

# FLUTTER - MONIALUSTAINEN MOBIILIKEHITYSTYÖKALU

Hiltunen Sampsa-Kristian

Opinnäytetyö  
Arktiset luonnonvarat ja talous  
Tieto- ja viestintätekniikka  
Insinööri (AMK)

2019

Arktiset luonnonvarat ja talous  
Tieto- ja viestintätekniikka  
Insinööri (AMK)

---

<b>Tekijä</b>	Sampsa-Kristian Hiltunen	<b>Vuosi</b>	2019
<b>Ohjaaja(t)</b>	Erkki Mattila		
<b>Toimeksiantaja</b>	pLAB		
<b>Työn nimi</b>	Flutter – monialustainen mobiilikehitystyökalu		
<b>Sivu- ja liitesivumäärä</b>	32		

---

Tämän opinnäytetyön aiheena oli monialustainen mobiilikehitystyökalu Flutter. Flutter on Googlen luoma ohjelmointityökalu, joka käyttää Dart -nimistä ohjelmointikieltä. Työn tarkoituksena oli selvittää Lapin Ammattikorkeakoulun ohjelmistolaboratorion (pLAB) ohjelmistokehittäjille, ovatko tämän ohjelmistotyökalun ominaisuudet käyttökelpoisia sekä tarpeeksi monipuolisia nykyaikaisessa mobiilikehityksessä.

Työssä testatut ominaisuudet olivat pLABin ohjelmistokehittäjien määrittelemät, mikä tarkensi työn kokonaisuutta. Työn soveltava osuus sisälsi monta yksittäistä mobiilisovellusta, joista jokainen sisälsi yhden tai useamman testattavan ominaisuuden. Tämä menetelmä tuntui parhaalta lähestymistavalta, sillä virheiden etsiminen ja korjaaminen olivat nopeampaa, kun mobiilisovellus on askeettinen sekä selkeä.

Kehitetyt mobiilisovellukset testattiin kahdella yleisimmistä käyttöjärjestelmistä, mitkä ovat Android sekä iOS. Kaikki mobiilisovellukset kehitettiin käyttäen Windows 10 käyttöjärjestelmää sekä Android Studiota. Tämän jälkeen kaikki mobiilisovellukset testattiin molemmilla laiteympäristöillä.

Testaus tapahtui käyttämällä älypuhelimia sekä tablettia emulaattorin sijaan, jotta testitulokset olivat selkeämpiä suorituskyvyn sekä ominaisuuksien toimivuuden kannalta. Testaustuloksien lisäksi tuotettiin erillinen dokumentaatio Flutterin ominaisuuksien käyttöönotosta pLABin ohjelmistokehittäjille, mikä edesauttaa sekä selkeyttää tämän ohjelmistotyökalun käyttöä.

Tässä raportissa kerrotaan eri kehitysympäristöistä sekä Flutterin eroavaisuuksista monialustaisiin kehitysympäristöihin. Pohdinnassa käydään läpi projektin sekä testauksen aikana ilmi tulleita ongelmia sekä kerrotaan avoin mielipide Flutterista.

**Avainsanat**                      Anrdoid, Dart, Flutter, iOS, monialustainen, natiivi, ohjelmistotyökalu, mobiilikehitys

Degree Programme in Information  
and Communication Technology  
Bachelor of Engineering

---

<b>Author</b>	Sampsa-Kristian Hiltunen	<b>Year</b>	2019
<b>Supervisor</b>	Erkki Mattila		
<b>Commissioned by</b>	pLAB		
<b>Subject of thesis</b>	Flutter – Crossplatform Framework		
<b>Number of pages</b>	32		

---

The subject of this thesis was mobile development via cross-platform framework, called Flutter. The objective was to clarify if Flutter is a viable and versatile enough in a contemporary cross-platform development. Flutter is created by Google and uses object-oriented programming language called Dart.

In this thesis the features analyzed were defined by pLAB's software developers. The practical part of the thesis consisted of a various number of developed applications that included one or more of these features. All of the applications developed, were tested on both Android and iOS platforms. In the development a Windows 10 operating system and Android Studio were used to create the applications. After the development process all of the applications were tested both on an Android and an iOS platform using actual devices, like phones and tablets, instead of emulators. This method seemed to be the easiest approach, because finding errors and testing them was much faster when the application was simplified and ascetic.

In addition, there is a documentation which included all of the test results and how to implement features in Flutter. This clarifies and helps the developers of pLAB to bring Flutter in to use. This thesis also relates to different frameworks and compare Flutter to other cross-platform frameworks. The conclusion of this thesis goes through the problems that appeared during this project and also gives the developers a point of view on Flutter's viability in a modern-day cross-platform development.

Key words

Android, cross-platform, Dart, Flutter, framework, iOS

## SISÄLLYS

1 JOHDANTO .....	6
2 KEHITYSYMPÄRISTÖT .....	7
2.1 Natiivit kehitysympäristöt .....	7
2.2 Monialustaiset kehitysympäristöt .....	9
2.2.1 Webview.....	10
2.2.2 React Native.....	10
2.3 Flutterin vertailu monialustaisiin kehitysympäristöihin.....	11
3 FLUTTER.....	14
3.1 Widgetit.....	15
3.2 Widgettien koostumus.....	15
3.2.1 Stateless widget .....	16
3.2.2 Stateful widget.....	16
3.3 Ohjelmointikieli Dart.....	17
4 TESTATUT OMINAISUUDET .....	18
4.1 Sijaintipalvelut.....	19
4.2 Tietoliikenne.....	20
4.3 Datarakenteet .....	22
4.4 Käyttöliittymä.....	23
4.5 Testaaminen .....	27
4.6 Suorituskyky.....	27
5 POHDINTA .....	29
LÄHTEET .....	31

## KÄYTETYT T JA LYHENTEET

API	ohjelmointirajapinta, joka mahdollistaa pääsyn toisen osapuolen sovellukseen tai alustaan
Eleet	gestures eli eleet, ovat käyttäjän suorittamia toimia puhelimen näytöllä, kuten pyyhkäisy tai painallus
Fps	frames per second eli sekunnissa näytölle piirtyvien kuvien määrä
Geofencing	GPS- tai RFID-tekniikan käyttö virtuaalisen maantieteellisen rajan luomiseksi
Json	JavaScript Object Notation on avoimen standardin mukainen tiedostomuoto tiedonvälitykseen
Kehitysympäristö	IDE eli ohjelmistopaketti, joka yhdistää kehittämiseen ja testaamiseen tarvittavat perustyökalut
Ohjelmistokehys	framework, ohjelmointityötä vähentävä ohjelmisto, jonka avulla luodaan alustava järjestelmä, jonka päälle sovellus luodaan
VS Code	Visual Studio Code on lähdekoodi editori

## 1 JOHDANTO

Teknologian ja elektroniikan kehittyessä älylaitteiden käyttö on kasvanut räjähdysmäisesti. Yhä useampi henkilö omistaa jonkinlaisen älylaitteen, kuten älypuhelimen, -kellon tai tabletin. Älylaitteiden suosion ja käytön kasvaessa myös erilaisten sovellusten määrä ja kilpailu kasvavat. Vuonna 2018 mobiilisovellus latauksia oli yli 200 biljoonaa ja luvun uskotaan kasvavan noin 260 biljoonaan vuoteen 2020 mennessä. (Statista 2018.)

Olipa kyseessä yritysten tablet-sovellusohjelma, suosittu mobiilipeli tai älykellon sovellus, se on mobiilikehitystä. Ensimmäisiä mobiilisovelluksia olivat laskin, kalenteri sekä java-pelit. Mobiilisovellusten kehittäminen on jokseenkin samanlainen, kuin web-sovellusten kehittäminen. Suurin ero on kuitenkin se, että mobiilisovellukset on usein kirjoitettu hyödyntämään mobiililaitteen ominaisuuksia, kuten kosketusnäyttöä tai kiihtyvyysmittaria. Tässä työssä kuitenkin keskitytään ainoastaan mobiilisovelluksien kehitykseen sekä niiden kehitysympäristöihin. (Simple programmer 2016.)

Flutter on 2017 keväällä julkaistu Googlen luoma ohjelmistokehitystyökalu, joka mahdollistaa nopean monialustaisen mobiilisovelluksien kehittämisen. Tämä ohjelmistokehitystyökalu on tuore, joka oli opinnäytetyötä aloittaessa vielä beta-vaiheessa, mutta oikea julkaisu, versio 1.0 tapahtui 4.12.2018. (Google Developers 2018.)

Työssä keskityttiin Flutterin ominaisuuksien toiminnallisuuteen sekä suorituskykyyn kehittämällä yksittäisiä ja täsmällisiä mobiilisovelluksia, jotka sisälsivät yhden tai useamman ominaisuuden. Nämä ominaisuudet olivat pLABin ohjelmistokehittäjien määrittelemät, jotka tarkensivat sekä selkeyttivät lopullisen työn kuvaa. Testituloksista sekä ominaisuuksien lisäämisestä tehtiin erillinen dokumentaatio pLABin ohjelmistokehittäjille.

Testaus suoritettiin kahdella eri mobiilikäyttöjärjestelmällä, Androidilla sekä iOS:lla, jotka ovat yleisimpiä mobiilikäyttöjärjestelmiä. Lisäksi Flutteria verrataan natiiveihin sekä muihin monialustaisiin ohjelmistotyökaluihin käytännön tasolla sekä sivutaan Flutterin käyttämää ohjelmointikieltä nimeltä Dart. Opinnäytetyö on tehty toimeksiantona pLABille sekä sen ohjelmistokehittäjille.

## 2 KEHITYSYMPÄRISTÖT

Kehitysympäristö on ohjelmistopaketti, joka yhdistää ohjelmistojen kirjoittamiseen ja testaamiseen tarvittavat perustyökalut. Kehittäjät käyttävät lukuisia työkaluja ohjelmistokoodin luomisessa, rakentamisessa ja testauksessa. Kehitysympäristöt sisältävät usein tekstieditorin, koodikirjaston, kääntäjän sekä testialustan. Tämä siis tuo yhteen monia kehitykseen liittyviä työkaluja yhdessä ja samassa paketissa. (TechTarget 2018.)

Nämä ohjelmistopaketit on suunniteltu yksinkertaistamaan ohjelmistokehitystä sekä tunnistamaan ja minimoimaan koodaus- sekä kirjoitusvirheet. Ilman erillistä kehitysympäristöä ohjelmoijan tulisi tehdä nämä kaikki erinäisillä työkaluilla. Kehitysympäristöt voivat siis parantaa ohjelmistokehittäjien tuottavuutta nopean asennuksen, versionhallinnan ja standardoinnin ansiosta. (TechTarget 2018.)

Kehitysympäristö sisältää tyypillisesti koodieditorin, kääntäjän tai tulkin sekä debuggerin, joita käytetään yhden graafisen käyttöliittymän kautta. Käyttäjä kirjoittaa ja muokkaa lähdekoodia koodieditorissa. Kääntäjä kääntää lähdekoodin luettavaksi kieleksi, joka on suoritettavissa esimerkiksi puhelimella. Debugger testaa ohjelmiston ratkaisemaan mahdolliset ongelmat tai virheet. (TechTarget 2018.)

Osa kehitysympäristöistä on avoimen lähdekoodin ohjelmistopaketteja, kun taas toiset ovat kaupallisia. Ne voivat olla itsenäisiä sovelluksia tai ne voivat kuulua osaksi laajempaa pakettia. Kehitysympäristön tulee vastata kehitettävän sovelluksen kanssa, jota halutaan tuottaa. Jos esimerkiksi kehittäjä haluaa luoda sovelluksen iOS:lle, tarvitaan kehitysympäristö, joka tukee Apple Swift –ohjelmointikieliä. (TechTarget 2018.)

### 2.1 Natiivit kehitysympäristöt

Sekä Apple että Google tarjoavat sovelluskehittäjille oman kehitystyökalunsa, käyttöliittymän ja ohjelmistokehityspaketin. Natiivit mobiilisovellukset luodaan käyttäen alustalle ominaisia ohjelmointikieliä. Esimerkiksi jotkut kielet, joita voidaan käyttää Android-sovellusten kehittämiseen ovat Java, Kotlin ja Python. iOS-käyttöjärjestelmässä kielet ovat Swift ja Objective-C. Natiivisovellukset luodaan

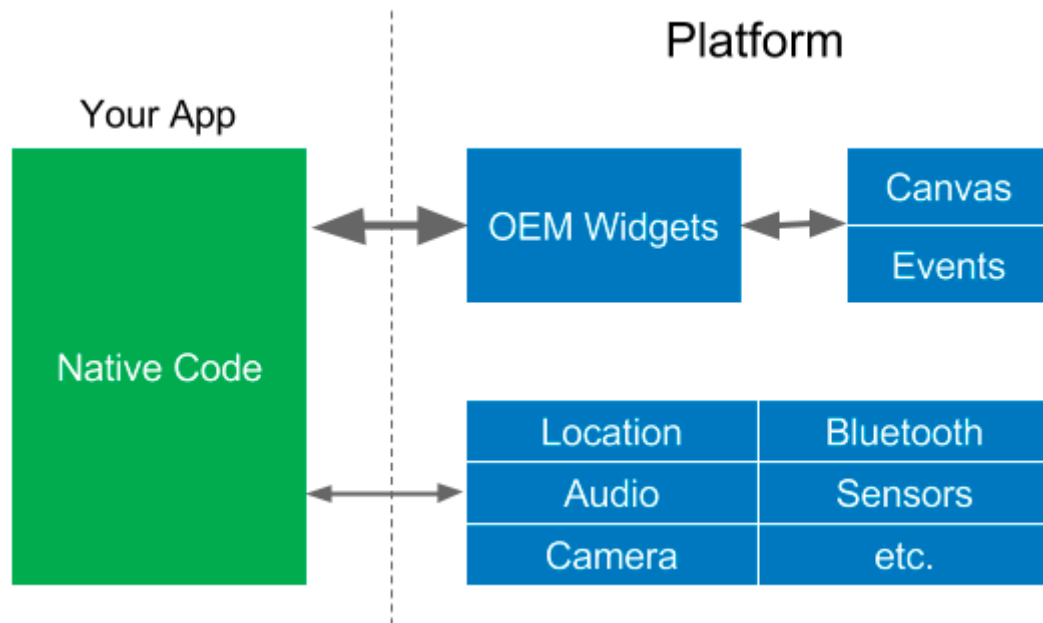
erityisesti käytettäväksi kohdealustalle, koska tämä tukee kaikkia alkuperäisiä teknologioita sekä kaikkia laitteen ominaisuuksia. (Codeburst 2018a.)

Natiivisovellukset tarjoavat parhaimman tuen ottaessaan käyttöön uusia ominaisuuksia. Sovellukset on rakennettu käyttämällä itse laitteiden toimittajien, Applen tai Googlen -ohjaimia sekä noudattavat usein näiden asettamia suunnittelutapoja. Useimmiten natiivisovellukset toimivat paremmin kuin monialustaisella rakennetut sovellukset, vaikka ero voi monissa tapauksissa olla hyvin pieni, riippuen monialustaisen taustalla olevasta teknologiasta. Suurin etu natiivisovelluksissa on, että ne voivat ottaa käyttöön uudet teknologiat, joita laitteiden toimittajat luovat beta-vaiheessa ilman kolmannen osapuolen integraatiota. Natiivisovellusten suurin epäkäytännöllisyys on koodien uudelleenkäytön puute eri alustoilla, minkä takia natiivisovellus kehitys voi käydä kalliiksi. (Codeburst 2018a.)

Koska natiivisovellusten kehittämisessä käytetyt tekniikat ovat alustaspesifisiä, natiivi-koodilla on suora pääsy laitteen käyttöjärjestelmään ja toimintoihin. Tätä kautta saavutetaan helpompi vuorovaikutus mobiililaitteiden alkuperäisten ominaisuuksien kanssa, mikä parantaa sovelluksen yleistä suorituskykyä, erityisesti grafiikka- tai multim mediasisällön esittämisessä. Tämän vuoksi monimutkaisten sovellusten rakentaminen natiivia koodia käyttäen on kannattavaa, sillä se vähentää sovellusten kaatumisen tai jäätymisen riskiä. (Codeburst 2018b.)

Sovellus kommunikoi alustan kanssa kuvion 1 mukaisesti luodakseen käyttöliittymän näkymän tai käyttäessä palveluita esim kameraa tai sensoreita. Tämä on yksinkertainen arkkitehtuuri, mutta sovellusten luominen natiivina useille käyttöjärjestelmille voi merkittävästi pidentää kehitysprosessia, sillä samaa koodia ei voi käyttää eri alustoilla. Natiivisovellukset integroituvat virheettömästi mobiilikäyttöjärjestelmään, käyttäjät voivat navigoida tutulla käyttöliittymällä ilman ylimääräisiä ongelmia, mikä johtaa positiivisiin käyttäjäkokemuksiin. (Hackernoon 2017a.)





Kuvio 1. Natiivisovelluksen kommunikaatio laitteen kanssa (Hackernoon 2017a.)

## 2.2 Monialustaiset kehitysympäristöt

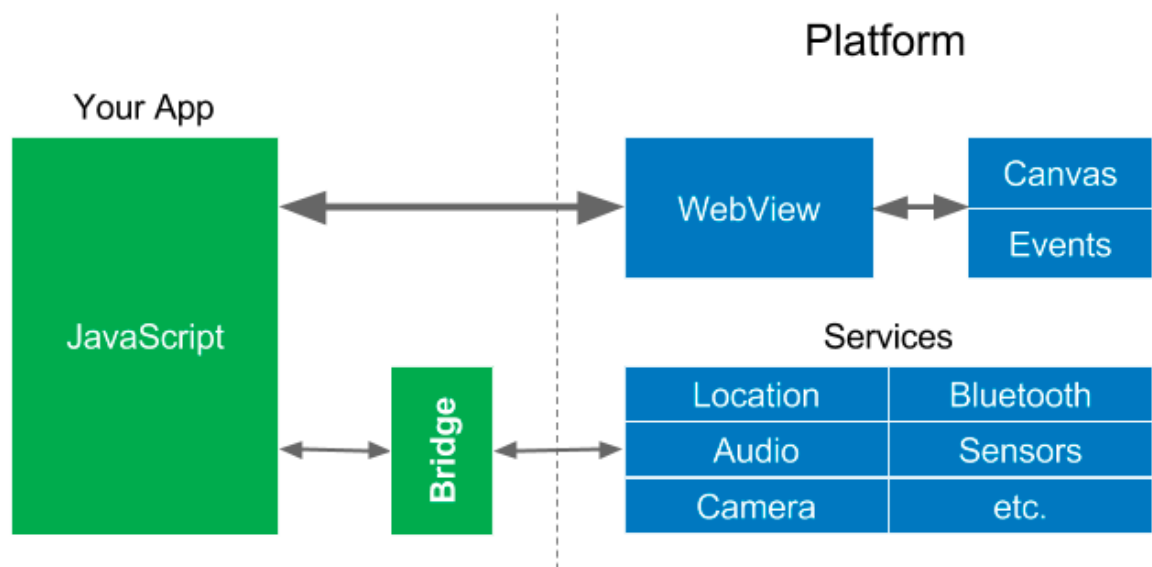
Monialustaiset kehitysympäristöt yleistyivät ensin Linux- sekä Mac OS-käyttöjärjestelmille. Älypuhelimien nousun myötä myös monialustaiset mobiilikkehitystyökalut yleistyivät, mitkä toivat yhteensopivuutta alustojen sekä laitteiden ympärille. (Techopedia 2019.)

Monialustaisilla kehitysympäristöillä voi luoda ohjelmiston tai ohjelman, joka toimii useilla eri alustoilla. Valmis monialustainen ohjelmisto voidaan ottaa käyttöön eri laitteissa ja alustoissa, huolimatta yhteensopimattomuusongelmista. Vaikka alustojen toiminnassa on eroja, käyttäjällä on samanlainen kokemus sovelluksesta kaikkien laitteiden kanssa. (Techopedia 2019.)

Monialustainen mobiilisovellusten kehittäminen on nykyään erittäin ajankohtainen aihe, jopa yritykset ottavat tämän huomioon suunniteltaessa uutta sovellusta tai päivitettäessä vanhempaa. Nykyaikana monialustaisia kehitysympäristöjä on runsaasti, joista jokaisella on hyvät ja huonot puolensa, mutta teollisuus, tiedot sekä kysyntä antavat suuntaa vaikutusvaltaisimmista kehitysympäristöistä. "Ohjelmoi kerran, käytä missä vain" -lähestymistavalla kehittäjät voivat käyttää yhtä koodia useilla alustoilla, mikä vähentää kustannuksia ja lyhentää kehitysaikaa verrattuna natiivisovelluksiin. (Bobology 2019.)

### 2.2.1 Webview

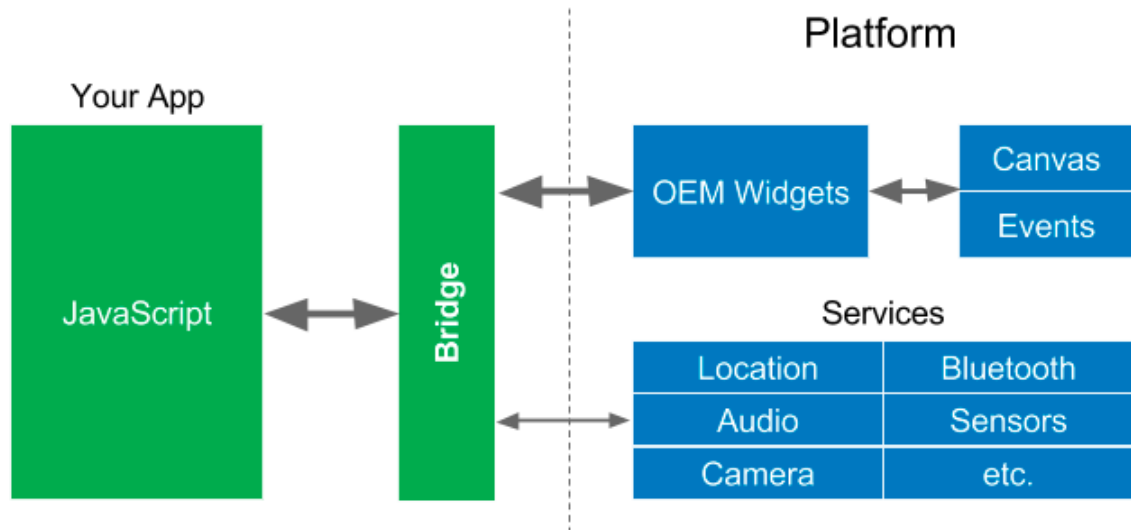
Ensimmäiset monialustaiset kehitysympäristöt perustuivat JavaScriptiin sekä WebViewiin. Sovellus luo HTML-koodin ja näyttää sen WebView -ohjelmana alustalla. JavaScriptin kaltaisilla ohjelmointikielillä on vaikea kommunikoida suoraan alustojen palveluiden kanssa, joten se kommunikoi siltauksen kautta, joka tekee kontekstikytkennät JavaScriptin ja natiivin välillä kuvion 2 mukaisesti. Koska alustapalveluja ei yleensä kutsuta usein, tämä ei aiheuta liikaa suorituskykyongelmia. (Hackernoon 2017b.)



Kuvio 2. WebView mobiilisovelluksen kommunikaatio alustan kanssa (Hackernoon 2017b.)

### 2.2.2 React Native

Vuonna 2015 React Native luotiin tuomaan reaktiivisen -tyylin näkymien monia etuja mobiilisovelluksiin. React Native on erittäin suosittu, mutta koska se käyttää JavaScriptiä alustan palveluiden lisäksi alustan elementeille, jotka ohjaavat sovelluksen näkymää ja käyttöliittymää, se täytyy siltauttaa myös näille kuvion 3 mukaisesti. Näkymää käsitellään useasti, jopa 60 kertaa sekunnissa animaatioiden, siirtymien aikana tai kun käyttäjä käyttää eleitä, kuten pyyhkäisyä. Tämä voi aiheuttaa suorituskykyongelmia, jos siltauksen kautta kulkee liikaa informaatiota. (Hackernoon2017c.)



Kuvio 3. React Native mobiilisovelluksen kommunikaatio alustan kanssa (Hackernoon 2017c.)

### 2.3 Flutterin vertailu monialustaisiin kehitysympäristöihin

Flutter on erilainen kuin useimmat muut vastaavat kehitystyökalut, sillä se ei käytä WebView-ohjelmaa, tai laitteen mukana tulevaa OEM-kääntäjää. Tämän sijasta Flutter käyttää sen omaa kehystä sekä renderintimoottoria piirtääkseen mobiilisovelluksen käyttöliittymän kuvion 5 mukaisesti. Esimerkiksi Androidissa on hyvin standardoituja käyttäytymismalleja eleille, kuten vaaka pyyhkäisy, joka vaihtaa näkymän välilehteä. Flutterissa voit kirjoittaa oman eleiden tunnistuksen. (Hackernoon2017d.)

Ohjelmointikieli Dart sisältää erittäin pienen syntaksin sekä on saavuttanut erittäin hyviä tuloksia eri vertailuarvoissa. Kuviossa 4 näkyy vertailu, missä suoritettiin samat tehtävät Dartilla, Nettyllä sekä Node.js:llä. Tehtävät sisälsivät JSON-tietojen hakemisen, niiden tallentamisen paikkalliseen muuttujaan sekä vastauksen lähettämisen. Vertailussa huomataan, että Dart suoritti määritetyt tehtävät nopeammin kuin Node.js. Vertailu on tehty käyttäen Apache -vertailutyökalua.

*Node.js:*

Concurrent requests	Average response time (ms)	Requests/second
10	3.323	3008.93
20	6.237	3206.63
40	12.065	3315.41
100	30.396	3289.87
200	61.997	3225.96
400	140.396	2849.08

*Dart:*

Concurrent requests	Average response time (ms)	Requests/second
10	1.651	6058.05
20	3.843	5204.23
40	7.692	5200.16
100	18.868	5299.96
200	39.045	5122.31
400	79.258	5046.82

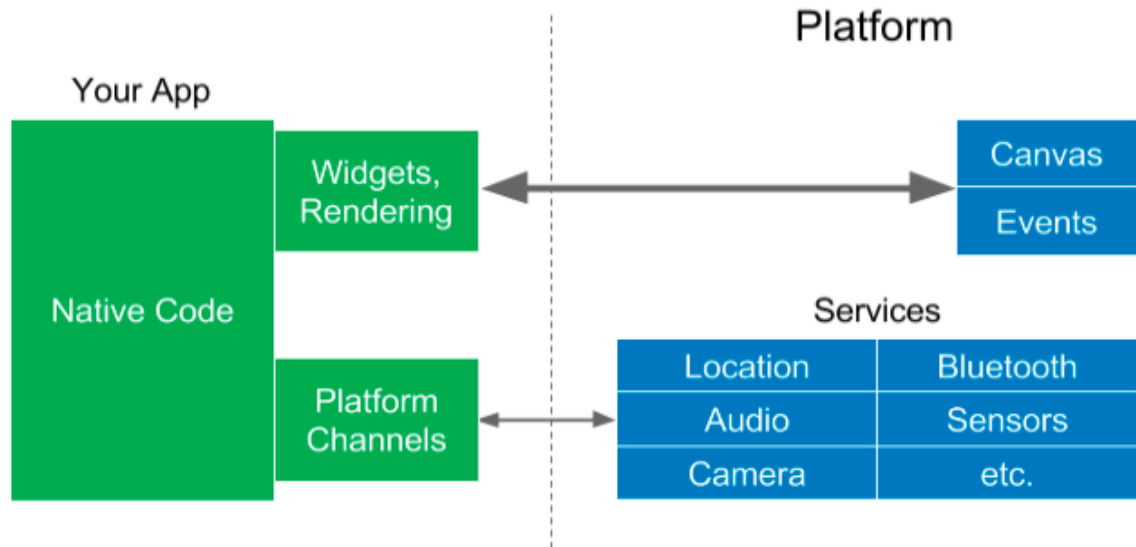
*Netty:*

Concurrent requests	Average response time (ms)	Requests/second
10	1.150	8697.03
20	2.001	9994.15
40	3.665	10914.36
100	8.770	11402.91
200	17.354	11524.76
400	35.218	11357.93

Kuvio 4. Apache –vertailutyökalun tulokset (Mrekuccis blog 2014.)

Flutter-sovellukset eivät käytä erillistä ohjelmointikielen kääntäjää, vaan se hyödyntää omaa AOT (Ahead of time) -kääntäjää, jonka avulla Flutter koodi käännetään alustakohtaiseksi koodiksi. Tämä edesauttaa sovelluksia käynnistymään ja toimimaan nopeammin. Flutter on rakennettu käyttäen C / C++, Dartia sekä Skiaa. AOT –kääntäjän lisäksi Flutter toteuttaa suurimman osan järjestelmästä

Dartissa, minkä johdosta Flutter-mobiilisovellukset ovat suorituskykyisiä. (Flutter Docs 2019a.)



Kuvio 5. Flutter-mobiilisovelluksen kommunikaatio alustan kanssa (Hackernoon 2017d.)

### 3 FLUTTER

Flutter on Googlen luoma monialustainen ohjelmistokehitystyökalu, jolla on mahdollista luoda laadukkaita mobiilisovelluksia nopeasti iOS- sekä Android-mobiilikäyttöjärjestelmille. Flutter tukee kehittämistä Linux-, Mac- sekä Windows-käyttöliittymillä. Tämä ohjelmistokehitystyökalu perustuu avoimeen lähdekoodiin, lisäksi se on täysin ilmainen sekä käyttää Dart ohjelmointikieltä. (Flutter Docs 2019b.)

Flutter sisältää nykyaikaisen ohjelmistokehityksen, 2D-renderointimoottorin, valmiita widgettejä sekä kehitystyökaluja, jota kuvio 6 havainnollistaa. Nämä kaikki komponentit toimivat yhdessä ja ne luovat kokonaisuuden tälle ohjelmistokehitystyökalulle, jolla voi kehittää, testata ja debugata mobiilisovelluksia. Flutterin ohjelmistokehitys on Reactin innoittama, mikä tekee siitä joustavan työkalujen suhteen. (Flutter Docs 2019b.)



Kuvio 6. Flutter arkkitehtuuri (Flutter Docs 2019b.)

Mobiilisovelluksia voi kehittää komentoriviltä minkä tahansa editorin kanssa, lisäksi se toimii eri ohjelmointiympäristöjen kanssa. Tässä työssä ohjelmointiympäristöinä toimivat Android Studio sekä Visual Studio Code. Nämä ohjelmistoympäristöt antavat kattavan tuen Flutterilla kehittämiseen sekä niiden sisäänrakennetut muokkausavustajat ovat monipuolisia. (Flutter Docs 2019c.)

Android Studio tarjoaa eniten ominaisuuksia Flutter kehittämiselle, kuten Flutter Inspectorin, joka analysoi käynnissä olevan mobiilisovelluksen widgettejä sekä mobiilisovelluksen suorituskykyä. Se tarjoaa myös tuen widget hierarkian helpolle kehittämiselle. VS Code puolestaan tarjoaa kevyemmän vaihtoehdon, koska se pyrkii käynnistymään nopeammin kuin muut ohjelmistoympäristöt. (Smashing Magazine 2018.)

Flutter tukee Hot-Reload-toimintoa mobiilisovelluksille, mikä sallii käynnissä olevan sovelluksen muuttamisen useissa tapauksissa, ylläpitämällä tilaa ilman, että sovellusta tulee pysäyttää tai rakentaa uudelleen. Tämä toiminto lisää huomattavasti kehitystehokkuutta. (Flutter Docs 2019d.)

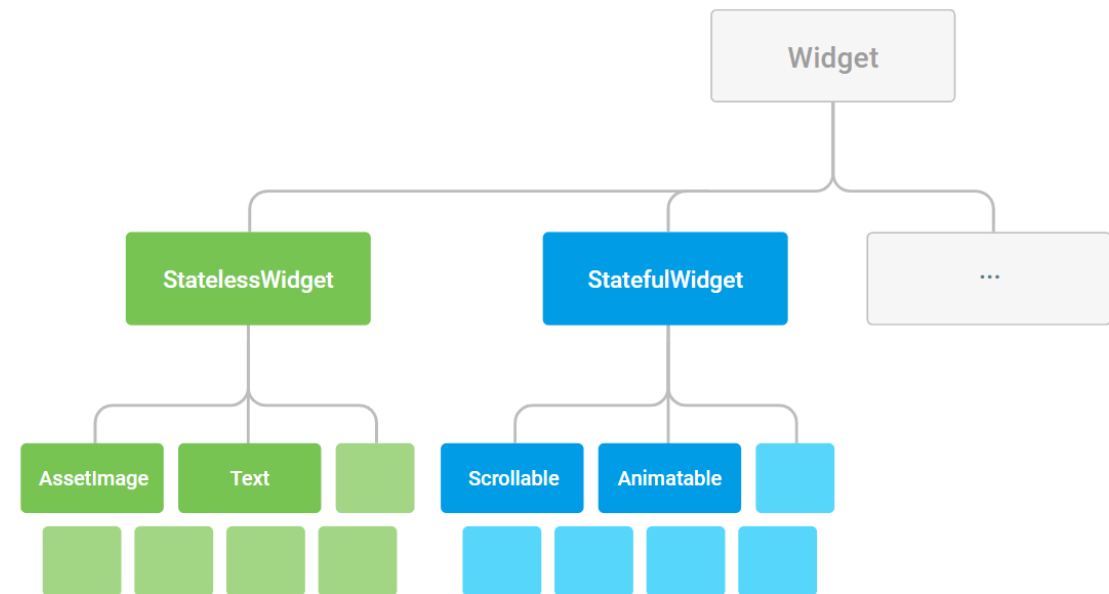
### 3.1 Widgetit

Widgetit ovat Flutter-sovelluksen käyttöliittymän perusrakenne. Jokainen widget on määritelmä käyttöliittymän osasta. Toisin kuin muut ohjelmistokehykset, jotka erottavat näkymät, ohjaimet, ulkoasut sekä muut ominaisuudet, Flutterilla on yhtenäinen oliiomalli, joka on widget.

Widget voi määrittää rakenteelliset elementit, kuten napit ja valikot, tyylitellyn elementin, kuten fontit ja värimaailman, ulkoasun, kuten sommittelun ja niin edelleen. Widgetit muodostavat koostumuksen perustuen hierarkiaan. Jokainen widget sisältää ominaisuuksia sekä perii ominaisuuksia sen parentilta eli yläluokalta. Ei ole erillistä sovellus-oliota. Sen sijaan root widget palvelee tätä roolia. (Flutter Docs 2019e.)

### 3.2 Widgettien koostumus

Widgetit koostuvat usein monista pienistä, yksitoimisista widgeteistä, jotka yhdistyvät tuottamaan tehokkaita efektejä. Kuvio 7 näyttää niiden koostumuksen. Esimerkiksi Container on yleisesti käytetty widget, joka koostuu useista widgeteistä, jotka ovat vastuussa ulkoasusta, paikannuksesta ja koosta. Luokkien hierarkia on matala, mutta laaja, mikä maksimoi mahdollisten yhdistelmien lukumäärän (Flutter Docs 2019e).



Kuvio 7. Widgettien koostumus (Flutter Docs 219e.)

### 3.2.1 Stateless widget

Stateless widgets eli tilattomat widgetit vastaanottaa argumentteja niiden parenttien widgetistä, jota ne säilyttävät lopullisissa jäsenmuuttujissa. Kun widgetiä pyydetään rakentumaan, se käyttää näitä tallennettuja arvoja polveuttaakseen uusia argumentteja widgettiin, jonka se rakentaa.

Tilanteissa, joissa luodaan widgettejä, joiden ei tarvitse hallita minkäänlaista muuttuvaa tilaa, käytetään tilatonta widgettiä. Kyseiset widgetit eivät ole dynaamisia. Ne eivät riipu mistään muusta, kuin niiden sisältämästä tiedoista. Tämä tarkoittaa sitä, että widget määräytyy ainoastaan sen rakenteeseen sisällyvistä argumenteista. Esimerkiksi teksti -widgetin tila ei muutu, sillä sen yliluokka ohjaa sitä. (Flutterdoc 2018a.)

### 3.2.2 Stateful widget

Stateful widgetit ovat dynaamisia. Ne antavat mahdollisuuden luoda widgettejä, jotka voivat dynaamisesti muuttaa sisältöään ajan kuluessa, eivätkä pohjaudu staattisiin tiloihin, joita syötetään niiden alustamisen aikana. Esimerkiksi kun widget toimii dynaamisesti, se kuuntelee muutoksia käyttäjältä ja päivittää tilansa, kun jotain muuttuu.



Kun näin tapahtuu, widget rakentuu uudelleen ja päivittyy muutoksien mukaisesti. Näin ollen se hallitsee omaa tilaansa, eikä ole riippuvainen sen ylläluokasta, joka tekisi niin sen puolesta. (Flutterdoc 2018b.)

### 3.3 Ohjelmointikieli Dart

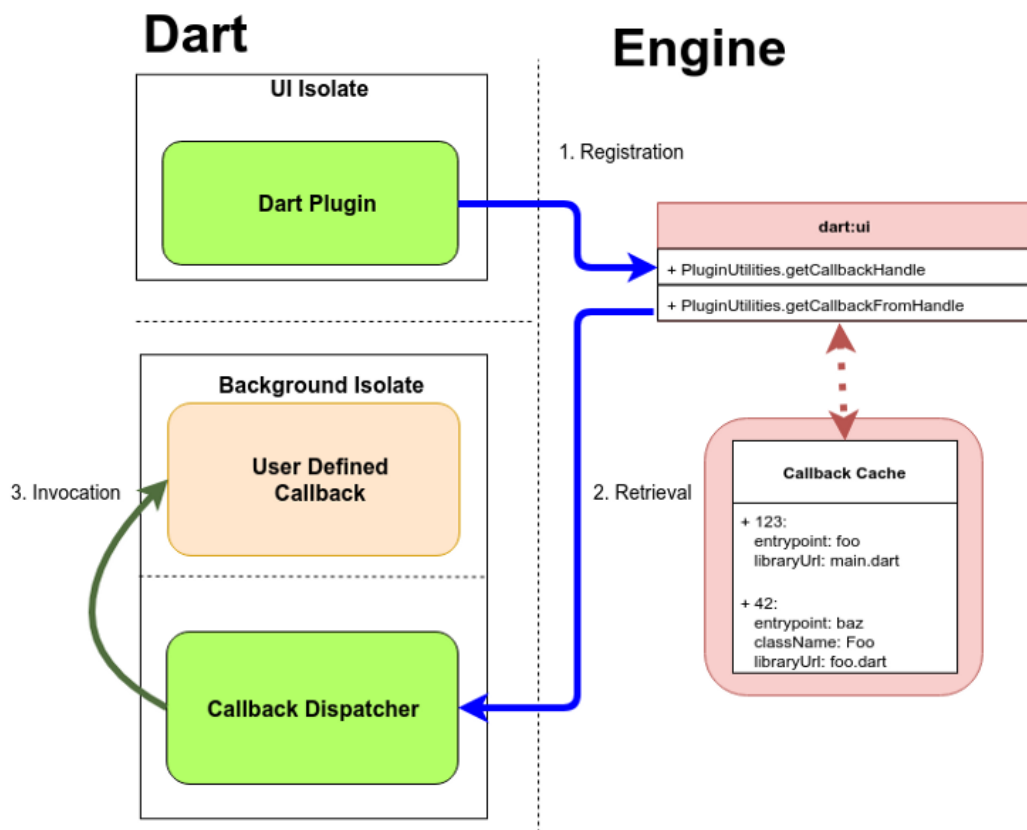
Dart on Googlen kehittämä ohjelmointikieli, joka on tarkoitettu palvelimille, selaimille sekä mobiilikehitykseen. Dart on luokkiin perustuva, yksiperintäinen, puhdas olio-ohjelmointikieli, jossa on C-tyylinen syntaksi, jonka voi valinnaisesti kääntää JavaScriptiin. Dart tukee monia rajapintoja sekä kokoelmia. Dartissa jokaisen olion runtime -tyyppi on edustettuna luokan ilmentymänä, jonka arvot voidaan saada käyttämällä getter -toimintoa. Tätä toimintoa käytetään arvojen hakemiseen tietyn olion ominaisuuksista.

Dart-ohjelmointikieli toimii yhdellä prosessorin ytimellä sekä isolaatissa ja kyseinen koodi voi käyttää luokkia sekä arvoja vain samalta isolaatilta. Eri isolaatit voivat kommunikoida keskenään lähettämällä arvoja porttien kautta, kuten ReceivePort ja SendPort. Isolaatit ajavat koodia niiden omassa tapahtumakierrossaan ja jokainen tapahtuma voi suorittaa pienempiä tehtäviä sisäkkäisessä mikrotyöjonossa. Jos koodi estyy esimerkiksi suorittamalla pitkän aikavälin laskutoimituksen, koko ohjelma kaatuu eli lopettaa toimintoihin vastaamisen sekä sulkeutuu.

Asynkroniset toiminnot antavat ohjelman suorittaa muuta työtä, sillä välin kun se odottaa operaation päättymistä. Dart käyttää futuuri -objekteja (futures), jotka esittävät asynkronisten tulosten toimintoja. Dart-ohjelmat järjestetään modulaarisesti yksiköiksi, joita kutsutaan kirjastoiksi. Kirjastot ovat kapselointiyksiköitä ja ne voivat olla keskenään rekursiivisia, mutta ne eivät kuitenkaan ole ensiluokkaisia. (Dartlang 2018, 9-10.)

#### 4 TESTATUT OMINAISUUDET

Monet hyödylliset ominaisuudet edellyttävät, että mobiilisovelluksella on mahdollisuus käsitellä tapahtumia ilman käyttäjän vuorovaikutusta sekä sovelluksen ollessa käynnissä taustalla. Monialustaisissa mobiilisovelluksissa tämä ei yleensä ole mahdollista, mutta Flutter tukee Dart-ohjelmointikielen suorittamisen taustalla, vaikka sovellus ei olisi aktiivisena päätelaitteen näytöllä. Kuvio 8:ssa on esimerkki, miten Dart-koodi suoriutuu taustalla. (Medium Flutter 2018.)



Kuvio 7. Dart-koodin suorittaminen taustalla (Medium Flutter 2018.)

Googlen antama tuki Flutterille on hyvä, minkä takia sille on monia hyödyllisiä kokoelmia tai laajennuksia, joiden toiminnot ovat valmiiksi toteutettuja. Flutter on kuitenkin uusi, eikä jokaiseen toimintoon löydy vielä laajennusta, mutta se tarjoaa mahdollisuuden kehittää omia laajennuspaketteja, joiden avulla voi kutsua järjestelmäkohtaisia ohjelmarajapintoja, mutta tässä työssä käytin vain valmiita laajennuksia. Tästä syystä pLAB halusi, että testaan heidän määrittämien ominaisuuksien tuen ja toimivuuden. (Flutter Docs 2019f.)

Flutter antaa mahdollisuuden luoda yksinkertaisia käyttöliittymiä nopeasti sekä on myös tehokas työkalu kauniiden mukautettujen mobiilisovellusten luomiseen, sen laajennettavuuden ansiosta. Erilliset laajennukset vaativat niiden lisäämisen Flutterin `pubspec.yaml` -metadatatiedostoon, jossa määritellään tämän laajennuksen nimi sekä versio. Tämän jälkeen kyseinen laajennus tulee vielä tuoda lähdekoodin puolelle, jotta Dart pystyy käyttämään sen ominaisuuksia.

Tässä työssä keskityin yleisessä käytössä oleviin perusominaisuuksiin, joita pLABin ohjelmistokehittäjät käyttävät. Kehitin pieniä sekä askeettisia mobiilisovelluksia, jotka sisälsivät yhden tai useamman listatun ominaisuuden. Tämän jälkeen testasin sekä raportoin kyseisten ominaisuuden toiminnallisuuden molemmilla alustoilla Androidilla sekä iOS:lla. Lähestulkoon jokainen ominaisuus tarvitsi oman erillisen laajennuksen. Tässä työssä käyn läpi vain laajimmat sekä käytetyimmät ominaisuudet.

#### 4.1 Sijaintipalvelut

Flutter tukee sijaintipalveluita hyvin sekä niiden käyttäminen onnistuu lisäämällä `MapView`-laajennus projektiin. Tämä on `Widget`, joka näyttää Google Map -näytymän, joka on rakennettu `PlatformView` API:n päälle. Tätä näkymää on helppo muokata eri tavoilla. Laajennus sisältää vaihtoehto parametrin, jonka avulla kehittäjä voi muokata esim. eleiden aktivoinnin tai poistamisen, kameran oletusnäytymän paikoituksen, näytymän tyylin, jonka voi asettaa satelliitti, hybridi, normaali tai maasto -näkyväksi. Tämän lisäksi kehittäjä voi muokata myös paljon muita toimintoja.

Geofencing ominaisuuden käyttäminen vaatii myös oman laajennuksen yhtensopivuuden vuoksi, koska Androidilla ja iOS:lla on omat API:t geofencing-tapahtumien rekisteröintiin ja käsittelyyn. Laajennuksella täytyi olla kyky luoda Geofence esiintymiä, jotka sisälsivät koordinaatit ja geofencen säteen, yksilöllisen ID:n sekä luettelon geofencing-tapahtumista, joita haluttiin kuunnella. Android tarjoaa laajemmin vaihtoehtoja geofenssien määrittelemiseksi kuin iOS. Androidin vaihtoehtoista pystyi rekisteröimään takaisin kutsun, kun määritetyn alueen geofence-tapahtuma vastaanotettiin. (Medium Flutter 2018.)

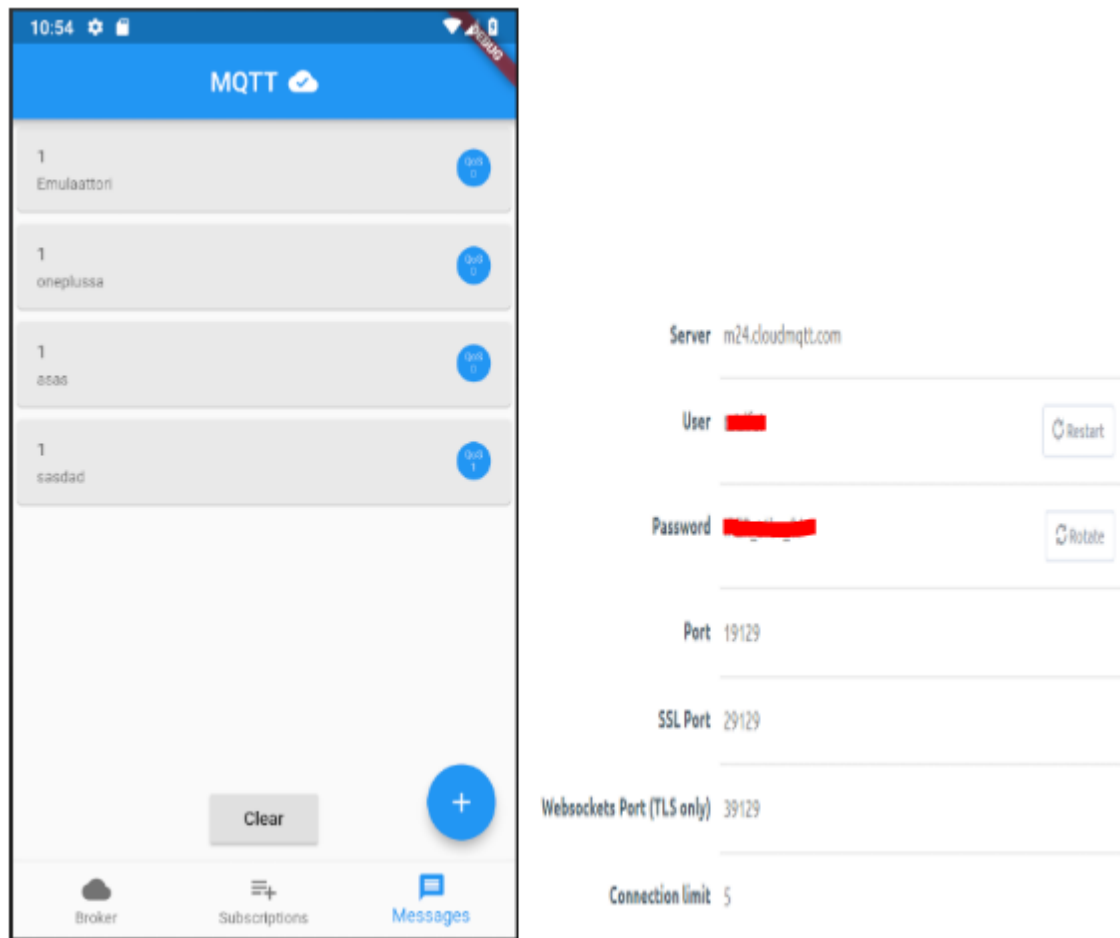
Kyseiset ominaisuudet toimivat Androidilla moitteettomasti, mutta iOS-alustalla oli ongelmia mapview-laajennuksen kanssa. Erilaisten asetusten määrittäminen sekä käyttöliittymän ulkoasu olivat helposti muokattavissa molemmille alustoille.

#### 4.2 Tietoliikenne

Tietoliikenne ominaisuuksien kehittämiseen Flutterilla on hyvä käyttää http-laajennusta. Tämä helpottaa verkkopyyntöjen suorittamista, kuten tietojen hake- mista JSON -tiedostomuodoista. Tiedostojen lataaminen HTTP-pyyntöllä vaatii Apple Transport Security -ominaisuuden poistamisen käytöstä, jotta se toimii iOS-alustoilla.

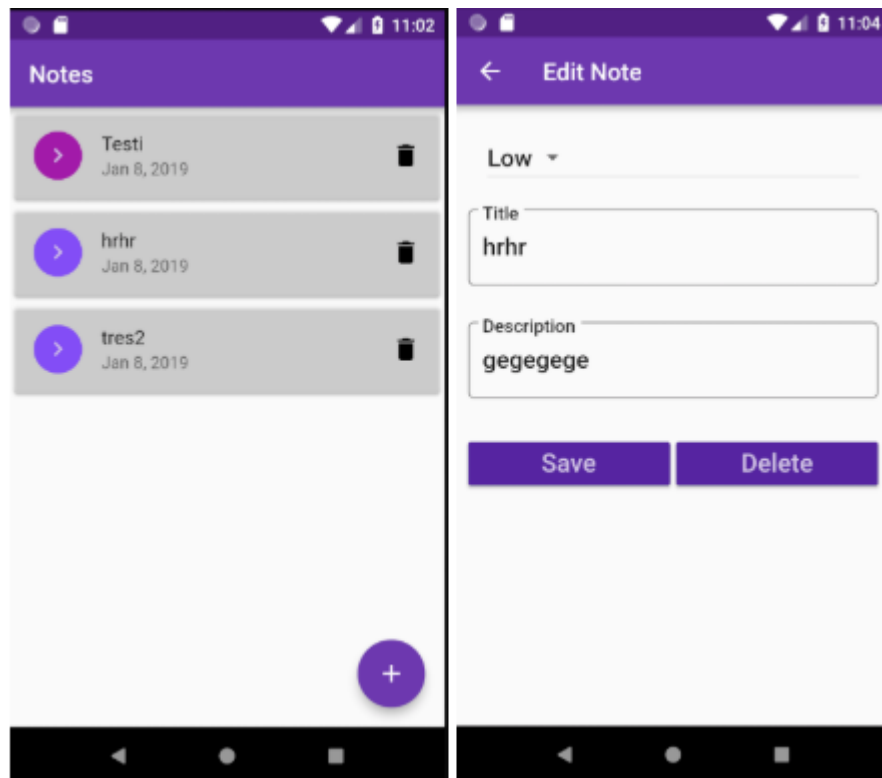
HTTP, eli verkkopyynnön tekeminen on helppoa, mutta saatu data on suositelta- vaa muuttaa Dart-objektiksi, sillä sen käsitteleminen on käytännöllisempää. Ta- vallinen JSON muuntaminen Flutterissa on hyvin yksinkertainen. Flutterissa on sisäänrakennettu dart:convert -kirjasto, joka sisältää suoran JSON-kooderin sekä dekooderin. Myös muita tiedostomuotoja on mahdollista ladata netistä, sekä tallentaa ne välimuistiin. Ladattujen tiedostojen näkymää voi kontrolloida sekä muokata helposti.

Testasin myös MQTT-protokollaa, joka vaatii mqtt-laajennuksen sekä muutaman kirjaston lisäämisen projektiin, jotta mobiilisovellus toimi. Käytin testaamisessa cloudmqtt -serveriä. Asensin tämän testisovelluksen emulaattorille sekä puheli- melle. Tämän jälkeen yhdistin emulaattorin sekä puhelimen käyttämään samaa palvelinta. Pystyin lähettämään sekä vastaanottamaan viestejä reaaliajassa mo- lemmilla laitteilla kuvio 9 mukaisesti. Ominaisuus toimi todella hyvin molemmilla alustoilla ilman ongelmia.



Kuvio 9. Mqtt-sovelluksen käyttöliittymä sekä cloudmqtt –serverin näkymä

Lisäksi testasin SQL -ominaisuutta, joka on yksi suosituimmista tavoista tallentaa tietoja paikallisesti. Flutter tukee SQLite -laajennusta, joka mahdollistaa tietojen tallentamisen lokaalisti. Kehitin mobiilisovelluksen johon käyttäjä pystyi tallentamaan muistiinpanoja, tarkastelemaan niitä sekä asettamaan tallennetulle muistiolle prioriteetin, jonka mukaan se sijoittui tallennettujen muistioiden listalla. Kuviossa 10 näkyy mobiilisovelluksen käyttöliittymän molemmat näytöt, joiden välillä käyttäjä voi navigoida. Mobiilisovellus toimii Androidilla, mutta iOS-alustalla oli ongelmia tallentaa tiedostoja pitkäaikaismuistiin.

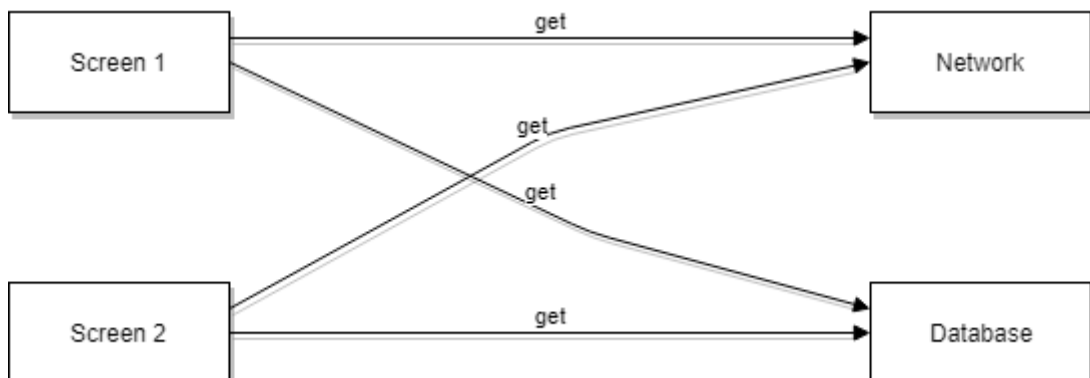


Kuvio 10. Muistilehtiö-sovelluksen käyttöliittymä

### 4.3 Datarakenteet

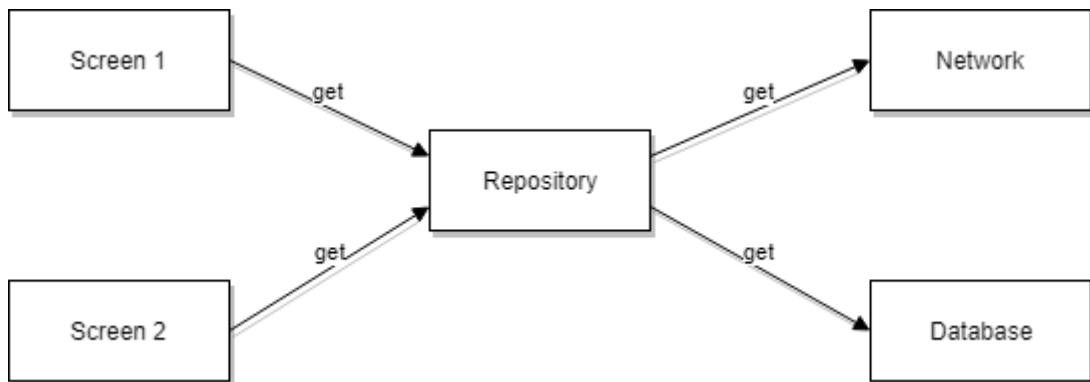
Useimmat nykypäivän mobiilisovellukset tarvitsevat yhteyden palvelimeen, josta se hakee sekä näyttää tietoja. Yhdessä välimuistin, tietokannan ja virheenkäsittelyn kanssa tämä voi olla hankalaa toteuttaa. Lisäämällä datarakenteita tietojen käyttö on erittäin kätevää ja helppoa.

Alla olevassa kuvio 11:ssä on esimerkki, jossa kumpikin näyttö tarvitsee tietoja ja pyytää niitä suoraan verkosta. Jos tiedot tallennetaan välimuistiin, tulee kummankin näytön pyytää tietoja ensin välimuistista, minkä jälkeen niitä kysytään verkosta.



Kuvio 11. Tietojen hakeminen palvelimelta sekä välimuistista ilman erillistä data-rakennetta (ProAndroidDev 2018.)

Arkisto on tässä kuvio 12:ssa tietolähde, joka pyytää verkosta ja välimuistista tietoja. Tietoja pyytävien näyttöjen ei tarvitse välttämättä tietää, mistä tiedot tulevat. Tämä mahdollistaa kaikkiin tietoihin liittyvien toimintojen sijoittamisen yhteen paikkaan eli tässä tapauksessa arkistoon. Refaktorointi ja muokkaaminen on sujuvampaa, sekä tietojen käyttö on entistä helpompaa lisäämällä datarakenteita.



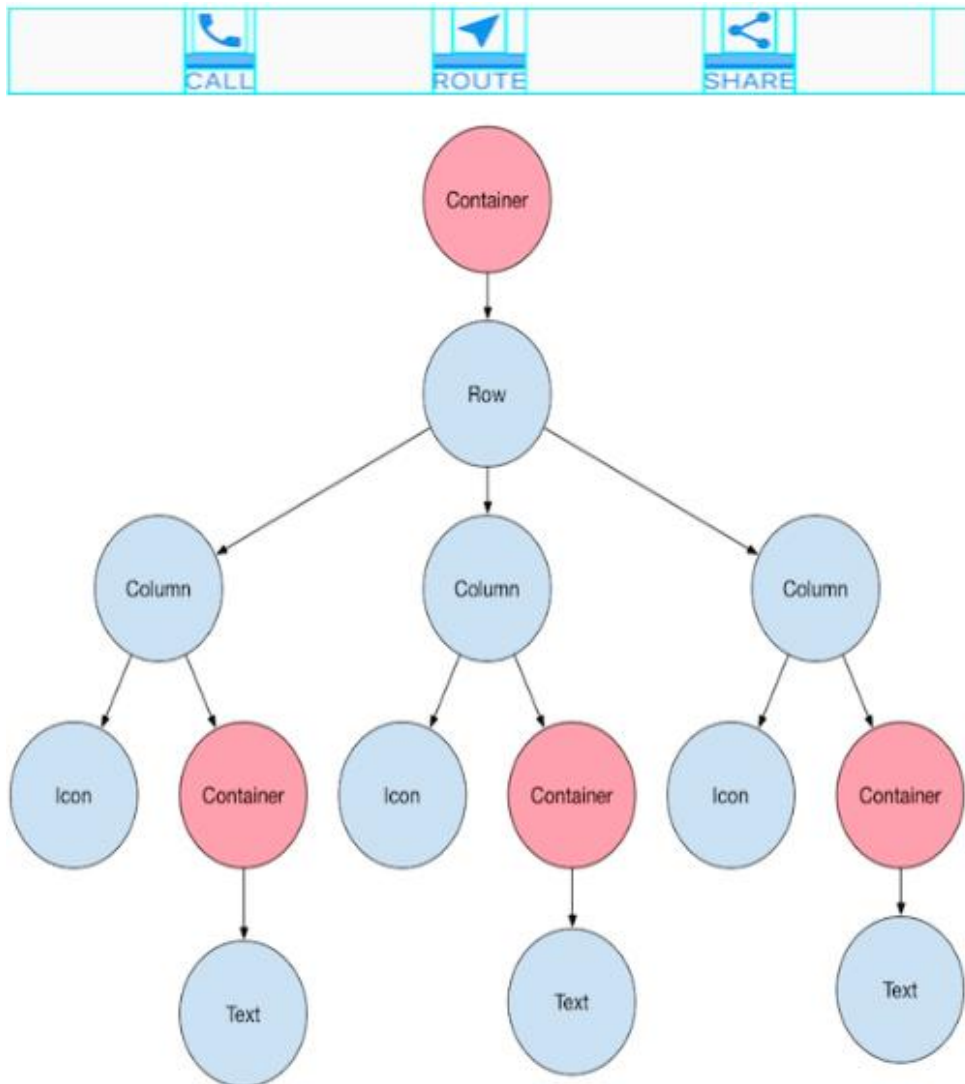
Kuvio 12. Tietojen hakeminen sekä näyttäminen datarakenteen kanssa (ProandroidDev 2018.)

Löysin valmiin projektin, jossa tämä ominaisuus oli toteutettu. Mobiilisovelluksen tarkoituksena oli näyttää kirjoja internetin välityksellä. Käyttäjä pystyi hakemaan kirjoja nimellä, lisäämään ne omaan kokoelmaan sekä jättämään arvosteluja. Sovellus käytti Google Books API:a kirjojen hakemiseen Googlen tietokannasta. Tämä sovellus toimi molemmilla alustoilla sekä lisätty datarakenne selkeytti lähdekoodia ja nopeutti kirjojen hakemista.

#### 4.4 Käyttöliittymä

Flutterin asettelumekanismin ja käyttöliittymän ydin on widgetit. Mobiilisovelluksessa näkyvät kuvat, kuvakkeet ja teksti ovat kaikki widgettejä. Myös rivit, sarakkeet ja ruudut, jotka järjestävät, rajoittavat ja kohdistavat näkymiä on widgettejä. Flutterin sisäänrakennettujen widgettien avulla on helppo kehittää kauniita mobiilisovelluksia, sillä se tarjoaa runsaan valikoiman niin Android kuin iOS -tyylisiä widgettejä, sekä mahdollisuuden luoda omia mukautettuja widgettejä, joilla luoda käyttöliittymä.

Voit luoda käyttöliittymän yhdistämällä widgettejä toisten widgettien sisään, mikä mahdollistaa samanlaisen käyttäjäkokemuksen alustasta tai laitteesta riippumatta. Kuvio 13:ssa on esimerkki, jossa näkyy kolme saraketta, joissa jokainen sarake sisältää kuvakkeen sekä tekstin. Lisäksi kuviossa näkyy, miten widgettien hierarkia tässä käyttöliittymässä toimii. (Layouts in Flutter 2019.)

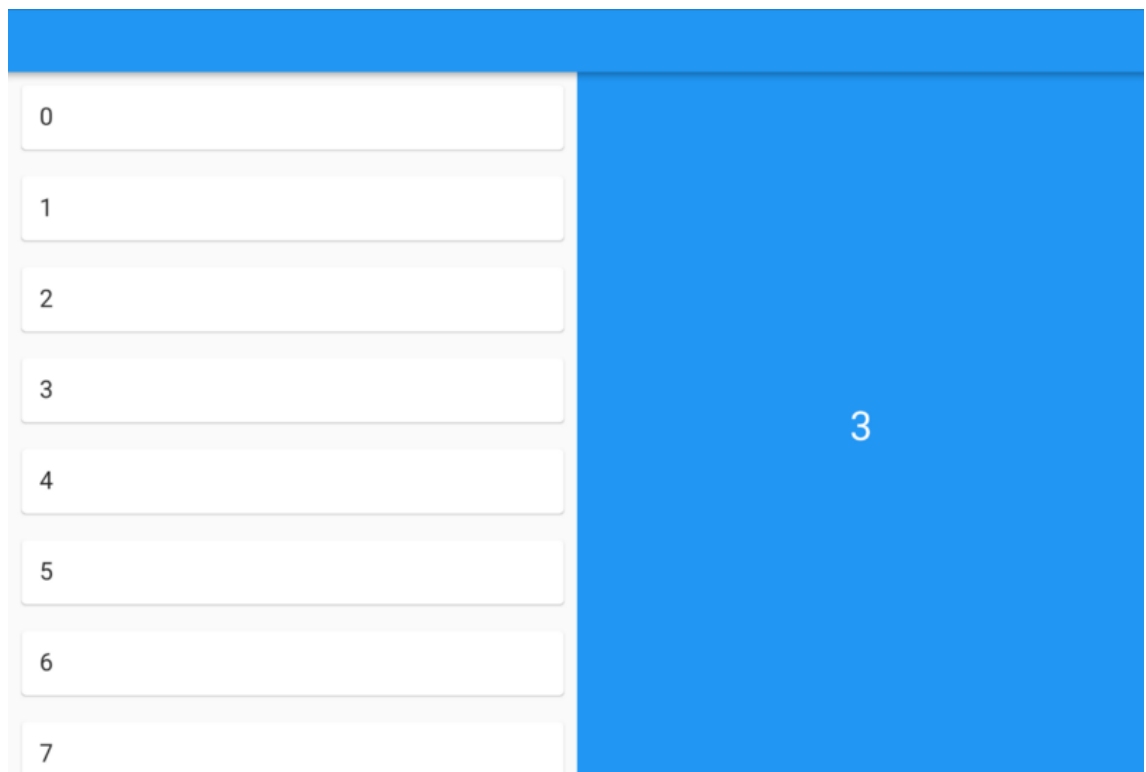


Kuvio 13. Esimerkki käyttöliittymän osasta sekä sen hierarkia widgetteinä (Layouts in Flutter 2019.)

Mobiilisovellusten on tuettava eri laitekokoja, pikselitiheyksiä ja suuntauksia. Lisäksi sovellusten täytyy skaalautua hyvin sekä pystyä käsittelemään suunnanmuutoksia säilyttäen näkymän sisältämät tiedot. Flutter tarjoaa eri mahdollisuuksia millä tavalla ratkaista nämä haasteet sen sijaan, että olisi vain yksi ratkaisu.



Tein mobiilisovelluksen, joka sisälsi nämä ominaisuudet sekä kaksi eri näkymää. Ensimmäinen näkymä sisälsi listan numeroista ja toinen näkymä esitti listasta valitun numeron. Määritin molemmille näkymille widgetit, joiden mukaan mobiilisovelluksen käyttöliittymä piirtyi näytölle. Ensimmäinen näkymä sisälsi OrientationBuilder -widgetin, joka tarkisti, onko laitteella riittävä leveys käsitellä molempia näkymiä yhtäaikaan. Jos leveys oli riittävä, molemmat näkymät esitettiin laitteen näytöllä kuten kuvio 14:ssä.



Kuvio 14. Mobiilisovelluksen näkymä tabletilla sekä vaaka-asennossa

Jos näin ei ollut, mobiilisovellus avasi uuden sivun toiselle näkymälle kuten kuvio 15:ssä. Nämä ominaisuudet toimivat testatuilla laitteilla ja käyttöliittymä piirtyi laitteen koon sekä vaaka- ja pystyasennon mukaan.



Kuvio 15. Mobiilisovelluksen näkymä puhelimella pystyasennossa

Myös oikean teeman määrittäminen käyttäjän alustalle sopivaksi onnistuu tarkistamalla, mikä alusta käyttäjällä on käynnissä. Tämän jälkeen käyttöliittymä valitsee, mitä käyttöliittymän komponentteja käytetään. Kuvio 16:ssa on esimerkki lähdekoodista, jonka avulla teema määräytyy alustan mukaan.

```
return new MaterialApp(
  // default theme here
  theme: new ThemeData(),
  builder: (context, child) {
    final defaultTheme = Theme.of(context);
    if (defaultTheme.platform == TargetPlatform.iOS) {
      return new Theme(
        data: defaultTheme.copyWith(
          primaryColor: Colors.purple
        ),
        child: child,
      );
    }
    return child;
  }
);
```

Kuvio 16. Lähdekoodi teeman määrittämiselle käytettävän alustan mukaan

#### 4.5 Testaaminen

Testaaminen tapahtui Android Studion sekä XCoden kautta, käyttäen ulkoisia laitteita. Ulkoisina laitteina toimivat; OnePlus 5, iPhone 4, iPhone 7 sekä iPad. Kaikki kehitetyt mobiilisovellukset testattiin ensin Android -alustalla, koska tätä ympäristöä käytettiin mobiilisovellusten kehittämiseen sekä virheiden korjaaminen oli helpompaa. Kun kaikki mobiilisovellukset saatiin toimimaan Android -alustalla, niitä siirryttiin testaamaan iOS -alustalla. Kaikki kehitetyt mobiilisovellukset testattiin jokaisella laitteella, jonka jälkeen testatun ominaisuuden toiminnallisuus sekä ongelmat kirjattiin erilliseen dokumentaatioon.

Testauksen raportoinnissa kirjattiin ylös mitkä ominaisuudet toimivat sekä jos kyseinen ominaisuus ei toiminut tietyllä laitteella tai antoi virheitä. Raportissa kerrotaan myös, miten testatut ominaisuudet otetaan käyttöön, mitä laajennuksia ne vaativat sekä miten niiden muokkaaminen tapahtuu. Lisäksi mobiilisovelluksien suorituskykyä tarkasteltiin, mutta vain käyttötuntuman perusteella. Oletusarvoisesti Flutter-sovellukset olivat suorituskykyisiä, ilman erillistä mikro-optimointia. Kehitetyt mobiilisovellukset olivat pieniä sekä niissä ei ilmennyt suorituskykyongelmia, jonka takia niiden tarkempi diagnosointi ei ollut tarpeellista.

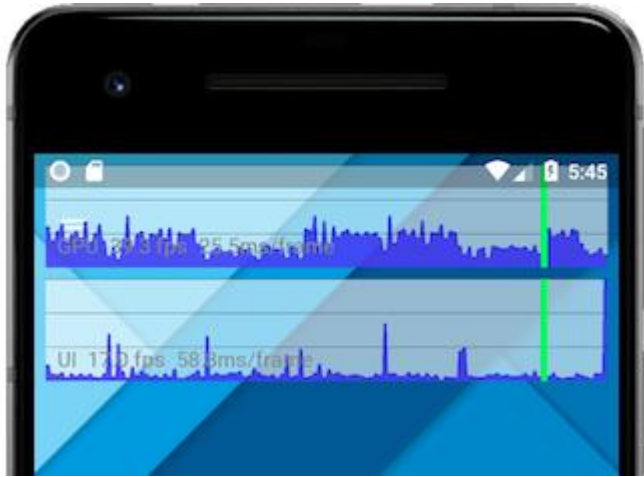
Kaikki pLABin listaamat ominaisuudet saatiin toimimaan Android -alustalla, mutta iOS -alustalla muutamien ominaisuuksien kanssa oli ongelmia, joita ei saatu ratkaistua. Tämän jälkeen mobiilisovellukset olivat valmiita esitettäväksi ohjaajalle.

#### 4.6 Suorituskyky

Flutter mobiilisovellusten suorituskyvyn profilointi tulisi suorittaa ulkoisella Android- tai iOS-laitteella, sillä emulaattorit eivät käytä samaa laitteistoa, joten niiden suorituskyky ominaisuudet ovat erilaisia sekä tietyt toiminnot eivät vastaa mobiilisovelluksen lopullista käyttäytymistä.

Flutter mahdollistaa suorituskyky -profiilin käytön eri ohjelmistoympäristöissä. Käyttöönotto on helppoa ja tapahtuu käytettävän ohjelmointiympäristön valikosta joka avaa uuden ikkunan ohjelmointiympäristön käyttöliittymään, josta voit valita eri päällysteitä, jotka näyttävät esimerkiksi mobiilisovelluksen fps:n. Tämä pääl-

lyste tulee näkymään laitteen yläreunaan kuvion 17 mukaisesti sekä käyttöliittymän ikkunassa, josta voi seurata suorituskyykyä reaaliajassa. Vihreät vertikaaliset palkit edustavat nykyistä kehystä. (Flutter performance profiling 2019.)



Kuvio 17. Flutter suorituskyyky -profiilin päällyste (Flutter performance profiling 2019.)

## 5 POHDINTA

Mobiilisovellusten kehittäminen sekä Androidille että iOS:lle voi viedä paljon resursseja. Monialustaisten kehitystyökalujen ansiosta kehittäjät voivat luoda mobiilisovelluksia nopeasti molemmille alustoille ja näin ollen vähentää kehitykseen kulutettua aikaa. Nykyaikana on paljon erilaisia monialustaisia työkaluja, mutta halusin oppia Flutterin sekä syventyä sen tarjoamiin ominaisuuksiin, sillä se on saanut paljon huomiota mediassa ja herätti näin ollen mielenkiintoni tätä ohjelmistotyökalua kohtaan. Yllätyin Flutterista, sillä se antaa hyvän sekä laajan tuen monille eri ominaisuuksille sekä sen saama tuki Googlelta ja yhteisöltä on vakuuttava.

Työn alussa minulla oli ongelmia Flutterin asennuksen sekä laajennusten käyttöönottamisen kanssa, mutta hetken totuttelun jälkeen mobiilisovellusten kehittäminen tällä uudella työkalulla oli vaivatonta. Myös Mac OS sekä iOS-alusta toivat omia haasteita, sillä en ollut aikaisemmin kehittänyt mobiilisovelluksia tälle alustalle sekä niiden kehitysympäristöt olivat uusia minulle. Näiden ongelmien ratkaisuun sain kuitenkin apua pLABin ohjelmistokehittäjiltä, mikä auttoi projektin läpiviemistä.

Opinnäytetyön soveltavassa osuudessa kehittämäni mobiilisovelluksien testaaminen vei oletettua kauemmin, koska Flutter päivittyi tasaisin väliajoin, eikä kaikki laajennukset päivittyneet tämän ohjelmistotyökalun mukana, jolloin ominaisuuksien implementointi oli vaikeaa. Lisäksi iOS-alusta toi omat haasteensa, sillä tämä alusta vaatii enemmän oikeuksien myöntämistä sekä tiettyjen ominaisuuksien estämistä, jotta tietyt mobiilisovellukset toimivat täysin.

Työn aikana opin paljon uutta mobiilisovellusten toiminnasta, mobiilikehityksestä sekä uuden ohjelmointikielen. Kokonaisuudessaan tämä opinnäytetyö oli loistava oppimiskokemus sekä tarpeeksi haastava.

Koska Flutterin ensimmäinen vakaa julkaisu tapahtui 9. maaliskuuta 2018, on liian aikaista tehdä vielä kunnollisia johtopäätöksiä. Kuitenkin tässä työssä esitetyistä ominaisuuksista ja toiminnallisuudesta tulee esiin, että se sopii erinomaisesti nopeaan ja luovaan mobiilikehitykseen.

Yhteenvetona koen, että Flutter ei korvaa natiivia mobiilikehitystä, sillä natiivisovelluksilla ei ole rajoitteita sekä niillä on täysi pääsy kaikkiin laitteen palveluihin ja ominaisuuksiin. Tämän takia monimutkaisia tai todella laajoja mobiilisovelluksia on helpompi kehittää natiivilla, koska kaikkia ominaisuuksia on mahdollista käyttää ja niiden toiminnallisuus on taattu. Lisäksi natiivina kehitettyjen mobiilisovellusten tiedostokoko on pienempi.

Flutter kuitenkin tarjoaa loistavan vaihtoehdon monialustaisessa mobiilikehityksessä. Jos Google jatkaa Flutterin tukemista sekä kehittämistä samalla tyylillä, uskon sen olevan suosituin monialustainen kehitystyökalu tulevaisuudessa, sillä se antaa mahdollisuuden kehittää suorituskykyisiä ja vakuuttavia mobiilisovelluksia nopeasti, jotka sopivat asiakkaan tarpeisiin ja vaatimuksiin. Mielestäni Flutter on paras valinta nykyaikaiseen monialustaiseen mobiilikehitykseen.

## LÄHTEET

Bobology 2019. The benefit of cross-platform software. Viitattu 8.3.2019  
<https://www.bobology.com/public/What-is-CrossPlatform-Software.cfm>.

Codeburst 2018. Native vs. cross-platform app development. Viitattu 17.2.2019  
<https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac>.

Dartlang 2018, 9-10. Overview. Viitattu 27.12.2018  
<https://www.dartlang.org/guides/language/specifications/DartLangSpec-v2.2.pdf>.

Flutterdoc 2018a. Stateless widgets. Viitattu 17.12.2018  
<https://flutterdoc.com/stateful-or-stateless-widgets-42a132e529ed>.

Flutterdoc 2018b. Stateful widgets. Viitattu 17.12.2018  
<https://flutterdoc.com/stateful-or-stateless-widgets-42a132e529ed>.

Flutter Docs 2019a. Technical Overview. Viitattu 4.3.2019  
<https://flutter.dev/docs/resources/technical-overview>.

Flutter Docs 2019b. Core principles. Viitattu 4.3.2019  
<https://flutter.dev/docs/resources/technical-overview>.

Flutter Docs 2019c. Set up an editor. Viitattu 8.3.2019  
<https://flutter.dev/docs/get-started/editor>.

Flutter Docs 2019d. Hot reload. Viitattu 8.3.2019  
<https://flutter.dev/docs/development/tools/hot-reload>.

Flutter Docs 2019e. Everything's a widget. Viitattu 11.3.2019  
<https://flutter.dev/docs/resources/technical-overview#everything-s-a-widget>.

Flutter Docs 2019f. Using packages. Viitattu 11.3.2019  
<https://flutter.dev/docs/development/packages-and-plugins/using-packages>.

Flutter performance profiling 2019. The performance overlay. Viitattu 3.4.2019  
<https://flutter.dev/docs/testing/ui-performance>.

Google Developers 2018. Flutter 1.0: Google's Portable UI Toolkit. Viitattu 5.12.2018  
<https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html>.

Hackernoon 2017a. The platform SDKs. Viitattu 4.3.2019  
<https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>.

Hackernoon2017b. WebViews. Viitattu 4.3.2019  
<https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>.

Hackernoon2017c. Reactive Views. Viitattu 4.3.2019  
<https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>.

Hackernoon2017d. Flutter. Viitattu 4.3.2019  
<https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514>.

Layouts in Flutter 2019. Layouts in Flutter. Viitattu 2.4.2019 <https://flutter.dev/docs/development/ui/layout>.

Medium Flutter 2018. Executing Dart in the Background. Viitattu 1.4.2019  
<https://flutter.dev/docs/development/packages-and-plugins/using-packages>.

Mrekuccis blog 2014. Asynchronous I/O micro war. Viitattu 15.4.2019  
<https://mrekucci.blogspot.com/2014/09/asynchronous-io-micro-war.html>.

ProAndroidDev 2018. Flutter bookshelf app. Viitattu 8.3.2019  
<https://proandroiddev.com/flutter-bookshelf-app-part-3-managing-data-the-right-way-30569abf9487>.

Simple programmer 2016. What is mobile development. Viitattu 16.12.2018  
<https://simpleprogrammer.com/what-is-mobile-development/>.

Statista 2018. Annual number of global mobile app downloads. Viitattu 10.12.2018 <https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/>.

Smashing Magazine 2018. Flutter mobile development. Viitattu 16.12.2018  
<https://www.smashingmagazine.com/2018/06/google-flutter-mobile-development/>.

Techopedia 2019. Cross Platform. Viitattu 8.3.2019  
<https://www.techopedia.com/definition/17056/cross-platform>.

TechTarget 2018. Integrated development environment. Viitattu 16.2.2019  
<https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment>.