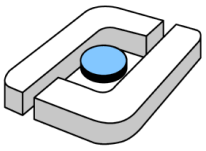


Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Schriftliche Projektarbeit zum Thema:

Exemplarische Umsetzungen und formale Kriterien für die Gestaltung wissenschaftlicher Arbeiten mit Microsoft Word

im Rahmen des Moduls
Programmierung 3 (MI),
des Studiengangs Informatik-Medieninformatik

Autor:	Tim Cirksena
	Tom Sattler
Matr.-Nr.:	950976
	966056
E-Mail:	Tim.cirksena@hs-osnabrueck.de
	Tom.sattler@hs-osnabrueck.de
Dozent:	Prof. Dr. Rainer Roosmann

Abgabedatum: 27.02.2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Source-Code-Verzeichnis

1 Einleitung

Die gegebene Aufgabe dieser Projektarbeit ist der Entwurf, die Entwicklung und das Umsetzten einer mobilen Anwendung mit Java für Android unter Verwendung von Android Studio. Das Thema dieser Projektarbeit ist eine mobile Anwendung, die es ermöglichen soll, dem Benutzer das Tracken seiner Fitnessstudiobesuche zu vereinfachen und durch intuitive Handhabung möglichst schnell an Information, über bisherige Ergebnisse, zu gelangen. Zu beachten ist außerdem das die App mehrere Kriterien erfüllen muss. Diese Kriterien sollen die Qualität der App sicherstellen, und es ermöglichen eine besonders hochwertige User Experience zu gewährleisten.

1.1 Ziel der Ausarbeitung

In dieser Ausarbeitung wird tiefgehend auf den Stand der Technologie für die Entwicklung mobiler Anwendung von Android eingegangen. Es werden Themen vermittelt, die die Auswahlmöglichkeiten der Architektur solcher Anwendungen erklären. Außerdem wird auf Themen wie Testing in Android, das sehr umfangreiche Android Jetpack und der Lebenszyklus der Aktivitäten in Android Apps, eingegangen. Das Ziel ist es dem Leser einen Einblick in die Entwicklung von mobiler Anwendung zugeben. Es soll die Grundlagen vermitteln und tiefergehende Fragen beantworten. Der/Die Leser/in soll verstehen warum das Entwickeln von Apps mit Android Studio eine gute Wahl ist, und womit er/sie sich beschäftigen muss, um Probleme vorab zu beseitigen.

1.2 Aufbau der Projektarbeit

Die Projektarbeit ist in folgende Themen unterteilt:

- Die Einleitung
- Der Aufbau
- Grundlegende Technologie
- Die Anwendung
- Fazit

Durch diese Strukturierung soll sich ein roter Pfaden durch das Projekt ziehen und ermöglichen den Zusammenhang der Themen verständlich zu machen.

2 Aufbau des Projektes, Projektorganisation

Um den Umfang, den Inhalt und den Aufwand des Projektes richtig einschätzen zu können haben wir bereits vorher einen groben Strukturplan erstellt. Dieser Strukturplan ist anhand einer Ideensammlung entstanden, diese haben wir schon vor dem wir angefangen sind gemacht. Dafür haben wir über Miro (ein online Whiteboard) alle möglichen Ideen zusammengefasst und miteinander verbunden. So konnten wir bereits in der anfänglichen Planungsphase bestimmen welche Funktionen viel Zeit kosten werden und welche eher schnell zu bearbeiten sind. Außerdem konnten wir uns so Zeit sparen, indem wir Funktionen, die sehr schwer umsetzbar wären, direkt ausschließen konnten. Abgesehen davon haben wir kontinuierlich Notizen zur notwendigen Infrastruktur bzw. Darstellungsoptionen an der Benutzeroberfläche notiert. Zum Beispiel:

- Darstellung von Übungen: Zugang zur Datenbank.
- Über das Menü der Übungen weitere Übungen hinzufügen: „Einfachheit bei Benutzung der mobilen Anwendung“.
- Im Kalender Tage farblich unterscheiden: „Bessere User Experience“.
- Von jeder Seite der mobilen Anwendung zurück zur Vor-Seite gelangen können.
- Übungen variieren voneinander stark: Viele Auswahl Möglichkeiten bereitstellen und Eingabe intuitiv machen.

Anhand dieser Stichpunkte konnten wir uns, zu jeder Zeit, wichtige Funktionen vor Augen halten und diese dann einfach aufteilen untereinander.

2.1 Vorschlag

Der Vorschlag wurde durch Brainstorming erarbeitet, wir haben vier verschiedene Kategorien aufgeschrieben und damit unsere Ideen eingegliedert. Im Rahmen des Projektes und der Gewichtslegung der Aufgaben haben wir hier sehr subjektiv entschieden. Zu den Kategorien gehören persönliche Präferenzen, Nützlichkeit, Notwendigkeit und Einzigartigkeit. Durch Visualisierung dieser Punkte konnten wir uns direkt sicher sein, dass das Projekt den richtigen Weg einschlagen wird.

2.2 Projektplan Strukturierung

Diesen Projektplan haben wir in vier verschiedene Hauptkategorien unterteilt, hauptsächlich dafür, um die Planung zu optimieren.

- Vorbereitende Zeit

  (Erfassen von vorhanden IT-Strukturen und deren vergleich)

- Datenbank

  (Aufbau der Datenbank, Eingabe und Struktur)

- Programmierung der Funktionen und Anpassung der Benutzeroberfläche

  (Normalerweise in zwei Kategorien unterteilt, durch Android Studio jedoch einfach zu verbinden)

- Testen der Anwendung

  (Testen, Präsentation und Nachbereiten)

2.3 Version Control durch GitHub

Bei der Entwicklung der App wurde als Version Control System GitHub verwendet. GitHub bot uns die Möglichkeit unseren Source-Code effizient miteinander zu teilen. Das Version Control System hat dafür gesorgt, dass wir asynchron am Projekt arbeiten und trotzdem die neuesten Fortschritte des Projektpartners, in unseren Source-Code einbinden konnten. Wir haben mit 3 Branches gearbeitet: Der main Branch und jeweils einer Branch pro Person. Die main Branch wurde genutzt, um den bereits funktionierenden Source-Code der beiden anderen Branches zu vereinen. Der individuelle Branch wurde genutzt, wenn an Implementierungen gearbeiteten wurde, die mehr Zeit in Anspruch nahmen und das Abspielen der App während der Entwicklung verhinderten. So konnten wir, wenn es nötig war, immer eine ältere, funktionierende Version des Source-Codes einbinden und wurden nicht von neuem Source-Code des Partners beeinflusst, bis dieser funktionstüchtig war.

3 Darstellung der Grundlagen

Die Programmierung der mobilen Anwendung wird mittels des Integrated Development Environment (IDE) Android Studio umgesetzt. In der Programmiersprache Java. Android Studio ist die offizielle IDE für die Entwicklung von Android Apps und basiert auf IntelliJ IDEA. Die Implementierung der Datenbank erfolgt über das Android Jetpack, welches die Room Datenbank Library beinhaltet. Room ist eine persistente Library, die einen abstrakten Layer über SQLite zur Verfügung stellt. Zur Bearbeitung der Benutzeroberflächen wird der interne Layout Editor von Android Studio benutzt.

3.1 Erweiterte Darstellung und Architekturmuster

Sobald es dazu kommt, dass Entwickler an einer echten mobilen Anwendung arbeiten, die dynamischer Natur ist und somit dauernd erweitert wird, um die Anforderungen des Benutzers gerecht zu werden, muss der Code dementsprechend strukturiert werden das eine Anpassung ohne große Veränderung möglich ist. Für modulares Design (getrennte Codeteile) bieten sich besonders Architekturmuster an. Die meistverwendeten Architekturmuster sind:

- MVC (Model-View-Controller)
- MVP (Model-View-Presenter)
- MVVM (Model-View-ViewModel)

Die Hauptidee dieser Architekturmuster ist die Organisation des Projektes, sodass der ganze Code beim Unit-Testing abgedeckt ist. Des Weiteren ist es sehr hilfreich in der Wartung der Software und des Projektes, Funktionen hinzuzufügen und zu entfernen. Entwickler können hiermit verschiedene Logikteile gesondert im Auge behalten und bearbeiten.

3.1.1 Model-View-Controller

Das Model-View-Controller-Muster ist die älteste Android-App-Architektur, die einfach vorschlägt, den Code in drei verschiedene Schichten aufzuteilen und somit zu strukturieren.

- **Model**: Die Model-Schicht dient zur Speicherung von Daten. Sie ist für die Handhabung der Domain-Logik (Businessrules) und die Kommunikation mit der Datenbank zuständig.

- **View**: User Interface-Schicht. Diese Schicht ist zuständig für die Visualisierung der Daten, die in der Model-Schicht gelagert sind.
- **Controller**: Schicht die die Hauptlogik beinhaltet. Hier wird auf das reagiert was der Benutzer der mobilen Anwendung macht und die Model-Schicht daraufhin geupdated und informiert.

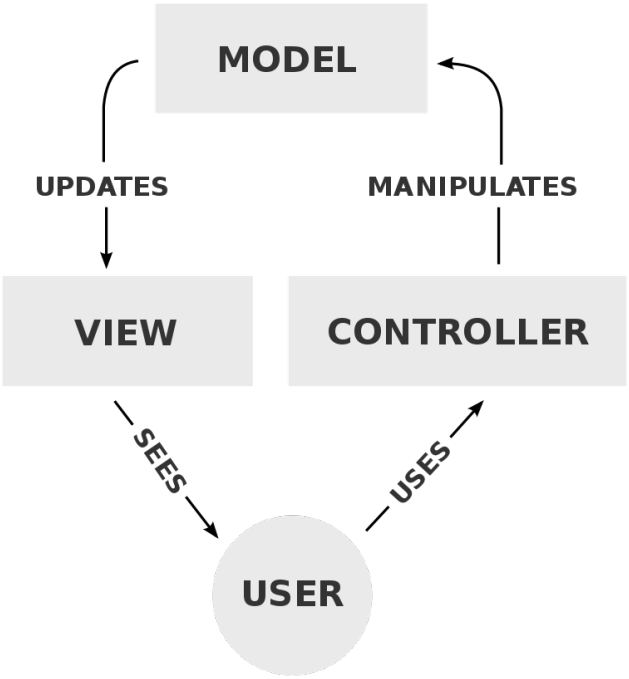


Abbildung 1: M-V-C als Graphik

Im MVC-Muster werden die Ansicht-Schicht und die Model-Schicht von der Art des Modells bestimmt. Die Anwendungsdaten werden von dem Controller aktualisiert und diese weitergeleitet an die View, diese ruft die Daten ab. In dieser Architektur kann die Logik unabhängig von der Benutzeroberfläche getestet werden, da sie getrennt sind. Es gibt mehrere mögliche Ansätze, um das Muster anzuwenden, eine davon ist es die Aktivitäten und Fragmente als Controller fungieren zu lassen. Hier sind sie verantwortlich für Datenverarbeitung und Aktualisierung der View. Sofern die View das Single-Responsibility-Prinzip einhaltet, besteht ihre Aufgabe darin, den Controller zu aktualisieren, sobald es zu Benutzerereignissen kommt. Die View zeigt lediglich Daten aus dem Model an, die Logik bleibt hier im Backend und wird nicht angezeigt.

Vorteile von dem MVC-Muster:

- Erhöhte Testbarkeit des Codes und weitreichende Erleichterung in Implementierung neuer Funktionen, da es die Trennung im hohen Maße unterstützt.

- Komponentenweisetest des Models und des Controllers sind leicht realisierbar, da sie keine Android-Klassen verwenden.

- Die View kann mittels UI-Tests getestet werden, sofern die View das Single-Responsibility-Prinzip einhält.

Nachteile:

- Der Zugriff von mehreren Views auf ein Model kann schnell zu einer sehr hohen Komplexität führen und dadurch fehlerhafte Funktionen bzw. Stellen des Codes schwer ausfindig machen lassen.

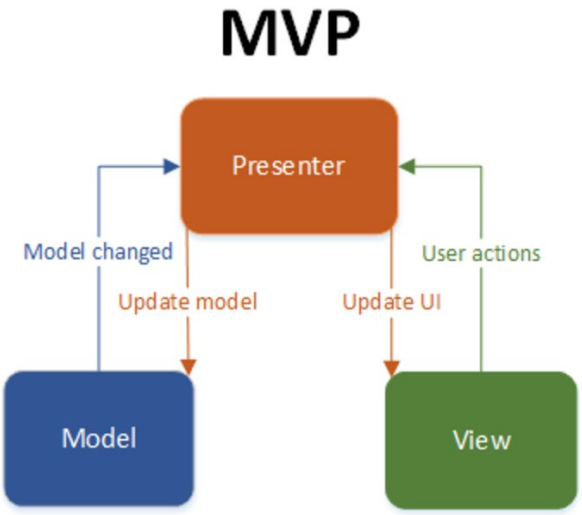
3.1.2 Model-View-Presenter

Das MVP-Muster ist die zweite große und bekannte Android-App-Architektur. Dieses Muster ist weithin angesehen und akzeptiert und wird weitreichend von Entwicklern empfohlen.

- **Model**: Dient wie in dem MVC-Muster zum Speichern der Daten. Außerdem auch für Handhabung der Domain-Logik und Kommunikation mit der Datenbank.

- **View**: Außerdem wie in dem MVC-Muster. Ist die sogenannte User Interface-Schicht, visualisiert Daten und verfolgt die Aktionen, die der Benutzer tätigt, um anschließend den Presenter (Moderator) zu benachrichtigen.

- **Presenter** (Moderator): Die Daten werden aus dem Modell aufgerufen und verwendet die UI-Logik an, um zu entscheiden, welche Daten angezeigt werden müssen. Der Status der View wird verwaltet und führt Aktionen von dem Benutzer aus die mittels Eingabebenachrichtigung erfolgt sind.



In diesem Muster ist die View und der Presenter eng miteinander verbunden und referenzieren aufeinander. Über eine Contract-Schnittstellenklasse wird die Beziehung zwischen View und Presenter definiert, außerdem dient es der Lesbarkeit und Verständlichkeit des Codes.

Vorteile:

- Einfache Wartung und Unit-Test, aufgrund von der Teilung der einzelnen Schichten
- Keine konzeptionelle Beziehung in Android-Komponenten

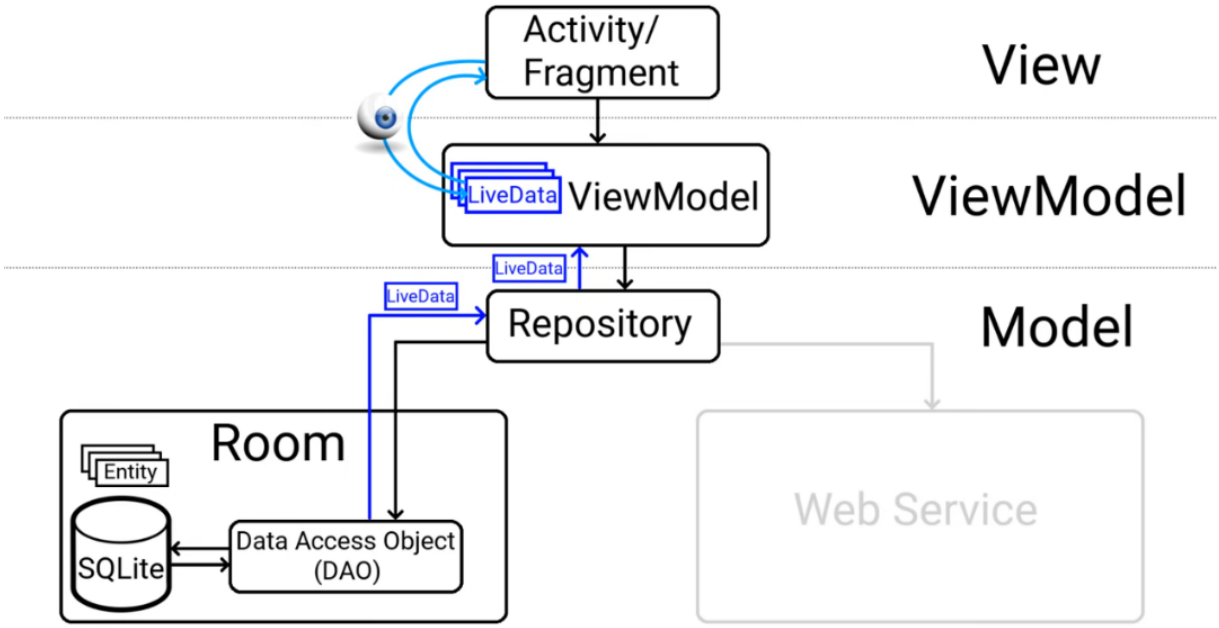
Nachteile:

- Durch Vernachlässigung des Single-Responsibility-Prinzips kann es dazu kommen das der Presenter zu einer sogenannten all-knowing-class wird. [2]

3.1.3 Model-View-ViewModel

Das MVVM-Muster wurde bei der Veröffentlichung der Architekturen von dem Android-Team empfohlen. Es wird sehr häufig in Bereichen der Web-Grafik und Präsentation-Software benutzt. Grundlegend verfolgt dieses Muster das Prinzip die zentrale Business-Logik der Anwendung von Datenpräsentationslogik zu trennen.

- **Model**: Diese Schicht ist für die Abstraktion der Datenquellen zuständig. Das Model und ViewModel arbeiten zusammen, um Daten abzusichern und aufzurufen.
- **View**: Hier wird der Zweck verfolgt das die View-Schicht die ViewModel-Schicht über Aktionen des Benutzers informiert.
- **ViewModel**: Dient als Bindeglied zwischen dem Model und der View. Und legt die Datenströme offen die für die View wichtig sind.



Das MVVM-Muster hat einige parallelen zu dem MVP-Muster. Die ViewModel-Schicht ersetzt die Presenter-Schicht. Jedoch wurden die Nachteile, die das MVP-Muster aufweisen mit dem MVVM-Muster behoben und auf folgende Weise gelöst:

1. Das ViewModel erhält keine Referenzen auf die View.
2. N zu 1 Beziehungen bestehen zwischen View und ViewModel.
3. Keine auslösenden Methoden zum Aktualisieren der View.

3.1.4 Android-Architekturmuster Fazit

Grundlegend ist zu sagen das es Sinnvoll ist für eine mobile Anwendung ein passendes Architekturmuster auszuwählen. Es ermöglicht künftige Änderung

des Codes und verleiht der Anwendung ein modulares Design, welches eine qualitativ hochwertige Prüfung und Wartung ermöglichen. Nachteile, die mit einer Architektur einhergehen sind meistens zeitaufwendiger und disziplinarischer Natur. So muss das Entwicklerteam streng darauf achten unangebrachte Änderungen zu vermeiden, da dies die Integrität der Architektur schaden kann. [3]

3.2 Testing in Android

Unit Testing soll versichern das der Entwickler qualitativ hochwertigen und fehlerfreien Code schreibt. Beim Durchlaufen von Testen können viele Informationen gewonnen werden. Die Korrektheit der Funktionen der mobilen Anwendung kann verifiziert- und somit eine fehlerfreie Publizierung gewährleistet werden. Unit Tests können unterschiedlich aussehen, es kann manuell, von dem Entwickler, durch die App navigiert- und jeder möglichen Variante der Benutzung eingeschlagen werden. Damit wird versucht versteckte Fehler zu finden. Der Nachteil vom manuellen Testen ist, dass es schlecht skalierbar ist. Die Automatisierung solcher Test kann mit Tools, die in Android Studio enthalten sind, durchgeführt werden. Die Vorteile sind das es schneller ist, einfacher zu wiederholen ist und grundlegend mehr umsetzbares Feedback liefert. Beispielsweise gibt es je nach Fach unterschiedliche Arten von Tests.

- Functional Testing: Macht die App das, was sie machen soll?
- Performance Testing: Erledigt die App Aufgaben schnell und effizient?
- Accessibility Testing: Ist die App zugänglich?
- Compatibility Testing: Funktioniert die App auf allen Geräten und API-Stufen?

Test können auch in ihrem Umfang aufgeteilt werden, da es stark abhängt in welcher Größe oder grad der Isolationen diese stattfinden.

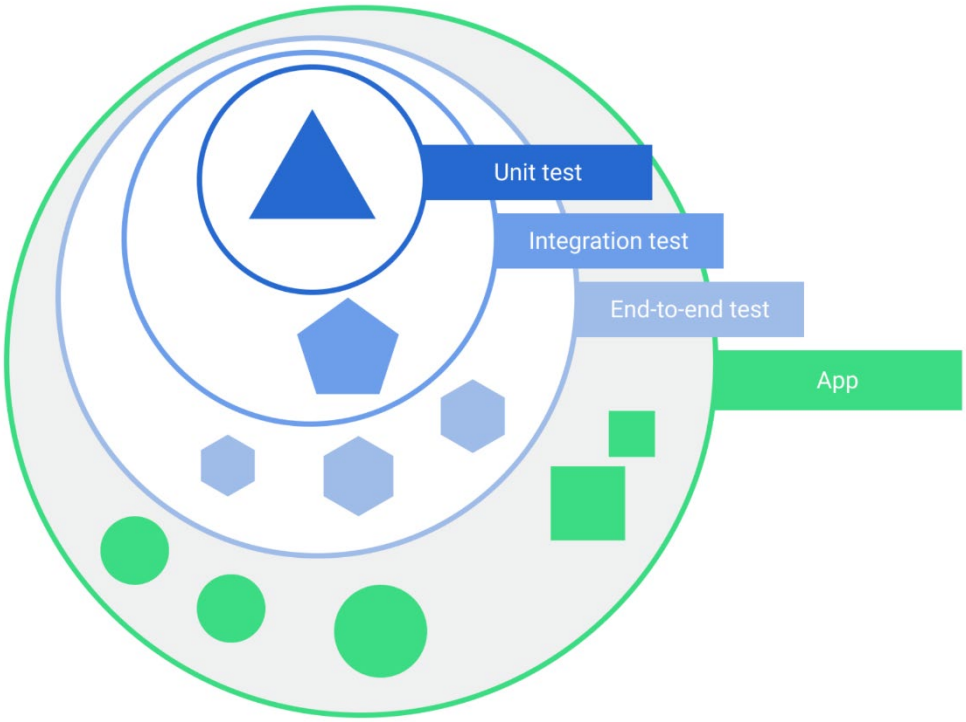


Abbildung 4: Testumfang

- Unit Tests beziehen sich nur auf sehr kleine Abschnitte der App und verifizieren somit eine Methode oder Klasse.
- End-to-end Tests, auch big tests genannt, verifizieren große Abschnitte einer App, wie einen Screen.
- Medium Tests sind zwei oder mehr Unit Test auf einmal und bilden das Mittelmaß.

3.2.1 Unterschied Local Test und Instrumented Test

Instrumented Tests laufen auf einem Android Gerät, entweder emuliert oder physikalisch. Die App wird auf das Gerät geladen und anschließend damit interagiert. Instrumented Tests sind für gewöhnlich UI Tests.

Local Tests werden auf einer Entwicklungsmaschine oder auf einem Server ausgeführt, deshalb nennt man sie auch Host-Side Test. Diese Tests sind meistens klein und schnell, isolieren ein Stelle.

3.3 Android Jetpack

Android Jetpack ist eine Sammlung von Bibliotheken und Werkzeugen. Diese Sammlung soll dabei Entwicklern helfen mobile Anwendungen mit Android leichter so erstellen. Es soll mithilfe von vorgegebenen Best Practices, komplexe Vorgehensweisen und Funktionen mit geringem Aufwand im Programmcode umsetzbar sein, indem hierfür weniger Boilerplate-Code benötigt wird. Bewährte Entwicklungsansätze, wie Data Binding, werden nun im Android Jetpack zusammengefasst und stätig durch weitere Konzepte ergänzt. Außerdem dient der Name dem Zweck der Vermarktung, das Jetpack soll hier als Metapher der Beschleunigten App Entwicklung dienen. Android Jetpack lässt sich in vier Kategorien einteilen:

1. Fundation: Das Fundament der Entwicklung von mobilen Anwendungen wird durch Werkzeuge zu der Sicherung der Abwärtskompatibilität gelegt.

2. Architecture: Hier enthalten sind die wichtigsten Architekturkomponenten wie das ViewModel, LiveData und Room, diese dienen der Strukturierung von modernen Android-Apps. Neuer Bestandteil ist die Navigationskomponente, die es dem Entwickler ermöglicht durch einfache Buttons komplexe Navigationsmuster zu erstellen.

3. Behavior: Diese Kategorie enthält wichtige Bibliotheken, die das Zusammenspiel unterschiedlicher Android-Dienste ermöglicht, regelt und vereinfacht. Ein Dienst ist das Benachrichtigen System.

4. UI: Hier werden sämtliche Layouts und weitere Ansätze zur Gestaltung erfasst. Alle dienen der Entwicklung attraktiver Android-Apps.

Abbildung 5: Android Jetpack

Android Jetpack besteht somit aus vielen verschiedenen Bibliotheken, die auf diese vier Hauptkategorien, aufteilt werden. Der modulare Aufbau von Jetpack dient dazu das jede dieser Bibliotheken einzeln genutzt werden kann. Ebenso können Projekte älterer Android-Versionen genutzt werden. [4]

3.4 Lokale Datensicherung mittels Room

Room ist ebenfalls Bestandteil von dem Android Jetpack. Es dient als Layer über die SQLite-Datenbank, vereinfacht Zugriffe auf diese und verhindert schwerwiegende Syntaxerrors im Datenbanksystem. Die SQLite-APIs sind zwar leistungsfähiger als Room, jedoch schwerer zu programmieren. Die Datenbank ist in drei Hauptkomponenten aufzuteilen

1. @Database

Dient als der Datenträger und Hauptzugangspunkt für unterliegende Verbindungen zu den persistierten, relationalen Daten der mobilen Anwendung.

2. @Entity

Eine Entität repräsentiert eine Tabelle in der Datenbank. Diese Klasse ist kommentiert mit einem *@Entity*. Alle Bestandteile dieser Entität müssen entweder public sein oder Getter und Setter Methoden besitzen. Jede Klasse muss einen Primary-Key besitzen, der diese Entität eindeutig identifiziert, mittels *autoGenerate* wird dieser Primary-Key automatisch erzeugt und inkrementiert. Meistens ist der Primary-Key einer Entität eine ID, die eine Zahl darstellt.

3. @DAO

DAO steht für Data-Access-Object, es beinhaltet alle Methoden, die dafür benötigt werden, um auf die Datenbanken zugreifen zu können. Es dient als Schnittstelle zwischen den Entitäten und der Datenbank. Mit Methoden wie @Insert, @Delete kann die mobile Anwendung mit den Entitäten interagieren. [5]

3.5 Activity-Lebenszyklus

Die Activity-Klasse ist eine sehr wichtige Komponente von Android Apps. Eine Activity stellt das Fenster dar, auf dieses kann die mobile Anwendung die Benutzeroberfläche zeichnen mit welchem wiederum der Benutzer interagieren kann. Es kann den ganzen Bildschirm füllen oder ein kleines Fenster sein, welches über anderen Activities schwebt. Eine Anwendung besteht aus vielen verschiedenen Activities die lose miteinander gekoppelt sind. Sie können sich untereinander aufrufen und somit öffnen, in dieser Zeit wird die aktuelle Activity gestoppt und die gestartete ausgeführt, im Hintergrund bleibt der Zustand der gestoppten Activity dem Android System erhalten. In der Art wie Activities gestartet und gehandhabt werden ist ein wesentlicher Bestandteil von Android Studio. Das Android System benutzt keine herkömmlich bekannte *main* Methode für das Starten der Activities, sondern mehrere spezielle Callback-Methoden, die die Activity stufenweise instanziieren. Diese Callback-Methoden werden in bestimmten Lebensphasen der instanziierten Activity aufgerufen. Aus diesem Grund werden die Callback-Methoden auch Lifecycle Callbacks genannt. Sofern eine Activity gestoppt wird, weil eine neue geöffnet wurde, informiert der Lifecycle Callback diese über ihre Zustandsänderung. Eine Activity kann in mehrere Zustände versetzt werden, für jeden Zustand gibt es in Android Studio eine Callback-Methode, die diesen bearbeitet. Bevor eine Zustandsänderung in Kraft tritt, müssen die notwendigen Arbeiten erledigt sein. Der Lifecycle einer Activity kann von mehreren Faktoren beeinflusst werden, dem eigenen Task, anderen Activities und dem Back Stack. Der Back Stack ist ein Stapel der genutzte Activities verwaltet. Eine Activity kann sich in folgenden Zuständen befinden: [6]

- Resumed-Zustand: Immer, wenn sich eine Activity im Fokus befindet und somit im Bildschirmvordergrund ist, ist sie im Resumed-Zustand. Hier

kann die App mit dem Benutzer interagieren. In diesem Zustand sollten unteranderem alle kritischen und rechenintensive Anwendungsprozesse erfolgen, das wären Animationen und Datenbankzugriffe. Es kommt erst zum Zustandswechsel, wenn der Fokus auf eine andere Activity gerichtet wird. Wenn somit eine andere Activity auf dem Bildschirmvordergrund angezeigt wird, wechselt die Activity in den Paused-Zustand, das Android System ruft die onPause Callback-Methode auf.

- Paused-Zustand: Die Ausführung kann durch bestimmte Ereignisse unterbrochen werden. Beispielsweise ist dies der Fall, wenn eine andere Activity in den Bildschirmvordergrund rückt oder auch wenn die ausgeführte Activity die erste überlagert. In dieser Situation wird die onPause Methode von dem Android System aufgerufen, die ausgeführte Activity wechselt dann von dem Resumed- in den Paused-Zustand. Der Paused-Zustand beendet die Activity nicht. Die Instanz der Activity wird weiterhin im Speicher gehalten und alle Informationen, die bislang bekannt waren, bleiben dem Fenster-Manager angefügt. Die pausierte Activity kann allerdings in Situationen in dem der Speicher sehr stark ausgelastet ist, durch das Android System beendet werden. Wenn die Activity wieder fortgesetzt werden soll, wird die onResumed-Methode aufgerufen, die Instanz der Activity ist noch unversehrt im Speicher und kann ohne Probleme geöffnet werden. In diesem Zustandswechsel müssen die Komponenten nicht erneut initialisiert werden, da diese gespeichert wurden.

- Stopped-Zustand: Wenn die Activity nicht mehr für den Benutzer sichtbar ist und somit vollständig verdeckt ist, wird das Zustand als Stopped oder als im Hintergrund betrachtet. Die Activities im Stopped-Zustand versuchen weiterhin, ihre Status- und Memberinformationen so lang wie möglich beizubehalten. Allerdings werden Activities die Stopped sind als die niedrigste Priorität der drei Zustände gesehen, daher beendet das Android System zuerst diese, um die nötigen Ressourcenanforderungen von Activities mit einer höheren Priorität zu erfüllen.

Festzuhalten ist, dass eine Activity die sich im Paused oder Stopped Zustand befindet jederzeit durch das Android System beendet werden kann. Entweder geschieht dies durch die finish-Methode oder durch das direkte Beenden des Prozesses der Activity. Falls diese wieder geöffnet wird, muss sie von ganz vorne erstellt werden.

3.6 Methoden des Activity-Lebenszyklus

Sobald eine Activity einen Zustandswechsel durchführt, wird diese über Callback-Methoden über die Zustandsänderung informiert. Alle Callback-Methoden können überschrieben werden und damit dem Entwickler helfen, das Programm auf entsprechende Zustandsänderungen richtig zu reagieren und zu bearbeiten. [7]

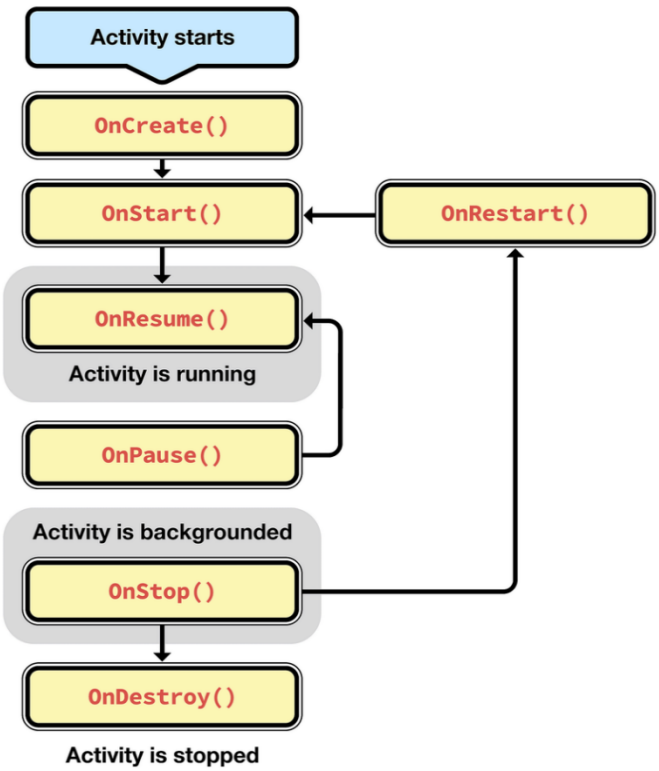


Abbildung 6: Methoden Activity-Lebenszyklus

- **onCreate Callback-Methode**

Wenn eine Activity gestartet und das erste Mal erzeugt wird, wird die onCreate Callback-Methode aufgerufen. Die Methode muss zwingend implementiert sein, um alle Startinitialisierungen auszuführen, die möglicherweise für die Activity erforderlich sind.

- Erstellen von Ansichten
- Initialisierung von Variablen
- Binden statischer Daten an Listen

Außerdem wird der Callback-Methode ein Bundle-Objekt übergeben, welches den vorherigen Zustand der Activity enthält, sofern sich dieser Zustand noch im Back Stack befindet. Im Anschluss von der onCreate Methode wird immer die onStart Methode aufgerufen.

- **onStart Callback-Methode**

Die onStart Methode wird aufgerufen, um die Activity darauf vorzubereiten in den Vordergrund der mobilen Anwendung zu rücken und interaktiv zu werden. In diesen Methoden werden die Komponenten initialisiert, die verantwortlich für das Verwalten der Benutzeroberfläche sind. Die Funktionen sollten schnell abgeschlossen werden können, da die Activity diese Methode auf dem Weg zum Resumed-Zustand aufruft und nicht lange darin verweilen sollte. Direkt, nachdem die onStart Methode beendet wurde, ruft das Android System die onResume Methode auf.

- **onResume Callback-Methode**

Wenn die Activity nun in den Vordergrund geöffnet wird, wird die onResume Methode aufgerufen. Die Activity ist nun interaktiv verwendbar für den Benutzer.

- **onStop**

Wenn sich die Activity nicht mehr in dem Bildschirmvordergrund befindet, wird die onStop Methode aufgerufen

- **onRestart**

Diese Methode wird aufgerufen so bald eine pausierte Activity in den Resume-Zustand wechseln soll.

- **onDestroy**

Die letzte Methode die vor der Zerstörung einer Activity aufgerufen wird. Entweder weil die Activity planmäßig durch finish beendet wird oder um für andere Prozesse Speicherplatz und somit Ressourcen zu schaffen.

Von allen Lifecycle Callback-Methoden muss nur zwingend die onCreate Methode implementiert werden. Alle anderen Methoden können optional, je nach

4 Anwendung der App

4.1 Funktionalität

Die von uns implementierte App bietet die Funktion Trainingseinheiten durch Übungen zu dokumentieren. Diese Übungen können von den bereits vorliegenden Übungen übernommen oder aber, falls nicht vorhanden, durch das Namen geben einer Übung, individuell hinzugefügt werden. In einer Übung werden die Werte Wiederholungen, Sätze, Schwierigkeit, Gewicht und das Datum gespeichert. Außerdem kann man eine Beschreibung und eine Vorlage hinzufügen. Durch das hinzufügen einer Vorlage, kann man auf die Übung durch den Reiter „Training aus Vorlage entnehmen“, alle bereits gespeicherten Übungen sehen, welche diese Vorlage zugewiesen bekommen. Auf dem oberen Teil der Startaktivität der App werden die Tage des aktuellen Kalendermonats gezeigt. Die Tage an denen trainiert wurde, werden in hellblau gekennzeichnet. Der aktuelle Tag hat einen roten Kreis um das „Kalenderkästchen“. Wenn man auf ein „Kalenderkästchen“ klickt, kann man alle Übungen sehen, die man an dem Tag hinzugefügt hat. Bei fälschlichem hinzufügen oder löschen einer Übung kann man wie bei allen Übungsauflistungen die Übung wieder löschen oder hinzufügen. Durch den unteren Teil der Startaktivität kann man einem Graphen entnehmen, an welchem Tag in diesem Monat man ein eher schweres oder eher leichtes Training hatte. Auf der x-Achse des Graphen sind die Tage von 1 bis zu dem letzten Tag des Monates dargestellt. Die y-Achse repräsentiert die Summe der Schwierigkeit der Übungen, die an diesem Tag verrichtet wurden. Der maximal dargestellte y-Wert ist auch der Wert des Tages, an dem die Summe der Schwierigkeit der Übungen am höchsten war. In der Statistik Aktivität kann man eine individualisierte Version des Graphen darstellen lassen. Die Statistik Aktivität bietet die Möglichkeit den Graphen auf der x-Achse in einen bestimmten Zeitraum innerhalb eines Jahres darzustellen. Auf der y-Achse werden nach Auswahl der Übung, entweder standardmäßig alle oder eine spezielle Übung repräsentiert.

Die App bietet also die Möglichkeit des Dokumentierens und Analysierens von verrichteten Übungen.

4.2 Navigation und Gliederung der Inhalte

4.2.1 Navigation zwischen den Aktivitäten

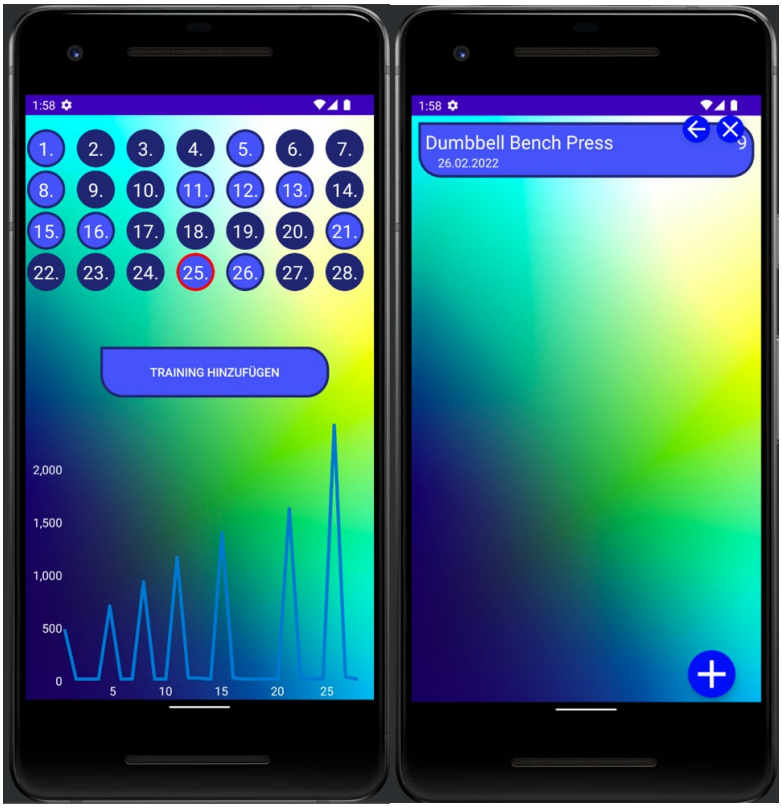


Abbildung 7:HomeView Abbildung 8: DayExerciseActivity

HomeActivity

Die HomeActivity stellt die Startaktivität dar. Auf der Home-Activity sind oben ein Kalender mit den monatsaktuellen Kalendertagen, in der Mitte ein Button „Training hinzufügen“ und unten eine dargestellte Statistik der bereits diesen Monat verrichteten Übungen. Durch eine touch-Ereignis auf einen der Tage im Kalender, wird die DayExerciseAktivität aufgerufen. Das „tappen“ auf den Training hinzufügen Button, öffnet dem Benutzer die StartTrainingAktivität. Wenn ein touch-Ereignis auf der Statistik passiert, öffnet sich die StatistikAktivität.

DayExerciseActivity

In der DayExerciseActivity werden in einer „Liste“ die verschiedenen an dem Kalendertag absolvierten Trainings angezeigt. Wenn man auf eines dieser Trainings „tapped“ wird man zu der AddEditExercise Aktivität der Übung weitergeleitet. Außerdem befindet sich oben ein weißer Pfeil nach links in einem „floating-Button“. Durch das Betätigen dieses Buttons kommt man zurück auf die HomeActivity.

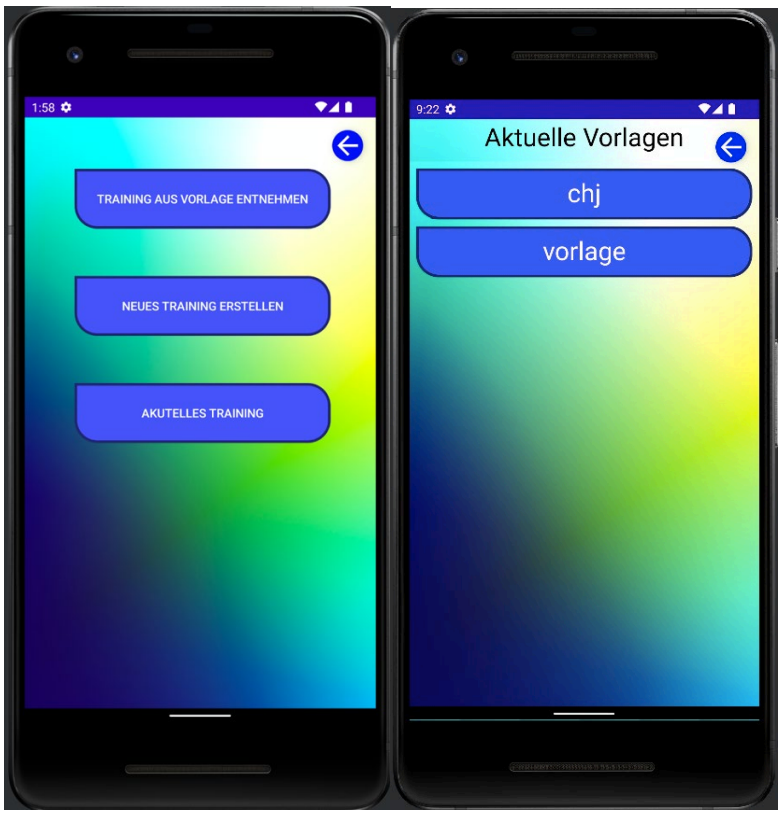


Abbildung 9: StartTrainingActivity

Abbildung 10: TemplateActivity

StartTrainingAktivität

Die drei Buttons der StartTraining Aktivität mit den Texten „Training aus Vorlage entnehmen“, „Neues Training hinzufügen“ und „Aktuelles Training“ führen zu je einer anderen Aktivität, mit der man ein Training auf einer anderen Art hinzufügen kann. Der Button, mit der Überschrift „Training aus Vorlage entnehmen“ führt zur Vorlage Aktivität. Durch den „Neues Training erstellen“ Button gelangt man an standardmäßige AddEditExercise Activity. Durch das „tappen“ auf „Aktuelles Training“, öffnet sich die ListExerciseActivity.

TemplateActivity

Die Vorlage Activity bietet zur Navigation einen „floating-Button“ mit einem Weißen Pfeil nach links in der rechten oberen Ecke der Aktivität. Der Button führt zurück zu der TrainingBeginnenActivity.

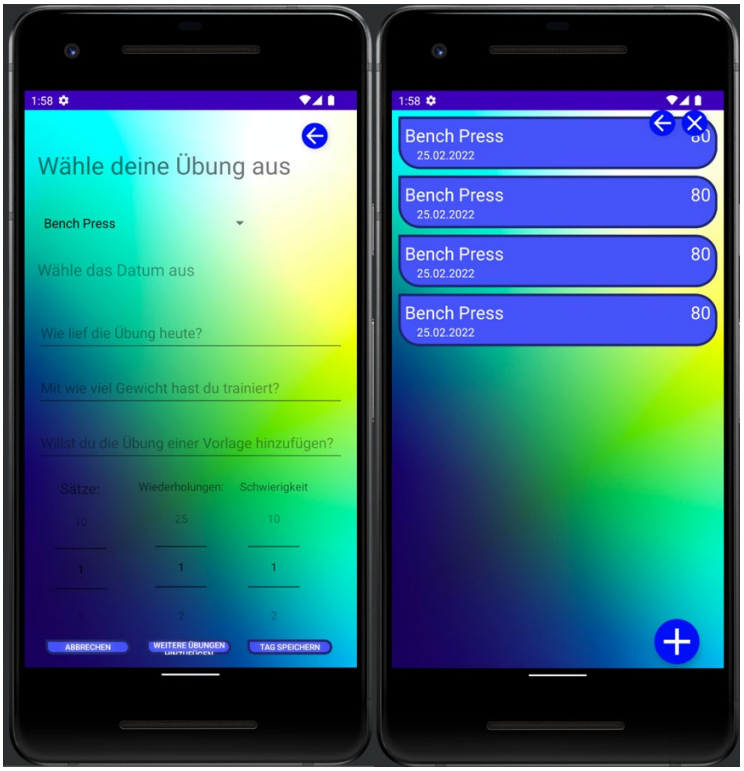


Abbildung 11: AddEditExerciseActivity Abbildung 12: ListExercisesActivity

AddEditActivity

Die AddEditActivity hat vier Buttons die alle als Möglichkeit zum Navigieren dienen. Drei dieser Button befinden sich unter den anderen Views im Bildschirm. Der Button, welcher unten links gelegen ist, hat die Überschrift „Abbrechen“. Unten in der Mitte ist ein Button mit dem Schriftzug „Weitere Übungen hinzufügen. Auf dem Button, der unten rechts liegt, steht „Tag Speichern“ in weißer Schrift. Wenn man den „Abbrechen“ Button betätigt, gelangt man zur ListExercisesActivity. Bei Auslösung des „Weitere Übung Hinzufügen“ Buttons gelangt man bei korrekter Eingabe der Auswahlfelder, zu einer leeren AddEditActivity. Der „Tag Speichern“ Button speichert den Tag und zeigt falls noch nicht gespeicherte, aber richtig ausgefüllte AddEditActivities vorhanden sind, diese an, damit sie auch noch gespeichert werden können. Nachdem die Übung gespeichert wurde, gelangt man zurück zur ListExercisesActivity. Wie die TemplateActivity bietet auch die AddEditActivity einen „floating-Button“ in der rechten oberen Ecke, der die ListExercisesActivity öffnet.

ListExercisesActivity

Die ListExercisesActivity bietet zur Navigation ebenso wie die Vorlage- und die AddEditActivity einen „floating-Button“ in der rechten oberen Ecke durch den man zurück zur StartTrainingActivity geführt wird.

4.2.2 Implementierung der Navigation durch Intent

```
Intent intent = new Intent( packageContext: this, MainActivity.class);
startActivity(intent);
```

Source-Code 1: Beispiel eines Intents

Die Möglichkeit während der Laufzeit die verschiedenen dargestellten Aktivitäten zu wechseln, geschieht durch Intent Objekte. Intent ist eine Klasse, die von der Klasse Object erbt und implementiert die Interfaces Parcelable und Cloneable. Ein Intent Objekt wird durch die „Anfangs-Aktivität“ und die „Ziel-Aktivität“, als Übergabeparameter im Konstruktor initialisiert. Durch die Methode „startActivity“, die den Intent als Übergabeparameter bekommt kann die Aktivität, wie im Intent definiert, gewechselt werden.

```
intent.putExtra(AddEditNoteActivity.EXTRA_TITLE, exercise.getName());
```

Source-Code 2: Beispiel eines putExtra Aufrufs

Außerdem bieten Intent Objekte die Möglichkeit „Extras“ von der einen zur anderen Aktivität zu liefern. Durch die „putExtra“ Methode des Intent Objekts, kann man einem Intent ein Extra geben. Dieses Extra braucht, um von der Ziel-Aktivität eingebunden zu werden einen Identifikationsstring. Diesen Identifikationsstring speichern wir als öffentlich statisch und finales String Objekt, wodurch man den nicht veränderbaren Identifikator auch außerhalb der Aktivitätsklasse aufrufen kann.

```
Intent data = getIntent();
String uebung = data.getStringExtra(AddEditNoteActivity.EXTRA_TITLE);
```

Source-Code 3: Beispiel eines getExtra Aufrufs

Die Extras eines Intents können aufgerufen werden in dem man ein neues Intent Objekt erstellt und diesem Intent Objekt mit getIntent (den Intent zuweist der die Activity mit startActivity geöffnet hat. Durch die Methode getExtra (In diesem Fall getStringExtra, da es sich einen String handelt) wird das Extra mit

dem Identifikator der Anfangs-Aktivität aufgerufen und gibt den dort eingetragenen Übergabeparameter vom Typ String zurück.

```
Intent intent = new Intent( packageContext: DayExercisesActivity.this, AddEditNoteActivity.class);
someActivityResultLauncher.launch(intent);
```

Source-Code 4: Beispiel someActivityLauncher

Wenn eine Activity für ein bestimmtes Ereignis geöffnet wird (Zum Beispiel das Ermitteln von Werten), wurde der Activity Result Launcher genutzt. Nachdem ein Ergebnis der aufgerufenen Activity ermittelt wurde, gibt man in der Methode setResult() einen Identifikator für den eingetretenen Fall und den Intent, mit dem die Acitivity für ein Ergebnis aufgerufenen wurde an. Mit diesem Identifikator kann die ursprüngliche „Anfangs“-Activity die verschiedenen Fälle, anhand des Identifikators vom Typ Integer unterscheiden und unterschiedlich mit den Fällen umgehen.

```
Intent data = result.getData();
if (result.getResultCode() == 77) {
```

Source-Code 5: Beispiel ResultCode Identifikator Bedingung

Das Einbinden des Intent wird nun durch die result.getData() Methode und nicht mehr durch getIntent() Methode realisiert, da man den Identifikator (Result Code) für die verschiedenen Fälle benötigt.

5 Implementierung der Datenpersistierung

Die Datenpersistierung wurde durch Android Room mit dem Model View Viewmodel Muster realisiert. Außerdem wurde die Database durch eine Repository Klasse ergänzt, welche für eine Skalierbarkeit der Database sorgt. Durch die Repository Klasse könnte beispielsweise eine Remote Database über eine API eingebunden werden.

5.1 Exercise

Die Java Klasse Exercise ist als Entität in dem SQLite Table „exercise_table“ implementiert worden. Der PrimaryKey integer id wird durch SQLite automatisch generiert. Die Klasse hält die Instanzvariablen: Name, Datum, Beschreibung, Schwierigkeit, Wiederholungen, Sätze, Gewicht, Vorlage, Position, Tag, Monat und Jahr. Alle Instanzvariablen sind mit der Sichtbarkeit private realisiert worden. Für alle Instanzvariablen wurden Setter und Getter implementiert. Die Instanzvariable Vorlage wird nicht im Konstruktor initialisiert, da eine Übung standardmäßig keine Vorlage haben soll.

5.2 Dao

Das Dao Interface sorgt, wie bereits in 3.4 beschrieben, für die Schnittstelle zwischen Datenbank und Entität. Der Zugriff geschieht durch sogenannte Queries.

```
@Query("SELECT * FROM exersice_table WHERE monat=:monat AND tag=:tag AND jahr=:jahr")
LiveData<List<Exercise>> getExercisesForDay(int tag, int monat, int jahr);

@Query("DELETE FROM exersice_table WHERE datum = :datum")
void deleteExercisesOfDay(String datum);

@Query("SELECT * FROM exersice_table WHERE monat >= :monat AND tag >= :tag AND jahr >= :jahr ORDER BY tag ASC")
LiveData<List<Exercise>> getAllExercisesLaterThan(int tag, int monat, int jahr);

@Query("SELECT * FROM exersice_table WHERE monat >= :monatMin AND tag >= :tagMin AND jahr >= :jahrMin AND " +
        "tag<= :tagMax AND monat<= :monatMax AND jahr<= :jahrMax  ORDER BY jahr,monat,tag ASC")
LiveData<List<Exercise>> getAllExercisesBetweenDates(int tagMin, int monatMin, int jahrMin, int tagMax,
                                                     int monatMax, int jahrMax);

@Query("SELECT * FROM exersice_table WHERE monat >= :monatMin AND tag >= :tagMin AND jahr >= :jahrMin AND " +
        "tag<= :tagMax AND monat<= :monatMax AND jahr<= :jahrMax  AND name = :name ORDER BY tag ASC")
LiveData<List<Exercise>> getAllExercisesBetween(int tagMin, int monatMin, int jahrMin,
                                                int tagMax, int monatMax, int jahrMax, String name);

@Query("SELECT * FROM exersice_table WHERE vorlage LIKE :vorlage GROUP BY name")
LiveData<List<Exercise>> getAllExercisesForVorlage(String vorlage);

@Query("SELECT * FROM exersice_table WHERE vorlage IS NOT NULL GROUP BY vorlage")
LiveData<List<Exercise>> getAllExercisesWithVorlage();
```

SOURCE-CODE 1: DAO QUERIES

Source-Code 6: Dao Queries

Neben den Standard Queries Insert, Update und Delete wurden noch zehn andere Queries eingebunden. Bis auf die Queries die Exercise Objekte löschen geben alle ein LiveData<List<Exercise>> Objekt zurück.

Der getExerciseForDay Query liefert alle Exercise Objekte mit dem übergebenen String Datum. DeleteExercisesOfDay löscht alle Exercises an dem übergebenen String Datum.

Der Query getAllExercisesLaterThan mit den Übergabeparametern: Integer Tag, Integer Monat und Integer Jahr gibt die Exercises zurück deren Tag, Monat und Jahr Instanzvariable größer gleich dem der Übergabeparameter sind. Die Exercise Objekte werden nach dem Tag aufsteigend sortiert zurückgegeben.

GetAllExercisesBetweenDates bekommt minimale und maximale Tage, Monate und Jahre Integer Werte als Übergabeparameter. Der Query liefert alle Exercise Objekte, welche größer gleich den Minima und kleiner gleich den Maxima sind. Die Exercise Objekte werden ebenfalls nach Tag, Monat und Jahr aufsteigend sortiert zurückgegeben.

Die implementierte Methode getAllExercisesBetween hat dieselbe Funktionalität wie die GetAllExerciseBetweenDates Methode mit dem Unterschied, dass hier zusätzlich nur Exercise Objekte zurückgegeben werden, bei welchen der String Name dem übergebenen String gleicht.

getAllExercises bekommt als Übergabeparameter einen String. Die Exercise Objekte, die in der LiveData Liste zurückgegeben werden, müssen ein String Objekt Vorlage tragen, was dem Übergabeparameter entspricht.

Der Query getAllExercisesWithVorlage gibt alle in der Datenbank gespeicherten Exercise Objekte zurück dessen String Vorlage ungleich null sind. [8] [9]

5.3 Database

Die implementierte Database Klasse ExercisesDatabase erweitert die abstrakte Klasse RoomDatabase. Außerdem ist in ihr die innere Klasse PopulateDbAsyncTask implementiert, welche die abstrakte Klasse AsyncTask erweitert. Die eingebundene Entität ist die Exercise. ExercisesDatabase hält folgende Instanzvariablen:

- Statische ExerciseDatabase instance mit privater Sichtbarkeit
- Abstraktes ExerciseDao exerciseDao mit öffentlicher Sichtbarkeit

Die implementierte Methode getInstance bekommt als Übergabeparameter den aktuellen Kontext initialisiert die Instanzvariabel instance durch den Room Databasebuilder. Dieser bekommt die Klasse selbst und den Namen der Database und deren Entität übergeben. Außerdem wird die fallbackToDestructiveMigration Methode aufgerufen, welche der instance Rechte gibt, sich selbst zu löschen und neu zu entwerfen, um für ältere Versionen kompatibel zu sein. Es wird noch ein Callback hinzugefügt und die Database wird gebaut und der instance Variable zugewiesen, die auch zurückgegeben wird.

Der roomCallBack wird ebenfalls implementiert, damit dieser ein neues Objekt der Inneren Klasse konstruieren kann.

Die Innere Klasse PopulateAsyncTask hält als Instanzvariabel das ExerciseDao Objekt. In dem Konstruktor wird ihm dieses als Übergabeparameter zugewiesen. Außerdem wird die Methode doInBackground implementiert, mit welcher Operationen im Hintergrund auf anderen Threats durchgeführt werden können. [8] [10]

5.4 Repository

Wie im Einleitungssatz bereits beschrieben, dient das Repository der Einbindung von mehreren Databases und somit der Skalierbarkeit der App. Die Repository Klasse Exercise hält drei Instanzvariablen vom Typ ExerciseDao(einmal) und LiveDate<List<Exercise>>(zweimal). Die Methoden des Repositorys lassen sich in 4 Typen einteilen:

- Konstruktor

- Methoden, welche mit Hintergrundthreads ausgeführt werden
- Methoden die implementierte Query Methoden des Daos zurückgeben
- Methoden, welche Instanzvariablen zurückgeben

Der Konstruktor bekommt Application Objekt Übergabeparameter und initialisiert mit diesem durch die Database getInstance Methode die Instanz Database für den Kontext. Außerdem wird das Dao durch die Database Methode der Instanzvariable exerciseDao zugewiesen. Durch das Dao werden jetzt die Instanzvariablen vom Typ LiveData<List<Exercise>> initialisiert.

Die Methoden getAllExercises und getAllExercisesWithVorlage geben die gehaltenen Instanzvariablen der Listen zurück.

```
private static class InsertExerciseAsyncTask extends AsyncTask<Exercise, Void, Void> {
    private ExerciseDao exerciseDao;

    private InsertExerciseAsyncTask(ExerciseDao exerciseDao) { this.exerciseDao = exerciseDao; }

    @Override
    protected Void doInBackground(Exercise... exercises) {
        exerciseDao.insert(exercises[0]);
        return null;
    }
}
```

Source-Code 7: InsertExerciseAsyncTask Klasse

Für die während der Laufzeit auf Hintergrundthreads ausführbaren Methoden wurden neue Klasse geschrieben, welche die AsyncTask Klasse erweitern. Sie halten eine Instanzvariable vom Typ exerciseDao und führen die gewünschte Methode des Daos in der doInBackground Methode aus. Im Konstruktor werden ihr als Übergabeparameter das Dao übergeben. Die Methoden arbeiten nur in der Database, also fügen entweder Elemente hinzu oder löschen sie.

Die Methoden, welche die implementierten Dao Methoden auf dem Hauptthread zurückgeben. Rufen die Query Methode der Instanzvariable ExerciseDao auf und geben an sie die Übergabeparameter weiter und haben denselben Rückgabewert, wie dessen Methode. [11] [8]

5.5 Viewmodel

Die Klasse ExerciseViewmodel erweitert die Klasse AndroidViewModel. Sie hält die Instanzvariablen zweier LiveData<List<Exercise>> Objekte und des Repositorys. Im Konstruktor wird durch den Übergabeparameter Application des super Konstruktors aufgerufen und mit diesem ein neues ExerciseRepository Objekt konstruiert. Durch die Instanvariable ExerciseRepository können nun alle Methoden dessen implementiert werden und die Instanzvariablen vom Typ LiveData<List<Exercise>> initialisiert werden.

6 Methoden zur Darstellung

6.1 GraphView

6.1.1 Vorstellung und unsere Implementation von GraphView

GraphView ist eine Android Library von Jonas Gehring die die Visualisierung von Listen in Form von Graphen realisiert. In der FiTracker App wurde die GraphView in der Statistik-Activity und der Home-Activity implementiert.

```
DataPoint[] s = new DataPoint[DataTimeConverter.getAmountDay()];
LineGraphSeries<DataPoint> series = new LineGraphSeries<>(s);
GraphViewHelper.setUpGraphView(graphView, DataTimeConverter.getAmountDay(), max);
GraphViewHelper.updateGraphView(graphView, series);
```

Source-Code 8: Implementierung eines Graphen mit der Hilfsklasse GraphViewHelper

Die GraphView Library arbeitet mit verschiedenen Datenstrukturen, von denen wir DataPoint und LineGraphSeries in unserem App-Projekt implementiert haben. Die GraphView kann nur Listen, die den Typ DataPoint tragen darstellen. Ein DataPoint hat einen x- und einen y-Wert vom Typ Double. Die LineGraphSeries erbt von der abstrakten Datenstruktur Klasse BaseSeries und implementiert das DataPoint Interface. Die abstrakte Klasse BaseSeries übernimmt in der Library die Funktion Datenstruktur und realisiert die nötigen Methoden, um mit der Datenstruktur zu interagieren. Die LineGraphSeries hat die nötigen Methoden implementiert um die darin enthaltenen DataPoint als Linie auf dem Graphen zu visualisieren. Der Konstruktor der LineGraphSeries wird ruft über den super Konstruktor den Konstruktor von BaseSeries auf und übergibt diesem als Übergabeparameter einen Array von DataPoints. Diese DataPoints werden in der FiTracker App mit zwei Arten von Übergabeparametern initialisiert. Auf der HomeActivity werden die DataPoints mit normalen Integer Werten für x und y initialisiert. In der StatistikActivity werden als x- Koordinaten Übergabeparameter vom Typ Date übergeben. Die DataPoints, welche Dates als x-Werte tragen konvertieren diesen Konstruktor durch die Funktion getTime(), welche von Java.util.Date implementiert ist, zu Long Werten, welche dann in dem Konstruktor selbst nochmal in einen Double konvertiert werden. [12]

6.1.2 Individuelle Darstellung mit GraphViewHelper

Die GraphView wird in der XML-Datei der Activity als View eingebunden.

```
//Hilfsmethode um den Graphen mit der richtigen größe(x- und y-Achse) und dem richtigen UI zu bauen
public static void setUpGraphView(GraphView graphView, int maxX, int maxY){
    graphView.getViewport().setXAxisBoundsManual(true);
    graphView.getViewport().setYAxisBoundsManual(true);
    graphView.getViewport().setMinY(0);
    graphView.getViewport().setMaxY(maxY);
    graphView.getViewport().setMinX(1);
    graphView.getViewport().setMaxX(maxX);
    graphView.setTitleColor(R.color.main_color);
    graphView.setTitleTextSize(18);
    graphView.getGridLabelRenderer().setGridStyle(GridLabelRenderer.GridStyle.NONE);
    graphView.getGridLabelRenderer().setHorizontalLabelsColor(Color.WHITE);
    graphView.getGridLabelRenderer().setVerticalLabelsColor(Color.WHITE);
}
```

Source-Code 9: setUpGraphView Methode der Hilfsklasse GraphViewHelper

Für die individuelle Darstellung der beiden GraphViews wurde eine Helfer Klasse GraphViewHelper implementiert. Diese öffentliche Helfer Klasse hält statische Methoden ohne Rückgabewert, welche die GraphView individualisiert visualisieren.

Ein Objekt vom Typ GraphView hält Objekte vom Typ GridLabelRenderer und ViewPort welche für die Visualisierung der View sorgen. Das Objekt GridLabelRenderer sorgt für die Darstellung des Gitternetzes und der Nummerierung des Graphen. Durch den ViewPort kann man die Werte der x- und y-Achse angeben (Beispielsweise maximale und minimale Werte der Achse).

Die Herangehensweise bei der FiTracker App war, die Graphen möglichst minimalistisch und aus ästhetischen Gründen ohne Koordinatengitter darzustellen. Auf der y-Achse ist der maximal erreichte Schwierigkeitswert das Maximum und 0 das standardmäßig das Minimum, für den Fall, das nicht trainiert worden ist. Auf der x-Achse wird ein Intervall von Tagen angezeigt, was auf der StatistikView selbst bestimmt werden kann und auf der HomeActivity der Erste bis zum letzten Tag des aktuellen Monats ist. [12]

6.2 Visualisierung dynamischer Listen durch RecyclerView

Die ViewGroup RecyclerView bietet durch eine Implementation mit dem ViewHolder Pattern die Möglichkeit eine dynamische Liste an Objekten darzustellen. RecyclerView ist eine optimierte ListView, welche seit der Android API

Lollipop ein Standard Widget ist. Ein Unterschied zur ListView und das, was der RecyclerView ihren Namen gibt, ist die Wiederverwertung von Views der dargestellten Items, welche aus dem Bildschirm „rausgewischt“ wurden. Diese Views der nicht mehr visualisierten Items werden, von beispielsweise in einer Liste, neu zu visualisierenden Items „übernommen“. Das macht die RecyclerView besonders effektiv für die Visualisierung von großen Listen und dynamischen Listen, wie unsere, welche schnell groß werden können. [13]

6.2.1 Standardmäßige Implementation der RecyclerView

```
RecyclerView recyclerView = findViewById(R.id.calendar_recycler);
recyclerView.setLayoutManager(new GridLayoutManager( context: this,  spanCount: 7));
adapter = new CalendarAdapter( clickListener: this);
recyclerView.setAdapter(adapter);
```

Source-Code 10: Initialisieren einer RecyclerView

Die RecyclerView **muss** nach ViewHolder Pattern mit einem Adapter, verknüpft werden. Ohne diesen Adapter können der RecyclerView ViewGroup keine Views hinzugefügt werden, welche in einem bestimmten Layout dargestellt werden sollen. Für die Art der Darstellung sorgt der LayoutManager mit dem die zu präsentierende Liste in eindimensionaler Listenform oder in zweidimensionaler Gitterform visualisiert werden kann.

Die Adapter Klasse hält die darzustellende dynamische Liste und die innere Klasse ViewHolder. Die Adapter Klasse muss die Methoden der statischen abstrakten Klasse RecyclerView.Adapter einbinden. Die Methoden onCreateViewHolder, onBindViewHolder und getItemCount werden überschrieben.

```
public CalendarViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View calendarView = LayoutInflater.from(parent.getContext()).inflate(R.layout.calendar_field, parent,  attachToRoot: false);
    return new CalendarViewHolder(calendarView);
}
```

Source-Code 11: OnCreateViewHolder Methode

Die Methode onCreateViewHolder, ruft den Konstruktor des ViewHolder auf und gibt das konstruierte Objekt zurück. Außerdem werden in der Methode ein View Objekt mit dem LayoutInflater und der Parent Activity als Übergabeparameter erstellt. Hier kann man den seine eigenes xml Layout aus den Resources einbinden, und so ein eigenes Design für die einzelnen dargestellten Objekte visualisieren.

```
@Override
public void onBindViewHolder(@NonNull CalendarViewHolder holder, int position) {
    holder.bind(daten.get(position));
    holder.setIsRecyclable(false);
}
```

Source-Code 12: onBindViewHolder Methode

```
public CalendarViewHolder(@NonNull View itemView){
    super(itemView);
    valueTextView = itemView.findViewById(R.id.feld_value);
}
```

Source-Code 13: Viewholder Konstruktor

Der Konstruktor ruft mit super den Konstruktor von der Klasse RecyclerView.Viewholer mit dem Übergabeparameter der View des Elements auf. Außerdem werden im Konstruktor den Instanzvariablen, der verschiedenen View Elemente, die in dem Layout itemView enthalten sind (beispielsweise TextView), die Views hinzugefügt die potenziell verändert werden sollen. Dies geschieht in der bind Methode, welche der User bei Bedarf (wenn zum Beispiel partikuläre visuelle Änderungen gemacht werden sollen) implementieren kann. Die bind Methode hat die Übergabeparameter des Elements der im Adapter getragenen Liste und bestimmt wie das übergebene Element visualisiert werden soll. Dies wird durch Methoden der Instanzvariablen Views bewerkstelligt. [13]

6.2.2 Darstellung von Übungslisten mittels RecyclerView

Abbildung 13: RecyclerView von der ListExercisesActivity

Die in der App an zwei Stellen benutzten Listen von Exercises wurden mit der RecyclerView umgesetzt. Dabei wurde die standardmäßige Implementierung umgesetzt (beschrieben in 5.2.1). Die Adapter Klasse ExerciseAdapter wird von der MainActivity und der DayExerciseActivity eingebunden. Die ExerciseAdapter Klasse hält eine ArrayList mit zugewiesenen Exercise Elementen. Eine Bind Methode war nicht nötig, da Die Exercises immer in gleicher Form visualisiert werden. Der ViewHolder hat drei Instanzvariablen, welche mit die drei, in dem Layout exercise_item.xml definierten, TextViews initialisiert werden. In der onBindViewHolder Methode werden jeder dieser dargestellten TextViews die Werte Datum, Name und Schwierigkeit der Exercise zugewiesen. Die exercise_item View an der Stelle des Elements in der Liste repräsentiert nun die Instanzvariablen des Elements in den angeführten TextViews.

6.2.3 Darstellung des Kalendermonats mittels RecyclerView

Abbildung 14: Recycleriew des Kalenders von der HomeActivity

Der Kalender auf der HomeActivity wurde mit einer RecyclerView realisiert. Grundsätzlich wurde auch hier das in 5.2.1 beschriebene ViewHolder Adapter Pattern umgesetzt. Die Adapter Klasse CalendarAdapter trägt eine Liste an Datum Objekten. Die Klasse Datum hat die Instanzvariablen day und trained, welche den aktuellen Tag als integer und ob an diesem trainiert wurde als boolean Wert repräsentieren. Die Datum Objekte an denen trainiert wurde, werden in

```
public void bind(final Datum datum){
            valueTextView.setText(kalenderTage[datum.getDay()]);
            valueTextView.setTextColor(Color.WHITE);
            if(datum.trained){
            itemView.setBackground(x.getResources().getDrawable(R.drawable.calendar_item_shape_trained));
            }
            else{
                itemView.setBackground(x.getResources().getDrawable(R.drawable.calendar_item_shape));
            }
            if(datum.getDay()==DataTimeConverter.getDay()-1){
                itemView.setBackground(x.getResources().getDrawable(R.drawable.calendar_item_shape_today));
            }
            itemView.setOnClickListener(view -> {
                listener.onItemClick(datum);
                });
    }
```

Source-Code 14: bind Methode der Kalender RecyclerView

Hier werden dem visualisierten Layout, aus dem in den Resources definierten String Array Kalendertage, der String an Stelle der Instanzvariable Tag des Datum Objektes zugewiesen. Nun werden durch verschiedene IF-Bedingungen die Fälle berücksichtigt durch die der itemView des Datums andere zu visualisierende Drawables zugewiesen werden, um Tage an den trainiert wurde und den aktuellen Tag zu markieren. Als Layout wird in der HomeActivity das Grid Layout gewählt. Das GridLayout wird mit der Zeilengröße 7 initialisiert welche eine Woche darstellt. [13]

6.3 Hilfsklassen

6.3.1 DataTimeConverter

Der DataTimeConverter ist eine Hilfsklasse mit statischen Methoden. Sie dient den anderen Klassen dadurch, dass sie die gehaltenen Kalender Daten umformatieren, zu Integer Werten umwandeln oder Werte des aktuellen Datums liefern kann.

Die Methoden

- getDayFromDate
- getMonthFromDate
- getYearFromDate

sorgen für die Umwandlung von einem Date String zu einem Integer Wert. Diese Umwandlung wird zum Vergleichen von Daten gebraucht. Die Implementierung der SQLite Queries und der Vergleich der Daten in der HomeActivity, bei der Erstellung der RecyclerView und dem Entwurf der Statistik, wurden durch den Vergleich von Integer Werten deutlich vereinfacht.

Die Methoden

- getDay
- getMonth
- getYear

geben den aktuellen Tag, Monat und das aktuelle Jahr als Integer Wert zurück. Die Methoden werden im Kalender implementiert um diese.

Die Methode getAmountDaysOfMonth liefert als Rückgabewert die Anzahl der Tage des aktuellen Monats.

Die Methode getDate gibt ein String Objekt des aktuellen Monats mit der Formatierung Tag.Monat.Jahr zurück.

Durch die Methode formattedDate werden Date Objekte in String Objekte mit der Tag.Monat.Jahr Formatierung umgewandelt und zurückgegeben.

6.3.2 GraphViewHelper

Die GraphViewHelper Klasse dient der übersichtlichen Anpassung der Graphen. Sie hält drei statische Methoden, die für die Visualisierung der Graphen zuständig sind. Die beiden Methode setUpGraphView und setUpGraphView withDate haben die Funktion die Graphen einheitlich zu visualisieren. Die Funktion updateGraphView sorgt dafür, dass der Graph bei veränderten Werten der LineGraphSeries neu dargestellt wird.

6.3.3 Date

Die Klasse Date wird von der RecyclerView als Element, das die Tage repräsentiert, implementiert. Date hat als Instanzvariablen den boolean trained und den Integer Wert day. Außerdem bietet die Klasse getter und setter für ihre Instanzvariablen an. Die Date Klasse wird in der HomeActivity und der CalendarAdapter Klasse zum Darstellen der Tage benötigt.

7 Zusammenfassung und Fazit

7.1 Zusammenfassung

Die von uns implementierte FiTracker App ermöglicht es, wie im Projektvorschlag beschrieben, dem User seine Fitness Erfolge zu tracken. Es wurde ein Splash-Screen mit dem Namen der App implementiert. Die HomeView der App wurde mit einem Kalender, einer Statistik und einem Training beginnen Button realisiert. Es können durch die Auswahl von Übungen in der AddEditActivity Übungen mit den Eigenschaften:

- Beschreibung
- Gewicht
- Datum
- Sätze
- Wiederholungen
- Schwierigkeit
- Vorlage

gespeichert werden. Wenn eine Übung hinzugefügt wurde, wird diese mit den anderen Übungen des Tages aufgelistet. Die App bietet die Möglichkeit in dem in der HomeView visualisierten Kalender, auf einen Kalendertag zu „drücken“ und die verrichteten Übungen des Tags zu sehen oder auch zu bearbeiten. Die Statistik auf der HomeView zeigt alle Übungen des Monats und deren Schwierigkeit mittels eines Graphen. Durch das „Drücken“ auf die Statistik gelangt man zu StatistikView, auf der man die Statistik individualisiert anzeigen kann. Individualisiert werden können das Intervall der Tage auf der x-Achse und die auf der y-Achse repräsentierte Schwierigkeit für die Übungen.

Die Datenpersistierung wurde mit SQLite und Android Room realisiert. Die Übung wurde als zu speichernde Entitätsklasse gewählt. Die Datenpersistierung wurde nach dem MVVM Pattern implementiert.

Die abzubildenden dynamischen Listen wurden mithilfe von Android RecyclerView dargestellt.

Die implementierten Graphen wurde mittels der Library GraphView umgesetzt.

7.2 Fazit

Abschließend kann gesagt werden, dass wir durch die Umsetzung des Projektes unsere Fähigkeiten der nativen Android Entwicklung in Java vertiefen konnten. Auch die Herangehensweise an Software-Entwicklungsprojekte in diesem Umfang konnte erlernt werden. Wir haben durch die Planung des Projektes viele verschiedene Möglichkeiten kennenlernen dürfen Software-Architekturmuster

umzusetzen, die nicht nur in der nativen Android Entwicklung, sondern auch in vielen anderen Bereichen der Software-Entwicklung eingesetzt werden können. Dieses gewonnene Wissen hat uns gezeigt, dass vor Allem die zeitliche Anordnung der Entwicklung eines Projektes wichtig ist und das durch diese im späteren Verlauf des Projektes viel Zeit gewonnen werden kann. Außerdem hat sich herausgestellt, dass der Umgang mit dem Version Control System GitHub elementar ist für die Zusammenarbeit an einem Softwareprojekt. Die effiziente Möglichkeit den Source-Code unter den Projektmitgliedern zu teilen, wird erst durch dieses gegeben und hat uns viel Zeit erspart. Abschließend können wir sagen das die Arbeit mit Android Studio und die Entwicklung unserer App eine sehr positive Erfahrung und ein großer Lernprozess war. Zukünftig wird uns die native Android App-Entwicklung und der Umgang mit Software-Architekturmustern deutlich leichter fallen.

8 Literaturverzeichnis

[1] RegisFrey, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg.

[2] SHISHIR, "medium.com," 27 05 2018. [Online]. Available: https://medium.com/programming-lite/mvp-in-android-part-i-97c83a8461ce.

[3] R. Mishra, „Geeksforgeeks," 24 01 2021. [Online]. Available: https://www.geeksforgeeks.org/android-architecture-patterns/. [Zugriff am 24 02 2022].

[4] P. D. E. Behrends, „Informatik-Aktuell," 2019 05 15. [Online]. Available: https://www.informatik-aktuell.de/entwicklung/programmiersprachen/mit-android-jetpack-die-app-entwicklung-beschleunigen.html. [Zugriff am 23 02 2022].

[5] Deepanshu, „Mindorks," 18 05 2020. [Online]. Available: https://blog.mindorks.com/introduction-to-room-persistent-library-in-android. [Zugriff am 18 02 2022].

[6] Programmierenlernen, „Chris," 29 12 2018. [Online]. Available: https://www.programmierenlernenhq.de/tutorial-android-activity-und-fragment-lifecycle/. [Zugriff am 23 02 2022].

[7] D. Dortinau, „docs.microsoft.com," 25 02 2022. [Online]. Available: https://docs.microsoft.com/de-de/xamarin/android/app-fundamentals/activity-lifecycle/. [Zugriff am 25 02 2022].

[8] arpitsrivastava288, „How to Build a Grocery Android App using MVVM and Room Database?," 27 12 2021. [Online]. Available: https://www.geeksforgeeks.org/how-to-build-a-grocery-android-app-using-mvvm-and-room-database/.

[9] Google Developers, „Google Developers," 24 02 2021. [Online]. Available: https://developer.android.com/reference/android/arch/persistence/room.

[10] C. yadav, "tutorialspoint," 05 11 2018. [Online]. Available: https://www.tutorialspoint.com/android-asynctask-example-and-explanation.

[11] digital-solutions, „digital-solutions," [Online]. Available: https://digital-solutions.consulting/blog/repository-in-androids-mvvm-architecture/.

[12] J. Gehring, „GraphView," 5 4 2019. [Online]. Available: https://github.com/jjoe64/GraphView/wiki.

[13] R. Friedel, „WillowTreeApps," [Online]. Available: https://www.willowtreeapps.com/craft/android-fundamentals-working-with-the-recyclerview-adapter-and-viewholder-pattern.

[14] Vecteezy, Artist, *Hantel.* [Art]. 2022.

[15] Gradienta, Artist, *Gradient Background.* [Art]. 2022.

[16] C. i. Flow, „https://codinginflow.com/," 13 09 2019. [Online]. Available: https://www.youtube.com/watch?v=HhmA9S53XV8&list=RDCMUC_Fh8kvtkVPkeihBs42jGcA&index=1. [Zugriff am 14 02 2022].

Eidesstattliche Erklärung

Hiermit erkläre ich/ Hiermit erklären wir an Eides statt, dass ich/ wir die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe/ haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

26.02.2022 Osnabrück — Tim Cirksena Tom Sattler
Ort, Datum — Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Urheberrechtliche Einwilligungserklärung

Hiermit erkläre ich/ Hiermit erklären wir, dass ich/wir damit einverstanden bin/sind, dass meine/ unsere Arbeit zum Zwecke des Plagiatsschutzes bei der Fa. Ephorus BV bis zu 5 Jahren in einer Datenbank für die Hochschule Osnabrück archiviert werden kann. Diese Einwilligung kann jederzeit widerrufen werden.

26.02.2022 Osnabrück — Tim Cirksena Tom Sattler
Ort, Datum — Unterschrift

(bei Gruppenarbeit die Unterschriften sämtlicher Gruppenmitglieder)

Hinweis: Die urheberrechtliche Einwilligungserklärung ist freiwillig.