Uudelleenvalokuvaussovelluksen kehittäminen Flutterilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus

kevät 2022

Henry Tiainen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä
Tekijä Henry Tiainen Vuosi 2022
Työn nimi Uudelleenvalokuvaussovelluksen kehittäminen Flutterilla
Ohjaaja Lasse Seppänen

Opinnäytetyöni tarkoituksena oli selvittää mobiilisovelluksen kehitysprosessi Flutter-ympäristössä dart-kielellä. Opinnäytetyön aihe on uudelleenvalokuvaussovelluksen ohjelmoiminen ilmaista karttapalvelua käyttäen. Mobiilisovelluksessa kehitetty koodi on avointa lähdekoodia. Sovelluksen on tarkoitus saada ihmiset kiinnostumaan heitä ympäröivistä historiallisista lokaatioista ja motivoida liikkumaan enemmän. Opinnäytetyöni toimeksiantaja oli Wikimedia Finland ry ja se on osa Helsinki Rephotography -hanketta.

Opinnäytetyön teoria koostuu pääosin Flutterin kehittäjien omista dokumentaatioista ja ratkaisuista erinäisiltä Flutter-aiheisilta foorumeilta. Tutkimusaineisto on kerätty Flutterin virallisesta dokumentaatiosta, joka käsittelee opinnäytetyöni aihetta sekä omasta tuotetusta valmiista sovelluskehitys työstä. Aineisto on analysoitu omalla käytännön osaamisella ja vertaamalla aineistoa muiden Flutter-kehittäjien keskustelupalstoilta saatuihin ratkaisuihin. Opinnäytetyöni tutkii miten Flutterilla voidaan suunnitella ja toteuttaa nykyaikaista käyttöliittymäsuunnittelua, miten Flutterilla pystyy kansainvälistämään ja kotoistamaan sovelluksen ja onko Flutter hyvä kehitysympäristö opettaa mobiiliohjelmointia. Tyypiltään opinnäytetyö on toiminnallinen. Työn toiminnallisen osan olen toteuttanut Android Studiolla hyödyntäen Sony Xperia 10 II -älypuhelinta.

Tutkimuksessa havaittiin, että Flutterilla voidaan saada aikaan ilman lisäkirjastojakin hyvä ja toimiva testisovellus Helsinki Rephotography -hankkeen valokuvakävelyitä varten. Nykyisen kehittämistyön perusteella voin suositella lisäkehitystä liittyen käyttöliittymään. Wikimedia Suomi ry oli tyytyväinen työn tuloksiin teorian ja soveltavuuden osilta. Ajapaik Flutter App on paraikaa testikäytössä ja lisäkehityksessä.

Avainsanat Flutter, dart, ohjelmistokehitys, mobiiliohjelmointi, käyttöliittymäsuunnittelu

Sivut 47 sivua ja 4 sivua liitteitä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology Abstract
Author Henry Tiainen Year 2022
Subject The Development of a rephotography application using Flutter
Supervisors Lasse Seppänen

The purpose of my thesis was to get an understanding on the process of developing a mobile application in Flutter using dart as the programming language. The subject of the thesis is the development of a rephotography application using a free map service-plugin. The code used in the development is open source by its' nature. The application's aim is to spread knowledge to people of the historic locations close to them and get people to walk more in their environment. The commissioner of the thesis was Wikimedia Suomi ry and it's a part of their Helsinki Rephotography-project.

The theory of the thesis is mainly based on the documentation of those who developed Flutter and from solutions on Flutter based forums. The research material was collected from Flutter's official documentation and from my own work developing the application. The data has been analyzed using my practical skills as well as cross-referring the material to known solutions from other Flutter developers on internet forums. My thesis examines how to design and implement modern user interface using Flutter, how to internationalize and localize the mobile application and how Flutter would work as a learning platform towards mobile programming. The thesis' type is functional. The development of the application has been done using Android Studio in tandem with Sony Xperia 10 II-smartphone.

The research demonstrates that even without the extensive use of plugin packages a rephotography application could be made using Flutter for the Helsinki Rephotography-project. Based on the current state of the application, I recommend further development of the user interface. Wikimedia Suomi ry was pleased with the work. Currently Ajapaik Flutter App is being tested in real environment and is being developed using my work as a basis.

Keywords Flutter, dart, software development, mobile programming, UI design

Pages 47 pages and 4 pages of appendices

Sanasto

Flutter	Googlen kehittämä, avoimen lähdekoodin omaava kehittämisympäristö
Dart	Ohjelmointikieli, jolla voidaan kehittää Flutter-sovellusta
Avoin lähdekoodi	Tietokoneohjelmien maksutonta tuottamis- ja kehittämismenetelmiä tukeva lähdekoodi, jota voi muokata ja jakaa tarpeittensa mukaisesti
Android Studio	Googlen ilmainen kehitysympäristö, jolla voi kehittää eri ohjelmointikielillä erilaisia Android- iOS- ja web-sovelluksia
Android	Googlen kehittämä mobiilikäyttöjärjestelmä
iOS	Applen kehittämä mobiilikäyttöjärjestelmä
JSON	Yksinkertaistettu ja standardoitu tiedostomuoto tietojen välitykseen
Muuttuja	Sovelluksen koodissa käytettävä nimellinen yksikkö, joka voi pitää sisällään kirjaimia, numeroita tai käyttäjän kirjoittamaa tekstiä
Funktio	Sovelluksen koodissa toimiva ohjelma, joka voidaan ajaa useampaankin kertaan haluttaessa
Paketti	Flutterissa hyödynnettävä paketti, jota voidaan käyttää valmiina kirjastona koodissa
SDK	Kirjasto ohjelmistokehitykseen käytettävistä työkaluista, joihin voi kuulua debuggeri, kääntäjä ja/tai ohjelmistokehys

Kansainvälistäminen	Sovelluksen kansainvälistäminen käyttäen i18n standardeja, jolla voidaan mahdollistaa sovelluksen kotoistaminen
Kotoistaminen	Sovelluksen lokalisointi tietylle kohdeyleisölle käyttäen l10n standardeja
Saavutettavuus	Sovelluksen käytön mahdollistaminen mahdollisimman monelle käyttäjälle
Metadata	Yksinkertaistettuna metadata tarkoittaa tietoa tiedosta. Metadata tai metatieto voi kertoa käyttäjälle esimerkiksi tiedon luojasta tai milloin ja miten tieto on koottu

Sisällys

Liitteet

1 Johdanto

Ihmisiä on kautta aikojen kiinnostanut ympäristönsä dokumentointi kirjoitusten, maalausten ja myöhemmin kuvien avulla. Valokuvausta on käytetty Suomessa dokumentoinnissa varhaisintaan 1857. Ensimmäinen dokumentoitu kuva Suomessa on otettu Helsingin Esplanadin puistossa vanhasta teatteritalosta. Vanhat kuvat historiallisista kohteista ovat yleisesti arkistoitu kirjastoihin tai ovat Museoviraston hallussa. Nykyään vanhoja kuvia skannataan elektroniseksi, jotta menneisyys saataisiin arkistoitua nykyajan standardien mukaisesti. Wikimedia Suomi Ry:n Helsinki Rephotography -hankkeessa tutkitaan vanhoja kuvia Helsingissä valokuvakävelyillä ja tämän takia uudelleenvalokuvaussovellus voisi olla tärkeä osa valokuvakävelyiden tulevaisuutta.

Opinnäytetyöni tarkoitus on näyttää uudelleenvalokuvaussovelluksen kehitysvaiheet kehittämällä sitä Flutterilla. Flutter on Googlen järjestelmäriippumaton mobiiliohjelmointialusta, jolla ohjelmoidaan dart-kielellä. Lähtökohtaisesti sovelluksessa oli valmiit raamit Ajapaikista tuotuihin albumeihin. Ajapaik on virolainen avoimeen lähdekoodiin perustuva yritys, joka on Helsinki Rephotography -hankkeen yhteistyökumppani. Albumien lisäksi sovellukseen oli myös integroitu laitteen kameran avaaminen vanhan kuvan uudelleenvalokuvaamista varten. Työni teoria pitää sisällään Flutterin ja dartin esittelemisen, opinnäytetyön lähtökohdat ja soveltavassa osiossa käytettävien ratkaisujen teorian. Opinnäytetyön soveltavassa osiossa käydään läpi Flutterin käyttöönotto, Android Studion liittäminen GitHubiin ja lisälaitteiden käyttöönotto sovelluksen emuloimista varten. Soveltavaan osioon kuuluu myös käyttöliittymäsuunnittelun ja saavutettavuuksien implementoimisen sovellukseen sekä karttapalvelujen käyttöönoton ja hyödyntämisen kuvien kanssa sovelluksessa. Wikimedia Finland ry toimii opinnäytetyöni tilaajana ja työni tavoitteena on lisäksi myös vastata heidän kysymyksiinsä Flutterilla kehittämisestä.

- Millainen on nykypäiväinen mobiilisovelluksen käyttöliittymä?
- Miten saan kansainvälistettyä ja kotoistettua mobiilisovelluksen Flutterilla?
- Pystyykö Flutterilla opettamaan mobiiliohjelmointia paremmin kuin muilla alustoilla?

2 Dart

Tässä luvussa käydään läpi opinnäytetyössäni käytettävän dart-ohjelmointikielen historiaa ja teorian tasolla olevaa syntaksia. Opinnäytetyön teoreettista syntaksia toteutetaan käytännössä opinnäytetyön soveltavassa osiossa.

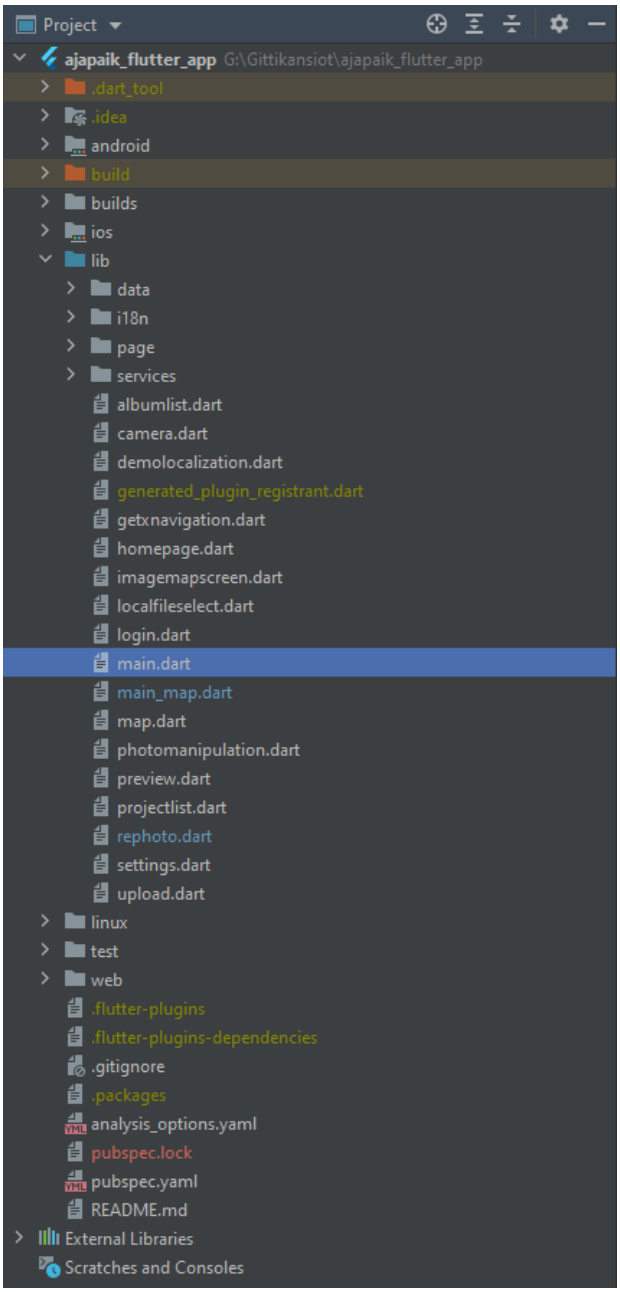
2.1 Historiaa dartista

Dart on Googlen kehittämä avoimeen lähdekoodiin perustuva ohjelmointikieli, jolla voidaan ohjelmoida Flutter sovelluksia. Google julkaisi dartin (ent. Dash) ohjelmointikielenä vuonna 2011. Dart kehitettiin alun perin korvaamaan JavaScript monimutkaisissa web-sovelluksissa ja sen myötä myös web-ohjelmoinnissa web-alustoilla. (Claburn, 2011) Dart käytti alkuvaiheessa omaa virtuaalikonettaan, mutta ideasta kuitenkin luovuttiin vuonna 2015 ja kehityksen suunnaksi otettiin ohjelmien kääntäminen JavaScriptiksi. Samana vuonna Dart Developer Summitissa julkaistiin Sky (nyk. Flutter). Nykyään dart on pääohjelmointikieli Googlen vuonna 2017 julkaisemassa järjestelmäriippumattomassa ohjelmointikirjasto Flutterissa.

2.2 Dartin syntaksin piirteet

Dart on yksinkertainen ja helposti opittava C-syntaksin omaava ohjelmointikieli. Se on pääasiallisesti käyttöliittymien kehittämiseen tarkoitettu ohjelmointikieli, jolla voidaan ohjelmoida mobiili- ja web-sovelluksia. Valmiissa Flutter-sovelluksessa dart kääntyy myös JavaScriptiksi dart2js:n kautta. (Dart, n.d.) Dartille ominaista on myös luokat, jotka ovat hyvin samanlaisia kuin Javassa käytettävät luokat. Dartiin kuuluu normaalisti nimetyt luokat, joilla yleensä luodaan backend-koodia. Käyttöliittymäkoodia kehittäessä Flutter-projektissa tulee huomioida kaksi erityistä luokkaa, joista ensimmäinen on StatelessWidget-luokka ja toinen StatefulWidget-luokka. Perusperiaate dartissa on, että luokkien sisälle ohjelmoidaan muuttujia ja funktioita ja niitä hyödyntäen voidaan tehdä olioita (engl. widgets), joilla rakennetaan toimiva kokonaisuus sovellukseksi. Kuvassa 1 näkyy Ajapaik Flutter App -projektipuu, jossa kaikki ohjelmakoodi kirjoitetaan dart-tiedostoihin.

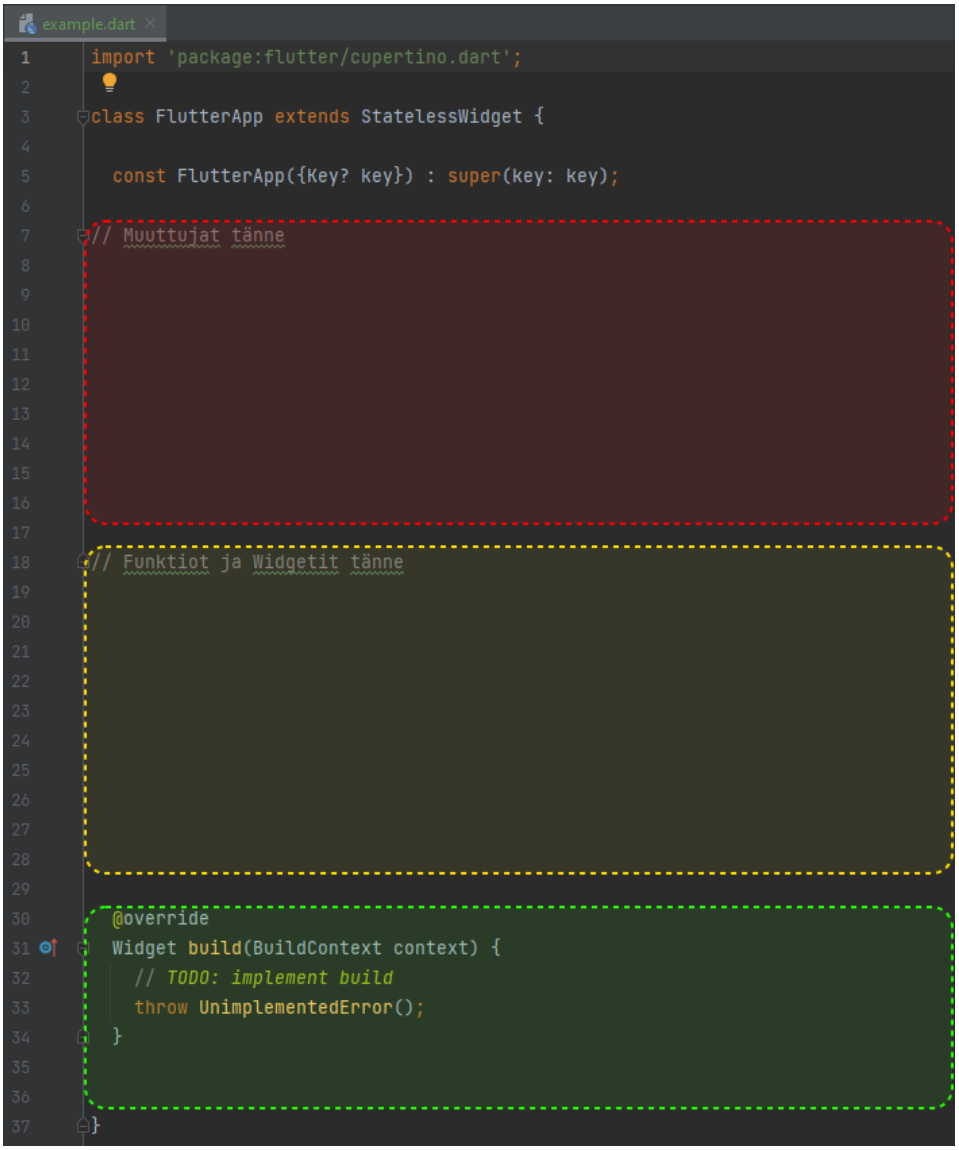
Kuva 1 – Ajapaik Flutter Appin projektipuu.



2.2.1 StatelessWidget-luokka

StatelessWidget-luokka on ns. tilaton luokka, joka ei säilytä ruudun tilaa. (Flutter, n.d.a.) Se on eritoten hyödyllinen sovellusta rakennettaessa silloin, kun sivun ei tarvitse muuttua käyttäjän toimintojen takia. Kuva 2 näyttää teoreettisella tasolla, mihin StatelessWidget-luokassa sijoitetaan muuttujat, funktiot, widgetit ja renderöitävä build-osuus.

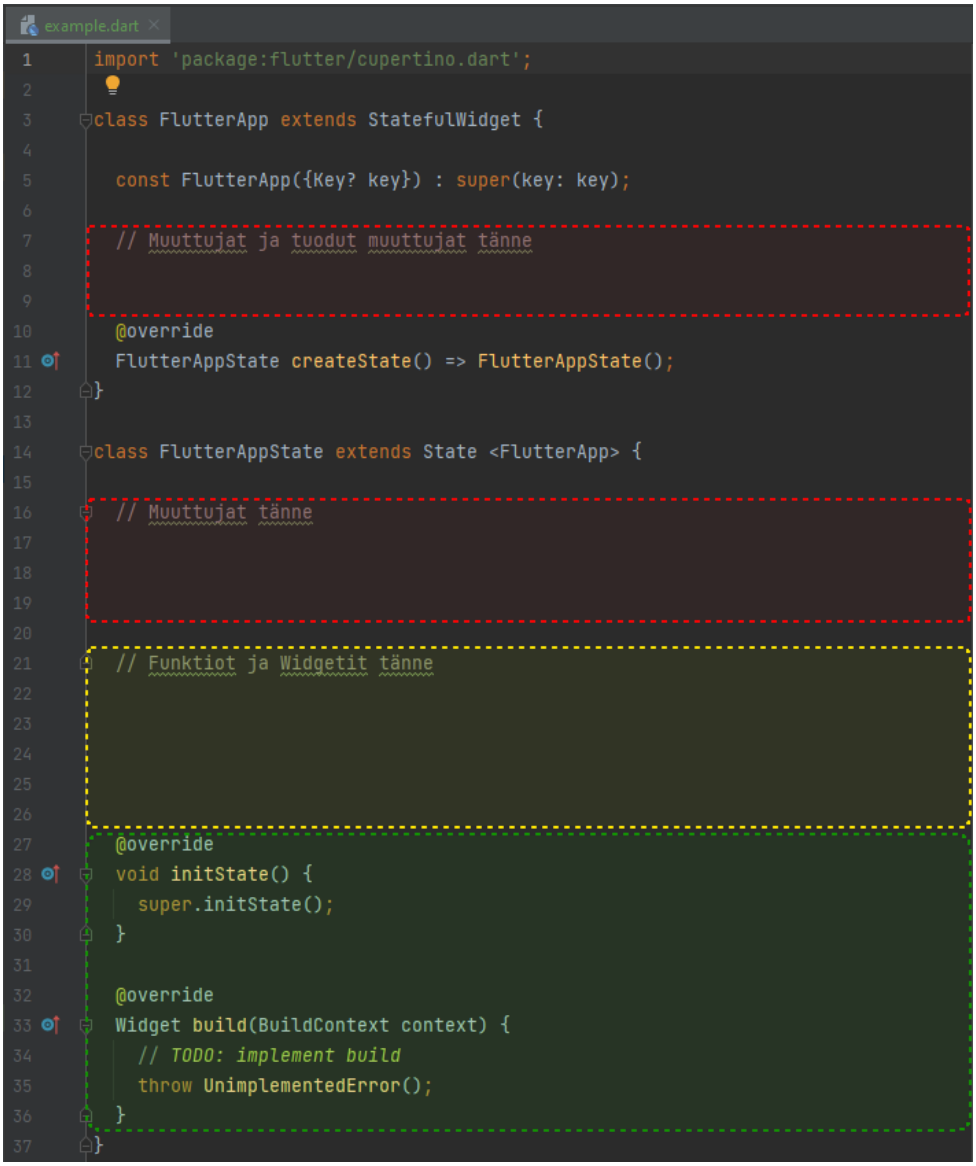
Kuva 2 – StatelessWidgetin yleinen rakenne ohjelmakoodissa.



2.2.2 StatefulWidget-luokka

StatefulWidget-luokka on tilallinen luokka, joka mahdollistaa dartissa käyttäjän toiminnoista johtuvan tilan muuttumisen. Näin ollen sovellukseen voidaan rakentaa muuttuvia olioita, jotka muistavat tilansa objektin eliniän ajan. (Flutter, n.d.b.)

Kuva 3 – StatefulWidgetin yleinen rakenne ohjelmakoodissa.



StatefulWidgetissä rakennetaan kaksi luokkaa, joista jälkimmäinen toimii ensimmäisen perijänä. StatefulWidgetissä periytyvän luokan syntaksi on muuten samanlainen, kuin StatelessWidget-luokan syntaksi. Kuva 3 näyttää teoreettisella tasolla, miten StatefulWidget rakennetaan toimivaksi kokonaisuudeksi.

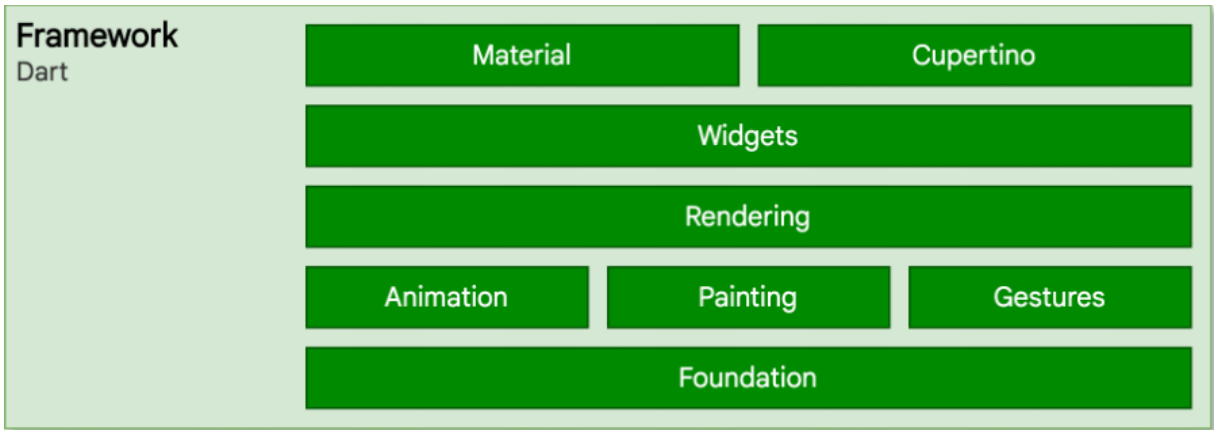
3 Flutter

Flutter on Googlen kehittämä järjestelmäriippumaton ohjelmointikirjasto. Flutter on tarkoitettu mobiilisovellusten ohjelmointiin ja siinä käytetään ohjelmointikielenä dartia. Kehitystyössä voidaan käyttää esimerkiksi Android Studio -nimistä editoria. Tässä kappaleessa käydään läpi Flutterin arkkitehtuuria ja koodin kääntymistä mobiili- ja web-kehityksessä.

3.1 Flutter ja mobiilisovellusten mobiilikehitys

Flutterin arkkitehtuurissa on tähdätty kerroksittaiseen ohjelmointiin, jossa jokainen yksittäinen kirjasto on riippuvainen niiden alemmista kerroksista. Jotkin kerrokset ovat vapaaehtoisia implementoida, joskin erittäin hyödyllisiä oikein käytettyinä.

Kuva 4 – Dartilla mobiilikehittämisen arkkitehtuurimalli. (Flutter, n.d.c.)



Kuvassa 4 alimmaisena näkyvä Foundation ja sen päällä olevat Animation, Painting ja Gestures edustavat sovellusrakenteen pohjaa. Foundation antaa luokillaan sovelluskehityksessä mahdollisuuden käyttää perustavanlaatuisia widgettejä.

Renderöintiin eli käyttöliittymän ruudulle piirtymiseen kuuluu elementtien piirtäminen, asettelu ja järjestys. Renderöintivaiheessa käyttäjä näkee konkreettisesti kaikki piirrettävät elementit ruudullaan.

Materialin alapuolelta löytyvät widgetit, jotka käsittelevät kaiken mitä ruudulle halutaan piirtää. Widgettien avulla pystytään määrittämään käyttöliittymään piirrettyjen elementtien tilat sekä

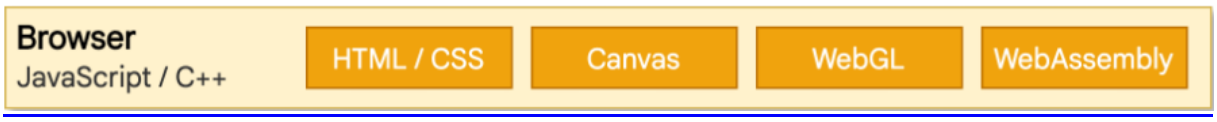
niiden visuaaliset rakenteet. (Flutter, n.d.d.) Widgeteillä pystyy sovelluksessa määrittelemään kaksi eri tilaa: StatelessWidget ja StatefulWidget. StatelessWidget-luokka on tarkoitettu widgeteille, joiden tilaa ei tallenneta käyttäjän toiminnoista riippumatta. Näitä widgettejä ovat esimerkiksi elementtien ikonit tai yleiset selitetekstit. StatefulWidget-luokka on tarkoitettu widgeteille, joiden tilan halutaan muistettavan koko objektin eliniän ajan. Käyttöliittymän tilaa vaihtaessa on aina pakko kutsua setState()-funktiota, jotta käyttöliittymän päivittäminen voi tapahtua.

Kuvassa päällimmäisenä näkyvät Material (Android) ja Cupertino (iOS), jotka edustavat rakenteen päällimmäistä kerrosta. Päällimmäinen kerros rakentaa MaterialApp-widgetin avulla perustan sovelluksen käännökselle Androidille ja iOS:lle. MaterialApp-widget pitää sisällään valmiita widgettejä. Näistä valmis-widgeteistä esimerkiksi voidaan ottaa Navigator-widget, jota kutsumalla voidaan vaihtaa ruutua toiseen. MaterialApp-widgettiä ei tarvitse käyttää sovelluskehityksessä, mutta siinä on paljon valmiita hyödyllisiä widgettejä nykyaikaisiin kehittämistarpeisiin. (Flutter, n.d.e.)

3.2 Flutter ja mobiilisovellusten web-kehitys

Flutterilla on mahdollista kehittää myös web-puolella. Web-puolen arkkitehtuurissa dart kääntyy joko HTML:ksi tai WebGL:ksi. HTML kääntyy käyttämällä hyödykseen CSS:ää ja Canvasta kun taas WebGL kääntyy käyttäen WebAssemblya.

Kuva 5 – Dartin verkkoselaimella kehittämisen arkkitehtuurimalli (Flutter, n.d.f.)

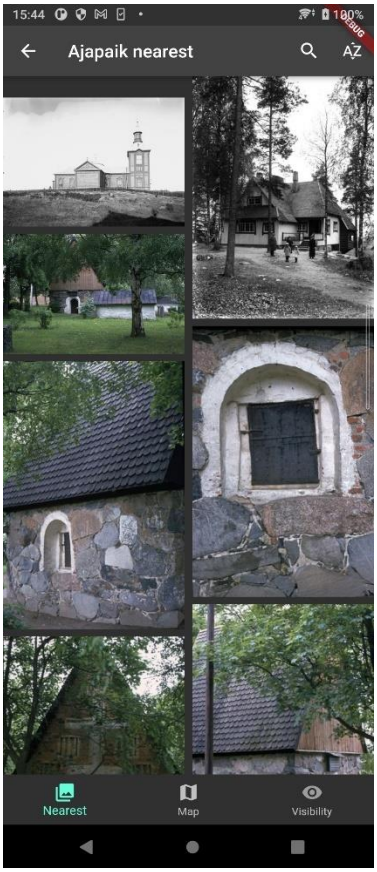


Sovelluskehityksessä dart kääntyy valmiiksi JavaScriptiksi, varmistaen moitteettoman web-puolen kehityksen kuvan 5 mukaisesti. Web-puolella kehittäessä sovelluskehitystä voidaan simuloida pääosin kaikissa verkkoselaimissa.

4 Ajapaik Flutter-sovellus

Lähtökohtana sovelluskehitykselle oli, että sovelluksessa oli sisällöllisesti valmiina albuminäkymä, kameran avaaminen ja sovellukseen sisäänkirjautuminen. Albuminäkymässä albumeita klikkaamalla pääsee niitä sisältävään kuvanäkymään, jossa kuvat on järjestetty ruudulle samalla tavalla kuin albumit kuvien 6 ja 7 mukaisesti.

Kuvat 6 ja 7 – Ensimmäisessä kuvassa näkyy albumien asettelu ja toisessa kuvien asettelu albumissa ”Ajapaik nearest”.



4.1 Albumit ja kuvat

Albumit kuvineen haetaan sovelluksessa GeoJSON-formaatilla. GeoJSON on kokonaisuudessaan yksi erillinen dart-tiedosto Ajapaik Flutter App -sovelluksen projektipuussa. Tällä formaatilla albumeja ja kuvia haettaessa saadaan albumien sisältämille kuville erilaisia tunnisteita, jotta yksittäisiä kuvia voidaan kutsua ohjelmakoodissa. Tunnisteisiin lukeutuu muun muassa kuvan nimi, kuvan verkko-osoite ja kuvanottokoordinaatit.

4.2 Kamera

Kameran käyttö sovelluksessa on ohjelmoitu erilliselle dart-tiedostolle. Camera.dart-tiedostossa on ohjelmoitu kameraan kuvafiltteri, jonka läpi kuva otetaan. Näin historiallisista kohteista saadaan helposti otettua uudet valokuvat, kun käyttäjä näkee mistä kulmasta kuva pitää ottaa. Jos käyttäjä on tyytyväinen ottamaansa kuvaa hän voi halutessaan tallentaa otetun kuvan Wikimedia Commonsin tai Ajapaikin palvelimelle. Tämä tapahtuu automaattisesti upload.dart-tiedoston kautta.

4.3 Sisäänkirjautuminen

Sovellukseen sisäänkirjautuminen tapahtuu sisäänkirjautumisruudussa, joka on ohjelmoitu login.dart-tiedostoon. Käyttäjän tunnistautuminen voidaan tehdä kirjautumalla Google-, Facebook- tai Wikimedia-tiliin. Wikimedia-tiliin kirjautumalla voidaan lähettää otettuja valokuvia Wikimedia Commons -palvelimelle.

4.4 Sovelluskehityksen menetelmät ja tavoite

Opinnäytetyössä tavoitellaan Ajapaik Flutter App -sovellukselle tietopohjaa, josta sovellusta tulevaisuudessa kehittävät voivat kerrata logiikkaa sen nykyisen toimivuuden taustoittamiseksi. Sovellusta on muiden aloittelevien kehittäjien tarkoitus muokata tulevaisuudessa oppimistarkoituksessa ja nykyinen versio on tehty Helsinki Rephotography -projektia varten. Opinnäytetyön menetelmänä toimivan sovelluskehityksen tarkoitus on saada uudelleenvalokuvaussovellus valmiiksi testikäyttöön osana Wikimedia Suomi ry:n Helsinki Rephotography-projektin palautusta varten.

Kehitystyön tavoitteena on saada aikaan toimiva versio, jossa on otettu huomioon modernit käyttöliittymäsuunnitteluun liittyvät ratkaisut. Nykyaikaisen sovelluksen tavoin kehitystyö tähtää myös sovelluksen kansainvälistämiseen ja lopuksi kotoistamiseen i18n- ja l10n-standardien mukaisesti.

5 Käyttöliittymäsuunnittelu ja saavutettavuus

Nykyajan mobiilisovelluskehityksessä tulee huomioida jokainen mahdollinen käyttäjä. Hyviin kehitysperiaatteisiin kuuluu laaja-alainen käyttöliittymäsuunnittelu ja saavuttavuuden käyttöönotto. Tässä luvussa tutustutaan siihen, mikä tekee käyttöliittymästä hyvän ja miten saavutettavuudella voidaan parantaa käyttäjäkokemusta.

5.1 Käyttöliittymäsuunnittelu mobiiliohjelmoinnissa

Mobiilisovelluksen käyttöliittymäsuunnittelua on ohjelmoijana tärkeä tarkastella loppukäyttäjän näkökulmasta. Hyvä käyttöliittymä on helppokäyttöinen, selkeä ja väljä. On myös hyvä huomioida, mihin elementtejä sijoittaa eri laitteiden kokoja ja muotoja silmällä pitäen. Joissakin sovelluksissa on myös tuki laitteen vaaka-asennossa selaamiseen, näin ollen on tärkeää myös rakentaa koherentti kokonaisuus elementeistä laitteen horisontaalista selaamista varten.

Jos suosittuja sovelluksia tarkastellaan, voidaan päätyä lopputulokseen, että jokainen sovellus on samankaltaistettu alla olevaan malliin. Samankaltaistuksella voidaan vaikuttaa käyttäjiin nopeuttamalla uuden sovelluksen käyttöönottoa ja potentiaalisesti saamaan käyttäjä palaamaan uudelleen sovelluksen pariin helppokäyttöisyyden ansiosta.

Kuva 8 – Nykyaikainen käyttöliittymäsuunnittelu suosituilla sovelluksilla. (Mielikäinen, 2022)

Nykyaikaisessa mobiilisovelluksen käyttöliittymässä ruudun yläosa (Kuvan 8, alue 1) on allokoitu puhelimen omille toiminnoille mm. indikoimaan aikaa, internet-yhteyttä ja kuuluvuutta. Ylä- tai alapalkin olemassaolo usein sulkee toisen pois. Ylä- tai alapalkki (Kuvan 8, alue 2 ja 5) pitää usein sisällään 5–7 painiketta, joihin riippuen sovelluksesta kuuluu mm. koti- ja etsi-painikkeet. Ruudun sivut (Kuvan 8, alue 4) ovat varattu muun muassa suosituissa kameraa käyttävissä sovelluksissa lisäpainikkeille ja keskiosa (Kuvan 8, alue 3) on varattu pääasiallisesti sisällölle.

5.2 Saavutettavuus mobiiliohjelmoinnissa

Saavutettavuus on tärkeä osa käyttöliittymäsuunnittelua ja siihen on hyvä pyrkiä sovelluksia ohjelmoidessa. Saavutettavuudessa pitää huomioida sovellukselle tarkoitettu ja sovellusta oletettavasti käyttävä kohdeyleisö. Saavutettavuuteen kuuluu esimerkiksi värisokeille tarkoitetut väriasetukset, elementtien ja tekstien koko tai mahdollisuus kuulla saneltu puhe tekstien sijaan. Saavutettavuus on tärkeä osa modernia sovelluskehitystä ja osittain lakisääteistäkin. Sillä voidaan varmistaa mahdollisimman monen käyttäjän sujuva käyttökokemus.

Flutterissa saavutettavuus merkitään lyhenteellä ”a11y”, joka on samalla standardoitu lyhenne saavutettavuudesta kaikessa verkkokehityksessä. Flutterissa saavutettavuuden kannalta tärkeät asiat ovat tarpeeksi suurien ja selkeiden fonttien käyttö, oikea määrä kontrastia ja mahdollisuus implementoida ruudunlukija. Kaikki kirjastossa olevat fontit ovat täysin muokattavissa teksti-widgeteiksi. Kontrastien suhteen Flutter suosittelee käyttämään W3C:n suosittelemaa kontrastirakennetta, joka on vähintään 4,5:1 pienelle tekstille ja vähintään 3:1 isolle tekstille. (Cooper ym., 2016) Ruudunlukijoista Flutter tukee mobiilialustoilla iOS:lle VoiceOver-nimistä ruudunlukijaa ja Androidille TalkBack-nimistä ruudunlukijaa. Web-puolella Flutter tukee työpöytäselaimilla MacOS:lle VoiceOver-ruudunlukijaa ja Windowsilla JAWs- ja NVDA-ruudunlukijoita. Web-kehityksessä kehittäjän pitää hyväksyä ruudunlukijan käyttö joko manuaalisesti tai ohjelmallisesti. Ohjelmallisesti hyväksyminen tapahtuu ohjelmakoodi 1 näyttämällä tavalla. (Flutter, n.d.g.)

Ohjelmakoodi 1 – Ohjelmallisesti tapahtuva ruudunlukijan käytön implementointi.

```
RenderBinding.instance.setSemanticsEnabled(true);
```

5.2.1 Kansainvälistäminen

Kansainvälistäminen (engl. internationalisation) ja kotoistus (engl. localisation) ovat saavutettavuuden muotoja, jotka täydentävät toisiaan sovelluksen suunnittelussa ja toteutuksessa. Kansainvälistämisellä sovelluskehityksessä pyritään mahdollistamaan lokalisaation käyttöönotto spesifille kohdeyleisölle. (Ishida ym., 2005a) Flutterissa kansainvälistäminen on hyvin dokumentoitu käytäntö ja sen merkintä tapahtuu viittaamalla siihen lyhenteellä ”i18n”. Kansainvälistäminen Flutterin sovelluskehityksessä tapahtuu MaterialApp- ja CupertinoApp-luokkien kautta. Myös luokat, jotka ovat kirjoitettu käyttäen WidgetsApp-luokkaa MaterialAppin tai CupertinoAppin sijasta voidaan myös kansainvälistää. (Flutter, n.d.h.)

5.2.2 Kotoistaminen

Sovelluskehityksessä sovellus voidaan kotoistaa kansainvälistämisen jälkeen. Flutterissa kotoistukseen viitataan lyhenteellä ”l10n”. Kotoistus käsittää muutokset, jotka ovat tarkoitettu spesifille kohdeyleisölle. Näihin muutoksiin lukeutuvat muun muassa tekstit, ajan erilaiset formaatit ja kulttuurillisesti helposti väärintulkittavat asiat. (Ishida ym., 2005b)

6 Lisäkirjastot

Flutter antaa kehittäjille mahdollisuuden hakea kirjastoja keskuspakettivarastosta, joka myös hallitsee kirjastojen automaattisen päivityksen. Paketeilla tarkoitetaan valmiita koodissa käytettäviä funktioita, jotka tarjoavat kehittäjille täsmäratkaisuja heidän tarpeisiinsa ohjelmointipuolella. Paketeiksi luetaan myös assetit ja fontit. Assetit on osio, joka voi pitää sisällään käyttäjän lataamia kuvia tai käyttäjän tekemiä JSON-tiedostoja.

Haettavia paketteja on tyypillisesti kahdenlaisia. Flutterin kehittäjien tekemiä lisäosia, jotka ovat useimmiten tehty kehittäjien pyynnöstä tai harrastelijoiden kehittämiä lisäosia. Varsinkin yksityisten kehittäjien tekemiä paketteja haettaessa on tärkeää huomioida, milloin viimeksi pakettia on päivitetty, mille alustoille kyseisellä paketilla on tuki ja onko paketilla vähimmäisvaatimusta mobiilikäyttöjärjestelmän versiolle. Yleisesti ottaen harvoin päivitetyt paketit saattavat lopettaa toimintansa Flutterin SDK:n päivittyessä, joka yleisesti saattaa jopa rikkoa koko sovelluksen toimivuuden. Jotkin lisäkirjastot eivät välttämättä tue jokaista alustaa. Joissakin karttapalvelujen lisäkirjastoissa ei ole ollenkaan esimerkiksi web-tukea, sillä ensisijaisesti karttapalvelut ovat yleisesti ottaen tarkoitettu mobiilialustoille. Joillakin lisäpaketeilla saattaa olla mobiilikäyttöjärjestelmän vähimmäisvaatimus. Näin ollen esimerkiksi iOS:lle kehittäessä tulee huomioida mahdolliset versioinnista johtuvat ongelmat, joissa esimerkiksi lisäpaketti vaatii vähintään iOS 9 tai uudemman version.

6.1 Lisäpakettien lataus ja asennus

Kaikki paketit ovat ladattavissa keskuspakettivarastosta https://pub.dev/packages -sivustolta ja ne asennetaan projektikansiosta löytyvään pubspec.yaml-tiedostoon. Pubspec.yaml on tiedosto, joka generoituu automaattisesti uutta Flutter-projektia aloittaessa. Tiedosto sijaitsee projektipuun yläosassa ja se sisältää metadataa projektista, jota Dart ja Flutter käyttävät toimiakseen. (Flutter, n.d.i.) Pubspec.yaml-tiedoston syntaksi on kirjoitettu YAML-kielellä ja siihen lisätyt paketit on pakko merkitä syntaksin mukaisesti ja oikeaoppisesti käyttäen TAB-painiketta välimerkin sijasta. Ohjeet asennukseen löytyvät lisäosan sivuilta asentamisen (engl. installing) osiosta. Paketteja voi asentaa komentojen 1 ja 2 mukaisesti Dart- ja Flutter-terminaaleilla sekä manuaalisesti. Ohjelmakoodi 2 näytetään, miten lisäosat asennetaan ohjelmallisesti pubspec.yaml-tiedostoon.

Komento 1 - Dart-terminaalilla asentaessa lisäosa asentuu viimeisimmällä versiolla.

```
$dart pub add lisäosan_nimi
```

Komento 2 - Flutter-terminaalilla asentaessa lisäosa asentuu viimeisimmällä versiolla.

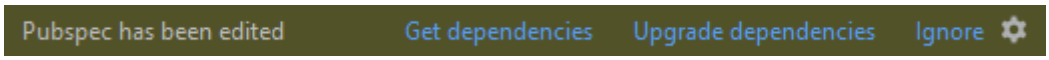
```
$flutter pub add lisäosan_nimi
```

Ohjelmakoodi 2 - Ohjelmallisesti asennettuna kehittäjä voi päättää, minkä version haluaa asentaa.

```
dependencies:
          lisäosan_nimi: ^1.2.3
```

Aina kun jotain lisätään tai poistetaan pubspec.yaml-tiedostosta Flutter kehottaa käyttäjää lataamaan paketit uudelleen kuva 9 näyttämällä tavalla, jotta ohjelmisto voi varmistaa pakettien toimivuuden yhdessä oman versionsa kanssa.

Kuva 9 – ”Get dependencies” lataa paketit uudelleen – ”Upgrade dependencies” päivittää automaattisesti **kaikki** lisäkirjastot viimeisimpiin versioihin – ”Ignore” jättää muutokset huomioimatta (ei suositeltu).



6.2 Lisäpakettien versioinnit

Paketteja päivittäessä pitää huomioida versioinnin muutokset. Flutterissa paketit käyttävät semanttista versiointia ja täten tulevat aina muodossa **^a.b.c+d.**

”a” merkitsee MAJOR-versionumeroa. Tämän numeron kasvaessa paketilla ei normaalisti ole taaksepäin yhteensoveltuvuutta aikaisemman version kanssa. Jos paketti on käytössä koodissa, voi kehittäjä olettaa sovelluksensa menneen rikki päivityksen myötä. Paketin sivustolla changelog-osiossa on ohjeet, miten vanhaa koodia tulee muuttaa, jotta se toimii päivitetyssä versiossa.

”b” merkitsee huomattavaa paketin muutosta. Muutos ei ole kuitenkaan niin suuri, että sillä tarvitsisi muuttaa myös MAJOR-versionumeroa. Tämän numeron muuttumisen myötä paketilla on

vielä mahdollisesti taaksepäin yhteensoveltuvuus. Kehittäjiä suositellaan silti menemään katsomaan changelog-osiosta muutokset, jotka mahdollisesti voivat rikkoa heidän sovelluksensa.

”c” merkitsee MINOR-versionumeroa. Paketin muutos on erittäin pieni. Tämän numeron muuttumisen myötä paketilla on yleisesti ottaen taaksepäin yhteensoveltuvuus. Päivitys pakettiin saattaa olla esimerkiksi jokin uusi ominaisuus tai jopa muutos readme.txt-tiedostossa.

”+d” merkitsee joko korjausta tai lisäominaisuutta MINOR-versionumeroituun pakettiin. +d-versiointia käytetään yleensä pelkästään 0-alkuisessa MAJOR-versionumeroinnissa.

Liitteessä 2 käydään läpi pubspec.yaml-tiedoston muoto ja millaiselta sen pitäisi näyttää, jotta kaikki lisäpaketit voi ottaa käyttöön ongelmitta.

7 Karttapalvelu

Flutterissa on mahdollista ottaa sovelluksen käyttöön karttapalveluksi kaksi isoa valmista lisäkirjastoa: Google Maps ja Flutter Map. Näistä isoista kirjastoista huomattavasti suositumpi on Google Maps, jonka kehittäjinä toimii Google. Google Maps tarjoaa lisäkirjastonsa täysin ilmaiseksi sovelluskehityksessä käyttöön, jos kartan lataukset ovat hyvin vähäisiä. Muutoin Google Maps käyttää dynaamista laskutusta, mikä riippuu karttapalvelun latausten määrästä ja Street Viewin käytöstä. (Google Maps Platform, n.d.)

Flutter Map on dartiksi käännetty lisäkirjasto Leafletistä. Leaflet on JavaScriptillä kirjoitettu avoimeen lähdekoodiin perustuva kevyt ja interaktiivinen karttapalvelu, joka on polveutunut OpenStreetMaps-nimisestä karttapalvelusta. (Agafonkin, 2010) Flutter Map on täysin ilmainen ja yhteensopiva useiden karttasovelluksia täydentävien lisäkirjastojen kanssa ja se on ilmainen vaihtoehto Flutterissa karttasovellusta kehittäville kehittäjille.

Karttapalvelua valittaessa on syytä ottaa huomioon sovelluksen käyttötarkoitus, jotta yllättäviltä kuluilta voi säästyä. Hinnoittelun lisäksi oleellinen ero on myös yksityisyydensuoja. Googlen karttapalveluja käytettäessä käyttäjä hyväksyy Googlen pilven käytön, joka tiettävästi kerää tietoja käyttäjästä muun muassa paikkatietojen avulla.

8 Sovelluksen viimeistely

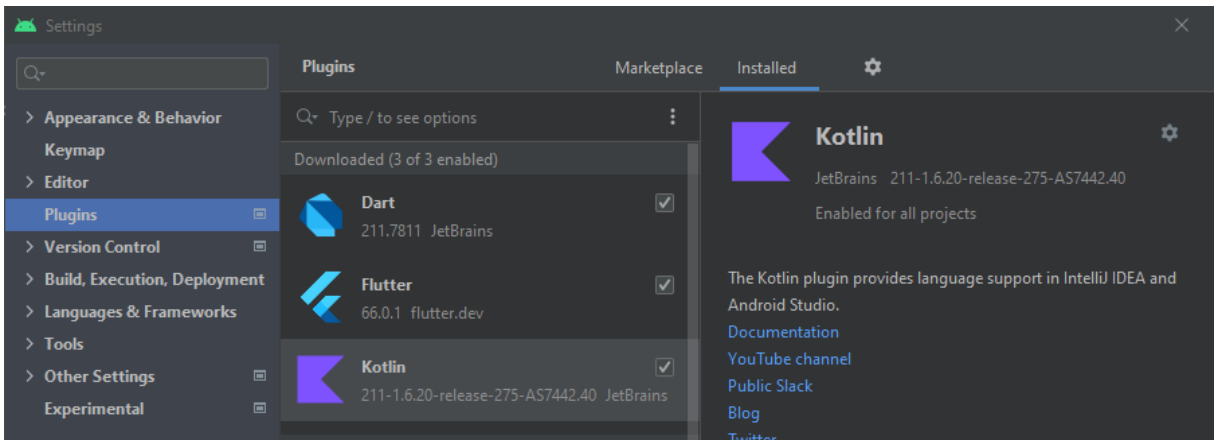
Flutter on tehnyt sovelluskehittäjille tarkistuslistan, jota suositellaan läpikäytäväksi oman sovelluksen kanssa ennen sovelluksen virallista julkaisua sovelluskauppoihin. Tarkistuslistan saavutettavuudesta tarkistettavat asiat ovat ruudunluvun testaus, suositusten mukaiset kontrastisuhteet, ikonien koot ja elementtien näkyvyys värisokeille tarkoitetuilla asetuksilla.

Sovelluksen toimivuuteen liittyvät asiat on myös hyvä tarkistaa. Näihin lukeutuu elementtien vuorovaikutukset ja käyttäjän syötteen kontekstuaaliset vaihdokset. Lopuksi on hyvä tarkistaa sovelluskehitysympäristön terminaalissa sovellusta normaalissa käytössä. Tämä on ainoa tapa varmistaa sovelluksen toimivuus virheilmoitusten varalta. Virheilmoitukset saattavat aiheuttaa muistivirheitä, jotka voivat johtaa sovelluksen kaatumiseen. Tarkistusten ja mahdollisten korjausten jälkeen sovellus voidaan julkaista Flutterin suositusten mukaisesti. (Flutter, n.d.j.)

9 Flutterin käyttöönotto

Flutteria voidaan kehittää useammalla editorilla. Android Studion, IntelliJ IDEAn ja VS Coden joukosta valitsin editoriksi Android Studion. Sovelluskehittämisen ensimmäinen askel Flutterilla on, että Android Studiossa ladataan Flutter- ja dart -lisäosat. Lisäosat ovat haettavissa Android Studion kauppapaikasta navigoimalla asetukset > lisäosat > kauppapaikka (engl. marketplace). Kun lisäosat ovat asentuneet löytyvät ne kuva 10 mukaisesti Android Studion asetuksista lisäosien ”asennettu”-välilehdestä (engl. installed).

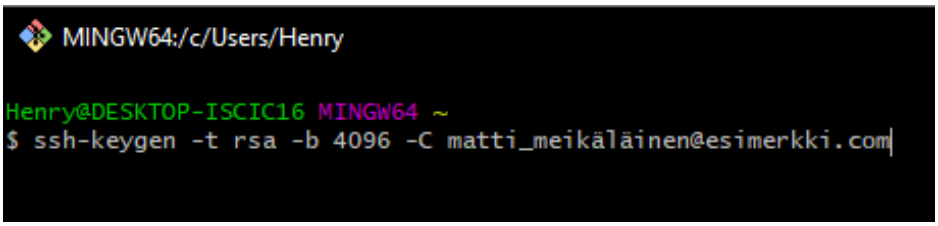
Kuva 10 – Android studion ”asennettu”-välilehti.



9.1 GitHubin käyttöönotto Android Studiossa

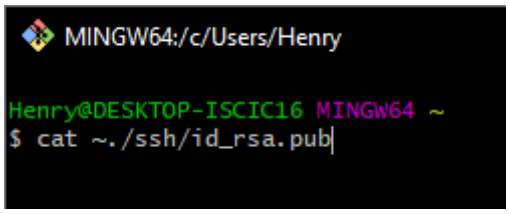
Seuraavaksi kehittäjän tulee liittyä Ajapaikin flutterapp-projektiin GitHubissa. GitHub on integroitu valmiiksi Android Studioon. Käyttöönotto tapahtuu tekemällä käyttäjän ensin GitHubiin, sen jälkeen lataamalla Gitin. Gitin asennuttua avaamalla Git Bash päästään terminaaliin, jossa luodaan SSH-avain ja liitetään se SSH-agenttiin. Tämä mahdollistaa projektin liittämisen GitHubissa kehittämistä varten.

Komento 3 – SSH-avaimen luominen. Matti_meikäläinen@esimerkki.com tulee korvata GitHubiin rekisteröimällä sähköpostilla.



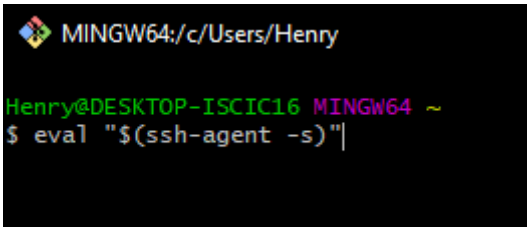
Komento 3 luodaan SSH-avain. Sen suorittamisen jälkeen terminaali kysyy mihin tiedostoon avain halutaan tallentaa ja halutaanko avaimelle vaihtaa nimi. Oletuksena avain tallentuu polkuun ”C:\Users\Tietokoneen_nimi\.ssh” ja on nimeltään ”id_rsa.pub”. Avaimen voi tallentaa myös salasanalla. Gitin generoitua SSH-avaimen se pitää avata komento 4 mukaisesti.

Komento 4 – Terminaaliin kirjoitettava komento, jolla avataan luotu SSH-avain.



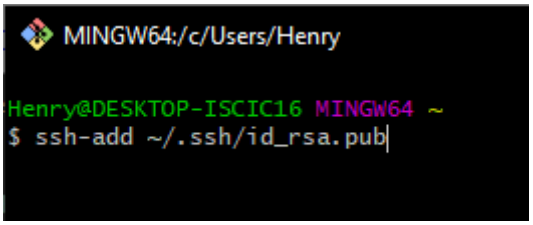
Oletuksena tiedoston nimi polussa on id_rsa.pub, mutta Gitin komennossa polun loppuosa pitää muuttaa, jos nimeä on muuttanut SSH-avainta luodessa. Terminaalin komentorivi palauttaa SSH-avaimen sisällön. Avaimen sisältö kopioidaan kokonaisuudessaan komentoriviltä ja kirjaudutaan GitHubiin. Kirjauduttua GitHubiin pitää navigoida oikean yläkulman avatariin ja sieltä navigoida ”Settings”. Asetuksissa löytyy sivun vasemmasta laidasta ”SSH and GPG keys”. Klikkaamalla ”New SSH key” voidaan liittää leikepöydällä oleva SSH-avain ja antaa tunnisteeksi projektille nimi, sillä GitHubissa voi olla useammassa projektissa eri SSH-avaimia ja näin ollen tunnisteen antaminen eri SSH-avaimille on tärkeää. Avaimen lisäämisen jälkeen on jälleen mentävä Git Bash-terminaaliin ja käynnistettävä SSH-agentti. Tämä tapahtuu komento 5 näyttämällä tavalla.

Komento 5 – SSH-agentin käynnistäminen tapahtuu terminaalissa kuvassa olevalla komennolla.



Kuvassa olevalla komennolla terminaalin pitäisi palauttaa SSH-agentin sarjanumero ja täten SSH-agentti on käynnissä. Tämän jälkeen lisätään komento 6 mukaisesti SSH-avain SSH-agenttiin.

Komento 6 – SSH-avaimen lisääminen SSH-agenttiin.



Lopuksi Git Bash-terminaalissa siirrytään polkuun, mihin halutaan kloonata ajapaik_flutter app. Liikkuminen poluissa Git Bashilla tapahtuu komento 7 näyttämällä tavalla.

Komento 7 – Esimerkkinä polku, johon on navigoitu kloonaamista varten ja komento, jolla kloonataan ajapaik_flutter app.

Sen jälkeen, kun kloonaus on suoriutunut onnistuneesti, voidaan sovellusta alkaa kehittämään GitHubin kautta. Vaihtoehtoisesti voidaan vielä tehdä oma haara (engl. branch) johon omaa kehitystyötä voidaan huolettomasti tallentaa. Oma haara projektiin voidaan tehdä komento 8 mukaisesti Git Bashin komentorivillä.

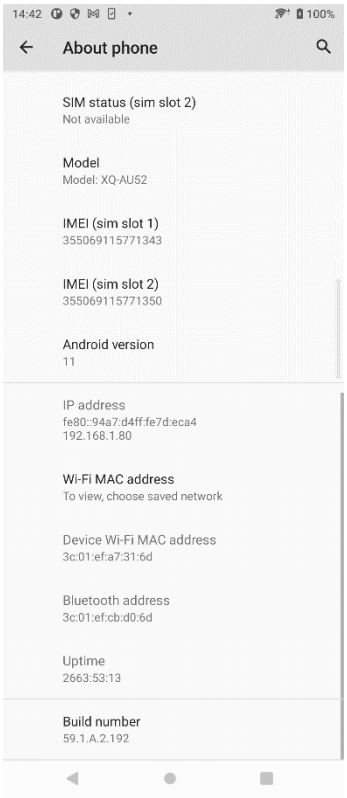
Komento 8 – Git-komento, jolla voidaan GitHubiin tehdä oma haara (engl. branch).

```
git branch oman_haaran_nimi
```

9.2 Lisälaitteiden käyttöönotto sovelluksen emuloimista varten

Android Studioon voidaan halutessa myös kytkeä lisälaite simuloimaan kehitettävää ohjelmaa. Android-laitteiden käyttöönotto kehitysympäristöön tapahtuu ottamalla käyttöön sovelluskehittäjän asetukset (engl. developer options). Tämä tapahtuu kuvan 11 mukaisesti navigoimalla Android-laitteesta kohtaan asetukset > tietoja puhelimesta. Riippuen järjestelmäversiosta, joissain Android-laitteissa pitää navigoida asetukset > järjestelmäasetukset > tietoja puhelimesta.

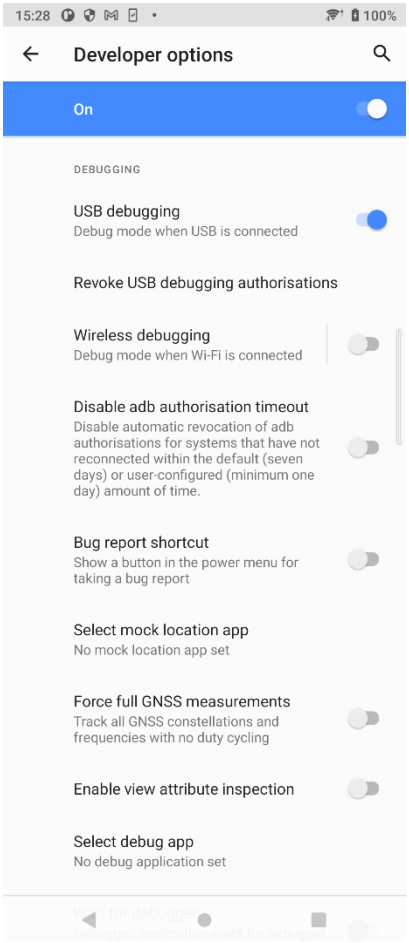
Kuva 11 – ”Tietoja puhelimesta”-valikko Android-laitteessa



Sovelluskehittäjän asetukset otetaan käyttöön painamalla koontiversionumero-kohtaa (engl. build number) seitsemän kertaa. Tässä kohtaa on hyvä huomioida, että laite ilmoittaa koontiversionumero-kohtaa painaessa, jos sovelluskehittäjän asetukset ovat laitteessa jo avattu. Kun sovelluskehittäjän asetukset ollaan otettu käyttöön, ne löytyvät kuvan 12 mukaisesti

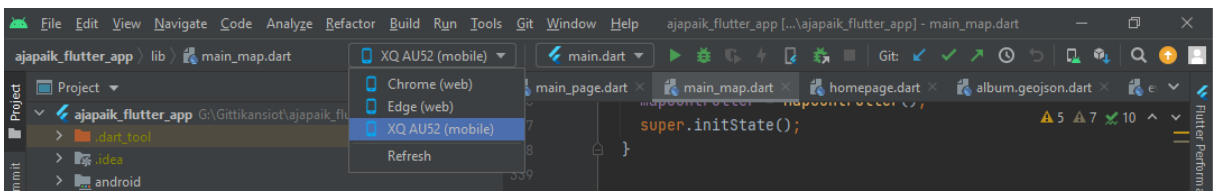
navigoimalla laitteesta asetukset (engl. settings) > järjestelmä (engl. system) > lisäasetukset (engl. advanced) > sovelluskehittäjän asetukset. Sovelluskehittäjän asetuksista navigoidaan kohtaan vianetsintä (engl. debugging) ja laitetaan päälle USB-vianetsintä. (engl. USB debugging)

Kuva 12 – USB-vianetsintä kytketty laitteessa päälle.



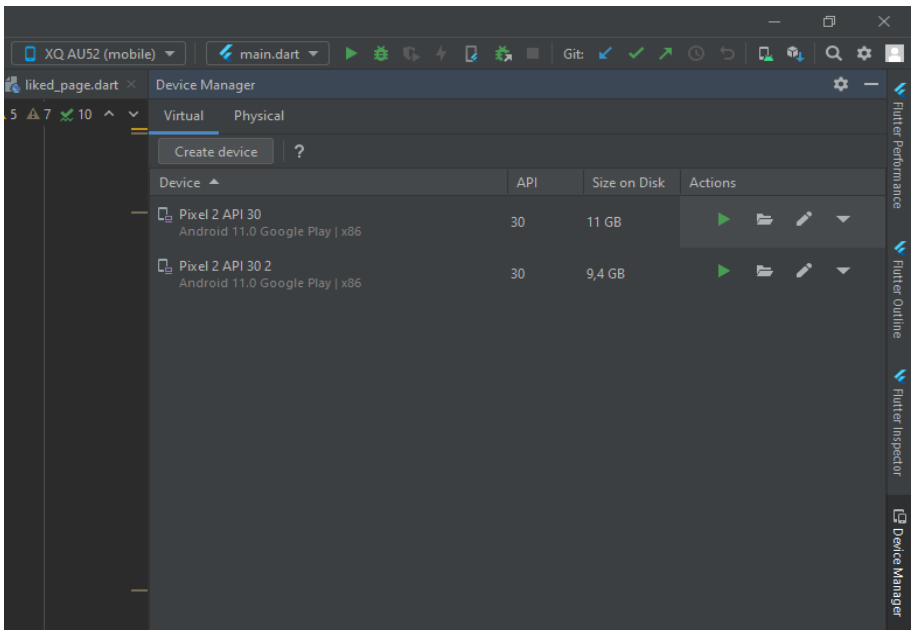
Kun USB vianetsintä on kytketty laitteesta päälle, pitäisi laitteen näkyä automaattisesti Android Studion ”Valitse Flutter-laite” -pudotusvalikossa seuraavalla käynnistyskerralla. Kuva 13 nähdään millainen näkymä Android Studiossa on valittaessa laitteita emulointia varten.

Kuva 13 – ”Valitse Flutter-laite” -kohdassa on Chrome- ja Edge -verkkoselaimet web-kehitystä varten oletuksena, jos kehittää Windowsilla ja käyttöjärjestelmään on asennettu Google Chrome.



Android Studiossa on myös mahdollista kehittää sovellusta emulaattorin avulla. Virtuaalisen laitteen saa luotua Android Studiossa navigoimalla laitehallintaan (engl. device manager) ja sieltä luomalla uuden laitteen kuvan 14 näyttämällä tavalla.

Kuva 14 – Laitehallinta, josta saa luotua virtuaalisen laitteen kehittämistä varten.



Virtuaalisesta laitteesta saa muokattua omiin kehittämistarpeisiin sopivan valitsemalla haluamansa lukuisista eri tyypeistä ja malleista. Sen jälkeen kun laite on luotu, se ilmestyy web-selainten ja fyysisten liitettyjen laitteiden joukkoon ”Valitse Flutter-laite” -pudotusvalikkoon.

10 Ajapaik Flutter App-sovelluskehitys

Tämä luku pitää sisällään kaiken Ajapaik Flutter Appiin implementoidut sovelluskehitysratkaisut. Soveltavassa osiossa on käytetty teoriaosiossa läpikäytyjä ratkaisumalleja. Sovelluksessa käytettävien muuttujien nimet ohjelmakoodissa ovat aina ohjelmakoodi 3 mukaisesti muuttujia kuvaavia ja niissä käytetään lowerCamelCase-formaattia.

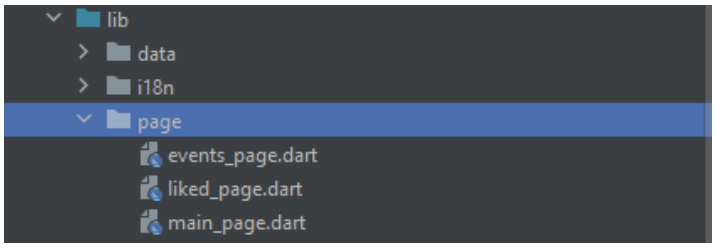
Ohjelmakoodi 3 – Muuttujien nimet ovat kuvaavia ja niissä käytetään lowerCamelCase-formaattia.

```
bool open = false;
bool busy = false;
bool toggle = false;
double userLatitudeData = 0;
double userLongitudeData = 0;
double mapLatitude = 0;
double mapLongitude = 0;
int maxClusterRadius = 100;
String orderBy = "alpha";
String orderDirection = "desc";
List<Marker> markerList = [];
List<Bounds> boundsList = [];
List<Color> _colors = [];
```

10.1 Sisältösivujen määrittäminen ja alapalkin toiminnot

Ajapaik Flutter appin juuritiedosto on main.dart. Main.dartissa sijaitsevassa widget buildissä palautetaan sovelluksen kotisivu. Sovelluksen kotisivu rakentuu tiedostossa HomePage.dart, jossa oletusetusivu on sisältösivu nimeltään Main_Page.dart. Sisältösivut sijaitsevat ”lib”-kansion alla olevassa sivukansiossa kuva 15 nähtävällä tavalla.

Kuva 15 – Sivu-kansio Ajapaik Flutter Appin projektipuussa.



Näin ollen sovelluksen kotisivun tarkoitus on pysyä aktiivisena niin pitkään, kunnes käyttäjä päättää tarkastella kuvia klikkaamalla niitä. Tämän ratkaisun mahdollisti ruutujen (engl. screens) implementoinnin sovellukseen.

Ohjelmakoodi 4 – Miten ruutujen implementointi tapahtuu ohjelmakoodissa.

```
final screens = [
  MainPage.network("",
  const LikedPage(),
  const Text(''),
  const EventsPage(),
  const Text(''),
];
```

Alapalkin (engl. BottomNavigationBar) toimivuus perustuu sisältöruutujen vaihtamiseen IndexedStack-olion kautta. Ohjelmakoodissa ruutujen vaihtaminen tapahtuu alapalkin kautta liitteen 3 näyttämällä tavalla. Homepage.dart-tiedostossa sijaitsevassa widget build-rungossa on IndexedStack-niminen olio. Tämä olio ottaa muuttujakseen funktion onItemTapped() indeksi arvon ja käyttää lapsena ohjelmakoodi 4 ruutuja. Alapalkissa olevat kohteet (engl. items) ovat bottomNavigationBar-olion luomia ja riippuen mikä kohde on valittuna, niin sen kohteen ikoni näkyy korostetusti turkoosina. Ohjelmakoodi 5 funktio käsittelee valitun kohteen tilan muutokset.

Ohjelmakoodi 5 – onItemTapped()-funktio muuttaa ruudun tilan valinnan käyttäjän toimintojen perusteella.

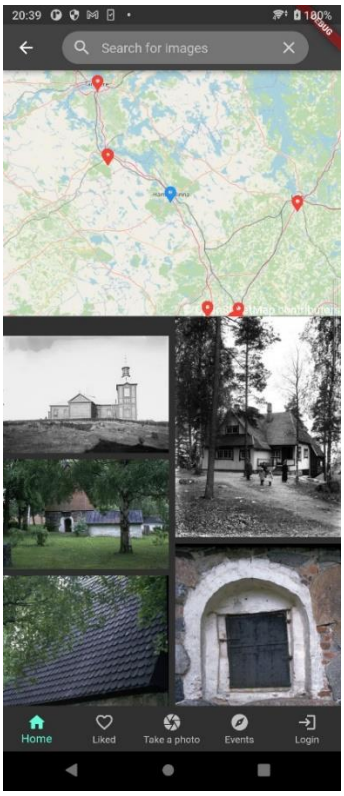
```
void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}
```

Käytännössä tämä ratkaisu mahdollisti pelkän sisällön muuttumisen sovelluksessa klikkailemalla alapalkissa olevia vaihtoehtoja. Ruutujen vaihto alapalkin kautta ei poista yläpalkkia (engl. appBar) eikä alapalkkia.

10.2 Hakupalkin implementoiminen

Yläpalkkiin implementoin kuvan 16 tavoin kuvien etsimispalkin. Palkkia voisi periaatteessa käyttää myös muissa sisältöruuduissa, jos sen ohjelmakoodia muokkaa.

Kuva 16 – Valmiissa sovelluksessa etsimispalkki on ruudun yläpalkissa



Etsimispalkin nykyinen toiminto on hakea kuvia avainsanoilla filtterin-omaisesti ja hakusanojen perusteella haut päivittyvät käyttäjälle sisältöruudussa. Käytännössä onnistuneeseen lopputulokseen pääsin alustamalla tekstikontrollerin (engl. TextController) ohjelmakoodi 6 mukaisesti ja implementoimalla hakupalkin.

Ohjelmakoodi 6 – Tekstikontrollerin alustaminen

```
final myController = TextEditingController();
```

Tekstikontrollerin alustamisen jälkeen piti yläpalkkiosioon tehdä tekstiä vastaanottava laatikko, jonka kautta kuvia pystyttäisiin suodattamaan. Tämä tapahtui yläpalkkielementissä ohjelmakoodi 7 näyttämällä tavalla.

Ohjelmakoodi 7 – Yläpalkin ohjelmallinen sisältö, joka pitää sisällään hakupalkin tyylilliset ja toiminnolliset ratkaisut.

```
appBar: AppBar(
  title: Container(
    alignment: Alignment.center,
    width: 300,
    height: 40,
    decoration: BoxDecoration(
      color: Colors.grey[600],
      borderRadius: const BorderRadius.all(
        Radius.circular(20),
      ),  // BorderRadius.all
    ),  // BoxDecoration
    child: Center(
      child: TextField(
        controller: myController,
        textInputAction: TextInputAction.go,
        textAlign: TextAlign.start,
        onSubmitted: (value) {
          setState(() {
            refresh();
          });
        },
        onChanged: (value) => onSearchTextChanged(value),
        decoration: InputDecoration(
            border: InputBorder.none,
            hintText: 'Search for images',
          prefixIcon: IconButton(
              onPressed: () {
                setState(() {
                  refresh();
                });
                onSearchTextChanged('');
              }, icon: const Icon(Icons.search)),  // IconButton
          suffixIcon: IconButton(
              onPressed: () {
                myController.clear();
              }, icon: const Icon(Icons.clear))),  // IconButton, InputDecoration
      ))),  // TextField, Center, Container, AppBar
```

Aluksi alustettiin kontti (engl. Container), jonka tarkoituksena oli ottaa vastaan lapsielementti. Konttia piti muuttaa niin, että se mahtuu yläpalkkiin ylittämättä sille määritettyä tilaa. Kontin taustan väriä piti muuttaa, jotta valkoinen kirjoitus näkyy paremmin. Lapsielementti kontille on tekstialue (engl. TextField). Tekstialueelle asetettiin tekstikontrolleri, jotta tekstialueelle kirjoitettu teksti muuttuu automaattisesti muuttujaksi. Sanan haku voidaan suorittaa puhelimen näppäimistöstä ENTER-painikkeesta tai tekstialueen vieressä sijaitsevasta ikonista. Kun käyttäjä hakee hakusanalla uusia kuvia, niin sovellus laukaisee ohjelmakoodi 8 näkyvän refresh()-funktion. Tämä funktio odottaa ohjelmakoodi 9 näkyvän getDataSourceUrl()-funktion tulosta. Sen saatuaan funktio hakee verkosta uudet tulokset käyttäen hakusanaa muuttujana ja tulokset saatuaan mahdollistaa StatefulWidget-näkymän uudelleenpiirtämisen.

Ohjelmakoodi 8 – Refresh()-funktiossa alustetaan tekstimuuttuja ”url” kutsumalla funktiota getDataSourceUrl().

```
void refresh() async {
  if (mapLatitude == 0 || mapLongitude == 0) {
    var geoPosition = await Geolocator.getCurrentPosition(
        desiredAccuracy: LocationAccuracy.high);
    mapLatitude = geoPosition.latitude;
    mapLongitude = geoPosition.longitude;
  }
  String url = getDataSourceUrl();
  if (tweenCompleted == true) {
    tweenCompleted = false;
    await (_albumData = fetchAlbum(http.Client(), url,
            latitude: mapLatitude, longitude: mapLongitude)
        .whenComplete(() => {
                tweenCompleted = true
            }));
  }
}
```

Ohjelmakoodi 9 – getDataSourceUrl()-funktio rakentaa url-muuttujan, joka ottaa parametrikseen hakukentästä tulevan tekstimuuttujan.

```
String getDataSourceUrl() {
  String url = widget.dataSourceUrl;
  if (url.contains("?")) {
    url += "&orderby=" + orderBy + "&orderdirection=" + orderDirection;
  } else {
    url += "?orderby=" + orderBy + "&orderdirection=" + orderDirection;
  }
  String searchkey=myController.text;
  url += "&search=" + searchkey;
  return url;
}
```

Ohjelmakoodissa 9 näkyvän GetDataSourceUrl()-funktion tarkoitus on käyttää tekstialueella olevaa hakusanaa lisäämällä se url-muuttujaan ja järjestää kuvat sen mukaan, mitä hakusanaa on haettu. Url-muuttujan avulla refresh()-funktio hakee GeoJSON-tiedoston, joka sisältää lähistöllä olevat kuvat hakusanalla suodatettuna. Tekstialueella on myös ruksi, mistä painaessa tekstialue tyhjenee.

10.3 Sisältösivun rakentaminen kuvien ja kartan kanssa

Sovelluksen pääsisältösivun rakentamisessa implementoin karttapalvelun sovellukseen, jotta kuvat voitiin siirtää karttaan merkeiksi. Merkkejä painamalla kuvia pystytään tarkastelemaan halutessa ruudun alhaalta nousevassa sisältölaatikossa. Sovelluksen karttapalvelu sisältää karttapalveluun tarvittavat käyttäjän, ja kuvien sijaintien piirtämisen mahdollistavat funktiot ja kartan dynaamisen liikuttelun, jossa kuvat vaihtuvat kartan kameran sijainnin mukaan.

Ohjelmakoodi 10 - Karttapalvelu implementoitiin projektiin lisäämällä tarvittavat lisäkirjastot Pubspec.yaml-tiedostoon.

```
29  dependencies:
30    flutter_map: 0.14.0
31    latlong2: ^0.8.1
32    location: ^4.3.0
33    geolocator: ^8.0.3
```

Ohjelmakoodi 10 esiintyvissä lisäkirjastoissa flutter_map on projektissa käytettävä karttapalvelu ja latlong2 mahdollistaa koordinaattien käytön muuttujina valmiissa karttasovelluksessa. Location-lisäkirjasto ja geolocator-lisäkirjasto toimivat yhteen, sillä location mahdollistaa laitteesta paikallistamisen ja geolocator taas mahdollistaa laitteen paikan seuraamisen lisäkirjastosta tulevan koodiin implementoitavan olion kanssa.

Ohjelmakoodi 11 – Kartassa käytettävä valmiiksi alustetut muuttujat.

```
bool open = false;
bool busy = false;
double userLatitudeData = 0;
double userLongitudeData = 0;
double mapLatitude = 0;
double mapLongitude = 0;
List<Marker> markerList = [];
List<Color> _colors = [];
late final MapController mapController;
StreamSubscription<Position>? _positionStream;
```

Muuttujia alustettiin sitä mukaan, mitä koettiin tarpeelliseksi karttasovelluksen lisätoimintoja kehittäessä. Huomion arvoisia muuttujia ohjelmakoodi 11 ovat kartan kontrolleri, jonka tuomat takaisinsoittotoiminnot (engl. callback functions) mahdollistavat karttasovelluksen dynaamisuuden ja paikallistamisvirta, (engl. position stream) joka mahdollistaa käyttäjän reaaliaikaisen paikannuksen.

Jotta karttasovellus saadaan käyttäjälle toimivaksi pitää se ensin suorittaa FutureBuilder-oliolla. Karttasovelluksen toimivuus perustuu eri funktioiden eri ajoituksiin, jotka halutaan koostaa yhteen samaan aikaan ajoituksista huolimatta. FutureBuilder-oliolla voidaan varmistaa, ettei laitteen sijaintipalvelujen hitaus vaikuta käyttäjän paikannukseen karttasovelluksessa. Sovelluksen siirtyessä etusivulle se käy aluksi ohjelmakoodi 12 näkyvät vaiheet läpi. Olio odottaa kaikkien funktioiden palautukset ja näyttää käyttäjälle latauspalkin. Kun olio on valmis palautusten kanssa, niin latauspalkki vaihtuu viewTab-funktioon, joka renderöi käyttäjälle karttasovelluksen kuvineen ja toimintoineen. Jos olion saamat funktioiden palautukset kohtaavat virheitä, niin sovellus ilmoittaa siitä ”snapshot error”-ilmoituksella. Näin ollen sovelluksen etusivu voidaan palauttaa renderöitäväksi ohjelmakoodi 13 mukaisesti.

Ohjelmakoodi 12 – FutureBuilder-olio, joka rakentaa karttasovelluksen taustalla ja näyttää pyörivää latauspalkkia.

```
body: Column(
  children: [
Flexible(
    child: FutureBuilder<List<Album>>(
      future: albumData(context),
      builder: (context, snapshot) {

        print("Future: main_page.dart)");
        if (snapshot.connectionState != ConnectionState.done) {
          return const Center(child: CircularProgressIndicator());
        }

        if (snapshot.hasError) (snapshot.error);

        return (snapshot.hasData)
            ? _viewTab(snapshot.data)
            : const Center(child: CircularProgressIndicator());
      },
    )),  // FutureBuilder, Flexible
],
),  // Column
```

Ohjelmakoodi 13 – Sisältösivun renderöitävä osuus.

```
@override
Widget build(BuildContext context) {

  return Scaffold(
    body: Column(children: [
      SizedBox(height: 300, child: _buildFlutterMap(context)),
      Flexible(
        child: photoView(context),
      )  // Flexible
    ]),  // Column
  );  // Scaffold
}
```

Karttasovelluksen sisältävää oliota tarkastellaan kahdessa eri osassa. Ensimmäinen osa pitää sisällään kartan dynaamisuuden kameran sijainnin perusteella ja toinen osio on kartan palautus renderöitäväksi toiminnallisuuksineen.

Ohjelmakoodi 14 – Karttasovelluksen ensimmäinen osa, jonka funktio mahdollistaa kartan dynaamisen toimivuuden.

```
Widget _buildFlutterMap(BuildContext context) {

  LatLng lastposition = LatLng(mapLatitude, mapLongitude);

  Future<void> timerFunction(mapPosition) async {

    List<Album>? a = await widget.callbackFunction(mapPosition);

    if (a != null) {
      print(a.first.features.first.properties.name);
      if (a.first.features.first.properties.name !=
          widget.albums!.first.features.first.properties.name) {
        setState(() {
          widget.albums = a;
        });
      }
    }
  }
```

Ohjelmakoodi 15 – Karttasovelluksen dynaamisuuden mahdollistuu vertailemalla koordinaatteja.

```
Future<List<Album>>? mapAlbumRefresh(MapPosition mapPosition) {
  if (mapPosition.center != null) {
    var center = mapPosition.center!;
    double lat2 = center.latitude;
    double lon2 = center.longitude;
    double distance =
        calculateDistance(mapLatitude, mapLongitude, lat2, lon2);

    if (distance > 0.1) {
      Future<List<Album>>? a;
      _albumData = a;
      mapLatitude = lat2;
      mapLongitude = lon2;
      refresh();
    }
  }
  return _albumData;
}
```

Ohjelmakoodi 16 – Internet-foorumeilta haettu valmis koodi, joka laskee etäisyyttä maapallolla ja ottaa huomioon maapallon pyöreyden. (Stack Overflow, 2019)

```
double calculateDistance(lat1, lon1, lat2, lon2){
  var p = 0.017453292519943295;
  var c = cos;
  var a = 0.5 - c((lat2 - lat1) * p)/2 +
      c(lat1 * p) * c(lat2 * p) *
          (1 - c((lon2 - lon1) * p))/2;
  return 12742 * asin(sqrt(a));
}
```

Ohjelmakoodissa 14 sijaitseva ”_buildFlutterMap”-olio renderöi käyttäjälle karttasovelluksen. Olion koordinaattimuuttuja ”lastposition” saa arvonsa ohjelmakoodi 15 näkyvästä mapAlbumRefresh()-funktiosta. Main_page.dart-tiedostosta löytyvä mapAlbumRefresh ()-funktio ottaa parametrikseen arvon mapPosition, jonka avulla voidaan saada vertailtavaksi kartan kameran keskipisteenä käytettäviä koordinaatteja. Jos funktion uudelleen laukaisemisen aikaan kartan kameran keskipisteen etäisyys on 0.1-arvoa (kts. Ohjelmakoodi 15) eri, kuin kartan alkuperäiset koordinaatit, niin refresh()-funktio antaa sovellukseen käytettäväksi sen alueen kuvat ja kameran keskipisteen koordinaatit vaihtuvat alkuperäisten tilalle. Ohjelmakoodi 16 käsittelee etäisyyksien laskentaa sovelluksessa. Funktio ottaa huomioon maapallon pyöreyden etäisyyksien laskennassa. Ohjelmakoodi 14 käytettävä timerFunction laukaistaan kolmen sekunnin välein ja

joka laukaisukerralla se odottaa mapAlbumRefresh()-funktiosta tulevaa albumidataa ennen seuraavaa laukaisua. Samainen funktio myös asettaa uudet kuvat kartalle, jos mapAlbumRefresh()-funktiosta palautuneiden kuvien nimet ovat erisuuret nykyisten piirrettyjen kuvien kanssa.

Kartan palautus käyttäjän ruudulle renderöitäväksi tapahtui Flutter Map-lisäkirjaston oliolla ”FlutterMap” ohjelmakoodi 17 mukaisesti.

Ohjelmakoodi 17 – FlutterMap-olion palautus funktioineen ja asetuksineen.

```
    return FlutterMap(
        mapController: mapController,
        options: MapOptions(
          onPositionChanged: (mapPosition, boolValue) async {
            lastposition = mapPosition.center!;
            t.cancel();
            t=Timer(const Duration(seconds: 3), ()=>timerFunction(mapPosition));
          },
          center: LatLng(mapLatitude, mapLongitude),
          interactiveFlags: InteractiveFlag.pinchZoom | InteractiveFlag.drag,
          zoom: 17.0,
          maxZoom: 18,
        ),  // MapOptions
        layers: [
          TileLayerOptions(
            urlTemplate: "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            subdomains: ['a', 'b', 'c'],
            attributionBuilder: (_) {
              return const Text("© OpenStreetMap contributors");
            },
          ),  // TileLayerOptions
          MarkerLayerOptions(markers: getMarkerList(context)),
          MarkerLayerOptions(markers: [
            Marker(
                width: 40.0,
                height: 40.0,
                point: LatLng(userLatitudeData, userLongitudeData),
                builder: (ctx) =>
                  const Icon(Icons.location_pin, color: Colors.blue)),  // Marker
          ])  // MarkerLayerOptions
        ]);  // FlutterMap
  }
}
```

FlutterMap-olio oli hyvin muokattavissa, mutta oletusarvoisesti se oli hyvin pelkistetty. Sovelluskehityksen tavoite kuitenkin oli päästä tilanteeseen, jossa sovellusta pystyttäisiin testikäyttämään valokuvakävelyillä. Tämä johti ongelmanratkaisutilanteeseen, jossa piti ratkoa miten karttaa liikutellessa käyttäjä voisi nähdä lisää kohteita riippuen kameran keskipisteestä, miten hakua pystyttäisiin rajaamaan siten, ettei uutta hakua tehdä joka kerta kun käyttäjä liikuttaa

karttaa ja miten voidaan pitää sovellus responsiivisena käyttäjälle ja palvelimelle toistuvista hauista huolimatta.

Olioon saatiin lisää toimintoja lisäämällä siihen karttakontrolleri, (engl. mapController) jotta kameran keskipisteen koordinaatteja voitiin käyttää erillisinä muuttujina ja kartan takaisinsoittofunktioita pystyttiin palauttamaan. Oliossa käytettävä takaisinsoittofunktio perustuu siihen, mitä automaattisesti tapahtuu karttaa liikuttaessa. Tämä automaatio asettaa lastposition-muuttujaan kartan senhetkisen keskipisteen, jotta keskipistettä voidaan vertailla mapAlbumRefresh()-funktiossa. Takaisinsoittofunktioon on asetettu kolmen sekunnin viiveen, jotta palautuksia ei tule liian montaa lyhyessä ajassa. Liian monta palautusta ilman viivettä johtaa sovelluksen kaatumiseen funktion aiheuttaman rasituksen takia.

Olio piirtää karttaan kuvat merkkeinä kohdassa MarkerLayerOptions, jonne annettiin arvoksi getMarkerList()-funktion parametrinään konteksti. getMarkerList()-funktiossa (kts. Liite 4) alustetaan tyhjä lista, joka ottaa arvokseen albumidatasta haetut kuvat. getMarkerList()-funktioon, tuli myös lisätä karttasovellukseen merkit, joita painamalla kartassa voidaan esikatsella merkin sisältämää kuvaa.

Liitteen 4 ohjelmakoodin näyttämällä tavalla listaan asetetaan albumista haetut lähialueen kuvat ja tehdään kuvista erilliset merkit (engl. markers) FlutterMap-oliota varten. Merkkien toimivuuteen on ohjelmoitu kaksi boolean-muuttujaa hallinnoimaan, milloin merkki on aktiivisena ja milloin merkki ei ole aktiivisena. Aktiivisena merkki on valkoinen ja passiivisena punainen. Alustettaviin muuttujiin on ohjelmoitu merkkien avaamista varten alustetun ”showBottomSheet”-materiaali. Nämä materiaalit pitävät sisällään merkin sisältämän kuvan esikatselun merkkiä painamalla. Kuvan esikatselua painamalla pääsee kuvan esikatseluruutuun, jonne ohjelmallisesti viedään myös kuvan tiedot. Lopuksi funktio palauttaa merkkilistan, jotta se voidaan antaa myöhemmin parametriksi karttaa renderöidessä.

Karttasovellukseen piti saada myös käyttäjää esittävä merkki. Tämä merkki on ohjelmoitu sovellukseen ja se on väriltään sininen. Käyttäjän merkin koordinaatit haetaan ohjelmakoodi 18 näyttämällä tavalla, jossa koordinaattien hakemisen lisäksi otetaan myös paikkavirta ja jos haettujen koordinaattien arvot ovat erisuuria haetaan ne uudelleen ohjelmakoodi 19 näyttämällä tavalla. Paikannusvirta mahdollistaa merkin liikkumisen laitteen sijainnin perusteella. Jos laite

liikkuu, niin sovellus palauttaa getCurrentLocation()-funktiota, joka päivittää käyttäjän sinistä merkkiä kartalla.

Ohjelmakoodi 18 – Funktio mahdollistaa paikannusvirran käyttämisen käyttäjän laitteen paikannusta varten.

```
void listenCurrentLocation() {
  LocationSettings locationSettings = const LocationSettings(
      accuracy: LocationAccuracy.best,
      // distanceFilter: 10,
      timeLimit: Duration(seconds: 5));  // LocationSettings
  _positionStream =
      Geolocator.getPositionStream(locationSettings: locationSettings)
          .listen((Position geoPosition) {
    if (geoPosition.latitude != userLatitudeData &&
        geoPosition.longitude != userLongitudeData) {
      setState(() {
        userLatitudeData = geoPosition.latitude;
        userLongitudeData = geoPosition.longitude;
      });
    }
  });
}
```

Ohjelmakoodi 19 – Funktio käyttää geolocator-lisäpakettia paikallistamaan laitteen koordinaatit.

```
void getCurrentLocation() async {
  var geoPosition = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.best);

  setState(() {
    userLatitudeData = geoPosition.latitude;
    userLongitudeData = geoPosition.longitude;
  });
}
```

Kun oliota ei enää tarvita siirryttäessä esimerkiksi sivulta toiselle, voidaan paikannusvirta automaattisesti sulkea käyttämällä ohjelmakoodi 20 näkyvää hävitysfunktiota, joka täten vapauttaa olioiden varaamat resurssit.

Ohjelmakoodi 20 – Hävitysfunktion sisällä olevat funktiot suljetaan sivulta siirtymisen jälkeen.

```
@override
void dispose() {
  _positionStream?.cancel();
  super.dispose();
}
```

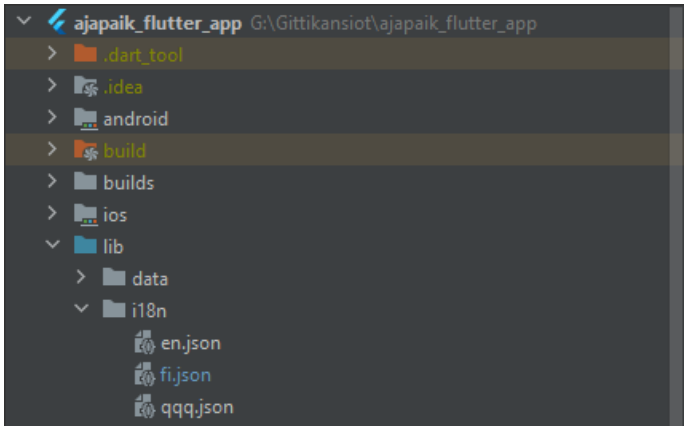
11 Sovelluksen kansainvälistäminen ja kotoistaminen

Tässä luvussa käydään läpi, miten sovellus on kansainvälistetty ja lokalisoitu Flutterin dokumentaatioiden mukaisesti. Sovelluksen tekstiosat on kehitetty englanniksi ja tekstiosien kotoistus, on todennettu toimivaksi vaihtamalla laitteen kieli suomeksi.

11.1 Sovelluksen kansainvälistäminen

Sovelluksen kansainvälistäminen tapahtui kuva 17 mukaisesti lisäämällä projektipuun lib-kansion alle uusi kansio nimeltä ”i18n”. Tähän kansioon lisättiin kaksi JSON-tiedostoa edustamaan natiivikieltä ja käännettävää kieltä maiden kansainvälisten lyhenteiden mukaisesti ”FI” ja ”EN”. Natiivikieli sovelluskehityksessä oli englanti. Kolmas JSON-tiedosto on nimeltään qqq.json ja se pitää sisällään käännettävien tekstimuuttujien metadatan.

Kuva – 17 Projektipuussa olevat kansainvälistämiseen tarkoitetut kansiot ja tiedostot.



Tämän jälkeen Pubspec.yaml-tiedostoon piti lisätä riippuvuuksien (engl. dependencies) alle ohjelmakoodi 21 mukaisesti uusi riippuvuus, jotta kotoistaminen voidaan suorittaa ohjelmakoodin

kautta. Riippuvuuksiin piti lisätä myös ohjelmakoodi 22 näkyvä ”assets”-osio, jotta sovellus löytää käynnistyessään projektipuusta kotoistamiseen tarvittavat JSON-tiedostot.

Ohjelmakoodi 21 – Pubspec.yamlissa käytettävä ohjelmakoodi, joka mahdollistaa kotoistamisen

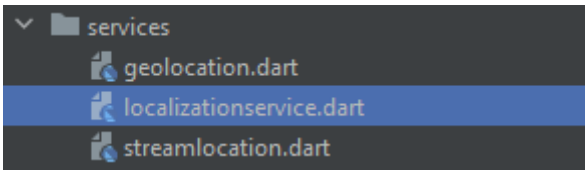
```
flutter_localizations:

        sdk: flutter
```

Ohjelmakoodi 22 – Pubspec.yaml-tiedostoon liitettävä ”assets”-osio.

```
85  flutter:
86    assets:
87      - lib/i18n/en.json
88      - lib/i18n/fi.json
89      - lib/i18n/qqq.json
```

Jotta sovelluksen käynnistämisen yhteydessä kotoistaminen toimii, tehtiin projektipuuhun localizationservice.dart-niminen tiedosto. Tiedosto on valmista ohjelmakoodia Flutterin virallisesta dokumentoinnista. Kuva 18 näkyvä localizationservice.dart-tiedosto on osa projektipuun ”Services”-kansiossa olevia dart-tiedostoja. Näitä tiedostoja ei tarvitse renderöidä sovelluksessa ja ovat nimensä mukaisesti palveluja.

Kuva 18 – Lib-kansioon lisättävä kotoistamisen mahdollistama .dart-tiedosto



Main.dart tiedostoon piti tuoda (engl. import) localizationservice.dart-tiedosto. Tuominen tapahtuu ohjelmakoodi 23 mukaisesti. Tiedostojen tuomiset tapahtuvat ”import”-osiossa poikkeuksetta aina dart-tiedoston ohjelmakoodirivillä 1.

Ohjelmakoodi 23 – Sivun tuominen toiselle sivulle, jotta siihen voidaan käyttää viittauksia

```
import 'localizationservice.dart';
```

Tuonnin jälkeen Main.dart-tiedoston Widget buildissä piti muuttaa GetMaterialApp-palautusta. Palautukseen piti saada localizationservice.dart-tiedostoon viittaava materiaali, jotta kotoistaminen voidaan suorittaa sovelluksen avaamisen yhteydessä. Ohjelmakoodi 24 muutettu GetMaterialApp sisältää Flutterin dokumentoinnista tuodun ohjelmakoodin, jotta kotoistaminen tapahtuu sovelluksen käynnistyessä.

Ohjelmakoodi 24 – Kotoistamiseen vaadittu ohjelmakoodi. (Flutter, n.d.k.)

```
return GetMaterialApp(
  supportedLocales: const [
    Locale('en', 'US'),
    Locale('fi', 'FI'),
  ],
  localizationsDelegates: const [
    AppLocalizations.delegate,
    GlobalMaterialLocalizations.delegate,
    GlobalWidgetsLocalizations.delegate,
  ],

  localeResolutionCallback: (locale, supportedLocales) {
    for (var supportedLocaleLanguage in supportedLocales) {
      if (supportedLocaleLanguage.languageCode == locale?.languageCode &&
          supportedLocaleLanguage.countryCode == locale?.countryCode) {
        return supportedLocaleLanguage;
      }
    }

    // If device not support with locale to get language code then default g
    return supportedLocales.first;
  },
```

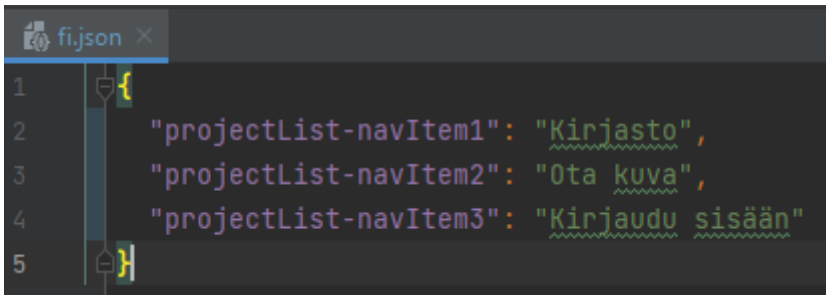
Näiden askeleiden jälkeen sovelluksen kansainvälistäminen on valmis ja pystytään jatkamaan sovelluksen kotoistamisella.

11.2 Sovelluksen kotoistaminen

Sovelluksen kotoistaminen tapahtuu i18n-kansioon luomissa JSON-tiedostoissa ja sovelluksessa käytettävissä tekstimuuttujissa. Nykyisessä sovelluksessa JSON-tiedostot ovat maakoodeiltaan suomi (FI) ja englanti (EN). Metadataa sisältävän qqq.json-tiedoston kautta sovelluksessa olevat tekstimuuttujat käännetään suomen tai englannin kielelle metadatassa selitettävän kontekstin mukaisesti. Tämä käännöstyö on tarkoitus toteuttaa translatewikissä, joka on yhteisöllinen käännösten tekemiseen tarkoitettu alusta ilmaisille ohjelmistoprojekteille. JSON-tiedostojen sisällä

käytettävä rakenne mukailee normaalia JSON-tiedostossa käytettävää formaattia. Aaltosulkeet pitävät sisällään rivi kerrallaan muuttujan ja käännettävän tekstin.

Ohjelmakoodi 25 – fi.json -tiedoston sisältö, jossa muuttujille annetaan arvot.



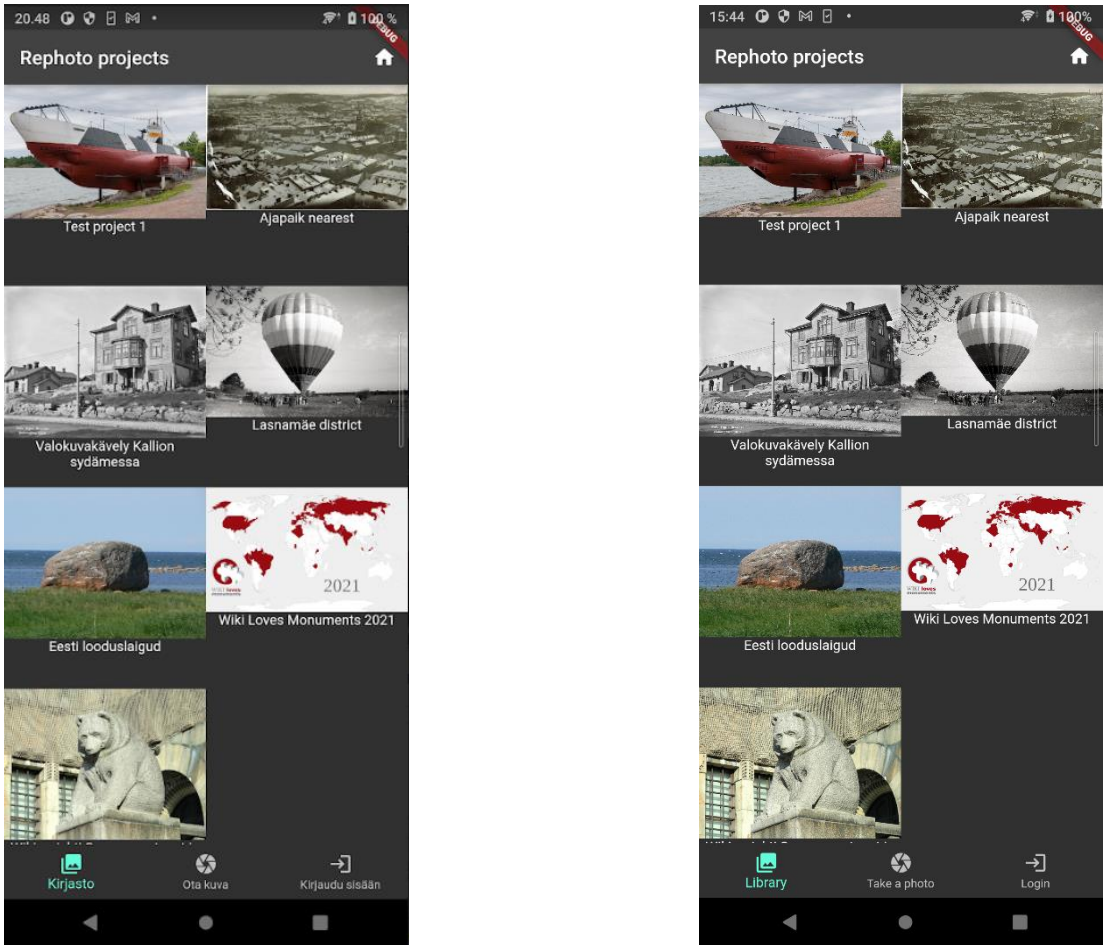
Ohjelmakoodin 25 mukaan kaikille JSON-muuttujille annetaan tunnistettava nimi tiedoston ja muuttujan mukaan. Esimerkissä on käytetty alapalkin ikonien tekstimuuttujia. Tämän jälkeen JSON-muuttujiin voidaan lisätä kotoistettava teksti natiivikielellä. Lopuksi sovelluksen ohjelmakoodissa korvattiin kaikki tekstiin viittaavat muuttujat ohjelmakoodi 26 mukaisesti.

Ohjelmakoodi 26 – Tekstimuuttujat sovelluksessa kotoistamisen jälkeen.

```
(AppLocalizations.of(context)!.translate('json_muuttuja'))
```

Käytännössä sovelluksen käynnistyessä GetMaterialApp kysyy localizationservice.dart-tiedostolta, mitä kieliä sovellus tukee. Maakoodin tullessa takaisin main.dart-tiedostoon koodinpätkä vertailee, onko takaisin tullut maakoodi sama, kuin mitä sovelluksessa on ennalta määritettyjä tuettuja kieliä. Jos maakoodi on sama, niin sovellus kotoistuu kyseiselle kielelle sovelluksen tekstiolioihin. Jos maakoodit ovat eri, niin sovellus kotoistuu ensimmäiselle kielelle, joka esiintyy tuetuissa kielissä. Ensimmäinen tuettu kieli sovelluksessa on englanti.

Kuva 19 – Sovellus natiivikielellä ja kotoistettuna.



Kuva 19 näkyy vierekkäin sovellus natiivikielellä ja kotoistettuna. Sovelluksen alapalkkien tekstit ovat kotoistamisen jälkeen riippuvaisia käyttäjän laitteen kieliasetuksista.

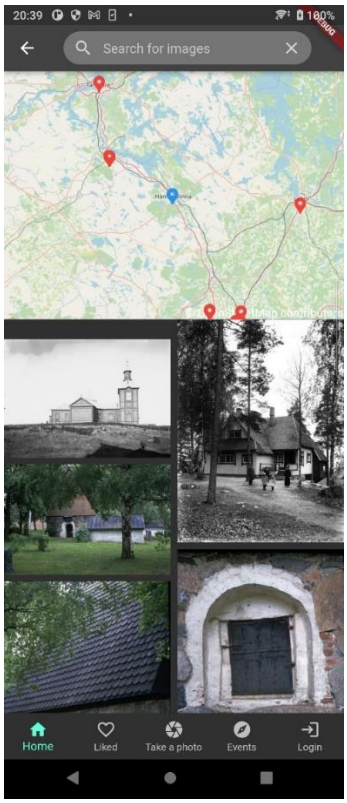
12 Johtopäätökset ja pohdinta

Opinnäytetyön tarkoitus oli toimittaa toimiva testisovellus valokuvakävelyitä varten. Tässä luvussa käydään läpi johtopäätökset lopputuloksesta ja vastataan viimeiseen tutkimuskysymykseen ohjaajan haastattelun avulla ja oman kokemuksen perusteella.

12.1 Opinnäytetyön oma-arviointi

Testisovellusta varten käytin laajasti opinnäytetyössäkin tutkimaani teoriaa sovelluksen hyödyksi, jotta paras mahdollinen tulos saataisiin aikaan. Testisovellus rakentuu ilman virheilmoituksia ja on näin ollen valmis tulevia testauksia varten. Kuvan 20 mukaisesti käyttöliittymäsuunnittelussa sovelsin tutkimaani teoriaa lopputuloksessa toimivasti ja sain kansainvälistettyä ja kotoistettua sovelluksen moitteettomasti. Sovellus käyttää käynnistyessä laitteessa olevaa kieltä ja jos kieli ei löydy kotoistettavista kielistä, niin sovellus käyttää kielenään englantia.

Kuva 20 – Testisovelluksen etusivu on teoriassa käydyn käyttöliittymäsuunnittelun mukainen ja valmis testattavaksi.



12.2 Opinnäytetyön työnantajan palaute

Palautetta työstäni sain runsaasti. Pääpainot työlläni oli toiminnallisuus ja visuaalisuus. Toiminnallisuuden osalta käyttöliittymässä suurin osa elementeistä toimi niin kuin oli tilattu. Käyttäjätestauksessa toukokuussa 2022 albumi ja karttatoiminnallisuus toimivat ongelmitta. Kuvien etsiminen toimii, mutta suodatusominaisuuden hakusanojen lisääminen ja poisto jäi tekemättä ajan loputtua. Karttatoiminnot toimivat myös niin kuin oli hahmoteltu toimivan, mutta ne tarvitsevat vielä laajaa optimointia funktioiden palautusten kanssa karttaa liikutellessa. Kartan pieneneminen kuvia selatessa ja kuvien pieneneminen karttaa selatessa ei ole implementoitu, mutta ne ovat suunnitteilla tulevaisuudessa. Visuaalisella puolella etusivun kanssa päädyin ratkaisuun, joka antaa konseptuaalisen käsityksen käyttöliittymästä karttapalvelusovelluksessa. Kuvien esikatselusivut oli myös tehty työnantajan ohjeiden mukaisesti myötäillen Ajapaik Android Appissä tehtyjä ratkaisuja. Työnantaja olisi jälkikäteen myös nähnyt konkreettisen lisäarvon, jos yksi opinnäytetyön tavoitteista olisi ollut sovelluksen julkaiseminen App Storeen mahdollisimman aikaisin. Tätä kautta olisi päästy nopeammin tilanteeseen, jossa olisi saatu laajempaa käyttäjän palautetta.

12.3 Ohjaajan haastattelu ja pohdinta Flutterista opetettavana ohjelmointikielenä

Wikimedia Foundation on kansainvälinen taho, joka pyörittää wikipediaa. Yli kymmenen vuotta sitten Wikipedia pyöri pääsääntöisesti vapaaehtoisvoimin, mutta entiteetin laajentuessa viimeisen kymmenen vuoden aikana tahon sisällä on oltu sitä mieltä, että Wikipedia hyötyisi ohjelmointiryhmien perustamisesta. Wikipediassa on paljon projekteja, missä käsitellään dataa ja näin ollen Wikimedialla on tarve ihmisille, joilla on ohjelmointikokemusta.

Projektini on osa Wikimedian Helsinki Rephotography -hanketta ja projektin tarkoitus oli myös luoda alusta tuleville Wikimedian mobiilisovellusprojekteihin osallistuville, jotka kokeilevat mobiiliohjelmointia ja oppia tekemästäni. Ohjaajani mukaan aikaisemmin Wikimedian projekteissa on pääsääntöisesti käytetty ohjelmointikielenä Javaa. Javan perusrakenteen on kuitenkin todettu olevan liian vaikeaa opetettavaksi tasolla, jolla pitäisi saada merkityksellisiä muutoksia aikaan mobiilisovelluksen käyttöliittymässä. Yksi isoimmista ongelmista Javan ohjelmoinnin kanssa on, että se vaatii liikaa ymmärrystä luokkien periytyvyyksien kanssa, jotta aloittelevat ohjelmoijat voisivat tehdä muutoksia valmiiseen ohjelmakoodiin ja näin ollen muuttaa esimerkiksi

käyttöliittymän elementtien asettelua. Tämä tekee projektien pitämisen aktiivisena pitkään vaikeaksi varsinkin, jos useampi ohjelmoijista on aloittelijoita. Ohjaajani mukaan Flutter ja dart ratkaisevat usean aloittelevan ohjelmoijan ongelmia yksinkertaisuudellaan. Dart helpottaa ohjelmoijan työtä luokkien vähäisyydellään ja muuttujien ja funktioiden lähetystä sivulta toiselle on helppoa ja loogista seurata ohjelmakoodissa. Dartin syntaksin yksinkertaisuus helpottaa aloittelevia ohjelmoijia jatkamaan ohjelmointia siitä, mihin ohjelmakoodi on jätetty muokattavaksi. Laitevaatimukset ovat myös kevyempiä Flutterilla, kuin Javalla, joka potentiaalisesti mahdollistaa useamman aloittelevan ohjelmoijan kehitystyön.

Rinnastaen omaa kokemusta ohjaajani haastattelun perusteella voin todeta että, Flutter alustana ja dart ohjelmointikielenä ovat oivat työkalut mobiiliohjelmoinnin opettamiselle. Dartin syntaksin helppous mahdollistaa nopean oppimiskaaren, jonka seurauksena ohjelmakoodin muokkaamista on helppo tehdä tavalla, joka tuntuu ohjelmoijalle merkitykselliseltä. Flutter ei nojaa liikaa lisäpakettien varaan, vaan antaa kehittäjälle työkalut tehdä muutoksia myös lisäpakettien sisällä oleviin elementteihin. Flutterissa käyttöliittymän muokkaaminen on isossa roolissa ja sen kanssa voi tehdä suuriakin muutoksia elementteihin. Tämä mahdollistaa innovatiivisen käyttöliittymäsuunnittelun ja sen toteutuksen sekä antaa vapaammat kädet keskeneräisen työn ymmärtämiselle ja sen muokkaamiselle.

12.4 Johtopäätökset

Opinnäytetyössäni sain dokumentoitua tarpeellisen teoriatiedon, joka mahdollisti sovelluskehityksen dokumentoinnin sen pohjalta. Koen, että sain vastaukset asettamiini tutkimuskysymyksiin projektin edetessä luontevasti ja vastaukset on esitetty opinnäytetyössä asianmukaisesti ja selvästi.

Opinnäytetyön ensimmäinen soveltava osio käsitteli myös tutkimuskysymyksenä esitettyä nykyaikaisen käyttöliittymäsuunnittelun mahdollisuutta Flutterissa. Vastauksen saaminen tuli luontevasti käyttöliittymää suunnitellessa. Flutter tarjoaa kehittäjälle kaikki työkalut mahdollistaakseen nykyaikaisen käyttöliittymäsuunnittelun ja toteutuksen. Käytin näitä työkaluja ratkaisussa, joka antoi konseptuaalisen käsityksen siitä, miltä käyttöliittymä voisi näyttää uudelleenvalokuvaussovelluksessa. Opinnäytetyön seuraava isompi osio liittyi sovelluskehitykseen. Sovellusta kehittäessä pyrittiin projektin monikielisyyden vuoksi vastaamaan kysymykseen, miten

Flutterilla pystytään kansainvälistämään ja kotoistamaan sovellus. Kysymykseen vastaaminen tapahtui luontevasti Flutterin suhteellisesti valmiiden dokumentaatioiden perusteella. Dokumentaatioita soveltamalla mahdollistin asianmukaisen kansainvälistämisen ja sovellus on ohjelmallisesti valmis kotoistettavaksi translatewikin avulla. Viimeinen tutkimuskysymys, onko Flutter hyvä kehitysalusta opettaa mobiiliohjelmointia oli kokonaisvaltaisempi kysymys, mihin vastausta pystyttiin miettimään vasta projektin tultua päätökseensä. Ohjaajan haastattelun ja oman kokemani perusteella tulin lopputulokseen, jossa pystyn suosittelemaan Flutteria mobiiliohjelmoinnin kehitysalustana aloitteleville kehittäjille. Helppokäyttöisyyden ja muokattavuuden ansiosta uudetkin kehittäjät pystyvät tuomaan projekteihin lisäarvoa ja samalla projektien aktiivisena pitäminen helpottuu.

Opinnäytetyöni lopputuloksena Wikimedia Suomi ry sai Ajapaik Flutter Appin testivalmiiksi ja dokumentaation sovelluksen kehittämisratkaisuista. Dokumentaatio pitää sisällään myös sovelluksen kehityksessä tarvittavien työkalujen käyttöönoton ja mahdollisten lisälaitteiden käyttöönoton.

13 Yhteenveto

Opinnäytetyön tavoitteena oli ottaa käyttöön Flutter ja opetella ohjelmointia dart-ohjelmointikielellä. Lopullisena tuotoksena oli tarkoitus dokumentoida Ajapaik Flutter Appin mobiilisovelluksen kehitystä ja tuottaa testivalmis sovellus Helsinki Rephotography -projektia varten. Koen, että sain vastauksia asettamiini tutkimuskysymyksiin tutkimani ja tekemäni pohjalta. Vastaukset olivat luonteeltaan johdonmukaisia päättelyjä oman tekemisen ja sovelluksen testivalmiuden varmistuttua. Työ pitää sisällään ohjeet aloittelevalle ohjelmoijalle Flutterin ja dartin käyttöönotosta, ohjelmakoodin syntaksin tarkastelun teoreettisella ja soveltamallani tasolla sekä sovelluksen toimivien osien tarkastelun ja logiikan purkamisen, jotta sovellusta voidaan kehittää opinnäytetyön pohjalta.

Opinnäytetyöni kautta opin käyttöliittymäsuunnittelua ja suunnittelun toteutusta oikeassa työympäristössä. Käyttöliittymä tehtiin uusiksi kolme kertaa ja samalla kokeiltiin eri elementtien toimivuuksia kokonaisuudessa, kunnes sopivimmat löytyivät testivalmista sovellusta varten. Osa ohjelmakoodista käsittelee myös mobiililaitteen ominaisuuksien seuraamista ja tarvittaessa manipuloi ruutua niiden mukaan. Opin myös mobiilisovelluksen ja sen käyttämän palvelimen vuorovaikutuksesta ja haetun datan varastoimisesta ja esittämisestä ruudulla eri tiloissa.

Osana jatkokehitystä suosittelen laajempaa testaamista iOS-alustalla, käyttöliittymän lisäoptimointia ja historiallisten kuvien laajempaa käyttöönottoa muiden palvelujen kautta. Koska sovelluksen pääkehitysalusta oli Android ja sovellusta kehittäneillä oli apunaan Android-laitteet niin sovellusta on testattu erittäin vähäisesti iOS-alustalla. Sen takia uskon, että sovellus poikkeaa iOS-laitteella ulkonäöllisesti tarkoitetusta lopputuloksesta. Tällä hetkellä käyttöliittymä toimii hyvin ja näyttää hyvältä, mutta etusivu kaipaa elementtien lisäoptimointia. Tarkoitus oli, että kartta pienenee, jos kuvia selaa ja kuvat pienenevät, jos kartan haluaa isommaksi skrollaamalla ylöspäin. Tätä toiminnollisuutta ei ole vielä implementoitu. Kuvat haetaan sovellukseen Wikimedia Commonsin palvelimilta. Tarkoitus oli myös liittää sovellukseen kuvia finna.fi-palvelun rajapinta, jolloin finnasta saatavat kuva-albumit olisi ollut myös mahdollista liittää sovellukseen tarkasteltaviksi. Tätä toiminnollisuutta ei ole myöskään vielä implementoitu.

Lähteet

Claburn, T. (2011). *Google Launches Dart Programming Language.* Viitattu 10.3.2022 https://www.informationweek.com/mobile/google-launches-dart-programming-language

Dart (n.d.). *dart2js: Dart-to-Javascript compiler*. Viitattu 8.3.2022 https://dart.dev/tools/dart2js

Flutter (n.d.a.). *StatelessWidget class*. Viitattu 21.4.2022 https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

Flutter (n.d.b.). *StatefulWidget class*. Viitattu 21.4.2022 https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

Flutter (n.d.c.). *Arch Overview | Mobile*. Viitattu 7.4.2022 https://docs.flutter.dev/assets/images/docs/arch-overview/web-arch.png

Flutter (n.d.d.). *Introduction to widgets*. Viitattu 22.4.2022 https://docs.flutter.dev/development/ui/widgets-intro

Flutter (n.d.e.). *Introduction to widgets | Using Material Components*. Viitattu 22.4.2022 https://docs.flutter.dev/development/ui/widgets-intro

Flutter (n.d.f.). *Arch Overview | Web*. Viitattu 7.4.2022 https://docs.flutter.dev/assets/images/docs/arch-overview/web-arch.png

Kuva 8. *Nykyaikainen käyttöliittymäsuunnittelumalli suosituilla sovelluksilla*. Ville Mielikäinen, 2022.

Cooper, M., Kirkpatrick, A., & O'Connor, J. (2016). *Understanding WCAG 2.0*. Viitattu 27.3.2022 https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html

Flutter (n.d.g.). *Accessibility | Screen readers*. Viitattu 27.3.2022 https://docs.flutter.dev/development/accessibility-and-localization/accessibility#screen-readers

Ishida, R., & Miller, S. K. (2005a). *Localization vs. Internationalization*. Viitattu 8.4.2022 https://www.w3.org/International/questions/qa-i18n

Flutter (n.d.h.). *Internationalizing Flutter apps*. Viitattu 8.4.2022 https://docs.flutter.dev/development/accessibility-and-localization/internationalization

Ishida, R., & Miller, S. K. (2005b). *Localization vs Internationalization*. Viitattu 8.4.2022 https://www.w3.org/International/questions/qa-i18n

Flutter (n.d.i.a.). *Flutter and the pubspec file*. Viitattu 10.3.2022 https://docs.flutter.dev/development/tools/pubspec

Flutter (n.d.i.b.). *Liite 2 | Esimerkki, miltä pubspec.yaml-tiedosto näyttää*. Viitattu 10.3.2022 https://docs.flutter.dev/development/tools/pubspec

Google Maps Platform (n.d.). *Pricing*. Viitattu 8.3.2022 https://mapsplatform.google.com/pricing/

Agafonkin, V. (2010). *Leafletjs.com | Overview*. Viitattu 8.3.2022 https://leafletjs.com/index.html

Flutter (n.d.j.). *Accessibility release checklist*. Viitattu 8.4.2022 https://docs.flutter.dev/development/accessibility-and-localization/accessibility#accessibility-release-checklist

Stack Overflow. 2019. *Total distance calculation from LatLng List*. Viitattu 24.3.2022 https://stackoverflow.com/questions/54138750/total-distance-calculation-from-latlng-list

Flutter (n.d.k.). *Internationalizing Flutter apps*. Viitattu 7.4.2022 https://docs.flutter.dev/development/accessibility-and-localization/internationalization

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyössäni en ole käyttänyt aineistoa, johon tarvitsisi lupaa tai tehdä sopimusta aineiston omistavan tahon kanssa. Kaikki opinnäytetyössäni käytettävät lähteet ja viittaukset ovat Flutterin omasta dokumentaatiosta tai erinäisistä artikkeleista ja ne ovat asianmukaisesti merkitty opinnäytetyön lähdeluetteloon. Soveltavassa osiossa luodut ja todeksi näytetyt ohjelmakoodit ovat opinnäytetyön tekijän luomaa tai muokkaamaa koodia.

Kehitysprojekti:

Opinnäytetyö ja siinä käytetyt työharjoittelussa otetut muistiinpanot ovat tallennettuna omalle tietokoneelle, henkilökohtaiselle Google Drivelle ja oppilaitokseni tarjoamalle OneDrivelle. Opinnäytetyössä luotu lopputulos on avointa lähdekoodia ja näin ollen tarkasteltavissa Ajapaikin GitHubissa kohdassa Ajapaik Flutter App. Linkki on lisätty

Liite 2: Ohjelmakoodi

```
name: <project name>
description: A new Flutter project.

publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:        # Required for every Flutter project
    sdk: flutter # Required for every Flutter project

  cupertino_icons: ^1.0.2 # Only required if you use Cupertino (iOS style) icons

dev_dependencies:
  flutter_test:
    sdk: flutter # Required for a Flutter project that includes tests

flutter:

  uses-material-design: true # Required if you use the Material icon font

  assets:  # Lists assets, such as image files
    - images/a_dot_burr.jpeg
    - images/a_dot_ham.jpeg

  fonts:              # Required if your app uses custom fonts
    - family: Schyler
      fonts:
        - asset: fonts/Schyler-Regular.ttf
        - asset: fonts/Schyler-Italic.ttf
          style: italic
    - family: Trajan Pro
      fonts:
        - asset: fonts/TrajanPro.ttf
        - asset: fonts/TrajanPro_Bold.ttf
          weight: 700
```

Liite 3: Ohjelmakoodi

```
return Scaffold(
        body: IndexedStack(
          index: _selectedIndex,
          children: screens,
        ),  // IndexedStack

    bottomNavigationBar: BottomNavigationBar(
      type: BottomNavigationBarType.fixed,
      currentIndex: _selectedIndex,
      onTap: _onItemTapped,
      items: <BottomNavigationBarItem> [
        BottomNavigationBarItem(
          icon: const Icon(Icons.home_rounded),
          label: (AppLocalizations.of(context)!.translate('homePage-NavItem1')
          )),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: const Icon(Icons.favorite_border),
          label: (AppLocalizations.of(context)!.translate('homePage-NavItem2')
          )),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: const Icon(Icons.camera),
          label: (AppLocalizations.of(context)!.translate('homePage-NavItem3')
          )),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: const Icon(Icons.explore),
          label: (AppLocalizations.of(context)!.translate('homePage-NavItem4')
          )),  // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon((loggedIn ? Icons.person : Icons.login)),
          label: (loggedIn
              ? (AppLocalizations.of(context)!
                  .translate('homePage-NavItemó'))
              : (AppLocalizations.of(context)!
                  .translate('homePage-NavItem5'))),
        )  // BottomNavigationBarItem
      ],  // <BottomNavigationBarItem>[]
    ),  // BottomNavigationBar
);  // Scaffold
```

Liite 4: Ohjelmakoodi

```
getMarkerList(context) {
  List list = widget.albums!.first.features;
  markerList.clear();
  for (int x = 0; x < list.length; x++) {
    if (list[x].geometry.coordinates.length > 0) {
      double latitude = list[x].geometry.coordinates[0];
      double longitude = list[x].geometry.coordinates[1];
      var m = Marker(
          width: 40.0,
          height: 40.0,
          point: LatLng(latitude, longitude),
          builder: (ctx) => IconButton(
              icon: Icon(Icons.location_pin, color: _colors[x]),
              onPressed: () {
                if (busy == true) {
                  return;
                } else {
                  if (open == true) {
                    Navigator.of(context).pop();
                    setState(() {
                      _colors[x] = Colors.red;
                    });
                    open = false;
                  } else {
                    busy = true;
                    open = true;
                    setState(() {
                      _colors[x] = Colors.white;
                    });
```

```
showBottomSheet(
        context: context,
        builder: (builder) {
          busy = false;
          return Container(
            height: 325,
            child: Column(children: [
              const SizedBox(height: 20),
              Row(children: [
                Expanded(
                  child: GestureDetector(
                    child: Image.network(
                        list[x].properties.thumbnail),  // Image.network
                    onTap: () {
                      Navigator.push(
                          context,
                          MaterialPageRoute(
                              builder: (context) =>
                                  RephotoScreen(
                                    historicalPhotoId: list[x].properties.id.toString(),
                                    historicalPhotoUri: list[x].properties.thumbnail.toString(),
                                    historicalName: list[x].properties.name.toString(),
                                    historicalDate: list[x].properties.date.toString(),
                                    historicalAuthor: list[x].properties.author.toString(),
                                    historicalSurl: list[x].properties.sourceUrl.toString(),
                                    historicalLabel: list[x].properties.sourceLabel.toString(),
                                    historicalCoordinates: list[x].geometry,
                                  )));  // RephotoScreen, MaterialPageRoute
                    }),  // GestureDetector
                )  // Expanded
              ])  // Row
            ]));  // Column, Container
```

```
                    }).closed.then((value) {
                  if (busy == false) {
                    open = false;
                  }
                  setState(() {
                    _colors[x] = Colors.red;
                  });
                  busy = false;
                });
              }
            }
          },
        ));  // IconButton, Marker
    markerList.add(m);
  }
 }
 return markerList;
}
```

Linkki

GitHub-linkki projektiin

https://github.com/Ajapaik/ajapaik_flutter_app