Tampereen yliopisto

Ville Karjalainen

KAHDEN SOVELLUSKEHYKSEN VERTAILU

Flutter ja React Native

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Huhtikuu 2019

TIIVISTELMÄ

Ville Karjalainen: Kahden sovelluskehyksen vertailu. Flutter ja React Native
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Huhtikuu 2019

Mobiililaitteiden määrä on kasvanut tasaista tahtia ja niille tehtävien sovelluksien määrä kasvaa myös. Mobiilisovelluksia voidaan kehittää joko natiivisovelluksena tai hybridisovelluksena. Natiivisovellukset tarjoavat parhaan käyttökokemuksen, mutta niiden kehittäminen vaatii enemmän resursseja. Hybridisovelluksien kehittäminen puolestaan tarjoaa nopean ja helpomman tavan kehittää sovellus, mutta se näkyy sovelluksen suorituskyvyssä negatiivisesti.

Hybridisovelluksien kehittämiseen on useita erilaisia sovelluskehyksiä ja niiden toimintaperiaatteet vaihtelevat paljon. Vaihtoehtoja ovat esimerkiksi web-teknologioita hyödyntävät sovelluskehykset tai natiivikoodia generoivat sovelluskehykset. Näistä jälkimmäinen tarjoaa paremman suorituskyvyn sekä käyttäjäkokemuksen, koska ne usein hyödyntävät alustakohtaisia ominaisuuksia ja rajapintoja.

Työssä vertaillaan kahta alustariippumattomien sovelluksien kehittämiseen tarkoitettua sovelluskehystä: Flutteria ja React Nativea. React Native on sovelluskehyksistä laajemmin käytetty ja Flutter uusi tulokas. Sovelluskehyksiä vertaillaan suorituskyvyn, muistinkulutuksen ja sovelluksien kokojen perusteella.

Vertailun perusteella Flutter osoittautui tehokkaammaksi sovelluskehykseksi kaikilla muilla mittareilla, paitsi käynnistymisajalla. Vertailu toteutettiin typistetyille sovelluksille ja pelkästään Android-alustalle. Työtä olisikin mielekästä jatkaa laajentamalla sovelluksia ja käytettyjä alustoja, jotta sovelluskehyksiä voitaisiin vertailla kattavammin.

Avainsanat: Mobiilisovellus, sovelluskehys, hybridisovellus, Flutter, React Native

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
iOS	iPhone OS, Applen kehittämä käyttöjärjestelmä
SDK	Software Development Kit

1. JOHDANTO

Mobiililaitteiden osuus koko maailman verkkoliikenteestä on tällä hetkellä lähes puolet [1]. Mobiililaitteiden suuri määrä ja laitteiden ominaisuuksien parantuminen tarjoavat houkuttelevat markkinat sovelluskehittäjille. Sovelluksien kehittäminen ei kuitenkaan ole suoraviivaista, vaan siihen on useita vaihtoehtoja ja valintaan sisältyy kompromisseja. Vaihtoehtoja sovelluksien kehittämiseen ovat esimerkiksi natiivisovellus tai hybridisovellus.

Natiivisovelluksien kehittäminen tarjoaa parhaan suorituskyvyn ja käyttäjäkokemuksen, mutta niiden kehittäminen vaatii paljon resursseja ja aikaa. Lisäksi, jos sovellus halutaan toteuttaa usealle alustalle, tulee työ tehdä lähes kokonaan uudelleen. Hybridisovellukset puolestaan ovat nopeampia kehittää, ja ne tuottavat usealla alustalla toimivan sovelluksen. Niiden suorituskyky ja ulkoasu eivät kuitenkaan tarjoa parasta mahdollista käyttäjäkokemusta. Tästä huolimatta hybridisovelluksien kehittäminen on saanut paljon huomiota, koska usealle alustalle yhdellä koodipohjalla kehittäminen on erittäin suuri etu. Viime vuosien aikana onkin ilmestynyt useita sovelluskehyksiä, jotka mahdollistavat usealle alustalle kehittämisen yhdellä koodipohjalla. Sovelluskehyksien toimintatavat vaihtelevat erittäin paljon, mutta suosituimpia ovat olleet natiivikoodia hyödyntävät ja generoivat toteutukset.

Tässä työssä keskitytään tarkastelemaan eri tavalla natiivikoodia hyödyntäviä sovelluskehyksiä, React Nativea ja Flutteria. Molemmat sovelluskehykset pyrkivät yhdistämään parhaat puolet natiivi- ja hybridisovelluksien kehittämisestä ilman kompromisseja. Sovelluskehyksiä vertaillaan tarkastelemalla niillä tuotettujen sovelluksien käynnistymisaikaa, keskusyksikön käyttöprosenttia, ajon aikana varattua keskusmuistia ja sovelluksien kokoa.

Luvussa 2 käydään tarkemmin läpi natiivisovelluksien sekä hybridisovelluksien kehittämisen edut ja haitat. Luvussa 3 esitellään vertailtavat sovelluskehykset ja niiden toimintatavat. Luvussa 4 esitellään vertailun mittarit, suoritetaan sovelluskehyksien vertailu ja käydään läpi vertailun tulokset, sekä pohditaan tuloksien oikeellisuutta. Viimeisenä luvussa 5 tehdään yhteenveto työstä.

2. MOBIILISOVELLUKSIEN KEHITTÄMINEN

Mobiilisovelluksien kehittämiseen on useita eri vaihtoehtoja, kuten natiivisovellus, websovellus ja hybridisovellus. Tämän luvun alaluvuissa käsitellään näistä vaihtoehdoista natiivisovelluksien ja hybridisovelluksien toimintaperiaatteet sekä edut ja haitat.

2.1 Natiivit mobiilisovellukset

Natiivit mobiilisovellukset kehitetään yhdelle alustalle kerrallaan käyttäen valitun alustan ohjelmointikieltä ja ympäristöä. Usein kuitenkin halutaan toteuttaa sovellus myös toiselle alustalle laajemman markkinaosuuden saavuttamiseksi. Toiselle alustalle natiivisovelluksen kehittäminen pitää aloittaa alusta alkaen, jolloin sama työ sovelluksen kehittämiseksi tehdään toisen kerran. Aiemmin saman natiivisovelluksen toteuttaminen usealle alustalle olisi vaatinut ainakin kolmea eri toteutusta: Android, iOS (iPhone OS) ja Windows. Nykyisin sovelluksia kehitetään käytännössä enää Android- ja iOS-laitteille, sillä molempien yhteenlaskettu osuus käytössä olevista käyttöjärjestelmistä on noin 97 % [2]. Tämän lisäksi Microsoft ilmoitti 2019 tammikuussa lopettavansa tuen Windows 10 Mobile -käyttöjärjestelmille vuoden 2019 joulukuussa [3].

Androidille kehitys tapahtuu käyttämällä Java- tai Kotlin-ohjelmointikieltä ja Android SDK:ta (Software Development Kit). Tällöin kehitysympäristönä toimii mikä tahansa laitteisto ja kehitys tehdään Android Studio -kehitystyökalulla. Kehitys iOS:lle tehdään Objective-C- tai Swift-ohjelmointikielellä ja iOS SDK:lla. Kehitysympäristöön iOS SDK:lla kuuluu Applen Xcode-kehitystyökalu, jonka käyttäminen edellyttää Applen Mac-laitetta, koska iOS SDK:ta ei ole mahdollista käyttää muilla järjestelmillä. Alustakohtaiset ohjelmointikielet, rajapinnat ja kehitysympäristöt ovat hyvin erilaisia, ja sen takia saman sovelluksen tekeminen kahdelle alustalle vaatiikin usein erilliset kehittäjät [17].

Erillään tehtävä natiivisovelluksen kehittäminen on kallista ja vaatii paljon resursseja. Erillään tehtävän kehityksen suurin ongelma on koodin uudelleen kirjoittaminen, koska toiselle alustalle on lähes mahdotonta käyttää suoraan koodia uudelleen ja samat toiminnot joudutaankin kirjoittamaan lähes kokonaan uudestaan. Uudelleen kirjoittaminen eri alustalle voi johtaa tilanteisiin, missä eri tiimit eivät enää ole samassa vaiheessa kehitysprosessia, ja toisen alustan sovellus on eri vaiheessa kehitystä. [17] [18]

Natiivisovelluksien kehittäminen tarjoaa myös huomattavia hyötyjä. Suurin hyöty natiivisovelluksissa on niiden tarjoama suorituskyky, varsinkin grafiikkaan liittyvissä toiminnallisuuksissa, kuten peleissä ja animaatioissa. Toinen merkittävä etu natiivisovelluksissa on niiden suora pääsy alustakohtaisiin rajapintoihin ja toimintoihin, kuten sijaintiin, ilmoituksiin ja laitteen kameraan. Ominaisuuksiin perustuvien etujen lisäksi natiivisovellukset on mahdollista pitää ajan tasalla kaikkien uusimpien toiminnallisuuksien ja käyttöliittymäkomponenttien osalta nopeasti.

2.2 Hybridimobiilisovellukset

Hybridimobiilisovellukset ovat sovelluksia, jotka toimivat usealla alustalla, yhdellä koodipohjalla. Hybridisovellus ei kuitenkaan ole yksiselitteinen käsitteenä, koska nimen määrittelyn jälkeen on kehitetty useita eri tapoja toteuttaa hybridisovelluksia. Tämän takia hybridisovelluksien nimeämiskäytännöt ovat sekavia ja eri toteutustavoille on yritetty esittää uusia nimeämiskäytäntöjä. [4] Yleisimmät toteutustavat hybridisovelluksien kehittämiseen on tuottaa sovellus joko web-teknologioita tai natiivikoodia generoivaa toteutusta käyttäen. Selkeyden vuoksi tässä työssä hybridisovelluksilla viitataan pelkästään web-teknologioita käyttäviin ratkaisuihin. Alustariippumattomiin natiivisovelluksiin puolestaan viitataan, kun sovelluksen kehittämiseen kirjoitettu koodi muutetaan natiivikoodiksi.

Hybridisovellukset käyttävät web-teknologioita, kuten HTML:ää (Hypertext Markup Language), CSS:ää (Cascading Style Sheets) ja JavaScriptiä, luodakseen sovelluksesta verkkosivun, joka näytetään WebView-komponentin avulla [5]. WebView-komponentti on sovelluksen sisäinen selain, jonka avulla verkkosivu ladataan niin, että se käyttäjälle näyttää sovellukselta. Näin toteutetut hybridisovellukset toimivat lähes millä laitealustalla tahansa, ja ne ovat usein hyvin kevyitä ja pystyvät rajoitetusti käyttämään alustakohtaisia natiivitoimintoja JavaScript -rajapinnan kautta [5]. Hybridisovelluksien etuja ovat yhden koodipohjan hyödyntäminen, nopea kehitys web-teknologioilla ja pienemmät kustannukset [5].

Malavoltan et al. tutkimuksen mukaan web-selaimen avulla toteutetut sovellukset eivät kuitenkaan ole saaneet positiivista palautetta käyttäjiltä suorituskyvyn ja käytettävyyden osalta. Sovelluksissa on myös arvioitu olevan huomattavasti enemmän ohjelmointivirheitä kuin natiivisovelluksissa. [16] Ahmadin et al. tutkimuksessa hybridisovelluksien suorituskykyä todetaan rajoittavan sovelluksen kehittämiseen käytetty web-teknologia ja

natiivitoimintojen rajallinen käytettävyys. Ulkoasua sovelluksissa rajoittavat sovelluskehyksien vaihtelevat käyttöliittymäkomponentit. [17]

Alustariippumattomat natiivisovellukset toteutetaan ilman WebView:tä. Sovelluksien koodi muutetaan natiiviksi, laitealustan mukaiseksi koodiksi. Muutos tehdään kääreen avulla tai käyttäen C++-kieltä, jota molemmat sekä iOS että Android tukevat. Tämän kaltaisia toteutuksia on useita ja niiden toimintaperiaatteista löytyy eroja, mutta lopputuloksena kaikista saadaan natiivisovelluksien kaltaisia alustariippumattomia sovelluksia. Alustariippumattomien sovelluksien suorituskyky vastaa lähes natiivisovelluksien suorituskykyä ja ulkoasukin on lähes identtinen. Vaikeasti erotettava ulkonäkö johtuu siitä, että useat sovelluskehykset hyödyntävät natiivikomponentteja. Tällöin käyttöliittymät muodostuvat samoista komponenteista ja niiden erottaminen toisistaan on vaikeaa.

3. VERTAILTAVAT SOVELLUSKEHYKSET

Työssä vertailtavat sovelluskehykset ovat React Native ja Flutter. Sovelluskehyksien valintaan on päädytty React Nativen suuren suosion ja Flutterin nopeasti kasvavan käyttäjämäärän vuoksi. Kirjoitushetkellä Github-sivulla React Nativella on 74 408 tähteä [6] ja Flutterilla 57 948 tähteä [7]. Sovelluskehyksien suosion ero Github-sivuilla on erittäin pieni, kun otetaan huomioon sovelluskehyksien julkaisupäivämäärät. React Native on julkaistu 2015 ja Flutter puolestaan 2017. Molemmat sovelluskehykset keskittyvät ratkaisemaan aikaisemmin mainittuja hybridisovelluksille tyypillisiä heikkouksia.

3.1 React Native

React Native on Facebookin kehittämä avoimen lähdekoodin sovelluskehys alustariippumattomien mobiilisovelluksien kehitykseen [8]. Se pohjautuu Facebookin kehittämään erittäin suosittuun React-kirjastoon. React Nativen toimintaperiaate onkin samanlainen kuin Reactin ja molempien ohjelmointikielenä toimii JavaScript [8]. Reactin lisäksi React Native hyödyntää paljon muitakin web-kehityksessä käytettyjä toiminnallisuuksia, kuten FlexBox-algoritmia ja CSS-tyylejä. Täten React Nativella kehittämiseen on helppo siirtyä web-kehityksestä, koska käytetyt teknologiat ovat ennestään tuttuja. React Nativen toiminnan perustana on sovelluksen sisäisen tilan päivittäminen, joka aiheuttaa ruudunpäivityksen. Ruudunpäivitys tapahtuu vain niille komponenteille, jotka liittyvät muuttuneeseen tilaan. Näin päivitykset ovat minimaalisia ja mahdollisimman vähän resursseja kuluttavia.

Minimaaliset päivitykset eivät kuitenkaan ole ainoa etu muihin web-pohjaisiin sovelluskehyksiin verrattuna. React Nativella tehty sovellus suoritetaan kohdelaitteessa JavaScriptCoren avulla. JavaScriptCore on Applen kehittämä avoimen lähdekoodin JavaScript-moottori, jonka avulla alustakohtainen natiivikoodi pystyy kommunikoimaan JavaScript-ohjelmien kanssa. JavaScriptCore mahdollistaa alustakohtaisien rajapintojen kutsumisen suoraan JavaScript-koodista, koska se kääntää JavaScriptin alustakohtaiseksi laitekoodiksi. [10] JavaScriptCoren tuoman edun lisäksi React Native muuntaa kehityksessä käytetyt komponentit natiivikomponenteiksi [9]. Näin sovelluksen komponentit toimivat ja käyttäytyvät samalla tavalla kuin natiivisovelluksien.

React Nativessa komponenttien tyylit määritellään JavaScriptinä. Jokainen komponentti hyväksyy style-objektin, joka sisältää camelCase-tyylillä kirjoitettuja avain—arvo -pareja.

Tyylien asettaminen on hyvin selkeää, koska objektien avaimet muistuttavat web-sovelluksissa käytettävien CSS-ominaisuuksien nimiä. Jos komponenteille ei anneta tyyliä ollenkaan, niin komponenttien ulkoasu on kohdealustan määrittelemän oletustyylin mukainen.

Komponentit asetellaan näytölle haluttuihin kohtiin käyttäen Flexbox-algoritmia, joka toimii tismalleen samalla tavalla kuin web-sovelluksissakin. Flexbox-algoritmin avulla komponentit asettuvat näytölle annettujen määrityksien mukaan. Algoritmi mahdollistaa komponenttien asettelun suhteessa toisiin komponentteihin, koska se huomioi muut komponentit näkymässä. Näin sovelluksen näkymät pysyvät samanlaisina myös eri kokoisilla ruuduilla, koska komponenttien sijainteja ei ole määritetty absoluuttisina arvoina koodiin. [22]

Sovelluksen suorittaminen laitteella on jaettu säikeisiin, jotta käyttöliittymän päivittäminen pysyy tasaisena [8]. Ensimmäinen säie on UI-säie, joka käynnistetään ensin ja se puolestaan käynnistää JavaScript-säikeen. UI-säie vastaa kaikesta ruudulle piirrettävästä toiminnasta sekä käyttäjän syötteiden seurannasta. Se myös vastaa Flexbox-algoritmilla aseteltujen käyttöliittymäkomponenttien absoluuttisen sijainnin laskemisesta laitteen ruudulla. JavaScript-säie vastaa kaikesta sovelluslogiikan suorittamisesta ja käyttäjän syötteisiin reagoimisesta. Näiden lisäksi käytössä on vielä kolmas säie, natiivi moduuli -säie, joka vastaa natiivirajapintojen kutsumisesta. Natiivi moduuli -säiettä käytetään esimerkiksi animaatioille, jotta voidaan keventää JavaScript-säikeen työmäärää. [11]

Suorittamisen jakaminen osiin säikeiden avulla mahdollistaa React Nativen tehokkuuden, mutta JavaScriptin käytössä on myös heikkoutensa. JavaScript-säie suorittaa tehtäviä kuten normaalit JavaScript-sovellukset, joten raskaat operaatiot voivat hidastaa muiden käskyjen suorittamista. Tämä puolestaan voi estää myös muiden säikeiden hyödyntämisen, koska ne odottavat käskyjä JavaScript-säikeeltä. Kyseisessä tilanteessa ruudunpäivityksissä voi ilmetä katkoksia, koska UI-säie odottaa käskyjä liian kauan. [11]

3.2 Flutter

Flutter on Googlen kehittämä avoimen lähdekoodin sovelluskehys alustariippumattomien mobiilisovelluksien kehitykseen. Sen tarkoitus on auttaa kehittäjiä rakentamaan korkealaatuisia ja suorituskykyisiä natiivisovelluksia eri alustoille ilman kompromisseja. Flutterin ohjelmointikielenä toimii Googlen kehittämä Dart. [12] Tämä tarkoittaa sitä, että

Flutterin käyttäminen edellyttää uuden ohjelmointikielen opettelua, koska Dart ei ole niin laajasti käytetty ohjelmointikieli. Dart muistuttaa kuitenkin hyvin paljon syntaksiltaan Java- ja JavaScript-ohjelmointikieliä, mikä helpottaa käytettävän ohjelmointikielen opettelua huomattavasti [12].

Flutterissa komponentteja kutsutaan widgeteiksi. Widgetit voivat olla joko tilallisia tai tilattomia, riippuen niiden käyttötarkoituksesta. Ruudunpäivitykset suoritetaan vain widgettien tilan muuttuessa ja sille osalle widgetin sisäistä tilaa, joka on muuttunut. [12] Toimintaperiaate on siis samankaltainen kuin React Nativella, mutta Flutterissa jokaiselle widgetille on kehitetty omat algoritmit ja datarakenteet [13]. Widgeteille optimoidut algoritmit ja datarakenteet mahdollistavat niiden tehokkaan prosessoinnin. Kehitystilassa Flutter hyödyntää JIT-kääntäjää (just in time -kääntäjä), jonka avulla muutokset nähdään reaaliajassa ja sovelluksen tila pysyy tallessa uudelleen lataamisesta huolimatta [15].

Toisin kuin React Native, Flutter ei käytä natiivikomponentteja, vaan Flutterissa käyttöliittymä on piirretty kankaalle hyödyntämällä Googlen kehittämää Skia-moottoria. Täten Flutterin widgetit ovat käytännössä vain kuvia näytöllä. Syy komponenttien piirtämiseen on paremman suorituskyvyn ja muokattavuuden tavoittelu. Flutterin on tarkoitus olla riippumaton natiivikomponenttien suorituskyvystä, ja sen takia niiden piirtäminen ruudulle mahdollistaa paremman optimoinnin. [15] Widgettien ulkoasuun on kuitenkin panostettu erittäin paljon, jotta ne näyttävät natiivikomponenteilta. Flutterin kehittäjätiimi ilmoittikin vuoden 2018 syyskuussa saaneensa iOS:ille tarkoitetut widgetit pikselin tarkkuudella vastaamaan Applen virallisia tyyliohjeita [14]. Androidille tarkoitetut widgetit ovat alusta alkaen olleet Material-tyyliohjeiden mukaisia.

Flutterissa tyylien asettaminen tehdään joko suoraan widgettikohtaisesti tai teemojen avulla. Ne widgetit, joille pystyy asettamaan tyylejä, sisältävät style-parametrin, johon voidaan sisällyttää widgettikohtaiset asetukset. Käytettäessä teemoja voidaan määrittää pääteema, jonka kaikki lapsi-widgetit perivät. Widgetit käyttävät pääteeman asetuksia, ellei lapsi-widgeteissä ole omaa teemaa, joka korvaa pääteeman.

Flutterissa widgetit asetellaan näytölle suoraan säiliöiden, rivien, sarakkeiden, pinon tai listanäkymän avulla. Asetteluun löytyy myös muita vaihtoehtoja, mutta mainitut asetteluwidgetit ovat yleisimmät. Asetteluwidgettien käyttäminen on suoraviivaista, sillä widgetit asettuvat ruudulle käytetyn asetteluwidgetin nimen mukaisesti. Rivi-widgetti asettaa widgetit vierekkäin, kolumni-widgetti puolestaan allekkain ja pino-widgetti asettelee widgetit päällekkäin.

Kun Flutter-sovellus rakennetaan julkaisuversioksi, siitä poistetaan kehitystyökalut. JIT-kääntäjän tilalle tulee AOT-kääntäjä (ahead of time -kääntäjä), joka mahdollistaa nopean sovelluksen käynnistyksen ja toiminnan. Sovelluksen rakentamisen yhteydessä koko sovellus käännetään alustakohtaiseksi laitekoodiksi. Näin sovelluksen suorittaminen ei vaadi mitään ylimääräisiä kääntäjiä tai tulkkeja ja sovellus on optimoitu kohdelaitteen alustalle. Kun sovellus on käännetty natiivikoodiksi, se pystyy kutsumaan alustakohtaisia rajapintoja suoraan, ilman mitään ylimääräisiä rajapintoja. Sovelluksen käynnistyksen yhteydessä ladataan natiivikoodiksi käännetty Flutter-kirjasto ja kirjoitettu Dart-koodi. Tämän jälkeen sovellus välittää kaikki käyttäjän syötteet Flutter-kirjastolle, joka reagoi syötteisiin. [15]

4. SOVELLUSKEHYKSIEN VERTAILU

Edellisessä luvussa esiteltyjen viitekehyksien vertailu suoritetaan pelkästään vertailusovelluksien avulla, koska molemmat sovelluskehykset tarjoavat lähes yhdenveroiset kehittäjätyökalut ja koodisyntaksin vertaaminen on hyvin subjektiivista. Tämän luvun alaluvuissa esitellään vertailun mittarit ja suoritetaan sovelluksen vertailu.

4.1 Vertailun mittarit

Sovelluksien vertailuun valitut mittarit ovat suorituskyvyn tarkastelu, sovelluksen käyttämä muisti ajon aikana sekä sovelluksen koko. Vertailtavat mittarit ovat tärkeitä ominaisuuksia mobiilisovellukselle, koska ne vaikuttavat suoraan siihen, minkälainen käyttäjäkokemus sovelluksesta jää käyttäjälle. Mittareita vertaillaan suoraan toisella sovelluskehyksellä tehdyn sovelluksen arvoihin. Valituille mittareille on alla esitetty tarkemmat kuvaukset niiden sisällöstä sekä perustelut miksi kyseistä mittaria tarkastellaan.

Sovelluksen suorituskyky on erittäin tärkeä mittari sovelluksen toiminnassa, ja sitä voidaan mitata monella eri tavalla. Tässä työssä suorituskyvyn mittaamiseksi tarkastellaan käynnistymisaikaa ja keskusyksikön käyttöprosenttia. Käynnistymisaikaa tarkastellaan kahdella mittarilla, kylmällä ja lämpimällä käynnistysajalla. Kylmä käynnistysaika tarkoittaa sovelluksen avaamiseen kuluvaa aikaa, kun sovellus on suljettu laitejärjestelmän kautta tai puhelin on käynnistetty uudelleen ennen sovelluksen avaamista [20]. Lämmin käynnistymisaika puolestaan tarkoittaa tilannetta, jossa sovellus on suljettu käyttäjän toimesta ja se käynnistetään uudelleen, ennen kuin laitejärjestelmä on lopettanut prosessia. Tällöin sovelluksen ei tarvitse luoda applikaatioprosessia käynnistymisen yhteydessä, vaan se löytyy laitteen muistista. [20]

Käynnistymisajan mittaamiseen käytetään Android Debug Bridge -komentorivityökalua (jatkossa adb), jonka avulla voidaan käynnistää sovellus ja selvittää sovelluksen käynnistymiseen kuluva aika. Keskusyksikön käyttöprosentti kertoo, kuinka paljon sovellus käyttää koko puhelimen keskusyksikön kapasiteetista. Mitä vähemmän sovellus tarvitsee laskentatehoa, sitä vähemmän sovellus kuluttaa akkua. Alhainen käyttöprosentti takaa myös sen, että sovellus ei häiritse muiden sovelluksien toimintaa käyttämällä liikaa resursseja. Kun sovellus vaatii liikaa resursseja, se ilmenee käyttäjälle hidastuneena toi-

mintana, koska laitteen resurssit eivät riitä sovelluksen ja taustatoimintojen pyörittämiseen. Keskusyksikön käyttöprosentti saadaan selvitettyä Android Studion Profiler-toiminnon avulla.

Suorituskyvyn lisäksi vertaillaan myös sovelluksen ajonaikaista muistinkulutusta. Mitä vähemmän sovellus käyttää ajonaikaista muistia, sen parempi. Tämä korostuu varsinkin laitteilla, joissa on vähän keskusmuistia. Tarkemmin ajonaikaisella muistilla tarkoitetaan sovelluksen käyttämää likaista yksityistä muistia. Likainen yksityinen muisti tarkoittaa muistia, joka on vain tietyn sovelluksen käytettävissä. Näin varattu muisti palautuu muiden sovelluksien käytettäväksi vasta kun sovellus suljetaan. [21] Myös sovelluksen varaama yksityinen ajonaikainen muisti saadaan selvitettyä adb:n avulla dumpsys -komennolla.

Ajonaikaisen muistin lisäksi sovelluksien käyttämää muistia verrataan myös tutkimalla sen kokoa laitteessa, kun sovellus on rakennettu julkaisuversioon. Julkaisuversiossa sovelluksista on jätetty pois kaikki kehittämistyökalut ja niiden koodi on käännetty alustakohtaiseksi. Tällöin sovelluksessa ei ole mitään ylimääräistä ja sen koon tulisi olla mahdollisimman pieni. Sovelluksista vertaillaan myös asennustiedostojen eli apk-pakettien kokoeroa. Asennustiedostojen koko antaa suuntaa sovelluksen koosta, jos se ladattaisiin Google Play -kaupasta tai App Storesta. Sen takia myös asennustiedoston koko on tärkeä ominaisuus ja senkin tulisi olla mahdollisimman pieni.

4.2 Vertailun toteuttaminen

Vertailua varten toteutetaan kaksi eri sovellusta, joista toiseen ei kuulu mitään toiminnallisuutta ja toisessa tehdään ruudunpäivityksiä tiheään tahtiin**.** Näin voidaan vertailla sovelluskehyksiä kattavammin. Ensimmäisessä sovelluksessa on vain keskitetty teksti ruudulla. Tämä auttaa selvittämään eri sovelluskehyksillä tehtyjen sovelluksien absoluuttisen minimikoon ja käynnistymisajan. Toinen sovellus on sekuntikello, jossa on mukana joitain käyttöliittymäkomponentteja ja tasaisesti päivittyvä tekstikomponentti. Sekuntikellon avulla voidaan vertailla, kuinka paljon sovellus kuluttaa laitteen resursseja ajon aikana. Sovelluskehyksistä käytetään kirjoitushetkellä uusimpia versioita. React Nativen versio on 0.59.1 ja Flutterin versio on 1.2. Sovelluksien vertailut suoritetaan samalla puhelimella koko ajan, jotta vertailu on tasavertaista. Käytettävä puhelin on Sony Xperia Z5 Compact, jonka Android käyttöjärjestelmän versio on 7.1.1.

4.2.1 Suorituskyky

Sovelluksien kylmä ja lämmin käynnistymisaika mitataan käynnistämällä laite uudelleen ennen mittauksen aloittamista. Käynnistyksen jälkeen ajetaan adb:n avulla komento

```
adb shell am start -S -W -R 10 <package-name>/.MainActivity
-c android.intent.category.LAUNCHER
-a android.intent.action.MAIN,
```

jossa S-lippu sulkee sovelluksen väkisin ennen avaamista, W-lippu antaa sovelluksen käynnistyä loppuun asti ja R-parametri kertoo kuinka monta kertaa komento suoritetaan [20]. Komento ajetaan ensimmäisellä kerralla antamalla R:n arvoksi 1. Tällöin saadaan selville sovelluksen kylmä käynnistysaika, koska laite on käynnistetty uudelleen ennen komennon ajamista ja sovellusta ei ole avattu käynnistyksen jälkeen. Seuraavaksi komento ajetaan antamalla R:n arvoksi 10. Näin sovellus avataan ja suljetaan kymmenen kertaa peräkkäin. Saadut arvot ovat sovelluksen lämpimiä käynnistysaikoja ja niistä lasketaan keskiarvo. Käynnistymisajan mittaamiset suoritetaan sovelluksien julkaisuversioille.

Keskusyksikön käyttöprosentti puolestaan saadaan selville Android Studion Profiler-toiminnon avulla seuraamalla sovellusta sen ollessa käynnissä. Profiler laskee sovellukselle keskiarvon keskusyksikön käyttöprosentille mittauksen ajalta. Jokaista sovellusta seurataan Profilerilla 30 sekuntia. React Nativella tehdyt sovellukset ajettiin dev-asetus pois päältä, jolloin sovelluksesta on poistettu käytöstä kehittäjätyökalut. Flutterilla tehdyt sovellukset ajettiin profile-tilassa. Tällöin Flutterilla tehdyt sovellukset vaihtavat JIT-kääntäjän AOT-kääntäjään, ja suorituskyky on lähempänä julkaisuversion suorituskykyä. Mittauksien lukuarvot on kirjattu taulukkoon 1.

Taulukko 1 *Sovelluksien käynnistymisajat ja keskusyksikön käyttöprosentit*

Sovelluskehys	Tarkasteltava sovellus	Kylmä käynnistysaika (ms)	Lämmin käynnistysaika (ms)	Keskusyksikön käyttöprosentti
React Native	Minimaalinen	286	190	0
	Sekuntikello	396	218	16
Flutter	Minimaalinen	552	338	0
	Sekuntikello	1307	471	12

Taulukkoon 1 kirjattujen tuloksien perusteella React Nativella tehdyt sovellukset käynnistyivät huomattavasti nopeammin kuin Flutterilla tehdyt. Minimaalisen sovelluksen kylmä käynnistysaika oli lähes kaksi kertaa nopeampi ja lämmin käynnistys aika noin 1,7 kertaa nopeampi React Nativella. Sekuntikellosovelluksella käynnistysaikojen erot ovat vielä suuremmat. Kylmä käynnistysaika on yli kolme kertaa nopeampi ja lämmin käynnistysaika yli kaksi kertaa nopeampi React Nativella. Taulukosta nähdään myös, että minimaalisten sovelluksien keskusyksikön käyttöprosentit olivat molemmilla sovelluskehyksillä 0 %. Tulos oli odotettavissa, koska minimaalisen sovelluksen ei ole tarkoitus suorittaa ruudunpäivityksiä ollenkaan. React Nativella tehty sekuntikello sovellus vaati kuitenkin 4 % enemmän resursseja keskusyksiköltä kuin Flutterin versio sovelluksesta.

4.2.2 Sovelluksen muistinkulutus

Sovelluksien koot mitattiin julkaisuversioille sovelluksista, koska kehitysversioiden vertaaminen ei anna todellista kuvaa sovelluksen koosta. Minimaalisissa sovelluksissa on mukana vain sovelluskehyksillä uuden projektin luonnin yhteydessä ladatut paketit. Minimaalisista sovelluksista on siivottu pois ylimääräiset komponentit, jotka molemmat sovelluskehykset tuottavat projektin luonnin yhteydessä. Sovelluksiin jää jäljelle vain keskitetty teksti ja saadaan absoluuttisen minimaalinen sovellus molemmilla sovelluskehyksillä. Sovelluksien julkaisuversioiden rakentamisen yhteydessä on otettu huomioon molempien sovelluskehyksien dokumentoinnista löytyvät ohjeet sovelluksien koon minimoimiseksi. Näin sovellukset ovat vertailukelpoisia, eikä niille ole tehty dokumentaation ulkopuolisia optimointeja.

Sovelluksien koko laitteessa on katsottu puhelimen laitehallinnan kautta. Sovelluksien asennustiedostojen koko nähdään tietokoneelta julkaisuversion rakentamisen jälkeen. Flutter luo vain yhden asennustiedoston, mutta React Native luo niitä neljä. Tämä johtuu siitä, että React Nativella tehtyihin projekteihin on asetettu päälle asetus, joka rakentaa erilliset paketit eri keskusyksikköarkkitehtuureille. Näin React Nativella saadaan asennustiedosto mahdollisimman pieneksi. React Nativen osalta asennustiedoston vertailuun on otettu mukaan kohdelaitteen arkkitehtuuria vastaava asennustiedosto. Sovelluksien koot laitteessa ja asennustiedostojen koot on kirjattu taulukkoon 2.

Taulukko 2 *Sovelluksien asennustiedostojen koko ja koko laitteessa*

Sovelluskehys	Tarkasteltava sovellus	Asennustiedoston koko (Mt)	Koko laitteessa (Mt)
React Native	Minimaalinen	8,21	32,62
	Sekuntikello	8,22	32,63
Flutter	Minimaalinen	4,85	14,94
	Sekuntikello	4,91	15,71

Taulukon 2 tuloksista nähdään, että kummatkin Flutterilla tehdyt sovellukset ovat yli kaksi kertaa pienempiä kuin React Nativella tehdyt. Myös Flutter sovelluksien asennustiedostojen koko on lähes kaksi kertaa pienempi kuin React Native sovelluksien.

Sovelluksien varaamaa keskusmuistia verrattiin pelkästään sekuntikellosovellukselle, koska siinä suoritetaan ruudunpäivityksiä. Varatun keskusmuistin määrä mitattiin adb:n avulla ajamalla komento

```
adb shell dumpsys meminfo<package-name> -d,
```

jossa d-lipun avulla saadaan enemmän tietoa sovelluksen muistin kulutuksesta [21]. Komennolla saadaan selville jaoteltu tulos siitä, kuinka sovelluksen käyttämä keskusmuisti jakaantuu eri osa-alueisiin. Tuloksen luvuista keskitytään kuitenkin vain likaisen yksityisen muistin määrään, koska sillä on suurin vaikutus laitteen muihin toimintoihin. Dumpsys komento suoritettiin heti sovelluksen avaamisen jälkeen, jolloin sekuntikelloa ei ole vielä käynnistetty. Näin saatiin selville sovelluksen käynnistyshetkellä varaama yksityinen likainen muisti. Mittauksen jälkeen sekuntikello käynnistettiin ja sen annettiin olla päällä 30 sekuntia, jonka jälkeen suoritettiin seuraava mittaus. Mittauksista saadut tulokset on kirjattu taulukkoon 3.

Taulukko 3 *Sekuntikellosovelluksien käyttämä likainen yksityinen muisti*

Sovelluskehys	Muisti hetkellä 0s (Mt)	Muisti hetkellä 30s (Mt)
React Native	43,9	64,3
Flutter	21,2	35,6

Taulukon 3 tuloksien perusteella nähdään, että Flutterilla toteutettu sekuntikello käytti huomattavasti vähemmän laitteen muistia. Flutterilla tehty versio sovelluksesta varasi

alle puolet React Nativen version varaamasta keskusmuistista sovelluksen käynnistyshetkellä. Kun sovellus oli ollut päällä 30 sekuntia Flutterin muistin tarve oli kasvanut 14,4 Mt ja React Nativen 20,4 Mt, jonka jälkeen Flutter sovelluksen varaama muisti oli edelleen lähes puolet React Nativen versiosta.

4.3 Vertailun tulokset

Saatujen tuloksien perusteella Flutter suoriutui paremmin kaikilla muilla mittareilla, paitsi käynnistysajalla. Taulukosta 1 nähtiin, että React Nativella tehdyt sovellukset käynnistyivät huomattavasti nopeammin. Tulos on yllättävä, kun otetaan huomioon taulukkoon 2 kirjatut sovelluksien koot. Voisi olettaa, että pienemmät sovellukset käynnistyisivät nopeammin. Tämän lisäksi Flutter-sovelluksien julkaisuversiossa käytettävän AOT-kääntäjän pitäisi tehdä sovelluksen käynnistämisestä nopeampaa kuin React Native -sovelluksien JIT-kääntäjän.

Taulukosta 1 nähdään keskusyksikön käyttöprosentin olevan lähes yhtä hyvä molemmilla sovelluskehyksillä. Tuloksen perusteella on vaikea erotella niistä tehokkaampaa, koska mittaus aika oli vain 30 sekuntia ja sovellus oli typistetty. Sovelluksien varaamassa yksityisessä muistissa puolestaan havaittiin suuret erot taulukossa 3. Flutterilla tehty sovellus tarvitsi huomattavasti vähemmän keskusmuistia, ja kyseisen sovelluksen muistin tarve kasvoi vähemmän sovelluksen ajon aikana.

Työn tulokset saattaisivat olla erilaiset, jos käytettävissä olisi ollut uudempi Android-puhelin ja iPhone. Varsinkin sovelluksien koko laitteessa ja käynnistysaika olisivat voineet olla hyvinkin erilaiset iPhonella, koska iOS-järjestelmissä on valmiiksi JavaScriptCore. Tällöin sovellukseen ei tarvitse sisällyttää JavaScriptCorea, kuten Android-alustalle tehdään React Nativella. Vaikutus todennäköisesti näkyisi tällöin myös asennustiedoston koossa. Käynnistysaikoihin iPhonella vaikuttaisi se, että iOS-järjestelmä ei tue JIT-kääntäjän käyttämistä. Näiden lisäksi Flutter on Googlen kehittämä, joten se saattaa vaikuttaa työssä saatuihin tuloksiin Android-alustalla.

5. YHTEENVETO

Työn tavoitteena oli vertailla alustariippumattomien mobiilisovelluksien kehittämiseen tarkoitettuja Flutter- ja React Native -sovelluskehyksiä. Vertailu suoritettiin tarkastelemalla sovelluksien käynnistysaikaa, keskusyksikön käyttöprosenttia, ajon aikana varattua keskusmuistia ja asennustiedostojen kokoa. Flutter osoittautui tehokkaammaksi sovelluskehykseksi kaikilla muilla mittareilla, paitsi käynnistysajalla. Tulos on hyvin lupaava Flutterin kannalta, koska Flutter on vielä alkutaipaleella ja React Native on laajasti käytössä mobiilisovelluksien sovelluskehityksessä. Vertailu toteutettiin kuitenkin vain Android-alustalle, joten tulokset saattaisivat muuttua, jos vertailu suoritettaisiin myös iOS-alustalle.

Vertailun perusteella nähtiin sovelluskehyksien erot suorituskyvyssä verrattuna toisiinsa, mutta vertailun sovellukset ovat hyvin typistettyjä. Täten olisikin mielekästä jatkaa työn vertailua laajentamalla vertailtavia sovelluksia, käytettyjä laitteita ja alustoja. Laajemmalla sovelluksella pystyttäisiin vertailemaan käyttöliittymäkomponenttien ulkoasua ja toiminnallisuutta sekä arvioimaan sovelluskehyksien käytettävyyttä. Sovelluskehyksiä olisi myös kiinnostavaa vertailla natiivisovelluksiin, jotta nähtäisiin kuinka lähellä natiivisuorituskykyä ne ovat. Tämä vaatisi kuitenkin enemmän aikaa ja osaamista molemmista Android- sekä iOS-alustoista.

LÄHTEET

[1] GlobalStats statcounter, http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet, viitattu 2.02.2019

[2] GlobalStats statcounter, http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#monthly-201301-201902, viitattu 2.02.2019

[3] Microsoft Windows support, https://support.microsoft.com/en-us/help/4485197/windows-10-mobile-end-of-support-faq, viitattu 6.2.2019

[4] Nunkesser, R. 2018, Beyond web/native/hybrid: A new taxonomy for mobile app development, 2018 ACM/IEEE 5th International Conference on Mobile Software Engineering and Systems, pp. 214–218. https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8543456

[5] Panhale, M. 2015, Beginning Hybrid Mobile Application Development, Chapter 3 - Building Blocks of HMAD, Apress

[6] React Native GitHub, https://github.com/facebook/react-native, viitattu 26.3.2019

[7] Flutter Github, https://github.com/flutter/flutter, viitattu 26.3.2019

[8] Paul, A. & Nalwaya, A. 2015, React Native for iOS Development, 1st edn, Chapter 2 - The Simplest Program: Hello World with React Native, Apress, Berkeley, CA.

[9] React Native Learn the basics, https://facebook.github.io/react-native/docs/tutorial, viitattu 26.3.2019

[10] Apple developer documentation, JavaScriptCore, https://developer.apple.com/documentation/javascriptcore, viitattu 26.3.2019

[11] React Native performance, https://facebook.github.io/react-native/docs/performance, viitattu 4.3.2019

[12] Flutter Technical overview, https://flutter.dev/docs/resources/technical-overview, viitattu 26.3.2019

[13] Flutter Inside Flutter, https://flutter.dev/docs/resources/inside-flutter, viitattu 26.3.2019

[14] Google developer blog, https://developers.googleblog.com/2018/09/flutter-release-preview-2-pixel-perfect.html, viitattu 27.3.2019

[15] Flutter FAQ, https://flutter.dev/docs/resources/faq, viitattu 26.3.2019

[16] I. Malavolta, S. Ruberto, T. Soru & V. Terragni 2015, End Users' Perception of Hybrid Mobile Apps in the Google Play Store, 2015 IEEE International Conference on Mobile Services, pp. 25–32. DOI: https://doi.org/10.1109/MobServ.2015.14

[17] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif & S. Ge. An Empirical Study of Investigating Mobile Applications Development Challenges, IEEE Access, vol. 6, pp. 17711–17728. DOI: https://doi.org/10.1109/ACCESS.2018.2818724

[18] M. E. Joorabchi, A. Mesbah & P. Kruchten 2013, Real Challenges in Mobile App Development, 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 15–24. DOI: https://doi.org/10.1109/ESEM.2013.9

[19] M. Willocx, J. Vossaert & V. Naessens 2016, Comparing Performance Parameters of Mobile App Development Strategies, 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 38–47. https://ieeexplore-ieee-org.libproxy.tuni.fi/document/7832966

[20] Android developers, App startup time, https://developer.android.com/topic/performance/vitals/launch-time, viitattu 28.3.2019

[21] Android developers, dumpsys, https://developer.android.com/studio/command-line/dumpsys#meminfo, viitattu 28.3.2019

[22] React Native Layout with Flexbox, https://facebook.github.io/react-native/docs/flexbox, viitattu 26.4.2018