SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN ALA

ANDROID-SOVELLUS JÄÄNMIT-TAUSTUTKALLE

TEKIJÄ/T Riikka Rantonen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä Riikka Rantonen	
Työn nimi Android-sovellus jäänmittaustutkalle	
Päiväys 22.11.2024	28/0
Yhteistyötaho Savonia ammattikorkeakoulu Oy ja Mestar Kuopio Oy	

Opinnäytetyöprojektina oli kehittää Android-käyttöjärjestelmää tukeva sovellus, joka vastaanottaa tutkasta tutkan jäästä mittaamia arvoja, käsittelee ne sekä lopuksi analysoi ja esittää jäänpaksuuden tulokset reaaliajassa käyttäjälle. Käyttäjä pystyy luomaan keräämistään mittaustuloksista myös raportin, joka sisältää mittaustulokset ja paikkatiedon. Tarve sovellukselle syntyi, kun tutkan valmistajan tarjoama sovellus ei vastannut tarpeeseen saada mittaustulos jäänpaksuudesta reaaliajassa. Reaaliaikainen tieto jäänpaksuudesta vähentää mm. henkilö- ja laitevahinkoja lisäten turvallisuutta mittausvaiheessa.

Tuloksena syntyi sovellus, joka kerää, taltioi ja pystyy visualisoimaan analysoidun tiedon sekä luomaan raportin saaduista tuloksista paikkamerkintöineen. Lisäksi onnistuimme luomaan yksinkertaisen analyysialgoritmin, joka tekee datasta jonkinlaisen tuloksen jään paksuudesta. Tulevaisuudessa sovellus ja analyysialgoritmit kehittyvät testausmahdollisuuksien myötä. Jatkokehityksenä sovellusta on mahdollista laajentaa lisäämällä tukia erilaisille mittauspinnoille sekä laajentamalla analysointimenetelmiä.

Avainsanat
Android, mobiilisovelluskehitys, UDP, Kotlin, SQLite, Compose, reaaliaikainen

SISÄLLYS

SANASTO

API	Järjestelmärajapinta
Android	Linux-pohjainen älylaitteille suunnattu käyttöjärjestelmä
Arkkitehtuuri	Ohjelmistojärjestelmän rakenne
Instanssi	Yksittäinen esiintymä oliosta tai luokasta
Integraatio	Kahden järjestelmän yhdistäminen
Kotlin	Moderni ohjelmointikieli
Lint	Automaattinen koodin analysoija, joka ilmoittaa koodivirheistä.
Relaatiotietokanta	Tietokanta, jonka tauluilla on viite toisiin tauluihin
SQL	Standardi kieli tietokantojen hallintaan
UDP-protokolla	Yksi Internet-protokollatyypeistä

1 JOHDANTO

Toimeksiantaja Mestar Kuopio Oy on Kuopiossa sijaitseva, Kuopion kaupungin ja sen konserniyhtiöiden omistama osakeyhtiö. Yhtiö tarjoaa yhdyskuntarakentamis- ja kunnossapitopalveluja. Talvisin yhtiön tehtäviin kuuluu jään paksuuden mittaus Kallavedessä liikunta- ja ulkoilupaikoilla sekä niistä raportointi. Mittauksien keruu tapahtuu nykyiseltään käsipelin kairaamalla ja käsin mittaamalla. Tämä on erittäin aikaa vievää sekä turvatonta. Yhtiöllä on käytössä maamittaustutka, joka tekee heijastekuvaa maan alta. Tutka ja sen tarjoama mobiilisovellus ei kuitenkaan anna reaaliajassa jään paksuuden mittaa käyttäjälle, vaan tulokset saadaan vasta mittauksen jälkeen erillisestä ohjelmasta.

Toimeksiantajalta syntyi idea kehittää kyseiselle tutkalle sovellus, joka laskisi mahdollisimman reaaliajassa jään paksuutta tutkan mitatessa sekä varoittamaan käyttäjää ohenevasta jäästä. Tämä nopeuttaisi mittauksien tekoa huomattavasti sekä vähentäisi laite- ja henkilöstövahinkoja. Sovellus pystyisi myös luomaan raportin tehdystä mittauksesta, mikä vähentäisi manuaalista työtä entisestään.

Tässä opinnäytetyössä toteutettiin Android-sovellus, joka ohjaa tutkaa ja vastaanottaa siltä mittaustietoja. Tällä sovelluksella on pyrkimys korvata työläät vaiheet jään mittauksessa. Kehitysvaiheessa käytiin keskustelua toimijan ja kollegoiden kanssa työnetenemisestä ja suunnitelmista. Tulevaisuudessa sovelluksen jään paksuuden mittauksen algoritmit kehittyvät ja tulokset tarkentuvat testauksien myötä. Työskentelyssä korostuivat itsenäinen tiedonetsintä, suunnitelmien teko, kommunikointi sekä opintojenaikana ansaidut ohjelmistokehityksen taidot.

2 LÄHTÖKOHDAT

2.1 Toimeksianto

Säännöllisiä jäänpaksuuden mittauksia hoitavalla yrityksellä Mestar Kuopio Oy:llä on hankittuna maanläpimittaukseen tarkoitettu tutka. Tutkan kehittäjä on kehittänyt tutkalle mobiilisovelluksen, jolla tutkaa ohjataan ja joka kerää mittaustiedot tiedostoiksi paikkatietojen kanssa. Valmistajan mobiilisovellus ei kuitenkaan anna jäänpaksuuden arvoa suoraan reaaliajassa, eikä siten vastaa täysin jäänmittaajan tarpeita. Lisäksi sovellus sisältää paljon ominaisuuksia, joita jäänpaksuuden mittaamiseen ei tarvita, mikä tekee sovelluksesta monimutkaisen käyttötarkoitukseen nähden. Asiakas halusi mobiilisovelluksen, joka laskee tutkasta saaduista tiedoista jäänmittauksen tuloksen reaaliajassa käyttäjälle, varoittaisi ohenevasta jäästä sekä luo raportin jäänpaksuuden arvoista sekä niiden sijainnista. Opinnäytetyö rajattiin tiedonkeruun ja visualisoinnin ympärille, jolloin paksuuden analyysin tekevä algoritmi jäi erilliseksi osuudeksi opinnäytetyön ulkopuolelle.

Reaaliaikainen jäänpaksuudenmittaus lisää työturvallisuutta ja vähentää laitteistovahinkoja ilmoittamalla jään heikkenemisestä. Lisäksi sovelluksen raportointityökalu vähentää työvaiheita luomalla halutun raportin sovelluksesta.

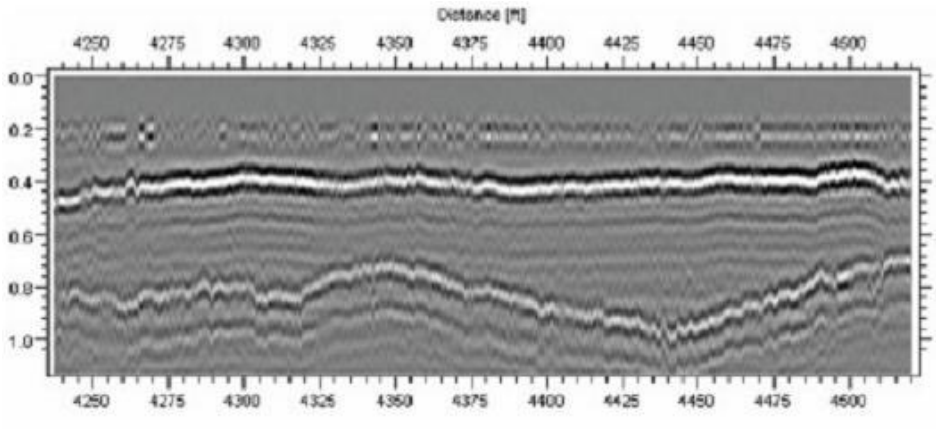
2.2 Tutkan ominaisuudet

Tutka lähettää impulsseja maahan ja nauhoittaa niistä syntyviä kaikuja. Näistä kaiuista pystytään matemaattisilla malleilla laskemaan mm. erilaisten materiaalien syvyyksiä.

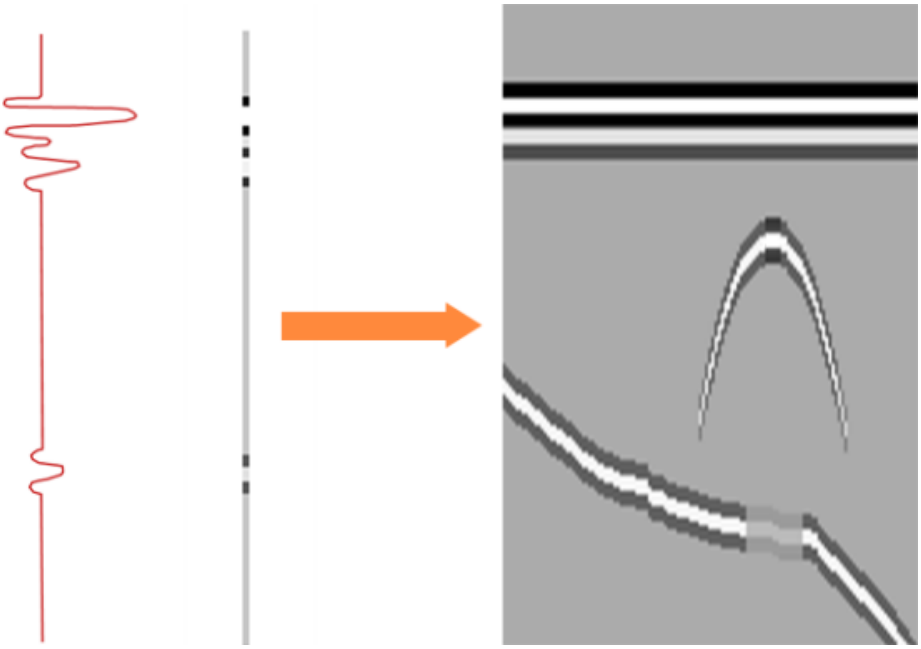
Tutka toimii samalla W-fi tukiasemana, jonka verkkoon mobiililaite yhdistetään kiinteällä IP-osoitteella sekä portilla. Kommunikointi laitteiden välillä tapahtuu UDP-protokollan kautta eli kaikki tutkalta lähtevät ja tutkalle saapuvat tiedot ovat tavujonoina. Tavujonojen purkamista ja lukemista varten kehitettiin luokkakirjaston, joka purkaa tavujonot olioiksi tunnistetun luokkamallin mukaan helpottamaan tiedon tulkitsemista.

Tutka lähettää yhdistetylle laitteelle heijastekuvaa (engl. Radargram) maan läpivalaisusta sekä GPS-tietoja reaaliajassa. Heijastekuva voidaan ilmaista mustavalkokuvana, jonka kukin väriarvo kuvaa näytteen arvoa (Kuva 1). Jokainen pitkittäinen pylväs heijastekuvassa voidaan ilmaista käyränä (Kuva 2). (DB Engineering & Consulting, n.d.). Sovelluksen tehtäväksi jää käsitellä, tallentaa ja esittää tieto käyttäjälle reaaliajassa.

Heijastekuva muodostuu jäljistä (engl. Trace). Yksittäistä pistettä jäljessä kutsutaan näytteeksi (engl. Sample) Jokainen näyte kuvastaa jollakin skaalalla, esimerkiksi -4000 – 4000, minkä vahvuisen heijasteen tutka on vastaanottanut.



Kuva 1. Esimerkki kuva heijastekuvasta.



Kuva 2. Esimerkki yhdestä jäljestä (engl. Trace). Yksi pikseli on yksi näyte, jolla on arvo. Mitä korkeampi arvo, sen vaaleampana se näyttäytyy heijastekuvassa sekä päinvastoin.

Koska mobiililaitteet tukevat lähtökohtaisesti vain yhtä verkkoyhteyttä eikä tutkassa ole mahdollisuutta datayhteyteen, sovellusta kehittäessä on ollut otettava huomioon offline-mahdollisuus erityisesti mittausvaiheessa.

2.3 Projektin rajaus ja määrittely

Projekti rajattiin siten, että siihen kuuluu käyttöliittymän luonti, mittauksen raakadatan keräys, niiden taltiointi, yhteyden ylläpito, mittauskertojen tallennus, paikkatietojen tallennus sekä analyysin tuloksen esittäminen. Koska analyysin teko vaatii syvempää ymmärrystä tilastoista, sen toteuttamista ei sisällytetty opinnäytetyöhön. Sovelluksen tuloksien yhteydessä tarkastellaan kuitenkin analyysimoduulia pintapuolisesti.

3 TEKNOLOGIAT

Sovelluskehityksessä käytettyjen teknologioiden joukkoa kutsutaan teknologiapinoksi (engl. Tech Stack). Jokainen kerros on rakennettu alla olevan päälle muodostaen pinon, mikä tekee teknologioista riippuvaisia toisistaan (Yaroslav Krutiak 2024). Tässä luvussa tarkastellaan mistä kaikista teknologioista sovellus on koottu.

3.1 Ohjelmointikieli

Lopputuotteen alusta määrittää pitkälti käytettävän ohjelmointikielen. Android-laitteelle yleisimmät ohjelmointikielet ovat Java, Kotlin ja C++, mutta tukee myös laajan kirjon muita kieliä.

Ohjelmointikieltä valittaessa on otettava huomioon mm. seuraavat seikat:

- Syntaksi ja oppimiskäyrä
- Suorituskyky ja tehokkuus
- Työkalut ja ekosysteemi
- Yhteisö ja tuki
- Alustan ja laitteen yhteensopivuus

Päädyimme Kotliniin sen ollessa Javaa modernimpi, tiiviimmän syntaksin omaava ohjelmointikieli. Sillä on myös laaja yhteisö ja se on edelleen jatkuvassa kehityksessä. Google on ilmaissut Kotlinin olevan virallinen kieli Android-kehitykselle, joten sen tuki tulee olemaan pitkään olemassa taaten sovellukselle pitkän elinkaaren ja ylläpidon (GeekForGeeks 2024). Kotlinin muita etuja ovat

- Koodin luettavuus
- Hyvä koodikääntäjä
- Tukee monialustaista kehitystä
- Java-yhteensopivuus

3.2 Tietokanta

Tietokannaksi valittiin Androidin natiivi relaatiotietokanta SQLite. Tietokanta on paikallinen, joten ulkoista taltiota ei tarvita. Paikallinen tietokantaratkaisu oli luontevin verkkoyhteyden rajallisuuden vuoksi. Tietokannan ylläpidon ja luonnin helpottamiseksi projektissa on käytössä Room-kirjasto.

Room-kirjasto luo paikallisen tietokannan ja hallitsee abstraktiotasoa helpottamaan kehitystä, jolloin SQLite API:n käyttöä ei tarvitse käyttää suoraan. Kirjasto sisältää ajonaikaisen SQL-lauseiden tarkastuksen, mikä vähentää huomattavasti virheitä. Kirjasto luo paikallisen tietokannan annetuilla määrityksillä ja palauttaa sen abstraktina luokkana. Tätä luokkaa kutsutaan niin sanotun sovelluskontin kautta, joka mahdollistaa yhden jaetun tietokantainstanssin. Tällä ratkaisulla tietokantainstanssia ei tarvitse alustaa jokaiselle näkymälle erikseen.

Tietokannan luonti tapahtuu luomalla abstraktiluokka, joka perii RoomDatabase-luokan (Kuva 3).

```kotlin
@Database(
    entities = [Trace::class, Project::class, GPS::class, Block::class, AnalysisResult::class],
    version = 11
)
abstract class TraceDatabase : RoomDatabase() {
    abstract fun traceDao(): TraceDao

    companion object {
        @Volatile
        private var INSTANCE: TraceDatabase? = null

        fun getDatabase(context: Context): TraceDatabase {
            return INSTANCE ?: synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    TraceDatabase::class.java,
                     name: "gpr_database"
                ).fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
```

Kuva 3 Esimerkki tietokantaluokasta, joka luo tietokannan.

3.3 Android Studio

Android Studio on virallinen, Googlen kehittämä IDE (Integrated Development Environment) eli integroitu kehitysympäristö Android-sovelluksille. Sen tuetut ohjelmointikielet ovat Java, Kotlin, Javascript sekä C++. Ensimmäinen versio julkaistiin yleiseen käyttöön 2014, jolloin se syrjäytti Eclipse ADT:n. (Android Developers 2024)

Pääominaisuuksia ovat

- Gradle-pohjainen kääntäjä
- virtuaalinen emulaattori
- GitHub-integraatio
- Lint-työkalut

Android Studio projektit koostuvat moduuleista, jotka ovat kooste lähdekoodeista sekä käännösskripteistä. Moduulirakenne helpottaa projektin osien erillisen testauksen ja käännöksen. Moduuleita voi olla useampi ja ne voivat olla itsenäisiä tai olla riippuvaisia toisistaan. Niitä voisi ajatella alaprojekteina. Moduulityypit ovat

- Android app (Android-sovellus)
- Feature (ominaisuus)
- Library (kirjasto)

3.4 Versionhallinta ja Git

Versionhallinta (engl. Version Control) ylläpitää ohjelmiston kehityksen haaroja sekä ohjelman konfiguraatioita. Versionhallinnassa jokainen muutos on jäljitettävissä. Näin voidaan tarvittaessa, kuten

ongelmatilanteissa, palata koodissa viimeisimpään toimivaan versioon. Jokaisesta ohjelmiston uudesta ominaisuudesta luodaan kehitysvaiheessa oksa tai haara (engl. Branch), joka liitetään valmistuessaan päähaaraan (engl. Main Branch). Pääoksasta luodaan lopullinen, tuotantoon menevä ohjelma.

Versionhallinta mahdollistaa useamman henkilön samanaikaisen työn samassa projektissa. Kukin kehittäjä kehittää omaa kopiota ohjelmasta ja luo haaran omasta versiostaan. Kun ominaisuus valmistuu ja se on saanut hyväksynnän, ominaisuus voidaan päivittää pääoksaan (Michael Ernst 2024).

Git on POSIX-järjestelmille suunniteltu versionhallintajärjestelmä. POSIX (Portable Operating System Interface) on joukko standardoituja käyttöjärjestelmärajapintoja, jotka perustuvat Unix-käyttöjärjestelmään. Uusimmat POSIX-spesifikaatiot – IEEE Std 1003.1-2017 – määrittelevät vakiorajapinnan ja -ympäristön, jota käyttöjärjestelmä voi käyttää POSIX-yhteensopivien sovellusten käyttöön (Robert Sheldon 2022).

3.5 Kirjastot

Kirjastot ovat valmiiksi kirjoitettuja ohjelmia, jotka suorittavat tiettyä tehtävää. Näitä voi sisällyttää omaan sovelluskehitykseen. Tämä auttaa kehittäjää keskittymään kehittämänsä sovelluksen ydintehtävään. Kirjastot voivat olla kirjoitettu eri kielellä, kuin varsinainen kehityksessä oleva ohjelmisto (Sencha 2022).

Tätä sovellusta tehdessä kirjastojen valintaan vaikuttivat kirjaston ylläpito, kirjastojen suosio sekä suorituskyky. Tässä kappaleessa käydään läpi keskeisimpiä kirjastoja, joita sovellukseen on valittu käytettäväksi.

3.5.1 Room

Room tarjoaa SQLitelle abstraktiokerroksen, joka mahdollistaa sujuvan tietokannan käytön.

Room-kirjaston etuja ovat

- Yksinkertaisuus
- Kääntämisenaikainen turvallisuus
- Kevyt ja nopea
- Skaalautuvuus
- Migraatiot
- Virallinen Googlen tuki

Roomilla on kolme pääkomponenttia, joista tietokanta kootaan (Save data in a local database using Room 2024).

1. Entity
   Entiteetti on kuvaus yhdestä tietokannan taulusta. Se sisältää sarakkeet nimineen ja niiden tyypit, pääavaimen sekä mahdollisen vakioarvon. (Kuva 4)

```
@Entity(tableName = "Project")
data class Project(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    var name: String,
    val date: String,
    val dataMode: Int,
    val isOpen: Boolean = false,
    var additionalDescription : String = ""
)
```

Kuva 4. Entity, joka kuvailee Project-taulun.

2. DAO
   Lyhenne sanoista Data Access Object. DAO:ssa määritetään kaikki SQL-lausekkeet, joita sovellus voi käyttää kun tarvitaan tietokantakutsuja. Jokainen DAO sisältää metodit, jotka mahdollistavat pääsyn tietokantaan.

3. Database
   Luokka, joka perii RoomDatabase-luokan. Tähän luokkaan määritetään tietokannan käyttämät Entity:t sekä DAO:t

3.5.2 Compose

Jetbrains Compose on Googlen kehittämä käyttöliittymäkehys Kotlinille. Composen tapa luoda käyttöliittymiä on funktionaalista. Compose hyödyntää Kotlinin toiminnallisen ohjelmoinnin ominaisuuksia, mikä tekee käyttöliittymän luonnista suoraviivaisempaa. Compose tukee myös muuttujien tilanhallintaa, mikä yksinkertaistaa koodia entisestään. Tämän ansiosta käyttöliittymistä voidaan luoda dynaamisia vähemmällä koodilla (Kotlin for Jetpack Compose 2024). Käyttöliittymät voidaan luoda joko ainoastaan Composella tai sitä voi yhdistää XML Views-teknologian kanssa.

3.5.3 OSMDroid

OSMdroid on avoimen lähdekoodin Android-kirjasto, joka tarjoaa OpenStreetMapDatan käytön. Kirjasto korvaa Androidin natiivin MapView-luokan (OpenStreetMap Wiki 2024).

OpenStreetMap valikoitui sovelluksen kartaksi sen ollessa lisenssivapaa ja mahdollisti offline-käytön. Kirjasto lataa sovelluksen välimuistiin (engl. Cache) kartat, kun sillä on pääsy verkkoon. Näin kartta saadaan näkyviin myös silloin, kun mittausta tehdään.

4 SOVELLUSKEHITYKSEN TULOKSET

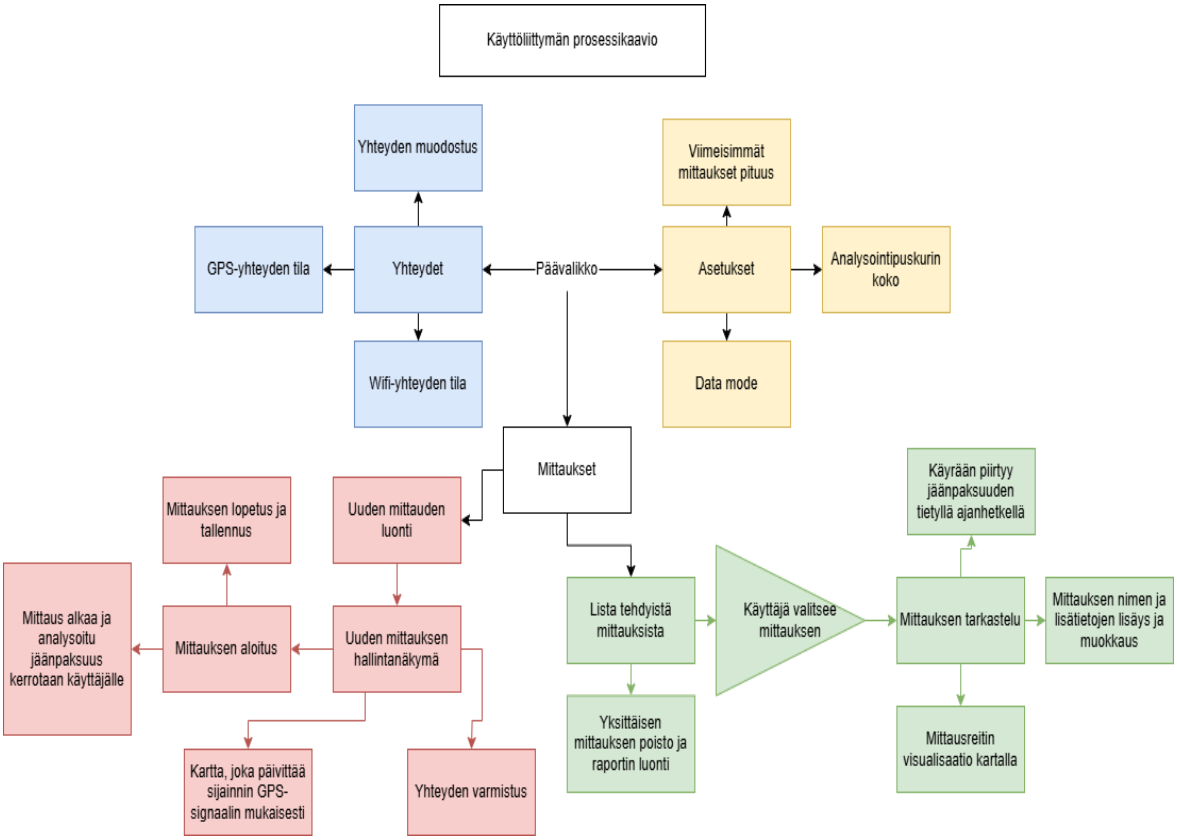
4.1 Prosessikaavio

Sovellusta suunniteltaessa pyrittiin mahdollisimman helppoon ja vaivattomaan käyttöön sekä loivaan oppimiskäyrään. Mittaukset tulisi saada tehtyä kentällä helposti ja vaivattomasti. Monimutkaisimmat asetukset on jätetty pois käyttäjän ulottuvilta, jotta ne eivät tekisi käyttöliittymästä liian monimutkaisia. Sovelluksen prosessikaavio kuvaa käyttöliittymän näkökulmasta (Kuva 5).

Sovelluksen avautuessa käyttäjälle aukeaa päävalikko. Päävalikosta käyttäjä voi valita jonkin seuraavista näkymistä: asetukset, mittaukset tai yhteydet.

Käyttäjän tulee aloittaa sovelluksen käyttö yhteyden muodostuksesta tutkaan. Sovellukseen on asetettu valmiiksi tarvittava osoite ja portti, joten käyttäjän ei tarvitse näistä huolehtia. Käyttäjä painaa ”Yhdistä”-painiketta, jolloin sovellus lähettää yhdistyspaketin tutkalle. Näkymä ilmoittaa käyttäjälle yhteyden tilasta tilan muuttuessa. Tällä näkymällä ilmoitetaan myös, onko tutkan GPS-vastaanotin saanut yhteyden satelliittiin.

Asetuksista käyttäjä valitsee asetukset tutkalle, analysoinnille sekä käyttöliittymälle. Tutkalle asetetaan data muoto riippuen, kuinka tarkan mittauksen tutkan tulee tehdä. Analyysiä varten valitaan analyysipuskurin koko. Tämä kertoo sovelluksen analyysimoduulille, kuinka suuresta otoksesta jäänpaksuuden tulkinta tehdään. Lisäksi mittausnäkymällä näkyvän käyrän pituus voidaan säätää tässä näkymässä.



Kuva 5. Sovelluksen prosessikaavio

Päävalikosta siirryttäessä Mittaukset-näkymään listaan piirtyy lista tehdyistä mittauksista. Kunkin mittauksen kohdalta käyttäjä voi valita mittauksen poiston, luoda raportin tai siirtyä tarkastelemaan mittaustietoja tarkemmin. Näkymässä on lisäksi painike uuden mittauksen luontiin.

Mittauksen tarkastelussa näytetään analysoidut jäänpaksuudet sekä mittauspisteiden sijainnit kartalla. Mittauksen nimeä ja kuvausta käyttäjän on mahdollista muokata.

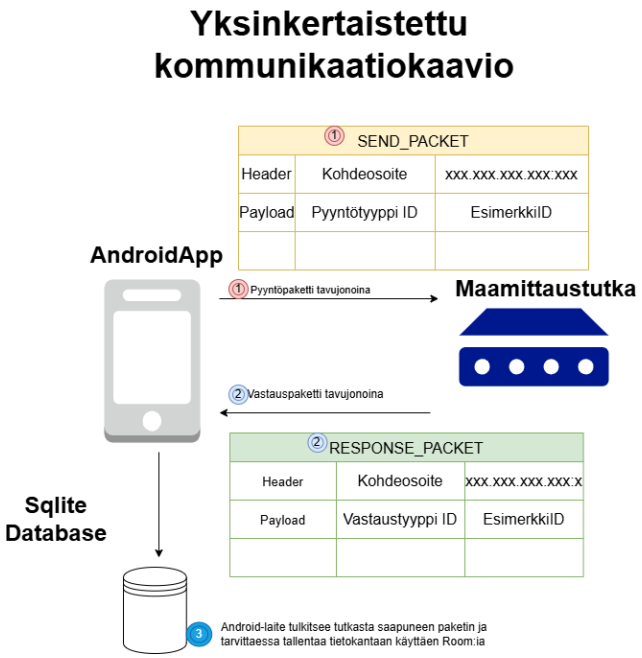
Uutta mittausta aloittaessa sovellus varoittaa käyttäjää, jos GPS-signaalia ei ole löytynyt. Tällöin GPS-tietoja ei voida tallentaa mittauksille. Käyttäjä voi käynnistää mittauksen painamalla ”Aloita mittaus”-painiketta, jolloin mittauksenaloituspaketti lähetetään tutkalle. Paketin vastaanoton jälkeen tutka lähettää mittaustieto- ja GPS-tietopaketit sovellukselle. Mittaustiedot lähetään analyysimoduulille, joka tulkitsee paketit ja palauttaa jäänpaksuuden arvon. GPS-data piirretään karttaan, josta käyttäjä voi katsoa kuljetun reitin reaaliajassa. Mittauksen voi keskeyttää käyttäjän valitsemalla ajanhetkellä.

4.2 Sovelluksen arkkitehtuuri

4.2.1 Tietoliikenne laitteiden välillä

Tutka sisältää oman Wi-Fi-tukiaseman, jonka tarjoamaan verkkoon Android-laite yhdistetään. Laitteiden välinen kommunikaatio toteuttaa UDP-protokollaa. Jotta laitteiden välinen kommunikaatio olisi mahdollista, Android-laitteen täytyy lähettää pyyntö tutkalle tutkan osoitteeseen oikeassa muodossa eli protokollan mukaisesti tavujono. Jos pyyntö onnistuu, tutka vastaa pyyntöön tietyssä tavujono muodossa. Tätä varten Android-sovellukselle on luotu luokkakirjasto, joka osaa purkaa tutkan lähettämät tavujonot objekteiksi helpottamaan tiedon käsittelyä. Android-sovellus käsittelee saapuneen tiedon ja tarvittaessa tallentaa tiedon tietokantaan (Kuva 6).

Molemman suuntaiset tavujonot sisältävät UDP-protokollan mukaisesti HEADER:in, jotka sisältävät kohde- ja lähtölaitteen tarvittavat tiedot, kuten kohde- ja lähtöosoitteen sekä portit. UDP-pakettien dataosuus sisältää myös tarvittavat tiedot, esimerkiksi komennon ID:n ja siihen liittyvät parametrit. Koska tutkan API-dokumentti on luottamuksellinen, ei sitä kuvailla tässä tarkemmin.

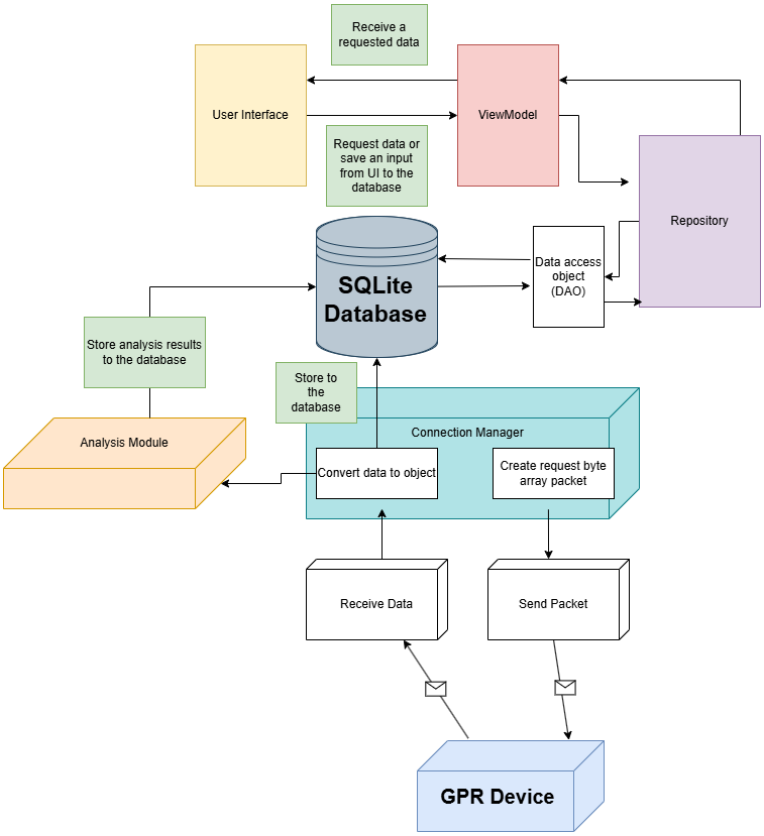


Kuva 6. Kommunikaatiokaavio laitteiden välillä.

4.2.2 ViewModel ja Repository Pattern Design

Sovellus noudattaa Repository Design Patternia (Kuva 7). Repository Pattern hallinnoi tietojen kulkua keskitetyssä paikassa, mikä vähentää koodin toistuvuutta ja parantaa ylläpidettävyyttä. Filosofiana on erottaa tiedonhankintalogiikka ja liikelogiikkakoodi erillisiksi osiksi ilman vahvoja riippuvuuksia. Esimerkiksi jos tietokanta muuttuu, arkkitehtuuria noudattamalla riittää tietokannan yhdistyslogiikan päivittäminen eikä muuta logiikkaa tarvitse muuttaa. (DevIQ 2024)

ViewModel luokka ylläpitää käyttöliittymän tiloja sekä toimii käyttöliittymän linkkinä tietokerroksen (engl. Data Layer) kanssa vastaanottavana ja lähettävänä osapuolena. ViewModelin etuihin kuuluu sen pysyvyys: Vaikka käyttöliittymällä tapahtuisi muutoksia, kuten näytön kääntyminen, käyttöliittymän ei tarvitse hakea tietoja uudelleen, vaan ViewModel muistaa jo sille asetetut arvot. (ViewModel overview 2024)



Kuva 7. Arkkitehtuurikaavio, kuinka tieto liikkuu laitteelta sovellukselle ja sovelluksen sisällä.

ViewModelille määritetään Repository-luokka, jota ViewModel käyttää. Repository on ViewModelista ensimmäinen taso data layerille. Tietokantaa käsiteltäessä käytetään tiedon saatavuus objektia (engl. Data Access Object) eli lyhyemmin DAO. DAO-luokat sisältävät tarvittavat metodit tietokantaan pääsyyn.

Tietokanta-luokassa määritetään abstraktifunktio projectDao() tyyppiä projectDao.

```
abstract class TraceDatabase : RoomDatabase() {
    abstract fun projectDao(): projectDao

    companion object {
        @Volatile
        private var INSTANCE: TraceDatabase? = null

        fun getDatabase(context: Context): TraceDatabase {
    …
    }
```

Repositorylle injektoidaan DAO interface. Näin Repository-lohko pystyy hyödyntämään DAO:ssa määritettyjä tietokantametodeja.

ProjectDao, joka hakee, luo ja poistaa projekteja:

```
@Dao
interface ProjectDao {

   // Get a project by name
    @Query("SELECT * FROM project WHERE project.id = :project_id LIMIT 1")
    suspend fun getProjectById(project_id : Long): Project

    // Get all projects and order by name
    @Query("SELECT * FROM Project ORDER BY name")
    suspend fun getProjects(): List<Project>


   // Insert a new project with a project object and return ID
    @Insert
    suspend fun insertProject(project: Project) : Long

    // Delete a project by ID
    @Query("DELETE FROM project WHERE id = :id")
    suspend fun deleteProject(id: Long): Int

…

}
```

Esimerkki Repositoryn koodista, jolle on injektoitu DAO ja käyttää DAO:n funktioita.

```
class ProjectRepository(private val projectDao: ProjectDao) : ProjectRepository {
 override suspend fun getProjects(): List<Project> = projectDao.getProjects()

    override suspend fun insertProject(project: Project) : Long = projectDao.insertProject(project)
    override suspend fun deleteProject(id : Long) = projectDao.deleteProject(id)
    override suspend fun getProjectById(projectId: Long): Project =
        projectDao.getProjectById(projectId)

…
}
```

Jotta DAO:n metodeja voidaan käyttää, luokalla tulee olla riippuvuus tietokantaan ja Context-luokkaan. Jotta näitä riippuvuuksia ei tarvitsisi Repository-luokalle parametrina lisätä, luotiin sovelluskontti (engl. Application Container), joka injektoi nämä Repository-luokalle. Kontti-luokka alustetaan sovelluksen Application-luokan sisällä, mikä on Androidissa perusluokka globaalin sovelluksen tilan ylläpitämiselle.

Kontin käyttö Application-luokassa. Kontti saa parametrina Application-luokan, joten tällöin kontilla on käytössä luokan ominaisuudet, kuten. Context-luokan.

```
class GprApplication : Application() {
    /**
     * AppContainer instance used by the rest of classes to obtain dependencies
     */
    lateinit var container: AppContainer

    override fun onCreate() {
        super.onCreate()
        container = AppDataContainer(this)

          …
    }
}
```

Kontin rajapinta (engl. Interface), sekä kontin luokka, joka perii sille luodun rajapinnan:

```
/**
 * App container for Dependency injection.
 */
interface AppContainer {
    val ProjectRepository: ProjectRepository
    val traceRepository: TraceRepository
}

/**
 * [AppContainer] implementation that provides instance of [OfflineGprProjectRepository]
 */
class AppDataContainer(private val context: Context) : AppContainer {
    /**
     * Implementation for [ProjectRepository]
     */
    override val ProjectRepository: ProjectRepository by lazy {
        ProjectRepository(TraceDatabase.getDatabase(context).projectDao())
    }
    override val traceRepository: TraceRepository by lazy {
        TraceRepository(TraceDatabase.getDatabase(context).traceDao())
    }

…
}
```

Repository-luokka saa nyt parametrikseen koko tietokantaluokan, jolle on annettu Context-luokka ja määritetty projectDao abstraktifunktio. Repository-luokan käyttö vaatii enää vain DAO:n parametrikseen. Tätä mallia kutsutaan Dependency Injection:iksi.

4.2.3 Connection Manager ja API

Connection Manager toimii keskitettynä luokkana sovelluksen ja tutkan välisen liikenteen ylläpitäjänä ja API:na. Yhteys noudattaa UDP-protokollaa. Luokka luo yhteyden, vastaanottaa ja lähettää viestejä ja seuraa laitteiden välistä viestintää. Vastaanotetut viestit puretaan vastaaviin luokkiin ja tehdään tarvittavat jatkotoimenpiteet, kuten tietokantaan kirjoittaminen.

Connection Manager luokan sisälle luotiin eräänlainen API. API:n kautta sovellus pystyy muodostamaan ja lähettämään oikean muotoisen tavujono paketin tutkalle. Esimerkiksi jos käyttäjä haluaa aloittaa mittauksen tekemisen, luokka lähettää start-komennon sisältävän paketin. Tutka osaa tulkita tämän ja vastaa lähettämällä sovellukselle mittauspaketteja.

UDP-yhteyden ollessa hyvin kevyt on sen haasteena yhteydettömyys. Laitteet eivät saa tietoa yhteyden olemassaolosta. Tätä varten API:lle toteutettiin ajastin, joka tietyn ajan välein tutkii, onko viestejä tutkalta saapunut. Jos aika ylittyy, voidaan todeta, että yhteyttä ei ole ja tästä ilmoitetaan käyttäjälle.

4.2.4 Analyysimoduuli

Analyysimoduuli on sovelluksessa erillinen luokka, joka käsittelee vastaanotettuja tutkan näytteitä, analysoi ne ja tallentaa tulokset tietokantaan. Sovelluksen vastaanotetut näytteet menevät puskurijonoon. Kun puskurijono täyttyy, näytteet siirtyvät analyysimoduulille. Näytteet esikäsitellään, kuten leikataan, lasketaan keskiarvot ja tehdään signaalin vahvistusta, ennen varsinaisen analyysin laskemista. Käsitellyistä näytteistä moduuli päättelee jään paksuuden. Moduuli on ensimmäinen versio jäänpaksuuden analyysin laskennasta ja tulee tulevaisuudessa kehittymään testauksien myötä.

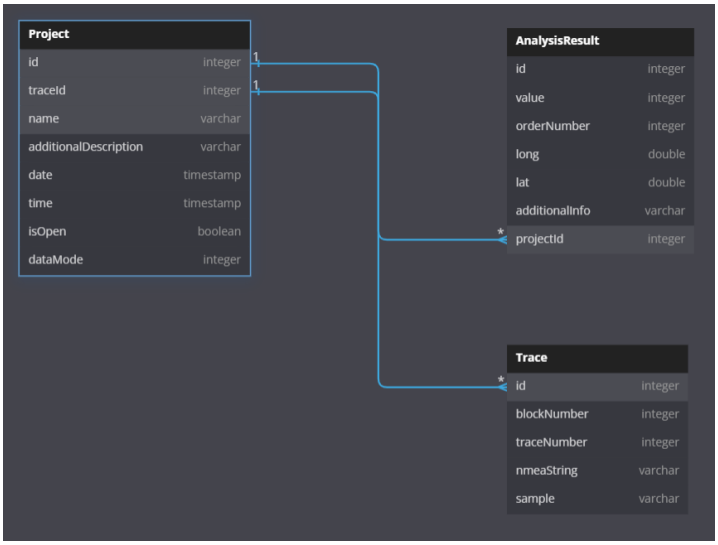
4.2.5 Raportin luonti

Raportin luonti osoittautui erääksi keskeisimmäksi lopputuotteeksi. Sovellus luo raportin CSV-muotoiseksi. CSV-taulukosta loppukäyttäjän on helppo jatkojalostaa tiedot esimerkiksi PowerBI:ssä.

Käyttäjä valitsee Mittaukset-näkymästä haluamansa mittauskerran, josta haluaa raportin luoda. Luo raportti -painikkeesta käyttöliittymä lähettää ViewModel:sta pyynnön Repository-luokalle, joka hakee tietokannasta AnalysisResult-taulusta rivit ID:n mukaan. Riveistä muodostuu lista, joka luodaan CSV:ksi. CSV-tiedostosta löytyvät sarakkeet ID, longitude, latitude, thickness (jäänpaksuus) sekä muut huomiot.

4.3 Tietokannan rakenne

Jokainen mittauskerta luo tietokantaan uuden rivin Project-tauluun. Sovellus tallentaa riviä luodessa päivämäärän (date-sarake), kellonajan (time-sarake) sekä generoi projektille nimen (name-sarake). DataMode sarakkeesta analyysimoduuli tietää, millä muodolla mittaus on tehty ja käsittelee mittaustulokset sen mukaan. IsOpen-sarake kertoo mittauksen tilan. Tällä varmistetaan mittaustietojen menevän oikean mittausprojektin riveihin. AdditionalDescription:iin tallentuu mahdollinen käyttäjän antama kuvaus tekstinä. Project-taulun pääavain toimii vierasavaimena muihin tauluihin. Näihin tauluihin suhde on yksi moneen (Kuva).



Kuva 7. Suunnittelutyökalulla tehty kaavio tietokannan tauluista ja niiden suhteista.

Kun sovellus vastaanottaa tutkalta mittaustietopaketin, sovellus tallentaa sen Trace-tauluun. Sample-sarakkeeseen tallentuu kaikki heijastekuvan yhden otoksen mittauspisteet. Analyysimoduuli laskee näistä mittauspisteistä jään paksuuden. Tauluun tallentuu mittauksen paikkatiedot NMEA-muodossa.

Lopuksi analyysimoduuli tallentaa käsitellyn tiedon AnalysisResult-tauluun. Kullekin tulokselle annetaan laskettu paksuus (taulussa value), longitudi (taulussa long) ja latitudi (taulussa lat) koordinaatit sekä mahdollinen muu tieto (taulussa additionalInfo). Tästä taulusta voidaan luoda CSV-raportti raportin luontivaiheessa.

4.4 Käyttöliittymä

Käyttöliittymässä on pyritty pelkistettyyn ulkoasuun. Mittauksien tapahtuessa paksuissa työvaatteissa sekä rankoissa olosuhteissa sovelluksen käytön tulee olla mahdollisimman vaivatonta ja suoraviivaista. Nämä ovat huomioitu mm.

- yksinkertaisilla käyttöliittymillä, joissa mahdollisimman vähän komponentteja ja tekstiä.
- isot kirjaimet ja näppäimet.
- kontrastit komponenttien välillä.

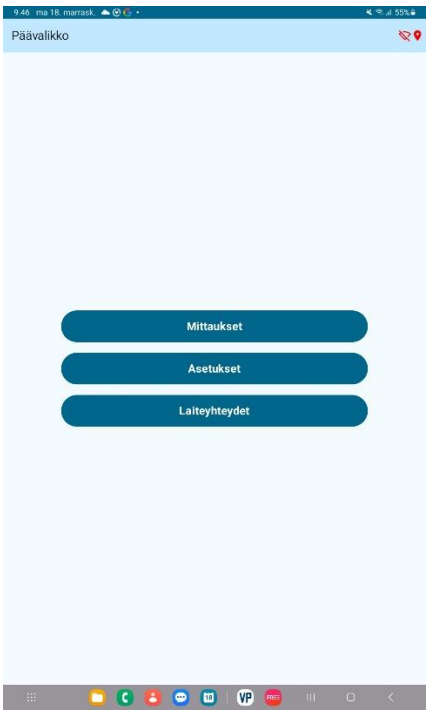
Mahdolliset säädökset ennen mittaukset voidaan toteuttaa sisätiloissa. Tällöin asetuksia ei tarvitse tehdä ulkona jäällä ennen mittausta. Itse mittaukseen pääsee nopeasti muutamalla painikkeen painalluksella.

Käyttöliittymä on toteutettu Compose-kirjastoa hyödyntäen. Komponenttipohjainen tapa luoda käyttöliittymiä helpottaa käyttöliittymän laajentamista, komponenttien riippuvuutta toisiinsa sekä vähentää toistuvaa koodia.

4.4.1 Päävalikko ja navigaatiopalkki

Sovelluksen yläreunassa sijaitsee navigointipalkki, joka näkyy käyttäjälle jokaisella näkymällä ja näyttää missä näkymässä käyttäjä käyttöhetkellä on. Navigointipalkista käyttäjä voi siirtyä edelliselle näkymälle nuolesta. Laite- ja GPS-yhteyden tilaa käyttäjä voi seurata reaaliajassa navigointipalkin ikoneista. Jos yhteys on muodostettu, ikoni on vihreä. Muodostamaton ja katkennut yhteys näkyy punaisena ikonina (Kuva 8).

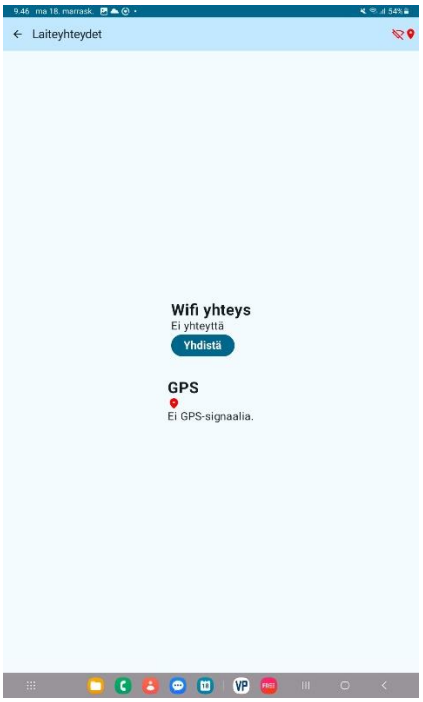
Päävalikkoon on suunniteltu isot näppäimet ja kirjaimet. Isot painikkeet helpottavat painikkeiden valitsemista paksummilla varusteilla, kuten toppahanskoilla. Päävalikko on myös hyvin yksinkertaistettu, jotta käyttäjä löytää nopeasti tarvitsemansa painikkeen.



Kuva 8. Sovelluksen päävalikko sekä navigointipalkki

4.4.2 Laiteyhteydet

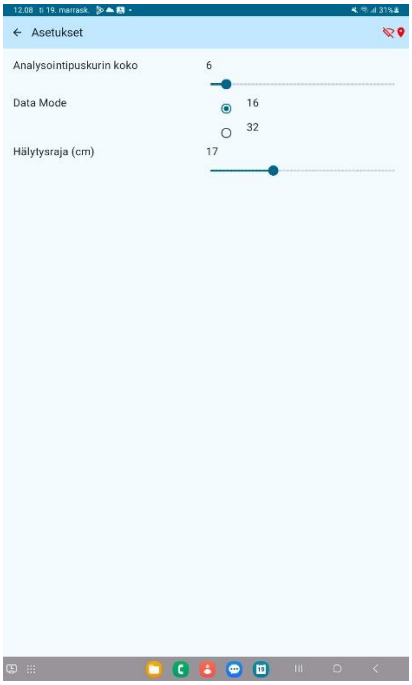
Laiteyhteydet -näkymällä käyttäjä voi seurata ja muodostaa laiteyhteyden. Yhteyksien alle muodostuu teksti yhteyden tilan mukaan. Värikoodatut ikonit nopeuttavat yhteystilan havaitsemista käyttäjälle, jossa punainen ikoni kertoo, että yhteyttä ei ole ja vihreä tarkoittaa, että yhteys on olemassa (Kuva 9). Tällä on huomioitu esimerkiksi lukihäiriöiset.



Kuva 9. Laiteyhteydet näkymä

4.4.3 Asetukset

Asetukset-näkymässä (Kuva 10) on kolme asetusta, joita käyttäjä voi säätää. Kun käyttäjä tekee muutoksia vaihtamalla elementtien arvoja, ne tallentuvat Android-laitteen sisäiseen Shared Preferences taltioon. Käyttäjän asettamat asetukset luetaan uudelleen näkymään, kun sovellus käynnistetään uudelleen.



Kuva 10. Asetukset-näkymä.

Analyysipuskurin koon skaala on 2–50. Koska skaala on tiukasti rajoitettu, syöttöelementiksi on valittu liu'uttaja (engl. Slider). Tästä käyttäjä havaitsee, kuinka suuri valittu arvo on koko skaalasta. Liukuelementin yläpuolella ilmoitetaan elementtiin asetettu arvo.

Data Mode kertoo tutkalle, kuinka tarkkaa tietoa tutka lähettää API:n yli sovellukselle. Jos käyttäjä ei ole tätä asetusta vielä asettanut, oletuksena on 16. Radiopainikkeet poissulkevat toistensa tilaa, joten käyttäjällä on aina vain toinen valittuna. Laitteelle asetettu tila näkyy aktiivisena radiopainikkeena. Asetuksen vaihtaminen lähettää tutkalle viestin uudesta asetuksesta, jolloin asetus tulee myös tutkalle voimaan.

Hälytysrajalla määritetään se paksuus, jolloin hälytys ilmoittaa mittaajalle liian ohuesta jäästä. Hälytysrajaa käyttäjä voi säätää liu'uttajalla. Elementin yllä oleva luku muuttuu valitun arvon mukaan, kuten analyysipuskurin kokoa säädettäessä.

4.4.4 Mittaukset-näkymä

Mittaukset-näkymässä (Kuva 11) käyttäjä voi poistaa aikaisempia mittauksia, tarkastella niitä tai luoda uuden mittauksen. Aikaisemmista mittauksista käyttöliittymä luo listan, missä kullekin mittaukselle käyttöliittymä piirtää nimen, päiväyksen sekä painikkeet poistoa, tarkastelua ja raportin luontia varten. Poiston ja raportin luonnin yhteydessä käyttäjältä kysytään ennen toimenpidettä varmistus, jotta peruuttamattomia vahinkopainalluksia ei tapahtuisi. Poistaessa mittausta, käyttöliittymä kutsuu

ViewModel-luokan sisälle määriteltyä deleteProject()-metodia. Room-kirjasto suorittaa DELETE SQL-lauseen ja poistaa myös kaikki riippuvuudet poistetusta projektista.

Kuva 11. Mittaukset näkymä

Tarkastele-painikkeesta käyttöliittymä pyytää tarkemmat tiedot mittauksesta tietokannasta. Valittu mittaus tallentuu ViewModelin tilaan aktiiviseksi projektiksi. Tästä tilasta seuraava näkymä renderöi ulkoasun oikeilla tiedoilla.

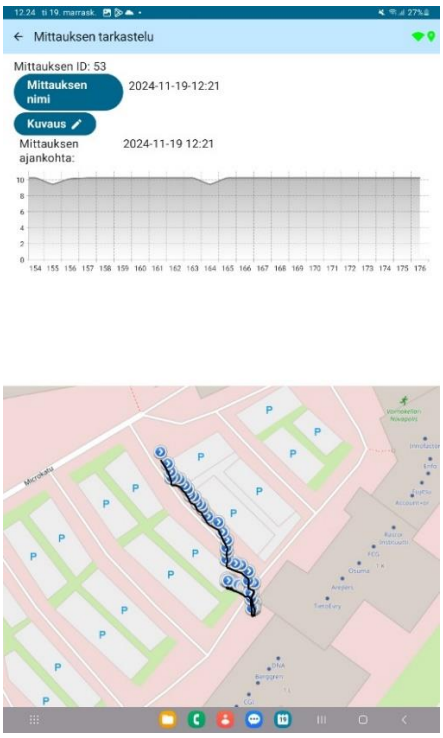
Uusi mittaus aloitetaan listan yläpuolella olevaa painiketta painamalla. Tässä myös huomioitu käyttäjä, jolla voi pakkasen vuoksi olla paksutkin käsineet mittausta tehdessä ja painikkeesta tehty leveä helpottamaan painamista. Painikkeesta painamalla sovellus siirtyy mittausnäkymään.

4.4.5 Mittauksen tarkastelu

Mittauksen tarkastelu -näkymä (Kuva 12) visualisoi valitun mittauksen reitin kartalla sekä jäänpaksuuden muutokset tiettynä ajanhetkenä. Painikkeista voidaan muokata projektin nimeä sekä antaa vapaaehtoinen kuvaus. Uudet tekstit tallennetaan tietokantaan.

Käyrästä käyttäjä voi seurata mittaushistoriaa. Y-akselissa on asteikko jäänpaksuudelle.

Karttaan piirtyy viivalla kuljettu reitti sekä mittauspaikat, johon jäänpaksuus on laskettu. Kartalle piirtyvät merkit kertovat merkin kohdalla olevan jäänpaksuuden.



Kuva 12. Mittauksen tarkastelu näkymä

4.4.6 Uusi mittaus

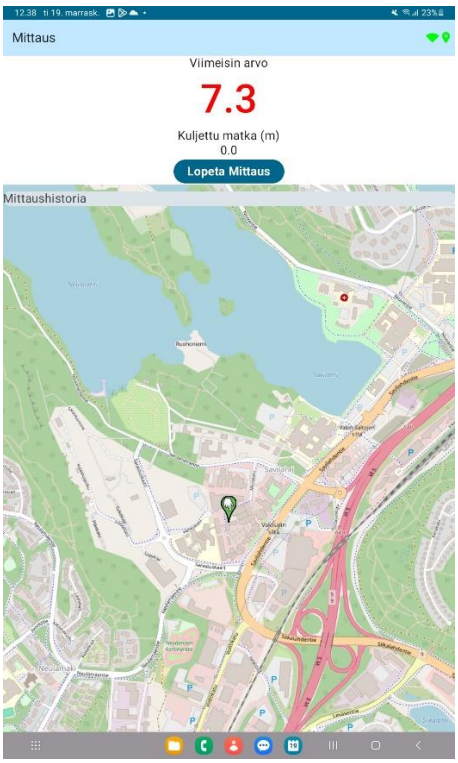
Mittausnäkymän alkuosassa näkyy mittauksenaloituspainike (Kuva 13). Painikkeesta painamalla sovellus lähettää tutkalle paketin, joka sisältää aloituskomennon. Paketin vastaanottaessa tutka alkaa lähettää mittaustietoja sovellukselle, jotka sovelluksen analyysipaketti käsittelee. Poistu-painike siirtää käyttäjän pois näkymästä, sillä käyttäjän poistuminen on estetty navigaatiopalkista, jotta mittauksien lähetys ei jäisi tutkalle päälle. Poistuminen lähettää tutkalle mittauksen lopetuspaketin, jolloin tutka keskeyttää mittauksen ja pakettien lähetyksen.

Jos käyttäjältä puuttuu GPS-signaali, siitä varoitetaan heti käyttäjää. Tämä ei kuitenkaan estä mittauksen tekoa, vaan mittaukset tehdään ilman paikkatietoja. Jos GPS-signaali löytyy, näkymän karttaan piirtyy käyttäjän nykyinen sijainti (Kuva 14). Mittauksen ollessa päällä, karttaan piirtyy viivalla kävelty reitti.

Mittaustiedot -paneelista painamalla käyttäjälle avautuu käyrä, voi seurata reitillä tehtyjä jään paksuuksia tietyn matkan välein. Mittaustiedot ovat tallennettu myös ViewModelin tilaan, joten käyttöliittymä ei tarvitse tehdä sovellusta kuormittavia kutsuja tietokantaan mittaushistorian saamiseksi.

Mittauksen lopettaessa tiedot tallentuvat tietokantaan ja käyttäjä ohjataan takaisin mittauksien listanäkymään. Kaikki mittauksen aikaiset prosessit ja tilat nollataan lähtöarvoihin.

Kuva 13. Mittausnäkymä



Kuva 14. Mittausnäkymä, jossa toimiva GPS-signaali.

5 JATKOKEHITYSEHDOTUKSET

Sovellukselle on luotu hyvät pohjat jatkokehitykselle. Tulevaisuudessa analyysialgoritmit tulevat sovelluksessa kehittymään testauksien myötä.

Tutkan ollessa maanmittaustutka olisi luonnollista, että sovellus voisi hyödyntää tulevaisuudessa tutkan ominaisuuksia laajemmin. Jotta tutka pystyy tulkitsemaan kaikukuvansa oikein, täytyy tutkalle lähettää materiaalin mukaiset asetukset, kuten aineen kiihtyvyys ja kuinka syvältä mitataan. Eri materiaaleille, kuten putkille, voisi tehdä valmiit profiilit vakioarvoilla, jolloin riittää, että käyttäjä valitsee halutun profiilin materiaalin mukaan, eikä arvo tarvitsisi manuaalisesti syöttää.

Karttareitin kopiointi reittiohjeeksi uudelle mittaukselle lisäisi jäänmittauksien tehokkuutta. Tällöin saataisiin myös historia selville, kuinka paksua jäätä on ollut samassa paikassa eri aikoina. Valmiiksi määritellyn reitin voisi näyttää kartasta käyttäjälle, jolloin käyttäjä ei tarvitse ulkoisia karttoja reitin seurantaan.

6 YHTEENVETO JA POHDINTA

Projektin aikana syntyi Android-sovellus, jonka tavoitteena on tarjota jäänmittauksen tekoon helpot työkalut mittausten tekemiseen. Vaikka sovellus on tässä vaiheessa vain tiedon kerääjä ja käyttöliittymän tarjoaja, sillä on potentiaalia tulevaisuudessa, kun analyysialgoritmit ja testaukset ovat saatu päätökseen.

Hyvä suunnittelu ja asioiden vertailu olivat avainroolissa tämän projektin onnistumiseen. Sovellus koostuu hyvin monesta eri osasta, joilla on tarkat omat tehtävät. Suunnitelmien laatiminen ja pohdinta kollegoiden ja asiakkaan kanssa vetivät selkeät rajat mitä tehtävää kunkin sovelluksen osan tulee toimittaa. Jokaisen osan ollessa itsenäinen kokonaisuus ja riippuvuuksien minimoimisen ansiosta päivittäminen ja lisäominaisuuksien laatiminen tulee olemaan vaivaton tulevaisuudessa.

Rinnakkaisuus tuli selkeästi esille. Prosesseja tapahtuu sovelluksessa samanaikaisesti paljon ja Android-laitteiden suorituskyky voi tulla herkästi vastaan. Modernien kielien, kuten Kotlinin, valinta tällaiseen projektiin oli toimiva ja takaa sovellukselle pitkän kehityskaaren sen ollessa jatkuvasti kehittyvä ja virallinen Android-ohjelmoinnin kieli.

Opinnäytetyössä haastavimmaksi osuudeksi nousi itse Android-kehitys. Android-kehitys poikkeaa perinteisemmästä ohjelmoinnista. Esimerkiksi Android Studio sisältää emulaattoria, koodieditoria, koodikääntäjää yms., jolloin jo itse kehitysalusta on erittäin raskas ja vaatia kehittäjältä pitkäpinnaisuutta muistin loppuessa. Android-projektit sisältävät myös paljon erilaisia moduuleita, konfiguraatioita ja muita skripti-tiedostoja. Omalle kohdalle Gradlen ymmärtäminen hidasti alussa projektin etenemistä. Lisäksi eri Java-versiot haastoivat tekijää, mutta onneksi kollegoista löytyi iso apu.

UDP-protokollan ja tutkan kanssa työskentely eivät ole arkipäivää tämän päivän tuoreelle ohjelmistokehittäjälle. Iloitsin erityisesti juuri näistä uusista aluevaltauksista ja integraatio Android-sovelluksen ja tutkan välillä osoittautui suorastaan innostavaksi. Oli jo projektin alkumetreillä ilo huomata, kuinka tärkeän tuotteen äärellä työskennellään. Toivottavasti tulevaisuudessa voisin hyödyntää tässä opinnäytetyöprojektissa hankkimiani tietoja ja taitoja.

LÄHTEET

Yaroslav Krutiak 2024. Verkkosivu. Viitattu 15.11.2024. https://spdload.com/blog/how-to-choose-a-tech-stack/

GeeksForGeeks 2024. Verkkosivu. Viitattu 11.11.2024. https://www.geeksforgeeks.org/kotlin-vs-java/

Michael Ernst 2024. Verkkosivu. Viitattu 15.11.2024. https://homes.cs.washington.edu/~mernst/advice/version-control.html

DB Engineerin Consulting n.d. Verkkosivu. Viitattu 15.11.2024. https://db-engineering-consulting.com/en/insights/ground-penetrating-radar/

Sencha 2022. Verkkosivu. Viitattu 11.11.2024. https://www.sencha.com/blog/difference-between-framework-vs-library-snc/

Android Developers 2024. Verkkosivu. Viitattu 11.11.2024. https://developer.android.com/studio/intro

Save data in a local database using Room 2024. Verkkosivu. Viitattu 10.11.2024. https://developer.android.com/training/data-storage/room

Kotlin for Jetpack Compose 2024. Verkkosivu. Viitattu 10.11.2024. https://developer.android.com/develop/ui/compose/kotlin

ViewModel overview 2024. Verkkosivu. Viitattu 10.11.2024. https://developer.android.com/topic/libraries/architecture/viewmodel

Robert Sheldon 2022. Verkkosivu. Viitattu 15.11.2024. https://www.techtarget.com/whatis/definition/POSIX-Portable-Operating-System-Interface

OpenStreetMap Wiki 2024. Verkkosivu. Viitattu 10.11.2024. https://wiki.openstreetmap.org/wiki/Osmdroid

DevIQ 2024. Verkkosivu. Viitattu 15.11.2024. https://deviq.com/design-patterns/repository-pattern