Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Realtime odečítání a vizualizace senzorických dat ze sporttesterů

2020

Ivo Horák

Vedoucí práce: Mgr. Radek Janoštík

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Ivo Horák
Název práce:	Realtime odečítání a vizualizace senzorických dat ze sport-testerů
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Radek Janoštík
Počet stran:	67
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Ivo Horák
Title:	Realtime reading and visualization of sensor data from sport testers
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Radek Janoštík
Page count:	67
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

V rámci bakalářské práce byla vyvinuta aplikace pro mobilní platformu Android a rozšířen firmware do senzoru od firmy Movesense. Aplikace umožňuje odečítání a vizualizaci senzorických dat v reálném čase z několika senzorů současně. Firmware senzoru dokáže detekovat krok a stav, ve kterém se sportovec právě nachází pomocí naměřených dat z akcelerometru.

Synopsis

The aim of this bachelor's thesis is to develop an Android application and extend firmware for Movesense sensors. The application is able to read and visualize data from multiple sensors simultaneously in real-time. The sensor's firmware can detect steps and distinguish between several states of the athletes using measured data from an accelerometer.

Klíčová slova: Android; senzor; detekce kroku; vizualizace

Keywords: Android; sensor; step detection; visualization

Chtěl bych poděkovat Mgr. Radkovi Janoštíkovi za cenné rady, testování a věcné připomínky při konzultacích a během tvorby této bakalářské práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V posledních letech se stalo mobilní zařízení (dále telefon) běžnou součástí každodenního života. S rostoucím výkonem telefonů rostou i možnosti, co tyto zařízení zvládnou. Jednou z hlavních předností telefonu je jeho velikost a s tím i úzce spojena jeho mobilita. Díky tomu lze mít telefon u sebe téměř vždy, což otevírá nové způsoby jeho využívání. Jeden z těchto způsobů může například být sledování denní aktivity, která je v dnešním světe velice opomíjena.

Každý telefon pohání operační systém. Mezi nejrozšířenější patří:

- **Android OS** je otevřený operační systém vyvíjený společností Google, převážně určen pro mobilní zařízení a tablety. K roku 2019 tento operační systém používalo 86.6 % mobilních zařízení na trhu. [1]
- **Apple iOS** je uzavřený operační systém, který vyvíjí společnost Apple. Apple iOS je výhradně určen pro telefony od společnosti Apple a k roku 2019 tento operační systém využívá 13.4 % mobilních zařízení. [1]

Každý telefon je vybaven nespočtem senzorů, mezi nejznámější patří akcelerometr, gyroskop, magnetometr nebo také snímač tepové frekvence. Proto mnoho sportovců telefon používá právě jako zařízení, které jim pomáhá analyzovat jejich výkon.

Pro sportovce je značně nekomfortní mít telefon u sebe během sportu, proto většina z nich preferuje tzv. chytré hodinky, které obsahují také mnoho senzorů a tímto odstraňují potřebu mít telefon během sportu. Po skončení tréninku jsou informace pomocí technologie Bluetooth přeneseny do telefonu a zde vizualizovány.

Obecným problémem chytrých hodinek je jejich přesnost. Jelikož hodinky nemají těsný kontakt s kůží sportovce, data, například z tepového snímače, musejí být softwarově upravena. Další nevýhodou může být i cena, která je poměrně vysoká.

Řešením tohoto problému mohou být hrudní pásy. Hrudní pásy nabízejí přesné měření tepové frekvence, jelikož mají těsnější kontakt s kůží sportovce za daleko nižší cenu. Naopak jejich problémem je to, že ve většině případů měří právě pouze onu tepovou frekvenci a nedisponují žádnou další funkčností.

V rámci této bakalářské práce byla vyvinuta aplikace pro Android OS, pomocí které je možné sledovat několik sportovců současně v reálném čase. To znamená, že je možné vidět počet kroků, spálené kalorie, celkovou vzdálenost, tepovou frekvenci a také elektrokardiogram[1] (dále EKG) již během výkonu každého z nich. Všechny tyto funkce a některé další jsou popsány v kapitole 7. Dále byl rozšířen firmware senzoru od společnosti Movesense, který umí detekovat krok a stav, ve kterém se právě nachází a poté tyto informace posílá telefonu, kde jsou vizualizovány.

[1] Elektrokardiogram je záznam časové změny elektrického potenciálu způsobeného srdeční aktivitou. [2]

2 Cíle

Před výběrem zařízení a začátkem vývoje byly stanoveny cíle, které by aplikace měla splňovat. Jedná se především o podporu skupiny sportovců, funkčnost v uzavřeném prostoru, funkčnost v reálném čase, kompatibilita a použitelnost. V této podkapitole jsou popsány všechny zmíněné cíle.

2.1 Podpora skupiny sportovců

Jedním z hlavních cílů byla podpora více sportovců. Aplikace by měla umožnit připojení několika sportovců k jednomu telefonu a všechna data vizualizovat na jednom místě. Tuto funkčnost nám nabízí například technologie Bluetooth, která podporuje souběžné připojení až 7 zařízení.

2.2 Funkčnost v uzavřeném prostoru

Funkčnost aplikace by neměla být ovlivněna tím, kde trénink probíhá. Musí fungovat i v hale, kde nemusí být dostupné připojení k internetu. Dále například GPS signál v halách nemusí být stabilní, tedy funkčnost celé aplikace by neměla záviset na této technologii. Pro přenos dat se proto využívá Bluetooth, který je nezávislý na prostředí.

2.3 Funkčnost v reálném čase

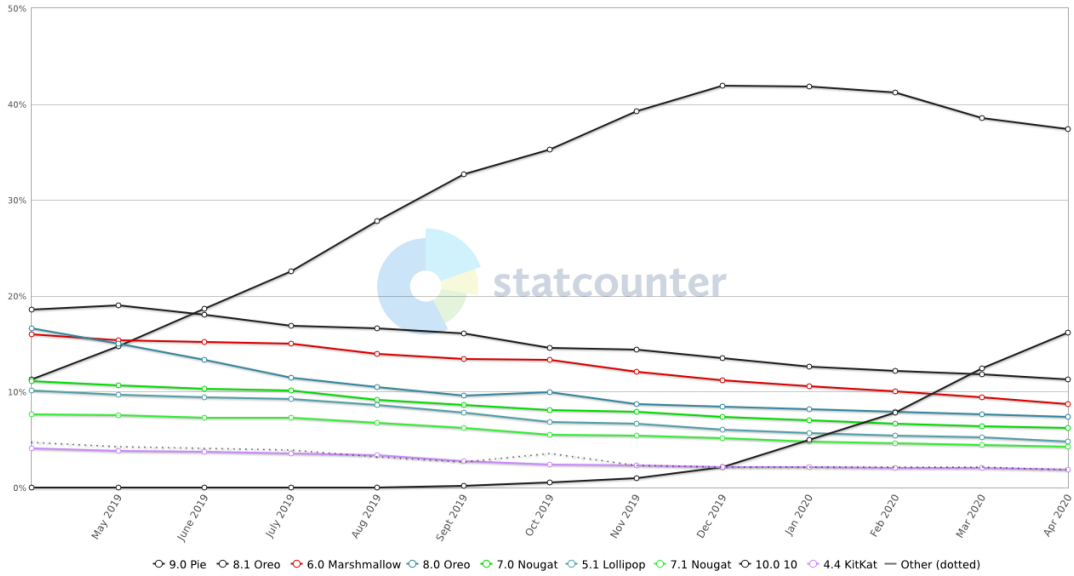
Aplikace by měla zobrazovat data ve chvíli, kdy jsou k dispozici. Z toho důvodu ukládání dat do paměti zařízení a poté synchronizace s telefonem není dostatečné řešení. Senzor odesílá data hned poté, co jsou k dispozici.

2.4 Kompatibilita

Android OS běží na mnoha zařízeních různých velikostí. Je proto žádoucí, aby aplikace plnila svou funkčnost jak na malých telefonech, tak i na tabletech. Proto se při vývoji aplikace využívaly tzv. virtuální jednotky, které jsou automaticky přepočítány na konkrétní rozlišení zařízení. Aplikace byla vyvíjena a poté testována na běžných telefonech o velikosti displeje 5,8 palců. Pro zajištění co největší dostupnosti aplikace bylo nutné brát v potaz i mnoho verzí operačního systému Android. Proto nejstarší verzí Androidu, kterou aplikace podporuje je verze Lollipop z roku 2014. Jak je možné vidět z grafu 1, Android nižší verze než je Lollipop využívají necelá 2 % telefonů.

2.5 Použitelnost

Při tvorbě aplikace byla snaha vytvořit co nejintuitivnější a nejpřehlednější uživatelské rozhraní. Proto byla hlavní část aplikace rozdělena do dvou sekcí - obecný přehled, kde lze vidět všechny připojené sportovce a jejich základní informace, a



Obrázek 1: Graf znázorňující distribuci jednotlivých Android verzí [3]

detailnější přehled každého sportovce. Tento přehled pak obsahuje i grafy, které se v reálném čase aktualizují.

3 Výběr zařízení

Tato kapitola popisuje projekt na úplném začátku, kdy proběhl průzkum již existujících řešení a jejich možností. Poté následoval výběr zařízení. Cílem bylo najít takové zařízení, které by nabízelo veřejné *Application Programming Interface*[2] (dále API) ke všem datům, které dané zařízení měří. Bylo přitom zapotřebí brát v úvahu důležitou vlastnost a to, aby nevyužívalo Internet a nebylo zcela závislé na GPS.

3.1 Současná situace mezi aplikacemi

Existuje nespočet aplikací, které se zabývají sledováním sportovce při sportu. Dokonce i sportovní značky mají vlastní aplikace jako jsou například Nike Run Club[3] nebo Adidas Training by Runtastic[4]. Obě tyto aplikace podporují připojení hrudního pásu. Tento pás měří pouze srdeční tep a naměřená data posílá zpět do telefonu. Telefon pak tato data vyhodnocuje a zpřesňuje tak své výpočty (např. spálené kalorie). Stále je však zapotřebí mít u sebe telefon, který měří další data, například uběhnutou vzdálenost, počet kroků. Další nevýhodou je fakt, že tyto aplikace nepodporují skupinu sportovců a v drtivé většině jejich funkcionalita závisí na GPS.

3.2 Problém chytrých hodinek

Mezi nejznámější prodejce chytrých hodinek patří Apple se svými Apple Watch, Fitbit a Garmin. Všechny tyto firmy již nabízí kompletní řešení pro sportovce. Jak již bylo zmíněno, cílem této práce bylo vytvořit aplikaci, která by podporovala skupinu sportovců a data přenášela v reálném čase bez použití internetu. Žádná z těchto firem nenabízí veřejné API, které by umožňovalo data číst přes Bluetooth nebo jinou technologii. Výjimkou byl Fitbit a Garmin, který nabízel jedno veřejné API a to pro srdeční tep pomocí ANT+.

ANT+ je technologie, která se používá specificky pro sportovní zařízení (senzory), konkrétně pro komunikaci mezi senzorem (například hodinkami) a telefonem. Tato technologie je k tomu přímo určená a ve srovnání s Bluetooth nabízí téměř neomezený počet současných připojení.

Jelikož srdeční tep je nedostačující informace, chytré hodinky nebyly vhodným zařízením pro splnění již výše uvedených cílů.

[2] Application Programming Interface (API) označuje v informatice rozhraní pro programování aplikací. Jde o množinu procedur, funkcí, tříd či protokolů, které může programátor využít během vývoje. [4]

[3] Aplikace dostupná z: https://play.google.com/store/apps/details?id=com.nike.plusgps&hl=cs

[4] Aplikace dostupná z: https://play.google.com/store/apps/details?id=com.runtastic.android.results.lite&hl=cs

3.3 Arduino

Další možností při výběru zařízení bylo Arduino. Arduino je open-source platforma určená pro tvorbu malých počítačů. Základem je deska, ke které lze připojovat další komponenty. Dostupné komponenty mohou být i senzory jako jsou akcelerometr, gyroskop, snímač tepové frekvence apod. Arduino nabízí i vývojové prostředí, které slouží k tvorbě softwaru. Toto řešení se zdá téměř perfektní, jelikož by bylo možné si vybrat specifické senzory a vše na míru aplikaci sestavit. Tímto by byla zajištěna kontrola nad softwarem i nad hardwarem.

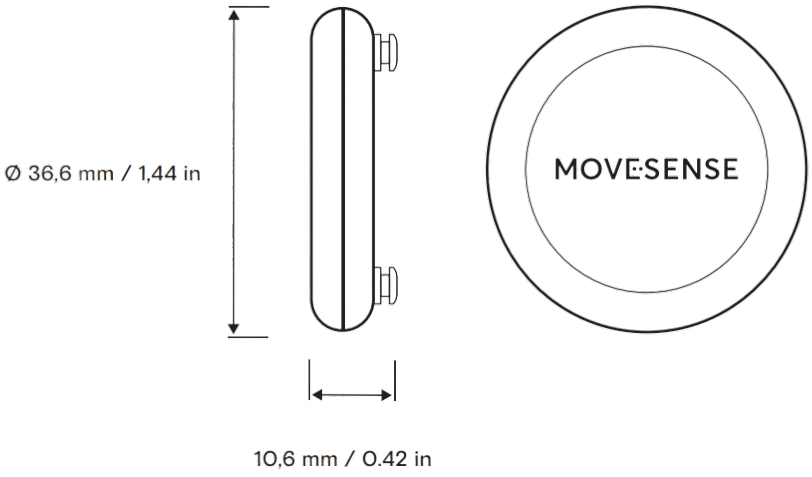
Hlavním problémem Arduina byla jeho velikost. Jelikož Arduino se používá převážně na prototypování, bylo by velmi náročné vytvořit „krabičku“, která by byla dostatečně malá na to, aby nevadila při sportu, a zároveň byla dostatečně přesná (některé senzory je potřeba kalibrovat). Jelikož by konstrukce takového zařízení byla poměrně náročná, nebylo Arduino použito.

3.4 Movesense senzor

Movesense je finská firma spadající pod Amer Sports, která zastřešuje firmy jako Salomon, Wilson, Atomic nebo Suunto. Nezaměřuje se na specifický sport nebo aktivitu, spíše zprostředkovává tzv. vývojářský balíček, který zahrnuje senzor a veřejné API pro komunikaci mezi telefonem a senzorem s využitím Bluetooth. Movesense obsahuje mnoho dílčích senzorů jako jsou akcelerometr, gyroskop, magnetometr, teploměr a měřič srdečního tepu. Svou velikostí patří mezi jedny z nejmenších sportovních senzorů na trhu. Jeho rozměry je možné vidět na obrázku 2. Movesense dále nabízí i příslušenství k senzoru jako je například hrudní pás nebo oděvní klipsy.

Movesense dovoluje psát i vlastní firmware do senzoru v jazyku C++. Tímto lze všechny výpočty provádět přímo na senzoru a pouze zasílat výsledky do telefonu. Firmware lze pak aktualizovat pomocí Bluetooth využitím aplikace, která je dostupná pro iOS v App Store nebo pro Android v Google Play.

Movesense je vhodným řešením pro tuto bakalářskou práci z důvodu své otevřenosti. Nabízí dobře zdokumentované API, jak pro komunikaci mezi telefonem a senzorem, tak i pro psaní vlastního firmwaru. Komplikací je pak skutečnost, že Movesense senzor nabízí pouze surová data, proto všechny informace, jako jsou uběhnutá vzdálenost, spálené kalorie nebo rychlost je z nich nutno nejprve vypočítat.



Obrázek 2: Rozměry Movesense senzoru [5]

4 Platforma Movesense

Jak již bylo nastíněno v kapitole 3.4, platforma Movesense se skládá ze dvou hlavních částí:

- Mobilní knihovna
- Senzorová knihovna

V této kapitole jsou popsány obě knihovny. První část této kapitoly se zabývá mobilní knihovnou, která je určena pro připojení a odpojení senzoru a dále pro výměnu dat se senzorem. V druhé části je popsáno, jak lze rozšířit firmware senzoru o vlastní uživatelsky definované služby. Informace vycházejí zejména z [6].

4.1 Mobilní knihovna

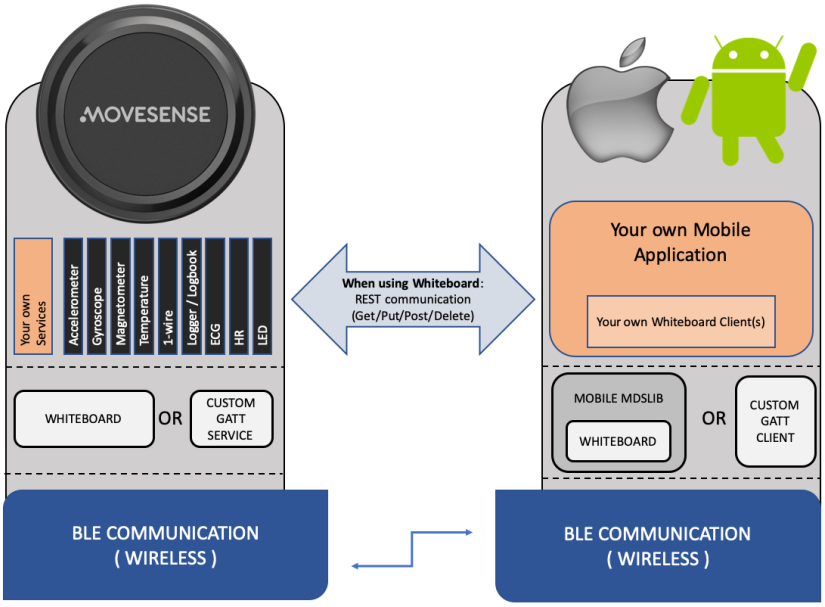
Movesense senzor se skládá z dílčích senzorů, které měří surová data. Mobilní knihovna (dále MDS) zprostředkovává API, které slouží ke komunikaci se senzorem - konkrétně s Whiteboard (tabulí). Whiteboard je detailněji popsána v sekci 4.2.1.1. V této chvíli stačí vědět, že se jedná o nízkoúrovňovou knihovnu, která zprostředkovává komunikaci mezi senzorem a telefonem. MDS je de facto obal (wrapper), který obaluje tuto nízkoúrovňovou komunikaci a vytváří tak jednoduchou asynchronní komunikaci.

Asynchronní komunikace znamená, že některé události (jedná se převážně o události u kterých není jisté, jak dlouho se budou vykonávat) jsou vykonávány mimo běh hlavního vlákna. Událost pak může informovat hlavní vlákno o jeho dokončení, selhání nebo postupu skrze zpětná volání, která jsou popsány v kapitole 4.1.2.1. V případě synchronní komunikace jsou všechny události vykonávány v hlavním vlákně a pokud by některá z událostí trvala déle, může nastat problém, kdy systém nebude reagovat na nové události včas. Proto mezi hlavní výhody asynchronní komunikace patří vyšší výkon a responzivita systému.

4.1.1 Komunikace mezi telefon a senzorem

Komunikace mezi senzorem a telefonem (znázorněna na obrázku 3) je založena na *Representational State Transfer* (dále REST) architektuře, která umožňuje klient/server komunikaci. V tomto případě je klientem telefon, který žádá senzor (server) o informace - konkrétněji žádá přímo jednotlivé služby z Whiteboard. Každá služba zprostředkovává svůj zdroj dat pomocí unikátního identifikátoru tj. cesty ke zdroji. REST rozhraní pak umožňuje jednoduchou práci se zdroji dané služby pomocí čtyř základních metod:

- **GET** (Retrieve) je základní metoda pro získaní zdroje.
- **POST** (Create) se používá k vytvoření dat.



Obrázek 3: Komunikace mezi senzorem a mobilní aplikací [6]

- **DELETE** se využívá ke smazání zdroje.
- **PUT** (Update) se používá k aktualizaci zdroje.

Tímto vzniká komunikační protokol s dvěma důležitými vlastnostmi:

- **Klient/server** architektura sloužící k oddělení odpovědnosti.
- Každý požadavek je **bezstavový**, musí tedy obsahovat vše potřebné ke svému vykonání.

V obecném případě vzniká více vlastností komunikačního protokolu, jako je například **cache**. Každý požadavek může být explicitně označený jako cachovatelný/necachovatelný, čímž se může drasticky snížit vytížení serveru. To však nelze použít v tomto případě, kde server je senzor a data se nepřetržitě mění. [7]

MDS definuje navíc metodu subscribe (odebírání), která se v běžné REST architektuře nepoužívá. Interně se jedná pouze o zavolání speciální POST metody, která spustí dlouhodobé odebírání dat. Subscribe umožňuje získávat data ze zdroje buď v pravidelných intervalech, nebo pokud dojde ke změně (například při detekci kroku). S touto metodou je i úzce spojena metoda unsubscribe, která odebírání dat ukončí. V tomto případě se jedná o zavolaní běžné REST metody DELETE. Volající je zodpovědný za zavolání této metody.

4.1.2 Struktura mobilní knihovny MDS

MDS se skládá ze dvou hlavních částí. První část nabízí API, které slouží k připojení senzoru k telefonu (dále MDS connectivity API). Druhá část nám nabízí

již výše zmíněné REST API pro komunikaci se senzorem (dále MDS REST API). Jelikož se v obou případech jedná o asynchronní volání výsledky jsou vráceny v tzv. zpětných voláních (callbacks).

4.1.2.1 Zpětné volání

Zpětné volání je jakýkoliv spustitelný kód, který je předán jako argument funkci s očekáváním, že bude někdy v budoucnu spuštěn (například jakmile budou data k dispozici). Obecně existují dva typy zpětných volání:

- **Blokující**, které je zavoláno před dokončením volající funkce.
- **Neblokující**, u kterého není určena doba, kdy bude zavoláno. Často se využívá v I/O operacích.

MDS využívá neblokující zpětná volání, tudíž je možné si představit senzor jako počítač a jeho hardwarová zařízení jsou dílčí senzory (akcelerometr, gyroskop atd.). V době, kdy jeden z těchto dílčích senzorů má k dispozici nová data, je vyvoláno přerušení, které je následně obslouženo.

4.1.2.2 MDS connectivity API

MDS connectivity API je první částí mobilní knihovny. Hlavním cílem tohoto API je zajištění připojení a následné odpojení od zařízení. Publikuje tedy dvě metody, které je možné vidět ve zdrojovém kódu 1.

```
1 public void connect(@NonNull String deviceAddress, @Nullable final
     MdsConnectionListener callback);
2 public void disconnect(@NonNull String deviceAddress);
```

Zdrojový kód 1: Metody pro připojení a odpojení od zařízení

Knihovna MDS nenabízí žádný způsob, jak zjistit MAC adresu Movesense senzoru, která je potřeba k jeho připojení. Proto k jejímu nalezení byla využita knihovna RxAndroidBle, která je založena na RxJave. RxJava spadá do reaktivního programování, které je orientované kolem datových toků a šíření změn.

Například v imperativním programování by nastavení $a := b + c$ znamenalo, že do a je přiřazen výsledek $b+c$ v okamžiku, kdy je výraz vyhodnocen a později lze hodnoty b a c změnit bez vlivu na hodnotu a. Nicméně v reaktivním programování se hodnota a automaticky aktualizuje vždy, když se změní hodnoty b a c, bez nutnosti znovu vykonávat výraz $a := b + c$. [8]

RxJava tohoto programovacího paradigma využívá a implementuje ho pomocí tzv. **observer pattern**. Observer pattern je návrhový vzor, který definuje dva důležité pojmy:

- **Vydavatel** (pozorovaný) je zdrojem dat.

- **Předplatitel** (posluchač) odebírá zdroj dat.

Mezi předplatiteli a vydavateli zavádí vztah N:1 (jeden vydavatel má mnoho předplatitelů). V případě, že na straně vydavatele dojde ke změně vztahu či výskytu nějaké události, upozorní na to automaticky všechny své předplatitele. [9]

Tohoto principu pak využívá výše zmíněná knihovna RxAndroidBle, kterou aplikace využívá k nalezení všech Movesense senzorů v okolí. Příklad, jak nalézt Bluetooth zařízení v okolí je možné vidět ve zdrojovém kódu 2. Tímto jakákoliv změna (například objevení nového zařízeni) bude automaticky informovat všechny předplatitele a ti na ni mohou reagovat.

```
Disposable scanSubscription = rxBleClient.scanBleDevices(
      new ScanSettings.Builder()
         .build()
      // možnost přidat filtr
)
   .subscribe(
      scanResult -> {
         // Zpracuj zde výsledek hledání
      },
      throwable -> {
         // zpracuj error zde. Např. není zaplý bluetooth na
            telefonu
      }
   );

// Pro ukončení hledání
scanSubscription.dispose();
```

Zdrojový kód 2: Nalezení všech bluetooth zařízení v okolí

Jak již bylo zmíněno, MDS connectivity API je založeno na asynchronním volání, proto je možné získávat průběžné informace ohledně připojení nebo odpojení skrze zpětné volaní, které je ukázáno ve zdrojovém kódu 3. Metoda *onConnectionComplete* nám vrací hodnotu *serial*, což je sériové číslo senzoru, které slouží pro komunikaci se senzorem.

4.1.2.3 MDS REST API

MDS REST API je druhou částí mobilní knihovny. Hlavním cílem tohoto API je zajistit komunikaci se senzorem. Jedná se především o možnost poslat naměřená data senzorem do telefonu. MDS REST API publikuje dohromady pět metod pro komunikaci se senzorem (viz zdrojový kód 4) a dvě zpětná volání. První zpětné volaní je určeno pro základní REST metody, druhé zpětné volání je určeno pro metodu subscribe, která je detailně popsána v této kapitole.

První čtyři metody patří mezi základní REST metody. Parametr URI v těchto metodách je cesta ke zdroji (službě). Skládá se z prefixu, sériového čísla senzoru a

```
1  public interface MdsConnectionListener {
2
3      /**
4       * Zavoláno pokud připojení mezi Mds a Whiteboard vrstvou
5       * proběhlo úspěšně
6       */
7      void onConnect(String macAddr);
8
9      /**
10      * Zavoláno pokud došlo k úplnému připojení k senzoru
11      *
12      */
13     void onConnectionComplete(String macAddr, String serial);
14
15     /**
16      * Zavoláno pokud došlo k chybě během připojení
17      *
18      */
19     void onError(MdsException error);
20
21     /**
22      * Zavoláno pokud bylo zařízení odpojeno
23      *
24      */
25     void onDisconnect(String macAddr);
26 }
```

Zdrojový kód 3: Zpětné volání pro connectivity API

cesty ke službě. Parametr *contract* obsahuje data potřebná k odeslání na senzor, takže například u metody GET, bude tento parametr prázdný řetězec. Posledním parametrem je pak zpětné volaní, které je možné vidět ve zdrojovém kódu 5.

Poslední metodou je speciální metoda *subscribe*. Má své vlastní zpětné volání, které je velice podobné zpětnému volání předešlých metod. I přesto, že jsou parametry metod stejné, jejich použití se liší. Parametr URI je v tomto případě vždy *suunto://MDS/EventListener*, *contract* je objekt ve formátu JSON[5], který se skládá ze sériového čísla senzoru, cesty ke službě a frekvence. Frekvence udává, jak často budou data posílána ze senzoru. Většina služeb, které frekvenci podporují, ji mají předem definované. Posledním argumentem je opět zpětné volání.

Všechny základní služby o sobě poskytují informace. Ty obsahují zmíněné podporované frekvence nebo také rozsah citlivosti (senzitivity). Nastavení správné citlivosti záleží na pohybu, který je měřen. Pokud je zapotřebí měřit drobné vibrace je nutné nastavit vysokou citlivost, naopak při měření vysokých amplitud, je vhodná nižší citlivost. Tyto informace je možné získat metodou GET na speci-

[5]JavaScript Object Notation (JSON) je způsob zápisu dat (datový formát) nezávislý na počítačové platformě, určený pro přenos dat, která mohou být organizována v polích nebo agregována v objektech. [10]

```
1     public void get(@NonNull String uri, String contract,
          MdsResponseListener callback);
2     public void put(@NonNull String uri, String contract,
          MdsResponseListener callback);
3     public void post(@NonNull String uri, String contract,
          MdsResponseListener callback);
4     public void delete(@NonNull String uri, String contract,
          MdsResponseListener callback);
5
6     public MdsSubscription subscribe(@NonNull String uri, String
          contract, MdsNotificationListener listener);
```

Zdrojový kód 4: Metody pro komunikaci se senzorem

```
1     public interface MdsResponseListener {
2
3     /**
4      * Zavoláno, jakmile volaní metody úspěšně skončilo
5      *
6      * @param výsledek volání
7      */
8     void onSuccess(String data);
9
10    /**
11     * Zavoláno, pokud volání metody skončilo chybou
12     *
13     * @param objekt obsahující error
14     */
15    void onError(MdsException error);
16 }
```

Zdrojový kód 5: Zpětné volání pro MDS REST API

fickou cestu (příklad cesty pro akcelerometr: *Meas/Acc/Info*). Příklad odpovědi je vidět ve zdrojovém kódu 6.

4.2 Senzorová knihovna

Senzorová knihovna slouží jako API pro tvorbu vlastních modulů, které mohou obsahovat uživatelsky definované služby. V první části kapitoly je popsána souborová struktura Movesense firmwaru, která úzce souvisí s definicí vlastních spustitelných modulů. Následuje detailnější popis Whiteboard a na závěr jsou popsány uživatelsky definované služby.

4.2.1 Souborová struktura Movesense firmwaru

Movesense firmware se skládá z pěti základních komponent:

```
{
  "Content":{
    "SampleRates":[
      13,
      26,
      52,
      104,
      208,
      416,
      833,
      1666
    ],
    "Ranges":[
      2,
      4,
      8,
      16
    ]
  }
}
```

Zdrojový kód 6: Odpověď senzoru na požadavek o informace k akcelerometru. „SampleRates“ jsou dostupné frekvence akcelerometru a „Ranges“ jsou dostupné citlivosti

- **App.cpp** je hlavním konfiguračním souborem firmwaru. Mohou zde být vypínány/zapínány volitelné moduly, definují se zde metadata nebo se zde registrují nové uživatelsky definované služby (viz zdrojový kód 7).
- **app_root.yaml** je konfiguračním souborem pro Whiteboard.
- **CMakeLists.txt** obsahuje skript na build (sestavení) celého firmwaru.
- **wbresources** je složka, obsahující soubory ve formátu YAML[6], které obsahují definice API pro uživatelsky definované služby.
- **movesense-device-lib** je hlavní knihovnou, která zprostředkovává základní funkčnost celého modulu.

4.2.1.1 Whiteboard

Whiteboard je klíčová knihovna nejen pro vnější komunikaci, ale i pro vnitřní. Její architektura je založena na tzv. mikroslužbě.

[6]YAML (YAML Ain't Markup Language) je formát pro serializaci strukturovaných dat. [11]

```
#include "PedometerService.h"
#include "movesense.h" // hlavní knihovna firmwaru
...

MOVESENSE_PROVIDERS_BEGIN(1)
// Uživatelsky definovaná služba
MOVESENSE_PROVIDER_DEF(PedometerService)

MOVESENSE_PROVIDERS_END(1)

MOVESENSE_FEATURES_BEGIN()
// Možnost vypínat/zapínat hlavní moduly Movesense firmwaru
OPTIONAL_CORE_MODULE(DataLogger, true)
OPTIONAL_CORE_MODULE(Logbook, true)
OPTIONAL_CORE_MODULE(LedService, true)

...

// Meta data firmwaru
APPINFO_NAME("Pedometer");
APPINFO_VERSION("1.1.1");
APPINFO_COMPANY("Movesense");
```

Zdrojový kód 7: Ukázka souboru App.cpp

Definice 1 (Mikroslužby)
Mikroslužby (případně mikroservisy) je jedna z variant softwarové architektury orientované na služby, kde jsou aplikace definovány jako soubor volně provázaných služeb. [12]

V tomto případě jsou jednotlivé dílčí senzory vlastní služby, které zprostředkovávají data Whiteboard. Tato data je pak možné využít pro tvorbu složitějších služeb (vnitřní komunikace) nebo je lze posílat přes Bluetooth do telefonu (vnější komunikace).

Whiteboard je de facto „mozkem“ veškeré komunikace. Pokud je vytvořena uživatelsky definovaná služba, musí být zaregistrována do Whiteboard, aby k ní mohlo být přistoupeno, tedy aby mohla být součástí vnitřní nebo vnější komunikace.

4.2.2 Uživatelsky definované služby

Uživatelsky definované služby, často nazývané uživatelsky definované moduly či pouze moduly, jsou hlavním důvodem, proč tvořit vlastní firmware. Je obecně nesprávný postup posílat surová data ze senzoru do telefonu a zde následně s těmito daty pracovat. Hlavním důvodem je fakt, že frekvence dat může být i 1024 Hz, což by mohlo zahltit Bluetooth zásobník telefonu a tímto by mohla být narušena komunikace se senzorem.

4.2.2.1 Životní cyklus modulů

Každý modul, uživatelsky definovaný nebo volitelný, má životní cyklus. Skládá se ze čtyř fází:

- **Inicializace modulu** je stav, kde jsou inicializovány hardwarové konfigurace. V této fázi nesmí být zažádáno o žádné služby z Whiteboard (například data z akcelerometru).
- **Spuštění modulu** je stav, kde se ve většině případů spouští hlavní logika modulu. Je tedy možné žádat o služby z Whiteboard.
- **Zastavení modulu** je stav, kdy spouštění modulu bylo zamítnuto.
- **Deinicializace modulu** je stav, kdy se například senzor připravuje k vypnutí. Jsou zde uvolněny všechny zdroje.

Všemi těmito stavy může modul procházet například při zapnutí a poté při vypnutí senzoru. Tyto stavy jsou důležité pro tvorbu uživatelsky definovaných služeb, jelikož odpovídají metodám, které musí programátor přepsat pro každou uživatelsky definovanou službu. Seznam metod je možné vidět ve zdrojovém kódu 8.

```
1    virtual bool initModule() OVERRIDE;
2    virtual void deinitModule() OVERRIDE;
3    virtual bool startModule() OVERRIDE;
4    virtual void stopModule() OVERRIDE;
```

Zdrojový kód 8: Hlavní metody uživatelsky definovaného modulu

4.2.2.2 ResourceProvider

ResourceProvider, poskytovatel zdrojů, je rozhraní pro tvorbu uživatelsky definovaných služeb. Každý modul může mít těchto služeb několik. S tímto je úzce spojena složka wbresources viz 4.2.2.3, kde je v každém YAML souboru popsáno API dané služby.

Nejprve je nutno dané služby zaregistrovat. Registrace služeb probíhá při inicializaci modulu, tedy v metodě *initModule*. Pokud má modul více služeb, je dobré tyto služby mít v poli a registrovat je najednou (viz zdrojový kód 9).

Při deinicializaci modulu je ovšem nutno tyto služby odregistrovat, aby již nebyly nabízené.

```
1  bool TestService::initModule()
2  {
3     // sProvederResources je pole ID služeb k registraci
4     if (registerProviderResources(sProviderResources) != whiteboard::
          HTTP_CODE_OK)
5     {
6        mModuleState = WB_RES::ModuleStateValues::INITFAILED;
7        return false;
8     }
9
10    mModuleState = WB_RES::ModuleStateValues::INITIALIZED;
11    return true;
12 }
```

Zdrojový kód 9: Příklad inicializace pole služeb

4.2.2.3 Wbresources

Aby bylo možné službu zaregistrovat je potřeba definovat její API. Touto definicí je pak určeno, které REST metody Whiteboard dané službě publikuje. Tyto definice uživatelských služeb jsou ve složce wbresources. Každá služba a její zdroje jsou definovány v separátním YAML souboru. Pro definici se využívá **Swagger**.

Swagger je sada nástrojů (framework) pro tvoru REST API. Výhodou Swaggeru je jeho jednoduchost, možnost testování existujícího API a automatická tvorba dokumentace (obr. 4).

Každý YAML soubor obsahuje základní informace o daném API. Mezi tyto informace patří název služby, verze, podmínky využití služby atd. Poté se již definují jednotlivé REST metody. Každá metoda musí mít definovanou cestu a možné odpovědi, u kterých je nutné definovat jejich schéma, tedy návratový typ odpovědi.

Pokud je definována konkrétní REST metoda, je nezbytné v modulu přepsat (override) tomu odpovídající metodu, která obstará výsledek tohoto volání. Seznam jednotlivých metod k přepsaní je ve zdrojovém kódu 11.

Speciální případ nastává, pokud uživatelsky definovaná služba definuje metodu subscribe. Zde je potřeba přepsat tomu určené metody popsané ve zdrojovém kódu 12. Ve většině případů se v těchto metodách spouští další logika služby. Mohlo by se jednat o detekci kroků, které chceme detekovat pouze v případě, že někdo tento zdroj odebírá. Pak je v metodě *onSubscribe* tato logika spuštěna a v metodě *onUnubscribe* je zastavena a jsou uvolněny všechny zdroje, které byly využity.

Pokud je k dispozici nová hodnota, například detekován krok, je nutné informovat všechny předplatitele pomocí metody *updateResource*. Tímto se docílí minimální vytíženosti Bluetooth (jsou přenášena pouze potřebná data a to ve chvíli, kdy jsou data k dispozici).

```
paths:
// cesta ke zdroji (URI)
  /Sample/Pedometer/StepCount:
  // typ HTTP metody
    get:
      description: |
        Get current number of steps
      responses:
      // úspěšná odpověď'
        200:
          schema:
             description: Return number of steps
             type: integer
             format: uint16

        204:
        // neúspěšná odpověď'
          description: Measurement source is unavailable
```

Zdrojový kód 10: Příklad definice GET metody pro zjištění aktuálního počtu kroků

```
virtual void onGetRequest(const Request& rRequest, const
    ParameterList& rParameters)
virtual void onPutRequest(const Request& rRequest, const
    ParameterList& rParameters)
virtual void onPostRequest(const Request& rRequest, const
    ParameterList& rParameters)
virtual void onDeleteRequest(const Request& rRequest, const
    ParameterList& rParameters)
```

Zdrojový kód 11: Metody obstarávají výsledek jednotlivých REST metod

4.2.2.4 ResourceClient

K vytvoření nové služby je ve většině případů zapotřebí již existujících zdrojů (akcelerometr, gyroskop atd.). ResourceClient je rozhraní zprostředkovávající již existující zdroje. Všechny zdroje mají své unikátní ID. Aby bylo možné získat data z již existujících zdrojů, je zapotřebí požádat o toto ID metodou *getResource* (zdrojový kód 13).

Po obdržení ID je možné s daným zdrojem pracovat pomocí následujících metod:

- **asyncGet**
- **asyncPut**
- **asyncPost**

```
1 virtual void onSubscribe(const Request& rRequest, const
    ParameterList& rParameters)
2 virtual void onUnubscribe(const Request& rRequest, const
    ParameterList& rParameters)
```

Zdrojový kód 12: Metody obstarávají výsledek metody subscribe, unsubscribe

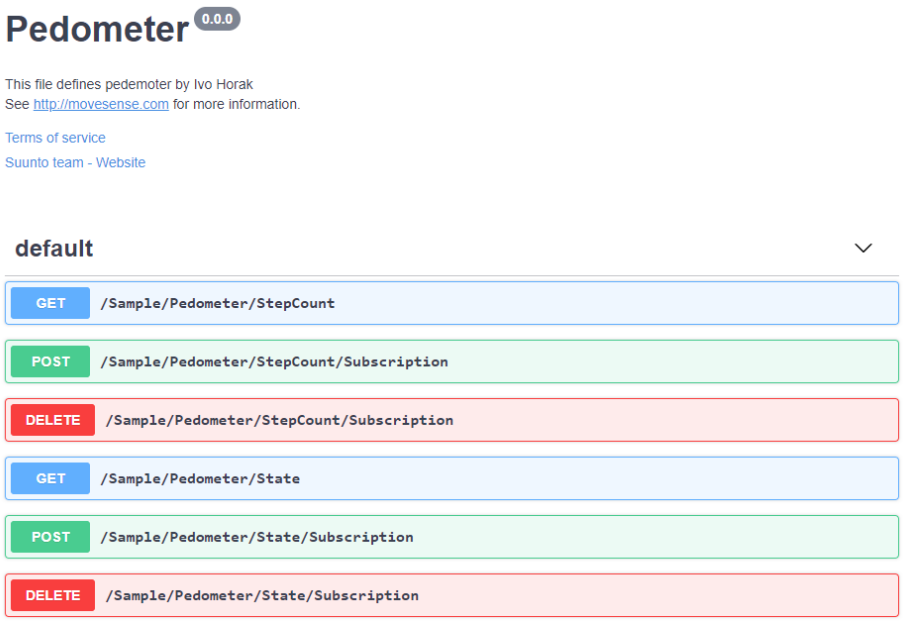
```
1 Result getResource(const char* pFullPath, ResourceId& rResourceId,
    const RequestOptions& rOptions = RequestOptions::Empty)
```

Zdrojový kód 13: Metoda pro získaní ID

- **asyncDelete**
- **asyncSubscribe**

Všechny tyto metody jsou asynchronní, tedy výsledek je obdržen ve zpětném volání. Každá z výše uvedených metod má své vlastní zpětné volání.

Pro ukončení využívaní zdrojů je nutné zdroje uvolnit metodou *releaseResource*.



Obrázek 4: Automaticky vygenerovaná dokumentace k firmwaru pro senzor. API definuje dvě metody GET pro získaní aktuálních kroků nebo stavu a dále definuje metody subscribe resp. unsubscribe

5 Platforma Android

V této kapitole je popsána platforma Android, od její historie až po architekturu systému a aplikací. Na závěr je popsán balíček zvaný Jetpack, který je dnes velice populárním. Informace převážně čerpány z [13] a [14].

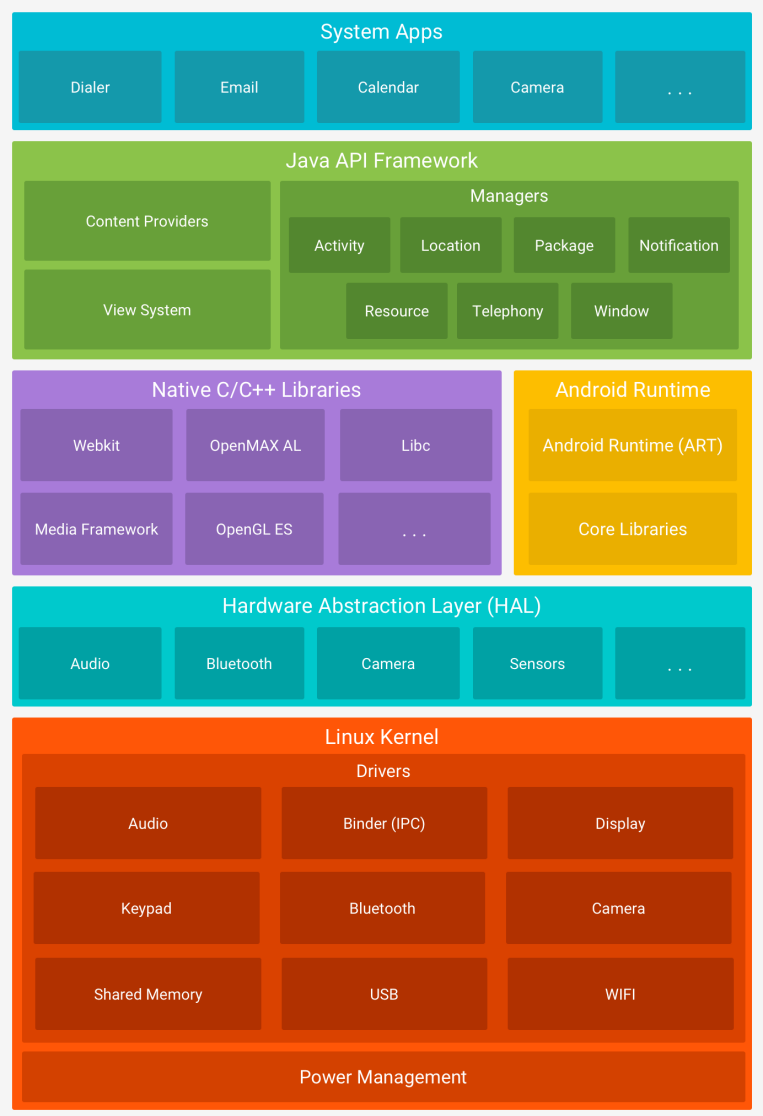
5.1 Historie

Společnost Android Inc. byla založena v říjnu roku 2003. Google Inc. ji v srpnu 2005 odkoupil a udělal z ní svou dceřinou společnost a pod vedením Andyho Rubina byla vyvinuta platforma založená na Linuxovém jádře. V září 2007 Google získal několik patentů v oblasti mobilních technologií a v listopadu téhož roku bylo vytvořeno konsorcium pod jménem Open Handset Alliance, které zahrnovalo mnoho společností zabývající se vývojem telefonů, čipů nebo mobilních aplikací. Těchto společností bylo dohromady 34 a mezi nejznámější patří Qualcomm, HTC, Intel, Samsung nebo LG. V den založení konsorcia byl vydán i její první produkt Android, otevřená mobilní platforma postavená na Linuxovém jádře verze 2.6. O týden později byl vydán první Android SDK pro vývojáře pod licencí open-source. [15]

5.2 Architektura systému

Architektura systému se skládá ze šesti základních komponent, které lze vidět na obrázku 5. Tyto části jsou stručně popsány v následujících bodech:

- **Linuxové jádro** zajišťuje základní funkcionalitu platformy Android. Patří zde například podpora hardwarových ovladačů, zabezpečení nebo správa paměti.
- **Hardware Abstraction Layer (HAL)** je abstraktní vrstva zprostředkovávající hardwarové možnosti zařízení nadřazenému aplikačnímu rozhraní (Java API Framework).
- **C/C++ knihovny** zahrnující například libc, SSL, SQLite nebo OpenGL.
- **Android Runtime** (dále ART) je hlavním důvodem, proč Android není jen běžnou implementací Linuxu pro telefony. Spolu se základními knihovnami, které nabízejí funkcionalitu, jak běžných Java knihoven, tak i specifickou pro Android, ART tvoří základní aplikační rozhraní, které pohání všechny aplikace.
- **Aplikační rozhraní** nabízí základní třídy k tvorbě aplikací.
- V **aplikační vrstvě** běží jak nativní aplikace, tak i aplikace třetích stran. Všechny aplikace jsou si rovny, jelikož využívají stejných knihoven, které nabízí aplikační rozhraní.



Obrázek 5: Architektura systému Android [13]

5.3 Architektura aplikace

Každá aplikace se skládá z několika znovupoužitelných komponent. Výhodou tohoto přístupu je, že aplikace mezi sebou mohou komponenty sdílet a tudíž je i využívat. Proto je například možné si vybrat jakou aplikaci chceme použít pro zobrazení webu (jestli defaultní prohlížeč nebo prohlížeč třetí strany).

Následující systémové komponenty jsou základními stavebními kameny každé aplikace:

- **Activity a Fragment Manager** řídí životní cyklus aktivit a fragmentů. Aktivity a fragmenty reprezentují danou „obrazovku“ aplikace. Jsou de facto kontejnery pro dílčí komponenty, které tato obrazovka obsahuje.

- **Views** jsou dílčí komponenty aktivit a fragmentů, které se používají na tvorbu uživatelského rozhraní.
- **Notification Manager** slouží k zobrazování notifikací pro uživatele v horní liště telefonu.
- **Content Providers** slouží ke sdílení dat mezi aplikacemi.
- **Resource Manager** složí k získání mimo kódových zdrojů jako jsou lokalizované řetězce nebo grafické rozložení (layout) fragmentů/aktivit.

5.4 Aktivity a fragmenty

Aktivita je základním stavebním prvkem pro vizuální část aplikace. Slouží jako vstupní bod pro interakci uživatele s aplikací. Z aplikace mohou být spuštěny jak aktivity patřící dané aplikaci, tak i aktivity jiných aplikací. Pokaždé, když je nová aktivita spuštěna, stará je zastavena a vložena na zásobník (back-stack). Tento zásobník pak slouží pro zpětnou navigaci, tzn. pokud uživatel zmáčkne tlačítko zpět, je aktivita ležící na vrcholu zásobníku obnovena. [16]

5.4.1 Fragment

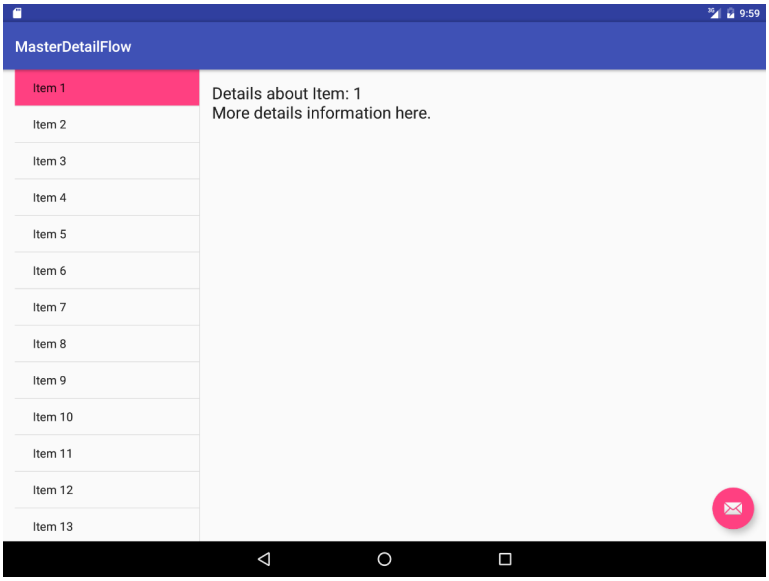
Logický celek, výseč uživatelského rozhraní, část aktivity. Fragmenty vznikly jako reakce na příchod zařízení s větším displejem (tablety). Většina uživatelských rozhraní se nedá použít na tablety, jelikož buďto vzniká UI s nevyužitým prostorem nebo jednotlivé prvky jsou roztaženy přes celý displej zařízení. Fragmenty umožnily tvorbu menších logických celků, které následně mohly být umístěny do aktivit nebo znovupoužity v jiných místech aplikace. Dobrým příkladem je tzv. master/detail flow (viz obr. 6), kde na telefonu by se pro detail položky a seznam položek použily dvě aktivity, ale na tabletu je vhodnější použít jednu. [16]

Fragmenty jsou de facto „light-weight“ aktivitami. Ačkoliv vznikly pro komfortní vyvíjení pro tablety, v dnešní době se ve velké míře používají i pro vývoj na telefony z důvodu jejich nenáročnosti na zdroje a zároveň velice jednoduchou komunikací mezi sebou (mají sdílenou aktivitu). Tímto vznikla nová architektura pro tvorbu aplikací nazývaná **single-activity application**.

5.4.2 Single-activity application

Jak je zřejmé z názvu, single-activity application má pouze jednu aktivitu, která slouží jako kontejner pro fragmenty. V tomto případě fragmenty reprezentují jednotlivé obrazovky aplikace (stále ale platí, že aktivita se může skládat z více fragmentů). Tímto vzniká architektura s mnoha výhodami:

- Výrazně menší náročnost na zdroje - zničení a vytvoření fragmentu (přechod na jinou obrazovku) je podstatně levnější než u aktivit.



Obrázek 6: Master/detail flow aktivita [17]

- Komunikace mezi obrazovkami a sdílení dat je mnohem jednodušší než v případě aktivity.
- Navigace mezi obrazovkami je velice jednoduchá využitím Navigation component (viz kapitola 5.5.3).

V roce 2018 se tato architektura stala doporučovanou pro tvorbu aplikací pro Android OS a vzniklo mnoho komponent, které vývoj pro tuto architekturu značně ulehčují. Mnoho z těchto komponent patří do Jetpack balíčku, který je popsán v následující kapitole.

5.5 Jetpack balíček

Jetpack balíček je odpověď Googlu na dlouhodobé problémy, které se při vyvíjení aplikací pro Android objevovaly. Jedná se především o problémy jako například správa životního cyklu aktivit a fragmentů, prevence úniku paměti nebo zachování dat při změně konfigurace (rotace displeje).

Jetpack je sada komponent, nástrojů a pokynů pro tvorbu robustních a dobře testovatelných aplikací. Výhodou je jeho modularita, tudíž není potřeba použít celý balíček, ale pouze části (komponenty), pro které se programátor rozhodne. Všechny komponenty jsou navíc zpětně kompatibilní.

Jetpack balíček je velmi obsáhlý, proto jsou zde popsány pouze části, které byly využity pro tvorbu aplikace.

5.5.1 LiveData

LiveData jsou jednou z hlavních součástí Jetpack balíčku. Využívají již zmíněného observer pattern (popsán v kapitole 4.1.2.2), to znamená, že spadají do

reaktivního programování.

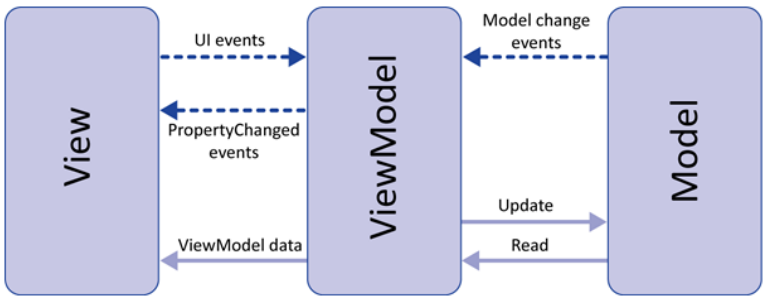
V tomto případě je **předplatitel** pouze jeden a to daný fragment nebo aktivita, **vydavatelem** pak mohou být například lokální (databáze) nebo vzdálené (internetové) zdroje. Hlavní výhodou LiveData je, že jsou si vědoma životního cyklu aktivit nebo fragmentů, tedy po jejich skončení (například přechod na jinou obrazovku) je automaticky ukončeno odebírání dat - nemůže dojít k úniku paměti. V tomto ohledu se například liší od RxJava, kde za uvolnění zdrojů zodpovídá programátor a proto k úniku paměti dojít může.

5.5.2 Návrhový vzor MVVM

MVVM, neboli Model-View-ViewModel, je návrhový vzor, kolem kterého je mnoho Jetpack komponent stavěno a dohromady tvoří dobře fungující celek. Jeho hlavní úlohou je oddělení logiky aplikace od uživatelského rozhraní. MVVM tedy odděluje data, stav aplikace a uživatelské rozhraní a tak tvoří kratší, přehlednější a dobře udržitelný kód.

MVVM se skládá z následujících částí (obrázek 7):

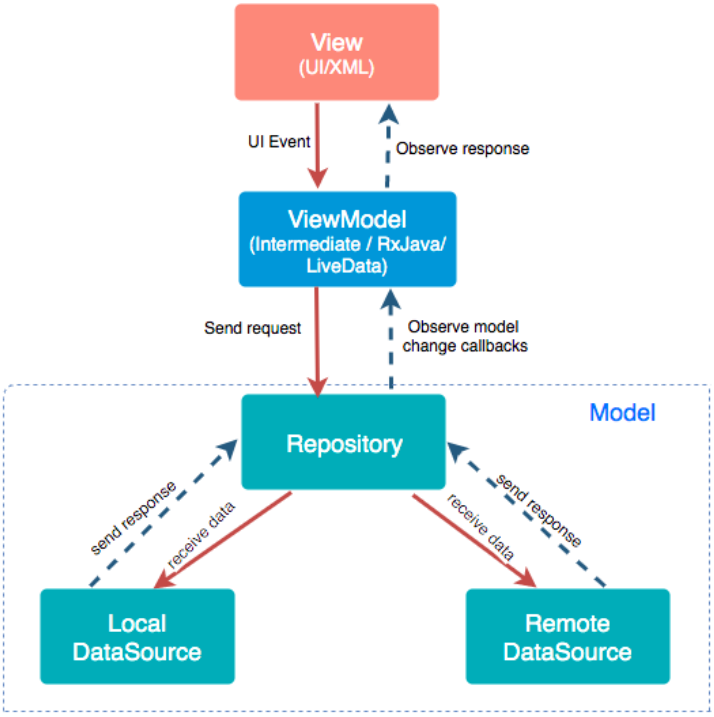
- **Model** obsahuje veškerou logiku a data, se kterými aplikace pracuje.
- **View** je vrstva uživatelského rozhraní.
- **ViewModel** spojuje obě předchozí vrstvy, tedy připravuje data z modelové vrstvy pro zobrazení v UI a na druhé straně reaguje na změny v UI, které propisuje zpět do Modelu.



Obrázek 7: Návrhový vzor MVVM [18]

Pokud by byl tento model přímo aplikován na Android aplikaci, tak struktura by mohla vypadat následovně (obrázek 8). Do **modelu** by patřil repozitář (lokální a vzdálené zdroje) a POJO[7] třídy daných dat. **ViewModel** je třída, která data z modelu zpracovává a vytváří z nich LiveData, která pak publikuje. Do **View** pak patří jednotlivé aktivity a fragmenty, které odebírají data z ViewModelu. [20]

[7]POJO (Plain Old Java Objects) jsou jednoduché objekty, které nejsou svázané technologicky specifickým rozhraním. Používají se pro uchování strukturovaných dat. [19]



Obrázek 8: MVVM architektura aplikovaná na Android aplikaci [13]

Tímto vzniká architektura, kde uživatelské rozhraní nemá žádné informace odkud data pocházejí a plní pouze jediný účel a to je zobrazovat. Je důležité si uvědomit, že tato data nejsou držena v uživatelském rozhraní a proto při změně konfigurace (rotace displeje) jsou pak k dispozici, jelikož ViewModel není závislý na životním cyklu aktivity (a tedy i fragmentu).

5.5.3 Navigation component

Navigation component je další komponentou z Jetpack balíčku. Vznikla hlavně z důvodu zjednodušení navigace mezi jednotlivými obrazovkami aplikace a velice dobře funguje se single-activity architekturou.

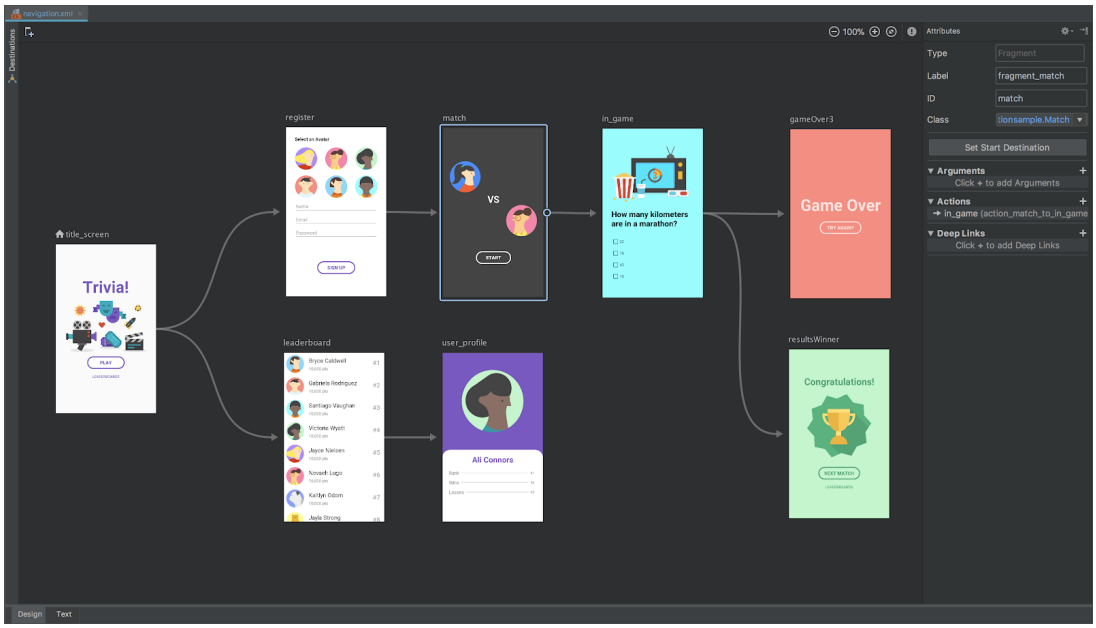
Skládá se z navigačního grafu, na kterém můžeme vidět všechny naše obrazovky (fragmenty). Pomocí grafického rozhraní (viz obr. 9) pak můžeme definovat kořenový fragment, tedy fragment, kterým aplikace bude začínat, a poté navigaci mezi jednotlivými fragmenty. Dále můžeme definovat argumenty, resp. data, která jsou mezi jednotlivými fragmenty posílána. Navigation component pak řeší transakce mezi jednotlivými fragmenty a také správu zásobníku (back-stack).

5.5.4 Room

Room je knihovna, která tvoří abstraktní vrstvu nad SQLite. Zprostředkovává jednoduchý přístup do databáze a zároveň zůstává stejně efektivní jako SQLite.

Mezi největší výhody patří:

- Kontrola SQL dotazů v době kompilace aplikace
- Snížení množství stále se opakujícího kódu tzv. boilerplate code
- Dobrá integrace s ostatními komponentami (LiveData)



Obrázek 9: Navigation component - grafické zobrazení [21]

Room se skládá ze tří základních komponent(viz obr. 10):

- Entity
- DAO (Data Access Objects)
- Database

5.5.4.1 Entity

Entity je třída reprezentující tabulku v databázi, která neobsahuje žádnou logiku (POJO třída). Pro každou třídu, která je anotována *@Entity*, Room vytvoří tabulku, kde jednotlivé atributy odpovídají sloupcům v tabulce. Tato anotace pak může obsahovat další specifikace jako jsou například indexy nebo cizí klíče. Dále pak každý atribut třídy může obsahovat vlastní anotace jako například:

- *@PrimaryKey* značí primární klíč tabulky.

- *@ColumnInfo* umožňuje definovat další informace o daném sloupci. Zde je možné specifikovat jméno sloupce, pokud není žádoucí, aby se jmenoval jako jeho odpovídající atribut.
- *@Ignore* je možnost, aby daný atribut byl ignorován, tedy aby se nevytvořil jeho odpovídající sloupec.
- *@Embedded* označuje atribut jako těsně vázaný na rodičovskou třídu. V SQL dotazu se lze odkazovat na atributy těsně vázaného atributu stejně, jako by byly součástí rodičovské třídy.

Příklad takové třídy je pak možné vidět ve zdrojovém kódu 14.

```
1  @Entity(tableName = "connected_devices",
2     foreignKeys = @ForeignKey(entity = Training.class, parentColumns
          = "id", childColumns = "training_id", onDelete = ForeignKey.
          CASCADE),
3     indices = {@Index(value = "training_id")})
4  public class ConnectedDevice {
5
6        @PrimaryKey(autoGenerate = true)
7        private long id;
8
9        @ColumnInfo(name = "heart_rate")
10       private int heartRate;
11
12    ...
13 }
```

Zdrojový kód 14: Příklad třídy reprezentující tabulku v Room databázi

5.5.4.2 DAO

DAO (Data Access Objects) je rozhraní (interface), kde jsou definovány metody, které se dotazují do databáze (viz. zdrojový kód 15). Jedná se tedy o vrstvu mezi aplikací a databází. Každá z definovaných metod buď nevrací nic, (například při aktualizaci hodnot tabulky) nebo vrací objekt třídy anotovaný anotací *@Entity*. DAO dále definuje tyto čtyři základní anotace pro práci s těmito objekty:

- *@Insert* slouží pro přidání nového záznamu do tabulky.
- *@Update* slouží pro aktualizování již existujícího záznamu.
- *@Delete* slouží pro smazání existujícího záznamu.
- *@Query* slouží pro definici vlastního dotazu do databáze.

```
1  @Dao
2  public interface ProfileDao {
3
4      @Insert
5      long insert(Profile profile);
6
7      @Delete
8      void delete(Profile profile);
9
10     @Query("select * from profile")
11     List<Profile> getAllProfiles();
12
13     @Query("select * from profile")
14     LiveData<List<Profile>> getAllProfiles();
15
16     ...
17 }
```

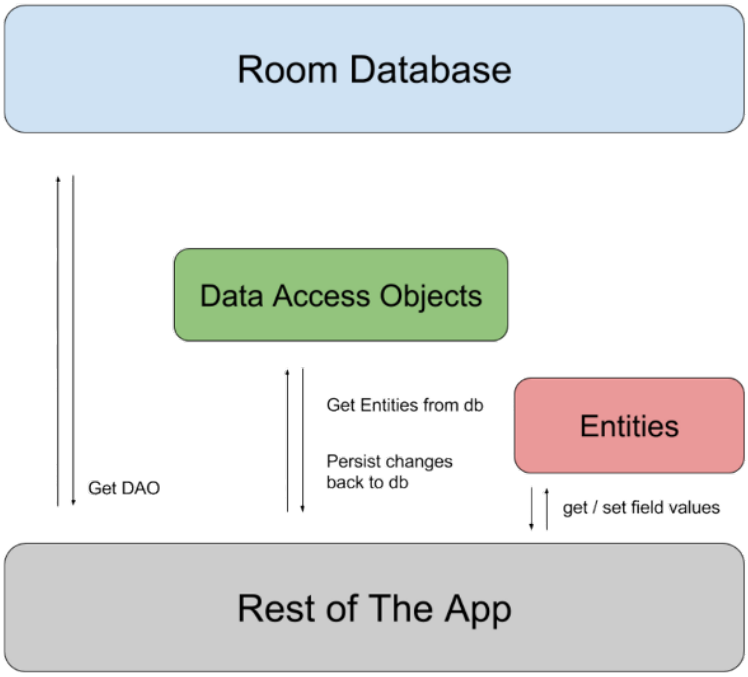
Zdrojový kód 15: Příklad DAO třídy definující metody pro práci s databází

Jak již bylo zmíněno, Room dobře spolupracuje s ostatními komponentami jako jsou například LiveData. Jednotlivé metody mohou navíc vracet i LiveData objekty, tedy objekty, které je možné pozorovat (viz kapitola 5.5.1). Je proto možné na jednom vlákně například přijímat data ze senzoru a ukládat je do databáze. Pokud jsou tato data pozorována pomocí aktivity nebo fragmentu, pak dochází k automatické notifikaci a data jsou okamžitě zobrazena.

5.5.4.3 Database

Poslední částí Room knihovny je Database. Database je abstraktní třída, která spravuje připojení s SQL databází a zároveň slouží jako hlavní přístupový bod do této databáze. Dále jsou v ní definovány všechny třídy, které patří do databáze a jsou anotovány pomocí *@Entity*. Dále obsahuje abstraktní metody pro získání DAO objektů.

Database pro programátora primárně slouží pro získaní DAO objektů pomocí kterých může pracovat s daty databáze.



Obrázek 10: Tří základní části knihovny Room [22]

6 Programátorská dokumentace

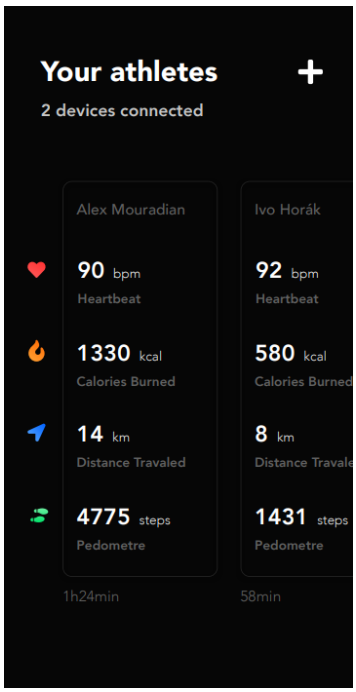
V této kapitole je popsána struktura projektu Android aplikace, jednotlivé třídy a matematický vzorec pro počítání kalorií. Dále budou ukázána uživatelská rozhraní, která byla navržena před vývojem aplikace. Bude popsán i firmware senzoru a algoritmus pro detekci kroku.

6.1 Uživatelské rozhraní

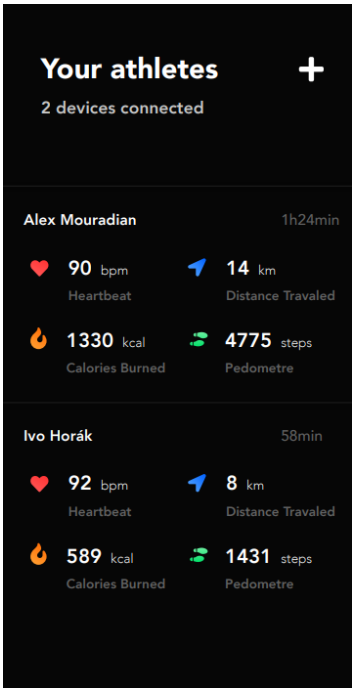
Před vývojem aplikace bylo navrženo několik základních vzhledů, jak by aplikace měla vypadat (viz obr. 11, 12, 13). Jednalo se primárně o přehled sportovců, jelikož je to jedna z nejdůležitějších obrazovek aplikace.

Přestože u varianty A je velmi příjemné, že ikonky daných informací se neopakují a na první pohled je zřejmé, jaký údaj co znamená, je zde problém v počtu sportovců, které je možné v danou chvíli zobrazit. Jak můžeme vidět na obrázku 11, lze zobrazit pouze dva sportovce aniž bychom museli listovat horizontálně. Proto tento design nebyl vhodný.

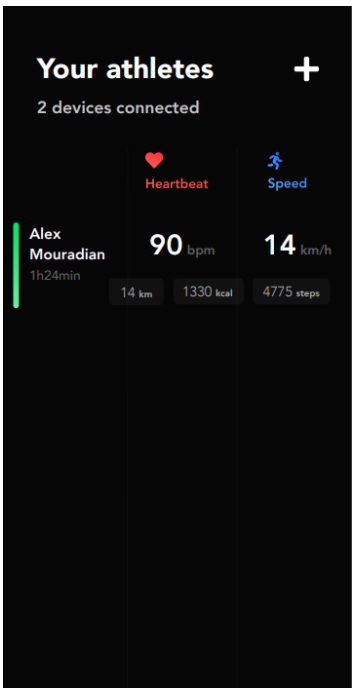
Varianty B a C pak sportovce zobrazují vertikálně a je možné zobrazit více sportovců najednou. Varianta C byla pokus o odstranění opakujících se ikonek, ale později bylo zjištěno, že tyto ikonky umožňují uživateli rychlejší orientaci na obrazovce a proto byly ponechány.



Obrázek 11: Varianta A

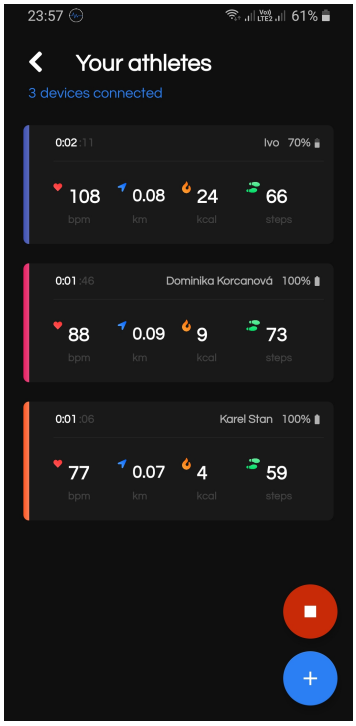


Obrázek 12: Varianta B



Obrázek 13: Varianta C

Po několika iteracích úprav byl navržen finální design aplikace, který umožňuje vidět až čtyři sportovce zároveň. Sportovci jsou od sebe barevně odlišeni (viz obr. 14).



Obrázek 14: Finální design aplikace

6.2 Struktura aplikace

Adresářová struktura projektu se skládá z následujících částí:

- **manifests** obsahuje soubor *AndroidManifest.xml*, který je hlavním konfiguračním souborem aplikace. Obsahuje informace jako název aplikace, její verzi, seznam všech aktivit nebo seznam všech systémových zdrojů, které aplikace používá.
- **java** je adresář, jehož součástí jsou všechny zdrojové kódy aplikace. Ty jsou pak dále rozděleny do podadresářů (packages).
- **java (generated)** obsahuje automaticky vygenerované třídy, které se generují například z externích knihoven, které jsou v projektu použity skrze anotace.
- **res (resources)** je adresář obsahující další zdroje pro aplikaci. Jedná se především o řetězce využívané v aplikaci, obrázky nebo XML rozložení (layout) daných fragmentů a aktivit.

6.2.1 Adresář res

Jak již bylo zmíněno, adresář *res* obsahuje všechny nekódové soubory. Je rozdělen do následujících podadresářů:

- **amin** je adresář, kde jsou definovány vlastní animace, které jsou použity při přechodu mezi jednotlivými obrazovkami.
- **drawable** obsahuje všechny grafické soubory jako jsou barevné pruhy v přehledu sportovců, ikony atd.
- **font** je adresář obsahující fonty.
- **layout** obsahuje grafické rozložení fragmentů a aktivit.
- **mipmap** je adresář, který obsahuje ikonky aplikace různých velikostí, které jsou použity například na domovské obrazovce telefonu nebo v menu aplikací.
- **navigation** je adresář obsahující *nav_graph.xml*, který je součástí Navigation component (popsána v kapitole 5.5.3) a definuje obecnou navigaci aplikace.
- **values** obsahuje několik dalších nekódových souborů. Nejdůležitější z nich jsou:
  - *colors.xml* obsahuje hexadecimální hodnoty všech barev, které jsou použity v aplikaci.
  - *dimens.xml* obsahuje nejdůležitější rozměry, které jsou použity napříč celou aplikací, například velikost jednotlivých nadpisů nebo velikosti ikonek.
  - *strings.xml* obsahuje lokalizované řetězce. Všechny řetězce v aplikaci jsou v samostatném souboru a proto rozšíření o další jazyk by znamenalo pouze vytvořit další soubor a všechny řetězce zde přeložit do příslušného jazyku.
  - *styles.xml* obsahuje základní definici témat, například hlavní téma, které se používá napříč celou aplikací. Mohou se zde definovat i menší témata například tlačítek nebo vstupních textů.

6.3 Popis tříd aplikace

Všechny třídy jsou uloženy adresáři *java*, kde jsou rozděleny do menších podadresářů. V této kapitole jsou popsány nejdůležitější z nich.

6.3.1 Aktivity

Jak již bylo zmíněno, aplikace využívá **single-activity architektury**. Celá aplikace má pouze jednu aktivitu a to MainActivity, která slouží pouze jako kontejner pro ostatní fragmenty.

6.3.2 Fragmenty

Každý fragment reprezentuje právě jednu obrazovku aplikace. Každá tato třída je potomkem třídy *Fragment* z AndroidX[8] knihovny. Níže je seznam všech fragmentů a jejich stručný popis:

- **BeginScreenFragment** je úvodní obrazovka s třemi navigačními tlačítky.
- **DashboardFragment** zobrazuje přehled všech připojených zařízení. Obsahuje dvě tlačítka pro připojení nového senzoru a pro dokončení tréninku.
- **GraphViewFragment** je obrazovka, která zobrazuje grafy konkrétního sportovce, které se v reálném čase aktualizují.
- **EditInfoFragment** slouží k editaci dat sportovce.
- **TrainingHistoryFragment** zobrazuje všechny již uskutečněné tréninky.
- **TrainingOverviewFragment** zobrazuje detailní informace o již uskutečněném tréninku.
- **HistoryGraphFragment** zobrazuje grafy konkrétního sportovce na daném tréninku. Jednotlivé grafy lze zobrazit přes celou obrazovku v samostatném fragmentu. Jedná se o fragmenty:
  - **HeartRateGraphFragment**
  - **StepsGraphFragment**
  - **DistanceGraphFragment**
  - **CaloriesGraphFragment**
- **ProfilesFragment** zobrazuje všechny definované profily.
- **EditProfileFragment** umožňuje editovat vybraný profil.
- **AfterConnectFragment** zobrazuje možnosti, jak zadat informace o sportovci po připojení senzoru k telefonu.
- **BasicInformationFragment** umožňuje zadat jméno a pohlaví sportovce. Dále umožňuje vybrat barvu, kterou sportovec bude reprezentován na tréninku.
- **DetailInformationFragment** umožňuje zadat detailnější informace o sportovci jako jsou váha, výška a věk. Tyto informace slouží k přesnějším výpočtům kalorií a uběhnuté vzdálenosti (viz kapitola 6.4).

[8] AndroidX knihovna je poměrně nová knihovna, která zastřešuje všechny komponenty z Jetpack balíčku. Také obsahuje například Support Library, která zajišťuje zpětnou kompatibilitu komponent.

Ke všem fragmentům existuje odpovídající **ViewModel**, který zpracovává data z repozitářů a publikuje do fragmentů jen ta, která jsou potřeba jako LiveData (viz kapitola 5.5.1). Fragmenty nemohou data nijak upravovat, ale pouze je odebírat a zobrazovat.

6.3.3 Databázové třídy

Jedná se o třídy, které slouží pro práci s databází Room. Tyto třídy je možné rozdělit do tří částí:

- **Modely** jsou třídy reprezentující jednotlivé tabulky databáze (jedná se o třídy anotované pomocí *@Entity* viz kapitola 5.5.4.1)
- **DAOs** jsou DAO třídy, které obsahují metody pro práci s databází (viz kapitola 5.5.4.2).
- **Repozitáře** jsou třídy, které tvoří abstraktní vrstvu mezi ViewModel a databází. ViewModels tedy nemají žádné informace odkud data pochazejí.

Konkrétní třídy rozdělené do zmíněných částí je možné vidět v tabulce 1.

Model	DAO	Repozitář
Profile	ProfileDao	ProfileRepository
ConnectedDevice	ConnectedDeviceDao	ConnectedDeviceRepository
Training	TrainingDao	TrainingRepository
AthleteData	AthleteDataDao	AthleteDataRepository

Tabulka 1: Rozdělení databázových tříd

6.3.4 Datové třídy

Datové třídy jsou POJO třídy, které se převážně vytváří z naměřených dat na senzoru. Instance těchto tříd vznikají konvertováním příchozích JSON řetězců na tyto datové třídy. Dále se datové třídy používají ke komunikaci se senzorem. Například pokud je potřeba zahájit odebírání dat, je vytvořen nový objekt zvaný *MdsSubscriptionURI*, který je následně konvertován do formátu JSON a odeslán na senzor. K těmto konverzacím se využívá populární knihovna **Gson** od společnosti Google. Níže je seznam všech datových tříd a jejich stručný popis:

- **EcgModel** reprezentuje naměřená EKG data.
- **HeartRateModel** reprezentuje naměřená data srdečního tepu.
- **LinearAccelerationModel** reprezentuje data naměřená z akcelerometru.
- **PedometerModel** reprezentuje data detekovaných kroků.
- **StateModel** reprezentuje data stavů, ve kterém se sportovec právě nachází.

6.3.5 Adaptéry

Adaptér je třída, která slouží k propojení seznamu dat s UI komponentou reprezentující seznam. Jedná se tedy o třídu, ve které je definováno, jak jednotlivé prvky seznamu budou vypadat a jaká data budou zobrazovat. Napříč celou aplikací se využíval tzv. RecyclerView, který je velice efektivní, jelikož již během listování jsou prvky, které nejsou vidět, recyklované a znovu použity na prvky, které je možné nově vidět (odtud název RecyclerView). Níže je seznam všech adaptérů a jejich stručný popis:

- **ConnectionAdapter** slouží k propojení seznamu všech zařízení, které se nachází v blízkosti telefonu s RecyclerView.
- **DashboardAdapter** propojuje všechny připojené sportovce a jejich informace jako jsou kroky, vzdálenost, spálené kalorie nebo srdeční tep s RecyclerView. Z důvodu, že tento seznam může být aktualizovaný i několikrát za sekundu, využívá se zde speciálního adaptéru tzv. ListAdapter. Od běžného adaptéru se liší v tom, jak jsou data aktualizována. V běžném adaptéru se po nastavení nového seznamu zavolá metoda *notifyDataSetChanged* a celý seznam se překreslí [9]. Tato metoda je velice náročná na zdroje a pokud bychom ji volali několikrát za sekundu, mohl by seznam začít problikávat nebo by se aplikace mohla začít sekat. ListAdapter využívá Eugene Myersova rozdílového algoritmu, který počítá nejmenší počet kroků, jak převést jeden seznam na druhý. Poté jsou aktualizované pouze prvky, které se opravdu změnily.
- **HistoryAdapter** slouží k propojení seznamu všech tréninků s RecyclerView.
- **ProfilesAdapter** slouží k propojení seznamu všech profilů s RecyclerView.
- **TrainingOverviewAdapter** slouží k propojení všech sportovců, kteří se účastnili vybraného tréninku s RecyclerView.

6.3.6 Dialogy

Pro výběr váhy, výšky a věku byly vytvořeny vlastní dialogy, které definují rozsah, ze kterého může uživatel vybírat a v případě váhy a výšky lze zvolit i jednotku. Níže je seznam všech dialogů se stručným popisem:

- **AgePickerDialog** slouží k vybrání věku.
- **WeightPickerDialog** slouží k vybrání váhy.
- **HeightPickerDialog** slouží k vybrání výšky.

[9] Existují i funkce jako například *notifyItemChanged, notifyItemRemoved* nebo *notifyItemInserted*, které nepřekreslují celý seznam, ale jen modifikovaný prvek.

6.3.7 Services

Během testování aplikace bylo zjištěno, že pokud uživatel déle jak 15 minut neinteraguje s aplikací (běží na pozadí nebo je telefon zamčen), tak Android OS automaticky aplikaci pozastaví. Tím způsobí odpojení všech senzorů a zastavení všech vláken, co aplikace využívala. Proto bylo nutné přepsat jádro aplikace do komponenty zvané **service**.

Service neposkytuje žádné uživatelské rozhraní a není vázána na životní cyklus aktivity (není tedy vázaná na životní cyklus celé aplikace). Běží na pozadí dokud ji uživatel nezastaví a nebo dokud není zastavena sama. Nejdůležitější typy services jsou:

- **Foreground service** slouží k vykonávání operací na pozadí. Nejsou na ni kladena žádné omezení, krom jednoho - uživatel musí být upozorněn, že service běží. V notifikační liště uživatel vidí, jaké foreground services právě běží. Například hudební přehrávač je dobrým příkladem této service. Uživatel pomocí notifikace může aplikaci do jisté míry ovládat (například přehrát další písničku).
- **Background service** slouží také k vykonání operací na pozadí. Od verze Android Oreo jsou na ni kladena velká omezení z důvodu bezpečnosti a plynulosti systému. Dříve tato service mohla běžet na pozadí téměř bez žádných omezení a proto se často stávalo, že některé services využívaly příliš mnoho zdrojů, i když jejich aplikace zrovna nebyly v popředí. Takto se například mohlo stát, že aplikace v popředí se sekala a uživatel neměl žádné tušení proč, jelikož tyto services nemusí upozornit uživatele o jejich běhu. Nová omezení nedovolují services, aby běžely na pozadí pokud jejich aplikace není v popředí. To znamená, že jakmile se aplikace dostane na pozadí, všechny její services jsou zastaveny.

Aplikace proto obsahuje jednu service, která je zapnuta poté, co je zapnut trénink. Uživatel po celou dobu běhu tréninku vidí v notifikační liště logo aplikace a je informován o tom, že trénink běží (jedná se tedy o foreground service).

6.3.8 Grafy

Grafy jsou jednou z nejdůležitějších částí aplikace. Uživateli umožňují zobrazení starších dat sportovce a tím je možné sledovat jeho trend během tréninku. Dále slouží k přehlednému zobrazení naměřeným hodnot. Všechny knihovny používané k zobrazení grafů napříč celou aplikací jsou licencované pod Apeche License, verze 2.0. V aplikaci se využívají tři typy grafů:

- K zobrazení **spojnicových grafů** se využívá knihovna *GraphView*. Do projektu byla přidána jako modul, jelikož bylo potřeba modifikovat vzhled grafů tak, aby zapadaly do celkové koncepce aplikace. Do knihovny byly například přidány stíny pod každý graf. GraphView podporuje automatické

překreslení, pokud je vložena nová hodnota do grafu. Osy grafu se také automaticky přizpůsobí nově vložené hodnotě.

- K zobrazení **paprskovitého grafu** je využita knihovna *MPAndroidChart*. Tato knihovna nabízí spoustu typů grafů, ale nepodporuje aktualizaci grafů v reálném čase. Proto nebyla využita k zobrazení spojnicových grafů, ale pouze pro zobrazení procentuálního poměru doby trvání mezi stavy, ve kterých se sportovec nacházel během tréninku.

- Posledním grafem je **sloupcový**, který se využívá k zobrazení spálených kalorií v jednotlivých částech tréninku. K zobrazení je využita knihovna *ChartProgressBar-Android*, která se liší od ostatních pouze designem, který dobře zapadá do aplikace. Graf dále nabízí možnost kliknout na jednotlivé sloupce pro zobrazení konkrétní hodnoty ve sloupci.

6.3.9 Pomocné třídy

Během vývoje vzniklo několik pomocných tříd, které ulehčily vývoj aplikace. Převážně se jedná o pomocné třídy, které umožnily konkrétní funkčnost, například potažením zprava doleva nad konkrétním sportovcem způsobí odpojení jeho senzoru od telefonu. Níže je seznam nejdůležitějších pomocných tříd se stručným popisem:

- **RxMds** je singleton[10] třída, která vznikla jako obal (wrapper) nad mobilní knihovnou MDS (viz kapitola 4.1). Slouží pro jednoduší asynchronní komunikaci se senzorem. Vytváří tak jednoduché API, které zajišťuje získání jakýchkoliv zdrojů senzoru. Například pro odebírání srdečního tepu je potřeba pouze zavolat metodu *subscribeToHeartRate* a předat sériové číslo senzoru (k identifikaci senzoru). V některých případech je možné zadat i frekvenci odběru, pokud to zdroj umožňuje.

- **Formatter** slouží k formátování často používaných řetězců.

- **Path** obsahuje všechny cesty ke zdrojům senzoru a také konstanty používané napříč celou aplikací.

- **DashboardItemTouchHelper, HistoryItemTouchHelper a ProfileItemTouchHelper** jsou potomky třídy *ItemTouchHelper.SimpleCallback*, která umožňuje definování dodatečné funkčnosti na jednotlivé prvky seznamu (RecyclerView).

[10] Singleton (česky jedináček nebo také unikát) je název pro návrhový vzor, používaný při programování. Využívá se k zajištění, aby v celém programu existovala pouze jedna instance určité třídy. [23]

6.4 Výpočet kalorií

Obecně je náročné vypočítat hodnotu spálených kalorií, jelikož se jedná o velice individuální proces. I přesto existují studie, které se snaží tuto hodnotu vypočítat. Výpočet kalorií v aplikaci je založen na této studií [24], která zohledňuje čtyři faktory.

Prvním faktorem je **srdeční tep**. Ve studii je zjištěno, že vztah srdečního tepu a spálených kalorií je lineární v rozsahu mezi 90 - 150 úderů za minutu. Při hodnotách mimo rozsah je pak výpočet méně přesný. V aplikaci není stanoven přesný rozsah pro srdeční tep, ale je ošetřeno, aby výsledná hodnota nebyla nesmyslná (například při hodnotě srdečního tepu 50 úderů za minutu by výsledná hodnota mohla být záporná).

Zbývajícími faktory jsou pak **váha**, **věk** a **pohlaví**, které jsou obecně hlavními faktory ve spalovaní kalorií. Výsledný vzorec upravený pro výpočet kalorií za minutu pro muže je dán:

$$\frac{-55.0969 + (0.6309 \cdot AvgSrdecniTep) + (0.1988 \cdot Vaha) + (0.2017 \cdot Vek)}{4.184} \quad (1)$$

a pro ženy:

$$\frac{-20.4022 + (0.4472 \cdot AvgSrdecniTep) + (0.1263 \cdot Vaha) + (0.074 \cdot Vek)}{4.184} \quad (2)$$

Aplikace každou minutu počítá průměrnou hodnotu srdečního tepu v dané minutě a tento průměr pak využije pro výpočet kalorií. Proto jsou kalorie v aplikaci aktualizovány každou minutu.

6.5 Výpočet vzdálenosti

Jelikož senzor nenabízí GPS a zároveň cílem této práce bylo vytvořit aplikaci, která nebude závislá na prostředí ve kterém se sportovec nachází, jediným způsobem, jak vypočítat uběhnutou vzdálenost je podle počtu kroků.

První je nutné zjistit, jak velký krok daný sportovec udělá. Podle článku [25] lze vypočítat velikost kroku na základě výšky a pohlaví. Vzorec pro muže je dán:

$$delkaKroku = 0.415 \cdot Vyska \quad (3)$$

a pro ženy:

$$delkaKroku = 0.413 \cdot Vyska \quad (4)$$

Jelikož vypočítaná hodnota je průměrnou hodnotou kroku při chůzi, bylo potřeba zjistit, o kolik procent se krok zvětší, pokud sportovec běží. Tuto informaci nebylo možné dohledat a proto na základě měření (měření bylo prováděno na 5 sportovcích) bylo zjištěno, že průměrné zvětšení kroku při běhu je **75 %**.

Velikost kroku se tedy odvíjí podle toho, zda sportovec běží a nebo jde. Pokud jde, je k výsledné vzdálenosti přičtena průměrná hodnota kroku, pokud běží je průměrná hodnota kroku zvětšena o 75 % a poté přičtena k výsledné vzdálenosti.

6.6 Struktura firmwaru

Základní struktura firmwaru senzoru již byla zmíněna v kapitole 4.2.1. Takto by vypadal firmware bez žádné uživatelsky definované služby. Senzor by byl stále zapnut a vysílal by do doby než by s k němu nepřipojil telefon (nebo dokud by se nevybila baterie).

Do firmwaru byly naprogramovány dvě uživatelsky definované služby. První služba slouží k detekci kroku a druhá k zapnutí/vypnutí senzoru. Tímto se rozšířila původní souborová struktura o čtyři soubory:

- **PedometerService.cpp** obsahuje zdrojový kód služby pro detekci kroku.
- **PedometerService.h** je hlavičkový soubor třídy PedometerService.
- **WakeUpService.cpp** obsahuje zdrojový kód služby pro zapnutí/vypnutí senzoru.
- **WakeUpService.h** je hlavičkový soubor třídy WakeUpService.

6.6.1 Služba pedometr

Jak již bylo zmíněno, hlavním cílem této služby je detekovat krok a poté tuto informaci poslat do telefonu. S detekcí kroku úzce souvisí i stav sportovce (senzoru), proto tato služba rozeznává několik stavů ve kterých se sportovec může nacházet. V této kapitole je popsán algoritmus pro detekci kroku a s ním úzce spojený algoritmus na rozeznávání stavů.

Pro detekci kroku byl použit velice jednoduchý algoritmus, který na základě změn na akcelerometru detekuje krok. Skládá z následujících kroků:

1. Přečti hodnoty z akcelerometru.
2. Aplikuj low-pass filtr[11] na každou osu akcelerometru pro vyhlazení dat.
3. Vypočítej velikost vektoru.
4. Pokud je velikost vektoru větší než mezní hodnota (threshold) a zároveň pokud je doba mezi posledním krokem alespoň 120 ms, pak detekuj krok a odešli tuto informaci telefonu.

[11] Jako low-pass filtr (dolní propust) se označuje lineární filtr, který nepropouští signál vyšších frekvencí. [26]

Mezní hodnota byla zvolena na základě měření na 10,8 ms^{-2}. Jedná se o tak vysokou hodnotu z důvodu, že akcelerometr měří i gravitační zrychlení a proto na vertikální osu akcelerometru působí gravitační síla s gravitačním zrychlením 9,81 ms^{-2}.

Doba mezi kroky je důležitá k zamezení detekci několika desítek kroků například při výskoku sportovce, kdy velikost vektoru by zcela určitě přesáhla mezní hodnotu a proto by bylo detekováno několik desítek kroků, které se reálně neuskutečnily.

6.6.1.1 Rozeznávání stavů

Jak již bylo zmíněno, senzor rozeznává i stavy, ve kterých se může nacházet. Stavy napomáhají ke správným výpočtům vzdálenosti, jelikož záleží zda například 100 kroků bylo naměřeno při chůzi, nebo při běhu. Senzor rozeznává tyto stavy:

- **Standing** reprezentuje stav, kdy sportovec stojí.
- **Walking** reprezentuje stav, kdy sportovec jde.
- **Running** reprezentuje stav, kdy sportovec běží.
- **Unknown** reprezentuje stav, kdy senzor neví v jakém stavu se nachází. Tento stav je použit jako výchozí při startu této služby.

Rozeznávání je založeno na velikosti vektoru, který je spočítán při detekci kroku. Pokud po dobu 2,5 s není zaznamenán krok (hodnota nepřesáhla mezní hodnotu 10,8 ms^{-2}), pak je odeslán stav *standing*. Pokud je krok zaregistrován, existují dvě další hodnoty, které reprezentují stavy *walking* a *running*. Pro stav *walking* je hodnota stejná jako v případě mezní hodnoty pro detekci kroku, tedy 10,8 ms^{-2}, a pro stav *running* je hodnota stanovena na 15 ms^{-2}. Rozeznávaní těchto stavů není založeno na tom, zda velikost vektoru byla větší než stanovené hodnoty, ale které hodnotě je velikost vektoru nejblíže.

Aby se snížila komunikace po Bluetooth na co nejmenší úroveň, senzor si pamatuje předchozí stav ve kterém byl a pokud je stejný jako nově detekovaný stav, pak stav není odeslán.

6.6.2 Služba pro probuzení senzoru

Jelikož základní firmware neobsahoval způsob, jak senzor vypnout, byla naprogramována jednoduchá služba, která upozorňuje uživatele o stavu senzoru (vypnut/zapnut) a dále nastaven limit jedné minuty, po kterém se senzor automaticky vypne, pokud se k němu nepřipojí telefon. Velmi podobná služba již byla naprogramována přímo od Movesense pro demonstraci, jak naprogramovat vlastní uživatelsky definovanou službu a dále jak používat jejich nízko úrovňové API k jednotlivým komponentám senzoru. Informace jsou proto čerpány z [27] a [28].

Senzor se skládá z dílčích komponent jako jsou akcelerometr, gyroskop atd. Obsahuje i komponentu zvanou **MAX3000x**, která se primárně využívá pro počítaní srdečního tepu a EKG. Právě s využitím této komponenty vznikla služba pro zapnutí senzoru dotykem na oba jeho konektory (je detekován srdeční tep). Jak je možné vidět ze zdrojového kódu 16, jedná se o velice nízkoúrovňové API, protože pro zapnutí této služby je potřeba vložit na zdroj služby hodnotu 1. Hodnota 0 by tuto možnost vypnula.

```
1 asyncPut(WB_RES::LOCAL::COMPONENT_MAX3000X_WAKEUP(),
2       AsyncRequestOptions(NULL, 0, true), (uint8_t)1);
```

Zdrojový kód 16: Asynchronní metoda PUT, která připraví senzor na příští zapnutí pomocí dotyku na konektory

Služba dále nabízí upozornění o tom, zda je senzor zapnutý a to tak, že každou sekundu se rozsvítí LED dioda. Tato funkčnost využívá další komponentu a tou je **LED**. Tu lze ovládat pomocí služby s cestou */Ui/Ind/Visual*. Příklad rozsvícení diody na krátký moment je možné vidět ve zdrojovém kódu 17.

```
1 //rozsvit' LED diodu
2 asyncPut(WB_RES::LOCAL::UI_IND_VISUAL(), AsyncRequestOptions::Empty,
3       WB_RES::VisualIndTypeValues::SHORT_VISUAL_INDICATION);
```

Zdrojový kód 17: Rozsvícení LED diody asynchronním voláním

Celková logika služby je pak jednoduchá. Po zapnutí senzoru (a tedy i služby), každou sekundu rozsviť diodu. Pokud se po dobu jedné minuty nikdo na senzor nepřipojil, připrav senzor na příští zapnutí pomocí dotyku na konektory a poté senzor vypni. Vypnout senzor je opět velmi podobné předchozím příkladům, jedná se pouze o jinou službu s jiným zdrojem a parametrem. Příklad je možné vidět ve zdrojovém kódu 18.

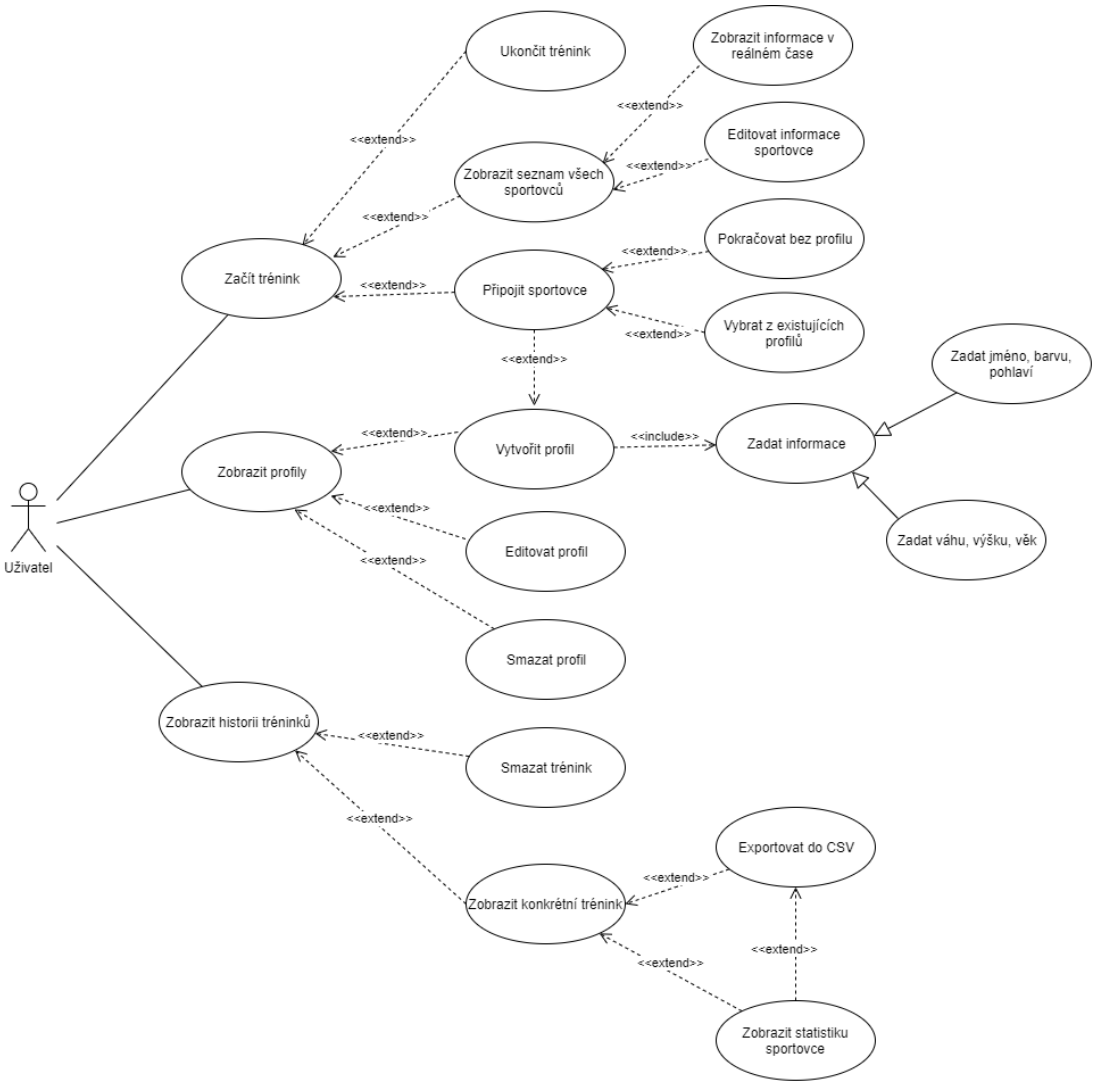
```
1 asyncPut(WB_RES::LOCAL::SYSTEM_MODE(), AsyncRequestOptions(NULL, 0,
    true),
2       WB_RES::SystemModeValues::FULLPOWEROFF);
```

Zdrojový kód 18: Vypnutí senzoru pomocí asynchronního volání

7 Uživatelská dokumentace

V této kapitole je popsána aplikace z uživatelského úhlu pohledu, tedy popis jednotlivých funkcí a k nim odpovídající snímky obrazovky.

Na obrázku 15 je diagram případů užití aplikace, kde je přehledně zobrazeno, jaké možnosti aplikace uživateli nabízí.



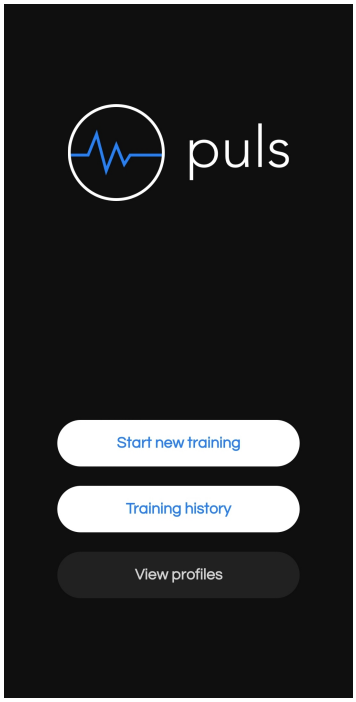
Obrázek 15: Diagram případů užití aplikace

7.1 Úvodní obrazovka

Ihned po zapnutí aplikace se zobrazí úvodní obrazovka (obr. 16), na které je na výběr ze tří základních možností:

- Tlačítko „Start new training“ začne nový trénink.

- Tlačítko „Training history“ zobrazí historii všech tréninků, které byly zaznamenány.
- Tlačítko „View profiles“ zobrazí přehled všech nadefinovaných profilů.



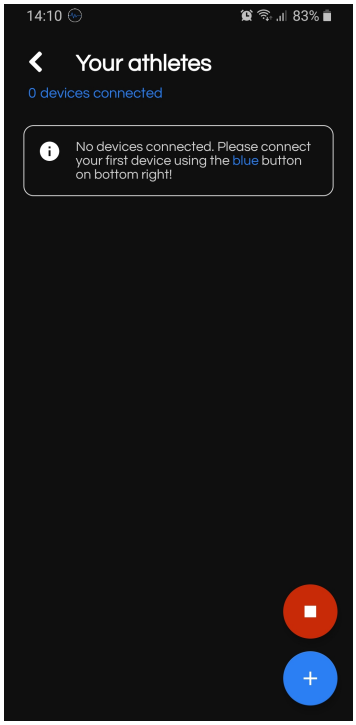
Obrázek 16: Úvodní obrazovka aplikace

7.2 Spuštění a ukončení tréninku

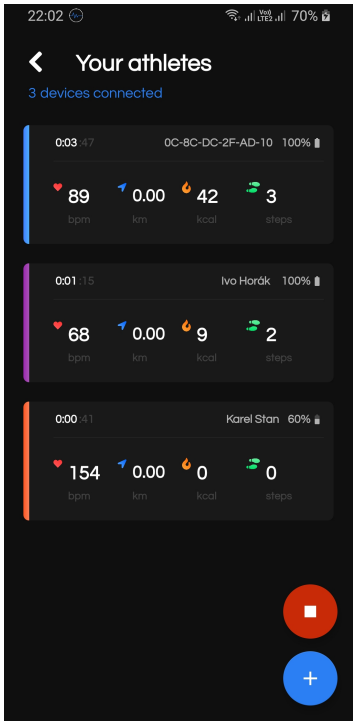
Po spuštění nového tréninku (obrázky 17 a 18) uživatel vidí seznam všech připojených sportovců a dále je zobrazena notifikace o tom, že trénink běží. Od této chvíle je možné aplikaci vypnout nebo telefon uzamknout, přičemž trénink stále poběží na pozadí. Uživatel zde může přidat nového sportovce nebo trénink zastavit tlačítky dole vpravo.

Pokud je seznam sportovců prázdný (obr. 17), uživatel vidí nápovědu, jak přidat nového sportovce. Tato nápověda zmizí po připojení prvního sportovce. V přehledu je pak možné vidět čtyři základní informace o sportovci: srdeční tep, uběhnutá vzdálenost, spálené kalorie a počet kroků. Dále u každého sportovce lze vidět jeho jméno, stav baterie senzoru a strávený čas na tréninku. Pokud uživatel nezadá jméno sportovce (viz kapitola 7.3), je jméno nahrazeno MAC adresou senzoru.

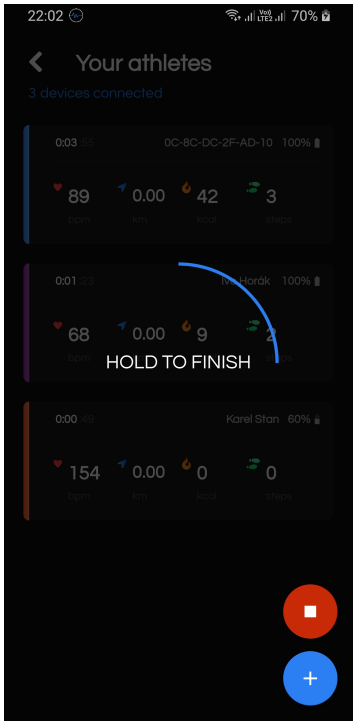
Stav baterie je počítán na základě jejího vnitřního odporu a napětí. Jelikož teplota a jakýkoliv pohyb senzoru ovlivňuje vnitřní odpor baterie a její napětí, výsledky nemusí být přesné. Energie je generována na základě chemických procesů, je tedy možné, že například úplně nová baterie se bude jevit jako prázdná



Obrázek 17: Prázdný přehled sportovců



Obrázek 18: Přehled se třemi sportovci



Obrázek 19: Ukončení běžícího tréninku

v extrémně nízkých teplotách (pokud by se jednalo o měření založeném na napětí), ale při měření vnitřního odporu se bude jevit jako plná.

Ukončit trénink lze podržením červeného tlačítka. Během držení tlačítka je pozadí aplikace ztmaveno (obr. 19) a lze vidět, jak dlouho uživatel tlačítko musí držet, aby trénink ukončil. Tímto způsobem je zamezeno nechtěnému ukončení tréninku. Pokud je ukončen trénink, který neobsahoval žádné sportovce, pak tento trénink není uložen.

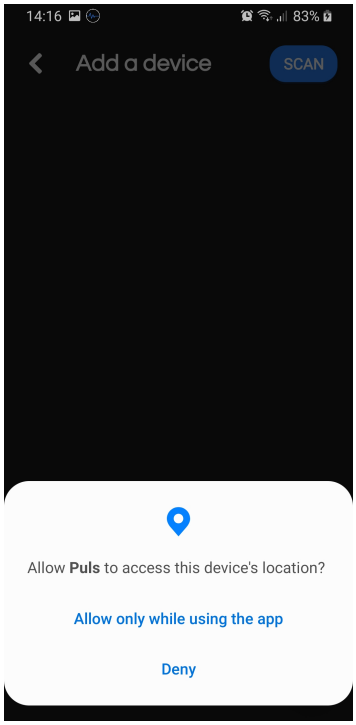
7.3 Připojení nového sportovce

Připojit nového sportovce lze modrým tlačítkem vpravo v přehledu sportovců daného tréninku. Pokud se jedná o připojení prvního sportovce od instalace aplikace, je uživatel vyzván k povolení, aby aplikace mohla využívat lokaci zařízení (obr. 20). Toto povolení je nezbytné pro hledaní Movesense senzorů v okolí telefonu pomocí Bluetooth.

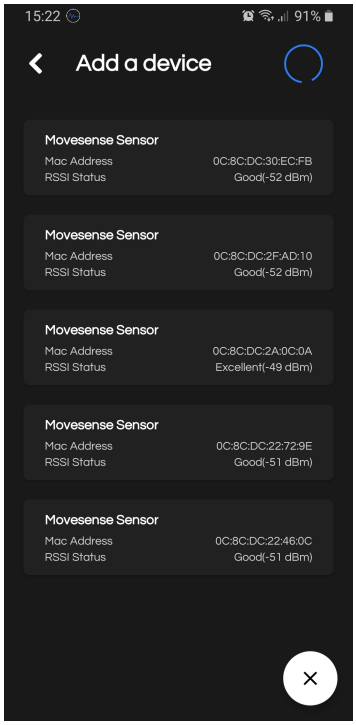
Po přijetí povolení je zahájeno vyhledávání senzorů v okolí. Všechny zařízení, ke kterým je možné se připojit lze vidět na obrazovce (obr. 21) a u každého z nich lze vidět MAC adresa a *Received Signal Strength Indicator* (RSSI) hodnota. RSSI hodnota informuje o kvalitě signálu mezi senzorem a telefonem.

Kliknutím na jeden z těchto senzorů pak skončí vyhledávání okolních zařízení a zahájí se připojování k vybranému senzoru (obr. 22).

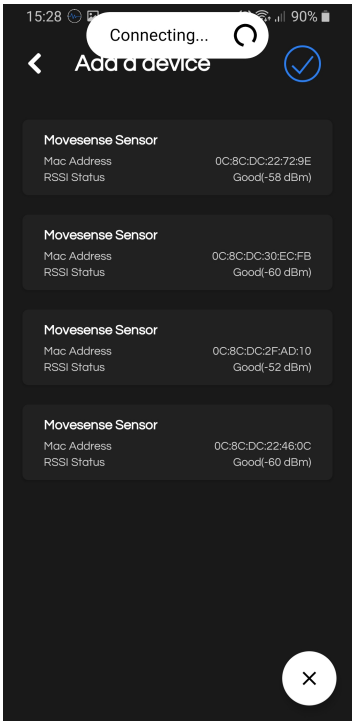
Po úspěšném připojení k senzoru je uživatel přesměrován na obrazovku, kde



Obrázek 20: Výzva k povolení lokaci telefonu



Obrázek 21: Vyhledávání senzorů v okolí



Obrázek 22: Připojení k senzoru

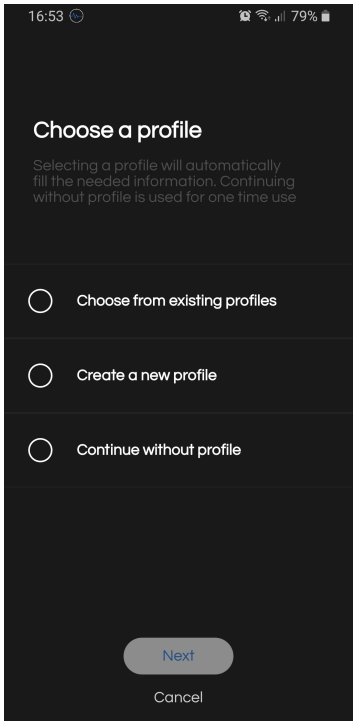
zvolí, jakým způsobem chce zadat informace o sportovci (obr. 23). Na výběr jsou tyto možnosti:

- Možnost „Choose from existing profiles“ využije informace z již existujícího profilu a použije je pro tohoto sportovce. Po tomto výběru je uživatel přesměrován zpět do přehledu sportovců.
- Možnost „Create a new profile“ vytvoří profil z informací, které následně uživatel zadá. Při dalším připojení uživatel znovu tyto informace nemusí vyplňovat a může použít již vytvořený profil. Při vytváření profilu je nutné, aby uživatel zadal všechny informace. Není možné vytvořit neúplný profil. Po výběru této možnosti je uživatel přesměrován k zadávání jednotlivých informací.
- Možnost „Continue without profile“ nevytvoří ze zadaných informací profil, ale pouze je použije pro daného sportovce. V tomto případě je možnost informace nezadat vůbec, ale výpočty pak budou využívat implicitních hodnot a proto výsledky nemusí být přesné. Po výběru této možnosti je uživatel přesměrován k zadávání informací, stejně jako v předchozím případě (viz obrázky 24 a 25).

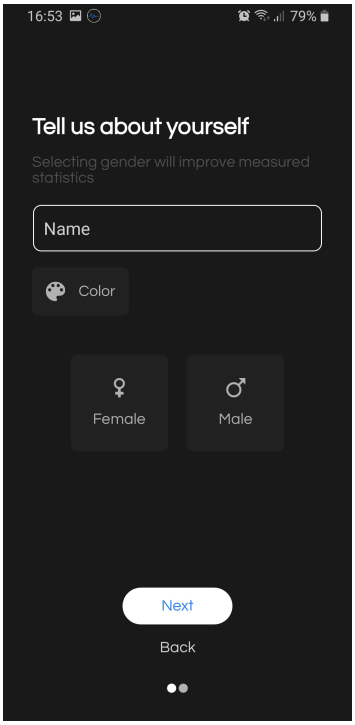
Připojený senzor lze i ručně odpojit. Tato možnost byla přidána hlavně z důvodu, kdy není dostatek senzorů nebo telefon neumožní připojení více senzorů

souběžně (omezení Bluetooth), ale uživatel i přesto chce měřit více sportovců. Ruční odpojení a poté znova připojení senzoru aplikace bere jako připojení nového sportovce a tímto je možné měřit více sportovců (senzory si navzájem sportovci půjčují).

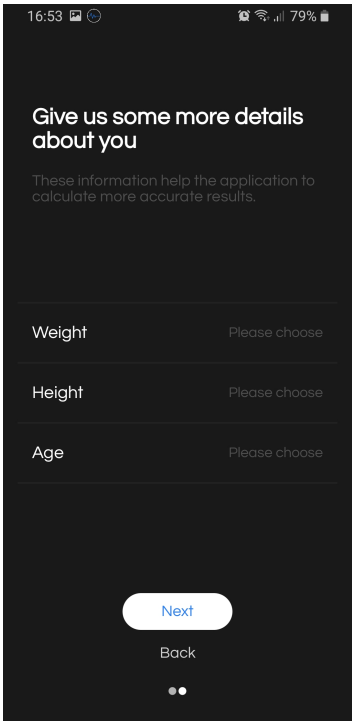
Pokud se sportovec vzdálí od telefonu příliš daleko, senzor se automaticky odpojí a sportovec z přehledu sportovců zmizí. V tom případě stačí, aby se sportovec přiblížil zpět k telefonu, senzor se automaticky připojí a sportovec dále pokračuje v tréninku (automaticky se zase zobrazí v přehledu sportovců). Během odpojení nejsou měřena data, takže jakákoliv aktivita sportovce po odpojení není nijak zaznamenána.



Obrázek 23: Výběr možností, jak zadat informace o sportovci



Obrázek 24: Obrazovka pro zadání jména, pohlaví a barvy



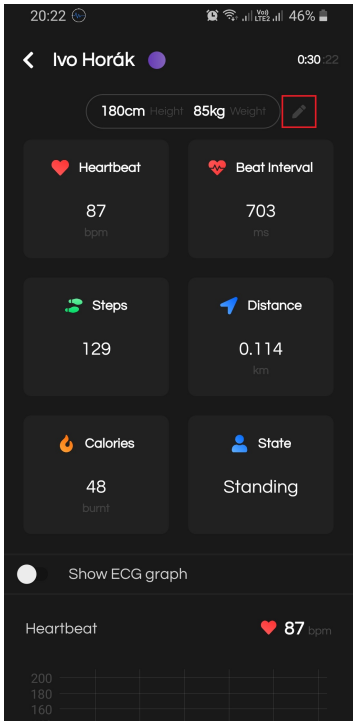
Obrázek 25: Obrazovka pro zadání váhy, věku a výšky

7.4 Detailní přehled sportovce v reálném čase

Kliknutím na konkrétního sportovce v přehledu sportovců se zobrazí jeho detailnější informace s grafy (obr. 25). Grafy se automaticky aktualizují a hodnoty na osách x a y se automaticky přizpůsobují nově naměřeným hodnotám.

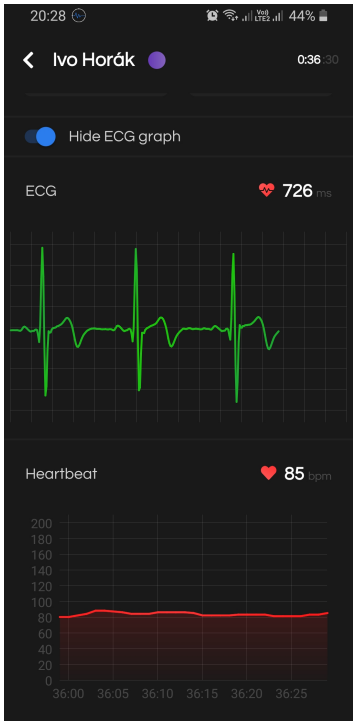
Mezi nově zobrazené hodnoty patří variabilita srdečního tepu (HRV) a stav sportovce. Variabilita srdečního tepu je prostým měřítkem časové variace mezi každým z úderů srdce. Stavy, ve kterých se sportovec může nacházet jsou již popsány v kapitole 6.6.1.1.

Dále jsou zde zobrazeny grafy k jednotlivým výpočtům (obr. 27). Speciálním grafem je pak graf zvaný elektrokardiogram (dále EKG). Tento graf není zobrazen ve výchozím zobrazení, jelikož pro jeho přesné zobrazení je potřeba, aby sportovec nebyl v pohybu (hrudní pás musí být v těsném kontaktu s kůží). Zobrazení grafu je tedy ovladatelné uživatelem a může ho zapínat a vypínat dle potřeby. Je důležité mít na paměti, že rychlost odebírání EKG je 128 Hz a tím může zahltit Blueotooth telefonu. EKG je možné vidět na obrázku 28.



Obrázek 26: Detailnější přehled sportovce

Obrázek 27: Grafy jednotlivých informací



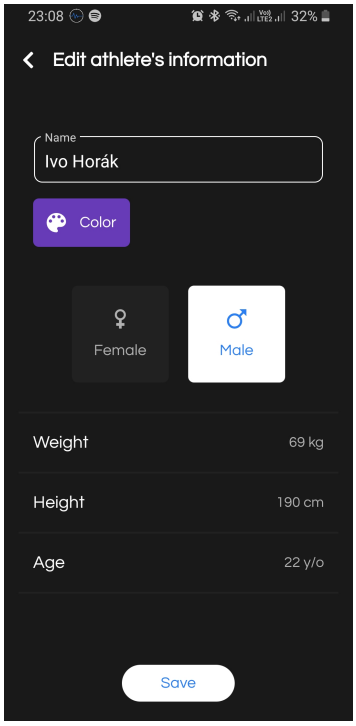
Obrázek 28: Elektrokardiogram

V detailním přehledu sportovce je i možnost editovat již zadané informace. Slouží k tomu „tužtička“, která se nachází napravo od základních informací (viz červený čtverec na obrázku 25). Uživatel se tímto dostane na editační obrazovku (obr. 29). V tomto případě jsou editována data sportovce, nikoli profilu a to i v případě, že údaje o sportovci byly zadány výběrem z již existujících profilů. Jakmile jsou data uložena všechny výpočty se provádí s nově upravenými hodnotami.

7.5 Zobrazení historie tréninků

Zobrazení historie tréninků je druhou možností, kterou je možné vybrat z úvodní obrazovky. Po kliknutí na tlačítko „Training history“ jsou zobrazeny všechny zaznamenané tréninky (obr. 30).

Každý trénink je možné vymazat přetažením zleva doprava po vybraném tréninku. O jeho odstranění je uživatel upozorněn zobrazením dialogu ve spodní



Obrázek 29: Editace údajů sportovce

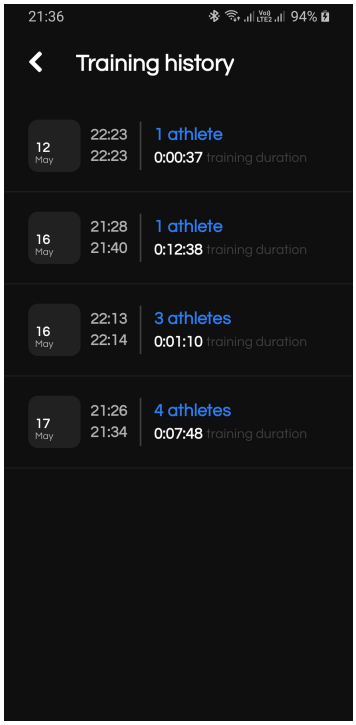
části aplikace. Na této liště je možnost kliknout na tlačítko „UNDO“, které vymazaný trénink obnoví. Dialog je zobrazen přibližně tři sekundy a poté zmizí. Po zmizení dialogu není možné nijak trénink obnovit.

Kliknutím na trénink se zobrazí seznam všech sportovců, kteří se účastnili tréninku. U každého sportovce jsou zobrazeny jeho základní údaje, tedy kolik udělal kroků, průměrný srdeční tep atd. Pokud byl sportovec během tréninku odpojen (zde se nejedná o případ, kdy se sportovec dostal mimo dosah Bluetooth, ale o případ, že ho uživatel ručně odpojil) je tato skutečnost zobrazena napravo od jeho jména přeškrtnutým symbolem Bluetooth (viz červený čtverec na obrázku 31).

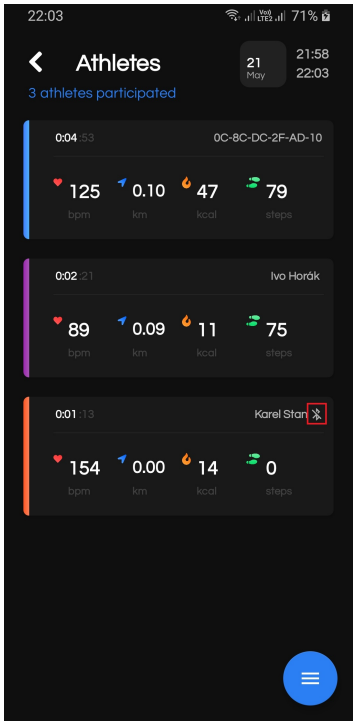
Modrým tlačítkem vpravo dole je možné exportovat trénink ve formátu CSV[12]. Uživatel má pak dvě možnosti, jak s vyexportovaným souborem naložit:

- První volba nabízí soubor uložit do paměti telefonu. Je zobrazen správce pro ukládání souborů, kde je možné soubor přejmenovat a vybrat jeho místo uložení.
- Druhá volba nabízí soubor sdílet. Je zobrazen dialog, ve kterém si lze vybrat, jakým způsobem chceme soubor sdílet (například poslat emailem nebo soukromou zprávou).

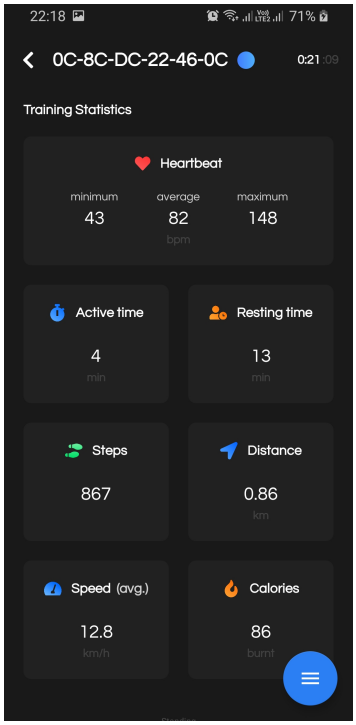
[12]CSV (Comma-separated values) je jednoduchý textový formát pro reprezentaci tabulkových dat. Soubor ve formátu CSV se skládá z řádků, ve kterých jsou jednotlivé položky odděleny znakem čárka (","). Hodnoty položek pak mohou být uzavřeny do uvozovek ("). [29]



Obrázek 30: Seznam zaznamenaných tréninků



Obrázek 31: Seznam sportovců na tréninku



Obrázek 32: Informace o vybraném sportovci

Kliknutím na sportovce jsou zobrazeny jeho detailní informace (obr. 33). Navíc se zde nachází informace jako průměrná rychlost, kolik minut strávil sportovec v pohybu a kolik minut strávil na střídačce.

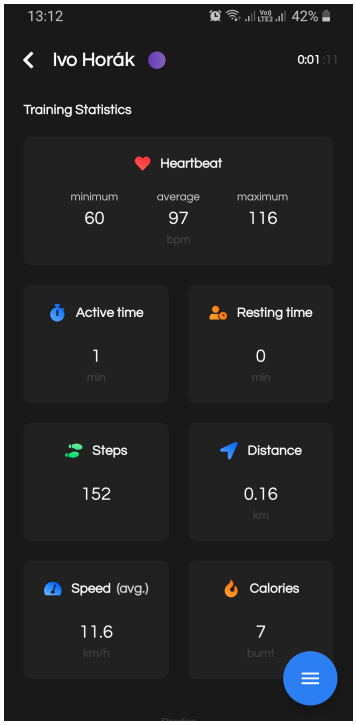
Průměrná rychlost se počítá na základě vzdálenosti a času, který sportovec strávil v pohybu. Nepočítá se zde s dobou, kterou strávil na střídačce.

Aktivní čas, tedy čas strávený v pohybu, je vypočítán na základě stavů - v tomto případě se jedná o stavy chůze a běhu. Přesněji, je to doba, kterou sportovec strávil v těchto stavech. Čas strávený na střídačce je vypočítán ze stavu stání.

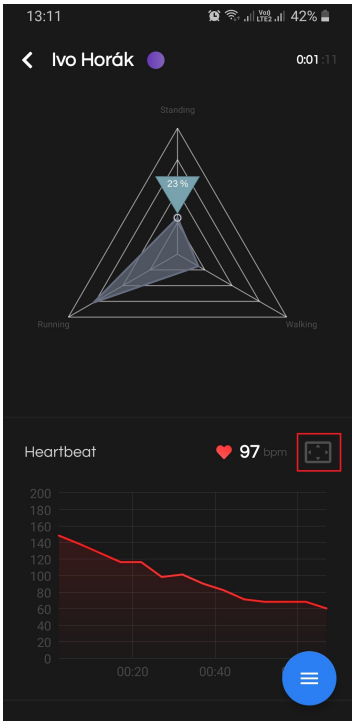
Mimo základních čtyř grafů (graf srdečního tepu, spálených kalorií, uběhnuté vzdálenosti a počtu kroků) je zde tzv. paprskový graf, který vyjadřuje procentuální poměr mezi dobou strávenou v jednotlivých stavech. Do grafu lze kliknout pro zobrazení procentuálních hodnot jednotlivých stavů.

Všechny grafy (kromě paprskového) lze zvětšit na celou obrazovku displeje (obr. 35). K tomu slouží ikonka vpravo nahoře nad každým grafem (viz červený čtverec na obrázku 34). Pokud je graf zobrazen na celou obrazovku, lze v něm přibližovat nebo se pohybovat po horizontální ose. Při přiblížení se pak automaticky přepočítávají hodnoty na obou osách.

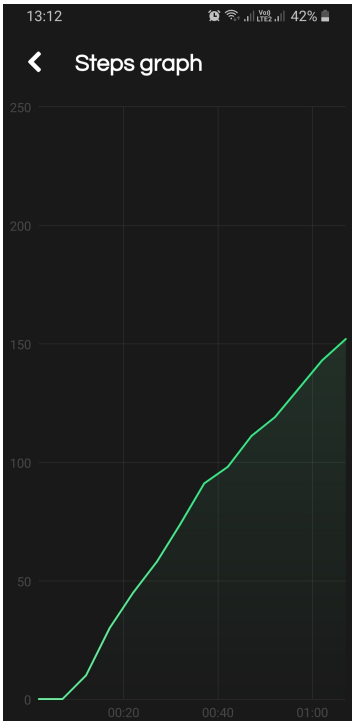
Graf spálených kalorií byl změněn ve sloupcový graf, který je rozdělen do šesti stejných částí. Pokud by byl trénink příliš krátký pro zobrazení šesti sloupců, bude pak zobrazen pouze jeden. Tímto je možné zjistit, v jaké části tréninku



Obrázek 33: Detail sportovce



Obrázek 34: Paprskovitý graf



Obrázek 35: Graf přes celou obrazovku

bylo nejvíce spáleno kalorií. I v tomto případě platí, že na jednotlivé sloupce lze kliknout pro zobrazení konkrétní hodnoty. Při zvětšení tohoto grafu je zobrazen klasický spojnicový graf.

Vyexportovat se dají i údaje konkrétního sportovce na tréninku. K tomu slouží modré tlačítko vpravo dole v detailu sportovce. Jsou zde na výběr stejné možnosti jako v případě exportu tréninku.

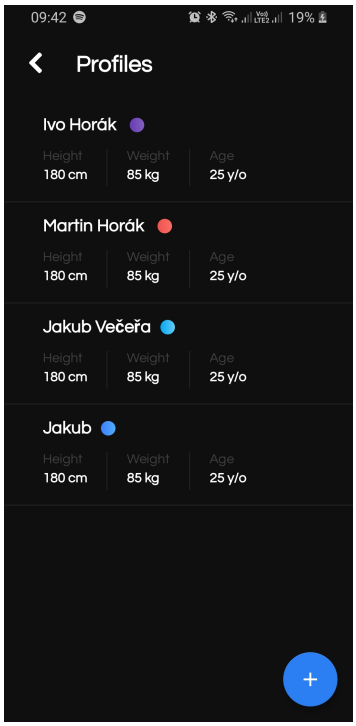
7.6 Zobrazení definovaných profilů

Zobrazení definovaných profilů je poslední možností, kterou je možné vybrat z úvodní obrazovky. Jak již bylo zmíněno, profily slouží primárně k tomu, aby uživatel nemusel každý trénink zadávat informace o sportovci neustále dokola. V ideálním případě by si měl připravit profily pro každého sportovce a poté pouze vybírat z nabídky. Přehled všech profilů je možný vidět na obrázku 36.

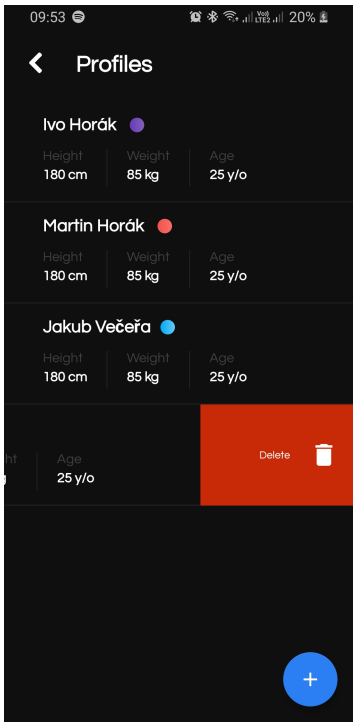
Přidat profil lze pomocí modrého tlačítka vpravo dole. Uživatel je tímto přesměrován na stejné obrazovky jako v případě připojení nového sportovce (viz obrázky 24 a 25). Pro vytvoření profilu musí uživatel vyplnit všechny informace. Pokud se uživatel pokusí uložit neúplný profil, aplikace mu to neumožní a zobrazí chybový dialog.

Profil je možné editovat kliknutím na něj. Uživatel je tímto přesměrován na editační obrazovku profilu, která je velice podobná editační obrazovce sportovce (obr. 29). Neúplný profil uložit nelze.

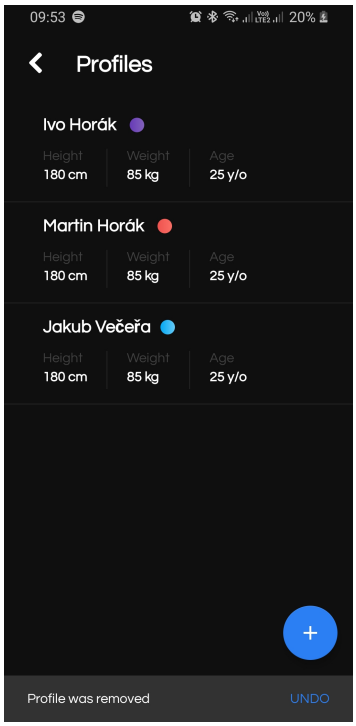
Stejně jako u tréninků, každý profil je možné odstranit a to přejetím po vybraném profilu zprava doleva (obr. 37). I zde se objeví dialog s možností „UNDO“, který vymazaný profil obnoví (viz obrázek 38).



Obrázek 36: Přehled všech profilů



Obrázek 37: Odstranění profilu



Obrázek 38: Možnost obnovit odstraněný profil

8 Plány do budoucna

V budoucnu bych rád pokračoval v dalším vývoji aplikace. Nyní aplikace nabízí základní funkčnost jako je zobrazení srdečního tepu, počtu kroků, uběhnuté vzdálenost, spálených kalorií atd., nicméně stále existují oblasti, ve kterých by aplikace mohla být daleko lepší. Aplikaci by mohla obsahovat například následující vylepšení a funkce:

- Hlubší analýza dat. Nyní jsou data pouze vypočítána a zobrazena. Aplikace by mohla podle vypočítaných dat například navrhnout, kdy by měl sportovec vystřídat.
- Mohl by být využit „chytřejší“ algoritmus pro detekci kroku. Stávající algoritmus je jeden ze základních, který využívá pouze akcelerometr. Takto jdou kroky velice jednoduše zfalšovat. Vylepšení by se hlavně týkalo využití více komponent senzoru (například gyroskop a magnetometr).
- Pokud by se senzor odpojil od telefonu (kvůli dosahu), naměřená data by mohla být ukládána do paměti senzoru a po automatickém připojení by tato data mohla být poslána a synchronizována s telefonem.
- Automatická aktualizace firmwaru. Firmware senzoru se nyní aktualizuje aplikací od firmy Movesense. Tato aplikace využívá pro aktualizaci veřejné knihovny, tudíž je možné implementovat vlastní systém pro aktualizace. Například po připojení senzoru by se senzor automaticky aktualizoval, pokud by existovala novější verze.
- Zobrazovat dlouhodobější statistiku konkrétního sportovce. Pokud by uživatel pokaždé při připojení stejného sportovce využil jeho profil, pak by se data ze všech tréninků, kterých se sportovec účastnil, mohla analyzovat a zobrazovat.
- Porovnání dvou a více sportovců. Aplikace by mohla umožnit vybrat konkrétní sportovce, kteří se účastnili tréninku a jejich data porovnat a tímto zobrazit například grafy, které by obsahovaly hodnoty vybraných sportovců dohromady.
- Zpřístupnění aplikace prostřednictvím Google Play.

Závěr

Cílem práce bylo najít řešení pro odečítání a vizualizaci senzorických dat ze sporttesterů pro skupinu sportovců. Vznikla tak aplikace na platformě Android, která komunikuje se senzorem od firmy Movesense. Tento senzor pak obsahuje rozšířený firmware pro detekci kroků a rozpoznání stavů, ve kterých se sportovci mohou nacházet. Aplikace umožňuje zahájit trénink, na který se může připojit několik sportovců souběžně a následně zobrazovat naměřená data jako jsou například spálené kalorie, počet kroků, uběhnutá vzdálenost nebo srdeční tep. K jednotlivým údajům pak existují i grafy, které se v reálném čase automaticky aktualizují. Jednotlivé tréninky jsou ukládány a aplikace umožňuje jejich zpětné zobrazení. Dále je možné v aplikaci definovat vlastní profily, které usnadňují vyplňování údajů o sportovci.

V budoucnu je pak hlavním cílem aplikaci zpřístupnit prostřednictvím Google Play a postupně používat složitější algoritmy pro analýzu naměřených dat. Mezi plánované vylepšení patří například automatická aktualizace firmwaru senzoru nebo dlouhodobější statistika individuálního sportovce.

Conclusions

The bachelor's thesis aimed to find a solution for real-time reading and visualization of sensor data from sport testers for a group of athletes. An Android application communicating with the Movesense sensor has been developed. The sensor's firmware has been extended so it can detect steps and states of the athletes. The application allows you to start a training session to which multiple athletes can be connected simultaneously. Measured data such as calories burnt, the number of steps, distance covered, or heart rate is visualized in the application in real-time. Each information has its own graph which is automatically updated whenever new data comes from the sensor. Every training session is stored in the application, therefore it can be reviewed later on. Furthermore, it is possible to define custom profiles in the application, which facilitates filling in information about athletes.

In the future, the main goal is to make the application accessible through Google Play and gradually use more complex algorithms to analyze measured data. Planned improvements are, for example, automatic sensor firmware updates or longer-term individual athlete statistics.

A Použité knihovny a jejich licence

V aplikaci bylo použito mnoho knihoven, které jsou licencované jako open source. Mezi nejznámější patří například Apache License 2.0, MIT license nebo GNU General Public License (GPL). V této kapitole jsou stručně popsány jednotlivé knihovny, jejich zdroj a jak byly použity v aplikaci:

- **RxAndroidBle** je knihovna, která ulehčuje práci s Bluetooth na platformě Android. Byla využita k vyhledávání okolních senzorů a k zjištění jejich MAC adresy. Knihovna je dostupná z https://github.com/Polidea/RxAndroidBle.
- **Butterknife** slouží k jednoduššímu propojení UI komponent s objekty v kódu, které tyto komponenty reprezentují. Knihovna je dostupná z https://github.com/JakeWharton/butterknife.
- **Gson** slouží k serializaci a deserializaci JSON objektů. Knihovna je dostupná z https://github.com/google/gson.
- **Loadtoast** je využit pro zobrazení dialogu během připojení k senzoru. Knihovna je dostupná z https://github.com/code-mc/loadtoast.
- **Lombok** je knihovna pro automatické generovaní getterů a setterů pro jednotlivé vlastnosti tříd. Knihovna je dostupná z https://projectlombok.org/setup/android.
- **Toasty** slouží k zobrazení informačních, upozorňujících a chybových hlášek. Knihovna je dostupná z https://github.com/GrenderG/Toasty.
- **MPAndroidChart** slouží k zobrazení spojnicových grafů. Knihovna je dostupná z https://github.com/PhilJay/MPAndroidChart.
- **Android-Iconics** slouží k zobrazení jakýkoliv ikonek z nejznámějších ikonových sad jako například *Fontawesome*. Knihovna je dostupná z https://github.com/mikepenz/Android-Iconics.
- **ColorPicker** slouží k zobrazení dialogu pro výběr barev pro jednotlivé sportovce. Knihovna je dostupná z https://github.com/Dhaval2404/ColorPicker.
- **DxLoadingButton** slouží k zobrazení animovaného tlačítka pro zahájení skenování okolních Movesense senzorů. Knihovna je dostupná z https://github.com/StevenDXC/DxLoadingButton.
- **MaterialSpinner** slouží k zobrazení rozbalovacího seznamu k výběru profilů. Knihovna je dostupná z https://github.com/jaredrummler/MaterialSpinner.

- **FloatingActionButtonSpeedDial** slouží k zobrazení menu při kliku na plovoucí tlačítko, například při exportu sportovce do CSV formátu. Knihovna je dostupná z https://github.com/leinardi/FloatingActionButtonSpeedDial.

B Obsah přiloženého DVD

Součástí práce je DVD s následující strukturou:

bin/
: Instalátor aplikace ve formě .apk balíčku a firmware, kterým je možné aktualizovat Movesense senzor.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src-android/
: Kompletní zdrojové kódy aplikace se všemi potřebnými knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření .apk balíčku pomocí Android Studia. Jedná se o celý Android Studio projekt.

src-firmware/
: Kompletní zdrojové kódy firmwaru se všemi potřebnými knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření zip balíčku pro aktualizaci Movesense senzoru.

readme.txt
: Instrukce pro instalaci a spuštění aplikace, včetně všech požadavků pro jeho bezproblémový provoz za účelem testování při tvorbě posudků práce a pro účel obhajoby práce.

install/
: Obsahuje instalační soubory nutné k vytvoření balíčku pro aktualizaci firmwaru senzoru.

U veškerých cizích převzatých materiálů obsažených na DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] *Smartphone Market Share. : OS Data Overview* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://www.idc.com/promo/smartphone-market-share/os⟩.

[2] Přispěvatelé Wikipedie. *Elektrokardiogram* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Elektrokardiogram&oldid=18257080⟩.

[3] *Mobile & Tablet Android Version Market Share Worldwide.* [online]. [cit. 2020-5-17]. Dostupný z: ⟨https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide⟩.

[4] Přispěvatelé Wikipedie. *API* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=API&oldid=18348465⟩.

[5] *MOVESENSE SENSOR.* [online]. 2016 [cit. 2020-5-17]. Dostupný z: ⟨https://www.movesense.com/wp-content/uploads/2016/11/Movesense_Sensor_Tech_Sheet_1.0.pdf⟩.

[6] *MOVESENSE SYSTEM OVERVIEW. : Software Overview* [online]. [cit. 2020-5-17]. Dostupný z: ⟨http://www.movesense.com/docs/system/system_overview/⟩.

[7] Přispěvatelé Wikipedie. *Representational State Transfer* [online]. 2019 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=17357691⟩.

[8] Přispěvatelé Wikipedie. *Reaktivní programování* [online]. 2019 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Reaktivn%5C%C3%5C%AD_programov%5C%C3%5C%A1n%5C%C3%5C%AD&oldid=17953630⟩.

[9] Přispěvatelé Wikipedie. *Observer (návrhový vzor)* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Observer_(n%5C%C3%5C%A1vrhov%5C%C3%5C%BD_vzor)&oldid=18410259⟩.

[10] Přispěvatelé Wikipedie. *JavaScript Object Notation* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=18470335⟩.

[11] Přispěvatelé Wikipedie. *YAML* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=YAML&oldid=18413745⟩.

[12] Přispěvatelé Wikipedie. *Mikroslužby* [online]. 2019 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Mikroslu%5C%C5%5C%BEby&oldid=17984829⟩.

[13] Google. *Platform Architecture* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://developer.android.com/guide/platform⟩.

[14] Meier, Reto. *Professional Android*. 4th ed. 2018. CMXXIX, 929 s. ISBN 978-1-118-10227-5.

[15] Přispěvatelé Wikipedie. *Android (operační systém): Historie* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Android_(opera%5C%C4%5C%8Dn%5C%C3%5C%AD_syst%5C%C3%5C%A9m)&oldid=18261680⟩.

[16] *Aktivita, Intent, Fragment a životní cyklus*. [online]. [cit. 2020-5-17]. Dostupný z: ⟨https://book.droidboy.cz/3-zivotni_cyklus.html⟩.

[17] *An Android Studio Master/Detail Flow Tutorial - Android 6*. [online]. [cit. 2020-5-17]. Dostupný z: ⟨https://www.techotopia.com/index.php?title=An_Android_Studio_Master%5C%2FDetail_Flow_Tutorial_-_Android_6&mobileaction=toggle_view_mobile⟩.

[18] Hazem Saleh. *MVVM architecture, ViewModel and LiveData (Part 1)* [online]. 2017 [cit. 2020-5-17]. Dostupný z: ⟨https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1⟩.

[19] Roman Pichlík. *Spring Framework – představení J2EE lightweight kontejneru* [online]. 2005 [cit. 2020-5-17]. Dostupný z: ⟨https://www.interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/⟩.

[20] *MVVM*. [online]. 2018 [cit. 2020-5-17]. Dostupný z: ⟨https://www.ackee.cz/blog/glossary/mvvm/⟩.

[21] Google. *An Android Studio Master/Detail Flow Tutorial - Android 6* [online]. 2018 [cit. 2020-5-17]. Dostupný z: ⟨https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html?m=1⟩.

[22] Google. *Save data in a local database using Room* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://developer.android.com/training/data-storage/room#java⟩.

[23] Přispěvatelé Wikipedie. *Singleton* [online]. 2020 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Singleton&oldid=18410280⟩.

[24] Keytel, LR; Goedecke, JH; Noakes, TD aj. Prediction of energy expenditure from heart rate monitoring during submaximal exercise. *Journal of sports sciences*. 2005, roč. 23, č. 3, s. 289–297.

[25] Guest, Richard; Miguel-Hurtado, Oscar; Stevenage, Sarah; Black, Sue. Exploring the relationship between stride, stature and hand size for forensic assessment. *Journal of forensic and legal medicine*. 2017, roč. 52, s. 46–55.

[26] Přispěvatelé Wikipedie. *Dolní propust* [online]. 2018 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/w/index.php?title=Doln%5C%C3%5C%AD_propust&oldid=16133314⟩.

[27] Movesense. *API reference*. 2020. Dostupný z: ⟨http://www.movesense.com/docs/esw/api_reference/#componentmax3000x⟩.

[28] Movesense. *Samples*. 2020. Dostupný z: ⟨https://bitbucket.org/suunto/movesense-device-lib/src/master/samples/hr_wakeup_app/⟩.

[29] Přispěvatelé Wikipedie. *CSV* [online]. 2019 [cit. 2020-5-17]. Dostupný z: ⟨https://cs.wikipedia.org/wiki/CSV⟩.