Sampo Ruohonen

Mobiilimonialustakehitysympäristöt ja suorituskyky

Tietotekniikan kandidaatintutkielma

5. maaliskuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Sampo Ruohonen

Yhteystiedot: saniosru@student.jyu.fi

Ohjaaja: Timo Tiihonen

Työn nimi: Mobiilimonialustakehitysympäristöt ja suorituskyky

Title in English: Mobile multiplatform development approaches and their performance

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 35+0

Tiivistelmä: Natiivi mobiilikehitys vaatii taitotietoa kunkin mobiilialustan työkaluista. Monialustakehityksellä voidaan kehittää sovellus monelle alustalle pienemmällä työmäärällä. Tässä tutkielmassa vertaillaan monialustakehitysteknologioiden ja -tapojen eroja. Web-sovellusten vahvuuksia ovat päivitettävyys, vähäinen tallennustilan käyttö käyttäjän laitteella ja kehityksen helppous web-kehittäjille. Ne ovat kuitenkin riippuvaisia verkkoyhteydestä. Kääntämiseen perustuva Flutter on lähes natiivin nopeutensa sekä natiivin tunnun ansiosta mielenkiintoinen vaihtoehto.

Avainsanat: kandidaatintutkielmat, monialustakehitys, mobiili, web-sovellukset, hybridisovellukset, ajoympäristösovellukset, tulkatut sovellukset

Abstract: Native mobile development requires knowhow about the tools of each mobile platform. Cross-platform development allows developing an app to multiple platforms with a smaller amount of work. This thesis compares the differences in cross-platform development technologies and styles. The advantages of web applications are the ease of updating, their need of little space on the user's device and simplicity of development for the web-developer. However, they require an internet connection. Compilation-based Flutter is an interesting option based on its close to native speed and native feel.

Keywords: Bachelor's Theses, cross-platform development, mobile, web applications, hybrid apps, runtime apps, interpreted apps

Taulukot

Sisällys

1 Johdanto

Mobiililaitteet ovat tulleet kaiken kansan käyttöön. Viime vuosina mobiililaitteiden kaksi suurinta käyttöjärjestelmää ovat olleet Android, jonka markkinaosuus on 74,1 %, ja iOS, 24,8 %. Muiden markkinaosuus on ollut pienempi: joulukuussa 2019 alle 1,5 % (Statista 2019). Natiivisovellusten kehitys eri käyttöjärjestelmille vaatii kaikille käyttöjärjestelmille kokonaan erikseen kehitettävän sovelluksen. On kuitenkin olemassa myös monialustaisia kehitysympäristöjä, jotka on luotu helpottamaan sovellusten julkaisua monelle alustalle. Nämä monialustaiset kehitysympäristöt voivat siis lyhentää aikaa saada sovellus markkinoille ja vähentää kehityskustannuksia. Tässä kirjallisuuskartoituksessa vertaillaan näistä monialustakehitysympäristöistä tehtyjen tutkimusten tuloksia ja eri monialustaisten kehitysympäristöjen ja natiiviympäristöjen vahvuuksia ja heikkouksia.

Tutkimuksessa kartoitetaan monialustaisten mobiilikehitysympäristöjen ja natiivisovellusten eroja sekä sovelluskehityksen että valmiiden sovellusten näkökulmasta. Tutkimuksen tarkoituksena on siis muodostaa lukijalle kuva mobiilikehitysympäristöistä ja näin auttaa tätä tekemään valinta, minkä ympäristön käyttöön päätyy vaatimusmäärittelyn perusteella. Tutkielman toisessa luvussa esitellään eri monialustakehitysympäristöjen tyyppejä ja niiden vahvuuksia ja heikkouksia, sekä annetaan esimerkkejä eri kategorioihin kuuluvista teknologioista. Kolmannessa ja neljännessä luvussa hyödynnetään Heitkötterin ym. (2013) luomaa mittaristoa sovelluskehysten luokitteluun. Kolmannessa luvussa vertaillaan sovellusten kehittämisnäkökulmaa kehittämisen helppoudesta ylläpidettävyyteen ja jatkokehitykseen. Neljännessä luvussa käsitellään valmiiden sovellusten laadullisia kriteereitä arvioiden pitkän aikavälin käyttökelpoisuutta eri ympäristöille, virrankulutusta, tallennustilan käyttöä sekä sovellusten nopeutta. Viidennessä luvussa koostetaan tässä tutkimuksessa saavutetut tulokset.

2 Sovelluskehityksen työkalut

Biørn-Hansen ym. (2020) ovat jakaneet mobiilisovelluskehitystyökalut viiteen monialustaiseen kategoriaan sekä natiivikehitykseen. Monialustaiset kategoriat ovat web- ja hybridisovellukset, ajoympäristöön perustuvat ja tulkatut, käännetyt sekä mallipohjaiset sovelluskehitystyökalut. Tässä luvussa esitellään näiden mobiilisovelluskehitystyökalujen tyyppejä ja kerrotaan niiden eroista. Tätä jaottelua pidetään mukana myöhemmin tutkielmassa, sillä kehitystyyleillä on yhdestä teknologiasta useampiin kategorian edustajiin yleistyviä ominaisuuksia.

2.1 Natiivimobiilisovelluskehitysympäristöt

Kaikille mobiilikäyttöjärjestelmille on oma ohjelmistokehityspakettinsa (engl. *SDK, software development kit*). Tällä ohjelmistokehityspaketilla voidaan jokaiselle käyttöjärjestelmälle kehittää natiivisovelluksia. Nämä sovellukset kehitetään kunkin käyttöjärjestelmän vaatimalla ohjelmointikielellä. Natiivisovellusta kehitettäessä voidaan käyttää suoraan käyttöjärjestelmän tarjoamia ohjelmointirajapintoja (engl. *API, application programming interface*) mobiililaitteen eri toiminnallisuuksille, kuten GPS-sensorille. Laadukas natiivisovellus on suorituskyvyltään paras mahdollinen kyseiselle käyttöjärjestelmälle. Natiivisovellus myös näkyy käyttäjälle tutun näköisenä, natiivin tuntuisena, sovelluksena. Natiivisovellus kuitenkin toimii vain tällä yhdellä käyttöjärjestelmällä. Natiivimobiilisovelluskehitysympäristö on iOS:lle Xcode-ympäristö ja ohjelmointikielet Swift tai Objective-C. Androidilla ympäristö on Android Studio, ja sillä käytettäviä kieliä ovat Java, Kotlin tai C++ (Grønli ym. 2014). Natiiviohjelmistokehityspakettia käyttäen päästään yleisesti parhaaseen valmiiseen sovellukseen. Tämä tarkoittaa kullekin käyttöjärjetelmälle parasta suorituskykyä ja parasta käyttäjäkokemusta. Näillä sovelluksilla pystytään hyödyntämään käyttöjärjestelmän tarjoamia ominaisuuksia täysin (Ebone ym. 2018). Koska eri käyttöjärjestelmillä on omat natiivit kielensä ja ympäristönsä, on käytännössä mahdotonta uudelleenkäyttää koodia eri käyttöjärjestelmille kehitetyissä sovelluksissa. Tästä syystä monelle alustalle kehitettäessä sovelluskehityksen työmäärä on suurin, sillä eri alustojen sovellukset täytyy kehittää lähes kokonaan alusta (Grønli ym. 2014).

2.2 Web-sovellukset

Web-pohjaiset sovellukset toimivat mobiililaitteen selaimessa kuten muutkin web-sivut. Ne on kehitetty JavaScriptilla, CSS:llä ja HTML:llä. Web-pohjaiset sovellukset käyttävät ajonaikaisena ympäristönä mobiililaitteen selainta, joten web-sovellukset toimivat kaikilla mobiilikäyttöjärjestelmillä ilman erillisen sovelluksen latausta ja asennusta laitteen sovelluskaupasta. Toisaalta web-sovelluksen avaaminen mobiililaitteen internetselaimen kautta ei ole yhtä helppoa kuin laitteeseen ladatun sovelluksen avaaminen suoraan laitteen sovellusvalikosta tai aloitusnäytöltä.

Web-sovellukset eivät myöskään välttämättä pääse käyttämään kaikkia mobiililaitteen rajapintoja kuten GPS:ää ja kameraa; ne rajoittuvat mobiililaitteen selaimen tarjoamiin rajapintoihin. Haittapuolena on myös pakollinen internetyhteys sovellusten käyttämiseksi, joka rajoittaa sovellusten käyttöä paikoissa joissa internetyhteys on heikko tai yhteyttä ei ole. Lisäksi mikäli sivu ei ole selaimen välimuistissa, kaikki lataukset sivulla joudutaan tekemään verkkoyhteyden kautta, jolloin sovelluksen nopeus on riippuvainen verkon nopeudesta. JavaScriptin avulla ei päästä suoraan käsiksi natiiviohjelmointirajapintoihin. HTML5 kuitenkin mahdollistaa Web-Storagen eli 5 MB:n suuruisen paikallisen tallennustilan käytön. Web-sovelluksen ja asennettavan sovelluksen välillä on suuria käyttöfilosofian eroja (Heitkötter ym. 2013a). Web-selain voidaan sulkea missä vaiheessa tahansa, eikä selaimen tarvitse ilmoittaa sulkemisestaan web-sivulle. Web-sovellukset eivät myöskään yleensä ole natiivin tuntuisia, vaan niiden käyttöliittymä eroaa kunkin mobiilikäyttöjärjestelmän käytänteistä (Heitkötter ym. 2013a).

Suuret web-palvelut tarjoavat usein sekä web-sovellusta että asennettavaa sovellusta omasta tuotteestaan, esim. Youtube. Esimerkkejä web-sovellukseen käytetyistä JavaScript-kirjastoista ovat jQuery Mobile, Sencha Touch ja React.js (Ciman ym. 2017).

Mielenkiintoinen kehitys web-sovelluksissa on siirtymä progressiivisiin sovelluksiin. Jotta web-sovellus olisi progressiivinen, sen tulisi Osmanin (2015) kuvauksen mukaan olla

- **progressiivinen**: jokaisella selaimella käytettävissä, mutta uusia ominaisuuksia tukeva selain tuo lisäarvoa
- **responsiivinen**: toimii minkä tahansa kokoisella ja tyyppisellä laitteella

- **käytettävissä ilman internetyhteyttä**: Service Worker mahdollistaa käyttämisen ilman internetyhteyttä
- **tuore**: se toimii aina uusimmalla versiolla Service Workerin päivitysprosessin myötä
- **turvallinen**: SSL/TLS mahdollistaa tietoturvallisen käytön
- **löydettävissä**: W3C-manifestit ja Service Workerin rekisteröinnin ansiosta progressiiviset web-sovellukset löytyvät hakukoneista
- **mielenkiintoa ylläpitävä**: push-ilmoitukset kannustavat käyttämään sitä uudelleen
- **asennettavissa**: sallivat käyttäjän asentaa hyödyllisimmät sovellukset laitteensa kotinäytölle
- **linkitettävissä**: progressiivisen web-sovelluksen voi helposti jakaa verkko-osoitteen avulla.

Progressiiviset web-sovellukset lisäävät toiminnallisuuksia käyttöön sen mukaan, kuinka uudella selaimella sovellusta käytetään. Sovellus toimii pelkkänä staattisena HTML-sivuna, mutta lisää toiminnallisuutta käyttöön jos selain niitä tukee. Progressiiviseen web-sovellukseen on liitetty mobiililaitteelle tallennettu JSON-kielinen metadatadokumentti eli manifesti. Manifestin ansiosta kotinäytölle asennettu sovellus toimii enemmän kuin natiivisovellus: sovellus voi käynnistyä koko näytölle ilman navigointipalkkia ja näytön suuntaa voidaan hallita. Lisäksi Androidilla on mahdollista asettaa käynnistyskuva (engl. *splash screen*) ja navigointipalkin teemaväri (Osmani 2015). Tämän lisäksi sovelluksen sisällöstä erillinen ulkokuori (engl. *application shell*) (Osmani ym. 2015) tekee mahdolliseksi käyttää paikallista tallennustilaa säilyttämään osaa käyttöliittymästä. Taustalla pyörivä Service Worker -taustaskripti mahdollistaa push-ilmoitukset (Majchrzak ym. 2018). Service Worker on käytännössä omalla säikeellään suorittava JavaScript-moduuli, joka mahdollistaa persistenttiyden ja tapahtuma-pohjaisen (engl. *event-driven*) kommunikoinnin muun sovelluksen kanssa (Malavolta ym. 2017). Progressiivinen web-sovellus voidaan asentaa mobiililaitteen kotinäytölle, toisin kuin perinteiset web-sovellukset. Progressiivisten web-sovellusten periaatteita voidaan toteuttaa monilla web-tekniikoilla. Osmani (2015) on käyttänyt testisovelluksensa toteuttamiseen React.js-kirjastoa.

2.3 Hybridisovellukset

Web-sovelluksista voidaan myös luoda asennettevia paketteja käyttämällä web-natiivi käärettä (engl. *web-to-native wrapper*). Näitä paketteja voidaan jakaa mobiililaitteiden sovelluskaupoissa. Tällöin sovellus ei sijaitse ulkoisella palvelimella, vaan sitä suoritetaan mobiililaitteessa WebView-komponentissa. WebView on upotettava selainikkuna, joka piilottaa selaimen normaalit kontrollit kuten osoitepalkin ja kirjanmerkit (Biørn-Hansen ym. 2020). Tätä kehitystapaa kutsutaan hybridikehitykseksi. Tällöin käyttäjän ei tarvitse käyttää web-työkaluilla kehitettyä sovellusta selaimen kautta, vaan sovelluksen voi ladata mobiililaitteeseen. Kääreiden etu on myös web-sovelluksia parempi tuki mobiililaitteen ohjelmointirajapintoihin, kuten GPS-sensoriin ja bluetoothiin (Willocx ym. 2016). Hybridisovelluksia voidaan jakaa alustojen omissa sovelluskaupoissa, asentaa laitteeseen ja käyttää ilman verkkoyhteyttä. Sovelluksen ulkoasu on kuitenkin luotu JavaScriptilla, CSS:llä ja HTML5:llä, joten hybridisovelluksetkin näyttävät enemmän web-sivuilta kuin mobiililaitteen sovellukselta. Esimerkkejä hybridisovelluskirjastoista ovat Apache Cordova (jota aikaisemmin kutsuttiin PhoneGapiksi), Framework7, Ionic, Onsen UI ja Quasar Framework.

2.4 Ajoympäristöön perustuvat ja tulkatut sovellukset

Tulkatut (engl. *interpreted*) sovellukset toimivat ajoympäristön (engl. *runtime*) avulla. Näiden sovellusten käyttöliittymä toimii kunkin alustan natiivikomponenttien avulla, eli ajoympäristösovellukset eivät käytä WebView-komponenttia hybridisovellusten tapaan. Sama liiketoimintalogiikan koodi toimii eri alustoilla, sillä ajonaikainen tulkki piilottaa eri alustojen erot sovelluslogiikalta (Willocx ym. 2016). Tyypillisesti ajoympäristösovelluksissa mobiililaitteen natiivikoodin toiminnallisuudet saadaan käyttöön liitännäisten siltauksen (engl. *bridging*) avulla (Biørn-Hansen ym. 2020). Ajoympäristösovellusten huono puoli ovat monet kehykset, joilla kaikilla on omat toistensa kanssa yhteensopimattomat liitännäiset. Esimerkkejä ajoympäristösovelluskehyksistä ovat JavaScriptia käyttävät Appcelerator Titanium, Facebookin React Native ja Progressin NativeScript, Microsoftin C#:ia käyttävä Xamarin ja omia merkintäkieliään käyttävät Qt ja Solar2d. React Nativessa, NativeScriptissa ja Titaniumissa JavaScript tulkataan alustan natiivikomponenteiksi ajonaikaisesti mobiililaitteella olevalla JavaScript-moottorilla, kuten JavaScriptCorella tai V8:lla. Xamarinilla saa tuotettua natiivi-

sovelluksia Androidille, iOS:lle ja macOS:lle, ja lisäksi samaa C#-koodia voidaan käyttää Windows-sovellusten tuottamiseen (Delıa ym. 2017). Solar2D, entiseltä nimeltään Corona, mahdollistaa sovellusten kehittämisen Androidille ja iOS:lle sekä lisäksi macOS:lle, Windowsille, Linuxille, tvOS:lle, Kindle Firelle ja Android TV:lle. Solar2D-sovellukset kirjoitetaan Lua-kielellä ja erillistä uudelleenkirjoitusta eri alustoille ei tarvita (Delıa ym. 2017). Titaniumissa käyttöliittymä kirjoitetaan JavaScriptilla käyttäen Titanium API:ta. Koodi pakataan ja edelleen luetaan käyttäen Titanium Mobilen moottoria. Titaniumin omistaja Axway on ilmoittanut lopettavansa Titaniumin kehittämisen ja siirtävänsä kehyksen avoimeksi lähdekoodiksi maaliskuussa 2022.

2.5 Kääntämiseen perustuvat monialustasovellukset

Ristiinkäännetyissä (engl. *cross-compilation* tai *trans-compilation*) tarkoitus on uudelleenkäyttää natiivisovellus tuottamalla syötesovelluksesta kohdesovellus tavukoodin tai korkeamman tason ohjelmointikielen avulla (Biørn-Hansen ym. 2020). Kääntämiseen perustuvat menetelmät tyypillisesti pystyvät kääntämään liiketoimintalogiikan koodin, mutta täyden sovelluksen saamiseksi joudutaan lisäämään toiminnallisuutta. Esimerkkejä kääntämiseen perustuvista menetelmistä ovat Googlen J2ObjC ja saman yrityksen Flutter. J2ObjC toimii nimensä mukaisesti muuttamalla Androidin Java-koodin Objective-C koodiksi iOS:lle. J2ObjC siis mahdollistaa Javalla kirjoitetun sovelluslogiikan koodin käyttämisen myös Androidilla. Se ei kuitenkaan tarjoa mitään monialustaisia käyttöliittymän suunnittelutyökaluja (Google 2020). Flutter eroaa tulkattujen sovellusten kategoriasta siinä, että Flutter ei käytä natiivikomponentteja käyttöliittymään, vaan kaiken piirtämisen tekee Skia Graphics Engine. Tämä moottori pystyy tekemään natiiveja käyttöliittymäkomponentteja Skia Canvasilla. Dart-koodi ja Flutter-ohjelmistokehityspaketti kännetään ennen ajoa (engl. *AOT, ahead-of-time compilation*) natiiveiksi ARM ja x86 -kirjastoiksi, eli tulkkia ei tarvita ajon aikana (Biørn-Hansen ym. 2020).

2.6 Mallipohjainen sovelluskehitys

Mallipohjaisella sovelluskehityksellä (engl. *model-driven software development*) tarkoitetaan tekstiin tai graafiseen mallinnuskieleen pohjautuvaa sovelluksen suunnittelua. Tässä tavassa kunkin alustan oma koodigeneraattori luo valmiin sovelluksen mallin pohjalta (Völter ym. 2013). Mallista luodaan alustan natiivikielen koodia, joka voidaan kääntää ja koota alustan omalla ohjelmistokehityspaketilla kullekin käyttöjärjestelmälle. Valmiiden sovellusten olisi siis mahdollista ideaalitilanteessa toimia kuten natiivisovellusten kullekin alustalle (Biørn-Hansen ym. 2020). Esimerkkejä kaupallisista mallipohjaisista työkaluista ovat WebRatio Mobile, BiznessApps ja Bubble. Akateemisia vaihtoehtoja ovat MD_2 (Heitkötter ym. 2013b) ja MAML (Rieger ym. 2018). Mallipohjaisia työkaluja on vertailtu muihin paradigmoihin ainoastaan Biørn-Hansen ym.in (2020) tutkimuksessa, joten mallipohjaisia työkaluja ei tulla tässä tutkimuksessa käsittelememään.

3 Sovellusten kehitys monialustaympäristössä

Tärkeä osa monialustakehitysympäristön toimivuutta on kehitystyön tehokkuus. Mitä pienemmällä työmäärällä kehittäjät pystyvät luomaan sovellukset Androidille ja iOS:lle, sitä nopeammin sovellus saadaan markkinoille, tai vaihtoehtoisesti sovellukseen voidaan samassa kehitysajassa lisätä enemmän käyttökelpoisia ominaisuuksia. Otettaessa uutta teknologiaa käyttöön suuri osa työajasta kuluu opetteluun. Oppimiskäyrän jyrkkyyteen vaikuttavat opittavien teknologioiden määrä sekä dokumentaation laatu.

Tämän luvun jäsennys perustuu Heitkötterin ym. (2013) laatimaan sovelluskehittämisen näkökulmaa koskevaan kriteeristöön. He ovat luoneet tutkimusta varten prototyyppisovellukset, joilla he ovat arvioineet monialustakehitysympäristöjä mittaristonsa pohjalta. Sovellukset ovat resurssienhallintasovelluksia, ja ne on kehitetty erikseen neljällä sovelluskehitystyylillä: web-sovelluksena, hybridisovelluksena PhoneGapillä, ajoympäristösovelluksena Titanium Mobilella sekä natiivisovelluksina Androidille ja iOS:lle.

3.1 Kehitysympäristö

Kehitysympäristöllä tarkoitetaan tietyn paradigman sovelluksen kehitykseen tyypillisesti liitettyjen työkalujen kypsyyttä. Tällaisia ovat ohjelmointiympäristö (engl. *IDE, integrated development environment*), sen virheenetsintätyökalut ja emulaattori sekä toiminnallisuudet, kuten automaattinen täydennys ja automaattitestaus (Heitkötter ym. 2013a).

Jotta sovellusta voi testata emulaattorin avulla, kunkin alustan ohjelmistokehityspaketti tulee asentaa. Web-sovellusten etuna on, että niitä voidaan tarkastella myös pelkästään selaimessa. Tämän hetken suosituin ohjelmointiympäristö on Visual Studio Code (StackOverflow 2021). Heitkötterin ym. (2013) mukaan web-kehitystyökalujen maturiteetti on hyvä. Natiivikehitystyökaluista on hyvä huomata, että Android Studiota voi käyttää Windowsilla, macOS:llä, Linuxilla sekä ChromeOS:llä (Google 2021c). IOS-natiivikehitystä hankaloittaa se, että Xcode on julkaistu vain macOS:lle (Apple 2021). Hybridisovelluskehysten kutsujen automaattinen täydennys ei välttämättä toimi (PhoneGap) (Heitkötter ym. 2013a).

	IDE	**Automaattinen täydennys**	**Automaatti-testaus**	**Hot Reload**
Web	Visual Studio Code	Kyllä	Kyllä	Sovellus web-selaimessa
Hybridi	Visual Studio Code	Ehkä	Kyllä	Sovellus web-selaimessa
Ajoympäristö: Titanium	Titanium Studio (Eclipse-pohjainen) VS Code ja plugin	Kyllä	Kyllä	Kyllä (Fastdev)
Kääntämiseen perustuva: Flutter	Android Studio (IntelliJ-pohjainen), VS Code ja plugin	Kyllä	Kyllä	Kyllä
Natiivi: Android	Android Studio	Kyllä	Kyllä	Kyllä
Natiivi: iOS	Xcode	Kyllä	Kyllä	Pluginilla

Taulukko 1. Kehitysympäristöjen vertailu

Vaikka kehitysympäristöissä on pieniä eroja, ovat kaikkien tutkittujen sovellusten kehityksen työkalut kypsiä. Eroja löytyy lähinnä hybridikirjastojen automaattitäydennyksen mahdollisesta puuttumisesta sekä tehtyjen muutosten testauksen nopeudesta. Web-työkaluilla kehitetty sovellus voidaan testata selaimessa heti muutosten tallennuttua palvelimelle, kun taas natiivisovellusten testaamiseen tarvitaan emulaattorin käyttöä.

3.2 Käyttöliittymäsuunnittelu

Natiivin tuntuisen sovelluksen kehittäminen web-ympäristössä on hankalaa, sillä CSS-lomakkeet määrittävät web-sovelluksen tyylin. Joitakin CSS-teemoja on luotu imitoimaan laitteiden natiivia tuntua. CSS3 mahdollistaa Heitkötter ym. (2013a) mukaan yksinkertaisen ja nopean käyttöliittymäkehityksen. Web-sovellus voidaan nopeasti ladata kohdelaitteella ilman tarvetta kääntämiselle, joten kehitys on nopeaa. Myös Adobe Dreamweaverin kaltaisten WYSIWYG-editoreja (engl. *what you see is what you get*) suunnittelutyökalujen käyttäminen on mahdollista.

Koska hybridikehys PhoneGap käyttää web-teknologioita käyttöliittymään, on käyttöliittymäsuunnittelu samalla tasolla web-sovellusten kanssa. Ajoympäristöön perustuva Titanium ei tarjoa WYSIWYG-editoria, joten käyttöliittymäsuunnitteluun on kulunut huomattavasti web-sovelluksia enemmän aikaa (Heitkötter ym. 2013a). Molemmat natiivisovelluskehitysympäristöt tarjoavat WYSIWYG-editorin (Google 2021c; Apple 2021). Heitkötter ym. (2013a) kehuvat erityisesti iOS-editorin olevan erittäin laadukas, mahdollistaen kehittäjän suunnitella suuren osan käyttöliittymästä kirjoittamatta lainkaan koodia.

Flutterille on tarjolla WYSIWYG-editori. Sen käyttöliittymäsuunnittelu tehdään deklaratiivisella tyylillä (engl. *declarative programming, React style*). Deklaratiivisessa ohjelmoinnissa ongelman ratkaisuun käytettävän algoritmin sijaan koodissa kuvaillaan haluttu lopputulos. Esimerkkejä tyylistä ovat SQL-kieli ja säännölliset lausekkeet (engl. *regular expressions*). Koska tyyli poikkeaa yleensä tutummasta imperatiivisesta tyylistä, voi Flutterin käyttöliittymäsuunnittelu olla haasteellisempaa aloittelijalle.

3.3 Sovelluskehityksen helppous

Gonsalvesin (2019) mukaan hybridisovelluksen luominen Cordovalla/Ionicilla on hankalaa, koska kehittäjän täytsyy tuntea useita teknologioita: TypeScriptia, Angularia, HTML:ää ja CSS:ää, sekä ymmärtää Ionic-käyttöliittymäelementtejä ja Cordova-liitännäisiä. Näiden teknologioiden käyttämisen avuksi laadittu dokumentaatio on pirstaloitunut moniin eri lähteisiin. Toisaalta Heitkötterin ym. (2013) mukaan web-teknologioiden dokementaatio on laadultaan yleensä korkealuokkaista. HTML-, CSS- ja JavaScript-sovelluskehitys on intuitiivista, joten Heitkötterin ym. (2013) mukaan sovelluskehitys on web-sovellusta kehittäessä helpompaa kuin muilla kehitystyyleillä. Kehittäjän vaaditaan vain normaalien web-teknologioiden tuntemusta eikä juuri muuta ohjelmistokehykseen liittyvää tietoutta. Hyvä JavaScript-ohjelmistokehys mahdollistaa lyhyen ja elegantin koodin. Esimerkiksi datan järjestäminen voi onnistua yhden avainsanan avulla, ja ohjelmistokehys tarjoaa useimmin tarvittavat toiminnallisuudet. PhoneGap-dokumentaatiota Heitkötter ym. (2013a) kehuvat selkeäksi, ja mukana on sekä yksinkertaisia että täydellisiä esimerkkejä. JQuery Mobilen dokumentaation he sanovat olevan heikompilaatuista; koodiesimerkkejä ei siinä ole.

Heitkötter ym. (2013a) kertovat ajoympäristöön perustuvan Titaniumin dokumentaation olevan laadukasta. Koodiesimerkkejä on paljon, joskaan ne eivät ole täysin kattavia. He myös kommentoivat tottumisen Titaniumin ohjelmointirajapintaan vievän ohjelmistoprojektin alussa aikaa, mikä johtuu siitä että tässä ohjelmointikehyksessä on paljon omaa koodia.

Gonsalves (2019) huomaa kääntämiseen perustuvan Flutter-sovelluksen kehittämisen opettelun olevan yksinkertaisempaa kuin Ionic/Cordova-sovelluksen. Flutter-kehittäjän täytyy opetella Dart-kieli, mutta sen lisäksi dokumentaatio on yhdellä sivulla ja kaikki tarvittavat liitännäiset yhdessä tietolähteessä (engl. *repository*).

Natiiviympäristöjen dokumentaation Heitkötter ym. (2013a) kertovat olevan ensiluokkaista ja tarjoavan useita koodiesimerkkejä. Lisäksi molemmille natiivialustoille on aloittelijaystävällisiä aloitusoppaita. Mikäli kehittäjä jo tuntee natiivin ohjelmointikielen, kehitys on nopeaa, joten riittää, että kehittäjä opettelee mobiilialustan erikoisuudet (Heitkötter ym. 2013a).

Minusta näennäinen helppous siis on osittain subjektiivinen kokemus. Heitkötter ym. (2013a) olivat selvästi jo kokeneita web-teknologioiden käytössä, joten web-kehitys onnistui heiltä

helposti. Toisaalta harjaantunut .NET-kehittäjä voisi mielestäni oppia Xamarinin C#-ympäristön helpommin kuin hänelle uudet web-teknologiat. Paljon siis riippuu kehittäjän aiemmasta kokemuksesta.

3.4 Ylläpidettävyys

Heitkötter ym. (2013a) arvioivat ylläpidettävyyttä ensisijaisesti koodirivien määrän perusteella. Esimerkkisovelluksissaan he käyttivät vähiten koodirivejä web-sovellukseen. PhoneGap-hybridisovellukset ovat koodiltaan yhteneviä web-sovellusten kanssa, jos alustan natiiviohjelmointirajapintoja ei käytetä. Pitkän aikavälin tuki JavaScriptille, HTML:lle ja CSS:lle on hyvä, ja päivityksiä ja parannuksia tekniikoihin tulee tasaisesti. JavaScript-sovelluskehyksen valinta voi kuitenkin Heitkötterin ym. (2013) mukaan olla haastellista. Yhdestä kehyksestä vaihtaminen toiseen, kun projektia on kehitetty jo yhdellä kehyksellä, voi olla hankalaa, aikaavievää ja kallista. Kun tällä hetkellä on useita suuria JavaScript-kirjastoja, ei oikean kehyksen valitseminen ole yksinkertaista.

Hybridi- ja tulkatulla kehitystavalla tehtyjen sovellusten koodiin voidaan tehdä päivityksiä ja julkaista ne ohittaen kauppapaikat Microsoft CodePushilla tai vastaavilla työkaluilla (Biørn-Hansen ym. 2020). Kääntämiseen perustuvilla kehyksillä, kuten Flutterilla, julkaistun sovelluksen päivittäminen onnistuu vain alustojen kauppapaikkojen kautta, eikä tukea vastaavaan ohittamiseen ole tulossa (Seidel 2018).

Ajoympäristöön perustuvan Titaniumin prototyyppisovelluksen koodirivien määrä oli Heitkötterin ym. (2013) mukaan suurempi kuin muiden paradigmojen sovelluksien koodirivien. He kuitenkin kehuvat sovellusten olevan hyvin jaettavissa modulaarisiin kokonaisuuksiin.

Natiivisovellusten koodirivien määrät ovat Heitkötterin ym. (2013) kummassakin prototyyppisovelluksessa suuremmat kuin muiden paradigmojen. Tämän he sanovat johtuvan olio-ohjelmointikielten luokista ja muista rakenteista. Vaikka koodia on paljon, on ylläpito helppoa, koska koodi on jaoteltu hyvin.

Web-sovellusten etuna on se, että käyttäjien sovellukset päivittyvät, kun palvelimella olevaa sovellusta päivitetään. Käyttäjän ei tarvitse ladata päivitystä sovellukseensa, eikä kehittäjän

tarvitse saada hyväksyntää kauppapaikalle päivityksen tekemiseen. Myös hybridi- ja tulkattujen sovellusten päivittämistä helpottaa kauppapaikan kiertäminen Microsoft CodePushin kaltaisilla työkaluilla. Sopivan ohjelmistoarkkitehtuurin valinta ja tähän arkkitehtuuriin taipuva kieli sallivat hyvin ylläpidettävän koodin luomisen.

3.5 Skaalautuvuus

Web-sovellusten skaautuvuus riippuu käytetystä sovelluskehyksestä, mutta pääsääntöisesti Heitkötterin ym. (2013) mukaan web-sovellukset voidaan jakaa moniin pieniin modularisaatiota tukeviin tiedostoihin. He kuitenkin kritisoivat jQuery-kirjastolla tehdyn projektin käyvän turhan monimutkaiseksi sen kasvaessa. PhoneGap-hybridisovellukselle he antavat samat arviot skaalautuvuudessa kuin web-sovellukselle.

Heitkötter ym. (2013a) arvioivat myös ajoympäristötyyppisten Titanium-sovellusten olevan helposti modularisoitavissa, mikä mahdollistaa helpon skaalautuvuuden. Lisäksi eri ikkunoita on mahdollista ajaa eri JavaScript-konteksteissa.

Natiivisovelluksissa liiketoimintalogiikan koodi ja käyttöliittymän koodi voidaan helposti eriyttää toisistaan. Myös kukin ohjelman näkymä voidaan jakaa ja kehittää erillisesti toisistaan. Tämä sekä olio-ohjelmoinnille tyypillinen luokkarakenne tekevät natiivisovelluksista helpompia skaalata kehitystiimin kesken kuin muilla paradigmoilla kehitettyjen sovellusten (Heitkötter ym. 2013a).

3.6 Mahdollisuudet jatkokehitykseen

Web-sovelluksena aloitettu projekti voidaan myöhemmin helposti muuttaa Cordovalla toimivaksi hybridisovellukseksi, mikäli tulee tarve käyttää laitteen ohjelmointirajapintoja projektin myöhemmässä vaiheessa. Myös liiketoimintalogiikan koodin uudelleenkäyttö Titanium Mobilessa on mahdollista, joten mahdollisuudet jatkokehitykseen ovat Heitkötterin ym. (Heitkötter ym. (2013a)) mukaan erinomaiset.

PhoneGap-hybridisovelluksen jakaminen web-palvelimella on mahdollista, mikäli ei ole käytetty alustakohtaisia ohjelmointirajapintoja. Tällöin sovellus on vain web-sovellus, ja hybri-

disovelluksen käyttäminen selaimessa on mahdollista. Biørn-Hansen ym. (2020) kritisoivat React Nativen ja NativeScriptin liitännäisiin pohjautuvaa integraatiota natiiveihin ominaisuuksiin siitä, että mikäli sovelluskehystä halutaan vaihtaa jossain projektin vaiheessa, täytyy käyttöliittymän ja logiikan lisäksi myös liitännäiset kehittää uudestaan. Heitkötter ym. (2013a) antavat samanlaisen arvion Titanium-sovelluksista. Ajoympäristöön perustuvan Titanium-sovelluksen liiketoimintalogiikan koodi voidaan uudelleenkäyttää eri ympäristössä, mutta koska Titanium vaatii suuren määrän omaa koodia, sen vaihtaminen muuhun on työlästä.

Natiivisovellusten koodin jakaminen eri käyttöjärjestelmän sovelluksiin ei ole mahdollista, koska eri alustoilla käytetään eri kieliä ja ohjelmointirajapintoja, joten yhdelle alustalle kehitettyä natiivisovellusta ei voi siirtää toiselle (Heitkötter ym. 2013a). Poikkeuksena tästä mainittakoon J2ObjC:n kaltainen natiivikoodin muuttaminen toiselle kielelle (Google 2020).

Parhaat jatkokehitysmahdollisuudet on mielestäni web-sovellusten JavaScript-, HTML- ja CSS -teknologioilla. Toisaalta JavaScript-kehykset vaikeuttavat kuviota, koska ne eivät ole toistensa kanssa yhteensopivia. Titaniumin kaltaisen ajoympäristö-sovelluksen liiketoimintalogiikan JavaScript-koodia voi uudelleenkäyttää esimerkiksi web-sovelluksessa, mutta toisaalta JavaScriptilla luotua käyttöliittymää ei.

3.7 Kehityksen nopeus ja hinta

Heitkötter ym. (2013a) mukaan web-sovellusten kehitystyökalut ovat laadukkaita, joten niiden debuggaus, testaus ja käyttöliittymäsuunnittelu on vienyt vähiten aikaa ja ollut kustannustehokasta. PhoneGap-hybridisovelluksen kehittäminen on tässäkin suhteessa tasoissa web-sovelluksen kanssa. Alustakohtaisten ominaisuuksien lisääminen PhoneGap-projektiin lisää sen kehitysaikaa.

Ajoympäristösovelluksen kehitys Titaniumilla on ollut aikaavievää, sillä kehitykseen tarvitaan paljon tälle ohjelmistokehykselle erityistä tietoutta. Lisäksi Titaniumilla kehitystä on hidastanut mitä näet, sitä saat -editorin puute (Heitkötter ym. 2013a). Kaikkein aikaavievintä ja kalleinta on natiivisovellusten kehittäminen kullekin alustalle erikseen. Kumpikin alusta vaatii niiden omien ohjelmointirajapintojen tuntemista (Heitkötter ym. 2013a). Tässä

kategoriassa natiivisovellukset siis selvästi häviävät monialustasovellusten kehittämiselle.

4 Laadulliset näkökulmat valmiisiin sovelluksiin

Sovelluskehitystyön tavoitteena on tuottaa valmis sovellusohjelma. Heitkötter ym. (2013a) ovat kehittäneet neljätoistakohtaisen kriteeristön monialustakehitystyökalujen evaluointiin. Kriteeristön sovelluskehitykseen liittyvät kohdat käsiteltiin luvussa 3. Tässä luvussa käsitellään valmiiseen sovellukseen liittyviä kriteerejä, joita ovat:

1. lisenssi ja hinnat
2. tuetut alustat
3. pääsy alustakohtaisiin ominaisuuksiin
4. pitkän aikavälin käyttökelpoisuus
5. sovelluksen ulkonäkö ja tuntu
6. sovelluksen nopeus
7. sovelluksen jakaminen.

Tässä luvussa keskitytään erityisesti sovellusten nopeuteen, sillä mielestäni se on tärkeä kriteeri käyttäjän arvioidessa sovelluksen laatua. Pidän myös virran- ja tallennustilan kulutusta tärkeinä laadullisina vaatimuksina erityisesti mobiililaitekontekstissa, jossa akku ja tallennustila asettavat rajoitteita. Lisäksi arvioin pitkän aikavälin käyttökelpoisuutta sovelluskehyksille, sillä tuen loppuminen ajaa sovelluskehyksen vääjäämättä vaikeuksiin tietoturva-aukkojen paljastuessa. Pääsyä alustakohtaisiin ominaisuuksiin, tuettuja alustoja, sovelluksen ulkonäköä ja tuntua sekä sovellusten jakamista sivuttiin tutkielman luvussa 2. Tutkielmassa ei käsitellä lisenssejä ja hintoja, koska ne muuttuvat jatkuvasti.

4.1 Pitkän aikavälin käyttökelpoisuus

Tässä tutkielmassa on viitattu useisiin eri monialustakehyksiin. Tällä hetkellä iOS ja Android ovat selvästi suosituimmat mobiilikäyttöjärjestelmät, eikä niille ole näköpiirissä varteenotettavia kilpailijoita. Windows Phonen suosio oli ja meni (Statista 2019). Applen ja Googlen ollessa maailman viiden suurimman yrityksen joukossa (Statista 2021) näyttää natiivisovellusten tuki olevan jatkossakin hyvää.

		Tähtiä	Käyttäjiä
Web	React	178 000	8 200 000
	Angular	78 000	2 100 000
	Vue	191 000	
Hybridi	Framework7	17 000	
	Ionic	46 000	
	Onsen UI	8 000	
	Quasar	20 000	
Ajoympäristö	React Native	100 000	800 000
	NativeScript	21 000	
Kääntäminen	Flutter	133 000	

Taulukko 2. Eri kehysten tähdet ja käyttäjät GitHubissa (Facebook 2021a; Google 2021a; You 2021; Kharlampidi 2021; Ionic 2022; Monaca 2021; Stoenescu 2021; Facebook 2021b; Progress 2021; Google 2021b)

Web-sovellusten tuki on hyvää, ja sellaisena se tulee jatkumaan. Uudet selaimet tukevat progressiivisia ominaisuuksia jo kattavasti (Santoni 2021). Suurin haaste web-sovelluksen kehityksessä on käytettävän JavaScript-kehyksen valinta. Tämän hetken suosituimpia kehyksiä ja kirjastoja ovat Facebookin React, Googlen Angular ja Evan Youn kehittämä Vue.js. JQuery Mobilen kehitys on lopetettu (jQueryMobile 2022). Sencha Touchin kehitys on myös lopetettu, ja se on sulautettu Sencha Ext Js -kehykseen, joka keskittyy datan visualisointiin (Sencha 2022).

Hybridisovelluskehyksistä vain PhoneGapin takana on suurempi yritys, mutta se on lopettanut PhoneGapin aktiivisen tuen (Adobe 2022). Hybridikehykset ovat selvästi web-kehyksiä pienempiä, mutta niistä Ionicin tuki vaikuttaa edelleen aktiiviselta ja se on hieman kilpailijoitaan suurempi (Ionic 2022).

Tulkatuista sovelluskehyksistä monessa lähteessä mainitun Appcelerator Titaniumin tuki on Axwaylta loppumassa maaliskuussa 2022. Vaikka Xamarin on Microsoftin projekti, on yhtiö siirtymässä Xamarinista kohti MAUI:ta (Ortinau 2021), joten Xamarinin valinta ei minusta uuteen projektiin vaikuta enää kovin järkevältä. Turvallisin vaihtoehto tulkatuista kehyksistä

on siis mielestäni Facebookin React Native. Progressin NativeScript on toinen mahdollinen valinta, mutta sen suosio on selvästi vähäisempää kuin React Nativen (Progress 2021). Solar2d ja Qt ovat selvästi pienempien tiimien projekteja ja siten epävarmemmalla pohjalla (QtGroup 2021; Shcherban 2021).

Googlen vielä suhteellisen tuore kääntämiseen perustuva Flutter vaikuttaa minusta olevan yksi turvallisimmista vaihtoehdoista. Vuonna 2017 julkaistulla teknologialla on käyttäjiä jo selvästi paljon, joten tuki sille jatkunee. Googlen J2ObjC:tä pitäisin käyttökelpoisena työkaluna, jos Java-koodista on tarve saada iOS-sovellus, mutta kääntämiseen perustuvista työkaluista valitsisin jatkuvan tuen perusteella mieluummin Flutterin.

Natiivisovellusten pitkän aikavälin tuki näyttää olevan vakaalla pohjalla. Web-sovelluksen JavaScript-kehykseksi varmin valinta vaikuttaa tällä hetkellä olevan React. Hyviä vaihtoehtoja ovat myös Angular ja Vue. Hybridikehystä valittaessa voisi Cordovaan pohjautuva Ionic olla tulevaisuuden kannalta paras valinta, koska yhteisö tukee sitä aktiivisesti. Tulkatuista sovelluskehyksistä valitsisin React Nativen sen suosion ja taustalla olevan Facebookin vuoksi. Kääntämiseen perustuvista kehyksistä turvallisimmalta vaikuttaa Googlen Flutter.

4.2 Virrankulutus

Mobiililaitteiden ollessa riippuvaisia akuistaan on virrankulutus tärkeä ei-funktionaalinen kriteeri sovelluskehityksessä. Energia ei ole riippuvainen pelkästään tehosta, sillä energiankulutukseen vaikuttaa myös käytetty aika. Prosessorin kellotaajuuden madaltaminen ei siis riitä, sillä silloin tehtävän tekemiseen kuluu enemmän aikaa ja potentiaalisesti enemmän energiaa (Hassan ym. 2014). Energiankulutus toki on myös laajemmin tärkeä puheenaihe, kun puhutaan ilmaston lämpenemisestä ja teknologian kehityksen vaikutuksesta siihen. Freitag ym. (2021) arvioivat ICT-sektorin (engl. *information communication technology*) hiilidioksidipäästöjen olevan 2,1 - 3,9 % maailman hiilidioksidipäästöistä, mikä merkitsee, että sektorin päästöt ovat korkeammat kuin globaalin lentoliikenteen.

Corbalan ym. (2018) arvioivat natiivin, hybridin, käännetyn ja tulkatun kehitystavan virrankulutusta Android-ympäristössä. He käyttävät suoraan Qualcomm-prosessoriin integroitua Trepn Profiler -työkalua energiankulutuksen mittaamiseen. Corbalan ym. (2018) kehittivät

Kehitystyyli	Kehys	**Laskenta**	**Video**	**Audio**
Natiivi	**Android NDK**	1,8		
	Android SDK	3,5	4,8	3,9
Hybridi	**Cordova**	1,6	13,9	4,3
Ajoympäristö	**Corona**	7,3	5,0	5,2
	NativeScript	1,8	11,1	4,2
	Titanium	1,7	5,3	4,2
	Xamarin	3,0	5,1	4,0

Taulukko 3. Corbalanin ym. (2018) tutkimuksen demosovellusten virrankulutukset eri kehitystyyleillä (mWh)

tutkimustaan varten kolme sovellusta. Yksi ohjelma kuormittaa prosessoria matemaattisilla liukulukulaskuilla. Toinen sovellus on video- ja kolmas audiotoistin. Sovellukset he kehittivät Cordovalla, Titanium Mobilella, NativeScriptillä, Xamarinilla, Coronalla sekä Androidille natiivilla SDK:lla. Matemaattisen sovelluksen seitsemäs tyyli on Android-natiivi NDK-kirjasto (engl. *native development kit*), joka on tarkoitettu parhaaseen suoritustehoon Androidilla. NDK:lla sovellus on kehitetty suoraan ARM-arkkitehtuurin natiivilla C++-kielellä normaalin SDK:n käyttämän Javan sijaan, eikä se siis käytä ART-virtuaalikonetta.

Matemaattisten laskujen ratkaisemiseen Corbalanin ym. (2018) tutkimuksessa Cordova, Titanium, Android NDK ja NativeScript kuluttavat vähiten energiaa ja ovat siinä suhteessa hyvin lähellä toisiaan. Xamarin ja Android SDK -sovellukset tulevat seuraavina, sillä ne kuluttavat jo huomattavasti enemmän energiaa. Suurin energiankulutus on Coronalla kehitetyllä sovelluksella. Mielenkiintoista tässä testissä on se, että Android SDK:lla kehitetty sovellus ei kuulu tehokkaimpien sovellusten joukkoon. Se johtuu tutkijoiden mukaan Javan matemaattisten kirjastojen tehottomuudesta verrattuna muiden sovelluskehitystyylien JavaScript-kirjastoihin.

Videontoistosovelluksessa tilanne on erilainen. Energiankulutuksessa neljä toistaan hyvin lähellä olevaa ovat Android SDK:lla, Coronalla, Xamarinilla ja Titaniumilla kehitetyt sovelluset. NativeScript- ja Cordova-sovellukset kuluttavat selvästi enemmän energiaa. Corbalan ym. (2018) arvioivat Cordovan heikon tuloksen johtuvan Cordovassa käytetyn HTML-

videotoistimen tehottomuudesta.

Audiotoistinten tapauksessa erot ovat hyvin pieniä. Viisi sovelluksista on energiankulutukseltaan kymmenen prosentin sisällä toisistaan. Ainoastaan Coronalla kehitetty audiosoitin hieman tehottomampi.

Näistä testeistä Corbalan ym. (2018) saivat siis selville, että on tärkeää tietää kehitettyyn sovelluksen tyyppiin epäsopivat kehitysympäristöt. Ääritapauksessa ajoympäristöön perustuvalla Coronalla kehitetty sovellus vei matemaattisia laskuja laskevassa sovelluksessa jopa 470 % enemmän energiaa kuin tehokkain, Cordovalla kehitetty hybridisovellus. Tehokkaiden sovellusten erot eivät olleet läheskään yhtä suuria. Huomattiin myös, että ajoympäristöön perustuva Titanium Mobile ei koskaan ollut tehottomimpien sovellusten joukossa, vaikkei se myöskään ollut kaikkein tehokkain missään testitapauksessa. Tästä voidaan päätellä myös, että tietyn tyyppinen sovelluskehitystyökalu ei takaa hyvää tai huonoa tulosta virrankulutuksessa. Käytettävän työkalun optimointi voi olla tehty hyvin kuten Titaniumissa tai heikosti kuten Coronassa.

Ciman ym. (2017) ovat käyttäneet testeissään sekä Android- että iOS-laitteita. He ovat käyttäneet virrankulutuksen mittaamiseen ulkoista Monsoon PowerMonitor -virtamittaria. Tutkimuksessaan he vertailivat web-sovellusta, PhoneGapilla kehitettyä hybridisovellusta, Titaniumilla kehitettyä tulkattua sovellusta sekä MoSyncilla kehitettyä ristiinkäännettyä sovellusta. He eivät käyttäneet hybridisovelluksessa tai web-sovelluksessaan JavaScript-kirjastoja tai HTML5-ominaisuuksia DOM-puun muuttamiseen. Cimanin ym. (2017) testisovellukset mittaavat dataa testilaitteiden sensoreista ja päivittävät sitä näytölle tekstinä. Kun testilaitteiden näytöllä suoritettiin pelkkää testisovellusta, kehyksillä ei ollut suuria eroja virrankulutuksessa. Virrankulutus oli kaikilla kehyksillä alle kuuden prosentin sisällä vähiten energiaa kuluttaneesta natiivisovelluksesta.

Kun mitattiin sensoreilta tietoa, erot kehysten välillä olivat selvästi suurempia. Kääntämiseen perustuva sovelluskehys MoSync suoriutui sensoreilta luettaessa heikoiten. Tutkijat kommentoivat tämän johtuvan ohjelmien toteutuksen eroista, MoSyncin käyttämässä C++-kielessä ei ilman apukirjastoja ole tutkimuksen aikaan ollut mahdollista lähettää ja kuunnella tapahtumia (engl. *event*). Ciman ym. (2017) olivat päättäneet toteutta tutkimuksensa käyttä-

mättä apukirjastoja, joten he joutuivat kuuntelemaan sensoria päättymättömässä silmukassa, mikä johti huomattavasti suurempaan virrankulutukseen.

Muista kehyksistä web-sovellukset kuluttivat eniten energiaa. Ciman ym. (2017) sanovat tämän johtuvan mobiiliselainten kutsujen natiiviohjelmointirajapintoihin huonosta optimoinnista. PhoneGapia ja Titaniumia he vertasivat kiihtyvyysanturin datan mittaamisessa ja tämän datan piirtämisessä. Kokeessa huomattiin PhoneGapin kuluttavan vähemmän energiaa, kun dataa piirretään ruudulle Androidilla, ja Titanium kulutti vähemmän energiaa iOS:llä. Jos dataa mitataan sensorista mutta ei piirretä näytölle, Titanium kuluttaa molemmilla käyttöjärjestelmillä vähemmän energiaa. Siis ajoympäristöön perustuvan Titaniumin JavaScript-käyttöliittymän päivittäminen on tehottomampaa kuin hybridikehys PhoneGapin käyttöliittymän päivitys. Ciman ym. (2017) toteavat Titaniumin olevan paremmin optimoitu iOS:lle, kun taas PhoneGap on Androidilla joskus parempi. Näin selkeän voittajan valitseminen näistä ei ole mahdollista.

Virrankulutuksen vertailua vaikeuttaa tutkimusten teon aikaan eri tyylien vaihdellen tukemat natiivirajapinnat. Hybridikehys Cordovalla ja ajoympäristöön perustuvalla Titaniumilla kehitetyt sovellukset suoriutuivat Corbalan ym. (2018) testeissä hyvin, mutta Cimanin ym. (2017) tutkimuksessa ne eivät olleet natiivisovellusten veroisia. MoSyncin tuloksista ei voi vetää paljon johtopäätöksiä, sillä kehyksen viimeinen vakaa julkaisu on vuodelta 2013.

4.3 Tallennustila

Gonsalves (2019) on tutkinut natiivien ja Flutterilla sekä Cordovalla ja Ionic Frameworkilla kehitettyjen vastaavien sovellusten pakattua ja asennettua kokoa laitteissa. Heidän mukaansa iOS:lle ja Androidille Cordova/Ionic-sovellus on pienin sekä asennettuna laitteeseen että pakattuna, eli ladattaessa sovelluskaupasta. Androidilla natiivisovellus vie seuraavaksi vähiten tallennustilaa ja Flutter-sovellus puolestaan eniten. IOS:llä natiivisovellus vie eniten tilaa, mutta Gonsalves (2019) epäilee tämän olevan natiivisovelluksen mukana pakattavien Firebase-podien koon aiheuttama anomalia.

Ebonen ym. (2018) mukaan natiivisovellukset Androidille ja iOS:lle veivät huomattavasti vähemmän tilaa kuin monialustasovellukset, jotka oli kehitetty Xamarin Formsilla, Xamari-

nilla, Cordovalla ja Titaniumilla.

Pelkkään web-sivuun pohjautuva sovellus saattaa olla paras tallennustilan kannalta, sillä mitään suoritettavia tiedostoja ei ole tällöin käyttäjän laitteella. Selaimet toki lataavat ja säilyttävät välimuistissa usein käytettyjen sovellusten sisältöä, ja progressiivisten web-sovellusten manifesti vie oman tilansa. Gonsalves (2019) ei testannut web-sovellusta, joten hänen testissään hybridisovellus Cordovalla ja Ionicilla vei vähiten tallennustilaa. Natiivisovellukset eivät tarvitse ylimääräisen ajoympäristön paketointia sovelluksen mukaan, joten ne pääsääntöisesti ovat pienempiä kuin monialustakehysten sovellukset (Ebone ym. 2018).

4.4 Nopeus

Nopeus on tärkeä laadullinen kriteeri sovelluksen arvioinnissa. Tässä luvussa kerrotaan eri ympäristöillä kehitettyjen sovellusten nopeudesta suorittaa tehtäviä. Tutkimuksissa on saatu hieman erilaisia tuloksia Androidilla ja iOS:llä, joten niitä analysoidaan käyttöjärjestelmittäin.

- **Barros ym. (2020)**

  Kehitystyylit: Flutter (kääntäminen), natiivisovellukset, React Native (ajoympäristö)

  Tehtävät: HTTP-pyyntö hakea JSON-dataa palvelimelta, piirtää viisi kohdetta näytön listalle, tallentaa yksi alkio paikalliseen tallennustilaan ja hakea viisi alkiota paikallisesta tallennustilasta.

  – Datan piirtäminen Androidilla ruudulle ja hakeminen paikallisesta tallennustilasta natiivisovelluksella nopeampaa kuin Flutterilla ja React Nativella.
  – IOS-natiivisovellus oli datan hakemisessa palvelimelta, datan piirtämisessä sekä datan hakemisessa paikallisesta tallennustilasta Flutteria ja React Nativea nopeampi.
- **Delıa ym. (2017)**

  Kehitystyylit: Cordova (hybridi), Corona (ajoympäristö), natiivi, NativeScript (ajoympäristö), web-sovellus, Xamarin (ajoympäristö)

  Tehtävä: Raskaiden liukulukulaskujen laskeminen.

  - Androidilla web-, Cordova-, NativeScript- ja Titanium-sovellukset nopeimpia, seuraavana nopeudessa natiivi- ja Xamarin-sovellukset, selvästi heikoin oli Corona-sovellus
  - Natiivisovelluksen heikkoutta selittää natiivin Java-koodin tarvitsema ajoympäristö Android Runtime (ART).
  - IOS:llä natiivisovellus oli jokaisessa testissä suvereeni voittaja. Web-, Corona- ja Xamarin-sovellukset olivat keskivaiheilla suorituskyvyltään. Hitaimpia olivat Cordova-, NativeScript- ja Titanium-sovellukset.
- **Ebone ym. (2018)**

  Kehitystyylit: Cordova (hybridi), natiivi, Titanium (ajoympäristö), Xamarin (ajoympäristö, vain Android), Xamarin Forms (ajoympäristö)

  Tehtävä: Eri kokoisten kohteiden piirtäminen ruudulle asetettujen parametrien mukaisesti.
  - Natiivi-, Xamarin- ja Xamarin Forms -sovellusten koontiajat (engl. *build*) kasvoivat samassa suhteessa näkymän kokoon, eli ne kokoavat koko näkymän ennen sen piirtämistä. Cordova- ja Titanium-sovellusten koonti hidastuu vain vähän näkymän koon kasvaessa, eli Ebonen ym. (2018) mukaan ne kokoavat vain tarvittavan osan näkymästä.
  - Xamarin Forms -sovellusten käyttöliittymän responsiivisuus hidastuu selkeästi näkymän kasvaessa, kun taas natiivi-, Xamarin ja Titanium-sovelluksilla käyttöliittymien responsiivisuus responsiivisuus hidastuu vain vähän.
  - Cordova-sovelluksilla käyttöliittymän piirto ja responsiivisuus on Androidilla ja iOS:llä erilainen, jota selittävät HTML- ja JavaScript-moottoreiden erot käyttöjärjestelmien välillä.
- **Gonsalves (2019)**

  Kehitystyylit: Cordova ja Ionic-kirjasto (hybridi), Flutter (kääntäminen), natiivi

  Tehtävät: Sovelluksen käynnistys, kaksi eri siirtymää.
  - Androidilla Flutter ja natiivisovellus nopeampia käynnistysajoissa kuin Cordova-sovellus, Flutter nopein näkymien välillä siirryttäessä.
  - IOS:llä Flutter-sovellus nopein käynnistymään ja siirtymään näkymien välillä. Cordova/Ionic-sovellus hitain käynnistymään.

  - IOS:llä näkymien välillä siirtymisessä natiivi- ja Cordova-sovellusten välillä ei eroa.

- **Saarinen (2019)**

  Kehitystyylit: Cordova (hybridi), Flutter (kääntäminen), natiivit, React Native (ajoympäristö), Titanium Mobile (ajoympäristö), Xamarin (ajoympäristö)

  Tehtävät: Sovelluksen käynnistys, siirtymät eteen- ja taaksepäin näkymissä, listan piirtäminen näytölle hakemalla esitettävä tieto paikallisesta tallennustilasta, värinämoottorin käynnistäminen, raskaiden laskujen laskenta. Tehtävien suoritusaikaa on mitattu koodissa sekä korkeanopeuskameralla.

  - Androidilla sovelluksen käynnistämisessä nopein oli natiivisovellus, seuraavana Cordova- ja Flutter-sovellukset. IOS:llä sovellusten käynnistysajoissa Titanium ja React Native olivat nopeimpia, Flutter ja Cordova hitaimpia.
  - Uuteen näkymään siirtymisessä natiivi-, Flutter- ja Cordova-sovellukset olivat kärjessä, Xamarin-sovellus seuraavana; ja hitaimmat olivat Titanium- ja React Native -sovellukset.
  - Androidilla palattaessa näkymässä taaksepäin Cordova-, natiivi- ja Xamarin-sovellukset olivat testin nopeimmat. Titanium- ja React Native -sovellukset tulivat seuraavina, ja Flutter-sovellus oli selvästi hitain. Myös IOS:llä Flutter oli hitain, muiden välillä ero oli hyvin pieni.
  - Listan piirtämisessä Androidilla Flutter-, natiivi- ja Titanium-sovellukset olivat nopeimmat. Cordova- ja React Native -sovellukset olivat seuraavina, selvästi hitain oli Xamarin-sovellus. IOS:llä muita hitaampia olivat Cordova- ja Titanium-sovellukset
  - Värinämoottoritestissä React Native -sovellus oli muita sovelluksia hitaampi Androidilla.
  - Raskaan laskennan testissä Androidilla natiivi-, Titanium- ja Cordova-sovellukset olivat nopeimmat, Xamarin-sovellus huomattavasti hitaampi, React Native -sovellus uudestaan huomattavasti tätä hitaampi ja Flutter-sovellus taas huomattavasti React Native -sovellusta hitaampi. IOS:llä Cordova-, Titanium- ja React Native -sovelluksilla suorittaminen kesti noin 27 sekuntia natiivisovelluksen, Xamarin-sovelluksen ja Flutter-sovelluksen suoriutuessa noin sekunnissa.

Saarisen (2019) mukaan Flutterin, React Nativen ja Xamarinin hitaus johtui niiden oletusarvoisesti käyttämistä 32-bittisistä binääreistä. Flutter, Xamarin ja React Native tukevat 64-bittistä toimintaa vuonna 2022, mutta tutkimuksen teon aikaan tuki ei ollut neuvottu dokumentaatiossa (Flutter), sitä ei ollut virallisesti julkaistu (Xamarin) tai sitä ei ollut vielä olemassa (React Native).

Androidilla hybridisovellukset Cordovalla toteutettuna ovat monessa testissä nopeimpia. React Native -ajoympäristösovellus on monessa testissä ollut hitaimpien joukossa, kylläkin Saarisen 2019 tutkimuksessa 32-bittisenä, joten suuria johtopäätöksiä ei voi oletettavasti nopeammasta 64-bittisestä toiminnasta tehdä. Ajoympäristösovellukset Titaniumilla kehitettynä ovat nopeita monissa kokeissa. Xamarinilla kehitetyt ajoympäristösovellukset ovat hitaimpien joukossa. Gonsalvesin 2019 tutkimuksessa kääntämiseen perustuva Flutter oli nopea.

Kuten ylläolevasta käy ilmi, iOS:llä monialustakehyksistä selvää voittajaa on vaikea löytää. Flutter on monen tutkimuksen nopein, vaikkakin se myös häviää Saarisen (2019) tutkimuksessa näkymässä taaksepäin palaamisessa ja käynnistysnopeudessa. Cordova vaikuttaa monessa tutkimuksessa häviävän nopeudessa muille kehyksille. Titanium on molemmissa ääripäissä, koska se käynnistyy nopeimmin mutta on hidas raskaassa matematiikassa. Myöskään Xamarin Forms ei ollut nopeimpien monialustakehitysympäristöjen joukossa, kun taas Xamarinin nopeus oli keskitasoa. React Nativekin hävisi raskaassa matematiikassa mutta oli nopeudessa muuten keskivaiheilla.

Delıan ym. (2017) tutkimuksessa web-sovellukset toimivat kohtuullisen nopeasti molemmilla alustoilla, ja jos sovellus ei tarvitse erikseen natiiveja ohjelmointirajapintoja, voisi se laskentatehon perusteella olla hyvä vaihtoehto sovelluskehykseksi.

5 Yhteenveto

Mobiilisovelluksen kehitykseen käytettäväksi teknologiaksi ei ole selvää parasta teknologiaa. Tästä syystä kehitysprojektia aloittaessa on hyvä tiedostaa käytettyjen valintojen merkitys.

Web-sovellukset ovat kohtuullisen suorituskykyisiä, mikäli alustan JavaScript-moottori on optimoitu hyvin. Niiden etuihin kuuluvat myös se, että käyttäjän ei tarvitse asentaa sovellusta laitteelleen ja sovellusta voidaan päivittää palvelimella, niin että käyttäjän ei tarvitse tehdä mitään toimenpiteitä. Web-sovellukset voivat siis olla hyvä vaihtoehto saada sovellus markkinoille suurelle yleisölle helposti; riittää kun kehittäjät tuntevat web-teknologiat, joilla on kattava dokumentaatio ja joiden tuki tulee säilymään hyvänä. Progressiivisten ominaisuuksien lisääminen mahdollistaa natiivinkaltaisia ominaisuuksia myös web-sovelluksiin. Haasteen web-sovelluksen kehityksessä aiheuttaa JavaScript-kehyksen valinta. Myös suurempi virrankulutus saattaa olla ikävä sivuvaikutus web-teknologioiden käytöstä.

Mikäli vaatimusmäärittelyssä todetaan tärkeäksi käyttäjien pysyvyys ja se halutaan saavuttaa natiivilla tunnulla ja asennettavuudella, eivät web- tai hybridisovellukset ole mukana kilpailussa. Natiivisovellukset ovat pääsääntöisesti suorituskykyisin tapa kehittää sovellus. Androidilla erityisesti tehokas nopeudeltaan ja energiatehokkuudeltaan on C++ NDK. Android Studio ja hot reload -ominaisuus auttavat sekä Android- että Flutter-kehityksessä. Flutter vaikuttaa olevan suorituskykyinen monialustakehys sekä Androidilla että iOS:llä. Flutter luo käyttöliittymään natiivikomponentteja, ja sen tuki tullee jatkumaan hyvänä. Flutter-sovellusten haittapuolena on se, että päivitykset joudutaan hyväksyttämään kauppapaikoilla eikä mahdollisuutta suoriin päivityksiin ole. Flutter-kehitykseen käytettävä Dart-kieli lienee kaikille kehittäjille uusi tuttavuus.

Mielenkiintoista olisi tutkia Androidin ja iOS:n tukemien ohjelmointikielten optimointia esimerkiksi kehittämällä sama sovellus sekä Javalla, Kotlinilla että C++:lla samoin kuin Swiftillä ja Objective-C:llä. Microsoftin uuden MAUI:n nopeuden ja virrankulutuksen vertailu muihin monialustakehyksiin olisi myös mahdollisuus jatkotutkimukselle.

Lähteet

Adobe. 2022. “Update for customers using PhoneGap and PhoneGap build”. Viitattu 20. helmikuuta 2022. https://blog.phonegap.com/update-for-customers-using-phonegap-and-phonegap-build-cc701c77502c.

Apple. 2021. “What’s Included in Xcode”. Viitattu 28. marraskuuta 2021. https://developer.apple.com/xcode/whats-new/.

Barros, Lucas, Flávio Medeiros, Eduardo Moraes ja Anderson Feitosa Júnior. 2020. “Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies.” Teoksessa *SEKE,* 186–191.

Biørn-Hansen, Andreas, Christoph Rieger, Tor-Morten Grønli, Tim A Majchrzak ja Gheorghita Ghinea. 2020. “An empirical investigation of performance overhead in cross-platform mobile development frameworks”. *Empirical Software Engineering* 25:2997–3040.

Ciman, Matteo, ja Ombretta Gaggi. 2017. “An empirical analysis of energy consumption of cross-platform frameworks for mobile development”. *Pervasive and Mobile Computing* 39:214–230. ISSN: 1574-1192. https://doi.org/10.1016/j.pmcj.2016.10.004.

Corbalan, Leonardo, Juan Fernandez, Alfonso Cuitiño, Lisandro Delia, Germán Cáseres, Pablo Thomas ja Patricia Pesado. 2018. “Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption”. Teoksessa *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems,* 191–201. MOBILESoft ’18. Gothenburg, Sweden: Association for Computing Machinery. ISBN: 9781450357128. https://doi.org/10.1145/3197231.3197242.

Delıa, Lisandro, Nicolás Galdamez, Leonardo Corbalan, Patricia Pesado ja Pablo Thomas. 2017. “Approaches to mobile application development: Comparative performance analysis”. Teoksessa *2017 Computing conference,* 652–659. IEEE.

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.

Facebook. 2021a. “React GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/facebook/react.

———. 2021b. “React Native GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/facebook/react-native.

Freitag, Charlotte, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair ja Adrian Friday. 2021. “The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations”. *Patterns* 2 (9): 100340. ISSN: 2666-3899. https://doi.org/https://doi.org/10.1016/j.patter.2021.100340. https://www.sciencedirect.com/science/article/pii/S2666389921001884.

Gonsalves, Michael. 2019. “Evaluating the mobile development frameworks Apache Cordova and Flutter and their impact on the development process and application characteristics”. Tutkielma, California State University, Chico.

Google. 2020. “What J2ObjC is”. Viitattu 30. marraskuuta 2021. https://developers.google.com/j2objc.

———. 2021a. “Angular GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/angular/angular.

———. 2021b. “Flutter GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/flutter/flutter.

———. 2021c. “Meet Android Studio”. Viitattu 28. marraskuuta 2021. https://developer.android.com/studio/intro.

Grønli, T., J. Hansen, G. Ghinea ja M. Younas. 2014. “Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS”. Teoksessa *2014 IEEE 28th International Conference on Advanced Information Networking and Applications,* 635–641. https://doi.org/10.1109/AINA.2014.78.

Hassan, Hesham, ja Ahmed Moussa. 2014. “Power Aware Computing Survey”. *International Journal of Computer Applications* 90 (helmikuu). https://doi.org/10.5120/15555-4256.

Heitkötter, Henning, Sebastian Hanschke ja Tim A. Majchrzak. 2013a. “Evaluating Cross-Platform Development Approaches for Mobile Applications”. Teoksessa *Web Information Systems and Technologies,* toimittanut José Cordeiro ja Karl-Heinz Krempels, 120–138. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-36608-6.

Heitkötter, Henning, Tim A Majchrzak ja Herbert Kuchen. 2013b. “Cross-platform model-driven development of mobile applications with md2”. Teoksessa *Proceedings of the 28th Annual ACM Symposium on Applied Computing,* 526–533.

Ionic. 2022. “Ionic GitHub”. Viitattu 20. helmikuuta 2022. https://github.com/ionic-team/ionic-framework.

jQueryMobile. 2022. “jQuery Mobile GitHub”. Viitattu 20. helmikuuta 2022. https://github.com/jquery-archive/jquery-mobile.

Kharlampidi, Vladimir. 2021. “Framework7 GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/framework7io/framework7.

Majchrzak, Tim A, Andreas Biørn-Hansen ja Tor-Morten Grønli. 2018. “Progressive web apps: the definite approach to Cross-Platform development?” Teoksessa *Hawaii International Conference on System Sciences 2018 (HICSS-51).*

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. https://doi.org/10.1109/MOBILESoft.2017.7.

Monaca. 2021. “Onsen UI GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/OnsenUI/OnsenUI.

Ortinau, David. 2021. “The New .NET Multi-platform App UI”. Viitattu 1. joulukuuta 2021. https://devblogs.microsoft.com/xamarin/the-new-net-multi-platform-app-ui-maui/#net-multi-platform-app-ui-maui.

Osmani, Addy. 2015. “Getting Started with Progressive Web Apps”. Viitattu 30. marraskuuta 2021. https://developers.google.com/web/updates/2015/12/getting-started-pwa.

Osmani, Addy, ja Matt Gaunt. 2015. “Instant Loading Web Apps With An Application Shell Architecture”. Viitattu 30. marraskuuta 2021. https://medium.com/google-developers/instant-loading-web-apps-with-an-application-shell-architecture-7c0c2f10c73#.1s0o3w42k.

Progress. 2021. “NativeScript GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/NativeScript/NativeScript.

QtGroup. 2021. “Qt GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/qt/qtbase.

Rieger, Christoph, ja Herbert Kuchen. 2018. “A process-oriented modeling approach for graphical development of mobile business apps”. *Computer Languages, Systems & Structures* 53:43–58.

Saarinen, Jarkko. 2019. “Evaluating cross-platform mobile app performance with video-based measurements”. Tutkielma, Tampereen Yliopisto. https://urn.fi/URN:NBN:fi:tuni-201905161720.

Santoni, Muriel. 2021. “Progressive Web Apps browser support compatibility”. Viitattu 1. joulukuuta 2021. https://www.goodbarber.com/blog/progressive-web-apps-browser-support-compatibility-a883/.

Seidel, Eric. 2018. “Code Push / Hot Update / out of band updates”. Viitattu 26. marraskuuta 2021. https://github.com/flutter/flutter/issues/14330.

Sencha. 2022. “Sencha Touch has been merged with Ext Js”. Viitattu 20. helmikuuta 2022. https://www.sencha.com/products/touch/.

Shcherban, Vlad. 2021. “Solar2d GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/coronalabs/corona.

StackOverflow. 2021. “2021 Developer Survey”. Viitattu 22. tammikuuta 2022. https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies/.

Statista. 2019. “Global Market Share Held by Mobile operating Systems Since 2009”. Viitattu 5. helmikuuta 2020. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

Statista. 2021. “The 100 largest companies in the world by market capitalization in 2021”. Viitattu 1. joulukuuta 2021. https://www.statista.com/statistics/263264/top-companies-in-the-world-by-market-capitalization/.

Stoenescu, Razvan. 2021. “Quasar Framework GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/quasarframework/quasar.

Willocx, Michiel, Jan Vossaert ja Vincent Naessens. 2016. “Comparing Performance Parameters of Mobile App Development Strategies”. Teoksessa *Proceedings of the International Conference on Mobile Software Engineering and Systems,* 38–47. MOBILESoft ’16. Austin, Texas: Association for Computing Machinery. ISBN: 9781450341783. https://doi.org/10.1145/2897073.2897092. https://doi.org/10.1145/2897073.2897092.

Völter, Markus, Thomas Stahl, Jorn Bettin, Arno Haase ja Simon Helsen. 2013. *Model-driven software development: technology, engineering, management.* John Wiley & Sons.

You, Evan. 2021. “Vue.js GitHub”. Viitattu 1. joulukuuta 2021. https://github.com/vuejs/vue.