OULUN AMMATTIKORKEAKOULU

Jesse Orre

REITTIMITTAUSOHJELMISTO LANGATTOMALLE ANTURILLE

REITTIMITTAUSOHJELMISTO LANGATTOMALLE ANTURILLE

Jesse Orre
Opinnäytetyö
Syksy 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Jesse Orre
Opinnäytetyön nimi: Reittimittausohjelmisto langattomalle anturille
Työn ohjaaja: Jukka Jauhiainen
Työn valmistumislukukausi ja -vuosi: Syyslukukausi 2021 Sivumäärä: 35

Työn aiheena oli tehdä Android-pohjainen mobiilisovellus, jolla pystyy tekemään värähtelymittauksia Nomen kehittämällä langattomalla anturilla Bluetooth Low Energy -teknologian välityksellä sekä lukemaan ja analysoimaan tietoa. Värähtelymittaukset toteutetaan ottamalla yhteys käyttäjän valitsemaan anturiin sovelluksella, jonka jälkeen anturi kiinnitetään mitattavaan pisteeseen ja tehdään mittaus valitsemalla kyseinen mittapiste sovelluksesta. Mittapisteet ja niihin liittyvät tiedot haetaan yrityksen pilvipalvelusta REST-kutsulla ja tallennetaan paikalliseen SQLite-tietokantaan. Sovellusta on tarkoitus pystyä käyttämään myös offline-tilassa paikallisen SQLite-tietokannan ansiosta.

Työn toteutuksen aikana perehdyttiin tarkemmin Android-ohjelmointiin, Bluetooth Low Energy -teknologiaan sekä SQLite-tietokantoihin. Työ toteutettiin käyttämällä Android Studio -kehitysympäristöä ja Java-ohjelmointikieltä.

Työn lopputuloksena syntyi sovellus, jolla pystyy tekemään värähtelymittaukset ja vastaanottamaan mitatun datan Nomen kehittämältä nmas chess -anturilta. Mitattu data lähetetään REST-kutsulla pilveen sekä tallennetaan paikalliseen SQLite-tietokantaan. Sovellusta voi käyttää myös offline-tilassa paikallisen SQLite-tietokannan ansiosta.

Asiasanat: Android Studio, Bluetooth Low Energy, SQLite, mobiilisovellukset, värähtelymittaus

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Jesse Orre
Title of thesis: Route Measurement App for Wireless Sensor
Supervisor: Jukka Jauhiainen
Term and year when the thesis was submitted: Autumn 2021
Number of pages: 35

The purpose of the work was to create an Android-application that is capable of doing vibration measurements and receiving the measured data using a sensor developed by Nome. The vibration measurements are done by selecting a sensor from a list on the application, after which the sensor is attached to the Point to be measured and the measurement is made by selecting that measurement point from the application. Measurement points and related data are retrieved from a cloud service with a REST call and stored in a local SQLite database. The application is also intended to be available offline thanks to the local SQLite database.

During the implementation of the work, Android programming, Bluetooth Low Energy technology and SQLite databases were studied in more detail. The work was carried out using the Android Studio development environment and the Java programming language.

The result of the work was an application that can make vibration measurements and receive the measured data from nmas chess sensor developed by Nome. The measured data is sent with a REST call to the cloud and stored in a local SQLite database. The application can also be used offline thanks to the local SQLite database.

Keywords: Android Studio, Bluetooth Low Energy, SQLite, mobile applications, vibration measurement

SISÄLLYS

SANASTO

ARM (Advanced RISC Machines) Brittiläinen mikroprosessoriarkkitehtuuri.

BLE (Bluetooth Low Energy) Matalavirtainen versio Bluetoothista.

Bluetooth SIG (Bluetooth Special Interest Group) Bluetooth-standardia ylläpitävä järjestö.

FFT (Fast Fourier Transform) Algoritmi diskreetin Fourier-muunnoksen ja sen käänteismuunnoksen laskemiseksi.

GAP (Generic Access Profile) Ohjaa yhteyksiä ja mainontaa Bluetoothilla.

GATT (Generic Attribute Profile) Määrittää tavan, jolla kaksi BLE-laitetta voivat siirtää tietoa edestakaisin.

IDE (Integrated Development Enviroment) Ohjelmointiympäristö eli ohjelma, jolla ohjelmoija suunnittelee ja kehittää ohjelmistoa.

IEEE (Institute of Electrical and Electronics Engineers) Kansainvälinen tekniikan alan järjestö.

IIoT (Industrial Internet of Things) Teollisuuden esineistä koostuva verkko.

ISM (Industrial, Scientific and Medical) Joukko radiotaajuusalueita, jotka on varattu muiden sähkölaitteiden kuin varsinaisten radiolaitteiden käyttöön.

REST (Representational State Transfer) Rest -ohjelmointirajapintojen avulla voidaan lähettää pyyntöjä HTTP -tai salatun HTTPS-protokollan yli toisille palvelimille, jotka palauttavat datan jossain muodossa kuten JSON tai XML.

SDK (Software Development Kit) Kokoelma ohjelmistokehityksen työkaluja yhdessä asennettavassa paketissa.

UHF (Ultra High Frequency) Radiotaajuusalue, joka kattaa 0,3–3 GHz:n taajuusalueen.

UUID (Universally Unique Identifier) Tietojärjestelmien tietoihin tarkoitettu yksilötunniste.

1 JOHDANTO

Työn tarkoituksena oli luoda Android-pohjainen mobiilisovellus, jolla voidaan suorittaa värähtelydatamittauksia langattomalla anturilla Bluetooth Low Energy -teknologian välityksellä, lähettää saatu data pilvitietokantaan sekä tallentaa niitä paikallisesti. Värähtelydatamittaukset on tarkoitus suorittaa reittimittaustyylisesti, mikä tarkoittaa sitä, että kaikki mittaukset suoritetaan käyttäen yhtä anturia, jota siirrellään mittapisteeltä toiselle mittauksien valmistuttua. Mobiilisovellus tulee toimimaan Nomen tarjoaman nmas chess IIoT -järjestelmän kanssa, joka on tarkoitettu koneiden ja laitteiden kunnonvalvontaan. Nmas chess IIoT -järjestelmän on tarkoitus havaita aikaisin alkavat laakeriviat ja muut häiriöt, jotka vaikuttavat myöhemmin koneen toimintaan.

Nome Oy on suomalainen yritys, joka tarjoaa mittaavan ja ennakoivan kunnossapidon tuotteita sekä palveluita. Nomen päämääränä on ylläpitää ja parantaa asiakkaiden tuotannon kilpailukykyä ennaltaehkäisemällä yllättäviä laiterikkoja, ja samalla tuotannon turvallisuus parantuu ja ympäristövahinkojen riski pienenee. (1.)

Työssä perehdytään enimmäkseen reittimittaussovelluksen toimintaperiaatteeseen sekä Bluetooth Low Energy -teknologian ja SQLite-tietokannan käyttöön sovelluksessa.

2 KÄYTETYT TYÖKALUT JA TEKNOLOGIAT

2.1 Android-käyttöjärjestelmä

Android on Linux-pohjainen käyttöjärjestelmä, joka on suunniteltu ensisijaisesti kosketusnäyttöön tarkoitettuihin mobiililaitteisiin, kuten älypuhelimiin ja tabletteihin. Android on yksi nykyään eniten käytetyistä mobiilikäyttöjärjestelmistä. Android-ohjelmisto sai alkunsa Kalifornian Palo Altossa vuonna 2003. (2.)

Android-ohjelmistoa tukeva laitteisto perustuu ARM-arkkitehtuurialustaan. Android on avoimen lähdekoodin käyttöjärjestelmä, joka tarkoittaa, että se on ilmainen ja kuka tahansa voi käyttää sitä. (2.)

Suositeltu ja kätevin tapa kehittää Android-sovelluksia on käyttää Java-ohjelmointikieltä. Vaikka Java on yleiskäyttöinen työkalu, sitä käytetään yhdessä Android Software Development Kit (SDK) -ohjelmiston kanssa Android Studio -ympäristössä sovellusten kehittämiseen. Toinen virallinen tapa on käyttää C++:aa Native Development Kitin (NDK) kanssa. (3, s 14.)

SDK tarkoittaa ohjelmistokehityspakettia tai lyhyesti devkit. Se on joukko ohjelmistotyökaluja ja ohjelmia, joita kehittäjät käyttävät sovellusten luomiseen tietyille alustoille. SDK-työkalut sisältävät useita asioita, kuten kirjastoja, dokumentaatioita, koodinäytteitä, prosesseja ja oppaita, joita kehittäjät voivat käyttää ja integroida omiin sovelluksiinsa. (4.)

2.2 Android Studio

Android Studio on virallinen integroitu kehitysympäristö (IDE) Android-sovelluskehitykselle, joka perustuu IntelliJ IDEA -kehitysympäristöön. IntelliJ.n koodieditorin ja kehittäjätyökalujen lisäksi Android Studio tarjoaa vieläkin enemmän ominaisuuksia, jotka parantavat käyttäjän tuottavuutta Android-sovelluksia rakennettaessa, kuten

- joustava Gradle-pohjainen rakennusjärjestelmä
- nopea ja monipuolinen emulaattori

- monipuolinen ja helppokäyttöinen kääntötyökalu
- valmiit koodimallit ja GitHub-integraatio
- laajat testaustyökalut
- työkalut suorituskyvyn, käytettävyyden, version yhteensopivuuden ja muiden ongelmien selvittämiseen. (5.)

2.3 Java

Java on Sun Microsystemsin vuonna 1991 luoma korkean tason luokkiin perustuva olio-ohjelmointikieli, joka on suunniteltu siten, että sillä on mahdollisimman vähäinen määrä toteutusriippuvuuksia. Java on yleiskäyttöinen ohjelmointikieli, jonka avulla sovelluskehittäjät voivat kirjoittaa koodin kerran ja ajaa sen missä tahansa, mikä tarkoittaa, että käännetty Java-koodi voi toimia kaikilla Javaa tukevilla alustoilla ilman uudelleenkääntämistä. (6.)

2.4 Bluetooth

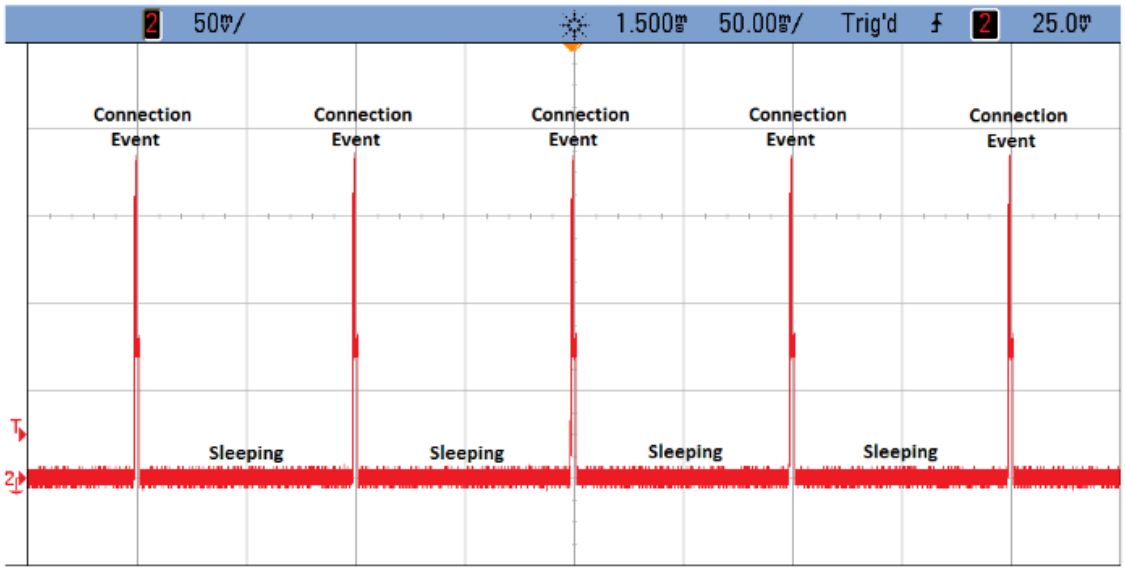
Bluetooth on lyhyen kantaman langattoman tekniikan standardi, jota käytetään tiedonsiirtoon erilaisten laitteiden välillä lyhyillä etäisyyksillä käyttämällä UHF-radioaaltoja ISM-kaistoilla, välillä 2,402–2,48 GHz, ja rakennettaessa henkilökohtaisia verkkoja. Sitä käytetään pääasiassa vaihtoehtona langallisille yhteyksille, tiedostojen vaihtamiseen lähellä olevien laitteiden välillä ja matkapuhelinten ja musiikkisoittimien liittämiseen langattomilla kuulokkeilla. (7.)

Bluetoothia hallinnoi Bluetooth Special Interest Group (SIG), jolla on yli 35 000 jäsenyritystä televiestinnän, tietojenkäsittelyn, verkostoitumisen ja kulutuselektroniikan aloilla. IEEE standardoi Bluetoothin nimellä IEEE 802.15.1, mutta ei enää ylläpidä standardia. Bluetooth SIG valvoo spesifikaation kehittämistä, hallinnoi pätevyysohjelmaa ja suojaa tavaramerkkejä. (7.)

2.5 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) -teknologia on matalavirtainen versio Bluetoothista, joka suunniteltiin toimimaan pienellä paristolla toimivien IoT-laitteiden kanssa. BLE:n matalan virrankulutuksen ansiosta BLE:tä käyttävät laitteet voivat kestää jopa useita vuosia. Matala virrankulutus johtuu siitä, että BLE-laite lähettää tietoa vain pienen prosenttiosuuden kokonaisajasta, kun laite on kytketty.

Tiedon lähetys tapahtuu lyhyissä jaksoissa, joiden välissä laite on matalavirtaisessa lepotilassa. (8.) (Kuva 1.)



KUVA 1. Virrankulutus BLE-yhteyden aikana (8, s 3)

Bluetooth Low Energy tarjoaa kaksi tapaa kommunikoida. Ensimmäinen käyttää mainoksia (BLE Advertising), joissa BLE-oheislaite lähettää paketteja kaikille ympärillä oleville laitteille. Vastaanottava laite voi sitten toimia näiden tietojen perusteella tai muodostaa yhteyden saadakseen lisätietoja. Toinen tapa kommunikoida on vastaanottaa paketteja yhteyden kautta, jolloin sekä oheislaite että keskuslaite lähettävät paketteja. (9.)

BLE-mainonta on yksi tärkeimmistä Bluetooth Low Energy -ominaisuuksista. Mainosten oikean käytön ymmärtäminen voi auttaa vähentämään virrankulutusta, nopeuttamaan yhteyksiä ja parantamaan luotettavuutta. Kahden laitteen välille ei voida luoda yhteyttä ilman mainoksia. Mainospakettien tietojen ja muodon määrittäminen on yleensä ensimmäinen asia, jonka parissa BLE-laitteita kehitetään. (9.)

Mainonta on rakenteeltaan yksisuuntaista. Keskuslaite ei voi lähettää mitään tietoja oheislaitteelle ilman yhteyttä, mutta yksi oheislaite voi mainostaa useille alueen keskuslaitteille ilman yhteyttä. (9.)

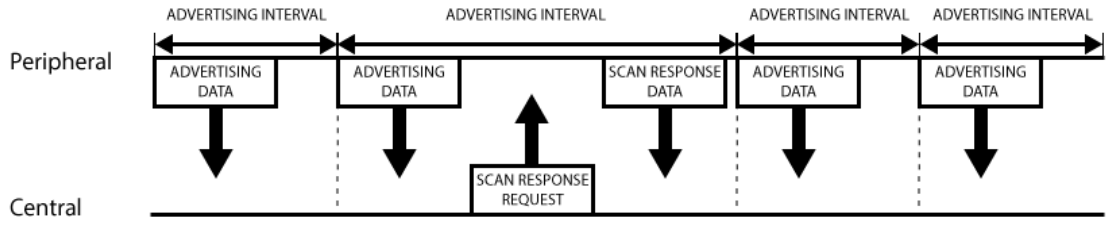
2.5.1 Generic Access Profile (GAP)

GAP eli Generic Access Profile ohjaa yhteyksiä ja mainontaa Bluetoothilla. GAP tekee laitteesta näkyvän ulkomaailmaan ja määrittää, miten kaksi laitetta voivat olla vuorovaikutuksessa keskenään. (10.)

GAP määrittelee eri roolit laitteille ja sillä on kaksi keskeistä käsitettä, jotka ovat keskuslaitteet (Central) ja oheislaitteet (Peripheral). Oheislaitteet ovat pieniä, pienitehoisia, resursseja rajoittavia laitteita, jotka voivat muodostaa yhteyden paljon tehokkaampaan keskuslaitteeseen. Oheislaitteita ovat esimerkiksi sykemittari, BLE-yhteensopiva läheisyystagi jne. Keskuslaitteet ovat yleensä paljon enemmän prosessointitehoa ja muistia vaativia laitteita, kuten matkapuhelin tai tabletti. (10.)

Mainoksia voidaan lähettää kahdella tavalla GAP:n avulla: mainostietojen tietopaketilla (Advertising Data) ja skannausvasteen tietopaketilla (Scan Response). Molemmat tietopaketit ovat identtisiä ja voivat sisältää enintään 31 tavua dataa, mutta vain mainonnan tietopaketti on pakollinen, koska tämä tietopaketti lähetetään jatkuvasti laitteesta, jotta kantaman sisällä olevat keskuslaitteet tietäisivät sen olemassaolosta. Skannausvasteen tietopaketti on valinnainen toissijainen tietopaketti, jota keskuslaitteet voivat pyytää, ja sen avulla kehittäjät voivat sovittaa hieman enemmän tietoa mainoksen tietopakettiin, kuten merkkijonot laitteen nimeä varten jne. (10.)

Kuvassa 2 näytetään GAP:n mainontaprosessi ja se, miten mainosten ja skannausvasteen tietopaketit toimivat.



KUVA 2. GAP:n mainontaprosessi (10)

Oheislaite asettaa tietyn mainosvälin, ja joka kerta, kun tämä väli kuluu, se lähettää uuden mainospaketin. Pidemmät viiveet säästävät virtaa, mutta heikentävät saavutettavuutta, jos laite mainostaa itseään vain 2 sekunnin välein 20 ms:n sijasta. (10.)

2.5.2 Generic Attribute Profile (GATT)

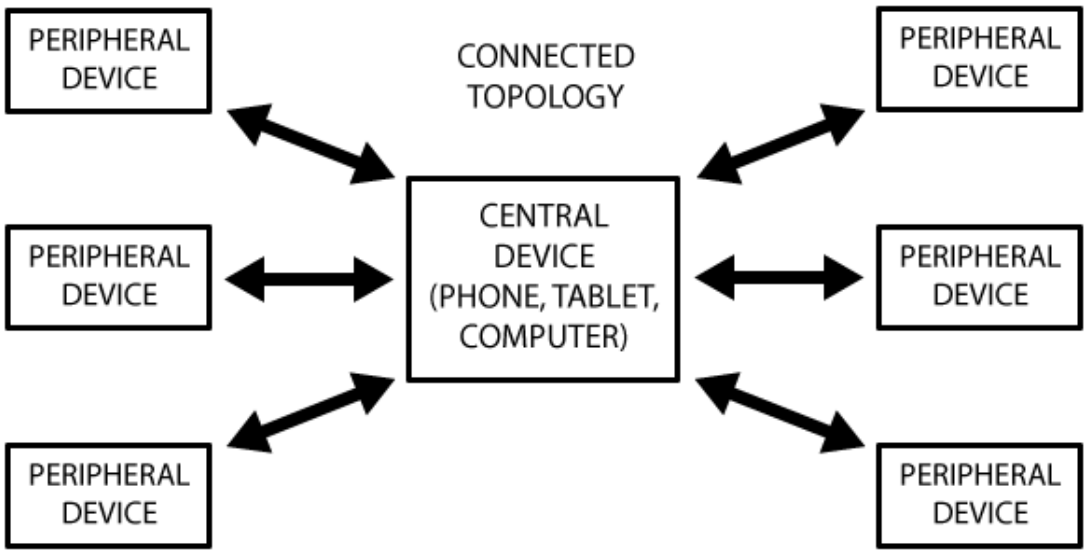
GATT on lyhenne sanoista Generic Attribute Profile, ja se määrittää tavan, jolla kaksi BLE-laitetta voivat siirtää tietoa edestakaisin käyttämällä palveluita (Service) ja ominaisuuksia (Characteristic). GATT käyttää yleistä dataprotokollaa nimeltä Attribute Protocol (ATT), jota käytetään palveluiden, ominaisuuksien ja niihin liittyvien tietojen tallentamiseen. (11.)

GATT tulee voimaan, kun kahden laitteen välille on muodostettu oma yhteys. Tämä tarkoittaa, että GAP:n hallitsema mainontaprosessi on käyty jo läpi. (11.)

GATT-yhteydet ovat yksinomaisia, mikä tarkoittaa sitä, että BLE-oheislaite voidaan liittää vain yhteen keskuslaitteeseen kerrallaan. Heti kun oheislaite muodostaa yhteyden keskuslaitteeseen, se lakkaa mainostamasta itseään, ja muut laitteet eivät enää voi nähdä sitä tai muodostaa siihen yhteyttä, ennen kuin olemassa oleva yhteys katkeaa. (11.)

Yhteyden muodostaminen on myös ainoa tapa sallia kaksisuuntainen viestintä, jossa keskuslaite voi lähettää merkityksellistä tietoa oheislaitteelle ja päinvastoin (11).

Kuvassa 3 näytetään, miten yhteys toimii BLE-laitteiden välillä.



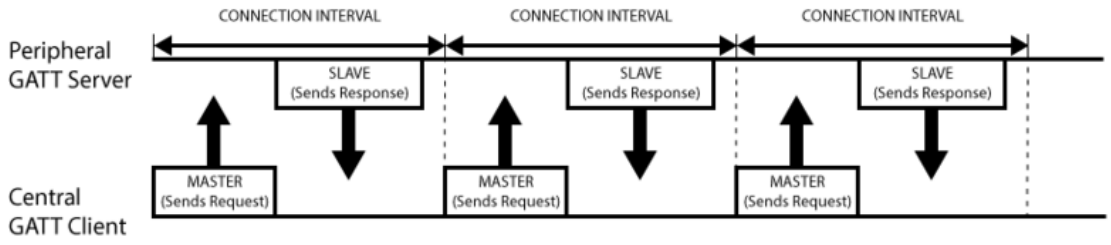
KUVA 3. Keskuslaitteen ja oheislaitteiden välinen yhteys (11)

Jos tietoja on vaihdettava kahden oheislaitteen välillä, on otettava käyttöön mukautettu järjestelmä, jossa kaikki viestit kulkevat keskuslaitteen läpi (11).

Oheislaite tunnetaan nimellä GATT-palvelin, ja se sisältää ATT-hakutiedot ja -palvelut sekä ominaisuudet. GATT-asiakas (puhelin/tabletti) lähettää pyynnöt GATT-palvelimelle. Kaikki tapahtumat aloittaa päälaite, GATT-asiakas, joka vastaanottaa vastauksen toissijaiselta laitteelta, GATT-palvelimelta. (11.)

Yhteyttä muodostaessa oheislaite ehdottaa ”yhteysväliä” (Connection interval) keskuslaitteelle, ja keskuslaite yrittää muodostaa yhteyden uudelleen jokaisen yhteysvälin jälkeen nähdäkseen, onko uutta tietoa saatavilla jne. Keskuslaite ei välttämättä pysty täyttämään pyyntöä, jos se on varattu puhumaan toisen oheislaitteen kanssa tai tarvittavat järjestelmäresurssit eivät ole käytettävissä. (11.)

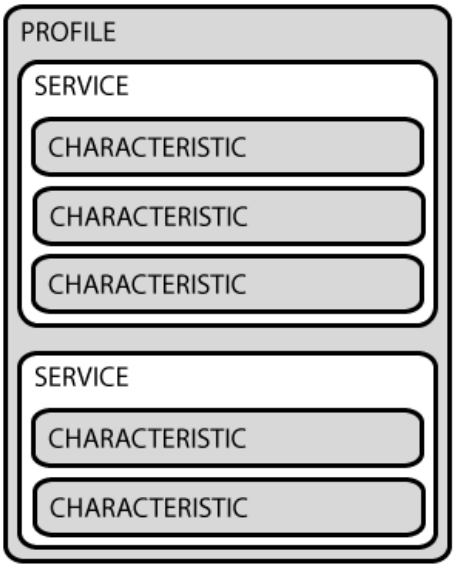
Kuvassa 4 havainnollistetaan tiedonsiirtoprosessia oheislaitteen (GATT-palvelin) ja keskuslaitteen (GATT-asiakas) välillä päälaitteen käynnistäessä jokaisen tapahtuman.



KUVA 4. GATT-palvelimen ja GATT-asiakkaan välinen tiedonsiirtoprosessi (11)

2.5.3 GATT-palvelut ja -ominaisuudet

GATT-tapahtumat BLE:ssä perustuvat korkean tason sisäkkäisiin objekteihin, joita kutsutaan nimillä profiilit (Profile), palvelut (Service) ja ominaisuudet (Characteristic), jotka näkyvät kuvassa 5 (11).



KUVA 5. Profiilit, palvelut ja ominaisuudet (11)

Profiilia ei ole olemassa BLE-oheislaitteessa, vaan se on yksinkertainen ennalta määritetty palvelukokoelma, jonka joko Bluetooth SIG tai oheislaitteiden suunnittelijat ovat koonneet (11).

GATT-palveluja käytetään tietojen jakamiseen loogisiin kokonaisuuksiin, ja ne sisältävät tiettyjä datapaloja, joita kutsutaan ominaisuuksiksi. Palvelulla voi olla yksi tai useampi ominaisuus, ja jokainen palvelu erottuu muista palveluista ainutlaatuisen numeerisen tunnuksen (UUID) avulla, joka voi olla joko 16-bittinen (virallisesti hyväksytyille BLE-palveluille) tai 128-bittinen (mukautetuille palveluille). (11.)

GATT-tapahtumien alimman tason käsite on ominaisuus, joka kiteyttää yhden datapisteen. Palveluiden tapaan jokainen ominaisuus erottuu ennalta määritetyn 16-bittisen tai 128-bittisen UUID-tunnuksen kautta. Ominaisuudet ovat tärkein kohta, jota käytetään BLE-oheislaitteiden kanssa. (11.)

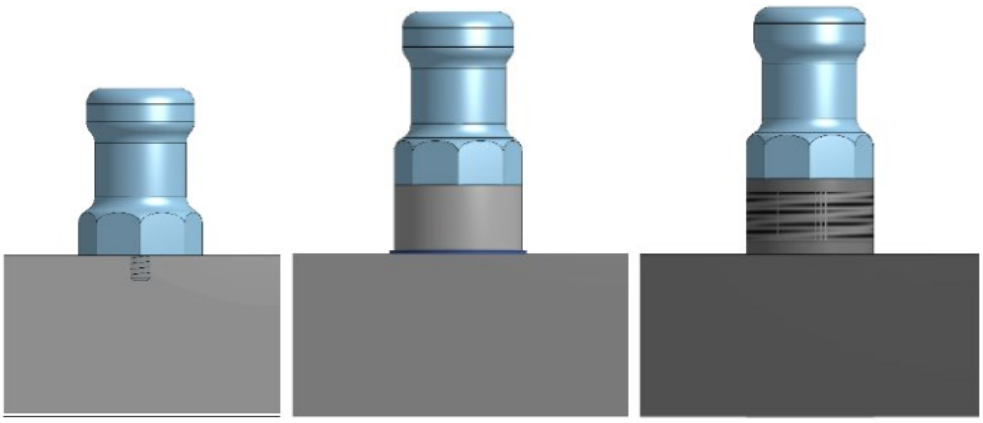
2.6 nmas chess -anturi

Nomen kehittämä nmas chess IIoT -järjestelmään tarkoitettu langaton anturi tukee seuraavia ominaisuuksia:

- aikatason lähetys

- 2 vuoden akunkesto
- näytteistaajuus 12 000 Hz
- tarkka pintalämpötilan mittaus (12).

Anturi on kiinnitettävissä joko M8-ruuviliitoksella suoraan laitteen runkoon, liimaamalla tai myös lisätarvikkeena saatavalla tasomagneetilla. Jotta mittauksista saadaan mahdollisimman luotettavaa dataa, tulee anturin kiinnityskohta olla tasainen pinta, jolla on anturin pohjaa laajempi ala. Mittapisteet valitaan siten, että anturi ja värähtelyn lähde ovat mahdollisimman lähellä toisiaan. (13.) (Kuva 6.)



KUVA 6. nmas chess -anturin kiinnitykset (13)

2.7 SQL ja SQLite

SQL (Structured Query Language) on standardoitu ohjelmointikieli, jota käytetään relaatiotietokantojen hallintaan ja erilaisten toimintojen suorittamiseen niissä oleville tiedoille. SQL:n käyttötarkoituksia ovat mm. tietokantataulukko- ja indeksirakenteiden muuttaminen, tietorivien lisääminen, päivittäminen ja poistaminen. SQL:n yleisimmät kyselyt ovat select, add, insert, update, delete, create, alter ja truncate. SQL:stä tuli relaatiotietokantojen yleisimmin käytetty ohjelmointikieli niiden ilmaantumisen jälkeen 1970-luvun lopulla ja 1980-luvun alussa. (14.)

SQLite on tietokantamoottori, jonka avulla käyttäjät voivat olla vuorovaikutuksessa relaatiotietokannan kanssa. SQLitessä tietokanta tallennetaan yhteen tiedostoon, eikä siinä ole erillistä palvelinsovellusta, mikä erottaa sen muista tietokantamoottoreista. Tietokannan käytön ja manipuloinnin edut ilman palvelinsovellusta ovat valtavat. SQLiteä käytetään maailmanlaajuisesti testaamiseen,

kehittämiseen ja kaikkiin muihin skenaarioihin, joissa on järkevää, että tietokanta on samalla levyllä sovelluskoodin kanssa. (15.)

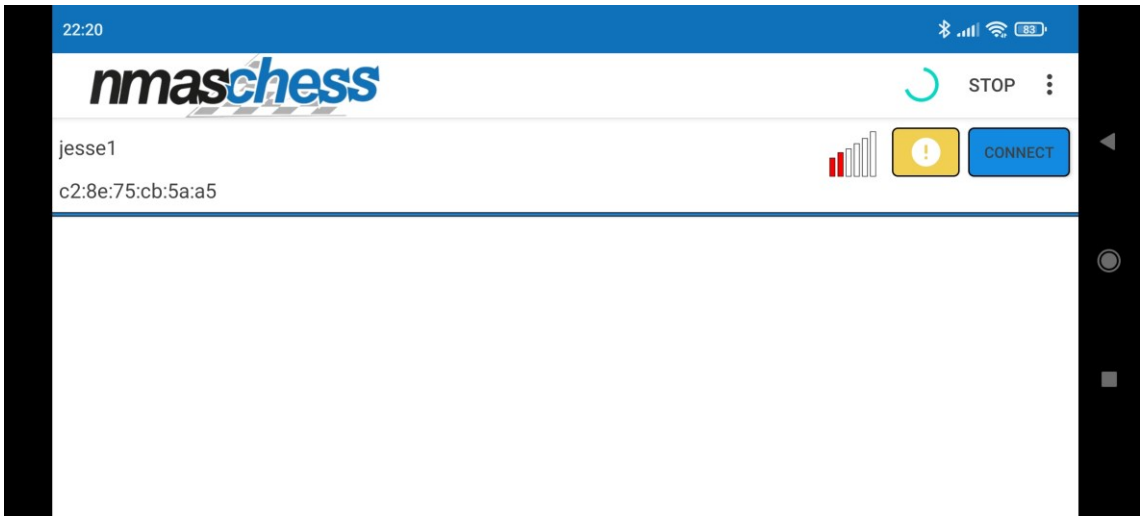
3 ANDROID-SOVELLUS

Reittimittaussovellus langattomalle anturille toteutettiin käyttämällä Android Studio -kehitysympäristöä sekä Java-ohjelmointikieltä. Sovelluksen kehityksessä ja testauksessa käytettiin apuna Android -mobiililaitteita sekä Nomen tarjoamaa langatonta nmas chess -anturia. Sovellus määriteltiin toimimaan mobiililaitteessa, jossa on Android 8.0 tai uudempi käyttöjärjestelmä.

Sovellus toteutettiin kolmella eri aktiviteetilla, joista ensimmäisessä on lista antureista, jotka ovat laitteen Bluetooth-kantaman sisällä. Kun käyttäjä valitsee haluamansa anturin listalta ja painaa Connect-painiketta, avautuu toinen aktiviteetti, jossa reittimittaus ja datan analysointi on tarkoitus suorittaa. Reittimittaus eli värähtelymittaus suoritetaan valitsemalla mittauspiste Nomen pilvitietokannasta tulevalta listalta ja painamalla mittausnappulaa. Mittauksen valmistuttua mitattu data lähetetään REST-kutsulla käyttäjälle määritettyyn pilvitietokantaan. Sovelluksen ollessa offline-tilassa mitattu data tallennetaan laitteessa sijaitsevaan paikalliseen SQLite-tietokantaan ja lähetetään pilvitietokantaan sitten, kun laite seuraavan kerran yhdistyy verkkoon. Sovelluksen SQLite-tietokantaan tallennetaan myös tietoa vanhemmista mittauksista, jotta datan analysointi onnistuu myös offline-tilassa. Kolmannessa aktiviteetissa käyttäjä voi kirjautua sisään sovellukseen nmas chess IIoT -järjestelmän käyttöön tarkoitetuilla tunnuksilla. Sisäänkirjautuminen on toteutettu REST-kutsulla. REST-kutsut sovelluksessa on toteutettu käyttäen Volley-kirjastoa.

3.1 Aloitusaktiviteetti

Sovelluksen ensimmäisessä aktiviteetissä on lista Bluetooth-kantaman päässä olevista nmas chess -antureista, joilla reittimittaus on tarkoitus suorittaa. Anturi on tunnistettavissa anturiin kiinnitetystä tarrasta, jossa lukee anturin MAC-osoite. Kuvassa 7 on anturin kohdalla sille määritetyn mittapisteen nimi sekä MAC-osoite. Keltaisesta napista on tarkoitus lisätä anturin käyttöoikeudet sisään kirjautuneen käyttäjän omistajalle, esimerkiksi jollekin yritykselle. Sinistä nappia painamalla avautuu toinen aktiviteetti, jossa kyseiseen anturiin muodostetaan yhteys ja jossa reittimittaukset on mahdollista suorittaa.



KUVA 7. Sovelluksen aloitusnäkymä

Jotta Bluetooth-ominaisuudet saadaan sovelluksessa käyttöön, on sovellukselle annettava lupa käyttää ja löytää Bluetooth-laitteita. Luvat otetaan käyttöön lisäämällä ne projektin AndroidManifest.xml -tiedostoon.

Sovellukseen lisättävät luvat ovat

- BLUETOOTH: Antaa sovelluksen muodostaa yhteyden Bluetooth -laitteisiin
- BLUETOOTH_ADMIN: Antaa sovelluksen etsiä ja muodostaa yhteyden Bluetooth -laitteisiin
- ACCESS_FINE_LOCATION: Android M (6.0) ja sitä uudemmat versiot edellyttävät sijaintilupaa, jotta sovellus saa BLE-skannaustulokset. (16.)

BLE-mainospaketteja vastaanottaakseen täytyy käyttää Android BLE API:ssa olevaa BluetoothLeScanner-luokkaa. Luokka tarjoaa menetelmiä skannaukseen liittyvien toimintojen suorittamiseen Bluetooth LE -laitteilla.

Kuvassa 8 näkyvässä funktiossa reagoidaan kuvassa 7 näkyvään START/STOP-painikkeeseen. Painiketta painamalla skannaus joko käynnistyy tai pysähtyy.

```
private void scanLeDevice(final boolean enable, boolean buttonPress) {
    final BluetoothLeScanner bluetoothLeScanner = mBluetoothAdapter.getBluetoothLeScanner();

    if (enable) {
        mScanning = true;
        ScanSettings.Builder builder = new ScanSettings.Builder();
        builder.setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY);
        builder.setMatchMode(1);

        bluetoothLeScanner.startScan( filters: null, builder.build(), mLeScanCallback);
    } else {
        if (!buttonPress) {
            mScanning = false;
            bluetoothLeScanner.stopScan(mLeScanCallback);
            mLeDeviceListAdapter.clear();
        } else {
            mScanning = false;
            bluetoothLeScanner.stopScan(mLeScanCallback);
        }
    }
    invalidateOptionsMenu();
}
```

KUVA 8. Bluetooth-skanneri

Bluetooth-skannerille annetaan parametreinä ScanSettings-olio, jolle asetetaan yhteyden kannalta optimaaliset asetukset, sekä ScanCallback-olio. Suodatukseen tarkoitettu ScanFilter-olio jätetään pois parametreistä, sillä BLE-laitteiden suodatus tapahtuu tässä tapauksessa ScanCallback-oliossa (kuva 9) sijaitsevassa if-lauseessa, jossa tarkistetaan havaitun laitteen valmistajan tunniste eli manufacturer ID.

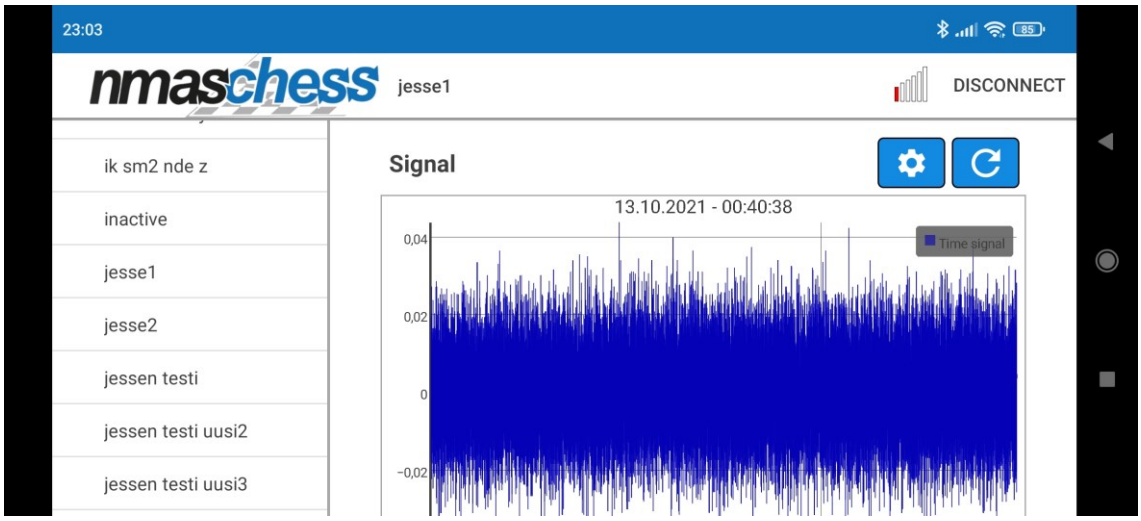
```
private ScanCallback mLeScanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, final ScanResult result) {
        ScanRecord record = result.getScanRecord();
        SparseArray<byte[]> manufacturerSpecificData = record.getManufacturerSpecificData();
        for (int i = 0; i < manufacturerSpecificData.size(); i++) {
            manID = manufacturerSpecificData.keyAt(i);
        }
        if (manID == nomeID){
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    mLeDeviceListAdapter.addDevice(result);
                    mLeDeviceListAdapter.notifyDataSetChanged();
                }
            });
        }
    }
};
```

KUVA 9. Skannerissa parametrinä käytetty ScanCallBack-olio

3.2 Reittimittausaktiviteetti

Kun anturi on valittu aloitusnäytössä ja Connect-nappia painetaan, aukeaa reittimittausaktiviteetti, jossa anturin eli oheislaitteen ja Android-mobiililaitteen eli keskuslaitteen välille muodostetaan yhteys. Yhteyden muodostamisen ohella haetaan lista mittapisteistä joko REST-kutsulla yrityksen pilvitietokannasta, jos laite on yhdistettynä verkkoon, tai paikallisesta SQLite-tietokannasta, jos verkkoa ei ole saatavilla. Kun mittapiste valitaan listalta, haetaan myös siihen kuuluvat mittausdatat edellä mainituin tavoin, jonka jälkeen mittausdata visualisoidaan eri graafeissa.

Kuvassa 10 näkyy vasemmalla lista mittapisteistä, jotka on haettu yrityksen tietokannasta. Oikealla näkyvästä päivitysnapista painamalla tehdään uusi värähtelymittaus anturilla. Ratasnapista päästään muokkaamaan anturin asetuksia sekä vaihtamaan mittapisteelle määritettyjä varoitus- ja hälytysrajoja. Kuvan oikea puoli, jossa näkyy värähtelymittauksesta saatu signaalidata, on rullattavissa alaspäin ScrollView-tyyppisen FrameLayout-komponentin ansiosta. ScrollView sisältää neljä eri graafia, joissa on värähtelymittaus ja siitä laskettu FFT-muunnos sekä trendidataa mittauksista saaduille arvoille ja mitatuille lämpötiloille. Datan visualisointi on tehty käyttäen jjoe64:n GraphView-kirjastoa. FFT-muunnos on tehty käyttäen Apache Commons Math-kirjaston FastFourierTransformer-luokkaa.



KUVA 10. Reittimittausaktiviteetti

Anturin isännöimän GATT-palvelimen yhteyden ja tietoliikenteen hallintaan on tehty Service-tyyppinen luokka nimeltä BluetoothLeService, joka käyttää hyväkseen useita eri luokkia Bluetooth API:sta, kuten BluetoothGatt, BluetoothManager ja BluetoothAdapter. Service on sovelluskomponentti, jonka tehtävä on suorittaa pidempiaikaisempia toimintoja ilman käyttäjän vuorovaikutusta. Service saattaa jopa jatkaa toimintaansa jonkin aikaa, vaikka käyttäjä on siirtynyt toiseen sovellukseen. (17.)

BluetoothGatt tarjoaa toiminnon, joka mahdollistaa kommunikoinnin Bluetooth Smart- tai Smart Ready -laitteiden kanssa. BluetoothManager on korkean tason hallintakomponentti, jota käytetään BluetoothAdapter-ilmentymän hankkimiseen ja yleiseen Bluetooth -hallintaan. BluetoothAdapterin avulla voidaan suorittaa perustavanlaatuisia Bluetooth-tehtäviä, kuten aloittaa laitteiden löytäminen, hakea lista liitetyistä laitteista, muodostaa BluetoothDevice eli ilmentymä laitteesta MAC-osoitteen avulla, sekä luoda BluetoothServerSocket-olio muiden laitteiden yhteyspyyntöjen kuunteluun ja Bluetooth LE -laitteiden etsimiseen. (18.)

BluetoothLeService on bound-tyyppinen Service, mikä tarkoittaa sitä, että se toimii niin kauan, kun se palvelee toista sovelluskomponenttia, joka on tässä tapauksessa reittimittausaktiviteetti, eikä se toimi taustalla loputtomasti. Jotta BluetoothLeService saadaan käyntiin, on sitä kutsuttava abstrak-

tin Context-luokan funktiolla bindService(), joka tarvitsee parametriksi ServiceConnection-rajapintaluokan. ServiceConnection valvoo yhteyttä BluetoothLeServiceen käyttäen callback-funktioita onServiceConnected() sekä onServiceDisconnected(). (Kuva 11.)

```
// Code to manage Service lifecycle.
private final ServiceConnection mServiceConnection = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName componentName, IBinder service) {

        mBluetoothLeService = ((BluetoothLeService.LocalBinder) service).getService();
        if (!mBluetoothLeService.initialize()) {
            Log.e(TAG,  msg: "Unable to initialize Bluetooth");
            connectProgressDialog.dismiss();
            finish();
        }
        connectProgressDialog = getDialogProgressBar( title: "Connecting...").create();
        connectProgressDialog.show();
        mBluetoothLeService.connect(mDeviceAddress);
```

KUVA 11. BluetoothLeServicen alustus ja Bluetooth-yhteyden muodostaminen

BluetoothLeService-luokka sisältää abstraktin luokan nimeltä BluetoothGattCallBack, joka toteuttaa callback-funktiot GATT-tapahtumille, jotka tapahtuvat reittimittausaktiviteetissä. BluetoothGattCallBack-luokka reagoi tapahtumiin kuten yhteyden luonti ja katkaisu, GATT-palvelun löytö, GATT-ominaisuuksien luku ja kirjoitus sekä signaalin vahvuuden luku. Lauetessaan callback-funktiot lähettävät reittimittausaktiviteetille takaisin broadcast-tyyppisiä lähetyksiä, jotta aktiviteetti pystyy reagoimaan kyseisiin BLE GATT-tapahtumiin. Jotta reittimittausaktiviteetti pystyy vastaanottamaan BluetoothLeService-luokan broadcast-lähetykset, täytyy aktiviteettiin luoda ilmentymä BroadcastReceiver-luokasta, jossa broadcast-lähetykset otetaan vastaan ja toteutetaan tarpeelliset operaatiot.

Kun oheis- ja keskuslaitteiden välille on muodostettu yhteys, laukeaa BluetoothGattCallBack-luokassa callback-funktio nimeltä onConnectionStateChange() (kuva 12), jossa käsitellään tarpeelliset operaatiot yhteyden luontia ja katkeamista varten. Kun yhteys on muodostettu, suoritetaan oheislaitteelle GATT-palvelun haku (Service discovery) käyttäen BluetoothGatt-luokan funktiota discoverServices(), joka etsii laitteelta siihen määritetyt palvelut. Tässä tapauksessa palveluita on vain yksi. Kun palvelu on löydetty, saadaan siitä haettua siihen kuuluvat ominaisuudet.

```
@Override
public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
    String intentAction;
    if (newState == BluetoothProfile.STATE_CONNECTED) {

        intentAction = ACTION_GATT_CONNECTED;
        mConnectionState = STATE_CONNECTED;
        broadcastUpdate(intentAction);
        Log.i(TAG, msg: "Connected to GATT server.");
        // Attempts to discover services after successful connection.
        Log.i(TAG, msg: "Attempting to start service discovery:" +
                mBluetoothGatt.discoverServices());

    } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
        intentAction = ACTION_GATT_DISCONNECTED;
        mConnectionState = STATE_DISCONNECTED;
        Log.i(TAG, msg: "Disconnected from GATT server.");
        broadcastUpdate(intentAction);
    }
}
```

KUVA 12. onConnectionStateChange()-funktio

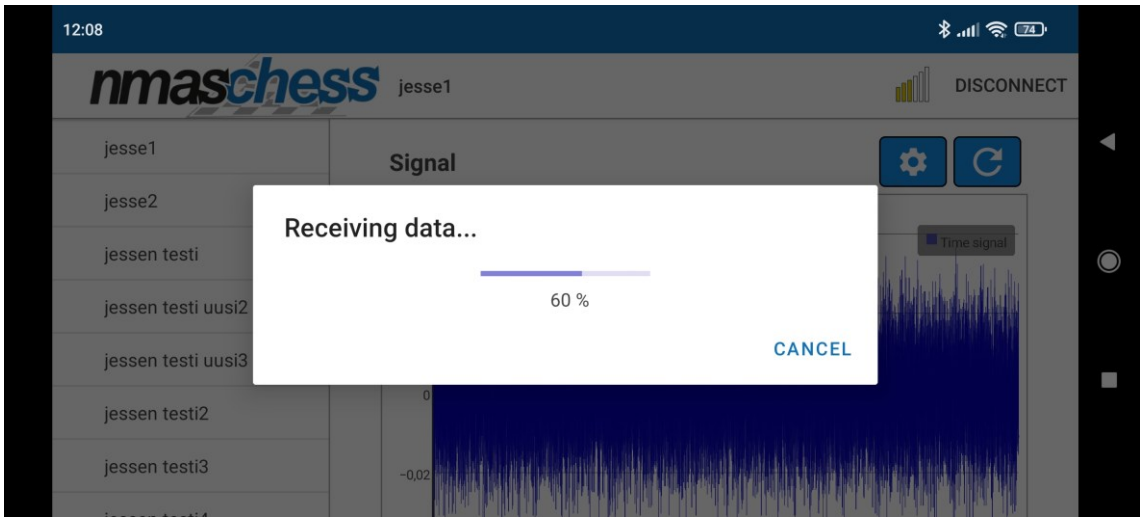
Värähtelymittausdatan lähetys anturilta mobiililaitteelle onnistuu käyttäen BLE-ilmoitusta (Notification). Ilmoitus on keino, jolla BLE-oheislaite ilmoittaa keskuslaitteelle, kun ominaisuuden arvo muuttuu. Anturilla on värähtelymittaukselle tarkoitettu ominaisuus, joka ilmoituksen aktivoituessa alkaa lähettämään vastaanottajalle dataa. (Kuva 13.)

```
mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);
Handler handler = new Handler(Looper.myLooper());
handler.postDelayed(new Runnable() {
    public void run() {
        BluetoothGattDescriptor descriptor = characteristic.getDescriptor
                (UUID.fromString(uuid));
        descriptor.setValue(enabled? BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
                : new byte[]{0x00, 0x00});
        mBluetoothGatt.writeDescriptor(descriptor);
    }
}, delayMillis: 100);
```

KUVA 13. GATT-ominaisuuden ilmoituksen aktivointi

Kun värähtelymittauksen ominaisuuden ilmoitus on aktivoitu ja ominaisuuden arvo muuttuu anturilla, laukeaa BluetoothLeService-luokassa funktio onCharacteristicRead() jonka if-lausekkeessa

tarkistetaan, onko kyseessä värähtelymittauksen ominaisuus. Jos on, niin lähetetään siitä broadcast-lähetys takaisin reittimittausaktiviteetille, jossa on mukana ominaisuuden uusi arvo. Tämä toistuu niin kauan, kunnes anturi on lähettänyt kaiken datan mittauksesta (kuva 14). Kun kaikki data on saatu vastaan, tehdään reittimittausaktiviteetilla tarvittavat operaatiot datan lähetystä, tallennusta ja visualisointia varten.



KUVA 14. Värähtelymittausdatan vastaanotto anturilta mobiililaitteelle

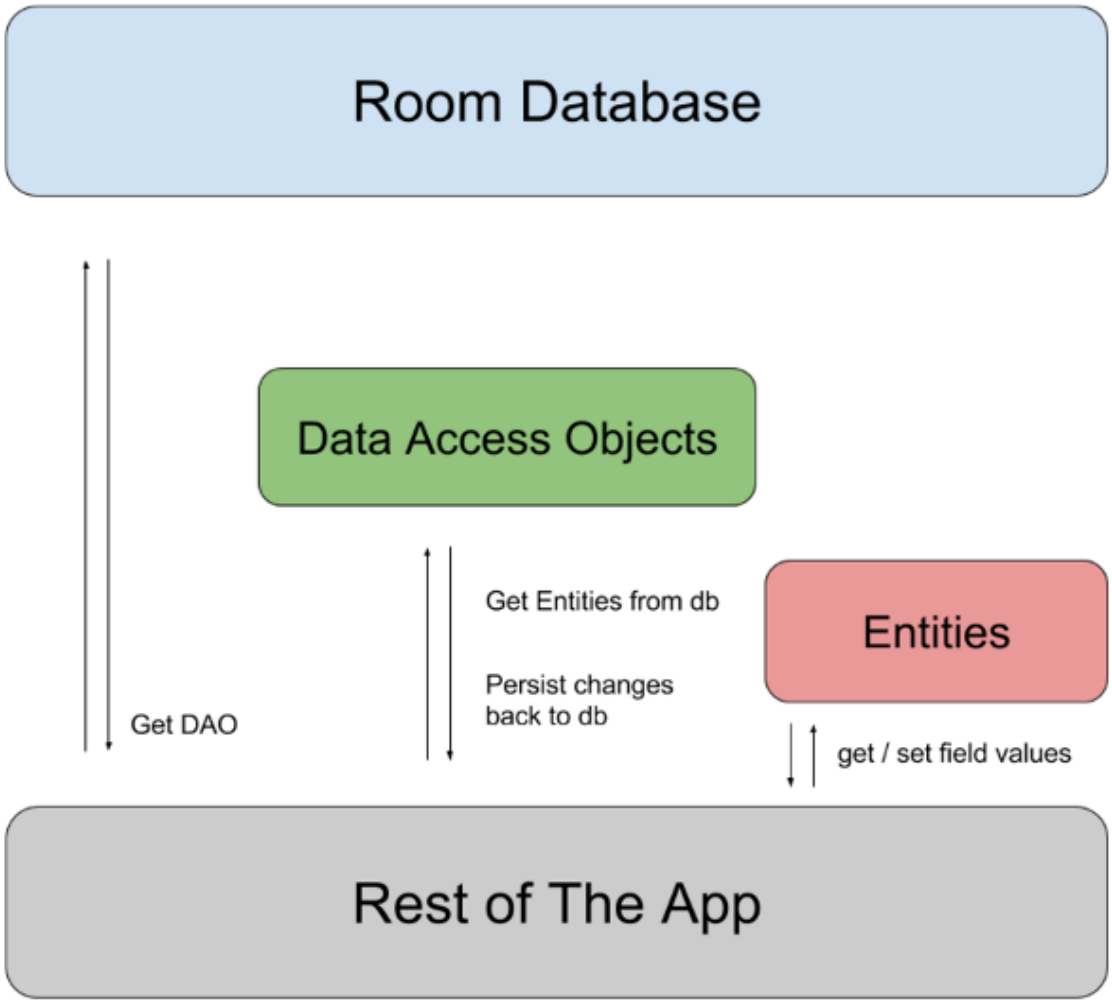
3.3 SQLite-tietokanta ja Room-kirjasto

Jotta sovellusta pystyy käyttämään offline-tilassa, luotiin sovellukseen paikallinen tietokanta käyttäen Room-kirjastoa, joka mahdollistaa tehokkaamman tietokannan käytön ja hyödyntää SQLiten täyden tehon tarjoamalla sille ylemmän tason abstraktiokerroksen. (19.)

Roomissa on kolme pääkomponenttia:

- Tietokantaluokka, joka sisältää tietokannan ja toimii päätukiasemana sovelluksen tietoihin.
- Entiteetit (Entity), jotka edustavat taulukoita sovelluksen tietokannassa.
- Data Access Objektit (DAO:t), jotka tarjoavat menetelmiä, joita sovellus voi käyttää tietokannan tietojen kyselyyn, päivittämiseen, lisäämiseen ja poistamiseen. (20.) (Kuva 15.)

Tietokantaluokka tarjoaa sovellukseen ilmentymiä tietokantaan liittyvistä DAO:ista. Sovellus voi käyttää DAO:ita tietojen hakemiseen tietokannasta. Tietokannan haku voi palauttaa joko yksittäisiä muuttujia tai kokonaisia entity-olioita. (20.)



KUVA 15. Diagrammi Room-kirjaston arkkitehtuurista (20)

Sovellukseen tehtiin luokka nimeltä MyDatabase, joka edustaa Room-kirjaston tietokantaluokkaa. Tietokantaluokalle on määritetty siihen kuuluvat entiteetit sekä luotu ilmentymät sovelluksen käyttämistä DAO:ista. (Kuva 16.)

```
@androidx.room.Database(entities = {LocationEntity.class, TimeSignalEntity.class,
        TrendDataEntity.class, TypeEntity.class}, version = 1, exportSchema = false)
public abstract class MyDatabase extends RoomDatabase {

    public static final String NAME = "Database";

    public int GPEAK = 1;
    public int ENVELOPE = 2;
    public int VRMS = 3;
    public int WARNING = 4;
    public int ALARM = 5;
    public int BATTERY = 6;
    public int TEMPERATURE = 7;

    public abstract LocationDao locationDao();
    public abstract TypeDao typeDao();
    public abstract TrendDao trendDao();
    public abstract TimeSignalDao timeSignalDao();
    public abstract  RawDao rawDao();
}
```

KUVA 16. Room-tietokantaluokka

LocationEntity-luokassa (kuva 17) määritetty taulukko sisältää listan mittapisteistä. TimeSignalEntity-luokassa määritetty taulukko sisältää viimeisempien värähtelymittauksien tietojen lisäksi kaiken värähtelymittausdatan, mitä ei ole vielä pystytty lähettämään REST-kutsulla pilveen. TrendDataEntity-luokassa määritetty taulukko sisältää kaiken mittauksiin liittyvän trendidatan. Trendidata on eroteltu toisistaan sen mukaan, minkä tyyppistä dataa on kyseessä. TypeEntity-luokassa määritetty taulukko sisältää trendidataan liittyvät datatyypit.

```
@Entity(tableName = "locations", indices = {@Index(value = {"location_name"}, unique = true)})
public class LocationEntity {

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "location_id")
    public int locationId;

    @ColumnInfo(name = "location_name")
    public String locationName;
```

KUVA 17. LocationEntity-luokka

Tietokannan luontia varten on tehty luokka nimeltä DatabaseApplication, joka perii Application-luokan. Application-luokka on globaali luokka sovelluksen tilan ylläpitämiseen. DatabaseApplication-luokan onCreate()-funktiossa luodaan objekti tietokannasta käyttäen Roomin databaseBuilder()-funktiota, jonka voi aktiviteettiin myöhemmin kutsua funktiolla getMyDatabase(). getMyDatabase()-funktion sisällä kutsutaan funktiota populateTypesData(), jonka tehtävä on esitäyttää tietokannan types-taulukko, kun tietokanta luodaan ensimmäisen kerran.

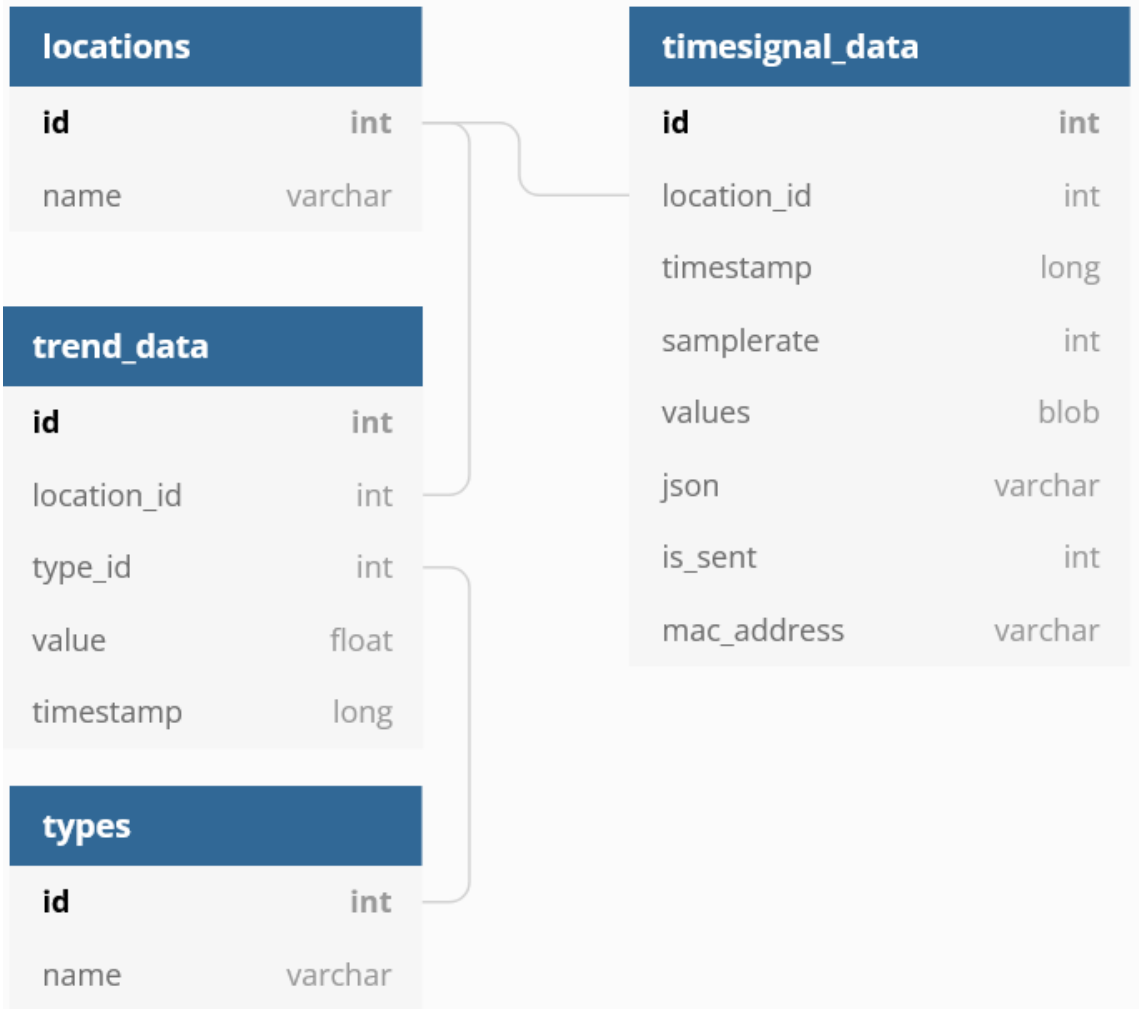
Jotta DAO:ihin pääsisi käsiksi, täytyy ensiksi hakea DatabaseApplication-luokassa luotu tietokantaluokan olio. Olion hakuun tarkoitettuun funktioon getMyDatabase() pääsee käsiksi DatabaseApplication-contextin avulla.

Kun tietokantaluokan olio on haettu, voidaan sen kautta kutsua DAO:ita, joilla voidaan kutsua DAO:iden sisällä määritettyjä SQL-kyselyfunktiota (kuva 18). Tietokannasta haetaan dataa, jos sovellus on offline-tilassa. Tietokantaan lisätään dataa, kun uutta on saatavilla, ja poistetaan dataa sitä mukaan, kun datan määrä ylittää tietokannan koodissa määritetyt rajat. Koska Room ei salli SQL-kyselyjä kutsuttavaksi pääsäikeellä, kutsutaan ne sovelluksessa eri säikeissä käyttäen Executor- sekä Thread -luokkia.

```
@Dao
public interface LocationDao {

    @Query("SELECT * FROM locations where location_id = :id")
    public LocationEntity getById(int id);

    @Query("SELECT location_id FROM locations where location_name = :name")
    public int findIdByName(String name);

    @Query("SELECT location_name FROM locations")
    public List<String> getAllLocations();

    @Query("INSERT INTO locations (location_id, location_name) VALUES (null, :locationName)")
    public void insertLocation(String locationName);

}
```

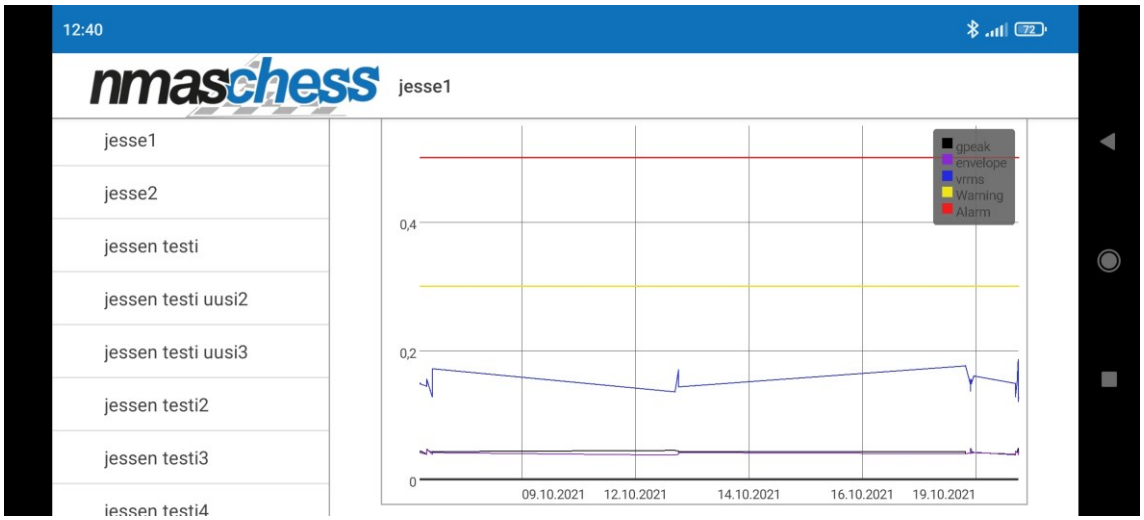
KUVA 18. Mittapisteden SQL-kyselyihin liittyvä LocationDao-rajapintaluokka

Kuvassa 19 nähdään sovelluksessa käytetyn tietokannan suhdekaavio. Locations tarkoittaa mittapisteitä, ja jokainen värähtelymittausdata sekä trendidata jaotellaan mittapisteen mukaan. Trendidata jaotellaan lisäksi sen mukaan, minkä tyyppistä dataa on kyseessä.



KUVA 19. Sovelluksen SQLite-tietokannan suhdekaavio

Kuvassa 20 nähdään paikallisesta SQLite-tietokannasta haettua trendidataa ”jesse1” -nimiselle mittapisteelle käyttäen TrendDao-rajapintaluokkaa.



KUVA 20. Paikallisesta SQLite-tietokannasta haettua trendidataa

Kuvan 20 trendidata on haettu käyttäen kuvan 21 SQL-kyselyä, jolla etsitään kaikki data, jotka liittyvät kyselyn parametreissä annettuihin arvoihin vanhimmasta uusimpaan.

```
@Query("SELECT * FROM trend_data where trend_location_id = :locationId and trend_type_id = :typeId ORDER BY trend_timestamp ASC")
public List<TrendDataEntity> selectAll(int locationId, int typeId);
```

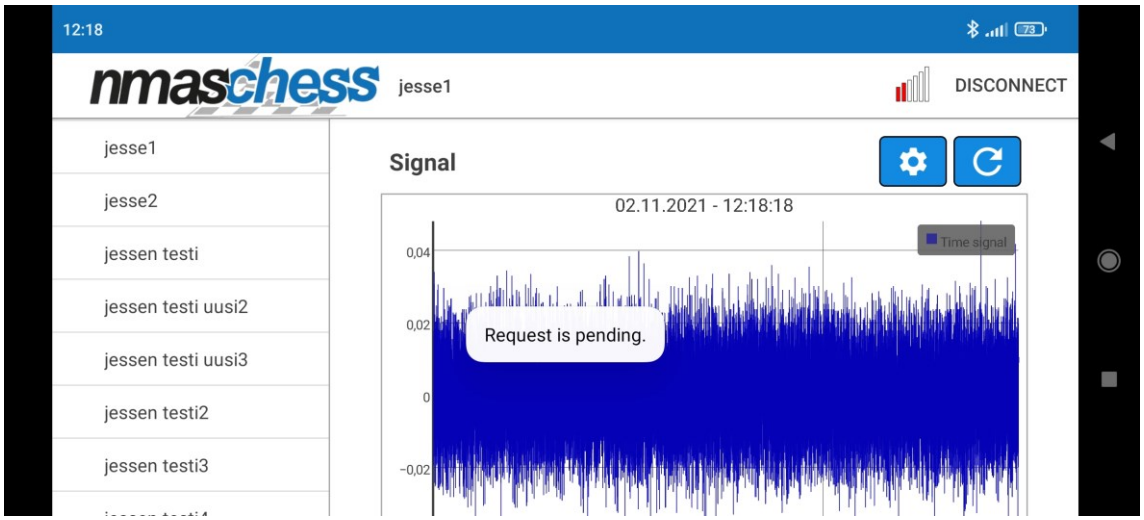
KUVA 21. Trendidatan hakuun tarkoitettu SQL-kyselyfunktio

Värähtelymittausdataan tarkoitetut kyselyt suoritetaan TimeSignalDao-rajapintaluokassa. Värähtelymittausdatan tallentamiseen on käytetty kuvan 22 SQL-kyselyä.

```
@Query("INSERT INTO timesignal_data (timeSignalId, timesignal_location_id, timesignal_timestamp, timesignal_rate," +
        " timesignal_values, timesignal_json, timesignal_is_sent, timesignal_mac_address) " +
        "VALUES (null, :locationId, :timestamp, :rate, :values, :json, :isSent, :address)")
public void insertTimeSignalData(int locationId, long timestamp, int rate, byte[] values, String json, boolean isSent, String address);
```

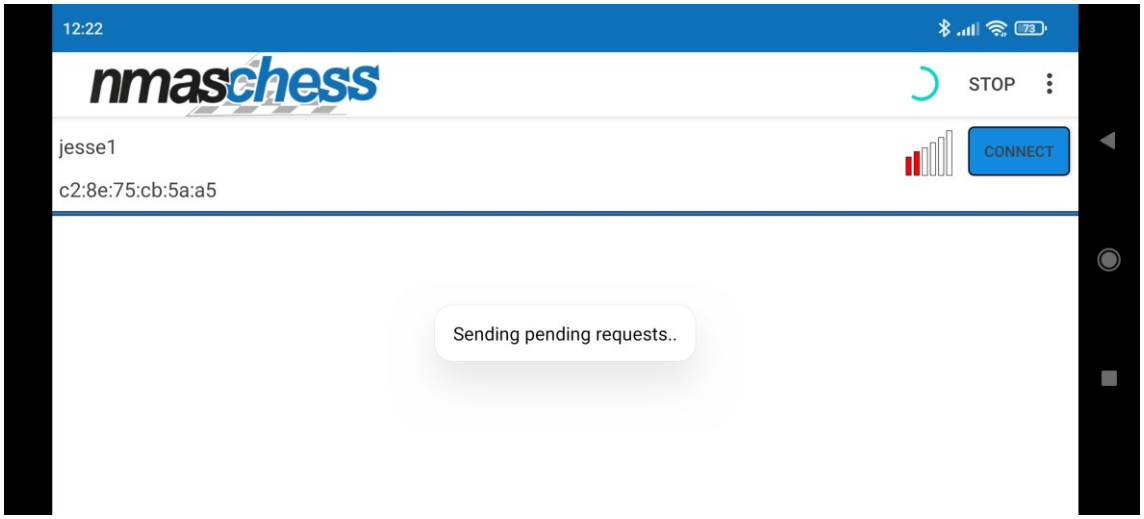
KUVA 22. Värähtelymittausdatan tallentamiseen tarkoitettu SQL-kyselyfunktio

Kun värähtelymittaus on suoritettu sovelluksen ollessa offline-tilassa, tulee näytölle ponnahdusviesti ”Request is pending”, josta käyttäjä tietää, että mittausdataa ei lähetetty vaan se tallennettiin paikallisesti ja lähetetään myöhemmin, kun verkko on taas saatavilla ja käyttäjä kirjautuu sisään sovellukseen. (Kuva 23.)



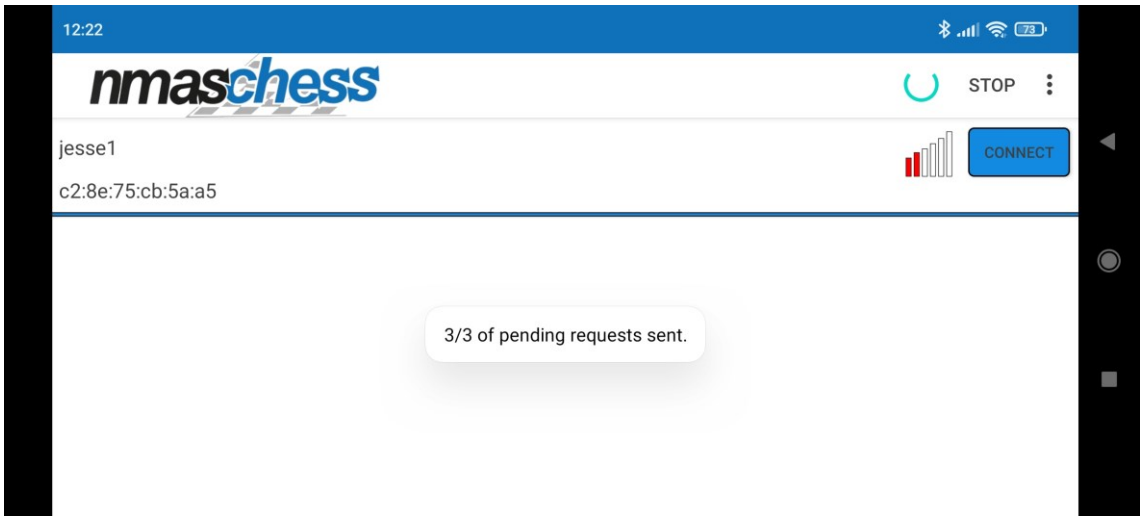
KUVA 23. ”Request is pending” -ilmoitus

Kun verkko on taas saatavilla ja käyttäjä kirjautuu sovellukseen sisään, tulee ruudulle ilmoitus ”Sending pending requests”, joka on merkki siitä, että sovellus pyrkii lähettämään tietokannassa olevat värähtelymittausdatat pilveen REST-kutsulla (kuva 24).



KUVA 24. ”Sending pending requests” -ilmoitus

Kun sovellus on käynyt jokaisen värähtelymittausdataan liittyvän REST-kutsun läpi, tulee ruudulle lopulta ilmoitus, jossa kerrotaan, kuinka monta lähetystä tietokannassa olevista paketeista meni läpi. (Kuva 25.)



KUVA 25. Värähtelymittausdatan lähetys onnistui

Onnistuneesti lähetetyt värähtelydatamittauspaketit poistetaan paikallisesta tietokannasta, kun taas epäonnistuneiden lähetysten paketit jäävät odottamaan seuraavaa kertaa. Värähtelymittausdatan poisto tietokannasta tapahtuu TimeSignalDao-rajapintaluokassa sijaitsevalla DELETE-tyyppisellä SQL-kyselyfunktiolla. (Kuva 26.)

```
@Query ("DELETE FROM timesignal_data where timeSignalId = :id ")
public void deleteById(int id);
```

KUVA 26. Värähtelymittausten poistoon tarkoitettu SQL-kyselyfunktio

4 POHDINTA

Työn tuloksena saatiin valmiiksi Android-mobiilisovellus, jolla pystyy suorittamaan värähtelymittauksia sekä vastaanottamaan mittausdataa Nomen kehittämältä nmas chess -anturilta Bluetooth Low Energy -teknologian välityksellä. Sovellus tallentaa mittauksista saatua dataa sekä REST-kutsuilla haettua mittausdataa sovelluksen sisäiseen SQLite-tietokantaan, mikä mahdollistaa sovelluksen käytön offline-tilassa. Lisäksi sovelluksella pystyy selaamaan ja analysoimaan aiemmin mitattua dataa. Sovellukseen voi ja täytyy kirjautua sisään, jotta sovelluksessa käytetyt REST-kutsut toimivat.

Työn alussa laaditut tavoitteet saavutettiin suurilta osin. Käytin työssä pohjana Nomelle aikaisemmin kehittämääni vastaavanlaista sovellusta, joka nopeutti sovelluksen valmistumista huomattavasti. Sovellus on muutamaa pientä ominaisuutta vaille täysin käyttökelpoinen.

Työn edetessä sain suuren määrän kokemusta Android-ohjelmien kehityksestä, Bluetooth Low Energy -teknologiasta sekä etenkin SQLite-tietokannan suunnittelusta ja käytöstä, josta aiempi kokemus ja osaaminen oli todella vähäinen. Kaiken kaikkiaan, onnistuin työssä mielestäni hyvin.

LÄHTEET

1. Nome 2021. Saatavissa: https://nome.fi/. Hakupäivä 2.9.2021.

2. Elprocus 2020. What is an Android Operating System & It's Features. Saatavissa: https://www.elprocus.com/what-is-android-introduction-features-applications/. Hakupäivä 6.9.2021.

3. Cardle, J. Paul 2016. Android App Development in Android Studio. Java + Android Edition for Beginners. Manchester Academic Publishers. Saatavissa: http://projanco.com/Library/Android%20App%20Development%20in%20Android%20Studio%20-%20Java%20plus%20Android%20edition%20for%20beginners.pdf. Hakupäivä 6.9.2021.

4. Valdellon, Lionel 2020. What is an SDK? Everything You Need to Know. CleverTap. Saatavissa: https://clevertap.com/blog/what-is-an-sdk/. Hakupäivä 6.9.2021.

5. Open Handset Alliance 2021. Intro. Saatavissa: https://developer.android.com/studio/intro/. Hakupäivä 24.06.2021.

6. GeeksforGeeks 2021. Introduction to Java. Saatavissa: https://www.geeksforgeeks.org/introduction-to-java/. Hakupäivä 25.06.2021.

7. Bluetooth. 2021. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Bluetooth/. Hakupäivä 25.06.2021.

8. Kamath, Sandeep & Lindh, Joakim 2012. Measuring Bluetooth Low Energy Power Consumption. Texas Instruments. Saatavissa: https://www.ti.com/lit/an/swra347a/swra347a.pdf?&ts=1589460592015/. Hakupäivä 26.06.2021.

9. Argenox Technologies LLC 2016. BLE Advertising Primer. Saatavissa: https://www.argenox.com/library/bluetooth-low-energy/ble-advertising-primer/. Hakupäivä 30.8.2021.

10. Townsend, Kevin 2021. Introduction to Bluetooth Low Energy. GAP. Adafruit Industries. Saatavissa: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap/. Hakupäivä 30.8.2021.

11. Townsend, Kevin 2021. Introduction to Bluetooth Low Energy. GATT. Adafruit Industries. Saatavissa: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt/. Hakupäivä 30.8.2021.

12. Nome 2021. nmas chess IIoT -järjestelmä. Saatavissa: https://nome.fi/product/nmas-chess-langaton-kunnonvalvontajarjestelma/. Hakupäivä 2.9.2021.

13. Nome 2021. Chess Käyttöohje. Saatavissa: https://nome.fi/wordpress/wp-content/uploads/2021/07/Chess_kayttoohje_web_A4_24062021-1.pdf. Hakupäivä 3.9.2021.

14. Sirkin, Jessica 2020. SQL (Structured Query Language). TechTarget. Saatavissa: https://searchdatamanagement.techtarget.com/definition/SQL. Hakupäivä 31.10.2021.

15. Ryzac Inc 2019. What is SQLite? Saatavissa: https://www.codecademy.com/articles/what-is-sqlite. Hakupäivä: 31.10.2021.

16. Ong, Chee Yi 2020. The Ultimate Guide to Android Bluetooth Low Energy. PunchThrough. Saatavissa: https://punchthrough.com/android-ble-guide/. Hakupäivä 12.10.2021.

17. Open Handset Alliance 2021. Services overview. Saatavissa: https://developer.android.com/guide/components/services. Hakupäivä 14.10.2021.

18. Open Handset Alliance 2021. Bluetooth overview. Saatavissa: https://developer.android.com/guide/topics/connectivity/bluetooth. Hakupäivä 14.10.2021.

19. Open Handset Alliance 2021. Room. Saatavissa: https://developer.android.com/jetpack/androidx/releases/room. Hakupäivä 28.10.2021.

20. Open Handset Alliance 2021. Save data in a local database using Room. Saatavissa: https://developer.android.com/training/data-storage/room. Hakupäivä 28.10.2021.