Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektro- und Informationstechnik

Studiengang Medieninformatik

Bachelorarbeit

von

Marvin **Jäckisch**

Entwicklung einer Smartphone-App zur Trainingsunterstützung

Development of a Smartphone App for Training Support

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektro- und Informationstechnik

Studiengang Medieninformatik

Bachelorarbeit

von

Marvin **Jäckisch**

Entwicklung einer Smartphone-App zur Trainingsunterstützung

Development of a Smartphone App for Training Support

Bearbeitungszeitraum:	von	28. Juni 2021
	bis	26. November 2021

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektro- und Informationstechnik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Jäckisch, Marvin**
Studiengang: Medieninformatik

Durchgeführt in (Firma/Behörde/Hochschule): CLEAN FITNESS GmbH, Maxhütte-Haidhof

Ausgabedatum: 28. Juni 2021 Abgabedatum: 26. November 2021

Titel:

Entwicklung einer Smartphone-App zur Trainingsunterstützung

Zusammenfassung:

Es wird eine App mithilfe von Flutter erstellt, die bei der 33-Tage Challenge zum Einsatz kommt. Bei der 33-Tage Challenge geht es darum, dass der Benutzer in den 33 Tagen die Disziplin hat, die vorgegebenen Trainings- und Ernährungspläne einzuhalten. Die App bietet eine Mediathek, in der verschiedene Videos und Anleitungen zu der Challenge abgespielt werden können. Es werden automatische Trainingspläne für den Benutzer erstellt, wobei anhand seines Levels (z. B. Übergewicht, körperlicher Zustand) verschiedene Übungen zusammengestellt werden. Wenn der Benutzer das Training startet, sieht er auf seinem Smartphone eine Anleitung zu der Übung und es startet ein Timer. Die Daten des Trainings werden per Apple Health/Google Fit gespeichert und es wird eine Statistik über das Training angezeigt (Kalorienverbrauch, durchschnittliche Herzfrequenz, usw.). Nach dem Training wird noch ein Vorschlag für eine vom Kalorienverbrauch abhängige Mahlzeit gemacht. Es wird außerdem in der App noch eine Statistik über die 33 Tage geben. Für den Zugriff auf Login- und Anamnesedaten wird die App per REST-API an eine bereits vorhandene Datenbank angebunden.

Schlüsselwörter: Flutter, App-Entwicklung, Fitness, Kalorienberechnung

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Körperliche Fitness war schon immer ein wichtiges Thema, vor allem im früheren Berufsleben. Durch neue Technologien und deren Fortschritte können viele Arbeitsschritte durch maschinelle Arbeit ersetzt werden. Gerade im heutigen Büroalltag ist es nach wie vor wichtig, seinen Körper fit zu halten. Deshalb entscheiden sich immer mehr Menschen, etwas für ihre Gesundheit und ihren Körper zu tun und Sport zu treiben.

Heutzutage wird das Smartphone oder eine Smartwatch immer mehr als Hilfsmittel bei sportlichen Aktivitäten verwendet. Eine Smartwatch kann unter anderem Herzfrequenz und Kalorienverbrauch messen, ein Smartphone kann bereits die zurückgelegten Schritte zählen und bei der Ausführung von Übungen helfen.

Durch die Arbeit bei CLEAN FITNESS wurde mir immer mehr bewusst, wie wichtig es ist, dieses Wissen an andere weiterzugeben und so kam ich auf die Idee, diese App zu entwickeln.

Die hier beschriebene Fitness-App soll Konzepte aufzeigen, die ein personalisiertes Training mit einer individuellen Mahlzeit verbinden. Darüber hinaus möchte sie mit der 33-Tage-Challenge dem Nutzer eine anwenderfreundliche Möglichkeit bieten, sich mit dem Thema Fitness und gesunde Ernährung auseinanderzusetzen.

1.2 Zieldefinition

Das Ziel der Arbeit besteht in der Konzeption und Umsetzung einer Fitness-App für die 33-Tage-Challenge, für Android- und iOS-Geräte.

Das Hauptaugenmerk liegt auf der Erstellung eines individuellen Trainingsplans, der Hilfestellung bei der Durchführung der Übungen und der Messung des Kalorienverbrauchs. Ein weiteres Ziel dieser Arbeit ist es, einen Überblick über den Trainingsverlauf der 33 Tage zu geben.

Kapitel 2

Grundlagen

2.1 CLEAN FITNESS GmbH

Das Unternehmen CLEAN FITNESS GmbH wurde 2019 in Maxhütte-Haidhof von Sascha Schulz gegründet. Seine Intention war es, gesunden Sport und gesunde Ernährung zu verbinden. Er machte sich selbstständig und begann, Menschen, die ihren Lebensstil verbessern möchten, Personal Training anzubieten. Er wollte seine Vision nicht klein halten und gründete das Franchiseunternehmen CLEAN FITNESS. Seitdem hat CLEAN FITNESS über 100 Fitnesstrainer ausgebildet und unterstützt sie beim Aufbau ihrer Selbstständigkeit. Darüber hinaus gibt es regelmäßige Schulungen, um Fitnesstrainer auf dem Laufenden zu halten.

Das in der Fitnessbranche tätige Unternehmen bietet ausgebildeten Fitnesstrainern eine Vielzahl von Dienstleistungen an. Sie bieten den Trainern eine eigene Homepage mit Suchmaschinenoptimierung sowie Unterstützung bei der Erstellung von Werbemitteln und steuerlichen Fragestellungen.

2.2 Die 33-Tage Challenge

Die 33-Tage Challenge wird von den Fitnesstrainern an ihre Kunden verkauft und unterstützen diese dann. Die Herausforderung besteht darin, dass der Kunde innerhalb von 33 Tagen Freude an Sport und gesunder Ernährung findet. Der Kunde erhält wöchentlich einen Ernährungsplan mit Einkaufsliste sowie einen Trainingsplan, mit dem er alle zwei bis drei Tage trainieren kann. Zu Beginn führt der Trainer die Anamnese des Kunden, die die aktuellen Körpermaße wie Gewicht, Alter oder Körperfettanteil ermittelt. Am Ende der 33 Tage prüft der Trainer noch einmal, wie sich die Gesundheit und Fitness des Kunden verändert haben.

2.3 Cross-Plattform-Frameworks

In diesem Abschnitt geht es darum, sich einen Überblick über die aktuellen plattformübergreifenden Frameworks zu verschaffen und die Vor- und Nachteile der einzelnen Frameworks zu erläutern.

Die Entwicklung einer nativen App und einer plattformübergreifenden App unterscheidet sich grundlegend dadurch, dass man bei einer nativen App sowohl für Android- als auch für iOS-Apps eine eigene App mit unterschiedlichen Programmiersprachen und Entwicklungsumgebungen entwickeln muss, wohingegen bei der Entwicklung mit einem Cross-Plattform-Framework für Android und auch iOS eine Codebasis benutzt werden kann.

Während es für die native Entwicklung vorgesehene Entwicklungsumgebungen wie XCode und Android Studio gibt, kann der Nutzer eines plattformübergreifenden Frameworks seine Entwicklungsumgebung frei bestimmen. So ist es beispielsweise möglich, die Anwendung mit dem kostenlosen Quelltexteditor *Visual Studio Code* von Microsoft [1] zu entwickeln. Eine wichtige Ergänzung ist, dass bei der Entwicklung für iOS immer ein Gerät mit dem Betriebssystem macOS verwendet werden muss.

2.3.1 Xamarin

Im Februar 2013 wurde Xamarin [2] als erstes plattformübergreifendes Framework veröffentlicht, das es Entwicklern erstmals ermöglichte, Android-, iOS- und OS-X-Apps zu entwickeln – mit nur einer Codebasis. Xamarin wurde 2011 gegründet und gehört seit 2016 zu Microsoft. Als Microsoft Xamarin kaufte, wurde .NET zu Xamarin hinzugefügt.

Für die Entwicklung mit Xamarin wird die Entwicklungsumgebung *Visual Studio* von Microsoft empfohlen, da es in der Entwicklungsumgebung bereits vorinstalliert ist. Die Anwendungen werden mit der Programmiersprache C# entwickelt und Xamarin unterstützt *XAML* (Extensible Application Markup Language).

Die Benutzeroberfläche einer Xamarin-Anwendung wird mithilfe von *Xamarin.Forms* und der *XAML*-Sprache erstellt. *Xamarin.Forms* ist ein Open-Source-Benutzeroberflächenframework. [3]

Mit *Xamarin.Android* und *Xamarin.iOS* wird der benutzergenerierte C#-Code in nativen Code kompiliert. Bei *Xamarin.Android* wird beispielsweise der C#-Code in die Zwischensprache IL kompiliert, die dann beim Start der Anwendung in eine native Assembly kompiliert wird. Bei *Xamarin.iOS* wird der C#-Code bereits beim Kompilieren in einen nativen ARM-Assembly-Code kompiliert, der dann direkt auf den Geräten ausgeführt werden kann. [3]

Xamarin.Essentials ermöglicht die Verwendung von nativen Gerätefeatures in einer Xamarin-Anwendung. Zu den verfügbaren Funktionen gehören:

- Geräteinformationen
- Dateisystem
- Beschleunigungsmesser
- Wählhilfe
- Text-zu-Sprache
- Bildschirmsperre

2.3.2 Ionic

Das Open-Source-Webframework *Ionic* wurde im November 2013 von dem Unternehmen Drifty Co veröffentlicht. Die Entwicklung von Ionic-Anwendungen basiert auf *HTML5*, *CSS*, *Sass* und *JavaScript/TypeScript*.

Mit der Veröffentlichung der Version 2 des Ionic-Frameworks am 1. April 2017 wurde das Framework überarbeitet und basiert seit dem auf dem Typescript-basierendem Front-End-Webapplikationsframework *Angular*. Bei diesem Übergang wurde auch der Wechsel von *Javascript* zu *TypeScript* durchgeführt.

Mit der Veröffentlichung der Version 4 des Frameworks am 23. Januar 2019 wurde die Multi-Framework-Kompatibilität des Frameworks implementiert. Ab dieser Version war es dann möglich, Ionic-Anwendungen nicht nur mehr mit Angular zu erstellen, sondern Entwicklern wurde auch die Frameworks *React* und *Vue* zum Entwickeln von Ionic-Anwendungen zur Verfügung gestellt. [4]

Das Ionic-Framework verwendet *HTML* und *CSS* für die Benutzeroberfläche, was es Benutzern in der Webentwicklungsbranche leicht macht, die Benutzeroberfläche der Anwendung zu erstellen. Das Framework bietet den Benutzern eine Menge von Angular Komponenten, um die Anwendung so aussehen zu lassen, als wäre sie mit einer nativen Programmiersprache entwickelt worden. [5]

2.3.3 React Native

Nachdem Facebook den großen Erfolg von React in der Webentwicklung gesehen hatte, beschloss Facebook, seinen Einfluss im Mobilfunkgeschäft auszuweiten. React Native startete 2013 als internes Hackathon-Projekt auf Facebook und hatte ursprünglich das Ziel, den Entwicklungsprozess für iOS und Android zu vereinheitlichen. [6]

React unterstützt natives JavaScript und neuerdings auch TypeScript als Programmiersprache. Die Benutzeroberfläche einer React Native-Anwendung ist in *JSX* implementiert. Im Gegensatz zu Ionic verwendet React Native nicht nur HTML-Seiten im Hintergrund, die über WebView gerendert werden, sondern übersetzt die Anweisun-

gen auch beim Kompilieren des Codes in native UI-Elemente. Dies macht es schwierig, eine React-Native-Anwendung von einer nativen Anwendung zu unterscheiden.[7]

Ein Vorteil der Verwendung von React Native besteht darin, dass es eine große Community gibt, die bereits sehr viele fertige Komponenten erstellt hat. Dadurch kann man sehr schnell eine funktionierende Anwendung erstellen. Man kann außerdem auf Hardware Sensoren zugreifen. [8] Die verfügbaren Sensoren sind:

- Accelerometer,
- Barometer,
- Gyroscope,
- Magnetometer,
- MagnetometerUncalibrated,
- Pedometer

Es ist auch möglich, nativen Code in eine React-Native-Anwendung zu integrieren. Dies erhöht die Flexibilität der Anwendung.

2.3.4 Flutter

Flutter ist Googles Antwort auf das Cross-Plattform-Framework von Facebook und wurde 2017 von Google veröffentlicht. Flutter-Anwendungen können nicht nur als Android- und iOS-Anwendungen erstellt werden, sondern es ist auch möglich, Flutter-Anwendungen als Web-App, Windows-, Linux-, maxOS- oder auch Google Fuchsia-Anwendungen zu erstellen. Das Framework ist selbst in C++ geschrieben und verwendet im Hintergrund die *Dart Virtual Machine*. Entwickler schreiben Flutter Apps in der *Dart* Programmiersprache. Der Vorteil von Dart gegenüber anderen Programmiersprachen liegt darin, dass Dart einen Transpiler besitzt, der den Quelltext in JavaScript übersetzt und dadurch die Web-Anwendungen ermöglicht. [9]

Bei Flutter ist die Benutzeroberfläche über Widgets aufgebaut und nutzt eine eigene Grafik-Engine im Hintergrund, um diese Widgets auf den Bildschirm zu zeichnen. Anstatt sich auf eine Reihe nativer Komponenten zu verlassen, verwendet Flutter eigene Widgets für Material Design (Android) und Cupertino (iOS). Ein Widget gruppiert Logik, Interaktion und Visualisierung innerhalb eines Objekts. Es gibt eine Reihe vorkonfigurierter Widgets, die bekannte und häufig benötigte Interaktionen wie Schaltflächen, Listen, Kontrollkästchen usw. anzeigen.

2.3.5 Trends der Cross-Plattform-Frameworks

Die plattformübergreifende Entwicklung hängt stark von der Beteiligung Dritter ab und ist daher immer von einer großen Nutzer- oder Community-Basis abhängig. Ohne diese aktive und fürsorgliche Community wird das Framework oder die Technologie heute unweigerlich in einen Abwärtstrend rutschen, da einige Funktionen und Support nicht mehr gewährleistet werden kann.

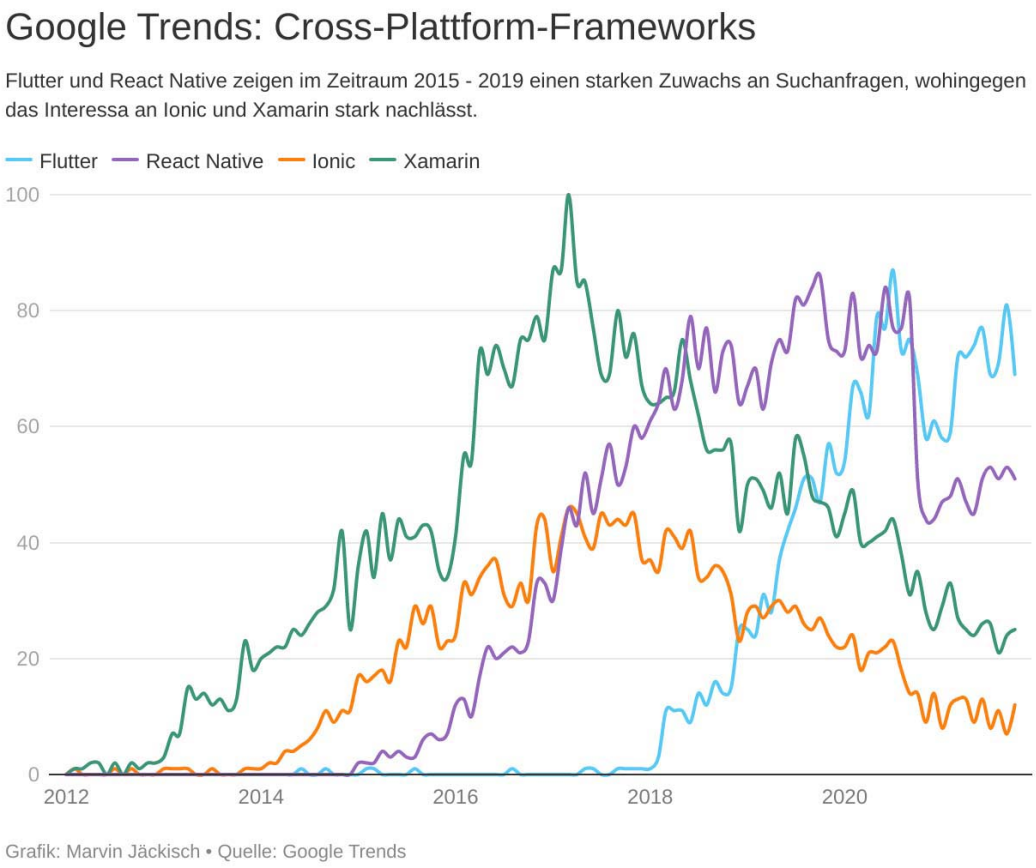

Abbildung 2.1: Google Trends zu Flutter, React Native, Ionic und Xamarin [10]

Abbildung 2.1 zeigt deutlich, dass das Interesse an den beiden Frameworks Ionic und Xamarin in der Google-Suche abnimmt. Beide Frameworks sind noch nicht sehr alt, aber da React Native und Flutter einen viel direkteren und moderneren Ansatz zur Lösung des Problems der plattformübergreifenden Entwicklung verfolgt haben, ersetzen sie nach und nach die anderen Frameworks.

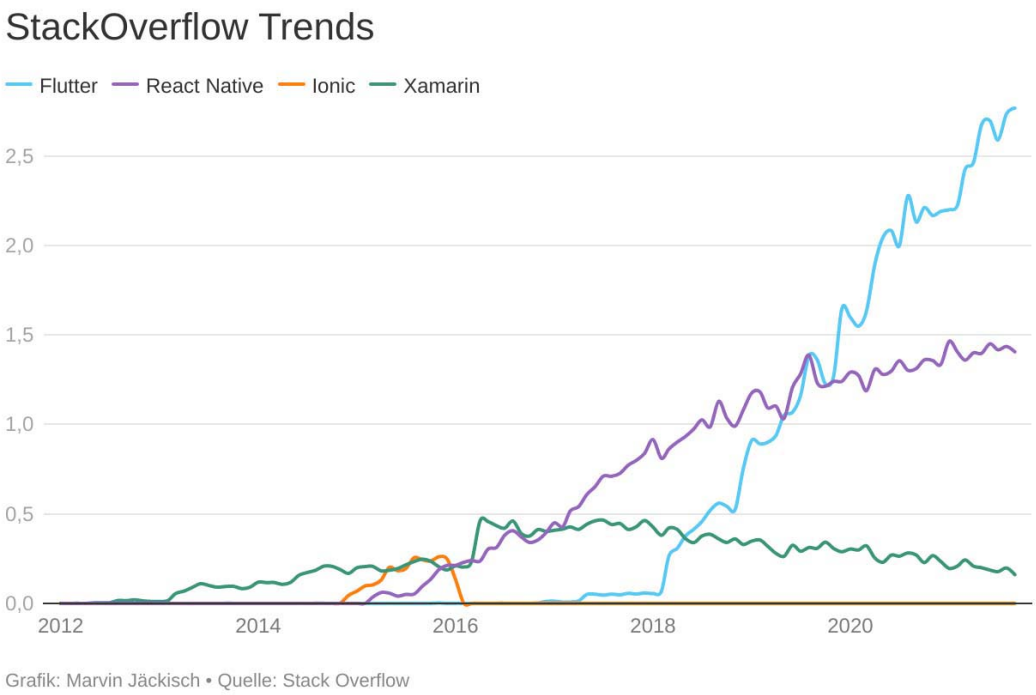

Abbildung 2.2: Stack Overflow Trends zu Flutter, React Native, Ionic und Xamarin [11]

In Abbildung 2.2 sieht man auch, dass bei den Stack Overflow [12] Fragen seit der Veröffentlichung des plattformübergreifenden Frameworks Flutter oder React Native relativ schnell eine höhere Nachfrage nach den Frameworks entstand als bei den Frameworks Ionic und Xamarin.

2.4 Smartwatches

Die beiden Haupttypen von Smartwatches, *Apple Watch* und *Wear OS*, teilen viele ihrer Funktionen. Die grundlegendste Funktionalität ist die Zeitanzeige und Benutzer können aus einer Bibliothek von vielen Watch-Faces wählen, welche noch andere Daten anzeigt, zum Beispiel einen Schrittzähler. Wenn die Apple Watch nicht verwendet wird, schaltet sie ihren Bildschirm aus, um Akkulaufzeit zu sparen, und beleuchtet den Bildschirm nur, wenn sie eine Handbewegung des Benutzers erkennt oder der Benutzer den Bildschirm berührt. Jede Uhr ist mit dem Telefon eines Benutzers gekoppelt und überträgt Informationen vom Telefon an die Uhr. Am Telefon empfangene Benachrichtigungen (von SMS, Telefonanrufen oder Anwendungen) werden auf die Uhr gespiegelt und erzeugen dort einen Ton oder eine kleine haptische Vibration am Handgelenk des Trägers. Wenn der Benutzer in wenigen Sekunden die Hand hebt, werden die Benachrichtigungsdetails angezeigt. Nach dem Lesen werden die Benachrichtigungen zu Referenzzwecken auf der Uhr gespeichert (im "Notification Center"), bis der Benutzer sie löscht. Sowohl Google- als auch Apple-Smartwatches ermöglichen es Benutzern, Nachrichten zu senden, Anwendungen zu starten oder andere Aufgaben mit Sprachbefehlen auszuführen, und andere Anwendungen können aus dem App Store heruntergeladen werden. Schließlich werden die Schritte und die Anstrengung des Benutzers von einer Vielzahl von Sensoren gemessen, die die körperliche Aktivität überwachen. Außerdem kann eine Smartwatch verwendet werden, um seine Schlafaktivität zu tracken und sich einen Überblick über sein Schlafverhalten zu verschaffen. [13]

2.4.1 Apple Watch

Die *Apple Watch* der ersten Generation wurde am 9. September 2014 vorgestellt und ist seit dem 24. April 2015 erhältlich. Die erste *Apple Watch* hatte folgende Sensoren verbaut, mit denen der Benutzer seine Aktivitäten messen konnte: [14]

- Herzfrequenzmesser
- Beschleunigungssensor
- Gyrosensor
- Umgebungslichtsensor

Die 2. Generation der *Apple Watch* wurde am 7. September 2016 vorgestellt und waren ab dem 16. September 2016 erhältlich. Sie zeichnete sich vor allem durch einen verbesserten Dual-Core-Prozessor und der GPS-Integration aus. Zur 2. Generation zählen die *Apple Watch Series 1* und *Series 2*. [15]

Am 12. September 2017 wurde dann die *Apple Watch Series 3* vorgestellt, welche ab dem 22. September 2017 erhältlich war. Die wichtigste Neuerung der 3. Generation war die Möglichkeit, mit einer eSim eine LTE-Verbindung mit der *App Watch* herstellen zu können. Dadurch wurde ermöglicht, beispielsweise Telefonate auch ohne eine Verbindung mit einem iPhone zu tätigen. Es wurden außerdem folgende Sensoren und Features hinzugefügt: [16]

- Barometrischer Höhenmesser
- Optischer Herzsensor
- Notruf SOS
- Lautsprecher
- Mikrofon
- Apple Pay

Mit der 4. Generation der *Apple Watch*, bei der die *Apple Watch Series 4* am 12. September 2018 vorgestellt und am 21. September 2018 erhältlich war, wurde an der *Apple Watch* das Display verändert, wodurch je nach Ausführung anstelle eines 38 mm bzw. 42 mm großem Display, jetzt ein 40mm bzw. 44 mm großes Display verbaut wurde. Außerdem wurden Sensoren zur Messung von Elektrokardiogrammen (EKG) verbaut, wodurch Hinweise auf Herzprobleme, wie beispielsweise Vorhofflimmern erkannt werden können. Außerdem kann die *Apple Watch* seit dem Stürze erkennen und falls es eingestellt wurde, auch bei einem Sturz automatisch den Notruf wählen. [17]

Die *Apple Watch Series 5* wurde am 10. September 2019 vorgestellt und war ab dem 20. September 2019 erhältlich. Bei der 5. Generation war die wichtigste Neuerung der neu verbaute Kompass und das Always-on-Display. Durch das neue Display wird das Display dauerhaft angezeigt, auch wenn der Benutzter das Display nicht angetippt hat oder seinen Arm nicht bewegt hatte. Um die Akkulaufzeit zu verlängern, werden Helligkeit und Bildwiederholfrequenz reduziert, wenn das Display nicht verwendet wird. [18]

Die 6. Generation wurde am 15. September 2020 vorgestellt und war ab dem 18. September 2020 erhältlich. Bei dieser Generation wurden zwei Geräte vorgestellt, die *Apple Watch Series 6* und die *Apple Watch SE*. Bei der *Apple Watch Series 6* wurde ein neuer Sensor verbaut, der Blutsauerstoffsensor, und das Always-on-Display wurde verbessert. [19] Mit der *Apple Watch SE* wurde eine günstigere Alternative zur *Apple Watch Series 6* angeboten. Hierbei verzichtet sie auf das Always-on-Display, das EKG und den Blutsauerstoffsensor. [20]

2.4.2 Smartwatches mit Wear OS

Wear OS ist ein Betriebssystem des Unternehmens *Google LLC*. Im Juni 2014 wurde es im Namen *Android Wear* veröffentlicht. Am 16. März 2018 wurde es von *Android Wear* zu *Wear OS by Google* umbenannt. [21]. 2021 vereinbarten Google und Samsung eine Kooperation, wodurch die Betriebssysteme *Wear OS* und *Tizen* in Zukunft zu *Wear* verschmelzen werden. Dadurch sollen in Zukunft mehrere Verbesserungen mit sich kommen, beispielsweise längere Akkulaufzeit, bessere Performance des Betriebssystems, einfachere Systemnavigation und verbessertes Tracking von Gesundheits- und Fitness-Daten. [22]

Android Wear unterstützt viele Sensoren, darunter Beschleunigungs- und Neigungssensoren zur Bewegungssteuerung sowie Sensoren zur Messung von Luftdruck und Herzfrequenzsensoren. Die Verbindung mit dem Smartphone erfolgt mit Bluetooth, teilweise auch mit WLAN und Mobilfunknetz. Geräte mit einer Anbindung an das Mobilfunknetz können auch ohne Smartphone genutzt werden zum Beispiel können sie ohne Smartphone einen Telefonanruf ausführen. Einige Modelle sind mit GPS- und NFC-Technologie kompatibel.

2.5 Firebase

Firebase ist ein Dienst von *Google LLC* und ist eine Art Backend-as-a-Service. Mit einem Backend-as-a-Service lagern Sie die Verantwortung für den Betrieb und die Wartung von Servern an einen Dritten aus und konzentrieren sich auf die Frontend- oder Client-seitige Entwicklung. Darüber hinaus bietet ein Backend-as-a-Service Tools, mit denen Backend-Code erstellt und der Entwicklungsprozess beschleunigt werden kann. Es verfügt über einsatzbereite Funktionen wie skalierbare Datenbanken, APIs, Cloud-Code-Funktionen, Social-Media-Integrationen, Dateispeicherung und Push-Benachrichtigungen. [23]

Firebase zielt darauf ab, drei Hauptprobleme für Entwickler zu lösen:

- Schnell eine App erstellen
- Eine App veröffentlichen und überwachen
- Benutzer-Authentifizierung und Kundenbindung zu erlangen

Zu den Funktionen von Firebase gehören Datenbanken, Authentifizierung, Push-Nachrichten, Analysen, Dateispeicher und vieles mehr (siehe Abbildung 2.3). [24]

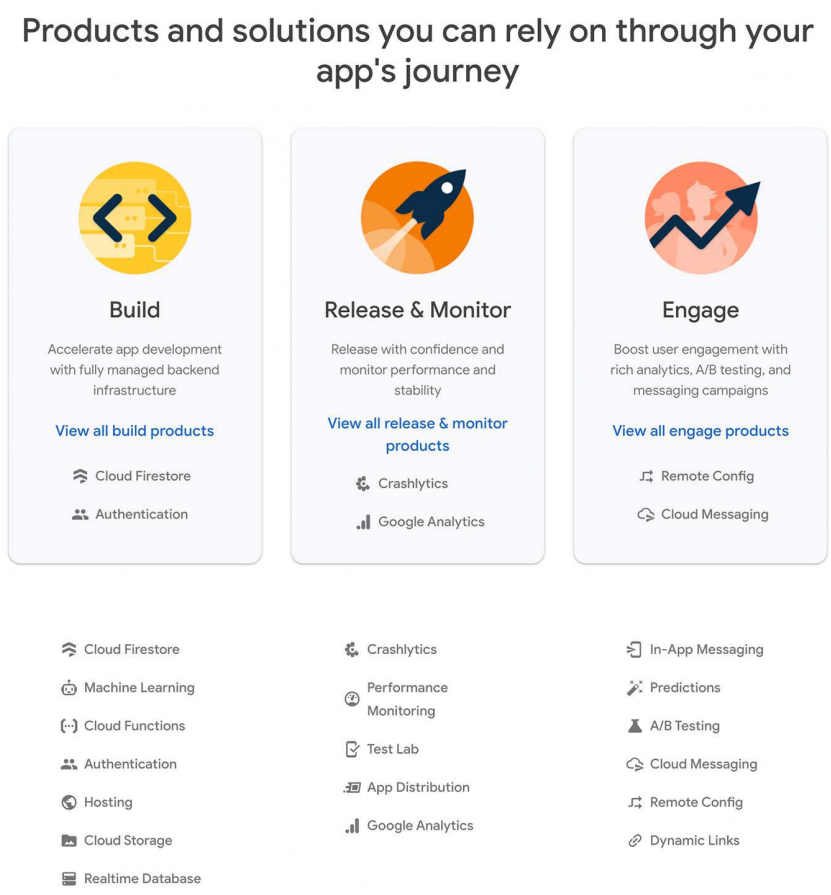

Abbildung 2.3: Einen Überblick über die Funktionen von Firebase [24]

2.5.1 Firebase Authentication

Mit Firebase können Nutzerregistrierungen und Loginfunktionalitäten ermöglicht werden. Dabei gibt es vorgefertigte SDKs und UserInterface-Bibliotheken, die der Entwickler neben eigenen Authentifizierungsmethoden verwenden kann. Durch die Bereitstellung dieser Funktion spart sich der Entwickler einen großen Anteil an Programmierleistungen, um ein eigenes Authentifizierungssystem einzurichten. Ebenfalls minimiert sich der Verwaltungsaufwand extrem, denn über das System können Konten zusammengeführt oder auch deaktiviert werden und das ohne großen Mehraufwand.

Abbildung 2.4: Firebase Authentifizierungsmöglichkeiten [25]

Wie in Abbildung 2.4 zu sehen bietet Firebase folgende Authentifizierungsmöglichkeiten:

- Anonym (Gast)
- E-Mail (Passwort)
- Facebook
- Telefonnummer
- Twitter
- Google
- GitHub

2.5.2 Cloud Firestore

Firestore dient dem Speichern und Synchronisieren von Daten zwischen Benutzern und Geräten mit Hilfe einer cloudgehosteten NoSQL-Datenbank. Cloud Firestore bietet Livesynchronisation und Offlinesupport sowie effiziente Datenabfragen.

Die Hauptmerkmale von Firestore sind:

- Flexibilität
  - Das Cloud Firestore-Datenmodell unterstützt flexible, hierarchische Datenstrukturen. Die Daten können in Dokumente und Sammlungen organisiert werden. Außerdem sind verschachtelte Untersammlungen möglich.
- Expressive Abfrage
  - In Cloud Firestore kann man Abfragen verwenden, um einzelne, bestimmte Dokumente oder alle Dokumente einer Sammlung abzurufen, welche den Abfrageparametern entsprechen. Die Abfragen können mehrere verkettete Filter enthalten, sowie Filter und Sortierungen kombinieren.
- Echtzeit-Updates
  - Cloud Firestore verwendet die Datensynchronisierung, um Daten auf jedem verbundenen Gerät zu aktualisieren.
- Offline-Unterstützung
  - Cloud Firestore speichert die Daten im Cache des Geräts, sodass die App Daten schreiben, lesen und abfragen kann, selbst wenn das Gerät offline ist. Wenn das Gerät wieder online ist, synchronisiert Cloud Firestore alle lokalen Änderungen mit Cloud Firestore zurück.

2.5.3 Cloud Storage

Cloud Storage dient dazu, Inhalte wie Bilder, Audio und Video in einem einfachen Objektspeicher abzulegen. Firebase verwendet ein einfaches Verzeichnis-/Dateisystem, um seine Daten zu strukturieren. Firebase führt Uploads und Downloads unabhängig von der Netzwerkqualität durch.

Außerdem lässt sich Cloud Storage in Firebase Authentication integrieren, damit Entwickler den Zugriff gestalten können, ohne die Privatsphäre der Benutzer zu verletzen. [26]

2.6 Ähnliche Trainingsapps

Im folgenden werde ich bereits bestehende Trainingsapps auf dem Markt untersuchen und den Vergleich zur 33-Tage Challenge herstellen.

2.6.1 Freeletics

Die App Freeletics wurde von der Freeletics GmbH entwickelt und hat weltweit bereits über 52 Millionen Benutzer.

Freeletics kommt als App auf dem Smartphone zum Einsatz. Zum einen gibt es die Möglichkeit Freeletics kostenlos zu nutzen. Hier finden Sie zum Beispiel eine Auswahl verschiedener Ausbildungen, die jeweils nach griechischen Göttern und Göttinnen benannt sind. Auf der anderen Seite gibt es das kostenpflichtige „Coach"-Feature, das der eigentliche Vorteil von Freeletics ist.

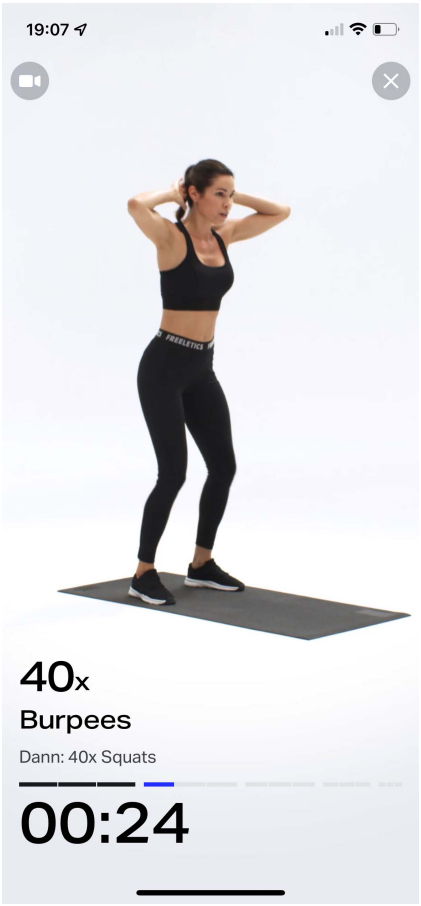

Abbildung 2.5: Das Training bei *Freeletics*

Mittlerweile gibt es über 90 Gods Workouts. Das Prinzip der Workouts ist immer, die angegebenen Übungen so sauber und schnell wie möglich zu absolvieren. Generell sind die Workouts recht abwechslungsreich. Der Benutzer kann beispielsweise auswählen, welche Körperregionen trainiert werden sollen, wie anspruchsvoll das Training sein soll, und er kann auch nach Trainingsdauer oder Schwierigkeitsgrad filtern. Es besteht auch die Möglichkeit, nur mit dem Körpergewicht zu trainieren oder alternativ Trainingsgeräte oder Laufsequenzen zu integrieren.

Mit den Trainings Journeys wird der Anwender für 6, 8 oder 12 Wochen an die Hand genommen. Die Reisen zielen auf unterschiedliche Fitnessziele ab, zum Beispiel in Form kommen, Gewicht verlieren, Muskeln aufbauen, den Körper straffen oder die Ausdauer steigern. Der Trainer berechnet das anfängliche Training basierend auf Informationen wie Fitnesslevel, Alter sowie Größen- und Gewichtsinformationen. Im weiteren Verlauf wird das Training dank künstlicher Intelligenz an die Leistung und den Fortschritt des Benutzers angepasst. [27]

2.6.2 adidas Training by Runtastic

Die adidas Training by Runtastic App wurde anfänglich unter dem Namen Runtastic Results veröffentlicht und wurde von der Firma Runtastic GmbH entwickelt. 2015 übernahm Adidas das Unternehmen und die Runtastic GmbH wurde dann eine Tochtergesellschaft von Adidas. Daraufhin wurde die Runtastic Results auch in adidas Training by Runtastic umbenannt.

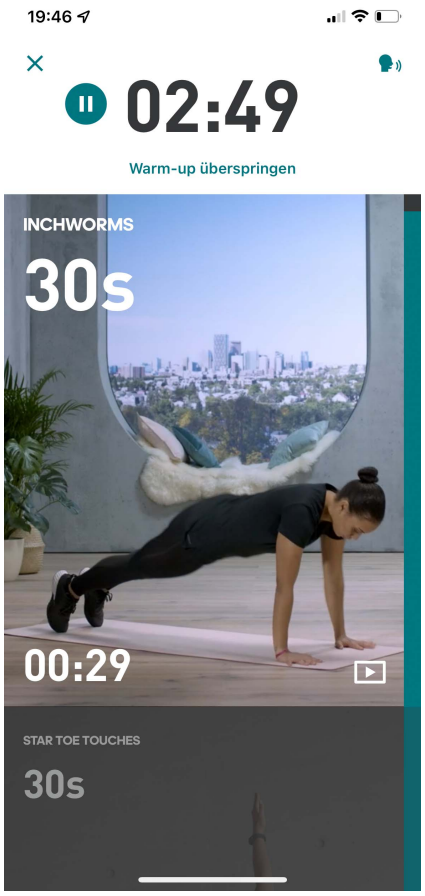

Abbildung 2.6: Das Training bei *adidas Training by Runtastic*

Die adidas Training by Runtastic App bietet Trainingspläne mit derzeit mehr als 190 verschiedenen Übungen, die alle ausschließlich mit dem eigenen Körpergewicht durchgeführt werden können, ohne Gewichte oder zusätzliche Geräte. Die einzelnen Workouts im persönlichen Trainingsplan bestehen aus mehreren Übungen.

Ein auf den Benutzer zugeschnittener Trainingsplan begleitet ihn 12 Wochen. Die Workouts bestehen aus Übungen, für die man nur das eigene Körpergewicht braucht. Die einzelnen Workouts dauern zwischen 15 und 45 Minuten und können immer und überall absolviert werden. Es gibt mehr als 180 Videos, die dem Benutzer dabei helfen, die Übungen korrekt nachzumachen. Auf Basis des Benutzerfeedbacks über die Vorwoche wird der Trainingsplan immer wieder neu für den Benutzer zusammengestellt.

Zusätzlich kann man sich gezielt Pläne für einzelne Körperbereiche erstellen lassen unter der Berücksichtigung der Zeit, die man hierfür aufwenden möchte, beispielsweise ein 20-Minuten-Programm nur für die Beine. Es wird außerdem noch ein Gesundheits- und Ernährungsguide angeboten, der den Benutzer auch außerhalb dem Training begleitet. [28]

2.6.3 Vergleich zur 33-Tage Challenge

Beim Blick auf die zuvor untersuchten Trainings-Apps wird deutlich, dass die 33-Tage-Challenge einige Features haben soll, die in den Trainings-Apps nicht vorhanden sind. Einerseits bot keine der untersuchten Apps eine After-Workout-Mahlzeit nach dem Training an. Mit der adidas Training by Runtastic App ist es möglich Ernährungstipps zu bekommen, Ernährungspläne oder ähnliches gibt es jedoch nicht.

In der 33-Tage-Challenge-App spielt die Kalorienberechnung eine große Rolle. Schaut man sich die Freeletics-App im Vergleich an, wird man feststellen, dass es bei diesem Training keine Art der Nachverfolgung des Kalorienverbrauchs gibt. Mit der adidas Training by Runtastic App ist es möglich die Anzahl der verbrauchten Kalorien nach dem Training anzuzeigen, diese wird jedoch nur mit Hilfe der Nutzerdaten, also Gewicht, Alter, etc. berechnet. Mit der 33-Tage-Challenge ist es möglich um das Training mit einer Smartwatch zu verfolgen und sich nach dem Training weitere Statistiken anzeigen zu lassen.

2.7 Kalorienverbrauchsberechnung

Zur Kalorienverbrauchsberechnung gibt es zwei Möglichkeiten, wie man sie mit Hilfe eines Smartphones berechnen kann. Dabei ist die erste Möglichkeit, eine Smartwatch zu benutzen, beispielsweise die *Apple Watch*.

2.7.1 Kalorienverbrauchsberechnung mit Smartwatch

Das Kalorien-Tracking der Apple Watch basiert auf einem wissenschaftlichen Prinzip namens Stoffwechselrate. Die Stoffwechselrate ist die Rate des Stoffwechsels einer Person oder wie schnell der Körper die durch die Nahrungsaufnahme aufgenommenen Kalorien verbraucht.

Jeder Mensch hat eine einzigartige Stoffwechselrate, aber Wissenschaftler können die Stoffwechselrate einer Person basierend auf ihrem Geschlecht, Gewicht und ihrer Größe genau vorhersagen.

Wenn man die Apple Watch einrichtet, gibt man diese Informationen in der Health-App an. Diese persönlichen Informationen werden dann verwendet, um den Grundumsatz, den Stoffwechsel unter normalen Ruhebedingungen, und den aktiven Stoffwechsel, den Stoffwechsel während des Trainings, zu berechnen. Die Apple Watch verwendet dann Herzfrequenz und Bewegung, um eine Zunahme oder Abnahme der Aktivität zu erkennen, indem sie ihren aktiven Stoffwechsel misst. Der aktive Stoffwechsel entspricht dann den während des Trainings verbrauchten Kalorien. [29]

2.7.2 Kalorienberechnung mit MET-Faktor

Eine weitere Möglichkeit der Kalorienverbrauchsberechnung ist die Berechnung der Aktivität mit Hilfe des MET-Faktors. Grundlage der Berechnung der körperlichen Aktivitäten sind die metabolischen Äquivalente (MET). Diese wurden von der University

of South Carolina ermittelt. [30] Die METs drücken die Sauerstoffaufnahme für all diese Aktivitäten aus. Als Basis für die Höhe der Sauerstoffaufnahme dient 1 MET, das etwa $3,5\ ml^{-1} \cdot kg^{-1} \cdot min^{-1}$ beträgt und der Sauerstoffaufnahme beim ruhigen Sitzen entspricht. Ein MET entspricht damit in etwa dem Grundumsatz.

Der Kalorienverbrauch wiederum hängt direkt von der Sauerstoffaufnahme ab, sodass mit Hilfe des MET-Faktors der Kalorienverbrauch mit folgender Formel berechnet werden kann:

$$Kilokalorie = MET \cdot Gewicht\ in\ Kilogramm \cdot Dauer\ in\ Stunden \tag{2.1}$$

Wenn man beispielsweise den Kalorienverbrauch einer 70 kg schweren Person, bei einer mäßige Anstrengung (3.8 MET) für 30 Minuten berechnen will, würde das Ergebnis 133 kcal ergeben ($3,8 \cdot 70\ kg \cdot 0,5\ h$). [30] Diese bereits berechneten MET-Werte sind unter Compendium of Physical Activities verfügbar.

Kapitel 3

Anforderungsanalyse

In diesem Kapitel werden die grundlegenden Anforderungen dieser Bachelorarbeit vorgestellt. Die Anforderung an Software besteht aus zwei Anforderungen, die als funktionale und nicht-funktionale Anforderungen bezeichnet werden. Außerdem werden die Herausforderungen der 33-Tage-Challenge vorgestellt.

3.1 Funktionale Anforderungen

Ziel ist es, eine Anwendung zu entwickeln, die die folgenden Spezifikationen umsetzt.

Erstellung von Trainingsplänen, angepasst an das Benutzerlevel

Es sind Trainingspläne zu erstellen, die sich dem Fitnesslevel des Nutzers anpassen. Das Fitnesslevel legt der Nutzer bei der Registrierung selbst fest und kann es nachträglich in der App ändern.

Trainingsablauf anzeigen

Das Training sollte mit dem individuellen Trainingsplan erstellt werden. Zu Beginn werden alle Übungen dieses Trainings in einer Übersicht angezeigt und der Nutzer sieht zu allen Übungen einen erklärenden Text und ein Anschauungsvideo. Wenn das Training beginnt, werden ein Timer und ein Video der aktuellen Übung angezeigt. Der Timer sollte angehalten und Übungen sollten übersprungen werden können. Nach jeder Übung sollte eine Pause eingelegt werden, in der der Nutzer bereits die nächste Übung sehen kann.

Statistik über Training anzeigen

Am Ende eines Trainings soll sich der Nutzer einen Überblick über seinen Kalorienverbrauch, seine Herzfrequenz und die Dauer des Trainings verschaffen können. Nutzt der Benutzer eine Smartwatch, werden diese Daten über Apple Health / Google Fit übertragen. Andernfalls wird der Kalorienverbrauch anhand des MET-Faktors, der

Dauer der durchgeführten Übungen und des Gewichts des Benutzers berechnet (vgl. Abschnitt 2.7.2). Die Herzfrequenz kann in diesem Fall nicht angezeigt werden.

After-Workout-Mahlzeit erstellen

Nach dem Training sollte eine Auswahl an After-Workout-Mahlzeiten basierend auf den verbrauchten Kalorien sowie eine Zutatenliste und Zubereitungshinweise für jede Mahlzeit angezeigt werden. Die Mahlzeiten sollen auch im Nachhinein einsehbar sein.

Statistiken über 33-Tage anzeigen

Es soll eine Übersicht über alle Trainings der 33-Tage dargestellt werden können. Diese Übersicht soll verschiedene Daten beinhalten, die Anzahl der durchgeführten Trainings, die insgesamt verbrannten Kalorien und die durchschnittliche Herzfrequenz, sowie die durchschnittliche Dauer über alle Trainings. Außerdem sollen Diagramme über die verbrannten Kalorien, der Herzfrequenz und der Dauer über die Trainings in den 33 Tagen angezeigt werden.

Mediathek mit Videos und Anleitungen zur 33-Tage Challenge

Dem Nutzer soll eine Mediathek mit diversen Videos und Anleitungen für die 33-Tage-Challenge zur Verfügung stehen. Diese Videos sollen in einer Datenbank gespeichert und nicht auf dem Gerät gespeichert, sondern beim Anschauen heruntergeladen werden.

Anmeldung/Registrierung des Benutzers

Der Benutzer soll sich in der App mit seiner E-Mail registrieren/anmelden können.

Sicherung der Daten auf einer Datenbank

Alle Benutzerdaten sollen in einer Datenbank gespeichert werden, damit der Benutzer seine Daten bei einer Neuinstallation der Anwendung nicht verliert. Zu diesen Daten gehören die Nutzerdaten, also das Gewicht und Fitnesslevel des Nutzers, die Statistiken der einzelnen Trainingseinheiten und der zuletzt erstellte Trainingsplan.

3.2 Nicht-funktionale Anforderungen

Benutzerfreundlichkeit

Die Anwendung sollte ein möglichst einfaches, strukturiertes und benutzerfreundliches Layout haben. Die Nutzung der Webanwendung soll für die Nutzer intuitiv sein. Der Benutzer soll in der Lage sein, mit geringem Aufwand und in kurzer Zeit durch die Anwendung zu navigieren und die Funktionen der Anwendung auszuführen.

Plattformübergreifend

Die Anwendung sollte unabhängig von der verwendeten Plattform funktionieren. Der Benutzer sollte sowohl auf Android- als auch auf iOS-Geräten die gleiche Benutzererfahrung haben und alle Funktionalitäten sollten auf beiden Plattformen verfügbar sein.

Mehrsprachige Anwendung

Die Anwendung soll auch von Benutzern genutzt werden können, die kein Deutsch sprechen. Damit soll sichergestellt werden, dass diese Anwendung auch außerhalb Deutschlands genutzt werden kann. Die verfügbaren Sprachen sollen *Deutsch* und *Englisch* sein.

3.3 Herausforderungen der 33-Tage Challenge

Die Herausforderungen der 33-Tage-Challenge bestehen im Gegensatz zu den in Abschnitt 2.6 vorgestellten Apps zum einen darin, den Nutzer für die 33 Tage aktiv zu halten. Bei *Freeletics* zum Beispiel ist es dem Nutzer selbst überlassen, wann er sein nächstes Workout macht und wie oft er beispielsweise ein Workout wiederholt. Die 33-Tage-Challenge sieht vor, dass der Benutzer jeden zweiten Tag trainieren sollte.

Eine weitere Herausforderung besteht darin, einen individuellen Trainingsplan für den Benutzer zu erstellen. Bei den anderen Apps ist dies nicht der Fall. Außerdem soll für die 33-Tage-Challenge eine After-Workout-Mahlzeit kreiert werden. Die vorgestellten Apps bieten keinerlei After-Workout-Mahlzeiten an.

Zudem besteht im Gegensatz zu den vorgestellten Apps eine Herausforderung darin, die Trainingsstatistiken über die Smartwatch abzurufen, bzw. diese zu berechnen, falls keine Smartwatch verfügbar ist. Im Gegensatz zur *Freeletics*-App bietet die *adidas Training by Runtastic*-App an, den Kalorienverbrauch während dem Training zu berechnen.

Kapitel 4

Vorgehen

4.1 Arbeitsweise

Um eine Übersicht über das Vorgehen des Projekts zu erhalten, wird in diesem Projekt die GitHub Issues verwendet. Ein Issue ist in der Regel dazu gedacht, um beispielsweise in einem Open-Source-Projekt den Entwickler der Anwendung über gefundene Fehler in der Anwendung zu informieren.

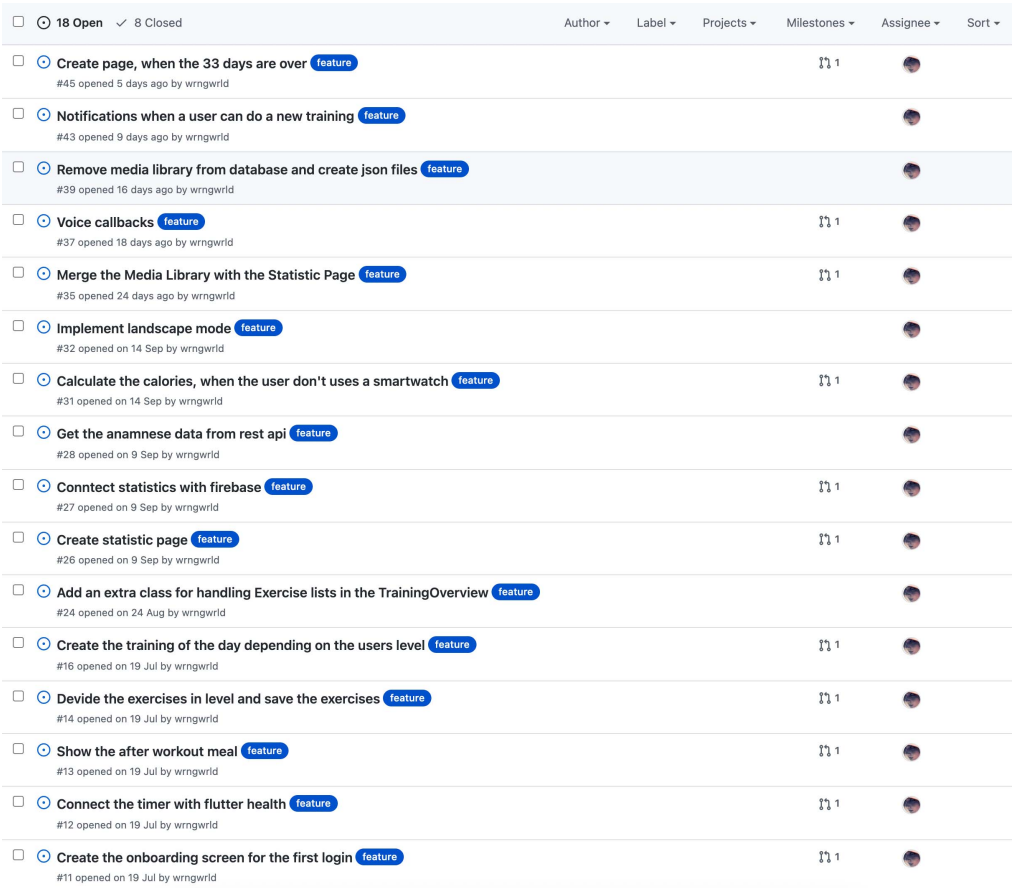

Abbildung 4.1: Ein Ausschnitt der Issues auf GitHub

Es kann jedoch auch dafür verwendet werden, seine Anwendung in verschiedene Teilaufgaben zu unterteilen. Dabei kann man für jede Teilaufgabe ein eigenes Issue anlegen. Jedem Issue kann man eine Beschreibung hinzufügen, eine verantwortliche Person festlegen und ein Label anlegen. Labels werden dazu verwendet, schnell zu erkennen, was für eine Art von Teilaufgabe dieses Issue ist. Beispielsweise kann man einem Issue ein Label *bug* geben, um zu erkennen, dass es sich in diesem Issue um einen Bug handelt.

Wie in Abbildung 4.1 zu sehen, habe ich in meinem Projekt einigen Issues das Label *feature* gegeben. Außerdem ist bei der Spalte *Assignee* der Verantwortliche dieses Issues angegeben.

4.2 Git-Workflow

In dieser Arbeit wurde das Programm Git als Basis für die Versionsverwaltung verwendet. Ein häufig verwendeter Workflow ist Git-Workflow. [31] In Abbildung 4.2 ist ein Übersicht über den Git-Workflow zu sehen.

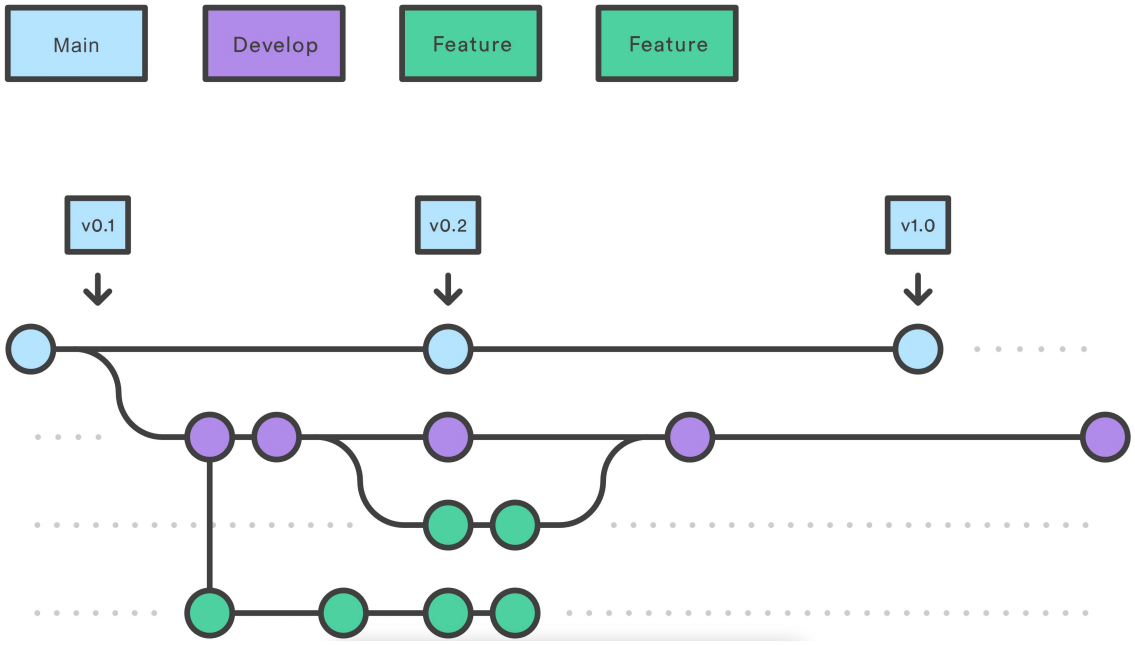

Abbildung 4.2: Übersicht über Git-Workflow [31]

Der *main*-Branch des Projekts enthält die gesamte Änderungshistorie und jeder *Commit* erhält die Versionsnummer als „Label“. Der Stand des *main*-Branches ist der Stand, der im Produktionssystem sichtbar ist.

Ein *develop*-Branch wird für die Entwicklung erstellt. Dieser dient als Branch für die Zusammenführung der einzelnen *feature*-Branches. Ein Feature ist wie im vorherigen Abschnitt beschrieben, eine einzelne Teilaufgabe des Projektes. Bei meinem Projekt habe ich für jedes Feature ein eigenes Issue auf GitHub angelegt. Für jedes Feature wird ein eigener Branch angelegt. Auf diesen Branch werden alle Änderungen committed, die zu diesem Feature gehören. Ein *feature*-Branch interagiert nie direkt mit dem *main*-Branch.

Sobald der Entwickler mit der Entwicklung des Features fertig ist, erstellt er auf GitHub einen Pull-Request, wodurch er eine Integration dieses *feature*-Branches auf den *develop*-Branch anfordert. Dieser Pull-Request wird dann von einem anderen Entwickler überprüft und es wird ein Code-Review durchgeführt. Sollten auf diesem Branch von dem anderen Entwickler Fehler gefunden worden sein, müssen diese korrigiert und eine erneute Überprüfung angefordert werden. Sollten auf diesem Branch keine Fehler mehr gefunden werden, kann dieser Branch in den *develop*-Branch integriert werden. [31]

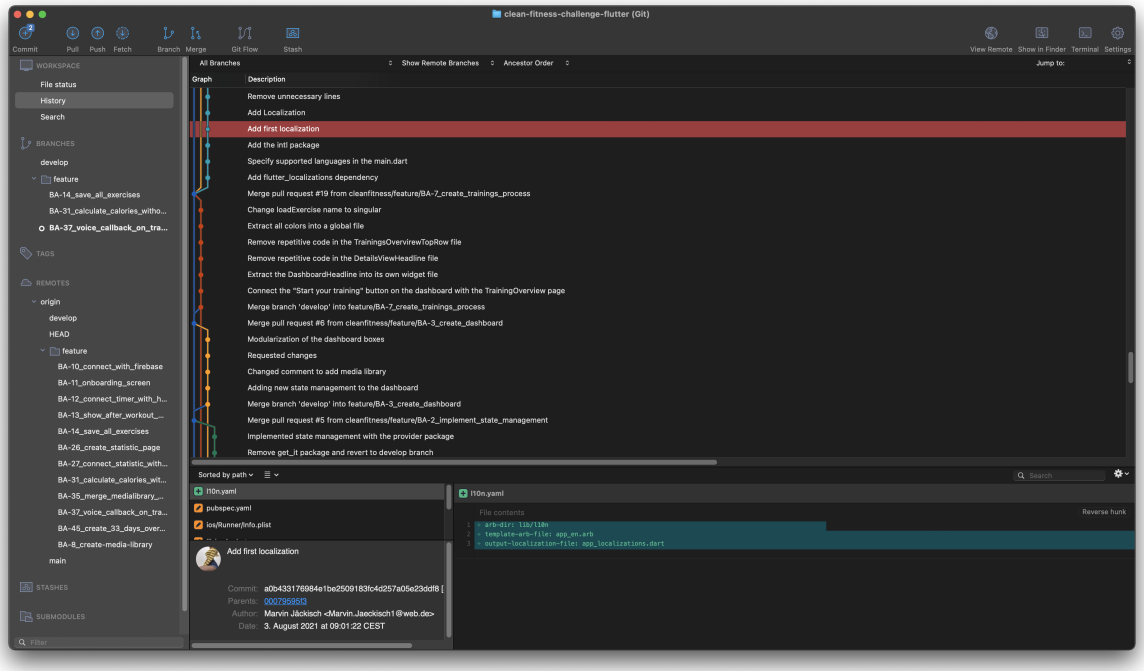

Abbildung 4.3: Ausschnitt aus der Änderungshistorie des Projektes von Sourcetree

4.3 Entwicklungsumgebung

In einem Flutter-Projekt gibt es zwei mögliche Entwicklungsumgebungen, in denen die Anwendung entwickelt werden kann, *Android Studio* [32] und *Visual Studio Code* [1]. Bei beiden Entwicklungsumgebungen sind Plugins verfügbar, welche die Entwicklung einer Flutter-App unterstützen. In diesem Projekt wurde das Programm *Visual Studio Code* verwendet (siehe Abbildung 4.4).

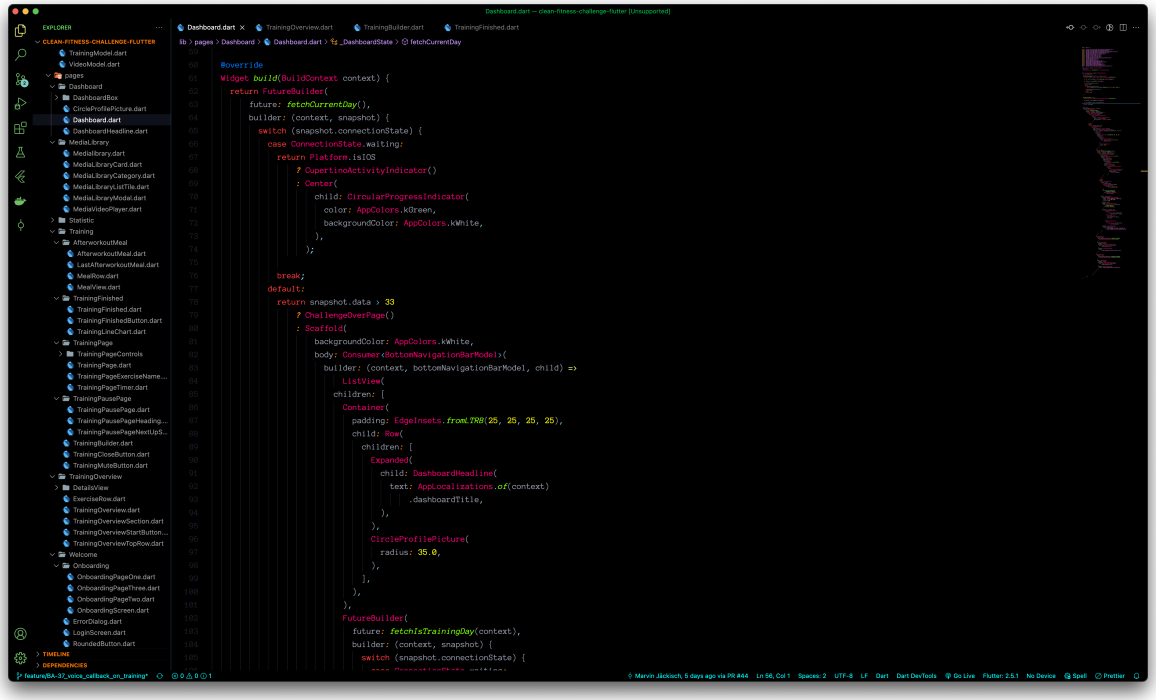

Abbildung 4.4: Screenshot der Entwicklungsumgebung *Visual Studio Code*

Zur Versionsverwaltung wurde das Programm Sourcetree [33] verwendet. In Abbildung 4.3 sieht man einen Ausschnitt der Änderungshistorie in diesem Programm. Sourcetree wurde hier verwendet, da der Entwickler dieser Anwendung außerdem der Entwickler des Git-Workflows ist. Daher wurden hier Funktionen verbaut, die diesen Workflow unterstützen. Zum Beispiel wird dem Benutzer in diesem Programm das Erstellen von neuen Features und Releases abgenommen, wie in Abbildung 4.5 zu sehen ist. Dadurch können auch beispielsweise Fehler durch Erstellen von Branches vermieden werden.

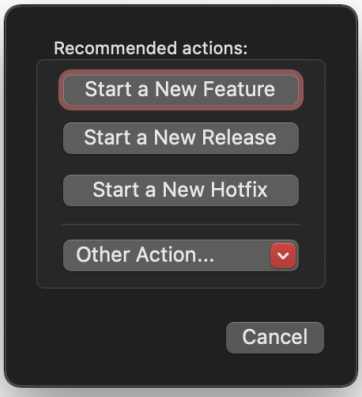

Abbildung 4.5: Unterstützung des Git-Workflows durch Sourcetree

Kapitel 5

Implementierung

In diesem Kapitel findet man nähere Informationen zur Umsetzung der einzelnen Anforderungen. Bei der Implementierung wurde sich an den definierten Anforderungen im Kapitel 3 gehalten.

5.1 State Management

Im weitesten Sinne besteht der App-Status aus allem, was während der Ausführung der App in den Speicher geladen wird, einschließlich der Assets der App und der Informationen zum Framework, wie Benutzeroberfläche, Animationsstatus, Schriftarten usw.

Diese werden jedoch bereits vom Flutter-Framework verarbeitet. Wenn man also über das State Management spricht, bezieht man sich auf die Daten, welche die Entwicklungsumgebung benötigt, um die Benutzeroberfläche jederzeit neu zu erstellen.

Die Zustände, die man verwalten muss, können in zwei konzeptionelle Typen unterteilt werden:

1. Ephemeral state
2. App state

Ephemeral state

Der Zustand, der für jedes Widget lokal ist, wird als kurzlebiger Zustand bezeichnet. Da der Zustand in einem einzigen Widget enthalten ist, sind keine komplexen Zustandsverwaltungstechniken erforderlich – es reicht aus, ein einfaches StatefulWidget zu verwenden, um die Benutzeroberfläche neu aufzubauen.

App state

Der Status, der von verschiedenen Widgets geteilt wird, wird als App-Status bezeichnet. Hierfür werden State Management Lösungen benutzt.

In dieser Anwendung wurde als State Management Lösung das Package provider [34] benutzt. Als Beispiel wird die Klasse *BottomNavigationBarModel* benutzt. Um diese Klasse zu verstehen wird zuerst einmal die Funktionsweise der `BottomNavigationBar` erklärt.

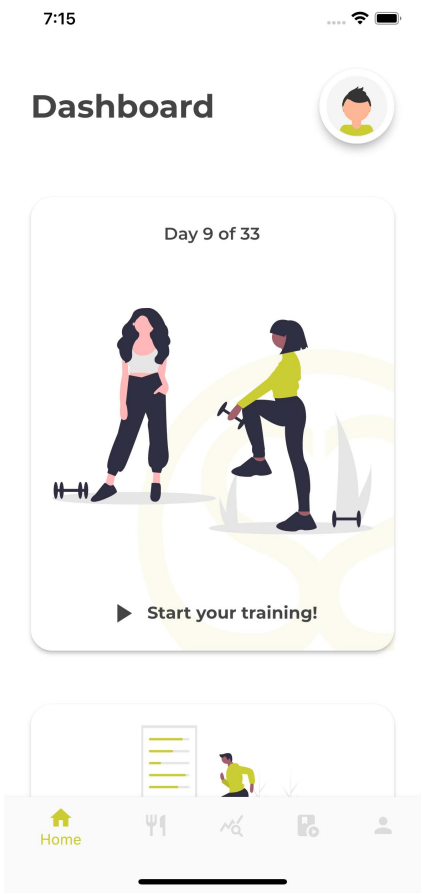

Abbildung 5.1: Dashboard mit BottomNavigationBar

Wie in Abbildung 5.1 zu sehen, wird in der Anwendung zur Navigation in den Hauptbildschirmen eine `BottomNavigationBar` verwendet.

Quellcodeausschnitt 5.1 beschreibt, wie in Flutter eine BottomNavigationBar implementiert wird. Hierzu wird in einer `Scaffold`, welche sozusagen als Grundgerüst eines *Views* dient, ein `BottomNavigationBar`-Widget übergeben, in welcher die verschiedenen `items` angegeben werden. Diese Items sind `BottomNavigationBarItem`-Widgets, welche die Icons mit Texten enthalten, die später auf der NavigationBar angezeigt werden.

Mit dem `currentIndex` wird zwischen den einzelnen Items gewechselt. Also wenn beispielsweise vier Items in der `BottomNavigationBar` angegeben wurden und der `currentIndex` auf 2 gesetzt wird, wird in dem Scaffold das dritte Item gerendert. Bei der `onTap`-Funktion erhält man den index des Items, welches der Benutzer angeklickt hat und man kann damit dann den currentIndex setzten.

```
1  Scaffold(
2    bottomNavigationBar: BottomNavigationBar(
3      items: [
4        BottomNavigationBarItem(
5          icon: Icon(Icons.person),
6          label: 'Profile',
7        ),
8      ],
9      currentIndex: bottomNavigationBarModel.selectedPage,
10     onTap: (index) => {
11       bottomNavigationBarModel.changeSelectedPage(
12         BottomNavigationBarPages.values[index],
13       )
14     },
15   ),
16 );
```

Listing 5.1: `BottomNavigationBar`

Damit man aus der ganzen App heraus auf die `BottomNavigationBar` zugreifen kann und nicht nur innerhalb des `BottomNavigationBar`-Widgets den `currentIndex` wechseln kann, wird hier eine State Management Lösung benutzt und es wird das Provider package verwendet. Hierzu habe ich die Klasse `BottomNavigationBarModel` (siehe Listing 5.2) erstellt und diese mit der Klasse ChangeNotifier erweitert. Die Klasse ChangeNotifier wird bereits durch Flutter bereitgestellt. Durch diese Klasse kann man auf die Änderungen in dieser Klasse einen Listener abonnieren. Innerhalb dieser Klasse ist eine Variable `selectedPage`, welche den `currentIndex` in der `BottomNavigationBar` repräsentiert. Außerdem enthält die Klasse eine Funktion `changeSelectedPage`, mit der man die Variable `selectedPage` ändern kann. In dieser Funktion wird außerdem die Funktion `notifyListeners()` aufgerufen, wodurch allen Listeners die Änderungen mitgeteilt werden.

```
1  class BottomNavigationBarModel extends ChangeNotifier {
2    /*
3      Represents the selected page of the BottomNavigationBar
4      0 = Dashboard
5      1 = Afterworkout Meal
6      2 = Statistic
7      3 = Media Library
8      4 = Profile
9    */
10   int selectedPage = 0;
11
12   void changeSelectedPage(BottomNavigationBarPages page) {
13     selectedPage = page.index;
14     notifyListeners();
15   }
16 }
```

Listing 5.2: `BottomNavigationBarModel`

Um nun in der Anwendung auf diese Klasse zugreifen zu können, wird der Scaffold, welcher die BottomNavigationBar enthält, mit einem ChangeNotifierProvider-Widget umhüllt (siehe Listing 5.3). Durch dieses Widget wird eine Instanz der BottomNavigationBarModel-Klasse erschaffen. Dieses Widget soll weit oben im Widget-Baum platziert werden, sodass alle Widgets darunter liegen, die diese Klasse benötigen. Damit man nun auf die Klasse zugreifen kann, wird noch die Scaffold mit dem Consumer-Widget umhüllt. Bei dem Consumer-Widget erhält man dann Zugriff auf das `bottomNavigationBarModel`-Objekt. Bei dem BottomNavigationBar-Widget wird dann der currentIndex durch `bottomNavigationBarModel.selectedPage` bestimmt. Jetzt kann man zum Beispiel die Funktion `bottomNavigationBarModel.changeSelectedPage(1)` aufrufen, wodurch sich in der Klasse die Variable `selectedPage` ändert und durch die Funktion `notifyListeners()` wird außerdem beispielsweise der currentIndex der BottomNavigationBar geändert.

```
ChangeNotifierProvider(
    create: (context) => BottomNavigationBarModel(),
    child: Consumer<BottomNavigationBarModel>(
      builder: (context, bottomNavigationBarModel, child) => Scaffold(
        bottomNavigationBar: BottomNavigationBar(
          ...
        ),
      ),
    ),
  );
```

Listing 5.3: `ChangeNotifierProvider`

Es gibt außerdem noch die Möglichkeit auf die BottomNavigationBarModel-Klasse unterhalb des ChangeNotifierProvider zuzugreifen, ohne das Consumer-Widget zu benutzten. Dafür benutzt man beispielsweise `Provider.of<BottomNavigationBarModel>(context, listen: false).changeSelectedPage(1);`. Der Vorteil in dieser Variante liegt darin, dass durch einen Aufruf des `notifyListeners()` innerhalb der Klasse, die Widgets nicht nochmal neu gebaut werden, wie bei dem Consumer-Widget.

5.2 Firebase

Die Plattform Firebase wird dazu verwendet, verschiedene Funktionalitäten der Anwendung zu realisieren. Um Firebase in einer Flutter-Anwendung benutzen zu können, muss das Package *firebase_core* hinzugefügt werden. Außerdem muss man die Anwendung unter console.firebase.google.com zu seinem Firebase-Projekt hinzufügen. Zudem werden noch die Packages *cloud_firestore* und *firebase_auth* für die weiteren Funktionalitäten benötigt.

Zuerst muss die Anmeldung/Registrierung des Benutzers implementiert werden. Dafür wurde eine weitere Klasse *ApplicationState* erstellt (siehe Listing 5.4), welche durch die Klasse *ChangeNotifier* erweitert wird. Hier wird alles implementiert, was mit dem Login/Registrierung und dem Upload und Download von Bildern und Daten bei Firebase zutun hat. Der ChangeNotifierProvider der ApplicationState wird über der MaterialApp eingebunden, da ApplicationState in der kompletten App verwendet werden soll.

Im Konstruktor dieser Klasse wird eine Funktion `init()` aufgerufen. In dieser Funktion wird überprüft, ob der Benutzer angemeldet ist. In der ApplicationState-Klasse wird noch eine Variable _loginState erstellt. Der Typ dieser Variable ist ein Enum, welches aus den zwei Werten *loggedOut* und *loggedIn* besteht. Die Variable *_auth* ist von Type FirebaseAuth und dort wird ein Listener in der `init()`-Funktion angelegt, welcher auf alle Änderungen des Benutzers in der App hört. Die Variable speichert außerdem den aktuell eingeloggten Benutzer, auch über das Beenden der Apps hinaus. Wenn also kein Benutzer eingeloggt ist, wird in der `init()`-Funktion der `_loginState` auf `loggedOut` gesetzt. Wenn ein Benutzer eingeloggt ist, wird es auf `loggedIn` gesetzt.

```
ApplicationState() {
  init();
}

FirebaseAuth _auth;
ApplicationLoginState _loginState;

Future init() async {
  await Firebase.initializeApp();

  _auth = FirebaseAuth.instance;

  _auth.userChanges().listen((user) async {
    if (user != null) {
      _loginState = ApplicationLoginState.loggedIn;
    } else {
      _loginState = ApplicationLoginState.loggedOut;
    }
  });
  notifyListeners();
}
```

Listing 5.4: `Implementierung der ApplicationState`

Damit der Benutzer sich einloggen kann, wird in der ApplicationState-Klasse eine Funktion implementiert, mit der sich der Benutzer mit Hilfe E-Mail und Passwort einloggen kann, vorausgesetzt dieser Account wurde bereits registriert. In dieser Funktion wird `_auth.signInWithEmailAndPassword` aufgerufen (siehe Listing 5.5). Diese Funktion wird bereits von Firebase vorgegeben. Sie gibt als Rückgabewert dann den eingeloggten Benutzer in die Variable _auth zurück. Danach setzt man die Variable *_loginState* noch auf *loggedIn* und ruft *notifyListeners()* auf.

Die Registrierung eines neuen Benutzers wird genauso aufgerufen, wie beim Login, außer dass anstelle von `signInWithEmailAndPassword` die Funktion `createUserWithEmailAndPassword` aufgerufen wird. Damit sich der Benutzer ausloggen kann, wird beim ausloggen die Funktion `_auth.signOut()` aufgerufen.

```
_auth.signInWithEmailAndPassword(
  email: email,
  password: password,
);

_loginState = ApplicationLoginState.loggedIn;

notifyListeners();
```

Listing 5.5: Anmeldung mit Firebase

`fetchPreference`-Funktion

Eine weitere Funktion in der ApplicationState-Klasse stellt die `fetchPreference`-Funktion (siehe Listing 5.6) dar. Durch diese Funktion wird dem Entwickler ermöglicht, eine bereits gespeicherte Key-String-Kombination aus der Firestore-Datenbank des aktuell angemeldeten Benutzers zu empfangen. Die Firestore-Datenbank besteht aus Kollektionen. In der Datenbank dieser Anwendung werden die Kollektionen auf der ersten Ebene die Benutzerkollektionen sein. Es wird also für jeden Benutzer ein eigenes Dokument innerhalb der *users*-Kollektion existieren, in der nur seine eigenen Daten gespeichert werden. In Dokumente kann man entweder Daten als Key-String-Kombinationen speichern oder man erstellt in diesem Dokument wieder eine Kollektion. Dadurch kann man die Datenbank so gestalten, wie es in der Anwendung benötigt wird. Abbildung 5.2 und 5.3 zeigen, wie eine Firestore-Datenbank aufgebaut ist. Um ein Datei aus einem Dokument zu erhalten, wird ein `DocumentSnapshot`, wie in Listing 5.6 gezeigt, erstellt. In diesem `DocumentSnapshot` kann man dann auf alle Dateien zugreifen, die in diesem Dokument existieren.

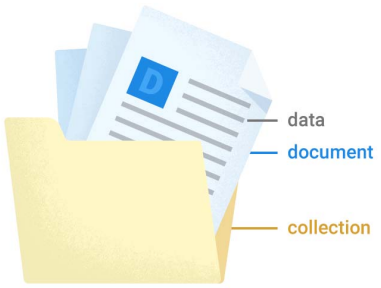

Abbildung 5.2: Aufbau der Firestore-Datenbank [35]

```
1 fetchPreference(String key) async {
2   DocumentSnapshot querySnapshot =
3       await _firestore.collection('users').doc(currentUser.uid).get();
4
5   Map<String, dynamic> data = querySnapshot.data();
6
7   return data[key];
8 }
```

Listing 5.6: `fetchPreference`-Funktion

`uploadPreference`-Funktion

Damit man auch eine *Preference* in die Datenbank hochladen kann, wurde die Funktion `uploadPreference` realisiert (siehe Listing 5.7). In dieser Funktion wird in den Eingabeparametern bestimmt, welche Daten mit welchem Key hochgeladen werden. Durch die `update`-Funktion auf ein Firestore-Dokument kann man dann die Daten hochladen.

```
1 uploadPreference(String key, String data) {
2   _firestore.collection('users').doc(currentUser.uid).update({key: data});
3 }
```

Listing 5.7: `uploadPreference`-Funktion

`setFirstLogin`-Funktion

Wenn sich ein neuer Benutzer in der Anwendung registriert, muss bereits eine *Preference* berechnet und auf die Firestore-Datenbank hochgeladen werden. Hierzu wird die `setFirstLogin`-Funktion (siehe Listing 5.8) verwendet. Diese Funktion berechnet das aktuelle Datum und speichert dieses sowohl in der Datenbank, als auch in den *SharedPreferences* als `firstLogin`. Dieses Datum wird dann in der Anwendung verwendet, um den Tag zu berechnen, an dem sich der Benutzer in der 33-Tage Challenge befindet.

```
setFirstLogin() async {
  final startHour = 4;

  SharedPreferences prefs = await SharedPreferences.getInstance();

  DateTime currentTimeStamp = DateTime.now().toUtc();

  String currentDay = DateTime.utc(
    currentTimeStamp.year,
    currentTimeStamp.month,
    currentTimeStamp.day,
    startHour,
  ).toString();

  _firestore
      .collection('users')
      .doc(currentUser.uid)
      .set({'firstLogin': currentDay});

  prefs.setString('firstLogin', currentDay);
}
```

Listing 5.8: `setFirstLogin`-Funktion

`fetchLoginPreferences`-Funktion

In der ApplicationState-Klasse existiert außerdem noch die Funktion `fetchLoginPreferences` (siehe Listing 5.9), welche aufgerufen wird, sobald der Benutzer sich anmeldet. In dieser Funktion werden alle bereits vorhandenen Daten auf das Smartphone geladen, welche in der Firebase-Datenbank gespeichert wurden. Bei der Funktion `fetchStatistics` werden beispielsweise die Statistiken aller bereits durchgeführten Trainings heruntergeladen. Diese Daten enthalten das Trainingsdatum, die Dauer des Trainings, die verbrauchten Kalorien, sowie die durchschnittliche Herzfrequenz während des Trainings. Diese Daten werden dann in den SharedPreferences Speicher der Anwendung geladen, wodurch sie dann auch über das Beenden der Anwendung bestehen bleiben.

Bei der Funktion `fetchAfterworkoutMeals()` werden die vom letzten Training berechneten Mahlzeiten von Firebase heruntergeladen und in den SharedPreferences ge-

speichert. Es werden außerdem noch die Variablen `firstLogin`, `lastTrainingDay`, `currentDay`, `userWeight` und `userLevel` heruntergeladen, bzw. berechnet, und in die SharedPreferences gespeichert. Die Variable `currentDay` entspricht dem aktuellen Tag in der 33-Tage Challenge und berechnet sich aus der Differenz zwischen dem heutigen Tag und der ersten Anmeldung, welche in der Variable `firstLogin` gespeichert wird. Die Variablen `userWeight` und `userLevel` werden bei der Registrierung vom Benutzer selbst bestimmt.

```
1  fetchLoginPreferences() async {
2    SharedPreferences prefs = await SharedPreferences.getInstance();
3
4    await fetchStatistics()
5    await fetchAfterworkoutMeals()
6
7    String firstLogin = await fetchPreference('firstLogin');
8    prefs.setString('firstLogin', firstLogin);
9
10   String lastTrainingDay = await fetchPreference('lastTrainingDay');
11   if (lastTrainingDay != null)
12     prefs.setString('lastTrainingDay', lastTrainingDay)
13
14   int currentDay = DateTime.parse(firstLogin).daysUntil(DateTime.now().
       toUtc())
15   prefs.setInt('currentDay', currentDay + 1)
16
17   String userWeight = await fetchPreference('userWeight');
18   if (userWeight != null) prefs.setInt('userWeight', int.parse(userWeight))
19
20   String userLevel = await fetchPreference('userLevel');
21   if (userLevel != null) prefs.setInt('userLevel', int.parse(userLevel));
22 }
```

Listing 5.9: `fetchLoginPreferences`-Funktion

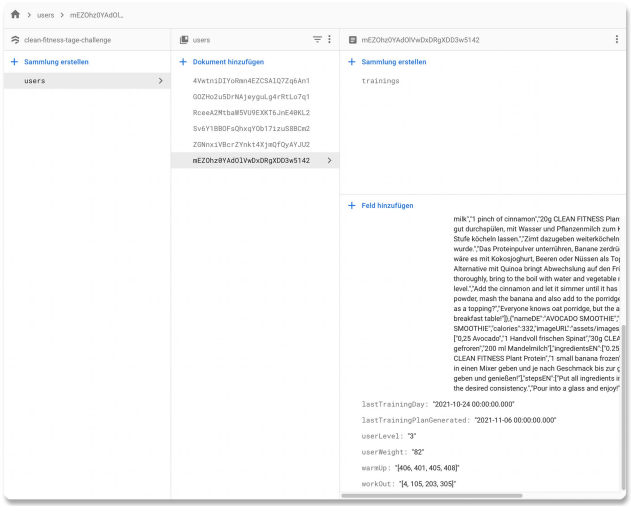

Abbildung 5.3: Auszug aus der Firestore-Datenbank

5.3 Anmeldung / Registrierung

Nachdem die Anwendung gestartet wurde, wird sofort die Klasse ApplicationState initialisiert. In dieser Klasse wird im Konstruktor geprüft, ob der Benutzer bereits angemeldet ist. Nach dem Starten der Anwendung wird auf das Widget *WelcomeScreen* navigiert. In diesem Widgets wird dann mit Hilfe des Providers ApplicationState geprüft, ob der Benutzer angemeldet ist oder nicht. Falls der Benutzer bereits angemeldet ist, wird er direkt weiter auf die Homepage der Anwendung weitergeleitet. Anderenfalls wird in diesem Widget ein Bildschirm mit Buttons angezeigt, die zur Anmeldung und Registrierung führen. (siehe Listing 5.10)

```
@override
Widget build(BuildContext context) {
  ApplicationState applicationLoginState = Provider.of<ApplicationState>(
    context);

  if (applicationLoginState.loginState == ApplicationLoginState.loggedOut)
    {
    return Scaffold(
      ...
    );
  } else {
    return HomepageWithBottomNavigationBar();
  }
}
```

Listing 5.10: `WelcomeScreen`-Widgets

Auf dem Widget `LoginScreen` (siehe Abbildung 5.4) wird ein Formular mit E-Mail-Adresse und Passwort angezeigt, sowie ein Button zur Anmeldung. In Flutter ist es möglich, bei einem TextFormField, welches ein Eingabefeld in einem Formular darstellt, einen *validator* zu übergeben (siehe Listing 5.11). Durch diesen Validator ist es möglich, die Eingabe eines Benutzers zu überprüfen, bevor das Formular abgeschickt wird. Als Validator wird in dieser Anwendung das Package *form_field_validator* verwendet. Bei dem E-Mail-Feld wird beispielsweise als Validator der `EmailValidator` und der `RequiredValidator` verwendet. Dadurch wird die Überprüfung der Eingabe des Benutzers vereinfacht.

```
TextFormField(
  validator: MultiValidator([
    EmailValidator(),
    RequiredValidator(),
  ]),
);
```

Listing 5.11: Ausschnitt aus dem `TextFormField`-Widget

Sobald der Benutzer sich über das Formular anmeldet und die Eingaben korrekt sind, wird die Funktion `applicationState.fetchLoginPreferences()` aufgerufen, wodurch alle bereits vorhandenen Daten auf das Smartphone geladen werden. Außerdem wird erneut auf das Widget WelcomeScreen navigiert. Dort wird wieder geprüft ob der Benutzer eingeloggt ist, was dieses mal auch korrekt ist und es wird die Homepage der Anwendung angezeigt.

Das Widget `SignUpScreen` (siehe Abbildung 5.4) dient als Registrierungsbildschirm und ist ähnlich aufgebaut wie der Anmeldungsbildschirm, nur dass hier außerdem noch ein Feld für den Vor- & Nachnamen eingefügt wurde. Außerdem wurde bei dem Passwort-Feld weitere Validatoren `MinLengthValidator(8)` und `PatternValidator()` hinzugefügt. Dadurch wird sichergestellt, dass der Benutzer ein ausreichend sicheres Passwort eingibt.

Nach erfolgreicher Registrierung wird die Funktion `applicationLoginState.setFirstLogin()` aufgerufen, wodurch das Datum der ersten Anmeldung gespeichert wird und es wird außerdem wieder auf das WelcomeScreen-Widget navigiert. Dort wird geprüft, ob der Benutzer angemeldet ist und ob der Benutzer bereits seine Anamnese durchgeführt hat. Dazu wird geprüft, ob in den Daten bereits die Variablen `userLevel` und `userWeight` existieren. Wenn diese nicht existieren, wird auf ein weiteres Widget `OnboardingScreen` navigiert. Dort werden dann das `userLevel` und das `userWeight` abgefragt, in der Datenbank und in den SharedPreferences gespeichert. Daraufhin wird wieder auf die Homepage der Anwendung navigiert.

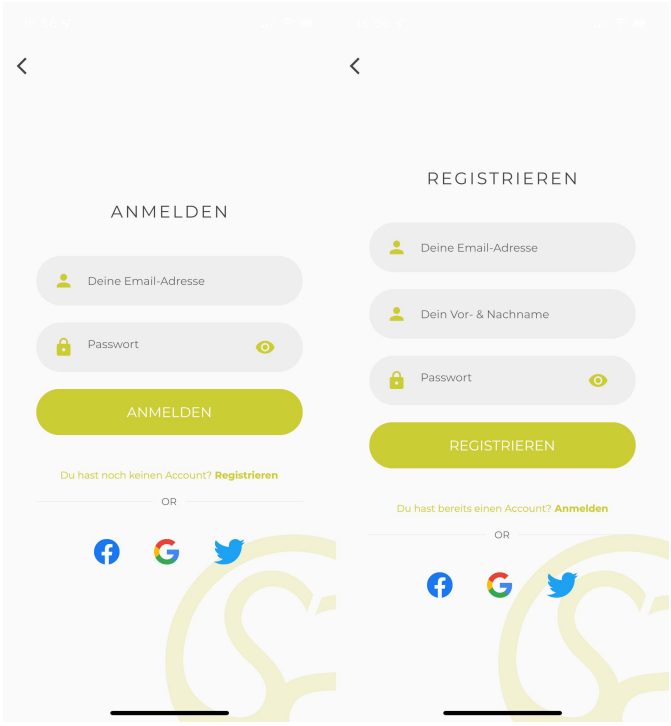

Abbildung 5.4: Login- und Registrierungsbildschirm

5.4 Lokalisation

Damit diese Anwendung auch von Benutzern verwendet werden kann, welche nicht Deutsch sprechen, werden in diese Anwendung alle Texte lokalisiert. Das bedeutet, dass die Texte in der Sprache angezeigt werden, die der Benutzer in seinem Gerät als Standartsprache eingestellt hat. Also hat der Benutzer beispielsweise Englisch als Gerätesprache eingestellt, werden alle Texte in der Anwendung auf Englisch angezeigt.

Damit das möglich wird, werden die Packages *flutter_localizations* und *intl* benötigt. *flutter_localizations* wird bereits vom Flutter-Framework bereitgestellt. Zum hinzufügen von unterstützten Sprachen wird in der MaterialApp die `localizationsDelegates` und die `supportedLocales`, wie in Listing 5.12 dargestellt, angegeben.

```
1  MaterialApp(
2    localizationsDelegates: [
3      AppLocalizations.delegate,
4      GlobalMaterialLocalizations.delegate,
5      GlobalWidgetsLocalizations.delegate,
6      GlobalCupertinoLocalizations.delegate,
7    ],
8    supportedLocales: [
9      Locale('de', ''),
10     Locale('en', ''),
11   ],
12 );
```

Listing 5.12: `localizationsDelegates` in der MaterialApp

Damit man jetzt eigene Texte festlegen kann, werden die zwei Dateien *app_de.arb* und *app_en.arb* in dem Ordner */lib/l10n/* erstellt. In dieser Datei werden die Textpassagen dann mit Hilfe eines Bezeichners bestimmt (siehe Listing 5.13). Zum Benutzen dieser Texte werden dann in der App anstelle der Strings `AppLocalizations.of(context).appTitle` verwendet.

```
1  {
2    "appTitle": "CLEAN FITNESS 33 Tage Challenge",
3    "dashboardBoxTop": "Tag {currentDay} von 33",
4    "@dashboardBoxTop": {
5      "placeholders": {
6        "currentDay": {
7          "type": "String",
8          "example": "3"
9        }
10     }
11   },
12   "dashboardMediaLibrary": "Deine Mediathek",
13   "dashboardStartTraining": "Starte dein Training!"
14 }
```

Listing 5.13: Auszug aus *app_de.arb*

Man kann bei diesen Textpassagen außerdem Variablen verbauen, wie beispielsweise bei `dashboardBoxTop` in Listing 5.13, wodurch man dann zum Beispiel Nummern nicht direkt in den Übersetzungen bestimmen muss, sondern sie während der Laufzeit bestimmen kann. Man kann dann mit `AppLocalizations.of(context).dashboardBoxTop(1)` beispielsweise eine Variable einfügen.

5.5 Trainingsplanerstellung

Damit die Trainingsplanerstellung realisiert werden konnte, mussten zuerst die verschiedenen Übungen unterteilt und auf dem Gerät gespeichert werden. Die Übungen wurden in drei Kategorien unterteilt, *Warm-Up*, *Work-Out* und *Cool-Down*. Außerdem wurden die *Work-Out*-Übungen noch in Level und Kategorien unterteilt. Es gibt drei Level, die der Benutzer selbst anhand seines Fitnesslevels unterscheiden kann. Er kann es entweder bei der Anamnese einstellen, welche bei der Registrierung in der Anwendung stattfindet oder es in den Einstellungen der Anwendung ändern. Die Kategorien bestehen aus *Front*, *Back*, *Side*, *Leg* und werden dazu verwendet, die Übungen in den verschiedenen Körperregionen einzuteilen.

Die Übungen wurden in einer *SQLite*-Datenbank gespeichert, welche beim ersten Starten der Anwendung erstellt und gefüllt wird. Damit eine *SQLite*-Datenbank realisiert werden kann, wird das Flutter-Package *sqflite* benötigt.

Zur leichteren Handhabung mit der Datenbank wurde die Klasse `DBProvider` erstellt. Diese Klasse dient dazu, die Datenbank zu erstellen und alle Zugriff auf die Datenbank zu handhaben. In dieser Klasse wurde ein Getter `database` initialisiert (siehe Listing 5.14). Sobald in der Anwendung die Datenbank aufgerufen wird, wird dieser Code ausgeführt. Dort wird geprüft, ob in der statischen Variable _database bereits eine Datenbank existiert. Falls bereits eine Datenbank initialisiert wurde, wird diese als Rückgabewert zurück gegeben. Anderenfalls wird die Funktion `initDB` aufgerufen, welche die Datenbank realisiert und diese Datenbank wird dann zurück gegeben.

```
1   static Database _database;
2
3   Future<Database> get database async {
4   if (_database != null) return _database;
5
6   _database = await initDB();
7   return _database;
8 }
```

Listing 5.14: Realisierung des Getters der Datenbank

In der `initDB`-Funktion (siehe Listing 5.15) wird zuerst der Pfad zu einem Speicherort bestimmt, der dazu verwendet werden kann, Benutzerdaten auf dem Smartphone zu speichern. Danach wird an diesen Pfad der Name der Datenbank angehängt. Im Anschluss wird mit der Funktion `openDatabase` diese Datenbank geöffnet. In dieser

Funktion kann man eine Funktion `onCreate` übergeben. Dort wird die Funktion `insertExercises` mit dem Parameter `db` aufgerufen.

```
initDB() async {
  Directory documentsDirectory = await getApplicationDocumentsDirectory();
  String path = join(documentsDirectory.path, 'DBv3.db');

  return await openDatabase(
    path,
    onCreate: (Database db, int version) async {
      await insertExercises(db);
    },
  );
}
```

Listing 5.15: `initDB`-Funktion

In der Funktion `insertExercises` wird mit Hilfe der `execute`-Funktion der `Database`-Klasse die Datenbanktabelle erstellt. Bei dieser `execute`-Funktion kann man SQL-Statements übergeben um somit seine Datenbank zu erstellen und aktualisieren. Es werden folgende Felder in der *Exercise*-Tabelle erstellt. Es wird eine *id* für jede Übung vergeben, womit man die Übungen individuell bestimmen kann. *nameDE* und *nameEN* stellen die Namen der Übungen dar. Es wurde zwei Sprachen der Namen gespeichert, damit man in der Benutzeroberfläche wieder zwischen der englischen und der deutschen Sprache unterscheiden kann. Die Felder *time* und *level* speichert die Dauer der Übung in Sekunden und das Level der Übung, welche '1', '2' oder '3' sein kann. Es wird der *MET* gespeichert. Dieser wird für die Berechnung des Kalorienverbrauchs benötigt, wie in Kapitel 2.7.2 beschrieben. Die Felder *categoryDE* und *categoryEN* beschreiben die beanspruchten Körperregionen dieser Übung. Es wird wieder die deutsche und die englische Ausführung dieser Kategorien gespeichert. Das Feld *exerciseTpe* beschreibt die Art der Übung. Es wird unter *Warm-Up*, *Work-Out* und *Cool-Down* unterschieden. *muscleGroup* wird zur Unterscheidung zwischen den Kategorien *Front*, *Back*, *Side* und *Leg* verwendet. Das Feld *image*, bzw. *secondSideImage*, beschreiben die Bilder der Übung, welche in der Trainingsübersicht angezeigt werden. *secondSideImage* wird dazu verwendet, um bei einer Übung mit der Kategorie *Side* ein Bild anzuzeigen, das die Ausführung der Übung auf der anderen Seite anzeigt. Das wird verwendet, da eine *Side*-Übung nur eine Körperseite trainiert und man dann noch die andere Seite zudem trainieren muss. Bei dem Feld *videoURL* wird die URL der Videoerklärung gespeichert. Bei *descriptionDE* und *descriptionEN* werden Beschreibungen der Übung gespeichert, wieder in den zwei Sprachausführungen.

Im Anschluss wird die Datenbank mit den verschiedenen Tabelleneinträgen gefüllt. Das wird mit dem SQL-Statement `INSERT INTO` durchgeführt.

Die Klasse `DBProvider` enthält außerdem noch die Funktionen `getExerciseByID` (siehe Listing 5.16) und `getAllExercises`. Dadurch wird der Zugriff auf die Datenbank vereinfacht, da man nicht bei jedem Datenbankzugriff mit Hilfe eines SQL-Statements die Daten abrufen muss. Bei der Funktion `getExerciseByID` wird eine

Abfrage auf die Datenbank durchgeführt, die genau die Übung mit der übergebenen ID zurückliefert. Bei der Funktion `getAllExercises` wird ein List-Objekt mit allen Übungen zurückgegeben.

```
1 getExerciseByID(int id) async {
2   final db = await database;
3   var res = await db.query('Exercise', where: 'id = ?', whereArgs: [id]);
4
5   return res.isNotEmpty ? Exercise.fromJson(res.first) : Null;
6 }
```

Listing 5.16: `getExerciseByID`-Funktion

Um den Umgang mit den einzelnen Übungen zu vereinfachen, wurde eine Klasse `Exercise` erstellt. In dieser Klasse hat jedes `Exercise`-Objekt die gleichen Attribute wie die *Exercises*-Tabelle in der Datenbank.

Die Erstellung des Trainingsplans wird in dem Widget `HomepageWithBottomNavigationBar` durchgeführt. Dieses Widget wurde ausgewählt, da dieses Widget als Homepage der Anwendung dient und die Erstellung somit bei jedem Öffnen der App angestoßen wird. In einem Flutter-Widget wird die `initState`-Funktion verwendet, damit der Entwickler Funktionen beim Initialisieren des Widgets ausführen kann. In der `initState`-Funktion wird die `generateTrainingPlan`-Funktion aufgerufen.

In der `generateTrainingPlan`-Funktion wird zu Beginn geprüft, ob der Benutzer in der Anwendung angemeldet ist und bereits eine Anamnese durchgeführt hat. Da der Trainingsplan anhand der Benutzerdaten erstellt wird, kann kein Plan erstellt werden, wenn der Benutzer nicht angemeldet ist und sein Fitnesslevel noch nicht ausgewählt hat. Wenn der Benutzer angemeldet ist und sein Trainingslevel bekannt ist, wird aus der Firebase-Datenbank die *Preference* `lastTrainingPlanGenerated` heruntergeladen. Diese Variable sagt aus, wann bei diesem Benutzer das letzte Mal ein Trainingsplan erstellt wurde. Es ist wichtig zu prüfen, ob an diesem Tag bereits ein Trainingsplan erstellt wurde, da der Trainingsplan nicht bei jedem Öffnen der Anwendung generiert werden soll, sondern es soll jeden zweiten Tag einen neuen Plan geben. Es wird im Anschluss geprüft, ob an diesem Tag bereits ein Trainingsplan erstellt wurde und ob an diesem Tag ein Training geplant ist. Das wird geprüft, indem aus der Firebase-Datenbank das Feld `lastTrainingDay` heruntergeladen wird, welches aussagt, wann das letzte Training abgeschlossen wurde. Es sollte nach jedem Trainingstag ein Tag Pause stattfinden. Es wurden dann die Anzahl der Tage zwischen dem letzten Trainingsdatum und dem heutigem Datum berechnet. Falls diese Anzahl größer als eins ist, wird an diesem Tag ein Training stattfinden und die Variable `isTrainingDay` wird auf `true` gesetzt..

Falls an diesem Tag noch kein Plan erstellt wurde und die Variable `isTrainingDay` gleich `true` ist, wird die Generierung der Trainingsplans begonnen. Dafür wurde zuerst die Funktion `loadDatabase` (siehe Listing 5.17) aufgerufen. Bei dieser Funktion

wird die SQLite-Datenbank geöffnet und es wird die Funktion `getAllExercises` dieser Datenbank aufgerufen. Bei dieser Funktion werden alle Einträge der Datenbank in einer `List` zurückgegeben.

```
Future<List<Exercise>> loadDatabase() async {
  final _resultDatabase = await DBProvider.db.getAllExercises();

  if (_resultDatabase != null)
    return _resultDatabase;
  else
    throw Exception('Failed to load Exercise!');
}
```

Listing 5.17: `loadDatabase`-Funktion

Daraufhin wird die Variable `userLevel` aus der Firebase-Datenbank heruntergeladen und in der Variable `userLevel` gespeichert. Dann wurden die Funktionen `generateWarmup`, `generateWorkout` und `generateCooldown` aufgerufen. Die Beschreibungen der Funktionen wird anhand der `generateWorkout`-Funktion erklärt, wie in Listing 5.18 zu sehen.

```
generateWorkout(int userLevel, List<Exercise> allExercises) async {
  List<String> exercises = [];

  List<String> exerciseTypes = ['front', 'back', 'side', 'leg'];

  for (String type in exerciseTypes) {
    List<Exercise> fetchedExercises = allExercises
        .where(
          (exercise) =>
              exercise.exerciseType == 'workOut' &&
              exercise.muscleGroup == type &&
              exercise.level == userLevel,
        )
        .toList();

    Exercise exercise = fetchedExercises.sample(1)[0];
    exercises.add(exercise.id.toString());
  }

  return exercises;
}
```

Listing 5.18: `generateWorkout`-Funktion

Zu Beginn wird eine leere Liste `exercises` erstellt, in der im Laufe der Funktion die generierten Übungen gespeichert werden. Dieser Funktion wird die Liste mit allen Übungen, sowie das Trainingslevel des Benutzers übergeben. Es wird eine Liste mit den Übungskategorien *Front*, *Back*, *Side* und *Leg* erstellt, worüber dann in der Funktion

iteriert wird, da der Workout-Abschnitt des Trainings immer aus vier verschiedenen Übungen mit den vier Übungskategorien besteht. Es wird dann eine Abfrage auf die Liste aller Übungen erstellt, wobei die Abfrageparameter aus `exerciseType`, `muscleGroup` und `level` bestehen. Bei dieser Abfrage wird dann eine Liste zurück gegeben, die alle Übungen, passend zu den Parametern, enthält. Aus dieser Liste wird dann eine zufällige Übung mit Hilfe der Funktion `sample` ausgewählt und die ID dieser Übung wird in der Liste `exercises` gespeichert. Sobald die Schleife durchgeführt wurde, wird die Liste aus der Funktion zurückgegeben.

Im Gegensatz zu der Funktion `generateWorkout` wurde bei den Funktionen `generateWarmup` und `generateCooldown` bei der Abfrage der Übungen aus der Liste aller Übungen nur der Parameter `exerciseType` verwendet, da im Warmup und im Cooldown des Trainings die Übungen nicht in Level oder in Kategorien unterteilt sind. Außerdem wurde bei diesen Funktionen in der Funktion `sample` auf den generierten Übungslisten anstatt einer Übung vier Übungen ausgewählt. Die Rückgabewerte der Funktionen `generateWarmup`, `generateWorkout` und `generateCooldown` wurden dann sowohl in den SharedPreferences als auch in der Firebase-Datenbank gespeichert.

Falls am aktuellen Tag bereits ein Trainingsplan erstellt wurde, wird anstelle der Trainingsplanerstellung der aktuelle Trainingsplan in die SharedPreferences gespeichert. Dafür wurde der Trainingsplan aus der Firebase-Datenbank heruntergeladen und in die SharedPreferences gespeichert.

5.6 Trainingsablauf

Bevor mit dem Training begonnen werden kann, erhält der Benutzer zunächst einen Überblick über das Training. Es wird das Widget `TrainingOverview` (siehe Abbildung 5.5) aufgerufen, sobald der Benutzer das Training über die Homepage starten will. In dieser Übersicht werden alle Übungen angezeigt, die zuvor bei der Trainingsplanerstellung generiert wurden. Beim Aufrufen dieses Widgets wird mit Hilfe eines `FutureBuilders` die Funktion `_getExercises` aufgerufen. In dieser Funktion wird zuerst wieder die Funktion `_loadDatabase` (siehe Listing 5.17) aufgerufen und der Rückgabewert wird in der Liste `_allExercises` gespeichert. Diese Liste enthält somit alle Übungen der Datenbank. Daraufhin werden die Listen `_warmUpExercises`, `_workOutExercises` und `_coolDownExercises` erstellt und ihnen wird der Rückgabewert der Funktion `getExerciseList` übergeben. Dieser Funktion wird die Liste des generierten Trainingsplans übergeben und die Liste aller Übungen. Diese drei Listen werden dann zusammen in eine Liste `exerciseList` gegeben und als Rückgabewert des Fu-

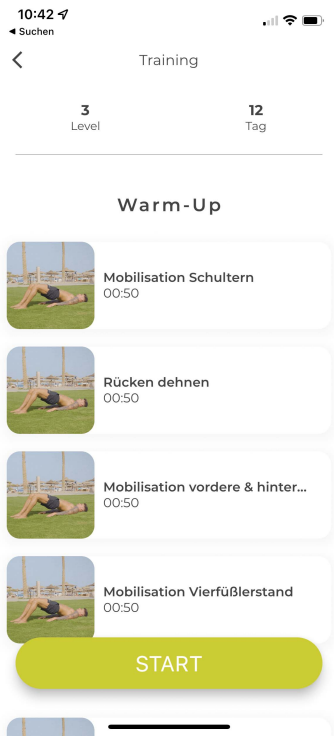

Abbildung 5.5: Aufbau der Trainingsübersicht

tureBuilders verwendet. Im Widget werden dann aufgrund der übergebenen Liste die Kategorien und die einzelnen Übungen erstellt. Sobald man auf eine dieser Übungen klickt, wird eine Detailansicht dieser Übung angezeigt. In dieser Ansicht wird ein Video abgespielt, welches die Übung erklärt, sowie die Dauer und die Beschreibung dieser Übung anzeigt. Das Video ist im *Firebase Storage* gespeichert. Dafür muss man im Dashboard des Firebase-Projekts das Video hochladen und man erhält dann den Link, den man für das Video benutzen kann.

Sobald der Benutzer auf den *START*-Button in der Trainingsübersicht (siehe Abbildung 5.5) klickt, wird in der Anwendung auf das `TrainingBuilder`-Widget navigiert und es wird die Liste `exerciseList` übergeben. In diesem Widget wird mit Hilfe eines PageViews der Trainingsablauf erstellt. Damit man nicht nur innerhalb des `TrainingBuilder`-Widgets auf den Trainingsablauf und auf die Navigation im Trainingsablauf zugreifen kann, wurde eine ChangeNotifier-Klasse `TrainingModel` erstellt. Diese Klasse enthält eine Variable `currentTrainingPage`, welche über die aktuelle Seite in dem PageView des `TrainingBuilder`-Widgets bestimmt. Außerdem gibt es noch die Funktionen `nextTrainingPage` (siehe Listing 5.19) und `previousTrainingPage`. Diese Funktionen ermöglichen es, die aktuelle Seite des PageView von überall in der Anwendung zu ändern.

```
1 nextTrainingPage() {
2   currentTrainingPage += 1;
3
4   notifyListeners();
5 }
```

Listing 5.19: `nextTrainingPage`-Funktion

Der `ChangeNotifierProvider` und der `Consumer` dieser Klasse wurde dann in dem `TrainingBuilder`-Widget hinzugefügt. Im PageView dieser Klasse wurde als `children` der Rückgabewert der Funktion `createTrainingPages` übergeben. Innerhalb dieser Funktion wird eine Liste `exercises` erstellt, in denen dann alle Übungen gespeichert werden und eine Liste `trainingPagesList`, worin die einzelnen Widgets des PageViews gespeichert werden. Dann wird über die Liste `exerciseList` iteriert. Dadurch erhält man drei Durchgänge für die einzelnen Trainingskategorien *Warmup*, *Workout* und *Cooldown*. Danach wird über alle Übungen der einzelnen Kategorie iteriert und für jede Übung wird ein Widget `TrainingPageTimer` zu der Liste `trainingPagesList` hinzugefügt. Außerdem wird in der Liste `exercises` die Übung hinzugefügt. Es wird im Anschluss geprüft, ob in der aktuellen Übung das Feld `secondSideImage` vergeben wurde. Falls dort ein Eintrag existiert, wird zudem noch ein `TrainingPageTimer`-Widget hinzugefügt für die zweite Seite dieser Übung. Es wird außerdem für jede hinzugefügte Übung noch eine weitere Seite zu der Liste hinzugefügt, da im Trainingsablauf vor jeder Übung eine kurze Erklärseite der kommenden Übung angezeigt werden soll. Diese Liste `trainingPagesList` dient dann als Rückgabewert der Funktion und als `children` des PageViews.

Das Widget `TrainingPageTimer` enthält einen Timer, sowie die Kontrollelemente für den Timer und das Video der Übung. In diesem Widget wird zuerst überprüft, ob es sich um eine Trainingsseite oder um eine Vorbereitungsseite handelt. Wie in Abbildung 5.6 zu sehen ist, sind die beiden Seiten ähnlich aufgebaut, nur dass die Dauer der Vorbereitung auf der Vorbereitungsseite auf 20 Sekunden festgelegt ist und die Dauer der Trainingsseite durch den Eintrag in der Datenbank bestimmt wird. Außerdem wurden die Elemente der Trainingsseite vertikal gespiegelt, sodass der Nutzer sofort sehen kann, auf welcher Seite er sich gerade befindet.

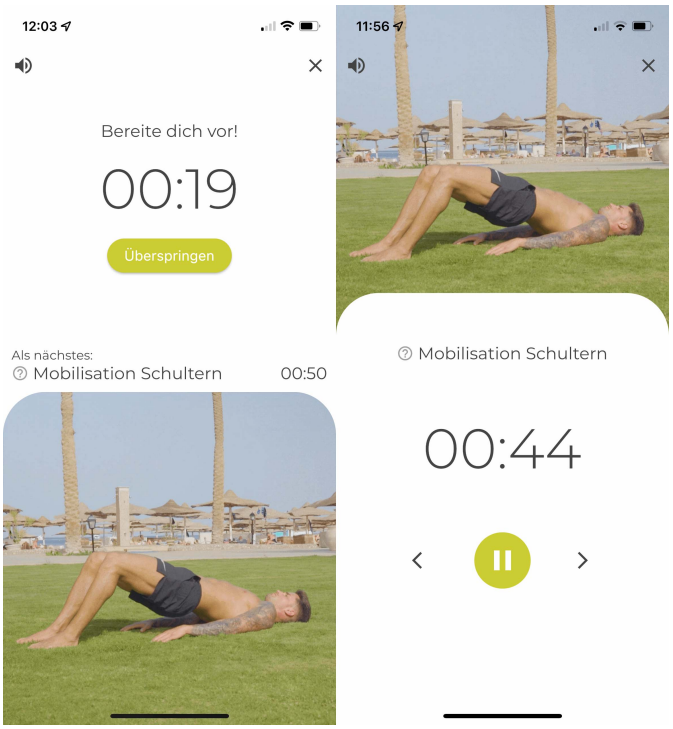

Abbildung 5.6: Vorbereitungs- und Trainingsseite

Beim Initialisieren des Widgets `TrainingPageTimer` wird in der `initState`-Funktion die Funktion `startTimer` aufgerufen. Mit diesem Timer wird die Variablen `_remainingTime` jede Sekunde um eins verringert. Außerdem wurde in der Klasse `TrainingModel` die Variablen `isRunning` und `remainingTime`, sowie die Funktionen `setRemainingTime`, `resumeTimer` und `pauseTimer` initialisiert. Dadurch wird es ermöglicht, die Steuerung des Timers auch von anderen Orten in der Anwendung aus zu steuern. `setRemainingTime` setzt die Variable `_remainingTime` auf den in der Funktion übergebenen Parameter. Die Funktion `resumeTimer` setzt die Variable `isRunning` auf `true`, `pauseTimer` setzt sie auf `false`.

Jedes Mal, wenn der Timer ausgeführt wird, was jede Sekunde passiert, wird zuerst überprüft, ob die Variable `isRunning` im Objekt `trainingModel` gleich `true` ist. Falls es korrekt ist, wird `_remainingTime` um eins verringert und im `trainingModel` wird die Funktion `setRemainingTime` mit `_remainingTime` aufgerufen. Außerdem wird noch geprüft, ob `_remainingTime` gleich null ist. In diesem Fall wird außerdem noch geprüft, ob die aktuelle Seite der PageView gleich die letzte Übung

des Trainings ist. Falls es die letzte Seite ist, wird der Timer pausiert. Anderenfalls wird die nächste PageView-Seite aufgerufen und die Zeit des Timers wird wieder auf die Dauer der Übung gesetzt. (siehe Listing 5.20)

```
1  Timer _timer;
2
3  _timer = Timer.periodic(Duration(seconds: 1), (_) {
4    if (trainingModel.isRunning) {
5      setState(() {
6        _remainingTime--;
7      });
8      trainingModel.setRemainingTime(_remainingTime);
9    }
10   if (_remainingTime == 0) {
11     if (widget.trainingModel.pagesCount > widget.pageId) {
12       widget.trainingModel.nextTrainingPage();
13       _remainingTime = widget.exercise.time;
14     } else {
15       trainingModel.pauseTimer();
16     }
17   }
18 });
```

Listing 5.20: Implementierung des Timers

Damit der Benutzer weiß, wann die Übung beginnt und wie lange sie dauert, wurden in der Anwendung Ansagen während des Trainings implementiert. Als Grundlage für die Implementierung dient das Package `flutter_tts`. Dadurch ist es möglich, eine Text-to-Speech Sprachausgabe zu entwickeln. Zu Beginn muss die Sprachausgabe initialisieren. Dafür werden bei dem `FlutterTts`-Objekt die Werte `setSpeechRate`, `setPitch`, `setVolume` und `setVoice` benötigt. Im Anschluss kann man dann, wie in Listing 5.21 beschrieben, mit der Funktion `flutterTts.speak()` die Ausgabe ausführen. In dieser Anwendung wurde beispielsweise bei der Vorbereitungsseite eine Sprachausgabe mit „In 10 Sekunden geht's los“ ausgeführt, wenn genau zehn Sekunden auf dem Timer übrig sind. Hier wurde wieder mit der Lokalisation der Ausgabe gearbeitet.

```
1 if (_remainingTime == 10) {
2   flutterTts.speak(AppLocalizations.of(context).startsInSeconds(10));
3 }
4 if (_remainingTime == 0) {
5   flutterTts.speak(AppLocalizations.of(context).letsGo);
6 }
```

Listing 5.21: Sprachausgabe implementieren

Es wurde außerdem in der `TrainingModel`-Klasse eine weitere Variable `isMuted` und eine Funktion `changeMute` hinzugefügt. Dadurch wurde sichergestellt, dass

man wieder von überall aus in der App die Sprachausgabe stoppen und aktivieren kann. Dafür wurde im `TrainingPageTimer`-Widget die Abfrage von Listing 5.22 implementiert. Dadurch wird die Sprachausgabe gestoppt und aktiviert, sobald jemand die Funktion `changeMute` in der `TrainingModel`-Klasse ausführt. Es wurden auch Buttons in der Benutzeroberfläche eingefügt, durch die man die Sprachausgabe stoppen und aktivieren kann.

```
1 trainingModel.isMuted
2   ? flutterTts.setVolume(0)
3   : flutterTts.setVolume(1.0);
```

Listing 5.22: `isMuted`-Abfrage

5.7 Kalorienberechnung

Die Kalorienberechnung des Trainings kann in zwei verschiedenen Varianten durchgeführt werden. Die erste Variante besteht darin, den Kalorienverbrauch von *Google Fit* oder *Apple Health* zu beziehen. Da bei dieser Variante erforderlich ist, dass der Benutzer beim Training eine Smartwatch trägt, musste eine weitere Variante der Kalorienberechnung entwickelt werden. Die andere Variante wäre, die Kalorienberechnung mit Hilfe des MET Faktors (siehe Abschnitt 2.7.2) und den tatsächlichen durchgeführten Übungen zu berechnen.

Berechnung mit dem `health`-Package

Damit man in Flutter die Daten von `Google Fit` oder `Apple Health` erhält, muss man das Package `health` verwenden. Um die Daten zu erhalten, muss man zuerst eine Genehmigungsanfrage an *Google Fit*, bzw. *Apple Health* stellen, damit man diese Daten verwenden darf. In dieser Anwendung wurde diese Anfrage beim Öffnen der Trainingsübersicht angefordert. Dafür muss man ein `HealthFactory`-Objekt erstellen und auf diesem Objekt die Funktion `requestAuthorization` aufrufen (siehe Listing 5.23). Dieser Funktion muss man dann die gewünschten Datentypen übergeben. In dieser Anwendung wird `ACTIVE_ENERGY_BURNED`, was die verbrauchten Kalorien darstellt, und `HEART_RATE`, die Herzfrequenz des Benutzers, benötigt.

```
1 HealthFactory health = HealthFactory();
2 await health.requestAuthorization([
3   HealthDataType.ACTIVE_ENERGY_BURNED,
4   HealthDataType.HEART_RATE,
5 ]);
```

Listing 5.23: Genehmigungsanfrage an *Google Fit*, bzw. *Apple Health*

Damit man im Anschluss die Dauer des Training erhält, wird beim Starten des Trainings die Startuhrzeit in den SharedPreferences mit dem Schlüssel `trainingStarted`

gespeichert. Sobald das Training beendet wird, wird zu dem `TrainingFinished`-Widget navigiert und dort die `finishTraining` Funktion aufgerufen. In dieser Funktion wird zuerst die Start- und Enduhrzeit bestimmt. Die Startuhrzeit wird aus dem SharedPreferences-Speicher geladen und die Enduhrzeit ist die aktuelle Uhrzeit. Damit wird die Dauer des Trainings berechnet und in der Variablen `trainingDuration` gespeichert.

Im Anschluss wird sowohl in dem SharedPreferences-Speicher, als auch in der Firebase-Datenbank die Variable `lastTrainingDay` gespeichert. Dadurch kann bestimmt werden, ob der Benutzer an diesem Tag bereits ein Training abgeschlossen hat und dass es in der Firebase-Datenbank keinen Tag geben wird, an dem zwei Trainingsstatistiken gespeichert wurden. Danach wird die Funktion `calculateCalories` mit den Parametern `startDate` und `endDate` aufgerufen.

In dieser Funktion wird zu Beginn noch einmal geprüft, ob die Genehmigungen der Daten von *Google Fit*, bzw. *Apple Health* erteilt wurden. Im Anschluss werden die Daten mit Hilfe der `health.getHealthDataFromTypes`-Funktion abgefragt. Die Parameter dieser Funktion sind `startDate`, `endDate` und `types`. `types` besteht dabei aus einer Liste aller Datentypen, die wir erhalten wollen. Bei uns sind das `ACTIVE_ENERGY_BURNED` und `HEART_RATE`. Es werden dann noch mit `HealthFactory.removeDuplicates` alle Duplikate aus der Liste entfernt. Es wird über die Liste aller Datenpunkte iteriert und es wird zwischen den einzelnen Datentypen unterscheiden. Falls der Datenpunkt vom Typ `HEART_RATE` ist, wird dieser zur Variable `sumHeartRate` addiert und die Variable `countHeartRate` wird um eins erhöht. Anderenfalls wird der Datenpunkt zur Variable `calculatedCalories` addiert. Im Anschluss wird noch die durchschnittliche Herzfrequenz während dem Training berechnet, indem die Summe aller Herzfrequenzen durch die Anzahl geteilt wird. Der Rückgabewert dieser Funktion ist die Variable `calculatedCalories`. (siehe Listing 5.24)

```
1 _healthDataList.forEach((x) {
2   switch (x.typeString) {
3     case 'HEART_RATE':
4       sumHeartRate += x.value;
5       countHeartRate++;
6       break;
7     case 'ACTIVE_ENERGY_BURNED':
8       calculatedCalories += x.value;
9       break;
10  }
11 });
12
13 averageHeartRate = sumHeartRate / countHeartRate;
```

Listing 5.24: Datenpunkte aus *Google Fit / Apple Health* verwenden

Im Anschluss werden die Werte dieses Trainings gespeichert. Um das zu vereinfachen wurde eine Klasse *StatisticModel* erstellt. Diese Klasse enthält die Variablen

`trainingDate`, `duration`, `calories` und `heartRate`. Die Werte dieses Trainings werden in einer Variablen statisticModel gespeichert und diese wird dann in den SharedPreferences-Speicher geladen und in die Firebase-Datenbank mit Hilfe der `uploadStatistic`-Funktion hochgeladen.

Auf dem `TrainingFinished`-Widget (siehe Abbildung 5.7) werden die Statistiken dieses Trainings angezeigt, es werden die Dauer des Trainings, die verbrannten Kalorien, die durchschnittliche Herzfrequenz und ein Diagramm über den Verlauf der Herzfrequenz dargestellt. Das Diagramm wurde mit Hilfe des *fl_chart*-Packages erstellt. Dafür wurde einer `LineChart` die Datenpunkte der Herzfrequenz, sowie die maximalen und minimalen Punkte der Herzfrequenz übergeben.

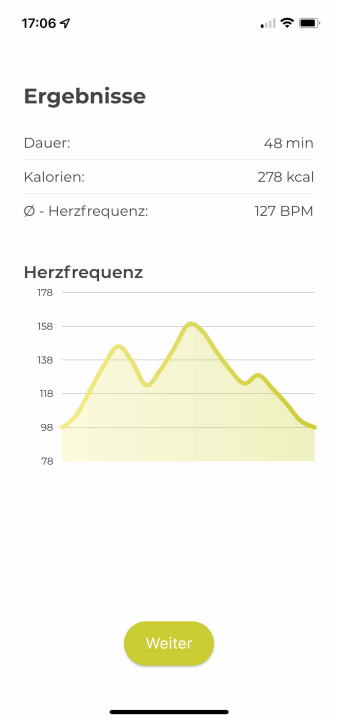

Abbildung 5.7: Statistik nach dem Training

Berechnung mit dem MET-Faktor

Zur Berechnung des Kalorienverbrauchs mit Hilfe des MET-Faktors musste zuerst bestimmt werden, welche Übungen der Benutzer während dem Training macht, welche Übungen er überspringt und wie lange er jede Übung ausführt. Dafür wurde bei der Generierung der Seite des PageViews in dem `TrainingBuilder`-Widgets bereits eine Liste mit allen Übungen erstellt, die auch die zusätzlichen *Side*-Übungen und die Vorbereitsungsseiten enthält. Außerdem wurde in der Klasse `TrainingModel` die neuen Variablen `trainingTimes` und `allExercises` und die Funktionen `setTraining` und `increaseTime` erstellt. Am Ende der Funktion `createTrainingPages` im `TrainingBuilder`-Widget wird dann mit der Liste `exercises` die Funktion `trainingModel.setTraining(exercises)` (siehe Listing 5.25) aufgerufen. In dieser Funktion wird für jede Übung ein Element in der Liste `trainingTimes` mit dem

Wert `0` hinzugefügt. Diese Liste wird dazu verwendet, für jede Übung die Dauer der Ausführung in Sekunden zu speichern. Außerdem wird in die Liste `allExercises` für jede Übung ein Element mit dem Wert der Übung als Objekt hinzugefügt.

```
List<int> trainingTimes = [];
List<Exercise> allExercises = [];

setTraining(List<Exercise> exercises) {
  exercises.forEach((element) {
    trainingTimes.add(0);
    allExercises.add(element);
  });
}
```

Listing 5.25: `setTraining`-Funktion

Die `increaseTime`-Funktion (siehe Listing 5.26) wird im Timer verwendet, um bei jeder Sekunde die Dauer der aktuellen Übung in der `trainingTimes`-Liste um eins zu erhöhen.

```
increaseTime(int index) {
  trainingTimes[index]++;
}
```

Listing 5.26: `increaseTime`-Funktion

Sobald der Benutzer das Training beendet, wird in der Funktion `finishTraining` zudem noch die Funktion `calculateAverageCalories` (siehe Listing 5.27) aufgerufen. In dieser Funktion wird zuerst das Gewicht des Benutzers aus dem SharedPreferences-Speicher geladen. Dann wird über alle Einträge der `trainingTimes`-Liste iteriert. Es wird geprüft, ob der aktuelle Eintrag eine Vorbereitsungs- oder eine Trainingsseite ist, da nur bei den Trainingsseiten die Kalorien berechnet werden sollen. Danach wird mit Hilfe der Dauer der Übung, des Gewichts des Benutzers und dem MET-Werts der Übung die Kalorien des aktuellen Eintrages berechnet. Danach wird dieser zu den Gesamtkalorien addiert.

```
  calculateAverageCalories() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    final userWeight = prefs.getInt('userWeight');
    double averageCalories = 0;

    for (int i = 0; i < trainingModel.trainingTimes.length; i++) {
      if (i % 2 == 1) {
        final exerciseTime = trainingModel.trainingTimes[i];
        final exercise = trainingModel.allExercises[i];

        double caloriesCurrentExercise = exercise.met * userWeight * (
    exerciseTime / 3600);
```

```
12          averageCalories += caloriesCurrentExercise;
13        }
14      }
15      return averageCalories;
16    }
```

Listing 5.27: `calculateAverageCalories`-Funktion

Nach dem Aufruf der Funktionen `calculateCalories` und `calculateAverageCalories` im `TrainingFinished`-Widget wird geprüft, ob Berechnung der Kalorien durch das health-Package Ergebnisse enthalten (siehe Listing 5.28). Falls diese Berechnung null ergeben, werden die Kalorienberechnungen mit dem MET-Faktor verwendet und man kann daraus schlussfolgern, dass der Benutzer keine Smartwatch während dem Training benutzt hat.

```
1 double calculatedCalories = await calculateCalories(startDate, endDate);
2 double averageCalories = await calculateAverageCalories();
3
4 calculatedCalories == 0 ? calories = averageCalories : calories =
      calculatedCalories;
```

Listing 5.28: Prüfung, ob Berechnung über *Google Fit* oder *Apple Health* Ergebnisse enthalten

5.8 After-Workout-Mahlzeit

Zur Bestimmung der After-Workout-Mahlzeiten wurden für den einfacheren Umgang mit den Mahlzeiten eine Klasse `AfterworkoutMeal` erstellt. In dieser Klasse wurden die Variablen `nameDE`, `nameEN`, `calories`, `imageURL`, `ingredientsDE`, `ingredientsEN`, `stepsDE` und `stepsEN` erstellt. `nameDE` und `nameEN` sind die Namen der Mahlzeiten, in deutscher und englischer Ausführung, `calories` entspricht dem Kalorienwert der Mahlzeit, `imageURL` enthält den Pfad zu dem zugehörigen Bild, `ingredientsDE` und `ingredientsEN` enthalten die Zutaten zur Zubereitung in Deutsch und Englisch und `stepsDE` und `stepsEN` enthalten die Schritte der Zubereitung in Deutsch und Englisch.

Im Anschluss wurde eine JSON-Datei mit einigen Rezepten erstellt. Dabei existiert jedes Feld, dass auch in der Klasse `AfterworkoutMealModel` erstellt wurde. (siehe Listing 5.29)

```
1 {
2     "nameDE": "TROPICAL SMOOTHIE BOWL",
3     "nameEN": "TROPICAL SMOOTHIE BOWL",
4     "calories": 342,
5     "imageURL": "assets/images/meals/tropical_smoothie_bowl.jpg",
6     "ingredientsDE": [
7       "1 kleine Banane, gefroren",
```

```
8       "100g Mango, gefroren",
9       "100 ml Mandelmilch, ungesüßt",
10      "10g Kokoschips"
11    ],
12    "ingredientsEN": [
13      "1 small banana, frozen",
14      "100g mango, frozen",
15      "100 ml almond milk, unsweetened",
16      "10g coconut chips"
17    ],
18    "stepsDE": [
19      "Alle Zutaten bis auf die Kokoschips in einen Mixer geben und cremig
    mixen.",
20      "In eine Schale geben, mit Kokoschips toppen und genießen!"
21    ],
22    "stepsEN": [
23      "Put all ingredients in a blender and mix until the desired
    consistency is achieved",
24      "Pour into a glass and enjoy!"
25    ]
26  },
```

Listing 5.29: Beispiel aus `AfterworkoutMeals.json`

Nachdem der Benutzer ein Training abgeschlossen hat und die Statistik-Seite des Trainings geschlossen wurde, wird in der Anwendung auf das Widget `AfterworkoutMeal` (siehe Abbildung 5.8) navigiert. Dieses Widget bekommt von der Statistik-Seite die verbrannten Kalorien des Trainings übergeben und zeigt passende Mahlzeiten zu dieser Kalorienanzahl an. In diesem Widget wird mit Hilfe eines `FutureBuilder`-Widgets die Funktion `fetchMeals` (siehe Listing 5.30) aufgerufen. In dieser Funktion wird zu Beginn die Funktion `fetchJson` aufgerufen, wodurch alle Mahlzeiten aus der `AfterworkoutMeals.json` Datei, in eine Liste gespeichert werden. Es wird eine neue Liste `selectedMeals` erstellt, in der die passenden Mahlzeiten gespeichert werden.

```
1 final numberOfSelectedMeals = 3;
2
3 fetchMeals(double calories, int numberOfMeals) async {
4     List<AfterworkoutMeal> allMeals = await fetchJson();
5     List<AfterworkoutMeal> selectedMeals = [];
6
7     for (AfterworkoutMeal meal in allMeals) {
8       if (calories - meal.calories > -50 && calories - meal.calories < 50)
    {
9         selectedMeals.add(meal);
10      }
11    }
12    if (selectedMeals.length == 0) {
13      selectedMeals = allMeals.sample(numberOfMeals);
14    } else {
15      selectedMeals = selectedMeals.sample(numberOfMeals);
```

```
16      }
17      saveMeal(selectedMeals);
18
19      return selectedMeals;
20    }
```

Listing 5.30: `fetchMeals`-Funktion

Es wird dann über die Liste aller Mahlzeiten iteriert und geprüft, ob die Kalorienanzahl der Mahlzeit nicht mehr als 50 Kalorien von den verbrauchten Kalorien abweicht. Wenn das erfüllt ist, wird diese Mahlzeit zu der `selectedMeals` Liste hinzugefügt. Im Anschluss wird geprüft, ob diese Liste Elemente enthält. Wenn sie keine Elemente enthält werden mit Hilfe der `sample`-Funktion drei zufällige Mahlzeiten aus der Liste aller Mahlzeiten ausgewählt. Anderenfalls werden aus der `selectedMeals`-Liste drei zufällige Mahlzeiten ausgewählt. Diese Mahlzeiten werden mit der `saveMeal`-Funktion in den SharedPreferences und in der Firebase-Datenbank gespeichert und werden als Rückgabewert der Funktion zurückgegeben.

Sobald der Benutzer auf eine dieser drei Mahlzeiten klickt, wird er in eine Detailansicht (siehe Abbildung 5.8) navigiert. Dort werden die Zutaten, sowie die Zubereitung dieser Mahlzeit angezeigt. Es wurde außerdem auf der Homepage ein Menü in der `BottomNavigationBar` hinzugefügt, wodurch der Benutzer auch nach dem Training noch die Rezepte und Zutaten der Mahlzeiten nachsehen kann. In diese Widget werden die Mahlzeiten von der Firebase-Datenbank heruntergeladen und dann auf dem Widget dargestellt.

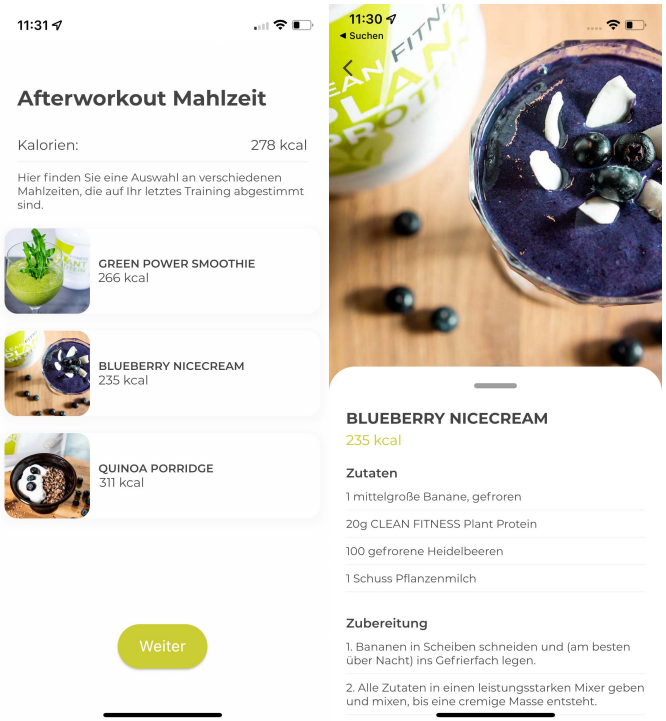

Abbildung 5.8: Die After-Workout-Mahlzeiten

5.9 Statistiken

Damit der Benutzer seinen Fortschritt über die 33 Tage sieht, wurde in der Anwendung eine Statistik über alle Trainings der 33 Tage erstellt (siehe Abbildung 5.9). Auf dieser Statistik werden die Anzahl der Trainings, die verbrannten Kalorien, die durchschnittliche Herzfrequenz, die durchschnittliche Dauer der Trainings und Diagramme über die 33 Tage für die verbrannten Kalorien, Herzfrequenzen und der Dauer der Trainings angezeigt.

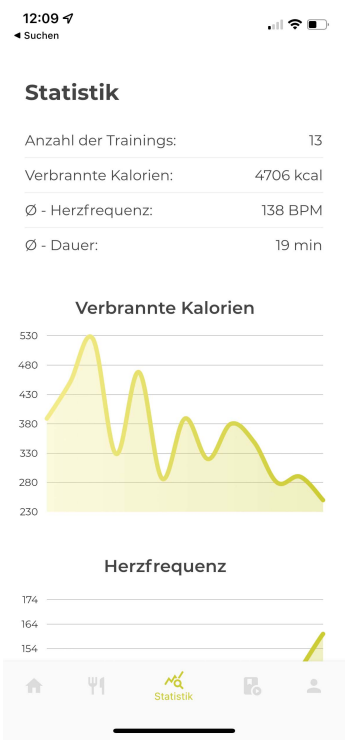

Abbildung 5.9: Die Statistik-Seite

In diesem Widget wird die Funktion `fetchStatistics` (siehe Listing 5.31) aufgerufen. Diese Funktion lädt zuerst die Statistiken aller Trainings aus der Firebase-Datenbank mit Hilfe der `applicationState.fetchStatistics`-Funktion. In dieser Funktion wird über alle Trainingsstatistiken iteriert. Dieser String wird dann in ein `StatisticModel`-Objekt gespeichert. Es wird geprüft, ob diese Statistik im Feld `heartRate` einen Eintrag besitzt. Wenn dieser Eintrag existiert, wird die Variable `countHeartRate` um eins erhöht und auf die Variable `sumHeartRate` wird die `heartRate` dieser Statistik addiert. Außerdem wird noch die Dauer auf `sumDuration` und die Kalorien des Trainings auf `sumCalories` addiert. `numberTrainings` wird außerdem noch um eins erhöht. Die durchschnittliche Dauer wird dann berechnet, indem die Summe aller Dauern durch die Anzahl der Trainingseinheiten geteilt wird und die durchschnittliche Herzfrequenz wird berechnet, indem die Summe aller Herzfrequenzen durch die Anzahl der Herzfrequenzen geteilt wird.

```
1  fetchStatistics() async {
2    List<String> trainingStats = await applicationState.fetchStatistics();
3
4    double countHeartRate = 0;
5    double sumDuration = 0;
6    double sumHeartRate = 0;
7    numberTrainings = 0;
8    sumCalories = 0;
9    averageDuration = 0;
10   averageHeartRate = 0;
11
12   for (String statistic in trainingStats) {
13     StatisticModel statisticModel = StatisticModel.fromJson(json.decode(
     statistic));
14     if (statisticModel.heartRate != 'NaN') {
15       countHeartRate++;
16       sumHeartRate += double.parse(statisticModel.heartRate);
17     }
18     sumDuration += double.parse(statisticModel.duration);
```

```
19        sumCalories += double.parse(statisticModel.calories);
20        numberTrainings++;
21      }
22      averageDuration = sumDuration / numberTrainings;
23      averageHeartRate = sumHeartRate / countHeartRate;
24    }
```

Listing 5.31: `fetchStatistics`-Funktion

5.10 Mediathek

Es sollte in der Anwendung außerdem noch eine Mediathek implementiert werden. Diese Mediathek stellt verschiedene Videos und Anleitung als Hilfestellung für die 33-Tage Challenge zur Verfügung. Es werden außerdem Motivationsvideos abgespielt, die dem Benutzer durch die 33-Tage Challenge helfen soll. Dabei wird jeden Tag ein neues Video angezeigt. Für den Umgang mit den Videos wurde eine neue Klasse `Video` erstellt. Das Feld `id` dieser Klasse wird zur eindeutigen Identifizierung der Videos benutzt. Das Feld `day` wird für die Motivationsvideos genutzt, damit man weiß, an welchem Tag dieses Video angezeigt werden soll. Die Felder `titleDE`, `titleEN`, `subTitleDE`, `subTitleEN`, `categoryDE`, sowie `categoryEN` sind der Titel, der Untertitel und die Kategorie des Videos, in englischer und deutscher Version. Es sind außerdem noch die Felder `imageURL` und `videoURL` erstellt worden. `imageURL` stellt das Vorschaubild und `videoURL` stellt die URL zu diesem Video dar.

Die Videos wurden in dem Firebase Storage hochgeladen, wie in Abschnitt 5.6 bereits beschrieben. Als Speicher für die Daten der einzelnen Videos wurde eine weitere Tabelle in der SQLite-Datenbank dieser Anwendung erstellt. Die Felder dieser Datenbank sind die gleichen Felder, wie in der Klasse `Video`. In der `DBProvider`-Klasse wurden außerdem noch weiter Funktionen erstellt, welche das Abrufen der Videodaten aus der Datenbank erleichtern. Es wurden die beiden Funktionen `getVideoByID` und `getAllVideos` hinzugefügt, wobei die Implementierung ähnlich wie bei Listing 5.16 ist. Außerdem wurden noch die Funktionen `getVideoByDay`, `getVideosByCategoryDE` und `getVideosByCategoryEN` erstellt. Die Funktion `getVideoByDay` (siehe Listing 5.32) wird dafür verwendet, das Motivationsvideos des übergebenen Tages zu erhalten. Die Funktionen `getVideosByCategoryDE` und `getVideosByCategoryEN` gibt eine Liste zurück, die alle Videos enthält, welche der übergebenen Kategorie zugehörig sind.

```
1 getVideoByDay(int day) async {
2   final db = await database;
3   var res = await db.query('Videos', where: 'day = ?', whereArgs: [day]);
4
5   return res.isNotEmpty ? Video.fromJson(res.first) : Null;
6 }
```

Listing 5.32: `getVideoByDay`-Funktion

Sobald der Benutzer auf der Homepage auf die Mediathek-Seite (siehe Abbildung 5.10) navigiert, wird in dem Widget `MediaLibrary` ein `FutureBuilder` mit der Funktion `getTodaysVideos` (siehe Listing 5.33) ausgeführt. Dadurch wird das Video des aktuellen Tages geladen. In dieser Funktion wird zuerst aus dem SharedPreferences-Speicher der aktuelle Tag geladen. Im Anschluss wird mit der Funktion `DBProvider.db.getVideoByDay` das Video des aktuellen Tages geladen. Dabei ist der Parameter der aktuelle Tag. Falls der Tag größer als 33 ist, wird dann der 33. Tag verwendet. Die Funktion gibt dann das erhaltene Video zurück. Es werden außerdem in diesem Widget noch Buttons für alle Motivationsvideos und Kategorien angezeigt.

```
Future<Video> getTodaysVideos() async {
  final prefs = await SharedPreferences.getInstance();
  int currentDay = prefs.getInt('currentDay');

  final _result = await DBProvider.db.getVideoByDay(currentDay <= 33 ?
   currentDay : 33);

  if (_resultDatabase != null)
    return _resultDatabase;
  else
    throw Exception('Failed to load Exercise!');
}
```

Listing 5.33: `getTodaysVideos`-Funktion

Klickt der Nutzer auf eine der Kategorien, wird in diesem Widget (siehe Abbildung 5.10) ein `FutureBuilder` mit der Funktion `getVideos` (siehe Listing 5.34) aufgerufen. In der Funktion wird geprüft, ob das System des Benutzers auf Deutsch oder Englisch eingestellt ist. Daraufhin wird entweder die `DBProvider.db.getVideosByCategoryDE` oder `DBProvider.db.getVideosByCategoryEN` ausgeführt. Das Ergebnis dieser Funktion wird dann in dem Widget verwendet, die Videovorschauen anzuzeigen.

```
Future<List<Video>> getVideos() async {
  final _resultDatabase = Localizations.localeOf(context).toString() == 'de
   '
      ? await DBProvider.db.getVideosByCategoryDE(categoryName)
      : await DBProvider.db.getVideosByCategoryEN(categoryName);

  if (_resultDatabase != null)
    return _resultDatabase;
  else
    throw Exception('Failed to load Exercise!');
}
```

Listing 5.34: `getVideos`-Funktion

Sobald der Nutzer entweder auf das heutige Video in der Bibliothek oder auf eines der Videos in einer der Kategorien klickt, wird ein `BottomSheet` mit Hilfe der `showModalBottomSheet`-Funktion aufgerufen. In diesem `BottomSheet` (siehe Abbildung 5.10) wird das passende Video zur angeklickten Vorschau, der Titel und der Untertitel des Videos angezeigt.

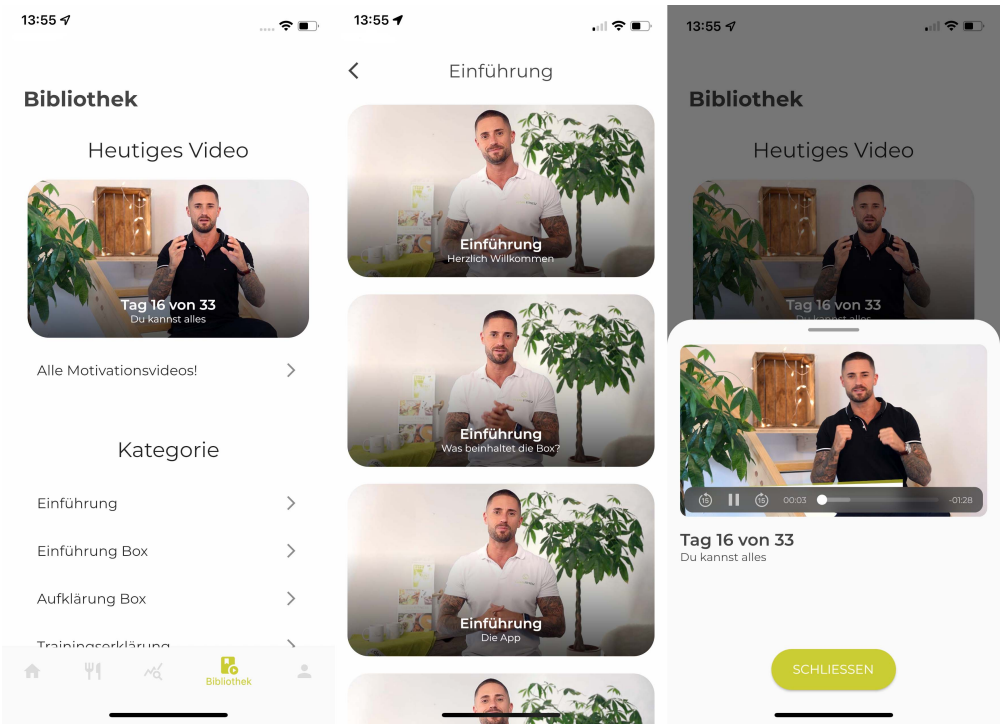

Abbildung 5.10: Die Mediathek

Kapitel 6

Fazit und Ausblick

6.1 Fazit

Ziel dieser Arbeit war es, eine voll funktionsfähige Anwendung zu entwickeln. Als Ergebnis dieser Bachelorarbeit entstand eine Smartphone-Anwendung mit Flutter, die es ermöglicht, die 33-Tage-Challenge mit dem Smartphone durchzuführen. Flutter macht die Anwendung modular und leicht erweiterbar, was bei der zukünftigen Entwicklung der Anwendung eine wichtige Rolle spielt. Das Resultat der Bearbeitung kann bereits problemlos dazu verwendet werden, eine 33-Tage Challenge mit Hilfe dieser Anwendung durchzuführen.

Zu den grundlegenden Funktionalitäten dieser Anwendung zählen zum Beispiel die Erstellung des Trainingsplans, die Durchführung des Trainings, die Berechnung der Kalorienanzahl, die Erstellung einer Auswahl an After-Workout-Mahlzeiten, die Erstellung von Statistiken über die gesamten 33 Tage und das Anzeigen einer Mediathek. Ebenso sollten eine Authentifizierung des Benutzers und die Sicherung aller Daten auf eine Datenbank implementiert werden. Ein wichtiger Meilenstein bei der Entwicklung war die Benutzerfreundlichkeit der Anwendung. Dabei sollte bei der Planung und Implementierung der Benutzeroberfläche und Funktionalität besonders auf die Benutzerfreundlichkeit geachtet werden.

Die in Kapitel 3 aufgeführten Anforderungen konnten in dieser Anwendung erfüllt werden. Die Umsetzung dieser Arbeit hat mein Wissen über die Cross-Plattform-Entwicklung mit Flutter und Firebase bereichert. Außerdem konnte ich mein bereits im Studium erworbenes Wissen einbringen und vertiefen.

6.2 Ausblick

Zur Weiterentwicklung der Anwendung kann gesagt werden, dass in Zukunft noch einige Aspekte dieser Anwendung verbessert und erweitert werden könnten. Eine Verbesserungen dieser Anwendung besteht in der Erweiterung der Daten in der `AfterworkoutMeals.json`-Datei. Aktuell kann es passieren, dass bei dem Vorschlag

einer Mahlzeit zufällige Mahlzeiten aus den Daten ausgewählt werden. Das liegt daran, dass nur ein bestimmter Bereich der Kalorien durch die Mahlzeiten abgedeckt ist. Wenn in Zukunft mehr Mahlzeiten in dieser Datei verfügbar sind, wird die Auswahl dieser Mahlzeiten verbessert.

Eine weitere Weiterentwicklung der Anwendung besteht in der Vervollständigung der Video und Bilder der Übungen. Der aktuelle Stand der Anwendung zeigt bei jeder Übung das selbe Bild/Video an, was in Zukunft nicht der Fall sein sollte. Außerdem wird in Zukunft für die Ansagen während dem Training keine Text-to-Speech Sprachausgabe mehr verwendet. Es werden Ansagen aufgenommen und diese anstelle der Text-to-Speech Ausgaben verwendet.

Außerdem könnte man in Zukunft die Anmeldung, bzw. die Registrierung der Benutzer über *Facebook*, *Google* und *Twitter*, wie in Abbildung 5.4 zu sehen, implementiert werden. In diesem Stand der Anwendung werden die Icons im Anmeldungs- und Registrierungsbildschirm als Platzhalter benutzt und die Funktionalitäten wurden noch nicht entwickelt.

Literaturverzeichnis

[1] Microsoft Corporation. Visual Studio Code - Code editing. Redefined. https://code.visualstudio.com/. Eingesehen am 24. October 2021 [Online].

[2] Microsoft Corporation. Xamarin. https://docs.microsoft.com/de-de/xamarin/. Eingesehen am 20. October 2021 [Online].

[3] Microsoft Corporation. Was ist Xamarin? https://docs.microsoft.com/de-de/xamarin/get-started/what-is-xamarin, September 2021. Eingesehen am 21. October 2021 [Online].

[4] Drifty Co. Ionic Framework. https://ionicframework.com/docs, Dezember 2020. Eingesehen am 21. October 2021 [Online].

[5] Cruxlab Inc. Xamarin vs Ionic vs React Native: differences under the hood. https://medium.com/swlh/xamarin-vs-ionic-vs-react-native-differences-under-the-hood-6b9cc3d2c826, September 2017. Eingesehen am 21. October 2021 [Online].

[6] Wenhao Wu. React Native vs Flutter, cross-platform mobile application frameworks. Master's thesis, Metropolia University of Applied Sciences, March 2018.

[7] Krystof Beuermann. Cross-Plattform-Apps entwickeln: React Native, Flutter und Co. https://entwickler.de/programmierung/cross-plattform-apps-entwickeln-react-native-flutter-und-co/, März 2019. Eingesehen am 20. October 2021 [Online].

[8] Meta Platforms Inc. Sensors - Expo Documentation. https://docs.expo.dev/versions/latest/sdk/sensors/. Eingesehen am 21. October 2021 [Online].

[9] Google LLC. Beautiful native apps in record time. https://flutter.dev/. Eingesehen am 21. October 2021 [Online].

[10] Google LLC. Flutter, React Native, Ionic, Xamarin - Google Trends. https://trends.google.de/trends/explore?geo=DE&q=Flutter,React%20Native,Ionic,Xamarin. Eingesehen am 20. October 2021 [Online].

[11] Stack Overflow Ltd. Stack Overflow Trends. https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native%2Cionic%2Cxamarin. Eingesehen am 20. October 2021 [Online].

[12] Stack Overflow Ltd. Stack Overflow. https://stackoverflow.com. Eingesehen am 21. October 2021 [Online].

[13] Stefania Pizza, Barry Brown, Donald Mcmillan, and Airi Lampinen. Smartwatch in vivo. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, Mai 2016.

[14] Apple Inc. Apple Watch (1st generation) - Technical Specifications. https://support.apple.com/kb/SP735, March 2019. Eingesehen am 23. October 2021 [Online].

[15] Apple Inc. Apple Watch Series 1 - Technical Specifications. https://support.apple.com/kb/SP745, März 2021. Eingesehen am 23. October 2021 [Online].

[16] Apple Inc. Apple Watch Series 3 - Technical Specifications. https://support.apple.com/kb/SP766, November 2021. Eingesehen am 23. October 2021 [Online].

[17] Apple Inc. Apple Watch Series 4 - Technical Specifications. https://support.apple.com/kb/SP778, März 2021. Eingesehen am 23. October 2021 [Online].

[18] Apple Inc. Apple Watch Series 5 - Technical Specifications. https://support.apple.com/kb/SP808, März 2021. Eingesehen am 23. October 2021 [Online].

[19] Apple Inc. Apple Watch Series 6 - Technical Specifications. https://support.apple.com/kb/SP826, Oktober 2021. Eingesehen am 23. October 2021 [Online].

[20] Apple Inc. Apple Watch SE - Technical Specifications. https://support.apple.com/kb/SP827, November 2021. Eingesehen am 23. October 2021 [Online].

[21] Dennis Troper. Android Wear, its time for a new name. https://www.blog.google/products/wear-os/android-wear-its-time-new-name/, Mar 2018. Eingesehen am 23. October 2021 [Online].

[22] Jessica Gietz. Smartwatch-Betriebssystem: Samsungs Tizen und Googles Wear OS verschmelzen zu Wear. https://blog.1und1.de/2021/06/15/smartwatch-betriebssystem-samsungs-tizen-und-googles-wear-os-verschmelzen-zu-wear/, Juni 2021. Eingesehen am 23. October 2021 [Online].

[23] George Batschinski. Backend as a Service – What is a BaaS? https://blog.back4app.com/backend-as-a-service-baas/. Eingesehen am 23. October 2021 [Online].

[24] Google LLC. Firebase. https://firebase.google.com/. Eingesehen am 23. October 2021 [Online].

[25] Google LLC. Firebase Authentication. https://firebase.google.com/docs/auth. Eingesehen am 23. October 2021 [Online].

[26] Google LLC. Cloude Storage. https://firebase.google.com/docs/storage. Eingesehen am 23. October 2021 [Online].

[27] Freeletics GmbH. Über Freeletics. https://www.freeletics.com/de/press/. Eingesehen am 23. October 2021 [Online].

[28] runtastic GmbH. Runtastic Homepage. https://www.runtastic.com/de/. Eingesehen am 24. October 2021 [Online].

[29] Erin MacPherson. How Accurate Are Apple Watch Calories? https://www.iphonelife.com/content/how-accurate-are-apple-watch-calories-how-to-ensure-theyre-accurate, September 2021. Eingesehen am 23. October 2021 [Online].

[30] Barbara E. Ainsworth. 2011 Compendium of Physical Activities. https://www.codinma.es/wp-content/uploads/2018/11/Second-compendium-METS-2011.pdf, 2011. Eingesehen am 23. October 2021 [Online].

[31] Atlassian. Pty Ltd. Gitflow Workflow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. Eingesehen am 24. October 2021 [Online].

[32] Google LLC. Android Studio. https://developer.android.com/studio. Eingesehen am 24. October 2021 [Online].

[33] Atlassian. Pty Ltd. Sourcetree Homepage. https://www.sourcetreeapp.com/. Eingesehen am 24. October 2021 [Online].

[34] Dash Overflow. provider | Flutter Package. https://pub.dev/packages/provider, September 2021. Eingesehen am 03. November 2021 [Online].

[35] Google LLC. Cloud Firestore. https://firebase.google.com/docs/firestore. Eingesehen am 08. November 2021 [Online].

Abbildungsverzeichnis

Listings