ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Bachelorarbeit
im Studiengang Medieninformatik

# Evaluation der Fitbit Activity-Tracker im Kontext von Mobile Medical Apps

vorgelegt von

**Moritz Martin**

April 2017

| | |
|---|---|
| 1. Gutachter | Prof. Dr. Manfred Reichert |
| Betreuer: | Marc Schickler |
| Matrikelnummer | 848224 |
| Arbeit vorgelegt am: | 13.04.2017 |

# Kurzfassung

In der Fitness-Branche herrscht nun seit einigen Jahren ein riesiger Aufschwung. Dieser wird unter anderem durch die neuen, technischen Möglichkeiten unterstützt. So ist es heutzutage üblich, statt einer Armbanduhr, einen Fitnesstracker oder auch eine Smartwatch mit sich zu tragen. Diese sind, dank der fortschreitenden Technik, mit allen nur denkbaren Sensoren ausgestattet, sodass man die, über den Tag getätigten Aktivitäten, exakt aufzeichnen kann. Dies dient den Benutzern somit sowohl als Übersicht, als auch zur Motivation, sich mehr zu bewegen. Außerdem können zum Beispiel mit Hilfe der Daten der Herzfrequenz Schlüsse über die Gesundheit des Nutzers gezogen werden.
Aufgrund der stetig wachsenden Nachfrage an Fitnesstrackern, stellen immer mehr Firmen diese Produkte her. Zusätzlich stellen die Hersteller meist eine Anwendung für das Smartphone zur Verfügung, mit welcher die Daten visualisiert und somit überwacht werden können.
Einer der bekanntesten Hersteller auf diesem Gebiet ist **Fitbit**. Diese bieten neben der offiziellen Fitbit-App auch die Möglichkeit an, als Drittanbieter mit einer eigenen Anwendung, an die Daten des Trackers zu gelangen. Dies wird über eine Schnittstelle realisiert, die vom Hersteller zur Verfügung gestellt und verwaltet wird. Über diese können Entwickler eine eigene Anwendung implementieren, mit welcher die Daten abgerufen, aber auch manipuliert werden können.
Die Entwicklung, der Umgang mit dieser API und das gesamte Ökosystem von Fitbit, soll durch diese Arbeit aufgezeigt werden. Hierfür wurde eine Testanwendung für das mobile Betriebssystem iOS entwickelt. Die Erfahrungen aus dieser Entwicklung und dem Umgang mit Fitbits Ökosystem werden daraufhin aufgezeigt und evaluiert. Mit dem Ergebnis der Evaluation wird anschließend die Relevanz der Schnittstelle und dessen Daten hinsichtlich der Anwendungszecke untersucht. Außerdem wurden die von dem Armband gesammelten Daten anhand eines Experimentes mit anderen Trackern verglichen und auf dessen Genauigkeit untersucht.

# Eigenständigkeitserklärung

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

Ulm, den 13.04.2017 Moritz Martin

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Neben der rasanten Entwicklung der Smartphones und Smartwatches etablierte sich in den letzten Jahren noch eine dritte Kategorie auf dem Technikmarkt, die Fitnessarmbänder.
Durch den anhaltenden, weltweiten Fitnesstrend, steigt die Zahl der Nutzer kontinuierlich. Laut einer Prognose werden im Jahr 2017 ca. 44 Millionen Fitnessarmbänder verkauft, dieser Wert lag 2015 noch bei ca. 30 Millionen. [1] Noch stärker steigt dabei der Absatz der Smartwatches, welche ebenfalls für die Überwachung der eigenen Fitness eingesetzt werden. So nutzt in Deutschland schon jeder Dritte Smartphonebesitzer einen Aktivitäts-Tracker. [2]
Diese sind jedoch schon lange nicht mehr nur einfache Schrittzähler, sondern können, je nach Hersteller und Modell, nahezu alle erdenklichen Werte des Körpers messen. Die wohl wichtigste Funktion dabei, ist die Messung der Herzfrequenz, da diese maßgeblich an der Gesundheit beteiligt ist. Viele Hersteller setzen bei ihren Produkten daher auf eine kontinuierliche Aufzeichnung dieser Werte. [8]
Einer der wohl bekanntesten Hersteller, der aus diesem Fitnesstrend hervorgegangen ist, ist **Fitbit**. Neben dem großen Angebot an Fitness-Trackern, welche von einfachen Bändern, ohne Display, bishin zu Smartwatch-ähnlichen Uhren reichen, bietet Fitbit eine Anwendung an, über welche die gesammelten Daten eingesehen werden können.
Zusätzlich ist es hier allerdings möglich, die Daten auch mit einer eigenen Anwendung über eine bereitgestellte Schnittstelle abzufragen. Diese wird von Fitbit gratis zur Verfügung gestellt. Doch ganz ohne Bedingungen ist die Verwendung der Schnittstelle nicht möglich. Daher sollen in der vorliegenden Arbeit, mit der Entwicklung einer Testanwendung diese Bedingungen geprüft und bewertet werden. Das Hauptaugenmerk wird dabei an vielen Stellen auf der Herzfrequenz liegen, da diese, wie schon erwähnt, wohl einer der wichtigsten Werte des Aktivitäts-Tracker darstellt.

## 1.1 Problemstellung

Der Zugriff auf die von der Uhr gesammelten Daten als Drittanbieter ist jedoch komplizierter als es zunächst scheint.

Im Gegensatz zu Fitbis hauseigener Anwendung, herrscht bei einer externen Anwendung keine direkte Datenübertragung zwischen der Uhr und der App. So können an dieser Stelle große Verzögerungen und somit Probleme auftauchen. Außerdem gibt es beim Abfragen der Daten einige Einschränkungen und Schwierigkeiten, welche die Verwendung zum Teil sehr einschränken können. Somit stellt sich die Frage, wie gut die Anwendung trotz dieser Probleme realisiert werden kann.
Wichtig ist dabei vor allem welche Daten der Uhr freigegeben werden, wie gut dessen Zugriff funktioniert und wie groß die Verzögerung im Vergleich zu einer direkten Bluetooth-Verbindung ist.

## 1.2 Zielsetzung und Motivation

Als Ziel der Arbeit gilt es, anhand einer eigenen Anwendung zu überprüfen, wie es sich mit der von Fitbit bereitgestellten Schnittstelle arbeiten lässt. Diese Anwendung wird dabei auf dem Betriebssystem iOS realisiert und soll die wichtigsten Daten des Armbandes abfragen und verarbeiten können. Wie schon erwähnt, gehören zu diesen unter Anderem die Herzfrequenz, aber auch die Schritte und die Daten, welche über den Schlaf gesammelt werden. Die Testanwendung sollte daher mindestens diese Daten visualisieren und gegebenenfalls bearbeiten können.
Zusätzlich sollen die gesammelten Herzfrequenz-Daten mit Hilfe eines Experimentes mit den Daten anderer Produkte verglichen werden.

## 1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist dabei in sieben Kapitel unterteilt.
Dabei werden zunächst in Abschnitt 2 die allgemeinen Grundlagen zum Hersteller Fitbit aufgezeigt. Hierzu zählen sowohl die angebotenen Produkte als auch die Anwendung, welche Fitbit für ihre Aktivitäs-Tracker zur Verfügung stellt. Außerdem werden an dieser Stelle weitere Hersteller mit einem ähnlichem Angebot aufgezeigt und mit Fitbit verglichen.
Im darauffolgenden Kapitel werden schließlich die technischen Grundlagen erläutert. Dabei wird erklärt, welche Entwicklungsumgebung und Werkzeuge verwendet wurden. Diese werden genauer beschrieben, um einen groben Überblick über die spätere Anwendung zu erhalten. Zusätzlich wird der Ablauf der Kommunikation zwischen der Anwendung und dem Armband aufgezeigt.
Kapitel 4 setzt sich mit den nötigen Anforderungen an die Anwendung auseinander. Diese sind dabei in funktionale und nicht-funktionale Anforderungen aufgeteilt.
Schließlich wird die Implementierung der Anwendung beschrieben. Hier wird der Aufbau der einzelnen Seiten und dessen Entwicklung genau erklärt. Dabei werden darüber hinaus einzelne Stellen aus dem Quellcode herausgenommen und kurz beschrieben.

Anschließend wird die Anwendung nun in Kapitel 6 mit den zuvor gestellten Anforderungen verglichen. Dabei wird sowohl bei den funktionalen, als auch bei den nicht-funktionalen Anforderungen untersucht, ob diese erfüllt wurden.
Mithilfe des erlangten Wissens der Implementation, wird in Kapitel 7 das Ökosystem Fitbits untersucht. Mit Bezug auf die Entwicklung werden dabei sowohl die negativen, als auch die positiven Aspekte aufgezeigt. Abgeschlossen wird die Evaluation schlussendlich durch ein Fazit am Ende des Kapitels.
Daraufhin werden die gesammelten Daten in Abschnitt 8 durch ein Experiment untersucht. Hier wird zunächst der Aufbau dargestellt, woraufhin die Durchführung und zum Schluss ein Fazit folgt.
Abschließend wird die Arbeit durch eine kurze Zusammenfassung und einer endgültigen Schlussfolgerung resümiert.

# 2
# Grundlagen

Das folgende Kapitel behandelt die allgemeinen Grundlagen über den Hersteller Fitbit. Dabei wird dessen gesamtes Ökosystem genau aufgezeigt. Außerdem werden anschließend weitere Hersteller genannt, welche als Vergleich dienen sollen und daher nur kurz beschrieben werden. Die wichtigsten Merkmale dieser werden schließlich mit Fitbit verglichen.

## 2.1 Fitbit

Die Firma Fitbit ist ein amerikanisches, noch sehr junges, Unternehmen. Gegründet wurde es im Mai 2007 und ist somit erst durch den wachsenden Fitnessmarkt entstanden. [3]
Wie man aus dem Namen schon schließen kann, befasst sich Fitbit ausschließlich mit dem Thema Fitness. Dabei hat Fitbit ein umfangreiches Ökosystem aus einer Hardware in Form von Aktivitäts-Trackern und der Software, welche für Smartphones aber auch als Webanwendung verfügbar ist, aufgebaut. Die Produkte sind dabei in den verschiedensten Preisklassen erhältlich. Das Ziel dieser ist aber immer das selbe. Sie sollen dem Benutzer dabei helfen die Aktivitäten zu überwachen und durch motivierende Statistiken zu mehr Bewegung animieren. Diese Punkte werden hauptsächlich über die Software realisiert, da diese zum Beispiel durch freischaltbare Erfolge oder Wettkämpfe mit Freunden den Benutzer motivieren kann. [15]

### 2.1.1 Produkte

Wie schon erwähnt, bietet Fitbit eine riesige Auswahl aus verschiedenen Produkten an. Diese beschränken sich dabei jedoch hauptsächlich auf Aktivitäts-Tracker in Form eines Armbandes. Neben diesen Armbändern bietet Fitbit lediglich eine Waage, die mit in das Ökosystem eingebaut werden kann und einen Anhänger, welcher als Schrittzähler und eingeschränkter Aktivitäts-Tracker fungiert, an.
Somit sind die Armbänder der wohl wichtigste Teil in diesem System. Das Angebot dieser be-

ginnt bei etwas günstigeren Modellen, ohne Display und mit eingeschränkter Funktionalität. Mit diesen kann der Benutzer somit nur die Aktivitäten und den Schlaf aufzeichnen und diese Daten in der Anwendung auf dem Smartphone beobachten.
Den vollen Funktionsumfang erlangt man durch die teureren Modelle, welche einer Smartwatch sehr stark ähneln. Hier können zunächst die gesammelten Daten, zum Teil über ein eigenes Display, angezeigt werden. Außerdem besitzen diese neben der Herzfrequenz noch weitere Funktionen. Je nach Modell sind zusätzlich eine GPS-Verbindung oder auch Befehle zur Smartphone-Steuerung enthalten. So kann man über die Uhr zum Beispiel Benachrichtigungen lesen oder auch die Musik auf dem Smartphone steuern. [4]

### 2.1.2 Fitbit Surge

In der vorliegenden Arbeit wurde zur Aufzeichnung der Daten und Verknüpfung mit der Testanwendung die **Fitbit Surge** verwendet. Preislich ist dies das teuerste und somit auch das umfangreichste Modell welches angeboten wird.
Neben den Standardfunktionen wie die Aktivitätsaufzeichnung, können hier auch die oben genannten Funktionen in Verbindung mit dem Smartphone verwendet werden. Außerdem ist die Uhr mit einem GPS-Tracker ausgestattet, was es ermöglicht, ohne Smartphone Sport zu machen und gleichzeitig die gelaufene Strecke aufzuzeichnen. Wichtig für die Testanwendung ist jedoch die Aufzeichnung der Herzfrequenz und des Schlafes, was die Uhr ebenfalls beherrscht und durchgehend misst.
Wie alle Produkte im Sortiment synchronisiert sich die Uhr automatisch über Bluetooth mit dem Smartphone, sodass die Daten in der Anwendung dargestellt werden können. [5]
Die Abbildung 2.1 zeigt dieses Modell in der schwarzen Ausführung. Hier ist zu sehen, dass die Uhr über ein schwarz-weiß Display verfügt und momentan der Modus zur Aktivitätsaufzeichnung aktiviert ist. Über diesen kann zum Beispiel ein Lauf gemessen werden, wobei unter anderem die Strecke, Zeit und Herzfrequenz, wie hier zu sehen, erfasst werden.
Darüber hinaus besitzt die Uhr noch weitere, vordefinierte Modi, um verschiedenste Sportarten individuell aufzeichnen zu können.


**Abbildung 2.1:** Abbildung der Fitbit Surge während der Aufzeichnung einer Aktivität [5]

## 2.2 Auswertung der Daten über die Fitbit-Anwendung

Zusätzlich zu den Produkten bietet Fitibit auf allen mobilen Plattformen (Android, iOS, Windows Phone) eine Anwendung an, über welche die gesammelten Daten überwacht, aber auch bearbeitet werden können. Voraussetzung für den vollen Funktionsumfang dieser Apps ist jedoch ein Aktivitäts-Tracker, da ohne dessen Aufzeichnungen ein Großteil der Funktionen nicht genutzt werden kann.

**Aufbau** Die Anwendung ist im wesentlichen sehr einfach aufgebaut. Nach der erfolgreichen Verbindung mit dem Armband gelangt der Benutzer auf das Dashboard. Wie in Abbildung 2.2 links kann man sich hier einen schnellen Überblick über alle wichtigen Daten verschaffen. Die Anordnung der einzelnen Kacheln kann der Benutzer dabei nach belieben verändern, sodass der wichtigste Punkt im größten, oberen Bereich der Seite erscheint. Außerdem können hier Informationen über das Armband wie zum Beispiel der Akkustand eingesehen werden.
Jede dieser Kacheln auf dem Dashboard ist nebenbei ein Button, welcher den Benutzer zu dem dazugehörigen Menü führt. Diese Untermenüs sind dabei sehr ähnlich aufgebaut und setzen sich im wesentlichen aus zwei Teilen zusammen. Zunächst ist im oberen Bereich des Displays ein Diagramm zu sehen, welches die Daten zu diesem Menüpunkt visualisiert. Darunter befindet sich jeweils eine Tabelle, die die Einträge des Diagramms auflistet. In der Abbildung 2.2 rechts wird ein Beispiel dazu aufgezeigt.
Außerdem gibt es auf jeder Seite eine Navigationsleiste am oberen Bildschirmrand und eine Tableiste am unteren. Die Navigationsleiste enthält weitere Buttons welche für das jeweils ausgewählte Menü zuständig sind, wohingegen die Tabbar Verlinkungen zu den wichtigsten Seiten der Anwendung aufführt.

**Funktionen** Der Funktionsumfang der Anwendung ist wie zu erwarten sehr hoch. Neben der Visualisierung aller Daten kann der Benutzer auch selber Einträge hinzufügen oder löschen. Außerdem können über die Anwendung alle Einstellungen bezüglich der Uhr, aber auch der des Nutzerprofils und somit der Datenauzeichnung - und Darstellung, vorgenommen werden.
Unabhängig von dem Aktivitäts-Tracker kann aber auch die Nahrungsaufnahme und das Gewicht eingetragen werden, um einen noch besseren Überblick über die Gesundheit zu bekommen. Außerdem bietet die App die Möglichkeit Freunde hinzuzufügen, um Statistiken mit diesen zu vergleichen und zu teilen, was wiederum die Motivation steigern soll.
Für Neueinsteiger existieren zusätzlich zahlreiche Anleitungen um den Weg in das Ökosystem und die Fitnesswelt zu vereinfachen.

### 2.2.1 Schnittstelle für Entwickler

Wie bereits erwähnt, stellt Fitbit neben der eigenen App auch eine API für Entwickler zur Verfügung. Über diese können Drittanbieter sowohl über das Web, als auch über das Smartphone mit einer eigenen Anwendung die gesammelten Daten auswerten.
Fitbit bietet hier eine riesige Auswahl an Möglichkeiten die Daten abzufragen und verspricht, weitere Befehle auf Anfrage einzubauen. Die Daten werden dabei über einen Server heruntergeladen. Auf die genaue Funktionsweise und die Befehle wird im Abschnitt 3.4 genauer eingegangen.


**Abbildung 2.2:** Screenshots aus der Fitbit-Anwendung. Links: Das Dashboard. Rechts: Das Untermenü zum Punkt „Schlaf“[6]

Um sich mit dem System vertraut zu machen, bietet Fitbit eine Umfangreiche Dokumentation mit zahlreichen Beispielen und Referenzen, welche zum Teil fertige Beispielabschnitte beinhalten. So wird es dem Entwickler sehr leicht gemacht sich in die API einzulesen und damit zu arbeiten. Außerdem gibt es noch ein Entwickler-Forum, in welchem einige Mitarbeiter des Herstellers vertreten sind, die die Fragen schnell und zuverlässig beantworten können. [7]

## 2.3 Vergleichbare Hersteller

In diesem Teilabschnitt werden die Ökosysteme vergleichbarer Hersteller aufgezeigt. Da diese nur als Referenz dienen, wird die Beschreibung sehr kompakt gehalten.
Um vergleichbare Ergebnisse ermitteln zu können, werden im Folgenden jeweils die Produkte mit dem größten Funktionsumfang und somit auch die teuersten dieser Kategorie verglichen.

### 2.3.1 UP by Jawbone

Jawbone ist ebenfalls ein sehr junges Unternehmen, welches im Dezember 1999 gegründet wurde. Zunächst beschränkte sich ihre Produktion auf Bluetooth-Lautsprecher und verschiedene Kopfhörer. Durch das wachsende Interesse an Fitnessprodukten stellte Jawbone jedoch im No-

vember 2011 die **UP**-Reihe vor. Ähnlich wie das Fitbitprodukt handelt es sich hierbei um ein Armband, über das die Aktivitäten aufgezeichnet werden können. [9]

**UP3** Das neuste Modell ist das UP4. Dies ist jedoch in Deutschland nicht verfügbar und bietet als einziges Upgrade einen NFC-Chip, welcher Bargeldloses bezahlen ermöglicht. Aus diesem Grund wird hier das UP3 vorgestellt (siehe Abb. 2.4).
Im Gegensatz zur Surge von Fitbit, muss der Benutzer bei diesem Armband auf ein Display verzichten. Dieses bietet lediglich Status-LEDs, welche den aktuellen Zustand des Bandes anzeigen können. Auch bei der Herzfrequenz gibt es gewisse Unterschiede. Das UP3 zeichnet die Frequenz nicht durchgehend auf, sondern misst nur den Ruhepuls über den gesamten Tag. Bis auf die fehlende GPS-Funktion und die Smartphone-Benachrichtigungen bietet das Band jedoch den gleichen Umfang wie das Fitbit Produkt. Auch hier werden automatisch Aktivitäten und der Schlaf erkannt und aufgezeichnet. Synchronisiert werden die Daten schlussendlich ebenfalls über Bluetooth, was bei nahezu allen Wearables üblich ist.
Die fehlenden Funktionen machen sich jedoch auch im Preis bemerkbar, da die Jawbone Produkte deutlich günstiger sind. [10]

**Anwendung** Anwendungen von Fitnessarmbändern unterscheiden sich von der Funktionalität und des Aufbaus nur sehr gering. Daher besitzt auch diese App ein Dashboard, über welches die über den Tag gesammelten Informationen anschaulich visualisiert werden. Zusätzlich können die einzelnen Daten auch in einer Detailansicht eingesehen werden.
Des Weiteren werden hier automatisch erkannte Aktivitäten dargestellt. Diese können jedoch auch über die Anwendung manuell hinzugefügt oder bearbeitet werden.
Zuletzt wird auch diese App als Motivator für den Benutzer verwendet. Durch Vergleiche mit Freunden oder motivierende Zusatzinformationen und Tipps wird dieses Ziel erreicht.
Abbildung 2.3 zeigt das Dashboard dieser Anwendung im Vergleich.

**Schnittstelle für Entwickler** Auch die Schnittstelle zu Jawbones UP ist im wesentlichen sehr ähnlich aufgebaut. Wie auch bei Fitbit können hier die gesammelten Daten des Benutzers über einen Server heruntergeladen werden. Diese sind dabei ebenfalls in die einzelnen Kategorien unterteilt. Zusätzlich bietet Jawbone jedoch ein SDK (**S**oftware **D**evelopment **K**it), welches dem Entwickler den Einstieg erleichtert und zusätzlich Zugang zu eigenen Methoden bietet. So können zum Beispiel Diagramme im Stil Jawbones realisiert werden, ohne dass diese extra implementiert werden müssen.
Jawbone bietet dem Entwickler Zugriff zu allen gesammelten Daten des Armbandes. Da hier jedoch weniger Werte aufgezeichnet werden, wie zum Beispiel bei der Herzfrequenz, ist Fitbits API etwas umfangreicher. [11, 12, 13]

### 2.3.2 Apples Apple Watch

Seit April 2015 führt Apple neben den bekannten Macs, iPhones und iPad nun auch die Apple Watch. Diese zählt zu der Kategorie der Smartwatches, besitzt jedoch, wie auch die Aktivitäts-Tracker, einige Sensoren um die Vitalwerte des Körpers zu messen. Da diese sich in der Funktionsweise sehr ähneln und Apple ebenfalls eine Schnittstelle für Entwickler zur Verfügung stellt, bietet dieses Produkt einen anschaulichen Vergleich mit Fitbit.

**Apple Watch Series 2** Die aktuellste Version ist dabei die Series 2 (siehe Abb. 2.4), welche, wie der Name schon schließen lässt, erst die zweite Generation dieses Produktes ist.
Der Funktionsumfang, was die Aufzeichnungen der Vitalwerte betrifft, unterscheidet sich im Vergleich zur **Fitbit Surge** jedoch kaum. Auch die Apple Watch bietet eine kontinuierliche Herzfrequenz-Aufzeichnung, einen integrierten GPS-Sensor und das automatische Erkennen von Aktivitäten.
Lediglich der Schlaf wird zunächst nicht aufgezeichnet. Die Uhr kann jedoch mithilfe einer Anwendung um diese Funktionalität erweitert werden. Hierfür existiert ein von Apple bereitgestellter Appstore, welcher ausschließlich Anwendungen für die Uhr enthält.
Zusätzlich besitzt die Apple Watch im Gegensatz zur **Surge** ein Farbdisplay und ist hochwertiger gebaut. Dies macht sich jedoch im Preis bemerkbar, da die Fitbit Produkte deutlich günstiger als die Apple Watch sind. [14]

**Anwendung** Die Apple Watch wird, da sie direkt in Apples Ökosystem integriert ist, mit mehreren Anwendungen gleichzeitig synchronisiert. Zunächst werden alle Daten in der **Watch-App** gesammelt. Neben den Übersichten zu den gesammelten Daten können hier Einstellungen bezüglich der Uhr vorgenommen werden. Da es sich dabei um eine Smartwatch und nicht nur um einen Aktivitäts-Tracker handelt, sind diese natürlich deutlich umfangreicher.
Neben dieser Anwendung, werden alle Daten mit Apples **Health-Kit** synchronisiert. In dieser App werden alle möglichen Vital-Daten zusammengeführt und visualisiert. Dabei werden nicht nur die Daten der Apple Watch dargestellt, sondern auch Werte, welche über das Smartphone oder weitere Wearables gemessen wurden. Somit besitzt der Benutzer hier einen Sammelpunkt, in welchem alle Daten zusammengefasst werden. Fitbit hingegen verzichtet auf die Synchronisation mit Apples Health-Kit, weshalb diese Daten an dieser Stelle nicht eingesehen werden können.

**Schnittstelle für Entwickler** Für Entwickler unterscheidet sich die Implementierung einer Anwendung gravierend. Der größte Unterschied ist zunächst, dass nicht nur die Daten eines Servers ausgelesen werden können, sondern, im Gegensatz zu Fitbit, eigens für die Apple Watch erstellte Anwendungen entwickelt werden können. Diese werden somit auf der Uhr installiert und können direkt mit dieser interagieren. Zwar benötigen viele Anwendungen bisher trotz allem die Unterstützung des iPhones, da die Uhr nicht genügend Leistung besitzt, so erweitert dies dennoch den Funktionsumfang enorm. Der Entwickler kann somit nicht nur bereits gesammelte Daten abrufen, sondern kann die Uhr selbst so programmieren, dass ein bestimmter Wert über einen bestimmten Zeitraum gemessen wird. So ist es zum Beispiel möglich, die oben genannte Schlafauszeichnung selbst zu entwickeln.
Damit sind dem Entwickler, was die Funktionalität und die Verarbeitung der gesammelten Daten betrifft, anders als bei Fitbit, kaum Grenzen gesetzt. Dabei darf jedoch erneut nicht unerwähnt bleiben, dass es sich bei der Apple Watch um eine Smartwatch und nicht ausschließlich um einen Aktivitäts-Tracker handelt. [16, 17, 18]

### 2.3.3 Garmin

Garmin ist ein Schweizer Unternehmen, welches im Jahre 1989 gegründet wurde. Zunächst spezialisierte sich dieser Hersteller auf Navigationsgeräte. Seit März 2014 bietet Garmin jedoch ebenfalls Aktivitäts-Tracker in verschiedenen Ausführungen und Formen an. [19]

**Vívoactive HR** Das neuste Modell ist dabei die Vívoactive HR. Bei dieser handelt es sich, wie bei der Fitbit Surge, um ein Fitnessarmband, welches zusätzlich ein kleines Display enthält. Dieses Gerät besitzt, im Gegensatz zu Fitbits Pendant ein Farbdisplay, was die Darstellung der Inhalte deutlich verbessert. Außerdem kann das Armband dank Wasserdichte auch zum Schwimmen getragen werden.
In den Funktionen unterscheiden sich die Geräte jedoch kaum, da hier ebenfalls neben der GPS-Verbindung und der Kontinuierlichen Herzfrequenz-Messung, hauptsächlich Aktivitäten erkannt und aufzeichnet werden können.
Zusätzlich bietet Garmin jedoch ebenfalls den **Connect IQ**-Bereich an, über welchen eigene Anwendungen oder Erweiterungen zu dem Armband heruntergeladen werden können. Dieser ist über die, auf dem Smartphone installierte App, erreichbar.
Die Abbildung 2.4 zeigt die hier genannten Produkte im Vergleich. [20]

**Anwendung** Die Anwendung unterscheidet sich im Wesentlichen kaum von den bereits genannten Herstellern. Hier werden hauptsächlich die gemessenen Daten des Armbandes in anschaulichen Diagrammen und Übersichten dargestellt. Des Weiteren können über diese einige Einstellungen an dem Tracker vorgenommen werden. Zusätzlich dient die App durch Ranglisten und einigen Benachrichtigungen zur Motivation. Abbildung 2.3 zeigt jeweils die Übersichtsseite der hier genannten Anwendungen. [21]

**Schnittstelle für Entwickler** Garmin bietet verschiedene Schnittstellen für die Entwicklung an. Darunter befindet sich zum Beispiel der einfache Zugang zu den sogenannten „Watch faces“[22], welche den Hintergrund und das Ziffernblatt der Uhr bestimmen. Wie auch Fitbit stellt Garmin zusätzlich eine Schnittstelle für die Fitnessdaten zur Verfügung. Über diese können Übersichten und Zusammenfassungen der jeweiligen Kategorien heruntergeladen oder manipuliert werden. Um Zugang zu dieser API zu bekommen, muss jedoch zunächst ein umfangreiches Formular ausgefüllt werden, durch welches Garmin die nötigen Informationen zu der geplanten Anwendung erhält.
Das Abrufen und Bearbeiten dieser Daten basiert jedoch auf der selben Architektur wie bei Fitbit und wird somit über einen Server mit einer bestehenden Internetverbindung realisiert. [22, 13]


**Abbildung 2.3:** Screenshots der App-Übersichten. Links: UP by Jawbone. Mitte: Garmin Connect Mobile. Rechts: Watch. [21, 23, 24]


**Abbildung 2.4:** Abbildungen der Produkte. Links: UP 3 by Jawbone. Mitte: Vívoactive HR. Rechts: Apple Watch Series 2. [10, 20, 14]

# 3
# Technische Grundlagen

Dieses Kapitel befasst sich nun mit den technischen Grundlagen. Dazu werden die Entwicklungsumgebungen und Bibliotheken, welche zur Entwicklung der Anwendung benötigt wurden, aufgezeigt und erläutert.
Zunächst beginnt dieser Abschnitt mit dem wichtigsten Teil, der Programmierumgebung und der dazugehörigen Programmiersprache. Anschließend werden die dort verwendeten Bibliotheken genannt, woraufhin die genaue Funktionsweise der Schnittstelle von Fitbit erklärt wird.

## 3.1 Programmierumgebung - Xcode

Für die folgende Anwendung wurde als Entwicklungsumgebung die Xcode Version 8.2.1 mit der aktuellen Programmiersprache Swift 3 (siehe Abschnitt 3.2) verwendet. Xcode ist eine Programmierungsumgebung, welche hauptsächlich zur Entwicklung nativer Anwendungen für Apple-Betriebsysteme (macOS, iOS, tvOS und watchOS) verwendet wird. Aus diesem Grund ist Xcode ausschließlich für macOS verfügbar und kann nicht auf Windwos installiert werden. Grundsätzlich setzen sich die in Xcode erstellten Projekte aus zwei Teilen zusammen. Zunächst gibt es den eigentlichen Code, welcher für die Logik und den Ablauf der Anwendung zuständig ist. Dazu ergänzend bildet den zweiten Teil der **Interfacebuilder**, mit welchem die UI-Elemente platziert und bearbeitet werden können.
In der folgenden Anwendung wurde jedoch auf diesen Interfacebuilder verzichtet. Dessen Funktionalität kann durch den Quellcode jedoch ersetzt werden. Der Verzicht des Interfacebuilders sorgt für einen einfacheren Umgang der Layout-Anpassungen auf den verschieden Displaygrößen. Da hier jedoch die grafische Oberfläche, mit welcher man die UI-Elemente der Anwendung platziert und bearbeitet, fehlt, ist der eigentliche Aufbau der Anwendung komplizierter und aufwändiger. Für diesen Fall existiert jedoch eine Bibliothek, welche den Entwickler dabei unterstützt die UI-Elemente zu organisieren. Diese wird im Abschnitt 3.4 PureLayout genauer aufgezeigt.
Getestet wurde die Anwendung auf den möglichen und aktuellen Simulatoren, welche bei Xcode

enthalten sind und auf einem iPhone 6s.
Wie es bei der Entwicklungsumgebung um Xcode üblich ist, wurde die Anwendung auf der Basis des aktuellen Betriebssystems iOS 10 entwickelt. Außerdem wurde zur Datenspeicherung die interne Datenbank „CoreData“ verwendet.

## 3.2 Swift 3

> „After Apple unveiled the Swift programming language, it quickly became one of the fastest growing languages in history. Swift makes it easy to write software that is incredibly fast and safe by design [...] Swift is intended as a replacement for C-based languages (C, C++, and Objective-C)“[25]

Wie schon erwähnt, wurde für die Anwendung die Programmiersprache **Swift** verwendet. Diese ist die relative junge, eigens von Apple entwickelte Spache. Die erste Swift-Version wurde erst auf der Entwicklerkonferenz im Jahr 2014 vorgestellt. Bis zum heutigen Zeitpunkt wird diese dabei stetig weiterentwickelt, weshalb im Sommer 2016 die Version Swift 3 vorgestellt wurde. Wie im Zitat zu lesen ist, gehört Swift zu den am schnellsten wachsenden Programmiersprachen der Welt. Dies ist vor allem auf die große Anzahl an iOS-Entwicklern und die große Nachfrage an Anwendungen aus Apples AppStore zurückzuführen.
Die Sprache basiert dabei auf den jeweils besten Aspekten der Programmiersprachen C, C++ und Objective-C. Aus diesem Grund war Swift bis zu der Version 2.2 kompatibel mit Objective-C. Mit der dritten Generation wurden jedoch größere Änderungen veröffentlicht, was sie zu einer eigenständigen Sprache ohne Kompatibilität entwickelt hat.
Swift kann sich durch mehrere Besonderheiten der modernen Programmierung auszeichnen. So muss der Entwickler zum Beispiel nicht hinter jede Zeile ein Semikolon setzten, was die Programmierung etwas vereinfacht und Fehler durch das Vergessen des Semikolons vermeidet. Eine weitere besondere Eigenschaft der Programmiersprache ist die Fähigkeit optionale Variablen zu erstellen. Wird eine Variable, wie im Listing 3.1 [26] mit einem Fragezeichen versehen, kann diese existieren, muss es aber nicht zwangsläufig. Durch ein Ausrufezeichen, wie in der Zeile 9 des Listings, kann die Variable jedoch dazu „gezwungen“ werden. Bei der Verwendung dieses Elementes sollte man jedoch vorsichtig sein, da es hier, bei falscher Anwendung, schnell zu Fehlern und somit zu Abstürzen führen kann. Das Programm erwartet nun, dass die Variable einen Wert besitzt. Sollte dieser jedoch nicht verfügbar sein, kann es zu einem Fehler führen.

**Listing 3.1:** Beispiel eines optionales Datentyps [26]

```
1 // -----?
2 if let roomCount = john.residence?.numberOfRooms {
3         print("John's residence has \(roomCount) room(s).")
4 } else {
5 print("Unable to retrieve the number of rooms.")
6 }
7
8 // -----!
9 let roomCount = john.residence!.numberOfRooms
```

Außerdem fällt bei der Deklaration der Variablen und dessen Datentypen eine weitere Besonderheit der Sprache auf.
Zunächst wird zwischen zwei Unterschiedlichen Varianten unterschieden. Diese können, wie das Listing 3.2 zeigt entweder konstant, was durch das Schlüsselwort *let* ausgelöst wird, oder variabel sein, was durch *var* realisiert wird.

**Listing 3.2:** Erstellen eines Datentyps

```
1 let constant = "Test"    // String
2 var variable = 55        // Int
3 let test = 12.0          // Double
```

Zusätzlich fällt auf, dass hier kein bestimmter Datentyp festgelegt werden muss, da Swift diesen anhand des Wertes automatisch erkennt und festlegt.

## 3.3 Bibliotheken

Neben den Standard Frameworks wurden für die Entwicklung folgende Bibliotheken verwendet.

**Charts** Charts unterstützt, wie der Name schon schließen lässt, den Entwickler beim Erstellen von Diagrammen aller Art. Dieses wurde verwendet, um die gesammelten Daten in einer anschaulichen Form darzustellen [27]. Das hier abgebildete Listing 3.3 zeigt, wie ein so ein Diagramm in Swift erstellt und verwendet wird.
Für dieses Beispiel werden zunächst zwei Testarrays, welche den späteren Datensatz darstellen, in Zeile 3 und 4 erstellt. Durch die darauf folgende *If-Abfrage* wird sichergestellt, dass diese nicht leer sind. In diesem Fall ist diese Abfrage nicht notwendig, da die Arrays direkt darüber befüllt werden. In einer echten Anwendung ist dies jedoch nicht immer eindeutig und schützt somit vor Abstürzen.
Über die *For-Schleife* von Zeile 6 bis 10 werden die Arrays in ein extra Array vom Datentyp **BarChartDataEntry** geschrieben, welches anschließend als Quelle für das Diagramm verwendet wird. Ab Zeile 15 folgen nun nur noch Einstellungen bezüglich des Diagramms. Hier kann von den Farben bis hin zu den Positionen und den Einstellungsmöglichkeiten nahezu alles verändert werden. Dazu sind in diesem Fall nur einige Beispiele der Möglichen Einstellungen aufgelistet.
Zuletzt muss nur noch der Fall eines leeren Diagramms implementiert werden. Hierfür existiert ein extra Textfeld, welches in diesem Fall erscheint. Somit genügt es, hier die gewünschte Nachricht einzutragen. Dieser Fall tritt auf, wenn die oben genannte *If-Bedingung* nicht erfüllt wird.

**Listing 3.3:** Erstellung eines Diagramms

```
1  var dataEntries: [BarChartDataEntry] = []
2
3  let xVals: [String] = ["So", "Mo", "Di", "Mi", "Do", "Fr", "Sa"]
4  let yVals: [Double] = [1.0,2.0,5.0,3.0,1.5,10.0,11.0]
5  if  yVals.count > 0{
6          for i in 0..<xVals.count {
7                  let dataEntry = BarChartDataEntry(x: Double(i), y
       : Double(yVals[i]))
8                  dataEntries.append(dataEntry)
9          }
10
11         let chartDataSet = BarChartDataSet(values: dataEntries,
       label: text)
12         let chartData = BarChartData(dataSet: chartDataSet)
13         barView.data = chartData
14
15         barView.legend.textColor = .white
16         barView.backgroundColor = .black
17         [...] // weitere Farbeinstellungen
18         barView.chartDescription?.text = "Beispiel"
19         barView.xAxis.labelPosition = .bottom
20         barView.leftYAxisRenderer.axis?.axisMinimum = 0
21         barView.drawGridBackgroundEnabled = true
22         barView.pinchZoomEnabled = false
23 }else{
24         barView.noDataText = "Keine Daten Vorhanden"
25         barView.noDataTextColor = .white
26 }
```

**PureLayout** Um die grafische Oberfläche der Anwendung aufzubauen, wurde hier nicht der integrierte „Interfacebuilder“ verwendet, sondern die Bibliothek PureLayout [28]. Dies ermöglicht es die Oberfläche programmatisch zu erstellen und bringt gegenüber dem Interfacebuilder den Vorteil, dass die Anpassung an die verschiedenen Displaygrößen deutlich vereinfacht wird. Die Platzierung der Elemente ist sehr intuitiv gehalten, wie am Beispiel des Listings 3.4 zu sehen ist. Dabei gibt es jedoch kein Einheitliches vorgehen wie ein Element auf der Benutzeroberfläche arrangiert werden muss da mehrere funktionierende Varianten existieren.
In diesem Beispiel wird zunächst in Zeile 1 ein Element in der Mitte des ViewControllers angeordnet. Daraufhin wird die Größe des zweiten Elementes bestimmt. Dieses ist dabei 100dp(Zeile 2) hoch und 180dp (Zeile 3) breit. Nach diesem Vorgang wird dieses an der linken Seite (Zeile 4) des Displays und an der Unterseite des ersten Elements „befestigt“ (Zeile 5). In beiden Fällen wird dazu ein kleiner Abstand eingebaut.

**Listing 3.4:** Beispiel zur Platzierung eines UI-Elementes

```
1 testElement.autoCenterInSuperview()
2 testElement2.autoSetDimension(.height, toSize: 100)
3 testElement2.autoSetDimension(.width, toSize: 180)
4 testElement2.autoPinEdge(toSuperviewEdge: .left, withInset: 2.0)
5 testElement2.autoPinEdge(.top, to: .bottom, of: testElement,
     withOffset: 10.0)
```

**SampleBit** Zuletzt wurde eine Bibliothek verwendet, welche den Login über OAuth2 mit Fitbit durchführt und vereinfacht.[29] Dafür müssen zunächst alle wichtigen Informationen zu dem Login gesammelt werden. Dazu zählen unter Anderem eine **Client ID** und ein **Client Secret**, welche durch Fitbit vergeben werden.
Anschließend wird mit Hilfe der Login-Methode (siehe Abschnitt 5.3.1) die Anmeldung durchgeführt. Die daraus erhaltenen Login-Daten werden schlussendlich, in einer Funktion die bei Erfolg aufgerufen wird, auf dem Gerät gespeichert. Außerdem wird der Benutzer an dieser Stelle in die eigentliche Anwendung geleitet. Für die Letzteren Punkte musste die Bibliothek jedoch erweitert werden, da diese lediglich den Login vollzieht.

## 3.4 Schnittstelle zu Fitbits Daten

Dieser Abschnitt befasst sich mit der von Fitbit bereitgestellten API. Dabei werden die einzelnen Elemente und die nötigen Schritte, die zur Entwicklung benötigt werden, aufgezeigt und erklärt.

### 3.4.1 Kommunikation der Schnittstellen

Zunächst muss zur genauen Erläuterung der Weg der Daten genauer untersucht werden. Als Entwickler einer eigenen Anwendung ist es nicht möglich über Bluetooth auf die Daten des Fitnessarmbandes zuzugreifen. Die gesammelten Daten müssen hierfür einen deutlich längeren und komplizierteren Weg zurücklegen.

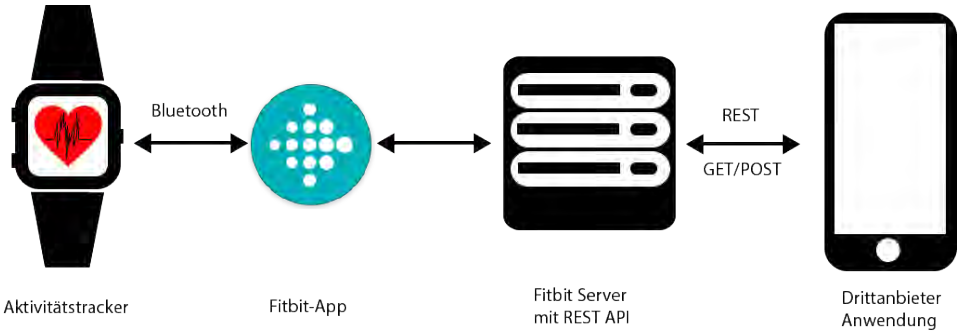



**Abbildung 3.1:** Kommunikation zwischen der Schnittstelle und der Anwendung

Die Abbildung 3.1 veranschaulicht diesen Weg. Wie man hier sehen kann, werden die gesammelten Daten zunächst an die Fitbit-App per Bluetooth gesendet. Damit ist diese Anwendung für die Datenübertragung unverzichtbar. Diese leitet nun wiederum die Daten mit einer bestehenden Internetverbindung an den Fitbit-Server. Auf diesen kann nun schließlich die eigene Anwendung zugreifen, um die Daten herunterzuladen. Die Probleme, welche durch diesen Ablauf zustande kommen, werden im Abschnitt 7.1.2 genau aufgezeigt.

### 3.4.2 RESTful-Service mit HTTP

Bei der Architektur, welche für das Abrufen der Daten verwendet wird, handelt es sich um einen sogenannten „REST-Service“(kurz für: **Representational State Transfer**). Dieser dient dabei als Schnittstelle zwischen dem Web-Dienst und der Anwendung, also dem Computer oder in diesem Fall dem Smartphone.
Umgesetzt wird diese Kommunikation durch das HTTP-Protokoll, was die wohl häufigste Methode ist. Dieses Protokoll bietet einige Standard-Methoden durch welche die Daten auf dem Server manipuliert werden können. Durch diese definiert sich damit die Art der Manipulation. Im Verlauf der Arbeit wurden dabei die Folgenden drei Funktionen verwendet.

- **GET**: Die am häufigsten verwendete Methode ist die **GET-Anfrage**. Mit dieser werden die Daten ohne Veränderung, von dem Server ausgelesen.
- **POST**: Durch den **POST** können neue Daten durch den Benutzer auf dem Server erstellt werden.
- **DELETE**: Hier werden, wie der Name schon sagt, Ressourcen von dem Server entfernt.

Zur Veranschaulichung zeigt das Folgende Listing 3.5 wie dieser Prozess in Swift realisiert wird.

**Listing 3.5:** Beispiel eines HTTP GET-Requests

```
1  let token = Data().getToken()
2  let myUrl = NSURL(string: "https://api.fitbit.com/1/user/-/
       profile.json")
3  let request = NSMutableURLRequest(url:myUrl! as URL)
4  request.httpMethod = "GET"
5
6  request.addValue("Bearer " + token, forHTTPHeaderField: "
       Authorization")
7  DispatchQueue.global().async {
8          let task = URLSession.shared.dataTask(with: request as
       URLRequest,
9          completionHandler:{
10         (data, response, error) in
11                 // Empfangen und auslesen der Daten
12                 [...]
13         }
14 }
```

Zunächst werden hierfür die benötigten Informationen und Variablen definiert. Da Fitbit eine Authentifizierung benötigt, wird dieser Schlüssel, welcher durch den Login vergeben wird, zunächst festgelegt. Daraufhin wird in Zeile 2 die URL für den Request bestimmt.
In Zeile 3 und 4 wird dem URL die Request-Methode hinzugefügt. Als letztes wird nun noch der Schlüssel zu Authentifikation angehängt. Nun kann der Request abgesendet werden. Als Antwort auf diese Anfrage liefert die Schnittstelle eine Datei im „JSON-Format“, welches im folgenen Abschnitt erläutert wird.

### 3.4.3 JSON

Die Abkürzung JSON steht für **J**ava**S**cript **O**bject **N**otation und wird häufig für Webanwendungen verwendet. Es eignet sich besonders für die Datenübertragung, da es ein sehr kompaktes, auf Textbasiertes Dateiformat ist, welches nur die wesentlichen Informationen beinhaltet. Somit können Ressourcen und damit auch Zeit erspart werden.
Der Inhalt dieser JSON Datei beruht dabei auf dem „Key-Value“-Prinzip. Dies bedeutet, dass die aufgelisteten Variablen zunächst mit einem Namen betitelt werden (Key), welcher mit dem dazugehörigen Wert (Value) verknüpft ist. Dies ist vor allem für den Entwickler wichtig, da dieser die Werte genau nach diesem Prinzip aus der JSON-Datei auslesen kann. Die einzelnen Objekte werden dabei durch Klammerung getrennt. Ein explizites Beispiel, wie dies in der Anwendung ausgelesen und verarbeitet wird, ist im Abschnitt 5.3.2 zu sehen und wird nochmal genauer erläutert.
Das Listing 3.6 zeigt wiederum ein Beispiel wie eine Antwort im JSON-Format aussehen kann. Hier kann man die verschiedenen Stufen und Aufteilungen des Objektes deutlich erkennen. Das Objekt welches in diesem Fall dargestellt wird ist ein „activities-heart“-Objekt, wie in Zeile 2 zu sehen ist. Dieses beinhaltet dabei weitere Variablen, welche einen einfachen Wert besitzen können, wie die Variable „dateTime“ in Zeile 4, aber auch weitere Objekte, wie bei dem Beispiel „heartRateZones“, welches wiederum mehrere Variablen besitzt.
Das Objekt „heartRateZones“ bildet somit ein Array mit 4 Einträgen (siehe Zeile 9 bis 35). Jeder Eintrag wird dabei durch geschwungene Klammern umschlossen und besitzt jeweils 5 weitere Variablen. Durch die eckigen Klammern endet das Array und die nächste Variable kann aufgeführt werden (Zeile 37).

**Listing 3.6:** Beispiel JSON-Format zu einer Herzfrequenz-Anfrage

```
{
 "activities-heart": [
   {
        "dateTime": "2015-08-04",
        "value": {
        "customHeartRateZones": [],
        "heartRateZones": [
                {
                        "caloriesOut": 740.15264,
                        "max": 94,
                        "min": 30,
                        "minutes": 593,
                        "name": "Out of Range"
                },
                {
                        "caloriesOut": 249.66204,
                        "max": 132,
                        "min": 94,
                        "minutes": 46,
                        "name": "Fat Burn"
                },
                {
                        "caloriesOut": 0,
                        "max": 160,
                        "min": 132,
                        "minutes": 0,
                        "name": "Cardio"
                },
                {
                        "caloriesOut": 0,
                        "max": 220,
                        "min": 160,
                        "minutes": 0,
                        "name": "Peak"
                }
                ],
        "restingHeartRate": 68
    }
  }
 ]
}
```

### 3.4.4 Authentifizierung - OAuth2.0

Um sicherzustellen dass der Benutzer nur die eigenen Daten auslesen kann, muss dieser sich verifizieren. Dies wird über den Standard **OAuth2.0** realisiert. Dies ist ein Service, welcher von zahlreichen Anwendungen zur Authentifizierung verwendet wird, darunter sind unter Anderem Facebook oder auch Twitter vertreten. Die Besonderheit bei OAuth ist, dass man dem Nutzer nur Zugriff über einen Teil der Daten ermöglichen kann, ohne alle Details der Anwendung herauszugeben. So wird dieses Verfahren häufig in Verbindung von Facebook verwendet. Der Nutzer gibt dabei einer Anwendung zum Beispiel die Berechtigung Inhalte auf die Pinnwand zu posten, ohne dass die privaten Daten oder die Freundesliste ausgelesen werden können.
Dies wird, wie im Fall der Fitbit-API über einen Token realisiert. Der Token ist ein individueller Schlüssel, welcher bei der Authentifizierung vergeben wird. Dieser wird daraufhin im weiteren Verlauf über den HTTP-Reqest mitgesendet um den Nutzer zu verifizieren. Wie in Abbildug 3.2 zu sehen ist, kann der Benutzer während der Anmeldung festlegen, welche Daten abgefragt werden dürfen und wie lange der Token verfügbar sein soll.

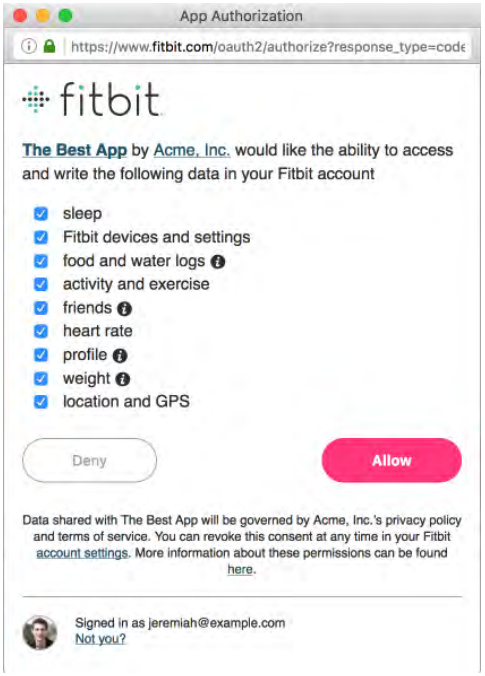



**Abbildung 3.2:** Beispiel einer Anmeldung über OAuth[32]

# 4 Anforderungsanlayse

Das nun folgende Kapitel beschreibt die Anforderungen, welche die Testanwendung erfüllen soll. Dafür werden die funktionalen und die nicht-funktionalen Anforderungen definiert und beschrieben, die in der späteren Implementierung umgesetzt werden sollen.

## 4.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die Aufgaben, die das System erfüllen soll.

- **Native Anwendung**: Die Anwendung soll nativ auf dem Betriebssystem iOS entwickelt werden. Daraus folgt, dass alle von der Plattform bereitgestellten Bedienungselemente verwendet werden sollen. Außerdem müssen dessen Richtlinien eingehalten werden.
- **Serverkommunikation**: Das System soll bei bestehender Internetverbindung erfolgreich und fehlerfrei mit dem Fitbit-Server kommunizieren können, sodass die Daten zu jeder Zeit aktualisiert werden können.
- **Benutzerverwaltung**: Der Benutzer soll erfolgreich an- und abgemeldet werden können. So kann man jederzeit den Benutzer wechseln.
- **Datenauswertung**: Das System soll mindestens die Daten der Herzfrequenz oder des Schlafes herunterladen und auswerten können.
- **Dynamische Requests**: Die Anfragen an den Server sollen unabhängig vom Benutzer und dessen Daten funktionieren. Dabei müssen alle möglichen Variablen innerhalb der Requests unterstützt werden.
- **Dynamische Auswertung**: Die aus den Anfragen erhaltenen Datensätze sollen entsprechend und fehlerfrei verarbeitet und dargestellt werden.
- **Diagramme**: Die Anwendung soll anschauliche Diagramme erstellen, über die der Benutzer die geforderten Daten einsehen und analysieren kann.

- **Anmelden**: Der Benutzer soll, sofern möglich, eigenständig beim Start der Anwendung angemeldet werden, um diesem Zeit zu ersparen.
- **Herzfrequenz**: Bei den Daten der Herzfrequenz soll jeder einzelne Eintrag einsehbar sein.

## 4.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen betrachten das gesamte System und beschreiben, wie dieses funktionieren soll.

- **Zuverlässigkeit**: Die Anwendung soll die gestellten Aufgaben in einem angemessenen Zeitintervall erfolgreich erledigen. Außerdem soll sie fehlertolerant sein, was bedeutet, dass das System Fehler erkennt und diese behandelt ohne abzustürzen.
- **Verfügbarkeit**: Da die Anwendung auf dem Betriebssystem iOS entwickelt wird, soll diese für alle iPhones mit der Version 10.0 oder höher geeignet sein.
- **Benutzerfreundlichkeit**: Die Anwendung soll intuitiv bedienbar sein, sodass der Benutzer sich ohne Hilfe zurechtfinden kann.
- **Erwartungskornform**: Die Benutzeroberfläche der Anwendung soll einheitlich sein und den allgemeinen Gestaltungsregeln entsprechen. Außerdem müssen die gestellten Aufgaben auch die erwarteten Ergebnisse liefern.
- **Korrektheit**: Die Anwendung soll stets korrekte Ergebnisse anzeigen.

# 5 Implementierung

Dieses Kapitel befasst sich mit der Umsetzung des Projektes. Dabei wird zunächst die grobe Dialogstruktur der Anwendung aufgezeigt, um einen besseren Überblick über das Programm zu bekommen. Im weiteren Verlauf werden dann die jeweiligen ViewController und dessen Inhalte genauer beschrieben. Der Fokus liegt dabei vor allem auf den HTTP-Requests und auf der Verarbeitung deren Antworten.

## 5.1 Dialogstruktur

Die Dialogstruktur, wie in Abbildung 5.1 zu sehen ist, zeigt den Aufbau der Anwendung. Jeder hier abgebildete Dialog steht für einen ViewController im Programm. Diese wiederum beinhalten, wie schon erläutert, den Code, welcher die Seite darstellt und dessen Logik verarbeitet. Beim Starten der App gelangt man zunächst auf die Login-Seite. Die Anmeldung ist dabei die Voraussetzung, damit die Anwendung weiter verwendet werden kann, da ohne diese keine Daten von dem Server ausgelesen werden können. Nach einer erfolgreichen Anmeldung gelangt man somit auf das Dashboard, welches die Hauptseite der Anwendung darstellt. Über dieses erreicht man somit alle weiteren Menüpunkte. Außer den Dialogen sind auf dem Diagramm noch Formulare zu sehen. Diese repräsentieren Aktionen des Benutzers, welche die Daten beeinflussen. So ist es zum Beispiel möglich einzelne Daten zu löschen oder sogar neue Daten manuell hinzuzufügen. Der genaue Ablauf wird jeweils im folgendem Abschnitt erläutert.

## 5.2 Architektur

Bevor nun die Testanwendung aufgezeigt und genauer erläutert wird, soll hier nochmals auf die besondere Architektur der Schnittstelle und der Anwendung eingegangen werden.
Der wohl logischste und auch schnellste Weg an die Daten des Armbandes zu kommen, wäre offensichtlich eine direkte Kommunikation per Bluetooth zwischen der Anwendung und dem

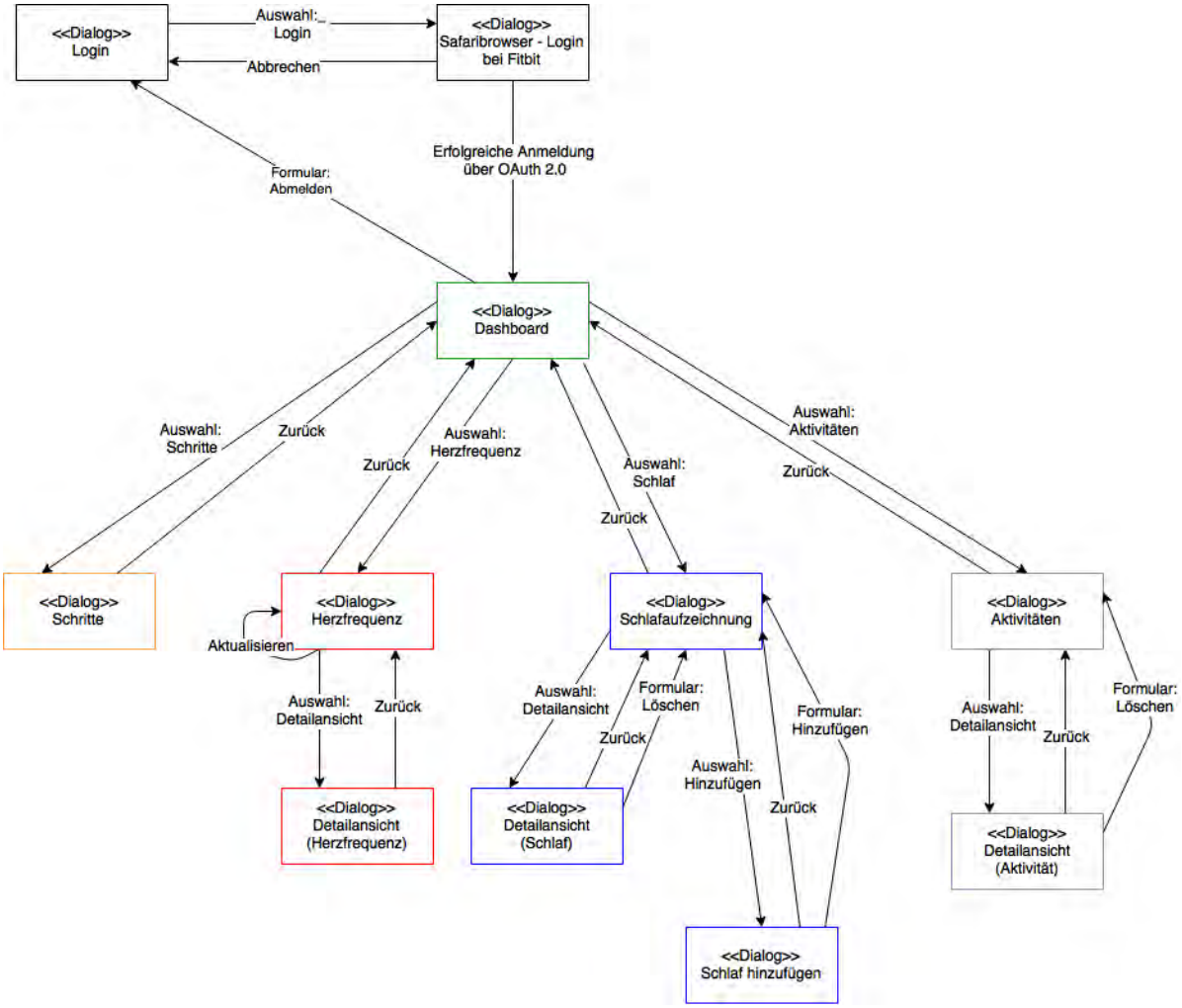



**Abbildung 5.1:** Dialogstruktur der gesamten Anwendung

Armband. Dies wird dem Entwickler jedoch, wie schon im Abschnitt 3.4.1 angesprochen, nicht erlaubt. Daher musste während der kompletten Implementierung der Umweg über den Fitbit-Server genommen werden. Die Abbildung 5.2 veranschaulicht dabei nochmals den genauen Ablauf der Kommunikation, welche zum Austausch der Daten notwendig ist.
Zunächst müssen die Daten zwischen dem Armband und der Fitbit-Anwendung über eine Bluetooth-Verbindung synchronisiert werden. Dieser Vorgang wird bei bestehender Verbindung in regelmäßigen Abständen automatisch durchgeführt, kann jedoch auch zu jedem Zeitpunkt, manuell getätigt werden.
Da die Daten nun auf dem Smartphone sind, können diese durch eine bestehende Internetverbindung auf den Fitbit-Server geladen werden. Von diesem können schlussendlich die aufgezeichneten Werte über die selbst entwickelte Anwendung, heruntergeladen werden. Da die Ressourcen jedoch nicht kontinuierlich auf den Server geladen werden, kann es an dieser Stelle zu größeren Verzögerungen kommen, weshalb die Werte, vor allem die Herzfrequenz, in der eigenen Anwendung nicht aktuell sein können. Dieses Problem wird im weiteren Verlauf häufig auftreten und genauer erläutert.

So war neben der Visualisierung der Datensätze die Kommunikation mit dem Fitbit-Server, aufgrund dieser Architektur der zentrale Punkt während der Implementierung, da diese für alle Daten, welche in der Anwendung dargestellt werden, verantwortlich ist. Daher wurde auf jedem ViewController mindestens ein individueller HTTP-Request zum Server gesendet. Auch dies wird im folgendem Abschnitt zur Entwicklung genauer erläutert.

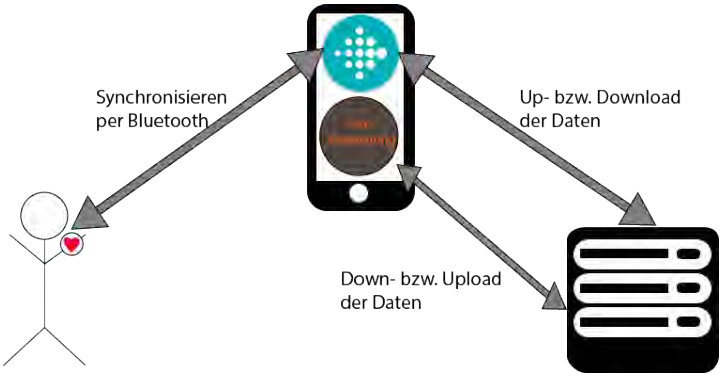



**Abbildung 5.2:** Architektur der Anwendung

## 5.3 Realisierung der Anwendung

In diesem Teilabschnitt wird nun der Aufbau der einzelnen ViewController aufgezeigt. Dies wird anhand einiger Screenshots zu der Anwendung veranschaulicht und erklärt. Außerdem werden jeweils deren Besonderheiten im Quellcode kurz erläutert.
Während der Implementierung wurde stets darauf geachtet, die Anwendung intuitiv und benutzerfreundlich zu gestalten. Dies spiegelt sich zum Beispiel im Farbschema wieder. Die komplette App ist dabei in einheitlichen Farben gehalten. So wurde jede Seite nach dem gleichen Prinzip aufgebaut. Auf einem schwarzen Hintergrund sind weitere „Container-Views“ platziert, welche wiederum die weiteren UI-Elemente beinhalten. Durch die Abgrenzung der Container ergibt sich ein übersichtliches Gesamtbild, durch welches die verschiedenen Menüpunkte schnell erkennbar sind.
Um Überschriften hervorzuheben sind diese in einem ausfälligen orange eingefärbt, wohingegen die restlichen *Labels* weiß sind.
Da die Seiten auch vom eigentlichen Aufbau sehr ähnlich sind, muss sich der Benutzer nicht auf jeder Seite neu orientieren. So ist die Anwendung sehr einfach zu Bedienen und benötigt keine Lernphase für die Benutzung.
Außerdem wurde zur Navigation durch die jeweiligen Menüs der iOS interne *NavigationController* verwendet. Dieser befindet sich immer am oberen Rand des Displays und zeigt dort stets an in welchem Menü sich der Benutzer gerade befindet. Zusätzlich kann dieser mit mehreren *NavigationBarButtons* bestückt werden. Nach dem iOS-Standard befindet sich hier auf der linken Seite jeweils der **zurück**-Button. Durch eine „Wisch-Geste“ nach rechts kann dessen Funktion jedoch ebenfalls realisiert werden.
Der Button auf der rechten Seite hingegen kann dabei für die unterschiedlichsten Funktionen verwendet werden. Häufig werden hier **Hinzufügen**-, **Löschen**- oder **Teilen**-Buttons angelegt.

Aufgrund dieser individuellen Belegung, wird auf dessen Funktionalität jeweils in der zugehörigen Beschreibung der Seite genauer eingegangen.

### 5.3.1 Login

Im folgenden Abschnitt wird nun zunächst der Login erläutert. Dessen Aufbau ist jedoch sehr einfach gehalten, da der eigentliche Login, nach Vorschriften von Fitbit, im Browser stattfinden muss. Anwendungen die diese Bedingung ignorieren, werden von Fitbit nicht akzeptiert und gesperrt.
Daher besteht die Login-Seite im wesentlichen nur aus vier Elementen. Einer Überschrift mit dem Namen der Anwendung, ein *Label*, welches den Ladezustand des Systems ausgibt und zuletzt zwei *Buttons*. Der obere Button dient dabei zur Bestätigung des Accounts, wohingegen der zweite Button für den Login zuständig ist.
Öffnet der Benutzer die Anwendung zum ersten Mal, ist der „Fortsetzen"-Button inaktiv, da sich der Benutzer zunächst anmelden muss. Dies ist durch den unteren *Button* möglich, welcher den Browser aufruft und somit die Anmeldung startet.(Siehe Abbildung 5.3, rechts). Hier findet nun der Login zu Fitbit statt. Mit Abschluss dieser Anmeldung gelangt man direkt in die Anwendung auf den nun folgenden ViewController, welcher das Dashboard repräsentiert. Für den Prozess des Logins wurde, wie bereits erwähnt, die Bibliothek *SampleBit* verwendet. Diese steuert den Ablauf der Anmeldung und liest nach einem erfolgreichem Abschluss die Daten aus.

**Listing 5.1:** Anmeldung mit OAuth

```
1  //Login:
2  guard let url = URL(string:
3  "https://www.fitbit.com/oauth2/authorize?response_type=token&
       client_id="+clientID+"&redirect_uri="+AuthenticationController.
       redirectURI+"&scope="+defaultScope+"&expires_in=604800")
4  let authorizationViewController = SFSafariViewController(url: url
       )
5  authorizationViewController.delegate = self
6  authorizationVC = authorizationViewController
7  viewController.present(authorizationViewController, animated:
       true)
8  //Callback:
9  if let token = AuthenticationController.extractToken(notification
       , key:
10 "#access_token") {
11   NSLog("You have successfully authorized")
12   [...]
13   Data().saveToken(tokenURL: token)
14   self?.delegate?.authorizationDidFinish(success)
15 } else {
16   print("There was an error")
17 }
```

Das Listing 5.1 zeigt wie dieser Prozess aufgerufen werden muss.
Zunächst wird in der Login-Methode die passende URL für die Anmeldung erstellt, welche anschließend über den Safaribrowser aufgerufen wird.
Mit Abschluss der Anmeldung wird der Code der Callback-Methode ausgeführt. Dabei wird überprüft, ob der Vorgang erfolgreich abgeschlossen wurde und daraufhin wird gegebenenfalls ein *Token* gespeichert. Der Token ist zum Einen der Indikator für einen erfolgreichen Login und zum Anderen wird dieser auch für die weitere Authentifizierung während der Benutzung der Anwendung verwendet. Dieser muss bei jedem Request mitgesendet werden, um den Zugriff auf die Datensätze sicherzustellen.
Da sich die Anwendung die Anmeldedaten speichert, kann dieser Schritt beim erneuten Öffnen jedoch umgangen werden. Nun wird zunächst überprüft, ob die Anmeldedaten noch gültig sind, da diese nach einem bestimmten Zeitraum von Fitbit deaktiviert werden. Sollten diese ihre Gültigkeit noch nicht verloren haben, kann man über den ersten *Button* direkt, ohne Anmeldung, auf das Dashboard gelangen. Zusätzlich zeigt der *Button* dabei den Name des Benutzers an, um Verwechslungen zu vermeiden.

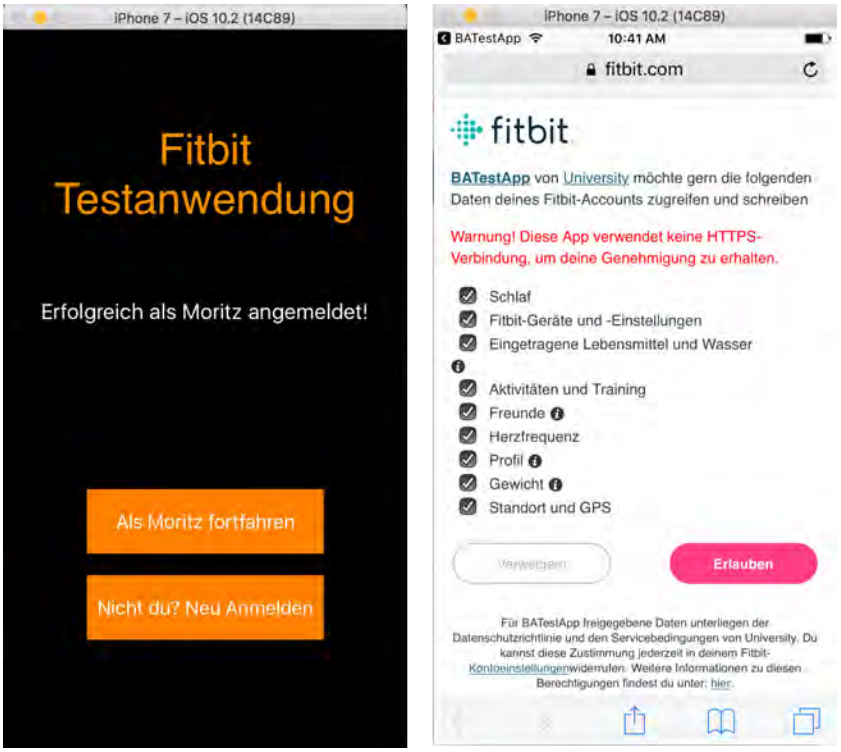



**Abbildung 5.3:** ViewController bezüglich dem Login. Links: Erfolgreiche Anmeldung. Rechts: Login über den Browser.

### 5.3.2 Dashboard

Nach dem erfolgreichem Login gelangt man auf das Dashboard. Dies ist durch die Abtrennung der rechteckigen Views sehr einfach und übersichtlich gehalten. So sieht man sofort auf den ersten Blick, welche Elemente zusammen gehören und kann alle Informationen ablesen, die von Interesse sind.

Wie man in Abbildung 5.4 erkennen kann, wird das Dashboard in vier Bereiche eingeteilt. Diese grenzen sich durch die eben erwähnten Container voneinander ab. Dabei besitzt jede Containerview weitere UI-Elemente, welche den Inhalt dieser repräsentieren. So beschreibt die Überschrift, die durch die Farbe orange und die Größe etwas hervorsticht, den jeweiligen Menüpunkt. Des weiteren besitzen die Container ein *Label*, welches die jeweiligen Daten anzeigt und ein zweites, das den Zeitpunkt der letzten Messung ausgibt. Zusätzlich besitzt die oberste View, welche zuständig für die Schritte ist, eine *ProgressBar*. Hierfür werden die Anzahl der Schritte im Verhältnis zu dem vom Benutzer eingetragen Tagesziel veranschaulicht. Hat man sein Ziel erreicht, ändert sich sowohl die Farbe, als auch das Label zu der *ProgressBar*. Gleichzeitig dienen die beschriebenen *Views* als *Buttons* um eine Verlinkung zu dem jeweiligen Menüpunkt zu erstellen. Hierfür wurden diese mit einem *TapGestureRecognizer* versehen, da Views ohne weitere Einstellungen nicht „klickbar“ sind. Durch das Betätigen des *Buttons*, gelangt man zu dem entsprechenden ViewController.
Am oberen Rand des Displays kann man die *NavigationBar* sehen, welche für die Übersicht und einige Funktionen zuständig ist. Daher zeigt diese den Titel der Seite an und besitzt an dieser Stelle einen *Button* auf der rechten Seite, welcher für das Abmelden des Benutzers zuständig ist. Wird dieser gedrückt, erscheint ein *Alert*-Fenster, in welchem der Nutzer bestätigen muss, dass dieser sich abmelden will. Mit Abschluss dieser Aktion, landet der Benutzer erneut auf der Loginseite.
Das wichtigste an dem Dashboard sind jedoch die *HTTP-Requests*, welche dafür sorgen, dass der ViewController mit Daten befüllt wird. Dazu werden in der *ViewDidLoad*-Methode die bei allen iOS-ViewControllern beim Erscheinen aufgerufen wird, die Requests erstellt und abgesendet. Diese müssen wiederum aus dem *JSON*-Format in einen, für das Programm lesbaren, Datentypen umgewandelt werden.

**Listing 5.2:** Auslesen einer JSON-Datei

```
1  case "heart":
2  if let hr = dict["activities-heart-intraday"] as?
3  [String: AnyObject]{
4          let hrJSON = hr["dataset"] as! [[String: AnyObject]]
5          for event in hrJSON{
6                  self.timeAr.append(event["time"] as! String)
7                  self.valueAr.append(event["value"] as! Double)
8          }
9          if valueAr.count != 0 {
10                 heartLbl.text = String(describing:(valueAr.last!))
11                 heartLbl2.text = "Letzter Eintrag " +
12                 String(describing: ((timeAr.last)!)) + " Uhr"
13         }else{
14                 heartLbl.text = "Keine Daten vorhanden"
15                 heartLbl2.text = "Keine Daten vorhanden"
16         }
17 }
```

Der dargestellte Code-Abschnitt zeigt diesen Prozess für die Daten der Herzfrequenz. Dieser Teilabschnitt befindet sich in einer Methode, welche das zu entpackende Objekt und ein Schlüsselwort zur Identifikation des *JSON*-Objektes enthält. In diesem Fall ist das Schlüssel-

wort „heart“. In Zeile 3 wird somit zunächst durch das Erstellen eines Optionalen Datentyps mit Hilfe von „if let [...]“, sichergestellt, dass das Objekt nicht leer ist und es zu keinen Fehlern führen kann. Dies sorgt dafür, dass der folgende Teil nur bei Erfüllung der Bedingung ausgeführt wird.
Anschließend wird das *JSON*-Objekt mithilfe einer *For-Schleife* durchgelaufen. Während dieses Prozesses werden die Daten per Key-Value Abfrage ausgelesen, was zum Beispiel in Zeile 6 und 7 beobachtet werden kann. Hier wird jeweils nach den Schlüsselworten **time** und **value** gesucht.
Zuletzt muss das zugehörige *Label* noch mit den neuen Daten beschrieben werden. Auch hier wird dabei noch einmal überprüft, ob das Objekt Daten enthält. Sollte dies nicht der Fall sein, wird dies mit einer entsprechenden Nachricht für den Benutzer über das *Label* ausgegeben. Beim Beschreiben des *Labels* in Zeile 12 und 13 fällt auf, dass nur der letzte Wert des Arrays verwendet wird. Das hängt damit zusammen, dass es bei den Requests nicht möglich ist nur den letzten gemessenen Wert auszugeben. Daher muss immer ein größerer Zeitabschnitt untersucht werden, wovon nur der letzte Wert schlussendlich verwendet werden kann. Auf dieses Problem wird in der Evaluation im Abschnitt 7.1.5 genauer eingegangen.


**Abbildung 5.4:** Ansichten des ViewControllers bezüglich des Dashboards.

### 5.3.3 ViewController - Schritte

Dieser ViewController besteht im wesentlichen aus zwei Teilen. Zunächst kann man im oberen Bereich ein Balkendiagramm sehen, welches die Anzahl der Schritte darstellt. Zu diesem gehört außerdem die *UISegmentedControl*, mit welcher man den Zeitraum der angezeigten Daten einstellen kann. Unter den Punkten „Woche“ und „Monat“ kann die Anzahl der Schritte an den jeweiligen Tagen eingesehen werden. In der Auswahl „Jahr“ hingehen, wird der Durchschnitt von jedem der letzten 12 Monate veranschaulicht, weshalb sich an dieser Stelle zusätzlich die Achsenbeschriftung und die Legende ändert.
Dafür benötigt jeder Zeitabschnitt des Diagramms eine eigene Aufbereitung der Daten. Um den Ablauf dieses Prozesses aufzuzeigen, ist im folgenden Quellcode die Bearbeitung der Daten zum Abschnitt „Jahr“ zu sehen.
Wie im Listing 5.3 zu sehen ist, war die Verarbeitung dieser Daten der wohl aufwändigste Teil dieser Seite. Dies ist unter Anderem auf die Einschränkung der vorgegeben Requests zurückzuführen. Da es keine Anfrage für den durchschnittlichen Wert eines bestimmten Zeitraumes an die Schnittstelle gibt, musste dies nachträglich im Code berechnet werden.

**Listing 5.3:** Umwandlung der Daten für die Jahresübersicht

```
1  var j = 0
2  for i in 0..<13{
3    var sum = 0
4    var counter = 0
5    let temp = Int(values[j].1.components(separatedBy: "-")[1])!
6    while newMonth {
7      if j >= values.count{
8        yVals.append(sum / counter)
9        break
10     }
11     if Int(values[j].1.components(separatedBy: "-")[1])! != temp{
12       if i != 1{
13         yVals.append(sum / counter)
14       }
15       newMonth = false
16     }
17     sum += values[j].0
18     j += 1
19     counter += 1
20   }
21   newMonth = true
22 }
```

Dazu wurde zunächst der Durchschnitt für jeden Monat errechnet. Um dies zu realisieren, durchläuft eine *For-Schleife* zunächst jeden einzelnen Monat. Diese werden in Zeile 5 mithilfe des Datums erkannt. In der *While-Schleife* ab Zeile 6 werden die Daten des Monats dann summiert und anschließend, wenn der letzte Tag des Monats erreicht ist, dem Datensatz „yVals“, welche für das Diagramm verwendet wird, hinzugefügt. Dabei wird der Wert noch durch die Anzahl der Tage geteilt, um den richtigen Durchschnitt zu erhalten.

Unter dem Diagramm ist auf dieser Seite noch eine *TableView* zu sehen. Diese veranschaulicht alle gemessenen Daten. So ist es möglich, die genaue Anzahl der Schritte, von jedem einzelnen Tag einzusehen. In der Abbildung 5.5, wird diese Seite dargestellt.

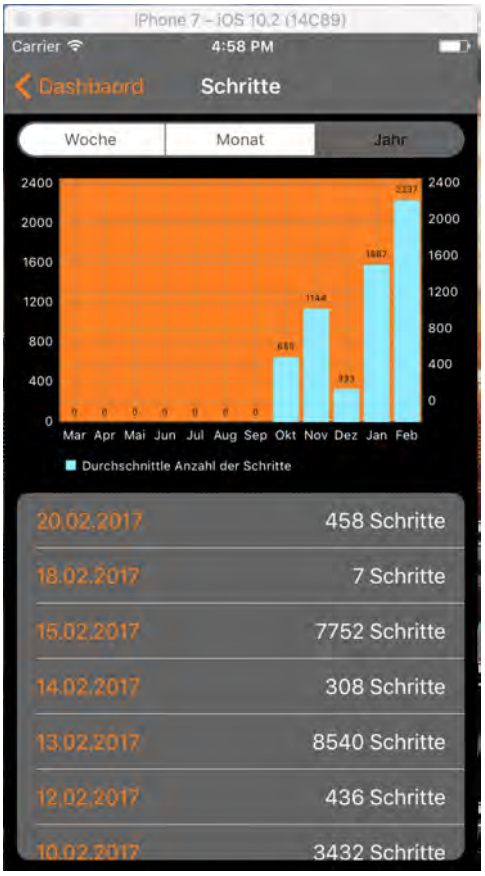



**Abbildung 5.5:** Ansicht des ViewControllers bezüglich der Schritte

### 5.3.4 ViewController - Herzfrequenz

Wie die Dialogstruktur zeigt, beinhaltet der Menüpunkt Herzfrequenz zwei ViewController. Dabei wird zunächst die Übersicht aufgezeigt und im Anschluss die Detailansicht. Dies entspricht auch dem Ablauf während der Verwendung der Anwendung.

**Herzfrequenz - Übersicht** Auch dieser ViewController wurde nach demselben Prinzip aufgebaut. Durch die rechteckigen *Views* grenzen sich auch hier klar drei Bereiche ab. In dem oberen Bereich ist erneut ein Diagramm zu sehen, welches nun, als Liniendiagramm, die Herzfrequenzen darstellt. Zugehörig zu diesem ist wieder die *UISegmentedControl*. Hier ist, da bei der Herzfrequenz dauerhaft Daten gesammelt werden, zusätzlich der Punkt „Stunde“ verfügbar. Unter den Punkten „Stunde“ und „Tag“ werden alle gemessenen Herzfrequenzen in diesem Zeitraum dargestellt, während unter den beiden anderen Punkten der Ruhepuls veranschaulicht wird. Ursprünglich sollte hier auch der Durschnittswert angezeigt werden, was aufgrund des verfügbaren Angebots an **REST-Anfragen** jedoch kaum möglich war. Dieses Problem wird im Abschnitt 7.1.5 genauer aufgeführt.
Um die unterschiedlichen Zeitabschnitte zu erhalten, reicht jedoch nicht nur ein einzelner Request aus, da Anfragen für einen einzelnen Tag und über einen langen Zeitraum unterschiedliche Datensätze liefern. Auch dieses Thema wird im Abschnitt 7.1.4 evaluiert.

**Listing 5.4:** Erstellung der verschiedenen Requests

```
1  var startTimeM = components.minute!
2  var startTimeH = components.hour! - 1
3  let endTimeM = components.minute!
4  let endTimeH = components.hour!
5  let timeString = String(format: "%02d", startTimeH) + ":" +
       String(format: "%02d",
6  startTimeM) + "/" + String(format: "%02d", endTimeH) + ":" +
       String(format:
7  "%02d", endTimeM)
8  scriptURL =
9  "https://api.fitbit.com/1/user/-/activities/heart/date/today/1d/1
       min/time/"+timeString+".json"
10 getRequests(scriptURL: scriptURL, unbox: "hour")
11 let endDate = components.year! - 1
12 let dateString = String(endDate)+"-"+String(format:
13 "%02d",components.month!)+"-"+String(format: "%02d",components.
       day!)
14 scriptURL = "https://api.fitbit.com/1/user/-/activities/heart/
       date/today/" +
15 dateString+".json"
16 getRequests(scriptURL: scriptURL, unbox: "year")
17 scriptURL =
18 "https://api.fitbit.com/1/user/-/activities/heart/date/today/1d/1
       min.json"
19 getRequests(scriptURL: scriptURL, unbox: "day")
```

In dem Listing 5.4 wird der Prozess zur Erstellung der Zeitabschnitte ausgeführt. Zunächst wurden von Zeile 1 bis 4 mehrere Variablen, welche die aktuelle Zeit ausgeben, erstellt. Anschließend müssen diese Variablen in einen String umgewandelt werden, der von Fitbit akzeptiert wird. Dieser wird daraufhin wie in Zeile 9 in die URL eingebaut. Über die „getRequest-Methode“ wird die Anfrage gesendet und wie auf dem Dashboard verarbeitet.
Ähnlich wird dabei das Jahr angefragt, wobei hier natürlich nicht eine Stunde, sondern ein ganzes Jahr abgezogen werden muss.
Anders funktioniert jedoch der Request für die aktuellen Tag, da Fitbit hier die Möglichkeit bietet, direkt in die URL das Stichwort „today“ einzubauen. So spart man sich das Erstellen eines Datums über den Code.
Wie man also sehen kann, werden alleine um diese Seite aufzubauen 3 Anfragen benötigt.
Unter dem Diagramm ist die zuletzt gemessene Herzfrequenz zu sehen. Diese kann man mit Hilfe des „Aktualisieren-Buttons“ erneut abgefragt werden. Da die Herzfrequenz jedoch nur ca. alle 10 Minuten auf dem Server aktualisiert wird, wird dabei zusätzlich der Zeitpunkt der letzten Messung ausgegeben.
Im unteren Bereich werden erneut alle Daten in einer *TableView* veranschaulicht. Dabei wird das Datum mit dem entsprechenden Wert dargestellt. Über diese Tabelle erreicht man außerdem per Klick auf einen Eintrag die Detailansicht, welche den ausgewählten Tag darstellt.
Die komplette Ansicht ist auf Abbildung 5.6 links zu sehen.

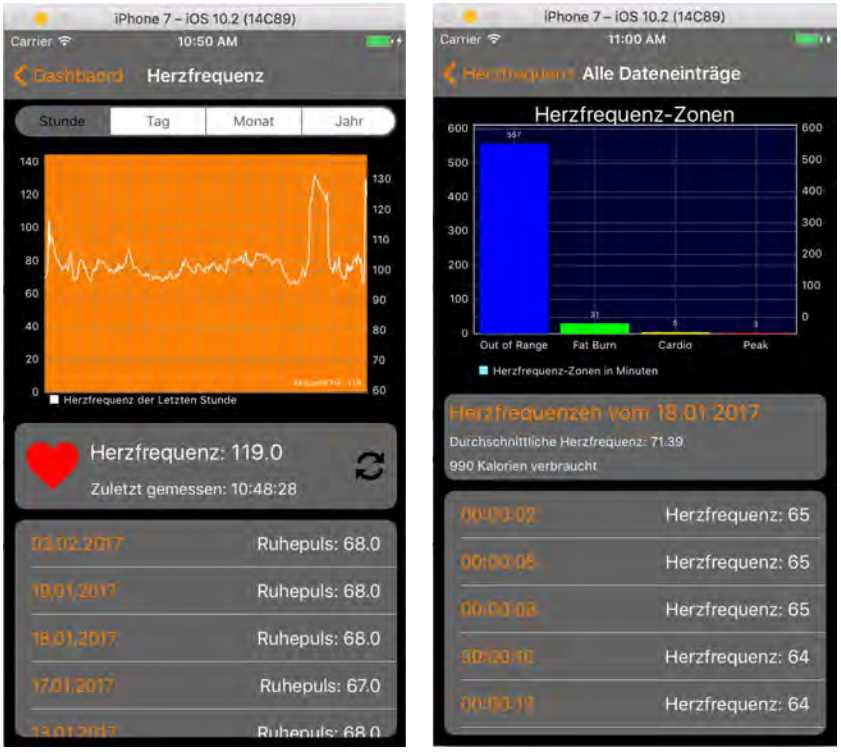



**Abbildung 5.6:** Ansicht der Herzfrequenz-Seiten. Links: Übersicht, Rechts: Detailansicht

**Herzfrequenz - Detailansicht** Auf dieser Seite sind nun die genauen Details des ausgewählten Tages zu sehen. Angefangen mit einem Balkendiagramm im oberen Bereich. In diesem werden die verschiedenen, von Fitbit definierten, Herzfrequenzzonen des Benutzers dargestellt.

Angezeigt werden diese in Minuten pro Zone, um diese anschaulich auf einen Blick zu visualisieren. In dem Container darunter wird außerdem die durchschnittliche Herzfrequenz und die Anzahl der verbrauchten Kalorien angezeigt. Zuletzt wird eine ausführliche Tabelle mit jedem Eintrag, den der Tracker an diesem Tag aufgezeichnet hat, aufgelistet. So hat man auf alle gemessenen Daten Zugriff. Diese Detailansicht ist jedoch immer erst am Folgetag verfügbar, weshalb die Herzfrequenz über den gesamten Tag erst verspätet einsehbar ist.

**Listing 5.5:** Erstellung eines Requests zu einem bestimmten Tag

```
1 let scriptURL =
2 "https://api.fitbit.com/1/user/-/activities/heart/date/"+date+"/1
    d/1sec/time/00:00/23:59.json"
```

Wie schon erwähnt und in der Abbildung 5.6 zu sehen, kann nun die durchschnittliche Herzfrequenz, im Gegensatz zur Übersichtsseite angezeigt werden. Dies liegt daran, dass hier erneut ein weiterer Request mit dem entsprechenden Tag erstellt werden muss. Dieser kann, da es sich hier nur um einen einzelnen Tag handelt, auch den Durchschnittswert und weitere Details anzeigen.
Das Listing 5.5 bildet hier die URL zu dem eben genannten Request ab. Dieser unterscheidet sich, wie auch der Request für die Herzfrequenz der letzten Stunde, aus dem Listing 5.4, grundsätzlich von den Anfragen über längere Zeiträume. Dies ist auf die Trennung der Zugriffsrechte von Fitbit zurückzuführen, was im Abschnitt 7.1.4 genauer untersucht wird.

### 5.3.5 ViewController - Schlafaufzeichnung

Zum Menüpunkt **Schlafaufzeichnung** gehören insgesamt drei ViewController. Zu der Übersicht und Detailansicht, welche schon bei dem Menüpunkt **Herzfrequenz** zu finden sind, kommt eine weitere Ansicht, die für das manuelle Hinzufügen eines Schlafes zuständig ist.

**Schlafaufzeichnung - Übersicht** Auch dieser ViewController unterscheidet sich vom Aufbau im wesentlichen kaum von den übrigen Übersichten. Auf der oberen Hälfte der Seite, ist ein Balkendiagramm zu sehen, das die geschlafenen Stunden anzeigt. Dieses unterscheidet sich jedoch farblich von den anderen Diagrammen, da hier statt einem Orange ein dunkles Lila verwendet wurde.
Unter diesem Diagramm wird erneut eine Tabelle mit allen Schlafaufzeichnungen und dem zugehörigen Datum aufgelistet. Wie bei der Herzfrequenz, gelangt man auch hier durch einen Klick auf einen beliebigen Eintrag auf dessen Detailseite. Diese Tabelle unterscheidet sich jedoch in ihrer Funktionalität von den bisher genannten Tabellen, da hier zusätzlich Einträge aus der Liste entfernt werden können. Mit einer Wisch-Geste nach links erscheint in der ausgewählten Zeile, wie es bei iOS üblich ist, ein *Löschen-Button*. Über diesen kann ein Eintrag nun dauerhaft aus der Datenbank entfernt werden, falls zum Beispiel ein Schlaf falsch oder ungenau aufgezeichnet wurde.
Außerdem fällt in der *NavigationBar* auf, dass im rechten Eck nun ein *Plus-Button* zu sehen ist, über welchen man den ViewController für das manuelle Hinzufügen eines Schlafes erreicht. Die komplette Ansicht ist in Abbildung 5.7 auf dem linken Bild dargestellt.


**Abbildung 5.7:** Ansichten der ViewController bezüglich des Schlafes. Links: Übersicht. Rechts: Detailansicht

**Schlafaufzeichnung - Detailansicht** In dieser Detailansicht muss zunächst zwischen zwei verschieden Fällen unterschieden werden. Da der Schlaf manuell, aber auch automatisch über den Tracker hinzugefügt werden kann, entstehen dabei verschiedene Datensätze. Der automatisch erkannte Schlaf ist dabei deutlich umfangreicher. Hier wird der Schlaf zu jeder Minute überwacht, wohingegen bei dem manuellen Eintrag nur die Start- und Endzeit eingegeben werden kann. So zeigt das Diagramm bei der automatisch erkannten Aufzeichnung die Dauer der verschiedenen Schlafqualitäten an. Die Qualität wird dabei in drei Zustände unterteilt, welche in Minuten als Balkendiagramm dargestellt werden. Bei manuellen Aufzeichnungen steht anstelle des Diagramms jedoch nur eine Meldung, welche den Nutzer auf die fehlenden Daten hinweist. Anders ist es bei den beiden Folgenden *ViewContainern*. Hier wird zunächst die genaue Dauer angezeigt, welche nebenbei durch eine *ProgressBar* veranschaulicht wird. Auch diese verändert, abhängig vom erreichen des „Schlafziels“, die Farbe und die zugehörige Beschreibung. Auf diese *View* folgen die genauen Einschlaf- und Aufwachzeiten. Zuletzt wird die Qualität des Schlafes aufgelistet. Auch diese kann nur bei einem automatisch erkannten Schlaf angezeigt werden. Ist dies der Fall, werden mehrere Indikatoren für die Qualität aufgezeigt. Zunächst die Effizienz, welche den Prozentsatz der Minuten im Tiefschlaf anzeigt. Daraufhin folgt eine Angabe, über die Anzahl, welche angibt, wie oft man aufgewacht ist. Woraufhin zuletzt angeben wird wie oft und wie lange man sich im Zustand „Ruhelos“ befunden hat.
Im Gegensatz dazu wird auch hier der Nutzer bei einem manuellem Eintrag durch ein *Label* darauf hingewiesen, dass diese Daten nicht verfügbar sind.
Außerdem kann der angezeigte Schlaf über den *Löschen-Button* in der *NavigationBar* entfernt werden. Nach Bestätigung dieser Aktion gelangt man somit wieder auf die nun aktualisierte

Übersicht. Auf der Abbildung 5.7 im Bild rechts ist diese Seite mit einem manuell aufgezeichnetem Schlaf zu sehen.
Die Unterscheidung zwischen den beiden verschiedenen Schlaftypen konnte jedoch nicht über ein Attribut im Datensatz erkannt werden. Hier liefert Fitbit keine Unterscheidung dieser Objekte. Die Daten, welche bei einem manuell hinzugefügten Schlaf nicht aufgezeichnet werden, befüllt Fitbit trotzdem mit Werten. So wird zum Beispiel die Qualität des Schlafes immer auf 100 Prozent gesetzt, was einen perfekten Schlaf, ohne Aufwachen beschreiben würde. Um so die Verfälschung der Daten und des Diagramms zu umgehen, mussten die Schlaftypen auch ohne ein passendes Attribut unterschieden werden.
Wie das Listing 5.6 in der Zeile 1 zeigt, wurden hierfür die Effizienz, die Minuten bis zum Einschlafen und der Zähler der Ruhelosigkeit untersucht. Da sich diese Werte bei einem selbst eingetragenem Schlaf nie unterscheiden, kann man so sicherstellen, um welchen Typ es sich handelt. Liegt zum Beispiel die Effizienz unter 100 Prozent, wie es bei einem „echten“ Schlaf üblich ist, wird der automatisch erkannte Schlaf ausgewählt.

**Listing 5.6:** Unterscheidung der verschieden Schlaf-Typen

```
1  if _efficiency == 100 && _minutesToFallAsleep == 0 &&
       _restlessCount == 0{
2          //manuell
3          notRealLbl.isHidden = false
4          barView.noDataText = "Diagramm nur bei gemessenen
5          Schlaf-Aufzeichnungen moeglich"
6          barView.noDataTextColor = .white
7          barView.backgroundColor = SleepViewController().
       hexStringToUIColor(hex:
8          "#000033")
9          [...]
10 }else{
11         //automatisch
12         [...]
13         setChart(data: chartData)
14 }
```

**Schlafaufzeichnung - Hinzufügen** Aufgrund der besonderen Funktionalität dieses View-Controllers sticht dieser im Aufbau des Layouts etwas hervor. Da hier keine Daten veranschaulicht werden müssen, besitzt diese Seite weder eine Tabelle noch ein Diagramm. Stattdessen sind im oberen Bereich zwei *Views* zu sehen, welche dabei den eingestellten Zeitpunkt des Einschlafens und Aufwachens anzeigen.
Der Standard „Einschlaf-Zeitpunk“, welcher beim Aufruf der Seite angezeigt wird, ist dabei immer die aktuelle Uhrzeit mit dem Datum des Vortages. Dies hat den Grund, dass Fitbit es nicht erlaubt, einen Schlaf in der Zukunft einzutragen. Somit ist das Datum des Vortages der erste mögliche, akzeptierte Eintrag. Am unteren Bildschirmrand ist ein *PickerView* zu sehen. Dieser zeigt sowohl das Datum als auch die Uhrzeit an. Über diesen können somit die gewünschten

Zeitpunkte eingestellt werden.
Beim Aufruf der Seite ist zunächst die Auswahl für den Einschlaf-Zeitpunkt aktiv. Dies wird dem Benutzer durch die Hervorhebung bestimmter Merkmale der *View* vermittelt. So werden zum Einen die *Labels* des aktiven Fensters eingefärbt. Wohingegen die *Labels* des inaktiven Fensters schwarz bleiben. Außerdem wird das Letztere durch einen niedrigeren *Alpha-Wert* etwas abgedunkelt und ausgegraut. *Alpha-Werte* sogen bei Grafiken für die Durchlässigkeit der Objekte. Dabei steht der Maximalwert (1,0) für absolut undurchlässig, wohingegen der Minimalwert (0,0) das Objekt unsichtbar macht.
Durch einen Klick auf die gewünschte *View* kann der Benutzer den Fokus wechseln. Außerdem springt der *Picker* bei jedem Wechsel zu dem entsprechenden Datum und dessen Uhrzeit. So sind beliebig viele Wechsel und somit Anpassungen der beiden Zeitpunkte möglich. Durch das einfache und minimalistische Design kommt es hier kaum zu Verwechslungen oder Fehlern.
Dies kann jedoch nie vollständig verhindert werden, weshalb in dem freien Zwischenraum eventuelle Fehler bei der Auswahl der Daten angezeigt werden.
Sollte der Benutzer zum Beispiel den Einschlaf-Zeitpunkt zeitlich hinter den Aufwach-Zeitpunkt legen, wird dies durch ein *Label* angezeigt. Um fehlerhafte HTTP-Posts zu vermeiden, wird der *Button*, welcher für den Upload zuständig ist nur aktiviert, wenn keine Fehler erkannt werden. Die hierfür erbrachten Vorkehrungen sind dabei in dem Listing 5.7 zu erkennen. In Zeile 1 wird dabei zunächst bei der Erstellung des *Pickers* ein maximales Datum gesetzt, welches nicht überschritten werden kann. In diesem Fall ist dies das aktuelle Datum mit der aktuellen Uhrzeit, welches den letzten akzeptieren Zeitpunkt darstellt.

**Listing 5.7:** Vorkehrungen zur Vermeidung von fehlerhaften Uploads.

```
1  timePicker.maximumDate = end as Date
2  [...]
3  if start.timeIntervalSince1970 >= end.timeIntervalSince1970{
4          errorLbl.text = "Der Endzeitpunkt muss vor dem Startpunkt
       sein"
5          startSmallerEnd = false
6  }else{
7          startSmallerEnd = true
8  }
9  if Int(end.timeIntervalSince(start as Date) * 1000) < 600000 && !
       startSmallerEnd {
10         errorLbl.text = "Der Schlaf ist zu kurz"
11         duration = false
12 }else{
13         duration = true
14 }
15
16 if duration && startSmallerEnd{
17         errorLbl.text = ""
18         navigationItem.rightBarButtonItem?.isEnabled = true
19 }
```

Die darauf folgenden Abfragen werden bei jeder neuen Zeiteinstellung untersucht. Dabei wird zunächst überprüft, ob das Startdatum weiter zurück liegt als das Enddatum. Daraufhin wird

geprüft ob der Schlaf länger als 600000 Millisekunden ist, was 10 Minuten entspricht. Diese Vorgabe wird von Fitbit gestellt, um einen Schlaf darzustellen. Nur wenn diese beiden Punkte erfüllt sind, wird der *Upload-Button* freigegeben, welcher wiederum den Schlaf auf den Fitbit-Server lädt.
Nach dem Klick auf diesen Button und der Bestätigung des Uploads, wird diese Aktion durchgeführt und der Benutzer landet auf der aktualisierten Übersicht.
Auf der Abbildung 5.8 sind diese beiden Seiten zu sehen. Die linke Ansicht zeigt den View-Controller während der Auswahl des Aufwach-Zeitpunktes, wohingegen das rechte Bild den erfolgreichen Upload durch einen Alert bekannt gibt.

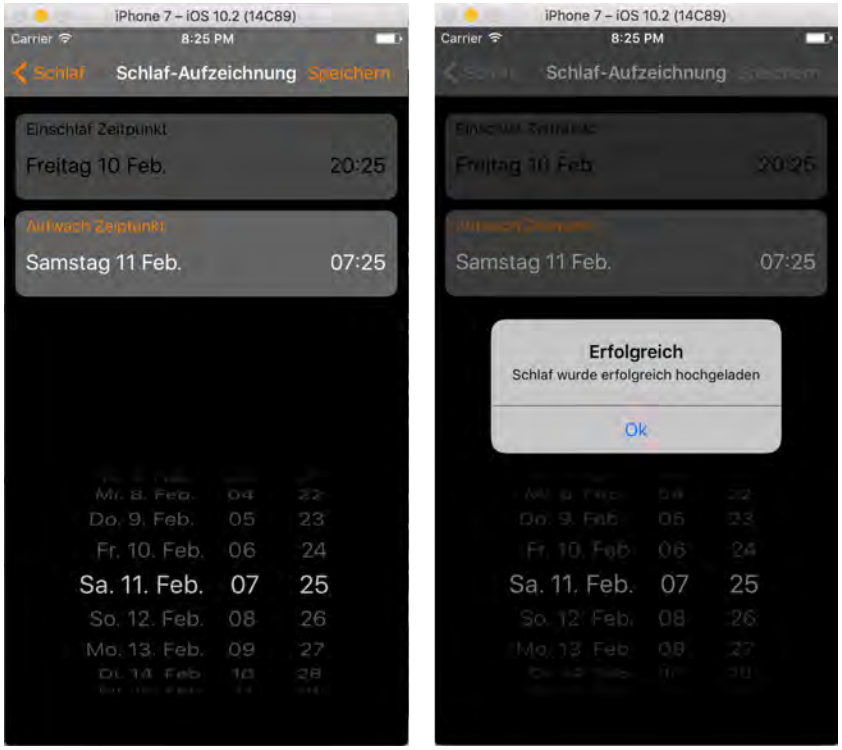



**Abbildung 5.8:** ViewController zur manuellen Aufzeichnung eines Schlafes. Links: Auswahl der Zeiten. Rechts: Erfolgreicher Upload des Schlafes

### 5.3.6 ViewController - Aktivitäten

Dieser Teilabschnitt behandelt den letzten Menüpunkt der Anwendung, welcher die Daten der Aktivitäten zeigt. Da die Datensätze der Aktivitäten deutlich komplexer und umfangreicher als die der vorherigen Punkte sind, unterscheidet sich der Aufbau dieses Menüs ein wenig. Aufgeteilt wird dies, wie in der Abbildung 5.1 der Dialogstruktur zu sehen ist, in zwei ViewController. Zunächst wird dabei erneut eine Übersicht angezeigt, über welche man schließlich zur Detailansicht gelangt.

**Aktivitäten - Übersicht** Hier fällt sofort der Unterschied dieses ViewControllers auf, da dieser lediglich eine Tabelle enthält welche alle Aktivitäten darstellt. Dies ist jedoch keine Standard-Tabelle, wie sie auf den bisherigen Seiten zu sehen war und in iOS verwendet wird, sondern eine eigens erstellte Tabelle. Im Gegensatz zur normalen Tabelle können hier beliebig viele Elemente mit einem eigenen Layout in den Zeilen platziert werden. Dazu müssen der Aufbau, und die zugehörige Zelle jedoch zunächst definiert werden. Das Listing 5.8 zeigt, etwas gekürzt, wie dies realisiert wird. Zunächst werden von Zeile 3 bis 6 die gewünschten UI-Elemente definiert, welche nun in der **override.init(...)**-Funktion platziert werden. Wie der Name der Methode vermuten lässt, wird hier die Standard-Zeile der iOS Tabelle überschrieben. Platziert werden die Elemente schlussendlich mit dem beschriebenen Framework *PureLayout*. Dies geschieht von Zeile 12 bis Zeile 16.

**Listing 5.8:** Erstellen einer eigenen Zelle

```
1  class CustomCellTableViewCell: UITableViewCell {
2
3          let distanceLbl = UILabel()
4          let timeLbl = UILabel()
5          let dateLbl = UILabel()
6          let nameLbl = UILabel()
7
8          [...]
9          override init(style: UITableViewCellStyle,
       reuseIdentifier: String?) {
10         super.init(style: style, reuseIdentifier: reuseIdentifier
       )
11         [...]
12         nameLbl.autoPinEdge(toSuperviewEdge: .left, withInset:
       5.0)
13         nameLbl.autoAlignAxis(toSuperviewAxis: .horizontal)
14         [...]
15         dateLbl.autoPinEdge(toSuperviewEdge: .right, withInset:
       5.0)
16         dateLbl.autoAlignAxis(toSuperviewAxis: .horizontal)
17         }
18 }
```

Außerdem wird auf dieser Übersicht, wie Abbildung 5.9 zeigt, kein Diagramm welches die letzten Datensätze visualisiert, dargestellt. Um also eine Aktivität genauer zu betrachten, muss hier ein Eintrag aus der Tabelle ausgewählt werden, welcher den Nutzer auf die passende Detailansicht führt. Zusätzlich ist es auch in dieser Tabelle möglich, Aktivitäten zu entfernen, was auf dem selben Prinzip der vorherigen Tabelle basiert. Nach Bestätigung wird der Eintrag dauerhaft aus der Tabelle und von dem Fitbitserver entfernt.

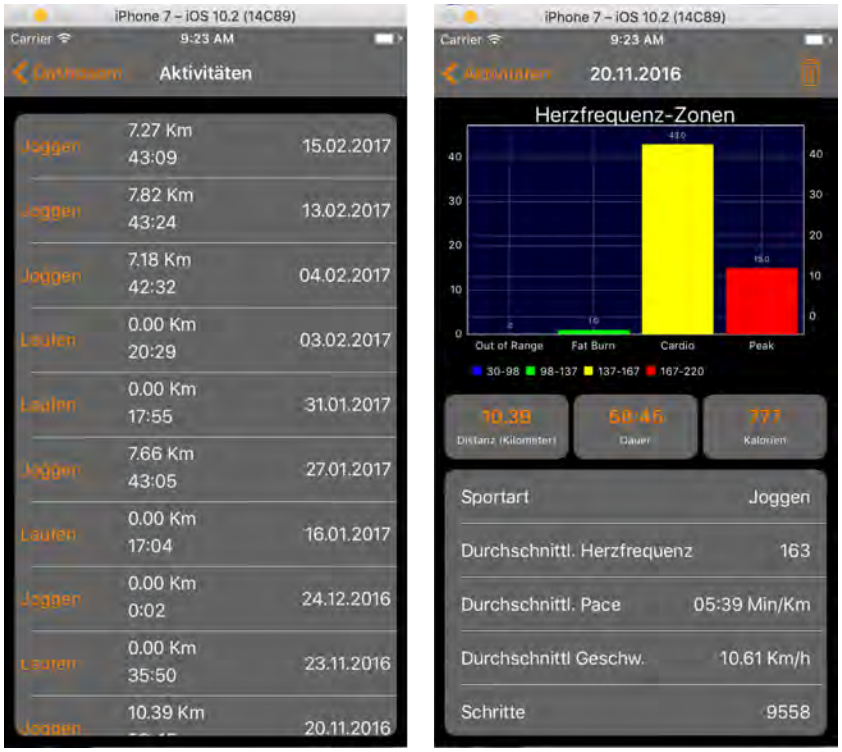



**Abbildung 5.9:** Übersicht der ViewController bezüglich der Aktivitäten. Links: Die Übersicht. Rechts: Die Detailansicht

**Aktivitäten - Detailansicht** Der Aufbau dieser Seite entspricht nun wieder dem bekannten Schema. Die Abbildung 5.9 rechts zeigt ein Beispiel dieses ViewControllers. Wie schon auf den vorherigen Ansichten, befindet sich im oberen Bereich der Seite das Diagramm, welches in diesem Fall ein Balkendiagramm ist, das die verschiedenen Herzfrequenzzonen visualisiert. Diese werden dabei auf den Zeitraum während der Aktivität beschränkt und teilen sich dabei erneut in vier Bereiche auf, welche man durch die farbliche Abtrennung deutlich erkennen kann. Auch hier wird die Zeit, wie üblich, in Minuten angegeben.
Darunter sind drei kleinere Container abgebildet, welche die wichtigsten Informationen einer Aktivität auf einen kurzen Blick darstellen. Die Werte sind dabei in orange eingefärbt, um diese zusätzlich hervorzuheben.
Der letzte Teil der Seite besteht erneut aus einer *Tabelle*. Hier werden die übrigen Daten, welche während der Aktivität aufgezeichnet werden, aufgelistet. Diese ist jedoch im Gegensatz zu den übrigen Tabellen nicht scroll- oder bearbeitbar, da diese eine feste Länge hat und lediglich zur Visualisierung der Werte dient.

Außerdem kann die Aktivität auch hier dauerhaft gelöscht werden. Über einen Klick auf den *NavigationBarButton* und dessen Bestätigung gelangt man somit wieder auf der aktualisierten Übersicht.

# 6 Anforderungsabgleich

In diesem Kapitel wird nun überprüft, ob die in Abschnitt 4 aufgestellten Anforderungen erfüllt wurden. Dazu werden erneut zunächst die funktionalen und anschließend nicht-funktionalen Anforderungen untersucht, ob diese erfüllt wurden.

## 6.1 Funktionale Anforderungen

Nun werden erneut die Anforderungen aus Kapitel 4.1 aufgelistet und mit der realisierten Testanwendung verglichen.

- **Native Anwendung**: Diese Anforderung wurde vollständig erfüllt, da die Anwendung nativ auf iOS realisiert wurde.
- **Serverkommunikation**: Diese Anforderung wurde erfüllt (siehe Kapitel 5).
- **Benutzerverwaltung**: Diese Anforderung wurde erfüllt (siehe Abschnitt 5.3.1).
- **Datenauswertung**: Diese Anforderungen wurden erfüllt (siehe Abschnitt 5.3.4 und 5.3.5)
- **Dynamische Requests**: Diese Anforderung wurde erfüllt, sofern der Benutzer den Bedingungen der OAuth-Autorisierung zustimmt.
- **Dynamische Auswertung**: Diese Anforderung wurde erfüllt (siehe Kapitel 5)
- **Diagramme**: Diese Anforderung wurde erfüllt (siehe Kapitel 5)
- **Anmelden**: Diese Anforderung wurde erfüllt (siehe Abschnitt 5.3.1)
- **Herzfrequenz**: Diese Anforderung wurde erfüllt (siehe Abschnitt 5.3.4, Detailseite)

## 6.2 Nicht-funktionale Anforderungen

In diesem Abschnitt werden die nicht-funktionalen Anforderungen erneut aufgelistet und ebenfalls mit der Testanwendung verglichen.

- **Zuverlässigkeit**: Diese Anforderung wurde erfüllt, ist jedoch von der Internetverbindung abhängig. Bei einer stabilen Verbindung, werden die Daten in kurzer Zeit heruntergeladen und verarbeitet. Ist diese jedoch instabil, kann es zu Verzögerungen führen.
- **Verfügbarkeit**: Diese Anforderung wurde erfüllt. Die Anwendung wurde mit Xcode und Swift für iOS 10 entwickelt.
- **Benutzerfreundlichkeit**: Diese Anforderung wurde erfüllt. Die UI-Elemente und der Aufbau der Anwendung halten sich dabei an das Design des Betriebsystems iOS.
- **Erwartungskornform**: Diese Anforderung wurde erfüllt, da jede Funktion die erwartete Aufgabe durchführt.
- **Korrektheit**: Diese Anforderung wurde erfüllt.

# 7 Evaluation des Ökosystems Fitbit

Aus den Erfahrungen der Entwicklung der Testanwendung wird nun das Ökosystem und vor allem die Entwicklungsumgebung um Fitbit genauer untersucht. Dabei werden sowohl die positiven als auch negativen Aspekte aufgezeigt, um zu veranschaulichen, wie gut dieses System mit der besonderen Architektur funktioniert, wie die Arbeit damit ist und ob es sinnvoll ist, eine Anwendung mit Anbindung an Fitbits Schnittstelle zu implementieren. Der wichtigste Teil hierbei ist erneut die Arbeit mit der Herzfrequenz. Dieser Punkt dient daher an einigen Stellen als Vorzeigebeispiel, welches stellvertretend für das gesamte System gilt. Außerdem müssen bei der Herzfrequenz einige Besonderheiten beachtet werden, auf welche im Folgendem ebenfalls genauer eingegangen wird.

## 7.1 Die Arbeit mit der Schnittstelle

Der Umgang mit der Schnittstelle und dessen Architektur stellt das zentrale Element bei der Entwicklung einer Anwendung dar, die auf den Daten Fitbits basiert. Der folgende Abschnitt behandelt alle Komponenten, mit welchen der Entwickler während der Realisierung konfrontiert wird.

### 7.1.1 Einstieg in das Projekt

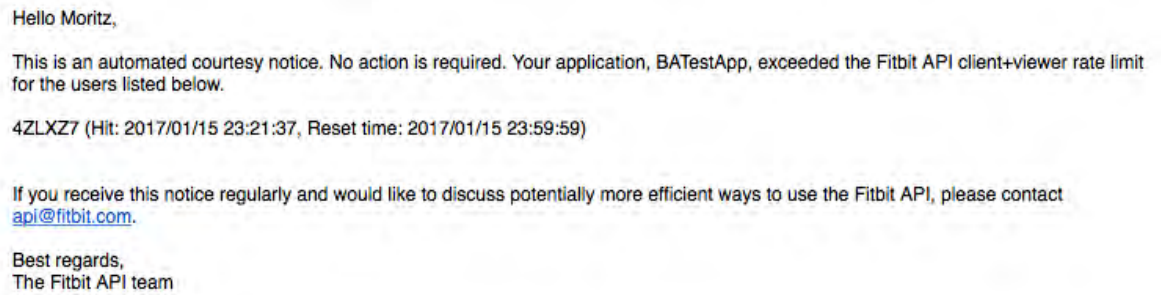
Fitbit gestaltet den Einstieg in ihre Schnittstelle, wie schon kurz im Abschnitt Grundlagen (2.2.1) angedeutet, sehr einfach. Auf der Developer-homepage [7] wird eine umfangreiche Dokumentation aufgelistet, welche jeden Schritt bis zum eigentlichen Beginn deutlich erklärt. Die einzelnen Schritte sind dabei übersichtlich gegliedert und erleichtern somit das Verständnis und den Ablauf zu Beginn des Projektes.
Neben dieser Dokumentation liefert Fitbit zur Veranschaulichung einige Referenzen, wie zum Beispiel einen Prototyp zum Ablauf des Logins, um bei komplizierteren Abschnitten für mehr

Klarheit zu sorgen.
Neben dem Einstieg in die Materie bietet die Dokumentation zusätzlich Beispiele zu den verschiedenen Requests, welche dabei ebenfalls in einzelne Kategorien unterteilt (Herzfrequenz, Schlaf, Aktivität, etc.) sind. Diese sind sehr ausführlich erklärt und werden mithilfe der URL des Requests und dessen Ausgabe veranschaulicht. So findet der Entwickler schnell zu jeder Kategorie den passenden Befehl, sofern dieser verfügbar ist.
Um diese Anfragen zusätzlich testen zu können, verweist Fitbit dabei auf eine Webanwendung, welche HTTP-Requests ausführt und dessen Ergebnisse als *JSON* ausgibt. So können die Anfragen auch ohne funktionierende App getestet werden, bevor diese schlussendlich implementiert werden.
Alles in allem ist der Einstieg in solch ein Projekt sehr einfach, da die Dokumentation übersichtlich und vor allem sehr umfangreich gestaltet ist. Zusätzlich können alle weiteren Eventualitäten in dem eigenen Forum besprochen werden, welches schnell auf individuelle Probleme eingehen kann.

### 7.1.2 Bedingungen

Die Entwicklung einer Anwendung bringt jedoch auch einige Bedingungen mit sich. Diese schränken den Entwickler während der Implementation, aber auch den Benutzer in der späteren Verwendung der App, zum Teil deutlich ein.

**Anzahl der Requests** Fitbit beschränkt die Anzahl der Requests als Nutzer einer eigenen Anwendung auf nur 150 pro Stunde. Dies mag im ersten Moment nach einer ausreichenden Anzahl klingen, jedoch stellt sich bei genauerer Betrachtung schnell das Gegenteil heraus.
Je umfangreicher die Anwendung ist, desto mehr Requests benötigt diese, da für jede Anfrage ein neuer Request gestartet werden muss. Will der Benutzer zum Beispiel die aktuelle Herzfrequenz, eine Übersicht zum letzten Monat und einen speziellen Tag anfragen, was in der Testanwendung bei jedem Aufruf der Herzfrequenzübersicht durchgeführt wird, sind dies allein schon drei Anfragen.
Da für jeden speziellen Zeitraum und jede Kategorie ein neuer Request erstellt wird, summiert sich diese Anzahl sehr schnell. In einem Anwendungsfall, in welchem der Benutzer alle Menüs einmal anschaut und jeweils den aktuellen Tag in der Detailansicht durchgeht, kann sich die Anzahl schnell auf 20 - 30 Anfragen summieren. Bei häufigem Aktualisieren, wie zum Beispiel während eines Trainings, erreicht man also sehr schnell das Limit von 150 Requests.
Auch während der Entwicklung kann dies oft zu Problemen führen. Da einige Fälle oft mehrmals getestet werden müssen, erreicht man auch hier, bei mehrmaligem Aufrufen einer speziellen Seite schnell das Limit von 150 Anfragen. Gerade bei der Realisierung der Diagramme mussten diese sehr häufig getestet und somit neu aufgerufen werden.
Ist das Limit an Requests erreicht, muss man bis zum Beginn einer neuen vollen Stunde warten, um neue Anfragen stellen zu können. Dies ist im schlimmsten Fall somit genau eine Stunde, im besten Fall jedoch nur einige Minuten. In der Abbildung 7.1 ist eine Benachrichtigung zu sehen, welche man von Fitbit beim Erreichen des Limits erhält. In diesem Beispiel war die Schnittstelle knapp 40 Minuten lang gesperrt.


Hello Moritz,

This is an automated courtesy notice. No action is required. Your application, BATestApp, exceeded the Fitbit API client+viewer rate limit for the users listed below.

4ZLXZ7 (Hit: 2017/01/15 23:21:37, Reset time: 2017/01/15 23:59:59)

If you receive this notice regularly and would like to discuss potentially more efficient ways to use the Fitbit API, please contact api@fitbit.com.

Best regards,
The Fitbit API team

**Abbildung 7.1:** Die Benachrichtigung, welche durch das Erreichen des Limits gesendet wird.

Diese Einschränkung kann, je umfangreicher die Anwendung ist, zu einigen Problemen führen. Jedoch lässt die Email schließen, dass Fitbit in Einzelfällen das Limit etwas erhöhen kann, um diese zu umgehen.

**Verzögerungen** Wie schon im Abschnitt 3.5.1 Kommunikation der Schnittstelle aufgezeigt wurde, kann die Anwendung nicht mit dem Aktivitärs-Tracker direkt kommunizieren. Die Daten müssen dabei erst den umständlichen Weg über die Fitbit-App auf den Server hinter sich bringen. Dies führt wiederum zu einigen Nachteilen. Der offensichtliche Nachteil ist zunächst, dass die Fitbit-App trotz allem unverzichtbar bleibt.
Außerdem kann dies zu deutlichen Verzögerungen führen. Diese variieren dabei zwischen zwei und 15 Minuten. Durch eine manuelle Synchronisation in der offiziellen Fitbit-App kann dieser Vorgang beschleunigt werden. Jedoch ist man auch hier, wie schon erwähnt, auf diese App angewiesen und muss diesen umständlichen Zwischenschritt durchführen um aktuelle Ergebnisse zu bekommen.
Diese Verzögerung bringt zusätzlich eine Einschränkung der Funktionalität mit sich. Gerade bei der Herzfrequenz ist es wichtig, einen möglichst aktuellen Wert zu bekommen. So kann der Benutzer, zum Beispiel während einer Aktivität, beim Erreichen einer zu hohen Frequenz gewarnt werden. Da der Wert jedoch, ohne manuelles Synchronisieren der Fitbit-App, ca. zehn Minuten nach der Aufzeichnung erst abgefragt werden kann, ist diese Funktion hinfällig.
Für einen verzögerten Überblick über die Vitalwerte ist die eigene Anwendung somit sehr zu empfehlen, da die Werte hier sehr detailliert abgerufen werden können. Die Echtzeitüberwachung ist jedoch vollkommen ausgeschlossen, wodurch die Entwicklung für ein Fitbitgerät sehr viele Anwendungszwecke und somit an Relevanz verliert.

### 7.1.3 Stabilität des Servers

Während der gesamten Realisierung und Nutzung der Testanwendung gab es keine Probleme bei der Kommunikation mit den Servern von Fitbit. Diese konnten zu jeder Zeit erreicht werden und haben keine unbekannten Fehler angezeigt. Außerdem können die Daten in kürzester Zeit heruntergeladen werden, was sich deutlich in der Testanwendung bemerkbar macht, da die Werte nahezu ohne Verzögerung bei einem Aufruf einer Seite erscheinen. Auch bei größeren Datenmengen, wie die der Herzfrequenz über einen gesamten Tag, sind hier kaum Ladezeiten zu erkennen.
So ist der Fitbit-Server, unter normalen Umständen, sehr zuverlässig, sodass hier während der

gesamten Entwicklung und Nutzung der Anwendung kaum Probleme oder Fehler entstehen können.

### 7.1.4 Varianten der Herzfrequenz

Bevor mit der Realisierung der Herzfrequenz-Requests begonnen werden kann, muss der Entwickler einige Dinge beachten. Die Herzfrequenz kann ohne weitere Einstellungen, zunächst nur bedingt verwendet werden.
Bei der Erstellung der Anwendung stellt Fitbit mehrere Varianten vor, welche den späteren Verwendungszweck und Funktionsumfang dieser festlegen. Der Entwickler kann hier zwischen „Server“, „Client“ und „Personal“ wählen. Für diese Arbeit kamen jedoch nur die Client- oder die Personal-Variante in Frage.
Durch die Personal-Variante erhält man auf alle verfügbaren Daten Zugriff, darunter auch auf die sogenannte „intraday time series“. Dies bedeutet, dass jeder gemessene Wert der Herzfrequenz abgefragt werden kann. So kann sich der Benutzer den Wert zu jeder Minute oder sogar Sekunde des Tages anzeigen lassen. Außerdem werden hier, bis auf die oben genannte Verzögerung, aktuelle Werte angezeigt.
Gerade während oder nach einer Aktivität, ist es sehr interessant zu sehen, wie sich die Herzfrequenz während dieser entwickelt hat. Zusätzlich wird hier ein Durchschnittswert und die jeweiligen Zeiten innerhalb verschiedener Zonen ausgegeben. Durch die Anpassung der Requests ist dabei jeder gewünschte Zeitraum bis zu einem ganzen Tag möglich. Das Listing 7.1 zeigt einen Beispiel Datensatz zu solch einer Anfrage. Ohne diesen Zugriff hingegen sind lediglich Zusammenfassungen über den kompletten Tag möglich. Diese zeigen somit nur den Ruhepuls und die Herzfrequenzzonen des Tages an. Einzelne Werte zu bestimmten Zeiten sind daher nicht ersichtlich.
Durch die Wahl der Personal-Variante beschränkt sich der Entwickler jedoch darauf, dass nur er selbst die Anwendung verwenden kann. Das heißt diese ist mit nur einem Fitbit-Account verbunden und kann von keiner anderen Person verwendet werden.
Die **Client**-Variante bietet, im Gegensatz dazu, Zugang für jeden beliebigen Nutzer. Hier verliert man allerdings zunächst den Zugriff auf die **intraday time series**, welcher aber unter gewissen Bedingungen erlangt werden kann. Hierfür bietet Fitbit die Möglichkeit dies durch eine E-Mail an den Support freizuschalten. In dieser muss die Anwendung beschrieben und deren Verwendungszecke aufgezeigt werden. Somit stellt Fitbit sicher, dass die App nicht zu kommerziellen Zwecken verwendet wird. Dieser Vorgang wurde für die, in dieser Arbeit beschriebenen Testanwendung, erfolgreich durchgeführt. Das Verfahren hat dabei problemlos innerhalb nur eines Tages funktioniert.

**Listing 7.1:** Beispielausgabe zu einer Intraday time series

```
    "customHeartRateZones": [],
    "dateTime": "today",
    "heartRateZones": [
     {
      "caloriesOut": 2.3246,
      "max": 94,
      "min": 30,
      "minutes": 2,
      "name": "Out of Range"
     },
     [...]
    ],
    "value": "64.2"
   }
  ],
"activities-heart-intraday": {
  "dataset": [
   {
    "time": "00:00:00",
    "value": 64
   },
   {
    "time": "00:00:10",
    "value": 63
   }
   [...]
```

### 7.1.5 Angebot an Requests

Im wesentlichen bietet Fitbit ein sehr breites Spektrum an Requests, sodass während der Entwicklung in diesem Bereich nur wenig Probleme aufgetreten sind. In den meisten Fällen ist das Angebot so groß, dass alle möglichen Anfragen abgedeckt werden. So bietet jede Kategorie die Möglichkeit, einen kompletten Datensatz, oder nur einzelne Werte für jedes gewünschte Datum herunterzuladen. Außerdem können diese Daten auch über einen längeren Zeitraum, wie zum Beispiel einem Monat, abgefragt werden. Dies ist jedoch gleichzeitig einer der Punkte, in welchen das Angebot zu knapp gestaltet ist. Die folgenden Probleme sind während der Entwicklung der Anwendung aufgetreten.

**Anfragen über einen langen Zeitraum** Bei Anfragen über einen größeren Zeitraum können ausschließlich einzelne Daten und nicht der komplette Datensatz zu der jeweiligen Kategorie abgerufen werden. Am Beispiel der Herzfrequenz wird nur der Ruhepuls und die Herzfrequenzzonen der jeweiligen Tage ausgegeben. Ebenso ist es auch bei den restlichen Kategorien, wie bei den Daten zum Schlaf. Um somit die Dauer und die Qualität des letzten Monates festzustellen, sind hier zwei Requests nötig. Das heißt, je mehr Informationen relevant sind, desto

mehr Request werden benötigt.
Wie schon im Abschnitt 5.3.4 *Herzfrequenz - Übersicht* angesprochen, sollte die Tabelle, welche die Werte der letzten Tage ausgibt, zusätzlich die durchschnittliche Herzfrequenz anzeigen. Da die Anfrage über einen längeren Zeitraum diesen Wert jedoch nicht liefert, muss für jeden Tag ein individueller Request getätigt werden. Dies ist jedoch nicht möglich, da an dieser Stelle beliebig viele Einträge in der Tabelle stehen können, woraus folgt, dass hierfür auch die gleiche Anzahl an Requests nötig ist. Zum Ersten würde dieser Prozess die Anwendung erheblich verlangsamen und zweitens würde man so sehr schnell das oben erwähnte Limit an Anfragen erreichen.
Diese Gründe tragen somit dazu bei, dass auf bestimmte Funktionen beziehungsweise Informationen verzichtet werden muss, da es durch die eingeschränkten Requests nicht möglich ist.

**Zugriff auf den letzten Datensatz** Ebenfalls fehlt die Möglichkeit, den letzten Datensatz der jeweiligen Kategorie abzufragen. Da dem Entwickler nicht immer bekannt ist, wann zum Beispiel der letzte Schlaf aufgezeichnet wurde, kann an dieser Stelle nicht nach einem bestimmten Datum gefragt werden.
Um die Anfrage zu vereinfachen, wäre eine Methode nützlich, über die man den zuletzt gemessenen Wert anfragen kann. Da dies jedoch nicht verfügbar ist, muss dieser Wert über einen umständlichen Weg heraus gefunden werden.
Gerade auf dem Dashboard (siehe Abschnitt 5.3.2) der Testanwendung sollen genau diese Daten angezeigt werden. Um den letzten bekannten Wert zu erhalten, musste hier zunächst eine Anfrage über einen längeren Zeitraum, wie beispielsweise einen Monat, getätigt werden. Diese liefert somit ein chronologisch sortiertes *Array*. Aus diesem kann schlussendlich der letzte Wert herausgenommen und verwendet werden.
Dieses Verfahren sorgt jedoch für einige Daten, die im Endeffekt nicht benötigt und somit umsonst heruntergeladen werden müssen. Vor allem bei einer schlechten Internetverbindung kann es so zu kleineren Verzögerungen kommen.

### 7.1.6 Detailgrad der Daten

Was die Datensätze betrifft, bietet Fitbit den Entwicklern ein sehr detailliertes Angebot. Wählt man die Option mit Zugriff auf den kompletten Herzfrequenz-Datensatz, kann man nahezu alle Daten, die das Armband sammelt, abrufen. Die einzige Ausnahme dabei sind die GPS-Daten, welche während einer Aktivität aufgezeichnet werden.
So ist es, wie schon erwähnt, möglich die Herzfrequenz zu jeder Sekunde an jedem Tag einzusehen. Auch die Schlafdaten bieten aufschlussreiche Informationen, da hier zu jeder Minute ein Wert aufgezeichnet wird, welcher angibt, wie gut der Schlaf des Nutzers ist.
Dieser hohe Detailgrad wird dabei bei jeder Kategorie gewährleistet. So können hier, sofern die nötigen HTTP-Requests getätigt wurden, die selben Daten, wie auch in der Anwendung von Fitbit selbst, ausgegeben werden. Daher steht die eigene App der Fitbit-App in diesem Punkt, abgesehen von den GPS-Daten, nichts nach.

## 7.2 Fazit

Die Entwicklung einer eigenen Anwendung, welche auf die gesammelten Daten eines Fitbit-Aktivitästracker zurück greift, bietet somit, aufgrund des Zugriffs auf nahezu alle aufgezeichneten Werte, grenzenlose Möglichkeiten, diese weiterzuverarbeiten. Dabei wird dem Entwickler durch eine sehr ausführliche Dokumentation mit einer hohen Anzahl von Beispielen der Einstieg und auch die fortlaufende Entwicklung sehr einfach gemacht. Sollten trotz allem Probleme auftreten, bietet Fitbit zusätzlich die Möglichkeit, Fragen in einem eigenem Forum zu stellen, wo diese zeitnah von Gleichgesinnten beantwortet werden können.
Auch das Angebot der möglichen Anfragen um Daten herunterzuladen, ist sehr groß und bietet in fast allen Fällen einen passenden Request. Zusätzlich verspricht Fitbit dieses Angebot stetig auszubauen und auf Wünsche der Entwickler einzugehen. Die in dieser Arbeit aufgelisteten Probleme bezüglich dieses Angebotes, könnten somit in naher Zukunft eventuell behoben werden.
Jedoch wird der Entwickler trotz allem deutlich eingeschränkt. Vor allem durch die großen Verzögerungen, welche auf die indirekte Kommunikation zurückzuführen sind, gehen viele Funktionen verloren. Außerdem muss ständig darauf geachtet werden, dass das Requestlimit nicht erreicht wird, da dadurch die komplette Anwendung bis zur nächsten vollen Stunde nicht mehr verwendet werden kann.
So wird die offizielle Fitbit-App, was die Visualisierung, das Ver- und Bearbeiten der Daten betrifft, immer mehr Funktionalität als die eigene Anwendung bieten. Außerdem ist diese stets auf einem aktuelleren Stand, da das Armband die Daten direkt per Bluetooth an die App überträgt. Zusätzlich sorgt Fitbit mit der beschränkten Freigabe der Herzfrequenz-Daten dafür, dass Anwendungen von Drittanbietern nur mit Fitbits Einverständnis veröffentlicht werden dürfen.
Alles in allem bietet Fitbit also eine sehr umfangreiche Schnittstelle, mit vielen Freiheiten für den Entwickler. Jedoch sorgen die Einschränkungen dafür, dass die offizielle Fitbit-Anwendung stets umfangreicher und aktueller sein wird. Diese kann außerdem nicht komplett ersetzt werden, da sie als Schnittstelle zum Hochladen der Daten auf den Server fungiert. Schlussendlich kann somit gesagt werden, dass die Entwicklung einer eigenen Anwendung nur dann Sinn macht, wenn die Daten nicht nur visualisiert, sondern eventuell noch mit Daten einer anderen Quelle verbunden oder gar auf eine ganz andere Weise, wie zum Beispiel durch ein Spiel, verarbeitet werden sollen. Vor allem Letzteres ist dabei besonderes interessant, da Spiele den Benutzer nachweislich auf längere Sicht zur Bewegung animieren können. [30, 13]

# 8 Experiment zum Datenabgleich

Das folgende Kapitel befasst sich nun mit den gesammelten Daten des Armbandes. Dabei soll überprüft werden, wie zuverlässig die Daten aufgezeichnet werden und wie genau diese sind. Zunächst wird dafür der Aufbau des Experimentes vorgestellt und anschließend die Durchführung dazu dokumentiert. Zuletzt werden die Ergebnisse ausgewertet und gedeutet.

## 8.1 Aufbau des Experimentes

In diesem Abschnitt wird der genaue Aufbau näher erläutert. Hierfür wird zunächst der Ablauf festgelegt, woraufhin anschließend die verwendeten Produkte vorgestellt werden. Außerdem wird in diesem Kapitel aufgezeigt, welche Daten genau verglichen werden sollen.

### 8.1.1 Ablauf

Um einen Zeitabschnitt mit möglichst vielen vergleichbaren Daten zu erhalten, wird das Experiment während eines Laufes durchgeführt. Hier werden nahezu alle relevanten Kategorien aufgezeichnet, welche überprüft werden können.
So wird die Strecke des Laufes vor dem Experiment mit **Google-Maps** festgelegt, um einen genauen Richtwert zu bekommen. Zur Herzfrequenz wird neben des Fitbit Aktivitäs-Trackers ein Brustgurt verwendet, welcher als Vergleichswert dienen soll. Dabei wird am Ende des Experimentes die Durchschnittsherzfrequenz der beiden verglichen. Zusätzlich soll ein bestimmter Abschnitt während des Laufes ausgewählt werden, aus welchem einzelne Werte zu einer bestimmten Zeit verglichen werden sollen.
Außerdem können abschließend die gelaufenen Schritte, die Kalorien und die Geschwindigkeit des Laufes untersucht werden.

### 8.1.2 Verwendete Produkte

Wie bereits in den vorangegangen Kapiteln, werden die Daten Fitbits über die *Surge* gesammelt. Um diese einzusehen, wird zusätzlich die hier erstellte Testanwendung verwendet. Über die Uhr kann die Aktivität manuell gestartet und auch beendet werden. Mit Hilfe der eingebauten GPS Funktion wird neben der Zeit und den Daten der Aktivität zusätzlich die gelaufene Strecke gemessen.
Um die Herzfrequenz zu vergleichen wurde hier der Bluetooth-Brustgurt **Heart Rate Strap CS009** von **ALATECH** [33] verwendet. Dieser kann neben der eigenen Anwendung **ALA COACH** zusätzlich mit der bekannten App **Runtastic** verbunden werden. Durch diese erhält man zusätzlich zu der Herzfrequenz die Distanz, Geschwindigkeit und alle weiteren Werte des Laufes. Aufgezeichnet wurde dies über ein iPhone 6s.

### 8.1.3 Strecke

Die Abbildung 8.1 zeigt die geplante Strecke, die für das Experiment gewählt wurde. Der grüne Punkt symbolisiert dabei den Start und der rote das Ende der Strecke. Wie am unteren Rand der Abbildung zu sehen ist, beträgt die Strecke 7,56 km.

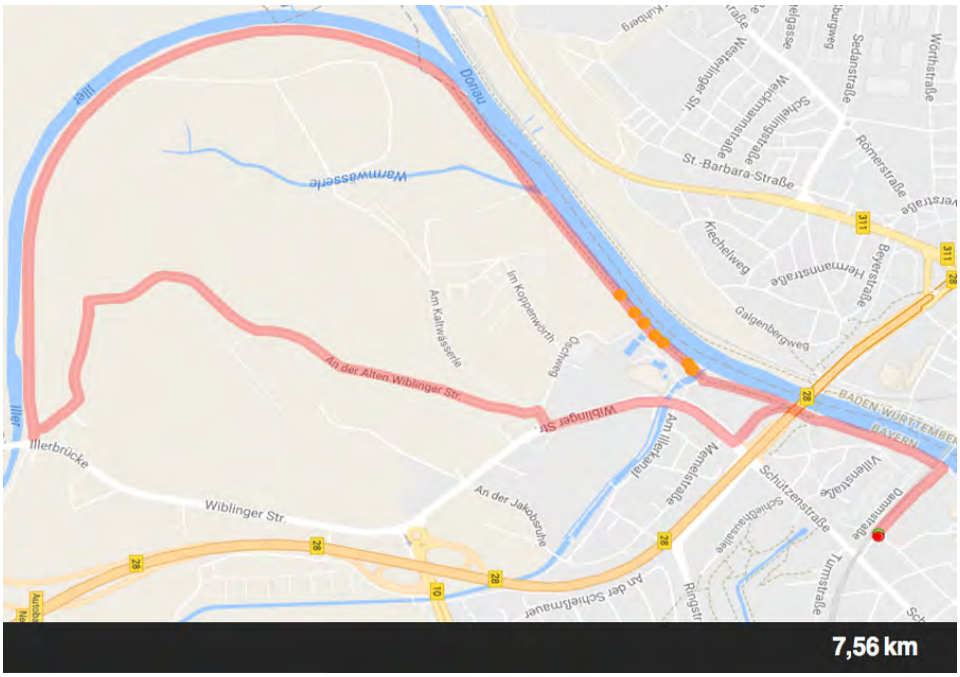



**Abbildung 8.1:** Abbildung der geplanten Strecke

## 8.2 Durchführung und Ergebnisse

Dieser Abschnitt befasst sich mit der Durchführung und der Auswertung der Ergebnisse. Dazu werden die einzelnen Werte, welche während der Aktivität gemessen wurden miteinander ver-

glichen und, wenn möglich, bewertet.
Da auch hier die Herzfrequenz im Fokus steht, wird diese als erstes aufgeführt.

### 8.2.1 Herzfrequenz

Die Abbildung 8.2 zeigt die durchschnittliche Herzfrequenz, welche während der Aktivität gemessen wurde. Die links abgebildeten Daten wurden mit Hilfe des Brustgurtes gemessen, die mittels der **Runtastic** App dargestellt werden. Der Wert auf der rechten Seite wird von der eigens erstellten Anwendung präsentiert, der mit der **Fitbit-Surge** ermittelt wurde.


**Abbildung 8.2:** Vergleich der durchschnittlichen Herzfrequenz. Links: Runtastic. Rechts: Testanwendung

Die durchschnittliche Herzfrequenz ist dabei erstaunlicherweise identisch, woraus man eine sehr akkurate Messung der beiden Geräte schließen kann. Nun gilt es somit noch zu überprüfen, ob auch einzelne Werte übereinstimmen.

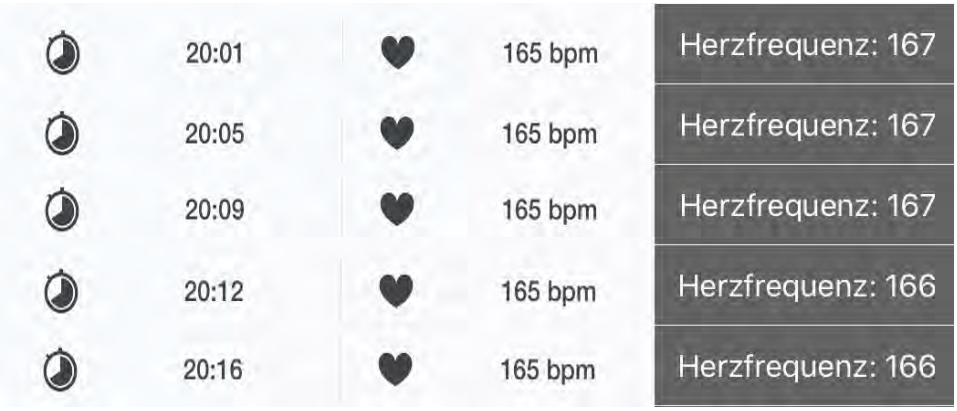



**Abbildung 8.3:** Vergleich eines bestimmten Abschnittes. Links: Runtastic. Rechts: Testanwendung

Wie die Abbildung 8.3 zeigt, unterscheiden sich die Daten dieses Abschnittes nur geringfügig. Im linken Bereich der Abbildung werden die Zeiten dargestellt, zu welchen die Herzfrequenz gemessen wurde. Da der Zeitpunkt und die Häufigkeit der Messungen bei beiden Produkten nicht beeinflusst werden kann, stimmen die Zeitstempel nicht exakt überein. Während der Brustgurt von ALATECH, wie hier in der Abbildung zu sehen, „nur“ alle vier Sekunden misst, zeichnet Fitbit sogar alle drei Sekunden auf. So startet der erste Wert der Messung aus der Abbildung bei Fitbit eine Sekunde später und endet schon zwei Sekunden früher. Um die Werte jedoch anschaulich vergleichen zu können, wurde dieser minimale Unterschied ignoriert, weshalb der Zeitstempel der Testanwendung hier nicht abgebildet wird.
Der Mittlere Teil der Abbildung repräsentiert somit die Daten des Brustgurtes, wohingegen der rechte Teil die Messungen des Fitbit-Armbandes darstellt. Wie hier deutlich zu sehen ist, bleibt der Wert des Brustgurtes durchgehend bei 165 Schlägen pro Minute. Im Gegensatz dazu schwankt die Messung von Fitbit (rechts) und ist zusätzlich etwas höher.

Da jedoch die durchschnittliche Herzfrequenz übereinstimmt und auch die Verteilung der Herzfrequenzzonen, wie Abbildung 8.4 zeigt, sehr ähnlich ist, kann man davon ausgehen, dass es sich hierbei lediglich um einzelne geringe Abweichungen handelt. Die Unterschiede der „grünen"- und der „gelben"-Zone in Abbildung 8.4 sind dabei auf die unterschiedliche Abstufung der Bereiche zurückzuführen. Fügt man diese auf einer gemeinsamen Skala zusammen, stimmen auch die Zonen nahezu perfekt überein. Zusätzlich haben beide Produkte eine maximale Frequenz von exakt 175 Schlägen pro Minute gemessen, was erneut für eine sehr ähnliche, genaue Messung spricht.

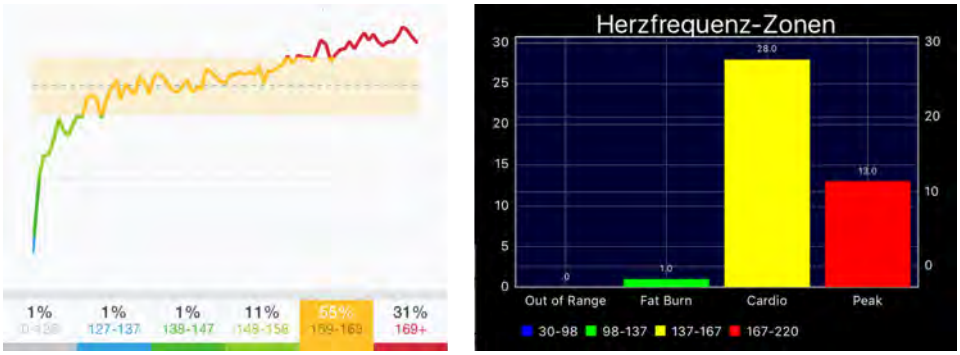



**Abbildung 8.4:** Vergleich Herzfrequenzzonen. Links: Runtastic. Rechts: Testanwendung

### 8.2.2 Allgemeine Daten der Aktivität

Die Abbildung 8.5 zeigt nun schlussendlich die allgemeinen Daten der Aktivität. Hierbei werden im linken Bereich der Darstellung wieder die Daten der **Runtastic**-App dargestellt und im rechten Bereich die Messungen der **Fitbit Surge** veranschaulicht.
Zunächst fällt auf, dass **Runtastic** mit einer gemessenen Strecke von 7,57 km nahezu den exakten Wert der Vorgabe erreicht hat, während Fitbit mit 7.17 km deutlich darunter liegt. Hier liegt die Anwendung von Runtastic jedoch auch klar im Vorteil, da die Aktivität über das Smartphone aufgezeichnet wird, welches die GPS-Daten mithilfe von Straßenplänen über das Internet abgleichen kann. Fitbit hingegen wurde ausschließlich über die Surge, also ohne Verbindung zum Smartphone, aufgezeichnet.
Aufgrund der unterschiedlichen Distanzen werden offensichtlich auch die Werte zu der Geschwindigkeit und der Pace beeinflusst. Somit entsprechen die Messungen von **Runtastic** eher der Realität, da die gemessene Strecke hier nahezu der Vorgabe entspricht.
Zuletzt müssen noch die übrigen Angaben, wie die Kalorien und die Anzahl der Schritte untersucht werden. Hier ähneln sich beide Werte erneut sehr stark. Die **Runtastic**-Anwendung hat dabei einen etwas höheren Kalorienverbrauch gemessen, was ebenfalls auf die Messunterschiede der zurückgelegten Distanz zurückgeführt werden kann.
Wie man in der Abbildung sehen kann, wird bei Runtatsic die genaue Anzahl der Schritte nicht aufgeführt. Mithilfe der durchschnittlichen Schritte pro Minute und der gemessen Zeit, kann der Gesamtwert jedoch errechnet werden.
Multipliziert man so diese beiden Werte, erhält man ein sehr ähnliches Ergebnis zu Fitbits Aufzeichnung.

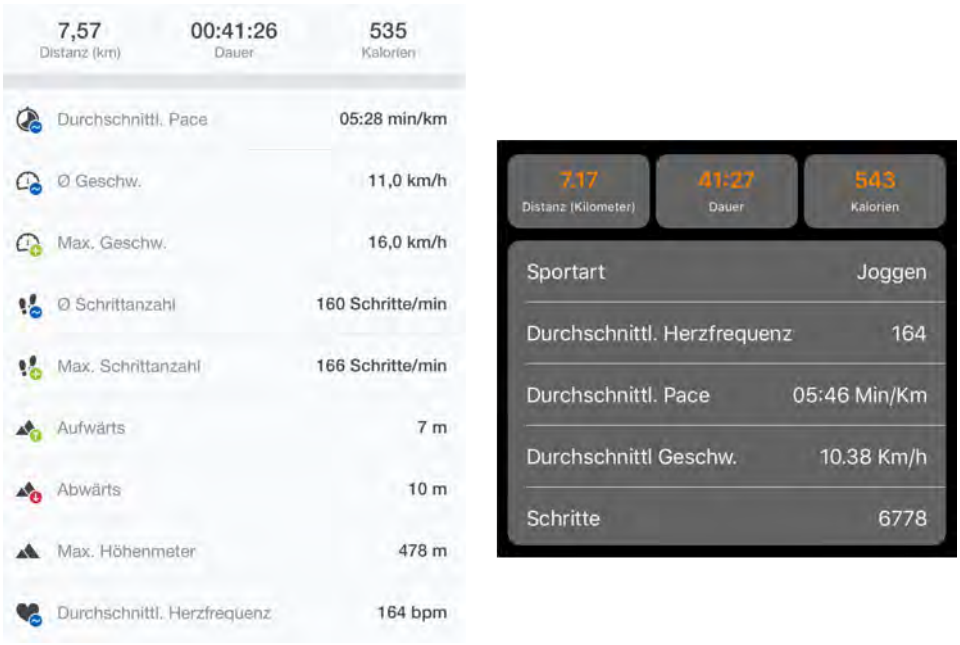



**Abbildung 8.5:** Vergleich der allgemeinen Daten. Links: Runtastic. Rechts: Testanwendung

### 8.2.3 Fazit

Abschließend kann man durch die hier aufgezeigten Ergebnisse feststellen, dass die beiden Produkte trotz geringer Unterschiede sehr ähnliche und vor allem genaue Daten zu dem aufgezeichneten Lauf zurückliefern. Erstaunlich ist dabei, dass die Herzfrequenz nahezu identisch gemessen wurde.
Trotz der kleinen Ungenauigkeit bezüglich der gelaufenen Strecke seitens Fitbit, sind auch die allgemeinen Werte zu der Aktivität sehr akkurat. Somit dienen schlussendlich beide Produkte als gute Übersicht und liefern verlässliche Ergebnisse.

# 9 Zusammenfassung

In dem folgenden Kapitel werden die wichtigsten Aspekte der Arbeit nochmals hervorgehoben. Neben der Zusammenfassung kann mit Hilfe des Experiments nun eine endgültige Schlussfolgerung gezogen werden.
Zum Einstieg in die Arbeit wurde zunächst Fitbit und das zugehörige Ökosystem vorgestellt. Um einen anschaulichen Vergleich zu ermöglichen, wurde dieses mit ähnlichen Herstellern und deren Produkten gegenübergestellt. Das Hauptaugenmerk lag dabei auf den Schnittstellen, welche für Entwickler freigegeben werden, um auf die Daten des Fitnesstrackers zuzugreifen. Dabei fiel vor allem die besondere Architektur der Kommunikation zwischen dem Tracker und der eigenen Anwendung auf dem Smartphone auf. Da die beiden Komponenten keine direkte Datenübertragung aufbauen können, musste dies über einen Server als Zwischenschritt realisiert werden.
Um den Umgang mit der Schnittstelle zu testen, wurden zunächst Anforderungen für eine Testanwendung aufgestellt, welche daraufhin auf dieser Basis realisiert wurde. Das Ziel der Anwendung war es, die Daten eines Fitbit-Armbandes auswerten zu können. Somit stand die Kommunikation mit der Schnittstelle im Zentrum dieser Entwicklung. Ein weiterer besonderer Aspekt war außerdem der Datensatz bezüglich der Herzfrequenz des Benutzers, da der Zugriff zu diesem mit einigen Bedingung verknüpft war.
Mit der Fertigstellung der Anwendung, wurde diese in Kapitel 6 mit den vorher festgelegten Anforderungen abgeglichen.
Abgeschlossen wurde die Entwicklung mit einer Auswertung bezüglich Fitbits Ökosystems. Dabei wurden die Probleme und Vorteile der Schnittstelle aufgezeigt und bewertet. Zuletzt wurde in Kapitel 8 mit Hilfe eines Experiments die Genauigkeit des Armbandes getestet.

## 9.1 Fazit

Das Ziel dieser Arbeit war es, mit Hilfe einer eigens entwickelten Testanwendung den Umgang mit Fitbits Schnittstelle, sowie den gesammelten Daten, überprüfen zu können. Dabei sollten vor allem die Bedingungen der besonderen Architektur und die Zugriffsrechte zu Daten wie zum Beispiel der Herzfrequenz untersucht werden. Die Umsetzung der Anwendung erfolgte hierfür auf dem Betriebssystem iOS für iPhones.
Die darauf folgende Evaluation zeigt, dass der Entwickler bei richtigen Einstellungen sehr viel Freiheit besitzt und auf nahezu alle Daten des Tracker zurückgreifen kann. Die aufgezeigten Bedingungen, wie zum Beispiel die Verzögerung, mit welcher die Daten auf den Server geladen werden oder das Limit an Anfragen an diesen, schränken den Entwickler jedoch auch stark in der Funktionalität ein. Wird die Anwendung also mit dem Ziel, lediglich die Daten abzufragen um diese zu visualisieren, entwickelt, wird die offizielle Fitbit-Anwendung stets umfangreicher und vor allem schneller sein. Außerdem kann diese nicht ersetzt werden, da sie als Zwischenschritt zur Übertragung der Daten fungiert.
Die Schnittstelle besitzt jedoch trotz all diesen Bedingungen ihre Existenzberechtigung, da die Daten, neben der Visualisierung, unzählige weitere Anwendungszwecke bieten. So können die Daten, mit Hilfe einer eigenen App, beispielsweise mit der iOS Anwendung Health verknüpft werden, da die offizielle Fitbit-Anwendung dies nicht unterstützt.
Außerdem könnten die Daten, sofern Fitbit die Einverständnis dazu gibt, durch eine Anwendung zum sogenannten Crowdsensing verwendet werden. Dabei werden alle Daten einer Gruppe von mehreren Menschen (**Crowd**) gesammelt und verarbeitet. Dies eignet vor allem für Studien oder Experimente bezüglich der Gesundheit. Da das Fitbit-Armband durchgehend die Herzfrequenz aufzeichen kann, und zu dieser auch jeder einzelne Wert eingesehen werden kann, liefert dies nützliche Informationen über die Gesundheit und den Zustand des Benutzers. [31]
Alles in allem kann die Schnittstelle also bei korrekter und sinnvoller Anwendung sehr nützlich sein. Da das Armband, wie das Experiment aus Kapitel 8 gezeigt hat, die Vitalwerte und Daten sehr akkurat aufzeichnet, bietet die Schnittstelle einen äußerst genauen und umfangreichen Datensatz bezüglich des Benutzers. So können vor allem im medizinischen Bereich sehr nützliche Anwendungen entstehen.

# Literaturverzeichnis


[1] Statistika: Absätze der Wearables https://de.statista.com/infografik/4323/prognose-zum-weltweiten-absatz-von-wearables/ Aufgerufen: 04-02-2017

[2] Bitkom: Ein Drittel nutzt Fitnesstracker https://www.bitkom.org/Presse/Presseinformation/Gemeinsame-Presseinfo-von-Bitkom-und-BMJV-Fast-ein-Drittel-nutzt-Fitness-Tracker.html Aufgerufen: 04-02-2017

[3] Fitbit: Über Fitbit https://www.fitbit.com/de/about , Aufgerufen: 16-03-2017

[4] Fitbit: Alle Produkte https://www.fitbit.com/de/store , Aufgerufen: 16-03-2017

[5] Fitbit: Detailseite der Surge https://www.fitbit.com/de/shop/surge, Aufgerufen: 16-03-2017

[6] ITunes: Beschreibung und Abbildung der Fitbit-App https://itunes.apple.com/de/app/fitbit/id462638897?mt=8 , Aufgerufen: 16-03-2017

[7] Fitbit: Developerseite https://dev.fitbit.com/de , Aufgerufen: 16-03-2017

[8] Lukasz Piwek , David A. Ellis, Sally Andrews, Adam Joinson: The Rise of Consumer Health Wearables: Promises and Barriers. In: PLoS Med 13(2): e1001953. doi:10.1371/journal.pmed.1001953. 2016

[9] Wikipedia: Informationen zum Hersteller Jawbone https://en.wikipedia.org/wiki/Jawbone_(company) , Aufgerufen: 16-03-2017

[10] Jawbone: Produktbeschreibung UP3 https://jawbone.com/fitness-tracker/up3, Aufgerufen: 16-03-2017

[11] Jawbone: Developerseite https://jawbone.com/up/developer/ , Aufgerufen: 16-03-2017

[12] Github: Beschreibung der Jawbone SDK https://github.com/Jawbone/UPPlatform_iOS_SDK , Aufgerufen: 16-03-2017

[13] Zhao Zhao, Ali Etemad, Ali Arya: Gamification of Exercise and Fitness using Wearable Activity Trackers. In: https://www.researchgate.net/publication/284887211. 2016

[14] Apple: Beschreibung der Apple Watch http://www.apple.com/apple-watch-series-2/, Aufgerufen: 16-03-2017

[15] Thomas Fritz, Elaine M. Huang, Gail C. Murphy, Thomas Zimmermann: Persuasive Technology in the Real World: A Study of Long-Term Use of Activity Sensing Devices for Fitness. In: Zurich Open Repository and Archive, University of Zurich. 2014

[16] Apple: Developerseite: Healthkit https://developer.apple.com/healthkit/, Aufgerufen: 16-03-2017

[17] Apple: Developerseite: WatchOS https://developer.apple.com/watchos/, Aufgerufen: 16-03-2017

[18] Apple: Developerseite: https://developer.apple.com/library/content/documentation/ General/Conceptual/ WatchKitProgrammingGuide/index.html , Aufgerufen: 16-03-2017

[19] Wikipedia: Erscheinungsdatum der Produkte https://en.wikipedia.org/wiki/List_of_ Garmin_products#Fitness

, Aufgerufen: 16-03-2017

[20] Garmin: Produktbeschreibung des Fitnessarmbandes https://explore.garmin.com/de-DE/vivo-fitness/ , Aufgerufen: 16-03-2017

[21] ITunes: Beschreibung und Bild der Garmin-App https://itunes.apple.com/de/app/garmin-connect-mobile/id583446403?mt=8, Aufgerufen: 16-03-2017

[22] Garmin: Informationen zur Developerseite https://developer.garmin.com, Aufgerufen: 16-03-2017

[23] ITunes: Beschreibung und Abbildung der UP-App https://itunes.apple.com/de/app/up-by-jawbone-aufzeichnen-mit-up-move-up24/id461125277?mt=8 Aufgerufen: 16-03-2017

[24] imore: Abbildung zur Apple Watch http://www.imore.com/apple-watch-activity-tracking-5-tips-you-need-know, Aufgerufen: 16-03-2017

[25] Swift: Zitat zu Swift ,https://swift.org, Aufgerufen: 01-02-2017

[26] Apple: Beispiel zu Optionalen Variablen ,https://developer.apple.com/library/content /documentation/Swift/Conceptual/ Swift_Programming_Language/OptionalChaining.html Aufgerufen: 10-03-2017

[27] Github: Qulle für die Bibliothek Charts https://github.com/danielgindi/Charts, Aufgerufen: 13-01-2017

[28] Github: Qulle für die Bibliothek PureLayout https://github.com/PureLayout/PureLayout, Aufgerufen: 13-01-2017

[29] Github: Qulle den Login mit OAuth: https://github.com/Stasonis/fitbit-api-example-swift, Aufgerufen: 26-03-2017

[30] Fabio Buttussi, Luca Chittaro, "Smarter Phones for Healthier Lifestyles: An Adaptive Fitness Game," IEEE Pervasive Computing, vol. 9, no. 4, pp. 51- 57, 2010.

[31] Rüdiger Pryss, Manfred Reichert, Berthold Langguth, Winfried Schlee: "Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy and Research", In: IEEE 4th International Conference on Mobile Services (MS 2015), IEEE Computer Society Press (2015)

[32] Fitbit: Abbildung OAuth https://dev.fitbit.com/docs/images/oauth-2-0-consent-screen-with-shadow.png, Aufgerufen: 10-02-201

[33] Alatech: Produktbeschreibung http://en.alatech.com.tw/CS009%20BLE%204.0%20Heart % 20Rate%20Strap/action-products_detail-did-1038.html, Aufgerufen: 18-03-2017