

Heidi Vulli

FLUTTER-OHJELMOINTIKEHYKSEN EVALUOINTI

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Lokakuu 2019

TIIVISTELMÄ

Heidi Vulli: Flutter-ohjelmointikehityksen evaluointi
Diplomityö
Tampereen yliopisto
Tietotekniikka, DI
Lokakuu 2019

Flutter on Googlen kehittämä alustariippumaton ohjelmistokehitys, joka käyttää ohjelmointikielenään Dart-kieltä, joka on myös Googlen luoma. Dart on luokkapohjainen, puhdas olioperusteinen ohjelmointikieli varustettuna staattisella tyyppityksellä, sekä yksinkertaisella periytymisellä. Kieli muistuttaa ilmaisultaan suosittuja ohjelmointikieliä, kuten C# ja JavaScript. Flutter eroaa muista alustariippumattomista teknologioista sen uniikin arkkitehtuurin avulla. Teknologia ei käytä hyväkseen natiiveja komponentteja, vaikka sen avulla pystytään tekemään natiivien sovellusten tasoisia applikaatioita. Sen sijaan Flutter käyttää omia komponenttejaan, widgeteitä, jotka ovat koko teknologian perusta. Widgetit voidaan jakaa kolmeen eri kategoriaan: peruswidgetit, Android-sovelluksille tyypillisiä ominaisuuksia omaavat Material-widgetit, sekä iOS-sovelluksille tyypillisiä ominaisuuksia omaavat Cupertino-widgetit. Flutterissa ohjelmia luodaan asettelemalla widgeteitä sisäkkäin. Flutterin omalaatuisen arkkitehtuurin johdosta teknologian suorituskyky on lähes kaksi kertaa parempi kuin React Nativella.

Diplomityön tarkoituksena oli evaluoida Flutteria useiden eri kriteerien avulla ja selvittää, voidaanko teknologia ottaa käyttöön finanssialan ohjelmointikonsultointiyrityksessä, Profit Software Oy:ssä. Evaluointikriteerit jaettiin neljään eri ryhmään: infrastruktuuri-, kehitys-, sovellus- ja käytettävyyskriteerit, joiden koettiin kattavan hyvin sovellusten kehitykseen liittyvät seikat, sekä Profitin tarpeet. Soveltuvuuden arviointia varten kehitettiin myös prototyyppisovellus Flutter-teknologialla, joka annettiin diplomityön valmistumisen jälkeen Profitille vapaasti käytettäväksi.

Tutkimuksen tulosten perusteella Flutter ei vielä työn kirjoitushetkellä ole tarpeeksi vakaa teknologia Profitin tarpeisiin ainakaan isoissa ohjelmointiprojekteissa. Vaikka teknologia on pyritty vakauttamaan julkaisemalla suurempia versioita vain kvartaaleittain, eivät ne ole aiemmin olleet taaksepäin yhteensopivia. Tässä vaiheessa teknologian kehitystaivalta tämä on vielä suuri riski, sillä uusien versioiden mukana tulevia uusia ominaisuuksia ei välttämättä pysty ottamaan käyttöön riskeeraamatta edellisessä toteutuksessa toimineita ominaisuuksia. Flutter ei myöskään vielä omaa valmiita kirjastoja ja toteutuksia läheskään yhtä paljon kuin kilpailijat, mikä hidastaa kehitysprosessia. Toisaalta kuitenkin Flutterin ohjelmointikieli Dart muistuttaa monelle kehittäjälle jo ennestään tuttuja ohjelmointikieliä, mikä nopeuttaa teknologian omaksumista. Tämän lisäksi Flutterin dokumentaatio on laaja ja aloittelevat kehittäjät löytävät helposti apua ongelmiinsa Flutterin nettisivujen kautta. Flutterissa on paljon hyvää ja sen tulevaisuus näyttää kirkkaalta, jos vain Google pysyy teknologiansa takana. Flutteria voidaan suositella käytettäväksi Profitilla jo vaikka heti pienissä mobiiliprojekteissa, joiden tarkoituksena on todistaa suunniteltu konsepti oikeaksi.

Avainsanat: alustariippumaton, Dart, evaluointi, Google, Flutter, JavaScript, mobiilikehitys, React Native

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

ABSTRACT

Heidi Vulli: Evaluation of the Flutter framework
Master's thesis
Tampere University
Information Technology, MSc
October 2019

Flutter is Google's cross-platform UI toolkit that uses Dart for its programming language. Dart is an object-oriented, pure class defined programming language that's also developed by Google. Dart is also optionally typed and it supports single-inheritance. The language resembles popular programming languages as C# and JavaScript. The main reason Flutter is different from other similar technologies is its unique architecture. Flutter doesn't utilise any native components, instead it uses its own components, widgets. Widgets in Flutter can be split in to three categories: basic widgets, Material-widgets that have typical properties for Android devices and Cupertino-widgets for iOS devices. Programs in Flutter are created by stacking widgets inside each other. Thanks to Flutter's architecture, its performance is twice as good as React Native's.

The motive for this thesis was to evaluate Flutter with several criteria and examine if the technology can be useful for a Finnish financial consulting enterprise, Profit Software Ltd. The evaluation criteria were divided in to four different groups: infrastructure, development, application and usability. In addition of the criteria, a demo application was created with the technology. The application was given to Profit after the thesis was complete.

The outcome of the research was that Flutter is still too immature and through that not yet suitable for Profit's needs in bigger programming projects. Though stable versions are published only in every quarter, they have not been backward compatible with previous versions. This is a big risk for Flutter because new features cannot necessarily be taken in use without breaking previously used features. Flutter also has a lot less packages than for example its rival React Native, which slows down the development process considerably. But on the other hand, Dart is intuitive and it's based on very popular languages that are well known among developers, which accelerates the development process. Furthermore, the documentation of Flutter is extensive which also eases the learning process. Flutter is a candidate in the field of programming frameworks if only Google will stay behind its technology. Flutter can be recommended for Profit already in smaller proof-of-concept mobile development projects.

Keywords: cross-platform, Dart, evaluation, Google, Flutter, JavaScript, mobile development, React Native

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

ALKUSANAT

Ajatus diplomityöstä laitettiin kytemään Profit Software Oy:n kanssa jo loppuvuodesta 2018 ja aihe saatiin iteroitua Jammun, Oskun ja ohjaajani Karin kanssa valmiiksi vuoden 2019 toukokuussa. Profitille iso kiitos siitä, että sain tehdä työn mielekkäästä aiheesta ja käyttää vapaasti aikaa siihen silloin kuin halusin. Kiitos myös Karille kommentteista ja tuesta työn kanssa.

Diplomityön ollessa viimeinen yliopistolle tehtävä työni, on myös hyvä aika kiittää kaikki opiskelutovereita kaikista näitä vuosista. Suurin kiitos Hiukkasen fukseille 2013 mahtavasta fuksivuodesta ja upeista hetkistä myös myöhempinä vuosina. Tämän lisäksi massiiviset kiitokset toiselle yliopistoperheelleni, Tampereen teekkerien PerinneSeuralle, porukkaan mukaan ottamisesta ja monista korvaamattomista muistoista. Kiitos myös kaikille muille matkassa mukana vaikuttaneille.

Tampereella, 20. lokakuuta 2019

Heidi Vulli

SISÄLLYSLUETTELO

1	Johdanto	1
1.1	Työn tavoitteet ja rajaus	1
1.2	Työn rakenne	2
2	Natiivit ja alustariippumattomat teknologiat yleisesti	3
2.1	Natiivit teknologiat	3
2.2	Alustariippumattomat teknologiat	4
2.2.1	Web-teknologiat	5
2.2.2	Hybriditeknologiat	6
2.2.3	Ristiinkäännetyt sovellukset	7
2.2.4	Skriptikielet	7
3	Dart	9
3.1	Kääntäminen Dartissa	9
3.2	Kapselointi	9
3.3	Tyypitys	10
3.3.1	Sisäänrakennetut tyypit	10
3.4	Muuttujat	13
3.5	Funktiot	13
3.6	Suoritusta kontrolloivat lausekkeet	15
3.7	Assert	15
3.8	Poikkeukset	16
3.9	Luokat	16
3.9.1	Abstraktit luokat	18
3.9.2	Periytyminen ja mixinit	18
3.10	Geneeriset kokoelmaluokat	19
3.11	Vertailu kilpailijoihin	19
4	Flutter	22
4.1	Flutterin arkkitehtuuri	22
4.2	Flutterin puurakenne	23
4.3	Widgetit Flutterin perustana	24
4.3.1	Tilattomat widgetit	24
4.3.2	Tilalliset widgetit	25
4.3.3	Periytyvät widgetit	26
4.3.4	Avaimet	27
4.4	Hot Reload ja Hot Restart	28
4.5	Vertailu React Native -ohjelmistokehykseen	29
4.5.1	Arkkitehtuuri ja suorituskyky	29

4.5.2	Kehityksen nopeus	29
4.5.3	Projektin pystytys	30
4.5.4	Yritykset teknologioiden taustalla	31
5	Tutkimus	32
5.1	Evaluointikriteerit	32
5.1.1	Infrastruktuurikriteerit	32
5.1.2	Kehityskriteerit	34
5.1.3	Sovelluskriteerit	37
5.1.4	Käytettävyysskriteerit	39
5.2	Prototyypisovellus	40
5.2.1	Aloituskäytännön näkymä	40
5.2.2	Ostotapahtuma-näkymä	40
6	Tulokset	41
6.1	Lisenssi	41
6.2	Tuetut kohdealustat	41
6.3	Tuetut kehitysalustat	42
6.4	Jakelukanavat	42
6.5	Ansaintamalli	42
6.6	Kansainvälistäminen	42
6.7	Teknologian kannattavuus pitkällä aikavälillä	43
6.8	Kehitysympäristö	45
6.9	Teknologian omaksuminen	46
6.10	Skaalautuvuus	46
6.11	Projektimenetelmien yhteensopivuus	47
6.12	Käyttöliittymäsuunnittelu	47
6.13	Testaus	48
6.14	Jatkuva käyttöönotto	48
6.15	Projektin konfigurointi eri ympäristöjen välillä	48
6.16	Ylläpidettävyys	49
6.17	Laajennettavuus	49
6.18	Integraatio natiiville koodille	50
6.19	Kehityksen nopeus	50
6.20	Laitekohtaisten ominaisuuksien saatavuus	51
6.21	Alustakohtaisten ominaisuuksien saatavuus	51
6.22	Tuki liitetyille laitteille	51
6.23	Syötteen heterogeenisyys	51
6.24	Ulostulomuotojen heterogeenisyys	52
6.25	Sovelluksen elinkaari	52
6.26	Palvelin-selain-integraatio	52
6.27	Turvallisuus	52

6.28 Sovelluksen liikutettavuus	53
6.29 Sovelluksen ulkonäkö ja tuntuma	54
6.29.1 Animaatiot ja transitiot	54
6.30 Suorituskyky	55
6.31 Käyttäjän autentikointi	55
7 Yhteenveto	56
Lähteet	58
Liite A Kahden eri widgetin paikan vaihtaminen napin painalluksesta	63
Liite B Kuvia demosovelluksesta	65
Liite C Flutterin lisenssi	67
Liite D Esimerkki eri ympäristöjen konfiguroinnista Flutterin avulla	68
Liite E Akun varaustason hakeminen natiiveista rajapinnoista	71

KUVALUETTELO

1.1	Eri mobiilialustojen markkinaosuusprosentit vuosina 2009-2019 [58].	1
2.1	Twitterin natiivisovellus Androidilla koodattuna (vasemmalla) ja PWA-sovellus (oikealla)	6
2.2	React Nativen arkkitehtuurikuvaus	8
3.1	TIOBE indeksit JavaScriptille ja Dartille.	20
3.2	Dartin, JavaScriptin ja TypeScriptin StackOverflow:n kysymyksien määrien kehitys vuosina 2012-2019.	20
4.1	Flutterin arkkitehtuurikuvaus	23
4.2	Flutterin puurakenne	24
4.3	Tilallisyen widgetien muokkaaminen ilman avaimia	27
4.4	Tilallisyen widgetien muokkaaminen avaimien kanssa	28
4.5	React Nativen ja Flutterin arkkitehtuurit vertailtavassa muodossa	30
6.1	Flutterin ja React Nativen StackOverflow:n kysymyksien määrien kehitys kvartaaleittain vuosina 2015-2019.	45
6.2	Kuvaus BLoC-mallista.	46
6.3	Natiivien rajapintojen kutsuminen Flutterista.	50
B.1	Kuvia demosovellukseen tehdyistä näkymistä.	65
B.2	Demosovelluksen eri osiot laajennettuina.	66

TAULUKKOLUETTELO

2.1	Ohjelmointikielet mobiilialustottain	3
6.1	Flutterin viimeisimmät julkaisut	44
6.2	Flutterin elinkaaren tiloja vastaavat tilat Androidilla ja iOSilla.	52

OHJELMA- JA ALGORITMILUETTELO

3.1	Sama muuttuja ensin kehittäjän tyypittämänä ja sen jälkeen Dartin tyypittämänä.	10
3.2	Numeroiden esittely	10
3.3	Esimerkki totuusarvoista	11
3.4	Esimerkki merkkijonojen ominaisuuksista	11
3.5	Esimerkki listoista	11
3.6	Esimerkki järjestämättömistä kokoelmista	12
3.7	Esimerkki assosiaatiotaulusta	12
3.8	32-bittisten Unicode-arvojen esittäminen	12
3.9	Vakioarvoisen muuttujan esittely	13
3.10	Esimerkki funktioiden esittely	13
3.11	Esimerkki funktion parametreista	14
3.12	Esimerkki asynkronisuudesta	14
3.13	Ilmaisu luokan esittelylle.	16
3.14	Ilmentymämuuttujien käyttö	16
3.15	Luokan rakentajien määrittely ja käyttö	17
3.16	Esimerkki Mixinin käytöstä	18
3.17	Esimerkki geneeristen tyyppien käytöstä	19
4.1	Esimerkki tilattomasta widgetistä	24
4.2	Esimerkki tilallisesta widgetistä	25
4.3	Esimerkki periytyvästä widgetistä	26
6.1	Irrelevantin datan siivous välimuistista 15 minuutin välein	53
A.1	Esimerkkisovellus, jonka avulla voidaan vaihtaa kahden <code>Tile</code> -widgetin paikkaa napin painalluksesta.	63
D.1	Konfiguraatiodostoto, jossa määritellään eri ympäristöjen välillä konfiguroitavat muuttujat.	68
D.2	Tuotantoympäristön konfigurointi	69
D.3	Kehitysympäristön konfigurointi	69
D.4	Varsinainen sovellus	69
D.5	<code>TervetuloaSivu</code> -widget, jossa käytetään konfiguroitua ympäristön nimeä.	70
E.1	Flutter-puolen toteutus sovellukselle.	71
E.2	Natiivien rajapintojen kutsuminen Kotlinilla.	72

LYHENTEET JA MERKINNÄT

AOT	Ennenaikainen kääntäminen (engl. Ahead-Of-Time)
API	Ohjelmointirajapinta (enlg. Application programming interface)
App store	Applen sovelluskauppa
CSS	Kokoelma tyyliohjeita, joita käytetään ohjelmointiprojekteissa (engl. Cascading Style Sheets)
Google Play	Android-laitteiden sovelluskauppa
HTML	Hypertekstin merkintäkieli (engl. Hypertext Markup Language)
iOS	Applen kehittämä käyttöjärjestelmä
JIT	Ajonaikainen kääntäminen (engl. Just-In-Time)
SDK	Software Development Kit
MVC	Ohjelmistoarkkitehtuuri, joka erottaa käyttöliittymän sovellusalue-tiedosta (engl. Model-View-Controller)
MVP	Minimaalinen versio kehitettävästä tuotteesta, jonka avulla voidaan validoida sen oletettu potentiaali (engl. Minimum Viable Product)
npm	JavaScript-pakettien hallintatyökalu (engl. Node Package Manager)
Ohjelmistokehys	Ohjelmistotuote, joka muodostaa rungon kehitettävälle ohjelmistolle ja nopeuttaa sen kehittämistä
POC	Konseptin todentaminen (engl. Proof-of-concept)
Push-ilmoitus	Laitteeseen lähetettäviä ilmoituksia, jotka herättävät lähes aina käyttäjän huomion
PWA	Progressiivinen web-sovellus (engl. Progressive Web Application)
UX	Käyttäjäkokemus (engl. User Experience)

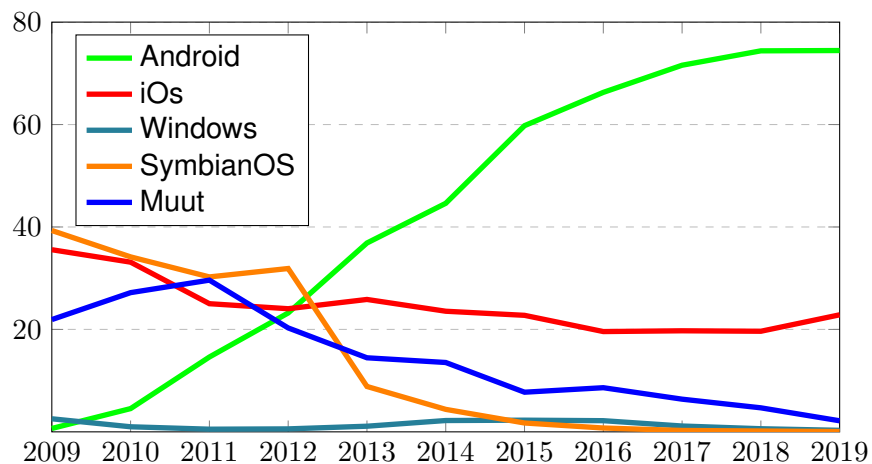
1 JOHDANTO

Tässä diplomityössä esitellään *Flutter*-ohjelmointikehystä. Esittely suoritetaan ensin teorian avulla ja lopussa teknologiaa evaluoidaan usean eri evaluointikriteerin avulla.

1.1 Työn tavoitteet ja rajaus

Työn tavoitteena on selvittää onko Flutter sopiva ohjelmistokehys finanssialan ohjelmointikonsultointiyritykselle, Profit Software Oy:lle, sen tulevissa projekteissa. Työssä kehitetään demo sovelluksesta, jonka avulla voi ostaa lippuja erilaisiin tapahtumiin. Sovelluksen tarkoituksena on auttaa teknologian evaluointia evaluointikriteerien kanssa.

Työssä rajoitutaan pääosin mobiilikehitykseen, sillä Flutterin web-kehitys ei ole vielä siinä pisteessä, että sitä olisi mielekästä evaluoida. Tämän lisäksi aluetta rajataan käsittelemään mobiilialustoista vain Androidia ja iOSia, sillä muiden alustojen markkinaosuus ei ole tällä hetkellä niin merkittävä, että niitä kannattaisi ottaa mukaan tutkimukseen. Eri alustojen markkinaosuuksia on kuvattu kuvaajassa 1.1, josta huomataan selkeästi Androidin ja iOSin merkityksellisyys.



Kuva 1.1. Eri mobiilialustojen markkinaosuusprosentit vuosina 2009-2019 [58].

1.2 Työn rakenne

Työn toisessa osiossa keskitytään taustoittamaan työtä avaamalla lukijalle natiivien ja alustariippumattomien teknologioiden teoriaa. Kolmas osio keskittyy Flutterin ohjelmointikieleen, *Dartiin*. Osiossa käydään läpi Dartin tärkeimmät ominaisuudet sekä vertaillaan kieltä JavaScriptiin ja TypeScriptiin. Neljännessä osiossa esitellään Flutter. Osiossa esitellään ensin Flutterin arkkitehtuuri, jonka jälkeen selitetään Flutterin puiden avulla miten kuvantaminen toimii. Näiden jälkeen esitellään Flutterin perustana olevat widgetit ja samalla kerrotaan miten avaimia (engl. keys) voidaan käyttää Flutterissa suorituskyvyn optimointiin. Viimeisenä osiossa vertaillaan Flutteria React Nativeen. Viides osio esittelee tutkimuksen tavoitteet sekä tutkimuksessa käytetyt evaluontikriteerit. Evaluointikriteereitä on 31 ja ne on jaettu neljään eri ryhmään: infrastruktuuri-, kehitys-, sovellus- ja käytettävyyskriteerit. Osion lopussa esitellään prototyypisovellus, jonka motiivina oli helpottaa Flutterin evaluointia sekä tarjota Profit Software Oy:lle demosovellus. Osiossa kuusi esitellään työn tulokset. Tulokset esitellään kriteereittäin. Työn lopussa on yhteenveto, jossa vedetään yhteen koko työ ja erityisesti tutkimuksen tulokset.

2 NATIIVIT JA ALUSTARIIPPUMATTOMAT TEKNOLOGIAT YLEISESTI

Mobiilisovellusmarkkinat kasvavat jatkuvasti ja nykyään lähes jokaisella yrityksellä on oma mobiilisovellus. Mobiilisovelluksia voidaan kehittää useilla eri teknologioilla, joita esitellään tässä osiossa. Tässä diplomityössä teknologiat jaetaan ensin natiiveihin ja alustariippumattomiin, jonka jälkeen alustariippumattomat jaetaan vielä pienempiin osiin Niclas Hanssonin ja Tomas Vidhallin tutkimuksessa [60] esittelemän jaon perusteella.

2.1 Natiivit teknologiat

Natiivitekniologioiden avulla kehitettävä mobiilisovellus toteutetaan hyödyntäen mobiilialustojen omia ohjelmointikieliä ja työkaluja. Natiivien sovellusten kehittämisessä käytetään alustakohtaisia rajapintoja ja komponentteja, joiden avulla saavutetaan kullekin alustalle ominaisia toimintoja.

Sovelluksen kehittäminen usealle eri alustalle natiivitekniologioiden avulla vaatii huomattavan määrän resursseja, sillä sama sovellus tulee kirjoittaa kahdella eri teknologialla alusta loppuun. Androidin ja iOSin tukemat ohjelmointikieliset ovat listattu taulukkoon 2.1. Tämä on natiivien teknologioiden ehdoton heikkous, sillä toisen alustan koodipohjaa ei usein, jos koskaan, pysty hyödyntämään toisella alustalla. [5]

Natiivilla teknologialla kehitetystä sovelluksesta voidaan saada parhaassa parhaassa tapauksessa hyvin tehokas, sillä se kehitetään alustalle tarkoitetulla teknologialla, joka on optimoitu juuri kyseisen alustan tarpeisiin. Natiivin teknologian avulla päästään myös käsiin kaikkiin alustan tarjoamiin ominaisuuksiin, ilman kolmannen osapuolen sovelluksia, joiden käyttöönotto voi alentaa sovelluksen suorituskykyä.

Natiiveja teknologioita kannattaa suosia silloin, kun sovelluksessa ei haluta tinkiä käyttäjäkokemuksesta. Jos osaavia kehittäjiä, aikaa ja rahaa on tarpeeksi, natiivi kehittäminen

Mobiilialusta	Ohjelmointikieliset
iOS	Swift ja Objective-C
Android	Kotlin ja Java

Taulukko 2.1. Ohjelmointikieliset mobiilialustottain

on aina hyvä vaihtoehto.

2.2 Alustariippumattomat teknologiat

Alustariippumattomilla teknologioilla tarkoitetaan sellaisia mobiilikehitysteknologioita, joiden avulla voidaan luoda yhdellä koodipohjalla sama sovellus eri mobiilialustoille. Jos sovellus halutaan kehittää vain yhdelle alustalle, on silloin usein järkevämpää käyttää natiivia teknologiaa, sillä teknologia on optimoitu kyseisen alustan tarpeisiin, kuten edellisessä osiossa kerrottiin. Muissa tapauksissa alustariippumattomat teknologiat ovat varteenotettava vaihtoehto, sillä niiden avulla saavutetaan laajempi käyttäjäkunta pienemmillä resursseilla. Sovellusta voidaan kehittää ja ylläpitää yhden koodipohjan avulla, mikä ansiosta säästetään merkittävästi resursseja. Myös testaaminen ja laadunvarmistus helpottuu, kun testattavana on yksi, yhteinen koodipohja.

Kuten jo aiemmin todettiin, tapauksissa, joissa ei haluta tehdä kompromisseja käyttöliittymä- ja käyttäjäkokemussuunnitelman suhteen, natiivit teknologiat vievät usein voiton. Yhden suunnitelman toteutuksessa sekä Androidille, että iOSille on omat haasteensa, sillä molempien käyttäjäkunnat ovat tottuneet tiettyihin alustakohtaisiin suunnitteluratkaisuihin. Tästä syystä myös monialustakehittäjien tulee olla tietoisia molempien alustojen toiminnasta ja suunnitteluratkaisuista, luodakseen natiivinoloisen kokemuksen myös monialustateknologioilla.

Jos monialustaisella sovelluksella halutaan käyttää hyödyksi laitteen alustakohtaisia ominaisuuksia, kuten kameraa tai GPS tulee sovellukseen ottaa käyttöön erilaisia liitännäisiä. Liitännäisten käytössä ongelmallista on epävarmuus niiden ajantasaisuudesta. Kun alustan SDK päivittyy, vaatii se usein myös liitännäisten päivittämistä. Jos liitännäistä ei päivitetä, voi aiheutua ongelmia versioiden yhteensopivuudessa.

Suorituskykyyn liittyvät kysymykset on myös hyvä ottaa huomioon teknologiaa valittaessa. Alustariippumattomilla teknologioilla toteutetut sovellukset ovat usein suorituskyvyllään heikompia kuin natiiveilla teknologioilla toteutetut sovellukset [27]. Syitä tähän avataan tulevissa kappaleissa tarkemmin.

Alustariippumattomat teknologiat ovat hyvä valinta, jos halutaan luoda sovellus nopeasti ja kustannustehokkaasti. Yksi ja sama kehitystiimi voi hoitaa sovelluksen kehittämisen kaikille alustoille, jolloin rekrytoinnissa ei tarvitse huomioida eri natiiviteknologioiden osaamista, vaan yhden alustariippumattoman kielen osaaminen riittää.

Tässä osiossa alustariippumattomat teknologiat jaetaan neljään osaan N. Hanssonin käyttämän jaottelun [60] mukaan: web-teknologiat, hybridisovellukset, ristiinkäännetyt sovellukset ja skriptikielitekhnologiat.

2.2.1 Web-tekniologiat

Yksi vaihtoehto alustariippumattomaan kehitykseen on web-tekniologiat, joilla tarkoitetaan usein HTML:n, CSS:n ja JavaScriptin yhdistelmää, jota myös HTML5:ksi kutsutaan. Näiden tekniologioiden avulla laitteeseen luodaan tavallisen ladattavan sovelluksen kaltainen web-sivusto, joka operoi täysin laitteen selaimessa. Sovellus käyttäytyy samalla tavalla kuin mikä tahansa internetsivusto, lukuunottamatta sovelluksessa käytettyjä alustalle ominaisia komponentteja ja niiden ominaisuuksia. [45] Tekniologia luokitellaan alustariippumattomaksi, sillä sovellusta pystytään käyttämään kaikilla mobiililaitteilla, joilla voi avata internetiselaimen. Jo aiemmin esiteltujen monialustaisten tekniologioiden vahvuuksien lisäksi web-tekniologioilla kehitettyjä sovelluksia ei tarvitse koskaan ladata laitteeseen, eikä tätä myötä myöskään päivittää, sillä sovellusta ajetaan laitteen selaimessa, jolloin käyttäjä pääsee aina käsiksi sovelluksen uusimpaan versioon.

Web-tekniologioiden suurin heikkous piilee siinä etteivät ne aina pääse käsiksi laitteen natiiveihin ominaisuuksiin selaimessa ajettavuuden takia. Tällöin kaikkia mahdollisia integraatioita ei voida toteuttaa, jonka seurauksena käyttäjäkokemus kärsii, aiheuttaen sen etteivät sovellukset yllä samalle tasolle natiivien vastineiden kanssa. [6]

Monella yrityksellä on laitteeseen ladattavan sovelluksen lisäksi myös vaihtoehtoisena versiona web-tekniologioilla toteutettu sovellus niille käyttäjille, jotka eivät halua tai pysty lataamaan sovellusta laitteeseen. Tunnetuimpia esimerkkejä ovat muun muassa YouTube ja Facebook.

Web-tekniologioiden uusi tulokas - PWA

PWA eli progressiivinen web-sovellus on kuin tavallinen verkkosovellus, mutta sen sijaan että sovelluksella olisi tarkat vaateet ympäristölleen, sovellus mukautuu aina sen hetkiseen ympäristöönsä, jolloin se muun muassa tietää mobiilissa adaptoitua pieneen näyttöön ja työpöytäkäytössä suurempaan [41].

Progressiiviset web-sovellukset käyttävät hyödykseen selaimen välimuistia, johon on mahdollista tallettaa huomattavia määriä dataa, jopa koko verkkosivusto ja sen sisältö. Jos sovellusta käytetään offline-tilassa, voidaan sivusto ladata välimuistista hetkessä. [71]

PWA-sovellukset voidaan ladata myös halutessaan laitteen aloitusnäytölle [64]. Kun sovellus avataan aloitusnäytöltä, on siitä piilotettu selaimessa näkyvä yläpalkki, joka sisältää osoiterivin ja valikot, ja se hakee välimuistista tilan, johon käyttäjä viimeksi sovellusta käyttäessään jäi. Sovellukset pystyvät myös käyttämään osaa alustojen natiiveista ominaisuuksista, kuten push-ilmoituksia [64], jotka ovat tärkeitä sitomaan käyttäjän käyttämään sovellusta. Kuvassa 2.1 on esimerkki siitä, miltä PWA-sovellus näyttää verrattuna natiiveilla tekniologioilla kehitettyyn sovellukseen.



Kuva 2.1. Twitterin natiivisovellus Androidilla koodattuna (vasemmalla) ja PWA-sovellus (oikealla)

2.2.2 Hybriditeknologiat

Hybriditeknologiat ovat verkkosovelluksiin verrattuna askel natiivimpaan suuntaan, sillä hybridisovelluksissa voidaan kutsua natiiveja API-rajapintoja, jolloin sovellus saa käyttöönsä alustakohtaisia ominaisuuksia, kuten kameran ja GPS:n. Hybridisovellukset voidaan julkaista Google Playssa, sekä Apple Storessa. Hybriditeknologiat käyttävät HTML5-teknologioita web-teknologioiden tapaan. Hybridisovelluksissa sovelluksen sisälle luodaan oma selainympäristö WebView-komponentin [13] avulla, jonka sisällä HTML5:n avulla tehty sisältö näytetään. Näitä teknologioita ovat muun muassa PhoneGap [45], Trigger [49] ja Ionic [53].

Hybridisovelluksien heikkoutena on huono suorituskyky, joka aiheutuu piilotetusta selainympäristöstä. Natiivin käyttäjäkokemuksen saavuttamiseksi kehittäjän tulisi kirjoittaa alustakohtaista koodia, mikä sotii monialustaisten teknologioiden peruseriaatteita vastaan. [65]

Teknologia on hyvä vaihtoehto yrityksille, jotka haluavat saavuttaa mahdollisimman suuren käyttäjäkunnan pienillä resursseilla. Hybriditeknologiat ovat myös silloin vartenotet-

tava ratkaisu, jos käyttöliittymäsuunnittelussa ei tarvitse välittää natiiveista ominaisuuksista.

2.2.3 Ristiinkäännetyt sovellukset

Ristiinkäännetty sovellus ei käytä hyödykseen alustojen omia ohjelmointikieliä. Tästä huolimatta sovellus voidaan silti kääntää täysin natiiviksi ristiinkääntämisen avulla. Koska sovellus käännetään natiiveiksi tiedostoiksi, pystytään näiden teknologioiden avulla käyttämään suoraan oikeita natiiveja komponentteja, jolloin saavutetaan natiivin näköinen ja tuntuinen sovellus. Heikkoutena on se ettei samaa koodipohjaa pysty käyttämään eri alustoilla täysin, sillä eri alustojen komponentit ovat erilaisia, eikä niitä pysty suoraan käyttämään ristiin toisilla alustoilla.

Googlen kehittämä Flutter, johon tämä diplomityö keskittyy, kuuluu tähän teknologiakategoriaan. Flutter kuitenkin poikkeaa normaaleista ristiinkäännetyistä teknologioista siinä ettei se käytä suoraan natiiveja komponentteja, vaan Google on korvannut ne omilla widgeteillään. Heikkoutena tässä tavassa on se, että ohjelmoijan täytyy luottaa Flutterin kehittäjiin, jotta he pitävät widgetit ajantasalla ja natiivien komponenttien näköisinä. Muita tähän kategoriaan kuuluvia teknologioita ovat muun muassa Xamarin ja Xamarin.Forms [88], joiden ohjelmointikielenä toimii C#. [60]

2.2.4 Skriptikielet

Skriptikieltä (engl. native scripting) käyttävät sovellukset ovat sellaisia, joihin on sisällytetty tulkki, jonka tehtävänä on suorittaa koodia käännösaikaisesti ja tehdä kutsuja natiiveihin API-rajapintoihin. Nämä teknologiat käyttävät enimmäkseen JavaScriptiä ohjelmointikielenään. Esimerkkejä teknologioista ovat NativeScript ja React Native.

Markkinoiden johtava teknologia: React Native

React Native on alustariippumaton avoimen lähdekoodin ohjelmistokehitys, joka on Facebookin kehittämä ja se on julkaistu vuonna 2015 [67]. Teknologia pohjautuu suosittuun web-kehitysteknologiaan: Reactiin, jonka ohjelmointikielenä toimii JavaScript.

React Nativen arkkitehtuuri koostuu kolmesta eri osasta: natiiveista komponenteista, JavaScript virtuaalikoneesta, sekä React Native sillasta, kuten kuvassa 2.2 on esitetty. Natiivit komponentit sisältävät sovelluksen natiivin lähdekoodin, joka käyttää hyväkseen natiiveja API-rajapintoja. Javascript virtuaalikoneena käytetään JavascriptCorea, jonka tehtävänä on suorittaa JavaScript-koodi, eli sovelluksen varsinainen logiikka, erillään natiivista koodista. Tarkemmin tämä tapahtuu eri säikeissä, jossa pääsääntö on kääntää natiivia koodia ja piirtää käyttöliittymän, sekä vastaa käyttäjän syötteiden vastaanottamisesta.



Kuva 2.2. *React Nativen arkkitehtuurikuvaus*

Javascript-säie taas ajaa virtuaalikonetta sekä vastaa React Native -sovelluksesta ja varsinaisesta sovelluksen logiikasta. Virtuaalikoneen ja natiivien komponenttien välillä toimii silta, joka toimii viestinviejänä näiden kahden säikeen välillä. Viestit viedään sillalle asynkronisesti, jotta säikeet eivät lukitse toisiensa toimintaa. [66]

Työn myöhemmissä osioissa palataan React Nativeen uudelleen, sillä sitä käytetään vertailukohtana Flutterille.

3 DART

Dart on Googlen kehittämä ohjelmointikieli, jonka ensimmäinen versio julkaistiin vuonna 2011. Kielen taustalla ovat Lars Bak ja Kasper Lund, joiden tavoitteina oli (1) luoda rakenteinen, mutta joustava ohjelmointikieli web-kehittämiseen, (2) kehittää Dartista tutunoloinen ja luonnollinen kieli, joka on helppo oppia, sekä (3) varmistaa hyvä suorituskyky kaikilla moderneilla web-selaimilla ja ympäristöillä. [10] Kieli yhdistää parhaat puolet staattisesti tyyditetyistä kielistä, kuten Java ja C#, sekä dynaamisista kielistä, kuten JavaScript, Python ja Ruby. Dart ja Google luottavat web-komponenttien olevan web-ohjelmoinnin tulevaisuus, jonka takia Dart sisältää komponenttikirjaston. Komponentti on ohjelman osa, mikä on kirjoitettu HTML:n ja Dartin tai JavaScriptin yhdistelmällä ja joka on yleiskäyttöinen erilaisissa projekteissa. Web-komponenttia kutsutaan Dartissa widgeteiksi.

Tämän osion kirjoittamishetkellä Dartin viimeisin julkaistu versio on kesäkuulta 2019 versio 2.4.0 ja sitä käytetäänkin pohjana tälle osiolla. Dart on luokkapohjainen, yksinkertaiseen periytymiseen (engl. single-inheritance) luottava puhdas olioperusteinen ohjelmointikieli staattisella tyyppityksellä. Seuraavissa aliosioissa käydään läpi Dartin tärkeimpiä ominaisuuksia muun muassa Dartin Language Tour -dokumentaation avulla [1].

3.1 Kääntäminen Dartissa

Dartissa on mahdollista käyttää sekä AOT-, että JIT-kääntäjää. AOT-kääntäjä kääntää ohjelman jo asennusvaiheessa, jonka takia asennus kestää pidempään, mutta sovelluksen käynnistäminen on nopeaa. JIT-kääntäjä taas kääntää ohjelman lähdekoodin ajonaikaisesti. Dart käyttää AOT-kääntäjää sovelluksien julkaisemiseen ja JIT-kääntäjää sovelluksien kehitysvaiheessa. JIT-kääntäjä mahdollistaa Flutterin Hot reload ja Hot restart -ominaisuudet, jotka esitellään osiossa 4.

Dartia on myös mahdollista kääntää JavaScriptiksi, mikä mahdollistaa Dartin käytön myös web-kehityksessä. [63]

3.2 Kapselointi

Dartissa on mahdollista asettaa objektit ja metodit joko yksityisiksi (engl. private) tai julkisiksi (engl. public), kuten monissa muissakin kielissä. Dartissa yksityiset tunnisteet aloi-

tetaan merkillä `_`. Yksityiset tunnisteet ovat käytössä vain siinä kirjastossa, jossa ne ovat määritelty.

3.3 Tyypitys

Dartissa on käytössä vahva tyypitys, jossa tyyppejä tarkistetaan sekä käännosaikaisen, että myös ajonaikaisen tarkastelun avulla. Näiden tarkastelujen avulla taataan ettei ohjelma voi missään vaiheessa mennä tilaan, jossa ilmaisu tulkittaisiin arvoksi, joka ei vastaa ilmaisun staattista tyyppiä (engl. sound type system). Dartissa muuttujien tyypittäminen ei ole pakollista, sillä jos kehittäjä ei ole antanut muuttujalla tyyppiä, Dart päättelee sen itse. Jos muuttujasta ei ole tarpeeksi informaatiota, Dart käyttää `dynamic`-tyyppiä. Esimerkissä 3.1 sama muuttuja tyypitetään, sekä kehittäjän että Dartin toimesta. Dart tulkitsee tässä tilanteessa `esityksenTiedot`-muuttujan tyyppiksi `Map<String, Object>`, sillä avain on molemmilla arvoilla merkkijono, mutta arvoilla on eri tyytit `String` ja `int`, joiden yhteinen ylätyyppi on `Object`.

```
1 Map<String, dynamic> esityksenTiedot = {'Esittaja' : 'Aki Artisti',
    'LipunHinta' : 300};
2 // Tyyppi: Map<String, Object>
3 var esityksenTiedot = {'Esittaja' : 'Aki Artisti', 'LipunHinta' : 300};
```

Ohjelma 3.1. Sama muuttuja ensin kehittäjän tyypittämänä ja sen jälkeen Dartin tyypittämänä.

3.3.1 Sisäänrakennetut tyytit

Dartiin on sisäänrakennettu tyyppejä, jotka esitellään tässä kappaleessa. Näitä ovat numeroihin liittyvät tyytit: `num`, `int` ja `double`, totuusarvo `bool`, merkkijonot `String`, listat `list`, kokoelmat `set` ja `Map`, sekä harvemmin käytetyt `Runes` ja `Symbol`.

num, int ja double Numeroita voidaan käsitellä kolmen eri tyytin avulla. `num` on numeroiden perustyyppi, josta muut numerotyytit ovat periytyneet. Se sisältää muutamia perusoperaatioita. `int`-tyypillä määritellään kokonaisluvut ja `double`-tyypillä desimaaleja esittävät luvut. Jos `num`-tyypiselle muuttujalle yritetään antaa arvo, joka ei ole tyyppiä `int` tai `double`, syntyy käännösaikainen virhe [61].

Esimerkissä 3.2 näytetään, miten numerotyyppejä voi määritellä ja huomattavaa onkin, miten Dart muuttaa tarvittaessa kokonaisluvut automaattisesti desimaaleiksi, kuten rivillä 6.

```
1 // Tyyppi: int
2 var x = 1;
3 // Tyyppi: double
```

```

4  var y = 1.1;
5  // Tyyppi: double. Arvo: 1.0.
6  double z = 1;

```

Ohjelma 3.2. Numeroiden esittely

bool Totuusarvoja `true` ja `false` voidaan ilmaista tyyppillä `bool`. Dartin tyyppityksen takia ilmaisia `if`(merkkijono) tai `assert`(merkkijono) ei voi käyttää, vaan arvoja tulee tarkastella eksplisiittisesti kuten esimerkissä 3.3.

```

1  var merkkijono = '';
2  assert(merkkijono.isEmpty) // true

```

Ohjelma 3.3. Esimerkki totuusarvoista

String Dartissa merkkijonot ovat muuttumattomia ja ne tyyppitetään avainsanan `String` avulla. Merkkijonon luonnissa voi käyttää joko yksin- tai kaksinkertaisia lainausmerkkejä. Tässä diplomityössä on valittu käytettäväksi yksinkertaisia. Useamman rivin merkkijonoja on mahdollista luoda käyttämällä kolmea lainausmerkkiä peräkkäin.

Merkkijonon sisään voidaan lisätä muuttujan arvo tai lauseke ilmaisulla `#{lauseke}`. Jos kyseessä on muuttuja, voidaan aaltosulut jättää pois. Dartissa merkki `$` tarkoittaa tällaisen ilmaisun alkamista. Sijoitettavassa muuttujassa ei saa olla `$`-merkkiä. Sijoituksessa evaluoidaan ensin vasemmalta katsottuna ensimmäinen lauseke objektiksi, jonka jälkeen objektille kutsutaan `toString()`-metodia. Viimeisenä tarkastellaan, onko metodin paluuarvo merkkijono. Jos ei, heitetään virhe. Jos on, jatketaan merkkijonossa seuraavaan mahdolliseen muuttujaan. Lopuksi nämä merkkijonot yhdistetään, jolloin saadaan lopullinen merkkijono. [23] Merkkijonojen perusominaisuuksia on esitelty ohjelmassa 3.4.

```

1  var x = 'Esimerkki';
2  var y = 'kahdella';
3  // Kaksi muuttujaa merkkijonon määrittelyssä
4  // Ensin evaluoidaan muuttuja x, sen jälkeen vasta y
5  var z = '$x $y muuttujalla'
6  // Monen rivin merkkijono
7  var monirivinen = '''
8  Usean rivin
9  merkkijono.
10 '''

```

Ohjelma 3.4. Esimerkki merkkijonojen ominaisuuksista

list Listat ovat järjestettyjä kokoelmia objekteista. Listojen indeksointi alkaa numerosta 0. Listojen esittely tapahtuu kuten ohjelmassa 3.5.

```

1  var lista = ['a', 'b', 'c', 'd'];
2  assert(lista.length == 4); // tosi

```

```

3  assert(lista[0] == 'a'); // tosi
4  lista[0] = 'x';
5  assert(lista[0] == 'x'); // tosi

```

Ohjelma 3.5. Esimerkki listoista

set Dartin tyyppi `set` on kokoelma järjestämättömiä objekteja. Tässä tyypissä jokaisen objektin tulee olla arvoltaan uniikki, eli se ei saa sisältää kaksoiskappaleita arvoista. Ohjelmassa 3.6 on esimerkki tällaisen kokoelman käytöstä.

```

1  var hedelmat = {'banaani', 'omena', 'päärynä'};
2  hedelmat.add('ananas');
3  assert(hedelmat.length == 4);
4  // hedelmat.add(2) aiheuttaa virheen, sillä hedelmät on tyyppiä Set<String>

```

Ohjelma 3.6. Esimerkki järjestämättömistä kokoelmista

Map `Map` eli assosiaatiotaulu on tyyppi, jossa aina yksi arvo liittyy yhteen avaimen. Arvot ja avaimet voivat olla eri objekteja. Sama avain ei voi esiintyä useasti kertaan yhdessä kokoelmassa. Ohjelmassa 3.7 on esimerkki `Map`-tyypin käytöstä.

```

1  // Tyyppi: Map<String, List<String>
2  var hedelmat = {
3    'happamat': ['lime', 'sitruuna']
4    'makeat': ['ananas', 'omena']
5  }
6  assert(hedelmat['happamat'] == ['lime', 'sitruuna']); // true
7  assert(hedelmat['kirpeat'] == null); // true

```

Ohjelma 3.7. Esimerkki assosiaatiotaulusta

Tyyppien `Map` ja `set` tyhjät ilmaisut ovat ulkonäöltään samanlaisia. Jos kehittäjä kirjoittaa ilmaisun `var x = {}`, Dart evaluoi sen `Map<dynamic, dynamic>`-tyypinä.

Runes `String`-tyypillä voidaan esittää vain 16-bittisiä Unicode arvoja, jonka takia 32-bittisille on tehty oma tyyppi: `Runes`. Esimerkissä 3.8 näytetään, miten voidaan tulostaa ☹️-merkki.

```

1  Runes hymiö = new Runes('\u{1f60e}');
2  print(new String.fromCharCode(hymiö));

```

Ohjelma 3.8. 32-bittisten Unicode-arvojen esittäminen

Symbol Symbolien avulla voidaan viitata tunnisteeseen. Tämä on kätevää silloin, kun tunnisteen nimi muuttuu, jolloin siihen liitetty symboli kuitenkin säilyy ennallaan.

3.4 Muuttujat

Muuttujat ovat objekteja, jotka instantioidaan luokasta `Object`. Jos muuttujan tyyppiä ei haluta rajoittaa vain yhteen, tulee muuttujalle antaa tyyppi `Object` tai `dynamic`. Dartin dokumentaatio suosii `Object`-tyypin käyttämistä, jos käyttötarkoituksena on hyväksyä mikä tahansa objekti [26]. Jos muuttujaa ei ole alustettu, se saa aina arvon `null`.

Muuttujan voi määrittellä vakioarvoiseksi kahdella eri avainsanalla: `const` ja `final`. Näistä `const`-muuttujat ovat käännösaikaisia eli niiden arvon tulee olla selvillä ennen käännöstä. Tällaisia arvoja ovat muun muassa numerot, merkkijonot tai toiset `const`-muuttujat. Tyypin `final` omaavat muuttujat taas eivät ole käännösaikaisia, eli ne voi alustaa myös myöhemmin. Esimerkissä 3.9 on esitelty vakioarvoisten muuttujien esittelyä.

```

1 // Määritellään muuttuja, joka ottaa arvokseen tyhjän taulukon
2 var muuttuja = const [];
3 // muuttuja ei ole vakioMuuttuja, joten sitä voidaan mutatoida
4 muuttuja = [1,2]
5 const vakioMuuttuja = [];
6 // Koska vakioMuuttuja on vakio, ei sitä voida mutatoida
7 vakioMuuttuja = [1] // virhetilanne

```

Ohjelma 3.9. Vakioarvoisen muuttujan esittely

Dartissa on mahdollista määrittää myös globaaleja muuttujia. Luokille voidaan määrittellä staattisia muuttujia avainsanalla `static`, jolloin muuttuja on käytössä koko luokassa, eikä vain sen yhdessä instanssissa.

3.5 Funktiot

Dartissa myös funktiot ovat objekteja, jonka ansiosta niitä voi antaa parametreina toisille funktioille tai tallettaa muuttujiin. Näitä toimintoja esitellään esimerkissä 3.10.

```

1 // Funktion määrittely
2 void printVihannes(String vihannes) {
3     print (vihannes);
4 }
5 // Sama funktio lambdana
6 void printVihannes(String vihannes) => print(Vihannes);
7
8 var vihannekset = ['porkkana', 'selleri'];
9 // Funktion antaminen parametrina toiselle funktiolle
10 vihannekset.forEach(printVihannes);
11
12 var hedelmat = ['banaani', 'omena'];
13 // Funktion tallentaminen muuttujaan
14 var printHedelma = (hedelma) => '$hedelma on hedelmä.';

```

```
15  assert(printHedelmä('Omena') == 'Omena on hedelmä');
```

Ohjelma 3.10. Esimerkki funktioiden esittely

Funktiolle annettavat parametrit ovat joko pakollisia tai vapaaehtoisia. Pakolliset parametrit erotetaan vapaaehtoisista avainsanalla `@required`. Vapaaehtoiset parametrit voidaan jakaa kahteen eri luokkaan: nimetyt ja paikkasidonnaiset parametrit. Nimetyt vapaaehtoiset parametrit määritellään aaltosulkujen sisällä, kuten esimerkin 3.11 rivillä 1. Funktiota kutsuessa ei ole väliä, missä järjestyksessä nimetyt parametrit ovat (rivi 2, ohjelma 3.11). Paikkasidonnaiset vapaaehtoiset parametrit sidotaan hakasulkeiden sisään ja ne tulee antaa funktiokutsussa täysin samassa järjestyksessä kuin funktion määrittelyssä. Paikkasidonnaisten parametrien käyttöä on esitelty esimerkin 3.11 riveillä 4-6. Jos vapaaehtoista arvoa ei anneta, se saa arvokseen `null`.

Parametreille voi myös antaa oletusarvoja. Esimerkissä 3.11 on rivillä 1 annettu nimetyille parametreille oletusarvo ja rivillä 4 on annettua oletusarvo paikkasidonnaiselle paramet-rille.

```
1  void nimetytParametrit({bool arvo1 = true, bool arvo2}) {...}
2  // Funktion kutsuminen
3  nimetytParametrit(arvo2: false, arvo1: true);
4
5  void paikkasidonnaisetParametrit(String arvo1, bool arvo2, [bool arvo3 =
6     false]) {...}
7  // Funktion kutsuminen ilman kolmatta parametria
8  paikkasidonnaisetParametrit('Esimerkki', false);
9  // Funktion kutsuminen kaikkien parametrien kanssa
10 paikkasidonnaisetParametrit('Esimerkki', false, true);
```

Ohjelma 3.11. Esimerkki funktion parametreista

Funktiot palauttavat aina arvon. Jos paluuarvoa ei ole määritely, se on aina `null`.

Funktioiden asynkronisuus hoidetaan Dartissa tyyppin `Future` avulla. `Future` on lupaus siitä, että funktion lopputulos toimitetaan tulevaisuudessa. `Future`:n rajapintojen käyttö voi olla vaikealukuista, jonka takia joissain tilanteissa kannattaa suosia muistakin ohjelmointikielistä tuttua `async-await`-ilmaisua. Asynkronisissa funktioissa avainsanalla `await` voidaan kutsua sellaisia funktioita, jotka halutaan suorittaa loppuun ennen seuraavaan ilmaisuun siirtymistä. Asynkroniset funktiot palauttavat `Future`-tyyppisen arvon. Esimerkissä 3.12 esitellään asynkronisia funktioita.

```
1  // Asynkroninen funktio Future-tyypin avulla
2  asynkroninenFunktioFuturella() {
3    // ...
4    etsiMuuttuja().then((muuttuja) {
5      return tulostaMuuttuja(muuttuja);
6    }).then(tulostaExitCode);
7  }
8
```

```

9 // Asynkroninen funktio asyn-await-ilmaisun avulla.
10 asynkroninenFunktioAsyncAwait() async {
11 // ...
12 var muuttuja = await etsiMuuttuja();
13 var exitCode = await tulostaMuuttuja(muuttuja);
14 await tulostaExitCode(exitCode);
15 }

```

Ohjelma 3.12. Esimerkki asynkronisuudesta

3.6 Suoritusta kontrolloivat lausekkeet

Dartissa suoritusta voidaan kontrolloida muistakin ohjelmointikielistä tutuilla lausekkeilla, kuten konditionaalisilla lauseilla ja erilaisilla silmukoilla. Tässä osiossa esitellään näitä avainsanoja.

If ja else Dartissa on käytössä muistakin kielistä tuttu ilmaisu `if-else if-else` konditionaalisia lauseita varten. Konditionaalisissa lauseissa voi käyttää vain tyypiltään `bool` olevia arvoja. Dart sisältää Javascriptin tapaan myös yhden rivin syntaksin `ehto ? ilmaisu1 : ilmaisu2`. Dartissa on myös käytössä syntaksi `ilmaisu1 ?? ilmaisu2`, jonka avulla voidaan palauttaa eri arvoja, jos toinen annetuista arvoista on `null`.

Silmukat Dartissa on mahdollista käyttää joko `for`- tai `while`-silmukkaa. `while`-silmukka voidaan laajentaa `do-while`-silmukaksi, jossa silmukan ehto evaluoidaan vasta silmukan suorituksen jälkeen. Jos silmukan suoritus halutaan lopettaa, voidaan käyttää avainsanaa `break`. Jos taas silmukan tietty iteraatio halutaan keskeyttää, voidaan avainsanalla `continue` hypätä suoraan seuraavaan iteraatioon. Joitain objekteja, kuten taulukkoa, on mahdollista iteroida läpi myös `forEach()`-metodilla.

Switch ja case Avainsanojen `switch` ja `case` avulla voidaan objekteja vertailla keskenään ja vaihtaa ohjelman suoritustapaa riippuen vertailun tuloksesta.

3.7 Assert

`assert(ehto, vapaaehtoinenViesti)` -lauseketta voidaan käyttää hyödyksi virheiden paikallistamisessa ohjelmiston kehitysvaiheessa. Ilmaus keskeyttää ohjelman suorituksen ja heittää poikkeuksen, jos `ehto` ei toteudu. Jos ehto toteutuu, ohjelman suoritus jatkuu normaalisti. `assert()` -syntaksia on käytetty jo useassa esimerkissä, muun muassa ohjelmassa 3.7.

3.8 Poikkeukset

Poikkeuksiin liittyy kolme avainsanaa: `throw`, `catch` ja `finally`. Ensimmäisen avulla poikkeus heitetään myöhemmin kiinnitettäväksi, toisen avulla tämä poikkeus napataan kiinni ja kolmannen avulla suoritetaan koodia piittaamatta siitä, heitettiinkö poikkeusta tai napattiinko sitä.

3.9 Luokat

Dartissa kaikki objektit ovat luokkien ilmentymiä ja kaikki luokat polveutuvat `Object`-tyypistä. Jokaisella luokalla voi olla maksimissaan yksi superluokka, mutta tämän lisäksi eri luokkien rakenteita voidaan käyttää hyväksi useaan kertaan eri luokissa. Näitä ominaisuuksia käsitellään myöhemmissä aliosioissa.

Luokat koostuvat neljästä osasta (ohjelma 3.13): ilmentymämuuttujat (engl. instance variables), rakentajat, metodit, sekä getter ja setter-metodit.

```

1  class EsimerkkiLuokka {
2      // Ilmentymämuuttujat
3      // Rakentajat
4      // Metodit
5      // Getter ja setter -funktiot
6  }
```

Ohjelma 3.13. Ilmaisu luokan esittelylle.

Ilmentymämuuttujat Ilmentymämuuttujat ovat luokille ominaisia ja ne määritellään heti sen alussa. Jos niitä ei alusteta, ne saavat arvon `null`. Kaikille ilmentymämuuttujille generoidaan automaattisesti getter-metodi ja niille, joita ei ole määritelty `final`-avainsanalla, generoidaan myös setter-metodi. Esimerkissä 3.14 esitellään ilmentymämuuttujien käyttöä. Rivillä 9 on käytetty setter-metodia ja riveillä 11 ja 13 getter-metodia.

```

1  class Tapahtuma {
2      DateTime ajankohta;
3      double hinta;
4      String nimi;
5  }
6
7  void main() {
8      var tapahtuma = Tapahtuma();
9      // x-muuttujan setter-metodi
10     tapahtuma.hinta = 50.0;
11     // x-muuttujan getter-metodi
12     assert(tapahtuma.hinta == 50.0); // true
13     // y-muuttujan getter-metodi
```

```

14     assert(tapahtuma.nimi == null); // true
15 }

```

Ohjelma 3.14. Ilmentymämuuttujien käyttö

Rakentajat Rakentaja alustaa luokan annettujen parametrien avulla. Rakentajat voivat olla joko nimeämättömiä tai nimettyjä. Nimeämättömät rakentajat ovat nimeltään samoja kuin luokan nimi ja nimettyjen luokkien ilmaisu on `luokanNimi.rakentajanNimi`. Jos luokalle ei erikseen määritellä rakentajaa, luokalle generoidaan automaattisesti sellainen, joka ei tee mitään.

```

1  class Tapahtuma {
2      DateTime ajankohta;
3      double hinta;
4      String nimi;
5
6      Tapahtuma.tamaPaiva(double this.hinta, String this.nimi) {
7          ajankohta = DateTime.now();
8      }
9  }
10
11 class Urheilutapahtuma extends Tapahtuma {
12     Urheilutapahtuma.tamaPaiva(double hinta, String nimi) :
13         super.tamaPaiva(hinta, nimi);
14 }
15
16 void main() {
17     var tapahtuma = Tapahtuma.tamaPaiva(500.00, 'Esiintyjä Esa');
18     var urheilutapahtuma = Urheilutapahtuma.tamaPaiva(35.00, 'Ilves-Tappara');
19 }

```

Ohjelma 3.15. Luokan rakentajien määrittely ja käyttö

Luokkien rakentajat eivät periydy aliluokalle, joten jos samaa rakentajaa halutaan käyttää, tulee aliluokan rakentaja määrittää osoittamaan yläluokan rakentajaan, kuten esimerkin 3.15 rivillä 12.

Rakentajat luovat normaalisti luokasta aina uuden instanssin. Tämä voidaan kuitenkin ohittaa avainsanalla `factory`, joka hyödyntää välimuistista löytyvää instanssia.

Metodit Metodeihin kuuluvat getter- ja setter-funktioiden lisäksi myös ilmentymäfunktiot. Getter ja setter-metodit luodaan jokaiselle muuttujalle automaattisesti ja niitä voi luoda omien tarpeiden mukaan lisää. Ilmentymäfunktiot pääsevät käsiksi avainsanaan `this`.

3.9.1 Abstraktit luokat

Abstrakteja luokkia ei voi alustaa, joten niitä on kätevä käyttää esimerkiksi rajapintojen määrittelyissä. Abstraktit luokat käyttävät usein abstrakteja metodeja, joissa toteutus jätetään muille luokille.

3.9.2 Periytyminen ja mixinit

Aliluokkia on mahdollista laajentaa yläluokista avainsanalla `extends`. Aliluokan sisällä voidaan viitata yläluokkaan avainsanalla `super`. Ohjelmassa 3.15 on riveillä 11-13 esimerkki aliluokasta. Aliluokan ei täydy hyödyntää kaikkia yläluokan ominaisuuksia. Jos jokin ominaisuutta halutaan aliluokassa muokata, se voidaan määritellä uudestaan avainsanalla `@override`.

Dartissa on periyttämisen lisäksi mahdollista myös "mixata" eri luokkia aliluokkaan, jolloin saadaan aikaan `mixin`. `mixin` määritellään muuten samalla tavalla kuin luokat, mutta niistä jätetään rakentaja-funktiot pois (ohjelma 3.16).

```

1  mixin Lippu {
2      bool vipLippu = false;
3      double hinta = 50.0;
4
5      void tulostaLippu() {
6          if(lipputyypä == 'vip') {
7              print('VIP-lippu, $hinta euroa.');
```

Ohjelma 3.16. Esimerkki Mixinin käytöstä

3.10 Geneeriset kokoelmaluokat

Dartissa on mahdollista määrittää muuttujan tyyppi myös geneeriseksi, jolloin muuttujaa ei tarvitse sitoa tiettyyn tyyppiin. Jos kehittäjä haluaisi luoda saman luokan eri tyyppi-sille parametreille, geneeristen tyyppien avulla ei olisi tarvetta luoda kahta eri luokkaa. Esimerkin 3.17 kaksi ensimmäistä luokkaa voidaan yhdistää viimeisen luokan avulla.

```

1  abstract class Hinta {
2      String getHintaByKey(String key);
3      void setHintaByKey(String key, String value);
4  }
5
6  abstract class Hinnasto {
7      Object getHinnastoByKey(String key);
8      void setHinnastoByKey(String key, Object value);
9  }
10 // Luokka on määritelty yleisen tyyppin T avulla.
11 abstract class Hinnat<T> {
12     T getByKey(String key);
13     void setByKey(String key, T value);
14 }
```

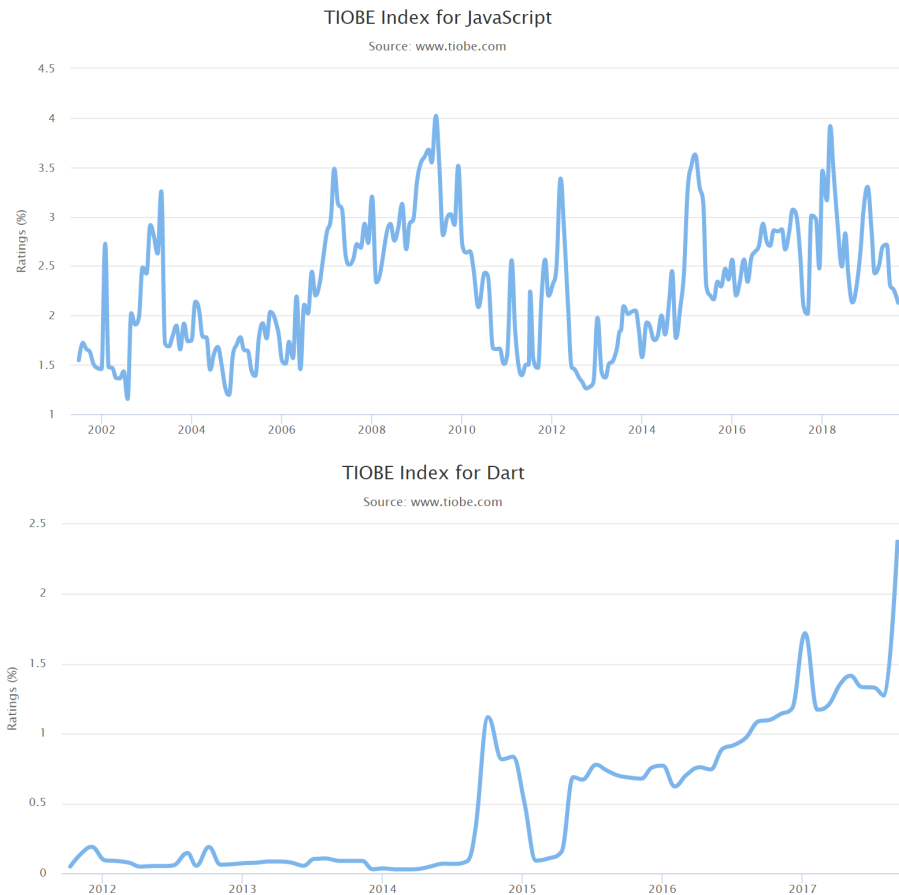
Ohjelma 3.17. Esimerkki geneeristen tyyppien käytöstä

Kelpavaa tyyppiä voidaan myös rajoittaa `extends`-ilmaisun avulla. Esimerkin tapauksessa luokkaa voitaisiin rajoittaa muokkaamalla riviä 11 muotoon `abstract class Hinnat<T extends RajoittavaLuokka> {}`.

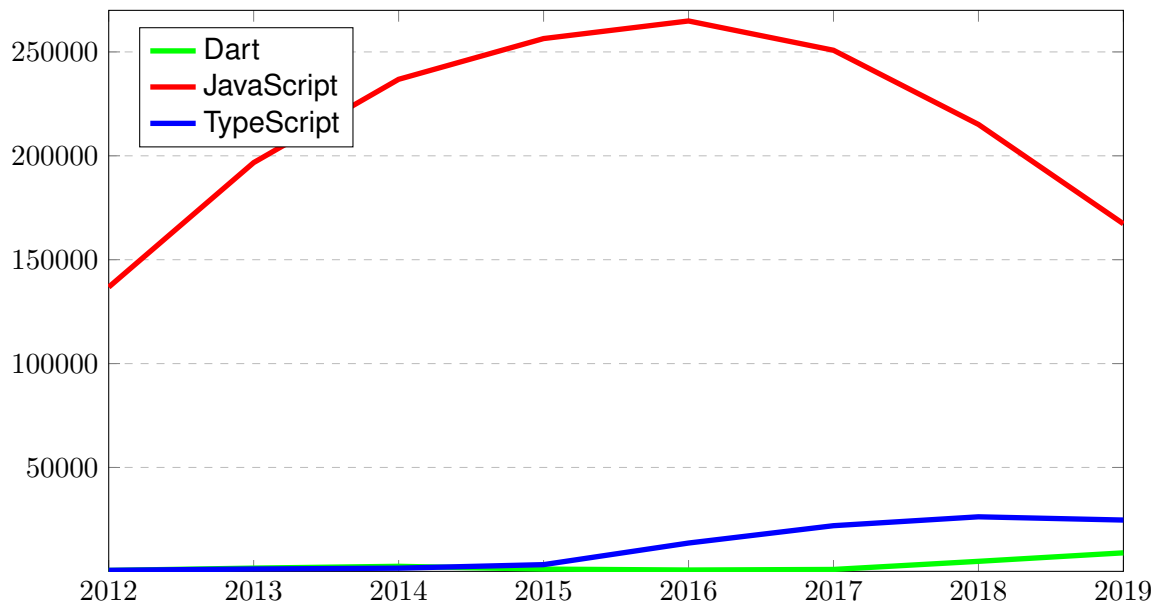
3.11 Vertailu kilpailijoihin

Dart muistuttaa ilmaisultaan paljon JavaScriptiä. Suurin ero ohjelmointikielten välillä on niiden suosio. Suosiota voidaan evaluoida TIOBE-indeksin avulla, joka arvioi teknologian suosiota muun muassa osaavien kehittäjien, kurssien, sekä hakukoneiden tuloksien määrän avulla. Kuvassa 3.1 on kuvaajat sekä JavaScriptin, että Dartin TIOBE-indeksille. Dart on TIOBE-tilastossa 26, JavaScript on sijalla 7 ja TypeScript 37. Kuvassa 3.2 kuvataan JavaScriptin, TypeScriptin, sekä Dartin vuosittaista kysymyksen määrää StackOverflow:ssa ¹. Selvästi vähiten kysymyksiä on tehty Dartista. StackOverflow kartoittaa vuosittain eri teknologioiden suosiota Developer Survey Results-tutkimuksessa. Vuoden 2019 tutkimuksessa JavaScript luokiteltiin suosituimmaksi ohjelmointikieleksi 67,8% turvin. TypeScript oli 10. 21,2%:lla ja Dart 22. 1,9%:lla. Kyselyyn vastasi noin 87 000 StackOverflow:n käyttäjää. [25] Näiden seikkojen perusteella, Javascript on selkeästi suosituimpi ohjelmointikieli kuin Dart.

¹Kuvaajassa esitetään kysymykset, joilla on tag-merkintänä dart, javascript tai typescript. Kysymykset ovat jaoteltu eri vuosille niiden luontipäivämäärän mukaan. Data on haettu <https://data.stackexchange.com/-sivuston kautta>.



Kuva 3.1. TIOBE indeksit JavaScriptille ja Dartille.



Kuva 3.2. Dartin, JavaScriptin ja TypeScriptin StackOverflow:n kysymyksien määrien kehitys vuosina 2012-2019.

Dartin ehdoton vahvuus sen kilpailijoihin nähden on nopeus. Vuonna 2010 tehdyn tutkimuksen [24] mukaan, kun Dartia ajetaan sen omalla virtuaalikoneella, se on noin kaksi kertaa nopeampi kuin JavaScript. Nopeuteen liittyy myös Dartin tapa käyttää sekä AOT-,

että JIT-kääntäjää sovelluksen kääntämisessä, jota esiteltiin jo aiemmin tässä osiossa.

JavaScriptin avulla on mahdollista kehittää sekä selain-, että palvelinpuolelle ohjelmia, minkä ansiosta kehittäjän ei tarvitse välttämättä osata kuin yhtä ohjelmointikieltä. Dartin dokumentaatiossa taas ei ole mainintaa kielen sopivuudesta palvelinpuolelle. JavaScriptiä käytetään selainpuolella suurilta osin web- ja mobiilikehitykseen erilaisten ohjelmistokehyksien kautta. Näitä ohjelmistokehyksiä ovat muun muassa React, Vue ja Angular. Mobiilikehityksessä JavaScriptiä käyttävät ohjelmistokehyksistä muun muassa React Native ja Angular. [17] Dartin ohjelmistokehykset taas rajoittuvat lähes täysin web-kehityksessä AngularDartiin ja Flutteriin, sekä mobiilikehityksessä Flutteriin [63].

JavaScriptissä mielipiteitä jakaa vahvan tyyppityksen puuttuminen. Tyyppityksen avulla on usein helpompaa löytää virheitä sovelluksesta, kun tiedetään odottaa mitä tyyppiä minäkään muuttujan pitäisi vastata. JavaScriptin pohjalta on rakennettu ohjelmointikieli TypeScript, joka ei sinänsä ole uusi ohjelmointikieli, vaan se tarjoaa JavaScriptiin uusia ominaisuuksia, kuten tyyppityksen. Dartissa käytetään vahvaa tyyppitystä.

4 FLUTTER

Flutter on Googlen kehittämä alustariippumaton sovelluskehitysteknologia, joka on saanut inspiraationsa Reactista [34] ja jonka ohjelmointikielenä toimii Dart.

4.1 Flutterin arkkitehtuuri

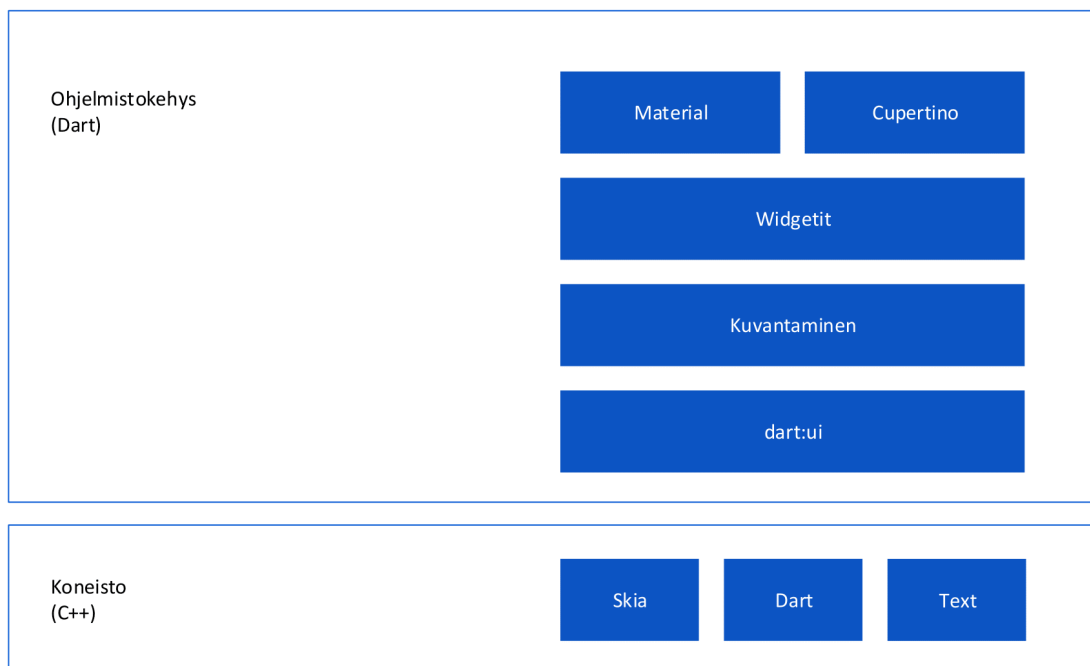
Flutterin arkkitehtuuri voidaan jakaa ohjelmistokehityksen ja Flutterin koneiston arkkitehtuuriin, kuten kuvassa 4.1 on esitetty. Koneisto koostuu Skiasta eli 2D grafiikoiden kuvantamiskirjastosta, Dartin virtuaalikoneesta, sekä Text-moottorista, joka vastaa tekstin kuvantamisesta.

Ohjelmistokehitys on rakennettu kerroksittain eri kirjastoista, jotka ovat riippuvaisia aina alemman abstraktiotason kirjastosta. Kirjastoja ovat `dart:ui`-, kuvantamis-, widget-, material- ja cupertino- kirjasto. Jokainen näistä kerroksista on kehittäjän ulottuvilla jatkuvasti, eli kehittäjiltä ei ole piilotettu mitään ja he voivat missä tahansa tilanteessa kutsua suoraan esimerkiksi `dart:ui`-kirjaston funktioita. Tämä helpottaa omien widgettien luomista, sillä kehittäjä voi käyttää samoja komponentteja avukseen kuin Flutterin kehittäjätkin.

Dart:ui-kirjasto Flutterin ensimmäisellä abstraktiotasolla on `dart:ui`-kirjasto, jonka tarkoituksena on kommunikoida Flutterin koneiston kanssa [47]. Jo pelkästään tämän tason avulla olisi mahdollista kirjoittaa kokonaisia sovelluksia, mikä ei kuitenkaan ole järkevää, sillä kehittäjä joutuisi laskemaan jokaisen komponentin koordinaatit itse ja myös pitämään niistä kirjaa.

Kuvantamiskirjasto Seuraava abstraktiotaso on kuvantamiskirjasto, joka hoitaa rasakat matemaattiset operaatiot `RenderObject`-objektin avulla. Tasolla luodaan Flutterin kuvantamispuu, joka esitellään tarkemmin myöhemmissä osioissa. Tämän kirjaston tuottamat laskutoimitukset ovat ohjelman suorituskyvyn kannalta kalliita, mistä johtuen niitä tallennetaan välimuistiin algoritmien avulla.

Widget-kirjasto Widget-kirjasto sisältää useita UI-komponentteja, jotka ovat täysin käyttökelpoisia sellaisinaan. Tämän tason komponentit ovat yleensä niitä, joita yhdistelmällä kehittäjät luovat omat komponenttinsa.



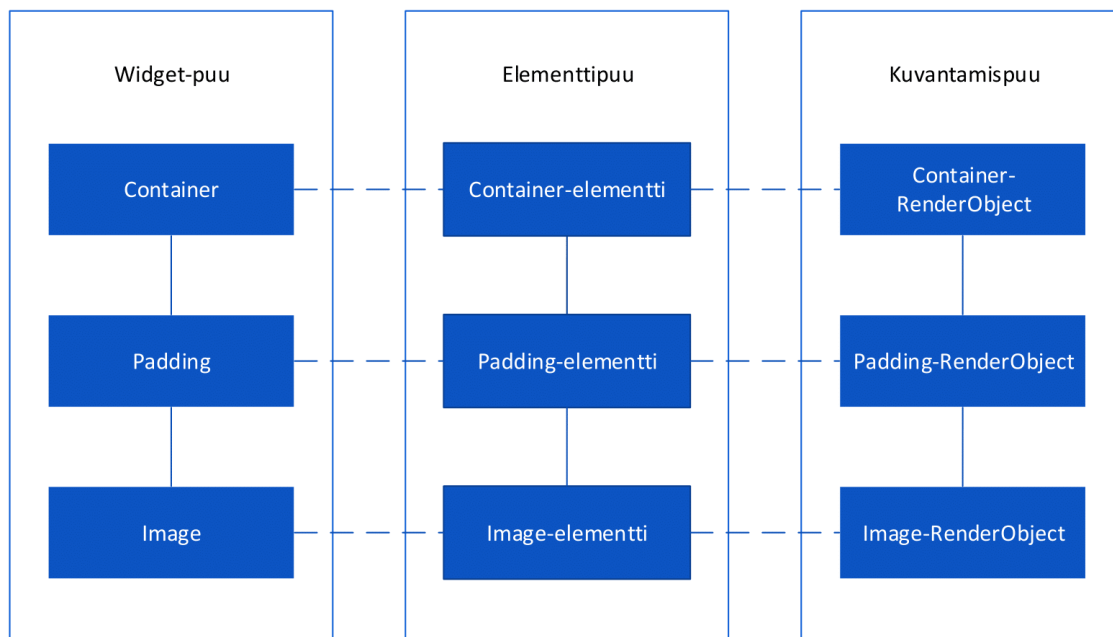
Kuva 4.1. Flutterin arkkitehtuurikuvaus

Material ja Cupertino -kirjastot Viimeinen abstraktiotaso on Material & Cupertino -kirjastot. Tällä tasolla widgetit ovat tyylitelty eri alustoille sopiviksi. Material-kirjaston widgetien tyyli on Android-sovelluksille tyypillinen ja Cupertino-kirjaston widgetit sisältävät komponentteja, joiden tyylit ovat iOS-sovelluksille tyypillisiä.

4.2 Flutterin puurakenne

Flutterissa widgetin kuvantaminen alkaa widget-puun luonnilla. Tämän jälkeen luodaan toinen puu, elementtipuu, johon luodaan jokaista widgetiä vastaava elementti `createElement()`-metodin avulla. Elementtipuun jälkeen luodaan vielä kolmas puu: kuvantamispuu, joka sisältää jokaista elementtiä vastaavan `RenderObject`-objektin. Nämä objektit luodaan elementeistä `createRenderObject()`-metodin avulla. Nyt jokaisella elementillä on referenssi sekä vastaavaan widgetiin, että `RenderObject`-objektiin. Kehittäjän ei tarvitse suoraan kommunikoida kuvantamispuun kanssa, vaan widgetit tekevät sen elementtien kautta. Tätä puurakennetta on havainnollistettu kuvassa 4.2.

`RenderObject`-objektit sisältävät logiikan widgetin kuvantamiseen. Näiden objektien luominen on kallis operaatio, jonka takia niille pyritään soveltamaan säilyvän tilan kuvantamista (engl. retained mode rendering). Tämä onnistuu elementtien avulla, jotka pitävät kirjaa widgetien tilasta, jos niillä sellainen on, sekä kertovat, onko elementtiin viitaava widget päivittynyt vertaamalla sitä `RenderObject`-objektiin. Jos widgetin tyyppi pysyy samana, Flutter ei luo uutta `RenderObject`-objektia, vaan se vain päivittää sen. Elementtipuuta käydään läpi ylhäältä alaspäin ja muuttumattomat elementit hypätään yli.



Kuva 4.2. Flutterin puurakenne

Flutterissa widgetit ovat muuttumattomia, joten jos jotain widgetin ominaisuutta muutetaan, tulee koko widget-puu rakentaa uudestaan. Kun widget päivittyy, vertaa se elementtiään, jos tyypit ovat eri, poistetaan sekä elementti, että `RenderObject` ja kaikki elementit ja `RenderObject`-objektit kyseisen elementin alapuolella. Jos tyyppi on sama, päivitetään `RenderObject`.

4.3 Widgetit Flutterin perustana

Flutterissa komponentteja kutsutaan widgeteiksi. Widgetejä ovat niin nappulat kuin asetteluun liittyvät komponentit. Flutterissa applikaatio luodaan asettelemalla widgetejä toistensa sisään, jolloin alemman tason widgetit perivät ylemmän tason widgetin ominaisuudet. Widgetit voidaan jakaa tilallisiin, tilattomiin ja periytyviin.

4.3.1 Tilattomat widgetit

Tilattomat widgetit määritellään kuten esimerkissä 4.1. Näillä widgeteillä ei nimensä mukaisesti ole tilaa, eli ne eivät voi muuttua dynaamisesti esimerkiksi käyttäjän painalluksesta. Tilattoman widgetin `build`-metodia kutsutaan yleensä kolmessa eri tilanteessa: widget lisätään widget-puuhun, widgetin isäwidget muuttuu ja kun periytyvä widget, josta widget riippuu, muuttuu. [76] `build`-metodin kutsuminen järkevästi on tärkeässä osassa ohjelman suorituskyvyn optimointia.

```

1 class Kuva extends StatelessWidget {
2   Kuva({@required this.kuva});

```

```

3
4   final ImageProvider kuva;
5
6   @override
7   Widget build(BuildContext context) {
8     return Container(
9       child: Padding (
10        padding: EdgeInsets.all(1),
11        child: Image(image: kuva),
12      ),
13    );
14  }
15 }

```

Ohjelma 4.1. Esimerkki tilattomasta widgetistä

4.3.2 Tilalliset widgetit

Tilalliset widgetit voivat muuttua dynaamisesti niiden eliniän aikana. Näille widgeteille määritellään tila, jota voi muuttaa komennon `setState` avulla. Tällöin widget rakennetaan uudestaan ja sen ulkonäkö voi muuttua.

Tilalliset widgetit ovat silti itsessään muuttumattomia ja ne säilyttävät muuttuvan tilansa erillisissä `State`-objekteissa, jotka luodaan widgetistä `createState`-metodilla, kuten esimerkissä 4.2. Metodia kutsutaan aina, kun tilallinen widget luodaan, jolloin voidaan olla varmoja siitä, että tila on oletusarvoinen. Jos kehittäjä on lisännyt widgetille globaalin avaimen (engl. global key) ja widgetiä siirretään widget-puussa, widgetin tila säilyy. [75] Avaimista kerrotaan lisää myöhemmissä osioissa.

```

1   class TilallinenWidget extends StatefulWidget {
2     // Widgetin tila konfiguroidaan tässä luokassa. Luokka sisältää muuttujat,
3     // jotka
4     // annetaan isäkomponentilta ja joita käytetään tilan rakennusmetodissa.
5     // Nämä kentät tulee aina merkitä final-avainsanalla.
6     @override
7     _TilallinenWidget createState() => _TilallinenWidget();
8   }
9
10  class _TilallinenWidget extends State<TilallinenWidget> {
11    int _summa = 0;
12
13    void lisääYksi() {
14      setState(() {
15        // setState-metodilla kerrotaan, että tilaa on muutettu,
16        // jolloin rakentajaa kutsutaan uudestaan.
17        // Jos _summa muutetaan ilman setState-metodia, ei
18        // rakentajaa kutsuta uudestaan ja mikään ei näytä

```

```

18         // muuttuneen.
19         _summa++;
20     });
21 }
22
23 @override
24 Widget build(BuildContext context) {
25     // Tätä metodia kutsutaan aina, kun setState on kutsuttu.
26     return Row(
27         children: <Widget>[
28             RaisedButton(
29                 onPressed: _lisaaYksi,
30                 child: Text('Lisää yksi'),
31             ),
32             Text('Summa: $_summa'),
33         ],
34     );
35 }
36 }

```

Ohjelma 4.2. Esimerkki tilallisesta widgetistä

4.3.3 Periytyvät widgetit

Periytyvät widgetit voivat välittää tilansa aliwidgeteilleen, jolloin tiedon välitys widgetien välillä helpottuu. Esimerkissä 4.3 on esitelty periytyvän widgetin käyttöä. [50]

```

1  class LipunHinta extends InheritedWidget {
2      final int hinta;
3      LipunHinta({this.hinta, Widget aliWidget}): super(child: aliWidget);
4      // Tämä metodi päättää, milloin widgetistä riippuvat widgetit tulee päivittää
5      bool updateShouldNotify(LipunHinta vanhaWidget) =>
6          vanhaWidget.hinta != hinta;
7      // of-metodin avulla, widgetin dataan päästään käsiksi helpommin
8      static LipunHinta of(BuildContext context) =>
9          context.inheritFromWidgetOfExactType(LipunHinta) as LipunHinta;
10 }
11
12 class EsimerkkiWidget extends StatelessWidget {
13     @override
14     Widget build(BuildContext context) {
15         final hinta = LipunHinta.of(context).hinta;
16         // ...
17     }
18 }

```

Ohjelma 4.3. Esimerkki periytyvästä widgetistä

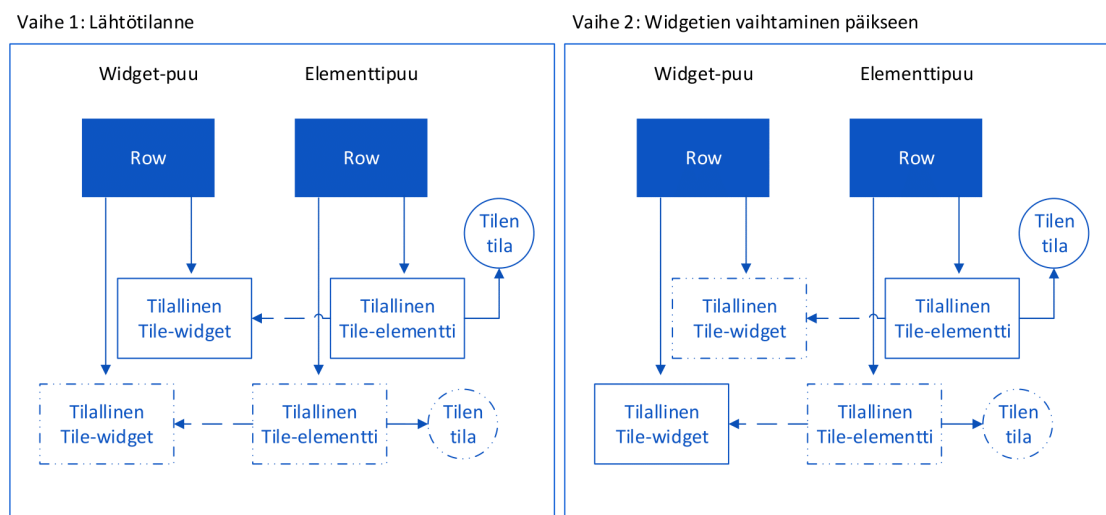
4.3.4 Avaimet

Flutterissa avaimia (engl. keys) voidaan antaa lähes kaikille widgeteille. Niiden tarkoituksena on erottaa kaksi, tyypiltään samaa, widgetiä toisistaan. Tämä tulee erityisen hyödylliseksi silloin, kun näitä widgetejä lisätään, poistetaan tai järjestellään uudestaan. Tilattomille widgeteille ei voi asettaa avaimia. [52]

Jo aiemmin esitelty elementtipuu sisältää informaation kyseisen widgetin tyypistä, sekä sen lapsikomponenteista. Elementtipuulla on myös tieto widgetin tilasta. Kun elementtipuu vertaa itseään widget-puuhun, se tarkistaa elementin ja widgetin tyypit, sekä mahdolliset avaimet, ja vertaa ovatko ne samoja.

Kuvissa 4.3 ja 4.4 on kuvattu avainten hyötyä hyvin yksinkertaisessa tilanteessa, jossa `Row`-widgetin sisällä on kaksi `Tile`-widgetiä, jotka vaihtavat paikkaa napin painalluksesta. Ensimmäisessä kuvasarjassa ei ole käytetty avaimia, mikä aiheuttaa ongelmia widgetien vaihtumisessa. Toisessa kuvasarjassa tämä on korjattu käyttämällä avaimia. Liitteessä A on esitelty sovelluksen toteutus molemmilla tavoilla.

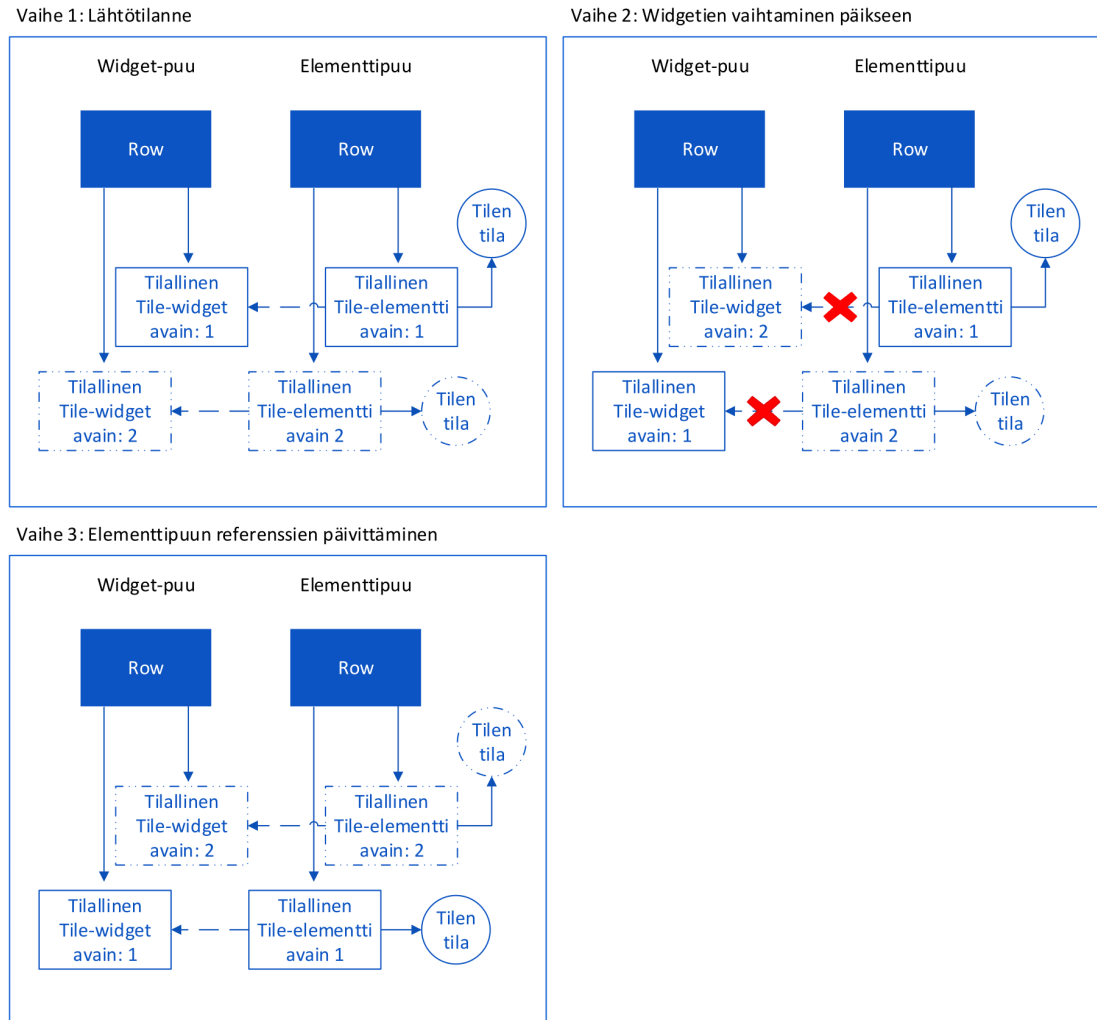
Ensimmäisessä kuvasarjassa 4.3 esitetään miten tilallinen widget, jolle ei ole määritelty avainta toimii esimerkin tilanteessa. Ohjelman käynnistyessä luodaan widget-puu, johon asetetaan, sekä `Row`-widget, että kaksi `Tile`-widgetiä. Widget-puun luonnin jälkeen luodaan elementtipuu, johon lisätään widget-puun widgetejä vastaavat elementit. Lopuksi elementtipuusta luodaan viittaukset widget-puuhun, sekä widgetien tiloihin, kuten 4.3 vaiheessa 1 on esitetty. Kun nappia painetaan, muuttuu tilanne vastaamaan kuvan vaihetta 2. Koska elementtipuu vertaa vain widgetin tyyppiä elementin tyyppiin, eivät elementit, eikä niiden tilat, vaihda paikkaa. Tästä johtuen myöskään käyttöliittymällä ei näy muutosta.



Kuva 4.3. Tilallisuuden widgetien muokkaaminen ilman avaimia

Kolmannessa kuvasarjassa 4.4 widgeteille on lisätty avaimet. Vaiheessa 2 elementit tarkistavat ensin ovatko widgetit samaa tyyppiä. Kun tämä tarkastelu onnistuu, vertaavat elementit widgetin avainta omaansa. Jos avaimet eivät vastaa toisiaan, viittaus elemen-

tin ja widgetin välillä poistetaan, kuten kuvasarjan vaiheessa 2 tapahtuu. Lopuksi, Flutter etsii elementeistä, jotka eivät viittaa widgetiin, sopivia avain-tyyppi -pareja, jotka voitaisiin linkittää widgeteihin. Jos tällainen elementti löytyy, viittaus widgetiin luodaan, kuten vaiheessa 3 tapahtuu. Avaimen ansiosta käyttöliittymällä tapahtuu widgetien paikkojen vaihto.



Kuva 4.4. Tilallisuuden widgetien muokkaaminen avaimien kanssa

Avaimet tulisi sijoittaa niin alas kuin vain on mahdollista, sillä mitä alempana avaimet ovat widget-puussa, sitä vähemmän elementtejä on vertailtavana. Mahdollisia avaintyyppejä on neljä: `ValueKey`, `ObjectKey`, `UniqueKey` ja `GlobalKey`. Näistä ensimmäiset kolme ovat lokaaleja ja viimeinen on globaali. Jokaisen avaimen tulee olla uniikki sen määrittelytasolla.

4.4 Hot Reload ja Hot Restart

Flutterissa on käytössä *Hot Reload*, joka mahdollistaa nopeiden muutosten tekemisen ja niiden lopputuloksen applikaatiossa hyvin nopeasti. Kun käyttäjä haluaa päivittää appli-

kaation Hot Reload -ominaisuudella, käännetään uudelleen vain ne kirjastot joissa on tapahtunut muutoksia, sovelluksen pääkirjasto ja ne kirjastot, jotka johtavat pääkirjastosta muuttuneeseen kirjastoon. Tämän jälkeen muuttuneet lähdekoodit lähetetään Dartin virtuaalikoneelle, jonka jälkeen se lataa kaikki kirjastot, joiden koodi on muuttunut. Luokat päivitetään ja Flutter rakentaa uudelleen widget-puun ja kuvantaa sovelluksen, jonka jälkeen muutokset ovat nopeasti ohjelmoijan näkyvillä. Tapauksissa joissa Hot Reload ei riitä, kuten silloin kun muokataan `initState()`-metodia, kehittäjä voi käyttää Flutterin *Hot Restart*-toimintoa.

4.5 Vertailu React Native -ohjelmistokehykseen

Tässä osiossa vertaillaan Flutteria ja React Nativea. Vertailuun on valittu React Native, koska se on yksi suosituimmista alustariippumattomista teknologioista tällä hetkellä [25].

4.5.1 Arkkitehtuuri ja suorituskyky

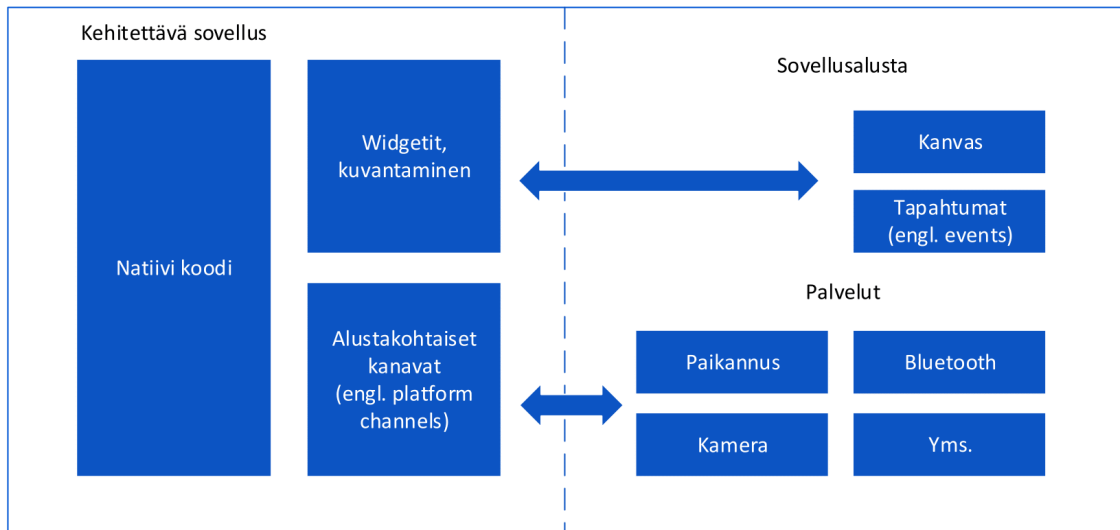
Suurin ero Flutterin ja React Nativen arkkitehtuureissa on widgettien käyttö. React Native kääntää kehittäjän kirjoittaman JavaScript-koodin React Native-sillan avulla natiiveiksi komponenteiksi, jotka sen jälkeen käännetään vielä natiiviksi koodiksi. Flutter sen sijaan käyttää suoraan omia widgetejään, jotka käännetään suoraan natiiviksi lähdekoodiksi. Flutterin avulla voidaan kontrolloida näytön jokaista pikseliä, mikä ei ole mahdollista React Nativessa sen käyttämän sillan johdosta. Flutterin arkkitehtuurin ja React Nativesta tutun sillan puutteesta johtuen, Flutter on suorituskyvyltään tehokkaampi kuin React Native. [35] Kuvassa 4.5 on esitelty Flutterin ja React Nativen arkkitehtuureja vertailtavassa muodossa.

4.5.2 Kehityksen nopeus

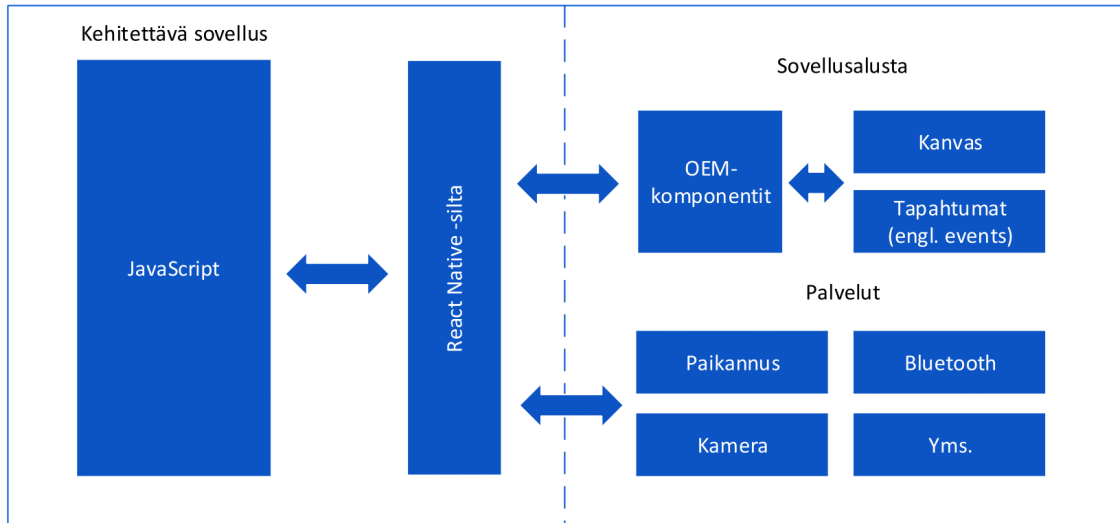
Flutter ja React Native tukevat molemmat erilaisia kirjastoja. Kirjastojen avulla ohjelmien kehittäminen nopeutuu, sillä kehittäjän ei tarvitse kirjoittaa kaikkea koodia itse, vaan voi sen sijaan käyttää valmiita toteutuksia hyväkseen. Flutterin nuoren iän johdosta, sille ei ole vielä samanlaista määrää valmiita toteutuksia kuin React Nativelle. React Native -hakusanalla löytyy npm:stä noin 20 000 pakettia, kun taas `dependency:flutter`-hakusanalla löytyy `pub.dev`:istä n. 4 000 pakettia.

Toinen kehityksen nopeuteen vaikuttava tekijä on dokumentaation selkeys. Flutterin dokumentaatio on laaja ja apua ongelmiin löytyy niin tekstimuodossa kuin videoina. Jos haluaa ymmärtää Flutterin syvimmän olemuksen ja kaivaa nettisivuja tarpeeksi pitkään, voi löytää osioita, joissa lukee vain TODO. Tätä työtä kirjoitettaessa osa dokumentaatiosta on jo muuttunut matkan varrella ja lisää tietoa lisätään sivuille jatkuvasti. React Nativen

Flutterin arkkitehtuurikuvaus



React Nativen arkkitehtuurikuvaus

**Kuva 4.5.** React Nativen ja Flutterin arkkitehtuurit vertailtavassa muodossa

dokumentaatio puolestaan ei ole yhtä laaja kuin Flutterin, muun muassa opettavia askel-askeleelta selitetyjä esimerkkejä ei löydy React Nativen dokumentaatiosta.

4.5.3 Projektin pystytys

Flutterissa projektin pystyttäminen on helppoa, sillä siihen on laaja dokumentaatio, jossa opastetaan askel-askeleelta miten projektin pystyttämisen tulisi tapahtua eri ympäristöissä. Flutter sisältää myös komentoliittymän *Flutter doctor*, joka opastaa käyttäjää projektin pystyttämisessä ja kertoo käyttäjälle mitä työkaluja on jo asennettu ja mitä on vielä asennettava, jotta Flutter toimisi oikein. Dokumentaatiossa on myös esitetty miten mikäkin editori tulisi konfiguroida, jotta se toimisi Flutterin kanssa saumattomasti yhteen.

React Nativen opas *Getting started guide* olettaa, että käyttäjällä on jo asennettuna kaikki tarvittava iOS ja Android -kehittämistä varten. Dokumentaatiossa hypätään suoraan projektin luomiseen, joten React Native -projektin pystyttäminen ei ole läheskään niin helppoa ja käyttäjäystävällistä kuin Flutterissa.

4.5.4 Yritykset teknologioiden taustalla

Molempien teknologioiden taustalla on isot yritykset: Flutterilla Google ja React Nativella Facebook. Facebook on toteuttanut React Nativen avulla suuri sovelluksia, joita ovat Facebook, Facebook Ads Manager, Facebook Analytics, sekä Instagram. Muita suuria yrityksiä, jotka käyttävät React Nativella tehtyjä sovelluksia ovat muun muassa Skype, Uber ja Pinterest. [85] Google käyttää Flutteria tällä hetkellä vain Google Ads-sovelluksessa. Muita isoja sovelluksia, jotka ovat tehty Flutterilla ovat muun muassa Alibaba ja Insight Timer ¹. [9]

¹Insight Timer -sovelluksella on yli 9 miljoonaa käyttäjää.

5 TUTKIMUS

Tutkimuksessa selvitetään, onko Flutter sopiva ohjelmistokehys Profit Softwaren tuleviin projekteihin. Tutkimukseen kuuluu demosovellus, jonka tarkoituksena on auttaa evaluoinnissa käytännön kokemuksen kautta. Tutkimuksessa rajoitutaan pääosin mobiilikehitykseen, sillä Flutterin muut kehitysmuodot ovat vielä siinä pisteessä, ettei niitä ole järkevää evaluoida näin kattavasti.

5.1 Evaluointikriteerit

Evaluointikriteerit iteroitiin lopulliseen muotoonsa Christoph Riegerin ja Tim A. Majchrzakin tutkimuksen [81] tekemän jaottelun, sekä asiakkaan, Profitin, vaatimusten kautta. Kriteerit jaotellaan neljään eri ryhmään: infrastruktuuri-, kehitys-, sovellus- ja käytettävyyss-kriteerit. Seuraavissa osioissa esitellään eri ryhmien kriteerit.

5.1.1 Infrastruktuurikriteerit

Alustariippumattomissa kehitysteknologioissa on aina tiettyjä rajoituksia infrastruktuuria koskien. Näitä rajoitteita tutkitaan tässä osioissa esiteltyjen kriteerien avulla.

Lisenssi

Teknologian käyttämä lisenssi on ehdottoman tärkeää selvittää ainakin niissä tapauksissa, kun sovellusta luodaan kaupalliseen käyttöön. Avoimen lähdekoodin ohjelmistokehityksiä jaetaan hyvin vapaasti, mutta niissä voi olla myös ominaisuuksia, jotka saa käyttöön vain ostamalla premium-version. Tällaisia ominaisuuksia voi olla esimerkiksi ylläpito ja konsultointi.

Tuetut kohdealustat

Alustariippumattomien teknologioiden tärkein ominaisuus on tarjota sovellus eri käyttöjärjestelmiä käyttäville laitteille samasta koodipohjasta. Näille teknologioille on tärkeää

selvittää, mitä kohdealustoja ja käyttöjärjestelmiä ne tukevat. Myös muiden kuin mobiilialustojen tuki evaluoidaan tässä kriteerissä.

Tuetut kehitysalustat

Tämän kriteerin avulla tutkitaan, mitä mahdollisia kehitysalustoja teknologialla on. Kehitysalustalla tarkoitetaan tietokonelaitteiston käyttöjärjestelmää.

Jakelukanavat

Sovelluksia jaetaan pääasiallisesti käyttäjille sovelluskauppojen kautta. Näistä suosituimpia ovat Applen Apple Store ja Googlen Google Play, jotka tarjoavat laajan valikoiman sovelluksia. Hyvä integraatio kauppojen kanssa muodostuu muun muassa mahdollistamalla kaupan ominaisuuksien käytön suoraan sovelluksesta, kuten sovelluksen arvosteleminen ja maksaminen sovelluksen sisällä. Kaikkia sovelluksia ei ole kuitenkaan mahdollista ladata sovelluskauppoihin, vaan niiden jakelu tapahtuu jostain muuta kautta. Esimerkiksi PWA-sovellukset on mahdollista ladata omaan laitteeseen vain selaimen kautta.

Ansaintamalli

Sovelluksien ansintamallit voidaan jakaa viiteen kategoriaan:

- Sovellus voidaan ostaa kertamaksulla ennen latausta tai ilmaisen jakson jälkeen.
- Sovellusta voidaan käyttää suurilta osin ilmaiseksi. Osan ominaisuuksista saa käyttöön vain maksamalla. Näissä sovelluksissa maksaminen tapahtuu usein sovelluksen sisällä.
- Sovelluksen käyttäminen vaatii useita maksuja: sovelluksen lataaminen saattaa maksaa, kuukausittaisesta käytöstä voidaan periä maksuja tai sovelluksen käyttäminen vaatii sovelluksen sisällä tapahtuvia maksuja.
- Sovelluksessa näytetään mainoksia.
- Sovellus on täysin ilmainen.

Ohjelmistokehitys voi tukea erilaisia ansaintamalleja. Tässä työssä ansaintamalliin ei keskitytä erityisemmin, sillä se ei ole erityisen tärkeä ominaisuus Profitille.

Kansainvälistäminen

Sovelluksien jakamisessa ei ole usein rajoitteita liittyen siihen, missä päin maailmaa sovellus ladataan. Vaikka sovellus olisi tehty vain suomeksi, voidaan se silti ladata omaan laitteeseen myös muualla maailmassa. Eri kielten tukeminen mahdollistaa sovelluksen

käyttäjäkunnan kasvun. Ohjelmistokehykset voivat helpottaa kehittämistä monikielisesti muun muassa tukemalla lokalisointia.

Kannattavuus pitkällä aikavälillä

Teknologian tulevaisuutta on tärkeä pohtia ennen kuin teknologiaan investoidaan. Kannattavuutta pitkällä aikavälillä on hankala evaluoida ilman kriteerin pilkkomista pienempiin osiin laajuutensa takia, joten kriteeri jaetaan kolmeen osaan: maturiteetti, vakaus ja aktiivisuus.

Teknologian maturiteetti Tässä osiossa tutkitaan teknologian ikää, historiaa sekä sovelluksia, joita on kehitetty jo aiemmin kyseisellä teknologialla. Historiaa tutkimalla voidaan vetää johtopäätöksiä muun muassa siitä, miten tietoturvakorjauksia korjataan.

Teknologian vakaus Vakautta voidaan arvioida menneiden ja tulevien julkaisujen frekvenssin perusteella. Evaluoinnissa käytetään apuna seuraavia kysymyksiä:

- Kuinka usein uusia toimintoja (engl. feature) julkaistaan?
- Ovatko päivitykset olleet taaksepäin sopivia? Jos ovat, kuinka pitkälle?
- Ovatko pienten ominaisuuksien julkaisutaajuudet lyhyitä?
- Julkaistaanko korjauksia ohjelmointivirheisiin ja tietoturvapäivityksiä kuinka nopeasti virheiden ilmaantuessa?

Yhteisön aktiivisuus Yhteisön aktiivisuutta voidaan evaluoida seuraavilla kysymyksillä:

- Onko ohjelmistokehyksellä aktiivinen yhteisö, joka raportoi ohjelmointivirheistä?
- Keskustelee yhteisö aktiivisesti ratkaisusta ongelmiin liittyen?
- Voiko yhteisö täydentää dokumentaatiota tarvittaessa?
- Onko yhteisö niin sitoutunut, että se voi ottaa vastuun teknologian kehittämisestä tarvittaessa?

Näiden lisäksi on tärkeää myös evaluoida teknologian tulevaisuutta uutisten, suunnitelmien ja huhujen perusteella. Jos teknologia on kytköksissä johonkin yritykseen tai yhdistykseen, evaluoinnissa olisi hyvä tarkastella myös pääosakkaita.

5.1.2 Kehityskriteerit

Kehityskriteerit ottavat kantaa millainen teknologia on kehittäjän näkökulmasta. Näissä kriteereissä pureudutaan erityisesti siihen, miten kehittäjää autetaan oppimaan teknolo-

giaa ja miten kehittäjää tuetaan teknologian käyttämisessä kehitysvaiheessa.

Kehitysympäristö

Kehitysympäristöjä on olemassa joka lähtöön. Erilaisten toimintojen rikkaus ja maturiteetti vaikuttavat kehittämisen tuotteliaisuuteen merkittävästi. Toiminnoilla tarkoitetaan muun muassa automaattista tekstinsyöttöä, kirjastojen dokumentaatioiden integraatiota, sekä virheiden jäljittäjää (engl. debugger). Jos toimintoja on liikaa, voivat ne vaikeuttaa kehittämistä. Jos taas toimintoja on liian vähän, heikentää se tuottavuutta. Jos teknologia tukee joitain kehitysympäristöjä, on kehityksen aloittaminen yleensä vaivattomampaa ja nopeampaa.

Teknologian omaksuminen

Teknologian nopea omaksuminen ja sen soveltaminen on tärkeää, koska uusia ohjelmistokehyksiä muodostuu jatkuvasti. Tämän kriteerin tarkoitus on esitellä kehityksen käyttämä teknologiakokonaisuus, ohjelmointiparadigmat, sekä dokumentaatio.

Skaalautuvuus

Suurissa projekteissa on tärkeää, että sovellus skaalautuu. Tämän voi toteuttaa käyttämällä erilaisia arkkitehtuurimalleja, kuten MVC-mallia. Ohjelmistokehykset saattavat rajoittaa skaalautuvuutta varsinkin silloin, jos kehyksellä on rajoitteita komponenttien jakamisessa pienempiin osiin. Uusien ihmisten lisääminen projektiin on tärkeää silloin, kun projekti kasvaa, joten suurienkin sovellusten tulee olla järkevästi jaoteltuja, jotta projektin rakenne on selkeä.

Projektimenetelmien yhteensopivuus

Tässä kriteerissä evaluoidaan, mahdollistaako teknologia erilaisten projektimenetelmien käytön. Tätä voidaan tutkia muun muassa arvioimalla kuinka nopeasti teknologian avulla saadaan aikaan MVP. Tässä kriteerissä ei enää evaluoida sitä, miten uusien tekijöiden lisääminen projektiin on mahdollista, koska se huomioitiin jo skaalautuvuus-kriteerissä.

Käyttöliittymäsuunnittelu

Käyttöliittymäsuunnittelu on tärkeässä osassa nykypäivän sovelluksia. Omat haasteensa suunnitteluun tuo erilaiset päätelaitteet, joiden näyttökoot vaihtelevat toisistaan. Teknologian tulisikin tukea kehitystä erilaisille laitteille responsiivisen suunnittelun avulla. Jos

responsiivisyyttä ei tueta, joutuu kehittäjä luomaan useita ulkoasuja, mikä kuluttaa huomattavasti enemmän resursseja.

Suunnittelua helpottaa, jos teknologian avulla on mahdollista nähdä tehdyt muutokset ilman pitkiä latausaikoja. Myös emulaattorin korvaaminen oikealla kohdelaitteella parantaa lopputuotteen käyttöliittymää.

Tässä kriteerissä on myös tärkeää ottaa huomioon, minkälainen tuki teknologialla on eri alustoille tyypilliselle suunnittelulle. Mitä vähemmän tukea on, sitä enemmän työtä vaaditaan kehittäjiltä, jos halutaan luoda aidon tuntuinen sovellus eri kohdealustoille.

Testaus

Sovellusten testaus on tärkeä osa niiden kehitystä. Tämän takia myös ohjelmistokehityksien tulee tukea erilaisia testausmuotoja. Perinteisten yksikkö- ja integraatiotestien lisäksi, mobiilisovelluksissa tulee ottaa huomioon myös erilaiset käyttötapaukset, kuten liikuminen sovellusten välillä. Myös mahdollisuudet sovelluksen ajonaikaiseen valvontaan, kuten kehittäjille tarkoitettu konsoli ja selkeä lokitus, parantavat sovelluksen testattavuutta. Näiden lisäksi myös sovelluksen ajaminen varsinaisella kohdelaitteella emulaattorin sijaan auttaa tiettyjen virheiden löytämisessä.

Jatkuva käyttöönotto

Eri ohjelmistokehitykset voivat tarjota helpotusta sovelluksen käyttöönottoon eri tavoilla. Toinen voi luoda vain lähdekoodin kohdealustalle, kun taas toinen saattaa antaa suoraan valmiit paketit ja vielä auttaa näiden julkaisussa. Jos projektissa on käytössä agile-projektimenetelmä, käytössä on luultavasti myös jonkinlainen putki, jonka avulla uudet versiot saadaan otettua käyttöön helposti. Tämä putki sisältää yleensä projektin rakentamisen (engl. build), automaattisesti ajettavat testit, sekä sovelluksen julkaisun.

Projektin konfigurointi eri ympäristöjen välillä

Yritysten ohjelmistoprojekteissa sovelluksista on usein monia eri versioita, jotka palvelevat eri tarkoituksia. Versiot jaetaan yleensä niiden käyttötarkoituksen mukaan tuotanto-, kehitys- ja testiversioon. Näiden lisäksi voidaan myös tarvita eri versiot eri käyttäjille ja esimerkiksi teemoille.

Ylläpidettävyys

Sovelluksen ylläpito ja sen helppous on tärkeä osa sovelluksen elinkaarta. Vaikka kehittäminen sujuisikin ongelmitta ja nopeasti, voi huono ylläpidettävyys lisätä sovelluksesta

koituvia kuluja merkittävästi.

Tässä työssä ylläpidettävyyttä evaluoidaan muun muassa koodin luettavuuden avulla. Tämän lisäksi myös tutkitaan dokumentointia tiedostojen sisällä, sekä koodin uudelleenkäytettävyyttä muissa projekteissa. Ylläpidettävyyteen liittyy myös jo aiemmin esitelty kriteeri: teknologian omaksuminen, sillä jos sovellusta ylläpitää eri henkilöt kuin sovelluksen alkuperäiset kehittäjät, tulee heidän myös osata kyseistä teknologiaa.

Laajennettavuus

Laajennettavuudella tarkoitetaan sitä, kuinka paljon ohjelmointikehykselle on luotu valmiita toteutuksia. Mitä enemmän toteutuksia on, sitä nopeampaa kehittäminen on.

Integraatio natiivin koodin kirjoittamiselle

Välillä sovelluksia kehittäessä hyvin varustellullakin teknologialla tulee vastaan tiettyjä ominaisuuksia, joihin teknologia ja sen kirjastot eivät vain yksinkertaisesti pysty suoraan itse. Näissä tilanteissa vaihtoehtoina on joko ottaa yhteyttä natiiveihin API-rajapintoihin tai kirjoittaa suoraan natiivia koodia, jos teknologia sitä tukee.

Kehityksen nopeus

Avain nopeaan kehitykseen on ohjelmistokehyksen helppokäyttöisyys, sekä kehitykselle jo valmiiksi toteutetut funktionaalisuudet, joita voidaan suoraan käyttää hyödyksi omissa projekteissa. Kehityksen nopeus muodostaa suuren osan projektin kustannuksista, sekä vaikuttaa myös tämän kautta suoraan projektin tuottavuuteen.

5.1.3 Sovelluskriteerit

Natiiveilla ohjelmistokehyksillä kehitetyillä sovelluksilla on mahdollista päästä käsiksi kaikkiin laite- ja alustakohtaisiin ominaisuuksiin. Hyvän ohjelmistokehyksen tulisi tarjota natiivinkaltainen kokemus sovelluksen käyttäjälle, joka voidaan usein saavuttaa vain käyttämällä laitteelle ja alustalle ominaisia ominaisuuksia. Tämän kriteerijoukon avulla evaluoidaan teknologian integraatioita.

Laitekohtaisten ominaisuuksien saatavuus

Jos teknologian avulla ei ole mahdollista tavoittaa laitekohtaisia ominaisuuksia, voi sovelluksen toiminnallisuus jäädä vajavaiseksi. Tässä kriteerissä tutkitaan miten teknologia

pystyy käyttämään muun muassa laitteen kameraa, mikrofonia, GPS:ää, sekä sensoreita.

Alustakohtaisten ominaisuuksien saatavuus

Tässä kriteerissä evaluoidaan teknologian yhdistämistä alustoille tyypillisiin ominaisuuksiin. Näitä ovat muun muassa akun ja internetyhteyden tiedot, pysyvyyden takaaminen, sekä yhteystietoihin pääseminen.

Tuki liitetyille laitteille

Laitteisiin pystyy nykyisin liittämään erilaisia laitteita muun muassa bluetoothin kautta. Liitettuja laitteita voi olla esimerkiksi kuulokkeet tai älykellot. Kattava ohjelmistokehitys pystyy viemään tietoa liitettyyn laitteeseen, sekä myös käyttämään hyväksi liitetyn laitteen välittämiä tietoja.

Syötteen heterogeenisyys

Mobiililaitteen käyttäjällä on useita eri tapoja syöttää tietoa laitteeseen. Näitä ovat muun muassa näppäimistö, hiiri, kosketusnäyttö, laitteen fyysiset näppäimet, sekä nykyään myös yleistyvät äänet, eleet ja kasvojentunnistus.

Ulostulomuotojen heterogeenisyys

Ulostulomuotojen heterogeenisyys hankaloittaa kehittämistä. Eri ominaisuuksilla varustetut näytöt vaativat huomiota, jotta sovellus skaalautuu kaikille näytöille yhtä hyvin. Näiden lisäksi myös sovelluksen projektointi, sekä äänipalaute ovat mahdollisia ulostulomuotoja.

Sovelluksen elinkaari

Sovelluksen elinkaari koostuu sovelluksen käynnistämisestä, keskeyttämisestä, jatkamisesta, sekä sulkemisesta. Sovellus keskeytyy, kun käyttäjä poistuu sovelluksesta, mutta ei sulje sitä. Sovelluksen tulisi jatkaa toimintaansa siitä mihin se keskeytyessään jäi, kun käyttäjä palaa takaisin sovellukseen, eli jatkaa sovelluksen käyttöä tauon jälkeen. Näiden toimintojen lisäksi eri alustoilla voi olla vielä lisätoimintoja.

Palvelin-selain-integraatio

Sovellusten liiketoiminnallinen logiikka käsitellään sovelluksen palvelinpuolella, jonka takia tiedonsiirto selain- ja palvelinpuolen välillä on hyvin tärkeää.

Turvallisuus

Sovelluksen ei tulisi käyttää alustan tai laitteen ominaisuuksia ilman lupaa ja silloinkin vain jos on pakko. Sovelluksen käyttämä, mahdollinen sensitiivinen data tulisi aina kryptata ja sitä ei saisi koskaan tallentaa sellaisenaan. Tiedonsiirroissa tulisi käyttää salattuja yhteyksiä. Ohjelmistokehykset voivat tarjota näihin erilaisia tukia. Parhaassa tapauksessa teknologiassa on olemassa jo valmiit työkalut, jolloin kokemattomat kehittäjät eivät joudu olemaan vastuussa turvallisuuteen liittyvää toiminnallisuutta.

5.1.4 Käytettävyysskriteerit

Käytettävyysskriteereissä evaluoidaan sovelluksen käytettävyyteen vaikuttavia asioita.

Sovelluksen ulkonäkö ja tuntuma

Monialustateknologioiden suurin ongelma on yleensä se, etteivät ne usein yllä natiiveilla teknologioilla kehitettyjen sovellusten tasolle käyttäjäkokemuksessa. Tässä kriteerissä evaluoidaan aidon natiivin tuntuman lisäksi myös animaatioita ja transitoita.

Suorituskyky

Sovellukset, joiden tehokkuus ei ole hyvä, eivät ole usein käyttäjäkokemukseltaan parhaita. Ihmisten kärsimättömyyden takia pitkät latausajat sovelluksen avaamisessa, näkymien vaihtamisessa, sekä käyttäjän syötteiden tulkinnassa saavat käyttäjän helposti turhautumaan.

Käyttäjän autentikointi

Nykyisin erilaisia kirjautumistapoja on useita, muun muassa sormenjälkitunnistus, kasvojen tunnistus ja perinteinen salasana. Hyvä ohjelmistokehys tukee vanhojen kirjautumistapojen lisäksi myös uusia innovatiivisia tapoja.

5.2 Prototyypisovellus

Tutkimuksessa tehtiin demosovellus Profit Software Oy:lle. Sovelluksen perustoimintona on selata ja ostaa lippuja erilaisiin tapahtumiin. Ostotapahtumasta on pyritty tekemään mahdollisimman kokonaisvaltainen ja siinä käyttäjä pystyy valitsemaan matkaseurueen koon, istumapaikkojen hintaryhmän, matkan tarkemman ajankohdan, hotellihuoneen tason ja matkustustavat kaupungin sisäisesti sekä kaupunkiin. Sovellus on rajoitettu vain selainpuolen toteutukseen ja applikaatiosta on tehty vain kaksi näkymää, joita käydään läpi tulevissa osioissa.

Sovelluksen kehittämisen motivaationa oli Profitin kiinnostus interaktiivisemmasta demosta kuin mitä UX-työkalut voivat tarjota projektien myyntiä varten. Diplomityössä sovelluksen kehittäminen oli hyvä tapa tutustua konkreettisesti Flutterin toimintaan. Yritykselle demotarkoitukseen tuleva sovellus oli myöskin mielekkäämpää toteuttaa kuin jokin oma projekti.

5.2.1 Aloituspääkymä

Aloituspääkymässä on mahdollista liikkua kolmen eri välilehden välillä joko pyyhkäisemällä sivulle tai painamalla välilehdistä. Ensimmäisellä välilehdellä näkyvät tulevat konsertit, toisella tulevat esitykset ja kolmannella tulevat urheilutapahtumat. Liitteen B kuvassa B.1 on vasemmalla kuva aloituspääkymästä.

5.2.2 Ostotapahtuma-näkymä

Ostotapahtuma-näkymässä on mahdollista valita kokonaisvaltainen paketti esitykseen saapumista varten. Näkymä on rakennettu painalluksesta laajentuviin laatikoista, joiden kapeassa näkymässä nähdään yhteenveto tehdyistä valinnoista ja laajentuvassa osiossa valitaan, mitä kustakin kategoriasta halutaan liukusäätimien avulla. Ensimmäisessä osassa valitaan matkaseurue, toisessa konserttipaikkojen hintaryhmä, kolmannessa vierailun ajankohta, neljännessä hotellihuoneen taso, viidennessä matkustustapa paikkakuntien välillä ja viimeisessä valitaan miten halutaan liikkua paikkakunnain sisällä. Näkymän alareunaan on kiinnitetty Hyvältä näyttää -nappi, jota painamalla käyttäjä siirtyy yhteenveto-sivulle. Liitteessä B on kuvia Ostotapahtuma-näkymästä. Kuvassa B.1 on oikealla kuva koko näkymästä niin, että mitään laatikkoa ei ole laajennettu. Kuvassa B.2 esitellään kaikkien laatikoiden sisältö laajennettuina.

6 TULOKSET

Tässä osiossa esitellään tutkimuksen tulokset, jotka ovat muodostettu erilaisten lähteiden, sekä demosovelluksen kehityksestä opittujen asioiden pohjalta. Osiossa 5 esiteltiin kriteereitä, joiden avulla tämän osion evaluointi on suoritettu. Kriteerit käydään läpi samassa järjestyksessä kuin edellisessäkin osiossa.

6.1 Lisenssi

Flutter on avoimen lähdekoodin ohjelmistokehys, joka on täysin ilmainen. Lisenssin suhteen Flutter voidaan jakaa kahteen osaan: koneisto ja kehys. Kehyksellä on vain yksi lisenssi, joka on lisätty työhön liitteenä C. Kehyksen lisäksi avoimien lähdekoodien projekteissa tulee ottaa huomioon lisenssien tarttuvuus, eli jokaisen projektiin liitetyn riippuvuuden oma lisenssi. Tämän johdosta myös Flutterin koneiston lisenssit tulee ottaa huomioon. Koneistossa lisenssejä on huomattavasti enemmän [38] kuin kehyksessä, mutta niissä ei ole merkittäviä sovellusta rajoittavia tekijöitä. Näiden lisenssin lisäksi Dartin erilaisilla kirjastoilla on omia lisenssejä, jotka tulee myös ottaa huomioon tapauskohtaisesti.

6.2 Tuetut kohdealustat

Kuten jo aiemminkin työssä on todettu, tällä hetkellä merkittävimmät markkinaosuudet mobiilialustojen suhteen omaavat selkeästi iOS ja Android, kuten kuvaajassa 1.1 on esitetty. Applen iOS-laitteista tuetaan kaikkia, joiden malli on vähintään iPhone 4s ja käyttöjärjestelmä iOS 8. iOS-laitteiden lisäksi Flutterilla voi kehittää sovelluksia myös kaikille ARM Android laitteille, joiden käyttöjärjestelmä on vähintään Android Jelly Bean, v16, 4.1.x. [34] Tuetut käyttöjärjestelmät ovat jo noin 8 vuotta vanhoja, minkä takia Flutter kattaa suuren osan käytössä olevista mobiililaitteista.

Flutter ei ainakaan vielä tue kehittämistä tableteille Material guide -oppaan mukaisesti, eikä teknologiaa myöskään ole kehitysvaiheessa testattu tableteilla. Flutterin kehittäjät kuitenkin uskovat samojen widgetien toimivan hyvin myös tableteilla, vaikka täyttä varmuutta tästä ei ole. Tästä huolimatta Flutterin avulla on jo kehitetty sovelluksia myös tableteille responsiivisuuden avulla. [79]

Mobiilikehityksen lisäksi Flutterista on jo olemassa versiot web-sovellusten, työpöytäso-

vellusten ja sulautetuiden järjestelmien sovellusten kehittämiseen. Versiot ovat kuitenkin vasta kehitysvaiheessa, eli ne eivät ole vielä vakaita eikä niiden avulla kannata kehittää kaupalliseen käyttöön tulevia sovelluksia. Web- ja työpöytäkehityksen mahdollistaminen tuo Flutteriin lisäarvoa ohjelmistokehityksenä, sillä niiden avulla samalla teknologialla on mahdollista kehittää sovelluksia lähes kaikkiin käyttötarkoituksiin.

6.3 Tuetut kehitysalustat

Teknologialla on mahdollista kehittää sovelluksia Linux, Mac ja Windows käyttöjärjestelmissä. iOS-sovelluksia ei usein pysty kehittämään ilman Mac-käyttöjärjestelmää, eikä Flutterkaan tee tähän poikkeusta. Codemagic-työkalun avulla on mahdollista julkaista Flutterilla kehitetty iOS-sovellus ilman Mac-tietokonetta, mutta ei kehittä. Kehittäminen tulee suorittaa Android-sovelluksen kautta. [48]

6.4 Jakelukanavat

Jakelukanavina tuetaan Google Play:ta ja Apple Storea. Hyvä integraatio sovelluskauppojen kanssa on mahdollistettu erilaisten kirjastojen avulla. Sovelluksen arvioimisen toteuttaminen onnistuu sille luodun kirjaston [8] avulla, sekä uudelleenohjaus kauppaan kyseisen sovelluksen sivulle onnistuu myös oman kirjaston [77] kautta. Molempien kauppojen kautta on mahdollista julkaista sovellus myös yksityiseksi, jolloin sovellukseen pääsee käsiksi vain kutsun tai linkin kautta, mikä on kätevää sovellusta kehittäessä.

6.5 Ansaintamalli

Flutter-sovellukset ladataan laitteeseen kauppojen kautta, mikä mahdollistaa maksullisen ja ilmaisen latauksen. Flutteriin on myös tehty kirjastoja, joiden avulla mahdollistetaan muun muassa mainokset [3] ja maksaminen sovelluksen sisällä [39].

6.6 Kansainvälistäminen

Teknologia ottaa huomioon saman sovelluksen käyttämisen eri kielillä lokalisoinnilla Dartin intl-kirjaston avulla. Lokalisointituki on toteutettu valmiiksi noin 52 kielelle, mikä ei kuitenkaan tarkoita, että kaikkien kielten tuki olisi aina automaattisesti päällä, vaan kehittäjän vastuulle on jätetty niiden kontrollointi. Myös uusien kielten lisääminen on mahdollista, mutta se vaatii ylimääräistä työtä. Dartin intl-kirjaston voi myös halutessaan korvata omalla toteutuksella tai jollain muulla kirjastolla, kuten i18n¹.

¹i18n on Profitilla aiemmin käytössä ollut lokalisoitikirjasto.

intl-kirjaston avulla on mahdollista formatoida numeroita ja päivämääriä, sekä vaihtaa tekstin lukusuuntaa horisontaalisesti. Kirjastosta puuttuu vielä ominaisuuksia, kuten numeroiden parsiminen, rahayksiköiden käyttäminen tekstin sijaan formatoinneissa, sekä aikavyöhykkeiden käyttö. Aikavyöhykkeitä ei voida käyttää kirjastossa, sillä Dartin `DateTime`-objekti ei tue eri aikavyöhykkeitä. [51] Näiden lisäksi Flutterin kehittäjät tekivät jo aikaisessa vaiheessa päätöksen, jonka seurauksena vertikaalisen tekstin tukea ei toteuteta Flutteriin, sillä se olisi vaikeuttanut Flutterin rakennetta merkittävästi [15].

6.7 Teknologian kannattavuus pitkällä aikavälillä

Teknologian kannattavuutta pitkällä aikavälillä evaluoidaan teknologian maturiteetin, vakauden ja yhteisön aktiivisuuden perusteella.

Teknologian maturiteetti Kuten jo aiemmin työssä on todettu, Flutter on uusi teknologia, minkä takia kehittäjäkunta on ainakin vielä pieni verrattuna esimerkiksi React Nativeen. Nuoresta iästä huolimattaan, Flutterin avulla on kehitetty sovellus muun muassa Alibaballe, jolla on noin 674 miljoonaa aktiivista käyttäjää vuosittain [4].

Flutter on jatkuvan kehityksen alla, minkä takia korjauksia ja uusia ominaisuuksia esitellään kuukausittain. Hotfix-korjauksia varsinaisiin kehityshaaroihin on julkaistu noin yhdestä päivästä kuukauteen sen jälkeen, kun varsinaiset haarat on julkaistu.

Teknologian vakaus Teknologian nuori ikä vaikuttaa sen vakauteen, sillä uusia ominaisuuksia ja korjauksia julkaistaan jatkuvasti. Taulukkoon 6.1 on kerätty viimeisimpiä julkaisuja ja niiden ajankohtia. Uusia isompia kokonaisuuksia on julkaistu noin kuukauden välein, pienempiä ominaisuuksia taas huomattavasti nopeammalla frekvenssillä.

Flutter pyrkii taaksepäin yhteensopivuuteen aina kun se vain on mahdollista. Maaliskuussa 2019 Flutter julkaisi ottavansa käyttöön AndroidX:n, joka on uusi kirjastojen hallintajärjestelmä vanhan Support Library:n tilalle. AndroidX:n päätarkoituksena on selkeyttää kirjastojen nimeämistä. Tämä muutos ei ollut taaksepäin yhteensopiva, jos projektissa oli jo otettu käyttöön Support Library. Flutterin viimeisimmissä vakaisissa versioissa (1.9.1, 1.7.8, 1.5.4 ja 1.2.1) on kaikissa esitelty taaksepäin yhteensopimattomia muutoksia. Näillä muutoksilla halutaan turvata se, että API-rajapinta on jatkossakin intuitiivinen huolimatta lisääntyneistä ominaisuuksista [68].

Flutteria kehitetään neljään eri haaraan: master, dev, beta ja stable, joista kehittäjät voivat valita omiin tarkoituksiin sopivan haaran. master-haarasta löytyy uusimmat muutokset ja haara voi olla hyvin epävakaa ja sisältää erilaisia ohjelmointivirheitä. dev-haarassa on aina uusin testattu kokonaisuus, johon pyritään tuomaan master-haara mahdollisimman usein. dev-haara eroaa master-haarasta testien kattavuudessa, sillä dev-haaran testit ovat kattavammat. beta-haaraan valitaan joka kuukausi paras dev-haarassa ollut kokonaisuus. stable-haaraa pyritään päivittämään vain kerran kvartaalissa, jolloin haaraan

Versio	Julkaisupäivämäärä	Versio	Julkaisupäivämäärä
v1.10.2	12.9.2019	v1.7.8+hotfix.3	9.7.2019
v1.9.1+hotfix.2	4.9.2019	v1.8.1	8.7.2019
v1.10.1	3.9.2019	v1.7.8+hotfix.2	2.7.2019
v1.10.0	2.9.2019	v1.8.0	1.7.2019
v1.9.1+hotfix.1	1.9.2019	v1.7.8+hotfix.1	2.7.2019
v1.9.7	28.8.2019	v1.7.11	27.6.2019
v1.9.6	28.8.2019	v1.7.10	25.6.2019
v1.9.5	23.8.2019	v1.7.9	24.6.2019
v1.9.4	23.8.2019	v1.7.8	22.6.2019
v1.9.3	21.8.2019	v1.7.7	21.6.2019
v1.9.2	20.8.2019	v1.7.6	21.6.2019
v1.9.1	18.8.2019	v1.7.5	19.6.2019
v1.9.0	14.8.2019	v1.7.4.	14.7.2019
v1.8.4	2.8.2019	v1.7.3	7.6.2019
v1.8.3	27.7.2019	v1.7.2	5.6.2019
v1.7.8+hotfix.4	19.7.2019	v1.7.1	4.6.2019
v1.8.2	11.7.2019	v1.7.0	31.5.2019

Taulukko 6.1. Flutterin viimeisimmät julkaisut

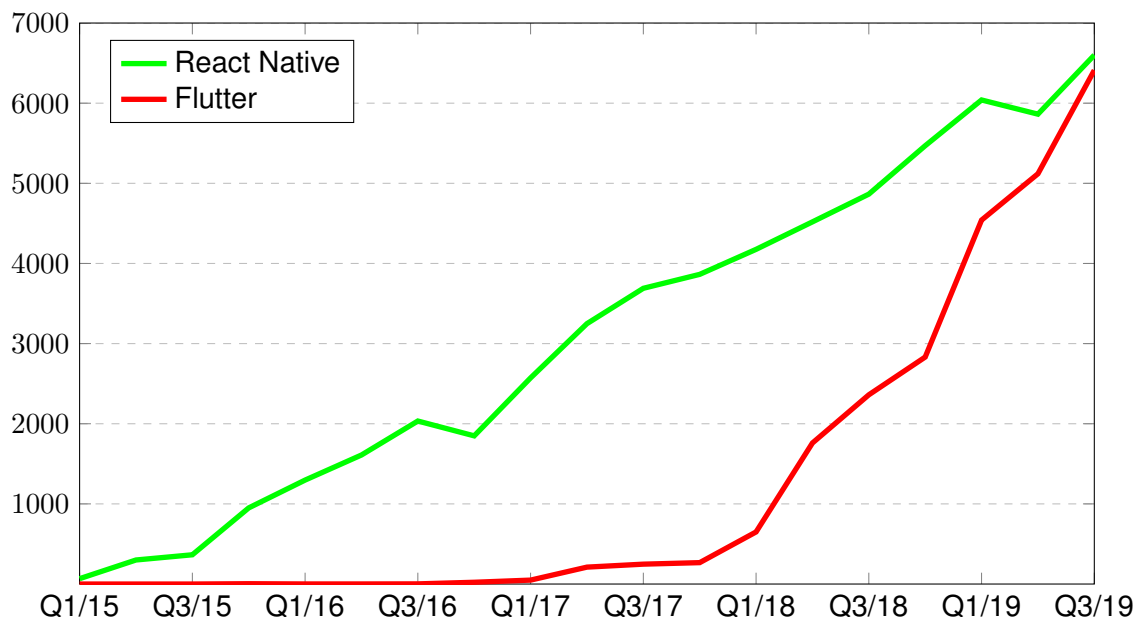
on tarkoitus viedä aina vain vakaita versioita teknologiasta. Kehittäjiä kehoitetaan käyttämään stable-haaraa tuotantojulkaisuissa. [31]

Yhteisön aktiivisuus Flutterin yhteisö on kasvava ja aktiivinen, josta merkinä on Stack Overflow:n kysymyksien määrä, joissa Flutter on viime kuukausien aikana saavuttanut React Nativea, kuten kuvaajasta 6.1 nähdään ². Flutterista on tehty yhteensä noin 24 000 kysymystä ja React Nativesta noin 60 000. Toisena yhteisön koon mittarina toimii tässä työssä Githubissa annetut tähdet, joita Flutterilla on 75 500 ja React Nativella 81 000. Viimeisenä mittarina toimii TIOBE-indeksi, jota vertailtiin jo aiemmin työssä JavaScriptin, TypeScriptin ja Dartin välillä kuvaajien 3.1 avulla. JavaScript on TIOBE-tilastossa sijalla 7, TypeScript 37. ja Dart 26. [82].

Tällä hetkellä ei ole todennäköistä, että Flutterin yhteisö pystyisi ottamaan vastuun Flutterin kehittämisestä, jos Google jostain syystä päättäisi hylätä teknologian, sillä Flutterin 13 449 kommitoinnista 12 558 on Googlen työntekijöiden tekemiä ³, mikä on noin 93% koko kehityksestä. Flutteria kuitenkin kehitetään jatkuvasti ja Googlen tavoitteena teknologialle on korvata JavaScript web-sovellusten kehittämisessä. Flutterin tulevaisuu-

²Kuvaajassa esitetään kysymykset, jolla on tag-merkintänä flutter/react-native. Kysymykset ovat jaoteltu eri kvartaaleille niiden luontipäivämäärän mukaan. Data on haettu <https://data.stackexchange.com/>-sivuston kautta.

³Kommitoinnit ovat laskettu Flutterin Githubin Contributors-sivun [32] kautta Master-haarasta aikavälillä 19.10.2014-1.10.2019. Kehittäjä on laskettu Googlen työntekijäksi, jos hänen omalla Github-sivullaan on siitä merkintä.



Kuva 6.1. Flutterin ja React Nativen StackOverflow:n kysymyksien määrien kehitys kvartaaleittain vuosina 2015-2019.

densuunnitelmia on kerätty Githubiin virstanpylväinä vuoden 2020 huhtikuulle asti [57]. Näiden perusteella voidaan uskoa, että Google ei olisi hylkäämässä Flutteria hetkeen.

6.8 Kehitysympäristö

Teknologiaa on mahdollista kehittää missä tahansa tekstieditorissa Flutterin komentorivityökalujen avulla. Suositeltua on myös käyttää Flutter-liitännäisiä valitun editorin kanssa. Dokumentaatioissa on suositeltu käytettäväksi, joko Android Studiota, IntelliJ:tä tai Visual Studio Codea. Näiden, sekä sopivien liitännäisten, lataamiseen on kattavat ohjeet Flutterin dokumentaatioissa. [72]

Flutter mahdollistaa sovelluksen ajonaikaisen monitoroinnin kehittäjien työkalujen (Dev-Tools) avulla suoraan selaimessa. Näiden työkalujen avulla voi seurata muun muassa muistin kulutusta. Testaukseen liittyviä työkaluja on esitelty enemmän Testaus-kriteerissä.

Flutterilla kirjoitettuja sovelluksia on mahdollista asettaa osaksi jatkuvaa integrointia ja julkaisua monien eri työkalujen kautta. Flutterin dokumentaatioissa on tarkat ohjeet fastlane-työkalun käyttöön. Tätä työkalua ei kuitenkaan tueta vielä Linux eikä Windows käyttöjärjestelmillä. Muita Flutterin tukemia työkaluja ovat GitLab CI/CD [43], Codemagic [44] ja demosovelluksessakin käytetty Bitrise [12].

6.9 Teknologian omaksuminen

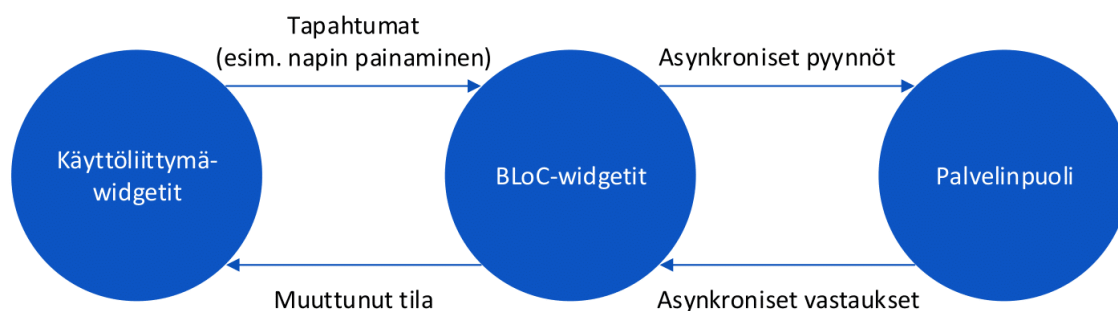
Kuten jo aiemmin on todettu, Flutter käyttää ohjelmointikielensä Dart:ia, joka ei ole vielä kovin yleisesti tunnettu. Dart on kuitenkin verrattaen helppo oppia, sillä se pohjautuu JavaScript, C# ja Java kieliin, joista ainakin yksi on yleensä tuttu kokemusta omaavalle kehittäjälle. Flutter käyttää ohjelmointiparadigmoinaan muun muassa olio-ohjelmointia, sekä imperatiivista ja funktionaalista ohjelmointia.

Flutterin oppimista helpottaa kattava dokumentaatio [33]. Uusille kehittäjille löytyy kattavat ohjeet projektin pystytykseen. Myös eri taustoista tuleville kehittäjille on omat osiot, joissa vertaillaan Flutterin eroja React Nativeen ja Xamarin.Formsiin. Näiden lisäksi on myös omat osiot Android- ja iOS-taustan omaaville sekä web-kehittäjille. Dokumentteista löytyy myös *Cookbook* [19], jonka tarkoituksena on esitellä ratkaisuja yleisimpiin ongelmiin esimerkkien kautta. Cookbook-osion lisäksi dokumentaatioissa on myös muita esimerkkejä, joissa opetetaan lukijaa askel-askeleelta. Flutterin API-dokumentaatio [30] on täysin oma kokonaisuutensa ja myös se on informatiivinen ja kattava. Flutterin lisäksi myös Dartin dokumentaatio [22] on helppolukuinen ja tarpeeksi laaja.

Flutterin sivuilla on myös olemassa *Codelabs*-osio [16], jossa on mahdollista opetella kehittämään sovelluksia tai niiden osia Flutterilla ilman työkalujen lataamista omalle koneelle. Osiossa on erilaisia harjoitustehtäviä, joita voi suorittaa suoraan omassa selaimessa.

6.10 Skaalautuvuus

Flutter mahdollistaa erilaisten sovellusarkkitehtuurien käytön. Googlen kehittäjät suosittelevat käytettäväksi Googlen kehittämää BLoC-mallia [78], jonka tarkoituksena on mahdollistaa widgetien uudelleenkäytettävyys. Kuvassa 6.2 on esitetty graafisesti BLoC-mallin toimintaa. Käyttöliittymä-widgetien tarkoituksena on vain näyttää käyttäjälle tietoa, niiden tarkoituksena ei ole tietää liiketoiminnallisesta logiikasta mitään, vaan tämä logiikka suoritetaan BLoC-widgetien avulla. BLoC-widgetin tarkoituksena on muuttaa käyttöliittymä-widgeiltä tuleva tapahtuma tilaksi, jonka voi antaa takaisin käyttöliittymä-widgeteille. Käytännössä BLoC-malli tekee tämän virtausten (engl. streams) avulla.



Kuva 6.2. Kuvaus BLoC-mallista.

6.11 Projektimenetelmien yhteensopivuus

Teknologia ei rajoita projektimenetelmien käyttöä. Sovelluksia on mahdollista kehittää niin ketterällä- kuin vesiputousmallilla. Ketterän kehityksen mahdollistaa muun muassa jatkuva käyttöönotto, jonka avulla on helppoa julkaista uusia versioita sovelluksesta, sekä projektien nopea pystytys.

6.12 Käyttöliittymäsuunnittelu

Responsiivisuuden voi toteuttaa Flutterissa muun muassa widgetien avulla. Yksi tapa on käyttää `LayoutBuilder`-luokkaa `BoxConstraints`-objektin kanssa, jolloin voidaan päättää mitä käyttäjälle näytetään eri kohdelaitteilla. Toinen yksinkertainen tapa on käyttää `MediaQuery.of()`-metodia rakentajafunktiossa, joka kertoo sovellukselle varatun tilan laitteen näytöllä, sekä muun muassa orientaation. Jälkimmäinen tapa on kätevämpi silloin, kun päätöksiä halutaan tehdä laitteen perusteella eikä vain yksittäisen widgetin. [20]

Flutterissa on mahdollista nähdä muutosten tulokset nopeasti Hot reload ja Hot restart -komentojen avulla. Komennot on esitelty jo aiemmin osiossa 4.

Teknologia mahdollistaa kehittämisen sekä Android, että iOS käyttöjärjestelmille myös käyttöliittymäsuunnittelun puolesta, sillä se sisältää kirjastot, jotka jäljittelevät alustoille tyypillisiä suunnitteluratkaisuja. Nämä kirjastot ovat Material Androidille ja Cupertino iOSille. Flutterin kanssa tulee kuitenkin muistaa, että ohjelmistokehitys ei käytä suoraan natiiveja komponentteja, vaan kehitykselle on luotu omat widgetit, jotka jäljittelevät natiiveja komponentteja. Tulevaisuudessa widgetien ajantasaisuus riippuu täysin Flutterin kehittäjistä.

Flutterille on olemassa erilaisia käyttöliittymäsuunnittelua helpottavia työkaluja, joiden avulla voi raahaamalla ja pudottamalla asetella widgetejä erilaisiksi kokonaisuuksiksi. Samalla nämä työkalut generoivat käyttökelpoista lähdekoodia, jonka käyttäminen osana omaa kehitystä voi nopeuttaa sovelluksen luontia merkittävästi. Monet näistä työkaluista ovat kuitenkin vasta kehitysasteella, kuten Flutter Studio [37] ja Widget-maker [86]. Myös suosittuun Figma-työkaluun on toteutettu kokeellinen Dart-generaattori, joka kääntää Figman komponentin Flutterin widgetiksi [29]. Työkalu käyttää hyväkseen Flutterin `dart:ui`-kirjastoa, jonka takia tuotettu koodi on hyvin matalan tason koodia ja vaikeaa ymmärtää. Esimerkiksi Figmalla tehty harmaa neliö -komponentti tuottaa generaattorin kautta Flutterin widgetin, jossa on 210 riviä koodia. Työkalu ei myöskään ymmärrä Figman vuorovaikutusominaisuuksia.

6.13 Testaus

Flutterissa kehittäjät ovat selkeästi halunneet luoda kattavat työkalut sovellusten testaamiseen. Testauksessa on mahdollista tehdä yksikkö-, widget- ja integraatiotestejä. Yksikkötesteissä on mahdollista käyttää mockattua dataa Flutterin *Mockito* [59] kirjaston kautta tai kirjoittamalla vaihtoehtoiset toteutukset dataa vaativille luokille. Widget-testit ovat samanlaisia kuin yleisemmin tunnetut komponenttitestit, eli ne testaavat yksittäisen widgetin toimintaa. Integraatiotesteillä testataan eri osakokonaisuuksien yhteensopivuutta, sekä sovelluksen suorituskykyä. Flutterissa kaikkia testejä on mahdollista testata automaattisesti, osana jatkuvaa integraatiota. Demosovelluksessa käytettiin Bitrise-palvelua julkaisemaan sovellusta ja ajamaan sille kirjoitettuja testejä automaattisesti koodipohjan muuttuessa.

Varsinaisten testien lisäksi Flutterissa on myös mahdollista seurata sovelluksen ominaisuuksia, kuten muistin käyttöä ja tehokkuutta *Dart DevTools* -työkalujen avulla selaimessa. Työkalujen ansiosta on myös mahdollista hidastaa animaatioita, mikä auttaa niihin liittyvien ongelmien kartoittamisessa. [36] Ohjelmointivirheiden löytämisen helpottamiseksi Flutterin kehittäjät ovat luoneet ohjelman, jonka tarkoituksena on analysoida koko koodipohja ja ilmoittaa mahdollisista ongelmista, joita ei muuten olisi huomattu. Ohjelman voi ajaa komentoriviltä komennolla `flutter analyze`.

Flutterin nettisivut sisältävät kattavat ohjeet testien kirjoittamiseen, sekä sovelluksen ajon aikaiseen monitorointiin. Flutterin kehittäjät ovat luoneet kirjastot, joiden avulla voi kirjoittaa yksikkö-, widget- ja integraatiotestejä, kun taas React Nativella virallinen tuki on vain Jest-testauskehykseen.

6.14 Jatkuva käyttöönotto

Flutterin dokumentaatioissa on ohjeet jatkuvalle käyttöönotolle [18]. Ohjeessa kehoitetaan käyttämään apuna fastlane-työkalua, jonka avulla pystytään automatisoimaan testausvaiheessa olevien sovellusten käyttöönottoa ja julkaisua [28]. Fastlane-työkalua voi tällä hetkellä käyttää vain Mac-käyttäjärjestelmän tietokoneilla.

Muita Flutterin tukemia työkaluja ovat Codemagic [44] ja Bitrise [12]. Molemmilla työkaluilla oli helppoa saada Flutter-projekti osaksi jatkuvaa integraatiota, eikä konfigurointia tarvittu juurikaan.

6.15 Projektin konfigurointi eri ympäristöjen välillä

Sovelluksen eri ympäristöjä, kuten esimerkiksi kehitys, testaus ja tuotanto, voidaan muokata eroamaan toisistaan konfiguraatioiden avulla. Liitteessä D on esitelty, miten ympäristön nimi voidaan konfiguroida riippumaan siitä, missä ympäristössä ollaan. Liitteessä

luodaan ensin konfiguraatio-tiedosto, johon luodaan `InheritedWidget`-widgetiä käyttäen `Konfiguraatio`-widget, jolle määritetään pakolliseksi muuttujaksi `ympäristönNimi`. Tämän jälkeen jokaiselle ympäristölle, eli tässä tapauksessa tuotanto- ja kehitysympäristölle, luodaan oma tiedosto, jossa määritellään `Konfiguraatio`-widgetin muuttujat juuri kyseiselle ympäristölle. Tämän jälkeen `Konfiguraatio`-widget voidaan ottaa käyttöön muualla ohjelmassa metodin `Konfiguraatio.of(context)` avulla. Eri ympäristöjä, sekä niiden konfiguraatioita, voidaan nyt käyttää ajamalla sovellusta eri komennoilla, esimerkiksi `flutter run -t lib/main_tuotanto.dart` ajaisi liitteen D sovelluksesta tuotantoympäristö-version. Jos tuotantoympäristön versio halutaan julkaista Androidilla, onnistuisi se komennolla `flutter build apk -t lib/main_tuotanto.dart`. iOSilla vastaava komento olisi `flutter build ios -t lib/main_tuotanto.dart`.

6.16 Ylläpidettävyys

Dart on pääpiirteittäin helposti luettava kieli johtuen sen pohjautumisesta jo olemassaoleviin kieliin. Dartin dokumentaatiossa on olemassa osio, jonka tavoitteena on opettaa lukijaa kirjoittamaan tehokasta koodia. Tämän osion yksi osa kertoo tyyleistä, joita kannattaa suosia, jotta koodi on yhtenäistä riippumatta kehittäjien määrästä. Toinen osion osa taas kertoo parhaat käytännöt dokumentointiin koodin sisällä. [26] Näiden lisäksi luettavuuteen vaikuttaa myös tietenkin sovellusarkkitehtuurin valinta ja se, miten hyvin näitä kaikkia seikkoja on noudatettu.

Widgetien uudellenkäytettävyys eri Flutter-projektien välillä on helppoa, sillä ne ovat omia komponenttejaan, jotka voidaan toteuttaa riippumattomiksi muista komponenteista. Flutterissa ei ole vielä olemassa vakaata ominaisuutta, jonka avulla Flutter-koodia voisi viedä toisella ohjelmistokehyksellä kirjoitettuun sovellukseen. Tämä ominaisuus on kehitysvaiheessa ja tulossa tulevaisuudessa sekä Androidille, että iOSille. [2]

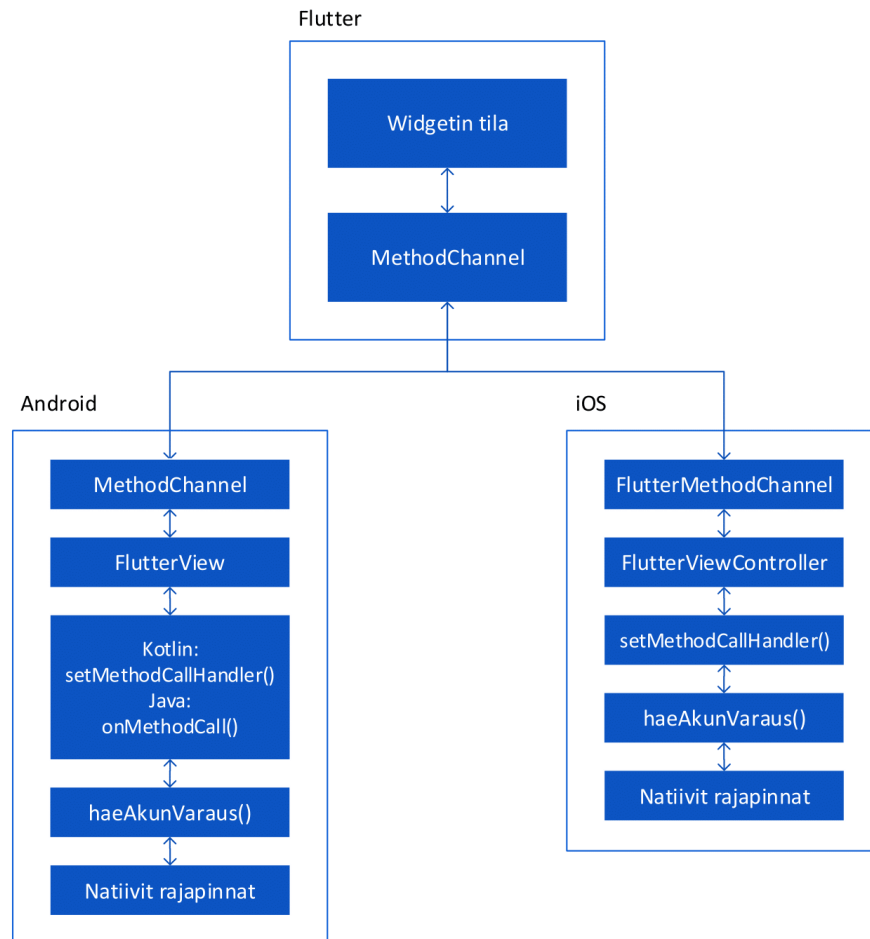
Suurena kysymyksenä teknologian ympärillä on se, kuinka hyvin natiiveja komponentteja jäljittelevät widgetit pidetään ajantasalla, jotta ne näyttävät ja toimivat samoin kuin natiivit. Tällä hetkellä tilanne näyttää valoisalta, sillä Flutteria kehitetään ahkerasti, mutta tulevaisuudesta ei ole varmuutta, koska widgetien ajantasaisuus riippuu täysin Flutterin nykyisistä ja tulevista kehittäjistä.

6.17 Laajennettavuus

Flutteria pystyy laajentamaan erilaisten kirjastojen avulla, joita voi ottaa yksitellen käyttöön omaan projektiin, kuten jo luvussa 4 kerrottiin. Peruserä on sama kuin esimerkiksi React Nativessa. Flutterin ja Dartin paketit löytyvät osoitteesta <https://pub.dev/>. Flutteria on mahdollista laajentaa kenen tahansa toimesta omilla kirjastoilla ja liitännäisillä.

6.18 Integraatio natiiville koodille

Flutterissa voidaan ottaa käyttöön joitain alustakohtaisia ominaisuuksia erilaisten kirjastojen avulla, mutta jos näitä ei ole valmiina kirjastoina tai liitännäisinä, tulee kehittäjän kirjoittaa itse natiivia koodia. Integraatio toimii siten, että Flutterista lähetetään viestejä iOS tai Android -puolelle `MethodChannel`-widgetin kautta, joka kommunikoi natiivin ja Flutter-puolen koodin välillä asynkronisesti. Kun `MethodChannel`-widgetin kautta vastaanotetaan viestejä natiivilla puolella, informaation avulla kutsutaan alustakohtaisia API-rajapintoja natiivin toteutuksen avulla. Tämän jälkeen Android tai iOS -puolelta lähetetään viesti takaisin Flutter-puolelle, jossa widgetin tilaa päivitetään. Kuvassa 6.3 esitellään tätä toimintaperiaatetta ja liitessä E on esimerkki akun varaustason hakemiselle natiivien rajapintojen avulla.



Kuva 6.3. Natiivien rajapintojen kutsuminen Flutterista.

6.19 Kehityksen nopeus

Kehitys Flutterilla on kohtalaisen nopeaa intuitiivisen syntaksin, sekä hyvän dokumentaation takia. Myös projektin pystyttäminen on nopeaa hyvien ohjeiden, sekä jo aiemmin se-

litetyn Flutter doctor -ohjelman takia. Valmiita toiminnallisuuksia Flutterilla ei kuitenkaan ole vielä lähellekkään yhtä paljon kuin esimerkiksi React Nativella, mikä hidastaa kehityksen nopeutta. Flutterille on julkaistu noin 4 000 kirjastoa, kun taas React Nativella niitä on noin nelinkertainen määrä.

6.20 Laitekohtaisten ominaisuuksien saatavuus

Flutter mahdollistaa sovelluksen pääsyn laitteen ominaisuuksiin hyvin, sillä ainakin perusominaisuudet ovat katettu eri kirjastojen avulla. Perusominaisuuksilla tarkoitetaan tässä laitteen kameraa [14], mikrofonia niin puheentunnistuksen [74] kuin nauhoittamisen [40] suhteen, sijaintia [42], sekä sensoreita [70].

6.21 Alustakohtaisten ominaisuuksien saatavuus

Osaan alustakohtaisista ominaisuuksista pääsee käsiksi suoraan Flutterin kirjastojen avulla. Niihin, joihin ei ole suoraa pääsyä, voi kehittäjät kirjoittaa suoraan natiivilla koodilla halutun toiminnallisuuden. Flutterissa on jo olemassaolevia liitännäisiä ainakin akun tilan [11] tutkimiseen, sekä selaimen avaamiseen suoraan sovelluksen sisältä [83].

Flutterin kautta on mahdollista myös säilöä dataa turvallisesti Flutterin tukemien liitännäisten avulla laitteeseen oman kirjaston avulla [73]. Kirjaston avulla voidaan muokata ja hakea tietoa iOSilla NSUserDefaults-rajapinnasta, sekä Androidilla SharedPreferences-rajapinnasta, jonka avulla dataa voidaan säilyttää pysyvästi. Rajapintoihin ei kuitenkaan tulisi säilöä sensitiivistä dataa ainakaan sellaisenaan.

6.22 Tuki liitetyille laitteille

Flutterissa on olemassa liitännäinen bluetoothille, jonka avulla laitteita voidaan yhdistää laitteeseen sovellusten kautta. Tämän lisäksi Flutterilla on myös mahdollista kehittää sovelluksia myös muun muassa älykelloihin. Tämä kuitenkin vaatii jonkin verran konfigurointia. [84]

6.23 Syötteen heterogeenisyys

Flutter tukee syötemuotoina ainakin näppäimistöä, kosketuksia, sekä ääniä ja eleitä. Kasvojen- ja sormenjäljentunnistusta voidaan käyttää käyttäjän tunnistautumiseen.

6.24 Ulostulomuotojen heterogeenisyys

Flutterin responsiivisuutta tutkitaan omassa kriteerissään, joten eri kokoisia näyttöjä tässä ei evaluoida. Flutter ei tue tällä hetkellä sovelluksen suoratoistoa esimerkiksi ChromeCastin kautta [87].

6.25 Sovelluksen elinkaari

Eri alustoilla sovelluksien elinkaaret ovat erilaisia. Androidilla elinkaari jaotellaan tiloihin onCreate, onStart, onResume, onPause, onStop ja onDestroy. iOSilla tuetut tilat ovat unattached, inactive, active, background ja suspended. Flutter ei tue kummankaan alustan kaikkia tiloja, vaan taulukkoon 6.2 on listattu Flutterin tukemat tilat ja tilat joihin ne linkittyvät Androidilla ja iOSilla.

Flutter	inactive	paused	resumed	suspending
Android	inactive	background	active	-
iOS	-	onPause()	onResume()	onStop()

Taulukko 6.2. Flutterin elinkaaren tiloja vastaavat tilat Androidilla ja iOSilla.

Kuten huomataan, Flutter ei tue kaikkia mahdollisia siirtymiä, minkä johdosta kehys ei pysty samoihin ominaisuuksiin kuin natiivit teknologiat, vaikka tärkeimmät tilat ovatkin tuettu.

6.26 Palvelin-selain-integraatio

Flutterin dokumentaatiossa on tuki palvelin- ja selainpuolen datansiirrolle JSON:n avulla. Sarjallistaminen voidaan tehdä joko manuaalisesti tai automaattisesti. Manuaalinen sarjallistaminen sopii pieniin projekteihin, joissa toimitaan yksinkertaisten JSON-objektien parissa. Monimutkaisempien objektien kanssa suositellaan kuitenkin käytettäväksi generaattoreita, jotka parsivat datan lähes automaattisesti käyttökelpoiseksi objektiksi. Esimerkiksi json_serializable-kirjasto [55] tarjoaa tällaisen generaattorin. Sarjallistamisen käyttöönotto, sekä manuaalisesti että automaattisesti, vaati konfigurointia, mutta siihen löytyy ohjeet Flutterin dokumentaatiosta. [54]

6.27 Turvallisuus

Flutter tarjoaa käyttäjän todentamiseen luotettavia liitännäisiä, kuten Google SignIn. Todentamisessa on suositeltavaa käyttää valmiita ratkaisuja omien sijasta varsinkin jos kyseessä on hieman kokemattomampi kehittäjä. Tällöin valmiit ratkaisut ehkäisevät yleisim-

piä virheitä. Profit Software Oy:n tapauksessa on kuitenkin parempi luoda omat ratkaisut käyttäjien hallinnoimiseen varsinkin finanssialan projekteissa, joissa sovelluksen turvallisuus on erityisen tärkeää, sillä kirjastoissa ja liitännäisissä piilee kuitenkin omat vaaransa.

Internetyhteyden kautta API-kutsuja tekevät sovellukset sisältävät yleensä jonkinlaisen sovelluksen sisäisen tietovaraston, jonka avulla vältytään hakemasta dataa hitaiden API-kutsujen kautta jatkuvasti. Dataa tulisi säilöä vain sen verran kuin on tarpeellista. Irrelevantti data tulisi siivota pois säännöllisesti esimerkiksi ajastimien avulla. Myös sovelluksen sulkemisen yhteydessä dataa tulisi poistaa. Datan siivoaminen onnistuu ohjelmassa 6.1 esitellyn esimerkin avulla.

```

1   void didChangeAppLifecycleState(AppLifecycleState state) {
2       if (state == AppLifecycleState.paused) {
3           new Future.delayed(
4               const Duration(minutes: 15), _cleanAllCache,
5           );
6       }
7   }

```

Ohjelma 6.1. Irrelevantin datan siivous välimuistista 15 minuutin välein

Sormenjäljentunnistus on yleistynyt viime aikoina tunnistautumismenetelmänä sovellusten sisäänkirjautumisessa. Vaikka sormenjälki ei olekaan yhtä vahva tunnistautumistapa kuin salasana, on se helppokäyttöisyytensä takia noussut suosituksi. Flutterissa sormenjäljentunnistus on mahdollista ottaa käyttöön liitännäisen [56] avulla.

Viimeksi käytettyjä sovelluksia pystyy yleensä selailemaan puhelimella niin, että niistä näytetään yksi näkymä pienennettynä. Tämä näkymä voi olla tarpeen suojata tilanteissa, joissa näytetään sensitiivistä dataa. Flutterista tähän ei löydy suoraan kirjastoa, sillä ominaisuus on täysin riippuvainen natiiveista rajapinnoista, sekä sovelluksen elinkaaresta. Flutterin avulla on kuitenkin mahdollista toteuttaa tämä suhteellisen helposti [69].

6.28 Sovelluksen liikutettavuus

Flutter on ainakin vielä pääasiallisesti keskittynyt tuottamaan mobiilisovelluksia, mutta sillä on myös valmiudet muillekin alustoille, kuten jo Tuetut kohdealustat -kriteerissä kerrottiin. Sulautetuille järjestelmille on oma API-dokumentaatio [21]. Osa Google Home Hub -sovelluksesta on toteutettu Flutterin sulautettujen järjestelmien tuella [80]. Google itse käyttää Flutteria Fuchsialle, joka on Googlen 2016 julkaisema käyttöjärjestelmä.

Puettaville laitteille (engl. wearables) on mahdollista tehdä sovelluksia Flutterilla, mutta se vaatii alustakohtaisen natiivin koodin kirjoittamista.

6.29 Sovelluksen ulkonäkö ja tuntuma

Kuten jo aiemminkin on todettu, Flutter käyttää omia widgetejä natiivien komponenttien sijaan. Nämä komponentit ovat kopioita natiiveista komponenteista. Komponentit eivät kuitenkaan ole rakennettu niin, että kehittäjä voisi vain valita käyttävänsä esimerkiksi dialogi-widgetiä ja teknologia osaisi tämän perusteella generoida näkymään oikean alusta dialogin alustakohtaisilla suunnitteluratkaisuilla, vaan kehittäjän tulee aina itse päättää käyttääkö Androidin vai iOSin widget-kirjastoa. Jos kehityksessä ei haluta tehdä kompromisseja, joutuu Flutter-kehittäjät kirjoittamaan eri alustoille eri toteutuksia. Tässä kuitenkin on hyvä huomioida ettei kaikkea koodia tarvitse kirjoittaa uudestaan, vaan vain alustakohtaisia ominaisuuksia kaipaavien komponenttien osat. Flutter pystyy selvittämään käytössä olevan alustan `defaultTargetPlatform()`-metodilla, jonka avulla alustakohtaisia suunniteluvalintoja voidaan tehdä. Joillekin ominaisuuksille Flutteriin on luotu automaattisesti oikein eri alustoilla toimivat widgetit. Tällainen ominaisuus on muun muassa navigointi.

6.29.1 Animaatiot ja transiitiot

Flutterissa on mahdollista luoda yksinkertaisten animaatioiden lisäksi myös hyvin monimutkaisia animaatioita. Esimerkkisovelluksen avulla pyrittiin tutustumaan erityisesti erilaisten animaatioiden ja transiitioiden käyttöön. Flutterissa animaatiot pohjautuvat [Animation](#)-objektiin, joka voidaan antaa suurimmalle osalle widgeteistä parametrina. Widgetit lukevat objektin nykyisen arvon, sekä seuraavat sen tilamuutoksia. Animaatiot voidaan jakaa kahteen osaan: Tween-animaatiot ja fysiikkaan pohjautuvat animaatiot. Tween-animaatioilla tarkoitetaan sellaisia, joilla on alku- ja loppupisteet, aikajana, sekä kaari, joka määrittää animaation ajastuksen, sekä nopeuden. Animaatioiden avulla on mahdollista luoda UX-suunnittelun perusanimaatioita, kuten animoitu lista, kahden sivun yhteisen elementin siirtyminen sivulta toiselle ja porrastetut animaatiot. Yhteisen elementin siirtyminen sivulta toiselle toteutetaan Flutterissa [Hero](#)-widgetin avulla. [Hero](#)-animaatiota ei ole mahdollista toteuttaa modaaleihin ainakaan suoraan samalla logiikalla kuin eri sivujen välille. [46] [Hero](#)-animaatioita käytettiin esimerkkisovelluksessa aloitusnäkyämästä ostotapahtuma-näkymään siirtymisessä.

Flutterissa transiitiot voidaan toteuttaa transiitioanimaatioihin tarkoitetuilla widgeteillä, kuten [FadeTransition](#), [SizeTransition](#) ja [SlideTransition](#). [7] Näille widgeteille kerrotaan vain alku- ja loppupisteet.

Flutterin animaatiot ja transiitiot kattavat hyvin ainakin perustarpeet. Suurimpaan osaan Flutterin widgeteistä on sisäänrakennettu jonkinlaisia widgetin liikuttamiseen liittyviä toiminnallisuuksia. Myös erilaisia animaatio-widgetejä on kattavammin kuin React Nativesa, jossa tuetaan oletuksena vain kahta animaatio-tyyppiä. [7] React Nativessa on kuitenkin olemassa useita kolmannen osapuolen toteutuksia monimutkaisemmille animaatioille.

6.30 Suorituskyky

Flutterin hyvä suorituskyky verrattuna React Nativeen verrattuna johtuu siitä, että Flutter käyttää omia widgetejään natiivien komponenttien sijasta. Tämän seurauksena React Nativesta tuttua React Native -siltaa ei tarvita, mikä nopeuttaa kääntämisprosessia, sillä Flutterin Dart-koodi käännetään suoraan natiiviksi lähdekoodiksi.

Dartissa käytetään sekä AOT-, että JIT-käännöstyyppiä, mikä myös parantaa ohjelmien suorituskykyä. AOT-kääntäjiä käyttävät kielet ovat yleensä nopeampia julkaistuissa sovelluksissa, kuitenkin hidastaen kehitystä merkittävästi. JIT-kääntäjät taas toimivat nopeammin kehittämisessä, mutta sovelluksen käynnistysaika on huomattavasti hitaampi. Flutter käyttää näppärästi hyödykseen kääntäjien parhaita puolia, sillä kehityksessä käytetään JIT-kääntäjää ja sovelluksen julkaisuvaiheessa AOT-kääntäjää. [63]

Animaatioissa Flutter pyrkii oletusarvoisesti saavuttamaan 60 fps suorituskyvyn, mutta jos laite tukee 120 fps suorituskykyä, on Flutterissa tuki myös siihen. React Native:ssä 60 fps saavutetaan usein vain kirjoittamalla natiivia koodia, sillä kommunikointi React Native -sillan kautta heikentää animaatioiden suorituskykyä merkittävästi. [36]

Flutterin sovellusten koot ovat aiheuttaneet paljon keskustelua ja ne ovatkin verrattaen suuria. Minimaalinen sovellus vie Androidille kehitettynä tilaa n. 4,5MB, mikä on huomattavasti suurempi kuin natiiveilla teknologioilla kirjoitetut. Flutterin moottori vie tästä tilasta n. 3,4MB, ohjelmistokehys, sekä sovelluksen koodi n. 900 KB, lisenssitiedosto 54 KB ja pakollinen Java-koodi n. 120 KB. iOS:llä sama sovellus vie n. 19,9 MB muistia, mikä johtuu enimmäkseen Applen tavasta salata tiedostoja. [34]

Suorituskyvyn maksimointia varten Flutter on koonnut dokumentaatioonsa hyviä käytänteitä, joiden avulla voi parantaa sovelluksensa suorituskykyä [62]. Flutterilla on myös työkaluja, joiden avulla voidaan tarkkailla sovelluksen suorituskykyä [36].

6.31 Käyttäjän autentikointi

Käyttäjän autentikointiin Flutter suosittelee käytettäväksi Googlen Firebaseia ja se onkin ainoa Flutterin dokumentaatioissa mainittu palvelinpuolen työkalu. Profitin näkökulmasta pakottaminen yhteen tiettyyn tapaan ei ole paras mahdollinen ratkaisu, sillä se saattaa toimia rajoittavana tekijänä osassa projekteja. Flutterilla on kuitenkin toteutettu myös sovelluksia, jotka käyttävät jo Profitille tuttuja työkaluja autentikoinnin kanssa, kuten AWS:n cognitoa.

7 YHTEENVETO

Mobiililaitteille tarkoitettujen sovellusten kehittämiseen on olemassa kaksi vaihtoehtoa: natiivit ja alustariippumattomat teknologiat. Natiivit teknologiat käyttävät alustoille optimoituja ohjelmointikieliä, joiden avulla saavutetaan usein paras loppusovellus. Natiivikehitys vie kuitenkin enemmän resursseja kuin alustariippumaton, sillä samaa koodipohjaa ei voi uudelleenkäyttää eri alustojen välillä. Alustariippumattomat teknologiat käyttävät yhtä koodipohjaa usealla eri alustalla. Yhden koodipohjan käyttäminen luo rajoitteita, sillä kaikkia eri alustojen erikoisominaisuuksia ei voida sen avulla saavuttaa. Esimerkiksi navigointi toimii eri alustoilla eri tavalla ja sen implementointi ei välttämättä ole mahdollista saman koodipohjan avulla. Näissä tilanteissa ratkaisuna on usein kirjoittaa natiivia koodia erikseen eri alustoille, mikä sotii monialustaisten teknologioiden peruseräitä vastaan.

Työssä tutkittiin Googlen kehittämää alustariippumatonta ohjelmistokehystä, Flutteria. Flutter käyttää ohjelmointikielenään myös Googlen kehittämää Dart-kieltä, joka on luokkajoinen, puhdas olioperusteinen ohjelmointikieli, varustettuna staattisella tyyppityksellä, sekä yksinkertaisella periytymisellä. Kieli on kehitetty muun muassa C# ja JavaScript kielen pohjalta ja se muistuttaakin ilmaisultaan niitä. Dart käyttää AOT-kääntäjää sovelluksen julkaisuvaiheessa ja JIT-kääntäjää kehitysvaiheessa, mikä mahdollistaa nopean kehityksen lisäksi myös nopeasti toimivan julkaistun sovelluksen.

Flutter-ohjelmistokehityksen pohjana toimivat teknologian omat komponentit eli widgetit. Teknologia ei käytä suoraan natiiveja komponentteja, kuten esimerkiksi React Native tekee, vaan sen sijaan se hyödyntää omia widgetejään. Flutterissa widgetit voidaan jakaa kolmeen osaan: peruswidgetit, Androidille tyypilliset ominaisuudet ja ulkonäön omaavat Material-widgetit, sekä iOS-käyttöjärjestelmää varten luodut Cupertino-widgetit. Näiden lisäksi widgetit voidaan jakaa vielä tilallisiin, tilattomiin ja periytyviin. Tilan hallinta Flutterissa voidaan toteuttaa tilallisten ja periytyvien widgetien avulla. Tilallisilla widgeteillä on nimensäkin mukaan tila, jota voidaan tarvittaessa muuttaa. Periytyvät widgetit taas voivat jakaa tilaansa lapsikomponenteilleen.

Työssä evaluoitiin Flutteria useiden evaluointikriteerien, sekä prototyyppisovelluksen avulla. Kriteereitä oli 31 ja ne jaettiin infrastruktuuri-, kehitys-, sovellus- ja käytettävyysskriteereihin. Prototyyppisovelluksessa luotiin lipunmyyntisovellus Profit Software Oy:lle, jonka avulla voi selata tulevia tapahtumia ja ostaa lippuja niihin. Sovellus rajoitettiin vain selainpuolen toteutukseen. Yritykselle tehty demosovellus oli mielekäs tapa tutustua Flutterin toimintaa konkreettisesti.

Tutkimuksen perusteella Flutter on suorituskyvyltään tehokas ohjelmistokehys. Tehokkuus johtuu teknologian uniikista arkkitehtuurista, jossa widgetit käännetään suoraan natiiviksi lähdekoodiksi, ilman esimerkiksi React Nativesta tuttua siltaa. Animaatioissa Flutter pyrkii vähintään 60 fps:n suorituskykyyn ja sillä on myös valmiudet 120 fps:n suorituskykyyn, mikä takaa sulavat animaatiot.

Flutterin jatkuva kehittyminen on sekä hyvä, että huono puoli. Jatkuva kehittyminen takaa nopeat korjaukset virheisiin, sekä uusien ominaisuuksien esittelyn tasaisin väliajoin. Kehitys voi aiheuttaa myös taaksepäin yhteensopimattomia ominaisuuksia, joita Flutterin tapauksessa on ollut lukuisia. Tämä johtuu siitä, että teknologiaa halutaan vielä kehittää niin, että API-rajapinta pysyy jatkossakin intuitiivisena, eikä niin sanottuja hotfixeja jätetä virallisiksi ratkaisuuksi. Teknologian jatkuva kehitys aiheutti myös diplomityön kirjoittamisessa haasteita, sillä noin puolivuotisen prosessin aikana teknologiasta on julkaistu neljä uutta, isoa versiota (v1.7.0-v.1.10). Tällä välillä Flutter on esimerkiksi esitelty sekä web-, että työpöytäsovellusten julkaisuun, minkä ansiosta teknologia voi tulevaisuudessa korvata jopa JavaScriptin.

Teknologia on helposti omaksuttavissa, sillä Flutterin dokumentaatio on laaja ja ohjelmointikieli Dart mukailee ilmaisultaan jo olemassaolevia kieliä. Helposti omaksuttava kieli ei vielä takaa sitä, että teknologiaa olisi nopea kehittää. Flutter on kilpailijoitaan vielä selvästi jäljessä erilaisten kirjastojen määrässä, mikä hidastaa kehitystä erityisesti monimutkaisimmassa ohjelmissa. Toisaalta pienissä projekteissa, joiden tarkoituksena on lähinnä todistaa suunniteltu konsepti oikeaksi, teknologia on varteenotettava vaihtoehto jopa Profitin tarkoituksiin.

Flutter on Googlen kehittämä ja suurin osa toteutuksista on Googlen omien kehittäjien tekemiä, minkä takia teknologia on hyvin riippuvainen yrityksen tuesta. Tässä vaiheessa ei ole vielä uskottavaa, että teknologia voisi pärjätä ilman sitä. Jos teknologia saa Googelta yhtä paljon huomioita tulevaisuudessa kuin tällä hetkellä, sen mahdollisuudet korvata JavaScript kasvavat merkittävästi. Toisaalta, jos Google lopettaa teknologiaan panostamisen, todennäköistä on, että teknologia unohtuu muiden samantyyppisten teknologioiden kanssa. Googlen tuki aiheuttaa myös kysymyksen siitä, haluaako yritys pakottaa kehittäjiä käyttämään muita Googlen sovelluksia Flutterin kehityksen yhteydessä, kuten esimerkiksi Firebasea. Tällä hetkellä tilanne näyttää kuitenkin ratkeavan erilaisten lisäkirjastojen avulla, joilla mahdollistetaan myös muiden kuin Googlen sovellusten käyttö.

Flutter on jo tällä hetkellä hyvä teknologia, joka sisältää useita hienoja ominaisuuksia. Teknologia ei kuitenkaan ole vielä niin kypsä, että sitä kannattaisi ottaa käyttöön finanssialalla, jossa teknologioilta odotetaan vakautta. Flutter on kuitenkin yksi niistä teknologioista, joita kannattaa pitää silmällä tulevaisuutta varten. Profitille teknologiaa voi suositella käytettäväksi jo heti pienissä POC-projekteissa. Suuremmissa projekteissa Flutteria ei kannata ottaa vielä käyttöön sen epävakauden takia.

LÄHTEET

- [1] *A tour of the Dart language*. URL: <https://dart.dev/guides/language/language-tour%5C#keywords> (viitattu 23. 07. 2019).
- [2] *Add Flutter to existing apps*. URL: <https://github.com/flutter/flutter/wiki/Add-Flutter-to-existing-apps> (viitattu 22. 10. 2019).
- [3] *ads 1.0.1*. URL: <https://pub.dev/packages/ads> (viitattu 01. 09. 2019).
- [4] *Alibaba Group Announces June Quarter 2019 Results*. URL: https://www.alibabagroup.com/en/news/press_pdf/p190815.pdf (viitattu 08. 09. 2019).
- [5] A. Amatya Suyesh ja Kurti. *Cross-Platform Mobile Development: Challenges and Opportunities* (2014). Toim. M. Trajkovik Vladimir ja Anastas, 219–229.
- [6] P. R. M. Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira ja F. A. da Silva. *Cross platform app: a comparative study*. *ArXiv* 7 (2015).
- [7] *Animation and motion widgets*. URL: <https://flutter.dev/docs/development/ui/widgets/animation> (viitattu 22. 10. 2019).
- [8] *app_review 1.0.0*. URL: https://pub.dev/packages/app_review (viitattu 01. 09. 2019).
- [9] *Apps take flight with Flutter*. URL: <https://flutter.dev/showcase> (viitattu 08. 10. 2019).
- [10] L. Bak. *Dart: a language for structured web programming*. 10. lokakuuta 2011. URL: <http://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html> (viitattu 23. 07. 2019).
- [11] *battery 0.3.0+5*. URL: <https://pub.dev/packages/battery> (viitattu 16. 09. 2019).
- [12] *Bitrise*. URL: <https://www.bitrise.io/> (viitattu 01. 10. 2019).
- [13] *Building web apps in WebView*. URL: <https://developer.android.com/guide/webapps/webview> (viitattu 25. 08. 2019).
- [14] *camera 0.5.4+2*. URL: <https://pub.dev/packages/camera> (viitattu 16. 09. 2019).
- [15] *Can to draw text at an angle using flutter?* URL: <https://github.com/flutter/flutter/issues/15710%5C#issuecomment-470418200> (viitattu 23. 07. 2019).
- [16] *Codelabs*. URL: <https://flutter.dev/docs/codelabs> (viitattu 22. 10. 2019).
- [17] *COLLECTION Front-end JavaScript frameworks*. URL: <https://github.com/collections/front-end-javascript-frameworks> (viitattu 07. 10. 2019).
- [18] *Continuous delivery with Flutter*. URL: <https://flutter.dev/docs/deployment/cd> (viitattu 16. 09. 2019).
- [19] *Cookbook*. URL: <https://flutter.dev/docs/cookbook> (viitattu 22. 10. 2019).
- [20] *Creating responsive apps*. URL: <https://flutter.dev/docs/development/ui/layout/responsive> (viitattu 15. 09. 2019).
- [21] *Custom Flutter Engine Embedders*. URL: <https://github.com/flutter/flutter/wiki/Custom-Flutter-Engine-Embedders> (viitattu 03. 10. 2019).
- [22] *Dart documentation*. URL: <https://dart.dev/guides> (viitattu 15. 09. 2019).

- [23] *Dart Programming Language Specification 5th edition draft*. URL: <https://dart.dev/guides/language/specifications/DartLangSpec-v2.2.pdf> (viitattu 06. 10. 2019).
- [24] *Dart versus Node js fastest programs*. URL: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/dart.html> (viitattu 07. 10. 2019).
- [25] *Developer Survey Results 2019*. URL: <https://insights.stackoverflow.com/survey/2019%5C#technology> (viitattu 07. 10. 2019).
- [26] *Effective Dart*. URL: <https://dart.dev/guides/language/effective-dart> (viitattu 23. 07. 2019).
- [27] A. P. ja Evelina Vorobyeva. *Evaluation of Cross-Platform Tools for Mobile Development* (2013).
- [28] *fastlane*. URL: <https://docs.fastlane.tools/> (viitattu 16. 09. 2019).
- [29] *Figma to Flutter*. URL: <https://www.figma.com/resources/api-and-extensions/figma-to-flutter/> (viitattu 02. 10. 2019).
- [30] *Flutter API reference documentation*. URL: <https://api.flutter.dev/index.html> (viitattu 15. 09. 2019).
- [31] *Flutter build release channels*. URL: <https://github.com/flutter/flutter/wiki/Flutter-build-release-channels> (viitattu 21. 09. 2019).
- [32] *Flutter Contributors*. URL: <https://github.com/flutter/flutter/graphs/contributors?from=2014-10-19&to=2019-10-01&type=c> (viitattu 01. 10. 2019).
- [33] *Flutter Documentation*. URL: <https://flutter.dev/docs> (viitattu 15. 09. 2019).
- [34] *Flutter FAQ*. URL: <https://flutter.dev/docs/resources/faq> (viitattu 19. 07. 2019).
- [35] *Flutter for React Native developers*. URL: <https://flutter.dev/docs/get-started/flutter-for/react-native-devs> (viitattu 29. 08. 2019).
- [36] *Flutter performance profiling*. URL: <https://flutter.dev/docs/testing/ui-performance> (viitattu 04. 10. 2019).
- [37] *Flutter studio*. URL: <https://flutterstudio.app/> (viitattu 02. 10. 2019).
- [38] *Flutterin koneiston lisenssit*. URL: https://raw.githubusercontent.com/flutter/engine/master/sky/packages/sky_engine/LICENSE (viitattu 21. 09. 2019).
- [39] *flutter_paystack 1.0.1*. URL: https://pub.dev/packages/flutter_paystack (viitattu 01. 09. 2019).
- [40] *flutter_sound 1.4.5*. URL: https://pub.dev/packages/flutter%5C_sound (viitattu 16. 09. 2019).
- [41] B. Frankston. *Progressive Web Apps [Bits Versus Electrons]*. *IEEE Consumer Electronics Magazine* 7.2 (2018), 106–117. ISSN: 2162-2248.
- [42] *geolocator 5.1.3*. URL: <https://pub.dev/packages/geolocator> (viitattu 16. 09. 2019).
- [43] *GitLab CI/CD*. URL: <https://docs.gitlab.com/ee/ci/> (viitattu 01. 10. 2019).
- [44] *Godemagic*. URL: <https://codemagic.io/start/> (viitattu 01. 10. 2019).
- [45] H. Heitkötter, S. Hanschke ja T. A. Majchrzak. *Evaluating Cross-Platform Development Approaches for Mobile Applications*. *Web Information Systems and Technologies*. Toim. J. Cordeiro ja K.-H. Krempels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, 120–138. ISBN: 978-3-642-36608-6.

- [46] *Hero Animations*. URL: <https://flutter.dev/docs/development/ui/animations/hero-animations> (viitattu 04. 10. 2019).
- [47] I. Hickson. *The Mahogany Staircase - Flutter's Layered Design*. Youtube. 2016. URL: <https://www.youtube.com/watch?v=dkyY9WCGMi0>.
- [48] *How to develop and distribute iOS apps without Mac with Flutter Codemagic*. 9. maaliskuuta 2019. URL: <https://blog.codemagic.io/how-to-develop-and-distribute-ios-apps-without-mac-with-flutter-codemagic/> (viitattu 01. 10. 2019).
- [49] *How Trigger.io works...* URL: <https://trigger.io/how-it-works/> (viitattu 17. 10. 2019).
- [50] *InheritedWidget class*. URL: <https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html> (viitattu 06. 10. 2019).
- [51] *intl 0.16.0*. URL: <https://pub.dev/packages/intl> (viitattu 22. 10. 2019).
- [52] *Introduction to widgets*. URL: <https://flutter.dev/docs/development/ui/widgets-intro%5C#keys> (viitattu 06. 10. 2019).
- [53] *Ionic docs. Web View*. URL: <https://ionicframework.com/docs/building/webview> (viitattu 17. 10. 2019).
- [54] *JSON and serialization*. URL: <https://flutter.dev/docs/development/data-and-backend/json> (viitattu 22. 10. 2019).
- [55] *json_serializable 3.2.2*. URL: https://pub.dev/packages/json%5C_serializable (viitattu 03. 10. 2019).
- [56] *local_auth 0.6.0+1*. URL: https://pub.dev/packages/local%5C_auth (viitattu 02. 10. 2019).
- [57] *Milestones*. URL: <https://github.com/flutter/flutter/milestones> (viitattu 21. 09. 2019).
- [58] *Mobile Operating System Market Share Worldwide*. URL: <http://gs.statcounter.com/os-market-share/mobile/worldwide/%5C#monthly-200901-201907> (viitattu 14. 07. 2019).
- [59] *mockito 4.1.1*. URL: <https://pub.dev/packages/mockito> (viitattu 22. 10. 2019).
- [60] T. V. Niclas Hansson. Effects on performance and usability for cross-platform application development using React Native. PhD thesis (2016), 4–6.
- [61] *num class*. URL: <https://api.dartlang.org/stable/2.5.1/dart-core/num-class.html> (viitattu 06. 10. 2019).
- [62] *Performance best practices*. URL: <https://flutter.dev/docs/testing/best-practices> (viitattu 04. 10. 2019).
- [63] *Platforms*. URL: <https://dart.dev/platforms> (viitattu 07. 10. 2019).
- [64] *Progressive Web Apps*. URL: <https://developers.google.com/web/progressive-web-apps/> (viitattu 22. 07. 2019).
- [65] C. P. Rahul Raj ja Seshu Babu Tolety. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 Annual IEEE India Conference (INDICON)*. 2012, 625–629.

- [66] *React Native Internals*. URL: <https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html> (viitattu 01. 10. 2019).
- [67] *React Nativen kotisivu*. URL: <https://facebook.github.io/react-native/> (viitattu 22. 09. 2019).
- [68] *Release Notes Flutter 1.9.1*. URL: <https://github.com/flutter/flutter/wiki/Release-Notes-Flutter-1.9.1> (viitattu 01. 10. 2019).
- [69] *Securing Flutter Apps*. URL: https://medium.com/@mehmetf%5C_71205/securing-flutter-apps-ada13e806a69 (viitattu 02. 10. 2019).
- [70] *sensors 0.4.0+1*. URL: <https://pub.dev/packages/sensors> (viitattu 16. 09. 2019).
- [71] *Service Worker API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API (viitattu 22. 07. 2019).
- [72] *Set up an editor*. URL: <https://flutter.dev/docs/get-started/editor?tab=vscode> (viitattu 01. 09. 2019).
- [73] *shared_preferences 0.5.3+4*. URL: https://pub.dev/packages/shared%5C_preferences (viitattu 02. 10. 2019).
- [74] *speech_recognition 0.3.0+1*. URL: https://pub.dev/packages/speech_recognition (viitattu 16. 09. 2019).
- [75] *StatefulWidget class*. URL: <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html> (viitattu 06. 10. 2019).
- [76] *StatelessWidget class*. URL: <https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html> (viitattu 06. 10. 2019).
- [77] *store_redirect 1.0.1*. URL: https://pub.dev/packages/store%5C_redirect (viitattu 22. 09. 2019).
- [78] S. Suri. *Architect your Flutter project using BLOC pattern*. 26. elokuuta 2018. URL: <https://medium.com/flutterpub/architecting-your-flutter-project-bd04e144a8f1> (viitattu 15. 09. 2019).
- [79] *Tablet Support*. URL: <https://github.com/flutter/flutter/issues/12310> (viitattu 01. 10. 2019).
- [80] F. Team. *Flutter: a Portable UI Framework for Mobile, Web, Embedded, and Desktop*. 7. maaliskuuta 2019. URL: <https://developers.googleblog.com/2019/05/Flutter-io19.html> (viitattu 03. 10. 2019).
- [81] C. R. ja Tim A. Majchrzak. Towards the definitive evaluation framework for cross-platform app development approaches. *Journal of Systems and Software* 153 (2019), 175–199. ISSN: 0164-1212.
- [82] *TIOBE Index for September 2019*. URL: <https://www.tiobe.com/tiobe-index/> (viitattu 22. 09. 2019).
- [83] *url_launcher 5.1.3*. URL: https://pub.dev/packages/url%5C_launcher (viitattu 16. 09. 2019).
- [84] *wear 0.0.3*. URL: <https://pub.dev/packages/wear> (viitattu 04. 10. 2019).
- [85] *Who's using React Native?* URL: <https://facebook.github.io/react-native/showcase/> (viitattu 08. 10. 2019).

- [86] *Widget-maker*. URL: https://norbert515.github.io/widget_maker/website/ (viitattu 02.10.2019).
- [87] *Would like a Cast / ChromeCast plugin for Flutter #18212*. URL: <https://github.com/flutter/flutter/issues/18212> (viitattu 03.10.2019).
- [88] *Xamarin.Forms*. URL: <https://dotnet.microsoft.com/apps/xamarin/xamarin-forms> (viitattu 17.10.2019).

A KAHDEN ERI WIDGETIN PAIKAN VAIHTAMINEN NAPIN PAINALLUKSESTA

Tässä liitteessä esitellään ohjelmakoodi sovellukselle, jossa vaihdetaan kahden `Tile`-widgetin paikkaa napin painalluksesta. Esimerkissä A.1 on kaksi eri versiota sovelluksesta. Versio, jossa ei oteta huomioon kommentoituja osuuksia (rivit 13, 14, 37) sovellus ei toimi niin kuin pitäisi, sillä `Tile`-widgetien paikat eivät vaihdu. Kommentoitujen osuuk-
sien kanssa sovellus vaihtaa `Tile`-widgetien paikkoja.

```

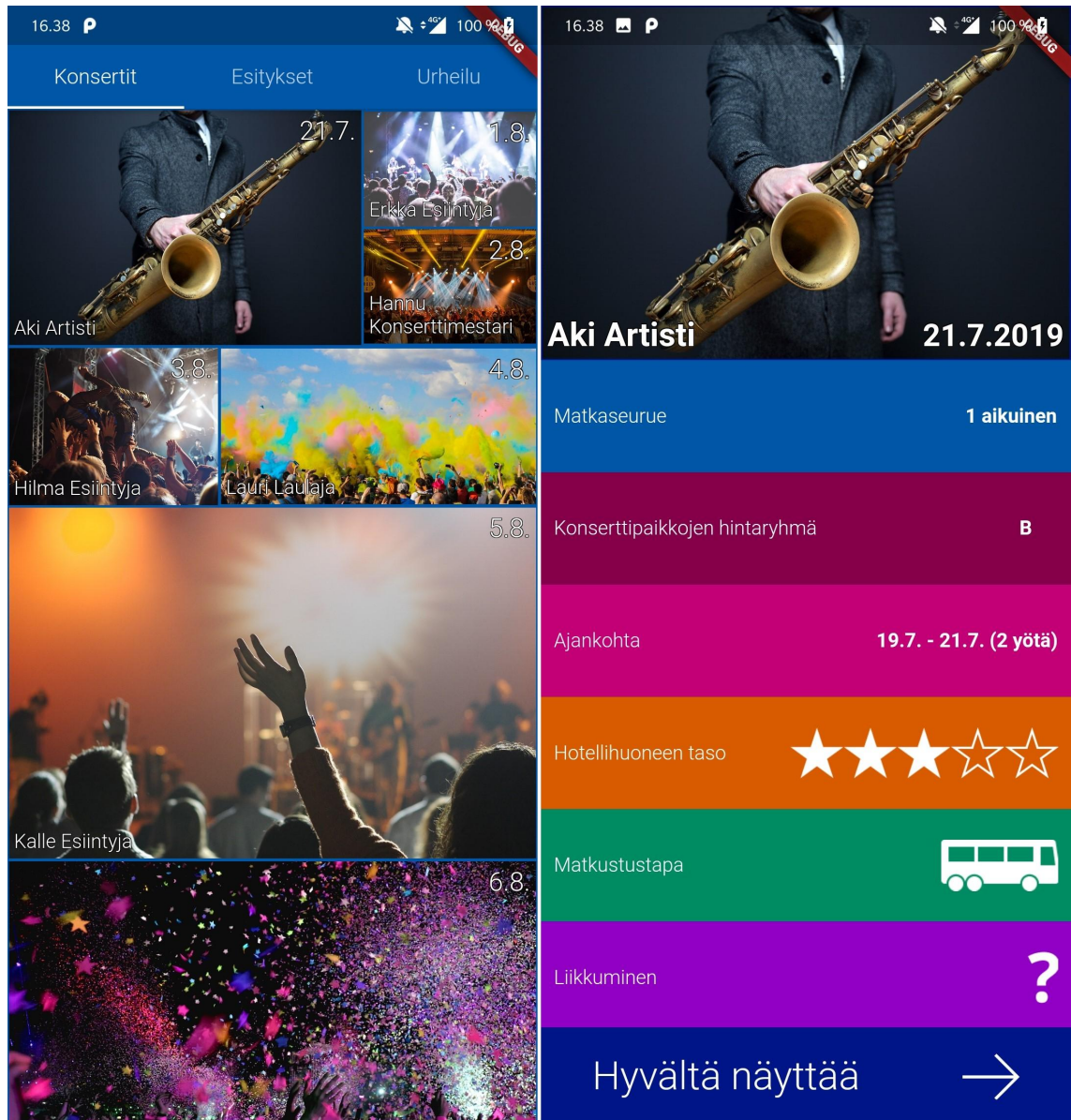
1  import 'package:flutter/material.dart';
2  import 'package:random_color/random_color.dart';
3
4  void main() => runApp(new MaterialApp(home: TileEsimerkki()));
5
6  class TileEsimerkki extends StatefulWidget {
7    @override
8    State<StatefulWidget> createState() => TileEsimerkkiTila();
9  }
10
11 class TileEsimerkkiTila extends State<TileEsimerkki> {
12   list<Widget> widgetit = [
13     TileWidget(/*key: UniqueKey()*/),
14     TileWidget(/*key: UniqueKey()*/),
15   ];
16
17   @override
18   Widget build(BuildContext context) {
19     return Scaffold(
20       body: Row(
21         children: widgetit,
22       ),
23       floatingActionButton: FloatingActionButton(
24         child: Icon(Icons.autorenew),
25         onPressed: vaihdaWidgetienPaikkaa,
26       ),
27     );
28   }
29   vaihdaWidgetienPaikkaa() {
30     setState(() {
31       widgetit.insert(1, widgetit.removeAt(0));

```

```
32     });
33   }
34 }
35
36 class TileWidget extends StatefulWidget {
37   //TileWidget(Key key) : super(key: key);
38   @override
39   State<StatefulWidget> createState() => TileWidgetTila();
40 }
41
42 class TileWidgetTila extends State<TileWidget> {
43   Color vari;
44
45   @override
46   void initState() {
47     super.initState();
48     RandomColor randomVari = RandomColor();
49     vari = randomVari.randomColor();
50   }
51
52   @override
53   Widget build(BuildContext context) {
54     return Tile(
55       color: vari, child: Padding(padding: EdgeInsets.all(70.0)));
56   }
57 }
```

Ohjelma A.1. Esimerkkisovellus, jonka avulla voidaan vaihtaa kahden *Tile*-widgetin paikkaa napin painalluksesta.

B KUVIA DEMOSOVELLUKSESTA



Kuva B.1. Kuvia demosovellukseen tehdyistä näkymistä.

The image shows a travel booking interface with the following sections:

- Matkaseurue (Travel group):** A blue section with the text "1 aikuinen" (1 adult). It features a slider with five human icons above and five below, indicating the number of travelers.
- Hotellihuoneen taso (Hotel room level):** An orange section with the text "3 tähteä" (3 stars). It features a slider with five star icons, where the first three are filled and the last two are empty.
- Matkustustapa (Travel method):** A green section with the text "Bussi" (Bus). It features a slider with four icons: a bus, a train, a car, and an airplane, with the bus icon selected.
- Ajankohta (Travel dates):** A pink section with the text "19.7. - 21.7. (2 yötä)" (19.7. - 21.7. (2 nights)). It features a date range slider with days labeled "Pe", "La", "Su 21.7.", "Ma", and "Ti". Below the slider, it says "Saapuminen" (Arrival) and "Lähtö" (Departure).
- Liikkuminen (Transportation):** A purple section with a question mark. It features a slider with five icons: a question mark, a bus, a taxi, a car, and a train, with the question mark icon selected.

Kuva B.2. Demosovelluksen eri osiot laajennettuina.

C FLUTTERIN LISENSSI

Copyright 2014 The Chromium Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

D ESIMERKKI ERI YMPÄRISTÖJEN KONFIGUROINNISTA FLUTTERIN AVULLA

Tässä esimerkissä esitellään lähdekoodi, jonka avulla eri ympäristöissä voidaan konfiguroinnin avulla näyttää käyttäjälle aina kyseisen ympäristön nimi.

```
1 // Tiedoston polku: lib/konfiguraatio.dart
2 import 'package:flutter/material.dart';
3 import 'package:meta/meta.dart';
4
5 // InheritedWidget-widgetin avulla Konfiguraatio-widgetiin päästään helposti
6 // käsiksi muista widgeteistä.
7 class Konfiguraatio extends InheritedWidget {
8   Konfiguraatio({
9     @required this.ymparistonNimi,
10    @required Widget child
11  }) : super(child: child);
12
13   final String ymparistonNimi;
14
15   static Konfiguraatio of(BuildContext context) {
16     return context.inheritFromWidgetOfExactType(Konfiguraatio);
17   }
18
19   @override
20   bool updateShouldNotify(InheritedWidget vanhaWidget) => false;
21
22 }
```

Ohjelma D.1. Konfiguraatitiedosto, jossa määritellään eri ympäristöjen välillä konfiguroitavat muuttujat.

```

1 // Tiedoston polku: lib/main_tuotanto.dart
2 import 'package:flutter/material.dart';
3 import 'konfiguraatio.dart';
4 import 'main.dart';
5
6 void main() {
7   var konfiguroituSovellus = new Konfiguraatio(
8     ymparistonNimi: 'tuotanto',
9     child: new Tervetuloa());
10
11   runApp(konfiguroituSovellus);
12 }

```

Ohjelma D.2. Tuotantoympäristön konfigurointi

```

1 // Tiedoston polku: lib/main_kehitys.dart
2 import 'package:flutter/material.dart';
3 import 'konfiguraatio.dart';
4 import 'main.dart';
5
6 void main() {
7   var konfiguroituSovellus = new Konfiguraatio(
8     ymparistonNimi: 'kehitys',
9     child: new Tervetuloa());
10
11   runApp(konfiguroituSovellus);
12 }

```

Ohjelma D.3. Kehitysympäristön konfigurointi

```

1 // Tiedoston polku: lib/main.dart
2 import 'package:flutter/material.dart';
3 import 'package:flutter_app/tervetuloaSivu.dart';
4 import 'konfiguraatio.dart';
5
6 class Tervetuloa extends StatelessWidget {
7   @override
8   Widget build(BuildContext context) {
9     var konfiguraatiot = Konfiguraatio.of(context);
10    return MaterialApp(
11      title: konfiguraatiot.ymparistonNimi,
12      theme: ThemeData(
13        primarySwatch: Colors.red,
14      ),
15      home: TervetuloaSivu(),
16    );
17  }
18 }

```

Ohjelma D.4. Varsinainen sovellus

```
1 // Tiedoston polku: lib/tervetulooSivu.dart
2 import 'package:flutter/material.dart';
3 import 'konfiguraatio.dart';
4
5 class TervetulooSivu extends StatefulWidget {
6   @override
7   _TervetulooSivu createState() => _TervetulooSivu();
8 }
9
10 class _TervetulooSivu extends State<TervetulooSivu> {
11   @override
12   Widget build(BuildContext context) {
13     var konfiguraatiot = Konfiguraatio.of(context);
14
15     return new Scaffold(
16       appBar: AppBar(
17         title: Text(konfiguraatiot.ymparistonNimi)
18       ),
19       body: Center(
20         child: Text('Tervetuloa!')
21       )
22     );
23   }
24 }
```

Ohjelma D.5. *TervetulooSivu*-widget, jossa käytetään kofiguroitua ympäristön nimeä.

E AKUN VARAUSTASON HAKEMINEN NATIIVEISTA RAJAPINNOISTA

Tässä liitteessä esitellään sovellus, jonka tarkoitus on hakea natiiveista rajapinnoista tieto laitteen akun varaustasosta ja näyttää se käyttäjälle. Tässä esimerkissä natiiveja rajapinnoja kutsutaan Kotlinin avulla.

```
1 import 'package:flutter/material.dart';
2 import 'package:flutter/services.dart';
3
4 void main() => runApp(AkunVaraustasoEsimerkki());
5
6 class AkunVaraustasoEsimerkki extends StatelessWidget {
7   @override
8   Widget build(BuildContext context) {
9     return MaterialApp(
10      title: 'Akun varaustaso demo',
11      theme: ThemeData(
12        primarySwatch: Colors.blue,
13      ),
14      home: AkunTiedot(),
15    );
16  }
17 }
18
19 class AkunTiedot extends StatefulWidget {
20   AkunTiedot({Key key}) : super(key: key);
21
22   @override
23   _AkunTiedotState createState() => _AkunTiedotState();
24 }
25
26 class _AkunTiedotState extends State<AkunTiedot> {
27   static const alusta = const MethodChannel('samples.flutter.dev/battery');
28   String _akunVaraustaso = 'Tuntematon akun varaustaso';
29
30   Future<void> _haeAkunVaraustaso() async {
31     String akunVaraustaso;
32     try {
33       // Haetaan natiiveista rajapinnoista metodia heaAkunVaraus
```

```

34     final int taso = await alusta.invokeMethod('haeAkunVaraus');
35     akunVaraustaso = 'Akun varaustaso on $taso %.';
36 } on PlatformException catch (e) {
37     akunVaraustaso =
38         'Akun varaustason hakeminen epäonnistui. Virheviesti: ${e.message}.';
39 }
40
41 setState(() {
42     _akunVaraustaso = akunVaraustaso;
43 });
44 }
45
46 @override
47 Widget build(BuildContext context) {
48     return Material(
49         child: Center(
50             child: Column(
51                 mainAxisAlignment: MainAxisAlignment.spaceEvenly,
52                 children: <Widget>[
53                     RaisedButton(
54                         child: Text('Hae akun varaustaso'),
55                         onPressed: _haeAkunVaraustaso,
56                     ),
57                     Text(_akunVaraustaso),
58                 ],
59             ),
60         );
61 }
62 }
63 }

```

Ohjelma E.1. Flutter-puolen toteutus sovellukselle.

```

1 // Tiedostopolku: akunTasoSovellus/android/app/src/
2 // main/kotlin/com/example/akunTasoSovellus/MainActivity.kt
3 package com.example.akunTasoSovellus
4
5 import android.content.Context
6 import android.content.ContextWrapper
7 import android.content.Intent
8 import android.content.IntentFilter
9 import android.os.BatteryManager
10 import android.os.Build
11 import android.os.Bundle
12
13 import io.flutter.app.FlutterActivity
14 import io.flutter.plugin.common.MethodChannel
15 import io.flutter.plugins.GeneratedPluginRegistrant
16

```

```

17 class MainActivity: FlutterActivity() {
18     private val CHANNEL = 'samples.flutter.dev/battery'
19
20     override fun onCreate(savedInstanceState: Bundle?) {
21         super.onCreate(savedInstanceState)
22         GeneratedPluginRegistrant.registerWith(this)
23         MethodChannel(flutterView, CHANNEL).setMethodCallHandler { call, result ->
24             if (call.method == 'haeAkunVaraus') {
25                 val batteryLevel = haeAkunVaraus()
26
27                 if (batteryLevel != -1) {
28                     result.success(batteryLevel)
29                 } else {
30                     result.error('EISAATAVILLA', 'Akun tasoa ei voitu hakea.', null)
31                 }
32             } else {
33                 result.notImplemented()
34             }
35         }
36     }
37
38     private fun haeAkunVaraus(): Int {
39         val akunVaraustaso: Int
40         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
41             val akkuManager = getSystemService(Context.BATTERY_SERVICE) as
42                 BatteryManager
43             akunVaraustaso =
44                 akkuManager.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY)
45         } else {
46             val intent = ContextWrapper(applicationContext).registerReceiver(null,
47                 IntentFilter(Intent.ACTION_BATTERY_CHANGED))
48             akunVaraustaso = intent!!.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) *
49                 100 / intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1)
50         }
51
52         return akunVaraustaso
53     }
54 }

```

Ohjelma E.2. Natiivien rajapintojen kutsuminen Kotlinilla.