

Sinipelto

OPPIMISPÄIVÄKIRJA
COMP.CS.220 Mobiiliohjelointi 2021

Oppimispäiväkirja
Heinäkuu 2021

SISÄLLYSLUETTELO

1. AJANKÄYTTÖ	4
1.1 Ajankäytön taulukko	4
2. HARJOITUSTEHTÄVÄT	9
2.1 Harjoitus 1	9
2.2 Harjoitus 2	10
2.3 Harjoitus 3	11
2.4 Harjoitus 4	12
2.5 Harjoitus 5	13
2.6 Harjoitukset 6 & 7 & 8	14
2.7 Harjoitus 9	17
2.8 Harjoitus 10	18
2.9 Harjoitus 11	21
2.10 Harjoitus 12	23
2.11 Harjoitus 13	26
2.12 Harjoitus 14 & 15	29
2.13 Harjoitus 16	32
2.14 Harjoitus 17	35
2.15 Harjoitus 18 – Palautekysely	39
3. VERKKOAINESTOT	42
4. LAAJA HARJOITUSTYÖ	44
4.1 Harjoitustyön suunnitelma	44
4.2 Harjoitustyön design	46
4.3 Harjoitustyön toteutus	47
4.4 Harjoitustyön käyttöohjeet	53
LÄHTEET	71

KUVALUETTELO

Kuva 1.	<i>Kuvankaappaus mobiililaitteen käyttöjärjestelmän tiedoista.</i>	9
Kuva 2.	<i>Kuvankaappaus harjoitus 4:n ratkaisusta. Yksinkertainen summalaskin liukuluvuilla.</i>	13
Kuva 3.	<i>Harjoitustehtävän 5 suoluution kuvankaappaukset laskunäkymästä ja historianäkymästä.</i>	14
Kuva 4.	<i>Kuvankaappauksia applikaatiosta tehtäviä 6, 7 ja 8 varten.</i> <i>Kyseessä on mobiilisovellus laskujen hallintaan.</i>	17
Kuva 5.	<i>Kuvankaappaus sovellussivusta Google Play-kaupassa.</i>	17
Kuva 6.	<i>Kuva harj. 9 käyttöliittymästä. Visuaalisia muutoksia käyttöliittymäpuolelle ei ole tehty edelliseen tehtäväään nähdyn.</i>	18
Kuva 7.	<i>Kuvankaappaus Firestoren hallinnan WebUI:sta. Tiedot ovat näppärästi tallentuneet pilvikantaan, ja synkronoivat muutokset reaalialjassa.</i>	18
Kuva 8.	<i>Kuvankaappauksia autentikoinnista. Kirjautumisenäkymä. Esimerkinä Facebook-sisäänskirjautuminen ja uloskirjautuminen sovellukseen.</i>	21
Kuva 9.	<i>Kuvankaappaukset tehtävästä 11. Sensorien dataa sekä emulaattorilla, että omalla puhelimella. Selkeästi nähtävissä, että oikealla puhelimella on enemmän sensoreita käytettävissään.</i>	23
Kuva 10.	<i>Kuvankaappauksia harjoituksesta 12. Sovelluksessa esikatselu ja tallennetun kuvan näkymät. Lisäominaisuutena vaihto etu- ja takakameran välillä.</i>	26
Kuva 11.	<i>Kuvankaappauksia harjoituksesta 13. Ensimmäinen kuva on emulaattorista, loput omasta laitteesta. Emulaattorissa ei ollut askelmittauksen sensoreita saatavilla, joten siitä erillinen ilmoitus käyttäjälle.</i>	29
Kuva 12.	<i>Kuvankaappauksia harjoituksista 14 & 15. Sovelluksessa kaksi osaa, hallintpaneeli yläosassa ja dynaaminen karttanäkymä alaosassa. Kuvassa myös tallennetut 10 sijaintia näkyvissä.</i>	32
Kuva 13.	<i>Kuvankaappaukset harjoituksesta 16. Ensimmäinen kuva tilanteesta, jossa kurssien päivitys epäonnistunut virheen takia. Toisessa kuvassa onnistunut muunnos eri valuutoiksi.</i>	35
Kuva 14.	<i>Kuvankaappauksia harjoituksesta 17. Kaksi ensimmäistä kuvaaa esittävät virhetilanteita (ei internet-yhteyttä, paikkaa ei löytynyt). Kaksi jälkimmäistä esittävät tulokset kahdesta eri validista lokaatiosta.</i>	39
Kuva 15.	<i>Kuvankaappauksia harjoitustyön applikaation alustavasta designistä.</i>	47
Kuva 16.	<i>Karkea tietomallinnus sovelluksen käytämistä tietorakenteista.</i>	47

1. AJANKÄYTTÖ

Tässä osiossa on käsitelty kurssin ajankäytöötä kurssin suorituksen aikana.

1.1 Ajankäytön taulukko

Oheisessa taulukossa 1 on taulukoituna kurssin ajankäyttö päivätaasolla. Taulukko sisältää tiedot siitä, mihin asioihin on käytetty aikaa, minkä verran ja minä ajankohtana. Käytetyn ajan sarakkeessa käytetty aika on numeroina, jotta se on helpompi esimerkiksi viedä Excel-laskentaohjelmaan ja laskea erinäisiä tilastoja ajankäytöön liittyen.

Taulukko 1. *Toteutunut ajankäyttö kurssilla päivätaasolla.*

Päivämäärä	Selite ajankäytölle	Käytetty aika (h)
14.01.2021	1. Luento 0 (1 h). Tutkittu kurssin ohjeistusta ja materiaaleja (2 h).	3,0
19.01.2021	Oppimispäiväkirjan luonti ja formatointi (1 h). Harjoitus 1 ja harjoitus 2 (1,5 h).	2,5
24.01.2021	Luennon 1 tallenne (1,5 h).	1,5
26.01.2021	Kotlinin perusteet Tutorialspointissa [7] (2 h). Harjoitus 3 (1,5 h). Arkkitehuuri-referenssi [11] (1,0 h).	4,5
09.02.2021	Luento 2 (1,5 h). Harjoitus 4 (1,5 h).	3,0
10.02.2021	Luento 3 (2,0 h). Harjoitus 5 (4,0 h).	6,0
12.02.2021	Room-tutoriaali ja Harjoitukset 6,7,8 aloitettu (4,0 h).	4,0

14.02.2021	Harjoitukset 678 jatkuu (4,0 h).	4,0
15.02.2021	Harjoitukset 678 jatkuu (4,0 h).	4,0
16.02.2021	Harjoitusten 678 viimeis- tely. Play-julkaisun virittely (4,0 h).	4,0
18.02.2021	Luento 4 (1,5 h).	1,5
19.02.2021	Luento 5 (1,5 h).	1,5
20.02.2021	Harjoituksen 9 aloittelua (2,0 h)	2,0
21.02.2021	Harjoitus 9 jatkoa (3,0 h)	3,0
04.03.2021	Harjoitus 9 viimeistely, rapsan täydennys (4,0 h). Luento 6 (1,0 h). Harjoi- tuksen 10 aloittelua (1,0 h).	6,0
07.03.2021	Harjoitustyön ideointi ja dokumentointi (3,0 h). Harj. työn käyttöliittymän suunnittelua (2,0 h).	5,0
10.03.2021	Luento 7 (1,5 h).	1,5
13.03.2021	Harjoitus 10 aloittelu, pro- jektiin valmistelu. Autenti- koinnin konfigurointea. (3,0 h)	3,0
14.03.2021	Harjoitus 10 jatkoa ja vii- meistely, dokumentointi, testaus (3,0 h).	3,0
15.03.2021	Korjausia ja parannuksia harjoitukseen 10 (4,0 h).	4,0

20.03.2021	Päiväkirjan korjailua, Harjoitus 11 aloittelua (4,0 h).	4,0
21.03.2021	Harjoituksen 11 viimeistely (2,0 h). Harjoituksen 12 valmistelu (0,5 h).	2,5
03.04.2021	Harjoituksen 12 toteutusta ja dokumentointia (3,0 h)	3,0
05.04.2021	Harjoituksen 12 viimeistely ja dokumentointi (2,0 h) Valmisteltu harjoitusta 13 (1,0 h).	3,0
06.04.2021	Luento 8 (1,5 h).	1,5
11.04.2021	Harjoituksen 13 työstö valmiiksi ja viimeistely (4,0 h).	4,0
12.04.2021	Harjoituksen 13 testaamista ja raportointia (0,5 h).	0,5
17.04.2021	Harjoitusten 14, 15 valmistelua ja aloittelua (2,5 h). Luento 9 tallenne (0,5 h).	3,0
18.04.2021	Harjoitusten 14, 15 jatko ja viimeistely, rapsan kirjoittelu (9,0 h). Rapsan siistintää korjailua ja täydetelyä (1,0 h).	10,0
22.04.2021	Harjoituksen 16 valmistelu ja aloitus (2,0 h).	2,0
23.04.2021	Harjoituksen 16 jatkoa. Rapsan kirjoittelua (6,0 h).	6,0

24.04.2021	Harjoituksen 16 jatkoja ja viimeistelyä. Rapsan korjailua ja täydentelyä (3,0 h).	3,0
25.04.2021	Harjoituksen 17 aloitusta, applikaatiopohja. Työstöä. Rapsan kirjoittelua (3,0 h).	3,0
02.05.2021	Harjoituksen 17 jatkoja. Rapsa. (7,0 h)	7,0
03.05.2021	Harjoituksen 17 jatkoja. Rapsa. (8,5 h)	8,5
04.05.2021	Harjoituksen 17 jatkoja. Rapsa. (6,0 h)	6,0
05.05.2021	Harjoituksen 17 jatkoja. Xml parserin ja uin työstöä. (3,0 h)	3,0
06.05.2021	Harjoituksen 17 jatkoja. Harjoituksen 17 viimeistely. Rapsan kirjoittelu. (4,0 h)	4,0
07.05.2021	Tehty harjoitus 18. Täydennelty ja korjailtu päiväkirjaa. (1,5 h)	1,5
01.07.2021	Aloitettu harjoitustyön toteutusta. Raportointia. (2,0 h)	2,0
02.07.2021	Jatkettu harjoitustyötä. (3,0 h)	3,0
03.07.2021	Jatkettu harjoitustyötä. Dokumentointi. (5,0 h)	5,0
04.07.2021	Jatkettu harjoitustyötä & dokumentointia. (3,0 h)	3,0

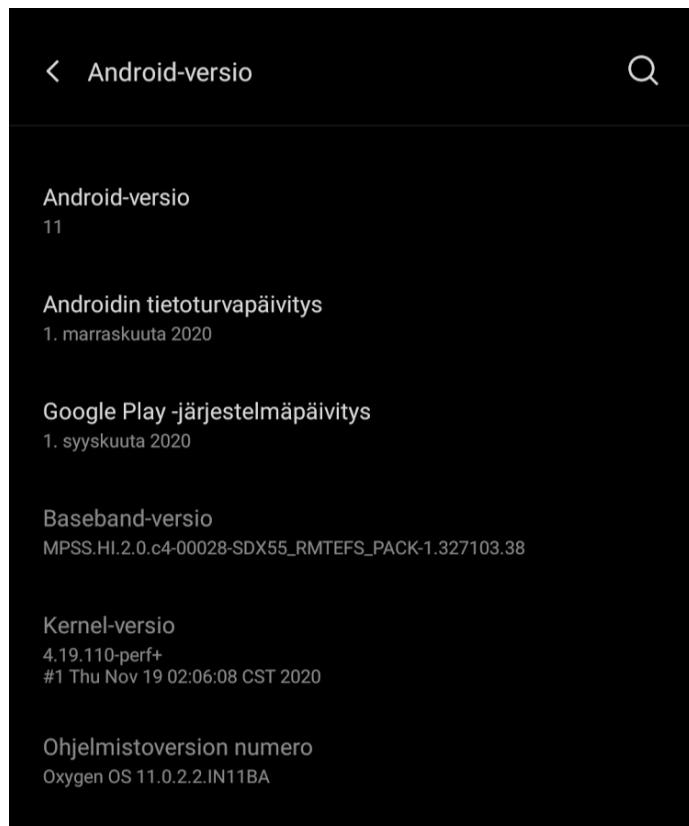
11.07.2021	Jatkettu harjoitustyötä. (4,0 h)	4,0
13.07.2021	Harjoitustyön jatkoa. (3,0 h)	3,0
14.07.2021	Harjoitustyö. (1,5 + 10,0 h)	11,5
15.07.2021	Harjoitustyö. (8,0 h)	8,0
16.07.2021	Harjoitustyö. (4,0 h)	4,0
19.07.2021	Harjoitustyö. (4,0 h)	4,0
24.07.2021	Harjoitustyö. (6,0 h)	6,0
26.07.2021	Harjoitustyö. (5,0 h)	5,0
27.07.2021	Harjoitustyö. (4,0 h)	4,0
28.07.2021	Harjoitustyö. (12,0 h)	12,0
29.07.2021	Harjoitustyö. (14,0 h)	14,0
30.07.2021	Harjoitustyö, korjauksia ja parannuksia. (6,0 h)	6,0
YHTEENSÄ	237,0 tuntia	237,0

2. HARJOITUSTEHTÄVÄT

Tässä osiossa on käsiteltynä kurssin aikana suoritetut harjoitustehtävät, niihin liittyvät ratkaisut muilta osin kuin lähdekoodin osalta, oma pohdinta, erilaiset kohdatut ongelmat ja muu niihin liittyvä maininnan arvoinen asia.

2.1 Harjoitus 1

Valitsin laitteeksi oman puhelimeni. Kyseisen laitteen on valmistanut OnePlus ja laitteen malli on nimeltään OnePlus 8 Pro. Malli julkaistiin alkuvuodesta 2020. Puhelimessa on Android-käyttöjärjestelmä. Kuvankaappaus käyttöjärjestelmän tiedoista kuvassa 1. Tiedoista käy ilmi Androidin pohjaversio, kyseisen järjestelmän kustomoitut build ja sen versionumero, kernelin (käyttöjärjestelmän ytimen) versio, sekä baseband eli radiopiiriin (GPS, WLAN, modeemi, Bluetooth, jne kommunikointiprotokollien piiri) firmwareen eli laiteisto-ohjelmiston versio. Lisäksi linkki esitteeseen puhelimen tarkemmista ominaisuuksista lähteissä [3].



Kuva 1. Kuvankaappaus mobiililaitteen käyttöjärjestelmän tiedoista.

Puhelimessa on siis Android 11, joka on tällä hetkellä uusin Android-versioita aktiivisessa tuotantokäytössä, ja julkaistiin syysuussa 2020 [2]. Puhelimessa on kyseisestä Android-

versiosta tehty mukautettu järjestelmäpaketti (ROM, lyhennetty sanoista ReadOnly Memory, jolla viitataan laitteessa olevaan Flash-muistin kertakirjoitteeseen muistiin), ja kyseisen ROM distribuution nimi on OxygenOS, joka on OnePlus:n itse valmistama järjestelmä omille puhelimille. Eurooppa-versio on nimeltään OxygenOS, kun taas siitä tehty vastaava Kiinan markkinoille suunnattu versio on nimeltään HydrogenOS [5]. Android 11-pohjaisen version numero on vastaava, OxygenOS 11 [5].

Sillä kyseessä on Android-puhelin, ohjelointikielinä toimivat erityisesti Java, jolla Android on pitkälti kehitetty, ja Kotlin [1]. Lisäksi löytyy myös erilaisia cross-platform-kieliä ja frameworkeja, esimerkiksi React Native, joka on Javascript-kielelle toteutettu kirjasto mobiiliapplikaatioiden kehittämiseen Android ja iOS-järjestelmille.

Ohjelointiympäristönä toimii esimerkiksi Kotlinilla tai Javalla kehitettäessä Android Studio, joka on saatavilla Windows, Linux, Mac ja ChromeOS -käyttöjärjestelmille [4]. Uusin versio Android Studiosta on 4.1.1, ladattavissa ilmaiseksi [4].

Laitteesta löytyy mobiilikehityksen kannalta oleelliset, tyyppiset mobiililaitteen ominaisuudet, kuten 4G-, ja 5G-modeemit (mobiilidatayhteys SIM-korttitunnistuksella), WLAN-piiri, Bluetooth 4.2, GPS-piiri softatuella (esim. Googlen location API), 128GB FLASH-tallennustila, 8GB RAM-muistia. Lisäksi laitteessa on seuraavia sensoreita: optinen sorjenjälkitunnistin, kiihtyyysanturi, gyroskooppi, läheisyysanturi ja kompassi. Edellä tarkeennettuna kyseisen omistamani mallin spesifikaatiot, joista tarjolla olevat vaihtoehtoiset spesifikaatiot on jätetty mainitsematta. [3]

2.2 Harjoitus 2

Tein harjoituksen 2 lähinnä ohimennen, sillä olen käyttänyt Gitlab-ympäristöä (sekä TUNI- että yksityistä puolta) lukuisia kertoja aiemmin. Lisäksi Git versionhallintana on itselleni äärimmäisen tuttu sekä koulun, että töiden puolesta.

Kloonasin repon koneelleni, ja alustin sinne tarvittavat konfiguraatiot Gittiä varten (attirbuutit, ignoret) kehitysympäristöjä varten, päivitin readmen, ja lisäsin myös päiväkirjatiedoston. Loin erilliset kansiot harjoituksille, ja harjoitustyölle repoon. Mitään ongelmia en git-repoa valmistellessani kohdannut.

En kuitenkaan luonut tähän harjoitukseen liittyviä hakemistoja tai vastaustiedostoa reposoon, sillä koin niiden luomisen epäolennaiseksi oppimisen kannalta tässä vaiheessa.

Koska puhelimessani on Android-järjestelmä, käytän ohjelointikielenä Kotlinia (Android-alustalle verrattain uusi kieli) käyttäen Android Studiota kehitysympäristönä. Nykyään Android Studio tarjoaa täyden tuen Kotlinin käyttöön [6], joten sen käytämisen

ei pitäisi tuoda ylimäärisiä haasteita tehtävien tai harjoitustyön tekemisessä. Tämän vuoksi konfiguroin repositorion näitä työkaluja varten.

Myös Androidin kehittäjät ovat todenneet Kotlinin itsessään erittäin hyväksi kieleksi, ja tarjoavat Kotlin-first lähestymistapa Android-kehitykseen [6]. Kotlin tarjoaa ekspressivistä ja suppeaa rakennetta, turvallista koodia (nullpointer-ohjelointivirheiden minimointi, muun muassa), yhteensopivutta Javan kanssa (Java-kirjastojen kutsuminen Kotlinista, ja päinvastoin, sekä asynkronisen koodin kirjoittamista [6]. Asynkronisuus on itselleni hyvin tuttua .NET-kehityksestä ja Javascriptista.

2.3 Harjoitus 3

Ennen asennusvaihetta perehdyin jonkin verran Kotlin-kieleen, joka on itselleni täysin uusi tuttavuus. Sen sijaan Javalla olen käyttänyt esimerkiksi työprojektissa, ja käynyt siihen liittyviä kursseja. Java on äärimmäisen samankaltainen tyyliiltään ja ominaisuuksiltaan kuin C#, joka on edesauttanut Javan opettelua kielenä. Molemmat ovat pure-OOP (Object-Oriented Programming) -paradigmaisia, staattisella ja vahvalla tyyppityksellä olevia käännettäviä (*compiled*) kieliä. Javan tapaan myös Kotlin käännetään ensin välikieliksi (*intermediary language*) eli tässä tapauksessa tavukoodiksi (*bytecode*), joka voidaan sitten ajaa JVM:llä (Java Virtual Machine). Käytännössä kyseiset kielet ovat keskenään muunneltavissa toisikseen.

Kotlinin valitsin siksi, että samalla oppisi uuden kielen, ja samalla saisi lisää kokemusta funktionaaliseen ohjelointiin, josta on hankittu pintaraapaisu Scalan ja Apache Spark - map-reduce-klusterikirjaston kanssa. Lisäksi tutustuin hieman Android-applikaatioille suosituun arkkitehtuuriin ja applikaation tyyppilliseen rakenteeseen, sekä komponentteihin [11].

Aloitin tehtävän asentamalla Android Studion uusimman jakeluversion asennuspaketin omalle Windows-koneelleni. Asentaessa valitsin kaikki komponentit asennettaviksi. Asennus sujui ilman mitään ongelmia. Tämän jälkeen ennen uuden projektin luomista kävin Studion asetukset, asentelin mm. Play-kaupan ekstensiöt, ja mukautin tarpeen muukaan muut asetukset itselleni sopivaksi.

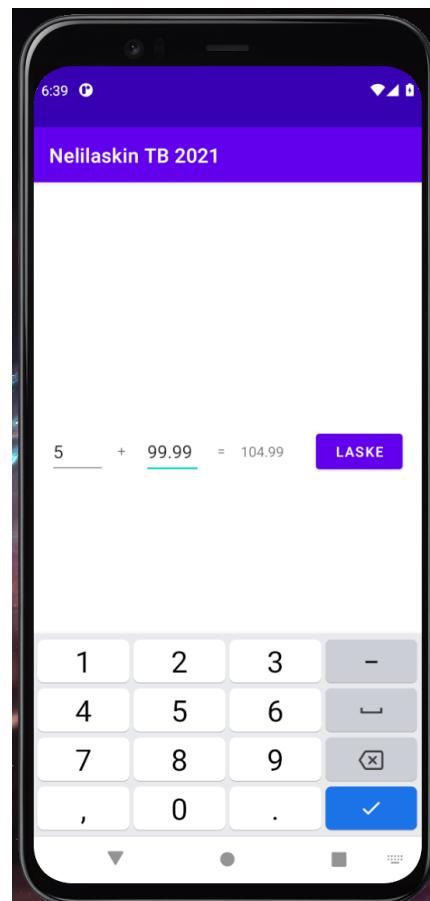
Seuraavaksi loin uuden "Hello World"-applikaation Android Studiossa seuraten Android Studion ohjeistusta [10]. Valitsin Empty Activity, ja loin uuden projektin repoon exercise-kansion alle. Säädin vielä AVDsta kuntoon uuden emulaattorin käyttämään laitetta Pixel 4, ja testasin käynnistää sen. Kun laite oli käynnissä, käänsin ja ajoin projektin, jolloin se asentui emulaattoriin, ja aukesi emulaattorissa onnistuneesti. Mitään vastoinväymisiä en tehtävän suorituksen aikana kokenut.

2.4 Harjoitus 4

Heti harjoitustehtävän alussa oli hieman ongelmia Android Studion kanssa. Ilmeisesti liian innokkaasti koodin kimpussa ähertäminen ja samaan aikaan emulaattorin buuttailu, kun Visual studio C++ projektit ja vagrant-vm:t pyörivät taustalla tuhannen muun applikaation kanssa, sai koko Windowsin grafiikkakälin tai jonkin välissä olleen kirjastokomponentin kaatumaan yhdessä Android Studion kanssa. Tässä onnistuin jopa vielä reporoamaan saman kaatumisen toisen kerran, ilmeisesti ei ole hyvä touhuta samaan aikaan Android Studion applikaatiokoodin kimpussa, kun AWD operoi Android-emulaattorin kanssa, vaan antaa sen rauhassa ensin käynnistää kokonaan loppuun asti, että esim. Gradle varmasti onnistuu buildissaan ja ajot menevät läpi. Myöhemmät vaiheet, kuten itse devaus ja testailu, sujuivat sittemmin aika lailla ongelmitta.

Toteutin tehtävän 4 applikaation Javalla (johtuen vaatimuksesta tehdä tehtävät 4 ja 5 eri kielillä). Varsinaisia ongelmia itse koodaamisvaiheessa ei juurikaan siis ollut. Suurimmat haasteet tulivat Designerin käytössä siiä, että sai elementit aseteltua näkymään ruudulle, yhdistellen niitä nuolilla. Koodipuolella suurin haaste oli taistella tyypimuunnosten kanssa, sillä kyseiset tyypimuunnosten lähestymistavat tuntuivat hirveän hankalilta siihen nähdyn, mitä olen tottunut esim. C# kanssa.

Lisäherkkuna harjoitukseen toteutin laskurin toimimaan myös liukulukujen (*double*) kanssa, eli käyttäjä voi syöttää inputtina joko kokonais- tai liukulukuja, niin halutessaan.



Kuva 2. Kuvankaappaus harjoitus 4:n ratkaisusta. Yksinkertainen summalaskin liukuluvuilla.

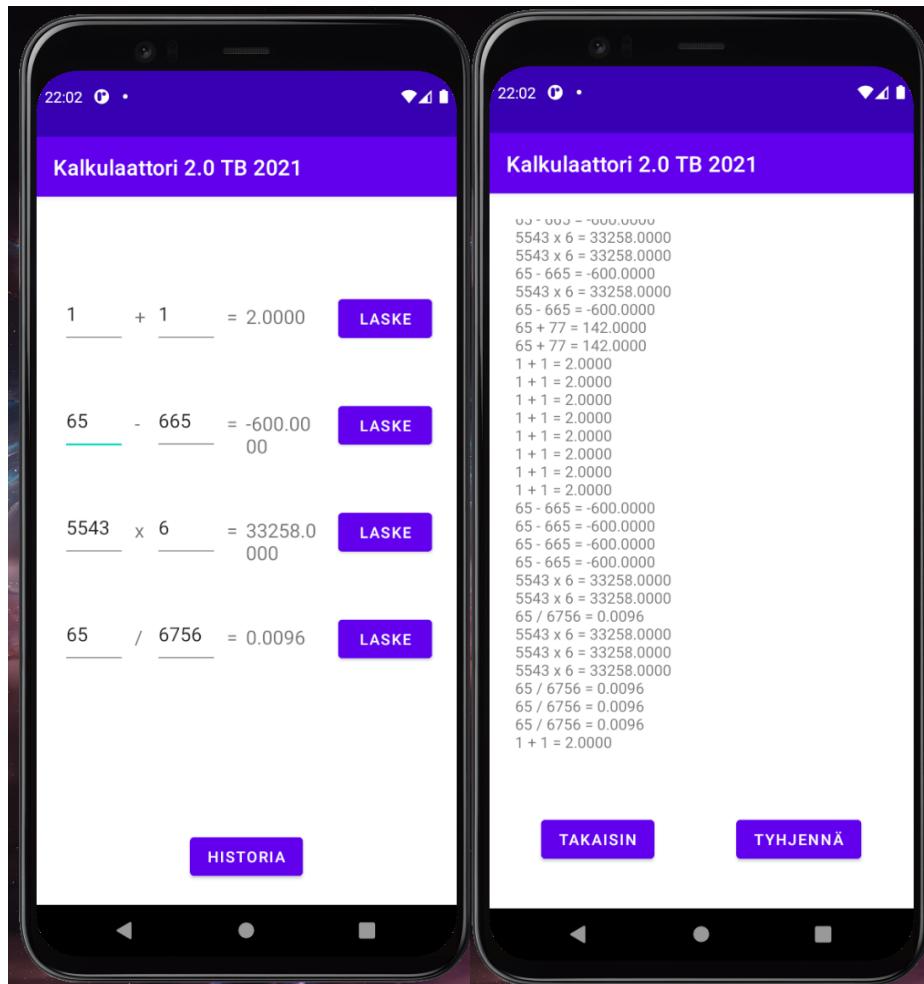
2.5 Harjoitus 5

Tämän harjoitustehtävän toteutin vastaavasti Kotlinilla perustuen tehtävänannon määritteisiin. Loin uuden projektin tehtävää varten hakemistoon exercise_5. Harjoitustehtävän tekeminen oli hyvin samankaltaista kuin aiemmassa harjoituksessa. Ehkä työläintä oli layoutin elementtien asettelu symmetrisesti ikkunaan niin, että kaikki elementit näkyivät ja sijaitsivat oikeilla paikoillaan. Lopulta asettelu onnistui aika mutkattomasti käyttäen kiinteitä lukuja ja asettaen constraintit järkevästi suhteutettuna mm. toisiin elementteihin. Itse taustalogiikka oli suhteellisen helppo toteuttaa. Hieman haasteita tuotti myös Kotliniin tottuminen kielenä, jossa suurimmat niin sanotut haasteet tulivat syntaksin puolesta.

Varsinaisen laskinlogiikan toteutus oli suoraviivainen operaatio. Toisen näkymän implementointi vaati hiukan enemmän aikaa, jotta sai kirjattua laskutapahtumia tiedostoon ja lokinäkymän hakemaan tapahtumahistorian näytölle.

Laskinnäkymästä pääsee nappulan kautta historianäkymään, johon haetaan tiedostosta suoritetut laskutapahtumat. Pienenä lisätyönä tein historianäkymästä skrollattavan, ja lisäsin tyhjennä-painikkeen, joka poistaa historian (tyhjentää tiedoston ja päivittää näytön

sisällön). Lisäksi käytin myös tässä tehtävässä liukulukuja, ja vastaukset pyörristyvät so-pivaan tarkkuuteen. Ohessa vielä kuvankaappaukset ratkaisusta.



Kuva 3. Harjoitustehtävän 5 soluution kuvankaappaukset laskunäkymästä ja historianäkymästä.

2.6 Harjoitukset 6 & 7 & 8

Toteutin harjoitukset 6, 7 ja 8 yhtenä pakettina. Koska kaikki tehtävät liittyvät samaan applikaatioon, käytin kaikille tehtäville yhteistä hakemistoa, johon loin uuden applikaation. Aloitin applikaation suorittamalla ensin "Room with a View" -tutoriaalin [13], jotta saisim perusymmärryksen tietokannan käsittelystä Android-applikaatiossa käyttäen tuo-reimpia työkaluja.

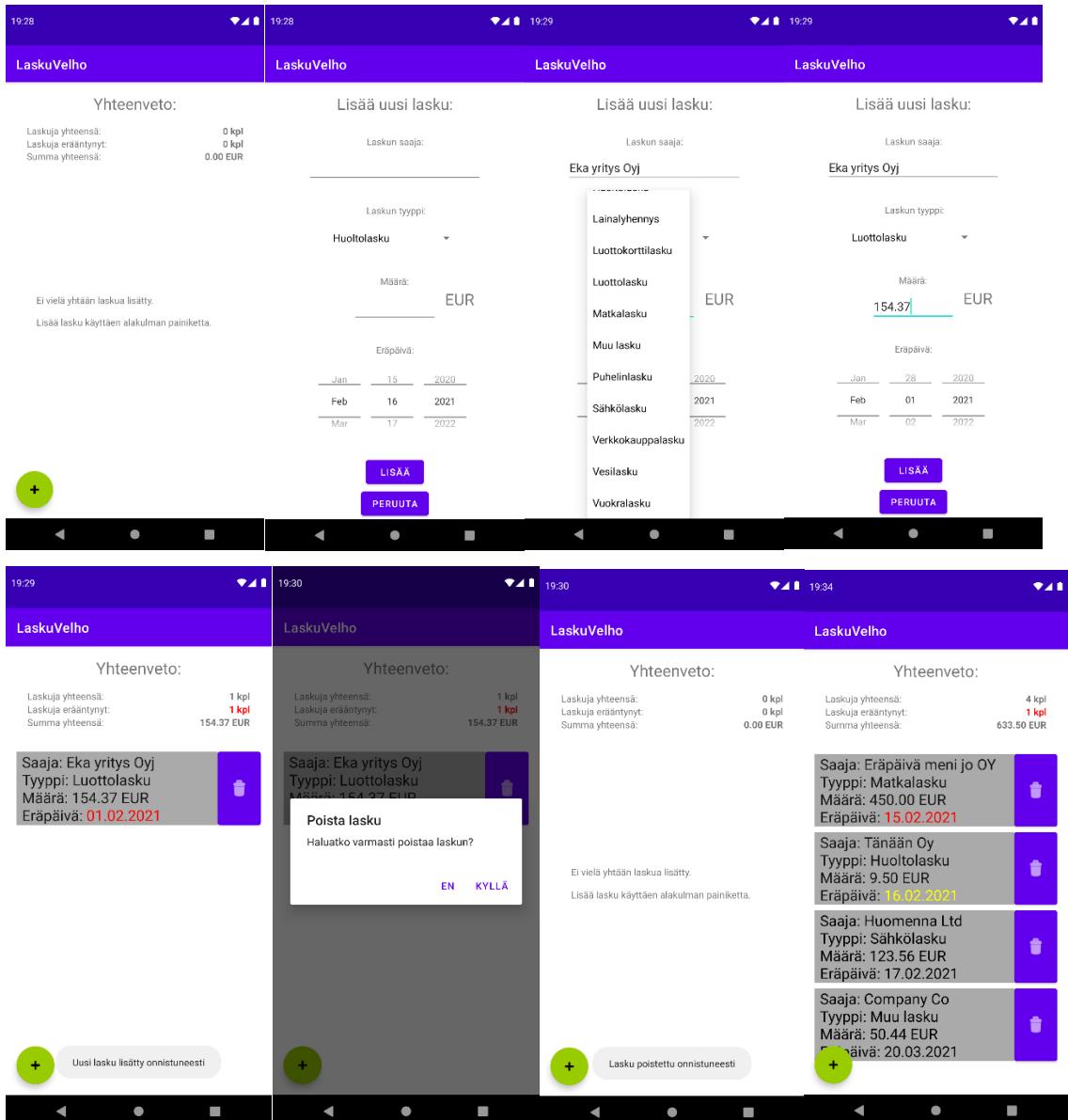
Tein tutoriaalista Kotlin-version. Tehdessäni tutoriaalia toteutin samalla omaa versiotani applikaatiosta, joka toimii laskujen kirjausjärjestelmänä. Applikaatioon voi syöttää laskujen tietoja, tutkia, ja poistaa niitä.

Tutoriaalissa oli jonkin verran erovaisuuksia verraten esimerkiksi luennolla tehtyyn esimerkkiin. Muun muassa LiveData-objektien sijaan neuvottiin käyttämään uudehkoa Kotlinin *kotlinx-coroutines*-kirjastoa ja sen asynkronista Flow-tietorakennetta DAO:n palauttamien listojen wräpperinä, suorittamaan datamuutosten observointia. Gettereitä eikä setttereitä tarvinnut kirjoitella attribuuteille erikseen. Lisäksi tietenkin oma ideani tuotti jonkin verran lisätöötä toteuttaa soveltuvalaksi tutoriaaliin perustuen. Esimerkiksi laskutyyppi-enumerointi vaati hieman suunnittelua järkevälle toteutukselle. Käytin myös rakenteita, joita ei käsitelty erikseen missään olemassa olevissa tehtävissä, joten tehtävää tehdessä oli jonkin verran soveltamista saada kustomoinnit toimimaan halutulla tavalla. Suurin osa ajasta ja työmääristä upposi tähän.

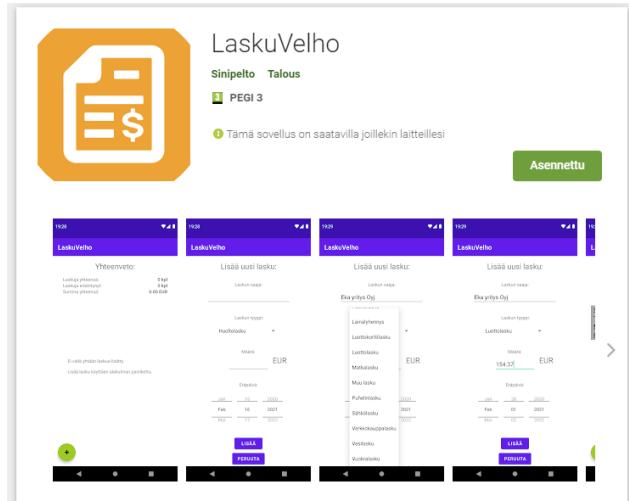
Harjoituksen 6 osalta siis aiheeksi valitsin laskujenkirjausjärjestelmän. Tietokannassa on yksi taulu, johon tallennetaan yhteensä 5 kenttää, joista 4 on käyttäjän kannalta merkityksellisiä: laskun tietokanta-ID, laskun saaja, laskun summa, laskun tyyppi ja laskun eräpäivä. Applikaatiossa on erillinen uuden laskun lisäysnäkymä, johon pääsee pluspainikkeesta vasemmasta alareunasta. Näkymässä syötetään uuden laskun tiedot, ja painetaan lisääpainiketta. Applikaatio tarkistaa, että kaikki kentät on syötetty, ja lisää uuden laskun kantaan, sekä näyttää sen pääänäkymässä. Harjoituksen 7 osalta pääänäkymään sisältyy generoitu listänäkymä RecyclerView-komponentin avulla niin, että kannasta haetaan sen mukaan, mitä näytölle mahtuu. Harjoituksen 8 osalta laskut on järjestetty niiden eräpäivän mukaan nousevaan järjestykseen niin, että vanhin lasku on ensimmäisenä. Laskun oikealla puolella on poistopainike, jolla laskun voi poistaa tietokannasta (ja näkymästä). Poistopainiketta painettaessa näytölle ilmestyy varmistusikkuna, joka varmistaa käyttäjältä poiston. Kyllä-painikkeella lasku poistuu, ei-painikkeella mitään ei tapahdu, ja varmistusikkuna häviää.

Vaadittujen ominaisuuksien lisäksi applikaatiossa on käytetty uusia tietotyyppejä, kuten päiväys, liukuluku, ja listavalikko (drop-down menu). Lisäksi recyclerview-näkymässä on käytetty korttinäkymää, jotta tekstinäkymä ja painike on saatu yhdistettyä. Yhteen listakomponenttiin. Lisäksi applikaatiossa on 2 erillistä activityä (näytä ja lisää -näkymät). Lisäksi syötetyt kentät tarkistellaan lisäysnäkymässä, ja virheellisistä tai tyhjistä syöttäistä ilmoitetaan käyttäjälle Toast-komponentin avulla. Tein myös pääänäkymän yläosaan yhteenvedon, jossa on suodatettuna esim. kaikkien laskujen määrä, erääntyneiden määrä (enemmän kuin 1 punaisella tekstillä), sekä laskujen summa yhteensä. Tämän lisäksi itse eräpäivät näkyvät värikoodattuina niin, että eräpäivän ollessa tämä päivä, näkyy se keltaisella, ja menneet eräpäivät punaisella, muutoin oletusväri on käytössä. Alla vielä kuvankaappaauksia applikaatiosta toiminnassa.

Applikaatio löytyy myös Googlen Play Storesta [16], jonne julkaiseminen oli oma prosessinsa, mutta hoitui suht mutkattomasti. Julkaisun tarkistaminen vie lähemmäs viikon, mutta tarkistusprosessi oli äärimmäisen kattava, ja sen ohella sai runsaasti tietoa ja vinkkejä sovelluksen paranteluun, esimerkiksi potentiaalisten ongelmien, grafiikoiden tai vaikka suorituskyvyn suhtein.



Kuva 4. Kuvankaappauksia applikaatiosta tehtäviä 6, 7 ja 8 varten. Kyseessä on mobiilisovellus laskujen hallintaan.



Kuva 5. Kuvankaappaus sovellussivusta Google Play-kaupassa.

2.7 Harjoitus 9

Room-layerin käyttö tässä kohtaa ei ole järkevää, sillä Firestore-tietokanta pitää sisälään tiedon lokaalin talletuksen siinä tilanteessa, jossa yhteyttä pilveen ei ole saatavilla, ja välimuistitetaan datan. Kun yhteys palaa, synkataan sitten tiedot jälleen pilven kanssa. Tämän vuoksi poistetaan paikallista tietokantaa Roomia käyttävä tietokantapalikka nykyisestä toteutuksesta ja muutetaan se käyttämään Firestore-tietokantaa.

Varsinaisia muutoksia kertyi suhteellisen vähän. Ensimmäisenä loin tilin Firebaseen, ja loin sinne projektin applikaatiota varten. Sen jälkeen otin Firebasesta Firestore-tietokannan ja analytics-raportointityökalun käyttöön suoraan Android Studiosta, ja tämä muodostii em. projektin alle tarvittavat resurssit. Latasin pilveen tunnistautumista varten tarvittavan json-tiedoston ja asetin sen paikalleen. Lisäksi konfiguroin tarvittavat palikat Gradleen.

Aloitin koodimuutokset konverteimalla vanhaa Sqlite-Room--rakennetta toimimaan Firestore-tietokannan kanssa. Aluksi poistin Room-kirjastoon liittyvät toiminnallisudet, ja muutin kannan sisäisen toteutuksen käyttämään Firestore-kantaa. Tämän jälkeen muutin dataluokan, ViewModelin, ViewHolderin ja repositorion toimimaan Firestore-kannan mukaisesti. Suurimmat haasteet olivat Firestore-kannan erilaisen toimintatavan ymmärtäminen, eli snapshotien ja dokumenttien toiminnallisuden ja asynkronisen luonteen, ymmärtäminen, sekä nykyisen olemassa olevan (Sqlite+Room) toteutuksen muokkaaminen sen kanssa yhteensopivaksi. Muutosten seuranta hoitui hyvin ViewHolderin ja Firestoren addSnapshotListener-callbackin, ja MutableLiveData-rakenteen kanssa yhteistyössä.

Ohjeita ja apuja Firestore-kannan käyttöön erityisesti nykyisen toteutuksen päälle muokaten, löytyi varsin vähän lähesti. Muutamasta hyvästä dokumentista ja artikkelista oli kuitenkin suuresti hyötyä toteutusta väsättäässä [18] [22] [25]. Näissä käsiteltiin muun muassa datan käsittelyä Firestoren ja Kotlinin kanssa yleisellä tasolla, sekä erityisesti Firestoren käyttöä yhdessä nykyisen ViewModelin ja LiveData -tietorakenteen kanssa. Esimerkiksi virallinen Firebasesen codelab esimerkiksi oli vain Javalle, mutta varsinaista Kotlin-codelabia ei ole olemassa, kuten Room with a View -codelabissa oli [21].



Kuva 6. Kuva harj. 9 käyttöliittymästä. Visuaalisia muutoksia käyttöliittymäpuolelle ei ole tehty edelliseen tehtävään nähden.

laskuvelho	invoices	JItgzXD9UywwA4bAlp1Y
+ Start collection	+ Add document	+ Start collection
invoices >	JItgzXD9UywwA4bAlp1Y >	+ Add field
	gbTgCEHRM7h2h9ccFHje	amount: 9.99 dueDate: March 5, 2021 at 2:56:46 PM UTC+2 invoiceType: 8 payee: "Firma Oy" uid: null

Kuva 7. Kuvankaappaus Firestoren hallinnan WebUI:sta. Tiedot ovat näppärästi tallentuneet pilvikantaan, ja synkronoivat muutokset reaalialajassa.

2.8 Harjoitus 10

Nyt harjoitus 9:n jälkeen ollaan tilanteessa, missä applikaatio toimii identtisesti harjoitukseen 6 7 ja 8 nähden. Koska data viedään nyt pilveen, ja kaikki applikaatiot käyttävät nyt

samaa kokoelmaa pilvessä, tarkoittaa se sitä, että kaikkien käyttäjien kaikki laskut näkyvät kaikille käyttäjille. Tämä on ongelmallista sikäli, että tarkoitus olisi eristää käyttäjien laskut omikseen, jolloin jokainen käyttäjä voi hallinnoida ainoastaan omia laskujaan. Tätä toiminnallisuutta varten tarvitsemme sovellukseen autentikoinnin. Pienenä lisähaasteena päätin toteuttaa myös email-kirjautumisen lisänä Facebook-autentikoinnin.

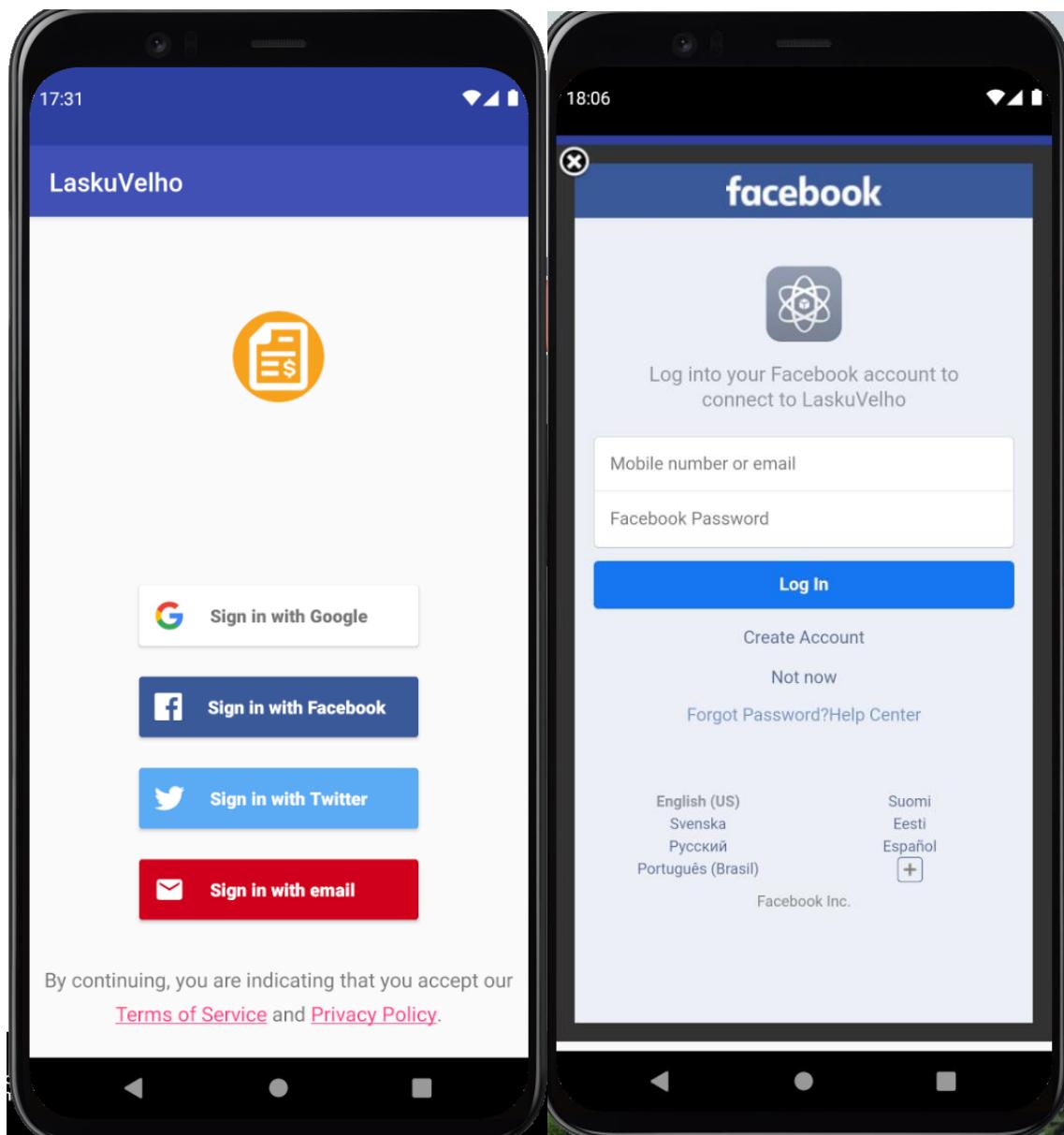
Autentikointi hoituu tietokannan tapaan myös Googlen Firebasen avulla. Ensin lisäilin tarvittavat paketit mukaan Gradleen, ja päivittelin niistä uusimmat versiot. Tämän jälkeen lisäsitin MainActivityyn tarkistuksen Firebasen avulla, onko käyttäjä kirjautunut sisään, ja autentikoimattoman käyttäjän ohjasin Firebasen valmiille autentikaatio-Activitylle.

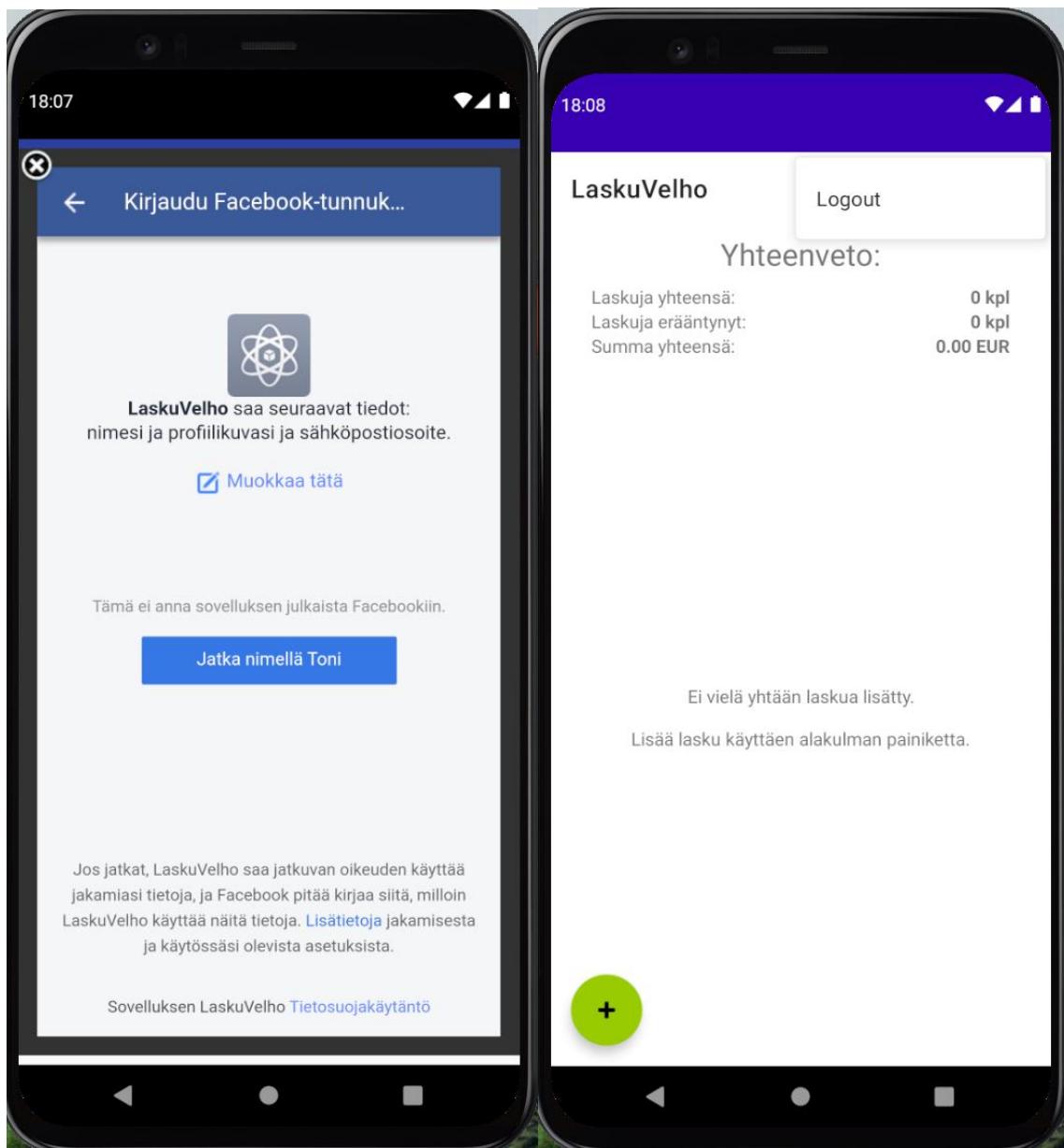
Tämän jälkeen konfiguroin Firebasesta tarvittavat säädöt autentikointeille. Lisäksi Facebook-kirjautuminen vaati jonkin verran lisäsäätiöä, myös Facebook Developers paneeliin kirjautumisen ja siellä vastavuoroisesti Firebase-applikaation konffaamisen, oauth redirect URLit, sertifikaatti-fingerprintien base64 muodot, Applikaatio-IDt ja secretit, yms. Eniten tehtävässä meni aikaa nimenomaan tähän konfiguroimiseen ja eri arvojen määstämiseen ja konfiguroimiseen eri paikkoihin.

Lopulta sain autentikoinnin pelittämään sekä sähköposti-salasanakirjautumiselle, että kolmannen osapuolen kirjautumisille (Google ja Facebook). Kun kirjautuminen rupesi pelittämään, säädin tietokannan käsittelyä niin, että laskutiedot tallentuvat vain kyseisen käyttäjän alle omaan instanssiinsa. Lisäksi säädin vielä Firebasen tietokannan policystä pakotetun autentikoinnin, ennen kuin kantaan saa luku-kirjoitusoikeudet.

Päätin jättää uloskirjautumisen toistaiseksi pois sovelluksesta niin, että käyttäjä kirjataan ulos vasta, kun hän manuaalisesti kirjautuu ulos, poistaa sovelluksen, tai tuhoa sen datan. Lisäsitin myös pää näkymään työkalupalkin, ja asetusvalikon, jossa on erillinen valinta uloskirjautumiselle. Tämän konfiguroin kutsumaan AuthUI:n Logout-toiminnallisuutta, ilmoittamaan tehdystä uloskirjautumisesta ja heittämään käyttäjä takaisin kirjautumisruutuun.

Hankalinta tässä harjoituksessa oli hyödyntää kolmannen osapuolen palveluita (Facebook, Twitter, jne.) sillä niiden käyttöönnottoprosessit veivät huomattavasti aikaa ja konfiguroimista, ennen kuin ne sai pelittämään sovelluksessa. Toisaalta jälleen Androidin oma ohje kirjautumisen toteutukseen oli erinomainen, ja sen avulla login-flow oli helppo toteuttaa käyttäen hyvin pitkälle valmiita palikoita, joita tarvitsi lähinnä kustomoida omien mieltymystensä mukaisiksi. Tässä mielessä harjoitus oli huomattavasti helpompi pariin edelliseen verrattuna.





tinuing, you are indicating that you accept
Uloskirjautuminen onnistui
Terms of Service and Privacy Policy.

Kuva 8. Kuvankaappaauksia autentikoinnista. Kirjautumisnäkymä. Esimerkkinä Facebook-sisäänkirjautuminen ja uloskirjautuminen sovellukseen.

2.9 Harjoitus 11

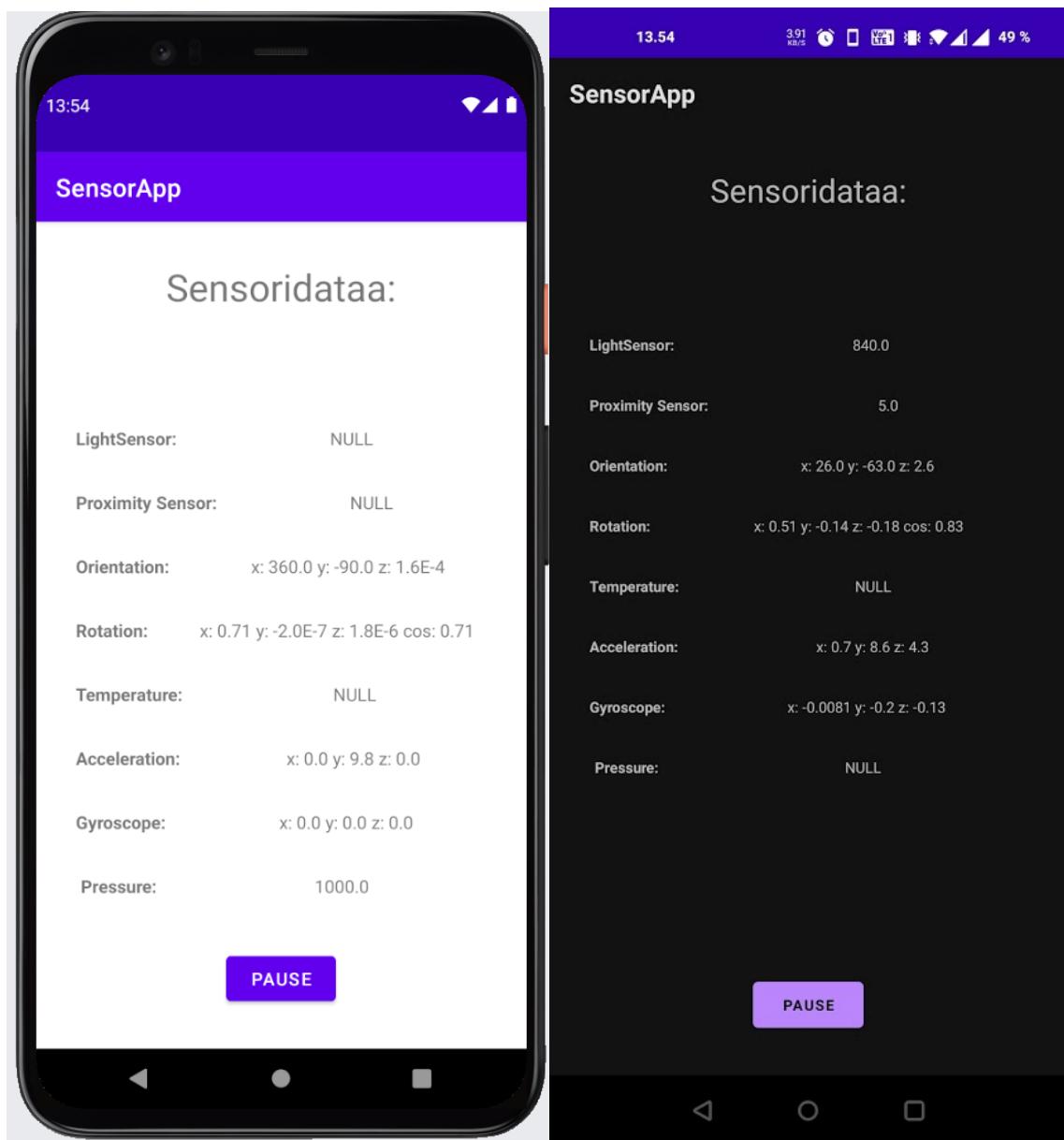
Tehtävässä 11 harjoiteltiin sensoridatan kaivamista puhelimen raudasta. Älypuhelimessa (ja monissa muissa vastaavissa mobiililaitteissa) on hyvin paljon erilaisia sensoreita, kuten liikkeeseen liittyvät sensorit: kiihtyvyysanturit, gyroskoopit, orientaatioon ja

rotaatioon liittyvät sensorit, lämpötilasensorit, mahdollisesti myös jopa kosteussensorit, läheisyysensorit, painesensorit, valosensorit, painovoimasensorit ja magneettikenttäsensorit [26]. Voimme myös lukea sensoreista saatavaa dataa applikaatiossa halutesamme, Androidin tarjoamien laitteistorajapintojen kautta, ja hyödyntää sitä applikaation logiikkassa, esimerkiksi tarjoamalla käyttäjälle palveluja (vaikkapa sovellus urheilun seurantaan) tai keräämällä BigDataa sensorien tuottamalla datalla.

Aloitin tehtävän luomalla yksinkertaisen käyttöliittymän, johon sensoridata tulee näkyville. Päätin tehtävänannon määrittämän muutaman sensorin lisäksi näyttää vielä muutaman ylimäääräisen sensoriarvon. Tämän jälkeen siirryin designista logiikkapuolelle, jossa asetin aktiviteetin periytymään sensorEventListener-luokasta kaapatakseni Sensoreihin kohdistuvia eventtejä. Tämän jälkeen lisäsin kuuntelijaan tarkastelun valitsemieni sensorien kohdalle, joista se poimii eventin ilmoittamat sensorin muuttuneet arvot tekstinäkymiin. Lisäksi määritin sensorien ja sensormanagerin latauksen muuttuihin on-createssa (optimointi) ja sensorien kuuntelijan liittäminen kyseiseen aktiviteettiin on-startissa. On-pausessa kuuntelijat irrotetaan, jotta sovellus ei esimerkiksi syö liikaa akkuvirtaa, kun se on vaikkapa taustalla, mutta ei kokonaan tapettuna. Hidastin myös näytteenottoonopeutta selkeästi (1 sekuntiin), joka on mahdollista API level 9:stä eteenpäin käyttää muita kuin esimääritettyjä sapling rate -arvoja. Tämä ei kuitenkaan vaikuttanut esimerkiksi orientaation tai rotaation sensorieventtien syntymisnopeuteen, vaan niitä tulvi entiseen tahtiin. Dokumentaatiossa mainitaankin, että eventtejä saattaa asetuksesta huolimatta tulla silti hitaanmin tai nopeammin [27].

Harjoitus oli huomattavasti paria edellistä helpompi, sillä ulkoisia kirjastoja ja niiden dokumentaatioita ei tarvinnut käyttää, vaan kaikki oli saatavilla suoraan natiivin tarjonnan kautta. Tämä helpotti pakettienhallintaa ja vähensi siihen kulunutta aikaa. Lisäksi Androidin oma dokumentaatio sensorien käyttöön [26] oli erittäin laadukas ja riittävä. Haastavinta oli orientaatio- ja rotaationsensorien käsitteily, sillä sen datan lukeminen ei toiminut ihan samalla periaatteella kuin monet muut sensorit, kuten kiihtyvyys, paine, valo, jne. Lisäksi sensorien testaaminen erityisesti virtuaalipuhelimella oli hyvin hankalaa, joten applikaatio oli pakko testata myös fyysisellä laitteella todetakseen sensoriarvojen realisisen käyttäytymisen.

Myös ajastuksen säätäminen toimivaksi tuotti hitusen haastetta ymmärtää ajastusluokkien toimintalogiikka kunnolla, mutta kun tähän liittyvät ongelmat selvisivät, ne olivat suhteellisen suoraviivaista ratkaista. Esimerkiksi samaa TimerTask objektiota voi ajastaa vain kertaalleen, joten tästä varten ei voinut käyttää kiinteää Task-objektia muistissa, vaan se piti luoda uutena apufunktion avulla. Kun ajastuksen sai lopulta pelittämään, se toimi erittäin hyvin ja tehokkaasti, sovellusta hidastamatta.



Kuva 9. Kuvankaappaukset tehtävästä 11. Sensorien dataa sekä emulaattorilla, että omalla puhelimella. Selkeästi nähtävissä, että oikealla puhelimella on enemmän sensoreita käytettävissään.

2.10 Harjoitus 12

Harjoituksessa 12 harjoiteltiin kuvan ottamista ja tallennusta mobiilisovelluksessa. Aloitin harjoituksen luomalla kamera-applikaatiolle uuden projektin.

Arvioitu työmääriä tähän harjoitukseen oli taas hieman suurempi, sillä vaatimus natiivin kameran käyttöön ilman kamera-applikaation käyttöä lisää työmääriä huomattavasti.

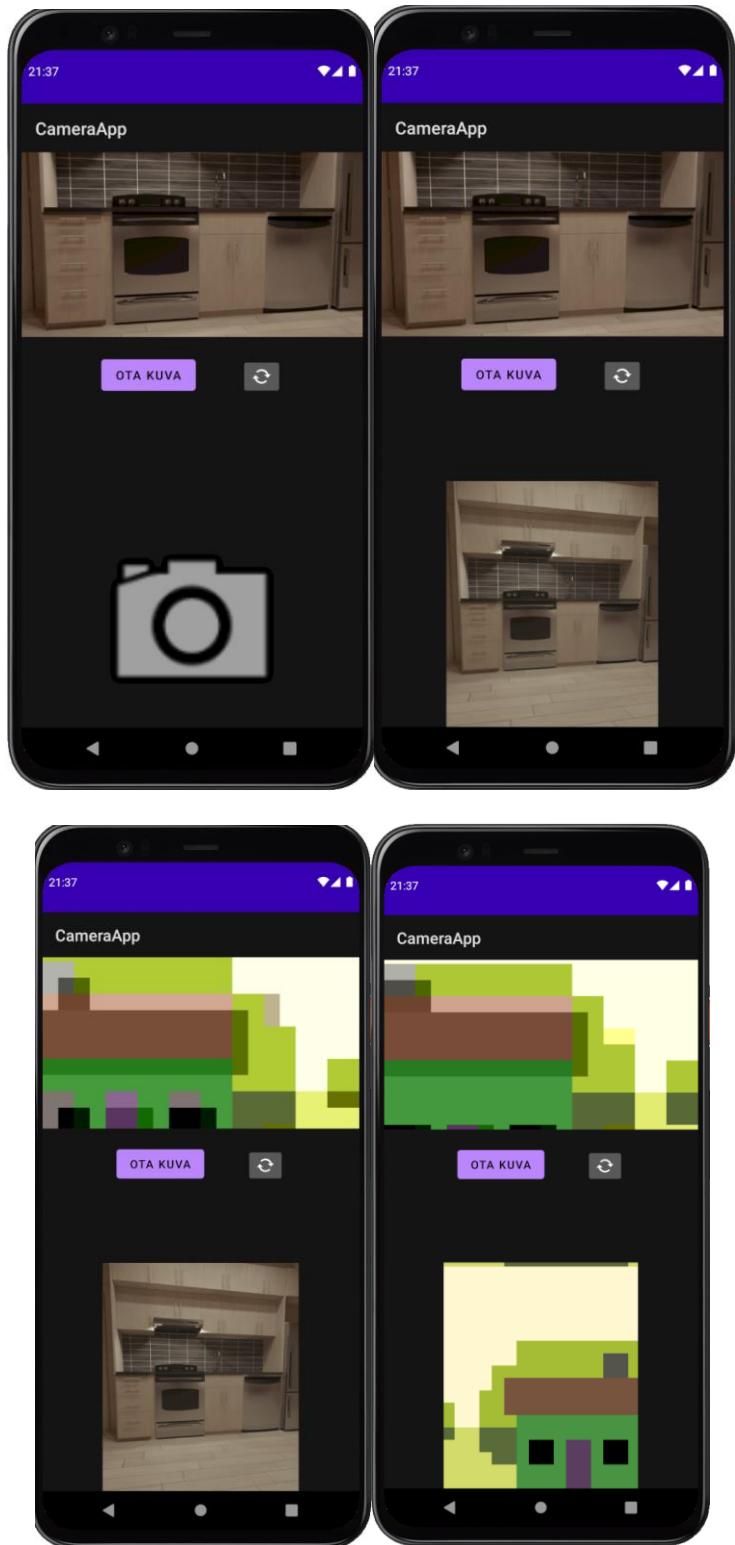
Ensimmäisenä katsoin applikaatiolle oikeudet kuntoon manifestista kameran käyttöön. Päädyin dokumentaation suosittelemana käyttämään CameraX-jetpack-kirjastoa van-

hentuneen Camera-apin sijaan. Asentelin paketin seuraavaksi projektiin käyttöön. Tämän jälkeen lähdin rakentamaan käyttöliittymää, johon tuli kaksi näkymää, kameranäkymä, valokuvanäkymä, kuvanottonappi, sekä lisämausteena kameran flippausnappi (etu- ja takakameran välillä vaihtava painike laitteissa, joissa on sekä etu- että takakamera).

Tämän jälkeen lähdin työstämään pää näkymän applikaatorunkoa, johon tulee logiikka oikeuksien hankinnasta, kuvan ottamisesta ja tallentamisesta sekä kameran valinnasta (flippaamisesta). Työstö sujui pääosin ongelmissa, kun seurasi codelabia CameraX-kirjaston käyttämiseen [30]. Muutamaan ongelmaan kuitenkin törmättiin matkan varrella.

Sopivien komponenttien valinta tuotti alkuun hieman haastetta, millä saisi näkymään kameran esikatselunäkymän, ja millä taas otetun kuvan. CameraX:stä löytyi suoraan sisäänrakennettuna tähän sopiva palikka: PreviewView. Sen sijaan ongelmallisempi oli otetun kuvan tallentaminen omaan näkymään. Tässä ratkaisuna toimi androidin oma ImageView, jota varten kuitenkin piti konvertoida CameraX:n tuottama ImageProxy objekti Bitmapiksi, ennen kuin sen sai lähtettävä ImageView-objektiin. Tähän löytyi kuitenkin onnekksi apuja, sillä konvertointi olikin varsin haastava prosessi lopulta toteuttaa. Lisäksi kuva tuli ensin 90 astetta väärässä rotaatiossa, joka piti myös saada tallennusruutuun korjattua.

Lisäksi codelabissa oli muutamia virheitä, jotka tuottivat hieman ylimääräistä päänvaiavaa, mutta selvisivät kuitenkin suhteellisen loogisesti (lähinnä päättelyssä olevia puutteita ohjeissa) Muilta osin tehtävä oli suhteellisen selkeä toteuttaa.



Kuva 10. Kuvankaappaauksia harjoituksesta 12. Sovelluksessa esikatselu ja tallennetun kuvan näkymät. Lisäominaisuutena vaihto etu- ja takakameran välillä.

2.11 Harjoitus 13

Harjoituksessa toteutettiin askelmittari, joka mittaa käveltyjä askelia sensorien avulla, ja mahdollistaa myös toiminnallisuuden lopettamisen, ja tulosten tallentamisen. Lähdin toteuttamaan sovellusta, joka käynnistyessä yrittää ladata tiedostosta tallennetun askelmäärän, jos sitä ei löydy, laskuri asetetaan nollaan. Sovellus näyttää tämänhetkisen askelien määrän. Sovellus on alussa taukotilassa, ja askelien kerääminen on pysäytettynä.

Sovelluksesta löytyvät painikkeet askelien keräyksen aloittamiseen ja pysäytämiseen, askelmäärän tallentamisen tiedostoon ja askelmittarin lukeman nollaamiseen. Kun aloituspainiketta painetaan, aloittaa sovellus askelien keräämisen, ja painike muuttuu pysäytys/taukopainikkeeksi, jota painamalla taasen askelien kerääminen pysähtyy (nykyinen askelmäärä jää näytölle).

Lähdin aluksi työstämään raakaversiota käyttöliittymästä, johon lisäilin tarvittavat komponentit. Karkean hahmottelun jälkeen asettelin komponentit sopiville paikoilleen ja täydensin niiden tiedot, sekä hienosäädin niiden ominaisuudet kuntoon. UI-puolen jälkeen siirryin työstämään logiikkapuolta ainoalle pääänäkymälle. Lisäs in ensi alkuun manifestiin ACTIVITY_RECOGNITION-permissionin saadakseni applikaatiolle asennustason oikeuden aktiviteettien seurantaan. Toteutin sitten sensorien hallinnan hyödyntäen sensoridata-codelabia [33] ja aikaisempaa harkkatehtävää 11 apuna käyttäen. Päädyin käyttämään Androidin sisäänrakennettua askelmittaria (TYPE_STEP_DETECTOR) [35], joka on sisäänrakennettu sensori askelten mittaan uusimmissa API-versioissa. Sensori palauttaa aina float-arvon "1.0", kun se havaitsee ihmisaskelta muistuttavan liikesarjan laitteen liikesensoreita hyödyntäen.

Tarjolla on myös aggregoiva TYPE_STEP_COUNTER sensori [34], joka kerää askeleita puhelimen päälläoloajan ja raportoi kokonaismäärää väliajoin [34]. Kyseinen sensori ei kuitenkaan sovella tähän tarkoitukseen, yhtä hyvin, sillä tarkoitus on poimia sovelluksen käynnistysajankohdan jälkeiset yksittäiset askeleet. Periaatteessa tuotakin sensoria voisi käyttää paremman puutteessa niin, että seurailee sensorin delta-arvoa (muutosta edelliseen), ja raportoi tiedon aina sen saapuessa, mutta tämä ei ole yhtä tarkkaa. Toteutin sensorin konfiguroinnin virhetarkasteluineen. Sovellus heittää ponnahdusdialogin, jos sensoria ei ole saatavilla, ja pakottaa käyttäjän sulkemaan sovelluksen.

Seuraavaksi toteutin hallinnan mittaan aloittamiseen ja pysäytämiseen boolean flagin avulla, ja toiminnallisuuden tätä hallitsevalle napille. Käytin tässä apuna start-stop nappulan click-kuuntelijaa, joka seuraa napin painalluksia, ja muuttaa lipun asentoa sen

mukaisesti, ja rekisteröi tai poistaa sensorin kuuntelijan sovelluksesta riippuen lipun asennosta. Näin maksimoidaan virransäästö, kun sovellus on pysäytystilassa.

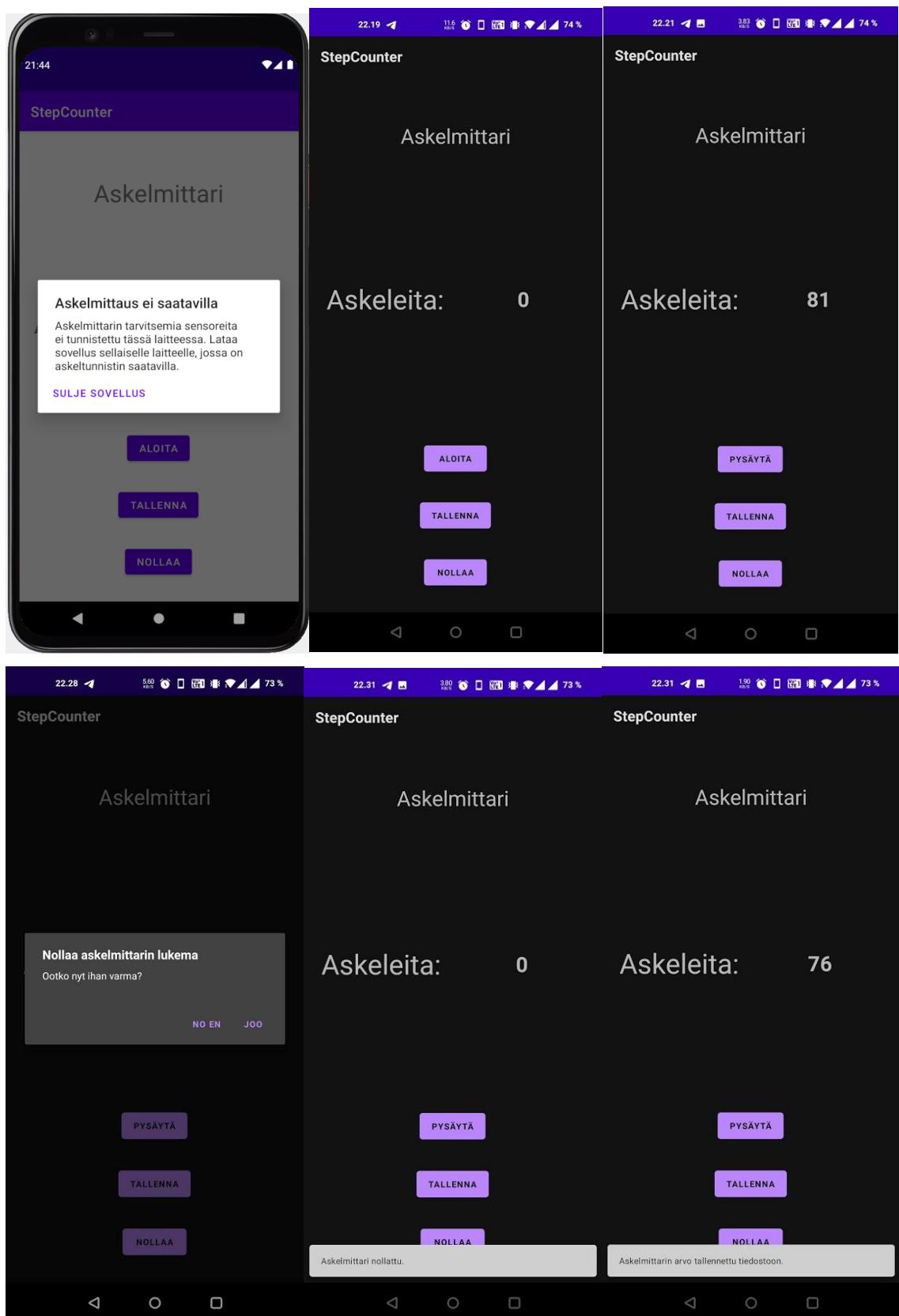
Toteutin myös toiminnallisuudet aloitukselle ja pysätykselle, nollaukselle ja arvon tallennukselle. Tallenna-nappula tallentaa nykyisen askelmittariin kerätyn arvon tiedostoon. Kun ohjelma avataan, luetaan tallennettu arvo tiedostosta muistiin.

Lisäsin myöhemmin kyselyn ACTIVITY-oikeuksien käyttöön runtime-tasolla niin, että oikeuden status tarkistetaan aina sovelluksen käynnistyksessä, sekä painettaessa käynnistyspainiketta. Mikäli käynnistyspainiketta painettaessa lupa on aiemmin evättyvä, ei painike käynnistä sensorien kuuntelua ollenkaan).

Tehtävä oli muutoin varsin suoraviivainen, joskin suurimmat haasteet olivat saada sensoriarvo ulos ja testattua sovelluksen toimintaa käytännössä. Emulaattorilla testaaminen oli luonnollisesti mahdotonta, joten varsinaisen sensorin testaaminen käytännössä jää aivan sovelluksen kehittämisen loppuvaiheille fyysisen laitteen (oman puhelimeni) kanssa. Alkuun oli hieman sähellystä permissioiden kanssa, sillä emulaattorilla ei tarvinnut miettiä varsinaisen sensorin käyttöä, ja oikealla laitteella testatessa tuli uutena tietona, että myös liikunta-aktiviteettien seuraamiseen tarvitsee pyytää eksplisiittisesti oikeus käyttäjältä, eikä pelkkä manifesti-ilmoitus riitä (eli kyseessä on *dangerous permission API-versioissa 29+*).

Lisätyönä hyödynsin oikeuksien käsitellyssä ulkoista kirjastoa, Dexter [37], jolla varmistan ja pyysin oikeudet Activity-permissionin käyttöön.

Testailin jonkin verran sovellusta, käyttäen STEP_DETECTOR sensoria ja huomasin, että sensori antaa aina pieni viiveen ennen askeleiden raportoimista. Kun alkaa kävelemään, niin vähän ajan päästä sensori raportoi "köntänä" suurin piirtein liikkeellelähdöstä kertyneet askeleet (usein esim. n 10 askelta), ja jos pysyy liikkeessä, se kerryttää siitä hetkestä alkaen askeleita tasaiseen tahtiin, kun käytössä oli SENSOR_INTERVAL_FASTEST, noin 1–2 sekunnin välein 1-2 askelta lisää, riippuen omasta nopeudesta (kävelien n. 1 askel ja juosten n. 2 askelta sekunnissa lisää). Lisäksi samalla tavalla esiintyi viivettä myös kävelemisen loputtua, ja sensori jatkoi raportoimista jonkin aikaa viiveellä kävelyn pysähtymisen jälkeen. Tässä mielessä sensori ei ole hyvä esimerkiksi tilanteessa, jossa sovelluksen pitäisi pystyä reagoimaan hyvin nopeasti tilanteeseen, jossa henkilö lähtee liikkeelle tai pysähtyy, vaan reagoinnissa syntyy aina n. 10 sekunnin viive.



Kuva 11. Kuvankaappaauksia harjoituksesta 13. Ensimmäinen kuva on emulaattorista, loput omasta laitteesta. Emulaattorissa ei ollut askelmittauksen sensoreita saatavilla, joten siitä erillinen ilmoitus käyttäjälle.

2.12 Harjoitus 14 & 15

Harjoitukset 14 ja 15 ovat nipputettu yhdeksi tässä raportissa, ja niihin viitataan tässä yhtenä isona harjoituksena. Aloitin harjoituksen luomalla sille projektipohjan ja päivittämällä konfiguraatiot ym. Ajan tasalle. Käytin harjoituksessa tukena sijainnin käsittelyä harjoitavaa codelabia [36]. Lisäsin manifestiin tarvittavat permissiot, ja käytin tässäkin edellisestä harjoituksesta tuttua permissiokirjastoa Dexter runtime-permissioyntöjen hallintaan [37].

Tuttuun tapaan lähdin itse logiikan osalta liikkeelle käyttöliittymästä, ja lähdin hahmottelemaan siihen tarvittavia komponentteja. Jaoin pää näkymän kahteen osaan, jossa ylempanä näkyy, nykyinen sijainti, sekä tallennusnappula, ja listattuna tallennettujen sijaintien määrä alapuolella. Lisäfeaturena nappula, jolla voidaan poistaa kaikki tähän asti tallennetut sijainnit. Tämän lisäksi hahmottelin alaosan karttanäkymän, jossa käyttäjän sijainti ja tallennetut pisteet näkyvät.

Seuraavaksi siirryin työstämään sovelluksen logiikkaa. Aloitin tällä kertaa fiksummin säätämällä permissiologiikan kuntoon aivan alkuun, lisäten pyynnöt tarvittaville oikeuksille (Internet, ja access fine location).

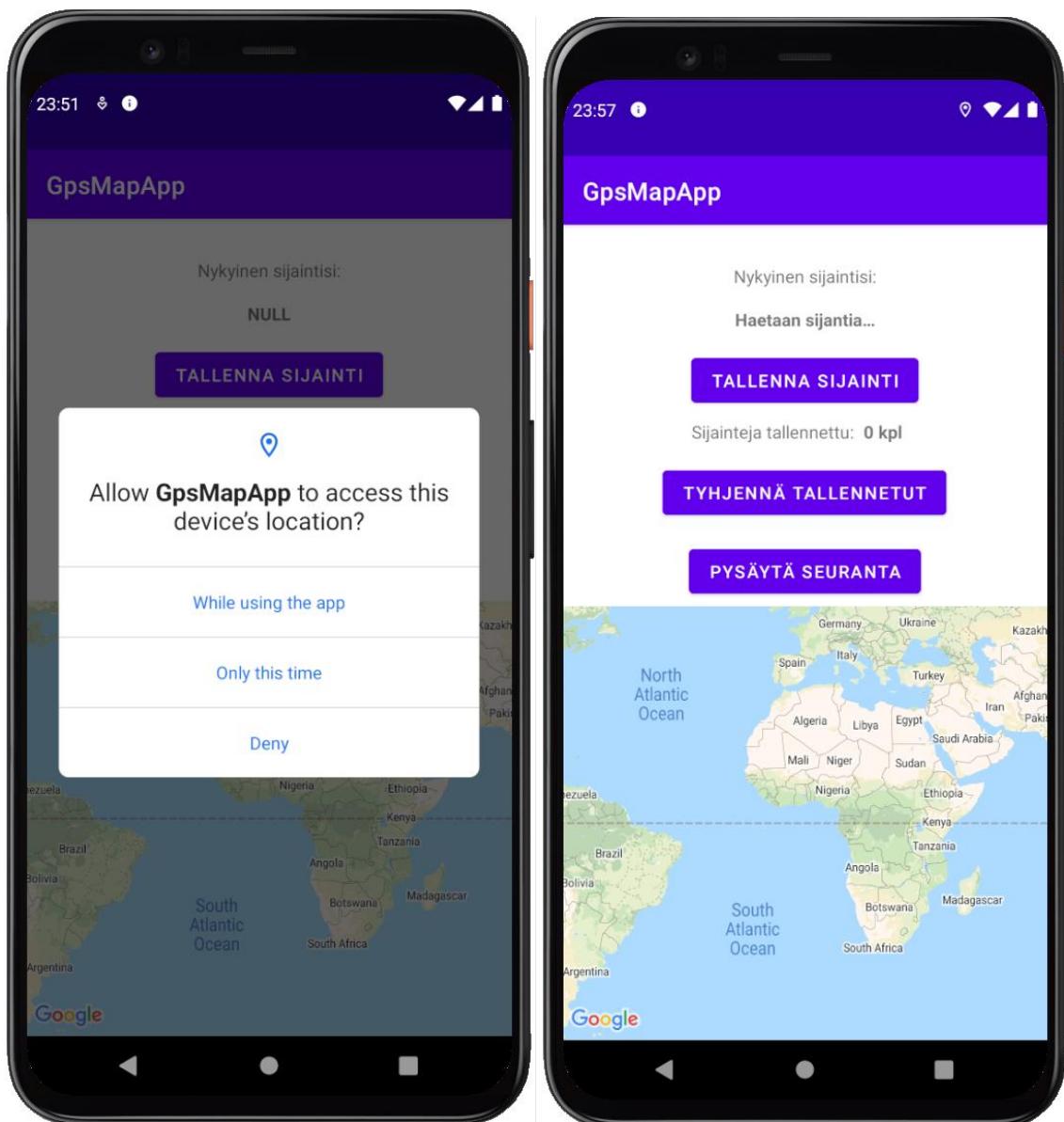
Seuraavaksi toteutin varsinaisen logiikan sijainnin selvitykseen ja päivittämiseen. Käytin tässä Location-codelabin [36] käyttämää LocationRequest-Response-Callback arkkitehtuuria, jossa ensin luodaan Locationrequest-tyyppinen sijaintipyyntö, johon täsmenneetään oleelliset tiedot, kuten sijainnin tarkkuus, päivitysväli, maksimikesto, jne. ja alustetaan location provider, joka oli tässä FusedLocationProviderClient. Tämän jälkeen luodaan callback-objekti, jota kutsutaan, kun sijaintiin saadaan päivitys annetun pyynnön mukaisesti, ja käsitellään saatu vastaus. Sen jälkeen toteutin logiikan sijaintiobjektien tallentamiseen ja siivoamisen. Tähän vielä ui:n päivitysten konffailu pääälle niin harjoituksen 14 osuus oli tehty.

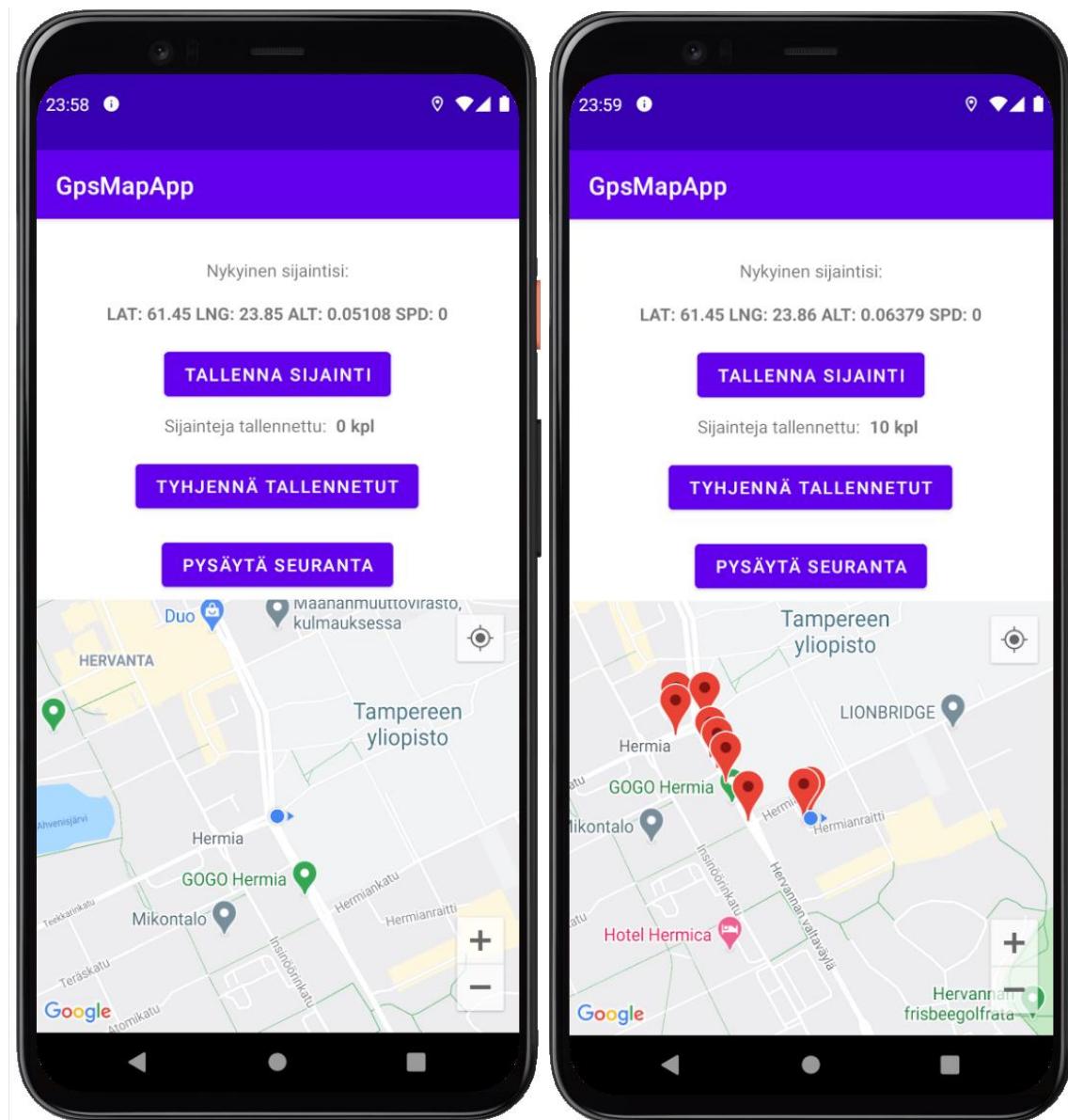
Saatuani edelliset toiminnallisuudet kunnolla pelittämään, siirryin työstämään varsinaista karttanäkymäosuutta. Päädyin toteuttamaan karttanäkymän käyttäen itse Googlen Maps SDK:ta, sillä minulla oli jo entuudestaan billing-tunnukset luotuna Google Cloudiin, joten sain suhteellisen vaivattomasti enabloitua Maps-API:t käyttööni ja loin itselleni mapsia varten IP-rajoitetun API-avaimen. Tutustuin aluksi itse SDK:hon ja sen tarjoamiin ominaisuuksiin ja rajapointoihin. Tämän jälkeen aloin tutkimaan tutoriaaleja karttanäkymän näyttämisestä Android-alustalla [39].

Alkuun määrititin api-avaimet karttanäkymään ja applikaatioon. Määritin seuraavaksi MainActivityn kuuntelemaan karttanäkymän valmistumista, ja callbackissa asetin map-olion paikoilleen muistiin. Tämän jälkeen testasin, että karttanäkymä latautuu ja toimii. Tässä suurimpana apuna oli karttanäkymän oman lifecyclen käytön ymmärtäminen ja liittäminen osaksi activityn toiminnallisuutta [42]. Erityisesti sen tajuaminen, että kutsuu lifecyclen vaiheita manuaalisesti, kun mapviewiä ei ole periytettynä vaan pelkästään luokkaan tallennettuna objektina.

Sitten konfiguroin kartan käyttämään MyLocation ominaisuutta, jotta laitteen nykyinen sijainti näkyisi kartalla, ja lisäksi enabloin oman sijainnin hakevan painikkeen [40]. toteutin vielä markerien asettamisen kartalle omaan sijaintiin ja niiden pyyhkimisen pois napin painalluksesta [41].

Lopuksi bonus-featureena toteutin kartan fokuksen käyttäjän sijaintiin, käyttäjän sijainnin seurannan kartalla, sekä nappulan sijainnin seuraamisen kytkemiseen päälle ja pois. Hieman hankaluksia toteutti permissioiden hallinnan, käyttäjän sijainnin saamisen näkyviin kartalla, sekä sijainnin seuraamisen näiden kaikkien yhteensovittamisen järkevästi ja niin, ettei missään tilanteessa syntyisi virheitä tai kaatumisia. Lisäksi alkuun oli haastavaa löytää sopivasti dokumentaatiota MapView tyyppisen karttanäkymän käsittelyyn, tai löytyneitä dokumentaatiota ja ohjeita oli hankala tulkita nimenomaisen implementaation edaksi, jonka vuoksi karttaosuuden työstäminen eteni alkuun hitaasti.





Kuva 12. Kuvankaappaauksia harjoituksista 14 & 15. Sovelluksessa kaksi osaa, halintapaneeli yläosassa ja dynaaminen karttanäkymä alaosassa. Kuvassa myös tallennetut 10 sijaintia näkyvissä.

2.13 Harjoitus 16

Harjoituksessa 16 keskityttiin datan hakuun ulkoisesta palvelusta rajapinnan kautta. Tässä osoittautui järkeväksi hyödyntää tehtävässä sopivaa kirjastoa datan hakemiseen ja käsitteilyyn sovelluksessa.

Johtuen syvällisestä henkilökohtaisesta inhostani XML:ää kohtaan, päädyin käyttämään valuuttakurssien lähteenä Fixer.io -nimistä JSON-rajapintaa tarjoavaa verkkopalvelua [43]. JSON eli JavaScript Object Notation on eräs datan esitymuodoista XML:än kanssa, mutta on modernimpi ja dataitehokkaampi ja tiiviimpi tapa kuljettaa dataa vain sen verran kuin sitä on, kun taas XML:ät tyypillisesti noudattavat tiettyä rakennetta eli schemaa, joka

edellyttää monimutkaisten tietorakenteiden ja valtavan metadatamääärän käytämisen viestin mukana. Lisäksi XML:ssä tägeillä on usein myös erikseen sulkeva tagi, joka sekä syö turhaan lisää merkkejä viestissä. Sen sijaan JSON on fleksimpi ratkaisu, jossai ei varsinaisesti ole ennalta määritettyä skeemaa, vaikkakin se voi ja on suositeltavaakin sellaista noudattaa, mutta esimerkiksi puuttuville datoille ei tarvitse muodostaa rakenteita ollenkaan tilaa säätääkseen, kunhan näiden mahdollisesti puuttuvien kenttien käsittely on myös rakennettu sujuvaksi.

Optimoidakseni virrankulutusta, päädyin toteuttamaan sovelluksen logiikan niin, että sovelluksen käynnistyessä valuuttakurssit haetaan kertaalleen sovelluksen muistiin. Tämän jälkeen sovellus päivittää taustalla uudet kurssit hieman API:n tarjoaman ilmaisversion mukaista päivitystahtia nopeammin, joka on siis keskimäärin kerran tunnissa. Eli sovellus tekee päivityksen 5 minuutin välein ajastetusti [46]. Lisäksi ajastus päivitptyy ensimmäisen kutsun jälkeen seuraamaan API:n ilmoittamaa päivitysaikaa ja siitä laskettua intervalia (5 minuuttia edellisestä päivityksestä). Lisäksi sovellus näyttää käyttäjälle, milloin tiedot on viimeksi päivitetty palveluntarjoajan päässä, eli tästä voi päätellä, kuinka tuoreita kurssit ovat.

Talletin valuuttakurssi-palveluun luomani API-avaimen turvallisesti erilliseen "keys.properties" -nimiseen tiedostoon, jota normaaliolosuhteissa en lisäisi versiohallintaan ollenkaan, mutta kurssihenkilökunnan sovellustestattavuuden puolesta tein tässä kohtaa poikkeuksen, ja lisäsin myös sen Gittiin. Tiedostossa on siis henkilökohtainen avain, millä CEX-rajapintaan voi tehdä kutsuja (kutsut siis kuluttavat kuukausittaista 1000 kutsun limiittiä, mikä on tässä käyttötarkoituksessa melkeinpä enemmän kuin tarpeeksi).

Aloitin jo erittäin tuttuun tapaan hahmottelemalla sovellukselle käyttöliittymän rungon designer-näkymässä. Lisälin komponentit eurojen syöttöön, kurssien päivitykseen ja alle arvojen näyttämiseen eri kohdevaluutoissa. Päädyin näyttämään kurssit euroista 6:lle muulle valuutalle: USA:n dollari, Englannin punta, Ruotsin kruunu, Venäjän rupla, Sveitsin frangi ja Kiinan juan.

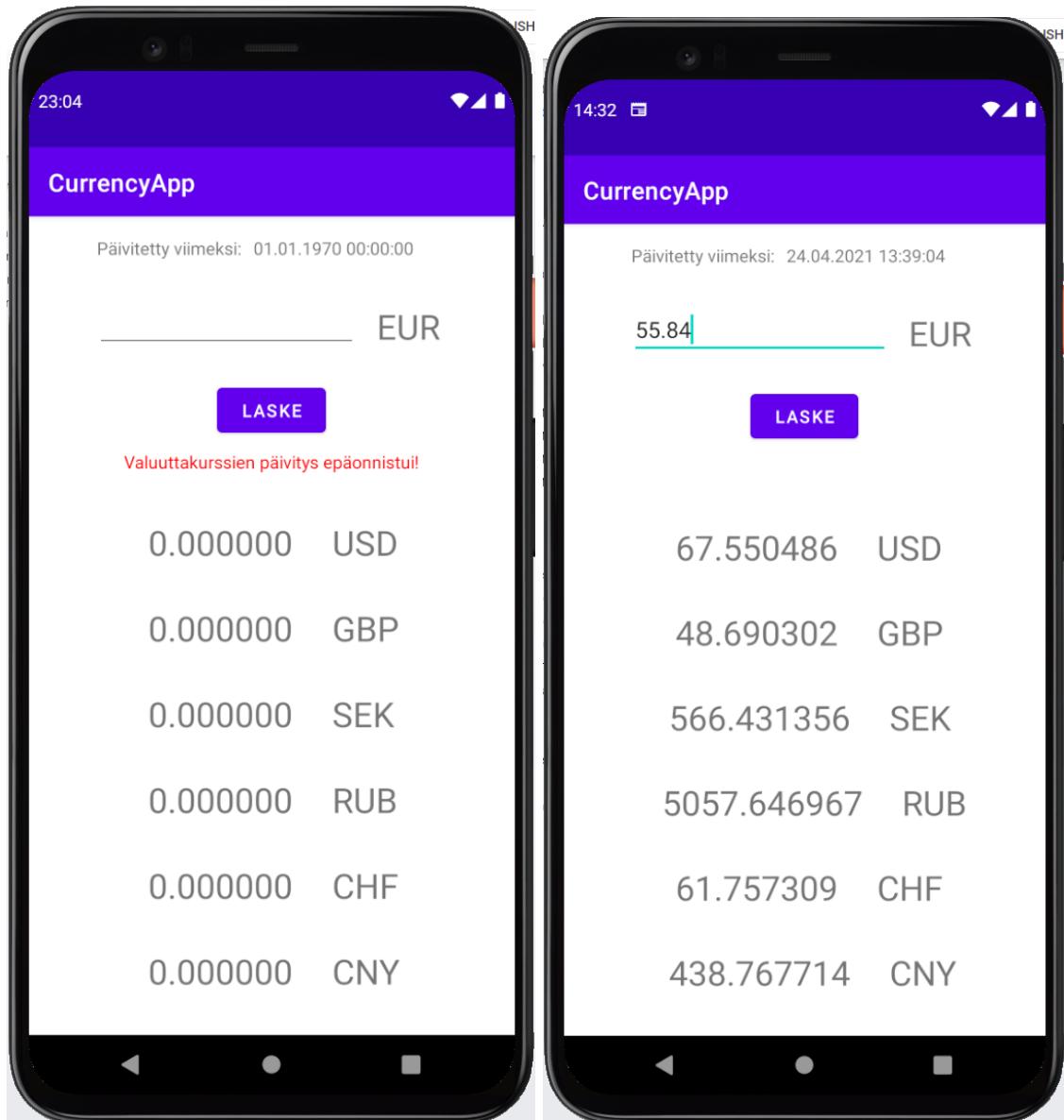
Seuraavaksi siirryin työstämään itse applikaation logiikkaa. Lisäsin alkuun manifestiin internet-oikeuden. Tämän jälkeen päädyin valitsemaan käytettäväksi kirjastoksi API-kutsuja varten Retrofit-nimisen kirjaston, joka on moderni ja käyttämältään arkkitehtuuriltaan erittäin selkeärakenteinen http-asiakaskirjasto [47]. Kirjaston kautta rakennetaan http-clientti käyttäen builderia ja syöttämällä palvelimen osoite (endpoint). Tämän jälkeen rakennetaan service-interfacet kohderesursseille vastaaviksi, jotka imevät tarvittavat parametrit ja resurssinimet funktion parametreina ja palauttavat Reponse-luokkaan wräpätynä palvelimen vastauksen rakennetta vastaavan toteutuskielen (tässä Kotlin) luokan.

JSON-objektienv konvertoimiseksi Kotlin-objekteiksi käytin Retrofitin ekstensiopalikkana suosittua Googlen toteuttamaa Gson-kirjastoa, ja sen GsonConverterFactory-komponenttia, jotta konversio tapahtuisi suoraan automaattisesti [48].

Tämän jälkeen lähdin rakentamaan tarvittavat dataluokat palvelun palauttaman json-rakenteen mukaisiksi. Toteutin seuraavaksi logiikan itse datan hakemiselle ja bindaamiselle vastaaviksi objekteiksi ja tallentamisen muistiin. Tämän jälkeen toteutin vielä logiikan taustapalvelulle tietojen päivittämiseksi, ja itse datan näyttämisen UI:ssa. Lopuksi toteutin vielä logiikan kurssien päivitysten ajastukselle, ja viimeisimmän päivitysajankohdan näyttämiselle. Käytin tässä Javan tarjoamia Timer- ja TimerTask luokkia [49, 50], joilla säädin päivitykselle ajastuksen 5 minuutin välein, jotta saataisin tuorein kurssi mahdollisimman pian sen jälkeen, kun se on palvelusta saatavilla (palvelussa se päivittyy n. 1 h intervalilla). Mikäli päivitys jostain syystä epäonnistui (verkkoyhteys poikki, palvelu alhaalla, tiedoissa virheitä, tms.), niin ajastetaan uusi päivitys huomattavasti lyhyemmän ajan päähän, jotta mahdollinen seuraava onnistunut päivitys saataisiin tehtyä mahdollisimman pian. Kirjoitushetkellä on päättetty ajaksi 5 sekuntia intervaliksi virhetilanteessa.

Mihinkään kovin suuriin ongelmiin en toteutusta väsatessa törmänyt, ja suurimmat ongelmat aiheutuivat juuri valitsemani rajapinnan rajoituksista. Palvelu tarjosi ilmaisille käyttäjille ainostaan selvätekstisen http-protokollan salatun HTTPS:n sijasta, jonka vuoksi Androidin itsesuojuvaisto heräili ja laittoi hanakasti tästä vastaan. Sain kuitenkin ylikirjoittua selvätekstieston kirjoittamalla erillisen configin, joka salli http-liikenteen juuri kyseiselle kohdedomainille poikkeuksena. Tähän löytyi onneksi apuja, jolla workaround saatiin toteutettua [45].

Lisäksi pieni haaste oli Androidin performanssисuojuksessa estää Network-kutsujen tekeminen pääsäikeessä (Main Thread), pitääkseen UI responsiivisena, heittäen NetworkOnMainThreadException -tyypisen poikkeuksen tällaisessa tilanteessa, kun retrofitillä yritti tehdä rajapintakutsua execute()-metodin kautta [47]. Tämä on erittäin hyvä asia, jotta ei vahingossaakaan tule jumiutettua käyttöliittymää raskailla network-operaatioilla. Tämän kiertäminen vaati retrofitin callback-tyylin käyttöä api-kutsussa niin, että valuutakurssien päivitys tehtiin enqueue()-metodilla, apufunktion kautta takaisinkutsuttuna, kun tulos oli vastaanotettu palvelimelta onnistuneesti kokonaan, pysähtymättä odottamaan valmistumista kutsukohdassa.



Kuva 13. Kuvankaappaukset harjoituksesta 16. Ensimmäinen kuva tilanteesta, jossa kurssien päivitys epäonnistunut virheen takia. Toisessa kuvassa onnistunut muunnos eri valuutoiksi.

2.14 Harjoitus 17

Ennen harjoituksen kimppuun käymistä perehdyin vähän aikaa XML:n käsittelyyn, ja tutkin tutkimisteni perusteella, että myös XML:n käsittely onnistuisi hyvin samalla, edellisessä tehtävässä käyttämälläni Retrofit-kirjastolla ja siihen sitten liitettyllä sopivalla XML parserilla. Päädyin siis myös tässä tehtävässä käyttämään samaa Retrofit-kirjastoa [47].

Tehtävä tuntui varsinkin alkuun, ainakin itselleni, jonkin verran haastavammaksi kuin edellinen tehtävä, johtuen palvelun rajapinnan monimutkaisudesta ja käytetyistä XML-rakenteista. Tämän vuoksi lähdin tekemään tarkempaa ja huolellisempaa alustavaa sel-

vittelyä ja suunnittelua rajapinnan käyttöön liittyen, ennen kuin hyppäsin koodin kimp-puun. Tutustuin myös rauhassa ensin tarjolla olevaan rajapintaan, sen metodeihin ja nii-den käyttämiin parametreihin, sekä rajapinnan tuottamiin tietorakenteisiin [51, 52].

Vaikka palvelun esimerkit oli laadittu käyttäen http-protokollaa, testasin, että palvelut pyörivät myös portissa 443 eli HTTPS-yhteydellä. Päädyin sovelluksessa käyttämään HTTPS-protokollaa, sillä liikenne sen läpi TLS-salattua (opendata.fmi.fi näyttäisi tukevan TLS1.2, 128-bittisellä sessioavaimella, joka on riittävän turvallinen tähän), vaikkakin kon-fidentiaalisuus tässä tehtävässä on lähinnä triviaalista (käytössä on julkisesti saatavilla oleva ja avoimesti käytettävissä oleva rajapinta).

Huomasin myös, että rajapinta etsii ilmeisesti karttapalvelun tjsp. Avulla haetun paikan ja laskee sille lähimmän mittaussensorin, ja palauttaa sen keräämän datan. Jos paikkaa ei löydy, palautuu virhevastaus:” No locations found for the place with the requested language”

Aloitin edeltävän tehtävän tapaan valmistelemalla sovelluksen lisäämällä tarvittavat oikeudet (Internet) ja tarvittavat kirjastot. Päädyin tässä käyttämään Retrofitin kanssa XML-prosessointiin kirjastoa nimeltä SimpleXML [57]. Käytin tässä vaihtelun vuoksi stringin sijaan okhttp3:n tarjoamaa `HttpUrl`-luokkaa, joka käyttää myös Builder-arkkiteh-tuuria Retrofitin clientin tapaan, ja rakensi kohde-URL:n tästä luokkaa käyttäen.

Ihan alkuun oli haasteita käsitellä Retrofitin baseurlia builder-tyyllillä, sillä Retrofitin Buil-derille ei kelvannut URL, joka sisältää paikkamääreen, ainoastaan juuritason serveri-osoite kelpasi sille. Tämä ei sinänsä ollut kuin muotoseikka, mutta vaati tarvitun servicen tekemisen hieman kompleksimmaksi.

Seuraavaksi määrititin rajapinnan palauttamien XML-rakenteiden mukaiset luokka-hierarkiat, joihin saadut XML-vastaukset mäpätään. Rakensi nämä sekä tuloksille, että virhevastaukselle. Päädyin aluksi määrittelemään nämä käsin, ja huomasin, että siihen uppoaa tolkuttomasti aikaa, ja lopputulos ei välttämättä edes toimi tai voi vaatia paljon korjailua.

Päädyin testaamaan rakennetta käytännössä niin, että asetin sovelluksen suorittamaan itsetestausta (self-test) pienen koodinpätkän avulla, joka käytännössä teki api-kutsun suoraan ohjelman käynnistyessä ajastetusti TimerTaskina. Näin paikkailin ja korjailin modeleita tehokkaasti saatujen virheilmoitusten perusteella. Tämä kuitenkin osoittautui kuitenkin lopulta erittäin työlääksi ja hankalaksi, ja lisäksi käyttämäni SimpleXML oli dep-rekoitunut, ja esim. listatyypiset elementit käyttäytyivät todella kummallisesti ja vaativat mitä ihmeellisimpiä parametreja ja annotaatioita [53].

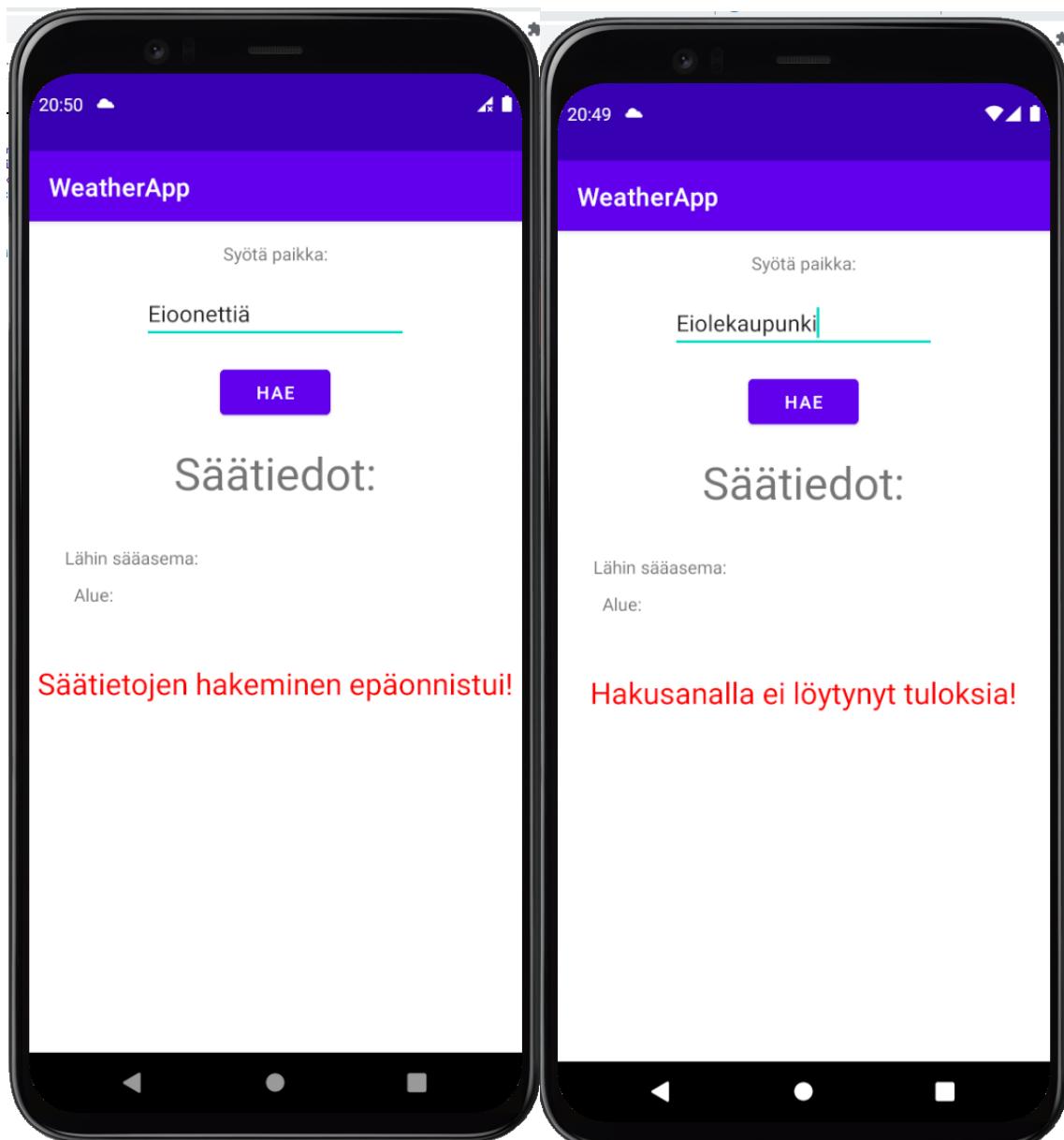
Tästä turhautuneena keskeytin sitten rakenteiden manuaalisen työstämisen, ja lähdin selvittelemään vielä uudelleen XML-spesifikaation luokkien generointia työkalulla. Tuttuun Javalla suosittuun työkaluun XJC, jolla pystyy XSD-skeemojen pohjalta generoimaan Java-luokat (muunnettavissa Kotlin-luokiksi) XML:ää vastaaviksi rakenteiksi, eli juuri siten miten olisin halunnut. Lisäksi työkalu oli saatavilla Javan JDK:ssa oletuksena mukana. [55]

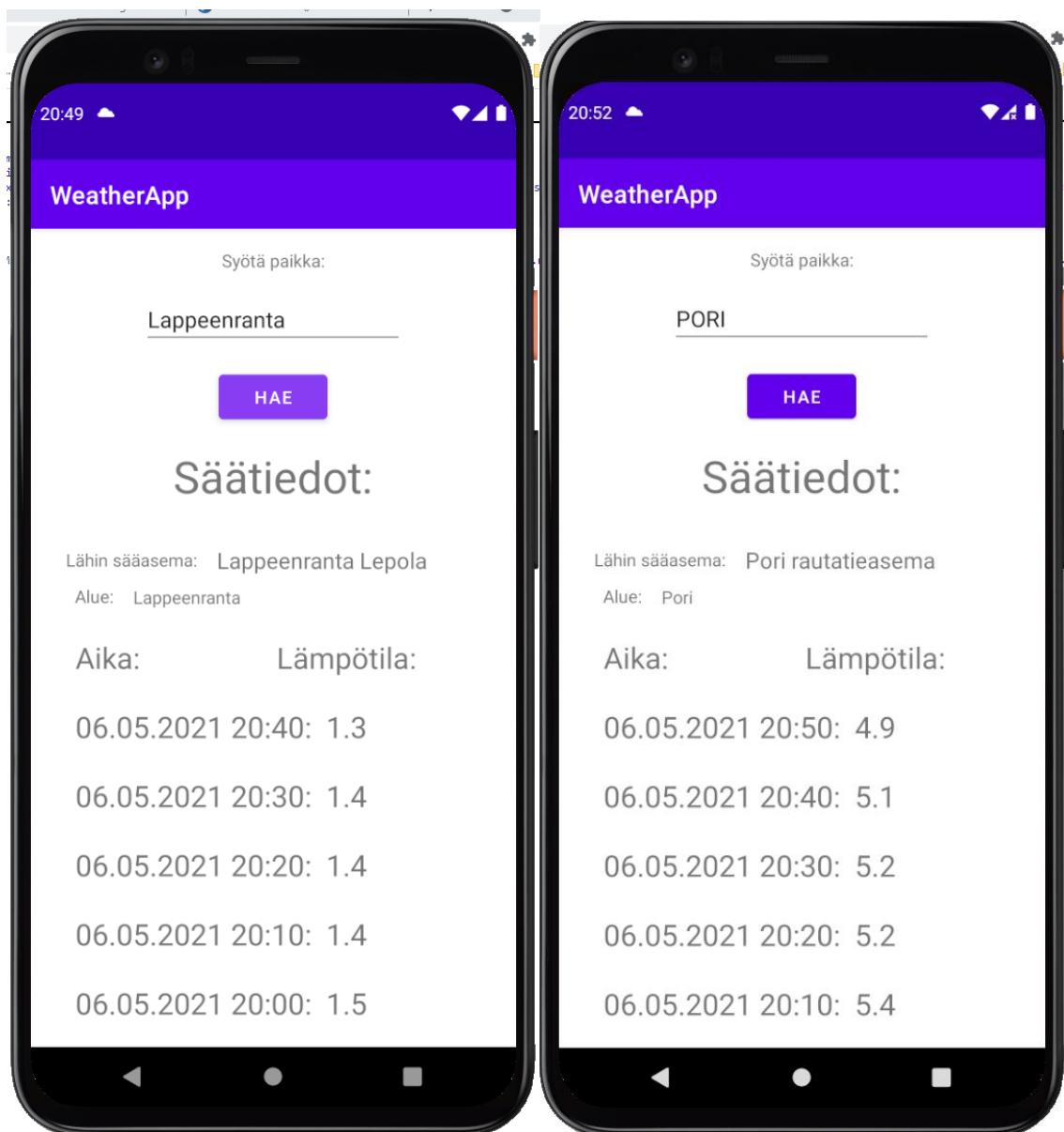
Kuitenkin vastaan tuli mitä kummallisimpia ongelmia työkalun käytössä XML:n käsitteessä, ja löysin jopa ilmatieteen laitoksen GitHubista muunnonkseen tarkoitettuja konfig-tiedostoja (mm. Xml-tyypeille XJB-määritteitä JAXB2-käyttöön) [54], mutta nämäkään eivät auttaneet eikä automaattinen käänös dataluokiksi lopulta onnistunut millään. Käsin tehdyt luokat löytyvät kuitenkin edelleen codebasesta reposta (models), jos niitä kiinnostaa tutkailla.

Päädyin sitten palaamaan takaisin manuaaliseen luokkien työstöön, mutta lopulta luovutin myös tämän kanssa, kun rakenteet eivät tahtoneet luonnistua automaattiselle parserille kelpaaviksi millään. Päädyin tämän jälkeen toteuttamaan parseri-implementaation käsin, käyttäen Androidin XMLPullParseriin pohjautuvaa XML-parseritoteutusta [57].

Aloitin ensin yritymällä työstää mahdollisimman geneerinen parseri, mutta tämänkin kanssa tuli jonkin verran hankaluksia geneerisyyden yms. kanssa, joten lopulta päädyin toteuttamaan rajapintaa varten spesifin toteutuksen XML-parserista, joka hakee XML:stä tarvittavat tiedot.

Tämän jälkeen formatoin tiedot sopivaan muotoon, ja esitin ne käyttöliittymässä. Toteutin sitten simpelin käyttöliittymän, jossa voi paikan perusteella hakea haluamaansa paikkaa, ja mikäli se löytyy FMI:n rajapinnasta, palautetaan 5 viimeisintä lämpötila-arvoa ai-kamäären kera. Mikäli paikkaa ei löydy tai ei ole esimerkiksi internet-yhteyttä, näyttää sovellus virheen mukaisen virheilmoituksen käyttäjälle säätietojen sijasta.





Kuva 14. Kuvankaappaauksia harjoituksesta 17. Kaksi ensimmäistä kuvaaa esittävät virhetilanteita (ei internet-yhteyttä, paikkaa ei löytynyt). Kaksi jälkimmäistä esittävät tulokset kahdesta eri validista lokaatiosta.

2.15 Harjoitus 18 – Palautekysely

Kurssi toimi etätoteutuksena oikein hyvin, eikä puhtaasti etäopetuksen kannalta kurssin toteutuksessa ollut mainittavia ongelmia. Luennot toimivat hyvin etänä, tarvittaessa tallenteena Teamsin kautta, ja muutoin ei ollutkaan merkittävää eroa normaaliiin.

Kurssilla käytettiin pitkälti läpi Android-kehityksen keskeisimmät elementit pääpiirteittäin. Mm. käyttöliittymän kehitys, logiikan kehitys, sensorit, kamera, paikannus, kartta, jne. Ehkä eniten hämärän peittoon jäi yhteydenmuodostus, kommunikaatiolaitteistoihin liittyvät seikat (esim. Bluetooth, NFC, ym.), pois lukien internet-yhteyden hyödyntäminen.

Näihin olisi ollut mielenkiitoista perehtyä tarkemmin, esimerkiksi tagin lukeminen tai toisen puhelimen kanssa kommunikointi NFC:llä, tai tiedon käsittely Bluetoothin avulla. Muutoin kurssin asiat soveltuivat aiheeseen erittäin hyvin.

Joskin monet aiheet ehdittiin sivuamaan vain hyvin pinnallisesti, esimerkiksi UI-elementtien hyödyntämiseen olisi toivonut monipuolisempaa käsittelyä eri elementtien käyttöön ja hyötyihin eri tilanteissa. Tarjolla olisi ollut rutkasti mm. erilaisia ryhmäytymis-, lajittelut-, widget- tyypisiä elementtejä (komponentteja), joita ei käsitelty ollenkaan. Plussaa kuitenkin RecyclerView-arkkitehtuurin läpikäymisestä, se on erittäin keskeinen (ja moderninakin) elementti datalistojen käsittelyssä.

Selvästi enemmän kurssilla kiinnosti – ja näkyi – koodi. Suurin osa luennoistakin oli pääasiassa perässä koodailua tai koodin seuraamista taikka sen esittelyä. Nykyisessä muodossaan tämä oli kurssin toteutukselle täysin luontevaa, eikä sitä olisikaan voinut oikein muutoin toteuttaa järkevästi. Tämä oli henkilökohtaisesti itselleni kuitenkin ihan ok, sillä tekeminen on aina ollut lähellä sydäntä, ja tässäkin oli mahtavaa päästä itse tekemään applikaatioita, ja toteuttamaan teoriaa käytännössä (joskin Yliopistotasolla suositaan kuitenkin teoreettisuutta enemmän).

Toisaalta itseäni kyllä kieltämättä kiinnostaisi perehtyä myös Androidin toimintoihin teoreettisellakin tasolla hieman paremmin. Kurssi voisi olla esimerkiksi jaettu kahteen osaan, joista ensimmäinen olisi selkeästi teoreettisempi, ja käsittelisi syvällisemmin esimerkiksi Sovelluksen elinkaaren toimintaa (lifecycle), jotka ovat paikoittain hyvinkin monimutkaisia esimerkiksi Konteksti-objektien ja tiettyjen näkymien kohdalla (Fragment vs. Activity vs. Application, ym.). Esimerkiksi Fragmenttejakaan ei käsitelty ollenkaan, niiden eroa ja hyötyjä pelkkiin Activity-näkymiin nähdyn olisi ollut kiinnostavaa perehtyä. Toinen vaihtoehto olisi jopa jakaa kurssi kahteen eri kurssiin, joista ensimmäinen osa käsittelisi pääasiassa teoriaa, ja toinen olisi tätä kurssia vastaava käytännön työkurssi.

En katsellut juurikaan online-kursseja tai niiden tarjontaa kurssin aikana, ja tukeuduin pääasiassa kurssin omaan aineistoon, sekä koodatessa Androidin omaa dokumentaatiota ja StackOverflown tukeen. Kotlinia varten tein läpi pari lyhyttä tutoriaalia hahmottaakseen Kotlinin perustoiminnallisuuden ja syntaksin. Koin itse riittäväksi hyödyntää Androidin omaa dokumentaatiota, joka oli mielestäni erittäin kattava, ja vähintäänkin aina sivusi työstettäviä asioita.

Harjoitukset olivat lähes kaikki miellyttäviä ja opettivat tehokkaasti, joskin osa oli ehkä vähän työläitä, ja toki niissä vaikutti oma työpanos ja laatu ja mahdolliset ylimääräiset lisäominaisuudet ja hienosäädöt, mitä ei välttämättä ollut tehtävänannossa vaadittu. Harjoituksissa oli pitkälti hyvin vapaat kädet toteuttaa harjoitus, mikä johti välillä siihen, että

myös työmäärä paisui aika lailla, vaikkakin tämä oli itseaiheutettua. Selkeästi työläimmät harjoitukset olivat 6,7,8 sekä 9,10 samaan applikaatiopohjaan yhdistetynä muodostuneena yhtenä kokonaisuutena, sekä viimeinen harjoitus 17 johtuen XML:n käsittelyn työläydestä. Harjoitukset 6–10 olivat työläitä mielestääni pääasiassa hyvällä tavalla, sillä niissä pääsi kuitenkin työllä tasaisesti eteenpäin, ja kaikki työ oli oppimisen kannalta kohtiin päin. Sen sijaan harjoitus 17 oli vähän ikävämmällä tavalla työläs, sillä siinä aikaa kului lähinnä taistellessa paikoillaan saman ongelman parissa, eikä niinkään tullut edistyä tai kasvattanut oppimiskäyrää työmäärän kasvaessa. Tuohon viimeiseen harjoitukseen olisi kaivannut ehkä enemmän tukea rajapinnan käytössä, tai vaihtoehtoisesti enemmän vapaavalintaisutta rajapinnan tai palvelun käyttöön, kuten edeltävässä harjoituksessa. Toisaalta tuli kuitenkin kerrattua XML:n käsittelyä, mikä on sinällään yleis-pätevä taito myös web-ohjelmoinnin kannalta.

Kokonaisuudessaan kurssi oli hieman työläs suhteutettuna 5 opintopisteeseen, eikä ainakaan oma työmääräni vastannut lähellekään 5 opintopisteen työmäärää vaan paisui huomattavasti sitä suuremmaksi. Eli ainakin hyvää arvosanaa tavoitellessa työmäärät eivät aivan vastaa opintopistemäärää. Toisaalta en kuitenkaan missään nimessä lähtisi karsimaan kurssin sisältöä, päinvastoin, kuten aiemmin mainittu, siihen voisi jopa lisäil-läkin vielä kontenttia. Sen sijaan opintopistemäärää voisi olla kohtuullista kasvattaa, kä-vin tämän kurssin käytännössä täysin ylimääräisenä, joten edes oma tilanne ei vaikuta tähän mielipiteeseen millään tavalla.

Pidin kurssia erittäin hyvänä lisänä koodarin arsenaaliiin, ja mobiiliipuoli on mielestääni olennaista ottaa haltuun. Kurssi oli kuitenkin pelkästään Android-alustalle pohjautuva, ja kurssilla käytettiin vain natiiveja Android-työkaluja. Itseäni olisi kiinnostanut perehtyä myös muihin alustoihin (esim. iOS), ja muihin työkaluihin, erityisesti React Nativeen, joka on moderni monialustainen framework mobiilisovellusten tekemiseen. Kurssia voisi siis ehkä muuttaa yleisempään suuntaan mobiilikehityksessä, ja huomioida myös muutkin alustat kuin Android, tai vaihtoehtoisesti sitten erotella näitä eri kursseiksi (esim. Mobiili-jatkokurssi, jne).

3. VERKKOAINEISTOT

Tässä kappaleessa on käsitelty ne verkkoaineistot, joita on hyödynnetty kurssin suoritukseen aikana. Verkkoaineistot on koottu taulukkoon, josta käy ilmi verkkoaineiston nimi, tyyppi. Lisäksi on lyhyesti ilmaistu verkkoaineiston käyttötarkoitusta (esimerkiksi harjotustestehtävän tukena) ja saatuja hyötyjä, jos näistä on koettu merkittävää hyötyä. Verkkoaineistoihin johtavat linkit on koottu lähteisiin lähdeluetteloon muodossa.

Taulukko 2. Kurssilla hyödynnetyt verkkoaineistot taulukoituna.

Aineiston nimi	Aineiston tyyppi	Käyttötarkoitus
Kurssin luennot ja luentokalvot	Luentomateriaali	Kurssin sisällön läpikäynti.
Tutorialspoint – Kotlin [7]	Kotlin online-tutoriaali.	Kotlin-kielen perusteiden läpikäynti
Android Developers [9]	Online-kurssit, dokumentaatio, codelabit, referenssit.	Androidin kehitykseen liittyvät asiat. Android-sovelluksen kehityksen tuki. Itseopiskelu eri toimintojen toteuttamiseen.
Stack Overflow [12]	Keskustelualusta ohjelmointiin liittyvistä aiheista. Toimii kysymys - vastaus periaatteella.	Referenssejä ohjelmoinnissa kohdatutujen yksittäisten haasteiden ratkaisun.
Kotlin Website [15]	Kotlin dokumentaatio, kotlinin ominaisuudet, ym.	Kotlinin toimintojen selvittäminen, saatavilla olevat tietorakenteet ja funktiot, kielen toiminnallisuus.

Google Firebase [17]	Firebaseen käyttöohjeet, codelabit, ja dokumentaatio.	Firebaseen käytönotto Android-sovelluksessa ja sen optimaalinen hyödyntäminen.
Medium [24]	Verkkosivusto asi-antuntijoiden koke- muksille.	Erilaiset haasteet applikaatiota luo- dessa, spesifi artik- keli, jossa asiaa on käsitelty.

4. LAAJA HARJOITUSTYÖ

Tässä kappaleessa käsitellään harjoitustyöhön liittyvät asiat, ja kappale sisältää harjoitustyön ratkaisun muita osin, lähdekoodia lukuun ottamatta, joka löytyy kurssille luo-dusta Git-repositoriosta Gitlabista. Kappaleessa on esitelty harjoitustyön suunnitelma, ominaisuudet, UI-design, dokumentaatio, sekä kuvankaappauksia sovelluksesta.

4.1 Harjoitustyön suunnitelma

Harjoitustyönä on tarkoitus toteuttaa sovellus, jossa sovelluksen käyttäjät voivat julkista haluamansa tekstin tai vaihtoehtoisesti kuvan, joka sitten näkyy kaikille sovelluksen käyttäjille. Käyttäjät voivat nähdä toistensa julkaisuja, ja poistaa omia julkaisujaan. Lisäksi käyttäjät voivat tarkastella muiden käyttäjien karkeaa sijoittumista karttanäkymässä (tarkkaa sijaintia ei paljasteta). Sovelluksen idea on siis samankaltainen kuin esim. Jodel - tai Twitter -sovelluksissa. Käyttäjät ovat toisilleen anonymejä, mutta käyttäjistä näytetään satunnaisarvottu numerosarja käyttäjän tunnistavana pseudonyminä. Toiset käyttäjät eivät kuitenkaan voi selvittää käyttäjän henkilöllisyyttä em. tokenin avulla, mutta esimerkiksi käyttäjät voivat nähdä, milloin sama henkilö on tehnyt uuden julkaisun (tämä toiminnallisuus saattaa vielä muuttua tai poistua!).

Sovelluksessa on 4 näkymää: kirjautumisnäkymä, jossa käyttäjä voi kirjautua tai rekisteröityä. Kirjautuminen tapahtuu Googlen Firebase-pilvipalvelua käyttäen. Uudet käyttäjät voivat rekisteröityä esim. Googlen tai Facebookin kautta, ja käyttäjiltä tallennetaan vaa-ditut autentikointitiedot rekisteröitymisen jälkeen. Lisäksi käyttäjälle arvotaan pseudonyymi identifioiva numerosarja. Numerosarja on 8 merkkiä pitkä, sisältää numerot 0-9, eli numeroavaruus yhteensä $10^{10} = 10\ 000\ 000\ 000$ mahdollista käyttäjää. Myös ar-vonnalle on jätettävä riittävästi pelivaraa satunnaisgeneroinnin törmäysten minimoi-miseksi. Tarkistus törmäysten varalta kuitenkin tehdään.

Seuraavaksi käyttäjä siirtyy pääänäkymään, jossa näkyvät kaikkien käyttäjien julkaisut RecyclerView-listänäkymässä. Julkaisujen tietokantahauissa pyritään hyödyntämään si-vutusta, eikä hakea kaikkia julkaisua kerralla, jotta sovellus ja tietokanta eivät kuormit-tuisi liikaa. Julkaisut voivat olla tyypiltään teksti- tai kuva-julkaisuja. Näkymän ylä- tai ala-reunassa on tekstilaatikko, ja sen vieressä lähetyspainike. Kun tekstikenttään on kirjoi-tettu tekstiä, ja painetaan lähetyspainiketta, muodostuu uusi julkaisu, joka päivitetään listänäkymään. Sovellus käyttää Firebase-pilvikantaa käyttäjien julkaisuiden tallennuk-seen (käyttäjän ID, teksti tai kuva). Käyttäjä voi nähdä kaikkien käyttäjien julkaisut, mutta

ei muuttaa muiden käyttäjien julkaisuja. Käyttäjä voi poistaa omia julkaisujaan, mutta ei kuitenkaan muokata niitä. Mikäli julkaisu poistetaan, siitä piilotetaan sisältö, mutta varsinainen julkaisu jäetään näkyviin esim. placeholder-tekstillä ("Tämä julkaisu on poistettu."). Muut mahdolliset tietokentät, kuten käyttäjän ID, julkaisupäiväys, karkeा sijainti, yms. jäetään näkyville. Näin estetään toiminnan väärinkäytö ja säilytetään jäljitettävyyss hallintotasolla mahdollisia väärinkäytö- tai vianetsintätapauksia varten.

Sovelluksessa voi julkaista myös kuvan, jolloin käyttäjältä kysytään käyttöä kameraan, tai hakemaan kuva Android-laitteen tallennustilasta. Kuvan lisäys postaukseen tapahtuu liitekuvaketta painamalla, tekstikentän vieressä. Kun kuva on otettu tai valittu, se julkaisaan tekstin tilalla. Mahdollisesti julkaisuun voi lisätä molempia, riippuen toteutuksen käytännöllisydestä.

Omat julkaisut -näkymässä näytetään RecyclerView-listausnäkymässä kaikki käyttäjän omat julkaisut. Lisäksi julkaisunäkymän yhteydessä niitä on mahdollista selata tai poistaa. Näkymästä on esimerkiksi helpompi nähdä myös vanhemmat omat julkaisut, jotka ovat hautautuneet pääänäkymässä selattavuuden kannalta liian kauaksi. Tähän näkymään ei toistaiseksi ole suunnitteilla muita toimintoja.

Karttanäkymässä on mahdollista selata käyttäjien määrä eri sijainneissa karkealla tasolla. Karttanäkymä toteutetaan Google Mapsin tarjoamien rajapintojen avulla. Näkymässä on esimerkiksi kaupunkien tai kaupunginosien kohdalla pallo, ja pallon keskellä numero osoittamassa aktiivisten käyttäjien lukumäärää. Tähän voi teknisellä tasolla tulla vielä muutoksia, riippuen toteutuksen vaikeuksista ja GDPR:n tuomista vaatimuksista ja rajoitteista. Erittäin pieni käyttäjämäärä voidaan esimerkiksi näyttää karkeasti ("< 10").

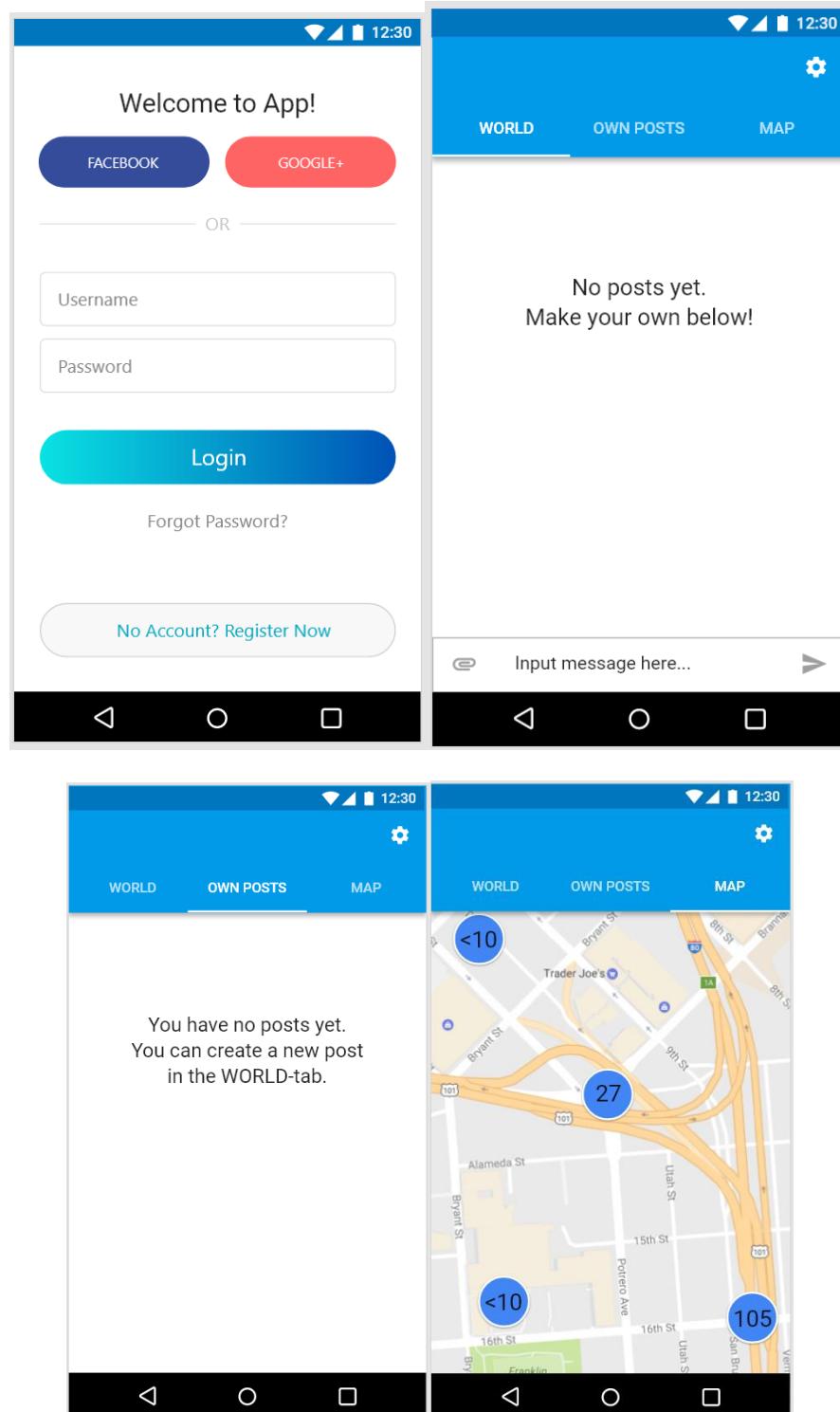
Sovelluksessa on koko sovelluksen ajan näkyvä navigointipalkki, jonka avulla eri näkymien välinen navigaatio onnistuu. Navigointipalkin lopullinen sijainti ei ole vielä päättetty, mutta alustavasti se sijaitsee sovelluksen ala- tai ylälaidassa oleva välilehijono.

Myös jonkinlainen asetusnäkymä, josta voisi tarkastella käyttäjän omia tietoja, tai esimerkiksi tehdä tietojen tarkastuspyyynnön (GDPR) ja poistaa oman tilinsä sekä kaikki tilin tiedot pysyvästi, on suunnitteilla sovellukseen.

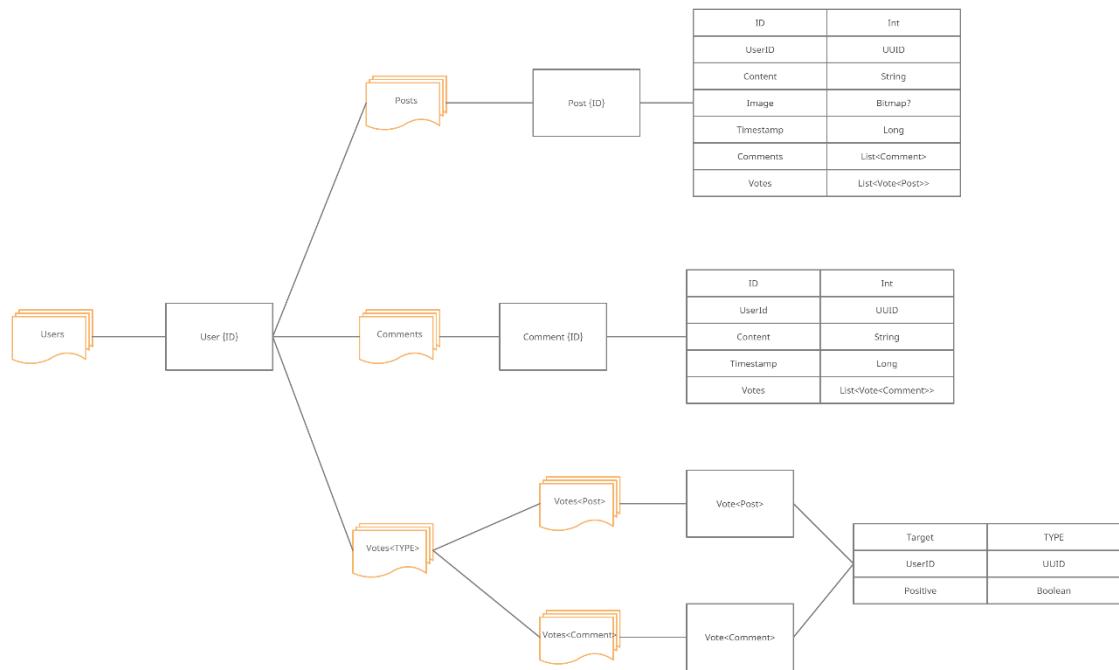
Mahdollisina lisäominaisuksina sovellukseen voidaan tietokannasta tai jopa mahdollisesti käyttöliittymätasolla assignoida erityisiä moderaattorikäyttäjiä, jotka kykenevät poistamaan (tai ehkä muokkaamaan) myös muiden käyttäjien kuin oman postauksia. Lisäksi lisäominaisuutena voisi olla mahdollisuus raportoida sovelluksen ylläpidolle (esim. em. moderaattoreille tai johonkin moderaattorikanavaan) väärinkäytöstä, tai muutoin

epämiellyttävistä, epäasiallisista tai muutoin sopimattomista julkaisuista, jolloin ne voidaan tarkastaa, ja mahdollisesti esimerkiksi poistaa. Lisäksi tapauksissa, joissa toiminta on jatkuva, käyttäjä voidaan estää kokonaan palvelusta.

4.2 Harjoitustyön design



Kuva 15. Kuvankaappauksia harjoitustyön applikaation alustavasta designista.



Kuva 16. Karkeaa tietomallinnus sovelluksen käyttämistä tietorakenteista.

4.3 Harjoitustyön toteutus

Aloitin harjoitustyön toteutuksen luomalla uuden Android-applikaation Android Studiolla tuttuun tapaan, päivittämällä konfiguraatiot ja versiot, ja kommitoimalla pohjan Git:iin. Tätä ennen päivittelin vielä kaikki työkalut uusimpiin versioihin, mukaan lukien Studion, Kotlinin, Gradlen, käytetyt kirjastot, ym. Lisäksi päivitin emulaattorin sovellukset ja Play-palvelut. Harjoitustyön sovelluksen nimivalinnoista päädyin lopulta valitsemaan nimen "Disqs", joka kuvailee hyvin sovelluksen käyttötarkoitusta, mutta on myös modernisoitu ja mahdollisimman lyhyt.

Projektiin alkuvälmiesteluiden jälkeen lähdin työstämään itse sovellusta. Konfiguroin alkuun projektille ympäristön Googlen Firebaseen, ja tämän jälkeen määritelin myös projektin käyttämään tästä ympäristöä, jotta Firebasen työkaluja voisi projektissa käyttää.

Aloitin sitten työstämään pääänäkymää designia mahdollisimman hyvin seuraten. Aloitin toteuttamalla "Tabbed Activity" -tyyliisen näkymän, käyttäen pohjana sen tarjoamaa viewholderia, adapteria ja fragment-pohjaista ratkaisua [58]. Tabit oli tässä toteutettu yhden Acitvityn sisään erillisinä näytettävinä Fragmentteina Activityn sisällä, joiden käsitteilyä ohjaa SectionsPagerAdapter. Lähdin tämän päälle toteuttamaan näkymiä kaikkiin postauksiin (World), omiin postauksiin (Own Posts), sekä karttanäkymään (Map).

Seuraavaksi siirryin toteuttamaan tarkemmin postausnäkymää, joka käyttää pohjanaan aiemmista tuttua RecyclerView-ratkaisua. Toteutin hieman aiempaa tähän liittyvää harjoitustehtävää mukailleen RecyclerView-näkymän Fragmentin sisään, ja tälle LiveData-ratkaisun ViewModelin sisään, sekä testidatan generoinnin näkymän testaamiseksi. Toteutin tähän modernin, dynaamisen tietojen päivitysmahdollisuuden ("pull-to-refresh" - tyylilä noudattavan) käyttäen SwipeRefreshLayout kirjastoa ja sen tarjoamaa refresh-kuuntelijaa [59].

UserID:nä käytetään tässä alkuperäisestä speksistä poiketen UUID-tyyppiä, joka löytyy Javasta sisäänrakennettuna. Kyseessä on siis (Immutable) "Universally Unique Identifier" (UUID) [60, 61] eli universaalisti identifioiva tunniste, joka nimensä mukaisesti viittaa universaalisti ainutkertaiseen tunnisteeseen, joka on tyypillisesti kerran luotu, myöhemmin muuttumaton eli pysyy luontihetkestä käytön lopetukseen asti muuttumattomana, yksilöiden yhtä ja tiettyä asiaa. Tämä oli numeroarvoa viisaampi ratkaisu, sillä oletusarvoisesti nämä tunnistheet ovat aina yksilöllisiä, ja niiden arvoavaruuus on riittävän suuri siihen, että törmäyksiä ei tarvitse pelätä, eikä erikseen tarkistaa [60]. Näin säästyy huomattavasti aikaa ja tehoa. Sen sijaan postauksilla ja kommenteilla on omat IDnsä, jotka ovat numeroarvoisia järjestyslukuja, jotta niiden avulla olisi helppo tehdä suodatusta ja lajittelua. TODO

Postausosion alettua näyttämään hyvältä, työstin sen pohjalta kommenttiosion, joka on hyvin pitkälti samankaltainen, muttamilla muutoksilla. Tälle tein omat layoutit ja loin tarvittavat luokat logiikan pohjaksi. Lisäksi toteutin myös näkymän pelkästään omille postauksille, joka käyttää periaatteessa täysin samaa logiikkaa kuin yleinen postausnäkymä, mutta suodattaa postauksen userID:n mukaan, eli pelkästään käyttäjän omat postaukset.

Lisäksi toteutin logiikan postausten ja kommenttien äänestykselle. Äänen postaukselle voi antaa oikeasta reunasta käyttäen + ja – painikkeita. Äänen per postaus tai kommentti, voi kuitenkin antaa vain kerran, eikä sitä voi muuttaa, joten ääntä annettaessa on oltava tarkkana. Tämän takana toimii yksinkertainen Vote-objekti, joka säilöö tiedon äänen suunnasta (positiivinen tai negatiivinen) yhden boolean-muuttujan avulla, lisäksi käyttäjän joka äänen antoi, jotta voidaan estää duplikaattiäiset, ja myös ajanhetken, jolloin ääni annettiin. Kun recyclerview ladataan, adapteri prosessoi jokaisen itemin kohdalla sen saadut äiset, ja mapittaa boolean arvon joko 1 = true tai -1 = false, ja listaa lopullisen summan äänimäärälle varattuun tekstikenttään. Jos käyttäjä antaa uuden äänen, tämä tieto päivitetään suoraan UI:hin, jotta muutos näkyisi välittömästi, mutta myös tietokantaan (joka menee sitten hieman viiveellä).

Saatuani postausnäkymät kunnolla toimimaan dummy datalla, lähdin työstämään tietokantalayeria Firestoren käyttöä ja oikeaa dataa varten. Tutkailin vielä eri tietokantoja ja vertailin Realtime Databasea ja Firestorea, mutta erityisesti hierarkisuuden ja sisäkkäisyksen kautta päädyin kuitenkin yhä samalle linjalle eli Firestoren käyttöön. Aloitin konffaamalla Firestoren Firebasessa, ja tekemällä tarvittavat konfiguraatiot projektiin [17, 1]. Tämän jälkeen lähdin korvaamaan dev-datankäsittelyn toteutuksia tietokantatoteutuksella. Dataluokkia joutui hieman tuunailemaan, jotta ne olisivat yhteensopivia Firestore-kannan typpien kanssa, esim. UUID:n tilalle String, ja lokaation tilalle GeoPoint. Tämän lisäksi default constructor-tarpeen poisto eli kaikista propertyistä nullable ja oletusarvoksi null. Pienen simplifioinnin jälkeen datatyypit näyttivät kelpaavan hyvin Firestorelle.

Seuraavaksi ongelmaa tuli heti asynkronisuuden kanssa – Firestore toimii pelkästään asynkronisella periaatteella, joten koodi on rakennettava siten, että se reagoi muutoksiin vasta kun tietokannan tulokset ovat saatavilla, johon voi mennä määrittelemätön aika. Yritin ensin ratkaista ongelmaa pakottamalla tietokantahaut synkronisiksi, mutta tässä huomasi nopeasti, että se ei ole ollenkaan järkevää ja lisäksi äärimmäisen hankalaan saada edes toimimaan. Lopulta päädyin toteuttamaan tietokannan käsittelyn callback-paradigman avulla luomalla Callback-interfacen, joka toteuttaa kaksi funktiota, onResult jota kutsutaan kun tietokantaoperaatio onnistuu ja siinä on dataa saatavilla, tai vaihtoehtoisesti onError funktio jota kutsutaan jos käsittelyssä kohdataan jokin virhe tai dataa ei löytynyt. Tällöin kutsujan on suoritettava datan käsittelyyn halutut operaatiot vasta kun onResult-funktiota kutsutaan. Tämä oli käytännössä lähinnä järjestelykysymys koodissa.

Seuraavaksi kohtasin ongelmia hierarkioiden käytännön toteutuksessa. Esimerkiksi huomasin, että jos postaukset asettelee userien alikokoelmiksi, on käytännössä mahdotonta hakea kaikkia postauksia kaikilta usereilta järkevästi. Sen sijaan esimerkiksi kommentit postausten subcollectionina ja votet edellämainittujen subcollectioneina toimi taas järkevästi, sillä tässä tapauksessa ei ole tarvetta hakea kommentteja tai ääniä useammasta kuin yhdestä postauksesta tai kommentista, ja siksi näiden kanssa hierarkisointi toimi odotetusti. Muutin siis ratkaisua hieman siten, että userit ja postaukset jaettiin kokonaan omiin kokoelmiinsa.

Jouduin vielä jonkin verran taistelemaan ideksoinnin kanssa, sillä osa kyselyistä koostui useampaan kenttään liitetystä filteroinnista ja järjestelystä, jonka vuoksi Firestore vaati luomaan kantaan ideksejä näihin kenttiin perustuen, esim. poistettujen postausten poissuodatus ja tämän jälkeen järjestely postausID:n mukaan laskevasti tuotti haasteita. Tässä oli aluksi ongelmaa, sillä oli vaikea hahmottaa miten indeksit olisi järkevä luoda.

Myöhemmin onnekseen lokituksen kautta virheviestin mukana tuli valmiiksi generoituja linkkejä tarpeellisten indeksien luontiin, ja näitä pystyi sitten käyttämään indeksien generointiseksi oikeanlaisiksi, ja ongelma(t) indeksoinnin kanssa lopulta ratkesivat.

Päädyin myös toteuttamaan autentikoinnin sovellukseen sellaisella logiikalla, että myös sovelluksen anonymi käyttö on mahdollista. Jos käyttäjä päätyy käyttämään sovellusta anonymisti, luodaan käyttäjälle lokaalisti tallennettu käyttäjä, josta välitetään vain uniikki ID tietokantaan postausten, kommenttien ja äänien mukaan. Tällöin, jos käyttäjä nollaavat sovelluksen datan esimerkiksi, hallinta käyttäjään menetetään lopullisesti, ja sen pystyy kaivamaan takaisin vain, mikäli tietää UserID:nsä, ja on sovelluksen kehittäjä, eli pystyy pakottamaan oman ID:nsä toiseksi. Jos käyttäjä siis haluaa, että omat sovelluksessaan luodut tietonsa pysyvät hallittavissa myös sovelluksen nollauksen ja poiston yli, on sovelluksen rekisteröidyttävä jollakin autentikointitavalla, jotta linkitys käyttäjän identiteetin ja UserID:n välillä voidaan muodostaa.

Päädyin tässäkin siihen tulokseen, että on järkevä hyödyntää valmista kirjautumisen käytöliittymän toteutusta Firebasen tarjoamana, jolloin riittää kirjautumisen konffaust ja login-flown rakentaminen toimivaksi ja sujuvaksi. Hyödynsin tässä vanhastaan harjoituksesta tuttuja tutoriaaleja, ja tuttuun tapaan autentikaation käyttöönotto sujui varsin ongelmitta. Myöhemmin olemassaolevien käyttäjien synkkaus, uusien luonti ja näiden erotteleva toiminnallisuus tuotti hieman haasteita, mutta kaikki näiltä osin kuitenkin saatiin ratkaistua. Otin UI:ssa käyttöön kirjautumiset Anonymisti, Googlen, Facebookin, Twitterin, Microsoftin, Githubin, Yahoon, Applen, ja Puhelinnumeron, sekä sähköposti-salasanayhdistelmän avulla. Lisäksi disabledin duplikaattikäyttäjät (samalla sähköpostilla uudelleen kirjautuva ohjataan oikeaa kirjautumisreittiä pitkin). Lisäksi varmistin, että näiden tapojen kautta kirjautuminen onnistuu sujuvasti. Osassa joutui tekemään enemmän konffausta itse palveluntarjoajien päässä, asettamaan Callback- ja datanpoisto- URLleja ja logiikkaa, hakemaan autentikaatiotunnusia ja avaimia Oauthia varten, yms. Lopulta toteutin autentikoinnin yhteyteen vielä logiikan jolla anonymit tilit pystytään myöhemmin linkittämään kirjautuneeseen käyttäjään. Tässä hyödynnettiin lokaalia applikaation tallennustila, ja käyttäjälle luodaan paikallisesti JSON-tiedosto, johon ensimmäisellä kerralla luodaan uusi käyttäjä, myös silloin kun käyttäjä kirjautuu anonymisti. Tähän liittyen rakenzin valikon, josta käyttäjä voi linkittää anonymin tilinsä todelliseen rekisteröityyn käyttäjätiliin. Tämän jälkeen kun käyttäjä suorittaa synkronoinnin, käyttäjän lokaalit tiedot, kuten sovelluksen oma userID, asetetaan pilvikäyttäjän tietoihin, jolloin esimerkiksi anonyminä luodut postaukset säilyvät käyttäjän hallinnassa. Mikäli käyttäjällä on kuitenkin aikaisemmin ollut jo tili samoilla tunnuksilla, priorisoidaan tunnusten takana oleva data,

eli esimerkiksi tällaisessa tilanteessa anonymi käyttäjä korvautuu olemassaolevan käyttäjän tiedoilla, ja omistus anonymeihin postauksiin menetetään. Lisäksi käyttäjän on mahdollisuus kirjautua ulos tilitään menun painikkeen kautta, ja halutessaan kirjautua uudelleen joko anonymisti tai sitten toisella käyttäjätilillä resetoimatta sovellusta.

Toteutin myös kuvan oton mahdollisuuden postauksen yhteyteen käytäen kamerapainiketta tekstikentän vasemmalla puolella. Kuvan lisäys toimii seuraavalla logiikalla: käyttäjä painaa kamerapainiketta, tällöin aukeaa kameränäkymä (kunhan lupa kamerankäytöön on myönnetty). Seuraavaksi käyttäjä voi perua operaation tai ottaa kuvan. Kun kuva on otettu, käyttäjä voi vielä perua kuvan ja ottaa uuden, tai hyväksyä kyseisen kuvan. Kun käyttäjä hyväksyy kuvan, palataan takaisin chattinäkymään, ja kuva on nyt liitetynä mukaan. Syötettyään vielä postaukselle tekstin ja painettua lähetysnappulaa, ilmestyy uusi postaus kuvan kera listalle. Kun kuvan otto ja liittäminen postaukseen onnistuu, tulee tästä ilmoitus käyttäjälle ruudun yläreunaan.

Toteutin kameran käytön käytäen uusinta CameraX JetPack-kirjastoa [30]. Loin kameränäkymälle erillisen fragmentin, joka lätkäistää nykyisen fragmentin tilalle, kun photopainiketta painaa. Kyseinen fragment suorittaa sitten kaikki kameraan liittyvän, eli oikeuksien tarkistus & pyyntö, kameran käynnistys, kuvan tallennus, ja kuvan välitys eteenpäin sopivassa muodossa, eli Bitmäppinä. Kun fragment käynnistyy, se alustaa näkymän, yrittää käynnistää kameran ja liittää sen fragmentissa olevaan Preview-näkymään, josta käyttäjä voi seurata kameran kuvaa reaalialkaisesti. Kun käyttäjä ottaa kuvan kuvauspainikkeesta, preview pysytetään, ja käyttäjältä kysytään, haluaako tämä käyttää kyseistä kuvaa vai ottaa uuden. Jos päädytetään uusimaan kuvan, käynnistetään kamera uudestaan. Kuvan ottamisen (tai peruutusnapin painalluksesta, jos kuvaa ei ole vielä otettu) activity palaa maailmafragmenttiin, ja ilmoittaa kuvan lisäyksen onnistumisesta (jos sellainen otettiin). Tämän jälkeen tekstikenttään vielä syötetään jotakin ja käyttäjän painaessa lähetysnappulaa, läheee uusi postaus kuvan kera. Postauksessa näkyy pienkuva, jonka vuoksi tähän implementointiin vielä erillisen kuvanäkymän (myös oma Fragment), joka näyttää kyseisen kuvan kokoruudussa. Kuvasta pääsee pois joko Paluunappia painamalla, tai sitten Uin kautta Back-nappulasta.

Ainoaksi ongelmaksi jota en saanut ratkaistua, jäi kuvanäkymän kuvan skaalaus koko ruudun täyttäväksi, tai edes vähän isommaksi. Tämä ei onnistunut järkevästi kuvaa skaalailemalla, kuvan kokoa muuttelemalla tai kuvan luontia uudelleen skaalattuna, vaan näissä joko syntyi kummallisia kaatumisiin johtavia virheitä tai sitten kuvan kuvasuhde meni niin rikki tai kuvasuhde niin huonoksi, että skaalaussessä ei ollut enää järkeä.

Toteutin vielä viimeisenä sovellukseen karttanäkymän, jossa on tarkoituksesta näkyä karkea sijainti sovelluksen kaikista käyttäjistä. Tarkoitus on hakea käyttäjän oma sijainti karttanäkymää avattaessa, ja tämän jälkeen sitten näyttää muiden käyttäjien karkea sijainti. Sovellus siis hakee ensin käyttäjän sijainnin, laskee siitä lähimman isomman keskittymän, esimerkiksi kaupungin tai kylän, ja merkitsee käyttäjän sijainnin siihen. Sitten se synkronoi tiedot kantaan, ja hakee muiden käyttäjien sijainnit vastineeksi, ja renderöi ne karttanäkymään.

Käytännössä tämän toteutus tapahtui myös vanhastaan tutun Google Maps SDK:n avulla. Enabloin Google Cloudin konsolista Mapsin SDK:n tälle projektille, sekä hain tarvittavan api-avaimen. Konffasin projektin ja tarvittavat kirjastot mapsin käyttöön. Tämän jälkeen lähdin työstämään sovelluksen karttanäkymän Fragmenttia. Tein koko fragmentin täytyväni karttanäkymän, johon konfiguroin Mapsin. Alkuun oli ongelmaa työstää navigatiota ja siihen reagointia järkeväksi, ja erityisesti tuotti ongelmia tunnistaa karttanäkymässä, kun käyttäjällä on se aktiivisena, tai päinvastoin kun Android luo näkymän vielä taustalla. Tähän yritin aluksi implementoida OnTabSelectedListener-luokkaa, jolla voisi OnTabSelected(tab) overridatulla metodilla tunnistaa tutkimalla onko tab.position Map-näkymän kohdalla. Tässä tuli kuitenkin ongelmia saada konteksti ja activity käyttöön, ja erittäin hankalasti ratkaistavia kaatumisia, joten tämän ratkaisun hylkäsini, ja lähdin sen jälkeen kokeilemaan erittäin hyvällä tuurilla löytynytä vihjetä käyttää sen sijaan natiivisti overridattavaa SetUserVisibleHint-metodia [63, 64]. Tällä pystyi tunnistamaan, kun Fragment on käyttäjällä näkyvillä, lisäksi pieni activity-tarkasteluun yhdistämällä ja pienellä kikkailulla onnistui vihdoin erottaa, milloin käyttäjä todella on näkymässä, ja haluttaa elementtejä sen mukaisesti järkevästi, muun muassa kysyä oikeuksia sijaintiin vain kun näkymä on avoinna, tai populoida karttaan dataa vain tällaisessa tilanteessa. Muutoin karttanäkymä on liitettynä fragmentin omaan elinkaareen (oncreate, oncreateview, onresume, onpause, ondestroy, ym.)

Kun sain karttanäkymän pelittämään hyvin, lähdin seuraavaksi työstämään itse sijaintia. Tässäkin hyödynsin vanhastaan tuttua FuseLocationProvider luokkaa ja sen johdannaisia, sekä LocationCallback- ratkaisua sijainnin hakemiselle [36, 62]. Alustavan viimeisimmän sijainnin sai suoraan providerista, ja callbackilla kerättyä LocationRequestin tuottamia uusia sijaintipäivityksiä. Kun sain sijainnin kunnolla pelittämään, lähdin rakentamaan logiikkaa karttapisteiden ym. Näyttämiselle. Päädyin toteuttamaan sen sellaisella pseudologiikalla, että ensin haetaan käyttäjän lähin isompi keskittymä, joka ei välttämättä ole aina käyttäjän tarkka sijainti, mikä on toivottavaakin yksityisyden kannalta. Tämän jälkeen tähän pisteeseen piirretään 10km säteinen ympyrä merkiksi siitä, että

tällä alueella on käyttäj(i)ä. Lisäksi kohtaan asetetaan merkki, jonka sisältönä on käyttäjien määrä alueella. Hyödynsin tässä Googlen tarjoamaa Places SDK ta, jonka avulla pystyy hakemaan sijainnin perusteella sen lähistöllä sijaitsevia kohteita, paikkoja esim. yrityksiä, kaupunkeja, nähtävyyksiä ym. Ja hakemaan näiden tietoja, kuten nimen, osoitteen, geopisteen, arvionnit, yms. Tämän jälkeen toteutin logiikan, jolla käyttäjän tietoihin kirjataan kyseinen paikka lähimmäksi pisteksi, ja tämän jälkeen haetaan kaikki sovelluksen käyttäjät, joilla on vastaavasti lähin piste kirjattuna, ja sovellus asettaa näihin kohtiin merkit sijaintien mukaisesti karttanäkymään. Lisäksi ympyröiden piirtoa on rajoitettu niin, että kahta ympyrää ei piirretä lähemmäksi kuin 70 % ympyrän säteestä eli 10 km:stä, vaan tällöin ne yhdistetään samaksi, ja käyttäjien lukumäärä ympyrässä kasvaa. Nämä vältytään liian monelta päälekkäiseltä ympyrältä.

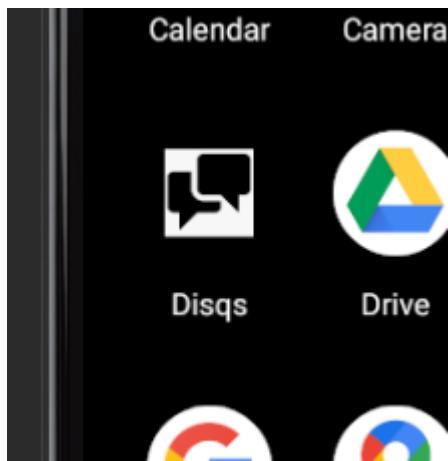
Tein lisäksi vielä kaikista teksteistä käänökset suomen kieleen, ja tästä oman string-xml-tiedoston sisältäen käänökset. Lisäsin myös sovellukselle oman logon, sekä tein sovellukselle julkaisun Play-kauppaan Play Consolen kautta, tuttuun tapaan, kuten aiemmassa harjoituksessa.

4.4 Harjoitustyön käytöohjeet

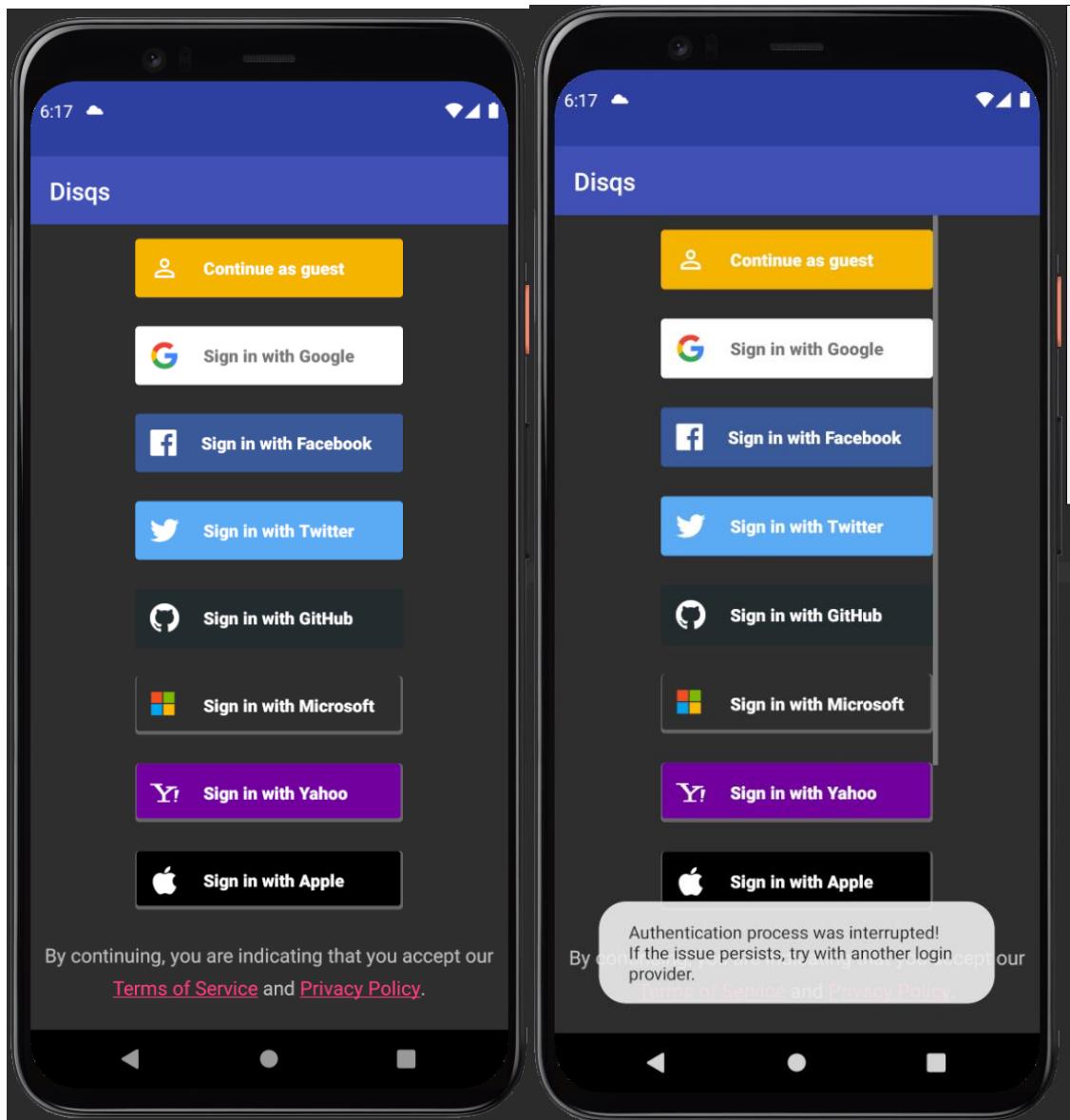
Sovellus on siis sosiaalisen median sovellus, jossa voi julkaisuta viestejä ja kuvia muille käyttäjille, lukea ja kommentoida omia sekä muiden käyttäjien julkaisuja, antaa plus- tai miinuspisteitä julkaisuille ja kommenteille, sekä lisäksi vielä seurata käyttäjien määriä ja karkeita sijainteja sovelluksessa.

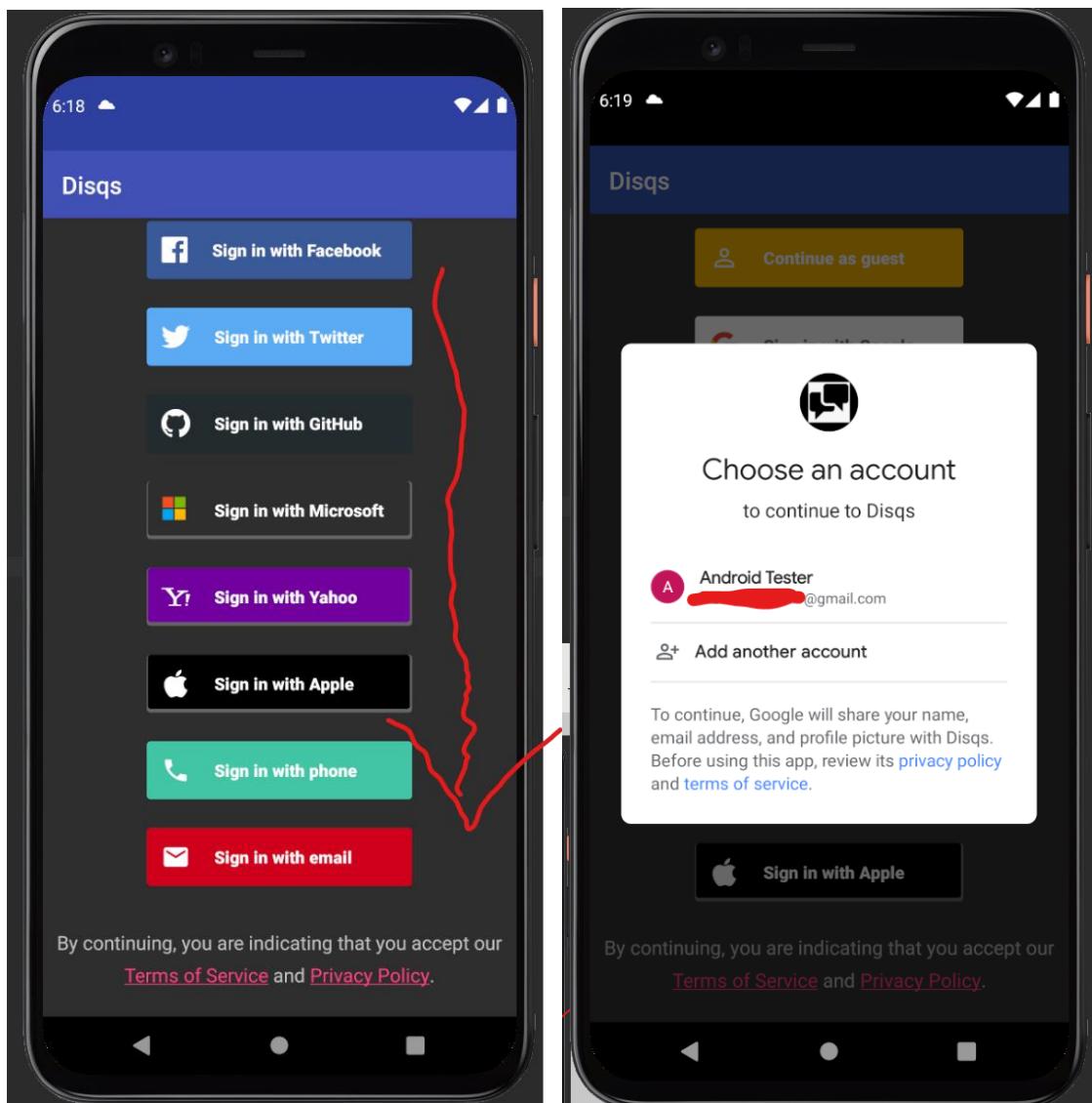
Sovellus on saatavilla Play-kaupassa linkin takaa [X]. Lisäksi sovellus pitäisi löytyä ihan nimellä hakemalla suoraan kaupasta. Sovelluksen alin tuettu SDK-versio on 26 (versus uusin 30), johtuen tietyistä sovelluksen käyttämistä kirjastoista ja työkaluista. Mikäli jostain syystä päätelaitte ei tue tätä SDK-versiota, sovellusta ei pysty asentamaan laitteelle ja on käytettävä joko toista (todennäköisesti uudempaa) laitetta, taikka sitten emulaattoria. Sovellus on lähtökohtaisesti tuettu (ja testattu) vain puhelimissa, ei esimerkiksi tabletteissa tai muissa laajakulmanäytöllä laitteissa.

Sovellus asennetaan kaupasta painamalla Play-kaupasta sovellussivulla Lataa/Asenna-painiketta (Install). Tämän jälkeen se asentuu laitteelle. Kun sovellus on asennettu, se avataan normaalisti Androidin sovellusvalikosta.

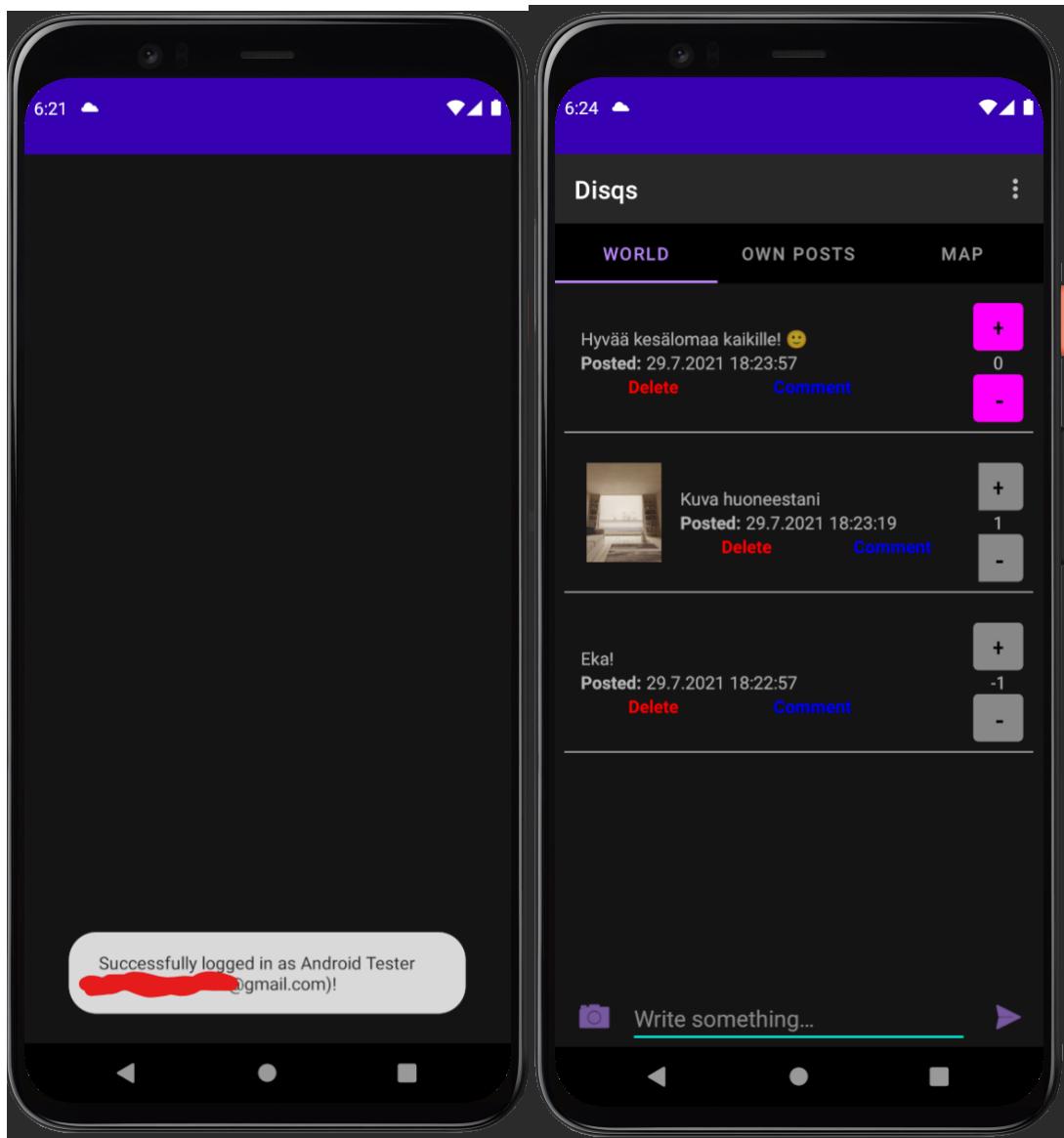


Ensimmäistä kertaa sovellusta avattaessa sovelluksessa aukeaa kirjautumisnäkymä. Tästä voi valita mieluisensa kirjautumistavan, tai esimerkiksi kirjautua anonyminä käyttäjänä. Näytöllä näkyy kerrallaan vain osa kirjautumistavoista, skrollaamalla listaa alas-päin saa näkyviin myös loput kirjautumisvaihtoehdot. Mikäli kirjautuminen epäonnistuu, voi esimerkiksi kokeilla toista kirjautumistapaa, jos jossain kirjautumistavassa on esimerkiksi tilapäinen käyttöhäiriö tai on tapahtunut odottamaton konfiguraatiomuutos, tms. Tästä tulee erillinen virheilmoitus käyttäjälle Toast-tyypillisellä ilmoituksella, ja käyttäjää ohjataan joko yrittämään uudelleen tai kokeilemaan toista kirjautumistapaa.

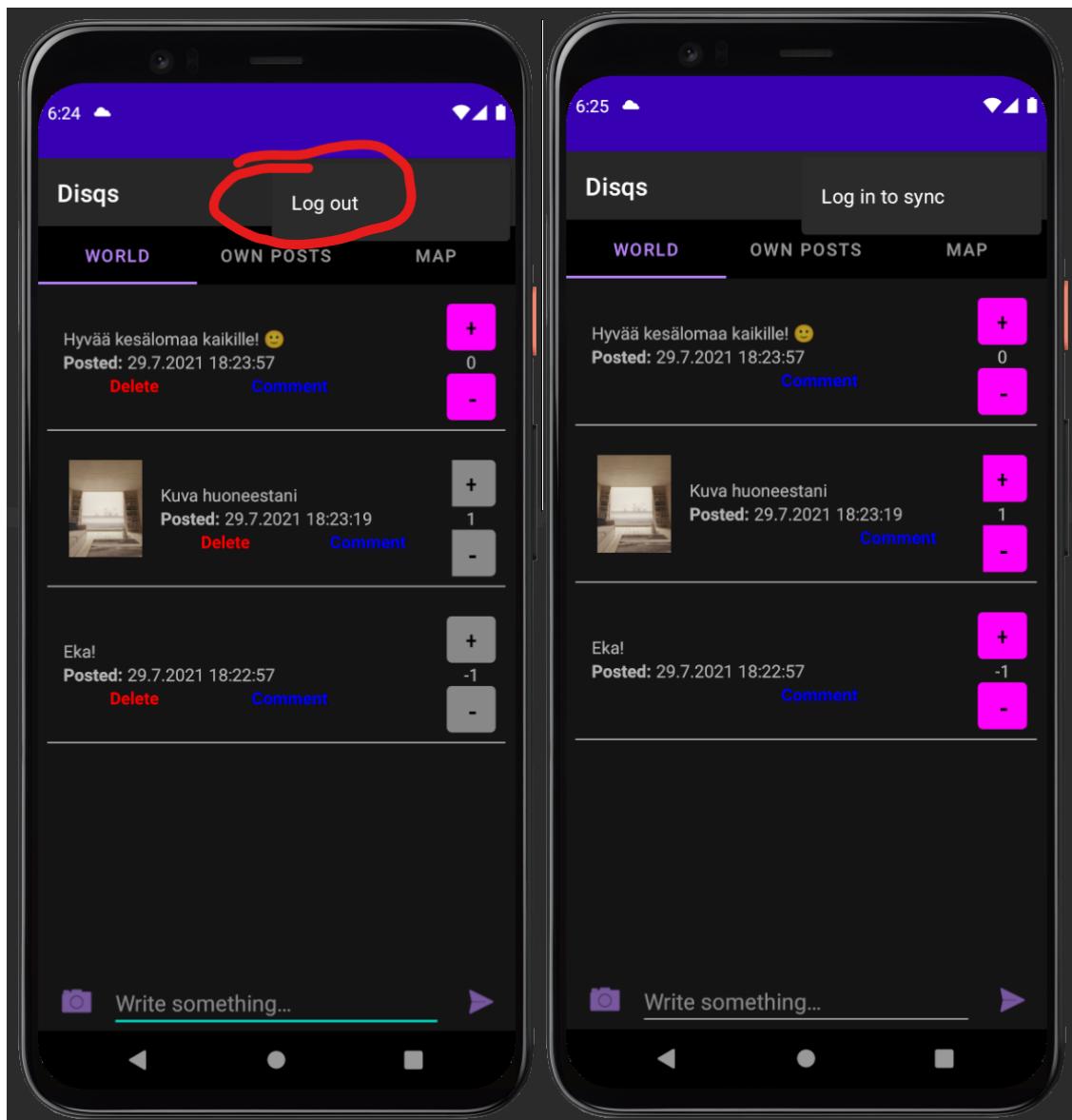




Kun kirjautuminen on onnistuneesti suoritettu, käyttäjä ohjautuu sovelluksen "Maailma"-näkymään, eli ns. päänäkymään, jossa näkyvät kaikkien sovelluksen käyttäjien luomat julkaisut. Lisäksi yläreunassa näkyvät mahdolliset muut näkymät, eli omien julkaisujen näkymä, ja karttanäkymä.



Mikäli kirjautuminen tehtiin ensimmäisellä kerralla anonymisti, on mahdollista linkittää anonymi tili oikeaan käyttäjätiliin oikeassa yläreunassa näkyvän pistevalikon kautta, painamalla "Liitä käyttäjätiliin"-painiketta. Tällöin käyttäjälle pitäisi aueta kirjautumisnäkymä, jossa anonymi kirjautuminen ei ole luonnollisesti mahdollista. Käyttäjä voi valita mieleisensä kirjautumistavan, ja onnistuneen kirjautumisen jälkeen käyttäjälle ilmoitetaan onnistuneesta käyttäjätilin synkronoinnista. Mikäli käyttäjällä ei ollut aikaisempaa käyttäjätiliä kyseisellä kirjautumistavalla, siirtyvät anonymisti tehdyt julkaisut luodun uuden käyttäjän nimiin, ja kirjautumisen jälkeen käyttäjä voi halutessaan niitä esimerkiksi poistaa. Lisäksi, jos vaikka kirjautuu samalla käyttäjällä toisella laitteella, näkyvät myös tästä omat julkaisut myös tällä laitteella, kunnes joko kirjautuu ulos, kirjautuu toisella käyttäjätunnusella tai poistaa/nollaa sovelluksen, ja kirjautuu sen jälkeen anonymisti.



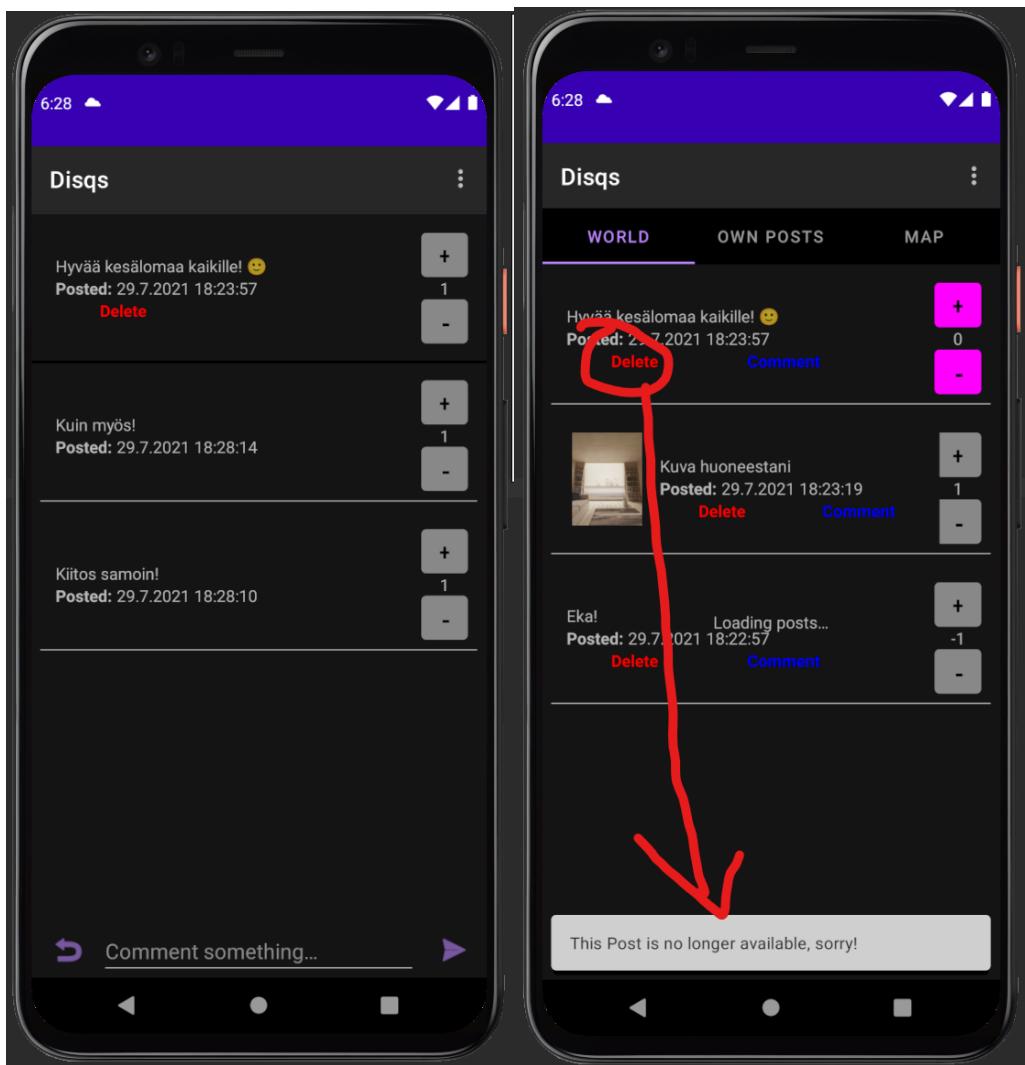
Yläolevassa kuvassa esitetty ensimmäinen kirjautuminen oikealla käyttäjällä, sen jälkeen kirjautuminen anonyminä käyttäjänä. Tästä näkee hyvin, että käyttäjän omistaessa julkaisut niitä pystyy poistamaan, kun taas anonyminä uutena käyttäjänä ei, ennen kuin kirjautuu uudelleen Sync-nappulasta vanhalle käyttäjätilille.

Jokaisen julkaisun kohdalla näkyy äänestyspainikkeet (plusääni ja miinusääni), sekä kommentointipainike. Lisäksi omien tekemien julkaisujen kohdalla näkyy myös punainen Poistopainike, jota painamalla julkaisu poistetaan sovelluksesta, eikä se näy kellekään.

Jokaista julkaisua tai kommenttia voi äänestää vain kerran, ja tästä indikoi painikkeiden väri. Jos painikkeet ovat kirkkaita, vaaleanpunaisia, tarkoittaa, että ääntä kyseiselle julkaisulle/komentille ei vielä ole annettu. Mikäli painikkeet näkyvät harmaana, on käyttäjä antanut jo postaukselle äänensä. Kun jommankumman äänen antaa, painikkeet

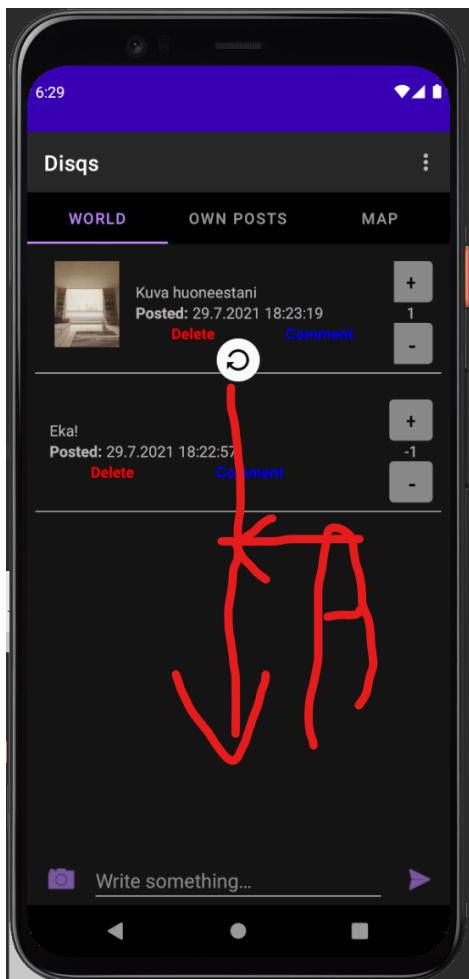
muuttuvat kirkkaista harmaiksi, ja ääni tallennetaan. Äänestyksen tulos näkyy painikkeiden välissä numerona. Numero on siis yhteenlaskettu plusääni (1) ja miinusääni (-1) laskettu summa.

Klikkaamalla julkaisun alapuolella näkyvää kommenttipainiketta, pääsee kyseisen julkaisun kommentinäkymään. Tästä näkymästä käyttäjä voi antaa tälle julkaisulle kommentteja, ja lukea olemassa olevia kommentteja, sekä antaa kommenteille ääniä. Lisäksi omien julkaisujen poisto onnistuu myös tässä näkymässä. Tällöin uusien kommenttien antaminen epäonnistuu, ja käyttäjälle näytetään virheilmoitus.



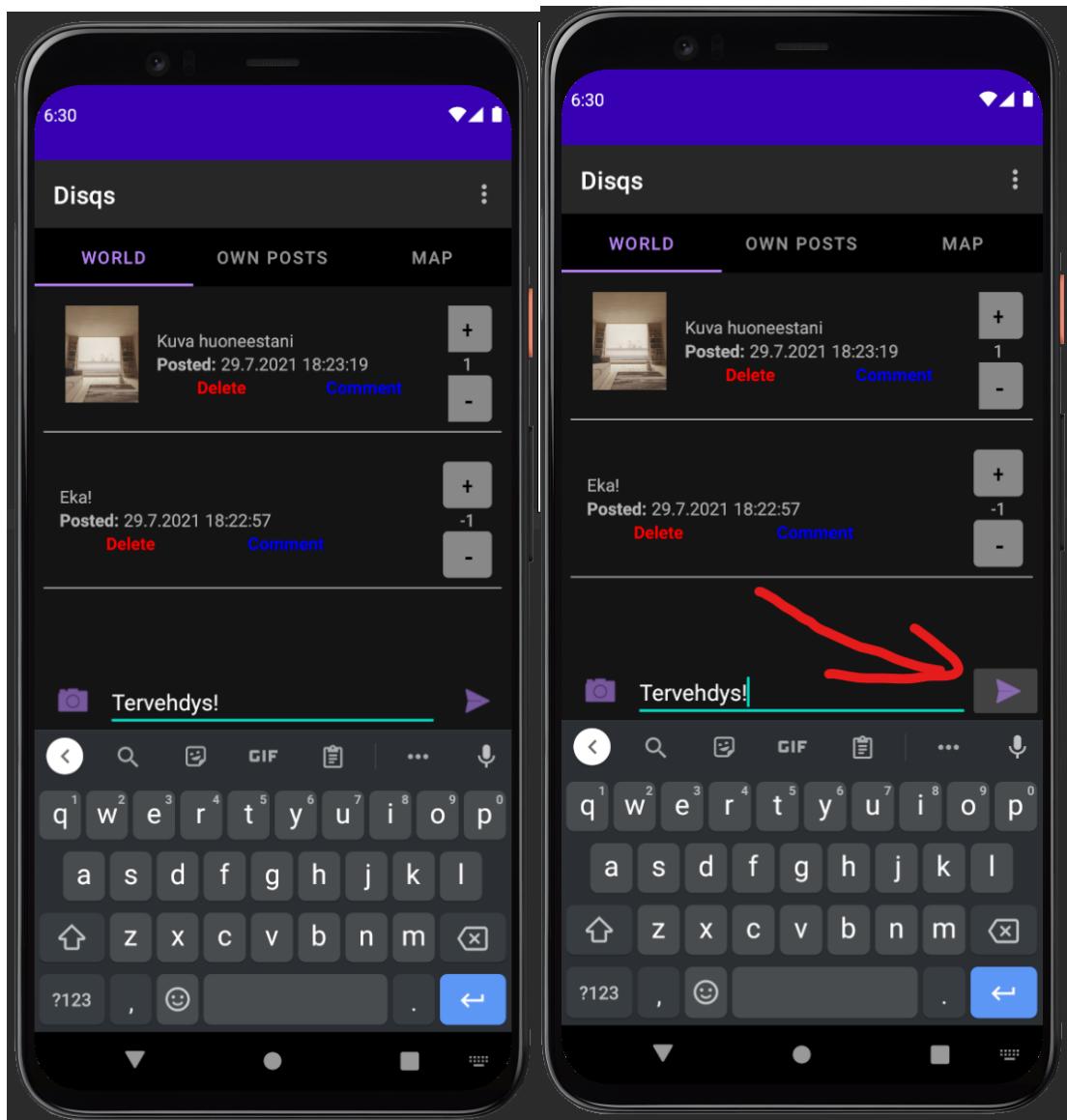
Uudet julkaisut eivät tule näkyviin automaattisesti, vaan käyttäjän on haettava uudet julkaisut itse. Tämä onnistuu helposti vetämällä näkymän yläreunasta kohti ruudun alareuna, jolloin näkyviin ilmestyy päivityssymboli. Tämä ilmaisee sitä, että sovellus hakee uusia päivityksiä listalle tietokannasta, ja näiden lataamisesta tulee näkyviin latausteksti latauksen ajaksi, jotta käyttäjä tietää odottaa uusien postausten latautumista. Päivitys

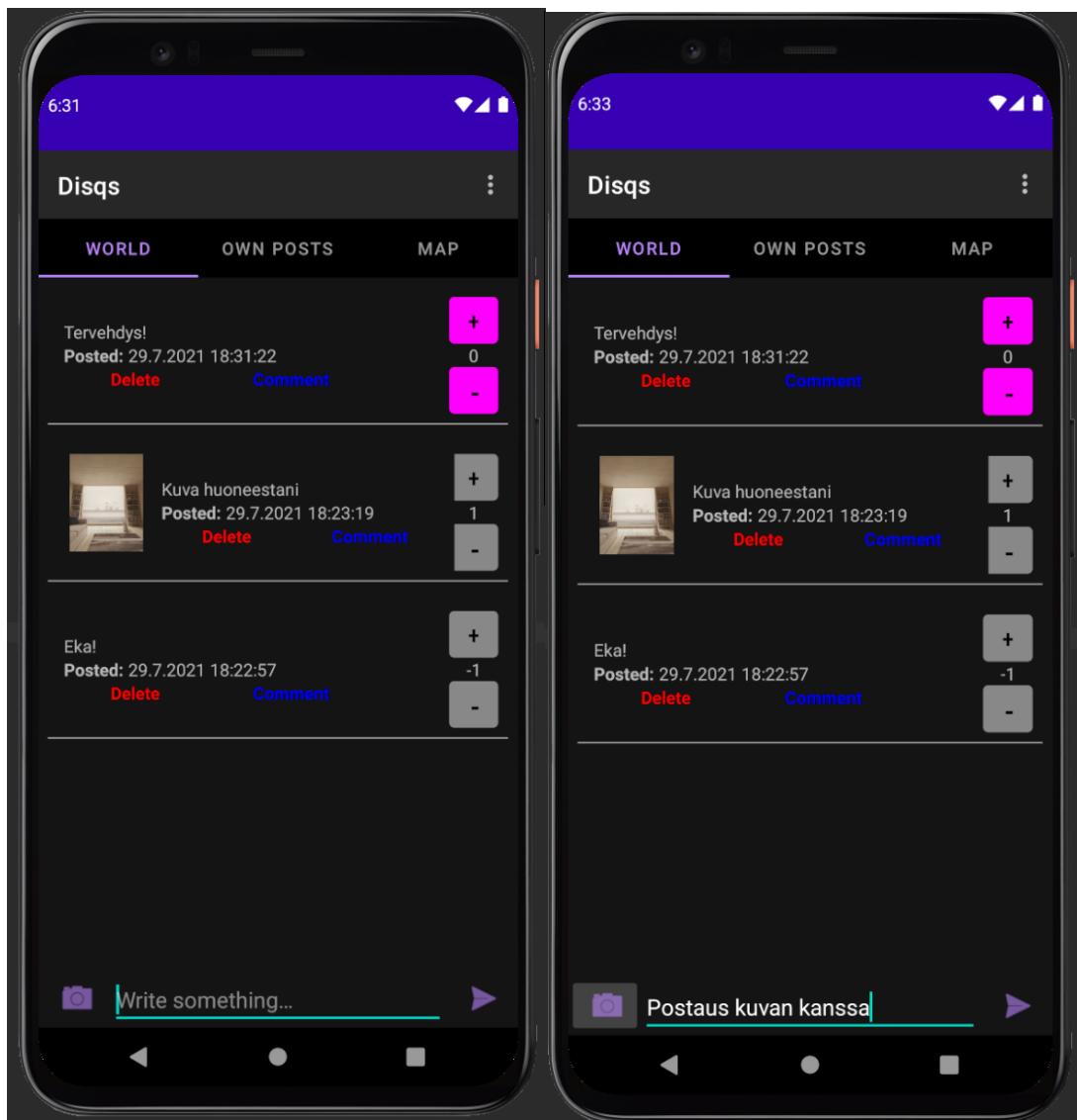
onnistuu niin maailmanäkymässä, kommenttinäkymässä, kuin omien julkaisujen näkymässä. Mikäli päivitys ei jostain syystä onnistu, tulee tästä virheilmoitus käyttäjälle.

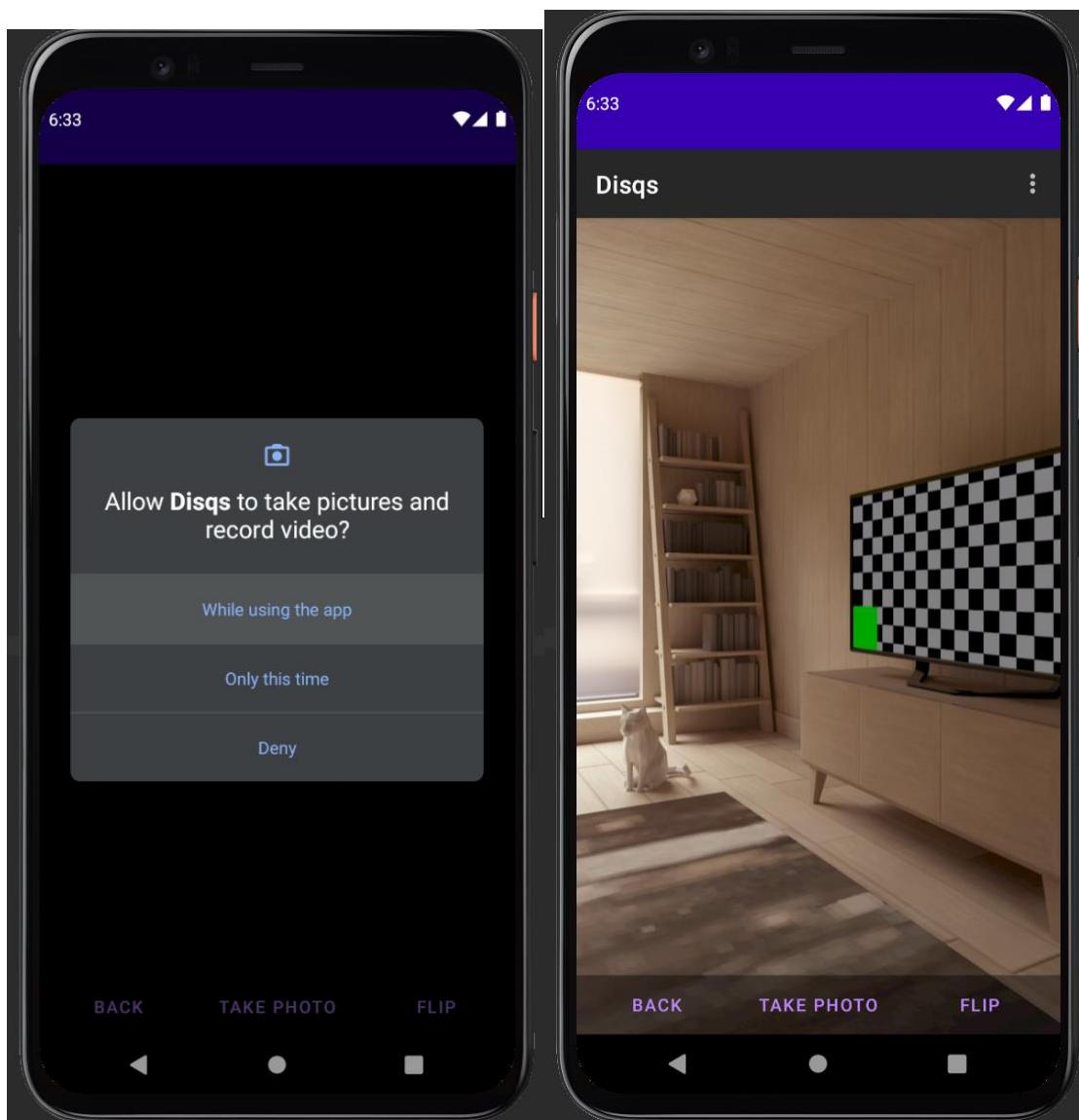


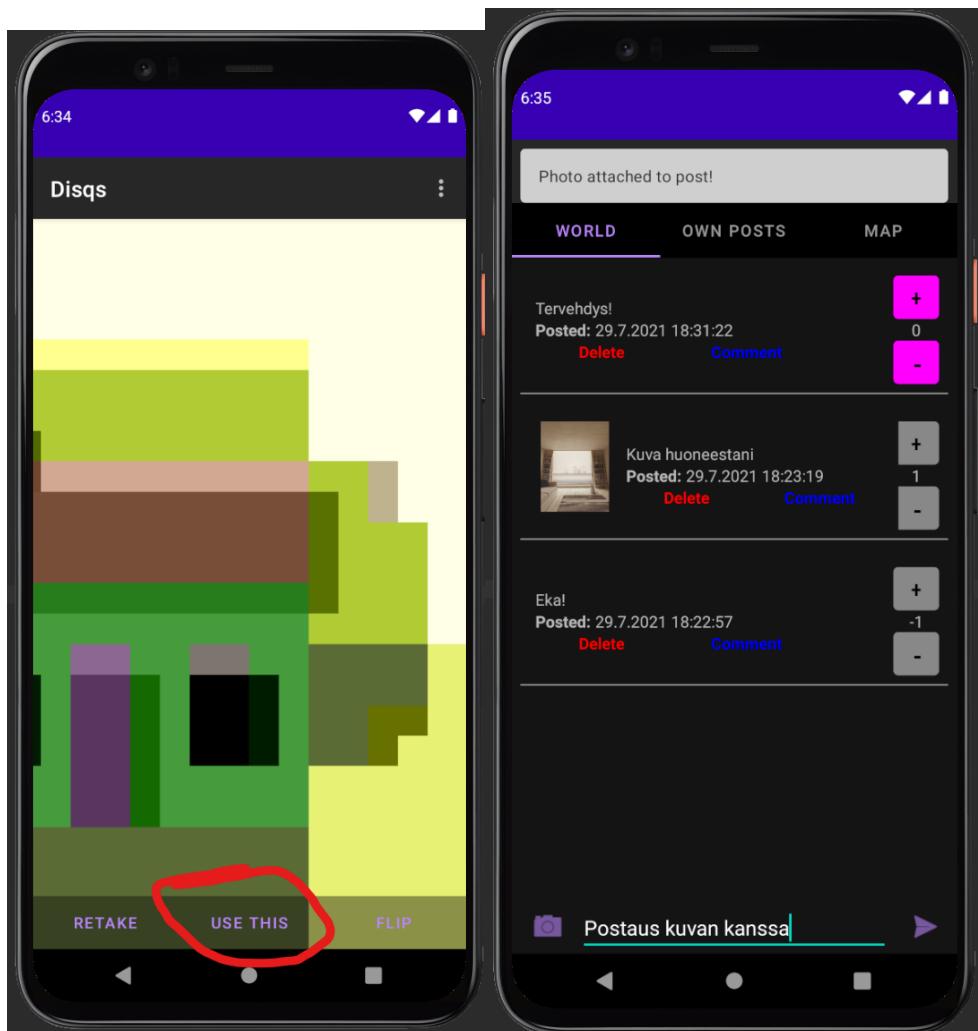
Uuden julkaisun luominen onnistuu Maailmanäkymässä näytön alareunassa olevien elementtien avulla. Mikäli haluaa julkaista vain tekstiä, onnistuu se painamalla tekstikenttää, jolloin näytölle aukeaa näppäimistö. Halutun tekstin jälkeen käyttäjä voi painaa paperilennokki-painiketta, jolloin tästä muodostuu uusi julkaisu. Mikäli käyttäjä haluaa luoda kuvallisen julkaisun, voi hän tekstin kirjoittamisen lisäksi painaa kamerapainiketta, jolloin aukeaa erillinen kameränäkymä. Oletuksena näkymässä näkyy takakameran kuva, mutta sen voi myös halutessaan vaihtaa kuvaamaan etukameraa. Mikäli käyttäjä ei halunnutkaan ottaa kuvaa, voi hän palata takaisin laitteen Back-nappulasta, tai ruudussa olevasta paluunappulasta takaisin Maailmanäkymään. Kun käyttäjä on tyytyväinen kuvakulmaan, tulee painaa keskellä olevaa painiketta, jolloin kuva jähmettyy. Tämän jälkeen käyttäjältä kysytään, onko kuva hyvä, ja halutessaan käyttäjä voi poistaa kyseisen kuvan, ja ottaa uuden. Mikäli kuva on hyvä, painetaan jälleen keskimmäistä painiketta, jolloin palataan takaisin maailmanäkymään, ja käyttäjälle ilmoitetaan onnistuneesta kuvan liittämisestä. Tämän jälkeen käyttäjä voi vielä halutessaan muokata tekstiä (jonka

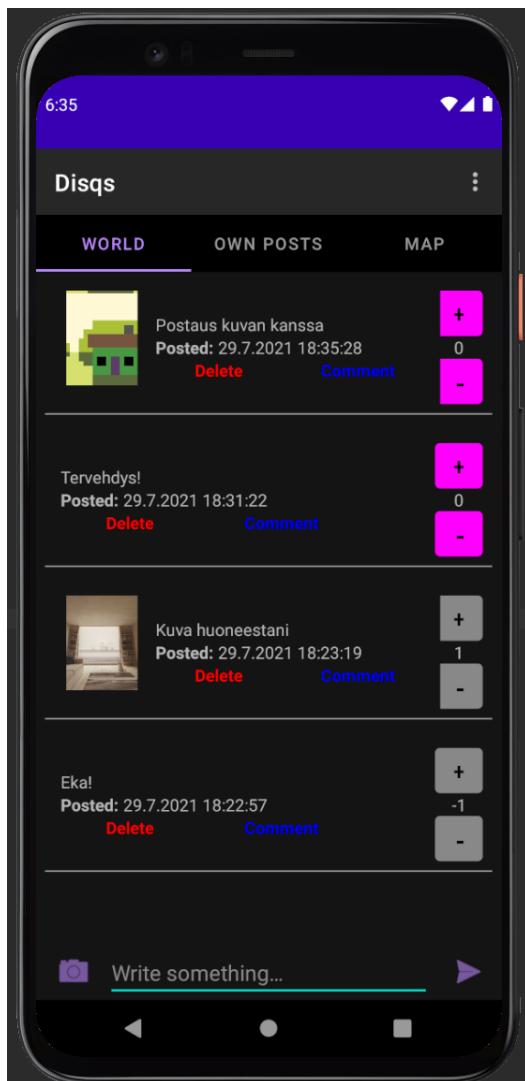
tulee olla syötettynä) ja lähettää julkaisun. Uusi postaus kuvan kera ilmestyy hetken kuluutta listan yläpäähän.



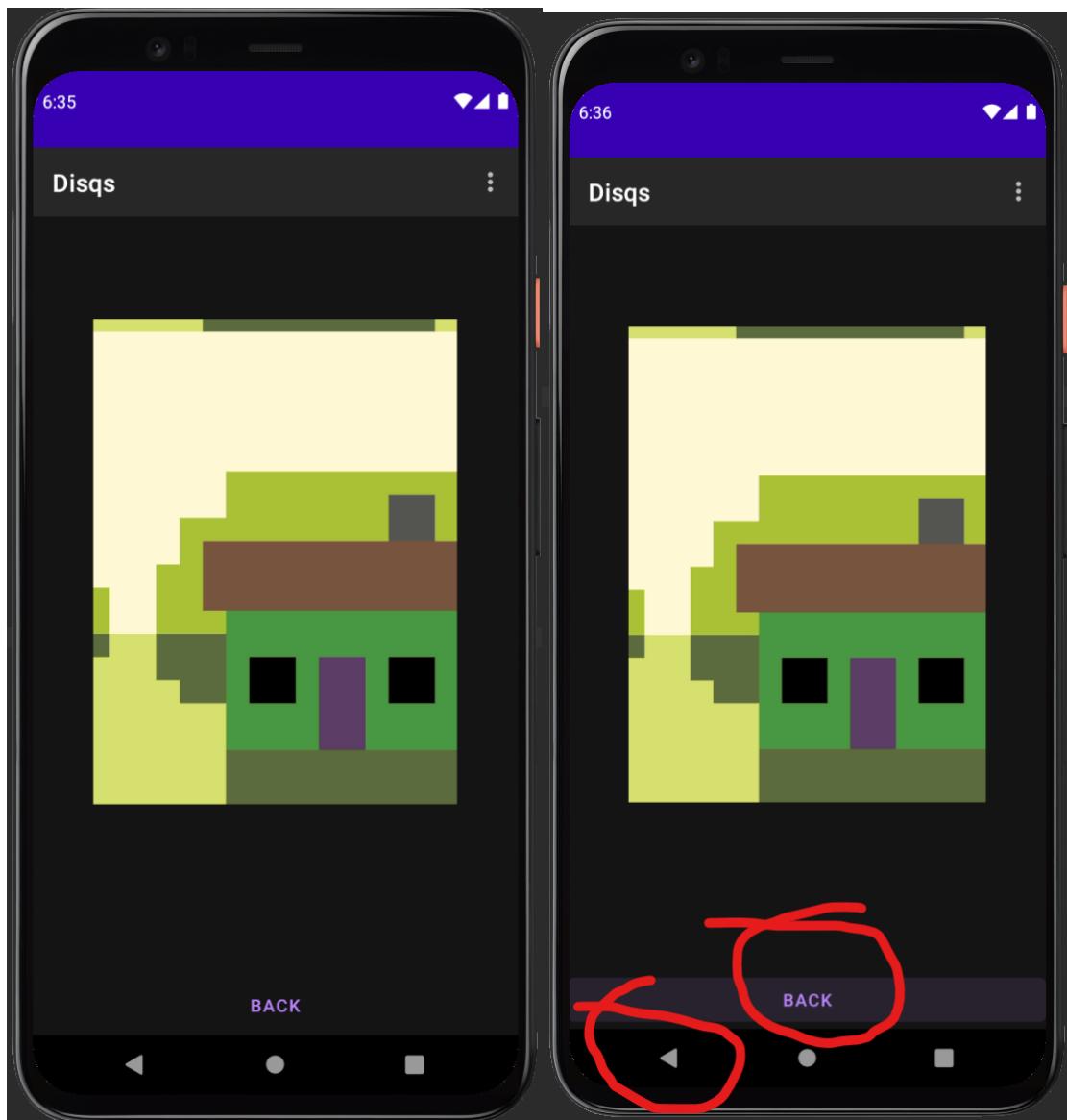








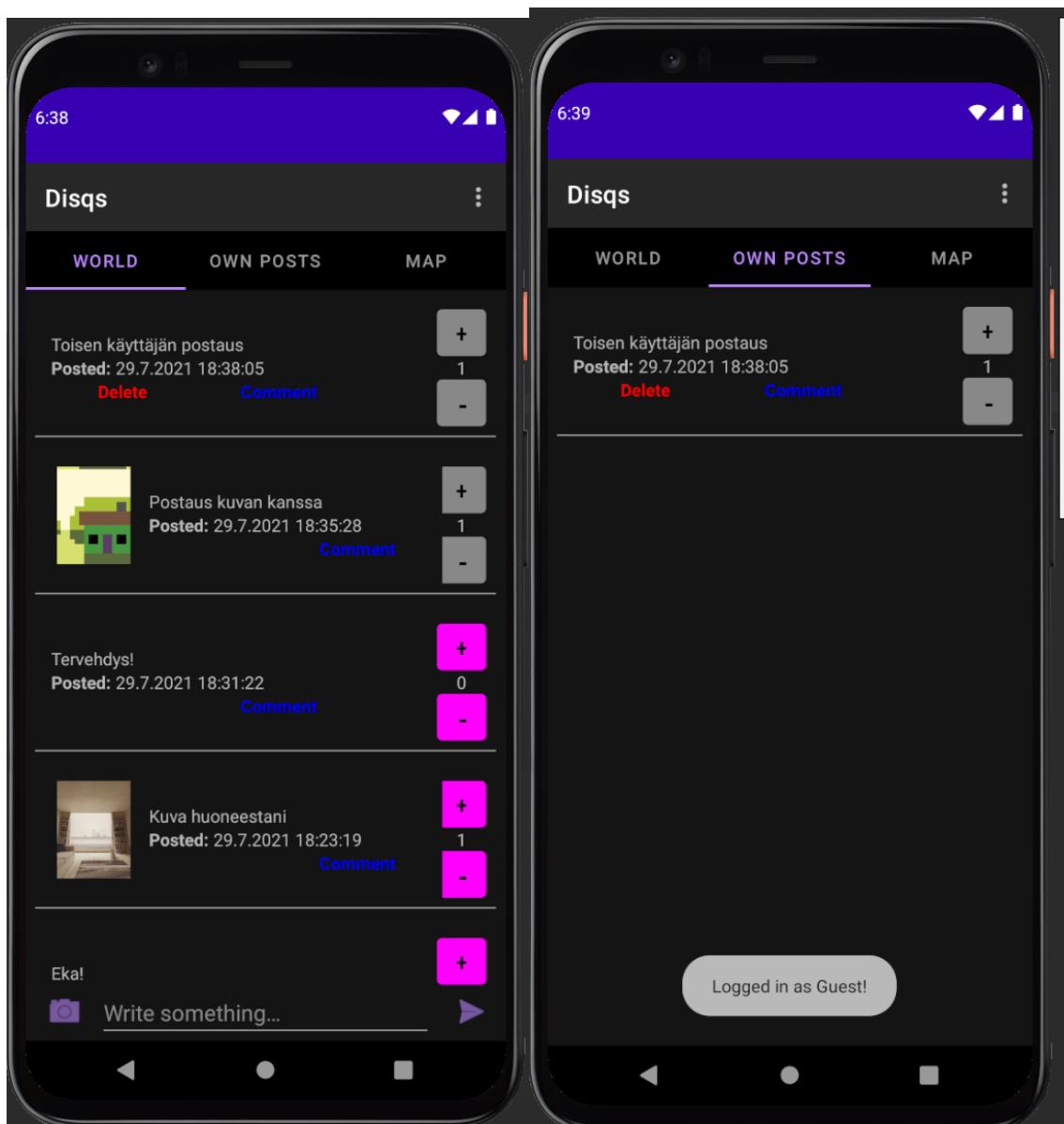
Kuvallisten postausten kuvia voi halutessaan tarkastella tarkemmin painamalla pikkukuvaa, jolloin aukeaa uusi, erillinen näkymä, jossa kuva näkyy hieman isompana. Tästä näkymästä voidaan myös palata takaisin käyttämällä joko laitteen paluunappulaa, tai sitten näytöllä näkyvää Back/Takaisin-painiketta.



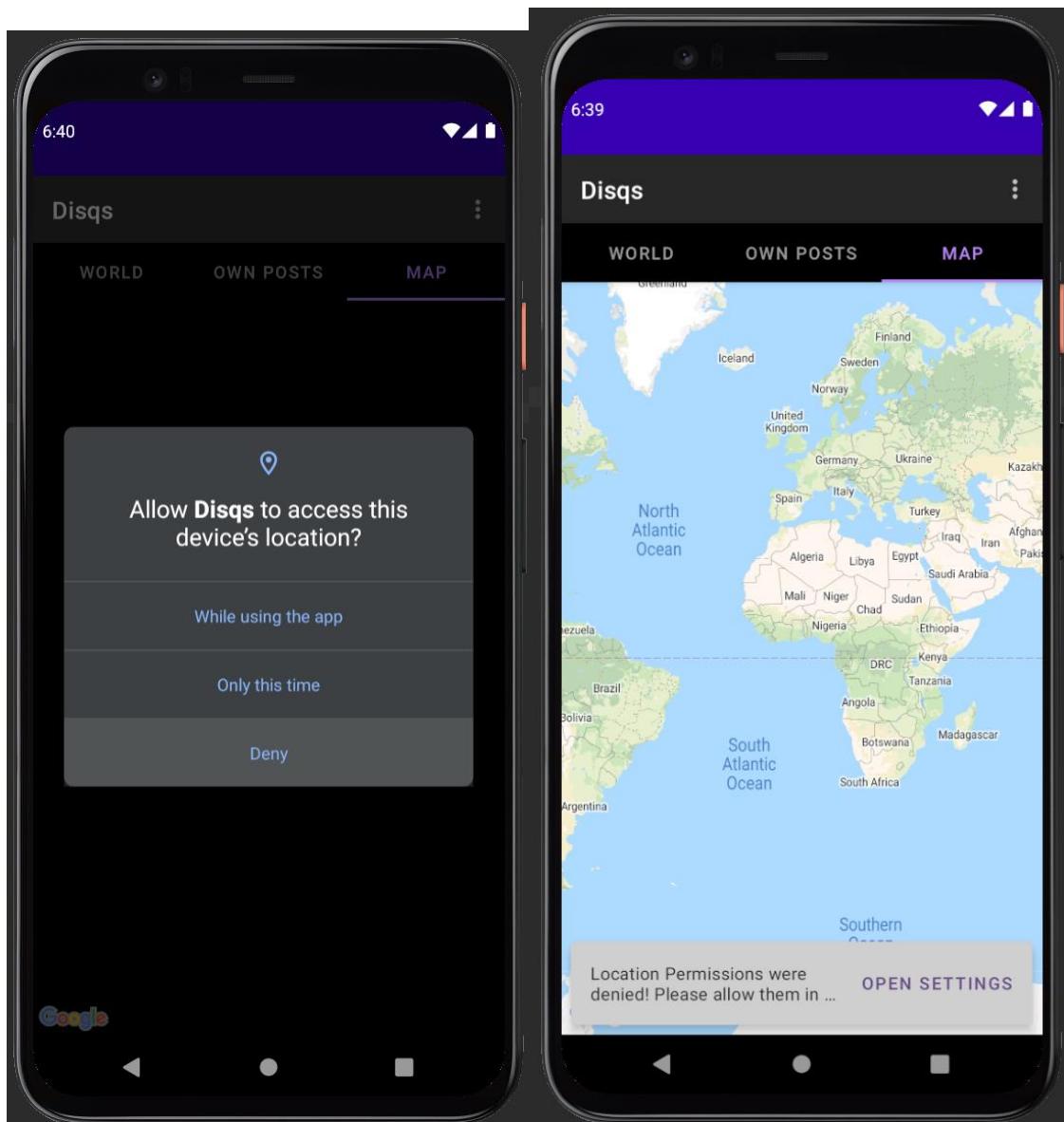
Vastaavasti kommentin luominen kommenttinäkymässä onnistuu samalla tavalla ruudun alareunasta. Kommenttiin ei tosin pysty lisäämään kuva, pelkästään tekstiä.

Yläreunan valikosta käyttäjä voi siirtyä myös muihin sovelluksen näkymiin joko pyyhkäisemällä ruutua vasemmalle tai oikealle, tai painamalla näkymän nimeä valikossa. Pyyhkäisy ei luonnollisesti kuitenkaan toimi Karttanäkymässä.

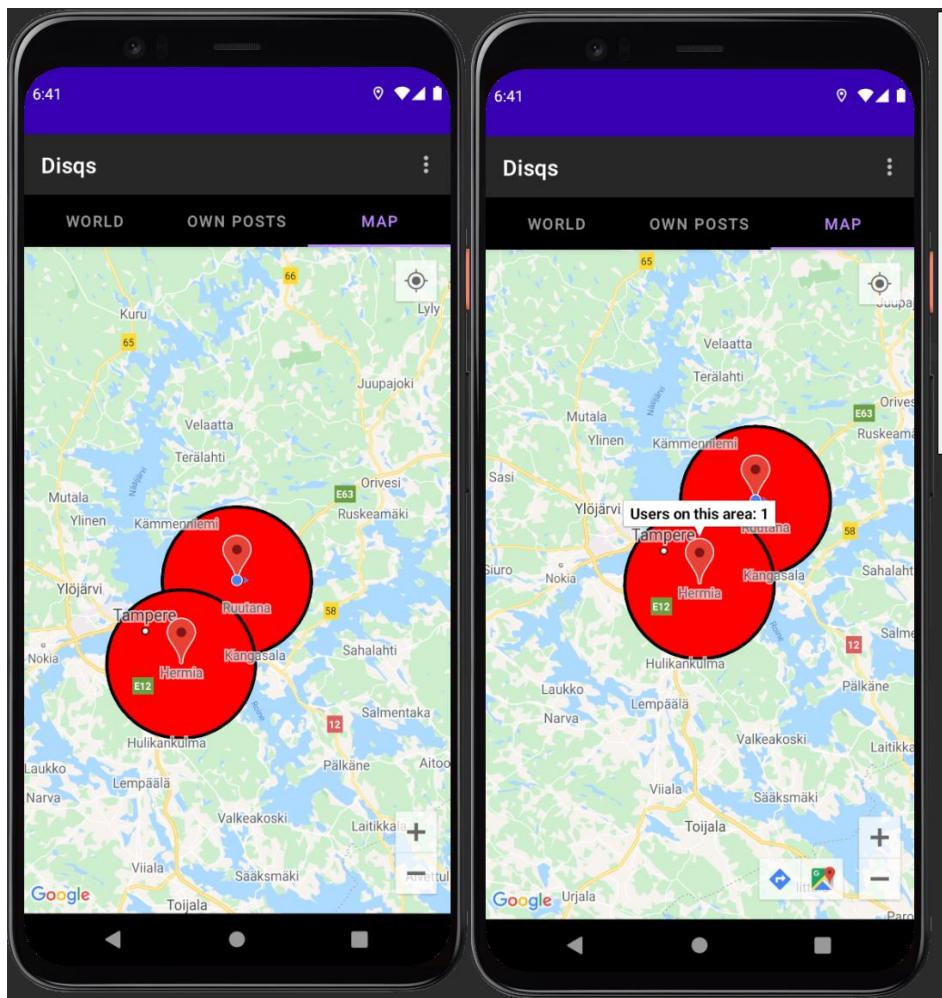
Omien julkaisujen näkymä on muutoin identtinen maailmanäkymään nähdyn, mutta siinä näkyvät ainostaan käyttäjän omat julkaisut. Tämä helpottaa huomattavasti, kun julkaisuja on tullut paljon, ja haluaa tarkastella jotakin paljon vanhempia julkaisujaan helposti ja nopeasti.

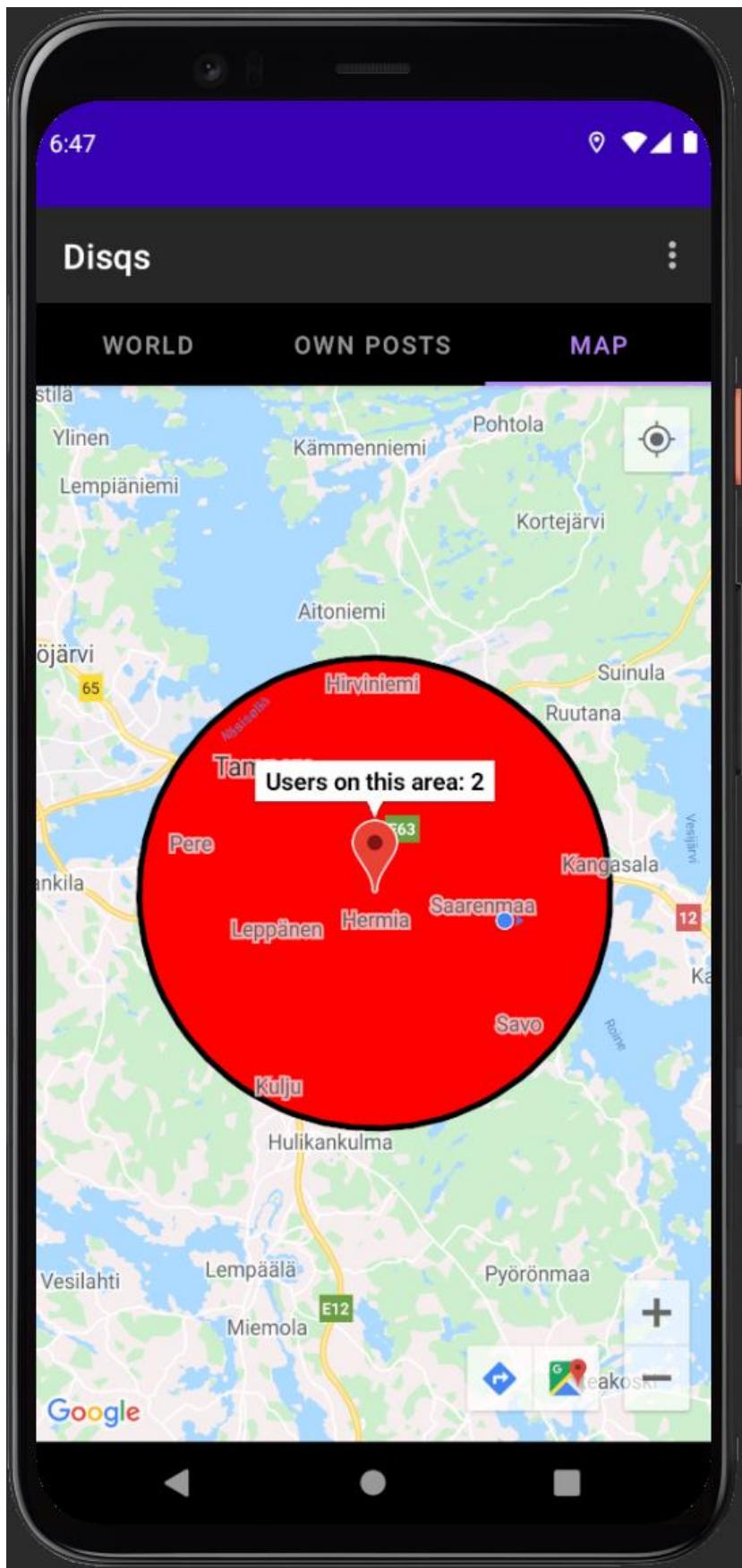


Vielä viimeisenä sovelluksessa on karttanäkymä, jonka avattaessa ensimmäistä kertaa kysytään käyttäjältä lupaa sijainnin käyttöön. Samoin kuten kameranäkymässä, jos käyttäjä ei myönnä sijaintiaan, kartta ei lataudu kokonaan, vaan käyttäjälle näytetään virheilmoitus ja pyydetään sallimaan sijainti sovelluksen asetuksista. Mahdollisesti hylkäämisen jälkeen karttanäkymän uudelleen avaamisen yhteydessä saatetaan lupaa kysyä vielä uudestaan, jos sitä ei vielä aiemmin ollut myönnetty. Käyttäjä voi hyväksyä luvan sijainnin käyttöön joko kertaluontaisesti nykyisellä sovelluksen käyttökerralla, tai pysyvästi, aina kun sovellusta käytetään niin, ettei lupaa tarvitse enää erikseen myöntää.



Kun käyttäjä viimein myöntää luvan sijainnin käyttöön, latautuu karttanäkymään käyttäjän oma sijainti, sekä lisäksi muiden käyttäjien karkeat sijainnit punaisina ympyröinä. Ensimmäisellä avauskerralla karttanäkymä fokusituu käyttäjän sijaintiin, mutta tämän jälkeen käyttäjä voi vapaasti liikutella karttaa, ja tutkia mm. muiden käyttäjien sijanteja ja lukumäärää kartalla.





LÄHTEET

- [1] Develop Android apps with Kotlin, <https://developer.android.com/kotlin>
- [2] What's the latest version of Android, <https://www.how-to geek.com/345250/whats-the-latest-version-of-android/>
- [3] GsmArena – OnePlus 8 Pro Full Phone Specifications, https://www.gsmarena.com/oneplus_8_pro-9919.php
- [4] Android Studio, <https://developer.android.com/studio>
- [5] OxygenOS, <https://fi.wikipedia.org/wiki/OxygenOS>
- [6] Android Kotlin-first approach, <https://developer.android.com/kotlin/first>
- [7] TutorialsPoint – Kotlin, <https://www.tutorialspoint.com/kotlin/index.htm>
- [8] AndroidAuthority, Kotlin tutorial for Android for beginners, <https://www.androidauthority.com/kotlin-tutorial-1134289/>
- [9] Android Developers, <https://developer.android.com/>
- [10] Build your first app, <https://developer.android.com/training/basics/firstapp/index.html>
- [11] Guide to app architecture, <https://developer.android.com/jetpack/guide>
- [12] Stack Overflow, <https://stackoverflow.com/>
- [13] Android Room with a View – Kotlin codelab, <https://developer.android.com/codelabs/android-room-with-a-view-kotlin>
- [14] Room Package Summary, <https://developer.android.com/reference/androidx/room/package-summary>
- [15] Kotlin Lang Docs, <https://kotlinlang.org/docs/home.html>
- [16] Harjoitusten 6,7,8 applikaatio Play Storella, <https://play.google.com/store/apps/details?id=fi.tuni.sinipelto.laskuvelho>
- [17] Google Firebase, <https://firebase.google.com/>
- [18] Firebase Firestore Using View Models and LiveData (Kotlin, Android), <https://medium.com/@deepak140596/firebase-firebase-using-view-models-and-livedata-f9a012233917>
- [19] Basic Example of LiveData and ViewModel, <https://medium.com/@taman.neupane/basic-example-of-livedata-and-viewmodel-14d5af922d0>
- [20] Firebase Firestore: How to convert document object to a POJO on Android, <https://www.javaer101.com/en/article/428580.html>

- [21] Cloud Firestore Android Codelab, <https://firebase.google.com/codelabs/firestore-android>
- [22] Adding data to Cloud Firestore using Kotlin, <https://medium.com/@winision/adding-data-to-cloud-firebase-using-kotlin-d337d841c643>
- [23] Building a Joke App with Cloud Firestore using Kotlin part 2, <https://wise4rmgo-dadmob.medium.com/building-a-joke-app-with-cloud-firebase-using-kotlin-part-2-dbcbb5b4c0eb>
- [24] Medium, <https://medium.com/>
- [25] Cloud Firestore Docs, <https://firebase.google.com/docs/firestore>
- [26] Sensors Overview, https://developer.android.com/guide/topics/sensors/sensors_overview.html
- [27] SensorManager registerListener method, [https://developer.android.com/reference/android/hardware/SensorManager#registerListener\(android.hardware.SensorEventListener,%20android.hardware.Sensor,%20int\)](https://developer.android.com/reference/android/hardware/SensorManager#registerListener(android.hardware.SensorEventListener,%20android.hardware.Sensor,%20int))
- [28] Android Developers, Take Photos, <https://developer.android.com/training/camera/photobasics>
- [29] Android Developers, Control the Camera, <https://developer.android.com/training/camera/cameradirect>
- [30] Android Codeblab, Getting started with CameraX, <https://codelabs.developers.google.com/codelabs/camerax-getting-started>
- [31] Android Developers, CameraX, <https://developer.android.com/training/camerax>
- [32] Android Alert Dialog, <https://developer.android.com/reference/android/app/AlertDialog>
- [33] Android Codelab: Getting sensor data, <https://developer.android.com/codelabs/advanced-android-training-sensor-data>
- [34] Android Step Counter, https://developer.android.com/reference/android/hardware/Sensor#TYPE_STEP_COUNTER
- [35] Android Step Detector, https://developer.android.com/reference/android/hardware/Sensor#TYPE_STEP_DETECTOR
- [36] Receive location updates in Android with Kotlin, <https://codelabs.developers.google.com/codelabs/while-in-use-location>
- [37] Dexter runtime permission library, <https://github.com/Karumi/Dexter>
- [38] Android Location Manager, <https://developer.android.com/reference/android/location/LocationManager.html>
- [39] Google Maps SDK for Android, <https://developers.google.com/maps/documentation/android-sdk/overview>

- [40] Adding a Map with Marker, <https://developers.google.com/maps/documentation/android-sdk/map-with-marker>
- [41] Select Current Place and Show details on Map, <https://developers.google.com/maps/documentation/android-sdk/current-place-tutorial>
- [42] Android MapView Tutorial, <https://www.zoftino.com/android-mapview-tutorial>
- [43] Fixer.io, <https://fixer.io/>
- [44] NetworkOnMainThreadException, <https://developer.android.com/reference/android/os/NetworkOnMainThreadException>
- [45] StackOverflow: Android 8: Cleartext HTTP traffic not permitted, <https://stackoverflow.com/questions/45940861/android-8-cleartext-http-traffic-not-permitted>
- [46] Fixer.io product plans, <https://fixer.io/product>
- [47] Retrofit, A type-safe HTTP client for Android and Java, <https://square.github.io/retrofit/>
- [48] Github: Google Gson, <https://github.com/google/gson>
- [49] Android Developers, TimerTask, <https://developer.android.com/reference/java/util/TimerTask>
- [50] Android Developers, Timer, <https://developer.android.com/reference/java/util/Timer>
- [51] Ilmatieteen laitoksen avoin data -rajapinta, <http://opendata.fmi.fi/>
- [52] Ilmatieteen laitos, latauspalvelun pikaohje, <https://www.ilmatieteenlaitos.fi/latauspalvelun-pikaohje>
- [53] Parameter does not have a match; SimpleXML, <https://stackoverflow.com/questions/45741617/parameter-does-not-have-a-match-simplexml>
- [54] Github, Fmidev / fmi-avi-xmlmodel, <https://github.com/fmidev/fmi-avi-xmlmodel>
- [55] Generate your JAXB classes in second, <https://thorben-janssen.com/generate-your-jaxb-classes-in-second/>
- [56] Android XML, <https://developer.android.com/training/basics/network-ops/xml#kotlin>
- [57] Simple XML Serialization, SourceForge, <http://simple.sourceforge.net/>
- [58] Create swipe views with tabs using ViewPager, Android Developers, <https://developer.android.com/guide/navigation/navigation-swipe-view>
- [59] Adding Swipe-to-Refresh To Your App, <https://developer.android.com/training/swipe/add-swipe-interface>
- [60] Universally Unique Identifier, Wikipedia, https://en.wikipedia.org/wiki/Universally_unique_identifier

- [61] Class UUID, Oracle Docs, <https://docs.oracle.com/javase/7/docs/api/java/util/UUID.html>
- [62] Android Permissions, Foreground Location, <https://developer.android.com/training/location/permissions#foreground>
- [63] StackOverflow, Using SetUserVisibleHint Method, <https://stackoverflow.com/questions/50424264/using-setuservisiblehint-method/50424392>
- [64] StackOverflow, How to determine when Fragment becomes visible in ViewPager, <https://stackoverflow.com/questions/10024739/how-to-determine-when-fragment-becomes-visible-in-viewpager>