DHBW
Duale Hochschule
Baden-Württemberg

Architekturentwurf

Native Android – App für persönliche Fitnessassistenz mit individuellen Plänen und Zielen

Stand:	15.06.2020
Autor:	Ferenc Horvay
Kurs:	TINF18B5
Zielgruppe:	Fitnessinteressierte

Versionstabelle

Versionsnummer:	**Autor:**	**Änderungsvermerk**	**Zu Anforderungsdokument Version**
1.0	Ferenc Horvay	Initiale Version	1.0 – 02.12.2019

Inhaltsverzeichnis

Inhaltsverzeichnis ..... **III**
Einleitung ..... **IV**
Abkürzungsverzeichnis ..... **V**
Glossar ..... **VI**
Abbildungsverzeichnis ..... **VII**
Tabellenverzeichnis ..... **VIII**
1. Architekturstrategie **1**
1.1. Flyweight Design Pattern 1
1.2. Decorator Design Pattern 1
1.3. Data Object Pattern 2
2. Statische Sicht – zu persistierende Daten **3**
2.1. ERM und Beschreibung derer Komponenten 3
2.2. Hilfsentitäten 5
2.3. Beschreibung und Definition der Relationen 5
3. Statische Sicht: Objekttypen zur Laufzeit **9**
3.1. UML Klassendiagramm 9
3.1.1. DAO Design Komponenten 9
3.1.2. Decorator Design Komponenten 10
4. Dynamische Sicht **12**
4.1. Funktionseinheiten 12
4.1.1. Trainingspläne 12
4.1.2. Kalender 12
4.1.3. Benutzer 13
4.2. Prozessbeschreibung für Funktionseinheit 14
5. Auswahl des Technologie Stacks **15**
5.1. Anforderungen an den Technologie Stack 15
5.2. Option 1: Webserver MySQL Datenbank 16
5.3. Option 2: SQLite 16
5.4. Option 3: Firebase 17
5.5. Option 4: Realm 18
5.6. Entscheidungsmatrix 19
5.7. Bewertung der Entscheidungsmatrix 20
6. Fazit **21**
7. Literaturverzeichnis **22**

Einleitung

Mehr und mehr Menschen verlieren aufgrund ungesunder Ernährung, sowie weniger Bewegung ihre Sportlichkeit oder legen an Gewicht zu. Ebenso kann ein Abonnement im Fitness Studio sehr viel Zeit und Geld in Anspruch nehmen. Hierzu soll im Rahmen des Projektes eine App entwickelt werden, welche durch von zu Hause aus ausführbaren Übungen, für welche man weder Geräte noch Gewichte braucht, einen passenden Trainingsplan ermitteln kann.

Die entstandene App soll einen Besuch im Fitnessstudio ablösen, indem es anhand eingegebener Körpermaße und Präferenzen einen passenden Trainingsplan erstellt. Dieser beinhaltet nur Übungen, welche mittels des eigenen Körpergewichtes absolviert werden können und somit auch weniger Zeit in Anspruch nehmen, als ein Fitnessstudiobesuch.

Der User soll anhand der angelegten Übungen der App ermutigt werden an den ausgewählten Tagen Sport zu treiben und durch die individuelle Berechnung des Trainingsplans auch nicht überanstrengt zu werden. Somit soll die App den Benutzer nach und nach dabei unterstützen sein Traumgewicht zu erreichen.

Das nachfolgende Architekturdokument soll dazu dienen, eine schnelle und effektive Entwicklung zu ermöglichen, Zeiten und Ressourcen präzise einplanen zu können und die Weiterentwicklung des Systems, sowie das Verständnis der einzelnen Komponenten zu erleichtern.

Abkürzungsverzeichnis

API ..... *Application Programming Interface*
CRUD ..... *create, read, update, and delete*
DAO ..... *Data Object Pattern*
JSON ..... JavaScript Object Notation

Glossar

Begriff	*Erläuterung:*
Native App	Mit einer nativen App ist eine Anwendung gemeint, welche speziell für das Betriebssystem des jeweiligen Endgerätes konzipiert und entwickelt wurde.
BMI	Der Body-Mass-Index ist ein grober Richtwert, mit der man das Körpergewicht eines Menschen im Verhältnis zu seiner Körpergröße bewerten kann.
Trainingsplan	Hiermit werden die Cardio-, Übungs- oder Muskelaufbaupläne zusammengefasst.
UML	Unified Modeling Language ist eine etablierte, objektorientierte, standardisierte und werkzeugunterstütze Modelliersprache für die Visualisierung, Beschreibung, Spezifikation und Dokumentation von Systemen

Tabelle 1 Glossar

Abbildungsverzeichnis

Abbildung 1 ERM-Diagramm der zu persistierenden Daten 3
Abbildung 2 UML Klassendiagramm 11
Abbildung 3 UML Aktivitätsdiagramm Training speichern 14
Abbildung 4 MySQL Datenbank Verbindung 16
Abbildung 5 SQLite Datenbank Verbindung 16
Abbildung 6 Firebase Datenbank Verbindung 17
Abbildung 7 Erstellen einer Relation in Realm 18
Abbildung 8 Erstellen einer Relation in SQLite 18

Tabellenverzeichnis

Tabelle 1 Glossar VI
Tabelle 2 Beschreibung der Entitäten im Kontext des ERM 4
Tabelle 3 Beschreibung der Hilfsentitäten 5
Tabelle 4 Termin Relation 6
Tabelle 5 User Relation 6
Tabelle 6 Fitnessplan Relation 7
Tabelle 7 Übung Relation 7
Tabelle 8 TerminToUser Hilfsrelation 8
Tabelle 9 UserToFitnessplan Hilfsrelation 8
Tabelle 10 FitnessplanToÜbung Hilfsrelation 8
Tabelle 11 Beschreibung der DAO Design Komponenten 9
Tabelle 12 Beschreibung der Decorator Design Komponenten 10
Tabelle 13 Beschreibung der Punktevergabe 19
Tabelle 14 Entscheidungsmatrix Datenbanksystem 19

1. Architekturstrategie

Für die Architekturstrategie des Projektes wurden folgende Design Patterns analysiert und die für das Projekt spezifischen Vor- und Nachteile genauer erläutert.

1.1. Flyweight Design Pattern

Das Flyweight design Pattern, ist ein strukturelles design pattern, welches verwendet wird, um die Anzahl an Objekten, welche zur Laufzeit erstellt werden zu minimieren. Diese Objekte sind nach ihrer Erzeugung unveränderbar und werden anschließend in einer HashMap abgespeichert.
Dieses Pattern bietet den Anwender die Möglichkeit die Hauptspeichernutzung seiner Applikation zu reduzieren und dadurch auch die Zeit der Erstellung dieser Objekte zu verringern.
Da die Architektur der zu erstellenden Android Applikation von diesem Design Pattern jedoch kaum profitieren würde, wird das Flyweight Design Pattern keinen Einfluss auf die Architektur der Applikation haben.

1.2. Decorator Design Pattern

Das Decorator Design Pattern ermöglicht das dynamische Hinzufügen von Fähigkeiten zu einer Klasse. Dazu wird die Klasse, welche wir erweitern möchten (Komponente), mit anderen Klassen (Dekorierer) dekoriert. Das heißt, der Dekorator umschließt die Komponente.
Dadurch können folgende Vorteile entstehen:

- Komponenten können mehrfach mit verschiedenen Dekoratoren umhüllt werden und damit in ihrem Verhalten flexibel und dynamisch modifiziert werden.
- Erweiterungen können ohne Codeänderungen an bestehenden Komponenten in das System integriert werden.
- Coderedundanzen werden vermieden, da das Verhalten eines speziellen Dekorators nur einmal implementiert werden muss.

Anhand der vorgestellten Vorteile des Decorator Design Pattern, in Verbindung mit der gegebenen Architektur der Applikation, wird das erwähnte Design Pattern als „Way-to-go“ bei der Entwicklung der Applikation verwendet.

1.3. Data Object Pattern

Das Data Access Object Design Pattern (DAO) ermöglicht es die Applikationsebene von der Persistierenden Ebene (in diesem Fall die Datenbank) zu trennen. Dadurch lässt sich jederzeit die unter der Applikation liegende Datenbank ohne Programmcodeänderung auswechseln. Hierbei muss lediglich die API welche zur Datenbank spricht ausgetauscht werden.
Da die Umsetzung dieses Patterns relativ einfach ist und eine Austauschbarkeit der persistierenden Schicht bietet, wird das DAO Pattern ebenso in der Architektur der Applikation berücksichtigt.

2. Statische Sicht – zu persistierende Daten

Im nachfolgenden Abschnitt werden die zu persistierenden Daten der nativen Android Fitness Applikation genauer beleuchtet. Diese Schicht wird genauer erläutert, um eine schnelle und effektive Entwicklung zu ermöglichen und im Falle einer Weiterentwicklung des Systems das Datenmodell langfristig erhalten bleiben kann. Da vorerst nicht zu erwarten ist, dass sich die Struktur der Daten innerhalb der Datenbank ändert wurde hierzu das Konzept einer relationalen Datenbank verwirklicht. Diese Entscheidung wird außerdem durch die Verwendung des DAO Design Patterns, wie in Kapitel 1.3 beschrieben gestärkt.

2.1. ERM und Beschreibung derer Komponenten

Im folgenden Entity Relationship Diagramm wurden die vier Entitäten mit ihren dazugehörigen Attributen und Primärschlüsseln genauer dargestellt.
Hilfsentitäten, welche Beispielsweise durch Auflösung der „n:m“, oder „1:n“ Beziehungen entstehen, werden dann im späteren Teil dieses Kapitels genauer beschrieben und definiert.

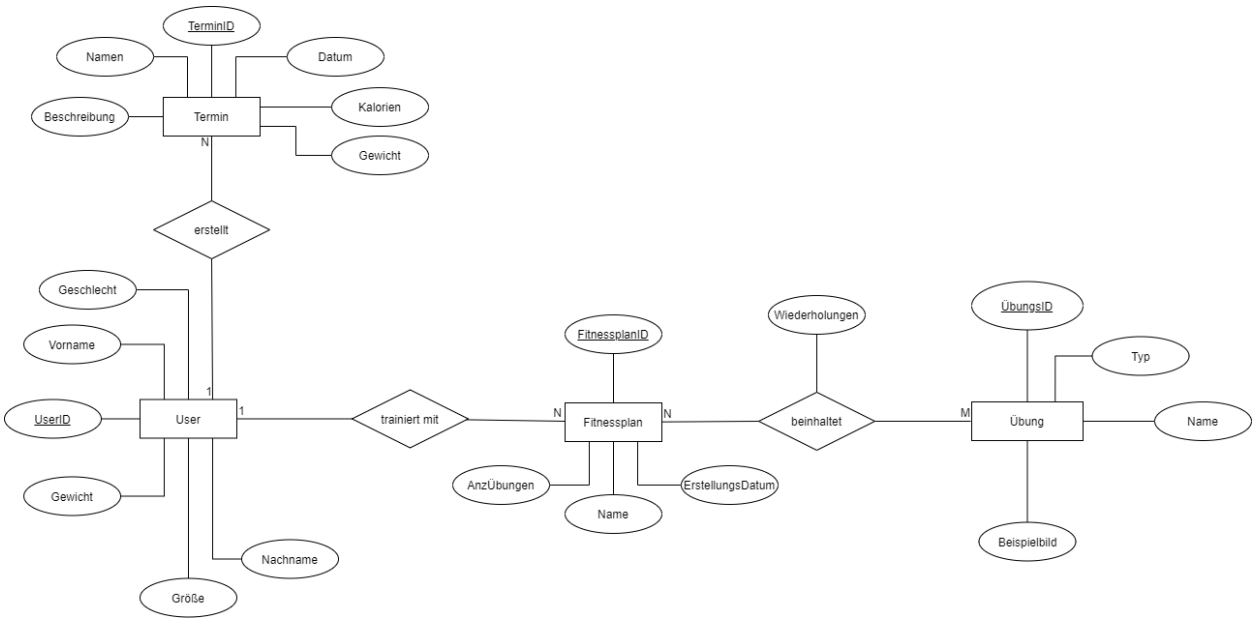

Abbildung 1 ERM-Diagramm der zu persistierenden Daten

Entität	Beschreibung
Termin	Termine, welche in der fertigen App innerhalb des Kalenders angezeigt werden, besitzen eine eigene Relation, welche mit dem Benutzer in Verbindung steht.
User	In der Relation des Benutzers werden allgemeine Informationen, die einen Benutzer betreffen, festgehalten.
Fitnessplan	Ein Fitnessplan wird ebenso als Relation dargestellt, da dieser, wie auch ein Termin, in direkter Verbindung mit dem Benutzer steht.
Übung	Eine Übung ist ein Teil eines Fitnessplans und besitzt die genauen Informationen, die eine Übung definiert, wie zum Beispiel die Art dieser (Muskel oder Ausdauerübung).

Tabelle 2 Beschreibung der Entitäten im Kontext des ERM

2.2. Hilfsentitäten

Durch die Auflösung der „n:m“, oder „1:n“ Beziehungen entstehen folgende Hilfsentitäten, welche ebenso in einer eigenen Relation dargestellt werden müssen.

Entität	Beschreibung
TerminToUser	Diese Hilfsentität löst die Beziehung zwischen einem User und mehreren Terminen auf. Sie beinhaltet deshalb den User, sowie dessen Termine.
UserToFitnessPlan	Ein User kann mit mehreren Fitnessplänen trainieren, weshalb diese Beziehung notwendig ist.
FitnessplanToÜbung	Da ein Fitnessplan mehrere Übungen verschiedener Arten beinhalten kann, muss an dieser Stelle ebenso eine Hilfsentität erstellt werden.

Tabelle 3 Beschreibung der Hilfsentitäten

2.3. Beschreibung und Definition der Relationen

Im nachfolgenden Abschnitt werden die zuvor definierten Relationen in tabellarischer Form inklusive. Beispieldaten beschrieben. Dabei wird darauf eingegangen, wieso und weshalb die spezifizierten Typen für die, aus dem ERM ablesbaren, Attribute festgelegt wurden.

Primärschlüssel werden zur besseren Übersicht in den folgenden Tabellen **Fett** dargestellt. Diese Schlüssel dürfen ebenso nicht null sein!

Termin					
int	varchar(255)	date	varchar(255)	int	int
TerminID	Name	Datum	Beschreibung	Kalorien	Gewicht
1	Trainiert	2020-02-05	Trainiert mit Fitnessplan XY		68
2	Burger gegessen	2020-01-17	Essen bei Burger Marie	950	

Tabelle 4 Termin Relation

Ein Termin besteht aus den hier gezeigten Attributen. Der Name, sowie die Beschreibung des Termins, können (fast) beliebig groß ausfallen und werden deshalb mit dem Datentyp varchar(255) abgespeichert. Da ein Termin später innerhalb eines Kalenders angezeigt werden soll, muss dieser Termin natürlich auch ein Datum besitzen, welcher mit dem Datentyp date abgespeichert wird. Optional sind in diesem Fall die Attribute Kalorien, oder Gewicht. Diese können vom Nutzer in den Termin gesetzt werden, um festhalten zu können wie viele Kalorien man zum Beispiel über einen Monat zu sich genommen hat, oder auch um ein Diagramm über den Gewichtsverlauf zu erstellen.

User					
int	varchar(255)	varchar(255)	double	int	varchar(9)
UserID	Vorname	Nachname	Gewicht	Größe	Geschlecht
1	Ferenc	Horvay	68,5	186	Männlich
2	Saskia	Esken	60,0	170	Weiblich

Tabelle 5 User Relation

Der User besitzt einen Vor- und Nachnamen, sowie ein Geschlecht. Diese Werte werden in einem varchar der Größe 255 bzw. 9 abgespeichert. Der varchar der Größe 9 lässt dank seiner Größe von 9 Bytes die Wörter „Männlich“, „Weiblich“ und „Divers“ zu, was allen grundlegend anerkannten Geschlechtern entsprechen sollte. Das Gewicht des Users wird in einem double abgespeichert, um im späteren Verlauf der App den BMI genauer berechnen zu können. Die Größe des Users wird in Zentimeter angegeben, um Kommafehler zu vermeiden. Aus diesem Grund handelt es sich hierbei um einen Interger-Wert.

Fitnessplan			
int	varchar(255)	int	date
FitnessplanID	Name	Ausführungen	Erstellungsdatum
1	Lockeres Workout	2	2020-02-05
2	Train harder	4	2020-03-17

Tabelle 6 Fitnessplan Relation

Ein Fitnessplan besteht aus einem Namen, welcher eine variable Länge besitzen darf. Zudem aus einem Erstellungsdatum, welches zur Sortierung aller Fitnesspläne von Vorteil ist und einem Integer-Wert, welcher angibt, wie oft dieser Plan bei einem Workout wiederholt werden soll.

Übung			
int	varchar(255)	varchar(12)	varchar(255)
ÜbungID	Name	Typ	Beispielbild
1	Bicycle Crunches	Muskelaufbau	@drawable/ByclCrunches
2	Hampelmann	Ausdauer	@drawable/hampelmann

Tabelle 7 Übung Relation

Eine Übung besitzt einen Namen und einen Typ, welche ebenso eine variable Länge besitzen können. Der Typ gibt in diesem Falle an, ob es sich um eine Ausdauer oder Muskelaufbau Übung handelt. Da das längere Wort Muskelaufbau in 12 Bytes passen, wurde dieser varchar auf die eben genannte Länge von 12 Bytes begrenzt. Da innerhalb von Android Studio Bilder, welche man in der App anzeigen möchte, im Ordner drawable abgespeichert werden, besitzt eine Übung ebenso einen varchar, welcher eine Länge von 255 Bytes besitzt. Dieser beinhaltet dann einen Verweis auf das Bild im oben genannten Ordner.

TerminToUser	
int	int
TerminID	**UserID**
1	1
2	1
3	1
4	2

Tabelle 8 TerminToUser Hilfsrelation

UserToFitnessplan	
Int	Int
UserID	**FitnessplanID**
1	1
1	2
1	3
1	4

Tabelle 9 UserToFitnessplan Hilfsrelation

FitnessplanToÜbung		
Int	Int	int
FitnessplanID	**ÜbungID**	Wiederholungen
1	1	15
1	2	15
1	3	20
1	4	10

Tabelle 10 FitnessplanToÜbung Hilfsrelation

Da diese Relation die Beziehung zwischen einem Fitnessplan und einer Übung wiederspiegeln soll, beinhaltet sie außer den Primärschlüsseln der jeweiligen Relationen, ebenfalls das Attribut Wiederholungen. Dieses Attribut ist ein Integer-Wert, welcher angibt, wie oft die Übung innerhalb des Fitnessplans wiederholt werden soll. Da eine Übung nur ganz oder gar nicht durchgeführt werden kann, ist dieser Wert ein Integer.

3. Statische Sicht: Objekttypen zur Laufzeit

Im nachfolgenden Kapitel werden alle zur Laufzeit benötigten Objekte in einem UML Klassendiagramm skizziert und im danach genauer beschrieben.

3.1. UML Klassendiagramm

Das nachfolgende Klassendiagramm beschreibt die einzelnen Klassen, welche nach den in Kapitel 1. beschriebenen Design Pattern entwickelt wurden.
Aus Gründen der Übersichtlichkeit wurden hierbei erstmals nur Getter-Methoden beschrieben. Vorerst kann davon ausgegangen werden, dass zu jeder beschriebenen Getter-Methode analog auch eine Setter-Methode implementiert wird.
Decorator Design Pattern

3.1.1. DAO Design Komponenten

Das DAO Design Pattern wird im folgenden UML Klassendiagramm umgesetzt, indem die zu persistierende Daten in drei verschiedene Java Objekte aufgeteilt wird:

Objekttyp	Beschreibung
Klasse	Klasse, welche als Container dient, um zum Beispiel den Namen, Alter, Gewicht, … eines Benutzers zu speichern.
Interface	Interface, welches eine abstrakte API darstellt, um CRUD Operationen auf der Datenbank auszuführen.
Klasse	Klasse, welche das oben genannte Interface implementiert und um ihre tatsächliche Funktionalität erweitert. Diese Klasse wird dann benutzt, um Benutzer Objekte zu speichern, löschen oder zu bearbeiten.

Tabelle 11 Beschreibung der DAO Design Komponenten

3.1.2. Decorator Design Komponenten

Das Decorator Design Pattern wird im durch die folgenden Java Objekte umgesetzt:

Objekttyp	Beschreibung
Interface	Wird von dem konkreten Muskel- oder Cardio- oder Übunsplan erweitert.
Abstrakte Klasse	Sie implementiert das Fitnessplan-Interface.
Klasse	Stellt später die Erweiterungen eines Fitnessplans dar, wie zum Beispiel Bicycle Crunches, welche den FitnessplanDecorator erweitern.
Klasse	Die drei Arten von Plänen (Muskel, Ausdauer und Übung), welche das Fitnessplan-Interface implementieren.

Tabelle 12 Beschreibung der Decorator Design Komponenten

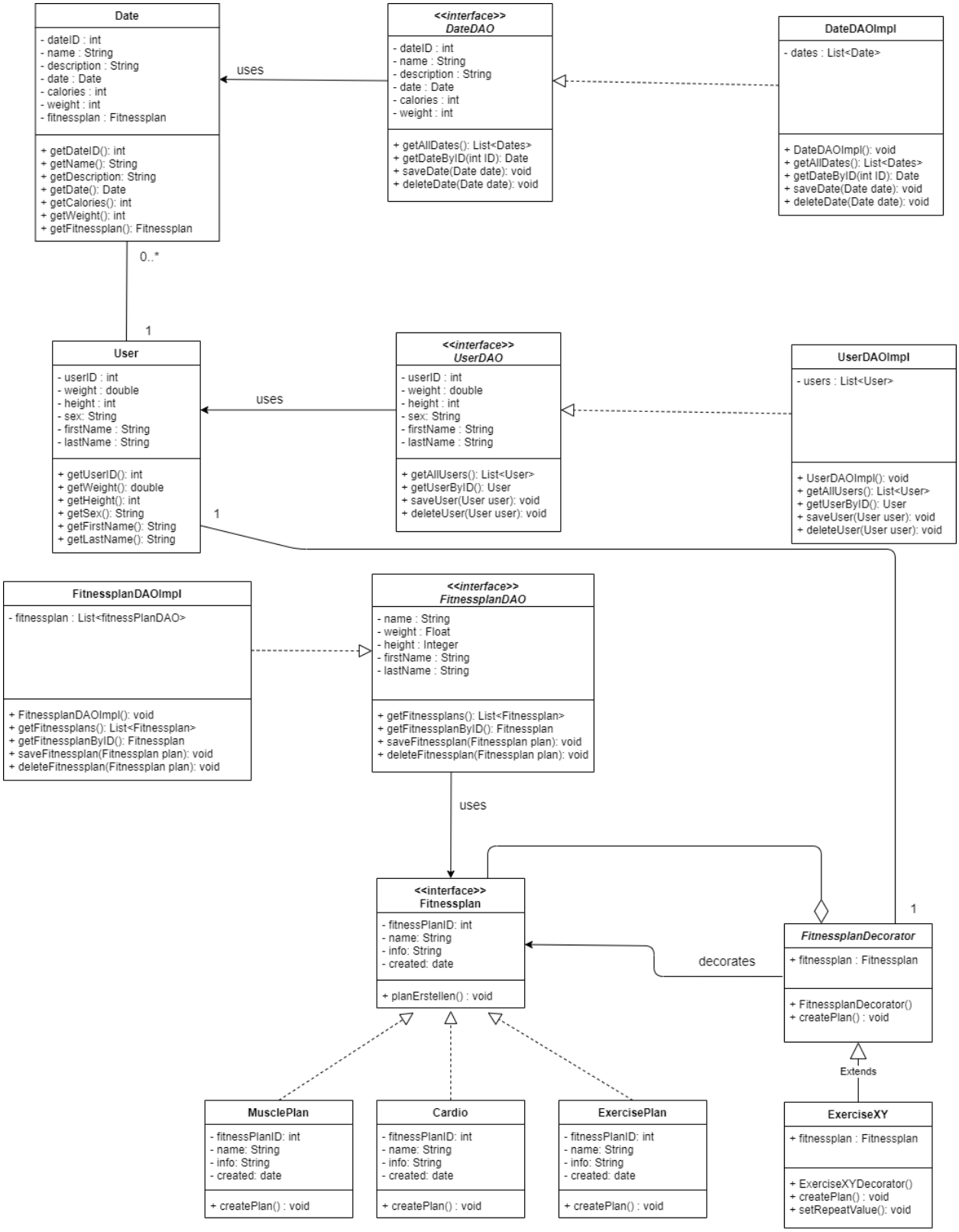

Abbildung 2 UML Klassendiagramm

4. Dynamische Sicht

4.1. Funktionseinheiten

Im nachfolgenden Kapitel werden die genaueren Funktionseinheiten definiert. Ein „R“ bezieht sich dabei auf die im Anforderungsdokument niedergeschriebenen Anforderungen.

4.1.1. Trainingspläne

- Cardioplan erstellen – Siehe R6
- Cardioplan bearbeiten – Siehe R6
- Cardioplan löschen – Siehe R12
- Muskelaufbauplan erstellen – Siehe R6
- Muskelaufbauplan bearbeiten – Siehe R6
- Muskelaufbauplan löschen – Siehe R12
- Übungsplan erstellen – Siehe R6
- Übungsplan bearbeiten – Siehe R6
- Übungsplan löschen – Siehe R12
- Alle Trainingspläne anzeigen – Siehe R11
- Trainingspläne suchen – Siehe R6
- Akustische Signale ausgeben – Siehe R15
- Beispiel Bilder/Videos zu einer Übung anzeigen – Siehe R25

4.1.2. Kalender

- Kalendereintrag erstellen – Siehe R22
- Kalendereintrag löschen – Siehe R22
- Kalendereintrag bearbeiten – Siehe R22
- Kalendereintrag farblich markieren – Siehe R18
- Gewicht in einem Kalendereintrag hinterlegen – Siehe R13
- Kalorien in einem Kalendereintrag hinterlegen – Siehe R17
- Training in einem Kalendereintrag hinterlegen – Siehe R23
- Kalendereinträge anzeigen – Siehe R22
- Kalendereinträge suchen – Siehe R22
- Gewichtdiagramm erstellen – Siehe R14
- Kaloriendiagramm erstellen – Siehe R14

4.1.3. Benutzer

- Benutzer erstellen – Siehe R1
- Benutzer löschen – Siehe R1
- Benutzer bearbeiten – Siehe R1
- Benutzer wechseln – Siehe R1
- Benutzer anzeigen – Siehe R1
- Gewicht setzen – Siehe R1
- Größe setzen – Siehe R1
- Alter setzen – Siehe R1
- Geschlecht setzen – Siehe R1
- BMI berechnen – Siehe R1
- BMI anzeigen – Siehe R2
- Standard BMI setzen – Siehe R4

4.2. Prozessbeschreibung für Funktionseinheit

Im nachfolgenden UML Aktivitätsdiagramm wird die Funktionseinheit des Kalenders genauer dargestellt. Dabei wird auf die Funktion eingegangen, wie ein Benutzer ein abgeschlossenes Training in den Kalender eintragen kann.

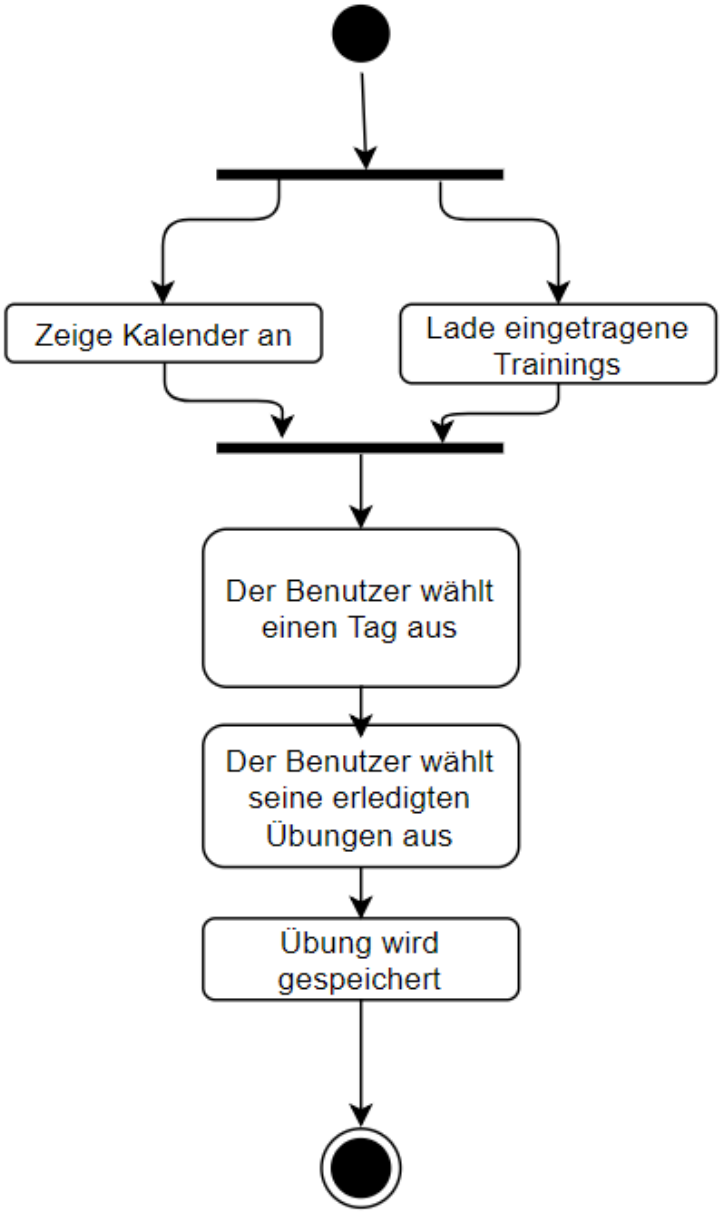

Abbildung 3 UML Aktivitätsdiagramm Training speichern

5. Auswahl des Technologie Stacks

5.1. Anforderungen an den Technologie Stack

Die Entscheidung über den Einsatz einer Datenbanktechnologie ist in der Konzeptionsphase eines Softwareprojekts essenziell. Hierbei werden konkrete Anforderungen und K.O. Kriterien definiert und evaluiert.
Wie im Anforderungsdokument niedergeschrieben, gab es mehrere Anforderungen an die Applikation und damit auch an die darunterliegende Datenbanktechnologie.

- Ein wichtiger Aspekt bei der Verwendung einer Datenbank ist die Ressourcenverwendung des Host-Systems. Die Applikation soll das Handy nicht unnötig in die Knie zwingen, geschweige denn, den Akku des Geräts unnötig belasten. Aus diesem Grund fließt dieser Aspekt in der Bewertungsskala mit 15% ein.
- In der Anforderung R-30 wurde niedergeschrieben, dass die Applikation auch ohne eine aktive Internetverbindung funktionieren soll. Deshalb darf keines der zu evaluierenden Datenbanksystemen nur mit einer aktiven Internetverbindung funktionieren. Da dieser Aspekt als K.O. Bedingung gilt, fließt er mit 30% in die Entscheidungsmatrix ein.
- Das Datenbanksystem sollte die Applikation nicht unnötig ausbremsen, aus diesem Grund wurde dieses Kriterium mit 20% in die Entscheidungsmatrix aufgenommen.
- Das Datenbanksystem sollte ebenso Programmiertechnisch einfach umsetzbar sein. Da dieser Aspekt den letztendlichen Nutzer der Applikation nicht betrifft, fließt dieser Aspekt nur mit geringen 15% in die Entscheidungsmatrix ein.

5.2. Option 1: Webserver MySQL Datenbank

Die erste Option zur Umsetzung der Datenbank innerhalb der App, ist die Variante einen externen MySQL Server über einen Connector innerhalb der App anzusprechen. Diese Version verlangt allerdings eine dauerhafte Verbindung zur Datenbank, welche nicht immer gewährleistet sein kann und außerdem nicht mit dem K.O. Kriterium zu vereinbaren ist.

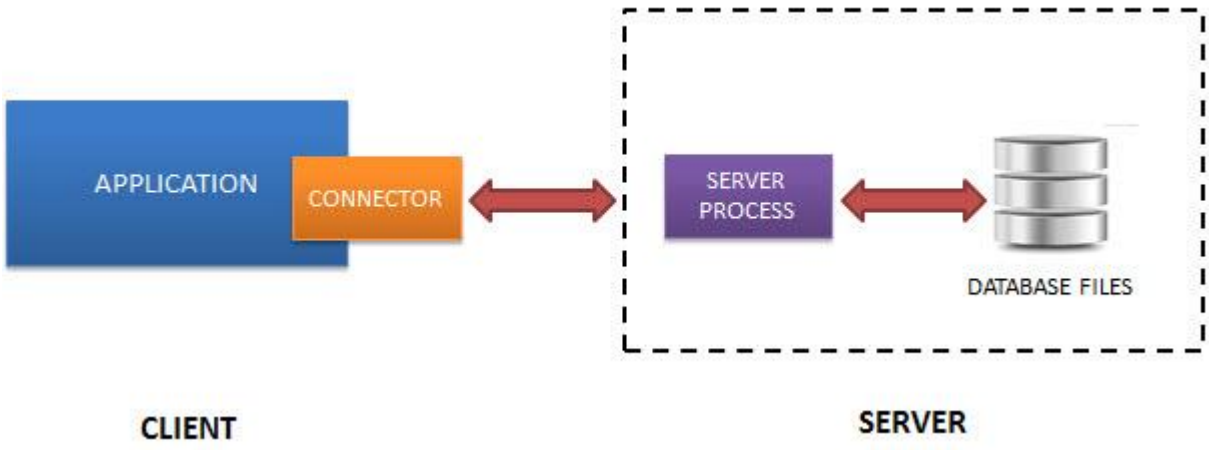

Abbildung 4 MySQL Datenbank Verbindung

5.3. Option 2: SQLite

Die zweite Option zur persistenten Speicherung aller Daten wäre die Verwendung einer SQLite Datenbank. Eine SQLite Datenbank definiert sich als kleinere Version der relationalen MySQL Datenbank. Sie ist speziell für die Verwendung innerhalb von Applikationen entwickelt worden und benötigt deshalb auch keine aktive Internetverbindung. Dennoch unterstützt diese Open Source Lösung alle Features, welche auch eine normale MySQL Datenbank unterstützen würde und ist auf dem Haupt- oder Sekundärspeichers des Host Geräts verwendbar. Außerdem ist die SQLite Datenbank dank ihrer Kompaktheit sehr schnell und verbraucht im Vergleich zu ihrem großen Bruder, MySQL, zur Laufzeit wenig Hauptspeicher.

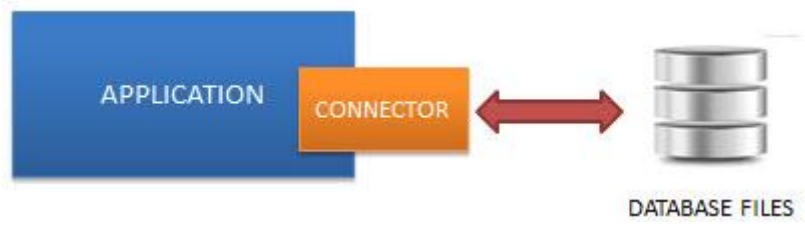

Abbildung 5 SQLite Datenbank Verbindung

5.4. Option 3: Firebase

Google's Firebase bietet Android Entwicklern die Möglichkeit Daten (im JSON Format) in einer in der Cloud gehosteten NoSQL Datenbank zu persistieren. Google setzt dabei auf eine Lokal gehostete Datenbank, welche sich bei einer aktiven Internetverbindung automatisch mit der in der Cloud befindlichen MySQL Datenbank synchronisiert. So bietet diese Option ebenso die Möglichkeit, ohne eine aktive Internetverbindung zu funktionieren, bringt dadurch aber erhöhte Kosten durch das Abonnement bei Google, sowie einen gewissen Mehraufwand, bei der Verwendung von Google's selbst entwickelten SDK.

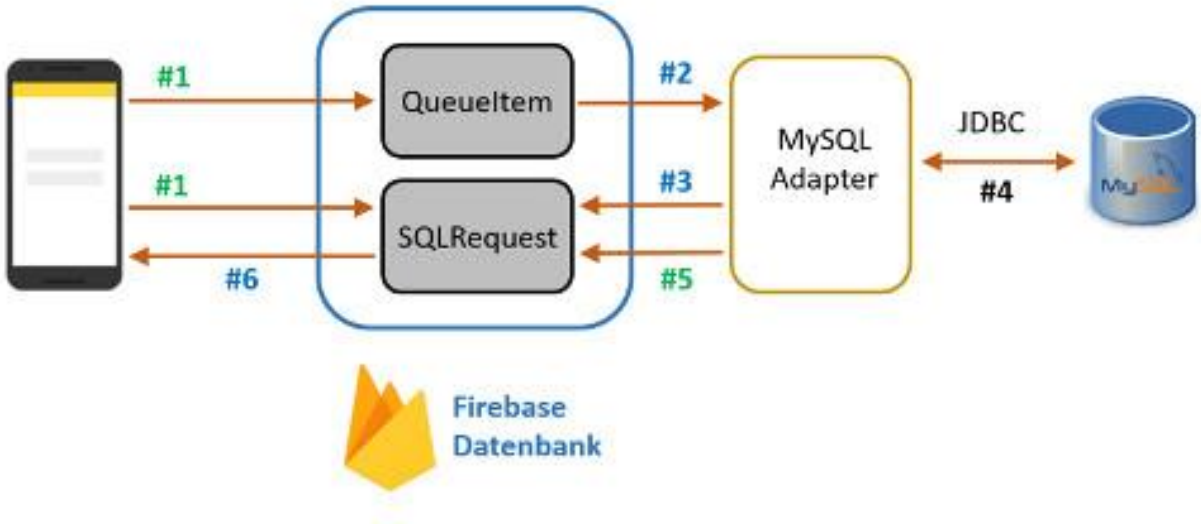

Abbildung 6 Firebase Datenbank Verbindung

5.5. Option 4: Realm

Realm ist eine Alternative zur SQLite Datenbank. Dabei ist Realm jedoch keine relationale, sondern eine NoSQL Datenbank, welche speziell für Mobilgeräte entwickelt wurde. Der Vorteil an Realm im Gegensatz zu einer SQLite Datenbank ist hierbei jedoch die Vereinfachung der SQL Transaktionen.

```
public class Favourites extends RealmObject{
    String title;
    String imageLink;
    @Nullable
    ImageResponse image;
public Favourites(String title, String imageLink, @Nullable ImageR
        this.title = title;
        this.imageLink = imageLink;
        this.image = image;
    }
}
```

Abbildung 7 Erstellen einer Relation in Realm

```
  private static final String SQL_CREATE_FAVOURITE =
            "CREATE TABLE " + FavouritesDBContract.FavouritesEntry
                    FavouritesDBContract.FavouritesEntry._ID + INT
                    FavouritesDBContract.FavouritesEntry.COLUMN_NA
                    FavouritesDBContract.FavouritesEntry.COLUMN_NA
                    " )";
  public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_FAVOURITE);
    }
```

Abbildung 8 Erstellen einer Relation in SQLite

5.6. Entscheidungsmatrix

Innerhalb der Entscheidungsmatrix kann ein Kriterium eine Punktzahl von 0 bis 15 Punkten erreichen. Wie eine Punktzahl zustande kommt, wird nachfolgend nochmals kurz erläutert:

Punkte	Beschreibung
13-15	Wenn die Leistung den Kriterien in besonderem Maße entspricht.
10-12	Wenn die Leistung den Kriterien voll entspricht.
9-7	Wenn die Leistung im Allgemeinen den Kriterien entspricht.
4-6	Wenn die Leistung zwar Mängel aufweist, aber im Ganzen den Kriterien noch entspricht.
1-3	Wenn die Leistung den Kriterien nicht entspricht, jedoch in absehbarer Zeit behoben werden kann.
0	Wenn die Leistung den Anforderungen* nicht entspricht und selbst in absehbarer Zeit nicht behoben werden können.

Tabelle 13 Beschreibung der Punktevergabe

Kriterien	Gewichtung	Option 1	Option 2	Option 3	Option 4
CPU / Memory verbrauch	15%	13	8	8	8
Offline Verfügbarkeit	30%	0	15	11	11
Schnelligkeit	20%	8	11	8	12
Sicherheit	20%	5	8	8	8
Bedienbarkeit	15%	8	8	8	10
Summe	100%	5,75	10,7	8,9	10

Tabelle 14 Entscheidungsmatrix Datenbanksystem

5.7. Bewertung der Entscheidungsmatrix

Anhand der Entscheidungsmatrix ist zu entnehmen, dass sich die technische Umsetzung der Datenbank am besten durch eine SQLite Datenbank verwirklichen lässt. Denn diese stellt die beste Alternative hinsichtlich der oben genannten Kriterien dar.

6. Fazit

Dieser Architekturentwurf beschreibt eine mögliche Architektur, um eine native Android Fitness-App zu entwickeln. Dabei wurden Anforderungen, welche innerhalb der ersten Lehrveranstaltung „Grundlagen des Softwareengineerings“ erhoben wurden.
Das dabei entstandene Dokument soll somit eine verstehbare Darstellung der zentralen Entwurfsentscheidungen aufzeigen, die Wiederverwendbarkeit der Architektur stärken, sowie den grundlegenden Bauplan für die spätere Entwicklung der Applikation gezeigt haben.

7. Literaturverzeichnis

- Anforderungsdokument Ferenc Horvay Stand 02.12.2019