Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Horváth Noel

2022

Szegedi Tudományegyetem
Informatikai Intézet

Mobilalkalmazás fejlesztése Mi Band 3 okoskarkötőhöz

(Development of a mobile application for Mi Band 3 smart bracelet)

Szakdolgozat

Készítette:
Horváth Noel
programtervező informatikus BSc szakos hallgató

Témavezető:
Dr. Bilicki Vilmos
adjunktus

Szeged
2022

Feladatkiírás

A szakdolgozat során egy olyan mobilalkalmazást kellett fejlesztenem a Mi Band 3 okoskarkötőhöz, ami képes az általa mért adatokat feldolgozni, megjeleníteni, nyilvántartani és exportálni. Készítés során egy mindennapi használatra alkalmas hibrid applikáció megvalósítása volt a cél, amiben felhasználom a modern webes technológiákat.

Tartalmi összefoglaló

Téma megnevezése

Mi Band 3 okoskarkötő mérési adatait feldolgozni, nyilvántartani, megjeleníteni és exportálni képes webes technológiákkal megvalósított mobilalkalmazás készítése a szakdolgozat témám.

Megadott feladat megfogalmazása

Feladatom egy olyan mobilapplikáció fejlesztése webes technológiákkal, ami képes a Mi Band 3 okos karkötővel együtt működni.

Megoldási mód

Olyan felületeket implementáltam hozzátartozó logikával és szolgáltatásokkal a mobilalkalmazáshoz, amelyek elvégzik a mérési adatok megjelenítését, tárolását, exportálását, feldolgozását és korábbi adatok szinkronizálását. Az szoftver felhasználója több Mi Band 3 eszközt is párosíthat, melyek között tetszőlegesen válthat az applikáción belül.

Alkalmazott eszközök, módszerek

Az applikáció létrehozásához Angular és Ionic keretrendszereket használtam fel, mint webes technológiákat. A mobil natív funkcionalitásainak eléréséhez, mint a Bluetooth, tárhely és fájlkezelés a Capacitor segít egy hibrid alkalmazást eredményezve, ami Android operációs rendszeren működik. A Bluetooth kommunikáció és az során elérhető adatok lekérésének megvalósítása a Cordova Bluetooth LE Plugin felhasználásával történt meg. Adatok tárolása a Firebase Firestore által nyújtott valós idejű NoSQL adatbázis használatával valósult meg. Mi Band 3 által mért adatok vizualizációjára pedig

a Chart.js-t használtam. Fejlesztői környezetnek a WebStorm-ot és az Android Studio-t választottam.

Elért eredmények

A szakdolgozatom végére egy olyan alkalmazást sikerült létrehoznom, amely a Mi Band 3 eszközzel párosítva képes a használója korábbi, jelenlegi aktivitását és pulzusát feldolgozni, megjeleníteni, eltárolni és exportálni internet kapcsolat megléte nélkül is, így akár alkalmas sportolási tevékenységek követésére és egészségügyi felhasználásra is. A szoftvert elkészülte után a mindennapjaim és főleg edzéseim során használtam az aktivitásom és pulzusom követésére.

Kulcsszavak

Mi Band, Firebase, Bluetooth, Ionic, Angular, Capacitor

Tartalomjegyzék

Motiváció

Szakdolgozatom során olyan applikáció elkészítése volt a cél, amely a használata során amellett, hogy feldolgozza és megjeleníti a mérési adatait a Mi Band 3 eszköznek, egy modern, letisztul és felhasználóbarát élményt nyújt az okoskarkötő tulajdonosainak számára. A téma ötlete onnan merült fel, hogy a szakmai gyakorlatom végzése során sikeresen megismerkedtem az Anuglar és Ionic keretrendszerekkel és a mobil alkalmazás fejlesztés mindig is érdekelt, így a kettőt kombinálva egy hibrid szoftver létrehozás mellett döntöttem, így szakdolgozatom motivációja az alkalmazás elkészítésével párhuzamosan ezen technológiák elsajátítása és megfelelő alkalmazása. Ezek mellett egy részében funkcionális bővítést végeztem egy már meglévő egészségügyi projekthez, ami a Mi Band 3 okoskarkötővel működve már képes volt az általa mért lépésszám, elégetett kalória és megtett távolság adatainak lekérésére, feldolgozásra és adatbázisba való feltöltésére. Mivel eddig még nem került elő Bluetooth Low Energy fejlesztéseim során, így jó ötletnek véltem egy olyan applikáció megvalósítását, ami képes a Mi Band mérési adatainak feldolgozására, tárolására, exportálására és megjelenítésére.

1. Terület áttekintése

Az alkalmazás tervezésének elkezdése előtt egy piackutatás végeztem, amely során felmértem a már meglévő megoldásokat, amely során 3 olyan applikációt találtam, melyek a Mi Band 3 fő képességeit felhasználják az általa mért adatok feldolgozására, tárolására és megjelenítésére. Mielőtt bemutatnám röviden ezeket a szoftvereket előtte ismertetném a választott eszköz képességeit.

1.1. Mi Band 3 funkcionális képességei

Mi Band 3 egy hordozható tevékenységkövető, amely Xiaomi harmadik generációs okoskarkötője a Mi Band márkanév alatt. A termék 2018-ban került forgalomba a következő funkciókkal ellátva, amelyek az alvás, pulzus, sportolási és aktivitás tevékenységek valós idejű követése. OLED érintőképernyővel ellátott okoskarkötő alkalmas SMS, hívás és alkalmazás értesítések jelzésére rezgéssel. Emellett még alkalmas ébresztőként vibrálni, párosított telefont megtalálni a közelben, feloldani a párosított telefon képernyőzárát, értesíteni, ha tétlen a használója és naptári események emlékeztetésére. Vezeték nélküli kapcsolatot eléréséhez a Bluetooth Low Energy 4.2

technológiát használja és kompatibilis minden olyan eszközzel, ami rendelkezik Bluetooth 4.0 verzióval.

1.2. Meglévő alkalmazások az eszközhöz

1.2.1. Mi Fit

A Mi Fit egy hivatalos alkalmazás a Mi Band 3 okoskarkötőhöz, ami további más Huami által készített okosórákkal, okos mérlegekkel, cipőkkel és Band termékekkel párosítható össze. Használatához egy fiók szükséges, hiszen a feldolgozott adatok lokális tárhely helyett felhő alapú tárhelyben kerülnek eltárolásra. Az applikáció egyszerű és könnyen használható felülettel biztosítja az eszköz teljes funkcionalitásának kihasználását, de forráskódja nem elérhető a közösség számára. A szoftver vizuális reprezentációt nyújt az aktivitási, alvási, sportolási, pulzus mérési adatokhoz és monitorozást tesz lehetővé sporttevékenység végzésekor. Emellett konfigurálhatóak az eszköz specifikus beállítások az applikáción belül, mint például: a nyelv, idő és dátum formátum, szinkronizálás. időjárás, értesítések, lépésszám cél, DND mód, kijelző fényerejének csökkentése, naptár események, pulzusmérés. Beállításokon túl rendelkezik még használati animációkkal, amelyek segítenek a helyes eszköz használatához és lehetőséget nyújt a közeli eszköz keresésben, ha nem találnánk meg folyamatos vibrálást biztosítva. Egyedi funkciója a barátok rendszere, amivel más felhasználókat tudunk barátként felvenni és megtekinthetjük az aktivitásukat. Nem hiába a Mi Fit a legteljesebb alkalmazás, ami elérhető Mi Band 3 okoskarkötőhöz iOS és Android operációs rendszereken, hiszen az eszköz gyártója által fejlesztett és karbantartott.

1.2.2. Master for Mi Band

Master for Mi Band egy Android operációs és iOS operációs rendszereket futtató eszközökön is elérhető applikáció Mi Band okoskarkötőkhöz. BLACKNOTE által fejlesztett zárt forráskódú megoldás, ami hasonló funkcionalitásokat nyújt a Mi Fit alkalmazáshoz kevésbé modern megjelenéssel. Előbbivel ellentétben nem teljesen ingyenes alkalmazás, hiszen bizonyos funkcionalitások használatához fizetni kell, hogy elérhetőek legyenek, mint a reklámmentesség, testreszabhatóság és értesítések kontrollálása a párosított eszközzel. Applikáció regisztrációt nem igényel és működése során a mért adatokat a lokális tárhelyen tárolja.

1.2.3. Gadgetbridge

Előző két bemutatott alkalmazással szemben a Gadgetbridge egy nyílt forráskódú alkalmazás, amely több különféle Bluetooth Low Energy technológiát használó okos eszközzel működik. Ez az ingyenes, felhő és regisztráció mentes megoldás Android platformra készült Java nyelven. Mi Band 3 funkcionalitásának kihasználásának terén elmondható, hogy egyes eszköz specifikus beállítási lehetőségek nem érhetőek el, mint a hivatalos alkalmazásban, de fő funkcionalitások az aktivitás, alvás, pulzusmérés, tevékenység követés szintén megtalálhatók ebben az alkalmazásban is.

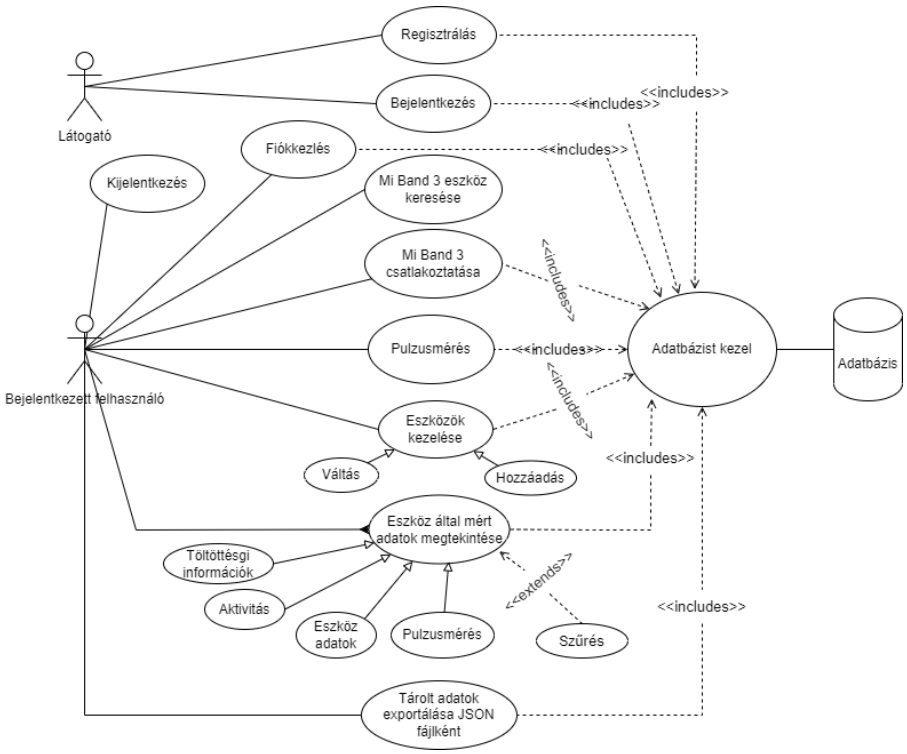
2. Funkcionális specifikáció

A saját nyílt forráskódú megoldás fejlesztését az előzőekben ismertetett alkalmazásokhoz hasonlóan, kíséreltem meg, hogy alkalmas legyen a Mi Band 3 okoskarkötő által mért adatok feldolgozására, tárolására és vizuális reprezentációjára Android operációs rendszer használata mellett. További Mi Band eszközzel való kompatibilitás és iOS platformon való használat megvalósítása a jövőben tervben van. Az alkalmazás a Mi Band 3 Companion nevet viseli, ami az okoskarkötővel való együttes használatra utal a hozzátartozó logóval együtt, mely a 2.1. ábrán tekinthető meg. Funkcionalitáshoz visszatérve felhasználókezelésben a Mi Fit példáját követi és regisztrációval felhasználói fiókok készíthetőek. Adattárolási szempontból az alkalmazás felhő alapú Firebase Firestore megoldást fogja alkalmazni, amely egyben alkalmas offline használatra is az adatok lokális gyorsítótárban való eltárolásával. A Mi Band 3 eszköz Bluetooth Low Energy kommunikációval csatlakozik a telefonhoz, mely után az adatok tárolása automatikusan megtörténik azok feldolgozása után. Az okoseszköz által tárolt korábbi adatok szinkronizálása automatikusan végrehajtódik első csatlakozás után, de lehetőség van manuális végrehajtásra is szükségszerűen anélkül, hogy újra kellene indítani az alkalmazást. A fejlesztett megoldás alkalmas az aktivitás vagy sportolási tevékenységek követésére, amely így akár egészségügyi területen is felhasználható lehet a pulzus mérési adatok követésével együtt.

2.1. ábra - Mi Band 3 Companion logója

2.1. Alkalmazás használatának ismertetése

Alkalmazásban végrehajtható műveleteket egy Use-case diagrammal fogom szemléltetni, amely a legalkalmasabb erre a feladatra. Ez tartalmazza, hogy a felhasználó milyen funkcionalitásokat tud véghez vinni az alkalmazás használata során. A fejlesztett applikáció használati eset diagramját a 2.2. ábra szemlélteti.



2.2. ábra - Use-case diagram

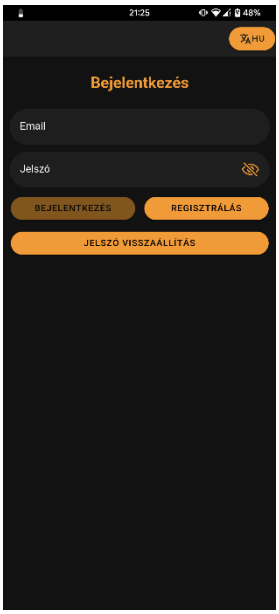
A fenti ábrán láthatóak egy adott felhasználó által elvégezhető műveletek és hogy azok igényelnek-e adatbázissal való kommunikációt. A Firebase által biztosított felhő alapú adatbázis online elérhetőség mellett offline módot is biztosít, így internet elérés megléte nélkül is használhatóak maradnak az adatbázis kommunikációt igénylő

funkcionalitások a felhasználói autentikáció kivételével. Az alkalmazás a felhasználók között két entitást különböztet meg a bejelentkezés státuszának tekintetében, amely során ilyen a látogató és a bejelentkezett felhasználó. Látogató számára két művelet érhető el a regisztráció és a bejelentkezés. Sikeres regisztráció és első bejelentkezés után elérhető lesz a felhasználó számára az összes funkcionalitása az alkalmazásnak, ha csatlakoztatunk legalább egy Mi Band 3 eszközt. Amíg a felhasználó nem jelentkezik ki addig az automatikus belépés nem igényel online adatbázis elérést, tehát offline módon használható marad az alkalmazás. Kapcsolódás létesítése egy okoskarkötővel keresés utáni kiválasztással tehető meg, ha található a közelben. Ez nem igényel adatbázis műveletet, de a kapcsolódás már igen, ami elmenti az eszköz információit és a csatlakozás időpontját. Az utóbbi szükséges ahhoz, hogy az alkalmazás újraindítása után az eszközök között meglehessen különböztetni a legutoljára használat okoskarkötőt, és az induláskor egy gyors kapcsolat létrehozását lehessen véghez vinni, ha a felhasználó nem jelentkezett ki. Egy felhasználó több eszközt is párosíthat a fiókjához, amik között tetszőlegesen válthat igényeinek megfelelően. Eszközkezelés igényel adatbázis olvasást és írást, mert az eszköz adatai tárolásra kerülnek a felhasználó adatainál. Kijelentkezés során nincs szükség az adatbázissal való kommunikációra, hiszen az alkalmazás kezeli ezt a folyamatot. Mi Band 3 által mért adatok 4 kategóriába csoportosíthatóak, amelyek a töltöttségi információk, az aktivitás, a pulzusmérés és az eszköz specifikus információk. Ezek részletesebb ismertetése a későbbi fejezetekben kerülnek tárgyalásra.

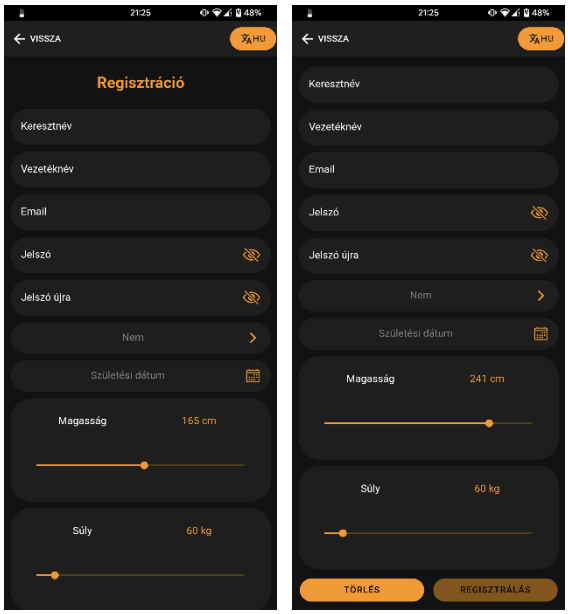
2.2. Felhasználó autentikálása

Autentikáció egy fontos egysége az alkalmazásnak, hiszen a mérések egy adott felhasználóhoz tartoznak, így tárolásához szükség van egy felhasználói fiókra. Az alkalmazást elindítva egy a 2.3. ábrán is látható bejelentkezési képernyő fogad, ahol az e-mail cím és jelszó megadásával be lehet jelentkezni, ha már létezik felhasználói fiók hozzá. Abban az esetben, ha nincsen akkor a „Regisztrálás” gomb megnyomása után a 2.4. és 2.5. ábrák által szemléltetett regisztrációs oldalon lévő űrlap kitöltésének és elküldésének elvégzésével lehet regisztrálni. Sikeres végrehajtás után egy aktivációs link kerül kiküldésre a megadott e-mail címre, aminek egyszeri megnyitása szükséges a fiók aktiválásához. Abban az esetben, ha az e-mail címre küldött link lejárt, akkor lehetőség van új aktivációs e-mail-t kérni bejelentkezés után felugró ablak segítségével. Ilyenkor nem lesz a felhasználó átirányítva másik oldalra, sem bejelentkeztetve, hiszen csak

verifikáció után lehetséges annak megtekintése. Regisztrációs űrlap felső részének kitöltése a keresztnév, vezetéknév, e-mail cím, jelszó és jelszó újra mezők helyes kitöltésével lehetséges. Ezek mellett szükséges még a regisztrálni kívánt felhasználónak a nemét, születési dátumát, magasságát és a súlyát megadnia, amely adatok a Mi Band 3 eszköz kapcsolódás utáni inicializálásához szükségesek, hogy a megtett távolságot és elégetett kalóriát ezek segítségével ki tudja számolni és ezen információkat meg is jelenítse a Mi Band 3 eszköz a kijelzőjén. A regisztrációs felület lehetőséget biztosít a megadott beviteli adatok törlésére ez által nem szükséges manuálisan kitörölni minden mező tartalmát a felhasználónak, hanem elég a „Regisztrálás” gomb mellett található „Törlés” gomb megnyomása.



2.3. ábra - Bejelentkezés képernyő



2.4. és 2.5. ábrák - Regisztrációs képernyő

A bejelentkezés oldalon lehetőség van még a 2.6. ábrán látható jelszó visszaállítás oldalra navigálni, ahol egy e-mail cím adható meg. A beviteli mező kitöltése és a „Kérés” gomb megnyomása után, egy jelszó visszaállító levél kerül kiküldésre, amivel új jelszó adható meg. Fontos, hogy a regisztrációhoz, első bejelentkezéshez, jelszó visszaállításhoz és verifikációs link kéréshez internetkapcsolat megléte szükséges. Automatikus bejelentkezés esetén offline mód támogatott a Firebase Authentication által, ezáltal nem kell minden alkalmazás indításkor bejelentkezni. Az alkalmazás biztonsága érdekében a jelszóvisszaállítás, regisztrálás és verifikációs link kérés óránként csak egyszer

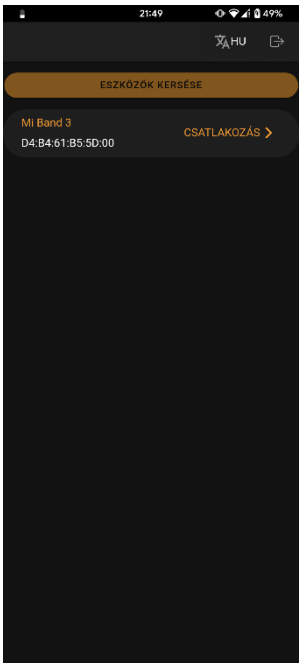
használható sikeresen, hiszen többszöri használat rosszindulatú célra is felhasználható lenne.

2.6. ábra - Jelszó visszaállítás képernyő

2.3. Eszköz keresése és csatlakozás

Bejelentkezés után, ha az alkalmazás nem talál elmentett eszközt a fiókhoz párosítva akkor egy eszköz keresési és csatlakozási felület jelenik meg. „Eszközök keresése” gomb megnyomása után megjelennek a talált Mi Band 3 eszközök, amelyekhez csatlakozhatunk a „Csatlakozás” gomb megnyomásával. Fontos, hogy a keresés vagy csatlakozás megkezdéséhez a telefonon bekapcsolt Bluetooth állapot szükséges a megfelelő engedélyek engedélyezése mellett. Android 11 és korábbi verziók esetében a helymeghatározás is szükséges az eszközök kereséséhez. Kényelmessé téve a Bluetooth és helymeghatározás bekapcsolását az alkalmazás automatikusan elvégzi, ha lehetséges. Keresés végrehajtása pár másodpercet vesz igénybe és a talált eszközök megjelennek jelerősségük szerint csökkenő sorrendben. Egy sorban megjelenő kártya egy eszköznek felel meg, ahol látható az eszköz neve, a MAC-címe és egy „Csatlakozás” gomb a 2.7. ábrán is látható módon. A megfelelő eszközhöz tartozó gomb megnyomása után a csatlakozás megkezdődik és sikeres párosítás és csatlakozás után a főoldalra kerül a felhasználó. Minden keresés után 10 másodpercet kell várnia a felhasználónak egy újabb végrehajtására, melyre azért van szükség, mert a Android 6-nál újabb verziók esetén

korlátozott számú Bluetooth szkennelés indítható percenként biztonsági okokból. Ennek a hátralévő idejét az „Eszközök keresés” gombon megjelenő számoló szemlélteti, amely során a gomb letiltásra kerül.

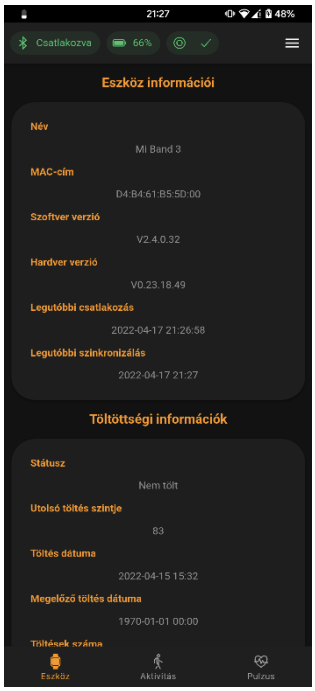


2.7. ábra - Keresési találatok megjelenítési felülete

2.4. Főoldal felülete

Sikeres kapcsolat létesítése után az felhasználó a 2.8. ábrán látható főoldalra kerül navigálásra, amely 3 fő részből áll, melyek egyike a tetején megjelenő eszköztár, ahol megtalálható az eszközzel való kapcsolat státusza, a csatlakoztatott eszköz töltöttségi szintje, a Mi Band 3 eszköz szinkronizálásának státusza és egy menü gomb. Ezek az elemek balról jobbra jelennek meg az előbbi felsorolásnak megfelelően. Csatlakozás státusz indikátor lehetőséget biztosít megnyomásra történő leválasztásra vagy csatlakozásra a jelenleg használat eszközről a Bluetooth kapcsolat státuszának megfelelően. Abban az esetben, ha nincs párosítva Mi Band 3 okoskarkötő akkor kapcsolódás létesítése ellenkező esetben kapcsolat bezárása indul. Hasonlóan a szinkronizálás státusz indikátor is használható megnyomással, amivel manuálisan elindítható az okoskarkötő korábbi adatainak szinkronizálása, ha nincs folyamatban egy korábbi. Alsó sávban egy navigációs menü található, mely a navigálást teszi lehetővé a főoldalon. A navigációs sáv és eszköztár közötti részben pedig a főoldal aloldali vagy

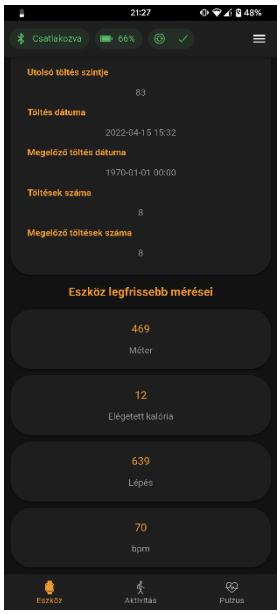
másképpen gyermek oldalai jelennek meg, amelyek rendre az eszköz, aktivitás és pulzus oldalak.



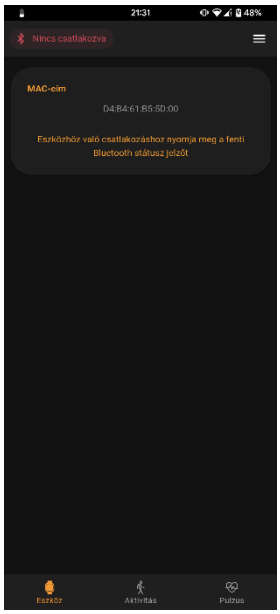
2.8. ábra - Főoldalon belüli Eszköz oldal felülete

2.5. Eszköz oldal

Eszköz oldal a már korábban említett főoldal gyermek oldala, ahol megtekinthető az eszköz neve, a MAC-címe, a szoftver verziója, a hardver verziója, a legutolsó csatlakozás és a szinkronizálás dátuma, amit a 2.8. ábra szemléltet. Ezen felül megtalálhatóak még a töltöttségi információk is (2.9. és 2.8. ábrák), amik rendre a töltöttségi státusz, utolsó töltés szintje, utolsó és azt megelőző töltés dátuma és a hozzájuk tartozó töltések száma. Eszköz által mért legfrissebb adatok is megjelennek az oldal alsó részében lentebb görgetés után. Ezen információk a megtett távolság méterben, elégetett kalória, lépések száma és pulzusszám a felsorolás sorrendjében fentről lefelé, melyek a 2.9. ábrán láthatóak. Ezen információk összessége csak abban ez esetben jelenik meg, ha az eszköz csatlakoztatva van, hogy biztosítva legyen az adatok valós idejű frissítése változásuk esetén. Ellenkezőleg a 2.10. ábrán látható felület fog megjelenni, ahol csak a használt eszköz MAC-címe és egy instrukció jelenik meg arról, hogy a felhasználó, hogyan kapcsolódhat újra a jelenlegi eszközéhez.



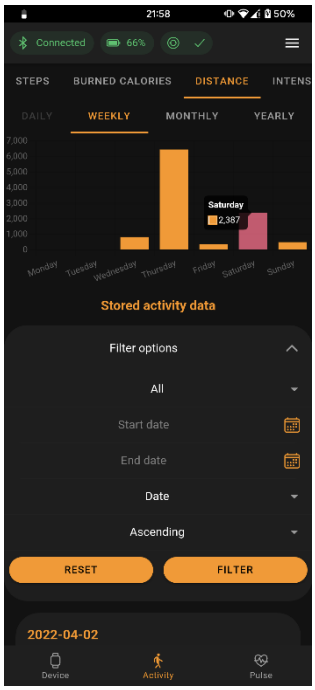
2.9. ábra - Eszköz oldal alsó része



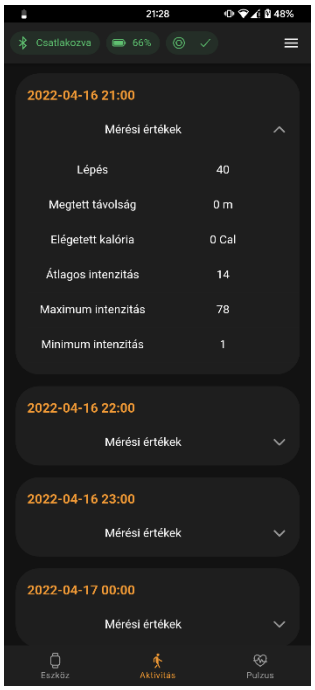
2.10. ábra - Eszköz oldal csatlakoztatott eszköz nélkül

2.6. Aktivitás oldal

Mérések feldolgozása és mentése automatikusan a háttérben történik meg, amint új adatot szolgáltat a csatlakoztatott eszköz. Ezen adatok típusának egyike az aktivitás, amelyet az ennek megfelelő névvel ellátott oldalon tekinthet meg a felhasználó az alsó navigációs sáv segítségével. Megtekintve a 2.11. ábrán látható képernyő fogad egy oszlop diagrammal és egy szűrési felülettel. Egy aktivitás mérés lépésszámból, elégetett kalóriából, megtett távolságból és intenzitásból tevődik össze, amelyeket heti, havi és éves bontásokban tekinthetőek meg. Vizualizációs felület felett található oldalra görgethető gombok segítségével kontrollálható a megjelenített adatok halmaza. Napi bontás is elérhető azon adatok számára, amelyeket az eszköz nem csak 24 órás összesítésben szolgáltat. Ez alól kivétel az elégetett kalória és megtett távolság. Intenzitás 0 és 255 közötti intervallumból vehet fel értéket, ezért számára elérhetőek az átlag, a minimum és a maximum diagram beállítási opciók a pontos reprezentálás érdekében. Adatbázisban tárolt aktivitások listázása a szűrési felület segítségével tehető meg. Lenyitása után a 2.11. ábrán látható opciók érhetőek el. Ezen feltételek a később bemutatásra kerülő aktivitás osztály adattagjai szerinti szűrést tesznek lehetővé sorrend megadásával kiegészítve. A „Filter” gomb megnyomása után az elemek 10-es nagyságrendben jelennek meg eleinte, de lefele görgetve további tölthető be. Egy találat a 2.12. ábrán látható lenyíló kártya formájában jelenik meg ahol az adott aktivitáshoz tartozó információk tekinthetőek meg.



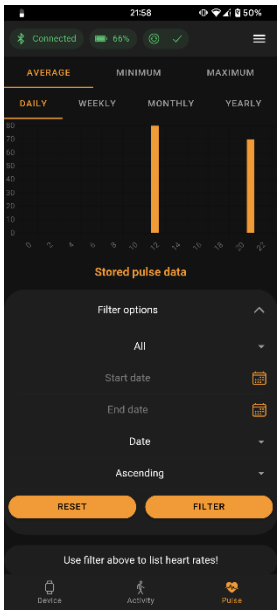
2.11. ábra - Aktivitás oldal



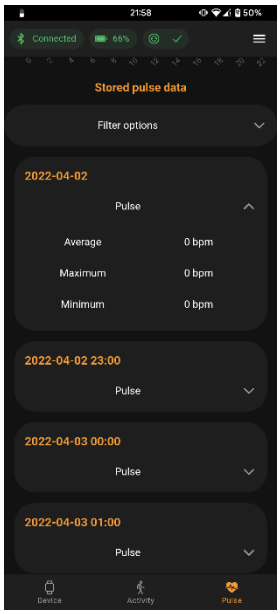
2.12. ábra - Aktivitás információi

2.7. Pulzus oldal

Aktivitás képernyővel megegyező felülettel jelenik meg a pulzus oldal a 2.13. ábrán látható módon, ahol a pulzus mérési adatok tekinthetőek meg vizualizációval és szűrési lehetőséggel. A korábban bemutatott oldalhoz hasonlóan az oszlop diagram adathalmaza beállítható a felette lévő opciók segítségével, ahol lehetőség van napi, heti, havi és éves bontások kiválasztására is. A később tárgyalásra kerülő pulzus osztály olyan adattaggal rendelkezik, amely tárolja az átlagos, minimum és maximum pulzusszám értéket, hiszen ezen adat is 0-255 közötti intervallumból vehet fel értéket. Szűrési feltételek megadása a pulzus adattagjai szerinti opciók kiválasztásával történhet meg. Listázás elvégzése után 10 darab pulzus kártya kerül megjelenítésre maximum. További betöltésére, ha van lehetőség akkor legörgetéssel tehető meg és egy lenyíló pulzus kártya tartalmazza a pulzus információkat az osztály adattagjainak megfelelően a 2.14. ábrán látható módon.



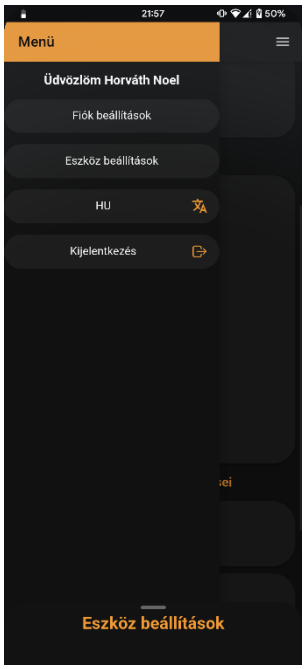
2.13. ábra - Pulzus oldal



2.14. ábra - Pulzus információi

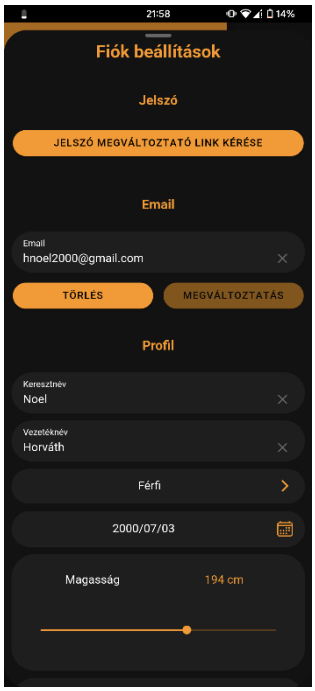
2.8. Menü és beállítások

Menü a főoldalon található eszköztár jobb szélén megjelenő gomb megnyomásával hívható elő. Megnyomás után egy balról benyúló felület jelenik meg a 2.15 ábrán látható módon, ahol lehetésége van a felhasználónak kijelentkeznie, beállítások felületet megjeleníteni és az alkalmazás nyelvét megváltoztatni. Ezek mellett egy szöveg is található a gombok felett, ami a jelenlegi felhasználót üdvözli a nevével.

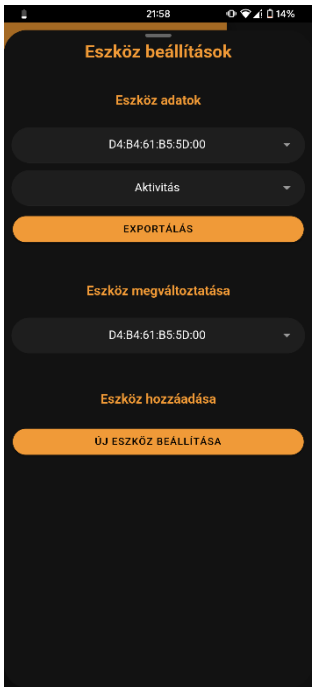


2.15. ábra - Főoldali menü

Az alkalmazás támogatja a magyar és angol nyelvet, így a nyelv megváltoztatás gomb megnyomása után felugró felületen ezen opciók közül lehet választani, ahol a jelenlegi nyelv kiemelt háttérszínnel van ellátva. A 2.16. ábrán látható fiók beállításokon belül módosítható a regisztráció során megadott e-mail cím és profil adatok, amelyek a Mi Band 3 eszközhöz szükségesek kalória és megtett távolság számításához az eszközön. Ezek mellett még jelszó megváltoztatás is kérhető, mely során egy link kerül kiküldésre automatikusan a felhasználó e-mail címére, amivel módosíthatja a jelszavát. Hasonlóan a jelszóvisszaállítás kéréshez ez a funkció is csak óránként egyszer vehető igénybe. 2.17 ábrán látható eszköz beállítások felületen megváltoztatható a jelenleg használt eszköz és emellett hozzáadható új eszköz is, amely a korábban bemutatott eszköz párosítás oldalra navigál. Ezen kívül elérhető még az adatok exportálása funkció is, amivel aktivitás vagy pulzus mérések menthetőek le a telefon tárhelyére a kiválasztott Mi Band 3 eszközhöz tartozó adatbázisban tárolt adatok közül JSON formátumban. Mentés után az elkészített fájl a Dokumentumok mappán belül létrehozott MiBand3Companion mappába kerülnek a mérés típusának megfelelő almappába rendezve. Az exportált JSON állomány neve tartalmazza az elkészítés dátumát és tartalma, pedig egy a formátumnak megfelelő adatstruktúra.



2.16. ábra - Profil beállítások



2.17. ábra - Eszköz beállítások

3. Felhasznált technológiák

A szakdolgozat céljául kitűzött alkalmazás fejlesztése során számos modern technológia lett felhasználva. A következőekben ezen fejlesztéshez segítséget nyújtó eszközöket, keretrendszereket és plugin-okat mutatom be röviden.

3.1. Angular

Az Angular egy Google által fejlesztett nyílt forráskódú keretrendszer webes alkalmazások készítésére TypeScript nyelven. Használatával úgynevezett egyoldalas (single-page) alkalmazások hozhatók létre, amik dinamikusan egy oldalt frissítenek több oldal betöltése helyett. Fő egységei a komponensek, service-ek és modulok. Modulok tartalmazzák egyes komponensek által használt komponensek és service-ek listáját. A komponensek felelősek a megjelenítés mögötti logikáért és biztosítják a megjelenítendő adatokat a hozzátartozó HTML sablonokkal. Ezen nézetek közötti navigálásról, pedig a Routing gondoskodik. Minden alkalmazás rendelkezik egy gyökér komponenssel, ami egy komponens hierarchiát valósít meg. A szolgáltatások (service) komponensek vagy további service-ek egymással való kommunikálására használható. Ezek az egységek mind dekorátorokkal ellátott osztályok. A keretrendszer által használt TypeScript hivatalosan a JavaScript bővítése, amivel szemben egyik előnye, hogy egy statikusan típusos nyelv, vagyis a változók és kifejezések típusa előre egyértelműen eldönthető, rögzítettek és futás során nem változnak, mint JavaScript esetében.

3.2. Ionic

Ionic szintén egy nyílt forráskódú keretrendszer, ami lehetővé teszi a webes technológiák felhasználásával mobil és asztali alkalmazások fejlesztését egy kódbázis használata mellett. Fő fókusza a felhasználói felületen van, amihez különböző komponenseket biztosít Angular, React és Vue,js keretrendszerek integrációjával. A keretrendszer segítségével hibrid alkalmazások valósíthatóak meg, amely webes és natív kódok keverékéből áll elő a natív alkalmazások teljesítményének és kinézetének megközelítéséhez. Komponensei előre implementáltak és úgy lettek megtervezve, hogy minden kijelzőn megfelelően működjenek. Natív funkcionalitás webes környezetben való eléréséhez Cordova és Capacitor plugin-ok használhatóak, amiket maga az Ionic vagy a közösség fejleszt.

3.3. Capacitor

Capacitor egy nyílt forráskódú projekt, ami a modern webalkalmazásokat natívan futtatja iOS, Android, Electron és webes platformokon, amely során egy könnyen használható interfészt nyújt a natív funkcionalitások elérésében. Ezzel a technológiával a hibrid alkalmazás különböző platformokon, úgy valósul meg hogy az adott platform natív alkalmazásában egy böngészőmotort használva jeleníti meg a böngészőre szánt tartalmat. A natív funkcionalitást a natív alkalmazásból kezeli és a webes kóddal plugin-ok segítségével kommunikál. A generált natív kód módosításához nem kell külön Cordova specifikus fájlokat konfigurálni és kompatibilitást biztosít a Cordova plugin-ok számára.

3.4. Cordova Bluetooth LE Plugin

Cordova Bluetooth LE Plugin Bluetooth Low Energy natív funkcionalitást biztosító plugin hibrid és webes alkalmazások számára Capacitor és Cordova technológiák használata mellett. Többi hasonló megoldással szemben ez rendelkezik a legteljesebb megvalósítással a Bluetooth LE eszközökkel való interakcióhoz Android, iOS és Windows rendszereken, mivel támogatja a perifériás és központi módokat, és lefedi az Android és iOS rendszeren elérhető API[1]-k többségét.

3.5. Firebase

Firebase egy modern felhőalapú szolgáltatás, ami a fejlesztők számára különböző eszközöket és szolgáltatásokat nyújt, amik segítenek az alkalmazás fejlesztésében, működtetésében és karbantartásában, amiket a Firebase SDK segítségével használhatunk alkalmazásokban. Cloud Firestore, ami egy NoSQL adatbázis és biztosítja a valós idejű szinkronizálás és offline adattárolást és elérést. Fontosabb szolgáltatásai még a Firebase Authentication, ami segít a felhasználók autentikációjában és a Cloud Functions for Firebase, ami pedig egy szerver nélküli keretrendszer, amely képes automatikusan futtatni JavaScript és TypeScript kódokat bizonyos események változásainak következtében. Ezeken felül további analitikai, hosztolási és gépi tanulással kapcsolatos eszközök érhetőek még el.

[1] Application programming interface (API) – alkalmazásprogramozási interfész

3.6. Chart.js

ChartJS egy közösség által karbantartott nyílt forráskódú könyvtár adatok vizualizációjára JavaScript nyelven. Segítségével 9 féle diagram típus használható, amelyek a line, bar, radar, doughnut and pie, polar area, bubble, scatter, area chart és az előbb említettek kevert változatai. Tovább ezen elemek tetszőlegesen testreszabhatóak a kívánt megjelenítés elérésében, melyeket HTML5 vászon segítségével jeleníti meg.

4. Alkalmazás szerkezeti felépítése

4.1 Modulok

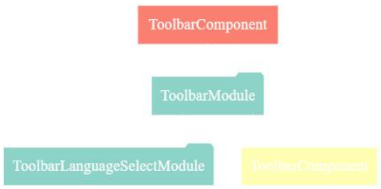
Angular használatával moduláris alkalmazások hozhatóak létre, melyet a keretrendszer saját moduláris rendszere biztosít, amit NgModules-nak neveznek. Egy NgModul a logikailag összefüggő komponensek, szolgálatatások és direktívák összessége. Segítségükkel használni tudunk más modulok által exportált funkciókat importálással, illetve exportálni egy modult más modulok számára. Egy Angular alkalmazás alapból rendelkezik egy modul osztállyal, amelyet gyökér modulnak neveznek. Minden modul külön fájlban definiált, amik a név.module.ts elnevezést követik és kódban egy dekorátorral ellátott osztályok. Egy alkalmazás több funkcionális modullal rendelkezhet, melyek a gyökér modulba vagy annak gyermek moduljaiba importálhatóak egy hierarchiát létrehozva, melynek mélysége bármilyen mély lehet. A Mi Band 3 Companion által használt modulok a következőek:

- ActivityItemModule
- ActivityModule
- ActivityRoutingModule
- AppModule
- AppRoutingModule
- CustomChartModule
- DataChartModule
- DatePickerModule
- DeviceModule
- DeviceRoutingModule
- DeviceSetupModule
- DeviceSetupRoutingModule

- HeartRateItemModule
- HeartRateModule
- HeartRateRoutingModule
- HomeModule
- HomeRoutingModule
- HomeTabBarModule
- ItemsFilterModule
- LoginModule
- LoginRoutingModule
- PasswordInputModule
- RangeItemModule
- RegisterModule
- RegisterModule
- RegisterRoutingModule
- ResetPasswordModule
- ScannedDeviceItemModule
- ToolbarLanguageSelectModule
- ToolbarMenuDeviceSettingsModalModule
- ToolbarMenuModule
- ToolbarMenuUserSettingsModalModule
- ToolbarModule

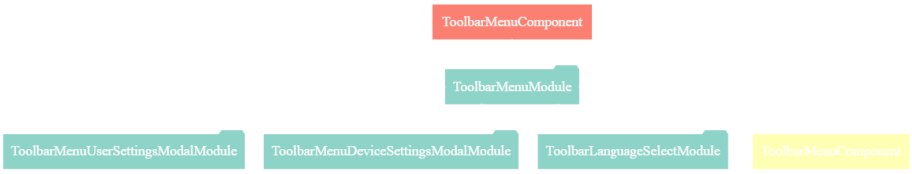
4.1.1 ToolbarModule

Az alkalmazás 32 modulja közül a ToolbarModule és annak egy alárendelt modul tartalmát fogom ismertetni a következőekben. Az előbbiek során említett modul a képernyőn megjelenő fenti sávban található eszköztár által használt funkcionalitásait és felületét fogja egységbe. Ábrája a 4.1. képen látható, amely során látható, hogy importálja a ToolbarComponent komponenst, amely a megjelenést és a hozzátartozó logikát tartalmazza és a ToolbarLanguageSelectModule-t, ami a nyelv megváltoztatás funkcióját és hozzátartozó felületet szolgáltatja. Továbbá a ToolbarModule exportálja a ToolbarComponent komponenst más komponensekben való használathoz.



4.1. ábra - ToolbarModule felépítése

ToolbarMenuModule az előbb tárgyalt modul gyermek modulja, mely 4.2. ábrán látható módon importálja a korábban említett ToolbarLanguageSelectModule-t, illetve a profil és az eszköz beállítások funkcionalitását tartalmazó ToolbarMenuUserSettingsModule-t és ToolbarMenuDeviceSettingsModule-t. Emellett exportálja még a ToolbarMenuComponent komponenst további használatra más egységekben való használatra.



4.2. ábra - ToolbarMenuModule felépítése

4.2. Interfészek

A modulok mellett a MI Band 3 Companion fontos építőelemei az interfészek, amelyek használatára lehetősége biztosít az alkalmazás által használt objektum orientált TypeScript programozási nyelv. Ezen absztrakt típusok fontosok hiszen segítenek elkülöníteni a megvalósítást a strukturális felépítéstől azzal, hogy megadják milyen adattagokkal és metódusokkal rendelkezhet egy objektum. Fontosabb jellemzői, hogy örököltethetőek és kiegészíthetőek, amivel flexibilitást és újrahasznosíthatóságot biztosítanak. Interfészek több további interfészt egészíthetnek ki a meglévő tagjaik kibővítésével. Implementálásuk osztályok által történhet meg az adattagjainak és metódusainak öröklésével és megvalósításával. A Mi Band 3 Companion alkalmazás fejlesztése során 30 interfész készült el, melyek az applikációban használt adatmodellek és szolgáltatások osztályainak struktúráját írják le külön fájlokba rendezve. A következő

alfejezetekben az előbb említett két típusból egy-egy fontosabb interfész kerül pontosabb ismertetésre.

4.2.1 IEntityModel

IEntityModel egy olyan generikus interfészt valósít meg, amely alapja az összes adatmodell interfésznek és így egyben osztálynak is és generikussága abban rejlik, hogy a paraméter listájában egy olyan generikus T típust vár, amelyet a metódusainak pontosabb leírására használ fel ezzel biztosítva az egység újrahasználhatóságát különböző típusokkal. Az interfész kódja a nevével megegyező IEntityModel.ts fájlban található meg, amit használni exportálással lehet más kódegységekben az export kulcsszóval való ellátottságnak köszönhetően. Felépítése során az IEntityModel 1 adattagot és 3 metódust definiál, melyet a 4.3. ábra szemléltet. Tagjai opcionálisak, vagyis az implementálásuk nem kötelező, így biztosítva azt, hogy az általa kiegészített interfészek használhatóak maradjanak metódus implementálás nélkül típusként is.

```
export interface IEntityModel<T> {
    id?: string;
    copy?(other: T): void;
    toString?(): string;
    isEqual?(other: T | undefined): boolean;
}
```

4.3. ábra - IEntityModel interfész felépítése

Interfész tagjainak szerepe az adatmodellekben:

- id: Egy szöveges azonosító az adatmodell számára.
- copy: A paraméterében kapott példányt lemásolja az jelenlegi osztály példányba.
- toString: Visszaadja az osztály példány szöveges reprezentációját.
- isEqual: Összehasonlítja a paraméterben megadott objektumot a jelenlegi osztály példánnyal.

4.2.2 IFirestoreBase

Az IFirestoreBase az előző adatmodell interfészhez hasonlóan is egy generikus interfész, amely szintén a nevével megegyező IFirestoreBase.ts fájlban került megvalósításra. Struktúrája a FirestoreBaseService nevezetű absztrakt osztály alapját képzi az egyes metódusok szignaturájának leírásával, melyet az összes további Firestore szolgáltatás

örököl ezen az osztályon keresztül. Az interfész a 4.4. ábrán is látható add, get, getWithValueChanges, list, listWithValueChanges, delete, update, paginate metódusok definíciójával rendelkezik. A visszatérési érték tekintetében találhatóak Observable típussal rendelkező függvények, amelyek során egy adatfolyamon keresztül lehet elérni az adatokat feliratkozással. Másik fajtája azon metódusok, amelyek Promise-ba csomagolva szolgáltatják vissza az adatot aszinkron módon. Pontosabb ismertetése ezen függvényeknek az implementációt megvalósító FirestoreBaseService szolgáltatás tárgyalása során kerül sor a következő (4.3.) Szolgáltatások alfejezetben.

```typescript
import { Observable } from 'rxjs';
import { PaginateOptions, PaginateResult, QueryOption } from '../../../shared/types/firestore.types';

export interface IFirestoreBase<T> {
    add(item: T, id?: string): Promise<void>;

    get(id: string): Promise<T | undefined>;

    getWithValueChanges(id: string): Observable<T | undefined>;

    list(queryOptions?: QueryOption<T> | QueryOption<T>[]): Promise<T[] | undefined>;

    listWithValueChanges(queryOptions?: QueryOption<T> | QueryOption<T>[]): Observable<T[] | undefined>;

    delete(queryOptions: { id: string } | QueryOption<T> | QueryOption<T>[]): Promise<void>;

    update(queryOptions: { id: string } | QueryOption<T> | QueryOption<T>[], data: Partial<T>): Promise<void>

    paginate(options: PaginateOptions<T>): Promise<PaginateResult<T> | undefined>;
}
```

4.3. ábra – IFirestoreBase interfész felépítése

4.3. Szolgálatások (Service)

Angular keretrendszerben a szolgáltatások egy specifikus feladat elvégzésére létrehozott dekorátorral ellátott osztályok. Fő célja, hogy biztosítsa az alkalmazás komponenseihez a különféle metódusokat és adatokat egy különálló egységként, így megtartva a modularitást és az újrahasználhatóságot. Alapértelmezetten minden szolgálatásból az alkalmazás életciklusa során egy példány keletkezik. Egy komponensben használt szolgáltatások függőségekként kerülnek injektálásra az Angular Dependency Injection mechanizmusával, mely után elérhetőek lesznek az adott service-ek osztályai a komponensen belül. Elsősorban a komponensek közös logikájának kiszervezésére használatos, de érdemes figyelni arra, hogy az ellátott folyamatok egy egységhez tartozzanak.

Következtekben a Mi Band 3 Companion főbb szolgáltatásait, azok fontosabb metódusait és működésüket fogom ismertetni. Minden szolgáltatás külön fájlban lett megvalósítva a név.service.ts elnevezést követve biztosítva az Angular által ajánlott szervezettséget és átláthatóságot.

4.3.1. BLE Connection Service

A BLE Connection Service olyan metódusokat tartalmaz, amelyek a Mi Band 3 eszközzel való kommunikációt teszik lehetővé az alkalmazásban. A szolgáltatás egy absztrakt osztályból pontosabban a BleBaseService osztályból származik, amely minden Bluetooth Low Energy (BLE) szolgáltatás ősosztálya.

Örökölt metódusai:

- write(): Paraméterben egy MAC-címet vagy IDevice objektumot, egy szolgáltatás címet vagy Service objektumot, egy karakterisztika címet vagy Characteristic objektumot, egy küldeni kívánt előjel nélküli egészek bájt tömbjét és egy írás típusát várja. A metódus a kapott értéket átkonvertálja kódolt szöveggé és azt az eszköz szolgáltatásában található karakterisztikájának elküldi a megadott választípusnak megfelelően.
- read(): A függvény egy MAC-címet vagy IDevice objektumot, egy szolgáltatás címet vagy Service objektumot és egy karakterisztika címet vagy Characteristic objektumot vár, ahonnan értéket olvas ki, majd visszatér azzal.
- subscribe(): Az előbb említett metódus paraméterezésétől annyiban tér el, hogy egy megfigyelőt (observer) vár és feliratkozik a szolgáltatás karakterisztikájára az eszköznek. A feliratkozás alatt szolgáltatott adatok feldolgozására pedig átadja a megfigyelőt.
- unsubscribe(): A paraméterben megadott karakteriszikáról leiratkozik, így a további értesítések ki lesznek kapcsolva az eszköz által.
- bytesToInt(): A függvény a kapott előjel nélküli egészekből álló 8 bites számok tömbjét alakítja át 10-es számrendszerbeli egész számmá.
- bytesFromInt(): Metódus a paraméterben kapott számot átalakítja egy előjel nélküli 0-255 érték tartománnyal rendelkező, vagyis 8 bites egész számok tömbjévé. Opcionális megadható a méret is, amely megadja, hogy milyen hosszú lehet a tömb. Ha nem lenne elég hosszú a tömb a szám átkonvertálása után akkor

0 bájtos értékekkel tölti fel azt, amíg a mérete nem éri el a paraméterben megadott szám értékét.

- dateTimeFromBytes(): A függvény a hívása során megadott 8 bites előjel nélküli egészek tömbjéből készít el egy Date objektumot a megadott kezdő tömb indextől kezdve. Visszatérésekor pedig egy FireTimestamp objektummal tér vissza, amit a Date objektumból konvertál át.
- dateTimeToBytes(): Egy Date vagy FireTimestamp objektumot vár a metódus paraméterként, amely dátumot átalakít egy 8 bites előjel nélküli egészek tömbjévé. Opcionális beállítások paraméterével megadható, hogy szerepeljen-e az eredményben a hét napjának száma, a 256-os töredéke a másodperceknek, a nappali időmegtakarítás mértéke vagy az időzóna értéke. Az options paraméter all adattagjának igazra állításával esetén az összes előbb felsorolt érték hozzáadásra kerül a visszatérési tömbhöz.

Gyerek osztály (BleConnectionService) metódusai:

- authenticateMiBand(): Paraméterben egy IDevice objektumot vár, ami a csatlakoztatni kívánt eszköz adatait tartalmazza. A függvény a csatlakozás utáni autentikációt hajtja végre, amely sikeres végrehajtása során az eszköz teljes funkcionalitása elérhetővé válik. Végrehajtás előtt feliratkozik az autentikációs szolgáltatás karakterisztikájára az értesítések fogadásához, majd az eszköz által küldött válaszok alapján végrehajtja a megfelelő adatok írását válaszul. Lépések pontosabb tárgyalására későbbi fejezetben kerül sor.
- createTimeoutForEnableBl(): Paraméterben milliszekundumok számát várja, amellyel egy várakozást létrehoz és végezte után pedig visszatér.
- enableBl(): Az függvény hívása során egy maximum várakozási időt vár másodpercekben, amivel a bekapcsolás utáni várakozás idő adható meg, amire azért szükséges, mert a Bluetooth bekapcsolás ideje nem minden eszköznél történik meg ugyanannyi idő alatt. A metódus bekapcsolja a Bluetooth funkciót az eszközön, ha nem volt még bekapcsolva és visszatér a bekapcsolás sikerességével.
- blAndPermissionChecks(): Metódus segítségével ellenőrizhető a Bluetooth állapota, ha nincs bekapcsolt állapotban akkor meghívja az enableBl() függvényt. Továbbá ellenőrzi a Bluetooth funkcionalitások használatához szükséges

engedélyek meglétét. Abban az esetben, ha hiányzik egy vagy több szükséges engedély, akkor engedélykérést indít.

- initializeBl(): Az alkalmazás számára inicializálja a Bluetooth elérést és használatot a Cordova BluetoothLE plugin initialize() metódusát használva és feliratkozik a Bluetooth állapot változásokra, amelyet a blStatusSubject adatfolyam adattagján meghívott next() függvénnyel továbbít.
- isConnected(): Paraméterben egy IDevice objektumot vagy szöveget vár, ami ebben az esetben MAC-címnek feleltethető meg és visszatérési értéke, hogy az adott címmel rendelkező eszköz csatlakoztatva van-e.
- wasConnected(): Hasonló az előbb említett metódushoz, csak visszatérési érteke, az hogy az adott MAC-címmel rendelkező eszköz volt-e már korábban csatlakoztatva.
- isBLEnabled(): Paramétert nem váró függvény a Bluetooth bekapcsoltságának állapotával tér vissza, mely abban az esetben igaz, ha bekapcsolt állapotban van a mobileszközön.
- scanForDevices(): Bluetooth szkennelést indít el és a paraméterben megadott milliszekundum eltelte után leállítja azt, majd továbbítja azokat a megtalált eszközöket a scanInfoSubject adatfolyamán, melyek a Mi Band 3 névvel rendelkeznek.
- getConnectObserver(): Egy megfigyelővel tér vissza, amely feliratkozás során kapott DeviceInfo objektum status adattagjának megfelelően elindítja az autentikációt sikeres csatlakozás után. Lecsatlakozás esetén pedig bezárja a használt Bluetooth erőforrást. Mindkét esetben kapcsolat információkat küld a connectionInfoSubject adatfolyamán keresztül, amit az arra feliratkozók megkaphatnak.
- disconnectAndClose(): MAC-címet vagy egy IDevice objektumot vár paraméterként és ez alapján lecsatlakozik a megadott eszközről és bezárja annak erőforrását, ha lehetséges.
- sendEncryptionKey(): Autentikáció során, ha a Mi Band 3 eszköz más privát kulccsal rendelkezik, mint amit az alkalmazás használ akkor újra el kell küldeni az eszköznek. A metódus ezt a folyamatot hivatott elvégezni. Paraméterben egy eszközt, szolgáltatást és karakterisztikát vár, ahova elküldheti azt az autentikáció során.

- getEncryptionKey(): Egy előjel nélküli egészeket tartalmazó 16 bájt hosszúságú konstans értékkel tér vissza, amit az autentikáció során a privát kulcs elküldésekor használ fel a szolgáltatás.

4.3.2. Firestore szolgáltatások

A FirstoreBaseService egy olyan absztrakt osztály, amely implementálja a generikus IFirestoreBase interfészt és azon metódusokat tartalmazza, melyek adatbázis műveleteket végeznek el. Továbbá őse az összes Firestore szolgáltatásnak, mint a FirestoreActivityService, a FirestoreAuthSecurityService, a FirestoreBatteryInfoService, a FirestoreHeartRateService és a FirestoreUserService. Az osztály generikus típusa csak olyan osztály típus lehet, amely az IEntityModel interfészből öröklődik. Adattagjai inicializálására szolgáló konstruktorában pedig meg kell adni az alkalmazás Firestore példányát, a megadott típus konstruktorát és Firestore adat konvertáló függvényeit tartalmazó objektumot. Emellett opcionálisan megadható az adatbázis kollekció elérési útvonala, ha későbbi inicializálásara lenne szükség.

Firestore szolgáltatások örökölt metódusai:

- getDocRef(): Egy generikus dokumentum referenciát ad vissza, a kollekció útvonala és a paraméterben megadott szöveges azonosító (id) alapján.
- createQuery(): Paraméterben egy generikus QueryOption objektumot vagy azok egy tömbjét várja, ami alapján egy generikus Firestore lekérdezést készít az átadott objektumok validálása után.
- add(): A metódus a kapott generikus objektumot elmenti az adatbázisban egy dokumentumként a használt szolgáltatás kollekciójába. Abban az esetben, ha a kapott objektum nem rendelkezik azonosítóval (id), akkor generál hozzá egyet.
- delete(): Paraméterben egy szöveges azonosítóval rendelkező objektum, egy QueryOption objektum vagy azok egy tömbje adható meg a függvénynek, mely alapján kitörli az adatbázisban található megfelelő dokumentumot vagy dokumentumokat, ha létezik vagy léteznek a szolgáltatás által használt kollekcióban.
- get(): Generikus metódus a megadott szöveges azonosító (id) alapján lekéri az adatbázisban található dokumentumot. Ha nem létezik akkor undefined értekkel

tér vissza, egyébként a talált dokumentum adatából képzett osztály példánnyal tér vissza.

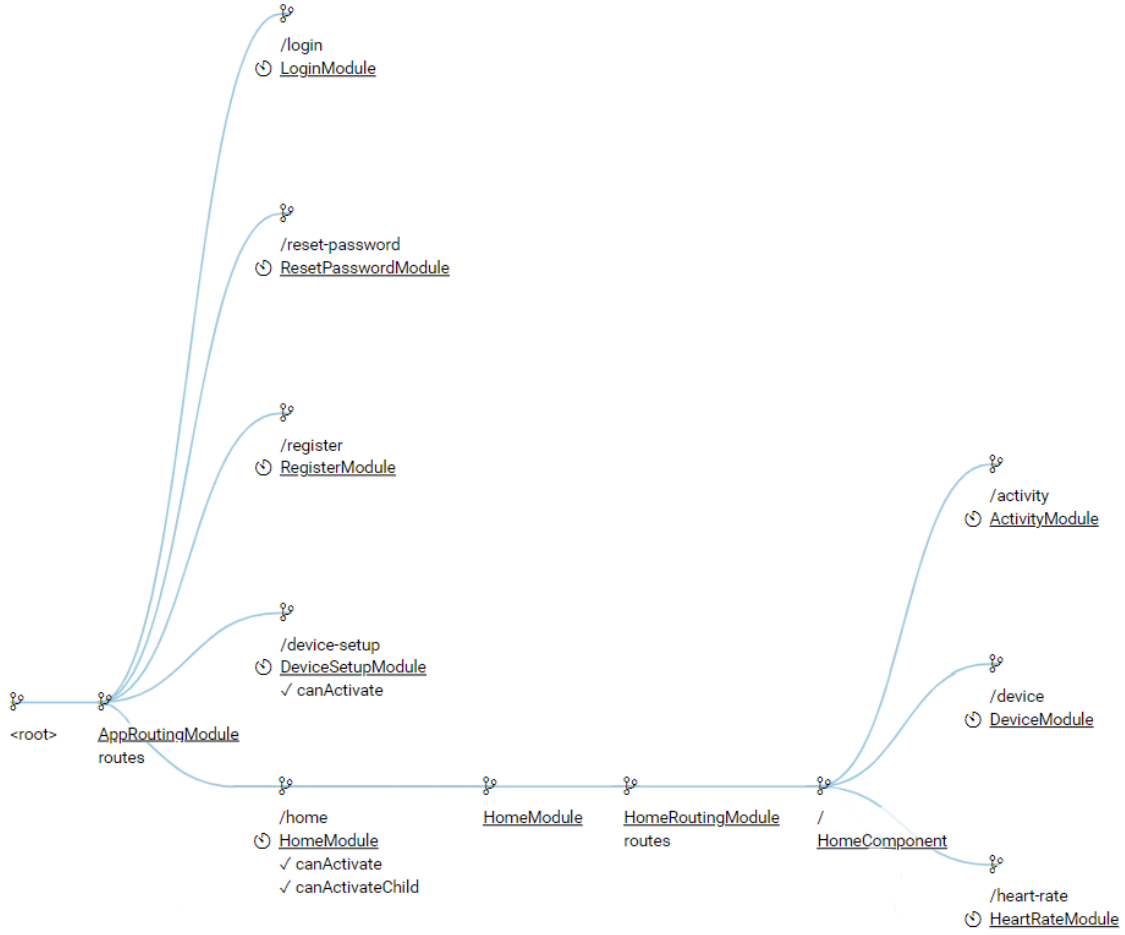
- getWithValueChanges(): Hasonlóan a get() függvényhez egy azonosítót vár, ami alapján visszaadja az adatbázisban található dokumentum változásait egy Subscribe típusú adatfolyam formájában, amire feliratkozva követhető a dokumentum értékeinek változásai.
- list(): Paraméterben egy QueryOption objektumot vagy azok egy tömbjét várja, amivel egy lekérdezést készítve lekéri a dokumentumokat és azok adataiból egy generikus objektumokból álló tömböt ad vissza, ha nem talált dokumentumokat a szolgáltatás által inicializált kollekcióban, akkor pedig undefined érték lesz az eredménye.
- update(): Metódus queryOptions paramétere alapján elkészített lekérdezéssel megtalálható dokumentumot vagy dokumentumok összességét frissíti a kollekcióban, a generikus és parciális data paraméter-ben található adattagoknak megfelelően.
- paginate(): Metódus limitált mennyiségű dokumentum lekérdezést tesz lehetővé a paraméterben megadott generikus PaginateOptions típusú objektum adattagjai alapján.
- listWithValueChanges(): List metódussal ellentétben egy feliratkozható Subscribe típusú adatfolyammal tér vissza, amivel lekérdezett dokumentumok változása követhető. A lekérdezést a paraméterben kapott QueryOption vagy azok tömbje alapján készíti el a korábban említett createQuery() függvényt használva.

4.4 Routing

Angular keretrendszerrel, úgynevezett single-page alkalmazások készíthetőek, amik nem igényelnek több különböző oldal betöltését használatuk során, helyettük egy oldalnézetet frissítenek dinamikusan a szükséges komponensek megjelenítésével és elrejtésével. Navigálást az Angular Router kezeli, amely az URL változása alapján cseréli ki a nézetet. URL címekben lehetőség van adatok átadására is nézetek között, hogy azok megfelelő információval rendelkezzenek. Nézetek útvonalait az átláthatóság és tisztább kód érdekében külön fájlban definiálva szokás megadni, amely biztosítja a navigációt a nézetek között. Benne található útvonalak (Routes) egy URL és egy komponens kötéstét adják meg. Továbbá minden Route elérhetősége konfigurálható Route Guard-ok

létrehozásával, amik a bennük definiált logika alapján döntik el, hogy a felhasználónak van-e jogosultsága egy adott oldalra navigálnia.

Következőekben a Mi Band 3 Companion által használt és a 4.5. ábrán látható routing felépítést fogom bemutatni, amely során megemlítem, hogy milyen modulok kerülnek betöltésre, ha a felhasználó egy bizonyos URL-re navigál.



4.5. ábra - Mi Band 3 Companion routing

Mint modulok esetén az útvonalak is egy hierarchiát alkotnak, ami az alkalmazás esetében a fenti ábrán látható. AppRoutingModule nevezetű gyökér modul kezeli a teljes útvonal forgalomirányítást, amelyben definiáltak a modulok és a hozzájuk tartozó útvonalak. Az előbb említett modulhoz tartozó útvonal a gyökér (root) útvonal, amely alapértelmezetten a „/”. Alkalmazás indulásakor a „/home” útvonal kerül betöltésre a HomeModule-lal, ha van bejelentkezett felhasználó és rendelkezik már párosított eszközzel. Útvonal megtekinthetőségének jogosultságát ez esetben a HomeGuard és DeviceGuard ellenőrzi a canActivate és canActivateChild metódusaival. Ha nincs

bejelentkezett felhasználó, akkor a HomeGuard a „/login” útvonalra navigál, mely esetben a bejelentkezés oldal fog megjelenni, ahonnan elérhetőek a regisztráció („/register”) és jelszó visszaállítás („/reset-password”) oldalak, melyekhez az előbbi felsorolásnak megfelelően a RegisterModule és a ResetPasswordModule tartozik. Ha pedig a felhasználó bejelentkezett és nincs eszköze, akkor a „/device-setup” útvonalra kerül navigálásra a DeviceGuard által, ahol az eszköz párosítás oldal lesz látható a DeviceModule betöltésével. A főoldal további gyermek útvonalakkal rendelkezik, amik a HomeRoutingModule nevezetű modulban definiáltak. Automatikusan az eszköz oldalra kerül navigálásra a felhasználó, ha van jogosultsága megtekinteni a „/home” útvonalat, mely esetben a DeviceModule kerül betöltésre és a teljes útvonal a „/home/device” lesz. Főoldali menün található „Kijelentkezés” gomb megnyomása után a bejelentkezés oldalra kerül vissza a felhasználó a „/login” útvonalon. Főoldalon történő további navigálásra az alsó navigációs sáv nyújt lehetőséget, mely használatával további 3 oldalra juthat el az alkalmazás használója. Egyik az előbb említett módon elérhető oldal az aktivitás oldal, melyhez a „/home/activity” útvonal tartozik az ActivityModule modullal. Továbbá megtekinthető a főoldalon még a korábban említett eszköz oldal („/home/device”) és a pulzus oldal a „/home/heart-rate” útvonalon, mely esetén a HeartRateModule töltődik be.

5. Adatmodellek

Ebben a fejezetben az alkalmazás által használt adatok modellezése kerül bemutatásra. Programozás során az adatmodell szerepe az összetartozó adatok egységbezárása és összességének reprezentációja a valóságnak megfelelően. Ezt az elvet követve az a Mi Band 3 Companion alkalmazás egy szorosan összekapcsolódó adatok összességét egy struktúrát leíró interfész és azt megvalósító osztály együttesen valósít meg egy modellt létrehozva. A következőekben bemutatásra kerülő adatmodellek közös tulajdonságai, hogy interfészei a korábban bemutatott IEntityModel interfészt bővítik ki, ami által rendelkeznek toString, copy, isEqual metódusokkal és azonosító (id) adattaggal. Ezen kívül az időpont tárolásra a FireTimestamp osztályt és annak IFireTimestamp interfészét használják. Ez az osztály a Firebase SDK által szolgáltatott Timestamp osztályt bővíti ki, amely a Firestore adatbázisában dátum elmentésére szolgáló típus. Továbbá minden adatmodell osztály rendelkezik egy getFirestoreConverter statikus metódussal, aminek visszatérési objektumával osztály példányt lehet átkonvertálni megfelelő dokumentum típussá adatbázisban való eltárolráskor és példányosítani lekérdezés során.

5.1. Eszköz adatmodell

Az eszköz adatmodell egy eszközzel kapcsolatos információkat hivatott egységbe zárni. Interfésze az 5.1. ábrán látható IDevice, amely a korábban bemutatott IEntityModel interfészt bővíti ki. Az adatmodellben tárolásra kerül az eszköz neve (name), a MAC-címe (macAddress), az utolsó csatlakozás dátuma (lastUsedDate) és az által szolgáltatott Bluetooth LE szolgáltatások (services). Interfész megvalósítása a Device osztályban valósult. Megvalósítása rendelkezik továbbá olyan privát statikus metódusokkal is, melyek adattag szerinti rendezést tesznek lehetővé, mint a sortByMacAddressAsc, sortByMacAddressDesc, sortByNameAsc, sortByNameDesc, sortByLastUsedDateAsc, sortByLastUsedDateDesc. Ezen összehasonlító függvények publikus elérése a getCompareFunction statikus metóduson keresztül érhetőek el az adattagnevének (propertyName) és rendezésirányának (direction) paraméterben való megadásával.

```
import { IService } from './IService';
import { IEntityModel } from './IEntityModel';
import { IFireTimestamp } from './IFireTimestamp';

export interface IDevice extends IEntityModel<IDevice> {
    name: string;
    macAddress: string;
    lastUsedDate: IFireTimestamp;
    services?: IService[] | undefined;
}
```

5.1. ábra – Eszköz adatmodell interfésze

5.2. Mérési adatmodellek

5.2.1 MeasurementValue adatmodell

Osztálya a MeasurementValue, ami az IMeasurementValue interfészt implementálja. A lenti ábrán látható, hogy átlag (avg), minimum (min) és maximum (max) numerikus adattagokkal rendelkezik, melyek a mérési értékek tárolására szolgálnak.

```
import { IEntityModel } from './IEntityModel';

export interface IMeasurementValue extends IEntityModel<IMeasurementValue>
    avg: number;
    min: number;
    max: number;
}
```

5.2. ábra – MeasurementValue adatmodell interfésze

A modellt továbbá használja az aktivitás adatmodell az intenzitás értékek és a pulzus adatmodell a pulzus értékek tárolására.

5.2.2 MeasurementInfo adatmodell

MeasurementInfo adatmodell a mérési adatok információit tárolja. Interfésze az IMeasurementInfo, amelyet a MeasurementInfo osztály valósít meg. A lenti ábrán látható adattagokkal rendelkezik a modell, melyek a mérés dátuma (date), a dátum típusa (type) és a mérési eszköz MAC-címe (deviceRef).

```
import { DateType } from '../../types/custom.types';
import { IEntityModel } from './IEntityModel';
import { IFireTimestamp } from './IFireTimestamp';

export interface IMeasurementInfo extends IEntityModel<IMeasurementInfo>
    type: DateType;
    date: IFireTimestamp;
    deviceRef: string;
}
```

5.3. ábra – MeasurementInfo adatmodell interfésze

5.3. Felhasználó adatmodell

Felhasználó adatmodell a felhasználó információinak eltárolására szolgál. Ezen adatok az a regisztráció során megadott e-mail cím (email), vezetéknév (firstName), keresztnév (lastName), születési dátum (birthDate), nem (gender), testsúly (weight), testmagasság (height), véletlenszerűen generált Mi Band 3 azonosító az eszköz inicializálásához (bandUserId), felhasználói azonosító (id) és az alkalmazás használata során párosított eszközök (devices). Az előbbiekben említett adattagok a lenti 5.4. ábrán tekeinthetőek meg, melyekkel az IUser interfész rendelkezik. Megvalósítása az User osztályban valósult meg, amely további metódusokkal rendelkezik.

```
import { IDevice } from './IDevice';
import { IEntityModel } from './IEntityModel';
import { IFireTimestamp } from './IFireTimestamp';
import { Gender } from '../../types/custom.types';

export interface IUser extends IEntityModel<IUser> {
    email: string;
    firstName: string;
    lastName: string;
    birthDate: IFireTimestamp;
    gender: Gender;
    weight: number;
    height: number;
    bandUserId: number;
    id?: string;
    devices?: IDevice[] | undefined;
    getFullName?(langCode?: string): string;
    getCurrentlyUsedDevice?(): IDevice | undefined;
}
```

5.4. ábra – Felhasználó adatmodell interfésze

Osztály további metódusai:

- getFullName(): Visszaadja a teljes nevét a felhasználónak a paraméterben megadott nyelv kód (langCode) által.
- getCurrentlyUsedDevice(): Metódus segítségével lekérhető az eszközök (devices) tömbjéből azon eszköz, amelyet a legutoljára használt az alkalmazással. Ha nincsenek eszközei a felhasználónak, akkor abban az esetben undefined értekkel tér vissza a függvény.

5.4. Aktivitás adatmodell

Aktivitás modell egy aktivitás mérés információinak egységbezárására szolgál. Struktúráját az IActivity modell definiálja, melyet az Activity osztály implementál. Ezen adatmodell által tárolt adatok a lépések száma (steps), a mérés információi (measurementInfo) MeasruementInfo típusként, a mérés azonosítója (id), a megtett távolság méterekben (distance), az elégetett kalóriák száma (calories) és az intenzitás értékei (intensity) MeasurementValue modellel reprezentálva. Továbbá az előbb említett interfész felépítése a lenti 5.5. ábrán tekinthető meg.

```
import { IEntityModel } from './IEntityModel';
import { IMeasurementInfo } from './IMeasurementInfo';
import { IMeasurementValue } from './IMeasurementValue';

export interface IActivity extends IEntityModel<IActivity> {
    steps: number;
    measurementInfo: IMeasurementInfo;
    id?: string;
    distance?: number;
    calories?: number;
    intensity?: IMeasurementValue;
}
```

5.5. ábra – Aktivitás adatmodell interfésze

5.5. Pulzus adatmodell

Pulzus mérések információinak tárolását a pulzus adatmodell valósítja meg, amely egy IHeartRate és azt megvalósító HeartRate osztályból tevődik össze. Ezen modell által tárolt adatok a pulzusszám (bpm), a mérési információk (measurementInfo) MeasurementInfo típussal és az azonosító (id). Továbbá az interfészének felépítése a lenti 5.6 ábrán látható.

```
import { IEntityModel } from './IEntityModel';
import { IMeasurementInfo } from './IMeasurementInfo';
import { IMeasurementValue } from './IMeasurementValue';

export interface IHeartRate extends IEntityModel<IHeartRate> {
    bpm: IMeasurementValue;
    measurementInfo: IMeasurementInfo;
    id?: string;
}
```

5.6. ábra – Pulzus adatmodell interfésze

5.6. Töltöttség adatmodell

Eszköz által szolgáltatott töltöttséggel kapcsolatos adatokat a töltöttségi modell hivatott eltárolni. Megvalósítása a BatteryInfo osztályban található meg, amely az IBatteryInfo interfészt implementálja. Ezen adatmodell adattagjai a mérés azonosítója (id), az eszköz töltöttség szintje (batteryLevel), a töltöttségi státusz (isCharging), az eszköz MAC-címe (deviceRef), az utolsó töltés dátuma (lastChargeDate), az utolsó töltés után kialakult töltöttségi szint (lastChargeLevel), a töltések száma (lastNumOfCharges), a megelőző töltés dátuma (prevChargeDate) és az utolsó töltés előtti töltések száma

(prevNumOfCharges). Ezen IBatteryInfo interfész és egyben adatmodell felépítése a lenti 5.7 ábrán tekinthető meg.

```
import { IEntityModel } from './IEntityModel';
import { IFireTimestamp } from './IFireTimestamp';

export interface IBatteryInfo extends IEntityModel<IBatteryInfo>
    batteryLevel: number;
    isCharging: boolean;
    id?: string;
    deviceRef?: string | undefined;
    lastChargeDate?: IFireTimestamp | undefined;
    lastNumOfCharges?: number | undefined;
    lastChargeLevel?: number | undefined;
    prevChargeDate?: IFireTimestamp | undefined;
    prevNumOfCharges?: number | undefined;
}
```

5.7. ábra – Töltöttség adatmodell interfésze

6. Adatok kinyerése és feldolgozása

Az alkalmazás fejlesztése során a Mi Band 3 eszközökkel való kommunikációt és nyers adatok Base64[2] dekódolását a Cordova Bluetooth LE Plugin biztosítja a mobiltelefon natív Bluetooth funkcionalitásának elérésével. Adatok feldolgozását pedig az alkalmazás BLE szolgáltatásaiban implementált algoritmusok végzik el. Mielőtt ismertetném az adatok feldolgozásának folyamatait, előtte ismertetném a Bluetooth LE technológiát.

6.1. Bluetooth Low Energy

Mi Band 3 okoskarkötő a Bluetooth Low Energy (BLE) vezeték nélküli technológiát használja. A klasszikus Bluetooth technológiával ellentétben BLE kis mennyiségű adatok küldésére és kevesebb energia fogyasztásra lett tervezve. A protokoll így olyan eszközökben és szenzorokban való használatra is alkalmas, amelyeknél fontos, hogy minél kevesebb energiát használjon a hosszabb üzemidő elérése érdekében. BLE a GATT Profile[3]-on alapszik, ami lehetővé teszi az adatcserét két eszköz közötti kapcsolat létrejötte után. Működése során megkülönböztetünk egy GATT szervert és klienst. A Mi Band 3 eszköz esetén a kliens az okostelefon, míg a szerver pedig maga az okoskarkötő. A GATT szerver szolgáltatásokkal rendelkezik, amelyek az eszköz funkcióinak felelnek meg. Minden szolgáltatáshoz egy egyedi UUID[4] azonosító tartozik és az egyes

[2] Olyan kódolási forma, amely bináris adatból ASCII karaktersorozat állít elő.
[3] GATT Profile (Generic Attribute Profile) – általános specifikáció profil
[4] UUID (Universally Unique Identifier) – egyedi azonosító szám

szolgáltatások karakterisztikákat tartalmaznak, amelyeken olvasási, írási és feliratkozási műveletek hajthatóak végre. Adatok kinyerésére és a Mi Band 3 eszköz beállításainak konfigurálására az egyes szolgáltatások karakterisztikáit lehet használni, amelyek a következő alfejezetekben kerülnek tárgyalásra.

6.2. Eszköz autentikálása

6.2.1 Advanced Encryption Standard (AES)

Advanced Encryption Standard egy titkosítási módszer, amelyet adatok titkosítására használhatunk. Mi Band 3 ezt a módszert az párosítás hitelesítéséhez használja, amely sikeres elvégeztével használhatóvá válnak a szolgáltatásai. Ez a titkosítási eljárás szimmetrikus kulcsú, vagyis egy privát kulcsot használ az adatok dekódolására és kódolására, amely a fogadó és a feladó birtokában van. A kulcs hossza lehet 128, 256 és 512 bit hosszúságú, amely a Mi Band 3 esetében 128 bites. Az okoskarkötő által használt titkosítási algoritmus pedig az AES/ECB/NoPadding.

6.2.2 Párosítás hitelesítése

Alkalmazás használata során a Mi Band 3 eszköz és az okostelefon közötti kapcsolat kezeléséért a már korábban bemutatott BleConnectionService[5] szolgáltatás felelős. Egy eszközhöz való csatlakozás előtt Bluetooth szkennelés szükséges a BLE hatáskörében. Eszközök keresését a scanForDevices metódus végzi el, amely csak az eszköz párosítás oldal által használt. Csatlakozást a connect metódus végzi el, amely szkennelést is végezhet abban az esetben, ha nem volt korábban csatlakoztatva a csatlakoztatni kívánt eszköz. Sikeres csatlakoztatás és szolgáltatások felderítése után meghívja az authenticateMiBand függvényt, amely a hitelesítés elvégzésére szolgál. Mi Band 3 hitelesítése csatlakozás után végezhető el, amelyre 30 másodpercig van lehetőség, aminek letelte után automatikusan megszakad a kapcsolat. Autentikáció elvégzése nélkül az okoskarkötő funkcionalitásai, szolgáltatásai nem elérhetőek, így fontos annak elvégzése. Első lépése a folyamatnak az autentikációs szolgáltatás hitelesítésért felelős karakterisztikájára való feliratkozás, amely íráskor 3 bájtos válasszal fog visszatérni. Minden Bluetooth kommunikáció által szolgáltatott válasz vagy olvasási művelet eredménye Base64 kódolással ellátót szöveg, amely dekódolása szükséges a nyers adatok

[5] https://github.com/noelhorvath/mi-band-3-companion/blob/main/src/app/services/ble/connection/ble-connection.service.ts

eléréséhez. Dekódolást a Cordova BLE Plugin encodedStringToBytes metódusa végzi el, amely visszaadja az eredeti bájtokat. Ezt az első lépést a 6.1 ábrán látható kódrészlet valósítja meg. Feliratkozás utáni következő lépés egy a 16 bájtos privát kulcs által titkosított vele azonos hosszúságú adat kérése az okoskarkötőtől. A kérés az autentikációs karakterisztikára való 0x0200 bájtok írásával tehető meg, mely elküldése előtt Base64-re kerül kódolásra.

```
private authenticateMiBand(device: IDevice): void {
    const service = this.miBand3.getServiceByName('authentication');
    const characteristic = service?.getCharacteristicByName('auth');
    if (service === undefined) {
        throw new Error('Failed to get authentication service');
    } else if (characteristic === undefined) {
        throw new Error('Failed to get authentication characteristic');
    }
    // Authentication steps
    // 1. set notify on (by sending 2 bytes request (0x01, 0x00) to the Descriptor === subscribe)
    this.subscribe(
        device,
        service,
        characteristic,
        {
            next: async (data: OperationResult) => {
                try {
                    if (this.connectionInfo.device !== undefined) {
                        if (data.status === BLESubscriptionStatus.SUBSCRIBED) {
                            this.logHelper.logDefault(this.authenticateMiBand.name, 'Subscribed to auth');
                            // authentication sub will fail immediately if the encryption key is same as on the Mi Band 3
                            // it is better to start with requesting a random key first
                            // if the device has different encryption/secret key => 100304 => send encryption key => back to 100101
                            // 2. Request random key by sending 2 bytes: 0x02 + 0x00
                            await this.write(data.address, data.service, data.characteristic, Buffer.from([0x02, 0x00]));
                        } else {
                            if (data.value !== undefined) {
                                const response = Buffer.from(this.ble.encodedStringToBytes(data.value)).subarray(0, 3).toString('hex');
                                this.logHelper.logDefault(this.authenticateMiBand.name, 'auth response', { value: response });
                                switch (response) {
```

6.1. ábra – Feliratkozás és kérés kódja az authenticateMiBand metódusban

A kérés sikeres megtörténése után az eszköz által küldött adat első 3 bájtja 0x100201 lesz további 16 bájtja pedig a dekódolandó bájtok sorozata. Abban az esetben, ha valamilyen okból nem képes az eszköz elküldeni akkor 0x100204 választ fog szolgáltatni, mely esetben a kapcsolat bezárásra kerül. Dekódolás korábban említett AES/ECB/NoPadding algoritmussal történik, mely végezte után az eredményt visszaküldi az eszköznek 0x0300 bájtokkal kiegészítve. Az enkriptálást a Node.js Crypto moduljában található függvények biztosítják az alkalmazás számára. Ezen válaszok kezelésének kódjai a 6.2 ábrán tekinthetőek meg.

```
case '100201':
    // 4. Encrypt the random 16 bytes number with AES/ECB/NoPadding with the encryption key that was sent at 2. step
    // authentication level 2
    this.logHelper.logDefault(this.authenticateMiBand.name, 'Requested random key received');
    const randomData = Buffer.from(this.ble.encodedStringToBytes(data.value)).subarray(3, 19); // remove the first 3 bytes
    const algorithm = 'aes-128-ecb';
    const cipher = crypto.createCipheriv(algorithm, this.getEncryptionKey(), '').setAutoPadding(false); // create encryptor + disable auto padding
    const encryptedData = cipher.update(randomData); // encrypt data
    cipher.final(); // stop cipher
    // 5. Send 2 bytes 0x03 + 0x00 + encrypted data
    await this.write(data.address, data.service, data.characteristic, Buffer.concat([Buffer.from([0x03, 0x00]), encryptedData]));
    break;
case '100204':
    // data request failed
    this.connectionInfoSubject.error('Requesting data failed!');
    await this.disconnectAndClose(data.address);
    break;
```

6.2. ábra – Kérés válaszok kezelése az authenticateMiBand metódusban

Ha az alkalmazás és eszköz által használt privát kulcs nem egyezik, akkor 0x100304 választ fog küldeni, amely azt jelenti, hogy el kell küldeni az eszköznek egy privát kulcsot. Az alkalmazás által használt „titkos” kulcs privát láthatóságú getEncryptionKey függvénnyel kérhető el, ami egy konstans 16 bájt hosszúságú kulcsot[6] ad vissza. A 0x100304 válasz kezelése során a sendEncryption metódus kerül meghívásra, amely elküldi az eszköznek a kulcsot.

```
case '100304':
    // encryption fails, because the device probably has a different secret key
    this.logHelper.logError(this.authenticateMiBand.name, 'Encryption failed, sending new encryption key!');
    await this.sendEncryptionKey(data.address, data.service, data.characteristic); // write 2 bytes (0x01 + 0x00)  + encryption key (16 bytes)
    break;
```

6.3. ábra – Kulcs küldése nem egyező kulcsok esetén az authenticateMiBand metódusban

A sikeres írás művelet végrehajtása után 0x100101 bájt adatok kerülnek átadásra, amely során a korábban említett kérés folyamat elindítása szükséges. Ebben az esetben a 0x0200 bájtok írása után a Mi Band 3 eszközön található gomb megnyomása szükséges a folytatáshoz. Ha a gomb nem kerül megnyomásra, akkor 0x100102 választ küld az eszköz, amely során újra el kell küldeni az alkalmazás által használt kulcsot. Sikertelen küldés esetén az okoskarkötő a 0x100104 választ szolgáltatja a feliratkozáson keresztül, amely során még egyszer elküldésre kerül a kulcs, de második esetben már a kapcsolat megszakításra kerül. Ezen válaszok által végrehajtott folyamatok kódja a lenti 6.4. ábrán tekinthető meg.

[6] Alkalmazás által használt privát kulcs a 0x30313233343536373839404142434445

```
case '100101':
    // if user did press the button on the Mi Band
    // authentication level 1
    this.logHelper.logDefault(this.authenticateMiBand.name, 'Sent key ok');
    this.logHelper.logDefault(this.authenticateMiBand.name, 'Paired successfully');
    // 3. Request random key by sending 2 bytes: 0x02 + 0x00
    await this.write(data.address, data.service, data.characteristic, Buffer.from([0x02, 0x00]));
    this.logHelper.logDefault(this.authenticateMiBand.name, 'Request sent for a random key');
    break;
case '100102':
    this.connectionInfoSubject.error('Pairing failed!\nPlease press the button on Mi Band to confirm pairing!');
    this.connectionInfoSubject.next(new ConnectionInfo(BLEConnectionStatus.CONNECTED, false, device));
    await this.sendEncryptionKey(data.address, data.service, data.characteristic); // send encryption key again
    break;
case '100104':
    this.connectionInfoSubject.error('Encryption key sending failed!');
    // re-send encryption key once if sending failed
    if (!this.sentEncryptionKeyAgain) {
        this.connectionInfoSubject.next(new ConnectionInfo(BLEConnectionStatus.CONNECTED, false, device));
        await this.sendEncryptionKey(data.address, data.service, data.characteristic); // send encryption key again
        this.sentEncryptionKeyAgain = true;
    } else {
        await this.disconnectAndClose(data.address);
    }
    break;
```

6.4. ábra – Kulcs küldése utáni válaszok kezelése az authenticateMiBand metódusban

Miután a kulcsok megegyeznek és a kért adat sikeresen dekódolva és visszaküldve lett, akkor a 0x100301 válasz fog érkezni, amely azt jelenti, hogy az autentikáció sikeresen megtörtént. Az authenticateMiBand metódusban a küldött adat kezelése során a kapcsolat adatfolyamán (connectionInfoSubject) az autentikáció státuszát igazra állítja és leiratkozik a használt karakterisztikáról. Továbbá ezen utolsó folyamat kódja a lenti 6.5. ábrán látható.

```
case '100301':
    // 6. Authentication done
    this.logHelper.logDefault(this.authenticateMiBand.name, 'Authenticated successfully');
    device.lastUsedDate = FireTimestamp.now();
    this.connectionInfoSubject.next(new ConnectionInfo(BLEConnectionStatus.CONNECTED, true, device));
    await this.unsubscribe(data.address, data.service, data.characteristic);
    break;
```

6.5. ábra – Sikeres authentikáció válaszának kezelése az authenticateMiBand metódusban

6.3. Adatok feldolgozása

Mi Band 3 által szolgáltatott aktivitás, pulzus és az okoskarkötő töltöttségével kapcsolatos adatok feldolgozásáért a BleDataService[7] nevezetű szolgáltatásban megírt algoritmusok felelősek. Ezen szolgáltatás a korábban említett BleBaseService[8] absztrakt osztályból származik, mint a BleConnectionService. Minden eszköz által küldött adat előjel nélküli

[7] https://github.com/noelhorvath/mi-band-3-companion/blob/main/src/app/services/ble/data/ble-data.service.ts
[8] https://github.com/noelhorvath/mi-band-3-companion/blob/main/src/app/services/ble/base/ble-base.service.ts

bájtok sorozata, melyet a használathoz 10-es számrendszerbeli egész számmá szükséges konvertálni, hogy dolgozni lehessen velük. Ezt a konvertálást a BleBaseService-ben megvalósított bytesToInt metódusa valósítja meg. A függvény algoritmusa végig megy a kapott bájtok tömbjén egy ciklussal az utolsó elemtől az elsőig és egy 0-val inicializált számlálót megszoroz a 256-tal és hozzáadja az i.-edik tömb elem értékét. Ha a ciklus lefutott akkor visszaadja a számot, amit a bájtok reprezentálnak, ha a megadott tömb nem üres. A bytesToInt metódus programkódja a lenti 6.6 ábrán látható.

```
public bytesToInt(bytes: Uint8Array): number {
    if (bytes.length === 0) {
        throw new Error('Failed to convert bytes to int: bytes array is empty!');
    }
    let res = 0;
    for (let i = bytes.length - 1; 0 <= i; i--) {
        res = res * 256 + bytes[i];
    }
    return res;
}
```

6.6. ábra – BleBaseService-ben található bytesToInt metódus kódja

A jelenlegi aktivitás adatai a 0000fee0-0000-1000-8000-00805f9b34fb UUID-val rendelkező szolgáltatás megfelelő karakterisztikáján („00000005-0000-3512-2118-0009af100700”) érhető el, amelyre az eszköz lehetőséget biztosít feliratkozásra és olvasásra. Egy napi valós idejű aktivitás mérés 13 bájt hosszúságú, amely első bájtja egy fejléc további 12 bájtja pedig tartalmazza a lépések számát (2-5), az elégetett kalóriák számát (6-9) és a megtett távolságot méterben (10-13). Ahhoz, hogy az aktivitás adatokat megkapja az alkalmazás fel kell iratkoznia az előbb említett karakterisztikára, amelyet a subscribeToRealTimeActivity metódus végez el. Ezen függvény az alkalmazás indulásakor hívódik meg automatikusan, ha van már eszköze a felhasználónak párosítva és az adatfolyamán kapott adatokat a processRealTimeActivity metódusnak továbbítja, mely a feldolgozást végzi el. A karakteriszikáról való olvasást a readRealTimeAcivity függvény valósítja meg, amely eszköz oldali mérési adatok inicializálás előtt kerül meghívásra. Továbbá az előbbi metódus a subscribeToRealTimeActivity algoritmusához hasonlóan szintén továbbítja az olvasott adatokat feldolgozásra a processRealTimeAcitivity számára. A feldolgozást végző függvény programkódja a 6.7 ábrán látható. Algoritmusa átkonvertálja a kapott adatott bájtok tömbjévé és a bytesToInt metódussal inicializál egy Activity típusú objektumot a megfelelő adatokkal. Értékadás

után elmenti az adatbázisba a feldolgozott aktivitást, majd továbbítja azt az activitySubject adatfolyamon a komponensek és szolgáltatások számára.

```
private async processRealTimeActivity(data: OperationResult): Promise<void> {
    if (data.value) {
        const bytes = this.ble.encodedStringToBytes(data.value);
        /* Realtime steps 5 bytes or 13 bytes if not used with Mi Fit it only shows steps count => 5 bytes
            0. header?
            1-4. steps
            5-8. calories (kcal)
            9-12. distance (meters)
        */
        const activity = new Activity();
        const current = new Date();
        const date = new Date(Date.UTC(current.getFullYear(), current.getMonth(), current.getDate(), 0, 0, 0, 0));
        activity.measurementInfo = new MeasurementInfo(data.address, FireTimestamp.fromDate(date), DateTypeEnum.DAILY);
        activity.steps = this.bytesToInt(bytes.subarray(1, 5));
        activity.distance = this.bytesToInt(bytes.subarray(5, 9));
        activity.calories = this.bytesToInt(bytes.subarray(9));
        const sameActivity = await this.activityService.list([
            { fieldPath: 'measurementInfo.date', opStr: '==', value: activity.measurementInfo.date },
            { fieldPath: 'measurementInfo.type', opStr: '==', value: activity.measurementInfo.type },
            { limit: 1 }
        ]);
        if (sameActivity !== undefined) {
            await this.activityService.update(
                { id: sameActivity[0].id },
                {
                    steps: activity.steps,
                    distance: activity.distance,
                    calories: activity.calories
                }
            );
        } else {
            await this.activityService.add(activity);
        }
        this.activitySubject.next(activity);
        this.logHelper.logDefault(this.processRealTimeActivity.name, 'realtime activity', { value: activity });
    }
}
```

6.7. ábra – Valós idejű aktivitást feldolgozó metódus programkódja

Mi Band 3 töltöttséggel kapcsolatos adatai az aktivitásnál említett szolgáltatás másik karakterisztikáján („00000006-0000-3512-2118-0009af100700”) keresztül érhető el. Erre való feliratkozást a subscribeToBatteryInfo metódus implementálja, amely eszköz csatlakozás után automatikusan meghívódik az eszköz feliratkozásokat és olvasásokat inicializáló initDeviceData függvény által az olvasást megvalósító readBatteryInfo-val együtt. Egy töltöttségi adat 3 vagy 20 bájtból áll annak tekintetében, hogy mely forrásból kapta meg az alkalmazás. Ha feliratkozás által kapott, akkor abban az esetben 3 bájt hosszúságú adat tartalmaz egy fejlécet az 1. bájton, a Mi Band 3 eszköz töltöttségi szintjét a 2. bájton és a töltöttség státuszát a 3. bájton. Töltöttségi státusz 1 értéke a töltést jelzi 0 pedig, hogy nem tölt jelenleg az eszköz. Olvasás által 20 bájt hosszúságú lesz az adat, amely az előbbi 3 információ mellett tartalmazza még az utolsó

előtti töltés dátumát (4-10) és töltések számát (11), utolsó töltés dátumát (12-18), töltések számát (19) és a szintjét (20). Mind az olvasott, mind a feliratkozás során kapott adatokat a processBatteryInfo metódus dolgozza fel. Mivel a subscribeToBatteryInfo függvény által kapott adatok kevesebb információt tartalmaznak, ezért a feldolgozást megvalósító metódusban, ha 3 bájt hosszúságú az adat, akkor a readBatteryInfo kerül meghívásra. A feldolgozó processBatteryInfo függvény algoritmusa egy BatteryInfo típusú objektum adattagjait inicializálja a megfelelő értékekkel az bytesToInt által végzett konvertálás után. Értékadás után pedig frissítésre kerül az adatbázisban tárolt töltöttségi adat. Az előbbiekben bemutatott feldolgozó metódus programkódja a lenti 6.8 ábrán látható.

```
public async processBatteryInfo(data: OperationResult): Promise<void> {
    if (data.value) {
        const bytes = this.ble.encodedStringToBytes(data.value);
        if (bytes.length < 3) {
            // get full battery info
            try {
                await this.readBatteryInfo();
            } catch (e: unknown) {
                this.logHelper.logError(this.processBatteryInfo.name, this.readBatteryInfo.name, { value: e });
            }
        } else {
            // process full info after reading from device
            const batteryInfo = new BatteryInfo();
            const unknown_value = bytes.at(0); // first byte ???
            batteryInfo.id = data.address;
            batteryInfo.deviceRef = this.connectionInfo.device?.macAddress ?? 'unknown';
            batteryInfo.batteryLevel = bytes.at(1) as number;
            batteryInfo.isCharging = bytes.at(2) === 1; // status => 1 === charging
            batteryInfo.prevChargeDate = this.dateTimeFromBytes(bytes, 3); // 7 bytes
            batteryInfo.prevNumOfCharges = bytes.at(10);
            batteryInfo.lastChargeDate = this.dateTimeFromBytes(bytes, 11); // 7 bytes
            batteryInfo.lastNumOfCharges = bytes.at(18);
            batteryInfo.lastChargeLevel = bytes.at(19);
            this.logHelper.logDefault(this.processBatteryInfo.name, 'batteryInfo', { value: batteryInfo });
            this.logHelper.logDefault(this.processBatteryInfo.name, 'batteryInfo unknown value', { value: unknown_value });
            this.batteryInfoSubject.next(batteryInfo);
            try {
                await this.batteryInfoService.update({ id: batteryInfo.id }, batteryInfo);
            } catch (e: unknown) {
                this.logHelper.logError(this.processBatteryInfo.name, 'update BatteryInfo error', { value: e });
            }
        }
    }
}
```

6.8. ábra – Mi Band 3 töltöttségi adatait feldolgozó függvény programkódja

A pulzus méréseket a 0000180d-0000-1000-8000-00805f9b34fb szolgáltatás 00002a37-0000-1000-8000-00805f9b34fb UUID-val rendelkező karakterisztikáján érhetjük el feliratkozással, amelyet a subscribeToHeartRate metódus valósít meg. Ezen adatok feldolgozását a processHeartRate függvény végzi el. Minden pulzus mérés 2 bájt hosszúságú és 1. bájtja egy fejléc, 2. bájtja pedig maga a pulzusszám. A feldolgozó algoritmus a 6.9 ábrán látható, amely átkonvertálja a kapott adatot és továbbítja a

heartRateSubject adatfolyamon számként. Mivel a korábbi pulzus mérések kerülnek csak mentésre, ezért a feldolgozott pulzusszám nem kerül elmentésre az adatbázisban ebben a metódusban.

```
private async processHeartRate(data: OperationResult): Promise<void> {
    if (data.value !== undefined) {
        // pushes data per second 1 heart rate measurement === 5 bpm data
        // 0. header?
        // 1. bpm
        const bpm = this.ble.encodedStringToBytes(data.value)[1];
        this.logHelper.logDefault(this.processHeartRate.name, 'bpm', { value: bpm });
        this.heartRateSubject.next(bpm);
    }
}
```

6.8. ábra – Pulzus mérést feldolgozó metódus programkódja

Mi Band 3 eszköz az aktivitás és pulzus méréseket percenként eltárolja, amelyeket az alkalmazás minden induláskor szinkronizál az adatbázisban tárolt adatokkal, de lehetőség van manuális végrehajtásra is a főoldalon. A korábbi adatok szinkronizálásának dátuma minden sikeres végrehajtás után frissül az adatbázisban és tárolt dátumtól kerülnek lekérésre az adatok az okoskarkötőtől következő szinkronizálást követően. Ezek feldolgozását a processPastActicityData metódus végzi el. Korábbi adatok fogadásához az aktivitásnál is említett szolgáltatás 00002902-0000-1000-8000-00805f9b34fb (szinkronizálás válaszok fogadására) és 00000005-0000-3512-2118-0009af100700 (korábbi adatcsomagok fogadására) karakterisztikáira szükséges feliratkozni, amelyet a kérési folyamat indításáért felelős fetchActivityData függvény végrehajt. Egy a processPastActivityData által feldolgozott adatcsomag az 1. bájtján a csomag sorszámát további bájtokon, pedig maximum 4 darab 4 bájtos korábbi adatot tartalmaz. Egy ilyen régebbi az eszköz által tárolt percenként értelmezendő 4 bájtnyi adat tartalmazza az aktivitás kategóriáját, intenzitását, lépések számát és a pulzust a felsorolás sorrendjében 1-1 bájton. A feldolgozó algoritmus a gyorsabb végrehajtás és adatfeltöltés érdekében a percenkénti aktivitásokból és pulzusmérésekből óránkénti méréseket készít a bejövő összes csomag megérkezte után, majd menti el azokat az adatbázisba.

7. Adattárolás Firestore használatával

7.1. NoSQL

A Mi Band 3 Companion által használt Firestore egy NoSQL adatbázist. Erre az adatbázis típusra gyakran hivatkoznak „nem SQL” és „nem relációs” elnevezésekkel, melyek hangsúlyozzák, hogy az SQL adatbázisoktól eltérően működik. Az SQL adatbázisokkal ellentétben nem előre definiált struktúrával rendelkezik, hanem dinamikus sémával, amely gyorsan változó és nagy mennyiségű strukturálatlan adatok tárolást teszi lehetővé. Séma típusának tekintetében léteznek dokumentum, kulcs-érték, oszlopalapú és gráf alapú NoSQL adatbázisok. Firestore a dokumentum sémát használja, amely során dokumentumokat gyűjteményekbe (collection) szervezi. Jellemzője még, hogy egy dokumentum bármely attribútuma alapján végezhető lekérdezés. Továbbá egy dokumentum több aldokumentummal rendelkezhet, amelyek hierarchiát alakítanak ki. NoSQL az adatokat horizontálisan skálázza tárhely szempontjából a kiszolgálók közötti particionálással vagyis, ha elfogy a kiszolgáló tárhelye, akkor új fizikai gépek kerülnek be a rendszerbe a tárhely növelése érdekében.

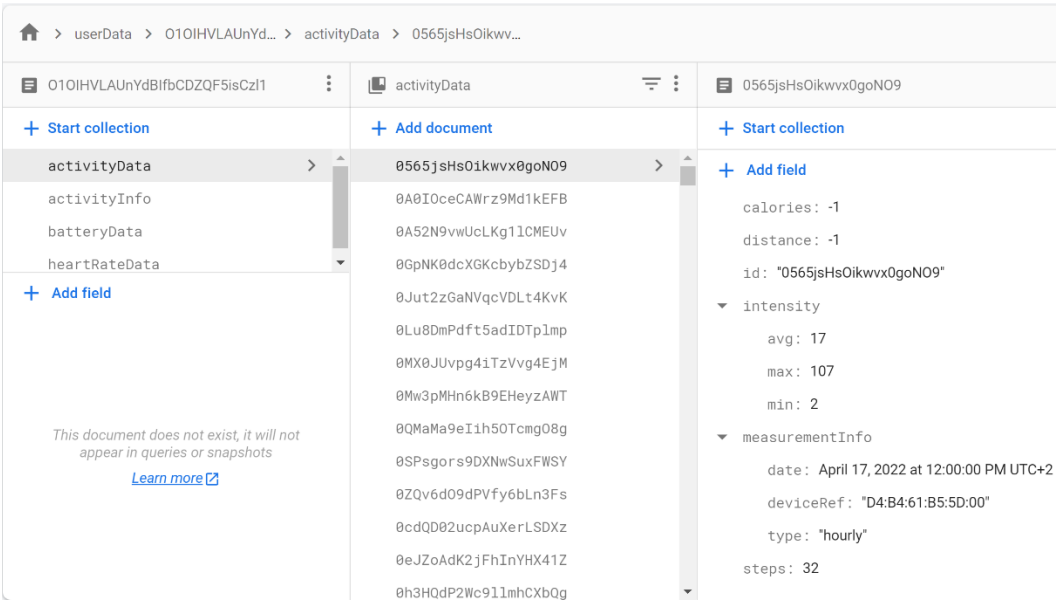
7.2. Firestore adattárolás

Firestore használatával az adatok dokumentumokban kerülnek tárolásra, amelyek gyűjteményekben vagy más néven kollekciókban helyezkedhetnek el. Továbbá a dokumentumok és a kollekciók további aldokumentumokba szervezhetőek egy hierarchiát kialakítva. Egy dokumentum bármilyen típusú adat tárolására alkalmas, annyi kivétellel, hogy a TypeScript osztály példányok esetén implementálni kell egy FirestoreDataConverter típusú objektumot a toFirestore és a fromFirestore metódusokkal. Egy toFirestore függvény az osztály példányt DocumentData típusú adattá alakítja az adatbázisban való eltároláshoz és a fromFirestore pedig egy dokumentumot példányosít az osztálynak megfelelően. Ezen konvertálók mind megvalósításra kerültek az eltárolt osztályokban statikus metódusokként, amelyet a Firestore szolgáltatások felhasználnak az adatbázis műveletek elvégzése során. A példaként a 7.1 ábrán az Activity (Aktivitás) osztály átalakító függvényei láthatóak az előbb említett módon megvalósítva, ahol az instantiate függvény végzi el a példányosítást.

```
public static getFirestoreConverter(): FirestoreDataConverter<Activity> {
    return {
        toFirestore: (instance: WithFieldValue<Activity> | Partial<Activity>): DocumentData => ({
            id: instance.id,
            steps: instance.steps,
            distance: instance.distance,
            calories: instance.calories,
            intensity: instance.intensity !== undefined
                ? MeasurementValue.getFirestoreConverter().toFirestore(instance.intensity as MeasurementValue)
                : instance.intensity,
            measurementInfo: instance.measurementInfo !== undefined
                ? MeasurementInfo.getFirestoreConverter().toFirestore(instance.measurementInfo as MeasurementInfo)
                : instance.measurementInfo
        }),
        fromFirestore: (snapshot: QueryDocumentSnapshot, options?: SnapshotOptions): Activity =>
            instantiate(snapshot.data(options) as IActivity, Activity)
    };
}
```

7.1. ábra – Activity osztály Firestore konvertáló programkódja

Az alkalmazás adatbázis struktúra szemléltetésére a felhasználó aktivitás adatainak adatbázisban tárolt adatstruktúráját mutatom be a következőekben. Ezen struktúra megegyezik a már korábban ismertetett Aktivitás adatmodell interfésze (IActivity) és osztálya (Activity) által meghatározott adattagokkal.



7.2. ábra – Mi Band 3 Companion adatbázis struktúrája

A 7.2 ábrán látható adatstruktúra egy felhasználó aktivitás mérés adatait tartalmazza. Az aktivitások a userData kollekcióban található felhasználó azonosítójával megegyező aldokumentumban lévő activityData gyűjteményben helyezkednek el. A benne lévő egyes dokumentumok egy véletlenszerűen generált azonosítóval azonosíthatóak, amely megegyezik az általa tárolt aktivitás id adattagjában tárolt értékkel.

A fenti dokumentumban látható továbbá, hogy az adatok kulcs-érték párokként, hierarchiába szervezettek és az IAcitvity interfész által definiált struktúrát követik.

8. Összegzés

A szakdolgozatomban törekedtem az alkalmazás által használt adatmodellek, módszerek, technológiák és kódrészek bemutatására. A Mi Band 3 Companion fejlesztése során sikerült egy olyan webes technológiákat használó szoftvert megvalósítanom, amely az okoskarkötő tulajdonosok hasznára válhat az eszköz által szolgáltatott adatok kezelésére. Az applikáció létrehozáskor igyekeztem egy mindennapi használatra alkalmas letisztult felületet készíteni, hogy minden funkcióját könnyen és egyszerűen lehessen használni. Az alkalmazás még teljesebbé tételéhez további funkcionalitásokkal lehet kibővíteni, mint például a szerver oldali statisztikák szolgáltatása napi és heti szinten egy statisztikák oldalon, további eszköz beállítások lépés és pulzus mérésekhez és felhasználói fiók, eszköz és mérések törlése. A szoftver továbbá tesztekkel is bővíthető, hogy a minősége biztosítása megfelelőbb legyen. Az alkalmazás kódjának alapja platformfüggetlen keretrendszerekkel lett megvalósítva, ezért további rendszerekre is kiadható a megfelelő módosítások és tesztelés elvégzésével.

Az szoftver készítésének megkezdése előtt nem rendelkeztem a Mi Band 3 eszköz programozással használható funkcionalitásainak és a BLE technológia ismeretével, amely nehézségeket okozott a kezdetben, de megfelelő dokumentumok és források tanulmányozásával elsajátítottam ezeket és végeredményben sikeresen megvalósítottam egy a témához kapcsolódó alkalmazást az okoseszközhöz. Fejlesztés során továbbá az Angular, Ionic, TypeScript és Capacitor technológiák mélyebb elsajátítása és megismerése sikerült, amelyek használatával magabiztosabban kezdhetek egy újabb projekthez. A megfelelő tudás megszerzéséhez hozzájárultak a különböző keretrendszerek és a TypeScript által nyújtott nagyszerű angol nyelvű dokumentációk is.

A korábban felsorolt bővítési lehetőségek és további platformokra való kompatibilitás kiterjesztése a jövőben tervben van. Ezen további munkálatok és változtatások az alkalmazás GitHub[9] oldalán lesznek elérhetőek.

[9] https://github.com/noelhorvath/mi-band-3-companion

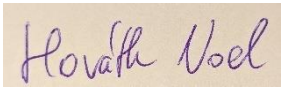
Felhasznált irodalom, források

1. Ionic: https://ionicframework.com/docs (utolsó látogatás dátuma: 2022.05.13)
2. Firebase: https://firebase.google.com/docs (utolsó látogatás dátuma: 2022.05.13)
3. ngx-translate: https://github.com/ngx-translate/core (utolsó látogatás dátuma: 2022.05.13)
4. Angular framework: https://angular.io/docs (utolsó látogatás dátuma: 2022.05.13)
5. Angular routing: https://angular.io/guide/router (utolsó látogatás dátuma: 2022.05.13)
6. Capactior: https://capacitorjs.com/docs (utolsó látogatás dátuma: 2022.05.13)
7. Bluetooth LE: https://developer.android.com/guide/topics/connectivity/bluetooth-le (utolsó látogatás dátuma: 2022.05.13)
8. Chart.js: https://www.chartjs.org/docs/3.7.1 (utolsó látogatás dátuma: 2022.05.13)
9. NoSQL adatbázis: https://azure.microsoft.com/en-us/overview/nosql-database/ (utolsó látogatás dátuma: 2022.05.13)
10. TypeScript: https://www.typescriptlang.org/docs/ (utolsó látogatás dátuma: 2022.05.13)
11. Cordova Bluetooth LE Plugin: https://github.com/randdusing/cordova-plugin-bluetoothle (utolsó látogatás dátuma: 2022.05.13)
12. Mi Band 3 Companion: https://github.com/noelhorvath/mi-band-3-companion (Az alkalmazás GitHub oldala)
13. NgCharts: https://valor-software.com/ng2-charts/ (utolsó látogatás dátuma: 2022.05.13)
14. Node.js: https://nodejs.org/en/docs/ (utolsó látogatás dátuma: 2022.05.13)
15. Rxjs: https://rxjs.dev/api (utolsó látogatás dátuma: 2022.05.13)
16. Mi Band 3 hivatalos oldala: https://www.mi.com/global/mi-band-3 (utolsó látogatás dátuma: 2022.05.13)
17. Ashutosh Maharaj (2020) - A Review on Advanced Encryption Standards (AES): https://www.researchgate.net/publication/338853730_A_Review_on_Advanced_Encryption_Standards_AES (utolsó látogatás dátuma: 2022.05.13)
18. Angular services: https://angular.io/guide/architecture-services (utolsó látogatás dátuma: 2022.05.13)
19. Gadgetbridge: https://codeberg.org/Freeyourgadget/Gadgetbridge (utolsó látogatás dátuma: 2022.05.13)
20. Master for Mi Band: https://play.google.com/store/apps/details?id=blacknote.mibandmaster (utolsó látogatás dátuma: 2022.05.13)
21. Mi Fit: https://play.google.com/store/apps/details?id=com.xiaomi.hm.health (utolsó látogatás dátuma: 2022.05.13)

Nyilatkozat

Alulírott Horváth Noel programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében. Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2021.05.13.



Elektronikus melléklet

A Mi Band 3 Companion alkalmazás forráskódja a források alatt megadott GitHub oldal mellett az alábbi linken is elérhető:
https://drive.google.com/file/d/1vbUcrcA7x6MYljjyqBywrlpGjzreHXkv