Pekka Hyytinen

Alustariippumaton mobiilisovelluskehitys

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pekka Hyytinen

Yhteystiedot: pekka.e.hyytinen@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Alustariippumaton mobiilisovelluskehitys

Title in English: Cross-platform mobile application development

Työ: Kandidaatintutkielma

Sivumäärä: 24+0

Tiivistelmä: Tässä tutkielmassa tutkitaan alustariippumatonta mobiilisovelluskehitystä. Tavoitteena on selvittää millaisia erilaisia alustariippumattomia kehitysympäristöjä ja arkkitehtuureja on olemassa. Lisäksi päämääränä on selvittää, kuinka erilaiset kehitysympäristöt ja arkkitehtuurit vertautuvat sekä toisiinsa että perinteiseen natiiviin Android/iOS kehitykseen.

Avainsanat: Alustariippumaton, natiivi, Android, iOS

Abstract: This bachelor thesis focuses on researching cross-platform mobile application development. The goal is to examine what different kinds of cross-platform frameworks and architectures exist and how they compare to each other as well as to traditional native Android/iOS development.

Keywords: Cross-platform native, Android, iOS

Kuviot

Sisältö

1 Johdanto

Nykypäivän mobiilisovelluskehityksessä on tärkeää pystyä tuottamaan nopeasti, luotettavasti ja kustannustehokkaasti sovelluksia mahdollisimman monelle laitteelle (Pinto ja Coutinho 2018). Alustariippumattomat kehitysympäristöt ovat osin syrjäyttäneet perinteisen kehityksen, jossa mobiilisovelluksen ohjelmakoodi joudutaan kirjoittamaan erikseen eri mobiilikäyttöjärjestelmille (Android/iOS), pystymättä jakamaan koodia eri alustojen kesken. Jaettuun koodipohjaan perustuvat alustariippumattomat kehitysympäristöt säästävät potentiaalisesti aikaa ja kustannuksia sekä kehitysvaiheessa että ylläpidossa (Raj ja Tolety 2012). Samaa koodipohjaa voidaan käyttää eri käyttöjärjestelmille tuotetuissa sovelluksissa. Työkalujen valinnalla on kuitenkin merkitystä. Tässä tutkielmassa pyritään ottamaan selvää eri alustariippumattomien kehitysympäristöjen/arkkitehtuurien eroista sekä hyödyistä ja haitoista verrattuna sekä toisiinsa että perinteiseen natiiviin kehitykseen.

Luvussa 2 esitellään teoreettinen tausta tarkemmin natiivin ja alustariippumattoman ohjelmistokehityksen osalta. Kerrotaan erilaisista arkkitehtuurisista ratkaisuista, teknisestä toteutuksesta sekä näiden eroista. Luku 3 tarkastelee nykypäivän suosituimpia alustariippumattomia kehitysympäristöjä, näiden teknisiä toteutuksia sekä erityispiirteitä. Luvuissa 2 ja 3 kerrotaan lisäksi eri lähestymistapojen ja kehitysympäristöjen adoptoinnin potentiaalisista hyödyistä ja haitoista. Lopuksi luvussa 4 kerrotaan yhteenveto havaituista tuloksista.

2 Mobiilisovelluskehitys

Tämän luvun tarkoituksena on luoda teoriapohja tutkielman keskeisimmille mobiilisovelluskehitykseen liittyville käsitteille. Luvussa esitellään erilaisia tapoja kehittää nykypäivän mobiilisovelluksia ja kerrotaan näiden tapojen hyödyistä, haitoista sekä eroista esimerkiksi arkkitehtuurin osalta.

2.1 Natiivi kehitys

Natiivimobiilisovellukset ovat sovelluksia, jotka kehitetään erikseen jokaiselle eri mobiilikäyttöjärjestelmälle. Nykyään tämä tarkoittaa käytännössä joko Android- tai iOS-käyttöjärjestelmää niiden hallitessa markkinoita (Raj ja Tolety 2012). Natiivisovelluksilla on täysi ja suora pääsy käyttöjärjestelmän toimintoihin ja niillä saavutetaan yleensä paras suorituskyky sekä käyttökokemus (Ebone, Tan ja Jia 2018). Kehitysympäristöinä toimivat pääasiassa alustojen viralliset ohjelmistokehityspaketit (engl. software development kit), ohjelmointikielet sekä koodieditorit. Androidilla ohjelmointikielinä käytetään Javaa ja/tai Kotlinia ja editorina Android Studiota. iOSilla ohjelmointikielinä toimivat Swift/Objective-C ja editorina Xcode. Tässä tutkielmassa natiivikoodilla tarkoitetaan edellä mainittuja ohjelmointikieliä, eikä esimerkiksi vielä alemman tason C/C++ kieliä tai konekieltä. Valmis sovellus paketoidaan natiiviksi sovelluspaketiksi (tiedostomuotona Androidilla .apk/.aab, iOSilla .ipa), jonka jälkeen se voidaan siirtää sovelluskauppoihin käyttäjien ladattavaksi.

Sovelluksen kehittämisestä natiivisti eri alustoille seuraa yleensä korkeammat kustannukset sekä haasteellisempi kehitystyö, koska kehittäjien täytyy osata käyttää eri ohjelmointikieliä sekä työkaluja. Kustannuksia taas korvaa natiivisovellusten usein tuottama parempi suorituskyky sekä vähäisempi resurssien, kuten keskusmuistin, suorittimen ja akun käyttö. (Pinto ja Coutinho 2018)

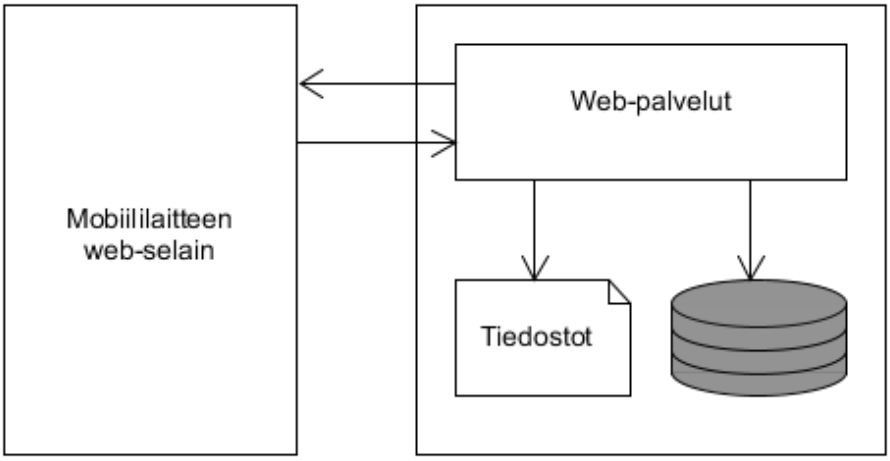
2.2 Alustariippumattoman kehitys

Alustariippumattomat mobiilisovellukset perustuvat jaettuun koodipohjaan. Sovelluksen ohjelmakoodi tarvitsee lähtökohtaisesti kirjoittaa ainoastaan kerran, minkä jälkeen sen voi kohdistaa useille mobiilikäyttöjärjestelmille. Suurimpana etuna on täten ajankäytön ja kustannuksien säästöt natiiviin kehitykseen verrattuna (Ebone, Tan ja Jia 2018). Yleisiksi haitoiksi taas voidaan lukea natiivia heikompi suorituskyky, sekä rajoittuminen kehitysympäristön tarjoamiin rajapintoihin laitteiden natiiviominaisuuksien käytössä (Ahmad ym. 2018).

Perinteisesti on puhuttu kolmesta eri tavasta toteuttaa mobiilisovelluksia, nämä ovat natiivi, web ja hybridi (Nunkesser 2018). Näistä web ja hybridi kuuluvat varsinaisesti alustariippumattomiin tapoihin tuottaa sovelluksia. Varsinkin hybridikategorian alle niputetaan usein hyvin erilaisia lähestymistapoja ja arkkitehtuurillisia ratkaisuja. Alustariippumattomien sovelluksien osalta ei vaikutakaan olevan olemassa mitään vakiintunutta termistöä eri lähestymistapojen luokitteluun, vaan eri lähteet esittävät usein toisistaan eroavia alakategorioita. Nunkesser (2018) esimerkiksi luokittelee yleisesti mobiilisovelluskehityksen lähestymistavat seuraavasti: Endeemiset sovellukset (endemic apps), web-sovellukset (web apps), hybridit web-sovellukset (hybrid web apps) hybridit sillatut sovellukset (hybrid bridged apps), järjestelmäkieliset sovellukset (system language apps) sekä vieraskieliset sovellukset (foreign language apps). Tässä luokittelussa endeemiset sovellukset vastaavat perinteistä natiivin kehityksen määritelmää ja web, hybridi- sekä vieraskieliset sovellukset alustariippumattomia lähestymistapoja. Nunkesserin (2018) mukaan mobiilikehityksen luokittelua on syytä päivittää vastaamaan nykyajan vaatimuksia, nimenomaan alustariippumattomien kehitysympäristöjen yleistymisen myötä. Raj ja Tolety (2012) luokittelevat varsinaiset alustariippumattomat arkkitehtuurityypit/lähestymistavat neljään kategoriaan jotka ovat web, hybridi (hybrid), tulkattu (interpreted) sekä ristiinkäännetty (cross-compiled). Tämä jaottelu esiintyi usein lähdekirjallisuudessa. ja tämän johdosta tässä tutkielmassa päädyttiin käyttämään samaa luokittelua. Tunnustetaan kuitenkin, että nämä määritelmät eivät kykene täydellisesti kuvaamaan kaikkien kehitysympäristöjen arkkitehtuuria. Kategoriat esitellään tarkemmin seuraavissa alaluvuissa.

2.2.1 Web

Web-sovelluksissa sovelluksen käyttöliittymä näytetään käyttöjärjestelmän web-selaimen kautta. Sovelluskauppojen kautta ladattavaa sovelluspakettia ei luoda, vaan sovelluksen ohjelmakoodi elää web-palvelimella ja toimii perinteisen web-sivun tavoin. Responsiivisten tekniikoiden avulla sovellus voidaan saada näkymään työpöytälaitteiden lisäksi optimaalisesti myös mobiililaitteilla.



Kuvio 1. Web-lähestymistapa (mukaillen Raj ja Tolety 2012)

Web-pohjaiset sovellukset ovat perinteisesti vaatineet toimiakseen internet-yhteyden, mutta uuden sukupolven progressiiviset web-sovellukset (engl. progressive web app) tuovat web-sovellukset yhä lähemmäksi natiivia. Ne kykenevät toimimaan myös offline tilassa sekä mm. lähettämään push-ilmoituksia ja tallentamaan sovellusikonin kotinäytölle jolloin ne voidaan käynnistää kuten tavanomaiset mobiilisovellukset (Malavolta ym. 2017). Progressiiviset web-sovellukset perustuvat palvelunvälittäjän (engl. service worker) käyttöön. Palvelunvälittäjä on omassa säikeessään toimiva ohjelmoitava tapahtumapohjainen välikäsi (käytännössä JavaScript -tiedosto), jonka avulla kehittäjät voivat määritellä kuinka sivun verkkopyynnöt käsitellään (Google 2019). Näin voidaan esimerkiksi mahdollistaa työskentely offline-tilassa sieppaamalla verkkopyyntö ja lähettämällä haluttu vastaus verkon sijaan välimuistista.

Web-sovellusten hyötynä voidaan pitää, että ne eivät vaadi asennusta laitteelle. Sovelluslogiikka elää palvelimella, joten käyttäjien ei tarvitse huolehtia mahdollisista päivityksistä vaan ne ovat välittömästi saatavilla (Raj ja Tolety 2012). Toisin kuin ladattavien sovellusten tapauksessa, web-sovellusten ei siis tarvitse käydä läpi sovelluskauppojen mahdollisesti pitkiä hyväksymisprosesseja. Haittoina taas voidaan pitää riippuvuutta verkkoyhteydestä, sekä tästä johtuvaa mahdollista hitautta (Raj ja Tolety 2012). Tosin kuten aiemmin on mainittu, progressiiviset web-sovellukset kykenevät näyttämään sovelluksesta riippuen ainakin jonkinlaista toiminnallisuutta myös offline-tilassa.

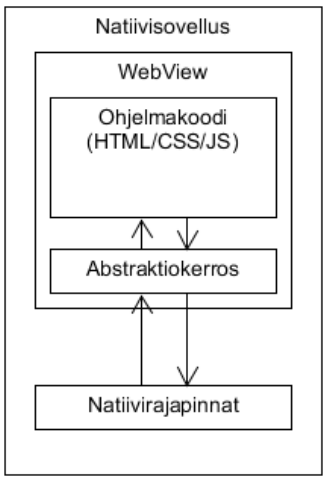
Rajn ja Toletyn (2012) mukaan web-sovellukset sopivat parhaiten sovelluksiin, joissa sovelluksen logiikka on suurimmaksi osaksi palvelimella ja sovelluksen tehtävänä on pääasiassa esittää tietoa. Se ei taas sovellu kovin hyvin lähestymistavaksi sovelluksiin, jotka tuottavat ja/tai prosessoivat paljon tietoa laitteella tai käyttävät hyväkseen laitteen sensoreita. Hyviä käyttökohteita voisivat olla siis esimerkiksi verkkokaupat, foorumit ja uutissivustot.

2.2.2 Hybridi

Hybridisovelluksilla tarkoitetaan yleensä sovelluksia jotka ohjelmoidaan web-teknologioilla (JavaScipt/HTML/CSS) ja paketoidaan natiivisovelluksiksi (Que, Guo ja Zhu 2016). Hybridisovelluskehyksissä sovelluksen käyttöliittymä luodaan laitteen WebView-näkymään koko näytössä. WebView (Androidilla WebView, iOSilla WKWebView tai UIWebView) on käyttöjärjestelmän kevyt web-selainkomponentti, joka kykenee näyttämään verkkosivuja sovelluksen sisällä. Sen lisäksi, että WebView on vastuussa sovelluksen näkymän renderöinnistä, toimii se samalla myös abstraktiokerroksena jonka läpi natiivirajapintoja voidaan kutsua (Biørn-Hansen ja Ghinea 2018). Hybridisovellukset eivät käytä muita natiiveja käyttöliittymäkomponentteja, vaan sovellus on käytännössä kuin web-sovellus, mutta paketoituna natiiviin muotoon. Paketoituna sovellus voidaan siirtää sovelluskauppoihin, josta käyttäjät voivat ladata sen kuten normaalin natiivin sovelluksen. Käyttöjärjestelmän toimintoihin kuten kameraan ja paikannukseen päästään käsiksi kehyksen tarjoamilla liitännäisillä, jotka tarjoavat yhteisen rajapinnan eri käyttöjärjestelmien toimintoihin (Pinto ja Coutinho 2018). Suurin osa hybridikehitysympäristöistä pohjautuu Apache Cordovaan. Cordova on avoimen lähdekoodin kirjasto, joka tarjoaa helpon rajapinnan laitteen natiiviominaisuuksien käyttöön sekä

sovelluksen paketointiin. (Biørn-Hansen ja Ghinea 2018).

Hybridisovelluksien haittoina voidaan yleisesti pitää natiivisovelluksiin verrattuna huonompaa suorituskykyä sekä natiivin ulkoasun ja käyttökokemuksen puutetta (Raj ja Tolety 2012). Kaikkia laitteen ominaisuuksia ei myöskään mahdollisesti voida hyödyntää, koska kehityksessä joudutaan usein tukeutumaan kehitysympäristön tarjoamiin liitännäisiin (Macquarrie 2018). Usein liitännäisiä/mukautettuja ominaisuuksia natiivi- ja hybridikehitysympäristön välille on mahdollista tuottaa itse. Toisaalta Macquairrien (2018) mukaan jos niitä joudutaan toteuttamaan paljon, voi kehitysaika muodostua jopa pidemmäksi kuin mitä se olisi, jos sovellus olisi toteutettu natiivisti. Lisäksi tuen saaminen uusille Androidin/iOSin ominaisuuksille hybridikehitysympäristössä kestää kauemmin verrattuna natiiveihin ympäristöihin, joissa ne ovat ensimmäisenä saatavilla. (Sharma 2019).

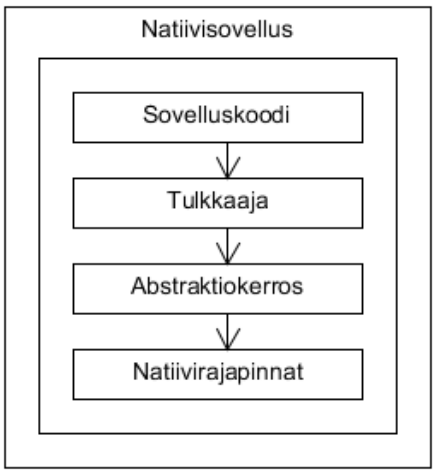


Kuvio 2. Hybridilähestymistapa

Hybridisovellus on hyvä ratkaisu, jos sovellus täytyy saada nopeasti markkinoille ja se ei vaadi kovin monimutkaisia ja/tai kustomoituja ominaisuuksia (Macquarrie 2018). Se voi olla myös hyvä ratkaisu sovelluksiin joissa laitteen sensoreita tarvitaan datan tuottamiseen, mutta dataa on tarkoitus käsitellä palvelimella eikä laitteella (Raj ja Tolety 2012).

2.2.3 Tulkattu

Tulkatussa (engl. interpreted) lähestymistavassa ohjelmakoodi kirjoitetaan käyttäen useimmiten jotain yleisesti käytettyä ohjelmointikieltä, jota tulkataan ajon aikana. Kirjoitetusta käyttöliittymäkoodista generoidaan natiiveja käyttöliittymäkomponentteja eri alustoille, ja laiteominaisuuksiin päästään käsiksi ylimääräisen ohjelmointirajapinnan kautta (Latif ym. 2016). Tulkatun sovelluksen ero käännettyyn sovellukseen verrattuna on se, että ohjelmakoodia tulkataan sovelluksen ajon aikana, kun käännetyssä sovelluksessa koodi analysoidaan ja käännetään kokonaisuudessaan ennen ajoa.



Kuvio 3. Tulkattu lähestymistapa (mukaillen Raj ja Tolety 2012)

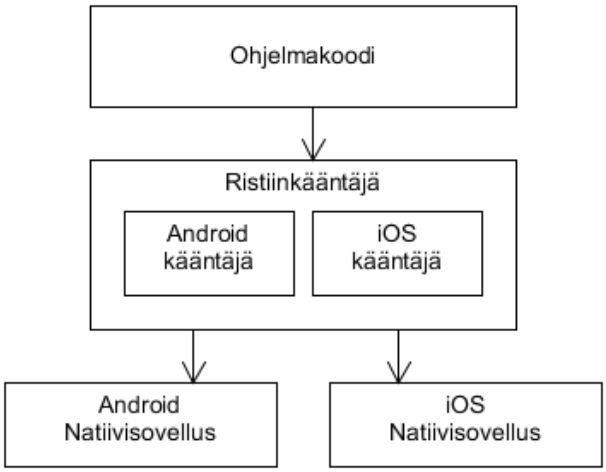
Sovelluksen lähdekoodi, joka tulkattujen kehitysympäristöjen tapauksessa on yleensä kirjoitettu JavaScriptillä, siirretään laitteelle ja tulkkaus tapahtuu käyttäen jotakin JavaScript-moottoria (engl. JavaScript engine). Javascript-moottoria kutsutaan tässä tapauksessa tulkiksi ja sen toteutus riippuu käytetystä alustasta. Esimerkiksi iOSilla käytetään pääasiassa JavascriptCorea. (Shah, Sinha ja Mishra 2019)

Hybridikehitysympäristöistä poiketen tulkatuille ympäristöille ei ole muodostunut standardia tapaa toteuttaa kommunikaatiota sovelluskoodin ja natiivikoodin välille. Tästä johtuen eri kehitystympäristöillä on hyvin erilaisia ratkaisuja kommunikaatiorajapinnan toteutukseen. (Biørn-Hansen ja Ghinea 2018).

Lähestymistavan etuna web- ja hybridilähestymitapoihin verrattuna on natiivien käyttöliittymäkomponenttien käytön mahdollistaminen sekä parempi suorituskyky. Sillä on kuitenkin yleensä ajonaikaisen tulkkaamisen johdosta huonompi suorituskyky verrattuna natiiviin ja ristiinkäännettyyn lähestymistapaan. Kuten hybridilähestymistavassa, sovelluskehitys on myös usein riippuvainen sovelluskehyksen tarjoamasta kokoelmasta ominaisuuksia. (Raj ja Tolety 2012)

2.2.4 Ristiinkäännetty

Ristiinkääntäjäksi kutsutaan kääntäjää, jota ajetaan eri käyttöjärjestelmässä kuin missä sillä käännettyä ohjelmaa ajetaan (El-Kassas ym. 2017). Ristiinkäännetyssä (engl. cross-compiled) lähestymistavassa kehittäjät tuottavat sovelluksen ohjelmakoodia jollakin yhteisellä kielellä ja ristiinkääntäjä kääntää sovelluksen lähdekoodin konekielelle kullekin kohdejärjestelmälle (Raj ja Tolety 2012).



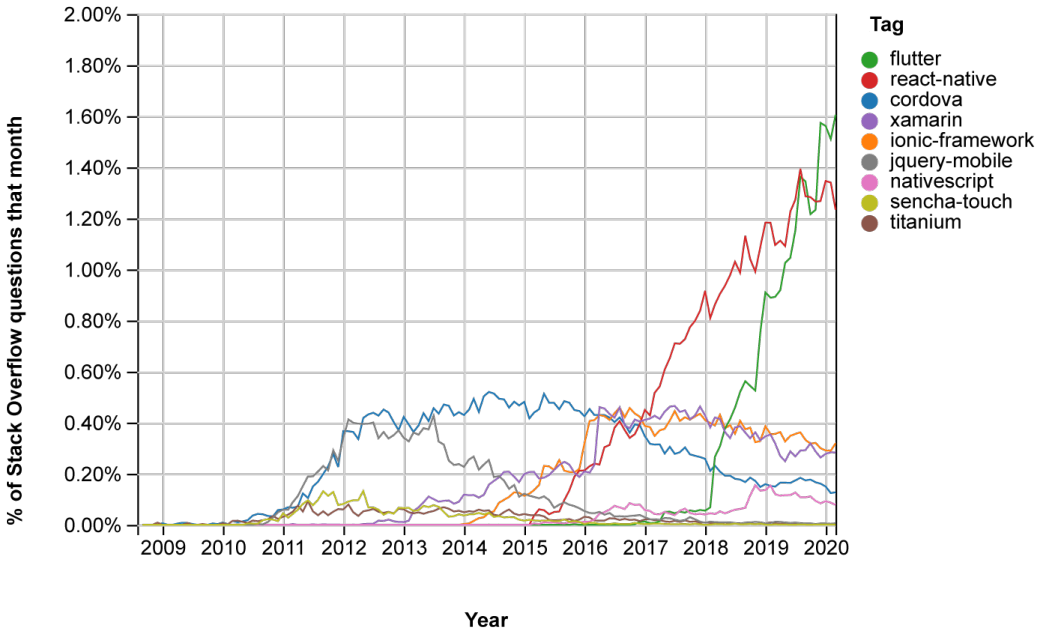
Kuvio 4. Ristiinkäännetty lähestymistapa (mukaillen Raj ja Tolety 2012)

Hybridistä ja tulkatusta lähestymistavasta poiketen ristiinkäännetyssä lähestymistavassa ei ole abstraktiokerrosta, jonka läpi natiivikoodin kanssa kommunikointi tapahtuu. Sen sijaan sen ohjelmistokehityspaketin toiminnallisuus kuvautuu suoraan eri käyttöjärjestelmien vastaaviksi toiminnoiksi. (Biørn-Hansen, Grønli ja Ghinea 2018). Ristiinkäännetyllä lähestymistavalla voidaan tuottaa sovelluksia, jotka näyttävät samalta ja käyttäytyvät kuin puhtaat

natiivisovellukset. Myös suorituskyvyn on mahdollista olla lähellä natiivisovellusta. (Ebone, Tan ja Jia 2018)

3 Alustariippumattomat kehitysympäristöt

Tämän luvun tarkoituksena on kertoa yleisimmistä nykypäivänä käytetyistä alustariippumattomista kehitysympäristöistä, niiden arkkitehtuurisista ratkaisuista sekä hyvistä ja huonoista puolista. Tässä luvussa ei käsitellä web-sovellusalustoja, vaan ainoastaan varsinaisia mobiilikehitysympäristöjä jotka tarjoavat mahdollisuuden paketoida sovellus ladattavaksi natiivipaketiksi. Käsittelyyn valitut kehitysympäristöt on valittu tarkastelemalla niiden suosituimmuutta mm. Stack Overflow trends -työkalun (ks. kuvio 5) sekä Google Trendsin avulla. Nämä antoivat hyvin samansuuntaisia tuloksia. Kaksi selkeästi suosituinta kehitysympäristöä ovat React Native ja Flutter. Tämän jälkeen on tasaisempaa, mutta seuraavina tulevat Xamarin sekä Ionic. Näistä neljästä jokainen edustaa hieman erilaista lähestymistapaa/arkkitehtuuria, joten ne sopivat hyvin vertailun kohteeksi tässä tutkielmassa.



Kuvio 5. Stack Overflow -kysymysten määrä kuukausittain
(Stack-Overflow 2020)

3.1 React Native

React Native on Facebookin vuonna 2015 kehittämä avoimen lähdekoodin sovelluskehitysympäristö, jossa ohjelmointi tapahtuu JavaScriptillä käyttäen React -kirjastoa. Käyttöliittymäkomponenttien kuvaaminen tehdään JavaScriptin syntaksia laajentavan JSX-merkkauskielen avulla, joka kääntyy käännösvaiheessa JavaScriptiksi (React 2020).

React Nativen arkkitehtuurillinen lähestymistapa tämän tutkielman kontekstissa on tulkattu ympäristö. Se tarjoaa JSX:n avulla kuvattavia adaptiivisia käyttöliittymäkomponentteja, jotka renderöityvät suoraan natiivikomponenteiksi eri käyttöjärjestelmillä. (React-Native 2020) Tällä tavalla on mahdollista saavuttaa samankaltainen tuntuma ja ulkoasu, kuin jos sovellus olisi kehitetty puhtaasti natiiveilla kehitysympäristöillä.

React Native käyttää JavaScript -moottoria sovelluskoodin tulkkaamiseen laitteella. Tämän lisäksi käytetään abstraktiokerrosta jota kutsutaan sillaksi (engl. bridge). Sillan avulla JavaScript-ohjelmakoodi kykenee kommunikoimaan eri käyttöjärjestelmien natiivikoodin kanssa tietyn protokollan avulla. Sillan kautta välitettävät viestit voivat olla esimerkiksi kutsuja käyttöliittymäkomponenttien luomiseen tai tietojen hakemiseen laitteistolta. (Gaba ja Ramachandran 2018). Nimestään huolimatta React Native ei siis tuota puhtaasti natiivia sovellusta, vaan sisältää välikerroksen jonka avulla JavaScriptillä tuotettu sovellus kykenee käyttämään natiiveja käyttöliittymäkomponentteja ja muita alustarajapintoja. Silta mahdollistaa myös kommunikoinnin natiivikoodilla kirjoitettujen moduulien (engl. native modules) kanssa. Jos React Native ei tue jotain natiiviominaisuutta, niin sen voi tarvittaessa toteuttaa itse natiivimoduulina ja hyödyntää käyttöjärjestelmän täyttä potentiaalia (React-Native 2020). Natiivimoduulit täytyy kirjoittaa erikseen jokaiselle käyttöjärjestelmälle jossa ominaisuutta halutaan tukea.

React Nativeen valmiina sisältyvien käyttöliittymäkomponenttien määrä on melko pieni, mutta tarjolla on paljon kolmannen osapuolen komponenttikirjastoja (Malik 2020). Malikin (2020) mukaan näiden laaduissa on eroja ja ulkopuolisten kirjastojen käyttöön voi sisältyä myös tietoturvariskejä. Monien kolmannen osapuolen kirjastojen ylläpito on myös lopetettu (Skuza, Mroczkowska ja Włodarczyk 2019). Tämä voi aiheuttaa ongelmia sovelluksen ylläpidon kannalta, kun vanhat kirjastot tulevat yhteensopimattomaksi päivitetyn koodipohjan

kanssa (Soral 2018).

React Nativella on muihin suosituihin kehitysympäristöihin verrattuna laajempi ja aktiivisempi kehittäjäyhteisö. Opetusmateriaaleja ja erilaisia laajennuskirjastoja löytyy suuri määrä, ja ohjelmointikielenä käytetään erittäin suosittua JavaScriptiä. Se on useita vuosia vanha, kypsä kehitysympäristö jonka ongelmia kehittäjillä on ollut aikaa korjata ja jota parannellaan ja kehitetään jatkuvasti. (Skuza, Mroczkowska ja Włodarczyk 2019). Sitä käyttävät sovelluksissaan myös useat suuret yritykset kuten Instagram, Walmart ja Facebook (Malik 2020).

3.2 Ionic

Ionic on Driftyn kehittämä avoimen lähdekoodin hybridi sovelluskehitysalusta. Ionic tukee eri JavaScript-kirjastoja, ja ohjelmakoodia voidaan tuottaa käyttäen joko Reactia, Angularia tai tulevaisuudessa myös Vuea (Ionic 2020). On myös mahdollista olla käyttämättä mitään JavaScript-kirjastoa, ja lisäksi Ionic-sovellus voidaan tarvittaessa muuntaa progressiiviseksi web-sovellukseksi (Lobastov 2019).

Tyypillisen hybridikehitysympäristön tapaan sovellusnäkymä luodaan laitteen WebView-komponentin sisälle, joten se ei käytä hyväkseen natiiveja käyttöliitymäkomponentteja. Ionic tarjoaa käyttöliittymäkirjaston, joka sisältää laajasti kustomoitavia sekä uudelleenkäytettäviä käyttöliittymäkomponentteja, jotka mukautuvat eri käyttöjärjestelmille (Ionic 2020). Taustalla käytetään Apache Cordovaa sovelluksen paketointiin eri mobiilikäyttöjärjestelmille, sekä sen liitännäisiä kommunikointiin natiivirajapintojen kanssa (Pinto ja Coutinho 2018).

Hybridiympäristöstään johtuen Ionic on suorituskyvyltään heikoin tässä tutkielmassa käsitellyistä kehitysympäristöistä. Soralin (2018) mukaan Ionic häviää esimerkiksi React Nativelle yleisesti suorituskyvyssä. Sillä on kuitenkin tuki yleisesti käytetyille web-teknologioille ja kirjastoille, joten kehittäjien löytäminen on helppoa ja sovelluksen saaminen markkinoille nopeaa (Lobastov 2019). Lobastovin (2019) mukaan sitä ei suorituskykynsä vuoksi kannata valita raskaisiin sovelluksiin, mutta useimmissa muissa tapauksissa sen suorituskyky on riittävä.

3.3 Xamarin

Xamarin on Microsoftin omistama, avoimen lähdekoodin kehitysalusta, jolla tuotetaan mobiilisovelluksia käyttäen Microsoftin .NET ohjelmistokomponenttikirjastoa, sekä ohjelmointikielenä C#:a (Microsoft 2020). Xamarinilla voidaan toteuttaa sovelluksia yksitellen eri käyttöjärjestelmille jakaen vain bisneslogiikan, tai käyttää Xamarin.Forms käyttöliittymäkirjastoa. Xamarin.Forms käyttää XML:ään perustuvaa XAML-merkkauskieltä käyttöliittymän kuvaamiseen, jonka avulla käyttöliittymäkoodia voidaan jakaa eri alustojen kesken (Microsoft 2020). React Nativen tapaan sen käyttöliittymäkomponentit kuvautuvat eri käyttöjärjestelmien natiivikomponenteiksi. Microsoftin mukaan keskimäärin 90% ohjelmakoodista voidaan jakaa kehitettäessä sovelluksia eri alustoille.

Xamarinin lähestymistavaksi voidaan määritellä tulkatun ja ristiinkäännetyn yhdistelmä. Tämä johtuu siitä, että Androidilla sovellusta ajetaan eri tavalla kuin iOS-käyttöjärjestelmässä. Androidilla Xamarin käyttää ajonaikaista kääntämistä, kun taas iOSilla sovelluskoodi käännetään ennen ajoa (Microsoft 2020). Xamarinin suorituskyvyn katsotaan yleisesti olevan lähellä natiivia (Altexsoft 2018). Ebonen ym. (2018) tutkimuksessa kuitenkin havaittiin, että käyttöliittymänäkymien renderöintiajoissa Xamarin.Formsilla tuotettu sovellus hidastui suurilla näkymillä huomattavasti verrattuna Xamarinilla erikseen, eri käyttöjärjestelmille kirjoitettuihin sovelluksiin. Altexsoft (2018) havaitsi myös tutkimuksessaan huomattavaa suorituskyvyn laskua eri operaatioissa käytettäessä Xamarin.Formsia.

Kuviosta 5 nähdään, että Xamarinin suosio on lähellä Ionicia, eikä yllä Flutterin tai React Nativen tasolle. Sillä on myös mm. huomattavasti vähemmän siihen liittyviä GitHub-repositorioita kuin muilla tässä tutkielmassa vertailtavilla kehitysympäristöillä. Xamarin on hyvä valinta kehitysympäristöksi, jos kehittäjät käyttävät jo valmiiksi .NET -tekniikoita tai paljon muita Microsoftin tuotteita. (Altexsoft 2018).

3.4 Flutter

Flutter on Googlen kehittämä avoimen lähdekoodin käyttöliittymäkirjasto ja ohjelmistokehityspaketti. Sovellusohjelmointi tapahtuu käyttäen Dart-ohjelmointikieltä, jonka syntaksi on samankaltainen mm. Javan ja JavaScriptin kanssa. Flutter tarjoaa kokoelman käyttöliittymää

kuvaavia komponentteja, joita kutsutaan widgeteiksi (engl. widget). (Flutter 2020). Widgetit ovat rakennuspalikoita jotka kuvaavat käyttöliittymän tyyliä, rakennetta ja ulkoasua (Shah, Sinha ja Mishra 2019). Käyttöliittymä kuvataan widgettejä puurakenteeksi yhdistämällä ja ne voivat olla tilattomia, tai sisältää tilan joka voi muuttua widgetin eliniän aikana (esim. teksti- tai värimuutos) (Flutter 2020). Flutter tarjoaa valmiina kahdentyyppisiä widgettejä, Cupertino-widgetit imitoivat iOSin tyyliä ja Material Design -widgetit googlen samannimistä muotoilukieltä. Flutter ei renderöi natiiveja käyttöliittymäkomponentteja, vaan widgetit ovat täysin alustariippumattomia. Androidille ja iOSille voi toteuttaa täysin samannäköisen sovelluksen, mutta myös tarvittaessa erotella tyylejä eri alustan mukaan. (Skuza, Mroczkowska ja Włodarczyk 2019)

Flutterin arkkitehtuuri sijoittuu tämän tutkielman kontekstissa ristiinkäännettyyn lähestymistapaan. Monista muista kehitysympäristöistä poiketen Flutter ei käytä WebView-komponenttia tai laitevalmistajien käyttöliittymäkomponentteja, vaan se hyödyntää omaa renderöintimoottoria omien käyttöliittymäkomponenttien piirtämiseen (Flutter 2020). Julkaisuversioissa se kääntää Dart-koodin suoraan konekielelle, ja kehityksen aikana sovellusta ajetaan virtuaalikoneessa joka mahdollistaa nopean uudelleenlataamisen sovelluskoodia muutettaessa (Flutter 2020). Suoraan konekielelle kääntämisestä voi seurata suorituskyvyllisiä etuja. Esimerkiksi Demedyukin ja Tsybulskyin (2020) mukaan Flutter suoriutuu muistinkäyttö- ja suoritinintensiivisistä operaatioista huomattavasti nopeammin kuin React Native ja on usein lähellä jopa natiivia.

Flutter on nuorempi kehitysympäristö kuin sen suurin kilpailija React Native. Sen kehittäjäyhteisö ei ole yhtä suuri ja ohjelmakirjastovalikoima on pienempi. Se myös käyttää verrattain tuntematonta ja vähän käytettyä Dart-ohjelmointikieltä (Skuza, Mroczkowska ja Włodarczyk 2019). Kuten kuviosta 5 huomataan, Flutterin nousu suosituksi kehitysympäristöksi on kuitenkin ollut todella nopea, joten sen tulevaisuus näyttää lupaavalta.

4 Yhteenveto

Tutkielmassa havaittiin, että jos sovelluksilta halutaan parasta mahdollista suorituskykyä ja laitekontrollia, natiivi on edelleen paras vaihtoehto. Alustariippumattomien ratkaisujen ylimääräiset abstraktiokerrokset madaltavat suorituskykyä, ja laiterajapinnat ovat usein pienemmät kuin natiiveilla ohjelmistokehityspaketeilla. Niiden avulla sovelluksia on kuitenkin potentiaalisesti huomattavasti nopeampi, halvempi ja helpompi tuottaa. On tärkeää tunnistaa millaisia ominaisuuksia sovellukselta vaaditaan, jotta voidaan valita oikea kehitysympäristö. Kehittäminen voi muodostua haastavaksi, jos valitaan alustariippumaton kehitysympäristö jonka liitännäiskokoelmat eivät ole riittävät sovelluksessa vaadituille ominaisuuksille. Suosituimmat kehitysympäristöt mahdollistavat omien liitännäisten tuottamisen, mutta kehittäjät eivät välttämättä ole varautuneet kirjoittamaan natiivikoodia kullekin alustalle erikseen. Tässä tapauksessa valinnasta saatavat hyödyt voivat pienentyä huomattavasti.

React Native ja Flutter ovat tällä hetkellä kaksi selkeästi suosituinta alustariippumatonta kehitysympäristöä. React Native on vanhempi, sen kehittäjäyhteisö on laajempi ja sille löytyy helpommin kolmannen osapuolen liitännäisiä. Flutterin suosio on kuitenkin nopeassa nousussa, sen yhteisö laajenee nopeasti ja suorituskyky on hyvä. Sillä on lisäksi takanaan Google, jolta varmasti löytyy resursseja kehittämiseen ja ylläpitoon. Muillekin kehitysympäristöille on kuitenkin paikkansa. Hybridikehitysympäristöillä kuten Ionicilla on helppo tuottaa sovelluksia web-teknologioilla, ja saada sovellus nopeasti markkinoille. Ne voivat olla hyvä valinta, jos ei vaadita natiivia ulkoasua tai parasta mahdollista suorituskykyä. Xamarin taas soveltuu valinnaksi, jos halutaan hyvää suorituskykyä ja .NET -tekniikat sekä Microsoftin tuotteet ovat tuttuja. Mobiililaitteiden ja internet-yhteyksien nopeutuessa on myös mielenkiintoista nähdä, millaiseksi progressiivisten web-sovellusten rooli muodostuu tulevaisuudessa.

Erilaiset alustariippumattomat lähestymistavat ja kehitysympäristöt eroavat toisistaan monin tavoin. Niiden yksityiskohtainen vertailu on kuitenkin monista syistä haastavaa. Kehitysympäristöjen suosio vaihtelee ja uusia versiopäivityksiä tulee jatkuvasti. Lisäksi erilaisten lähestymistapojen lisääntyminen ja arkkitehtuurien muuttuminen tekee termistön määrittelyn vaikeaksi. Akateemiset tutkimukset aiheesta menettävät merkityksellisyyttään nopeasti, ja

ajantasaista tietoa on haasteellista löytää.

Lähteet

Ahmad, A., K. Li, C. Feng, S. M. Asim, A. Yousif ja S. Ge. 2018. “An Empirical Study of Investigating Mobile Applications Development Challenges”. *IEEE Access* 6:17711–17728. ISSN: 2169-3536. doi:10.1109/ACCESS.2018.2818724.

Altexsoft. 2018. *Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison.* Saatavilla WWW-muodossa, https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/, viitattu 30.4.2020.

Biørn-Hansen, Andreas, ja George Ghinea. 2018. “Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development”. Teoksessa *HICSS.*

Biørn-Hansen, Andreas, Tor-Morten Grønli ja Gheorghita Ghinea. 2018. “A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development”. *ACM Comput. Surv.* (New York, NY, USA) 51, numero 5 (marraskuu). ISSN: 0360-0300. doi:10.1145/3241739. https://doi-org.ezproxy.jyu.fi/10.1145/3241739.

Demedyuk, Ihor, ja Nazar Tsybulskyi. 2020. *Flutter vs Native vs React-Native: Examining performance.* Saatavilla WWW-muodossa, https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980, viitattu 29.4.2020.

Ebone, A., Y. Tan ja X. Jia. 2018. “A Performance Evaluation of Cross-Platform Mobile Application Development Approaches”. Teoksessa *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 92–93. Toukokuu.

Flutter. 2020. *Flutter Docs.* Saatavilla WWW-muodossa, https://flutter.dev/docs/, viitattu 30.3.2020.

Gaba, Rahul, ja Atul Ramachandran. 2018. *React Native internals - React Made Native Easy.* Saatavilla WWW-muodossa, https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html, viitattu 28.4.2020.

Google. 2019. *Introduction to service worker.* Saatavilla WWW-muodossa, https://developers.google.com/web/ilt/pwa/introduction-to-service-worker, viitattu 27.4.2020.

Ionic. 2020. *Ionic Docs.* Saatavilla WWW-muodossa, https://ionicframework.com/docs/, viitattu 11.3.2020.

El-Kassas, Wafaa S., Bassem A. Abdullah, Ahmed H. Yousef ja Ayman M. Wahba. 2017. "Taxonomy of Cross-Platform Mobile Applications Development Approaches". *Ain Shams Engineering Journal* 8 (2): 163–190. ISSN: 2090-4479. doi:https://doi.org/10.1016/j.asej.2015.08.004. http://www.sciencedirect.com/science/article/pii/S2090447915001276.

Latif, Mounaim, Younes Lakhrissi, El Habib Nfaoui ja Najia Es-Sbai. 2016. "Cross platform approach for mobile application development: A survey", 1–5. Maaliskuu. doi:10.1109/IT4OD.2016.7479278.

Lobastov, Ihor. 2019. *The Good and the Bad of Ionic Mobile Development.* Saatavilla WWW-muodossa, https://dzone.com/articles/the-good-and-the-bad-of-ionic-mobile-development, viitattu 30.4.2020.

Macquarrie, Ashley. 2018. *Hybrid Apps vs. Native Apps: Which Should You Build?* Saatavilla WWW-muodossa, https://themanifest.com/mobile-apps/hybrid-apps-vs-native-apps-which-should-you-build, viitattu 15.4.2020.

Malavolta, I., G. Procaccianti, P. Noorland ja P. Vukmirovic. 2017. "Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps". Teoksessa *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 35–45.

Malik, Nyma. 2020. *React Native: Pros and Cons.* Saatavilla WWW-muodossa, https://citrusbits.com/react-native-pros-and-cons/, viitattu 28.4.2020.

Microsoft. 2020. *Microsoft Xamarin documentation.* Saatavilla WWW-muodossa, https://docs.microsoft.com/en-us/xamarin/, viitattu 29.4.2020.

Nunkesser, R. 2018. "Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development". Teoksessa *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 214–218. Toukokuu.

Pinto, C. M., ja C. Coutinho. 2018. "From Native to Cross-platform Hybrid Development". Teoksessa *2018 International Conference on Intelligent Systems (IS),* 669–676. Syyskuu. doi:10.1109/IS.2018.8710545.

Que, P., X. Guo ja M. Zhu. 2016. "A Comprehensive Comparison between Hybrid and Native App Paradigms". Teoksessa *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN),* 611–614. Joulukuu. doi:10.1109/CICN.2016.125.

Raj, Rahul, ja Seshu Babu Tolety. 2012. "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach". Teoksessa *2012 Annual IEEE India Conference (INDICON),* 625–629.

React. 2020. *React Docs.* Saatavilla WWW-muodossa, https://reactjs.org/docs, viitattu 15.4.2020.

React-Native. 2020. *React Native Docs.* Saatavilla WWW-muodossa, https://reactnative.dev/docs/, viitattu 11.3.2020.

Shah, K., H. Sinha ja P. Mishra. 2019. "Analysis of Cross-Platform Mobile App Development Tools". Teoksessa *2019 IEEE 5th International Conference for Convergence in Technology (I2CT),* 1–7.

Sharma, Sagar. 2019. *Pros Cons of Hybrid Mobile Apps Development.* Saatavilla WWW-muodossa, https://www.credencys.com/blog/pros-cons-of-hybrid-mobile-apps-development/, viitattu 15.4.2020.

Skuza, Bartosz, Agnieszka Mroczkowska ja Damian Włodarczyk. 2019. *Flutter vs. React Native – What to Choose in 2020?* Saatavilla WWW-muodossa, https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2020, viitattu 28.4.2020.

Soral, Rakshit. 2018. *React Native vs Ionic: Comparing performance, User Experience and much more!* Saatavilla WWW-muodossa, https://www.simform.com/react-native-vs-ionic/, viitattu 19.2.2020.

Stack-Overflow. 2020. *StackOverFlow Trends.* Saatavilla WWW-muodossa, https://insights.stackoverflow.com/trends/, viitattu 15.4.2020.