nachvollziehbar zu sein, damit es als Vorlage für zukünftige Entscheidungen über das zu verwendende Framework dienen kann.

1.3 Struktur der Arbeit

Zu Beginn werden in Kapitel 2 einige technischen Grundlagen erklärt, die für das Verständnis der vorgestellten Frameworks nötig sind. Darauf folgt ein Kapitel, in dem die Architektur, sowie die grundlegende Funktionsweise jedes Frameworks erklärt wird. Anschließend folgt in Kapitel 4 die Auswertung der Frameworks in drei Schritten. Dazu werden die Kriterien für die Bewertung definiert, die Frameworks anhand dieser Kriterien gegenübergestellt und im Kontext von drei Anwendungsfällen ausgewertet. Im darauffolgenden Kapitel 5 werden verwandte Arbeiten vorgestellt, worauf in Kapitel 6 eine abschließende Zusammenfassung der Arbeit folgt.

2

Grundlagen

Dieses Kapitel vermittelt wichtige Grundlagen, die für den weiteren Verlauf der Arbeit wichtig sind. In Kapitel 2.1 wird ein Überblick zu den verschiedenen Herangehensweisen der Anwendungsentwicklung für mobile Endgeräte gegeben. In Kapitel 2.2 werden moderne Konzepte von JavaScript erläutert, die in den behandelten Frameworks zum Einsatz kommen.

2.1 Ansätze zur Entwicklung mobiler Anwendungen

Mobile Anwendungen sind mittlerweile in vielen Fällen keine rein nativen Anwendungen mehr oder werden zumindest nicht mehr in der nativen Programmiersprache der Zielplattform entwickelt. Je nachdem, welche Art der Entwicklung gewählt wird, ergeben sich andere Einschränkungen bezüglich der Programmiersprache (z.B. Java/Swift oder HTML5 und JavaScript), der Architektur (z.B. Model-View-Controller) und der Funktionalität (z.B. neue wenig verbreitete Hardwarefunktionen). Dementsprechend muss vor Beginn der Entwicklung genau überlegt werden, welcher Ansatz gewählt werden soll. Im Folgenden werden fünf aktuell verwendete Ansätze vorgestellt, siehe [1].

- **Webanwendung:** Diese Art von Anwendung ist in der Regel in den gängigen Web-Technologien (HTML5, CSS und JavaScript) entwickelt und mit Frameworks wie jQuery Mobile für mobile Endgeräte optimiert. Ausgeführt wird die mobile Anwendung in einem mobilen Webbrowser. Ein daraus resultierender Vorteil ist, dass die Anwendung nicht installiert werden muss. Allerdings unterliegt sie den

Abbildung 2.1: Verschiedene Arten zur Entwicklung von mobilen Anwendungen nach [1]

Limitierungen des Browsers, der sie ausführt und kann somit, im Vergleich zu einer nativen Anwendungen, wesentlich weniger Funktionalität anbieten.

- **Native Anwendungen:** Native Anwendungen verwenden die Standard Programmierschnittstellen und verwenden die nativen Programmiersprachen (z.B. Java bei Android bzw. ObjectiveC bei iOS). Anwendungen, die mit solchen Technologien entwickelt werden, erfordern den höchsten Aufwand, da man sämtliche plattformspezifischen Besonderheiten beachten muss. Der größte Vorteil ist jedoch, dass sämtliche bereitgestellten Funktionen der entsprechenden Plattform genutzt werden können und die bestmögliche Performance der Anwendung erreicht werden kann.

- **Hybride Anwendungen in nativem Container:** In diesem Fall werden die selben Technologien wie bei der Web Anwendung verwendet. Allerdings wird die Anwendung nicht in einem Webbrowser ausgeführt, sondern läuft auf einem nativen Container (z.B. iOS WebView). Dieser Container kann wiederum auf Gerätefunktionen wie Sensordaten oder den Netzwerkstatus zugreifen und bietet somit mehr Funktionalität als ein klassischer Webbrowser.

- **Hybride Anwendungen mit nativer Komponente:** Diese Art von hybriden Anwendungen verwenden zwar ebenfalls Webtechnologien, allerdings wird je nach Framework ein Teil der Anwendung in nativen Code übersetzt. Eine Möglichkeit

wäre beispielsweise, die Benutzeroberfläche nativ zu halten, um das gleiche Look and Feel einer nativen Anwendung zu bieten. Der Übergang zwischen diesen Anwendungen und nativer Anwendungen ist dabei fließend und unterscheidet sich stark von Framework zu Framework.

- **Cross-Compiled Anwendungen:** Cross-Compiled Anwendungen werden, wie der Name vermuten lässt, von einer vom Framework festgelegten Programmiersprache in die plattformspezifischen nativen Sprachen übersetzt. Welche Sprache für die Entwicklung verwendet wird hängt dabei stark vom Framework ab (z.B. C# bei Xamarin). Manche Frameworks entfernen sich dabei sogar von klassischen Programmiersprachen und verwenden eigene deklarative Sprachen (z.B. MD2 [2]). Das Ergebnis ist jedoch in jedem Fall eine native Anwendung. Funktionalität und Entwicklungsaufwand lassen sich für diese Anwendungen nur schwer klassifizieren, da sie sich von Framework zu Framework stark unterscheiden.

2.2 JavaScript

In diesem Kapitel werden zunächst die für das Verständnis der Frameworks wichtigsten Neuerungen von ECMAScript erklärt. Die darauffolgenden Abschnitte behandeln Erweiterungen zu JavaScript, die in den Frameworks Ionic (TypeScript und Angular) und React Native (JSX) verwendet werden.

2.2.1 ECMAScript 2015

ECMAScript 2015 ist die sechste Version des standardisierten Sprachkerns (daher auch ECMAScript 6 genannt) von JavaScript und bringt so viele neue Konzepte, die zwar noch nicht von allen Web-Browsern unterstützt werden, aber in den meisten JavaScript-Frameworks zum Einsatz kommen. Im Codebeispiel 2.1 werden die sechs wichtigsten Neuerungen diskutiert. Der Code ist an einigen Stellen bewusst umständlich geschrieben, um die Unterschiede zu ECMAScript 5 deutlich zu machen.

```
1  //rectangle.js
2  export default class Rectangle {
3
4      constructor(height, width){
5          this.height = height;
6          this.width = width;
7      }
8
9      get area(){
10         let area = 0;
11         if(this.height > 0 && this.width > 0){
12             area = this.height * this.width;
13         }
14         return area;
15     }
16
17     set area(value){
18         this.area = value;
19     }
20 }
21
22 export let createRecArray = () => {
23     let array = [];
24     array.push(new Rectangle(2,3));
25     array.push(new Rectangle(3,3));
26     array.push(new Rectangle(4,3));
27 }
28
29 export let printRecAreas = (recArray) => {
30     for(let rec of recArray){
31         console.log(rec.area);
32     }
33 }
```

Listing 2.1: Wichtige Neuerungen in ECMAScript 6

```
1  //main.js
2  import * as rec from './rectangle';
3
4  let array = createRecArray();
5  printRecAreas(array); //prints 6 9 12
6
7  [a,b] = array;
8  console.log(a.area); //prints 6
9  console.log(b.area); //prints 9
```

Listing 2.2: Wichtige Neuerungen in ECMAScript 6

- **let:** Die erste auffällige Neuerung ist das Schlüsselwort `let`, welches an Stelle von `var` verwendet wird. In den meisten Fällen kann man sie äquivalent verwenden, da der einzige Unterschied darin liegt, dass Variablen, die mit `let` deklariert werden, block-scoped sind. Über `var` deklarierte Variablen sind function-scoped. Man könnte beispielsweise die Deklaration der Variable area (Zeile 10) in den

`if`-Block ziehen, wenn man `var` verwendet. Macht man dies jedoch mit `let`, ist der `return`-Wert undefined.

- **for/of-Schleife:** Mit der neuen `for..of-Schleife` können sogenannte `iterable objects`, wie Arrays und Sets durchlaufen werden[1]. Damit werden die `areas` (Zeile 30/31) der rectangles aus dem Array ausgegeben.

- **Arrow-Funktionen:** Die Arrow-Funktionen stellen eine Kurzschreibweise von Funktionsdefinitionen dar und erlauben damit in manchen Fällen wesentlich kompakteren Code. In Zeile 22 wird diese Schreibweise verwendet, um die Funktion `createRecArray()` zu definieren, die keine Parameter erwartet. In Zeile 29 wird auf die selbe Art und Weise `printRecAreas()` dieses mal mit Parameter definiert.

- **Klassen:** Mit ECMAScript 6 wurde eine Syntax für Klassen eingeführt, wie wie man sie aus klassischen objektorientierten Sprachen kennt. Die Klasse `Rectangle` (Zeile 2-20) stellt ein solches Beispiel dar. Im Konstruktor könnte alternativ mit dem Schlüsselwort `super` der Konstruktor einer Eltern-Klasse aufgerufen werden, wenn die Klasse `Rectangle` mit `extends` von einer entsprechenden Klasse erben würde. Die Getter und Setter funktionieren jedoch anders als man es z.B. aus Java kennt, denn sie werden nur für dynamisch berechnete Werte wie die Variable `area` verwendet. Aus diesem Grund existieren für `height` und `width` keine entsprechenden Methoden. Diese Variablen werden durch den Konstruktor implizit definiert und entsprechen in etwa den `public`-Variablen aus Java. Es gibt bislang noch keine Möglichkeit, `private`-Variablen wie in anderen objektorientierten Programmiersprachen zu deklarieren.

- **Module** gab es bis ECMAScript 6 nicht ohne Weiteres in JavaScript. Es gab zwar mehrere Standards (z.B. CommonJS und Asynchronous Module Definition[2]) aus der Community, die allerdings untereinander nicht kompatibel sind. Mit ECMAScript 6 sind Module jetzt fester Bestandteil von JavaScript. Im Code wird die Klasse Rectangle mittels `export` in Zeile 2 und die zugehörigen Funktionen in Zeile 22/29

[1] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/for...of, 29.05.2017
[2] http://requirejs.org/docs/commonjs.html, 29.05.2017

als Modul für main.js zur Verfügung gestellt. Dort wird in Zeile 2 alles aus dem Modul mit `import *` importiert.

- **Destructuring:** Seit ECMAScript 6 existiert eine neue Syntax, die es erlaubt, aus Objekten und Arrays direkt Variablen zu erzeugen. Genau das geschieht in Zeile 7 aus Listing 2.2. In der Variable `array` befinden sich drei Rectangle-Objekte, wovon die ersten beiden jeweils `a` bzw. `b` zugewiesen werden. So können beliebig viele Zuweisungen aus einem Array oder Objekt auf eine Zeile gekürzt werden.

- **Promises:** JavaScript bietet ab ECMAScript 6 die Möglichkeit mit Promise-Objekten asynchrone Berechnungen durchzuführen. Dabei gibt eine asynchrone Funktion ein Promise-Objekt zurück, das bei Erfolg das Ergebnis der Berechnung enthält oder andernfalls einen entsprechenden Fehler liefert. Beim Aufrufen der Funktion wird mit der Syntax `asyncFunc().then(result => {...}).catch(error => {...});` [3] jeweils definiert was passieren soll, nachdem das Promise-Objekt zurückgegeben wurde. Je nachdem ob die Berechnung erfolgreich war, wird die auf `then` folgende Funktion ausgeführt oder im Fehlerfall die auf `catch` folgende Funktion.

2.2.2 TypeScript

TypeScript ist ein Superset von JavaScript, das nach JavaScript übersetzt werden muss, bevor es von Browsern ausgeführt werden kann. Somit umfasst es alle Features von JavaScript, sowie einige zusätzliche z.B. im Bereich der Objektorientierung. Zwei wesentliche Eigenschaften, die TypeScript von JavaScript unterscheiden, sind die starke Typisierung und dass es sich bei TypeScript um eine kompilierende Sprache handelt. Beim Kompilieren wird in klassisches JavaScript übersetzt. Dabei kann zwischen ECMAScript 3, 5 und 6 als Zielversion gewählt werden. Im Folgenden werden zunächst die Komponenten der Sprache kurz erläutert und anschließend die Umsetzung der Typisierung und der Objektorientierung erklärt.

[3] http://exploringjs.com/es6/ch_promises.html#sec_examples-promises, 11.07.2017

Komponenten von TypeScript

TypeScript besteht im Wesentlichen aus drei Komponenten[4]:

- Die Sprache selbst bestehend aus Syntax, Schlüsselwörtern und den Typ-Annotations.
- Der **TypeScript Compiler**, der TypeScript in JavaScript übersetzt. Beim Compiler (tsc) handelt es sich schlicht um eine JavaScript-Datei, die eine TypeScript-Datei (.ts) als Eingabe erwartet und eine JavaScript-Datei daraus erzeugt (z.B. `tsc test.ts` erzeugt eine neue Datei `test.js`).
- Der **Language Service**, der Funktionen, wie man sie aus IDEs (z.B. IntelliJ) kennt, bereitstellt. Dazu gehören Funktionen wie automatische Statement-Komplettierung, Code-Formatierung oder farbliche Hervorhebung.

Typisierung in TypeScript

Eine der zentralen Erweiterungen zu JavaScript in TypeScript ist die starke Typisierung, wie man sie aus Hochsprachen, wie Java oder C#, kennt. Alle unterstützten Typen werden in der folgenden Tabelle mit kurzer Beschreibung aufgeführt. Syntaktisch werden Variablen nach dem Schema `let [identifier] : [type-annotation] = value;` definiert.

Eine Besonderheit gegenüber anderen stark typisierten Sprachen sind die Union-Type-Variablen und der `any` Typ. Letzteres ermöglicht die Typisierung zu umgehen, da eine mit `any` deklarierte Variable Werte von allen Typen annehmen kann. Falls man mehr als einen Typ verwenden möchte, ohne jedoch alle zuzulassen, kann man an Stelle von `any` mit `|` getrennt mehrere Typen angeben.

[4] https://www.tutorialspoint.com/typescript/typescript_overview.htm, 29.05.2017

Tabelle 2.1: Datentypen von TypeScript[5]

Datentyp	Schlüsselwort	Beschreibung
Boolean	`boolean`	Repräsentiert die logischen Werte `true` und `false`
Number	`number`	Repräsentiert alle numerischen Werte, egal ob ganze Zahl oder Bruchzahl. Neben der dezimalen Darstellung sind auch binär, oktal und hexadezimal zulässig.
String	`string`	Repräsentiert alle textuellen Datentypen
Enumeration	`enum`	Erlaubt es numerischen Werten semantische Namen zu geben.
Void	`void`	Wird verwendet für Funktionen ohne Rückgabewert.
Null	`null`	Repräsentiert die Abwesenheit eines Objekts
Undefined	`undefined`	Repräsentiert den Standardwert aller Variablen, die nicht initialisiert wurden

Klassen in Typescript

```
1  class Car{
2      static topSpeed:number = 250;
3      private _ps: number;
4      public age: number;
5
6      constructor(ps:number, age:number){
7          this._ps = ps;
8          this.age = age;
9      }
10
11     toString():void{
12         console.log("This Car has " + this._ps + " PS");
13     }
14 }
15
16 let car = new Car(120,5);
17 console.log(car._ps); //error _ps is private
18 console.log(car.age); //5
19 car.toString(); //This Car has 120 PS
```

Listing 2.3: Klassen in TypeScript

Mit ECMAScript 6 werden zwar bereits Klassen in JavaScript eingeführt, jedoch ohne den objektorientierten Ansatz vollständig umzusetzen, wie beispielsweise in Java. TypeScript erweitert JavaScript an dieser Stelle um die Schlüsselwörter `public, private und protected`, um die aus anderen Sprachen bekannte Kapselung zu realisieren. `public` und `private` verhalten sich dabei genau so, wie es in Java der Fall ist. Das `protected`

[5] https://www.typescriptlang.org/docs/handbook/basic-types.html, 31.05.2017

Schlüsselwort führt bei TypeScript dazu, dass im Gegensatz zu mit `private` deklarierten Variablen auch innerhalb von erbenden Klassen auf die Variable zugegriffen werden kann. Darüber hinaus können in TypeScript auch statische Variablen mit `static` deklariert werden. In Listing 2.3 wird beispielhaft eine Klasse `Car` definiert. Methoden und Variablen sind, falls sie nicht anders deklariert werden, automatisch `public`, daher funktioniert der Aufruf von `toString()` (Zeile 19).

2.2.3 Angular

In diesem Kapitel werden einige Grundlagen des Web-Frameworks Angular erklärt, da das Ionic Framework direkt auf Angular aufbaut. Dabei beschränkt sich dieses Kapitel lediglich auf die Komponenten, die in Ionic unverändert zum Einsatz kommen. Auf die Funktionsweise des Routings wird beispielsweise nicht eingegangen, da das Navigieren zwischen den verschiedenen Views einer Anwendung anders abläuft als in Ionic.

Components

```
1  import { Component, OnInit } from '@angular/core';
2
3  import { Hero }        from './hero';
4  import { HeroService } from './hero.service';
5
6  @Component({
7    selector: 'my-dashboard',
8    templateUrl: './dashboard.component.html',
9    styleUrls: [ './dashboard.component.css' ]
10 })
11 export class DashboardComponent implements OnInit {
12   heroes: Hero[] = [];
13
14   constructor(private heroService: HeroService) { }
15
16   ngOnInit(): void {
17     this.heroService.getHeroes()
18       .then(heroes => this.heroes = heroes.slice(1, 5));
19   }
20 }
```

Listing 2.4: Components in Angular, siehe [3]

Components beinhalten die Logik einer Angular Anwendung. Jede Component ist dabei immer nach dem selben Schema aufgebaut, wie z.B. die `DashboardComponent` aus dem offiziellen Angular Tutorial.

Zunächst wird das Component-Symbol importiert, um eine Component deklarieren zu können. Darüber hinaus werden in diesem Fall Daten (Zeile 3) und ein Service (Zeile 4) aus anderen Quelldateien der Anwendung importiert. Darauf folgt der Component-Decorator `@Component`, der die Metadaten für die Component enthält. In der Regel beinhaltet dieser immer einen `selector`, um die Component innerhalb einer Parent-Component referenzieren zu können, eine `templateUrl`, die auf den html-Markup verweist und eine oder mehrere `styleUrls`, die auf CSS-Dateien verweisen. Die Templates sind für die Darstellung der Anwendung zuständig und werden wie gewohnt durch die CSS-Dateien formatiert. Das Kernstück der Component ist die Klasse, die, wie aus anderen Sprachen üblich, Attribute (Zeile 12) und Methoden (Zeile 16) enthält. Im Beispiel 2.4 wird zudem eine Instanz eines `HeroService` erzeugt (Zeile 14). Was Services sind und wie sie funktionieren, wird in einem der nächsten Unterkapitel erklärt.

Templates und Directives

Templates sind für die Darstellung einer Angular Anwendung zuständig. Dabei handelt es sich um html-Markup, welches mit Angularbefehlen, die für die Darstellung und Manipulation von Daten genutzt wird, angereichert ist.

```
1 <div *ngIf="hero">
2   <h2>{{hero.name}} details!</h2>
3   <div><label>id: </label>{{hero.id}}</div>
4   <div>
5     <label>name: </label>
6     <input [(ngModel)]="hero.name" placeholder="name"/>
7   </div>
8 </div>
```

Listing 2.5: Components in Angular, siehe [3]

In Listing 2.5 kommt diese Syntax drei mal zum Einsatz. Die einfachste Variante ist in Zeile 2 zu sehen, in der mit den doppelten geschweiften Klammern auf das `hero`-Objekt zugegriffen wird und das `name`-Attribut angezeigt wird. Dies dient ausschließlich der Anzeige von Daten. Sollen die Daten hingegen durch den Nutzer bearbeitet werden

können, muss stattdessen `[(ngModel)]` (Zeile 6) verwendet werden, um einen bidirektionalen Datenfluss zu gewährleisten. Darüber hinaus können innerhalb von templates sogenannte `directives` verwendet werden. So wird beispielsweise in Zeile 1 mit `*ngIf="hero"` sichergestellt, dass die Anwendung nicht abstürzt, falls `hero` keinen Wert enthält. Das `div`-Element wird in diesem Fall nicht angezeigt. Ein weiteres Beispiel wäre `<li *ngFor="let hero of heroes»hero.name</li>`, mit dem aus einem Array namens `heroes` eine Liste erzeugt wird. Sämtliche directives können in der offiziellen Dokumentation[6] gefunden werden.

Services

Services stellen eine Möglichkeit dar, um wiederverwendbaren Code zu schreiben. So kann ein Data Service wie der `HeroService` in Listing 2.4 in jede beliebige Component injiziert werden. Da Services asynchron sind, erhalten wir eine Promise als Antwort auf unsere Anfrage (`getHeroes()`) und verarbeiten diese mit `then`, sobald die Daten zur Verfügung stehen.

[6] https://angular.io/docs, 28.08.2017

3

Cross-Plattform Frameworks

3.1 MD^2

MD^2 (model driven) wurde an der WWU Münster entwickelt. In [2] wird das Framework als *modellgetriebenes Cross-Plattform-Framework für die Implementierung mobiler Applikationen* beschrieben, wobei die Anwendungen in der eigens entwickelten domänenspezifischen Sprachen MD^2-DSL spezifiziert und zu nativem Code transformiert werden. MD^2 wurde gezielt auf datengetriebene Business Apps zugeschnitten. Diese umfassen die typischen CRUD-Funktionalitäten (`Create`, `Read`, `Update`, `Delete`) und besitzen sehr simple Benutzerschnittstellen, die sich im Wesentlichen aus Textfeldern, Labeln, Buttons, Menüs und Listen zusammensetzen. Dementsprechend eignet sich dieses Framework nur für die spezifizierte Domäne und ist insbesondere ungeeignet für Anwendungen, die komplexe Algorithmen ausführen müssen oder mehr Freiheiten bei der Gestaltung der Benutzeroberfläche benötigen. Die unterstützten Plattformen beschränken sich dabei auf iOS und Android. In diesem Kapitel wird zunächst kurz auf die Architektur des Frameworks eingegangen und im Anschluss die Funktionsweise der domänenspezifischen Sprache MD^2-DSL nach [2] erklärt.

3.1.1 Architektur

Das MD^2-Framework fällt in die Klasse der Cross-Compiled Anwendungen (siehe Kapitel 2.1). Dabei wird der in MD^2-DSL geschriebene Quellcode mittels drei verschiedener Code-Generatoren übersetzt. Zwei davon dienen der Übersetzung für die beiden unterstützten Zielplattformen iOS und Android. In beiden Fällen werden komplett native

Anwendungen erzeugt. Bei Android werden Versionen ab 3.0 unterstützt. Der dritte Generator generiert für den Backend-Server eine JavaEE-Anwendung, die als Serverschnittstelle für die Android- und iOS-Anwendung fungiert. Diese Anwendung ist zwar vollständig lauffähig und setzt sämtliche CRUD-Funktionalitäten um, ist aber eher als Startpunkt für eine individuelle Implementierung eines Servers gedacht. Ein detaillierte Beschreibung der Generatoren kann in [4] nachgelesen werden.

Das Framework folgt dem MVC-Entwurfsmuster (Details im nächsten Kapitel), u.a. um die Übersetzung in nativen Code zu vereinfachen. In Android existiert beispielsweise mit der Aufteilung in XML (View) und Java (Controller) jeweils ein Gegenstück zu den Definitionen der Benutzerschnittstelle und der Logik in MD^2-DSL. Die Datenhaltung (Model) erfolgt über herkömmliche Java-Objekte, die mit Annotationen versehen werden, um die Serialisierung nach JSON zu steuern. Sämtliche Daten werden lokal in JSON gespeichert und gegebenenfalls an den Server gesendet.

3.1.2 MD^2-DSL

Domänenspezifische Sprachen (DSL) können als Programmiersprachen auf sehr hohem Abstraktionsniveau angesehen werden, die speziell auf einen bestimmten Problemkontext bzw. eine bestimmte Domäne zugeschnitten wurden. DSLs sind normalerweise auch ohne Programmierkenntnisse leicht erlernbar, da sie überwiegend deklarativ, weniger technisch und insbesondere weniger umfangreich sind. In [2] wird die DSL, die für dieses Framework entwickelt wurde, ausführlich vorgestellt und dient daher als Grundlage für dieses Kapitel.

Im Fall der MD^2-DSL handelt es sich bei der Domäne um Business Applikationen, also mobile Anwendungen, die in einem geschäftlichen Kontext verwendet werden. Die Sprache wurde nach einem *Top-Down*-Ansatz entwickelt, d.h. die einzelnen Sprachelemente sollen jeweils eine spezifische Anforderung der fertigen Anwendung abbilden. Diesem Ansatz folgend, hat sich nach [2] die folgende Liste ergeben.

Nach [2] soll die Sprache es Anwendungsentwicklern ermöglichen,

1. Datentypen zu definieren,
2. lokal und serverseitig Datensätze dieser Typen anzulegen, zu lesen, zu aktualisieren und zu löschen, d.h. die typischen CRUD-Operationen auf Instanzen der Typen auszuführen;
3. die Benutzeroberfläche mit verschiedenen Layouts und typischen Steuerelementen zu beschreiben, besonders wichtig sind dabei Registerkarten (Tabs);
4. die Benutzernavigation zwischen den Ansichten zu steuern;
5. Datenbindung und Eingabevalidation zu definieren;
6. auf Benutzerereignisse und Statusänderungen zu reagieren; und
7. gerätespezifische Funktionen wie GPS oder Kamera zu nutzen.

Neben diesen funktionalen Anforderungen setzt MD²-DSL einige nicht-funktionale Anforderungen um, wie Trennung von Logik, Daten und GUI sowie Modularisierung. Dabei folgt die Sprache einer MVC-Architektur, d.h. Datenmodell (Model), Benutzerschnittstelle (View) und Logik (Controller) müssen jeweils in separaten Dateien gespeichert werden. Da MD²-DSL vorwiegend deklarativ ist, beschreiben Entwickler, was die Anwendung erreichen soll, anstatt den algorithmischen Ablauf zu formulieren. Deutlich wird das anhand der Benutzeroberfläche, bei der das Aussehen und die Zusammensetzung beschrieben werden muss. Wie diese schrittweise aufgebaut wird, ist für den Entwickler irrelevant.

Datenmodellierung

Daten werden in MD²-DSL durch sogenannte Entities beschrieben. Nach dem Entity-Schlüsselwort mit Bezeichner folgt eine Liste von Eigenschaften mit Typangabe, wobei neben den klassischen Typen (String, Integer, usw.) auch andere Entities als Typ ver-

wendet werden können (Zeile 7). Zudem können Typparameter genutzt werden, um eine Eigenschaft als optional zu spezifizieren (Zeile 4) oder den Wertebereich einzuschränken. Neben Entities werden lediglich Enumerations unterstützt, um die Komplexität der Sprache möglichst klein zu halten. Konzepte wie Vererbung werden beispielsweise absichtlich nicht unterstützt.

```
1 entity PRODUKT {
2   name : string
3   preis : integer
4   beschreibung : string(optional)
5 }
6 entity BESTELLUNG {
7   produkt : PRODUKT
8   email : string
9 }
```

Listing 3.1: Datenmodellbeispiel aus [2]

Benutzerschnittstellen

```
1  TabbedPane APPFENSTER {
2    SUCHENTAB -> Suchen
3    BESTELLENTAB -> Bestellen(tabTitle "Bestellungen")
4    INFOTAB(tabTitle "Info")
5  }
6  FlowLayoutPane SUCHENTAB(vertical) {
7    Label sucheLbl {text "Suche nach Produkt" style GROSS}
8    TextInput suchFeld { label "Produktname"
9      tooltip "Geben Sie den Namen des Produkts ein ..."
10   }
11   Button sucheBtn ("Suchen")
12 }
13 FlowLayoutPane BESTELLENTAB(vertical) {
14   Label bestellenLbl { text "Bestellung aufgeben" style GROSS }
15   Label info("Bitte geben Sie Ihre E-Mail-Adresse an ...")
16   AutoGenerator bestellung { contentProvider BestellungProvider }
17   Button bestellenBtn ("Bestellen")
18 }
19 FlowLayoutPane { ... }
20 style GROSS { fontSize 20 textStyle bold }
```

Listing 3.2: Benutzerschnittstellenbeispiel aus [2]

Für das Erstellen einer Benutzerschnittstelle existieren zwei Sprachkomponenten. Zum einen Inhaltselemente, wie Labels, Buttons oder Texteingabefelder und zum anderen Container, in denen die Inhaltselemente gruppiert werden. Die verschiedenen Contai-

nerarten werden `Panes` genannt und stellen jeweils ein anderes Layout dar und können beliebig ineinander geschachtelt werden.

Im Beispiel 3.2 werden `FlowLayoutPanes` in `TabbedPanes` geschachtelt. Dabei repräsentiert jede `FlowLayoutPane` eine Registerkarte. Um den Code übersichtlich zu halten und eine Modularisierung der Oberfläche sowie die Wiederverwendbarkeit einzelner Komponenten zu ermöglichen, müssen Anzeigeelemente nicht direkt geschachtelt werden. Stattdessen können die einzelnen Elemente separat definiert (Zeile 6ff und 13ff) und in der Vaterkomponente referenziert (Zeile 2-4) werden. Zusätzlich zu den Standardelementen wie Labels gibt es die sogenannten `AutoGenerators`, die eine Standarddarstellung für Entities darstellen, mit entsprechenden Eingabefeldern und Labels für die Eigenschaften der Entity. Außerdem stellen sie implizit eine Datenverbindung zwischen den Inhaltselementen und den Objekten eines Datenlieferanten (`contentProvider`) aus dem Contoller her.

Kontrolllogik

Die wichtigste Funktion des Controllers ist Model und View zusammen zu führen. Darüber hinaus kann auf globale Ereignisse reagiert werden, wie den Abbruch der Netzwerkverbindung oder einen sehr niedrigen Akkustand. Auch die Nutzung gerätespezifischer Funktionen (z.B. GPS) ist im Controller möglich.

Ein `main`-Block wird für jede Anwendung benötigt, da dort neben der Definition von Name und Version der Anwendung die Initialisierung angestoßen wird. Damit wird der Anfangszustand der Benutzeroberfläche festgelegt und es wird die Ausführung erster Aktionen gestartet. Aktionen sind die am häufigsten verwendeten Elemente im Controller, denn sie dienen der Definition aller dynamischen Aspekte. Mit den individuellen `CustomActions` wird eine Sequenz von Aktionsfragmenten definiert. Da ein Großteil der Logik ereignisbasiert abläuft, ist eine Art von Fragmenten dafür zuständig die Ausführung von Aktionen an Ereignisse zu binden. Konkret wird im Beispiel die Aktion `produktLaden` an das Drücken des Suche-Buttons in der `SUCHENTAB`-View gebunden (Zeile 10). Andere Fragmente werden zu Navigation (Zeile 18) oder für die CRUD-Operationen Speichern (Zeile 21), Laden (Zeile 15) und Löschen verwendet.

Datenquellen werden über Datenlieferanten (`contentProvider`) angebunden, wobei lokale und entfernte Datenquellen (Zeile 27) mit der Anwendung verknüpft werden können. Ein Datenlieferant verweist immer nur auf einen durch einen Filter (Zeile 24) eingeschränkten Ausschnitt der Datenquelle. Der Typ wird eines solchen Lieferanten wird genau wie bei den Eigenschaften der Entities angegeben und kann dementsprechend entweder auf einen vordefinierten Datentyp oder auf eine Entity (Zeile 26) verweisen.

```
1  main {
2    appName "Bestellapp" appVersion "ATPS 2013" modelVersion "1.0"
3    startView APPFENSTER.Suchen
4    onInitialized init
5  }
6  action CombinedAction init {
7    actions ereignisseRegistrieren validatorenBinden
8  }
9  action CustomAction ereignisseRegistrieren {
10   bind action produktLaden on SUCHENTAB.sucheBtn.onTouch
11   bind action bestellungAufgeben on BESTELLENTAB.bestellenBtn.onTouch
12 }
13 action CustomAction validatorenBinden { ... }
14 action CustomAction produktLaden {
15   call DataAction(load suchergebnis)
16   call NewObjectAction(bestellungProvider)
17   call AssignObjectAction(use suchergebnis for bestellungProvider.produkt)
18   call GotoViewAction(APPFENSTER.Bestellen)
19 }
20 action CustomAction bestellungAufgeben {
21   call DataAction(save bestellungProvider)
22 }
23 contentProvider PRODUKT suchergebnis { providerType backend
24   filter first where PRODUKT.name equals APPFENSTER.Suchen->SUCHENTAB.suchFeld
25 }
26 contentProvider BESTELLUNG bestellungProvider {providerType backend }
27 remoteConnection backend {uri "http://..."}
```

Listing 3.3: Kontrolllogikbeispiel aus [2]

Bei Datenbindungen zwischen Datenlieferanten und Inhaltselementen gibt es zwei Besonderheiten, die den Entwickler entlasten sollen. Zum einen stellt eine solche Verbindung Konsistenz zwischen GUI und Daten sicher und zum anderen wird bei einem Eingabefeld implizit gegen den zugehörigen Datentyps validiert (inklusive der im Datenmodell definierten Einschränkungen). Dieser Validator kann im Controller durch entsprechende Aktionsfragmente ergänzt werden, um beispielsweise zu überprüfen, ob die Eingabe einem regulären Ausdruck entspricht.

3.2 React Native

Seit 2015 steht das von Facebook entwickelte Cross-Plattform-Framework *React Native* kostenlos zur Verfügung. Unterstützte Zielplattformen umfassen iOS- und Android-Anwendungen. Bei den Anwendungen handelt es sich um hybride Anwendungen mit nativer Komponente. Bei React Native ist der Anteil des nativen Codes sehr variabel. Während die Benutzeroberfläche komplett nativ zusammengebaut wird, kann die Logik der Anwendung sowohl in JavaScript als auch in Objective-C bzw. Java programmiert werden. In diesem Kapitel wird zunächst die Architektur des Frameworks erklärt. Anschließend wird anhand einiger Codesbeispiele die grundlegende Funktionsweise erklärt. Abschließend wird gezeigt, wie nativer Code in React Native verwendet wird und wann dies sinnvoll ist.

3.2.1 Architektur

React Native baut direkt auf der JavaScript-Bibliothek *React* auf. Die Stärke von React liegt in der Erstellung von Benutzerschnittstellen. Interessant ist dabei ein Feature, das die Entwicklung von React Native möglich gemacht hat: Der Virtual DOM ist eine im Zwischenspeicher vorgehaltene Repräsentation des DOMs, der im Browser gerendert wird. Im Web-Kontext existiert dieses Feature aus Performancegründen, da Veränderungen am DOM zunächst nur im Zwischenspeicher berechnet werden und anschließend nur die Unterschiede zum ursprünglichen DOM im Browser neu gerendert werden. Abbildung 3.1 illustriert diesen Vorgang.

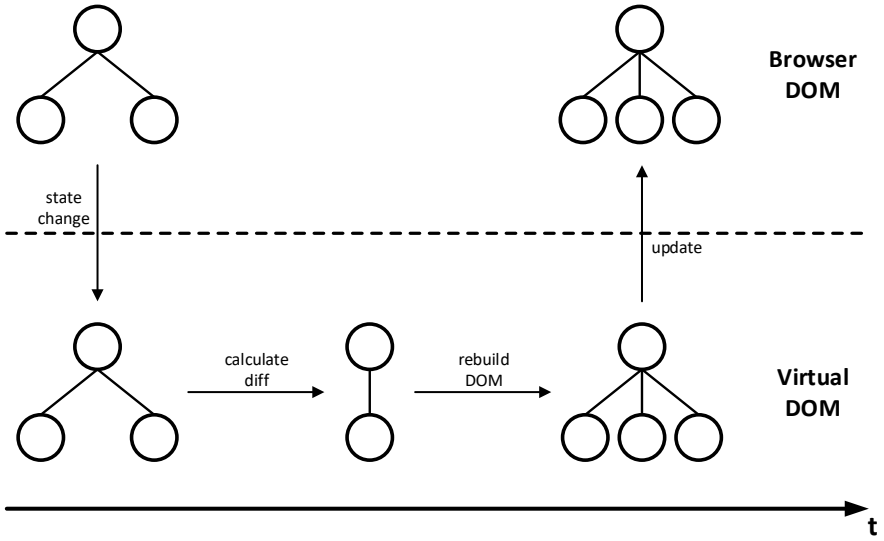


Abbildung 3.1: Virtual DOM von React nach [5]

Der große Vorteil an diesem Verfahren liegt für React Native in der zusätzlichen Abstraktionsschicht zwischen dem Code des Entwicklers und dem, was letztlich vom Browser gerendert wird. Konkret bedeutet das, anstatt einen DOM im Browser zu aktualisieren, werden Objective-C und Java APIs aufgerufen, um entsprechende iOS- oder Android-Komponenten zu rendern.

Wie React Native intern funktioniert, wurde auf der ReactiveConf 2015 [6] vorgestellt, kurz nachdem das Framework zum Open-Source-Projekt erklärt wurde. Abbildung 3.2 veranschaulicht abstrakt, wie eine React Native Anwendung aufgebaut ist. Auf der einen Seite ist der native Code in Java und C++, der insbesondere für das Rendering der GUI und den Zugriff auf die gerätespezifischen APIs zuständig ist. Auf der anderen Seite ist die JavaScript-Virtual Machine (JS VM), in der der JavaScript Code ausgeführt wird, in welchem die Anwendung entwickelt wurde. Die `Bridge` ist für die Kommunikation zwischen nativem Code und JavaScript Code zuständig. Jede Aktion startet dabei auf der "nativen Seite". Wenn zum Beispiel die Anwendung neu startet, wird in Java der Befehl `AppRegistry.runApplication()` aufgerufen. Dieser Aufruf wird serialisiert

und über die `Bridge` and die JS VM geschickt. Die JS VM sendet daraufhin eine Antwort, in der festgelegt wird, welche Komponenten in der Benutzeroberfläche gerendert werden müssen.

Abbildung 3.2: Architektur von React Native am Beispiel von Android nach [6]

In React Native gibt es 3 Threads. Die `UI Event Queue` ist der Main-Thread, den es in jeder Art von Anwendung gibt. Dazu kommen mit der `Native Modules Event Queue` und der `JS Event Queue` zwei weitere Threads. Abbildung 3.3 zeigt beispielhaft, was bei einer Nutzerinteraktion (z.B. ein Touch-Event) in den jeweiligen Threads berechnet wird. Die Aufteilung in zwei Threads mit nativem Code und einem mit JavaScript Code hat mehrere Gründe. Zum einen reagiert die Benutzeroberfläche auch nach dem Touch-Event noch auf die Bedienung des Nutzers, da das Ereignis in einem eigenen Thread behandelt wird. So kann zum Beispiel weitergescrollt werden, während auf die Berechnung der `JS Event Queue` gewartet wird. Darüber hinaus wird der Main-Thread zusätzlich entlastet, indem das Neuberechnen des Layouts in die `Native Modules Queue` ausgelagert wird. Zudem können komplexe Berechnungen, für die JavaScript weniger geeignet ist, ebenfalls in die `Native Modules Event Queue` ausgelagert werden (siehe Kapitel 3.2.3).

Die bisherigen Erklärungen haben sich auf Android bezogen. Die iOS-Architektur ist sehr ähnlich und funktioniert in den Grundzügen identisch. Genauere Einblicke bietet ein Blogeintrag[1] von Tadeu Zahallo, der ebenfalls an der Entwicklung von React Native beteiligt war, primär im Bereich von iOS.

[1] https://tadeuzagallo.com/blog/react-native-bridge/, 06.11.2017

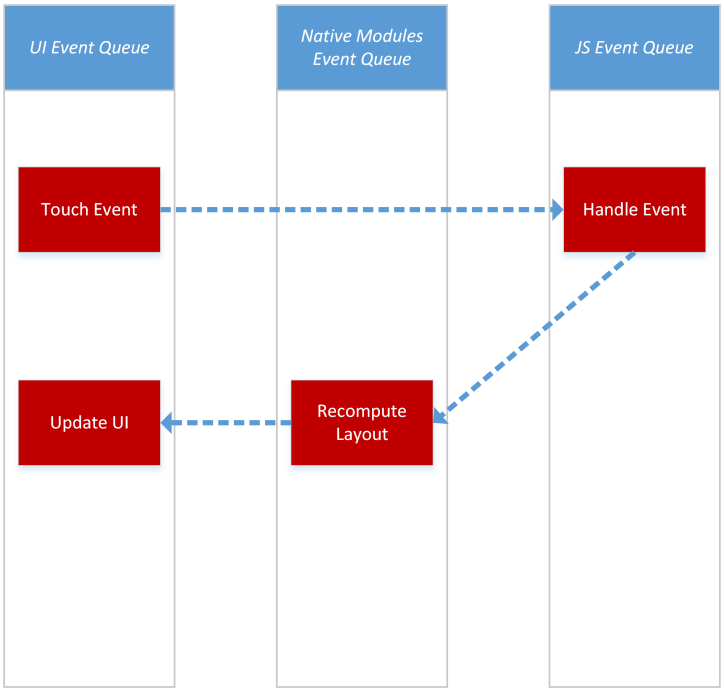


Abbildung 3.3: Abarbeitung eines Touch-Events nach [6]

3.2.2 Grundlegende Funktionsweise

Um die grundlegende Funktionsweise von React Native zu verstehen, betrachten wir zunächst den Quellcode, der beim Erzeugen einer neuen React Native Anwendung angelegt wird.

Zunächst muss in jeder Datei `React` importiert werden und in der Regel auch `Component` (Zeile 1). `Components` sind das Kernstück jeder React Native Anwendung. Sie beinhalten eine Beschreibung der GUI in der `render()`-Methode (Zeilen 10 ff), die sich wiederum aus anderen `Components` zusammensetzen kann. In diesem Fall handelt es sich dabei um die im Framework vordefinierten `Components View` (Zeile 12) und `Text` (Zeile 13). Dieser XML-ähnliche Markup nennt sich JSX (JavaScript XML) und erlaubt eine beliebig feine Modularisierung der Benutzeroberfläche, da theoretisch jeder Abschnitt in der `render()`-Funktion in eine eigene `Component` ausgelagert werden könnte. Der `style` der einzelnen `Components` wird entweder am Ende der Datei beschrieben (Zeile 22 ff) oder in eine eigene Datei ausgelagert und importiert. Damit die Anwendung nach

dem Start etwas anzeigt, muss eine `Component` mit `registerComponent()` (Zeile 33) als Einstiegspunkt festgelegt werden.

```
1  import React, { Component } from 'react';
2  import {
3    AppRegistry,
4    StyleSheet,
5    Text,
6    View
7  } from 'react-native';
8
9  export default class TestProjekt extends Component {
10   render() {
11     return (
12       <View style={styles.container}>
13         <Text style={styles.welcome}>
14           Welcome to React Native!
15         </Text>
16         ...
17       </View>
18     );
19   }
20 }
21
22 const styles = StyleSheet.create({
23   container: {
24     flex: 1,
25     justifyContent: 'center',
26     alignItems: 'center',
27     backgroundColor: '#F5FCFF',
28   },
29   welcome: {...},
30   instructions: {...},
31 });
32
33 AppRegistry.registerComponent('TestProjekt', () => TestProjekt);
```

Listing 3.4: `index.android.js` aus der Sample React Native Anwendung

Das Verhalten der GUI wird ebenfalls innerhalb der `Component`-Klassen festgelegt. Dafür müssen, je nach `Component`, verschiedene Callbacks definiert werden, abhängig von den Ereignissen, auf die man man reagieren möchte. Die React Docs [7] beschreiben, wie auf die jeweiligen Ereignisse reagiert werden kann. Beim vordefinierten `TextInput` stehen zum Beispiel `onChangeText`, `onSubmitEditing` und `onFocus` zur Verfügung. Für jedes Ereignis, das verarbeitet werden soll, wird üblicherweise eine eigene Methode geschrieben, beginnend mit einem Unterstrich (z.B. `_updateWelcomeText()`).

Anwendungslogik, die über das Verhalten der GUI hinaus geht, kann beliebig in weitere `.js`-Dateien ausgelagert werden oder alternativ als native Komponente realisiert wer-

den. Das Framework selbst gibt dafür keine Richtlinien vor. Falls Daten lokal auf dem Smartphone gespeichert werden sollen, bietet React Native lediglich den sogenannten `AsynchStorage`[2], einen key-value basierter Speicher, der global innerhalb der Anwendung verfügbar ist. Eine ausführlichere Einführung zur Programmierung mit React Native ist in [8] zu finden, welches teilweise als Grundlage für dieses Kapitel dient.

3.2.3 Native Komponenten

Eines der wichtigsten Alleinstellungsmerkmale gegenüber anderer Frameworks, die auf Web-Frameworks basieren, liegt in der Möglichkeit neben JavaScript auch in den nativen Sprachen programmieren zu können. Das hat mehrere Vorteile bzw. Anwendungsgebiete. Man könnte beispielsweise bereits existierenden Code einer nativen Android- oder iOS-Anwendung wiederverwenden oder mit gerätespezifischen APIs kommunizieren, die (noch) nicht von React unterstützt werden. Ebenfalls denkbar wären komplexe Berechnungen, die von Multithreading profitieren.

Um nativen Code zu verwenden, muss ein `Native Module`[3] geschrieben werden. Jedes dieser `Modules` nutzt das selbe Codegerüst (Listing 3.5). Zunächst müssen einige Komponenten von der `bridge` importiert werden, um die Kommunikation zwischen JavaScript und Java zu ermöglichen. Die Klasse `MyModule` muss anschließend das Interface `ReactContextBaseJavaModule` erweitern. Im Konstruktor muss lediglich `super(reactContext)` aufgerufen werden. Damit man im JavaScript-Code auf dieses `Module` zugreifen kann, muss die `getName()`-Methode überschrieben werden. Ein solches `Module` kann beliebig viele Funktionen zur Verfügung stellen, die jeweils in einer entsprechenden Methode implementiert werden. Wichtig ist, dass jede dieser Methoden mit `@ReactMethod` annotiert ist.

[2] https://facebook.github.io/react-native/docs/asyncstorage.html, 06.11.2017
[3] https://facebook.github.io/react-native/docs/native-modules-android.html, 06.11.2017

```java
package com.depends.mymodule;

import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactMethod;

public class MyModule extends ReactContextBaseJavaModule {
  public MyModule(ReactApplicationContext reactContext) {
    super(reactContext);
  }

  @Override
  public String getName() {...}

  @ReactMethod
  public void myMethod(...) {...}
}
```

Listing 3.5: Native Module Codegerüst

Ein Aufruf durch JavaScript sieht dann wie folgt aus:

```javascript
import { NativeModules } from "react-native";
NativeModules.MyModule.myMethod(...);
```

Listing 3.6: Aufruf eines Native Modules in JavaScript

In iOS funktioniert das Einbinden von nativem Code ähnlich und kann in der Dokumentation[4] nachgelesen werden. Bevor man jedoch ein `Native Module` selbst schreibt, sollte man auf der von Facebook eingerichteten Seite[5] nachschauen, ob ein solches `Module` nicht bereits existiert. Sollte dies nämlich der Fall sein, lassen sich die meisten der dort gelisteten Komponenten sehr einfach mit dem Node Package Manager (npm) zum Projekt hinzufügen. Dazu muss man die Komponente einfach mit `npm install <react-native-module> -save` hinzufügen und mit `react-native link` eine Anpassung der iOS- und Android-Projekte anstoßen, die beim Initialisieren eines React Native Projekts erzeugt wurden.

[4] https://facebook.github.io/react-native/docs/native-modules-ios.html, 06.11.2017
[5] https://js.coach/react-native, 06.11.2017

3.3 Ionic

Das Ionic Framework ist ein Open Source Projekt aus der Kategorie der hybriden Anwendungen, die in einem nativem Container ausgeführt werden. In seiner Historie seit dem ersten Release im Mai 2015 hat sich das Framework mehrfach stark verändert. Die größte Generalüberholung kam mit der Veröffentlichung von Ionic 2 im Jahr 2016, das auf dem ebenfalls stark veränderten Web-Framework Angular 2 aufsetzt. Beide Frameworks haben mittlerweile die Versionsnummern aus dem Namen gestrichen und sind schlicht als Angular und Ionic bekannt. Die grundlegenden Konzepte sind (seit der Version 2), mit der Einführung von TypeScript als neue Programmiersprache, jedoch gleich geblieben.

In diesem Kapitel wird zuerst die Architektur des Frameworks beschrieben. Anschließend wird anhand einiger Codebeispiel erklärt, wie mit Ionic eine Anwendung entwickelt werden kann. Darüber hinaus wird das Command Line Interface vorgestellt, das speziell dafür entwickelt wurde, das Erstellen und Verwalten eines Ionic-Projekts zu vereinfachen. Alle Erklärungen basieren auf der offiziellen Dokumentation [9] und [10].

3.3.1 Architektur

Ionic baut sehr stark auf Angular auf. Das gilt insbesondere für die Benutzeroberfläche und Logik der Anwendung. Ionic fokussiert sich dabei stark auf das *look and feel* einer mobilen Anwendung und bedient sich ansonsten weitestgehend an den Funktionalitäten, die Angular bereitstellt. Eine Ausnahme stellt die Navigation dar, deren Funktionsweise nachfolgend erklärt wird. Um daraus eine mobile Anwendung zu generieren, nutzt Ionic mit Apache Cordova [11] ein Framework, das für jede Zielplattform einen nativen Container bereitstellt, der über sogenannte *Cordova Plugins* auf Gerätefunktionen, wie Kamera oder GPS, zugreifen kann. In Abbildung 3.4 ist die Architektur einer *Cordova* Anwendung grafisch dargestellt. Die eigentliche Ionic Anwendung entspricht der *Web App* aus der Abbildung. Der Zugriff auf *Cordova* APIs wird dabei von Ionic jeweils über ein zum Plugin korrespondierenden Modul abstrahiert. Das bedeutet, soll beispielsweise die Android Version des Geräts angezeigt werden, wird über ein entsprechendes Modul auf

das `Device`-Plugin zugegriffen, welches wiederum über die APIs des Betriebssystems die Geräteinformationen abfragt.

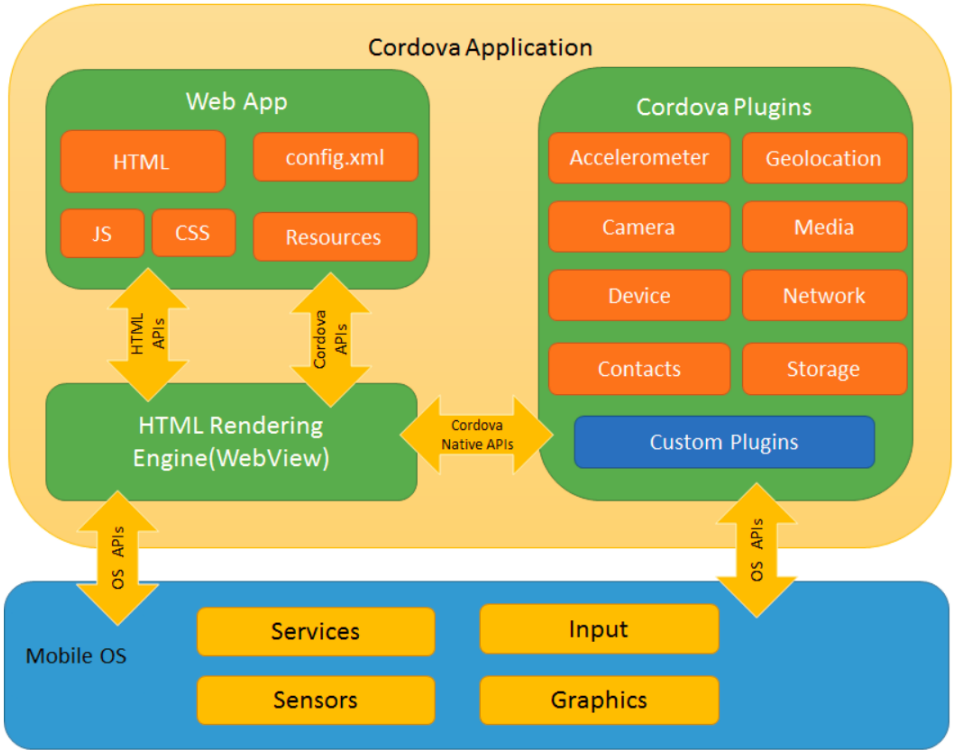


Abbildung 3.4: Architektur einer Cordova Anwendung nach [11]

In Ionic bzw. Angular werden die Module als `ngModules` bezeichnet. In erster Linie dienen sie der Strukturierung einer Anwendung. So können beispielsweise mehrere `Components` in einem Modul zusammengefasst werden. Jede Anwendung muss mindestens ein solches Modul definieren, das als *root-module* fungiert. Darüber hinaus werden Module hauptsächlich verwendet, um externe Bibliotheken in einer Anwendung einzubinden, was zum Beispiel für sämtliche *Cordova Plugins* der Fall ist.

3.3.2 Grundlegende Funktionsweise

Ein guter Startpunkt, um zu verstehen, wie eine Ionic Anwendung funktioniert, ist die Struktur des *src*-Ordners. In Listing 3.7 ist exemplarisch die Struktur des Ionic Tutorials aus [9] abgebildet.

```
src
├── app
│   ├── app.component.ts
│   ├── app.html
│   ├── app.module.ts
│   ├── app.scss
│   └── main.ts
├── assets
│   └── icon
├── index.html
├── manifest.json
├── pages
│   ├── hello-ionic
│   ├── item-details
│   └── list
├── service-worker.js
└── theme
    └── variables.scss
```

Listing 3.7: Ordnerstruktur des Ionic Tutorials aus [9]

Im Ordner `app` befinden sich einige Dateien, die zur Initialisierung der Anwendung benötigt werden. In `app.module.ts` wird das bereits erwähnte *root-module* definiert, das den Rest der Anwendung steuert. Dort wird außerdem die Datei `app.component.ts` als *root-component* festgelegt, welche als Einstiegspunkt in unsere Anwendung fungiert. Das heißt, sie wird immer als erste `Component` geladen und dient typischerweise als eine Art leere Hülle, in die andere `Components` geladen werden. Im `assets`-Ordner befinden sich statische Inhalte wie das Ionic Logo oder andere Bild-Dateien. Mit der `index.html` startet unsere Anwendung und hat deshalb das `ion-app`-Element im body, über den die *root-component* geladen wird. Im `theme`-Ordner befindet sich lediglich die Datei `variables.scss`, die für das *look and feel* der Ionic `components` verantwortlich ist und damit das Aussehen der gesamten Anwendung beeinflusst. Der Großteil des Codes, der vom Entwickler geschrieben wird, landet im `pages`-Ordner, wo für jede Seite ein einzelner Ordner erstellt wird. Für jede Seite wird dann jeweils eine `<Seitenname>.ts`, `<Seitenname>.html` und `<Seitenname>.scss` benötigt, in denen Logik, Markup und Styling der Seite definiert werden.

Diese Struktur ist der von Angular sehr ähnlich. Auch die Seiten bestehen aus den gleichen Dateien wie bei einer Webanwendung mit Angular. Interessant wird es bei der Navigation, die über den Ionic-eigenen `NavController` gesteuert wird. Dabei funktioniert die Navigation als simpler Stapel. Die oberste Seite auf dem Stapel ist die, die aktuell in der Anwendung angezeigt wird. Nun kann eine neue Seite mit `NavController.push(<nächsteSeite>)` auf den Stapel gelegt werden, um zu dieser zu navigieren. Möchte man stattdessen eine Seite zurück, reicht ein Aufruf von `NavController.pop()`. Die Verwendung des `ion-navbar`-Elements vereinfacht die Navigation noch weiter, da es automatisch für Seiten, die nicht ganz unten im Stapel liegen, einen *Back-Button* erstellt.

Im Gegensatz zu Angular wird in Ionic eine Möglichkeit benötigt lokal auf dem Gerät Daten abzuspeichern. Zu diesem Zweck steht in Ionic das Modul `native-storage` zur Verfügung. Bevor das Modul jedoch genutzt werden kann, muss das zugehörige Cordova-Plugin mit `ionic cordova plugin add cordova-plugin-nativestorage` hinzugefügt und mit `npm install -save @ionic-native/native-storage` installiert werden. Damit können dann *Key-Value*-Paare gespeichert werden.

```
1  import { NativeStorage } from '@ionic-native/native-storage';
2
3  constructor(private nativeStorage: NativeStorage) { }
4
5  this.nativeStorage.setItem('myitem', {property: 'value', anotherProperty: 'anotherValue'})
6    .then(
7      () => console.log('Stored item!'),
8      error => console.error('Error storing item', error)
9    );
10
11 this.nativeStorage.getItem('myitem')
12   .then(
13     data => console.log(data),
14     error => console.error(error)
15   );
```

Listing 3.8: Persistenes Speicher von Daten in Ionic[6]

Im Listing 3.8 wird ein Datensatz gespeichert und wieder geladen. Dafür muss das Modul importiert (Zeile 1) und ein `nativeStorage`-Objekte über den Konstruktor verfügbar gemacht werden (Zeile 5). Anschließend können die Methoden `getItem()` und `setItem()` zum Speichern und Laden von Datensätzen verwendet werden. Beim Rück-

[6] https://ionicframework.com/docs/native/native-storage/, 28.11.2017

gabewert handelt es sich um ein *Promise*, da die Zugriffe auf den Speicher asynchron ablaufen.

3.3.3 Command Line Interface

Das Command Line Interface (CLI) von Ionic ist ein sehr mächtiges Werkzeug, das in erster Linie dazu dient, Projekte zu erstellen und zu verwalten. Außerdem wird die Anwendung damit im Browser getestet, auf ein Smartphone deployed oder für den App-/Playstore ohne zusätzliche Debugging-Möglichkeiten gebaut. Das Ionic CLI beinhaltet alle Funktionen von Cordova, da es im Grunde genommen nur eine Erweiterung des Cordova CLIs ist. Im Folgenden werden die wichtigsten Funktionen erklärt.

Um eine neue Anwendung anzulegen, existiert der Befehl `ionic start <Projekt-name> <Templatename>`. Damit wird die gesamte Projektstruktur erstellt, sodass die Anwendung direkt im Browser gestartet werden kann. Dies wird über den Befehl `ionic serve` ausgeführt. Das ist die bequemste Möglichkeit, um vor allem in frühen Entwicklungsphasen eine Anwendung zu testen. Wie diese Anwendung aussieht, hängt zu Beginn vom verwendeten *Template* ab. Fünf solcher *Templates* stehen dafür zur Auswahl.

- **tabs:** Damit wird ein simples 3 Tab-Layout erstellt.
- **sidemenu:** Hier erhält man das typische Browsermenü, das man mit einer swipe-Bewegung von links ins Bild ziehen kann.
- **blank:** Wie der Bezeichner vermuten lässt, wird mit diesem *Template* lediglich ein weißer Bildschirm ohne Inhalt angezeigt.
- **super:** Mit diesem *Template* erhält man eine Anwendung mit 14 vorgefertigten Seiten, von denen man die gewünschten verwenden/modifizieren und den Rest löschen kann. Der große Vorteil bei diesem *Template* ist, dass die vorgefertigten Seiten sämtliche *best-practices* der Entwicklung mit Ionic umsetzen und so neuen Entwickler eine gute Orientierung bieten.
- **tutorial:** Hiermit wird die aus dem vorhergehenden Kapitel dargestellte Projektstruktur des offiziellen Tutorials erzeugt.

Natürlich muss früher oder später auch auf einem Endgerät getestet werden. Im Fall von Android geht das sehr simpel mit dem Befehl `ionic cordova run android –device`. Bei iOS muss dagegen vorher der Code signiert werden[7]. Dieser Schritt wird in Android nur dann notwendig, wenn die Anwendung in den Playstore gestellt werden soll. Um die Anwendung für den App-/Playstore bereitzustellen wird der Befehl `ionic cordova build android/ios –prod –release` verwendet.

Neben dem Erzeugen und Starten des Projekts bzw. der Andwendung ist eine der wichtigsten Funktionen des CLIs das Hinzufügen von Cordova Plugins, da diese für die Nutzung von Gerätefunktionen erforderlich sind. Der Befehl dafür lautet `ionic cordova plugin add <pluginname>`. Analog dazu kann es mit `ionic cordova plugin rm <pluginname>` bei Bedarf auch wieder entfernt werden. Eine vollständige Liste der Plugins inklusive Beschreibung und Beispielen kann in der offiziellen Dokumentation [9] unter *Ionic Native* gefunden werden.

Darüber hinaus kann das Ionic CLI auch einige Codegrundgerüste erstellen. Der meistgenutzte Befehl in dieser Hinsicht ist `ionic generate page <Seitenname>` zum erzeugen neuer Seiten. Analog können mit `ionic generate` noch viele andere Vorlagen[8], unter anderem komplette `components`, erzeugt werden.

3.4 Weitere Frameworks und Ausblick

MD^2, React Native und Ionic stellen nur eine begrenzte Auswahl der derzeit existierenden Cross-Plattform Frameworks für mobile Anwendungen dar. In diesem Kapitel wird daher auf zwei weitere aktuell populäre Frameworks eingegangen, die es aus verschiedenen Gründen in dieser Arbeit nicht näher betrachtet werden.

Eines dieser Frameworks ist *Framework7*[9], das ursprünglich Teil der in dieser Arbeit zu betrachtenden Frameworks sein sollte. Allerdings bietet das Framework gegenüber Ionic wenig Mehrwert. Es nutzt ebenfalls Webtechnologien und die entstehende Anwendung

[7] https://ionicframework.com/docs/intro/deploying/, 01.12.2017
[8] https://ionicframework.com/docs/cli/generate/, 01.12.2017
[9] https://framework7.io/, 10.12.2017

wird von Cordova gekapselt, um sie auf mobilen Endgeräten installieren zu können. Allerdings integriert Framework7 Cordova nicht, sondern ist lediglich ein HTML-Framework, das versucht, das native *look and feel* mit Webtechnologien nachzuempfinden. Möchte man folglich die Anwendung für den App-/Playstore ausliefern, muss man sich zusätzlich mit Cordova vertraut machen. Vorteile gegenüber Ionic bietet Framework7 in erster Linie in der Flexibilität, da es als HTML-Framework beliebig mit anderen Frameworks kombiniert werden kann. Es könnte insbesondere für Webentwickler interessant sein, die eine Alternative zu Angular benötigen oder großen Wert auf ein natives *look and feel* legen. Unterm Strich war es kein geeigneter Kandidat für die Auswertung, da es mit weiteren Technologien kombiniert werden muss und in Bezug auf Ionic redundant gewesen wäre.

Ein wesentlich interessanteres Framework ist *Xamarin*[10]. Damit lassen sich mit einer Codebase Android und iOS Anwendungen entwickeln. Xamarin kommt als Erweiterung zur Visual Studio IDE und nutzt die Programmiersprache C#, die anschließend in die verschiedenen nativen Sprachen (z.B. Java und Objective-C) übersetzt wird. Damit fallen Xamarin Anwendungen unter die Kategorie der Cross-Compiled Anwendungen. Das Framework wird von der gleichnamigen Firma entwickelt, die seit 2016 eine Tochtergesellschaft von Microsoft ist. In der Grundversion mit *Visual Studio Community* ist das Framework kostenlos, solange das Entwicklerteam aus 5 (oder weniger) Personen besteht. Bei Open Source Projekten und im Bereich der Lehre gilt diese Einschränkung nicht[11]. Für größere Unternehmen existieren die beiden kostenpflichtigen Versionen *Visual Studio Professional* und *Visual Studio Enterprise*. Bei beiden Versionen sind Zusatzangebote, wie der technische Support, mit inbegriffen. Darüber hinaus bietet Xamarin ein kostenpflichtiges online Training unter dem Namen *Xamarin University* an. Der Umfang dieses Frameworks ist deutlich größer als der von React Native, Ionic, und MD^2, da auch für Windows und MAC OS entwickelt werden kann. Da zudem MD^2 die Kategorie der Cross-Compiled Anwendungen bereits abdeckt und es den Umfang dieser Arbeit sprengen würde, wurde auf die detaillierte Betrachtung und Bewertung des Frameworks verzichtet.

[10] https://www.xamarin.com/, 10.12.2017
[11] https://www.xamarin.com/license, 11.12.2017

Natürlich gibt es noch ein Vielfaches mehr an Frameworks, vor allem im Bereich der hybriden Anwendungen. Es ist schwierig eine Prognose zu treffen, ob sich eine der Technologien gegen den Rest durchsetzen wird. Allerdings könnte das neue Betriebssystem von Google namens *Fuchsia*[12] eine entscheidende Rolle spielen. Zum Zeitpunkt, als diese Arbeit verfasst wurde, waren noch nicht viele Informationen über das neue Betriebssystem bekannt. Google hat insbesondere noch nicht bekannt gegeben, wofür Fuchsia entwickelt wird. Eine Möglichkeit wäre, dass es Android auf dem Smartphonemarkt ablösen soll. Das hätte einige Vorteile, da Google Fuchsia von Grund auf neu entwickelt, anstatt wie Android auf Linux aufzubauen. Im Moment ist Googles eigene Programmiersprache *Dart*[13] die Hauptprogrammiersprache, mit der Anwendungen für Fuchsia programmiert werden sollen. Das ist deshalb interessant, weil Google mit *Flutter*[14] bereits ein Cross-Plattform Framework vorgestellt hat, das Dart verwendet. Falls Google tatsächlich plant Android mit Fuchsia abzulösen und damit Androidentwickler *zwingt* auf Dart umzusteigen, könnte sich Flutter oder ein anderes Framework, das in Zukunft mit Dart arbeitet, bei der Cross-Plattform Entwicklung durchsetzen. Leider bleibt das bis zu einer offiziellen Bestätigung reine Spekulation, nicht zuletzt weil Google schon mehrfach große Projekte wieder eingestampft hat.

[12] https://github.com/fuchsia-mirror, 11.12.2017
[13] https://www.dartlang.org/, 11.12.2017
[14] https://flutter.io/, 11.12.2017

4

Evaluation

In diesem Kapitel folgt die Evaluation der Frameworks auf Basis einer Auswahl der in [12] und [13] vorgeschlagenen Kriterien. Zunächst werden diese Kriterien vorgestellt, um die Vor- und Nachteile der einzelnen Frameworks aufzuzeigen. Darauf folgt ein textueller Vergleich, bei dem die Frameworks anhand der zuvor definierten Kriterien gegenübergestellt werden. Abschließend wird mit dem sogenannten Analytischen Hierarchieprozess (AHP) nach [14] bewertet, welches Framework sich in einem vorher definierten Anwendungsfall am besten eignet.

4.1 Kriterien

Im Folgenden werden die Kriterien der Evaluation vorgestellt. Dabei wird auch auf Kriterien eingegangen, die aus verschiedenen Gründen nicht in die Bewertung mit einfließen. Harte Kriterien werden in der abschließenden Bewertung berücksichtigt, weiche Kriterien werden nur in der Gegenüberstellung in Kapitel 4.2 betrachtet.

Ein Kriterium, das nicht Teil der Evaluation sein wird, sind die unterstützten Zielplattformen. Hintergrund für diese Entscheidung ist eine Pressemitteilung des Marktforschungsunternehmens Gartner im August 2017[1], laut der der Marktanteil von Android (87.7%) und iOS (12.1%) zusammen bei nahezu 100% liegt. Dementsprechend sind die übrigen Zielplattformen für die meisten Entwickler mobiler Anwendungen irrelevant und somit ist es auch für diese Bewertung uninteressant, da alle behandelten Frameworks mindestens Android und iOS unterstützen.

[1] https://www.gartner.com/newsroom/id/3788963, 20.11.2017

Skalierbarkeit ist ein weiteres Kriterium, das nicht Teil dieser Auswertung ist, da es im Rahmen dieser Arbeit nicht möglich war, jedes der vorgestellten Frameworks auf Skalierbarkeit zu prüfen. Es können zwar Annahmen getroffen werden, wie gut ein Framework für größere Entwicklerteams und Projekte geeignet ist. Diese begründen sich jedoch im Wesentlichen auf die Modularisierung des Quellcodes, welche mit dem Kriterium der Wartbarkeit bereits diskutiert wird.

4.1.1 Harte Kriterien

KH 1: Lizenz und Kosten: Neben der Lizenz und den damit verbunden Kosten werden mit diesem Kriterium kostenpflichtige Zusatzangebote der jeweiligen Frameworks gegenübergestellt. Das könnten beispielsweise Cloud-Dienste sein, die eine Anwendung auf verschiedenen emulierten Konfigurationen automatisch testen oder Support durch die Entwickler des Frameworks.

KH 2: Unterstützung plattformspezifischer Funktionen: Je nach Framework kann auf unterschiedliche Gerätefunktionen zugegriffen werden. Dabei ist es zum einen interessant, was bereits unterstützt wird, zum anderen wird bewertet wie schwierig es ist, eine Anwendung um Funktionen zu erweitern, die nicht standardmäßig vom Framework bereitgestellt werden. Neben den Gerätefunktionen wie GPS oder Kamera ist hier auch von Interesse, ob es möglich ist, nativen Code einzubinden bzw. wie umständlich sich diese Einbindung gestaltet.

KH 3: Entwicklungsunterstützung: Bei diesem Kriterium wird verglichen, was das Framework bietet, um dem Entwickler die Arbeit zu erleichtern. Darunter fällt die Unterstützung von Entwicklungsumgebungen, die oftmals durch Plugins erweitert werden, um Features, wie eine automatische Vervollständigung, anzubieten, aber auch jede Art von Unterstützung bei der Fehlersuche. Das können zum Beispiel sinnvolle Fehlermeldungen mit Lösungsvorschlag sein oder Browsererweiterungen für die hybride Anwendungsentwicklung, die das Debuggen vereinfachen.

KH 4: Benutzeroberfläche: In vielen Frameworks steht das Design der GUI im Vordergrund. Daher bietet sich die Art, wie GUIs erstellt werden, sowie das *look and feel*

als möglicher Vergleichspunkt an. Dabei ist auch die Performance sehr wichtig, da der Nutzer die Anwendung nicht bedienen kann, solange die Benutzeroberfläche (nach)lädt. Generell gilt hier: Je näher man in den genannten Punkten einer nativen Anwendung kommt, desto besser. Das heißt nicht unbedingt, dass es identisch zu nativen Elementen aussehen soll, sondern dass Reaktionszeit und Benutzbarkeit auf einem ähnlichen Niveau liegen.

KH 5: Einstiegsschwierigkeiten: Die größte Einstiegsschwierigkeit ist in der Regel die speziellen Technologien eines Frameworks zu erlernen. Dementsprechend wird damit bewertet, wie schwierig es ist, sich in das jeweilige Framework einzuarbeiten. Da sich das natürlich einerseits stark vom Entwickler, als auch von der Technologie unterscheiden kann, wird auch in Betracht gezogen, ob die Technologien in anderen Kontexten bereits verwendet werden. Webtechnologien sind beispielsweise sehr weit verbreitet und ein Framework, welches diese explizit nutzt, ist folglich für viele Entwickler einfacher zu erlernen.

KH 6: Wartbarkeit: Bei einigen Frameworks lässt sich der Quellcode einer Anwendung besser strukturieren als bei anderen. Manche Frameworks halten in ihrer Dokumentation sogar *Best-Practices* fest, denen Entwickler folgen sollten. Daraus ergibt sich logischerweise eine höhere Wartbarkeit mit zunehmender Größe eines Projekts.

4.1.2 Weiche Kriterien

KW 1: Performance: Die Performance von mobilen Anwendungen, welche mit Cross-Plattform Frameworks entwickelt wurden, ist mittlerweile sehr nah an der nativer Anwendungen. Das heißt, der Nutzer sollte bei der Reaktionszeit der Anwendung in der Regel keinen Unterschied feststellen können. Mit diesem Kriterium sollen mögliche Randfälle diskutiert werden, in denen die Frameworks Probleme mit der Performance verursachen können, beispielsweise beim Start der Anwendung. Da sich die Performance nur in besonderen Fällen signifikant unterscheidet, eignet sie sich nicht als hartes Kriterium.

KW 2: Langlebigkeit: Ein wichtiges Kriterium insbesondere für kleinere Firmen stellt die Langlebigkeit eines Frameworks dar, da ein Umstieg auf ein anderes Framework mit mehr oder weniger hohen Kosten verbunden ist. Gute Indikatoren für eine hohe Langlebigkeit sind in erster Linie regelmäßige Updates, eine große und aktive Community und Beiträge zur (Weiter-)Entwicklung des Frameworks durch große Firmen, wie Google oder Facebook. Selbst mit all diesen Indikatoren kann ein vermeintlich langlebiges Framework schnell irrelevant werden, wenn sich beispielsweise ein kommerzieller Unterstützer plötzlich aus der Entwicklung zurückzieht. Aus diesem Grund wird die Langlebigkeit lediglich als weiches Kriterium aufgeführt.

4.2 Vergleich der Frameworks

In diesem Kapitel werden die Frameworks anhand der definierten Kriterien verglichen. Für die harten Kriterien ergibt sich aus dem Vergleich zudem ein Ranking der drei vorgestellten Frameworks, welches im nächsten Kapitel für die Auswertung mit AHP benötigt wird. Allerdings sind diese Rankings nicht endgültig, da je nach Anwendungsfall andere Aspekte eines Kriteriums eine zentrale Rolle spielen können. Daher kann sich die Bewertung der Frameworks in einem gegebenen Fall noch etwas verschieben. Diese Rankings werden zudem in Tabelle 4.1 zusammengefasst.

4.2.1 Lizenz und Kosten

Alle behandelten Frameworks sind Open-Source-Projekte unter entsprechenden Lizenzen, die eine kommerzielle Nutzung, Vervielfältigung und Modifizierung erlauben. Bei React Native handelt es sich um die BSD 3-clause License[2], MD[2] ist unter der Apache License 2.0[3] und Ionic unter der MIT License[4] lizenziert. Die Nutzung der Frameworks ist demnach in allen Fällen kostenlos und uneingeschränkt möglich. Unterschiede gibt es

[2] https://github.com/facebook/react-native/blob/master/LICENSE, 04.12.2017
[3] https://github.com/wwu-pi/md2-framework/blob/master/LICENSE, 04.12.2017
[4] https://opensource.org/licenses/MIT, 04.12.2017

Tabelle 4.1: Framework Ranking

Kriterien	**Ranking**
Lizenz und Kosten	1. Ionic
	2. React Native, MD^2
Plattformspezifische Features	1. React Native
	2. Ionic
	3. MD^2
Entwicklungsunterstützung	1. Ionic
	2. React Native
	3. MD^2
Benutzeroberfläche	1. React Native
	2. Ionic
	3. MD^2
Einstiegsschwierigkeiten	1. MD^2
	2. React Native
	3. Ionic
Wartbarkeit	1. Ionic
	2. MD^2
	3. React Native

vor allem bei den kostenpflichtigen Zusatzangeboten. Sowohl MD^2 als auch React Native bieten keinerlei Zusatzangebote. Bei Ionic kann hingegen ein Pro-Account erstellt werden, um eine Reihe kostenpflichtiger Software und Leistungen in Anspruch zu nehmen. Diese umfassen den *Ionic Creator*, mit dem per Drag and Drop sehr schnell Prototypen von Anwendungen erstellt werden können und eine Reihe von Cloud-Diensten, die den Deployment-Prozess (vom Quellcode bis zum App-/Playstore) automatisieren. In diesem Punkt kann Ionic am meisten überzeugen, während sich MD^2 und React Native den zweiten Platz teilen.

4.2.2 Unterstützung plattformspezifischer Funktionen

In MD^2 werden die gängigen Gerätefunktionen wie GPS und Kamera unterstützt, neuere Funktionen wie Near Field Communication (NFC) jedoch nicht. Das Framework lässt sich zudem nicht ohne Weiteres (d.h. nicht ohne am Framework selbst Änderungen vorzunehmen) um eine solche Funktion erweitern. Ionic und React Native unterstützen jeweils wesentlich mehr Funktionen und sind einfacher um neue Funktionen erweiterbar. Bei Ionic muss für jede neue Gerätefunktion ein Cordova Plugin entwickelt werden, wohingegen bei React Native lediglich der native Code eingebunden wird. Dementsprechend schneidet React Native aufgrund der einfachen Erweiterbarkeit hier am besten ab, gefolgt von Ionic und schlussendlich MD^2.

4.2.3 Entwicklungsunterstützung

Entwicklungsumgebungen (IDEs) speziell für die Frameworks gibt es leider nicht oder nur mit eingeschränktem Funktionsumfang. MD^2 wird über ein Plugin für Eclipse entwickelt, für Ionic und React Native existieren diverse Plugins für WebStorm. Für React Native gibt es zwar mittlerweile eine spezielle IDE, die aber nur auf MAC OS läuft und auch nur die Entwicklung für mobile iOS-Anwendungen unterstützt. Leider bietet keine dieser IDEs, egal ob mit oder ohne Plugins, die gleichen umfangreichen Möglichkeiten wie Android Studio oder Xcode für native Anwendungen. Syntaxkorrektur und Autovervollständigung werden unterstützt, für Debugging bzw. Tests werden jedoch zusätzliche Werkzeuge benötigt. Ionic hat durch die Verwendung von TypeScript als primäre Programmiersprache einen Vorteil gegenüber React Native, da TypeScript kompiliert werden muss. Dadurch kann der Code besser durch IDEs überprüft werden, ohne an Mächtigkeit gegenüber JavaScript zu verlieren, da jeder JavaScript-Code auch gültiger TypeScript-Code ist.

Zum Debuggen stehen für Ionic und React Native einige Browsererweiterungen zur Verfügung, wobei natürlich auch simple Ausgaben über die Konsole des Browsers mit `console.log()` eine Möglichkeit der Fehlersuche darstellen. In React Native gibt es darüber hinaus einen roten Errorscreen, wenn es während der Ausführung zu einem Fehler kommt, der oftmals sogar einen Vorschlag liefert, wie der Fehler zu lösen ist. Für

MD^2 gibt es keine solchen Werkzeuge, da automatisch korrekter nativer Code aus der domänenspezifischen Sprache erzeugt wird. Sollte es trotzdem zu Fehlern kommen, muss der native Code mit Hilfe von Android Studio oder Xcode debugged werden.

Unterm Strich werden die Entwickler von Ionic-Anwendungen am besten unterstützt, da die Browsererweiterungen und Plugins für WebStorm sehr ausgereift sind, dicht gefolgt von React Native. MD^2 bietet in diesem Bereich am wenigsten an.

4.2.4 Benutzeroberfläche

Die Benutzeroberflächen können mit MD^2 nicht besonders vielfältig gestaltet werden, da in der Zieldomäne des Frameworks hauptsächlich Formulare abgebildet werden müssen. Diese Einschränkung ist aber nicht unbedingt ein Nachteil, da die Erstellung von Benutzeroberflächen dadurch wesentlich einfacher wird. Ionic und React Native bieten jeweils sehr vielfältige Möglichkeiten Benutzeroberflächen zu gestalten. React Native erzeugt eine komplett native GUI aus JSX (JavaScript XML), wohingegen Ionic seine Oberfläche mit Webtechnologien (HTML und CSS) beschreibt, die dann in einer WebView gerendert werden. Welche der beiden Varianten besser ist, ist zu einem gewissen Grad Geschmackssache. Schöne Benutzeroberflächen lassen sich mit beiden Frameworks erstellen, wobei React Native das *look and feel* einer nativen Anwendung hat, was mit Ionic ohne Weiteres nicht erreicht werden kann. JSX hat außerdem den Vorteil, dass es ein wenig kompakter als HTML-Markup und leichter lesbar ist. Alles in allem hat React Native im Vergleich zu Ionic leicht die Nase vorn. MD^2 fällt aufgrund des geringen Funktionsumfangs etwas aus der Reihe, da es in seiner Domäne genau so gute Benutzeroberflächen erzeugt wie die anderen Frameworks und dafür weniger Entwicklungszeit benötigt, aber in jeder anderen Domäne deutlich abfällt.

4.2.5 Einstiegsschwierigkeiten

MD^2 ist von den drei Frameworks am leichtesten zu erlernen. Das kommt zum einen vom deskriptiven Charakter der zugrundeliegenden Sprache und zum anderen von der begrenzten Mächtigkeit, da sich das Framework auf eine einzige Domäne beschränkt.

Es richtet sich auch explizit an weniger technikaffine Entwickler bzw. wurde mit dieser Vorgabe im Hinterkopf entwickelt. React Native liegt, wie die Bezeichnung vermuten lässt, die JavaScript-Bibliothek React zugrunde. Mit JSX kommt in React Native eine Alternative zum HTML-Markup zum Einsatz. Diese Technologien sind relativ leicht zu erlernen. Die größte Herausforderung besteht darin, die eingesetzten Neuerungen des ECMAScript 6 (ES6) Standards zu verstehen. Bei Ionic verhält es sich ähnlich, da Webentwickler schnell einen Zugang finden sollten. Allerdings ist das Programmiermodell, das in Angular zum Einsatz kommt, einzigartig und braucht daher etwas mehr Einarbeitungszeit als andere Webtechnologien. Aus diesem Grund fällt der Einstieg in Ionic am schwersten. Diese Bewertung ist natürlich hinfällig, falls im Entwicklerteam bereits mit Angular oder React gearbeitet wird und der Umstieg auf Ionic oder React Native damit extrem leicht fällt.

4.2.6 Wartbarkeit

MD^2 profitiert bei der Wartbarkeit wieder von der eingeschränkten Mächtigkeit der Technologie. Zudem werden Model, View und Controller strikt im Quellcode getrennt. Daher sollte die Wartbarkeit auch bei größeren Projekten kein Problem sein. Ionic kann ebenfalls mit Trennung verschiedener Aspekte punkten. Mit den `Services` kann Code jederzeit ausgelagert werden, was bei einer MVC-Architektur für das Model und Teile des Controllers Sinn ergeben würde. Außerdem wird üblicherweise die Benutzeroberfläche in sogenannten *Templates* ausgelagert. Darüber hinaus werden eher deskriptive Elemente des Quellcodes in Annotationen ausgelagert. Dazu bietet die offizielle Dokumentation nicht nur Vorschläge für *Best-Practices*, sondern setzt diese auch beim erstellen von Codeskeletten durch das Ionic *Command Line Interface* direkt um. Ionic ermöglicht durch diesen Aufbau und seine Werkzeuge eine gute Wartbarkeit, solange man sich an die vorgestellten *Best-Practices* hält. In React Native ist dies leider nicht der Fall. Falls eine Trennung von Logik und Benutzeroberfläche gewünscht ist, muss man diese selbst vornehmen. Die Benutzeroberfläche kann zwar beliebig modular aufgebaut werden, der Entwickler wird dabei allerdings nicht aktiv unterstützt und besitzt viele Freiheiten bei der Strukturierung der Oberfläche bzw. des darunterliegenden Quellcodes.

Dementsprechend muss ein Entwicklerteam, das eine hohe Wartbarkeit gewährleisten möchte, bereits zu Beginn des Projekts möglichst genau festlegen, wie der Quellcode strukturiert werden soll. Die beiden anderen Frameworks lösen das wesentlich besser, wobei Ionic das beste Gesamtpaket aus Struktur des Quellcodes, *Best-Practices* und seinem *Command Line Interface* bietet.

4.2.7 Performance

Eine endgültige Aussage über das Abschneiden der Frameworks bezüglich der Performance zu treffen ist sehr schwierig und bedürfte umfangreicher Tests mit verschiedenen Anwendungen. Insofern wird hier nicht die Performance verglichen, sondern auf mögliche Probleme in Bezug auf die Performance aufmerksam gemacht. Da es sich bei der generierten Anwendung durch MD^2 um eine vollständig native Anwendung handelt, sollte kein spürbarer Unterschied zu klassischen nativen Anwendungen bestehen. Ionic hat dagegen mit den Limitierungen der verwendeten Webtechnologien zu kämpfen. Jede Gerätefunktion muss über ein Plugin von Cordova angesprochen werden und hängt in Sachen Performance stark davon ab, wie gut dieses implementiert ist. Das kommt insbesondere bei weniger populären Plugins zum tragen. Auch bei der Oberfläche ist Ionic von der in Cordova enthaltenen WebView abhängig. Ionic profitiert zwar von einer großen aktiven Community, die immer wieder für Optimierungen sorgt, allerdings sollten die genannten Umstände bei der Entscheidung für ein Framework im Hinterkopf behalten werden. React Native hat im Gegensatz dazu den Vorteil, dass die Oberfläche komplett nativ gerendert wird und grundsätzlich jede Funktion in nativen Code ausgelagert werden kann. Allerdings gibt es auch bei React Native mögliche Stolpersteine im Bereich der Performance. Die größten Geschwindigkeitseinbußen können bei der Kommunikation zwischen nativem Java-/Objective-C-Code und JavaScript-Code entstehen. Das liegt daran, dass gegenseitige Aufrufe jedes Mal serialisiert werden müssen, bevor sie über die `Bridge` geschickt werden können.

4.2.8 Langlebigkeit

Im Bereich der Langlebigkeit kann MD^2 nicht allzu gut abschneiden, da weder eine große Community, noch ein kommerzieller Unterstützer dahinter stehen. Offensichtlich wird dies wenn die GitHub-Repositories der Frameworks als Vergleich hergezogen werden. Zum Zeitpunkt, als diese Arbeit geschrieben wurde, hatten React Native[5] und Ionic[6] jeweils mindestens eine dreistellige Anzahl an Issues und Pull-Requests. Bei MD^2 gibt es dagegen keine Pull-Requests und nur fünf Issues[7], wovon vier aus dem Jahr 2013 stammen. Diese Zahlen überraschen nicht, wenn man bedenkt, dass hinter React Native Facebook steht und Ionic auf Angular aufbaut, das von Google mitentwickelt wird. Insofern kann bei beiden Frameworks davon ausgegangen werden, dass sie mindestens mittelfristig weiterhin mit vielen Updates/Verbesserungen versorgt werden.

4.3 Auswertung mit dem Analytischen Hierarchieprozess

Im folgenden wird der analytische Hierarchieprozess (AHP) vorgestellt und Schritt für Schritt auf die Problemstellung aus Kapitel 1.1 angewandt. Eine detaillierte Erklärung, inklusive der mathematischen Grundlage der Methodik, ist in [14] zu finden. Die Auswertung erfolgt anschließend mit Hilfe einer Webanwendung (BPMSG[8]).

4.3.1 Analytischer Hierarchieprozess

Der Analytische Hierarchieprozess ist eine Methode aus der Entscheidungstheorie und wurde 1980 vom Mathematiker Thomas L. Saaty entwickelt. Ziel dieser Methode ist es, komplexe Entscheidungen zu vereinfachen, indem sie hierarchisch strukturiert werden. Die hierarchische Struktur macht die Entscheidung zudem besser nachvollziehbar und erfordert eine rationale Auseinandersetzung mit dem Problem. In Abbildung 4.1 wird der Ablauf des AHPs vereinfacht dargestellt. Die einzelnen Schritte der mathematischen

[5] https://github.com/facebook/react-native, 05.12.2017
[6] https://github.com/ionic-team/ionic, 05.12.2017
[7] https://github.com/wwu-pi/md2-framework, 05.12.2017
[8] https://bpmsg.com/academic/ahp.php, 27.11.2017

Auswertung wurden weg gelassen, da sie für die Auswertung mit der Webanwendung unerheblich sind.

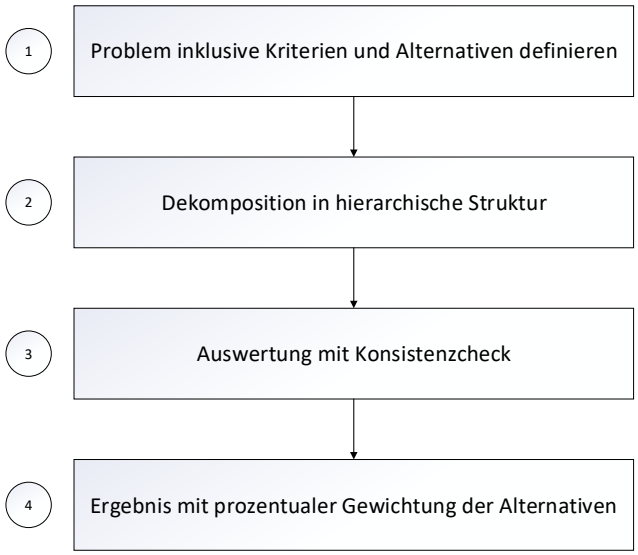


Abbildung 4.1: Vereinfachter Ablauf AHP nach [14] und [15]

Im ersten Schritt erfolgt die Definition der Problemstellung inklusive der Kriterien, nach denen ausgewertet werden soll und der Alternativen. Die Alternativen sind hier die drei vorgestellten Frameworks MD^2, React Native und Ionic. Die Kriterien wurden bereits in einem vorhergehenden Kapitel definiert und die Problemstellung entspricht der Problemstellung dieser Arbeit. Konkret stellt sich für jeden Anwendungsfall die Frage *"Welches Framework soll für den vorliegenden Anwendungsfall verwendet werden?"*.

Anschließend erfolgt die Dekomposition in eine hierarchische Struktur. Das Ergebnis dieses Schrittes ist in Abbildung 4.2 dargestellt. Auf der ersten Ebene befindet sich die Frage, die das Entscheidungsproblem beschreibt. In der mittleren Ebene sind die Kriterien, in die sich die Problemstellung aufspaltet. Die Kriterien können je nach Bedarf beliebig weit verfeinert werden. In der gegebenen Hierarchie könnte man zum Beispiel das Kriterium *Plattformspezifische Features* in einzelne Features (GPS, Kamera, ...) aufgliedern, wenn man nicht vergleichen will, welche Features unterstützt werden,

sondern wie gut die einzelnen Features in den jeweiligen Frameworks funktionieren. In der untersten Ebene finden sich die Alternativen wieder. Um die Abbildung übersichtlich und kompakt zu halten, wurden die Alternativen nicht für jedes einzelne Kriterium aufgeführt. Es muss natürlich trotzdem jede Alternative im Bezug auf jedes Kriterium bewertet werden.

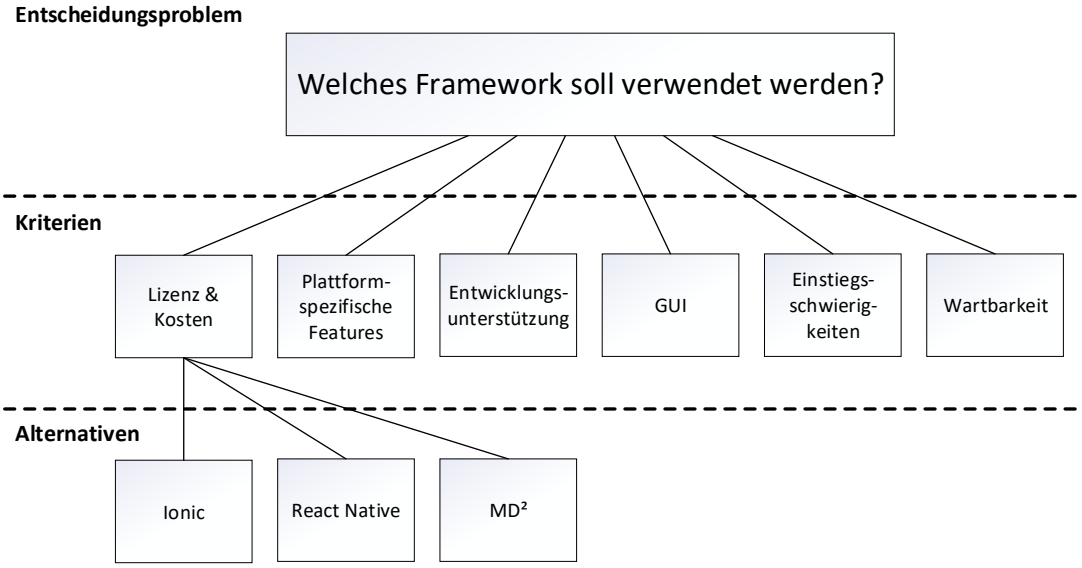


Abbildung 4.2: Entscheidungsbaum nach Schritt 2 des AHP

Sobald die Hierarchie steht, kann mit der Auswertung begonnen werden. Dabei wird auf jeder Ebene bewertet, welche Kriterien relevanter sind bzw. welche Alternative besser ist. Beim gegebenen Entscheidungsbaum bedeutet das, es muss für jeden Anwendungsfall entschieden werden, welche Kriterien wie wichtig sind. Entsprechend benötigt jeder Anwendungsfall ein eigenes Ranking der Kriterien. Es könnte beispielsweise sein, dass eine Anwendung unbedingt die GPS Position benötigt, um zu funktionieren, was zu einer hohen Gewichtung für die *Plattformspezifischen Features* führen würde, wohingegen eine andere Anwendung keines dieser Features nutzen möchte, was eine sehr niedrige Gewichtung zur Folge hätte. Analog würden bei Verfeinerung dieses Kriteriums einzelne Features unterschiedlich gewichtet werden. Auch die Alternativen müssen auf diese

Weise gewichtet werden. Dafür werden die Rankings, die sich aus dem Vergleich in Kapitel 4.2 ergeben haben, herangezogen.

Damit die Gewichtung nachvollziehbar ist, werden alle Kriterien auf einer Ebene paarweise miteinander verglichen und alle Alternativen in Bezug auf ein Kriterium paarweise verglichen. Dabei wird eine Skala von 1 (gleich bedeutend) bis 9 (viel besser/wichtiger) verwendet. Möchte man beispielsweise das Ranking 1. React, 2. Ionic, 3. MD^2 erhalten, muss bewertet werden wie viel besser React im Vergleich zu Ionic und MD^2 ist und wie viel besser Ionic im Vergleich zu MD^2 ist. Aus diesen drei Vergleichen wird dann eine Gewichtung in Prozent errechnet. Diese Gewichtung könnte im oben genannten Beispiel so aussehen: Ionic 30%, React 50%, MD^2 20%. Ordnet man nun die berechneten Prozentzahlen in absteigender Reihenfolge, erhält man das Ranking. Dieses Verfahren hat zwei wesentliche Vorteile. Zum einen spiegelt die Bewertung dadurch kleine und große Unterschiede in den jeweiligen Kategorien wieder, zum anderen können so Inkonsistenzen bei der Entscheidungsfindung aufgedeckt werden. Eine Inkonsistenz wäre zum Beispiel, wenn Ionic und MD^2 als gleich bedeutend (1) bewertet werden, obwohl React viel besser (9) als MD^2 und nur unwesentlich besser (2) als Ionic bewertet wurde. In der Webanwendung wird daher jede paarweise Gewichtung auf Inkonsistenzen geprüft.

Nachdem alle Kriterien und Alternativen gewichtet und auf Inkonsistenzen geprüft wurden, berechnet die Webanwendung das Ergebnis. Dieses Ergebnis ist wiederum eine Gewichtung der Alternativen, allerdings in Bezug auf die Problemstellung.

4.4 Anwendungsfälle mit Auswertung

Im Folgenden werden drei Anwendungsfälle beschrieben, inklusive einer Gewichtung der Kriterien. Danach folgt direkt die Auswertung bzw. das Ergebnis aus der Webanwendung. Um das Ergebnis in Gänze nachzuvollziehen kann der Code aus dem Anhang A in die Webanwendung[9] unter *Input/Edit Hierarchy* eingegeben werden. Darüber hinaus ist dort auch die detaillierte Gewichtung der Alternativen zu finden, die nach Erzeugen der Hierarchie eingegeben werden muss. Dafür muss unter der *Decision Hierarchy* auf

[9] https://bpmsg.com/academic/ahp-hierarchy.php, 28.11.2017

Alternatives geklickt werden und auf der folgenden Seite zunächst oben die Anzahl der Alternativen bestimmt und jede Alternative benannt werden. Anschließend kann für jedes Kriterium mit einem Klick auf *AHP* in der Spalte *Compare* die Gewichtung der Alternativen im Bezug auf das Kriterium der Zeile festgelegt werden. Bei den Kriterien kann ebenfalls im Anhang der ausführliche, paarweise Vergleich eingesehen werden, aus dem die Gewichtung resultiert.

Die Bewertung der Alternativen in Bezug auf die Kriterien, Lizenz und Kosten, Features, Entwicklungsunterstützung und Wartbarkeit ist in Tabelle 4.2 aufgeführt. Bei den Kriterien Benutzeroberfläche und Einstiegsschwierigkeiten variiert die Bewertung von Anwendungsfall zu Anwendungsfall, da im Einzelfall die Vorkenntnisse der Entwickler mit einbezogen werden können. Zudem muss für jeden Fall bewertet werden, wie sehr die eingeschränkten Möglichkeiten von MD^2 beim Erstellen der Benutzeroberfläche ins Gewicht fallen.

Tabelle 4.2: Bewertung der Frameworks nach Kriterien

Kriterium	**Framework**	**Gewichtung**	**Rang**
Lizenz und Kosten	React Native	25.0%	2
	MD^2	25.0%	2
	Ionic	50.0%	1
Plattformspezifische Features	React Native	62.5%	1
	MD^2	13.7%	3
	Ionic	23.8%	2
Entwicklungsunterstützung	React Native	32.0%	2
	MD^2	12.2%	3
	Ionic	55.8%	1
Wartbarkeit	React Native	9.7%	3
	MD^2	33.3%	2
	Ionic	57.0%	1

4.4.1 Portierung einer Webanwendung

Viele der heutigen mobilen Anwendungen sind eine Portierung einer bestehenden Webanwendung oder Webseite. In manchen Fällen wird die mobile Anwendung parallel zur Webanwendung entwickelt. Damit die Webanwendung nicht für zwei weitere Plattformen angepasst werden muss, bietet sich die Verwendung von Cross-Plattform Frameworks an. Das gilt insbesondere für Frameworks, die auf Webtechnologien aufbauen und somit die Anpassung erleichtern.

Für diesen Anwendungsfall (AF 1) ergibt sich, nachdem alle Kriterien paarweise gegeneinander abgewägt wurden, die Gewichtung aus der Tabelle 4.3.

Tabelle 4.3: Gewichtung der Kriterien

Kriterien	**Gewichtung**	**Rang**
Lizenz und Kosten	6.7%	5
Plattformspezifische Features	3.7%	6
Entwicklungsunterstützung	7.1%	4
Benutzeroberfläche	43.6%	1
Einstiegsschwierigkeiten	24.3%	2
Wartbarkeit	14.5%	3

Die Benutzeroberfläche ist hier mit Abstand das wichtigste Kriterium, da es bei einer Portierung wichtig ist, dass sich Nutzer in der mobilen Anwendung möglichst so zurecht finden, wie sie es von der Webanwendung gewohnt sind. Ebenfalls sehr wichtig ist eine leichte Erlernbarkeit der neuen Technologien, da ansonsten für die Portierung entweder neues Personal eingestellt werden muss oder das bestehende Entwicklerteam während der Einarbeitungszeit nicht produktiv arbeiten kann. Mit der Entwicklung einer zusätzlichen mobilen Anwendung muss zudem mehr Code gewartet werden, insbesondere, wenn sie immer auf dem gleichen Stand wie die Webanwendung sein soll. Die Entwicklungsunterstützung spielt eine weniger zentrale Rolle, da komplexe Algorithmen eher auf Serverseite berechnet werden und Code aus der Webanwendung in Frameworks, bei denen Webtechnologien zum Einsatz kommen, wiederverwendet werden

kann. Lizenz und Kosten unterscheiden sich bei den Frameworks nur in dem Zusatzangebot der Clouddienste, die wichtiger wären, wenn nur mobile Anwendungen entwickelt werden. Eine Webanwendung ist nicht auf plattformspezifische Features angewiesen, dementsprechend ist dieses Kriterium auch für die Portierung relativ uninteressant.

Die Bewertung der Frameworks in Bezug auf Einstiegsschwierigkeit und Benutzeroberfläche ist in Tabelle 4.4 aufgeführt. Bei den Einstiegsschwierigkeiten sind alle Frameworks identisch gewichtet, da je nach verwendeter Technologie in der Webanwendung (z.B. React oder Angular) die benötigte Zeit zum Einarbeiten extrem schwankt. Wird React oder Angular in der Webanwendung verwendet, ist React Native bzw. Ionic für die Entwickler sehr leicht zu erlernen. Falls keine der beiden Technologien bereits im Einsatz sind, ist MD^2 am leichtesten zu erlernen. Für die Benutzeroberfläche eignen sich React Native und Ionic gleichermaßen. MD^2 ist dagegen, aufgrund der Fokussierung auf bestimmte GUI-Elemente für Formulardaten, eher ungeeignet.

Tabelle 4.4: Bewertung der Frameworks nach Kriterien

Kriterium	**Framework**	**Gewichtung**	**Rang**
Benutzeroberfläche	React Native	47.2%	1
	MD^2	8.4%	3
	Ionic	44.4%	2
Einstiegsschwierigkeiten	React Native	33.3%	1
	MD^2	33.3%	1
	Ionic	33.3%	1

Anhand der Gewichtung der Kriterien und der Bewertung der Frameworks nach Kriterien liefert die Webanwendung folgendes Ergebnis:

Tabelle 4.5: Ergebnis für AF 1

Framework	**Gewichtung**	**Rang**
React Native	36.4%	2
MD^2	19.6%	3
Ionic	44.0%	1

Hier können sich die beiden Frameworks, die (teilweise) mit Webtechnologien arbeiten, durchsetzen, was aufgrund des Kontextes wenig überraschen sollte.

4.4.2 Geschäftsanwendungen

In großen Unternehmen kommt meist eine Art von ERP-System zum Einsatz. Die zugehörige Software ist meistens sehr komplex, da sämtliche Prozesse innerhalb der Firma abgebildet werden sollen. Das macht die Nutzung für Mitarbeiter oft relativ kompliziert, was insbesondere bei weniger technikaffinen Mitarbeitern zu Problemen führen kann. Aus diesem Grund kann es sinnvoll sein, manche Prozesse (oder Teilprozesse) in einer einfach zu bedienenden mobilen Anwendung abzubilden, sodass die betroffenen Mitarbeiter nur mit der mobilen Anwendung, anstatt der komplexen ERP-Software, interagieren müssen.

Tabelle 4.6 zeigt das Ergebnis der paarweise bewerteten Kriterien für diesen Anwendungsfall (AF 2).

Tabelle 4.6: Gewichtung der Kriterien

Kriterien	**Gewichtung**	**Rang**
Lizenz und Kosten	6.7%	6
Plattformspezifische Features	11.0%	3
Entwicklungsunterstützung	9.8%	4
Benutzeroberfläche	7.1%	5
Einstiegsschwierigkeiten	39.8%	1
Wartbarkeit	25.6%	2

Lizenz und Kosten fallen wieder nur sehr wenig ins Gewicht, da die betroffenen Unternehmen die Clouddienste von Ionic aus Datenschutzgründen wahrscheinlich nicht nutzen wollen. Eine solche mobile Anwendung wird speziell für ein Unternehmen bzw. vom Unternehmen selbst entwickelt. In den App-/Playstore schaffen es diese Anwendungen daher in den meisten Fällen nicht. Die Benutzeroberfläche muss dabei in erster Linie ihren Zweck erfüllen und übersichtlich sein. Eine grafisch ansprechende GUI ist eher

zweitrangig. Bei den plattformspezifische Features sind vor allem GPS und Kamera von Interesse, damit Anwendungen für Außendienstmitarbeiter umgesetzt werden können, die beispielsweise einen Schaden an einer Maschine mit Bildern dokumentieren müssen oder deren Standort während der Arbeitszeit abrufbar sein soll. Eine hohe Wartbarkeit sollte ebenfalls gegeben sein, damit Änderungen, die am ERP-System erfolgen, schnell in die mobile Anwendung übertragen werden können. Anwendungen dieser Art sollen meist schnell entwickelt werden und wenig Ressourcen verbrauchen, da sie oft nur indirekten produktiven Mehrwert für das Unternehmen bieten. Aus diesem Grund sollten die Einstiegsschwierigkeiten möglichst gering sein.

Die Bewertung der Frameworks in Bezug auf Einstiegsschwierigkeit und Benutzeroberfläche ist in Tabelle 4.7 aufgeführt. MD^2 ist von den drei Frameworks ohnehin bereits am einfachsten zu erlernen. Dieser Umstand wird in diesem Anwendungsfall weiter verstärkt, da MD^2 für genau diese Art von mobilen Anwendungen entwickelt wurde. Dementsprechend setzt das Framework sich bei diesem Kriterium deutlich ab. Die Benutzeroberfläche lässt sich mit jedem der Frameworks gleichermaßen gut umsetzen, da das Aussehen eine untergeordnete Rolle spielt.

Tabelle 4.7: Bewertung der Frameworks nach Kriterien

Kriterium	**Framework**	**Gewichtung**	**Rang**
Benutzeroberfläche	React Native	33.3%	1
	MD^2	33.3%	1
	Ionic	33.3%	1
Einstiegsschwierigkeiten	React Native	28.6%	2
	MD^2	57.1%	1
	Ionic	14.3%	3

Gibt man nun alle Bewertungen in die Webanwendung (BPMSG) ein, liefert diese das Ergebnis aus Tabelle 4.8. Keines der Frameworks ist in diesem Fall ungeeignet, aber MD^2 kann sich aufgrund seiner Spezialisierung auf die gegebene Domäne als bestes Framework knapp durchsetzten.

Tabelle 4.8: Ergebnis für AF 2

Framework	Gewichtung	Rang
React Native	27.9%	3
MD^2	38.0%	1
Ionic	34.1%	2

4.4.3 Anwendungen mit Facebook Integration

Ein großer Teil aktueller mobiler Anwendungen integriert Facebook, um beispielsweise den Facebook Login als Autorisierungsmethode zu nutzen. *Facebook for developers*[10] bietet darüber hinaus viele weitere Funktionen im Bereich der Kundenanalyse, Marketing oder Monetarisierung.

Für diesen Anwendungsfall (AF 3) ergibt sich, nachdem alle Kriterien paarweise gegeneinander abgewägt wurden, die Gewichtung aus der Tabelle 4.9.

Tabelle 4.9: Gewichtung der Kriterien

Kriterien	Gewichtung	Rang
Lizenz und Kosten	13.7%	3
Plattformspezifische Features	36.7%	1
Entwicklungsunterstützung	6.3%	6
Benutzeroberfläche	24.8%	2
Einstiegsschwierigkeiten	6.8%	5
Wartbarkeit	11.7%	4

Da es sich bei der Facebook Integration um die Erweiterung einer mobilen Anwendung um zusätzliche Features handelt, wird das Kriterium plattformspezifische Features am stärksten gewichtet. Zum einen soll es mit der Integration von Facebook möglich sein, den eigenen Standort oder Bilder zu teilen, zum anderen spielt dabei auch eine Rolle, wie kompliziert es ist, diese Funktionen der Facebook-Entwicklerprodukte in die

[10] https://developers.facebook.com/products, 04.12.2017

eigenen Anwendung zu integrieren. Die Zielgruppe von Facebook hat zudem einen hohen Anspruch an die Benutzeroberfläche, wodurch dieses Kriterium ebenfalls stärker gewichtet ist. Clouddienste, die Ionic anbietet, könnten hier ebenfalls von Vorteil sein, damit die Anwendung schnell im App- und Playstore landet und möglichst schnell vielen Testern zur Verfügung steht. Dazu trägt auch eine hohe Wartbarkeit positiv bei. Entwicklungsunterstützung und -schwierigkeiten spielen hier eine eher untergeordnete Rolle.

Die Bewertung der Frameworks in Bezug auf Einstiegsschwierigkeit und Benutzeroberfläche ist in Tabelle 4.10 aufgeführt. Das Spektrum der Anwendungen, die von einer Facebook Integration profitieren ist sehr groß. Daher ist es schwer bis unmöglich eine allgemeingültige Aussage über die Entwicklungsschwierigkeiten der einzelnen Frameworks zu treffen, da auch der Anteil der Facebook-Entwicklerprodukte stark schwankt. Bei der Benutzeroberfläche ist MD^2 durch seine Einschränkungen für viele mobile Anwendungen, insbesondere bei medialen Inhalten, eher ungeeignet. React Native und Ionic schenken sich, wie in Kapitel 4.2.4 beschrieben, relativ wenig beim Erstellen der GUI.

Tabelle 4.10: Bewertung der Frameworks nach Kriterien

Kriterium	**Framework**	**Gewichtung**	**Rang**
Benutzeroberfläche	React Native	48.4%	1
	MD^2	9.2%	3
	Ionic	42.3%	2
Einstiegsschwierigkeiten	React Native	33.3%	1
	MD^2	33.3%	1
	Ionic	33.3%	1

Anhand der Gewichtung der Kriterien und der Bewertung der Frameworks nach Kriterien liefert die Webanwendung folgendes Ergebnis:

Tabelle 4.11: Ergebnis für AF 3

Framework	**Gewichtung**	**Rang**
React Native	43.8%	1
MD^2	17.7%	3
Ionic	38.5%	2

MD^2 findet sich hier abgeschlagen auf dem letzten Platz wieder, da es mehr Anwendungsfälle abdecken müsste, um für dieses Szenario geeignet zu sein. React Native kann sich dagegen durchsetzen, weil es insbesondere mit der leichten Erweiterbarkeit um neue Features punkten kann.

5

Verwandte Arbeiten

In diesem Kapitel werden einige verwandte Arbeiten vorgestellt, die sich ebenfalls mit Entwicklungsansätzen beschäftigen, mit denen eine mehrfache Umsetzung für jede Zielplattform entfällt.

[16] untersucht die Möglichkeit, native Anwendungen komplett durch Webanwendungen zu ersetzen, die für mobile Endgeräte optimiert wurden. Eine 6-monatigen Testperiode zeigte, dass die Webanwendungen nicht mit den nativen Anwendungen mithalten konnten, wenn beispielsweise eine exakte GPS-Position bestimmt werden musste. Falls jedoch keine Gerätefunktionen wie Kamera oder GPS genutzt wurden, stellten die Webanwendungen eine brauchbare Alternative zu nativen Anwendungen dar.

[17] ist eine Erhebung der Literatur im Bereich der Cross-Plattform Entwicklung, mit Schwerpunkt auf hybriden und webbasierten Ansätzen zur parallelen Entwicklung für mehrere Zielplattformen. Das Ergebnis der Literaturanalyse ist, dass die mobilen Cross-Plattform-Ansätze zwar noch nicht vollständig ausgereift sind im Vergleich zum nativen Ansatz, aber ein extrem großes Potential haben. Das gilt besonders für die schnelle Entwicklung von Prototypen.

In [13] wird diskutiert, wie sich eine frühe Version von PhoneGap und ähnliche Frameworks zu dieser Zeit im Vergleich mit nativer Entwicklung schlagen. Der Schwerpunkt lag dabei auf Performance im Bereich CPU, Speicherverbrauch und Energieverbrauch. Darüber hinaus wurden einige Kriterien vorgestellt, anhand derer man Ansätze zu Entwicklung mobiler Anwendungen vergleichen kann. [12] verfolgt einen ähnlichen Ansatz wie diese Arbeit. Zunächst werden Kriterien definiert, mit denen anschließend Webanwendungen, Cross-Plattform-Anwendungen und native Anwendungen verglichen werden. Um die Entwicklungsansätze in den verschiedenen Kriterien zu bewerten, wurde

ein Prototyp einer Management-Anwendung mit jedem der Ansätze umgesetzt. Darüber hinaus schlagen die Autoren vor, für die Entscheidung über den verwendeten Ansatz eine *Multi-Criteria*-Methode (wie z.B. der Analytische Hierarchieprozess) zu verwenden. [18] versucht eine Grundlage für zukünftige Analysen der Cross-Plattform-Entwicklung zu schaffen. Dafür werden erneut Kriterien aufgestellt und verschiedene Entwicklungsansätze gegenübergestellt. Anschließend wird der präferierte Ansatz eingesetzt, um eine typische mobile Anwendung (einen RSS feeds client) zu entwickeln.

[19] evaluiert React Native in Bezug auf Performance, *look and feel* und wie viel Code für beide Plattformen (iOS und Android) verwendet werden kann. Dafür wurde eine Anwendung für beide Plattformen entwickelt (jeweils nativ und mit React Native). Das Ergebnis war vielversprechend, da keine für den Nutzer spürbaren Performancedefizite festgestellt werden konnten. Ungefähr 75% des Codes konnten für beide Plattformen verwendet werden.

[20] beschäftigt sich mit dem Problem, Cross-Plattform Frameworks zu testen. Dabei geht es insbesondere darum, ob sich die Anwendung auf allen Plattformen gleich verhält. Dafür haben sich die Autoren auf Cross-Compiling Frameworks fokussiert und ein eigenes Testwerkzeug namens X-Checker entwickelt, das inkonsistentes Verhalten auf den verschiedenen Zielplattformen aufdeckt. X-Checker wurde dann am Xamarin Framework getestet und konnte 47 Bugs aufdecken, die an die Entwickler gemeldet wurden und mittlerweile teilweise gepatched wurden.

In [21] wird das Ergebnis einer Studie präsentiert, die versucht, einen Zusammenhang zwischen Entwicklungsansatz einer mobilen Anwendung und vom Nutzer wahrgenommener Qualität festzustellen. Dazu wurden mehr als 750.000 Nutzerbewertungen von Android und iOS Versionen von 50 verschiedenen Anwendungen in vier Kategorien klassifiziert: Performance, Benutzbarkeit, Sicherheit und Zuverlässigkeit. Dabei wurde festgestellt, dass es bei hybriden Anwendungen wesentlich mehr Nutzerbeschwerden gibt als bei nativen Anwendungen. Speziell bei der Facebook Anwendung wurde nach dem Umstieg von einer hybriden zu einer nativen Anwendung ein Rückgang der Beschwerden über Performance und Zuverlässigkeit festgestellt.

Diese Arbeit unterscheidet sich in zwei wesentlichen Punkten von den genannten Arbeiten. Zum einen wird versucht sämtliche Cross-Plattform Entwicklungsansätze gegenüberzustellen, anstatt nur einen Ansatz mit nativen Anwendungen und/oder Webanwendungen zu vergleichen. Zum anderen erfolgte die Auswertung nicht auf Basis einer Anwendung, die in allen Ansätzen umgesetzt wird, da sich diese Ergebnisse nicht allgemein auf die Frage *"Welches Framework ist für einen gegebenen Anwendungsfall am besten geeignet?"* übertragen lassen. Stattdessen wurde die Perspektive eines Entwicklerteams eingenommen, das vor dieser Frage steht und eine Entscheidung auf Basis der individuellen Gegebenheiten treffen muss. Dafür wurde der Entscheidungsprozess beispielhaft an drei Anwendungsfällen durchgeführt.

6

Zusammenfassung

In dieser Arbeit wurden verschiedene Entwicklungsansätze der mobilen Cross-Plattform Entwicklung vorgestellt. Dazu wurde mit MD^2, React Native und Ionic stellvertretend jeweils ein Framework vorgestellt, das den jeweiligen Ansatz umsetzt. Um die Unterschiede der Frameworks und deren Ansätze deutlich zu machen, wurde die zugrundeliegende Architektur diskutiert und vorgestellt. Anschließend wurden noch einige weitere Frameworks angesprochen und ein Ausblick über mögliche zukünftige Entwicklungen gegeben. Bei der Vorstellung der Frameworks ging es nicht darum, jedes im Detail zu erklären; es sollte viel mehr ein Überblick zu den aktuell verfügbaren Technologien geschaffen werden.

Darauf folgte die Evaluation, in der zum einen die Vor- und Nachteile der Frameworks herausgearbeitet wurden und zum anderen eine Bewertung anhand von drei Anwendungsfällen erfolgte. Dafür wurden zunächst Kriterien definiert, die sowohl für die endgültige Bewertung, als auch für einen textuellen Vergleich herangezogen wurden. Letzterer sollte die Unterschiede bzw. Stärken und Schwächen der Frameworks hervorheben. Für die endgültige Bewertung wurden drei klassische Anwendungsfälle definiert und der Analytische Hierarchieprozess (AHP) angewendet. AHP wurde aus zwei Gründen verwendet. Erstens macht die Methode den Entscheidungsprozess nachvollziehbar und zweitens kann das Vorgehen auf eine zukünftige Entscheidung eins zu eins übertragen werden. Es wurde bewusst darauf verzichtet, eine allgemeingültige Aussage über die Frameworks zu treffen, da jedes der drei Frameworks in einem vorteilhaften Szenario überzeugen konnte.

Die Evaluation hat gezeigt, dass die Entscheidung für ein bestimmtes Framework von sehr vielen Faktoren abhängig ist. An erster Stelle stehen die Anforderungen der zu ent-

wickelnden Anwendung. Kann ein Framework diese nicht adäquat erfüllen, scheidet es logischerweise als Kandidat sofort aus. Ein besonders spannendes Thema ist in diesem Zusammenhang die Erweiterbarkeit der Frameworks, um zukünftige oder (noch) nicht implementierte Funktionen hinzuzufügen. Je einfacher dieses Konzept ist, desto besser oder schneller kann sich ein Framework an neue Anforderungen anpassen. Neben weiteren objektiven Faktoren, wie *Wartbarkeit* oder *Kosten*, spielen auch subjektive Faktoren eine nicht unerhebliche Rolle. Das betrifft insbesondere die *Benutzeroberfläche*, bei der sich der Geschmack der Nutzer und aktuelle Trends (z.B Material-Design) entscheidend auf die Akzeptanz der Anwendungen auswirken können. Die Art und Weise, wie die GUI entwickelt wird, kann zudem für viele Entwickler Grund genug sein, ein Framework nicht zu verwenden. Das gleiche gilt für sämtliche verwendeten Technologien im Framework, allen voran die dominierende Programmiersprache. Es ergibt daher Sinn, Frameworks zu betrachten, die Technologien verwenden, mit denen die Entwickler bereits vertraut sind, nicht zuletzt auch aufgrund der verkürzten Einarbeitungszeit. Zieht man all diese Punkte in Betracht, wird deutlich, wie wichtig es ist, alle an der Entwicklung beteiligten Mitarbeiter in diese Entscheidungsfindung miteinzubeziehen. Um die vielfältigen Meinungen und Anliegen besser zu strukturieren, bietet sich der Analytische Hierarchieprozess an, da der gesamte Entscheidungsprozess nachvollziehbarer wird.

Abschließend kann diese Arbeit niemandem die Entscheidung für oder gegen ein bestimmte Framework abnehmen. Vieles muss im Einzelfall neu bewertet werden und eine allgemeingültige Aussage ist ohnehin nicht möglich. Stattdessen ist es sinnvoll, die hier definierten Kriterien als Grundlage für eine eigene Bewertung heranzuziehen, diese dann zu modifizieren und einen eigenen Entscheidungsbaum aufzubauen.

Literaturverzeichnis

[1] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using vital sensors in mobile healthcare business applications: challenges, examples, lessons learned. In: International Conference on Web Information Systems and Technologies. (2013) 509–518

[2] Heitkötter, H., Majchrzak, T.A., Kuchen, H.: MD-DSL-eine domänenspezifische Sprache zur Beschreibung und Generierung mobiler Anwendungen. In: Software Engineering (Workshops), Citeseer (2013) 91–106

[3] Alphabet Inc.: Angular Tutorial. (`https://angular.io/generated/live-examples/toh-pt6/eplnkr.html`) zuletzt besucht: 7. Nov. 2017.

[4] Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with MD 2. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM (2013) 526–533

[5] Eisenman, B.: Learning React Native: Building Native Mobile Apps with JavaScript. Ö'Reilly Media, Inc."(2015)

[6] Konicek, Martin: Under The Hood of React Native. `http://www.modeling-guidelines.org/` (2015) zuletzt besucht: 26. Okt. 2017.

[7] Facebook Inc.: React Native Docs. (`http://facebook.github.io/react-native/docs/getting-started.html`) zuletzt besucht: 2. Nov. 2017.

[8] Eisenman, B.: Learning React Native: Building Native Mobile Apps with JavaScript. 1st edn. O'Reilly Media, Inc. (2016)

[9] Drifty Co.: Ionic Docs. (`https://ionicframework.com/docs/`) zuletzt besucht: 28. Nov. 2017.

[10] Ravulavaru, A.: Learning Ionic - Second Edition. Packt Publishing (2017)

[11] Apache Software Foundation: Apache Cordova. (`https://cordova.apache.org/`) zuletzt besucht: 7. Nov. 2017.

[12] Heitkötter, H., Hanschke, S., Majchrzak, T.A. In: Evaluating Cross-Platform Development Approaches for Mobile Applications. Springer Berlin Heidelberg, Berlin, Heidelberg (2013) 120–138

[13] Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, IEEE (2013) 323–328

[14] Charouz, J., Ramík, J.: A multicriteria decision making at portfolio management. E+ M Ekonomie a Management (2010) 44

[15] Schaude, J.: Technical Conception and Implementation of an IT-System supporting the flexible Distribution of Documents within a large scale Sales Organization. Master's thesis, University of Ulm (2015)

[16] Jobe, W.: Native Apps Vs. Mobile Web Apps. iJIM **7** (2013) 27–32

[17] Amatya, S., Kurti, A.: Cross-platform mobile development: challenges and opportunities. In: ICT Innovations 2013. Springer (2014) 219–229

[18] Xanthopoulos, S., Xinogalos, S.: A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In: Proceedings of the 6th Balkan Conference in Informatics. BCI '13, New York, NY, USA, ACM (2013) 213–220

[19] Hansson, N., Vidhall, T.: Effects on performance and usability for cross-platform application development using React Native. Master's thesis, Linköping University (2016)

[20] Boushehrinejadmoradi, N., Ganapathy, V., Nagarakatte, S., Iftode, L.: Testing cross-platform mobile app development frameworks (t). In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, IEEE (2015) 441–451

[21] Mercado, I.T., Munaiah, N., Meneely, A.: The Impact of Cross-platform Development Approaches for Mobile Applications from the User's Perspective. In: Proceedings of the International Workshop on App Market Analytics. WAMA 2016, New York, NY, USA, ACM (2016) 43–49

A

AHP-Webanwendung Eingaben

In diesem Anhang sind die Eingaben für das *Input/Edit Hierarchie*-Feld in der AHP-Webanwendung aufgeführt. Darunter folgt eine Reihe von Screenshots und Tabellen die festhalten, wie die Kriterien und Alternativen paarweise bewertet wurden. Die Tabellen sind an die Screenshots der Webanwendung angelehnt, wobei immer Framework A besser oder genauso so gut ist wie Framework B. In der letzten Spalte folgt die Bewertung auf einer Skala von 1 (gleich gut) bis 9 (deutlich besser).

```
1 Welches Framework?:Lizenz und Kosten,Features,
2 Entwicklungsunterstuetzung, Benutzeroberflaeche,
3 Einstiegsschwierigkeiten,Wartbarkeit;
```

Listing A.1: Hierarchie ohne Gewichtung

```
1 Welches Framework: Lizenz und Kosten=0.06743485,
2 Features=0.03706208,
3 Entwicklungsunterstuetzung=0.07134021,
4 Benutzeroberflaeche=0.43607379,
5 Einstiegsschwierigkeiten=0.24328944,
6 Wartbarkeit=0.14479963;
```

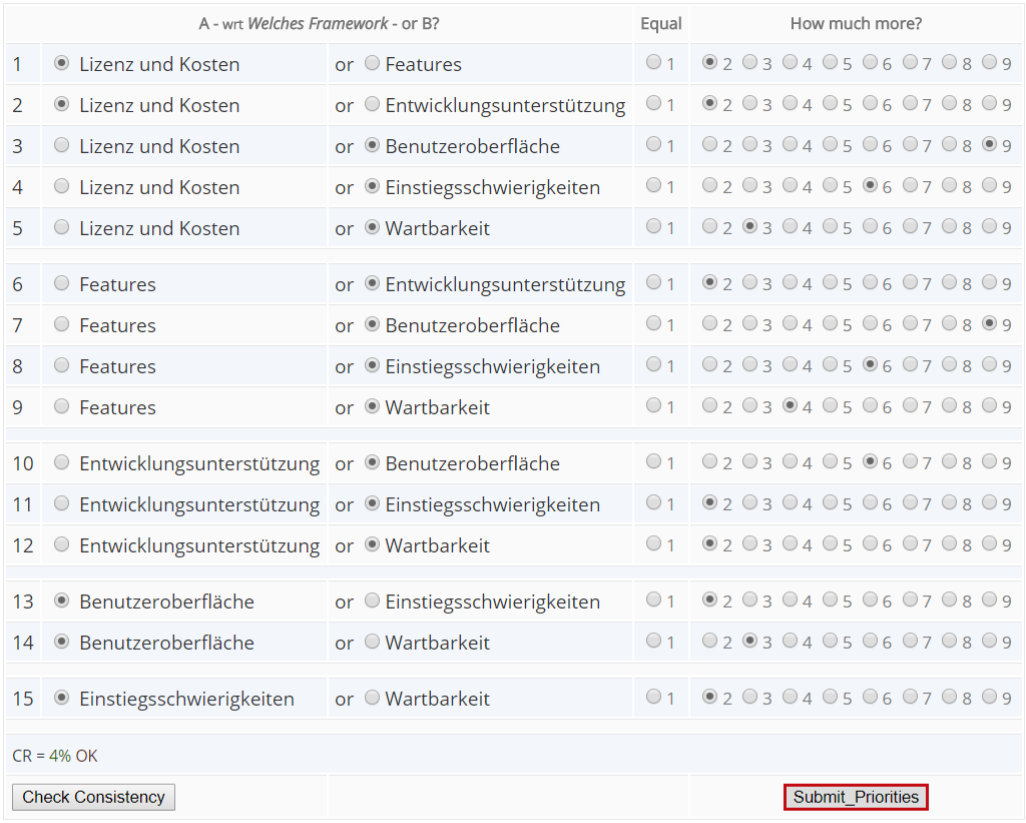
Listing A.2: Portierung einer Webanwendung

```
1 Welches Framework: Lizenz und Kosten=0.06668306,
2 Features=0.10956559,
3 Entwicklungsunterstuetzung=0.09789046,
4 Benutzeroberflaeche=0.07120779,
5 Einstiegsschwierigkeiten=0.39843917,
6 Wartbarkeit=0.25621393;
```

Listing A.3: Geschäftsanwendung

```
1 Welches Framework?: Lizenz und Kosten=0.06624404,
2 Features=0.35204531,
3 Entwicklungsunterstuetzung=0.07445324,
4 Benutzeroberflaeche=0.24845945,
5 Einstiegsschwierigkeiten=0.08404425,
6 Wartbarkeit=0.17475372;
```

Listing A.4: Anwendung mit Facebook Integration



	A - wrt *Welches Framework* - or B?			Equal	How much more?
1	Lizenz und Kosten	or	Features	1	2 3 4 5 6 7 8 9
2	Lizenz und Kosten	or	Entwicklungsunterstützung	1	2 3 4 5 6 7 8 9
3	Lizenz und Kosten	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
4	Lizenz und Kosten	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
5	Lizenz und Kosten	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
6	Features	or	Entwicklungsunterstützung	1	2 3 4 5 6 7 8 9
7	Features	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
8	Features	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
9	Features	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
10	Entwicklungsunterstützung	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
11	Entwicklungsunterstützung	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
12	Entwicklungsunterstützung	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
13	Benutzeroberfläche	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
14	Benutzeroberfläche	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
15	Einstiegsschwierigkeiten	or	Wartbarkeit	1	2 3 4 5 6 7 8 9

CR = 4% OK

Check Consistency Submit_Priorities

Resulting Priorities

	Category	Priority	Rank
1	Lizenz und Kosten	6.7%	5
2	Features	3.7%	6
3	Entwicklungsunterstützung	7.1%	4
4	Benutzeroberfläche	43.6%	1
5	Einstiegsschwierigkeiten	24.3%	2
6	Wartbarkeit	14.5%	3

Abbildung A.1: Paarweise Vergleich für Anwendungsfall Portierung einer Webanwendung

	A - wrt *Welches Framework* - or B?			Equal	How much more?
1	Lizenz und Kosten	or	Features	1	2 3 4 5 6 7 8 9
2	Lizenz und Kosten	or	Entwicklungsunterstützung	1	2 3 4 5 6 7 8 9
3	Lizenz und Kosten	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
4	Lizenz und Kosten	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
5	Lizenz und Kosten	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
6	Features	or	Entwicklungsunterstützung	1	2 3 4 5 6 7 8 9
7	Features	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
8	Features	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
9	Features	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
10	Entwicklungsunterstützung	or	Benutzeroberfläche	1	2 3 4 5 6 7 8 9
11	Entwicklungsunterstützung	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
12	Entwicklungsunterstützung	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
13	Benutzeroberfläche	or	Einstiegsschwierigkeiten	1	2 3 4 5 6 7 8 9
14	Benutzeroberfläche	or	Wartbarkeit	1	2 3 4 5 6 7 8 9
15	Einstiegsschwierigkeiten	or	Wartbarkeit	1	2 3 4 5 6 7 8 9

CR = 1.6% OK

Check Consistency Submit_Priorities

Resulting Priorities

	Category	Priority	Rank
1	Lizenz und Kosten	6.7%	6
2	Features	11.0%	3
3	Entwicklungsunterstützung	9.8%	4
4	Benutzeroberfläche	7.1%	5
5	Einstiegsschwierigkeiten	39.8%	1
6	Wartbarkeit	25.6%	2

Abbildung A.2: Paarweise Vergleich für Anwendungsfall Geschäftsanwendung

	A - wrt *Welches Framework?* - or B?			Equal	How much more?
1	Lizenz und Kosten	or	Features	1	2 3 4 5 **6** 7 8 9
2	Lizenz und Kosten	or	Entwicklungsunterstuetzung	1	2 **3** 4 5 6 7 8 9
3	Lizenz und Kosten	or	Benutzeroberflaeche	1	2 3 **4** 5 6 7 8 9
4	Lizenz und Kosten	or	Einstiegsschwierigkeiten	1	2 **3** 4 5 6 7 8 9
5	Lizenz und Kosten	or	Wartbarkeit	1	**2** 3 4 5 6 7 8 9
6	Features	or	Entwicklungsunterstuetzung	1	2 3 **4** 5 6 7 8 9
7	Features	or	Benutzeroberflaeche	1	**2** 3 4 5 6 7 8 9
8	Features	or	Einstiegsschwierigkeiten	1	2 **3** 4 5 6 7 8 9
9	Features	or	Wartbarkeit	1	**2** 3 4 5 6 7 8 9
10	Entwicklungsunterstuetzung	or	Benutzeroberflaeche	1	2 **3** 4 5 6 7 8 9
11	Entwicklungsunterstuetzung	or	Einstiegsschwierigkeiten	**1**	2 3 4 5 6 7 8 9
12	Entwicklungsunterstuetzung	or	Wartbarkeit	1	**2** 3 4 5 6 7 8 9
13	Benutzeroberflaeche	or	Einstiegsschwierigkeiten	1	2 **3** 4 5 6 7 8 9
14	Benutzeroberflaeche	or	Wartbarkeit	1	**2** 3 4 5 6 7 8 9
15	Einstiegsschwierigkeiten	or	Wartbarkeit	1	**2** 3 4 5 6 7 8 9

CR = 7.6% OK

Check Consistency Submit_Priorities

Resulting Priorities

	Category	Priority	Rank
1	Lizenz und Kosten	13.7%	3
2	Features	36.7%	1
3	Entwicklungsunterstuetzung	6.3%	6
4	Benutzeroberflaeche	24.8%	2
5	Einstiegsschwierigkeiten	6.8%	5
6	Wartbarkeit	11.7%	4

Abbildung A.3: Paarweise Vergleich für Anwendungsfall Facebook Integration

Tabelle A.1: Paarweise Vergleich der Alternativen für Anwendungsfall Portierung einer Webanwendung

Kriterium	**Framework A**	**Framework B**	**Bewertung**
Lizenz und Kosten	React Native	MD^2	1
	Ionic	React Native	2
	Ionic	MD^2	2
Features	React Native	MD^2	4
	React Native	Ionic	3
	Ionic	MD^2	2
Entwicklungsunterstützung	React Native	MD^2	3
	Ionic	React Native	2
	Ionic	MD^2	4
Wartbarkeit	MD^2	React Native	4
	Ionic	React Native	5
	Ionic	MD^2	2
Benutzeroberfläche	React Native	MD^2	6
	React Native	Ionic	1
	Ionic	MD^2	5
Einstiegsschwierigkeiten	React Native	MD^2	1
	React Native	Ionic	1
	MD^2	Ionic	1

Tabelle A.2: Paarweise Vergleich der Alternativen für Anwendungsfall Geschäftsanwendung

Kriterium	**Framework A**	**Framework B**	**Bewertung**
Lizenz und Kosten	React Native	MD^2	1
	Ionic	React Native	2
	Ionic	MD^2	2
Features	React Native	MD^2	4
	React Native	Ionic	3
	Ionic	MD^2	2
Entwicklungsunterstützung	React Native	MD^2	3
	Ionic	React Native	2
	Ionic	MD^2	4
Wartbarkeit	MD^2	React Native	4
	Ionic	React Native	5
	Ionic	MD^2	2
Benutzeroberfläche	React Native	MD^2	1
	React Native	Ionic	1
	Ionic	MD^2	1
Einstiegsschwierigkeiten	MD^2	React Native	2
	React Native	Ionic	2
	MD^2	Ionic	4

Tabelle A.3: Paarweise Vergleich der Alternativen für Anwendungsfall Facebook Integration

Kriterium	**Framework A**	**Framework B**	**Bewertung**
Lizenz und Kosten	React Native	MD^2	1
	Ionic	React Native	2
	Ionic	MD^2	2
Features	React Native	MD^2	4
	React Native	Ionic	3
	Ionic	MD^2	2
Entwicklungsunterstützung	React Native	MD^2	3
	Ionic	React Native	2
	Ionic	MD^2	4
Wartbarkeit	MD^2	React Native	4
	Ionic	React Native	5
	Ionic	MD^2	2
Benutzeroberfläche	React Native	MD^2	6
	React Native	Ionic	1
	Ionic	MD^2	4
Einstiegsschwierigkeiten	MD^2	React Native	1
	React Native	Ionic	1
	MD^2	Ionic	1

Abbildungsverzeichnis

2.1 Verschiedene Arten zur Entwicklung von mobilen Anwendungen nach [1] 6

3.1 Virtual DOM von React nach [5] 24
3.2 Architektur von React Native am Beispiel von Android nach [6] 25
3.3 Abarbeitung eines Touch-Events nach [6] 26
3.4 Architektur einer Cordova Anwendung nach [11] 31

4.1 Vereinfachter Ablauf AHP nach [14] und [15] 49
4.2 Entscheidungsbaum nach Schritt 2 des AHP 50

A.1 Paarweise Vergleich für Anwendungsfall Portierung einer Webanwendung 71
A.2 Paarweise Vergleich für Anwendungsfall Geschäftsanwendung 72
A.3 Paarweise Vergleich für Anwendungsfall Facebook Integration 73

Tabellenverzeichnis

2.1 Datentypen von TypeScript[1] 12

4.1 Framework Ranking 43
4.2 Bewertung der Frameworks nach Kriterien 52
4.3 Gewichtung der Kriterien 53
4.4 Bewertung der Frameworks nach Kriterien 54
4.5 Ergebnis für AF 1 54
4.6 Gewichtung der Kriterien 55
4.7 Bewertung der Frameworks nach Kriterien 56
4.8 Ergebnis für AF 2 57
4.9 Gewichtung der Kriterien 57
4.10 Bewertung der Frameworks nach Kriterien 58
4.11 Ergebnis für AF 3 59

A.1 Paarweise Vergleich der Alternativen für Anwendungsfall Portierung einer Webanwendung 74
A.2 Paarweise Vergleich der Alternativen für Anwendungsfall Geschäftsanwendung 75
A.3 Paarweise Vergleich der Alternativen für Anwendungsfall Facebook Integration 76

Name: Dominik Schwer Matrikelnummer: 802070

Erklärung

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

Ulm, den .....

Dominik Schwer