BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

Változatos egészségügyi szenzorok integrációja felhasználóbarát mobil mérési folyamatba

Tudományos Diákköri Konferencia

Szerzők
Kovács Levente
Mihályi László
Tóth Ádám

Konzulens
Dr. Forstner Bertalan

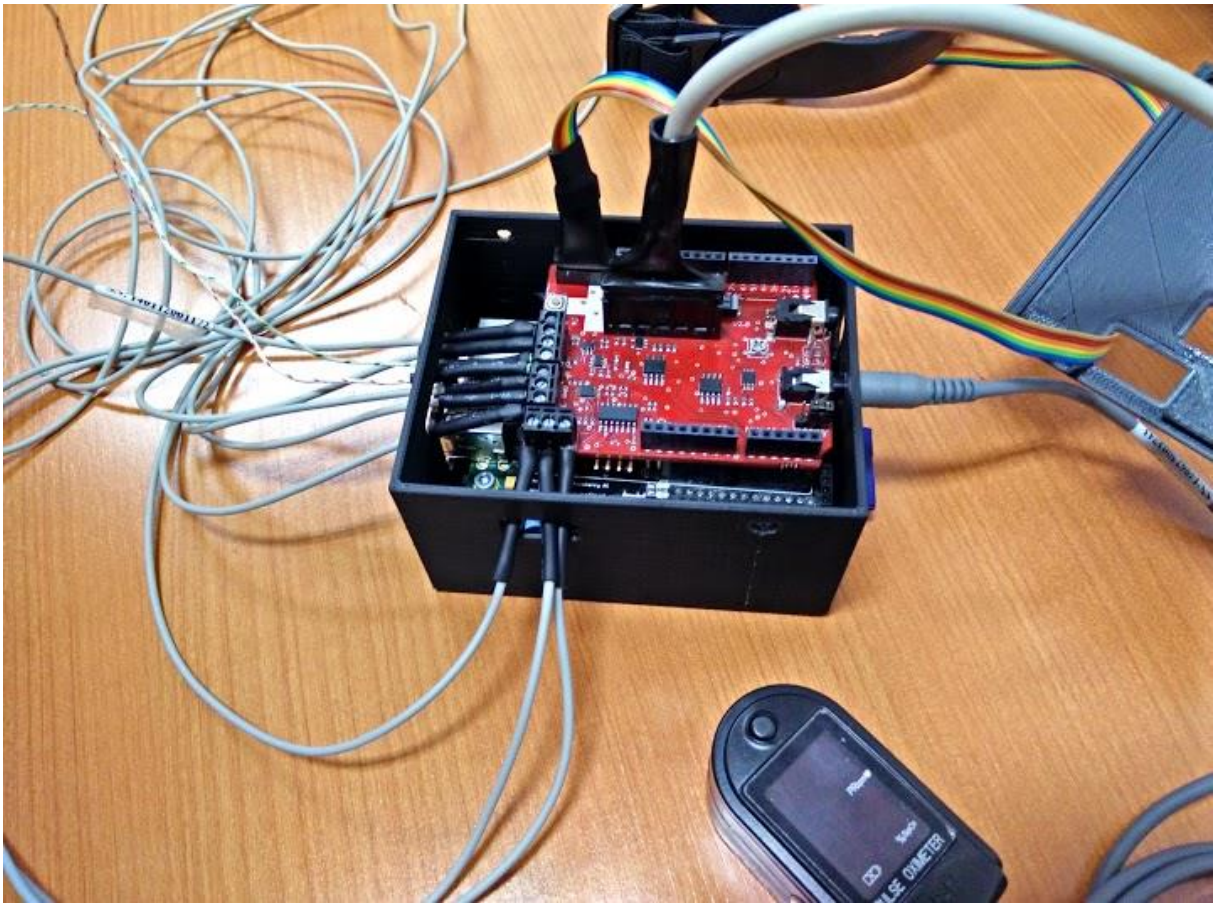
2015.11.17.

Tartalomjegyzék

1. Bevezetés

Jelen dolgozatunk a Budapesti Műszaki és Gazdaságtudományi Egyetem által meghirdetett 2015 évi Tudományos Diákköri Konferenciára elkészített rendszerünk bemutatására szolgál.

A feladatunk egy olyan elosztott architektúrájú rendszer elkészítése volt, melyben egészségügyi szenzorok adatait dolgozzuk fel és tesszük elérhetővé mobil felületen. A kitűzött célunk nemcsak a vezeték nélküli szenzorok bevonása volt, így egy Raspberry Pi[1] alapú kontroller segítségével vezetékes mérőket, mint amilyen a Cooking Hacks[2] cég által összeállított népszerű egészségügyi szenzorok bevonását is elérhetővé tesszük, amit az 1. ábra szemléltet. A mérési folyamat során az adatokat a Pi egy szerverhez juttatja el, amely ezeket feldolgozza és eltárolja. Ehhez a szerverhez tudnak csatlakozni az Android kliensek, amelyek kezdeményezik az egyes méréseket, és le tudják kérdezni a felhasználóik mérési eredményeit. A felhasználók visszajelzést kapnak arról, hogy a mért értékek az egészséges értékeknek megfelelnek-e. Amennyiben az érték egészségtelennek számít a szakértői adatok vagy a felhasználóhoz kötődő trend alapján, úgy tájékoztatást ad a rendszer, hogy mik az ajánlott teendők.



1. ábra. A méréseket végző Raspberry Pi

A projektet egy három fős csapatban készítettük el úgy, hogy mindenki a projekt egy-egy szegmensének elkészítéséért volt felelős. Az Android alkalmazást Tóth Ádám, a Raspberry-n futó programot Kovács Levente, a szerveralkalmazást pedig Mihályi László implementálta. Az adatok

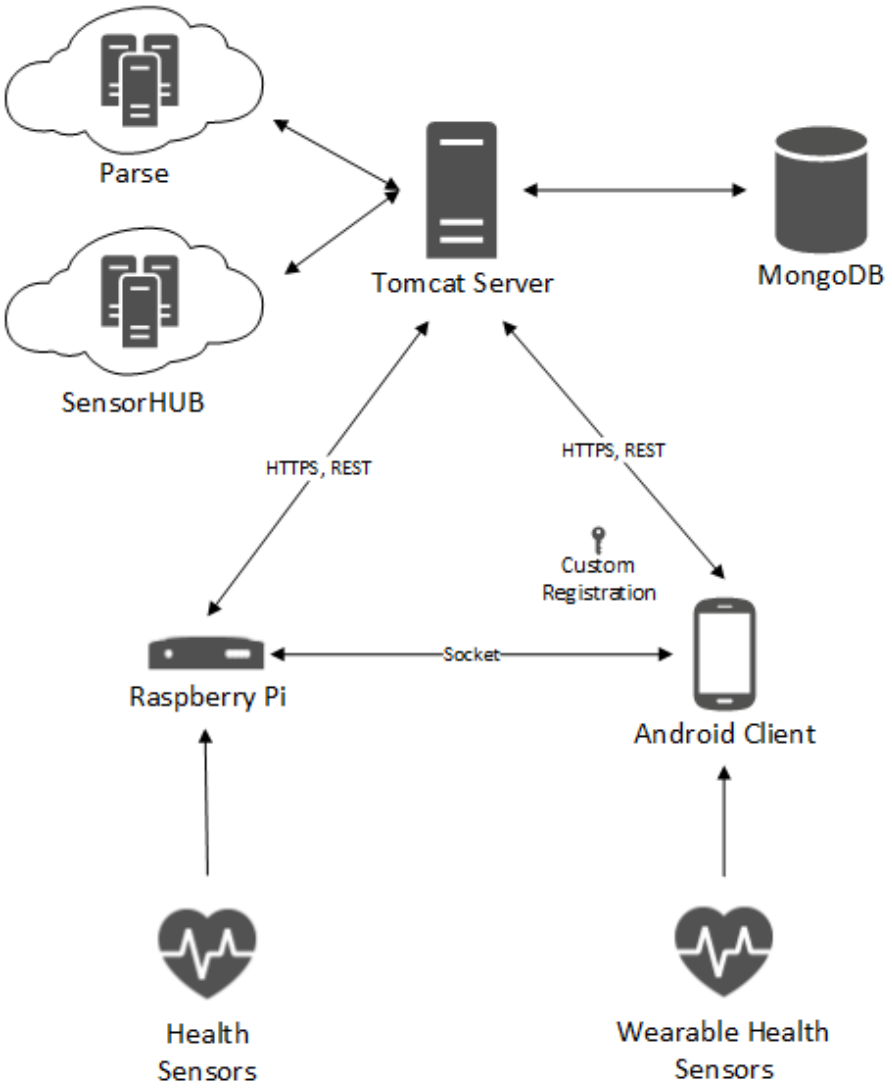
[1] https://www.raspberrypi.org/
[2] https://www.cooking-hacks.com/

elemzésére szolgáló SensorHUB platformon való fejlesztést Kovács Levente és Mihályi László végezte. A projekt készítése során konzulensünk Dr. Forstner Bertalan volt.

A dolgozatunk további részei a következőképpen épülnek fel. A 2. fejezetben bemutatjuk a rendszerünket, illetve a projektet hajtó motivációt. Ezt követik a rendszer szegmenseinek részletes bemutatása. A 6. fejezetben a SensorHUB-ról és a megvalósított korreláció keresésről írunk, majd a 7. fejezetben a rendszerünk szegmensei közötti kommunikációról, a 8. fejezetben pedig a felhasználók és a mérések hitelesítésére térünk ki. Ezt követően a rendszerünk teszt programjairól lehet olvasni, majd a dolgozatunkat az összefoglalás zárja.

2. A projekt bemutatása

A rendszerünk architektúrája meglehetősen összetett, egy magas szintű összefoglalását mutatja a 2. ábra. Alapvetően három szegmenset különböztethetünk meg. Központban található a szerver, melynek a felelőssége az adatok fogadása, illetve tárolása. Ezeken túl interfészt nyújt az Android[3] alkalmazások számára, melynek segítségével a felhasználók hozzáférnek saját mérési adataikhoz és azok kiértékeléséhez.



2. ábra. A rendszer általános architektúrája

A szerverhez HTTPS protokollt felhasználva REST API [**1**]-n csatlakozik a Raspberry Pi. Hozzá vannak csatlakoztatva a Cooking Hacks cég által összeállított szenzorok. A méréseket közvetlenül a Pi végzi, amit a felhasználó telefonja segítségével tud irányítani. Azonban az okostelefonnak is lehetősége van mérések lebonyolítására vezeték nélküli szenzorok esetén. Erre példa a Zephyr HxM[4], ami egy viselhető szívritmus mérő eszköz, ami már illesztve van a rendszerünkhöz. További szenzorok

[3] https://www.android.com/
[4] http://www.zephyranywhere.com/

integrálásán is dolgozunk, így például Fitbit Aria[5] okosmérleg, vagy a Quardio[6] cég által készített Quardiocore. De népszerű okosórák és okospólók támogatásán is dolgozunk.

A Raspberry Pi-hez hasonlóan tudnak az Android kliensek is csatlakozni a szerverhez annyi kivétellel, hogy a nyújtott interfészt csak előzetes regisztrációt követően érhetik el. A rendszerünkben profillal rendelkező felhasználóknak sikeres bejelentkezés után lehetőségük van új mérést indítani, majd a saját eredményeiket és azok kiértékeléseit megtekinteni.

A szerverhez csatlakoznak még különböző felhőszolgáltatások. Ilyen a mobilalkalmazás fejlesztők körében népszerű Parse[7], illetve az Automatizálási és Alkalmazott Informatikai tanszéken fejlesztett SensorHUB platform is. A kutatásunk során először a Parse BaaS (Backend as a Service) platformját vizsgáltuk és az integrálást követően azt tapsztaltuk, hogy big data analízisre nem megfelelő az ilyen rendszerű szolgáltatás. Szemben a SensorHUB-bal, aminek a felhasználásával lehetőségünk nyílik a mérési adatokon korreláció keresést végezni.

2.1. A kutatás motivációja

Napjainkban az IoT rendszerek egyre népszerűbbek [**2**], számos területen megtalálhatóak már. Ez alól a mobil egészségügy sem lehet kivétel, folyamatosak a szakmai kutatások. Azonban az egészségügyi adatokat az országok nagyon szigorúan kezelik [**3**], a betegekről semmilyen információ sem hagyhatja el az országhatárokat. További terjedést korlátozó problémát jelent a mérőberendezések ára[8], amit a legtöbb ember nem engedhet meg magának.

Célunk egy olyan rendszer megtervezése és implementálása volt, amely képes megfelelni a legszigorúbb feltételeknek is a világ bármely pontján. Törekedtünk a lehető legolcsóbb megvalósításra, ennek érdekében számtalan újszerű megoldást dolgoztunk ki.

A személyes adatokat olyan módon kezeljük, hogy a mérést végző eszköz semmilyen információja nincs a felhasználóról. A mérés feldolgozó szervernek pedig csak anonim adatok állnak rendelkezésre, ezzel megakadályozva a felhasználók azonosítását.

Ahhoz, hogy a mérőeszközök megengedhető áron elérhetőek, mégis megfelelő minőségűek legyenek, a rendszerünket a szenzorok megosztására (sharing economy) [**4**] alapoztuk. Ez az újszerű megoldás lehetővé teszi, hogy a mérőegység központilag működjön akár patikákban, vagy orvosi rendelőkben is. Azonban lehetőséget biztosítunk személyes felhasználásra is, amely például viselhető szenzorok használatát is megengedi. A költséghatékonyság jegyében a központi mérőegységek Raspberry Pi mikrokontrollerekre terveztük. De platform függetlenül is futtatható a szoftver, mivel Java nyelven[9] lett implementálva.

További újszerű megoldás jelent, hogy a felhasználók okostelefonjukat használhatják a mérés vezérlésére, illetve a folyamat felügyeletére, így a központi egységhez nem volt szükség külön kijelzőre, illetve beviteli eszközre. Ezzel növeltük a felhasználói élményt, mivel a jól megszokott készüléküket használják erre, mégis csökkentettük a rendszer összköltségét.

[5] https://www.fitbit.com/aria
[6] https://www.getqardio.com/
[7] https://www.parse.com/
[8] https://www.cooking-hacks.com/ehealth-sensors-complete-kit-biometric-medical-arduino-raspberry-pi
[9] http://www.java.com/en/about/

Újszerű még, hogy egy big data [**5**] rendszeren egészségügyi adatok közötti korreláció keresést végzünk, ezzel lehetővé téve, hogy korai előrejelzést nyújtsunk a felhasználóknak az adataikból kikövetkeztetett betegségeikről. Emellett olyan új összefüggéseket is felfedezhetünk, amelyek az orvostudomány számára eddig még nem voltak ismertek, ezzel segítve az egészségügy fejlődését.

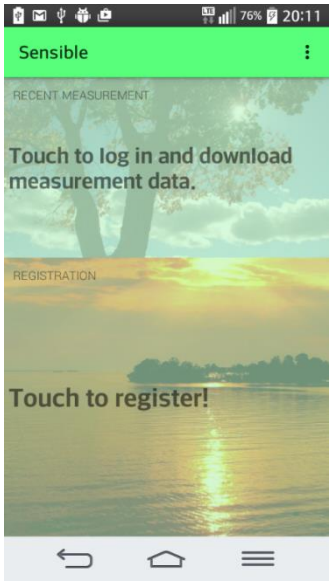
A kutatásunk létjogosultságát alátámasztja az a tény is, miszerint a General Electric[10] is külön részlegen foglalkozik ezzel a területtel, míg ez a projekt a Magyar Telekom megkeresésére indult, ami Magyarország egyik legnagyobb vállalata.

2.2. Szcenáriók

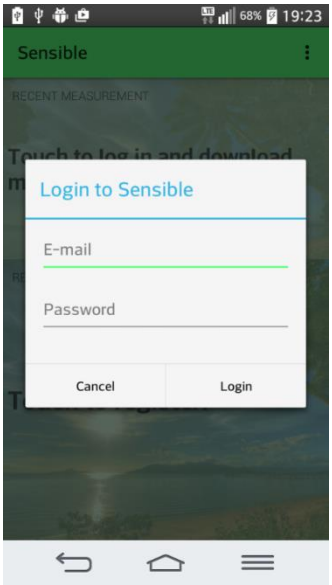
Ebben a fejezetben a jelenleg implementált funkciók felhasználói szempontú bemutatása található. Ezzel az általános használati esetek bemutatása a cél. A második rész egy egyszerű mérési folyamat bemutatásán keresztül vázolja fel a kliens alkalmazás elsődleges feladatait.

2.2.1. Elérhető funkciók

Az alkalmazás indításakor a 3. ábrán látható képernyő fogadja a felhasználót. Ebben az esetben a felhasználó még nem regisztrált, vagy nincs bejelentkezve. A képernyő két része a menü elemeiként szolgálnak. Ezek aktiválása a belépéshez vagy a regisztrációhoz tartozó felugró ablak megjelenését eredményezik. A következő képen a bejelentkező ablak megjelenése látható (4. ábra). Ezen adhatja meg a felhasználó a belépéshez szükséges e-mail címet és a hozzá tartozó jelszót.



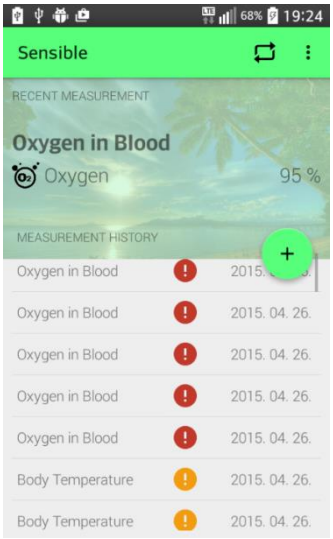
3. ábra. Kezdőképernyő



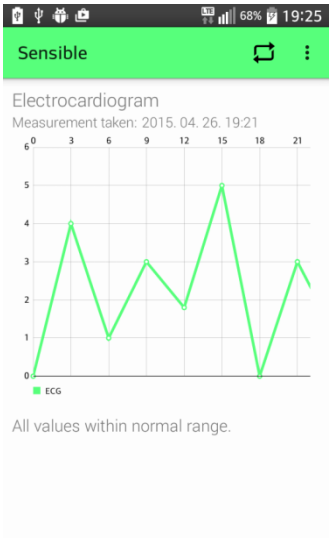
4. ábra. Bejelentkezési felület

A belépést követően megjelenik az alkalmazás elsődleges felülete (5. ábra). Ezen a felületen találhatók meg a funkciókat ellátó gombok, valamint az előző mérésekből származó eredmények is.

[10] http://www.ge.com/hu/hu/company/factsheet_hu.html



5. ábra. Alkalmazás fő felülete



6. ábra. Mérési eredmény diagramja

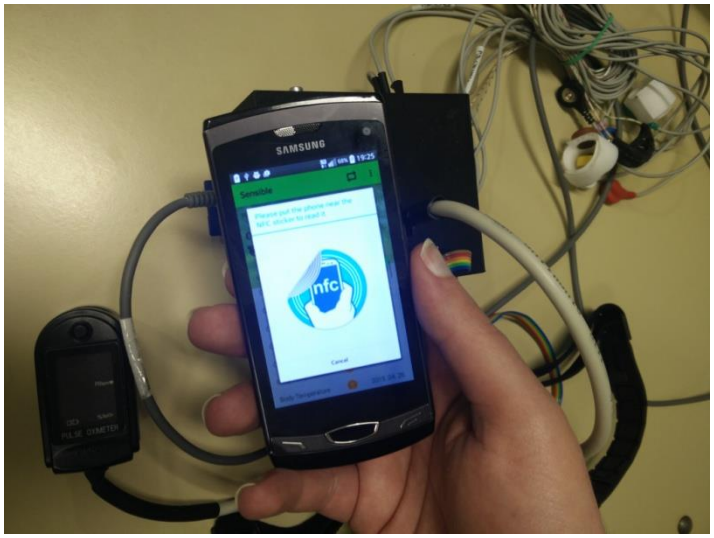
Az ActionBaron található egy frissítés gomb, ami adatok ismételt letöltésére, azok frissítésére szolgál. A menüből elérhető a kijelentkezés. A felső harmadban látható Recent Measurement rész tartalmazza az utolsó mérés eredményét. Érintés esemény hatására megjelenik a hozzá tartozó részletező képernyő. Folytonos értékű mérések esetében az eredmények diagramon keresztül is láthatóak (6. ábra).

A kapott adatokhoz egy rövid szöveges értékelés is tartozik, ez alapján adhatók meg különböző egészségügyi összefüggésen alapuló javaslatok, figyelmeztetések is. Ezek a figyelmeztetések a lista képernyőn is láthatók. A probléma súlyosságától függően a listaelemekre jelzések kerülnek. A főképernyőn szereplő listaelemek szintén kiválaszthatóak a részletező oldal megjelenítéséhez.

A FloatingActionButton feladata egy új mérési folyamat megkezdése. Ennek részletezése olvasható a következő pontban.

2.2.2. Mérési folyamat ismertetése

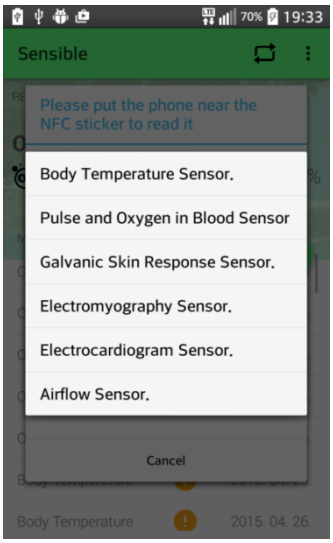
A lebegő gomb megnyomását követően egy felugró ablak jelzi, hogy megkezdődött a folyamat. Első lépésben a mérőállomáshoz tartozó NFC kód beolvasása szükséges. Ezt szemlélteti az alábbi ábra.



7. ábra. NFC beolvasása

A kód leolvasása után a kliens a szerver felé továbbítja az információt. Az azonosítást követően a felhasználói alkalmazás megkapja a kapcsolódáshoz szükséges eszköz adatait, valamint a hitelesítéshez szükséges tokent. Amennyiben ezek a folyamatok sikeresen zajlanak le, a Rasberry Pi eszköz visszaküldi az elérhető mérőeszközök listáját (8. ábra).

Az így kapott listából kiválasztható a vizsgálathoz alkalmazandó eszköz. A méréshez használt eszköz kiválasztása a mérési folyamat elindítását is eredményezi. A mérés befejeztével az eredmények elérhetővé válnak az eszközön, valamint a szerver is tárolja azokat.



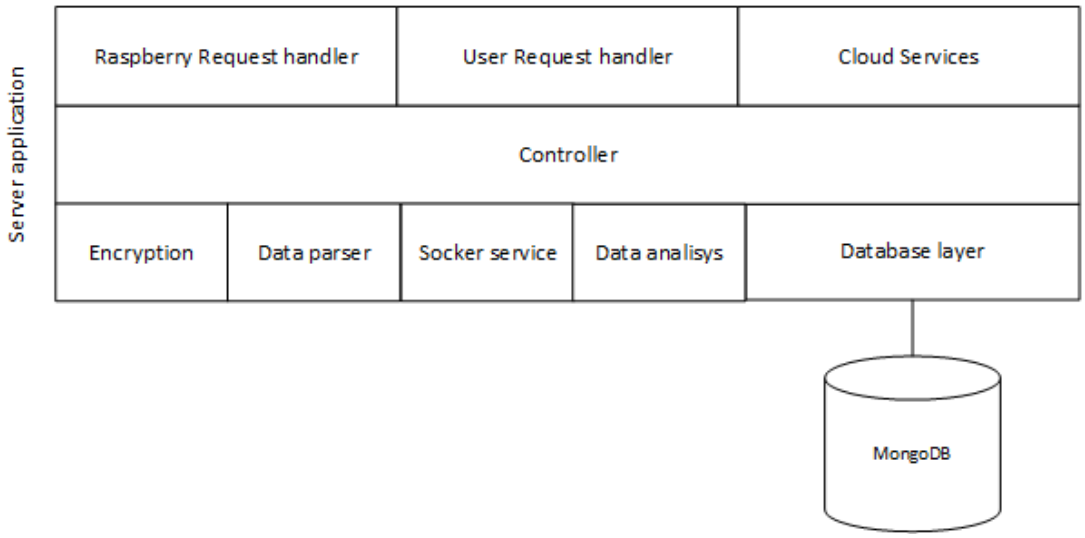
8. ábra. Szenzor kiválasztása

3. Szerver

A szerver a rendszer központi egysége. Összeköttetésben áll a kliensekkel, fogadja a mérési adatokat a Raspberry Pi-től és lehetőséget biztosít a felhasználó számára, hogy a saját mérési adatait megtekinthesse. A következő alfejezetekben a szerver részletes felépítéséről, szolgáltatásairól lehet majd olvasni.

3.1. A szerver architektúrája

A szerveralkalmazás Java programozási nyelven íródott, melyet egy Tomcat[11] szerver szolgál ki. Fő feladata, hogy REST API-t nyújtson az Androidos alkalmazások és a Raspberry Pi-k számára. Ennek egyszerű biztosítása érdekében a Jersey[12] nevű web szolgáltatást használtuk fel. A szerver alapértelmezetten két fő interfészt biztosít a kliensek számára, ahogy ez az alábbi ábrán is megfigyelhető. A Raspberry Request handler biztosítja a szerveralkalmazás Pi-k, míg a User Request handler a felhasználók által történő használatát.



9. ábra. Szerver architektúra

A kliensektől a szerverre érkező adatok feldolgozáson esnek keresztül, melyet a Controller réteg végez el. Itt a lehetséges további szolgáltatások felhasználásával (például titkosítás, adatok feldolgozása, elemzése, kiértékelése, azok felhőszolgáltatásba történő továbbítása) a szükséges műveletek elvégződnek. Természetesen a legalapvetőbb funkciót is ellátja a szerveralkalmazás, azaz az adatok tárolását. Erre a célra az elosztott működésre is alkalmas MongoDB[13]-t használjuk.

3.2. Adatbázis

A rendszerben adatokat tárolunk a felhasználókról, illetve természetesen a mérési adatokról is. Alapvető adatokat tárolunk a mérőeszközökről (jelen esetben a Pi) is azért, hogy később beazonosítható legyen, ha folyamatosan rossz mérési eredmények születnek egy elromlott szenzor, vagy készülék miatt. Így ezáltal a hibás mérésnek bélyegzett adatokat nem vesszük figyelembe a korreláció keresés során, ezzel biztosítva a pontosabb eredményeket, kimutatásokat.

[11] http://tomcat.apache.org/
[12] https://jersey.java.net/
[13] https://www.mongodb.org/

A felhasználókról alapvetően csak és kizárólag olyan információkat tárolunk, amelyek segítségével pontosabban tudjuk elvégezni a kiértékeléseket. Egy felhasználónak regisztráció során a szokásos felhasználónév, jelszó páros mellett lehetősége van megadni a születési évét, jelenlegi testmagasságát, illetve testsúlyát. Emellett pedig még a nem megadására is lehetőséget biztosítunk. Ezek mind olyan adatok, amelyek segítségével pontosabban tudjuk eldönteni egy-egy mérési eredményről, hogy azaz egészséges tartományba tartozik, vagy nem. Hat éves kisgyermeknél az ideális vérnyomásérték[14] lányok esetében 100/62, míg fiúk esetében 102/60. Azonban egy 30 éves felnőtt szervezet esetében ezek az értékek nőknél 113/72, férfiaknál pedig 122/77. Idősebb korban pedig mindkét nem esetében 130/80 felett található a vérnyomás egészséges tartománya.

Ezekből az adatokból jól látszik, hogy amennyiben egy kisgyermeknél a vérnyomás 120/70, akkor teljesen másképpen kell eljárni, mint egy felnőtt esetében, hiszen a gyermekeknél ez az érték kritikusan magasnak számít, míg az idősebbeknél ez egy ideális eredmény. Ezeket befolyásolják még a testsúly és a testmagasság értékei is, amelyek minél magasabbak, úgy a vérnyomás értéke is növekszik. Mivel az egyes mérési értékek ideális tartományai változatlanok például a túlsúly esetében is, ezért ezekkel az adatokkal csak visszajelzést tudunk adni a felhasználók számára, hogy ez is okozhatja az egészséges tartományon kívül eső mérési eredményt. Ezáltal a testsúly csökkentésével a vérnyomás értéke is rendeződhet.

A rendszer tervezése és implementálása során nem tartottuk jó megoldásnak azt, ha a célközönséget egy adott korosztály, annak átlagos testmagasságához és testsúlyához igazítsuk. Ennek következtében minden felhasználó a saját felelősségére megadott értékeinek megfelelően tudunk részletesebb és pontosabb diagnózist felállítani.

A mérési eredmények tárolásához NoSQL adatbázisra volt szükség. A rendszer esetében azért nem volt jó választás a relációs adatbázis, mert vannak olyan szenzorok, amelyek nem egy-egy átlagolt mérési eredményt szolgáltatnak, hanem például folytonos adatokat. Átlagolt eredményekkel dolgozik például a vérnyomásmérő, folytonos adatokkal pedig az EKG.

Folytonos adatokat mérő szenzorok esetében egy-egy mintavételből nem lehet következtetéseket levonni. Egyértelmű, hogy egy EKG mérés mintavételezési adatainak átlagolásából nem lehet következtetéseket levonni a szívműködést kísérő elektromos változásokról. Az összes mintavétel egységesen szolgáltatja az információkat a szív működésével kapcsolatban. Mivel egy-egy mérés során nem feltétlen lehet azt garantálnunk, hogy ugyanannyi mintavételezés történik, így kézenfekvő volt ezeket az adatokat egy kollekció objektumában eltárolni, amelyre egy nem relációs adatbázis biztosít lehetőséget.

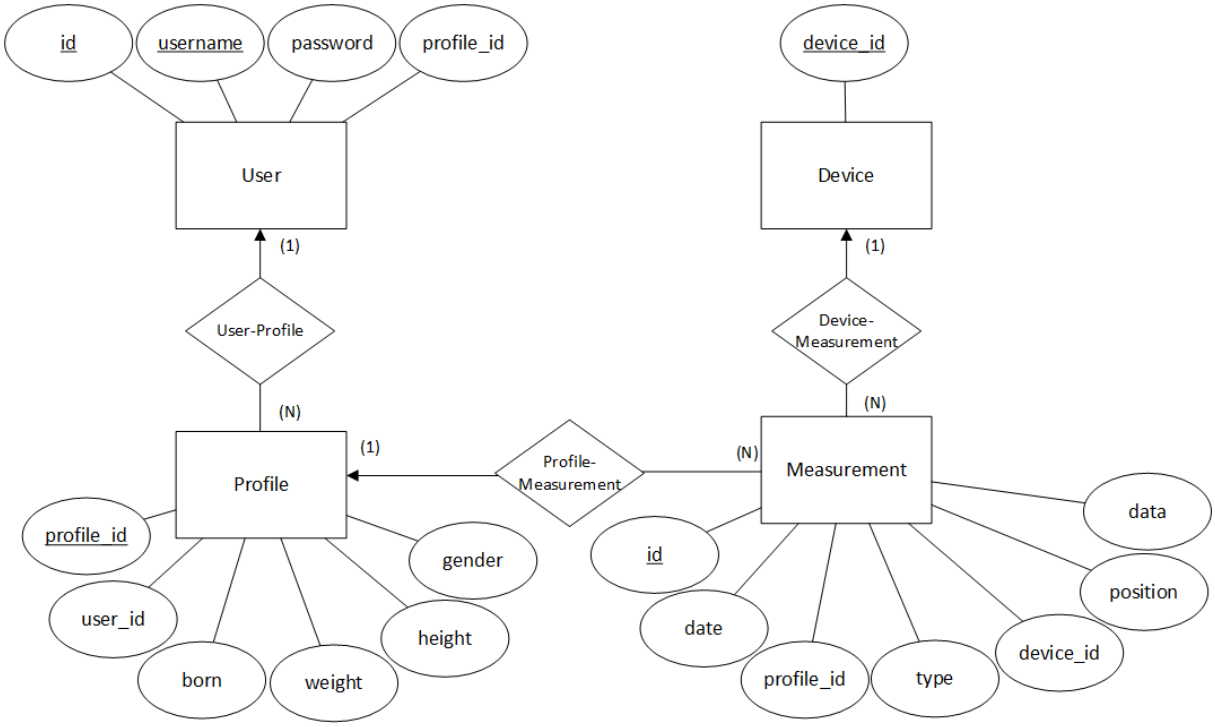
A mérési adatok egységes kezelése érdekében az átlagolt mérési eredményeket is nem relációs adatbázisban tároljuk. Így lett biztosítva az, hogy mindegyik szenzortípus eredményeit egyetlen sémával le lehet írni. Ennek következtében a szerveralkalmazásnak az adatok lekérdezése során nem kell meghatároznia az egyes mérési eredmények típusát, mert anélkül is ki tudja szolgálni a kérést.

[14] Az ideális mérési eredmények a Webbeteg.hu weboldaláról származnak.

3.2.1. Adatbázis implementációja

Az előzőekben már volt arról szó, hogy az adatok tárolására egy nem relációs adatbázis használatát véltük a legjobb megoldásnak. Ennek implementálására az elosztott rendszerek között egyik legjobban elterjedt adatbázis-kezelőt, a MongoDB [**6**]-t választottuk.

A MongoDB-ben dokumentumokat tárolhatunk, amelyeknek illeszkedniük kell az előzőekben megadott kollekciók sémáira. Az egyes dokumentumok attribútumainak típusai ugyan meghatározhatóak, ugyanakkor lehetőségünk van általánosan definiálni őket. Pontosan ezt használtuk ki arra a célra, hogy a szerveralkalmazásnak az adatok lekérdezése során ne kelljen meghatároznia az adatok típusát.



10. ábra. Az adatbázis ER-modellje

Az adatbázisunkban jelenleg négy kollekciósémát különböztethetünk meg, ahogy ez a fenti ábrán látható. Ezek sorra a Profiles, Users, Devices és Measurements.

3.2.1.1. Profiles kollekció

A Profiles kollekcióban tároljuk a felhasználók profil adatait. Ezeket regisztráció során lehet meghatározni, habár lehetőséget biztosítunk a későbbi megváltoztatására is. Ilyen profil adat például a születés éve, a felhasználó neme, vagy aktuális testmagassága, testsúlya.

A profiladatok tárolását meg lehetett volna valósítani a felhasználói adatokkal egy kollekcióban. Ezt a megoldást azért vetettük el, mert a mérések kiértékelését követően a rendszerünk nem tárolja el az eredményeket, mivel azok származtatott értékek, költségesebb eltárolni ezeket, mint újra kiszámolni.

A mérési eredmények kiértékelésében és a diagnózis felállításában fontos szerepet játszanak a profiladatok. Ennek függvényében ha a felhasználó megváltoztatta volna a profiladatait, akkor a korábbi méréseinek értékelései is módosultak volna.

Azonban jelenlegi megoldásunknak köszönhetően minden változtatást eltárolunk, így visszamenőleg ismét ki tudjuk értékelni a méréseket, illetve a korrelációkeresést sem fogja negatívan befolyásolni egy időközben megváltoztatott profiladat. Ehhez azonban még a kollekcióban el kellett tárolni a felhasználó egyedi azonosítóját is, akihez az adott profil tartozik.

3.2.1.2. Users kollekció

A felhasználók a rendszer használatához szükséges adatait a Users kollekcióban tároljuk. Ilyen adat például a felhasználónév és a jelszó. A jelszavaknak természetesen csak egy hash függvény által kódolt változatát tároljuk, ami egyértelműen nem fejthető vissza. A felhasználónév pedig megegyezik az e-mail címmel, ezzel könnyítve és gyorsítva a regisztrációs folyamatot.

Természetesen ezeken kívül még el kell tárolnunk a felhasználóhoz éppen aktuálisan tartozó profil azonosítót. Minden elvégzett mérés kiértékeléséhez ez a profilazonosító lesz felhasználva, ezekkel a paraméterekkel lesz a korrelációkeresés is elvégezve.

3.2.1.3. Devices kollekció

A rendszerünkben alapvető adatokat tárolunk a mérőeszközökről is. Erre azért van szükség, mert amennyiben egy mérőeszköz elromlik, úgy az általa közvetített rossz mérési eredményeket ugyanúgy beleszámolnánk a korreláció keresésbe, mint a helyes adatokat. Illetve az eszközök rendellenes működésére felhívható a tulajdonosok figyelme.

A mérőeszközöknek is rendelkezniük kell egy olyan azonosítóval, ami egyedivé teszi őket a kollekcióban. Erre a célra lehetőség szerint az eszközök MAC címeit használjuk fel, amely felhasználó általi módosítása a legtöbb esetben nem lehetséges.

3.2.1.4. Measurements kollekció

Természetesen a mérések eredményeit is tárolni szeretnénk, erre a Measurements kollekciót használjuk. A NoSQL sajátosságait felhasználva lehetőségünk van a rendszerünkben jelenlevő mind a három típusú eszköz adatainak a tárolására. Így az egyszerű egy értékeket közvetítő (például a hőmérsékletmérő), a folytonos adatokkal dolgozó (például az EKG), illetve a strukturális adatokat szolgáltató vérnyomásmérő értékei is ebben a kollekcióban találhatóak.

A mérések konkrét értékein túl eltároljuk a mérés keletkezésének pontos időpontját, a mérést végző felhasználó aktuális profil azonosítóját, amely segítségével a kiértékelés és a korrelációkeresés végezhető el, a mérőeszköz azonosítóját, melynek felhasználásával eldönthető, hogy az adott eszköznek volt-e már helytelen mérési folyamata. Ezeken túl természetesen a mérés típusát is meg kell határoznunk, hogy a mérés melyik szenzor felhasználásával történt, illetve egy kiegészítő információt is mellékelünk, amelyből kiderül, hogy a felhasználó milyen pozícióban volt a mérés elvégzésekor - feltételezve, hogy használta a pozíciót meghatározó szenzort.

3.2.2. Adatbázis bővíthetősége

A tervezés során kritérium volt, hogy a későbbiekben könnyedén lehessen a rendszerben újabb szenzorokat támogatni, ennek megfelelően lett az adatbázis felépítve.

A mérési típusok úgy lettek meghatározva, hogy azokban kizárólag csak elemi mérések eredményei szerepeljenek. Erre azért volt szükség, mert vannak olyan szenzorok is, amik egyszerre több egészségügyi éréket mérnek, azonban nem feltétlen tartoznak össze. Ilyenre példa a Cooking Hacks szenzorcsomagjában szereplő Pulzus és véroxigén szint mérő szenzor. Mivel egy vérnyomásmérő is

tud pulzus adatot mérni, így értelmetlen ezeket az adatokat több helyen tárolni az adatbázisban, kézenfekvő őket elemi szinten szétválasztani.

Ennek a megoldásnak köszönhetően újabb szenzorok támogatása nem jelent strukturális átalakítást a későbbiekben. Mivel a mérések típusai az adatbázisban nem külön kollekciókban jelennek meg, így elegendő csak azzal az információval rendelkezni, hogy az új szenzor egyszerű mintavételezett, folytonos, vagy strukturális adatokkal dolgozik-e. Mivel a három típus minden lehetőséget lefed, így külön átalakítást nem igényel egy új szenzor felvétele.

3.3. Szolgáltatott interfészek

A szerver két típusú kliens kéréseit szolgálja ki. Az egyik a mérési eredményeket szolgáltató eszköz (Raspberry), a másik pedig a mért eredményeket lekérdező Android alkalmazás. Mind a kettő kliens számára definiálni kellett egy interfészt, ahol elérhetik a szerver szolgáltatásait.

3.3.1. Raspberry Pi interfésze

A Raspberry Pi feladata a felhasználó által kezdeményezett mérés elvégzése. Az általa használt interfészeket szemlélteti az alábbi táblázat. Ahhoz, hogy a mérőeszközökkel a kapcsolatot fel lehessen építeni, bejelentkezés szükséges. Ekkor a Pi az egyedi azonosítóját elküldi a szerverre, ahol bekerül az aktív eszközök listájára.

Interfész	Cím	Kérés típusa	Média típusa	Adat
Bejelentkezés	Sensible/pi/login	Post	Application/json	DeviceID
Token validálás	Sensible/pi/checktoken	Post	Application/json	Kódolt Token
Adatok felküldése	Sensible/pi/upload	Post	Application/json	Kódolt mérési és META adatok

1. táblázat. Raspberry Pi interfész

Ezek mellett a Raspberry-nek ellenőriztetnie kell a felhasználótól kapott tokent, hogy az érvényes-e. Erre azért van szükség, hogy csak olyan mérést lehessen kezdeményezni, amelyet a szerver előzőleg már hitelesnek talált. Erről bővebben a Hitelesítés fejezetben lehet olvasni.

Továbbá a Pi-nek szüksége van egy olyan interfészre, melyen keresztül a mért adatokat el tudja küldeni a szervernek. Ekkor a küldött információ tartalmazza a mérési eredményeken túl a tokent, ami segítségével a hitelesítés zajlik, a mérőeszköz azonosítóját, és a mérés dátumát, amik a kódolt tokenben szereplő adatokkal lesznek összevetve. Ezen adatok mellett még természetesen a szenzor egyedi azonosítója is megtalálható, illetve a beteg pozíciója a mérés során.

Ezeken a lehetőségeken kívül a Raspberry Socket kapcsolatot is tud létesíteni a szerverrel, amelyet csak akkor használunk ha probléma van kapcsolat felépítésével. A Socketen keresztül történő kommunikáció előfordulhat akkor, ha az Android kliens nem tud kommunikálni a Pi-vel. Erről bővebben a Kommunikáció fejezetben írtunk.

3.3.2. Android kliens interfésze

A felhasználóknak lehetősége van a saját mérési eredményeinek a megtekintésére. Ezzel kapcsolatban a szervernek rendelkeznie kell olyan interfészekkel, amelyeken keresztül a felhasználó számára biztosítani lehet az adatok elérését. Alapvetően megkülönböztethetünk hitelesítéssel és a mérésekkel kapcsolatos adatokat.

3.3.2.1. Hitelesítés

A hitelesítéssel kapcsolatos interfészeket ábrázolja a 2. táblázat. A felhasználók számára biztosítani kellett a rendszerbe való regisztrációjukat. Ekkor lehetőségük van a felhasználónevükön és jelszavukon túl még a mérési eredményeik pontosabb kiértékelése céljából a születési évüket, jelenlegi testmagasságukat, testsúlyukat és nemüket megadni.

Interfész	Cím	Kérés típusa	Média típusa	Adat
Bejelentkezés	Sensible/auth/login	Post	Application/json	Felhasználónév, jelszó
Kijelentkezés	Sensible/auth/logout	Get	Plain text	-
Regisztráció	Sensible/auth/register	Post	Application/json	Profil adatok
Adatok módosítása	Sensible/auth/updateprofile	Post	Application/json	Profil adatok

2. táblázat. Hitelesítéssel kapcsolatos interfészek

Ezeken túl még az adatok módosítására is lehetőséget kell nyújtani. Az adatok módosítása során is ugyanazokat a paramétereket tudják megadni a felhasználók, mint a regisztráció során. Ezzel kapcsolatban felmerült, hogy problémát jelenthetnek hosszú távon a felhasználó nevek változtatásai, mivel ezek egyedi azonosítói a felhasználóknak. Azonban a rendszerben az alkalmazás használóihoz egy másik egyedi azonosító rendelésével - ami egyben az adatbázisban az elsődleges kulcs is - megoldható a probléma. Ekkor már csak arra kell figyelni, hogy az adatok módosítását követően az új felhasználónév is egyedi maradjon a rendszerben.

Egyszerűbb megoldást jelentett volna az, ha nincs engedélyezve a felhasználók számára a nevük változtatása. Azonban a felhasználók kényelmét figyelembe véve a tervezéskor úgy döntöttünk, hogy ne kelljen regisztrációkor külön megadni az e-mail címet és a felhasználónevet, mivel előbbi elegendő a számunkra. Ezek mellett a felhasználók többségének problémát jelent egy felhasználónév kitalálása, így kézenfekvő volt a kettőt együtt kezelni.

Természetesen szükség van a két alapvető funkció, vagyis a bejelentkezés és a kijelentkezés biztosítására is. Bejelentkezéskor a felhasználók elküldik a felhasználónevüket és a jelszavukat, amit a szerveralkalmazás összevet az adatbázisban található információkkal. Amennyiben helyes adatokat küldött el, úgy a szerver egy munkamenetet biztosít a számára, ami 30 perc tétlenséget követően megszűnik. Amennyiben a felhasználó az Android alkalmazásban a kijelentkezést választja, akkor a felhasználó munkamenete érvénytelenítve lesz.

3.3.2.2. Adatelérés

Ahhoz, hogy a felhasználó kapcsolatba tudjon lépni az Android alkalmazáson keresztül a Raspberry Pi-vel, előbb meg kell tudnia az adott mérőeszköz IP címét a szervertől. Ekkor a felhasználó elküldi a szervernek a Pi egyedi azonosítóját és amennyiben az megtalálható az aktív eszközök között, úgy válaszul megkapja az IP címet és a mérés hitelesítésére szolgáló Tokent.

Amennyiben az Android kliens és a Raspberry között nem lehet felépíteni a kapcsolatot, úgy a kommunikáció Socketen keresztül történik, amelyet a szerver biztosít. Ebben az esetben a szerver csak a mérések hitelesítésére szolgáló Tokent küldi el a telefonos alkalmazásnak, a Pi IP címét nem. Az alábbi táblázatban az adatelérésekkel kapcsolatos interfészek olvashatóak.

Interfész	Cím	Kérés típusa	Média típusa	Adat
Kapcsolat felépítése	Sensible/client/createconnection	Post	x-www-form-urlencoded	DeviceID
Adatok letöltése	Sensible/client/download/{from}/{dir}/{quantity}/{type}	Get	Plaint text	Adat filterek

3. táblázat. Adateléréssel kapcsolatos interfészek

Ezek mellett a felhasználók számára talán a legfontosabb interfész a mérési adataik lekérdezésére szolgál. Ekkor a felhasználó megadja azt a szenzort, amelynek méréseire kíváncsi. Lehetőség van arra is, hogy a mérési értékeket - szenzortól függetlenül - ömlesztve lehessen letölteni. Mindezek mellett a felhasználó megadhatja azt is, hogy mennyi eredményt szeretne letölteni, illetve egy dátumot, amelytől számítva vagy frissebb, vagy régebbi értékekre szeretne szűrni, mely irányt szintén meghatározhatja.

3.3.3. Teszt interfészek

Egy rendszer fejlesztése során fontos a tesztelések biztosítása, így a megfelelő teszt interfészek definiálásával a szerveralkalmazás olyan részeit is ellenőrizni lehetett, ami közvetlenül nem jelenik meg egyik kliens számára sem. Ezen interfészek találhatóak a 4. táblázatban.

A szerver a mérési, illetve a felhasználói adatokat NoSQL adatbázisban tárolja. Az adatbázisban lévő dokumentumok, kollekciók és adatok vizsgálatára egyszerű és kiforrott megoldás hiányában nem volt lehetőség. Azonban a megfelelő interfészek definiálásával a teljes adatbázis ellenőrizhetővé vált.

Interfész	Cím	Kérés típusa	Média típusa	Adat
MongoDB adatbázis lista	Sensible/test/mongo/dblist	Get	Plain text	-
MongoDB adatbázis törlés	Sensible/test/mongo/delete	Post	Application/json	Hitelesítő jelszó
MongoDB kollekció lista	Sensible/test/mongo/colllist/{db}	Get	Plain text	Adatbázis neve
MongoDB adatok listája	Sensible/test/mongo/{sensor}	Get	Plain text	Szenzor neve
Mérési adatok kódolása	Sensible/test/encode	Post	Application/json	Mérési adat

4. táblázat. Tesztelési interfészek

A fejlesztés során nem mindig volt lehetőségünk komplett teszteléseket végezni, amelyben mind a Raspberry, mind az Android alkalmazás, mind pedig a szerver részt vegyen. Emiatt a szerverre való adatfeltöltés tesztelése az implementált titkosítás miatt nehézkessé vált. Definiálni kellett egy olyan interfészt, amelyre a mérési adatokat tartalmazó JSON [**7**]-t elküldve visszakapjuk azt a titkosított szöveget, amelyet a szerver az adatok feltöltése során dekódolni tud. Ennek köszönhetően a szerver összes fő funkciója könnyen ellenőrizhetővé vált.

3.4. Kapcsolat a felhőszolgáltatásokkal

Annak érdekében, hogy az adatokat redundánsan, könnyen visszaállítható formában tároljuk, illetve azokon korreláció keresést végezzünk, különböző felhőszolgáltatásokat kellett igénybe vennünk. Rendszerünket két platformba integráltuk. Az egyik a mobilfejlesztők körében népszerű Parse, a másik pedig az Automatizálási és Alkalmazott Informatikai Tanszéken fejlesztett SensorHUB.

3.4.1. Parse

A Parse alapvetően egy adott mértékű forgalomig ingyenes BaaS (Backend as a Service), mely a mobilfejlesztők körében népszerű. Az adatok tárolása és azok lekérdezése, illetve a felhasználók hitelesítése és authorizálása úgy lett kialakítva, hogy a lehető legegyszerűbben lehessen a rendszereket hozzáintegrálni és kezelni.

A Parse is egy nem relációs adatbázis-kezelő rendszert használ hasonlóan az általunk készített szerverhez, melynek köszönhetően ugyanolyan adatstruktúrát lehetett kialakítani ebben a platformban, mint amilyet rendszerünkben definiáltunk. Az alábbi képen a Parse felhasználói felülete látható a mérési adatokkal.

date Number	sensorType Number	deviceID Number	position Number	profileID String	data Object	objectId String
1444590099035	1	109	1	561ab1e4e4b0d91…	{"value":40.455056739194546}	5S25KhpVyD
1444590097301	8	109	1	561ab1e4e4b0d91…	{"value":[73.03892483019757,2…	8uIGKRqRpf
1444590095614	8	109	1	561ab1e4e4b0d91…	{"value":[109.84740947217387,…	KsbQD9DBvV
1444590093944	2	109	1	561ab1e4e4b0d91…	{"value":100.00413578000826}	zyYB4ScaG4
1444590092269	6	109	1	561ab1e4e4b0d91…	{"value":[23.16544247022011,8…	yHjQxMKeGU
1444590090631	2	109	1	561ab1e4e4b0d91…	{"value":90.13764385456913}	x4QOuHjbii
1444590088902	7	109	1	561ab1e4e4b0d91…	{"value":[107.71261636964007,…	BDj8VHKb0o
1444590087209	7	109	1	561ab1e4e4b0d91…	{"value":[140.3453766746483,1…	tXAtRARXxR
1444590085566	5	109	1	561ab1e4e4b0d91…	{"value":9.28259365566365}	pAelL7Jip9
1444590083896	3	109	1	561ab1e4e4b0d91…	{"value":{"dia":77.5610220487…	uCeXaVfBkp
1444590081819	7	109	1	561ab1e4e4b0d91…	{"value":[48.92578527609754,1…	Jcj4GL23Dc
1444590080140	4	109	1	561ab1e4e4b0d91…	{"value":51.676961194087895}	xZc5bjSn5N
1444590078474	5	109	1	561ab1e4e4b0d91…	{"value":4.546960847841021}	9HHoSsYo1b
1444590076595	6	109	1	561ab1e4e4b0d91…	{"value":[5.030747484330972,8…	LoSnXVaz59
1444590074922	1	109	1	561ab1e4e4b0d91…	{"value":38.818144729869395}	qGhNqEiWAR

11. ábra. A Parse grafikus felülete

A Parse is - hasonlóan a piacon jelenlévő BaaS rendszerekhez - HTTP protokollt használja a kommunikációra, így az adatok küldésére a platform által definiált interfészekekre kellett a kéréseket elküldeni a megfelelő fejléccel és törzzsel. Az adatküldés sikerességére, vagy hibájára a HTTP státusz kódok segítségével következtethetünk, melyeket a rendszerünkben figyelembe kellett venni.

Az integrálást követően a korreláció keresés megvalósíthatóságának a komplexitását vizsgáltuk. Arra a következtetésre jutottunk, hogy a Parse által nyújtott Cloud Code szolgáltatás nem ideális összefüggések keresésére, mivel az általuk biztosított korlátozott erőforrások nem alkalmasak ilyen számításigényes feladatokra.

3.4.2. SensorHUB

A SensorHUB[15] az Automatizálási és Alkalmazott Informatikai Tanszék saját fejlesztésű IoT platformja. Ez egy olyan elosztott rendszer, melyet a Hadoop menedzsel. A felhasználók mérési adatai közötti összefüggés vizsgálatára használjuk. Ebben a fejezetben a két rendszer integrációjáról írunk, a SensorHUB-ra készített szoftverek és adatbázis szerkezet bemutatását külön fejezetben részletezzük.

Az integrációja során számtalan problémával álltunk szembe. Első körben problémát jelentett az, hogy a korreláció keresés miatt a felhasználók profil adatait is fel kellett küldeni a SensorHUB rendszerébe. Mivel a Parse-ba eredetileg csak a mérési eredményeket továbbítottuk, így a kialakított interfészt bővíteni kellett.

A profil adatok felküldése további problémákat is felvetett. Szerettünk volna megőrizni a felhasználók anonimitását, ahogy ezt több ország biztonsági szabályozása is megköveteli. Ennek érdekében transzformációkat végeztünk az adatokon, aminek köszönhetően nem lehet egyértelműen felhasználóhoz kötni az adatokat.

A SensorHUB platform jelenleg is fejlesztés alatt áll, így merült fel a tanúsítvány érvényesítési probléma. Napjainkban azok a web alkalmazások, amik felhasználói adatokat, jelszavakat kezelnek, a biztonság érdekében HTTPS protokoll felhasználásával kommunikálnak. Ennek használata maga után vonja a tanúsítványok érvényesítését, hitelesítését, mely magas költségeket jelentenek egy fejlesztés alatt álló rendszer esetében.

Ennek megfelelően lehetőséget kellett biztosítani a felhőszolgáltatások felé létrehozott interfészeknél, hogy a kommunikáció során ne ellenőrizze feltétlenül a tanúsítványokat. Ez ugyan nem tanácsos lépés, azonban elkerülhetetlen volt a kommunikáció biztosításához.

3.5. Mérési adatok értékelése

A rendszerünkben a felhasználók mérései kiértékelésre kerülnek annak érdekében, hogy a lehető legtöbb információval tudjuk ellátni őket. Ennek megfelelően a támogatott szenzorok egészségügyi értékeinek alakulásával is foglalkoznunk kellett.

A felhasználókról a kiértékelések pontosítása érdekében profil adatokat is tárolunk. Erre példa a testmagasság, a testtömeg, a születési év, vagy a nem is. Ezek felhasználásával állapítottuk meg az egészségügyi értékek megfelelő tartományait.

Ezen tartományok meghatározásakor két csoportot hoztunk lére. Az egyik csoportba olyan egészségügyi mutatókat válogattunk, amelyek függnek a születési évtől, illetve a felhasználó nemétől, míg a másik csoportba az összes paramétertől függetlenek kerültek. Előbbire példa a vérnyomás, amelynek egészséges tartományának változásáról a szerver oldali adatbázis fejezetben már részletesen is írtunk. Utóbbira pedig példa a vércukorszint, amely születési évtől, és nemtől függetlenül mindig 3.3 és 6.1 között megfelelő.

Megfigyelhető, hogy egészséges tartományokat a testmagasság és a testsúly nem befolyásolja. Ugyan az orvostudományban már megállapították például azt, hogy a túlsúlyos emberek hajlamosabbak a magas vérnyomásra, de ez nem jelenti azt, hogy emiatt náluk a küszöbértékek egyaránt magasabbak

[15] https://www.aut.bme.hu/Pages/Research/SensorHUB

lennének. Így a rendszerünk a felhasználók számára a BMI (testömeg) indexük alapján csak azt jelzi, hogy a magasabb, vagy éppen az alacsonyabb értéket a nem megfelelő testsúlyuk is előidézheti.

Ezeket a tartományokat a szerveren egy XML állományban tároljuk, amelyből a mérések kiértékelése során automatikusan kerül kiválasztásra a megfelelő. Az alábbi példában a 21-40 év közötti nők pulzus és vérnyomásértékeinek tárolása látható. Az age attribútumban mindig a tartomány alsó határát tároljuk csak el, annak felső határát a következő bejegyzés jelenti.

```
<Pulse gender="woman" age="21" low="60.0" high="100.0" />
<BloodPressureSys gender="woman" age="21" low="123.0" high="133.0" />
<BloodPressureDia gender="woman" age="21" low="78.0" high="85.0" />
```

Miután egy mérési érték nem esik bele egyik mérési tartományba sem, úgy az nem tekinthető egészségesnek. Ekkor meghatározunk egy előjeles számot. Amennyiben egy mérési érték alacsonyabb, mint az egészséges tartomány, úgy egy negatív számot rendelünk hozzá a méréshez, és annak abszolút értékes nagyságát a küszöbértékektől való távolság határozza meg. Ha a mért érték magasabb volt a megengedettnél, úgy egy pozitív számot adunk vissza. Természetesen azt is jeleznünk kellett, ha egy mérési érték az egészséges tartományon belülre esik, ekkor a méréshez nullát rendelünk hozzá.

A meghatározott számértékeket ezt követően a szerver alkalmazás Push notification üzenetben küldi el a felhasználó okostelefonjára a Google Cloud Messaging (GCM) [**8**] felhasználásával. A számértékhez tartozó szöveges jelentés hozzárendeléséért már a kliens alkalmazás a felelős, mivel csak így oldható meg hatékonyan a többnyelvű támogatás.

A mérésekhez tartozó előjeles értékek nem kerülnek eltárolásra a szerveren. Amiatt döntöttünk így, mert ezek származtatott értékek, és könnyen meg lehet őket újra határozni. Ennek köszönhetően nem növeltük az adatbázisunkban a redundanciát.

4. Raspberry Pi

4.1. Hardveres architektúra

A mérések lebonyolítására egy Raspberry Pi mikrokontrollert használunk, egy Cooking Hacks nevű cég szenzorjaival. Ezek a szenzorok alapvetően Arduinohoz készültek, viszont egy konverter lapka segítségével a Pi-n is használhatóak. Az eszköz köré az Automatizálási és Alkalmazott Informatikai Tanszéken készült egy 3D nyomtatott doboz is, ezzel esztétikusabbá téve a kivitelezést. Ez látható a 12. ábrán.

A projekthez a Raspberry Pi 1 B típusú modellt használtuk, ami többek között az alábbiakat tartalmazza:

- 512 MB RAM
- 2 USB 2.0 port
- 100 Mbit/s Ethernet port
- 700 MHz ARM processzor

12. ábra. A Pi szenzorokkal a 3D nyomtatott dobozában

A Pi energia ellátása micro USB protról történik. A vetzetékes szenzorok energia ellátása általában a Pi-ről történik, de vannak kivételek például: a vérnyomásmérőnek saját tápellátása van.

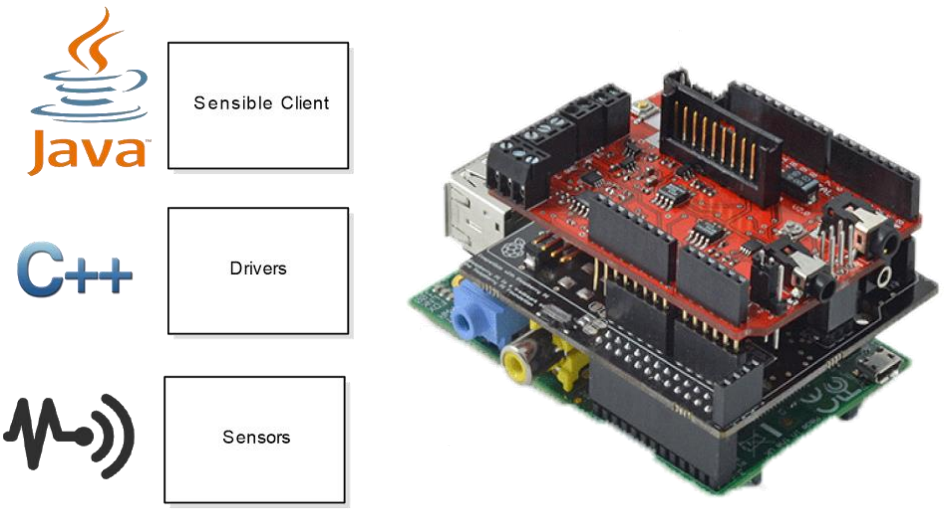
4.2. Szoftveres architektúra áttekintése

Az eszközre Raspbian[16] operációs rendszer van telepítve, ami a Debian Linux kifejezetten Raspberry Pi-re optimalizált változata, erre kellett tehát fejleszteni.

A szenzorokat C++ illesztő programok segítségével mintavételezzük, ezek kezelik a jelenleg analóg vagy digitális porton, vagy épp vezeték nélkül csatlakozó mérőeszközöket. Az illesztő programok

[16] https://www.raspbian.org/

végzik a szenzorral való kommunikációt és információt is szolgáltatnak róla. Ilyen információ lehet, hogy pontosan milyen típusú adatokat mér, vagy, hogy éppen csatlakoztatva van-e a rendszerhez. A Raspberry architekturális áttekintése az alábbi képen látható.



13. ábra. A Pi architektúrája

A Sensible kliens fogadja és szolgálja ki a hozzá bejövő kéréseket, de számos egyéb feladatot is végez. Itt történik a mérést végző fél hitelesítése a szerver segítségével és a méréshez szükséges illesztő programok menedzselése. A szerver által nyújtott segéd szolgáltatásokat is a kliens kezdeményezi, végül a mérés feltöltése is a feladatai közé tartozik.

A kliens és a szenzorok közti laza csatolásnak köszönhetően a pi kliens számára a szenzorral való kommunikáció transzparens, tehát könnyen lehet bővíteni a rendszert új szenzorokkal, amihez akár teljesen más technológiával kapcsolódik a mérő eszköz, anélkül, hogy a kliens kódját változtatni kéne.

4.3. Az illesztő programok

4.3.1. Az illesztő programok szerepe

Az illesztő program lehetővé teszi az szenzorok és a mérést végző eszköz közti lazább csatolást, így a rendszerünknek nem szükséges ismernie az összes kommunikációs formát, amivel felveheti a kapcsolatot a szenzorokkal és a speciális jelzéseket vagy pontatlanságokat sem kell ismernie az eszköz megfelelő használatához. Illesztő programot harmadik fél is fejleszthet ezzel is növelve a rendszer támogatottságát és bővíthetőségét.

4.3.2. Az elérhető szenzorok

A szenzorokkal való interakcióhoz drivereket használunk. Ezeket a rendszer induláskor betölti az erre dedikált mappából. Ezt követően elkéri tőlük az alapvetőbb információkat. Ilyen információ lehet, hogy milyen adatok mérésére képes a szenzor vagy, hogy éppen csatlakoztatva van-e. Ezek az információk a könnyű parse-olhatóság érdekében JSON formátumban vannak. Egy ilyen rekord például:

```
{"type":2,"connected":true,"details":[2,4]}
```

Jelen esetben ezt azt jelenti, hogy a kettes szenzor csatlakoztatva van, és mér 2-es és 4-es típusokat, azaz pulzust és vér oxigént. Ha valaki a rendszeren mérést szeretne indítani, akkor ezek alapján állítjuk össze a lehetséges mérési opciókat.

4.3.3. Az eredmények

Magát a mérést is a driver végzi, hiszen ez ismeri a legjobban a mérő hardver tulajdonságait. A bejövő érvényes mérési kéréseket egyszerűen továbbítjuk az illesztő program felé, ami biztosítja számunkra a mérések rész eredményeit és a végeredményt is. Egy ilyen rekord a következőképpen néz ki:

```
{"values":[{"type":2,"data":74},{"type":4,"data":88}]}
```

A mérések részeredményei a mérést kérelmezőnek továbbításra kerülnek egy socket kapcsolaton keresztül, hogy az esetlegesen hosszabb mérésekből hamarabb lásson eredményt a felhasználó.

Az illesztő program a mérés befejezését egy egyszerű flag-el tudja jelezni. A program ezt a rekordot kezeli a mérés végeredményeként, tehát ez kerül a szerver felé továbbításra. Ezt az adatot már nem küldjük el a kliens számára, hiszen ő ezt már az összes eddigi minta formájában megkapta. Egy ilyen rekord:

```
{"done":true,"values":[{"type":2,"data":72},{"type":4,"data":91}]}
```

4.3.4. Hiba

Előfordulhatnak hibák is a mérés közben például: a szenzorral megszakadt a kapcsolat. Egy ilyen hibát a drivernek kell kezelnie, de ha nem tudná megoldani a problémát, jelezheti azt a program számára. Ekkor a hiba továbbításra kerül a mérést indítónak, akit értesítünk a problémáról.

Az illesztő program egy ilyen üzenete lehet például:

```
{"error":"The sensor has been disconnected."}
```

4.4. A Sensible kliens főbb komponensei

4.4.1. Driver Manager

A Driver Manager az illesztő programokat kezelő funkcionális komponens. A program indításakor az erre kijelölt helyen levő drivereket végigkérdezi az állapotukról és listát vezet róluk, emellett ezek betöltése és futtatása is az ő feladata, amennyiben ezt egy másik komponens kéri tőle.

4.4.2. Discovery Service

A Discovery Service egy, a kliens által használt hálózati kommunikációs szolgáltatás. Ide érkeznek meg a hálózatról kapott felfedező csomagok. Ezt a szolgáltatás feldolgozza, és ha megfelelő formátumúnak találja, akkor válaszcsomagot állít elő a csomagban található IP címre.

4.4.3. Remote Parser

Ez a komponens az okos telefonnal végzett kommunikációs folyamatot vezérli. Az egyes bejövő kérések ide futnak be. Ezeket először értelmezi, majd ezután továbbítja a megfelelő végrehajtó komponensek.

4.4.4. Executer

A mérés végrehajtásáért felelős komponens. A Driver Manager által betöltött driver futását felügyeli és elvégzi vele a szükséges kommunikációt. Az illesztő program által mért értékekből részleges mérési értékeket képez és eldobja azokat a részeket, amit a felhasználó nem kért. A részleges mérési értékeket és az esteleges hibákat a felhasználó fele továbbítja, míg sikeres mérés esetén az adat feltöltést elindítja. Ha a rendszerben vannak úgynevezett kiegészítő szenzorok, amik önmagukban nem hordoznak egészségügyi információt például: pozíció, környezeti hőmérséklet azok is itt kerülnek hozzáadásra a feltöltéshez.

4.4.5. HTTP segéd komponens

A szerver irányába menő http kérések támogatására két segéd osztályt hoztunk létre. Ezek lehetőséget adnak kérések elemi szintű újrapróbálására hálózati hiba esetén adott időközönként adott darab számban, azonban az egyéb okból sikertelen kéréseket nem próbálja újra. A válaszokból válasz objektumot készít, ami tartalmazza a válasz kódot és az üzenetet vagy hiba üzenetet is, amennyiben volt ilyen.

4.5. Kommunikáció a mérést indítóval

4.5.1. A hitelesítés

A kommunikáció elején a felhasználó egy úgynevezett tokent kell, hogy küldjön, ami többek közt a mérés indításra való jogosultságát hivatott ellenőrizni. A tervezés során gondoltunk rá, hogy minimalizáljuk az ezzel való visszaélés lehetőségét, így a tokenek egyszer használatosak, tehát minden mérés után új példány szükséges belőle. Ennek a tartalmát csak a szerver ismeri, a mérő program nem tudja értelmezni. Ehelyett a szervertől kérdezi meg, hogy ez a token érvényes-e és feljogosít-e a rajta végzett mérés elvégzésére.

4.5.2. A mérés indítása

Sikeres hitelesítés után JSON formátumban elküldésre kerül a rendelkezésre álló szenzorokból és az általuk mért adatokból készített lista. Egy ilyen lista formátuma:

```
[{"type ":1, "details ":[1, 2]}, … {"type":n, "details":[d, k, x]}]
```

A felkínált listából választhat a felhasználó, hogy milyen paramétereket szeretne mérni. Ezt egy ugyan csak JSON formátumú válasz listában közli. Ez a lista a fentivel megegyező formátumú, azonban details paramétert nem kötelező specifikálni. Lehetséges egy szenzor által mért adatoknak csak egy részhalmazát mérni. Ekkor az adott szenzor details attribútumában lehet megadni a választott értékeket, ha ez nincs specifikálva vagy üres, abban az esetben az összes mért értéket használjuk.

4.5.3.A részeredmények

A mérés során keletkeznek részeredmények is. Ezeknek a célja, hogy az esetlegesen hosszú mérési és értékelési folyamat során korábbi előrejelzést adjunk a felhasználónak az eredményéről. A részeredményeket az illesztőprogram generálhatja és az Executer továbbítja a felhasználó felé a socket kapcsolaton. Egy ilyen részeredmény állhat egy vagy több mintavételből és típusból is, hogy a gyakran mintavételező szenzorok ne feltétlen küldjenek indokoltatlanul sok csomagot a hálózaton. Ezzel csökkentve a TCP fejléc által jelentett overhead méretét is a kommunikáció során. Egy ilyen rekord a következőképpen nézhet ki:

```
[{"type":9,"data":[241]}, {"type":7,"data":[410, 423, 457, 508]}]
```

5. Android kliens

Ebben a fejezetben az Android kliens alapvető céljainak is lehetőségeinek bemutatása olvasható. Tartalmazza a mobil alkalmazás architektúráját, felépítését, főbb komponenseit. Rövid ismertetést ad az elérhető funkciókról, valamint a felhasználói program mérési folyamatban betöltött szerepéről.

5.1. Kliens célja

A Sensible rendszerben a mérések vezérlése és az eredmények megjelenítése az Android kliens feladata.

Egy mérési folyamat elkezdéséhez szükséges az Android applikáció megléte. A mérések csak érvényes regisztrációt követően végezhetők el, bejelentkezett állapotban. A regisztráció során a bejelentkezéshez tartozó adatok megadása mellett, további személyes, egészségügyi adatok is bekérésre kerülnek. Az egészségügyi adatok megléte a vizsgálatok kiértékelésében játszik fontos szerepet.

Egy vizsgálat megkezdéséhez az alkalmazással egy NFC tag [**9**] felolvasásával lekérdezhető egy mérőállomás azonosítója. Ezt az azonosítót a szerver kezeli, majd értesítést küld a mérőeszközt vezérlő Rasberry Pi eszköznek és a kliensnek is. Az így végzett kommunikáció részletezése a szerver, valamint a Rasberry Pi dokumentációjának képezi részét. Az Androidos eszköz egy szerver által visszaadott IP cím és egy ellenőrző token segítségével kapcsolódhat a mérőberendezéshez. Ezt követően a mérőállomás egy lista formájában kínálja fel a kiválasztható mérőeszközöket. Itt a felhasználó kiválaszthatja a kívánt eszközt, ezzel elindítva a mérést. A mérésre kiválasztható szenzorok listája az állomáson megtalálható aktív eszközökből áll elő. Az összetett szenzorok esetében lehetséges csak bizonyos részeredmények mérése is.

A mérés befejezését követően az eredmények tárolásra kerülnek a szerveren, majd a kliens is értesül az új adatokról. Itt válik fontossá a felhasználói program másik fő funkciója, a megjelenítés. Mérési folyamat közben a folytonos jellegű adatok esetében azonnali részeredmények is érkeznek a klienshez.

Lehetőség nyílik a friss adatok mellett, a korábbi mért eredmények megtekintésére is. A kapott eredményekhez értesítés is tartozik, ami az esetlegesen felmerülő egészségügyi problémákról, veszélyekről ad tájékoztatást. Minden mérési eredményhez tartozik részletező képernyő. Ebben a nézetben megtekinthetők az esetleges figyelmeztetéshez tartozó információk. A mérés típusától függően (folytonos értékű vizsgálatok) lehetőség nyílik az eredmények diagramon való megjelenítésére is. Ezáltal a kapott adatok szemléletesebben is megjelennek.

5.2. Alkalmazás felépítése

Ebben a fejezetben az alkalmazás struktúrájának ismertetése olvasható. Tartalmazza az alapvető architektúrális elemek bemutatását.

A felhasználói program több, meghatározó és a működésben fontos szerepet játszó külső osztálykönyvtár beépítésével készült. Ezekről a fejezetben bővebb leírás is található. Ennek oka, hogy több olyan könyvtár is található köztük, amely nem a megszokott Android viselkedést valósítják meg. Eltérnek a napjainkban használatos megoldásoktól és módszerektől.

5.2.1. Felhasznált könyvtárak bemutatása

5.2.1.1. Flow & Mortar

A Square fejlesztői csapat által létrehozott, nyílt forrású, jelenleg is fejlesztés alatt álló rendszere[17]. A készítése mögötti motivációt az Android fragmensekkel való bővítésekor keletkezett, megnövekedett életciklus kezeléssel kapcsolatos bonyodalmak és problémák jelentették. Célja, hogy az alkalmazás komponensei egy Activityben kezelhetőek legyenek, fragmensek felhasználása nélkül is. A különböző elemeket egyszerű view komponensek megvalósításával és kombinálásával valósítja meg. A felépítés az MVP architektúrális mintát követi, a szerepkörök jól elkülönülnek.

A Flow elsődleges feladata, hogy a képernyőhöz tartozó elemeket kezelje. Minden egységet úgynevezett screennek tekint. A screenek egyszerű Java osztályok. Fontos részét kepezik a Flow által szolgáltatott annotációk, valamint a Blueprint interfész, ami a Mortar megvalósítást hivatott kiszolgálni. Ehhez kapcsolódik a Dagger által szolgáltatott dependency injection megoldás is. Ennek segítségével az alábbi módon valósul meg a könyvtár számára a screen használata. injektálási módszer.

```
@dagger.Module(injects = HomeView.class, addsTo = Main.Module.class)
class Module {
}
```

Az annotációban szereplő injects paraméter a screenhez kapcsoló View osztály számára jelzi az injektálás szükségességét. Az addsTo paraméter a Mortar bejegyzésért felelős. A következő kódrészletben a Viewban szereplő @Inject annotációval ellátott referencia látható, ami a screen presenteréhez biztosít hozzáférést.

```
@Inject
HomeScreen.Presenter presenter;
```

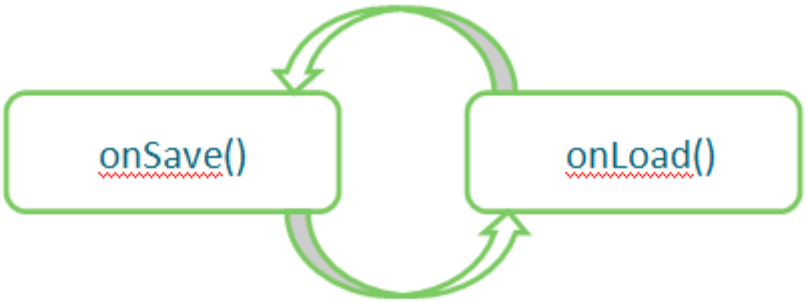
A presenter felelős az adatok kezeléséért, valamint a hozzá kapcsolódó view-kon keresztül az eseményekhez tartozó metódusok végrehajtásáért.

```
@Singleton
public static class Presenter extends ViewPresenter<HomeView>
```

Fontos megjegyezni, hogy az állapot megőrzése miatt egy singleton objektumnak kell lennie a presenternek. Így a külső változásokra való reagálás, mint például az orientáció megváltozása nem befolyásolja a komponens állapotát, adatait. Amikor pedig a presenter kikerül a megjelenített elemek halmazából, képes elmenteni az állapotát, majd később visszaállítani a saját Bundle elemét felhasználva.

Így a komponenshez tartozó életciklus modell a működés közben az alábbi képen látható módon valósul meg.

[17] https://github.com/square/flow
https://github.com/square/mortar



14. ábra. Komponens életciklus modell

Ezzel a nem túl összetett megoldással a fragmensek kezelését teljes mértékben kihagyva valósul meg az alkalmazás komponenseinek viselkedése.

5.2.1.2. Retrofit

Egyszerű segédkönyvtár REST kommunikáció és szolgáltatások használatához [18]. A REST szolgáltatásokat Java interfészeken keresztül valósítja meg.

A hívási típusokat annotációkon keresztül (@GET, @POST…) biztosítja. A metódus megfelelő annotálásához tartozik a metódusok paramétereinek megjelölése is. Itt megemlíthető egy kérés @Path jelzéssel való ellátása, ami a hivatkozott elérési út egy részét befolyásolja. Formok esetében a @Field, egy POST kérésnél pedig a @Body, mint a kérés body része használható. További megoldások a dokumentációjában megtalálható módon valósíthatók meg.

5.2.1.3. Picasso

Egyszerű és gyors képmegjelenítő eszköz. A képek letöltésére, módosítására, megjelenítésére is használható[19].

Lehetőséget biztosít képek aszinkron módú letöltésére, majd annak megjelenítésére. A kezelt képekhez transzformációs opciókat is biztosít. Automatikus memória és cache kezelést valósít meg. Támogatja a placeholderek és a hiba esetén megjelenő képek hozzáadását is.

5.3. Architektúra ismertetése

Az alkalmazás Android Studio fejlesztőkörnyezetben készült. Ennek megfelelően a szükséges Android konfigurációk gradle állományokban definiáltak.

Az applikációhoz szükséges minimális követelmény az API szint SDK 16 (Jelly Bean) verziójában van meghatározva. A target SDK szintje az API 21 támogatást jelöli meg (Lollipop). Ezek alapján lehetőség van az új, korszerű Lollipop komponensek felhasználása, ugyanakkor támogatást is biztosít a régebbi készülékek számára. Ezáltal a Support Library-k segítségével biztosított a régebbi operációs rendszerű eszközök támogatása is Android 4.1-ig visszamenőleg.

A fejezet további részében az alkalmazás főbb osztályainak részletezése olvasható. Ezek az osztályok jelentik az alkalmazás alapvető részeit.

[18] https://github.com/square/retrofit
[19] https://github.com/square/picasso

5.3.1. SensibleAndroidBaseActivity

Az alkalmazás egyetlen activityből áll. Ez az abstract osztály a SensibleAndroidBaseActivity, ami az ActionBarActivity activityből származik. Ebben az osztályban szerepel a Flow inicializálása, valamint a Mortarhoz szükséges előkészületek implementációja is. Az alábbi kódrészletben látható az onCreate() függvény törzsének részlete, ami beállítja a Mortar működéséhez szükséges konfigurációt.

```
MortarScope parentScope = Mortar.getScope(getApplication());
activityScope = Mortar.
     requireActivityScope(parentScope, getBlueprint());
Mortar.inject(this, this);

activityScope.onCreate(savedInstanceState);
```

Ezután következik a layout beállítása, ami alapján a flow kezdésnél értelmezett view tartalma is inicializálásra kerül.

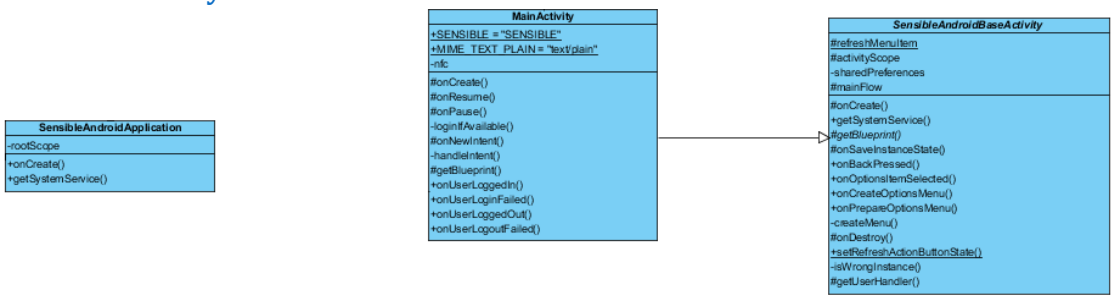
```
setContentView(R.layout.activity_main);

MainView mainView = (MainView) findViewById(R.id.container);
mainFlow = mainView.getFlow();
```

Ebben az osztályban található még az ActionBarhoz tartozó konfiguráció is. Mivel az alkalmazás összesen egy activityből épül fel, a beállítások itt elvégzett egyszeri megadása az alkalmazásra vonatkozó globális tulajdonságait jelenti.

Az alkalmazáson belüli navigációért a Flow felelős. A különböző view-k közti váltások ebből kifolyólag felüldefiniálják az alapértelmezett működést. Az alábbi kódrészleten a visszalépés eseményének felüldefiniálása látható.

```
/** Inform the view about back events. */
@Override public void onBackPressed() {
     // A visszalepes a Flow-ra van bízva. Amennyiben nem lehetseges a muvelet,
     // az eredeti, beepitett kezelese mod lep eletbe.
     if (!mainFlow.goBack()) super.onBackPressed();
}
```

5.3.2. MainActivity



15. ábra. Az Activity elhelyezkedése a modellben

Az SensibleAndroidBaseActivity leszármazottja, annak megvalósítója. Implementálja az NFC kezeléséhez szükséges Handler vezérlését, valamint megvalósítja UserHandlerListener interfészt. Ez az interfész tartalmazza a felhasználó sikeres, illetve sikertelen be- valamint kijelentkezéséhez szükséges eseménykezelő függvényeket. A fenti ábrán látható az alkalmazás Activity főbb függvényeivel és tagváltozóival.

5.3.3. HomeView

Az elsődleges, összetett nézet. Ez jelenik meg az alkalmazás megnyitásakor. Tartalmazza a bejelentkezéshez szükséges, illetve a bejelentkezés utáni állapothoz kapcsolódó kezdeti elemeket. Az osztály egy RelativeLayout leszármazott. Az osztályhoz tartozó XML állomány gyökéreleme az így létrehozott osztály. A felület részei ebben definiáltak. [**10**]

Az osztály tartalmazza a HomeScreen által szolgáltatott presenter referenciát, aminek keresztül képes a komponens másik részével kommunikálni. Ez a Flow & Mortar ismertetésének megfelelően injektálva van az osztályba:

```
@Inject
HomeScreen.Presenter presenter;
```

Az osztály kezeli a felületen megjelenő elemeket. Az ezekhez az elemekhez tartozó műveletek elvégzését már a presenterben megvalósított viselkedés hivatott véghezvinni. Látható a szerepkörök jól elhatárolt megjelenése. Az alábbi kódrészlet egy példával szemlélteti az említett működést.

```
@Override
public void onClick(View view) {
   if (view.equals(layoutRegistration)) {
      presenter.showRegisterDialog();
   }
   …
}
```

Egy érintés esemény kezelése olvasható a kódban. Amennyiben az aktivált elem a regisztrációs elemhez tartozik, az eseményhez tartozó metódus hívása megtörténik a presenter objektumon keresztül.

A Flow működéséhez további két függvény kiemelése lényeges. Az onFinishInflate(), valamint az onDetachedFromWindow(). Ezekben szerepel a presenter számára a láthatóság és annak megszűntetésére szolgáló hívások. A felület betöltését (inflate) követően szerepel a megjelenítésért felelős értesítés. A detached állapot hívásában pedig a leválasztáshoz szükséges hívás:

```
// felcsatolas
presenter.takeView(this);
// levalasztas
presenter.dropView(this);
```

Ebből is jól látható, hogy a View számára teljesen transzparens módon történik a megjelenés és az eltűnés.

5.3.4. HomeScreen

A HomeViewhoz tartozó, Flow & Mortar komponens implementációja. Megvalósítja az alapvető működést és a szükséges funkciókat, amikkel az alkalmazás részévé válhat. Itt található meg a HomeView által használt presenter implementációja is.

```
@Layout(R.layout.screen_home)
public class HomeScreen implements Blueprint {…}
```

Az osztály definíciójánál található Layout annotáció a Flow által használt jelölés, ami az osztályhoz tatozó screen_home felületet rendeli a screenhez. A BLueprint interfész implementálása teszi

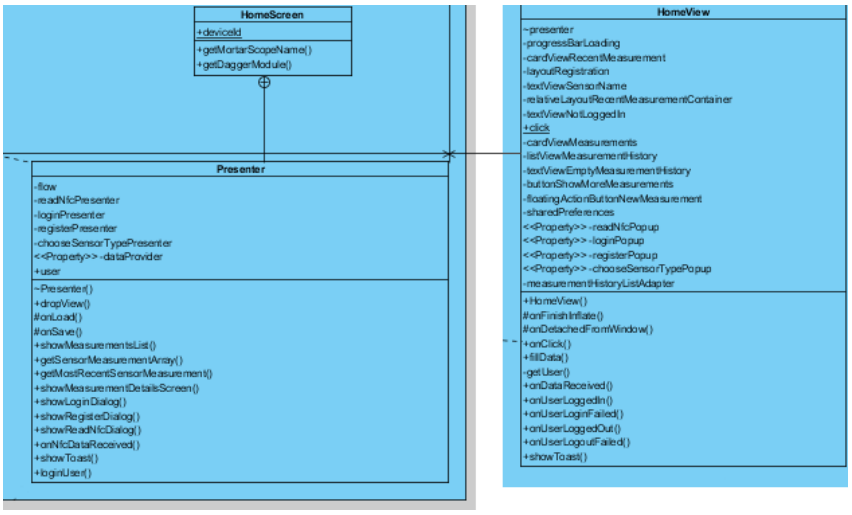
lehetővé, hogy a presenter objektumhoz tartozó állapot elmenthető, valamint visszaállítható legyen az alkalmazáson belül, specifikusan ehhez a komponenshez tartozóan.

Egy másik fontos részlet a Dagger segítségével végzett injektálás. Az injects jelzés lehetővé teszi, hogy a futás során a HomeView-ban bemutatott presenter referencia elérje a megfelelő objektumot, az addsTo paraméter pedig a Mortar bejegyzést végzi el.

```
@dagger.Module(injects = HomeView.class, addsTo = Main.Module.class)
class Module {
}
```

A presenter implementációjánál fontos megemlíteni a @Singleton annotációt. Ezzel érhető el az állapot megmaradása konfigurációváltozás mellett is. A HomeScreen presentere egy összetett vezérlő, melyben a főképernyőhöz tartozó további vezérlők laknak. A ViewPresenter<HomeView> ősosztályból származó presenter implementálja az onSave() és onLoad() függvényeket. Ezeken keresztül valósítható meg az állapot bundle objektumba való mentése, valamint az állapot visszaállítása. Ezek jelentik a teljes életciklus kezelését is.

A HomeScreen.Presenter implementálja a View által, presentereken hívott függvényeket. Ezek vagy egy-egy művelet végrehajtását, vagy az általa tartalmazott gyerekelem presenterének megfelelő műveletét hívják meg. Az alábbi kódrészleten látható a bejelentkezésnél használt felugró ablak megjelenítéséhez használt presenter hívás, valamint a tartalmazott elem presenterének hívása egy új, üres user objektummal. A Screen, a hozzá tartozó presenter és a HomeView főbb komponensei láthatóak az alábbi ábrán.



16. ábra. HomeScreen és HomeView a modellben

```
public void showLoginDialog() {
   loginPresenter.show(new User());
}
```

5.4. Kommunikáció a rendszer további részeivel

A rendszer további komponenseivel történő kommunikáció bővebb ismertetése a kapcsolódó szerver és Rasberry Pi dokumentációban olvasható. Ebben a fejezetben az Android oldali megoldások bővebb kifejtése található. Az első részben a szerverrel való kapcsolat kifejtése, majd a mérőberendezés irányába történő adattovábbítás szerepel.

5.4.1. Kommunikáció a szerverrel

A szerverrel kapcsolatos kommunikáció során REST hívásokon keresztül történik az adatcsere. Ehhez a folyamathoz a RetroFit könyvtár megoldásai implementáltak. A következőkben a kommunikációt vezérlő osztályok részletezése következik.

5.4.1.1. SensibleService

Ebben az interfészben szerepelnek a definiált REST kommunikációs hívások. A hívási interfész függvényei a RetroFit annotációival vannak ellátva. Például a bejelentkezéshez használható hívás:

```
@POST("/auth/login")
public void login(
   @Body UserBase user,
   Callback<Response> callback
);
```

Látható a POST típusú hívás, melyet az útvonal (path) meghatározása követ. A függvényben @Body annotációval szerepel a kérés törzse, jelen esetben egy felhasználó adatmodellje. Minden hívás utolsó paramétere egy Callbackre hivatkozik, ami a válasz feldolgozásáért felelős. A metódus egy hívására az alábbi kódrészlet ad példát:

```
public void loginUser(User user) {
   SensibleService service = SensibleClient.provideSensibleService();

   service.login(user, new LoginCallback(INSTANCE, user));
}
```

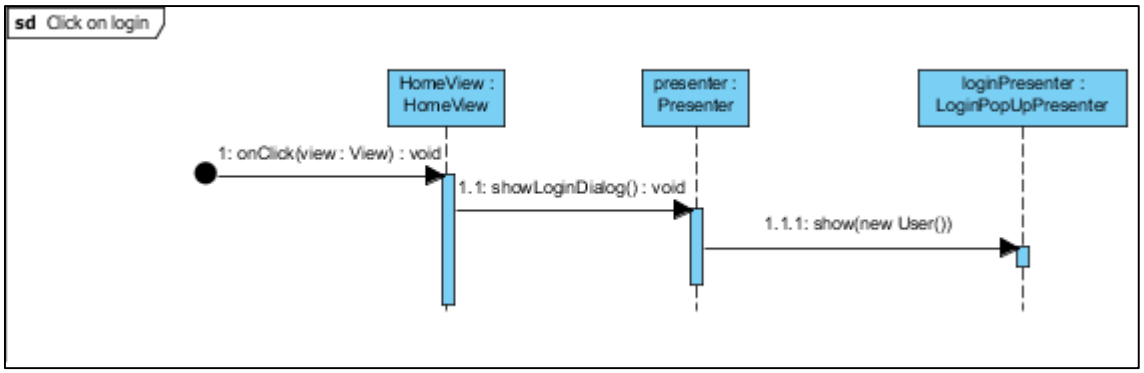
A válasz megérkezésekor egy LoginCallback objektum feladata a kapott eredmények feldolgozása.

5.4.1.2. LoginCallback

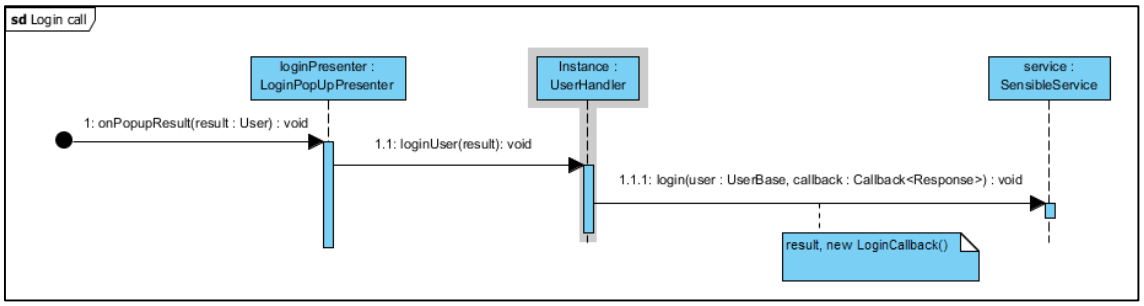
A Callback implementálja a RetroFit által szolgáltatott Callback<Response> interfészt. Az implementálandó függvények a sikeres és a sikertelen válaszok feldolgozására szolgálnak.

```
@Override
public void success(Response response, Response response2) {
   …
   userHandler.onLoginSuccess(user);
}

@Override
public void failure(RetrofitError error) {
   …
   userHandler.onLoginFailed();
   …
}
```

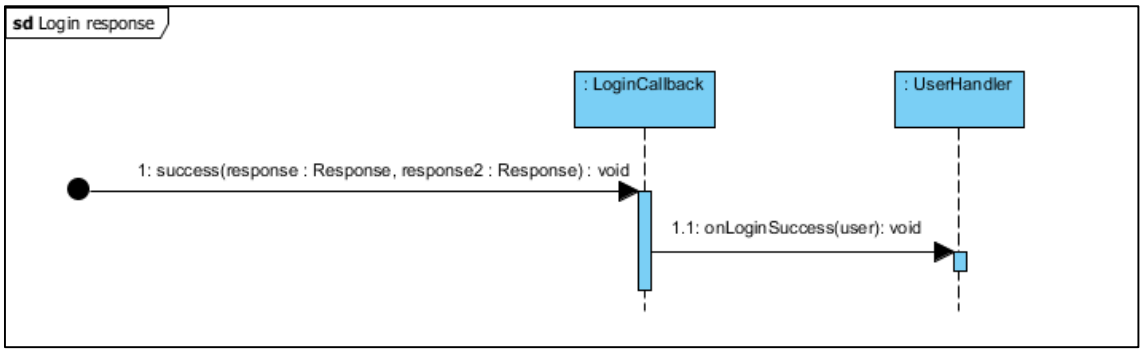
A sikeres, illetve sikertelen válaszok esetén a válaszból kiolvasható a kapott adat vagy hibaüzenet. Ebben az esetben a bejelentkezés sikeressége, vagy sikertelensége mellett a felhasználó állapotát kezelő UserHandler objektum megfelelő metódusa hívódik meg.



17. ábra. Login ablak megnyitása



18. ábra. REST hívás indítása a bejelentkezési adatokkal



19. ábra. REST hívás sikeres válaszának feldolgozása, bejelentkezés

5.4.2. Kommunikáció a Raspberry Pi eszközzel

A szervertől kapott token és IP cím párossal történik a Pi felé továbbított információátvitel.

5.4.2.1. SensiblePiClient

Egy külön szál végzi a Socket alapú kommunikációt. A socket egy általános Java Socket objektum. Az írással történik a Rasberry felé a kommunikáció a szerverrel kapott token és az NFC taggel leolvasott szenzorazonosító alapján. A beolvasás az egyszerű Java által használt stream alapú módon, soronként olvassak be a kapott adatokat. Az így kapott adathalmaz String formában kerül a MeasurementParserhez, ami az adatok feldolgozását fogja elvégezni. A létrehozott kapcsolat után az adatok feldolgozásának hívása következik. Itt a mérőállomás által visszaadott szenzorok listája készül el, amely alapján a felugró ablakban a felhasználó már meghatározhatja a használni kívánt eszközt.

A kiválasztott mérés elindítását követően az előző megoldással megegyező módon, soronként érkezhetnek a részeredmények és zajlik tovább a kommunikáció.

6. SensorHUB

A SensorHUB az Automatizálási és Alkalmazott Informatikai Tanszék saját fejlesztésű IoT platformja. Ez egy olyan elosztott rendszer, melyet a Hadoop menedzsel. A felhasználók mérési adatai közötti összefüggés vizsgálatára használjuk. A következő fejezetekben bemutatjuk a SensorHUB azon részeit, amiket a rendszerünk használ, illetve részletesen írunk a korreláció keresés megvalósításáról.

6.1. A SensorHUB fájl struktúrája

A SensorHub-ra feltöltött adatok először a HDFS[20]-re kerülnek fel, ami az Apache Hadoop elosztott fájlrendszere. Itt egy speciális mappa struktúrában tárolódnak. A projekt gyökér mappáján belül külön mappa van az éveknek, azokon belül a hónapoknak és így tovább egészen az órákig. Ha egy adat jön be, akkor a megfelelő mappában egy fájl kerül létrehozásra számára, ami kezdetben tmp, azaz ideiglenes kiterjesztésű. A további feltöltések egy ideig ebbe szintén a fájlba kerülnek, de idővel a fájl végleges lesz és az új kérésének ismét új fájl kerül létrehozásra.

6.2. Külső és belső táblák

Az adatbázisban tárolásnál két lehetőségünk volt a SensorHub-on. A külső táblák a fájlrendszerben létező fájlok sorait tünteti fel adatbázis rekordokként. Ezeken a rekordokon lehet aztán különböző kiválasztásokat végezni. Ezt a táblát azonban nem tudtuk közvetlenül a feltöltött adatokon használni több okból kifolyólag:

- Az externál táblát bár mappából létre lehet hozni, de ez ilyen több szintű mappa rendszeren nem működik.
- A feltöltött rekordok között több különböző adat típus van, mint például profil adat és testhőmérséklet. Ezeknek az adat struktúrái nem kompatibilisek egymással így külön táblában kell őket tárolni.
- A kommunikáció JSON formátumban történik, így ez kerül a fájlokba is, ezekből azonban nem hozható létre externál tábla.

Az internál tábla mögött ezzel szemben fizikailag létező adatbázis van, nem csak leképezésről van szó. Ennek használata esetén az adat egy fix struktúrában tárolódik soronként a műveleti memóriában, ezzel gyorsítva az ahhoz való hozzáférés sebességét. A műveleti memória viszont véges, úgyhogy óvatosan kell, hogy bánni az internál táblákban tárolt adatok mennyiségével.

6.3. Beszúrás az internál táblába

Az adatokat alapvetően két féleképpen lehetséges az internál táblába beszúrni. Egyszerre az externál táblából való select-el vagy egyenként JDBC[21]-n (Java Database Connector) keresztül. Mind a két lehetőséget megvizsgáltuk.

JDBC használata esetén minden SQL utasításhoz külön feladat készül a rendszerben Mivel egy felhasználó egyszerre csak egy feladatot futtathat a rendszerben és a program futtatásához is kellett egy, így az ezzel való próbálkozás dead lock-hoz vezetett. A problémát megpróbáltuk egy második account használatával orvosolni, így külön accounton futott a program és a program által kezdeményezett adatbázis művelet. A várakozásoknak megfelelően így már lefutott a program, azonban futási sebessége messze elmaradt a várakozásainktól. Mivel külön taszkként futott minden adatbázis művelet, ezért külön futási környezet készült számára külön naplókkal és várnia kellett a

[20] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[21] http://www.oracle.com/technetwork/java/javase/jdbc/index.html

műveletnek a futási jogra is. Ez akkora overheadet okozott, hogy egy rekord beszúrása kb. 8-10 másodpercet igényelt így óránként csak 360 rekordot tudnánk beszúrni az adatbázisba, ha folyamatosan szúrnánk őket be az adatbázisba. Nem gondoltuk, hogy ez alkalmas big data jellegű műveletekre.

Az externál táblába való beszúráshoz egy speciális sql scriptet használtunk, aminek segítségével lehetséges az externál táblában levő JSON objektumot közvetlenül beszúrni az internál táblába, így a fájlokat csak alap szinten kéne, hogy feldolgozzuk. A script a következőképpen nézett ki:

```
insert into sensible_profiles_internal
SELECT
    cast(spi.born as int) as born,
    cast(spi.weight as int) as weight,
    cast(spi.profileID as string) as profileID,
    cast(spi.height as int) as height,
    cast(spi.gender as int) as gender
FROM sensible_profiles_external spe
  LATERAL VIEW json_tuple(
   spe.value, 'born', 'weight', 'profileID', 'height', 'gender') spi
AS born,
 weight,
 profileID,
 height,
 gender;
```

Bár ennek a futása gyorsabbnak bizonyult a JDBC használatánál továbbra is lassú volt az adat beszúrás. Emellett az internál tábla egyre több műveleti memóriát használt a SensorHub szerverein, így végül az externál tábla használata mellett döntöttünk.

6.4. Beszúrás az externál táblákba

Az externál tábla tulajdonképpen a fájlrendszer egy részének a leképezése adatbázis táblára, ahogy azt a külső és belső táblák című fejezetben is olvasni lehet. Ebből következően, ha a fájlok tartalma változik, akkor az adatbázis tartalma is ugyanúgy változik. Az externál táblába való beszúráshoz tehát a leképezett mappába kell az adatokat beszúrnunk. Ez azonban más volt, mint amikor csak közbülső lépésként csináltuk ezt, mert akkor az sql lekérdezés végezte a JSON objektum parse-olását, ekkor azonban a táblánknak csak egy oszlopa volt, amiben szövegként szerepelt az objektum, nem volt alkalmas arra, hogy komplex szűréseket vagy lekérdezéseket végezzünk rajta.

A most létrehozandó tábla már több oszloppal is rendelkeznie kell, amik típusosak is, ezzel jó alapot biztosítva a hatékony korreláció kereséshez. Ehhez a fájlokban található JSON objektumot CSV (Comma Separated Values) formátumúra alakítjuk, ami alkalmas a fenti követelményeknek megfelelő leképezésre. Egy ilyen tábla létrehozására a lenti sql scriptet használtuk.

```
CREATE EXTERNAL TABLE sensible_profile_csv (
     born int,
     weight int,
     profileid string,
     height int,
     gender int
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION
'mappa';
```

Ez a megvalósítás már lehetővé teszi a gyors beszúrást és nem is használ sok műveleti memóriát a SensorHub szerverein.

6.5. Korreláció keresés

A statisztikában a korreláció két tetszőleges érték közti lineáris kapcsolat nagyságát és irányát jelzi. A nulla azt jelenti, hogy a két érték között biztosan nincs lineáris összefüggés. Negatív értékek fordított arányosságot jelentenek, míg a pozitívak egyenes arányosságot. A korreláció csak lineáris kapcsolat kimutatására alkalmas, így például egy szám és annak a négyzete közti összefüggésre nem.

A két változó közti korreláció keresés eredményét az úgynevezett parciális korreláció is befolyásolhatja, amikor egy harmadik, az előbbi kettővel valószínűleg kapcsolatban álló adat is van. Ekkor lehetséges a harmadik érték hatását kiszűrni, ez azonban komolyan megnövelheti a feladat komplexitását, ha sok értéktípust vizsgálunk.

6.5.1. A Pearson korrelációs koefficiens

A korreláció keresőnk a Pearson féle korrelációs koefficiens [**11**], [**12**] mintájára lett kialakítva, ami a két értékpár közötti lineáris kapcsolatot egy -1 és 1 közötti számmal jellemzi, ahol a nullához közeli eredmények statisztikailag elhanyagolhatónak tekinthetőek, míg a nullától távoliak erős lineáris kapcsolatra utalnak. Egy mintából az alábbi képlet felhasználásával számíthatjuk ki ezt az értéket.

$$r = r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

1. egyenlet. Pearson-féle korrelációs koefficiens

Megfigyelhető, hogy a képletben ismerni kell az általagot, tehát az adathalmazon több, mint egyszer végig kéne menni a képlet felhasználásához, valamint az, hogy a számokból rögtön kivonjuk, az átlagukat nagyban megnöveli a számításhoz szükséges műveletek számát lépésenként nézve. Ezen problémák miatt a Pearson képlet egy átrendezett formáját használjuk a programban. Az átrendezett képlet alább látható.

$$r = r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

2. egyenlet. Átrendezett Pearson képlet

A képlet megfigyelésével látható, hogy nincs többé szükség az átlag ismeretére a számításhoz és műveletek jelentős része elvégezhető az összegzés után, így nagy adat sorok esetén is csak egy lépésre van szükség a teljes számítás lefutása során.

Fontos továbbá, hogy mivel a számításhoz nem szükséges a teljes adathalmaz átlaga, így hatékonyan elvégezhető a számítás elosztott rendszerben, olyan módon, hogy minden gép elvégzi az elemek kiszámítását az adatok egy részén, majd ezekből a részekből lineáris időben elő lehet állítani a teljes adathalmazra számított értéket. Ez a megoldás többszálú számításra is lehetőséget ad.

Az implementáció előnye, hogy a parciális korrelációt [**13**] figyelmen kívül hagyva a számításhoz szükséges műveletek száma lineárisan függ az adatok mennyiségétől a memória igénye pedig

konstans, amennyiben feltételezzük, hogy az összegzés eredménye során előálló számok konstans memóriában elférnek, így tehát egy meglehetősen hatékony algoritmusról beszélhetünk.

A megoldás hátránya, hogy csak lineáris összefüggést tudunk vele kimutatni, pedig könnyen elképzelhető egyéb kapcsolat is az egészségügyi adatok között, így csak korlátozott megoldást nyújt a megoldandó feladatra.

6.5.2. Az összefüggő adatpárok

Az előbbi fejezetekben beszéltünk arról, hogy a korreláció keresést összefüggő adatpárokon kell elvégezni. A mi esetünkre levetítve ezt azt jelenti, hogy olyan adat párokat keresünk, ahol ugyanattól a felhasználótól származik mindkét mérés, valamint időben közel is vannak egymáshoz, ellenkező esetben az értékpár nem feltétlenül összefüggő. Az adatbázisunkban azonban nem ilyen értékpárokban vannak letárolva az adatok, így első feladatunk az össze tartózó értékpárok előállítása volt, amiket majd használhatunk az algoritmus bemeneteként.

Ennek az összerendelésnek alapvetően három féle típusa van. Amennyiben az egyik fél egy profil adat, úgy a hozzárendelés egyértelmű, hiszen a profil adatok historikusan vannak tárolva, így minden méréshez a mérés időpontjában aktuális profiladat van hozzárendelve.

Ha a két adat eltérő típusú (pl.: sima – folytonos) különböző táblákban vannak tárolva. Ekkor egyszerűen az egyik táblához hozzá illesszük a másikat a fentebb említett feltételekkel. Egy vetítés alkalmazása után a lekérdezés eredményét már fel is lehet használni a számításhoz.

Abban az esetben, ha azonos típusú adatokat szeretnénk összehasonlítani (pl.: folytonost – folytonossal) akkor ezek ugyan abban a táblában találhatóak, tehát a táblát önmagára illesszük a megfelelő feltételekkel. Ekkor a sima illesztéssel szemben figyelni kell arra is, hogy ne jelenjenek meg duplikálva a rekordok az eredményben. Ezt a problémát az alábbihoz hasonló feltétellel szoktuk megoldani.

```
a.type > b.type
```

6.5.3. Korreláció keresés folytonos adatokon

A fenti képlet jól használható két érték közti lineáris összefüggés keresésre, azonban folytonos adataink is vannak, amik adat tömb formájában vannak reprezentálva. Ezekre az értékekre nincs definiálva a korreláció keres. Erre a problémára keresünk megoldást ebben a fejezetben.

Ha csak szimpla értékeket tudunk a számításban felhasználni logikus lépésnek tűnik a folytonos adatok felbontása egyszerű értékekre. Az egyes mintavételek azonban önmagukban nem hordoznak információt, így a szétszedésre egy más módszerre álltunk elő.

Hullámforma analízis segítségével ki lehet nyerni a folytonos adatok különböző jellemző paramétereit. Ilyen például a frekvencia, a variancia és az átlag. Az így leképzett adat (pl.: a szív ritmus variancia) már hordoz az egészségügyi állapotra jellemző információt így van értelme a rajta végzett korreláció keresésnek, továbbá a használt képlettel is kompatibilisek, mivel már nem egy adat tömbről, hanem csak egy értékről beszélünk.

6.5.4. Megbízhatósági súlyozás

Az adatokat világszerte több forrásból, különböző szenzoroktól kaphatjuk így biztosak lehetünk benne, hogy előfordulnak hibás és pontatlan adatok. Ezek az adatok, ha megfelelő számban vannak, elfedhetik a keresett összefüggéseket.

Ennek a problémának a korai orvoslására találtuk ki a megbízhatósági súlyozást. Az egyes mérések értékelésekor a szerver észreveszi, ha egy érték nem reális és az adott mérő eszközről számon tartja a hibás mérések arányát.

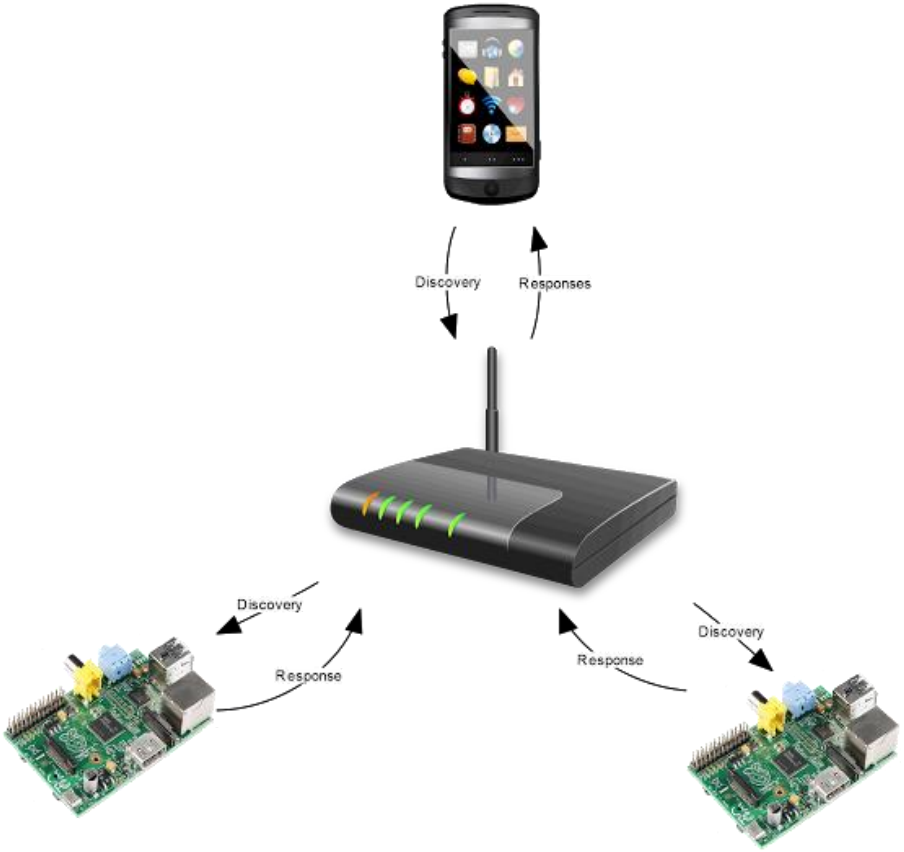
A korreláció keresés segítése céljából az eszközök megbízhatósági arányát is továbbítja a SensorHUB felé, így az ott végzett műveleteknél lehetőség van az adott eszközön mért adatok súlyozására, vagy az alacsonyabb megbízhatósági rátájú eszközök kizárására is a vizsgálatból.

7. Kommunikáció

A mérés indításához először csatlakozni kell az eszközre, ez alapvetően három különböző módon lehetséges. Ezekről az alábbi alfejezetekben írunk részletesen.

7.1. Azonos alhálózat

Az okost telefonnal lehetőség van alhálózati broadcast [**14**] üzenetet küldeni, ezt ábrázolja az alábbi kép is. Amennyiben az adott alhálózaton levő mérő eszközök ezt a csomagot fogadják, akkor válaszolnak neki. A válasz üzenetből kiderül a készülék IP címe és az eszköz azonosítója is, így az okos telefonnak minden csatlakozáshoz szükséges információja rendelkezésre áll.



20. ábra. Alhálózati felderítés

Ennek a megoldásnak előnye, hogy egyszerű bekonfigurálni, hiszen nincs szükség port nyitására. Az eszköz azonosító leolvasásához nincs szükség NFC olvasóra, bármilyen WLAN kapcsolattal rendelkező készülék hozzájuthat ehhez. A két eszköz közötti kommunikáció szempontjából is előnyös ez a konstrukció, hiszen a lehető legkevesebbet kell, hogy utazzanak a hálózatban a csomagok.

7.2. Nyilvános IP cím

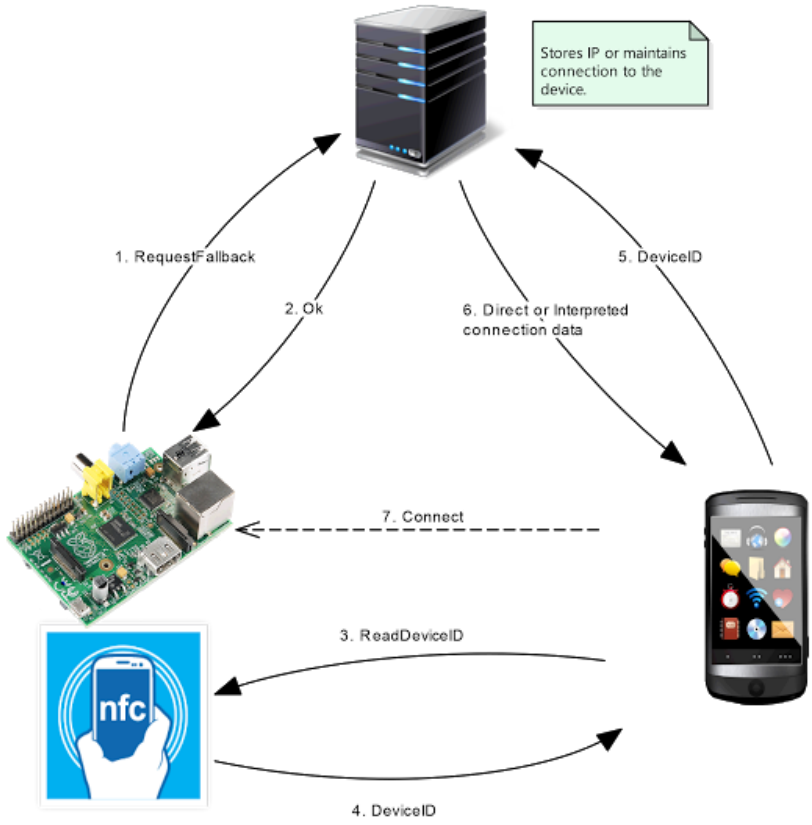
Amennyiben a Pi külső hálózatról elérhető lehetőség van nyilvános IP címmel való csatlakozásra. Ezt a folyamatot a szerver is segíti. A mérőeszköznek lehetősége van jelezni a szervernek, aki innen megtudja az eszköz aktuális nyilvános IP címét. Az ilyen módon bejelentkezett eszközök számára egy DDNS[22] (Dynamic Domain Name Service) szolgáltatást nyújtunk.

[22] http://compnetworking.about.com/cs/domainnamesystem/g/bldef_ddns.htm

Ennek a segítségével a felhasználók kényelmesen csatlakozhatnak az eszközhöz anélkül, hogy IP címét észben kéne tartani, ami ráadásul változhat is. Ilyenkor azonban az eszköz azonosítóhoz kell hozzájutni valahogyan. Ezt jelenleg egy, az eszközre helyezett NFC matricával biztosítjuk, mivel a mérés elvégzésénél amúgy is a közelben kell, hogy legyen a felhasználó. Ennek a megoldásnak az előnye, hogy nem szükséges a közös alhálózat többé, viszont bonyolultabb beüzemelni, hiszen port forward-ra van szükség ilyenkor, amihez az alhálózaton fenntartott IP címek regisztrálására is szükség lehet. További probléma, hogy az interneten utaztatott csomagoknál ingadozó késleltetés és csomag vesztés is megjelenhet.

7.3. Közvetített kapcsolat

Amennyiben a fenti két opció segítségével nem sikerülne kapcsolódni lehetőség van közvetített kapcsolat kezdeményezésére. Ezt mutatja be az alábbi ábra. Ekkor mindkét eszköz kapcsolatot kezdeményez a szerver felé. Az üzeneteket mindkét fél a szervernek küldi, az pedig tovább küldi azokat a másik fél számára. Erre a kapcsolódási módra elsődlegesen azért volt szükség, mert gyakran kellett olyan környezetben tesztelnünk, ahol az előző két kapcsolódási mód nem működött.



21. ábra. A szerver által támogatott kapcsolatfelvétel

A gyakorlatban ez a mód nagyon költséges, hiszen egy kapcsolathoz egyszerre két nyitott socket-re lenne szükség, ráadásul hosszabb utat tesz meg a csomag, mint a közvetlen kapcsolat esetén, ezzel tovább rontva annak a paramétereit. Mégis van létjogosultsága ennek a megoldásnak, hiszen bármilyen hálózati topológia esetén működik és nincs szükség hálózati konfigurációra sem.

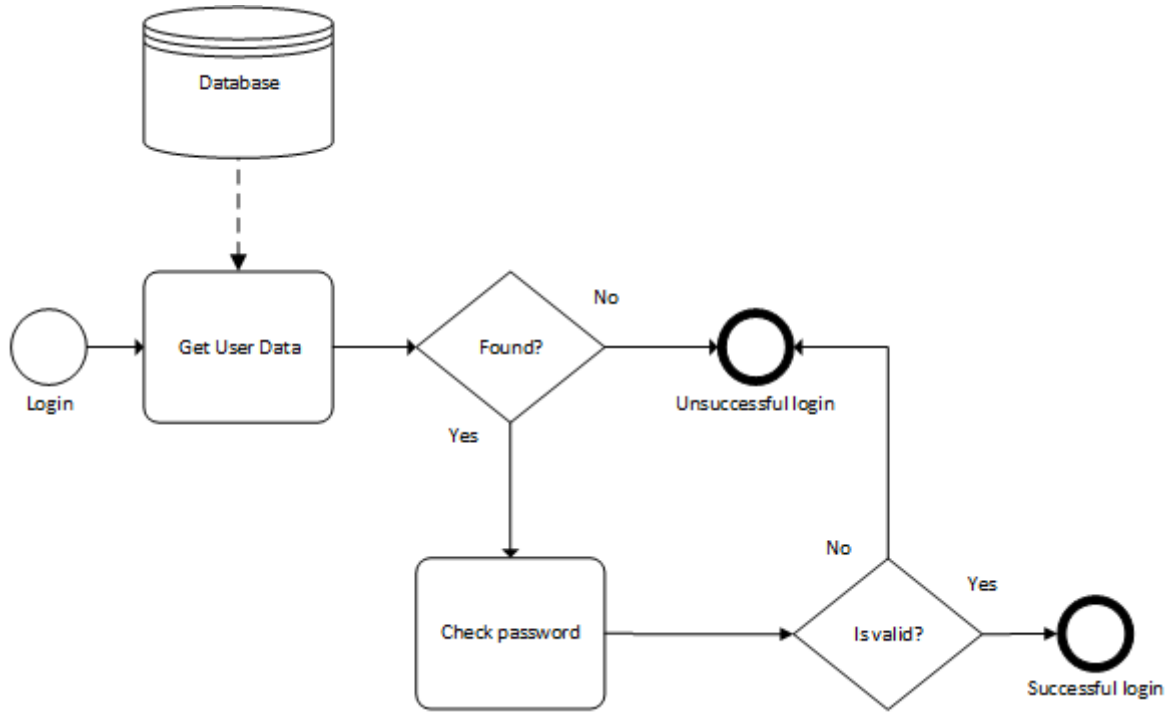
8. Hitelesítés

A rendszerünkben alapvetően kétféle hitelesítést alkalmazunk. Egyrészről a felhasználókat, másrészről pedig a méréseket kell vizsgálnunk. Előbbire azért van szükség, hogy ne férjenek más adataihoz hozzá jogtalanul a felhasználók. A mérés hitelesítésére pedig azért van szükség, hogy a nem valós mérési értékek ne kerülhessenek fel az adatbázisba, illetve védelmet nyújtunk a man-in-the-middle [**15**] jellegő támadásokkal szemben is, mivel a mérési adatok titkosított csatornán utaznak így az adatok lehallgatására, vagy módosítására nincs lehetőség. Ezeknek a megvalósításáról a következő alfejezetekben írunk.

8.1. Felhasználók hitelesítése

A felhasználók a rendszert csak előzetes regisztrációt követően használhatják. Regisztráció során egy e-mail cím, jelszó párossal azonosítják magukat, ahol e-mail címük egyben a felhasználó nevük is és ez egyértelműen azonosítja őket a teljes rendszerben. Jelszavukat titkosítva tároljuk az adatbázisban, ennek egyértelmű visszafejtése - a kódolási eljárásnak köszönhetően - a rendszer üzemeltetői számára sem lehetséges.

A szerver valamennyi szolgáltatását a felhasználók csak bejelentkezést követően érhetik el, melynek folyamatábráját az alábbi kép szemlélteti. Bejelentkezés során a szerver elsőként ellenőrzi, hogy a megadott felhasználónév szerepel-e az adatbázisban. Amennyiben nem, akkor értelemszerűen felesleges a folyamatot folytatni. Létező felhasználónév megadását követően a titkosított jelszót a szerver karakterenként veti össze az adatbázisban tárolt titkosított jelszóval. Amennyiben egy karakter is eltér, úgy a jelszó hibás volt és a bejelentkezés nem történhet meg. A titkosított jelszó tulajdonképpen egy betűkből, számokból és speciális karakterekből álló, látszólag értelmetlen string.



22. ábra. Bejelentkezés menete

Sikeres bejelentkezést követően a szerveren egy munkafolyamat lesz létrehozva a felhasználók számára, amely harminc perces tétlenséget követően automatikusan megszűnik. Ennek

köszönhetően amennyiben a felhasználó elfelejt kijelentkezni, a feleslegessé vált munkafolyamata nem fog tovább erőforrásokat foglalni a szerveren.

8.2. Mérések hitelesítése

A mérések hitelesítésére egy tokent alkalmazunk, amelyet a szerver generál le és értelmezni is csak ő tudja. Mikor a felhasználó egy mérést kezdeményez, akkor az adott Raspberry egyedi azonosítóját elküldi a szervernek. Ekkor kerülnek összegyűjtésre az elkészítéséhez szükséges adatok, vagyis a mérést kezdeményező felhasználó egyedi azonosítója, a felhasználó profiljának egyedi azonosítója, a mérést végző Raspberry IP címe, a felhasználó munkafolyamatának egyedi azonosítója, valamint egy generált token kulcs kerül bele.

Ennek a szerepe, hogy a kezdeményezett méréshez létrehozott tokent csak és kizárólag az arra jogosult személy tudja felhasználni, aki a folyamatot indította. Továbbá felhasználásával garantálni tudjuk, hogy a egy példányt kizárólag csak egyszer lehet felhasználni.

Annak érdekében, hogy a tokenben szereplő adatokat a felhasználók ne tudják manipulálni, az egészet titkosítanunk kellett. Erre az AES-128 nevű titkosítási eljárást alkalmaztuk, amely a Rijndael-féle kriptográfiai technológián alapszik. Amennyiben a felhasználó nem rendelkezik azzal 128 bit hosszú kulccsal, amivel a titkosítást végeztük, úgy nem tudja értelmezni a szervertől kapott információt.

A Német információbiztonsági hivatal 2015-ös ajánlása [**16**] szerint a 128 bit hosszú szimmetrikus titkosítási kulcs 2021-ig megfelelő védelmet nyújt, míg az Amerikai nemzetbiztonsági hivatal (NSA) idei ajánlása [**17**] szerint AES használata esetén a szigorúan titkos adatokat 256 bites kulccsal javasolt titkosítani. A Java programozási nyelvben a 128 bitet meghaladó kulcsméret használata jelenleg csak egy kriptográfiai kiegészítő csomag[23] telepítése után érhető el, így az ajánlások alapján és kompatibilitási okokból a 128 bites kulcshosszúság mellett döntöttünk.

A szerveren el sem tároljuk az aktív tokeneket, hiszen ha a szerver dekódolni tudja azt, akkor biztosan ő kódolta le, így a forrás megbízhatónak tekinthető. Ettől függetlenül további ellenőrzéseken esik át a dekódolást követően.

Először a token kulcsot vizsgáljuk meg, amit a felhasználó munkamenetében is eltárolunk. Ennek köszönhetően garantálni tudjuk, hogy csak a mérést kezdeményező felhasználó legfeljebb egyszer tudja felhasználni azt. A hitelesítés során ezt a két kulcsot vetjük össze elsőként és ha ezek nem egyeznek meg, úgy a mérés már nem tekinthető hitelesnek, így törlésre kerül.

A hitelesítés során a szerver megvizsgálja, hogy a tokenben található IP cím megtalálható-e az aktív mérőeszközök listájában. Amennyiben nem, akkor a Raspberry nincs is bekapcsolt állapotban, így a mérést el sem végezheti, így feltételezhető, hogy a mérési folyamatot manipulálni szeretné egy harmadik személy.

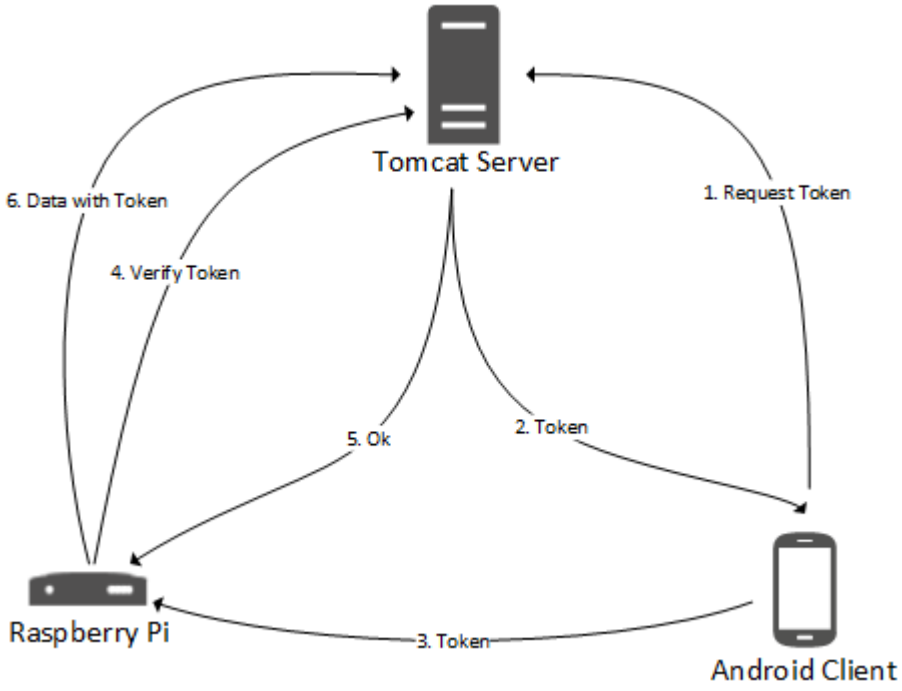
A token érvényessége alapvetően kétféleképpen járhat le. Az első verzió, hogy a felhasználónak lejár az aktív munkamenete, így az abban tárolt hitelesítési információk elvesznek. Ezt okozhatja, hogy kijelentkezik, vagy a megengedett 30 percnél hosszabb ideig tétlen marad. Ekkor a validálás sikertelen lesz, hiszen ehhez nincsenek meg a szükséges információk. A második verzió, hogy a

[23] http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

felhasználó a mérés végéig aktív marad. Ekkor a szerver az egyszeri felhasználás biztosítása érdekében érvényteleníti a munkamenetben található validáláshoz szükséges adatot.

A token használata egy további hitelesítési problémát is megold. Célunk volt az, hogy csak azokkal a Raspberry-kkel lehessen mérést kezdeményezni, amelyek előzőleg már bejelentkeztek a szerverre. Ennek köszönhetően garantálni tudjuk, hogy csak olyan mérést indíthat el a felhasználó, amit előzőleg a szerver már hitelesített. Ez megoldás arra, hogy harmadik személy ne tudja a rendszert lemásolva a felhasználók adatait megszerezni egy-egy hamis mérésen keresztül.

Ezt úgy oldottuk meg, hogy amikor az Android alkalmazás mérést kezdeményez, akkor a méréshez kiválasztott szenzor azonosítója mellett a szerver által leküldött tokent is továbbítja a Raspberry számára. Ez látható az alábbi ábrán is, amely bemutatja a token szegmensek közötti utaztatását. Ekkor a Pi felküldi azt a szerverhez, ahol átesik a szükséges ellenőrzésen. Ezt követően az eredményekről a szerver értesíti a Raspberry-t. Amennyiben a validálás sikeres volt, a mérés elkezdődik. Ellenben ha a szerver nem találta érvényesnek azt, akkor a Raspberry megszakítja a mérési folyamatot.



23. ábra. Token küldése a rendszerben

Amennyiben a mérés kezdeményezését a szerver hitelesnek találta, akkor a Raspberry elvégzi a szükséges folyamatokat. A mérést a tokennel együtt küldi fel a szervernek, ahol az ismét átesik a szükséges validációs eljáráson. Erre azért van szükség, hogy harmadik személy ne tudjon a Raspberry nevében tetszőleges mérési adatokat felküldeni. Amennyiben a szerver érvénytelennek találja, akkor nem dolgozza fel a mérési értékeket.

További biztonsági rést jelentene a rendszeren az, ha egy harmadik személy megszerzi azokat az információkat, hogy az adott mérési eredményeket a szerver mely interfészére kell elküldeni, illetve, hogy a küldendő információt milyen formátumban továbbítja a Raspberry. Ebben az esetben egy érvényes token birtokában bármilyen mérési adatokat fel lehetne küldeni.

Ezt a problémát a már említett AES-128 titkosítási eljárás segítségével oldottuk meg. A Raspberry a szerver felé küldött információkat egy 128 bit hosszú kulccsal titkosítja, mely kulcsot a szerver is ismeri, így dekódolni is tudja. Azonban harmadik személy a kulcs birtokának hiányában sem olvasni, sem írni nem tudja az adatokat, így teljes mértékben garantálva azt, hogy a mérési folyamat külső beavatkozás nélkül zajlódjon le.

9. Tesztprogramok

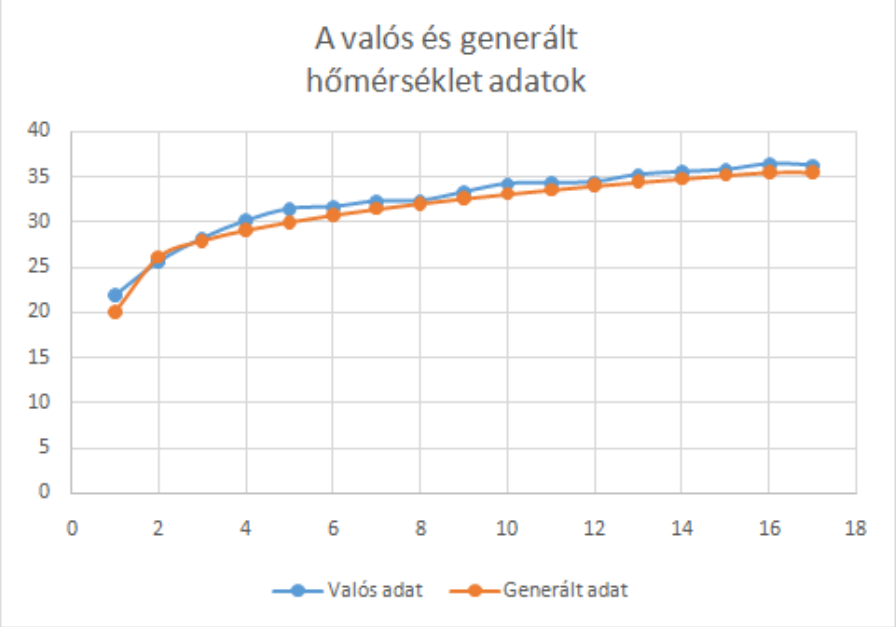
Egy rendszer tervezése, fejlesztése és üzemeltetése során elengedhetetlen annak részletes, átfogó tesztelése. Annak érdekében, hogy a rendszer egyes szegmensei hiányában is lehessen ellenőrizni a működést, különböző tesztprogramokat, és drivereket készítettünk, melyeket a következő fejezetekben mutatunk be.

9.1. Teszt driverek

A mérési folyamat tesztelésére lehetőségünk van szenzorok nélkül is. Ehhez csak egy olyan illesztő program készítésére van szükség, ami nem szenzorról olvassa a kiadott értékeket. A tesztelések során több realisztikus értéket szolgáltató teszt driver is készült. A realisztikus értékre azért volt szükség, mert így a mérést kiértékelő modulokat is tudjuk tesztelni. Nem készült tesztprogram az összes létező szenzorhoz, viszont az összes jelenleg támogatott adattípust lefedik, így a rendszer átfogó tesztelésére is alkalmasak. Az elkészült teszt programokról egy rövid bemutató található ebben a fejezetben.

9.1.1. A hőmérséklet mérő

A teszt programot a valós szenzoron megfigyelt működés alapján terveztük. A megfigyelt viselkedés a következő: a mérés elején a szenzor közel szoba hőmérsékletet jelez. A kijelzett érték először nagyobb lépésekben közelíti a testhőmérsékletet, de a lépések egyre kisebbek lesznek, végül pedig elsimul az álltaluk rajzolt görbe. A működését a következőképpen szimuláljuk: legeneráljuk a környezeti és testhőmérséklet közti különbséget egy megengedett intervallumon belül. A kijelzett érték először a környezeti hőmérséklethez közeli, de polinom segítségével közelítjük a kívánt végeredményt. A valós és generált adatok közti hasonlóságot, a lenit grafikonon lehet megfigyelni.



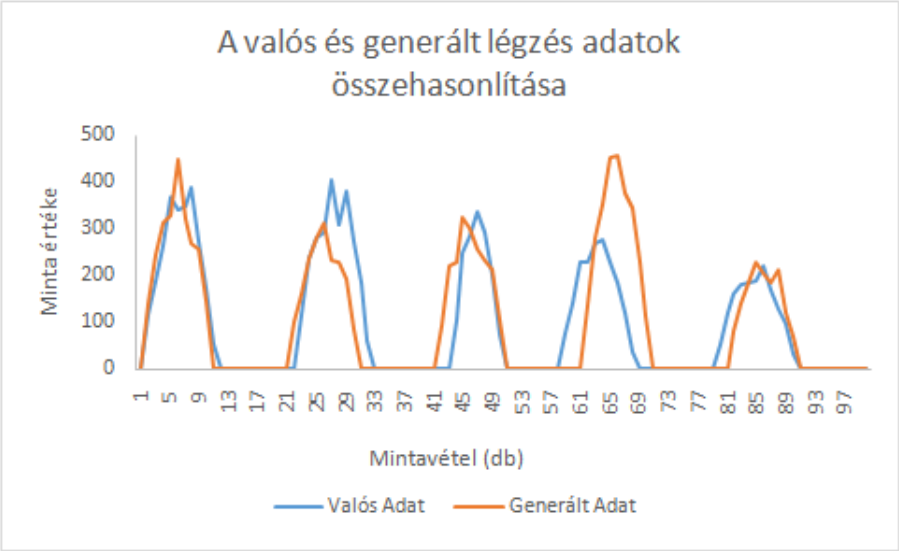
24. ábra. A valós és generált hőmérséklet adatok

9.1.2. Légzés mérő

A légzés mérő szerepe, hogy folytonos adatot mér, így szemben a hőmérséklet mérővel a mérés végeredménye nem írható le egyetlen számmal, ez tehát a rendszer egy külön részét hivatott tesztelni.

A légzés mérőt teszt programot működését a valós szenzoron megfigyelt viselkedés alapján terveztük. A megfigyelt viselkedés a következő: A szenzor csak a kilégzést méri, belégzés esetén nullát mutat. A nulla és nem nulla szakaszok közelítőleg periodikusan váltják egymást. A kilégzés intenzitása növekszik, majd elér egy csúcspontot, ahonnan kezdve csökken szinusz-szerű hullámformát leírva.

A működés szimulációját a következőek szerint oldottuk meg: Az adatokat a szinusz függvény felhasználásával állítjuk elő, azonban ha ez negatív lenne nullát adunk vissza. A függvényt periódusonként állítjuk elő. Az aktuális periódus amplitúdója véletlenszerűen kerül meghatározásra egy előre definiált tartományon belül. Az egyes mintákat is eltorzítjuk, hogy ne tisztán szinuszos periódusokat kapjunk, hanem a légzéshez jobban hasonlító mintát. A generált és valós légzési adatok közti hasonlóság az alábbi grafikonon megfigyelhető.



25. ábra. A valós és generált légzés adatok összehasonlítása

9.1.3. Pulzus és véroxigén mérő

A pulzus és véroxigén szintmérő tesztprogram szerepe a több értéket is mérni képes szenzorok reprezentálása, ez tehát a rendszernek a fenti kettőtől eltérő részét hivatott tesztelni. Az ilyen típusú szenzorok közül ez volt az egyik legegyszerűbb működéstű, ezért is készült ezt használtuk ilyen célra.

A példa adatok generálásását a valós szenzor működésének megfigyelése alapján mintáztuk. A megfigyelt viselkedés a következő: Az első adat csak több másodperc után jön, utána azonban meglehetősen stabilan tartja a mért értékeket, csak kisebb változások történhetnek.

A példa programunk a következőképpen működik: Pár másodpercig nem csinálunk semmit, majd kiadunk egy véletlenszerűen generált eredményt egy előre definiált tartományból. A program a további mintavételek esetén a generált értékhez közeli adatokat szolgáltat. Az eltérés iránya egyaránt lehet pozitív és negatív is.

9.2. Mérés generáló

Az előző fejezetekben olyan programokat mutattunk be, amelyek a valós mérési értékeknek megfelelő eredményeket generálnak le a Raspberry Pi és szenzorok hiányában. Azonban az adatok létrehozása ebben az esetben is időigényes - ahogyan a valós mérések esetében is -, így egy olyan program elkészítésére is szükség volt, ami nagy mennyiségű adatot képes legenerálni rövid idő alatt.

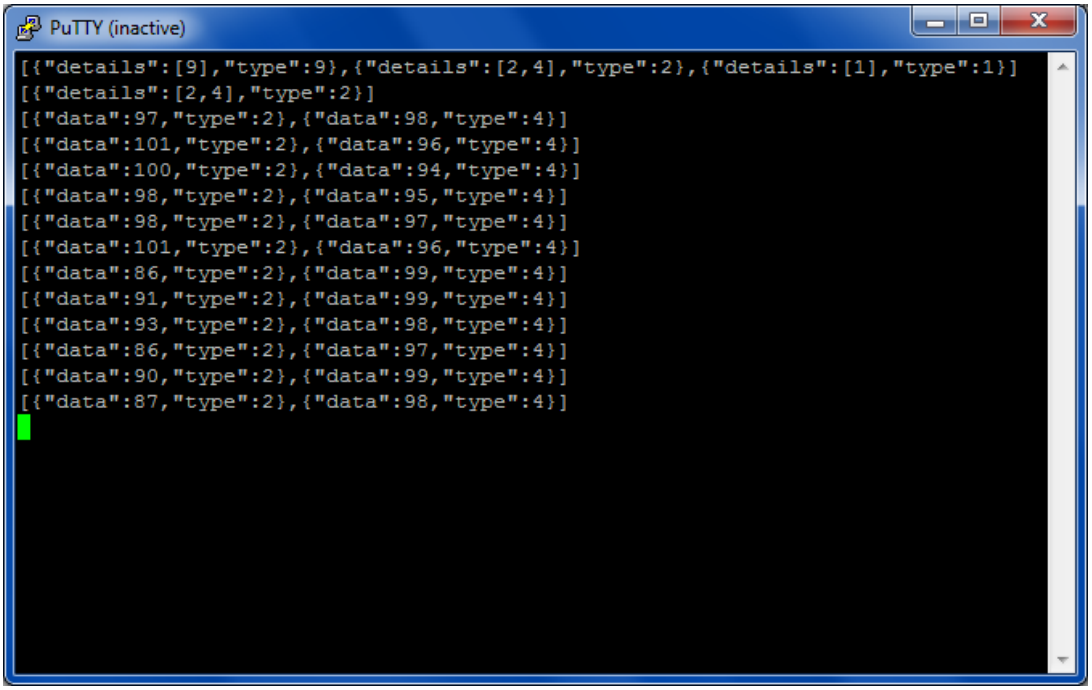
A program parancssori argumentumként kapott értékek alapján hozza létre a méréseket. Meg lehet adni, hogy milyen típusú szenzor adatokat készítsen, de azok mennyiségét és véletlenszerűségét is szabályozni tudjuk.

A fejlesztés során nem volt mindig lehetőségünk az Automatizálási és Alkalmazott Informatikai Tanszék által biztosított teszt szerver használatára. Így a kéréseket azok a számítógépek szolgálták ki, amiken a fejlesztés történt. A tesztprogramban azt is kényelmesen meg lehet határozni, hogy éppen melyik kiszolgáló aktív, így hova küldje a kéréseket.

Az adatok generálása során vizsgáltuk a szerverek és a generáló program teljesítményét is. Az eredményekből megállapítható volt, hogy kevesebb, mint öt perc leforgása alatt közel 1500 mérési értéket tudunk előállítani, kiértékelni és eltárolni. Ezek a tesztelések rámutattak a szerver alkalmazás szűk keresztmetszeteire, azokra a kódrészletekre, melyeken optimalizálást kellett végrehajtanunk.

9.3. Socket kommunikáció tesztelése

A socket kommunikáció tesztelésére a PuTTY nevű nyílt forrású telnet és SSH klienst használjuk. A PuTTY lehetővé teszi a socketen kapott információ nyers megjelenítését, illetve a felhasználó által begépelt adatok kiküldését gyorsan és egyszerűen, így kommunikációs hiba esetén a feleket külön-külön tudjuk tesztelni a hiba gyors beazonosításához. Az alábbi ábrán látható a PuTTY használat közben.



26. ábra. A PuTTY használat közben

10. Összefoglalás

Az általunk készített rendszer segítségével a felhasználók központilag elhelyezett mérőberendezések segítségével követhetik egészségi állapotukat, mely méréseket a megadott paraméterek függvényében ki is elemzünk. Az összegyűjtött adatokon korreláció keresést végzünk, amely felhasználásával még időben értesíthetjük a felhasználóinkat a rendellenességek közötti összefüggésekről. A jövőben az sem kizárt, hogy a felgyülemlett adatok között olyan kapcsolatokat is felfedezhetünk, amelyek az orvostudomány számára jelenleg még ismeretlenek.

A projekt során számtalan számunkra új technológiát megismertünk, melyek közül az IoT világa emelhető ki. A jelenlegi tendencia szerint az informatikában egyre nagyobb a hangsúly ezen a területen, így mindannyiunk számára nagyon hasznos tapasztalatot jelentett ennek megismerése.

A tervezés és fejlesztés során végig arra törekedtünk, hogy a lehető legrugalmasabb megoldást valósítsuk meg. Fontos volt számunkra, hogy ne zárjuk ki új szenzorok integrálásának lehetőségét, figyelmet fordítottunk újabb kliensek felvételére is. Munkánk során többször át kellett dolgozni a megoldásainkat, azonban a kialakított architektúránknak köszönhetően ezekkel a módosításokkal mindig gyorsan végeztünk.

10.1. Továbbfejlesztési irányok

A meglevő korreláció keresést egy komplexebb összefüggéseket is kimutatni képes konstrukcióra lehetne cserélni, ezzel növelve a feltárt összefüggések spektrumát.

A rendszer felhasználási körét más területekkel is bővíteni lehetne. Így például az okosotthon szenzorkészlete, vagy a sport közbeni állapotmonitorozás is integrálható lehetne a rendszerünkbe. Az okosotthont felhasználva olyan környezeti információkat tudnánk megállapítani a mérések során, amely befolyásolhatják az eredményeket. Ilyen a helyiség hőmérséklete, vagy a páratartalma is.

Szeretnénk a felhasználási köröket az aktuális trendek felé is kiterjeszteni. Terveink között szerepelnek a Quantified Self [**18**] és Self awareness [**19**] használatának lehetővé tétele az alkalmazásban, ezzel növelve a rendszerünk népszerűségét.

Rendszerünkben található hordozható, viselhető szenzorok segítségével már jelenleg is tudjuk monitorozni a sport közben a felhasználók egészségi állapotát, azonban az ilyen típusú terhelést nem kezeljük külön. Megfigyelhető, hogy futás közben mindenkinek magasabb a pulzusszáma, amit jelenleg a rendszerünk egészségtelennek minősítene. Ezzel a módosítással pedig lehetőségünk nyílna a sportolók számára olyan előrejelzéseket is szolgáltatni, amik például az izomsérülésüket is megelőzhetnék, illetve az edzésterveket is javítani tudnák.

Jelenleg még nem alakult ki egyértelműen a kutatást kezdeményező Telekom bevezetési stratégiája. Szó volt arról, hogy a szolgáltatás T-pontokba történő kihelyezésével használnák fel a rendszerünket, azonban a viselhető eszközök számtalan újabb felhasználási területet biztosítanak.

Hiszünk abban, hogy a fejlett technológia által elérhetővé vált szenzorok fontos szerepet fognak játszani a közeljövő hatékony egészségügyi ellátásában, a betegségek időben történő felismerésében, illetve a nagy adathalmazokból leszűrhető összefüggések segítségével történő megelőzésében. Azt gondoljuk, hogy kutatómunkánkkal egy lehetséges irányt kidolgozva hozzájárultunk ehhez a fejlődéshez. A pozitív visszajelzések motivációt jelentenek nekünk ennek az útnak a folytatására.

Irodalomjegyzék

[1] Mark Massé, *REST API Design Rulebook*. United States: O'Reilly Media, Inc., 2012.

[2] Peter Sloot, *Future Generation Computer Systems*., 2015.

[3] Amy J Barton, "The regulation of mobile health applications," *BMC Medicine*, május 2012.

[4] Juho Hamari, "The Sharing Economy: Why People Participate in Collaborative Consumption," *Journal of the Association for Information Science and Technology*, 2015.

[5] Paul Zikopoulos, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*.: McGraw-Hill Osborne Media, 2011.

[6] Kyle Banker, *MongoDB in Action*. Greenwich, CT, United States: Manning Publications Co., 2011.

[7] Nurzhan Nurseitov, Comparison of JSON and XML Data Interchange Formats: A Case Study, 2009.

[8] Shuang Zhao, "Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device messaging service," in *Annual Computer Security Applications Conference*, New York, United States, 2012, pp. 119-128.

[9] Tom Igoe, *Beginning NFC*.: O'Reilly Media, 2014.

[10] Satya Komatineni, *Expert Android*.: Apress, 2013.

[11] Lawrence I-Kuei Lin, *A Concordance Correlation Coefficient to Evaluate Reproducibility*, 45th ed.: International Biometric Society, 1989.

[12] Ziad S. Saad, *A new method for improving functional-to-structural MRI alignment using local Pearson correlation*. Houston, Texas, United States, 2008.

[13] Peter Fransson, *The precuneus/posterior cingulate cortex plays a pivotal role in the default mode network: Evidence from a partial correlation network analysis*. Paris, France, 2008.

[14] T. Cover, "Broadcast channels," *Information Theory, IEEE Transactions*, vol. 18, no. 1, pp. 2-14, január 1972.

[15] Ulrike Meyer, "A man-in-the-middle attack on UMTS," *ACM*, pp. 90-97, október 2004.

[16] Bundesamt für Sicherheit in der Informationstechnik. (2015, január) bsi.bund.de. [Online]. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile

[17] National Security Agency, USA. (2015, augusztus) nsa.gov. [Online]. https://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[18] Melanie Swan, "Emerging Patient-Driven Health Care Models: An Examination of Health Social Networks, Consumer Personalized Medicine and Quantified Self-Tracking," *Int. J. Environ. Res. Public Health*, február 2009.

[19] Dawn Freshwater, *Therapeutic Nursing*. London: SAGE Publications, 2002.