

SWE04 – Softwareentwicklung – Mobile Anwendungen

Projektaufgabe – Teil 3 (Phase 3)

Abgabe von Gruppe:

01-Bianca Bernhardt, 12-Sonja Gradwohl

Repository link: <https://github.com/sogr85/MobileApplications>

Seit Phase 3 (dieser Phase) verwenden wir zum Austausch und Zusammenfügen der jeweiligen Programmierabschnitte ein „private repository“ auf github.

Inhalt

1. Funktionalitäten in der App die nichts mit der UI zu tun haben und im Hintergrund ablaufen.	2
Tab Messung:	2
Tab Statistik:	2
Tab Einstellung:	2
Tab Entspannung:	3
2. Mögliche Gefährdungspotentiale für die Userexperience	3
Entspannung / Media Player:	3
Statistik / Datenbank:	3
Einstellungen / GPS:	3
Funktionalitäten im Vordergrund:	3
Tab Messung:	3
Tab Statistik:	3
Tab Einstellungen:	3
Tab Entspannung:	4
3. & 4. Services und Toasts	4
Measurement Intent Service, GPS Intent Service & FetchAddressIntentService	4
5. Datenbankmodell	9
6. Abfrage – API (PulsdataDao) Queries, die in unserer App gestellt werden müssen:	11

1. Funktionalitäten in der App die nichts mit der UI zu tun haben und im Hintergrund ablaufen

Tab Messung:

In der Standardmessung gibt es eine vorab definierte Zeit von 1 Minute, die vom User nicht verändert werden kann.

Ablauf Messung: User startet Messung beim Drücken des Herzsymbolbuttons. Nach Ablauf der Minute wird der Wert auf Sekunde 60 automatisch gewertet, aufgezeichnet und an die Datenbank übermittelt. Dies passiert im Hintergrund.

In der Individualmessung kann der User die Zeit selbst bestimmen max. 60 Minuten. Jeder gemessene Wert pro Minute wird ebenfalls an die Datenbank übertragen.

Die Meldungen (Puls im Normalbereich, Puls erhöht, Puls leicht erhöht) sind mit dem Herzbutton der die Messung durchführt, verknüpft. Normalbereich eines durchschnittlichen Erwachsenen liegt zwischen 60-80 Schlägen pro Minute, sollte der Puls bis einschließlich 80 Schlägen sein steht unter dem Messwert die Information, „ihr Puls ist im Normalbereich“ in grüner Farbe, ist der Puls zwischen 81-99 ist die Schrift orange und die Meldung „ihr Puls ist leicht erhöht“ und ab 100 wird der User mit einer roten Schrift darauf hingewiesen „Achtung Puls ist erhöht – Zeit für Entspannung“. Diese Selektion läuft im Hintergrund ab.

Die Meldungen „Berühren zum Starten“, „Berühren zum Stoppen“ sowie „Puls gefunden“, „Puls nicht gefunden“ und „Armband eng an Kontaktstelle am Handgelenk anlegen“ werden auch parallel zur Messung im Hintergrund generiert, indem beispielsweise kein Pulssignal gefunden wird oder das Signal unzureichend übermittelt wird.

Tab Statistik:

Eine Auswertung über einen bestimmten Zeitraum wird mit der Kalenderfunktion vom User ausgewählt (Start-Enddatum). Als Ergebnis wird am gleichen Bildschirm der Durchschnittspulswert über die Einzelwerte der Tage über den Zeitraum berechnet und angezeigt. Diese Berechnung wird auch fernab vom User im Hintergrund durchgeführt. Auch die Lokalität – Heim oder Arbeit wird im Hintergrund durch die GPS Funktion ermittelt und ausgeworfen. Einstellungsmöglichkeit beschränkt sich hier auf „GPS-Koordinaten – „Arbeit“ und „GPS-Koordinaten – „Heim“. Als Ergebnis steht somit z.B. „75BPM Heim“, wenn sich der User überwiegend zu Hause aufgehalten hat (im ausgewählten Zeitraum).

Tab Einstellung:

GPS sucht im Hintergrund die Lokalität (Koordinaten) wo sich die Person zum Zeitpunkt der Messung aufhält. Da diese Funktion die Benutzererfahrung gefährden würde ist es sinnvoll dies im Hintergrund auszuführen.

Die Verknüpfung zur Datenbank passiert auch im Hintergrund.

Auch Bluetooth Low Energy muss durch den User vorab aktiviert werden damit Daten im Hintergrund zwischen Sensor und Mobiltelefon übertragen werden können.

Tab Entspannung:

Die einzelnen Musiktiteln in den jeweiligen Rubriken werden im Hintergrund durch einen Media Player abgespielt sowie auch die Preference „zuletzt gehört“ im Hintergrund des Media Players abgespeichert wird.

2. Mögliche Gefährdungspotentiale für die Userexperience

Entspannung / Media Player:

Würden die Titeln in der jeweiligen Rubrik sich nicht direkt in der App abgespeichert befinden und beispielsweise Titeln von einem externen Server heruntergeladen werden müssen, könnte es bei schlechter Internetverbindung zu Performanceproblemen kommen.

Szenario: User muss zu lange auf Abspielen des Titels warten bevor dieser wiedergegeben werden kann. Dies hätte ein schlechte Userexperience zur Folge.

Statistik / Datenbank:

Würde das Abrufen der Pulswerte aus der Datenbank zu viel Zeit benötigen würden Ergebnisse mit großer Zeitverzögerung ausgeworfen werden. Folge dadurch – schlechte Userexperience, da er zu lange auf sein Ergebnis warten muss.

Einstellungen / GPS:

Würde das Suchen der Lokalität während der Messung in Vordergrund ausgeführt werden könnte das zu erheblichen Performanceproblemen führen, da das GPS dauernd versucht die Lokalität aktuell zu halten.

Funktionalitäten im Vordergrund:

Tab Messung:

Individualmessung – User kann eine Zeit zwischen 1-60 Minuten einstellen.

Herzbutton, mittels dieses Buttons kann der User die Messung selbst starten und stoppen.

Tab Statistik:

Hier gibt der User den Zeitraum (Start- Enddatum) ein um den gewünschten durchschnittlichen Stresslevel innerhalb dieser Zeitspanne zu ermitteln. Durch Auslösen des Ergebnisbutton erhält der User seinen durchschnittlichen Pulswert über den Zeitraum und zudem die Lokalität wo dieser sich überwiegend bei den Messungen aufgehalten hat.

Tab Einstellungen:

GPS, BLE und Datenbankverbindung werden durch den User aktiviert.

Tab Entspannung:

Der User kann eine der vier Rubriken auswählen und seinen gewünschten Titel abspielen oder pausieren. Zudem kann ein User über den Schnellzugriff „zuletzt gehört“ direkt zu dem Titel gelangen den er zuletzt abgespielt hat und erspart sich so die Suche.

3. & 4. Services und Toasts

Wir haben einen Service-Ordner mit 3 Klassen, welche die Services „FetchAddressIntentService“, „GpsIntentService“ und „MeasurementIntentService“ beinhalten. Bei allen 3 Services handelt es sich um Background-Tasks, die im Hintergrund durchgeführt werden.

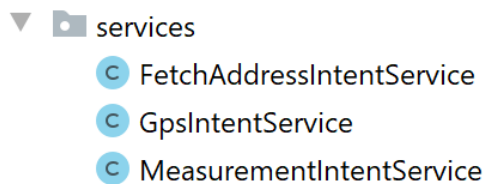


Abbildung 1: Serviceaufzählung 1

Measurement Intent Service, GPS Intent Service & FetchAddressIntentService

Der MeasurementIntentService liefert zyklisch Messdaten.

Durch das Drücken des Herzbuttons im Tab Messung startet man den Measurement Intent Service.

Die Werte basieren auf random-generierten Testpulsdaten, welche später final in der nächsten Phase durch den Sensor ermittelt werden.

```
while(!shouldStop){
    int randomPulseValue;
    // TODO Do the measurement with real hardware

    // TODO for now random value is generated for pulse value
    //https://stackoverflow.com/questions/21049747/how-can-i-generate-a-random-number-in-a-certain-range/21049922
    int min = 40;
    int max = 250;
    randomPulseValue = new Random().nextInt( bound: (max-min)+1)+min;

    // Set data
    broadcastIntent.putExtra(PULSE_VALUE, randomPulseValue);

    // Send broadcast
    sendBroadcast(broadcastIntent);

    // Sleep 100 Milliseconds.
    SystemClock.sleep( ms: 10000);
}
```

Abbildung 2: Schleife zur Messung im Measurement-Intent Service

In der Klasse `Tab_Messung` werden die jeweiligen Intent-Services in der Methode „public View onCreateView“ mittels „registerReceiver“ aufgerufen.



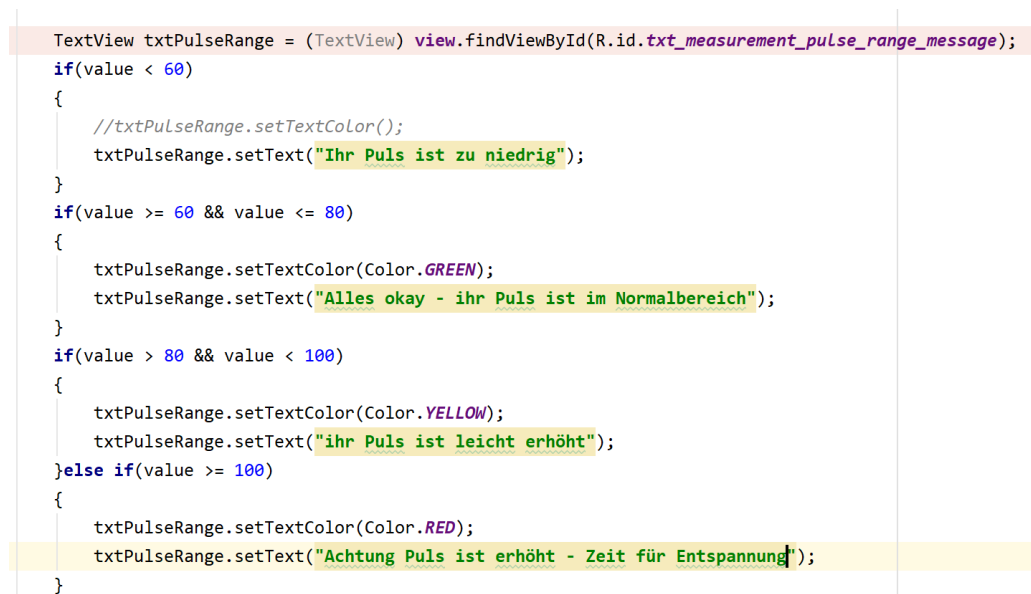
```

62 //-----
63 //Override
64 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
65
66     view = inflater.inflate(R.layout.fragment_messung, container, attachToRoot: false);
67
68     ImageButton btn = (ImageButton) view.findViewById(R.id.image_btn_heart);
69     btn.setOnClickListener(this);
70
71     getActivity().registerReceiver(receiver, new IntentFilter(MeasurementIntentService.DO_MEASUREMENT));
72     //getActivity().registerReceiver(receiver, new IntentFilter(GpsIntentService.FETCH_GPS_DATA));
73     getActivity().registerReceiver(receiver, new IntentFilter(FetchAddressIntentService.FETCH_ADDRESS_DATA));
74
75     //resultReceiver = new AddressResultReceiver(new Handler());
76
77     return view;
78 }

```

Abbildung 3: in Tab Messung - registerReceiver

In der `Tab_Messung` befinden sich ebenfalls die jeweiligen Abfragen, je nach Höhe des Pulsbereiches, welche mittels „if-Abfragen“ in die jeweiligen Kategorien gegliedert sind.



```

TextView txtPulseRange = (TextView) view.findViewById(R.id.txt_measurement_pulse_range_message);
if(value < 60)
{
    //txtPulseRange.setTextColor();
    txtPulseRange.setText("Ihr Puls ist zu niedrig");
}
if(value >= 60 && value <= 80)
{
    txtPulseRange.setTextColor(Color.GREEN);
    txtPulseRange.setText("Alles okay - ihr Puls ist im Normalbereich");
}
if(value > 80 && value < 100)
{
    txtPulseRange.setTextColor(Color.YELLOW);
    txtPulseRange.setText("ihr Puls ist leicht erhöht");
}
else if(value >= 100)
{
    txtPulseRange.setTextColor(Color.RED);
    txtPulseRange.setText("Achtung Puls ist erhöht - Zeit für Entspannung!");
}
}

```

Abbildung 4: if-Abfragen Pulshöhe

Zudem werden auch jeweilige Toasts zu Testzwecken angezeigt, welche Auskunft über das erfolgreiche Übermitteln der Pulswerte an die Datenbank widerspiegelt und auch die jeweiligen GPS Daten, welche „long, lat, street, zipcode, city und datetime“ sind, ausgeben.

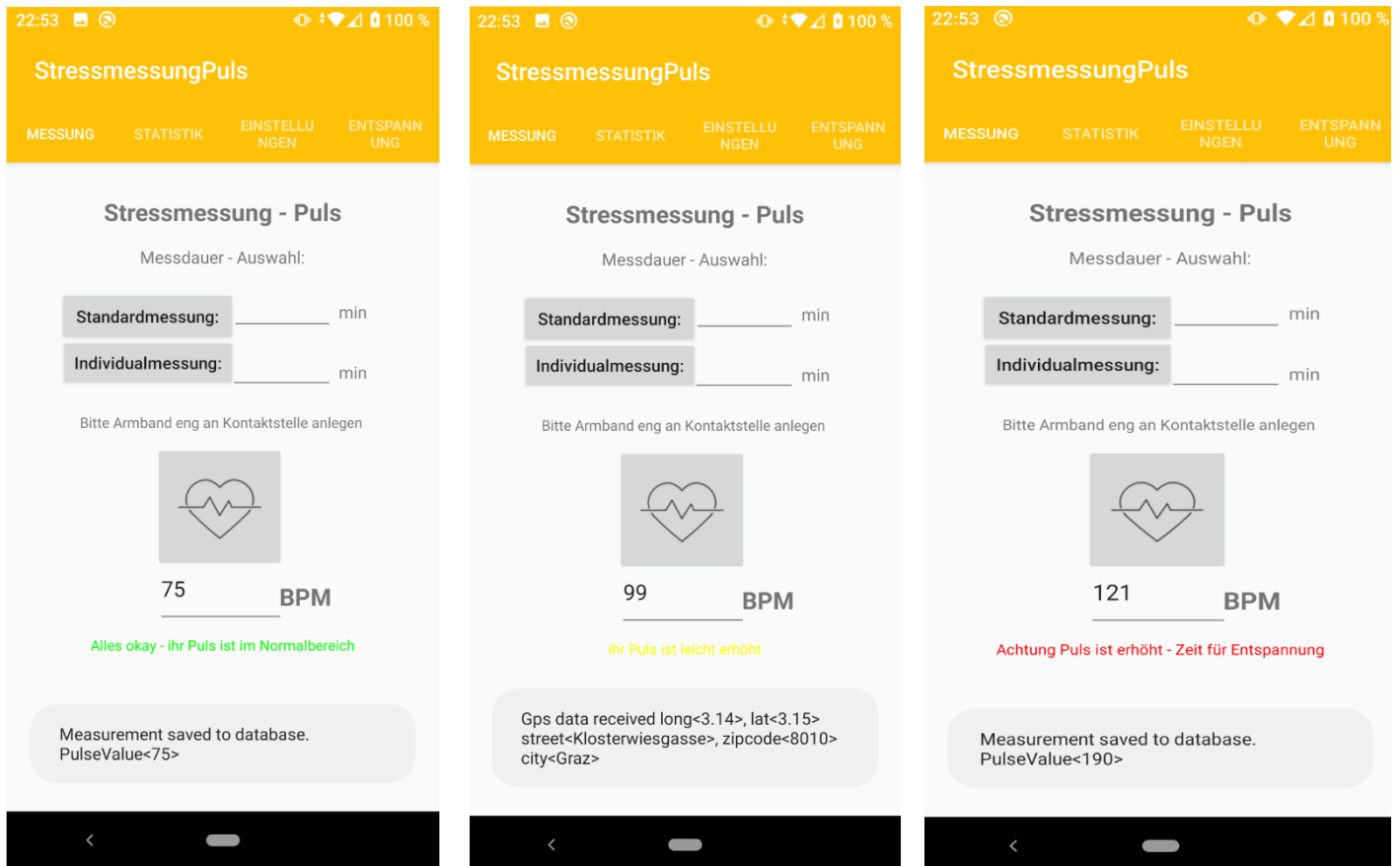


Abbildung 5: Pulshöhe Notifications + Toasts

In der Tab_Messung befindet sich der dazugehörige Code zu den obigen Toasts:

```
getActivity().runOnUiThread(  
    new ToastRunnable( _message: "Gps data received " +  
        "long<" + this.longitude + ">, lat<" + this.latitude + "> " +  
        "street<" + this.street + ">, zipcode<" + this.zipCode + "> " +  
        "city<" + this.city + ">"));  
}
```

Abbildung 6: Beispiel Codeimplementierung eines Toasts „GPS Data Received“

Verwendung der App mit GPS-Funktion:

1. Schritt: Am Mobiltelefon muss vor Öffnen der App bzw. innerhalb der App vor dem Start einer Messung, die Standortaktivierung und das WLAN/Mobiles Internet aktiviert sein.
2. Schritt: In Einstellungen wird der Switch-GPS-Button aktiviert.
Daraufhin folgt eine Permission-Abfrage, ob wir den GPS-Zugriff erlauben wollen.
3. Schritt: Nach Aktivierung wechseln wir auf das Tab „Messung“, wo wir die jeweilige Messung mit einem Touch auf den Herz-Button starten können.
4. Schritt: Nun wird der jeweilige Pulswert angezeigt inkl. aller GPS-Daten.

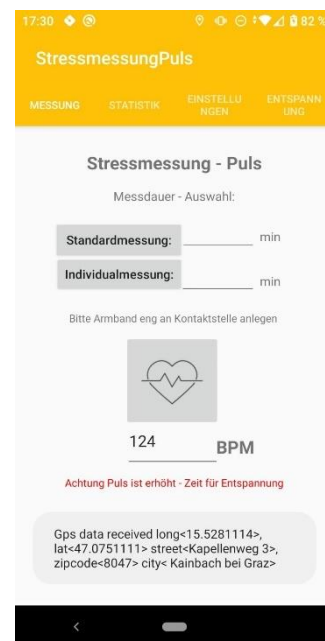
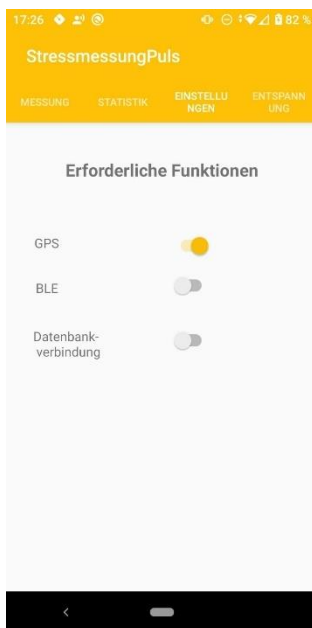


Abbildung 7: Funktionalität mit GPS-Aktivierung

Die Messung kann aber alternativ auch ohne die Aktivierung der GPS-Funktion genutzt werden. Als Resultat erhalten wir dann nur die Pulswerte, ohne GPS-Daten. Dies erkennt man daran, dass „long, lat“ auf dem Wert „0“ stehen. Siehe Abbildung 8.

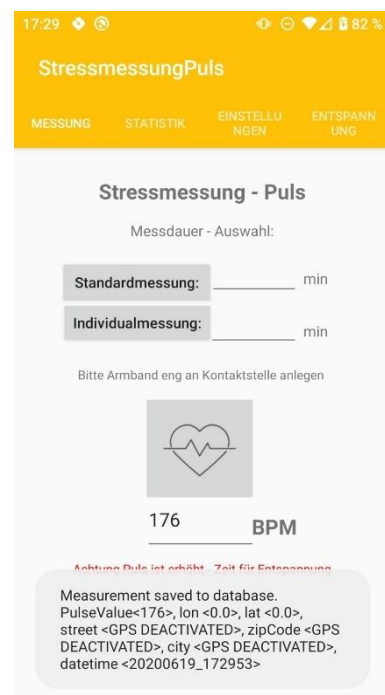


Abbildung 8: Funktionalität ohne GPS-Aktivierung

GPS Daten Ermittlung durch LocationCallback, die immer wieder die aktuelle Lokalisation ermittelt und diese als Intent an den FetchAddressIntentService weiterleitet.

```
LocationRequest locationRequest = new LocationRequest();
locationRequest.setInterval(10000);
locationRequest.setFastestInterval(3000);
locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

LocationServices.getFusedLocationProviderClient(getActivity())
    .requestLocationUpdates(locationRequest, new LocationCallback(){

        @Override
        public void onLocationResult(LocationResult locationResult) {
            super.onLocationResult(locationResult);
            //LocationServices.getFusedLocationProviderClient(getActivity())
            //    .removeLocationUpdates(this);
            if (locationResult != null && locationResult.getLocations().size() > 0) {
                int latestLocationIndex = locationResult.getLocations().size() - 1;
                double latitude =
                    locationResult.getLocations().get(latestLocationIndex).getLatitude();
                double longitude =
                    locationResult.getLocations().get(latestLocationIndex).getLongitude();

                Intent intent = new Intent(getActivity(), FetchAddressIntentService.class);
                intent.putExtra(FetchAddressIntentService.LATITUDE, latitude);
                intent.putExtra(FetchAddressIntentService.LONGITUDE, longitude);
                getActivity().startService(intent);
            }
        }
    })
```

Abbildung 9: LocationCallback in Tab_Messung

Der FetchAddressIntentServices kann mit den erhaltenen Variablen Longitude und Latitude aus den GPS Daten die vollständige Adresse mit Street, Zip_Code und City liefern.

```
public class FetchAddressIntentService extends IntentService {

    // TODO: Rename actions, choose action names that describe tasks that this
    // IntentService can perform, e.g. ACTION_FETCH_NEW_ITEMS
    public static final String FETCH_ADDRESS_DATA = "com.example.stressmessungpuls.services.action.fetchAddressData";

    // input/output variables
    public static final String LONGITUDE = "LONGITUDE";
    public static final String LATITUDE = "LATITUDE";
    // output variables
    public static final String STREET = "STREET";
    public static final String ZIP_CODE = "ZIP_CODE";
    public static final String CITY = "CITY";

    public FetchAddressIntentService() { super(name: "FetchAddressIntentService"); }

    @Override
    protected void onHandleIntent(@Nullable Intent intent) {

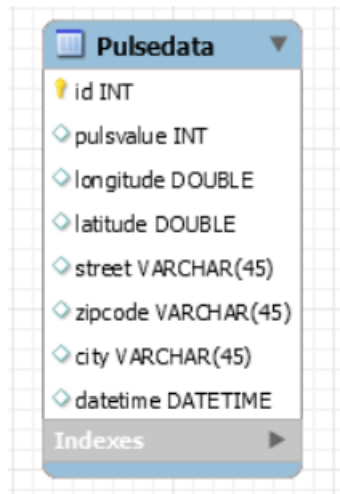
        // https://o7planning.org/de/10421/die-anleitung-zu-android-services

        if (intent != null){
            String errorMessage = "";
            double longitude = intent.getDoubleExtra(LONGITUDE, defaultValue: 0.0);
            double latitude = intent.getDoubleExtra(LATITUDE, defaultValue: 0.0);

            Geocoder geocoder = new Geocoder(context: this, Locale.getDefault());
            List<Address> addresses = null;
```

Abbildung 10: FetchAddressIntentService

5. Datenbankmodell



Da sich nur ein User mit der Datenbank verbindet ist ein etwaiger Username nicht relevant.

Der Primärschlüssel ist die ID und Fremdschlüssel wird keiner benötigt, da es nur eine Tabelle gibt mit 8 Spalten.

Die ID generiert sich in der Datenbank automatisch.

Der Pulswert wird vom Sensor bezogen, der Ort wird durch Aktivierung GPS in Einstellungen ermittelt und Datum + Uhrzeit werden vom Mobiltelefon ausgelesen.

Pulsedata - Table										
Table Name: Pulsedata		Schema: mydb								
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
pulsvalue	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
longitude	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
latitude	DOUBLE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
street	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
zipcode	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
city	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
datetime	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Abbildung 11: Datenbankmodell mit Attributen

Wir haben uns bzgl. Datenanbindung für die „Rooms-Datenbank“ entschieden, da wir uns mit dieser gut zurecht gefunden haben.

Der Datenbank-Ordner „database“ besteht aus der abstrakten Klasse „AppDatabase“, der Klasse „Pulsedata“ und dem Interface „PulsedataDao“.

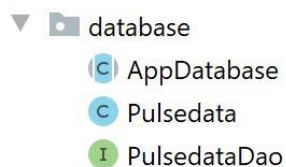


Abbildung 12: Ordner database 1

Wie auch in der Modellierung haben wir in der Implementierung als den Primärschlüssel die „id“ gewählt und diesen autogeneriert.

```
import androidx.room.*;

@Entity
public class Pulsedata {
    @PrimaryKey(autoGenerate = true)
    public int id;
```

Abbildung 13: Klasse Pulsedata mit Primary Key

In der Tab_Messung ist die Datenbank Runnable, dass für uns die Pulsdaten in die Datenbank speichert.

```
public class DatabaseRunnable implements Runnable {
    private Pulsedata data;
    private AppDatabase database;
    public DatabaseRunnable(AppDatabase _database, Pulsedata _data) {
        this.database = _database;
        this.data = _data;
    }

    @Override
    public void run() {
        database.pulsedataDao().insertAll(this.data);
        getActivity().runOnUiThread(new ToastRunnable(
            _message: "Measurement saved to database. PulseValue<" + this.data.pulsevalue + ">, " +
                "lon <" + this.data.longitude + ">, lat <" + this.data.latitude + ">, " +
                "street <" + this.data.street + ">, zipCode <" + this.data.zipcode + ">, " +
                "city <" + this.data.city + ">, datetime <" + this.data.datetime + ">"));
    }
}
```

Abbildung 14: InsertAll Befehl in Tab_Messung

6. Abfrage – API (PulsdataDao)

Queries, die in unserer App gestellt werden müssen:

```
@Dao
public interface PulsdataDao {
    @Query("SELECT * FROM pulsedata")
    List<Pulsdata> getAll();

    @Query("SELECT * FROM pulsedata WHERE id IN (:id)")
    List<Pulsdata> loadAllByIds(int[] id);

    @Query("SELECT * FROM pulsedata WHERE pulsevalue LIKE :pulsevalue AND " +
        "city LIKE :city AND " +
        "datetime LIKE :datetime LIMIT 1")
    Pulsdata findByDate (int pulsevalue, String city, String datetime);

    @Query("SELECT AVG (pulsevalue) from pulsedata")
    int getAverage();

    @Insert
    void insertAll(Pulsdata... pulsedataplural);

    @Delete
    void delete(Pulsdata pulsedata);
}
```

Abbildung 15: Auszug aus dem Bild PulsdataDao

Da wir für den User eine Mittelwertberechnung über einen von ihm definierten Zeitraum anbieten, haben wir dies bereits vorimplementiert. Nachfolgend die Query dazu:

```
@Query("SELECT AVG (pulsevalue) from pulsedata")
int getAverage();
(Auszug aus dem Bild Abbildung 13.)
```