Erkko Mäkinen

Monialustaisten mobiilisovellusten kehittämistavat

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2017

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Erkko Mäkinen

Yhteystiedot: erkko.e.makinen@student.jyu.fi

Työn nimi: Monialustaisten mobiilisovellusten kehittämistavat

Title in English: Different approaches for building cross-platform mobile applications

Työ: Kandidaatintutkielma

Sivumäärä: 27+0

Tiivistelmä: Mobiililaitteiden käyttö on lisääntynyt huomattavasti lähivuosina, joten sovelluksia kehitetään yhä enemmän. Sovellusten kehittäminen kuhunkin ympäristöön natiivisti on kuitenkin kallista ja vaatii runsaasti aikaa, sillä sovellukset joudutaan luomaan alustoille erikseen. Monialustainen mobiilisovelluskehitys pyrkii tuomaan ratkaisun tähän ongelmaan mahdollistamalla sovelluksen toiminnan usealla alustalla yhden ohjelmakoodin pohjalta.

Tässä tutkielmassa käydään läpi mitkä hyödyt ja haasteet ovat tyypillisiä natiiveille web-pohjaisille, hybrideille, tulkatuille ja käännetyille sovelluksille. Lisäksi tutkielmassa vertaillaan neljän erilaisen ohjelmistokehyksen ominaisuuksia, sekä mitä rajoitteita ja mahdollisuuksia ne asettavat kehittäjille. Lopuksi käydään läpi mikä monialustaisista sovellustyypeistä soveltuu parhaiten tietynlaisille mobiilisovelluksille.

Avainsanat: monialustainen, mobiili, sovelluskehitys

Abstract: The popularity of smartphones has been rising significantly and applications are being built at a rising pace. However, it is expensive to develop applications separately for different operating systems because they are based on different programming languages and development tools. Cross-platform development tries to solve this problem by making it possible to reach different platforms by relying only on one codebase.

This thesis goes through different cross-platform mobile application types and evaluates their advantages and challenges. There are also four different cross-platform frameworks that are compared to each other. In the last chapter there is a discussion about which development approach fits best for the specific mobile application types.

Keywords: cross-platform, mobile, application, software development

Kuviot

Sisältö

1 Johdanto

Mobiililaitteiden yleistyttyä mobiilisovelluksia kehitetään nykyään yhä enemmän, ja mitä erilaisempiin käyttötarkoituksiin. Monet työpöytäsovellukset ovatkin löytäneet tiensä mobiililaitteille niiden suorituskyvyn kasvaessa. Suosiota selittää myös se, että mobiililaitteet kulkevat käyttäjiensä mukana, ja niiden käyttö on helppoa sekä nopeaa (Taneja, Taneja ja Bhullar 2016). Laitteiden laskentatehon ja ominaisuuksien kasvaminen on myös edistänyt mobiilisovelluksien mahdollisuuksia.

Mobiililaitteen ominaisuuksia päästään käyttämään siihen asennetun käyttöjärjestelmän kautta. Käyttöjärjestelmiä on tarjolla useita erilaisia, mutta merkittävimpinä näistä mainittakoon Android ja iOS, jotka pitävät sisällään selvästi muita korkeamman markkinaosuuden, noin 96% tämän tutkielman kirjoitushetkellä (Netmarketshare 2017). Ongelmana alustojen moninaisuudessa on pääasiassa se, että ne pohjautuvat usein erilaisiin ohjelmointikieliin ja kehitystyökaluihin (Charkaoui, Adraoui ja Benlahmar 2014a). Kehitettäessä sovelluksia usealle alustalle natiiveja kehitysympäristöjä käyttäen, kehittäjältä vaaditaan paljon alustakohtaista tietämystä ja osaamista. Sen lisäksi ohjelman kirjoittaminen jokaiselle alustalle erikseen vaatii huomattavasti ylimääräisiä työtunteja ja muita kustannuksia (Taneja, Taneja ja Bhullar 2016). Ongelman ratkaisemiseksi on kuitenkin kehitetty erilaisia ohjelmistokehyksiä, joiden avulla kerran kirjoitettua ohjelmakoodia voidaan hyödyntää useille alustoille. Näiden työkalujen käyttö on lähivuosina saavuttanut suosiota, sillä yritykset haluavat saada tuotteensa saataville mahdollisimman suurelle ihmisjoukolle nopeasti ja vähällä vaivalla. Lisäksi erillisten natiivien sovellusten ylläpitäminen useille alustoille on haastavaa ja kallista. Kuitenkaan täysin toimivaa lähestymistapaa monialustakehitykseen ei vielä ole, sillä se tuo mukanaan omat rajoitteensa (El-Kassas et al. 2015).

Monialustakehitykseen on tarjolla useita erilaisia ohjelmistokehyksiä, jotka eroavat toisistaan enemmän tai vähemmän. Ne voidaan jakaa karkeasti kahteen eri luokkaan: web-pohjaisiin ja natiiveihin ohjelmistokehyksiin (Boushehrinejadmoradi et al. 2015). Web-pohjaisten ohjelmistokehysten avulla sovellukset luodaan käyttäen apuna yleisiä web-teknologioita. Näitä ovat mm. HTML5, CSS, ja JavaScript. Natiiveilla ohjelmistokehyksillä sovellus ohjelmoidaan tyypillisesti tietyllä ohjelmointikielellä, jonka jälkeen se käännetään muille halutuille

kohdealustoille. Näiden kahden lähestymistavan välillä on kuitenkin eroja ja välimuotoja, jotka tulee ottaa huomioon kehitystapaa valittaessa.

Tässä tutkielmassa on tarkoitus käydä läpi yleisesti monialustaisten mobiilisovellusten kehittämistä, kuten mitä erilaisia tapoja sovellusten rakentamiseen on, mitä hyötyjä ja haittoja näiden tapojen välillä on, ja kuinka ne eroavat toisistaan. Lisäksi tutkielmassa vertaillaan neljän eri ohjelmistokehyksen ominaisuuksia monesta eri näkökulmasta. Tutkielman lopussa tarkastellaan mitkä monialustaisista kehitystavoista soveltuvat parhaiten palvelinpainotteisille, sensori/IO -pohjaisille, itsenäisille ja asiakas-palvelin sovelluksille.

2 Mobiilisovellusten tyypit

Monialustaisten mobiilisovellusten kehittämiseen on tarjolla useita erilaisia kehitystapoja, jotka johtavat erilaisiin lopputuloksiin. Kullakin eri menetelmällä on oma tarkoituksensa omine hyötyineen ja haittoineen, joten sovelluskehittäjän tulee tarkkaan miettiä mitä kehitystapaa oman tuotteen kehittämiseen aiotaan käyttää hyödyn maksimoimiseksi. Tässä tulee ottaa huomioon mitä vaatimuksia sovelluksella on ja millaiselle kohderyhmälle se aiotaan suunnata. Monialustaisten mobiilisovellusten kehitystavat voidaan Raj ja Toledyn (2012) mukaan luokitella pääasiassa web-pohjaiseen, hybridiin, tulkattavaan ja käännettävään lähestymistapaan. Tässä luvussa käydään läpi kunkin mobiilisovellustyypin ja kehitystavan perustoiminta sekä niiden tuomia hyötyjä ja haasteita.

2.1 Natiivit sovellukset

Jokaisella eri alustalla on oma natiivi infrastruktuurinsa sovelluksien kehittämiseen (Raj ja Tolety 2012). Natiivilla kehitystavalla sovellus ohjelmoidaan tietylle kohdealustalle sen tukemaa ohjelmointikieltä ja kehitystyökaluja (engl. Software development kit) hyödyntäen. Esimerkiksi Androidille sovellukset ohjelmoidaan tyypillisesti Javalla ja vuorostaan iOS:sille Objective C:llä tai Swiftillä (Tunali ja Erdogan 2015). Natiiveilla sovelluksilla on pääsy laitteen kaikkiin ominaisuuksiin ohjelmointirajapintojen kautta, joten kehityksessä on kaikki mahdollisuudet käytettävissä. Volkan ja Zafer (2012) kirjoittavat, että sovelluksen ollessa täysin sidottuna alustaansa, se näyttää ja tuntuu juuri kyseiselle alustalle tehdyltä. Tämä puolestaan tekee käyttäjäkokemusesta miellyttävämmän ja sulavamman. Lisäksi, koska natiivit sovellukset ajetaan laitteen natiivissa ympäristössä, niiden suorituskyky on mahdollisimman korkea (Hartmann, Stead ja DeGani 2011).

Natiivit sovellukset ladataan tyypillisesti mobiililaitteille alustakohtaisista sovelluskaupoista (El-Kassas et al. 2015). Tämä ei kuitenkaan tarkoita, että kaikki sovelluskaupoissa olevat sovellukset ovat natiiveja, sillä sieltä on ladattavissa myös hybridejä sovelluksia, jotka yhdistelevät natiivien ja web-pohjaisten sovellusten piirteitä. Mobiilisovelluksille on suuri etu siitä, että ne ovat ladattavissa sovelluskaupoista, sillä käyttäjät löytävät sovellukset sieltä hel-

posti, ja jokainen sieltä ladattavissa oleva sovellus on tarkistettu sen henkilökunnan toimesta (Ohrt ja Turau 2012). Käyttäjän ei siis tarvitse miettiä onko sovelluksissa tietoturvaan tai mihinkään muuhunkaan liittyviä riskejä. Yksittäisiä poikkeuksia toki on, mutta ne läpäisevät monipuolisen tarkastuksen äärimmäisen harvoin ja ne poistetaan paljastuttuaan välittömästi. Lisäksi sovelluskauppojen etuna on se, että latauspaikkana toimii yksi ja sama vakiintunut ympäristö.

Edellä mainittujen hyötyjen lisäksi on natiivilla kehitystavalla myös huonot puolensa. Kehittäminen natiivisti on yleisesti ottaen kallista, sillä ohjelma on kirjoitettava jokaiselle alustalle erikseen ja ylläpito on tällöin haastavaa sekä hidasta (El-Kassas et al. 2015). Tämä kehitystapa koituu kalliiksi varsinkin niille, jotka haluavat sovelluksen saataville mahdollisimman monelle alustalle. Lisäksi sovelluksen hyväksyminen sovelluskauppaan voi olla pitkä ja hankala prosessi. Tämä on kuitenkin alustakohtaista, sillä sovellusten hyväksymiskriteerit voivat vaihdella toisistaan (Ohrt ja Turau 2012). Esimerkiksi Apple on tunnettu tiukoista ehdoistaan liittyen sovelluksien hyväksymiskriteereihin.

2.2 Web-pohjaiset sovellukset

Web-pohjaisessa kehityksessä (engl. web-based development) mobiilisovellukset luodaan hyödyntäen yleisiä web-teknologioita, kuten HTML:ää, JavaScriptiä sekä CSS:ää. Täten niiden käyttö tapahtuu selainten kautta tietyn palvelimen välityksellä, eikä niitä tarvitse erikseen asentaa laitteelle (Latif et al. 2016). Sovellus tulee siis kehittää vain kerran, ja selainten ollessa standardoituja, sitä voidaan käyttää sujuvasti monilla eri alustoilla. Aiemmin mainitut web-teknologiat ovat myöskin helpompi omaksua verrattuna muihin kehitystapoihin, joten varsinkin aloittelevalle kehittäjälle tämä on hyvä vaihtoehto. Lisäksi web-pohjaisten sovellusten ylläpito on helppoa, koska kaikki päivitykset sovellukseen ja dataan tehdään palvelimen välityksellä (El-Kassas et al. 2015). Ei siis tarvitse tehdä samoja muutoksia useissa eri ympäristöissä, jolloin ylläpitäminen ei ole niin kallista ja työlästä.

Web-pohjaisissa sovelluksissa on kuitenkin myös huonot puolensa. Yleisin tapa nykyään mobiilisovellusten jakamiselle on alustakohtaiset sovelluskaupat, ja koska web-pohjaisia sovelluksia käytetään selaimen välityksellä, niiden suosio ja näkyvyys voi kärsiä tästä (H. Heit-

kötter ja Majchrzak 2012). Sovelluskaupoissa tuotteiden monetisaatio on myöskin yksinkertaisempaa kuin web-sovelluksilla, sillä sovelluksen lataamisesta voidaan veloittaa jokin tietty summa. Siinä, ettei sovelluksia ladata ja asenneta sovelluskaupoista, on kuitenkin myös hyvä puolensa, sillä sovelluksia päästään käyttämään URL:in kautta suoraan ilman ylimääräistä odottelua.

Web-pohjaiset sovellukset ovat yleensä ainakin osittain riippuvaisia internet-yhteydestä. Vaikka yhteys löytyykin nykyään lähes jokaiselta mobiilikäyttäjältä, se ei ole kaikissa olosuhteissa mahdollinen, jolloin sovelluksen käyttö voi rajoittua tai jopa estyä kokonaan. Suuri osa web-pohjaisista mobiilisovelluksista käyttää kuitenkin local storagea tai selaimen välimuistia tallentamaan sivun sisältöä, jolloin sovellusta voidaan käyttää myös yhteydettömässä tilassa, ja sovelluksen sisältö latautuu nopeampaa. Tämä ei ole kuitenkaan ratkaise ongelmaa, jos dataa tulee vastaanottaa palvelimelta tai lähettää sitä sinne reaaliajassa. Web-pohjaisten sovellusten nopeus kärsii myös siitä, että HTML ja JavaScript ajetaan selaimen, eli ylimääräisen kerroksen kautta (El-Kassas et al. 2015).

Vaikka web-pohjaisten sovellusten ohjelmakoodi tulee kirjoittaa kerran, sen täytyy lisäksi tukea erilaisia resoluutioita, jotta käyttäjäkokemus säilyisi hyvänä (Latif et al. 2016). Tämä tuo mukanaan lisätyötä, sillä käyttöliittymän manuaalinen testaaminen eri resoluutioille voi olla työlästä ja pitkäveteistä. Tätä varten on kuitenkin kehitetty erilaisia ohjelmistokehyksiä, joilla responsiivisten verkkosivujen tekeminen on helppoa. Tällaisia ovat esimerkiksi Jquery Mobile, Sencha Touch ja jQTouch (Raj ja Tolety 2012).

Vielä muutamia vuosia sitten web-pohjaisilla sovelluksilla oli hyvin rajallinen pääsy laitteen natiiveihin ominaisuuksiin, mikä saattoi ajaa sovelluskehittäjiä muita kehitystapoja kohti. Nykyään mahdollisuudet ovat kuitenkin kasvaneet huomattavasti, sillä selainten kautta päästään hyödyntämään jo suurinta osaa näistä ominaisuuksista erilaisten ohjelmointirajapintojen avulla. Ominaisuuksia joihin natiiveilla sovelluksilla päästään, mutta web-pohjaisilla sovelluksilla ei päästä on tämän tutkielman tekohetkellä mm. kontaktit, kalenteri ja järjestelmän asetukset (Bar 2017). Jotkut käyttäjät voivat pitää näitä puutteita jopa etuna tietoturvan suhteen.

Vuonna 2015 Google esitteli uuden menettelytavan web-pohjaisten sovellusten kehittämi-

seen: progressiiviset web-sovellukset. Ne eroavat perinteisemmistä web-pohjaista sovelluksista siten, että ne hyödyntävät webin moderneja mahdollisuuksia tehdäkseen web-sivusta mahdollisimman sovellusmaisen (LePage 2017). Jotta web-pohjaista sovellusta voitaisiin sanoa progressiivikseksi, sen täytyy täyttää tietyt tarkasti määritellyt kriteerit. Näihin kuuluu mm. sovelluksen toimivuus kaikille käyttäjille selaimesta riippumatta, responsiivisuus, yhteysriippumattomuus hyödyntäen service workeriä, ja monia natiiveille sovelluksille tyypillisiä ominaisuuksia, kuten push-viestit ja sovelluksen lisääminen laitteen sovellusvalikkoon (LePage 2017). Lisäksi niille on ominaista hyödyntää app-shell mallia, jolloin käyttöliittymää ei ladata aina uudestaan sovellusta käytettäessä, vaan ainoastaan tarvittava sisältö haetaan palvelimelta. Tämä nopeuttaa sovellusten toimivuutta huomattavasti.

2.3 Hybridit sovellukset

Hybridit sovellukset (engl. hybrid applications) yhdistelevät sekä natiivien että web-pohjaisten sovellusten hyötyjä ja menetelmiä (Charkaoui, Adraoui ja Benlahmar 2014b). On kaksi erilaista tapaa näyttää web-sisältöä mobiililaitteella: perinteisen selaimen tai web-näkymän kautta, mikä on alustan natiivi komponentti web-sisällön esittämiseen ruudulla. Androidissa web-näkymänänä käytössä on WebView, ja iOS:issa UIWebView (Trice 2013). Hybridien sovellusten toiminta perustuu siihen, että web-näkymä upotetaan natiivin sovelluskuoren sisään, ja HTML-sisältö näytetään siinä. Sovelluksessä näytettävä sisältö voi olla paikallista, palvelimelta ladattavaa, tai molempia edellä mainituista vaihtoehdoista.

Hybridit sovellukset luodaan käyttäen samoja web-teknologioita kuin mitä web-pohjaisten sovellusten kehittämisessä käytetään. Suurimpana erona web-pohjaisiin sovelluksiin on se, että hybrideillä sovelluksilla pääsee käsiksi laitteen natiiveihin komponentteihin erillisen JavaScript -ohjelmointirajapinnan kautta (El-Kassas et al. 2015). Laitteen natiivien komponenttien hyödyntämismahdollisuudet voivat tuoda sovellukselle uusia ulottuvuuksia verrattuna perinteisiin web-sovelluksiin, joilla pääsy laitteistoon on rajatumpaa ja vaihtelevaa selaimittain ja versioittain. Lisäksi Hybridit sovellukset ovat ladattavissa alustakohtaisista sovelluskaupoista, jossa ne ovat hyvin näkyvillä ja helposti saatavilla (Raj ja Tolety 2012).

Hybridien sovellusten pääasiallisena etuna on se, että käyttöliittymää voidaan uudelleenkäyt-

tää useilla eri alustoilla, samalla hyödyntäen laitteiden natiiveja ominaisuuksia (Raj ja Tolety 2012). Sovellus tulee siis luoda vain kerran, vaikka kunnollista käyttäjäkokemusta tällöin vielä ei kuitenkaan saada aikaiseksi. Lisäksi hybridit sovellukset eivät välttämättä tarvitse internet-yhteyttä toimiakseen, vaan ne voivat säilöä aiemmin haettua dataa ja hyödyntää sitä laitteen ollessa yhteydettömässä tilassa.

Hybridien sovellusten haasteena on kuitenkin se, että ne ovat hitaampia kuin natiivit sovellukset. Tämä johtuu siitä, että hybridien sovellusten arkkitehtuurissa on useita erillisiä kerroksia, jolloin ohjelmakoodin suorittaminen vaatii ylimääräisiä askeleita. Sovellusten mahdollinen riippuvuus internet-yhteydestä voi myös hidastaa sisällön latautumista ja prosessointia. Lisäksi käyttäjäkokemuksen maksimoimiseksi sovellukset yleensä tarvitsevat jonkinlaista modifointia eri alustoille, jotta käyttöliittymä tuntuisi ja näyttäisi mahdollisimman paljon natiivilta sovellukselta. Tätä varten on luotu erilaisia ohjelmistokehyksiä ja lisäosia, joilla käyttöliittymän toteutukseen liittyviä ongelmia saadaan vähennettyä (Latif et al. 2016).

2.4 Tulkatut sovellukset

Tulkatut sovellukset (engl. interpreted applications) käyttävät alustakohtaisia natiiveja käyttöliittymäkomponentteja, kun taas ohjelman logiikka on alustariippumattomassa muodossa (Raj ja Tolety 2012). Niiden toiminta perustuu ajonaikaiseen ohjelman tulkkaamiseen, tähän kehitettyä moottoria hyödyntämällä. Tämä menettelytapa voidaan luokitella kolmeen eri alakategoriaan: virtuaalikone-, web-pohjaiseen-, ja ajonaikaiseen tulkkaamiseen (El-Kassas et al. 2015). Toimintalogiikka monelle alustalle voidaan määritellä joukolla erilaisia käskyjä, joko XML:ää tai jotain muuta kuvailevaa kieltä hyödyntäen (Raj ja Tolety 2012).

Tulkattujen sovellusten etuna on se, että ohjelman toimintalogiikka tulee kirjoittaa vain kerran, ja sovellukset tuntuvat ja näyttävät natiiveilta sovelluksilta. Lisäksi ne ovat ladattavissa alustojen sovelluskaupoista aivan kuten suurin osa mobiilisovelluksista. Laitteen natiiveihin ominaisuuksiin päästään myös käsiksi helposti tietynlaisen abstraktiokerroksen kautta (Raj ja Tolety 2012).

Tulkattujen sovellusten pääongelmana on se, että kehitystyö riippuu vahvasti käytettävästä ohjelmistokehyksestä ja sen ominaisuuksista (Latif et al. 2016). Alustojen ja laitteiden ke-

hittyessä myös ohjelmistokehysten on mukauduttava muutoksiin, mikä ei kuitenkaan aina tapahdu hetkessä. Esimerkiksi uutta alustakohtaista käyttöliittymäkomponenttia tai ominaisuutta ei päästä hyödyntämään ellei ohjelmistokehys tue niitä (Latif et al. 2016). Lisäksi tulkattujen sovellusten suorituskyky on natiiveihin sovelluksiin verrattuna hieman alhaisempi (Raj ja Tolety 2012).

2.5 Monialustaiset käännetyt sovellukset

Tämän lähestymistavan (engl. cross-compiled applications) toiminta perustuu korkeamman tason ohjelmointikielellä tuotettuun ohjelmakoodiin, jonka kääntäjä kääntää natiiviksi sovellukseksi tietyille kohdealustoille. Kääntäjällä on siis vastuu menetelmän tehokkuudesta ja luotettavuudesta (Raj ja Tolety 2012).

Etuna tässä menetelmässä on se, että kohdealustoille käännetyt sovellukset saavat kaikki natiivien sovellusten edut, kuten laitteiston ja ohjelmiston rajattoman hyödyntämisen. Käännettyjen sovellusten nopeus on myös mahdollisimman korkea, sillä natiiveilla sovelluksilla ei ole ylimääräisiä kerroksia hidastamassa ohjelman ajamista, vaan ohjelmakoodi on valmiiksi käännettynä konekielelle (Latif et al. 2016).

Suurimpana haasteena käännetyillä sovelluksilla se, että käyttöliittymä on tehtävä vartavasten jokaiselle alustalle erikseen (Raj ja Tolety 2012). Tämä johtuu siitä, että käyttöliittymäkomponentit ovat alustan natiiveja komponentteja, eli niitä ei voida käyttää muilla alustoilla. Lisäksi vaikka alustakohtaisiin ominaisuuksiin päästäänkin käsiksi, niin ne joudutaan kirjoittamaan jokaiselle alustalle erikseen, sillä niiden ominaisuudet vaihtelevat alustoittain.

Markkinoilla olevista työkaluista löytyy vielä paljon puutteita, koska ne ovat vielä kehittymässä. Varsinkin erilaisten ohjelmointivirheiden tunnistaminen ja korjaaminen erityisesti monikäännösvaiheessa voi olla hidastava tekijä sovelluskehittäjille (Raj ja Tolety 2012).

Seuraavan sivun kuviossa 1. esitetään web-pohjaisten, hybridien ja natiivien sovellusten rakenteelliset eroavaisuudet. Tulkatut ja käännetyt voidaan luokitella natiiveiksi sovelluksiksi, vaikka niiden kehitystavat poikkeavat alustakohtaisesta natiivista kehityksestä.

Kuvio 1. Web-pohjaisten, hybridien ja natiivien mobiilisovellusten eroavaisuudet (mukaillen Worklight, 2011a)

3 Ohjelmistokehykset

Mobiilisovelluskehittäjillä on yleensä tavoitteena saada tuotteensa saataville mahdollisimman suurelle ihmisjoukolle, mutta tämän saavuttamiseksi he joutuvat tekemään valintoja kehitystapojensa suhteen. Ensiksi tulee arvioida, mitä vaatimuksia sovelluksella on, ja mitkä kehitystavoista sopivat parhaiten yhteen näiden vaatimusten kanssa. Erilaisille sovellustyypeille on lukuisia ohjelmistokehyksiä, joista tulee tehdä valinta. Kehittäjien on kuitenkin hyvä tietää, mitä ominaisuuksia mitkäkin ohjelmistokehykset pitävät sisällään, ja miten ne eroavat toisistaan. Tässä luvussa käydään läpi neljän tunnetun ohjelmistokehyksen peruspiirteitä ja toimintaa. Ohjelmistokehykset on valittu tutkielmaan siten, että niiden käyttämien menetelmien ja niillä kehitettyjen sovellusten välillä on eroavaisuuksia. Tämä antaa mahdollisuudet monipuoliseen vertailuun niiden ominaisuuksien suhteen. Vertailun tuloksia käydään läpi tämän luvun viimeisessä alaluvussa.

3.1 Phonegap/Cordova

Phonegap, joka tunnetaan myös nimellä Apache Cordova, on Nitobin vuonna 2008 kehittämä, avoimeen lähdekoodiin perustuva ohjelmistokehys. Adobe osti Nitobin itselleen vuonna 2011, ja Phonegapin koodipohja lahjoitettiin Apache Software Foundationille projektiin nimeltä Cordova. Phonegap on jakeluversio Cordovasta, mutta vielä tällä hetkellä niillä ei ole toiminnallisia eroavaisuuksia.

Phonegapin avulla sovelluskehittäjät voivat luoda sovelluksia käyttäen yleisiä web-teknologioita, kuten HTML:ää, CSS:ää ja JavaScriptiä (Tunali ja Erdogan 2015). Tuloksena syntyvät sovellukset ovat ominaisuuksiltaan hybridejä, sillä ne yhdistelevät piirteitä sekä natiiveista että web-pohjaisista sovelluksista. Sovellukset ovat käytännössä web-pohjaisia, sillä ulkoasun renderöinti tapahtuu web-näkymää hyödyntäen. Niillä on kuitenkin puhtaasti web-pohjaisiin sovelluksiin erona se, että niillä on pääsy laitteen natiiveihin ominaisuuksiin monipuolisen JavaScript -ohjelmointirajapinnan kautta (Hartmann, Stead ja DeGani 2011).

Phonegapin arkkitehtuuri koostuu kolmesta eri kerroksesta: web-sovelluksesta, web-näkymästä ja PhoneGap -lisäosista, jotka sisältävät toimintalogiikan eri alustojen natiivien komponent-

tien hyödyntämiseen. Web-sovellus on vuorovaikutuksessa web-näkymän kanssa JavaScript -rajapinnan kautta, joka taas on vuorovaikutuksessa PhoneGap lisäosien kanssa natiivien ohjelmointirajapintojen kautta (Trice 2013). Nämä lisäosat pääsevät hyödyntämään käyttöjärjestelmän ominaisuuksia, kuten kameraa, GPS:ää, Bluetoothia ja kiihtyvyysanturia.

3.2 Xamarin

Xamarin on kehitysalusta, jolla voidaan kehittää iOS-, Android-, ja Windows sovelluksia C# -ohjelmointikielellä. Xamarin on alunperin polveutunut avoimen lähdekoodin Mono projektista, joka on .NET:iin perustuva ohjelmistokehitysympäristö (Tunali ja Erdogan 2015). Microsoft ilmoitti helmikuussa 2016 ostavansa yrityksen itselleen.

Xamarinin toiminta perustuu luodun C# -koodipohjan kääntämisen halutuille kohdealustoille. Xamarin-ohjelman rakenne jaetaan kahteen osaan: ohjelman käyttöliittymään ja ytimeen, joka sisältää sovelluksen toimintalogiikan (Boushehrinejadmoradi et al. 2015). Koska Xamarin kuitenkin käyttää käyttöliittymässä alustan natiiveja elementtejä, käyttöliittymä tulee suunnitella jokaiselle kohdealustalle erikseen. Tästä huolimatta resursseja säästyy huomattavasti, sillä sovelluslogiikka voidaan pääosin uudelleenkäyttää eri alustoille. Xamariniin on kuitenkin tarjolla maksullinen työkalu nimeltä Xamarin.Forms, jonka avulla käyttöliittymä voidaan luoda yhdellä kertaa kaikille kolmelle alustalle. Sen tarjoamat komponentit ovat kuitenkin vielä rajallisia (Tunali ja Erdogan 2015).

Xamarin toimii Windows ja Mac tietokoneilla, mutta iOS sovellusten rakentamiseen ja testaamiseen vaaditaan erikseen Mac tietokone. Yleisimmin kehitysympäristönä käytetään joko yrityksen omaa Xamarin Studiota tai Microsoftin Visual Studiota (Tunali ja Erdogan 2015).

3.3 Appcelerator Titanium

Appcelerator Titanium on avoimeen lähdekoodiin perustuva kehitysalusta, jonka avulla voidaan luoda monialustaisia mobiilisovelluksia käyttäen JavaScriptiä. Sen kehitti yritys nimeltä Appcelerator, ja se esiteltiin vuonna 2008. Titaniumista piti alunperin tulla monialustaisten työpöytäsovellusten kehitystyökalu, mutta tätä varten jakaantui erikseen projekti nimeltä

TideSDK, jolloin projektin pääpaino siirtyi mobiilisovelluskehitykseen.

Titaniumin erona suurimpaan osaan muihin ohjelmistokehyksiin on se, että sen toiminta perustuu ajonaikaiseen järjestelmään. Sovellusten logiikka ja data tuotetaan javascriptillä Titaniumin ohjelmointirajapintoja hyödyntäen, jonka jälkeen ohjelmakoodi pakataan Titaniumin moottorin kanssa (Tunali ja Erdogan 2015). Käyttöliittymän toteutuksessa käytetään alustan natiiveja komponentteja (H. Heitkötter ja Majchrzak 2012). Kun ohjelma on käynnissä, Titaniumin moottori tulkitsee JavaScript koodia ja luo käyttöliittymän sekä muut tarvittavat toiminnot ajonaikaisesti tarpeiden mukaan. Titaniumilla luodut sovellukset ovat ladattavissa alustakohtaisista sovelluskaupoista.

3.4 Smartface App Studio

Smartface on melko uusi iOS ja Android sovelluksien kehittämiseen tarkoitettu ohjelmistokehys. Se käyttää ohjelmointikielenään JavaScriptiä ja toimii tulkkaamalla ohjelmakoodia ajonaikaisesti tietyn alustan natiiviksi koodiksi. Se on luotu C++ -ohjelmointikielellä, ja sen arkkitehtuuri on suunniteltu siten, että sovellusten suorituskyky säilyisi mahdollisimman korkeana (Tunali ja Erdogan 2015).

Smartface koostuu sen tarjoamasta ohjelmointiympäristöstä sekä ajonaikaisesta moottorista, joka tulkitsee JavaScript koodia. Smartfacen ohjelmointiympäristö on windowsiin pohjautuva, mutta sillä voidaan luoda ja testata myös iOS sovelluksia, eli riippuvuutta Mac-tietokoneisiin ei ole. Smartface on myöskin verrattavista ohjelmistokehyksistä ainut, joka tarjoaa käyttöliittymän kehitykseen kokonaisvaltaisen WYSIWYG (engl. What You See Is What You Get) -editorin, jonka avulla käyttöliittymää voidaan tarkastella ja muokata ilman sovelluksen ajamista (Tunali ja Erdogan 2015). Tämä ei kuitenkaan aina toimi täydellisesti, sillä ulkoasuun voi vaikuttaa monet asiat, kuten näytön resoluutio ja asetukset.

3.5 Ohjelmistokehysten vertailua

Tässä alaluvussa vertaillaan edellä käsiteltyjen ohjelmistokehyksien eroavaisuuksia monesta eri näkökulmasta. Vertailun kriteereinä ovat ohjelmistokehyksien tukemat alustat, kehi-

tysympäristöt, työkalut ja niillä tuotetun koodin uudelleenkäytettävyys. Lisäksi arvioidaan ohjelmistokehyksillä tuotettujen sovellusten ulkoasua ja tuntumaa käyttäjäkokemuksen näkökulmasta.

Phonegap/Cordova tarjoaa vähintään osittaisen tuen jopa seitsemälle eri alustalle, mikä on tässä tutkielmassa vertailluista ohjelmistokehyksistä selvästi eniten. Sen avulla kehitetettyjä sovelluksia voidaan käyttää monilla eri alustoilla, koska sovelluksen sisältö näytetään puhelimen natiivissa web-näkymässä, joka löytyy lähes jokaisesta eri käyttöjärjestelmästä. Xamarinilla ja Titaniumilla taas on tasavertaisesti tuki Androidille, Windows Phoneille ja iOS:ille (Tunali ja Erdogan 2015). Smartface tukee ainoastaan Androidia ja iOS:sia, mutta koska nämä kaksi ovat selvästi suosituimmat alustat, suurin osa potentiaalisista asiakkaista saavutetaan myös Smartfacen avulla. Se on lisäksi vertailtavista ohjelmistokehyksistä ainut, jolla pystytään kehittämään ja testaamaan iOS-sovelluksia Windows ympäristössä. Tämä on suuri etu varsinkin windows-kehittäjille, joille vartavasten Mac-tietokoneen hankinta tuo runsaan lisäkustannuksen (Tunali ja Erdogan 2015).

Sovelluskehittäjillä voi olla hyvinkin erilaiset taidot ohjelmointikielien ja -taitojen suhteen. Suurimmalla osa it-alalla työskentelevistä on kuitenkin hallinnassa HTML, CSS tai Javascript ainakin jollain tasolla. Lisäksi ne ovat helppo oppia ja käytössä kaikkialla . Vertailtavista ohjelmistokehyksistä PhoneGap/Cordova hyödyntää juurikin näitä yleisiä webteknologioita, joten kynnys sen käyttämiseen ja opetteluun ei ole korkea. Titanium ja SmatFace käyttävät myös ohjelmointikielenään JavaScriptiä, joten varsinkin web-kehitystä tunteville ne ovat hyviä valintoja (Tunali ja Erdogan 2015). Xamarilla sovellukset luodaan C# -ohjelmointikielellä, joten se on oiva valinta varsinkin niille, joilla on aiempaa kokemusta Microsoftin .Net -ympäristöstä. Lisäksi koodin automaattinen tarkastaminen ennen ohjelman rakentamista ja automaattinen täydennys toimivat C#-ohjelmissa tarkemmin XAML-tuen ansioista (Tunali ja Erdogan 2015).

Jokaisella eri ohjelmistokehyksellä tuotetulla sovelluksella on ainakin osittain samaa koodipohjaa, mutta toiset vaativat enemmän alustakohtaisia muutoksia siihen. Koska Phonegap-sovellukset käyttävät ainoastaan HTML:lää, CSS:sää ja JavaScriptiä, niillä koodiin ei tarvitse välttämättä tehdä mitään muutoksia alustojen välillä. Se on kuitenkin suositeltavaa, jotta käyttäjäkokemus säilyisi hyvänä. Xamarinissa ohjelmakoodista voi uudelleenkäyttää

suurimman osan, mutta käyttöliittymät tulee tehdä eri alustoille erikseen, ellei hyödynnä maksullista Xamarin.Forms:ia. Se on työkalu jonka avulla luotu käyttöliittymä voidaan suoraan kääntää usealle alustalle. Myös Titanium ja SmartFace vaativat alustakohtaisia muutoksia käyttöliittymään, mutta Titanium näistä kahdesta hieman enemmän (Tunali ja Erdogan 2015).

Sovelluksen käytettävyys ja käyttäjäkokemus ovat ensiluokkaisen tärkeitä ominaisuuksia, jotta ihmiset ovat halukkaita käyttämään niitä (Tunali ja Erdogan 2015). Ohjelmistokehyksistä parhaiten nämä ominaisuudet täyttävät Xamarin, Titanium ja SmartFace, sillä niillä kehitetyt sovellukset ovat täysin natiiveja. Eri alustoille ja resoluutioille erikseen modifoidut käyttöliittymät saavat ne tuntumaan aivan laitteen natiivilla kehitysympäristöllä tuotetuilta sovelluksilta. PhoneGap/Cordova on tässä muita selvästi huonommassa asemassa, sillä usein ainut natiivi elementti näillä tuotetuissa sovelluksissa on web-näkymä, jossa web-pohjainen sisältö näytetään. Täten responsiivisten ja näyttävien sovellusten toteuttaminen on vaikeampaa, ellei käytä siihen runsaasti aikaa.

Kehittäjille on suuri helpotus jos kehitystyökaluilla on suuri käyttäjäkunta, jotka keskustelevat ja auttavat toisiaan ongelmatilanteissa. Tämä säästää paljon aikaa ja vaivaa, sillä monet kehittäjät törmäävät samoihin ongelmiin ja ratkaisua ongelmaan ei aina välttämättä tarvitse löytää itse. PhoneGap/Cordova, Xamarin, ja Titanium ovat kaikki hyviä vaihtoehtoja tämän kriteerin suhteen. SmartFace on näistä ohjelmistokehyksistä uusin vaihtoehto, joten se ei ole ainakaan vielä saavuttanut suurempaa suosiota (Tunali ja Erdogan 2015).

4 Sopivan kehitystavan valitseminen

Oikeanlaisen kehitystavan valitseminen monialustaiselle mobiilisovellukselle voi olla haastavaa, sillä lähestymistapoja on useita erilaisia, ja muutokset tällä saralla voivat olla hyvinkin nopeita. Pääasiassa sopiva kehitystapa riippuu sovelluksen vaatimuksista, esimerkiksi siitä, mitä sen avulla tulee voida tehdä, tai mitä laitteen ominaisuuksia sen tulee päästä hyödyntämään.

Mobiilisovelluksia on tyypeiltään laidasta laitaan, mutta kaikilla niillä on joitain piirteitä, joiden avulla ne voidaan luokitella neljään eri pääkategoriaan: palvelinpainotteisiin-, sensori/IO pohjaisiin-, itsenäisiin- ja asiakas-palvelin -sovelluksiin. Jotta sovelluskehitykselle saataisiin paras lopputulos niin ajankäytön kuin kustannustenkin suhteen, on mietittävä tarkkaan mitä menetelmää prosessissa käytetään.

Tässä luvussa käydään läpi millainen kehitystapa tulisi valita kunkin sovellustyypin kohdalla ja miksi näin tulisi tehdä. Sen lisäksi tarkastelua heijastetaan tietyissä tapauksissa myös siihen, mitä keinoja ei olisi mielekästä valita kehitykseen. Natiiveilla sovelluksilla viitataan tässä luvussa pääosin tulkattuihin ja käännettyihin sovelluksiin.

4.1 Palvelinpainotteiset sovellukset

Palvelinpainotteisten sovellusten koko toimintalogiikka ja sovelluksen tarvitsema data sijaitsee palvelimella, ja itse asiakkaan puoleisen sovelluksen osan tehtävänä on vastaanottaa informaatiota (Raj ja Tolety 2012). Tämä saatu informaatio näytetään ohjelman ohjeiden mukaisesti käyttöliittymässä, jonka kautta asiakas voi olla vuorovaikutuksessa palvelimen kanssa. Tällaisten sovellusten tapauksessa web-pohjainen kehitystapa olisi ideaali, sillä selainten ollessa pääosin standardeja käyttöliittymä tulisi tehdä vain kertaalleen, responsiivisia modifiointeja lukuunottamatta. Muutenkin koko koodipohja tulee luoda vain kerran, ja tarvittavat muutokset ohjelmaan tehdään ainoastaan palvelimelle (Raj ja Tolety 2012). Web-pohjaisilla sovelluksilla on kuitenkin tietyt rajoitteensa, ja jos kehittäjät haluavat laajentaa sovelluksensa toiminnallisuutta, niin se voi vaatia jopa uudelleenkehitystä eri mentelmää hyödyntäen. Tällainen voisi tapahtua esimerkiksi silloin, kun sovelluksen tarvitsee käyttää mobiililaitteen

sellaisia natiiveja ominaisuuksia, joihin selainten ohjelmointirajapintojen kautta ei päästä käsiksi. Koska sovelluksen käyttämä data ja toimintalogiikka sijaitsee palvelimella, natiivin sovelluksen valitseminen tähän tarkoitukseen ei ole mielekästä.

4.2 Sensori/IO -pohjaiset sovellukset

Sensori/IO -pohjaiset sovellukset hyödyntävät nimensä mukaan pääosin laitteiston natiiveja ominaisuuksia. Tällöin vaihtoehdoista voidaan sulkea pois ainakin web-pohjainen kehitystapa, sillä pääsy mobiililaitteen laitteistoon on rajoitettua ja sen hyödyntäminen on hitaampaa kuin natiiveilla sovelluksilla. Tällaisissa sovelluksissa on tärkeä ottaa huomioon, onko sen tarkoituksena toimia palvelimen välityksellä, vai painottuvatko toiminnot laitteeseen itseensä. Jos sovelluksen tulee olla vuorovaikutuksessa palvelimen kanssa, parhaana vaihtoehtona toimisi hybridi kehitystapa, sillä se mahdollistaa pääsyn laitteen natiiveihin ominaisuuksiin, mutta on muuten toiminnaltaan web-pohjainen (Raj ja Tolety 2012).

Jos sovelluksen toiminta ei painotu palvelimelle, parhaana vaihtoehtona voidaan pitää käännettyä sovellusta. Ne ovat natiiveja ja valmiiksi konekielelle käännettyjä, joten niiden nopeus, tuntuma, ja pääsy laitteistoon ovat erinomaisia. Toiseksi parhaana voidaan pitää tulkattuja sovelluksia, sillä nekin omaavat natiivien sovellusten tuomia hyötyjä. Ne ovat kuitenkin tulkkaamisen takia suorituskyvyltään hieman hitaampia kuin käännetyt sovellukset (Raj ja Tolety 2012).

Web-pohjaisilla sovelluksilla on rajatuin pääsy alustan ominaisuuksiin, joten useimmissa tapauksissa se olisi huonoin vaihtoehto tähän kategoriaan. Kuitenkin, jos selaimet tukevat sovelluksen toimintaan tarvittavia laitteen ominaisuuksia, niin nekin ovat toimiva vaihtoehto progressiivisten web-sovellusten muodossa.

4.3 Itsenäiset sovellukset

Itsenäisillä sovelluksilla tarkoitetaan tässä luvussa laitteeseen asennettavia sovelluksia, joiden avulla voidaan tuottaa ja prosessoida dataa laitteen sisäisesti. Lisäyksenä tähän voidaan luokitella myös asennetut sovellukset, joissa kaikki toiminta tapahtuu laitteen resursseja

käyttäen, mutta siinä näytettävää dataa haetaan palvelimelta. Pääpaino näissä siis keskittyy toiminnan osalta laitteeseen itseensä, joten tähänkin soveltuvat parhaiten käännetyt ja tulkatut sovellukset, joiden avulla natiivit ominaisuudet ovat monipuolisesti käytettävissä (Raj ja Tolety 2012). Lisäksi niiden suorittaman natiivin koodin ansiosta niiden suorituskyky on parhaimmillaan, mikä on yleensä tälläisten sovellusten yksi pääprioriteeteistä.

4.4 Asiakas-palvelin-sovellukset

Asiakas-palvelin-sovelluksien toiminta perustuu siihen, että datan prosessoinnissa vastuu on yhtälailla sekä palvelimella että laitteella itsellään. Tällöin paras vaihtoehto on hybridi kehitystapa, sillä se nimenomaan hyödyntää molempien, web-pohjaisten ja natiivien sovelluksien etuja. Suurin osa toiminnasta tapahtuu palvelimen kautta, mutta samalla pääsy laitteen natiiveihin komponentteihin säilytetään (Raj ja Tolety 2012).

5 Yhteenveto

Mobiilisovelluskehittäjille on tärkeää, että heidän tuotteensa ovat mahdollisimman suuren käyttäjäkunnan saavutettavissa. Tämä onnistuu helpoiten siten, että sovellus on saatavilla usealle eri alustalle. Alustojen natiivit kehitysympäristöt ovat hyvä vaihtoehto sovelluskehitykseen, mutta tällöin sovellus joudutaan luomaan jokaiselle alustalle erikseen. On kuitenkin olemassa myös ratkaisuja, joiden avulla sovellukset saadaan vähemmällä työmäärällä useammalle alustalle yhdellä kertaa. Tämän tutkielman tarkoituksena oli selvittää mitä erilaisia menetelmiä monialustaisten mobiilosovellusten kehittämiseen on ja vertailla näitä kehitystapoja toisiinsa. Tämän lisäksi vertailun kohteena oli neljä erilaista ohjelmistokehystä, jotka hyödyntävät toiminnassaan näitä erilaisia menetelmiä.

Mobiilisovellukset voidaan luokitella natiiveihin, web-pohjaisiin, hybrideihin, käännettyihin ja tulkattuihin sovelluksiin. Web-pohjaiset sovellukset toimivat täysin selaimen välityksellä, joten niitä päästään käyttämään URL:ien kautta. Niiden etuna on sovellusten kehittämisessä käytettävien teknologioiden helppous sekä se, että kaikki muutokset tulee tehdä vain yhteen ja samaan ohjelmakoodiin palvelimelle. Laitteen kaikkia natiiveja komponentteja ei kuitenkaan päästä hyödyntämään selaimen kautta, mikä voi tuoda omat rajoitteensa sovelluskehitykseen. Lisäksi haastena on sovellusten markkinointi, sillä niitä ei voida ladata sovelluskaupoista.

Hybridit sovellukset hyödyntävät toiminnassaan web-teknologioita, mutta ne ovat web-pohjaisista eroten ladattavissa erillisinä sovelluksina mobiililaitteisiin. Tällöin koko sovellus perustuu yhteen koodipohjaan ja laitteen ominaisuuksia päästään käyttämään monipuolisesti JavaScript -ohjelmointirajapinnan kautta. Koska sovelluksen sisältö näytetään kuitenkin erillisessä web-näkymässä, niin hybridien sovellusten suorituskyky ja käyttöliittymätoteutus ei yllä natiivien sovellusten tasolle.

Tulkatut sovellukset perustuvat ohjelmakoodin ajonaikaiseen tulkkaukseen erillisen ohjelmistotulkin avulla. Sovelluksen toimintalogiikka tulee kirjoittaa kerran, mutta käyttöliittymään tulee tehdä muutoksia alustoittain, sillä tulkatut sovellukset hyödyntävät niissä laitteen natiiveja käyttöliittymäkomponentteja. Tulkatut sovellukset ovat suorituskyvyltään tehokkai-

ta, vaikka puhtaasti natiivien sovellusten tasolle ei kuitenkaan päästä ajonaikaisen tulkkaamisen johdosta. Haasteena kehittäjille tämä mobiilisovellustyyppi tuo sen, että kehitystyö riippuu suuresti käytettävästä ohjelmistokehyksestä ja sen tarjoamista ominaisuuksista.

Käännetyt sovellukset ohjelmoidaan tietyllä ohjelmointikielellä ja ohjelmakoodi käännetään natiiveiksi sovelluksiksi eri alustoille. Sovellukset ovat tällöin puhtaasti natiiveja, eli suorituskyky ja laitteiston hyödyntämismahdollisuudet ovat erinomaisia. Alustakohtaisia muutoksia joudutaan kuitenkin tekemään käyttöliittymän ja mahdollisesti myös sovelluslogiikan suhteen.

Tutkielmassa oli vertailussa neljä eri ohjelmistokehystä: Phonegap/Cordova, Appcelerator Titanium, Xamarin ja Smartface App Studio. Phonegap/Cordova perustuu hybriditekniikkaan, Xamarin sovelluksen kääntämiseen, ja Titanium sekä Smartface ohjelman ajonaikaiseen tulkkaamiseen. Ohjelmistokehyksien vertailukriteereinä olivat niiden tukemien alustojen määrä, kehittämisessä hyödynnettävät tekniikat, ohjelmakoodin uudelleenkäytettävyys, tuki ja käyttäjäkunta, sekä niillä kehitettyjen sovellusten tarjoama käyttäjäkokemus.

Phonegap/Cordova kattaa vähintään osittaisen tuen jopa seitsemälle eri alustalle, kun taas Smartfacella ja Titaniumilla on tuki vain Androidille ja iOS:ille. Näiden ohjelmistokehysten välissä on Xamarin, joka tukee Androidin ja IOS:in lisäksi Windows Phonea. PhoneGap/Cordova, Titanium ja Smartface soveltuvat hyvin varsinkin web-kehittämiseen perehtyneille, sillä ohjelmointikielenä toimii JavaScript, joka on nopea oppia ja yksi maailman käytetyimmistä ohjelmointikielistä. Xamarin perustuu C# -ohjelmointikieleen, joten se on hyvä valinta varsinkin niille, joilla on kokemusta Microsoftin .Net -ympäristöstä.

Sovellusten käyttäjäkokemukseen vaikuttavat suurelta osin sovellusten suorituskyky ja käyttöliittymätoteutus. Kaikki muut ohjelmistokehykset hyödyntävät käyttöliittymissään alustakohtaisia käyttöliittymäkomponentteja paitsi Phonegap/Cordova, joten se ei yllä tällä saralla muiden tasolle. Lisäksi sen hyödyntämän hybriditekniikan takia Phonegap/Cordova -sovellusten suorituskyky on muita alhaisempi. Xamarinilla tuotettujen sovellusten suorituskyky on muita vaihtoehtoja parempi, mutta Titanium ja Smartface yltävät lähes tälle tasolle.

Jokaisella vertailussa olleella ohjelmistokehyksillä, Smartfacea lukuunottamatta, on huomattavan laaja käyttäjäkunta, joten ongelmiin voi etsiä helposti ratkaisuja erilaisten palstojen

kautta. Lisäksi ongelmiin voi kysyä apua, ja vastauksen saattaa saada hyvinkin nopeasti. Smartface on melko uusi ohjelmistokehys, joten sen suosio voi vielä kasvaa vuosien varrella.

Viimeisessä kappaleessa tarkasteltiin, mikä monialustaisista kehitystavoista olisi paras vaihtoehto erityyppisille mobiilisovelluksille. Sovellukset oli jaettu neljään eri kategoriaan: palvelinpainotteisiin, sensori/IO -pohjaisiin, itsenäisiin sekä asiakas-palvelin sovelluksiin. Palvelinpainotteisilla sovelluksille niiden data ja toimintalogiikka sijaitsee palvelimella, joten tähän parhaana ratkaisuna voidaan pitää web-pohjaisia sovelluksia. Sovellusta päästään siis käyttämään selaimen välityksellä, ja sovellus pohjautuu täysin yhteen koodipohjaan.

Sensori/IO -pohjaisten sovellusten pääprioriteettinä on tarjota tehokas ja monipuolinen pääsy laitteiston ominaisuuksiin, joten parhaiten tähän kategoriaan soveltuvat käännetyt tai tulkatut sovellukset. Itsenäisten sovellusten toiminta perustuu laitteiston omien ominaisuuksien hyödyntämiseen, joten myös tähän sopivat parhaiten käännetyt tai tulkatut sovellukset.

Asiakas-palvelin sovelluksilla sekä palvelimella että asiakkaalla on tärkeä rooli sovelluksen toiminnassa, joten parhaana vaihtoehtona voidaan pitää hybridisovellusta, joka yhdistää natiivien ja web-pohjaisten sovellusten tarjoamia hyötyjä. Tällöin web-pohjautuvaa sovellusta voidaan käyttää sujuvasti samalla, kun laitteiston ominaisuuksia päästään hyödyntämään monipuolisesti.

Suuri osa mobiilisovelluskehittäjistä valitsee vielä ennemmin natiivin, kuin monialustaisen kehittämistavan. Tämä johtuu siitä, että monialustakehitys tuo mukanaan tiettyjä rajoitteita ja haasteita, jotka voivat poissulkea tämän vaihtoehdon. Aihe vaatiikin lisää tutkimista ja kehittämistä, jotta siitä voisi tulla entistäkin vakiintuneempi valinta kehittäjille. Muutokset tällä saralla ovat hyvin nopeita, joten muutaman vuoden päästä monialustaisen kehittämisen tilanne voi olla aivan toinen.

Lähteet

Bar, Adam. 2017. “What Web Can Do Today”. https://whatwebcando.today/.

Boushehrinejadmoradi, N., V. Ganapathy, S. Nagarakatte ja L. Iftode. 2015. “Testing Cross-Platform Mobile App Development Frameworks (T)”. Teoksessa *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE),* 441–451. doi:10.1109/ASE.2015.21.

Charkaoui, S., Z. Adraoui ja E. H. Benlahmar. 2014a. “Cross-platform mobile development approaches”. Teoksessa *2014 Third IEEE International Colloquium in Information Science and Technology (CIST),* 188–191. doi:10.1109/CIST.2014.7016616.

———. 2014b. “Cross-platform mobile development approaches”. Teoksessa *2014 Third IEEE International Colloquium in Information Science and Technology (CIST),* 188–191. doi:10.1109/CIST.2014.7016616.

H. Heitkötter, Sebastian Hanschke, ja T. A. Majchrzak. 2012. “Evaluating Cross-Platform Development Approaches for Mobile Applications”. http://www3.nd.edu/~cpoellab/teaching/cse40814/crossplatform.pdf.

Hartmann, G: G. Stead ja A. DeGani. 2011. “Cross-platform mobile development”. Teoksessa *a Department of the Navy Grant N62909-11-1-1032.* https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf.

El-Kassas, Wafaa S., Bassem A. Abdullah, Ahmed H. Yousef ja Ayman M. Wahba. 2015. “Taxonomy of Cross-Platform Mobile Applications Development Approaches”. *Ain Shams Engineering Journal.* ISSN: 2090-4479. doi:http://dx.doi.org/10.1016/j.asej.2015.08.004. http://www.sciencedirect.com/science/article/pii/S2090447915001276.

Latif, M., Y. Lakhrissi, E. H. Nfaoui ja N. Es-Sbai. 2016. “Cross platform approach for mobile application development: A survey”. Teoksessa *2016 International Conference on Information Technology for Organizations Development (IT4OD),* 1–5. doi:10.1109/IT4OD.2016.7479278.

LePage, Pete. 2017. “Your First Progressive Web App”. https://developers.google.com/web/fundamentals/getting-started/codelabs/your-first-pwapp/.

Netmarketshare. 2017. “Mobile/Tablet Operating System Market Share”. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1.

Ohrt, J., ja V. Turau. 2012. “Cross-Platform Development Tools for Smartphone Applications”. *Computer* 45, numero 9 (): 72–79. ISSN: 0018-9162. doi:10.1109/MC.2012.121.

Raj, C. P. 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. doi:10.1109/INDCON.2012.6420693.

Taneja, K., H. Taneja ja R. K. Bhullar. 2016. “Cross-platform application development for smartphones: Approaches and implications”. Teoksessa *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom),* 1752–1758.

Trice, Andrew. 2013. “PhoneGap Explained Visually” (). http://www3.nd.edu/~cpoellab/teaching/cse40814/crossplatform.pdf.

Tunali, V., ja S. Z. Erdogan. 2015. “COMPARISON OF POPULAR CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT TOOLS”. Teoksessa *2. Ulusal Yönetim Bilişim Sistemleri Kongresi (YBS2015).* https://www.academia.edu/16743609/Comparison_of_Popular_Cross-Platform_Mobile_Application_Development_Tools.