Tampereen yliopisto

Anton Kaarakainen

ALUSTARIIPPUMATTOMAT SOVELLUS-KEHYKSET ANDROID-SOVELLUSKEHI-TYKSESSÄ

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Toukokuu 2021

TIIVISTELMÄ

Anton Kaarakainen: Alustariippumattomat sovelluskehykset Android-sovelluskehityksessä
Kandidaatintutkielma
Tampereen yliopisto
Tietotekniikka
Toukokuu 2021

Mobiilialustoille sovelluksia kehitettäessä on perinteisesti jouduttu toteuttamaan jokaisen alustan sovellus kunkin alustan natiiveilla teknologioilla. Alalla on kuitenkin pyrkimys siirtyä käyttämään alustariippumattomia ratkaisuja, joissa samaa ohjelmakoodia voidaan hyödyntää useamman alustan sovelluksessa. Yleensä eri alustojen käyttämät teknologiat poikkeavat toisistaan, jolloin yhden ohjelmiston kehittäminen usealle alustalle on hyvin resurssi-intensiivistä. Ratkaisuksi on kehitetty erilaisia sovelluskehyksiä (engl. frameworks) ja kirjastoja, joiden avulla yksi ohjelmistoprojekti voi tuottaa sovelluksen, joka toimii usealla eri järjestelmällä. Usein tuettuina järjestelminä ovat paitsi Android ja iOS, myös muut alustat, kuten erilaiset älytelevisiot. Web-teknologioita hyödyntämällä saadaan kehitettyä yksi sovellus käytännössä kaikille nykyaikaisia selainteknologioita hyödyntäville järjestelmille.

Tässä työssä tutustutaan Android-sovelluskehitykseen liittyviin sovelluskehyksiin, jaotellaan niitä kategorioihin ja verrataan niitä sekä natiivikehitykseen, että toisiinsa. Tarkoituksena on selvittää millaisia etuja ja haittoja sovelluskehysten hyödyntämiseen liittyy. Vertailtavia osa-alueita ovat sovelluskehityksen kustannukset, sovelluksen suorituskyky, käyttäjäkokemus, tietoturva ja sovelluksen jakeluun liittyvät seikat. Sovelluskehyksillä kehitettävät sovellustyypit on jaettu viiteen kategoriaan: perinteisiin web-sovelluksiin, hybridisovelluksiin, progressiivisiin web-sovelluksiin, tulkattuihin sovelluksiin ja käännettyihin sovelluksiin.

Työssä todetaan, että alustariippumattomia sovelluskehyksiä hyödyntävät sovellukset eivät yleisesti yllä samalle laatutasolle natiivisovellusten kanssa suorituskyvyn tai käyttäjäkokemuksen osalta. Myös laitteisto-ominaisuuksien tuki vaihtelee suuresti eri teknologioiden välillä. Alustariippumattomien ratkaisuiden suurimmat hyödyt liittyvät sovelluskehityksen nopeutumiseen ja ketteryysetuihin, sekä niiden kautta saavutettaviin kustannusetuihin. Uudet sovelluskehykset ovat myös jatkuvassa kehityksessä, mikä voi aiheuttaa odottamattomia ongelmia sovelluskehitysprosessissa. Sovelluskehysten käyttöä kannattaa harkita tapauskohtaisesti ja oikeiden teknologioiden valintaan on hyvä kiinnittää huomiota.

Avainsanat: Android, mobiilisovellus, alustariippumaton sovelluskehitys, sovelluskehys, hybridisovellus, web-sovellus, natiivisovellus

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface, ohjelmointirajapinta
APK	Android Package, Android-sovellusohjelmien pakettitiedosto, joita käytetään sovelluksien asentamiseen
CSS	Cascading Style Sheets, web-dokumentin tyyliohjeet sisältävä tiedostotyyppi
DOM	Document Object Model, dokumenttioliomalli
HTML	Hypertext Markup Language, hypertekstin merkintäkieli
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
KMM	Kotlin Multiplatform Mobile, alustariippumaton kehitysympäristö
NDK	Native Development Kit, työkalusetti C-kielien käyttämiseen Android-ympäristössä
PWA	Progressive Web Application, progressiivinen web-sovellus
RAM	Random Access Memory, hajasaantimuisti
SDK	Software Development Kit, ohjelmistokehityspaketti
TWA	Trusted Web Activity
URL	Uniform Resource Locator
UWP	Universal Windows Platform, Microsoftin kehittämä alusta Windows-järjestelmien yhteisille sovelluksille
WORA	engl. Write Once, Run Anywhere

1. JOHDANTO

Mobiililaitteiden markkinat ovat jakautuneet kahden ekosysteemin hallitsemaksi duopoliksi. Statcounterin tilaston [1] mukaan, vuonna 2021 mobiililaitteiden käyttäjämäärän ennakoidaan saavuttavan maailmanlaajuisesti 3,8 miljardia käyttäjää. Googlen Android on määrällisesti suurin ekosysteemi noin 72 prosentin markkinaosuudella. Applen mobiililaitteiden hyödyntämä iOS puolestaan pitää hallussaan noin 27 prosenttia markkinoista. Yhdessä nämä kaksi ekosysteemiä hallitsevat siis noin 99 prosenttia koko markkinasta. [1] Tässä työssä käsitteitä ekosysteemi, alusta ja järjestelmä käytetään synonyymeinä.

Mobiilialustoille kehitettäviä ohjelmistoja suunnitellessa perinteinen ratkaisu on kehittää erillinen sovellus kullekin järjestelmälle, niille suositetuilla omilla teknologioillaan. Tässä ratkaisussa on etunsa, mutta luonnollisesti useamman sovelluksen rinnakkainen kehittäminen vaatii paljon enemmän resursseja kuin yhteen sovellukseen keskittyminen. Markkinan jakautuminen käytännössä kahteen eri käyttöjärjestelmään on tuonut uusia mahdollisuuksia näiden alustojen sovelluskehitykseen.

Alalla on selkeä pyrkimys edetä suuntaan, jossa yhdestä ohjelmakoodipohjasta voitaisiin rakentaa sovelluksia useammalle alustalle. Erilaiset sovelluskehitykseen luodut ohjelmistokehykset, eli sovelluskehykset (engl. frameworks), tarjoavat jokainen hieman erilaisen ratkaisun alustariippumattomuuden saavuttamiseen. Tässä työssä tutustutaan Android-sovelluskehitykseen liittyviin sovelluskehyksiin ja tehdään niihin liittyvää vertailua. Sovelluskehyksien vertailtavia osa-alueita ovat sovelluskehityksen kustannukset, sovelluksen suorituskyky, käyttäjäkokemus, tietoturva ja sovelluksen jakeluun liittyvät asiat.

Aluksi luvussa 2 käsitellään Android-sovelluskehitykseen liittyviä vaihtoehtoja yleisellä tasolla ja avataan natiivikehityksen ja alustariippumattomien ratkaisujen eroja. Luvussa 3 esitellään erilaisia alustariippumattomia tapoja kehittää sovelluksia ja tarkastellaan niihin liittyviä esimerkkikehyksiä. Käsiteltävät teknologiat on jaettu aihetta käsittelevän kirjallisuuden perusteella kategorioihin. Lopulta luvussa 4 analysoidaan sovelluskehyksiin liittyviä etuja ja haittoja sekä verrataan niitä toisiinsa ja Android-natiivikehitykseen valituilla osa-alueilla.

2. ANDROID-SOVELLUSKEHITYS YLEISESTI

Jokaiselle nykyaikaiselle käyttöjärjestelmälle on lähtökohtaisesti olemassa jonkinlainen virallinen tapa kehittää sille sovelluksia. Yleensä eri alustojen käyttämät teknologiat poikkeavat kuitenkin toisistaan, jolloin yhden ohjelmiston kehittäminen usealle alustalle on hyvin resurssi-intensiivistä. Yhdeksi ratkaisuksi ongelmaan tarjotaan alustariippumattomia teknologioita.

2.1 Natiivi sovelluskehitys

Tämän työn kontekstissa natiivisovelluksella tarkoitetaan sovellusta, joka on kehitetty käyttöjärjestelmää varten luoduilla työkaluilla ja niihin liittyvillä ohjelmointikielillä. Googlen suosittelema ohjelmointikieli Android-sovelluksille on Kotlin. Muita tuettuja kieliä ovat muun muassa Java ja C++. [2] Android-ympäristön natiivi sovelluskehitys tapahtuu Android Software Development Kitin (SDK) avulla. Tarvittaessa myös C++ -kielinen koodi käännetään laitekohtaiselle konekielelle Native Development Kitin (NDK) avulla. [3]

Android SDK pakkaa sovelluksen koodin muun datan ja resurssitiedostojen kanssa APK-tiedostoon (Android package), jolla sovellus asennetaan käyttäjälaitteelle. Asennettua sovellusta suoritetaan Androidilla omassa virtuaalikoneessaan eli hiekkalaatikossa (engl. security sandbox) eristettynä muista sovelluksista. [2]

Lähes kaikki Android-alustalla länsimaissa käytetyistä natiivisovelluksista ladataan Google Play -sovelluskaupasta. Vuonna 2020 Google Play -kaupasta ladattiin sovelluksia maailmanlaajuisesti lähes 110 miljardia kertaa. [4] Tarjolla on myös useita pienempiä sovelluskauppoja.

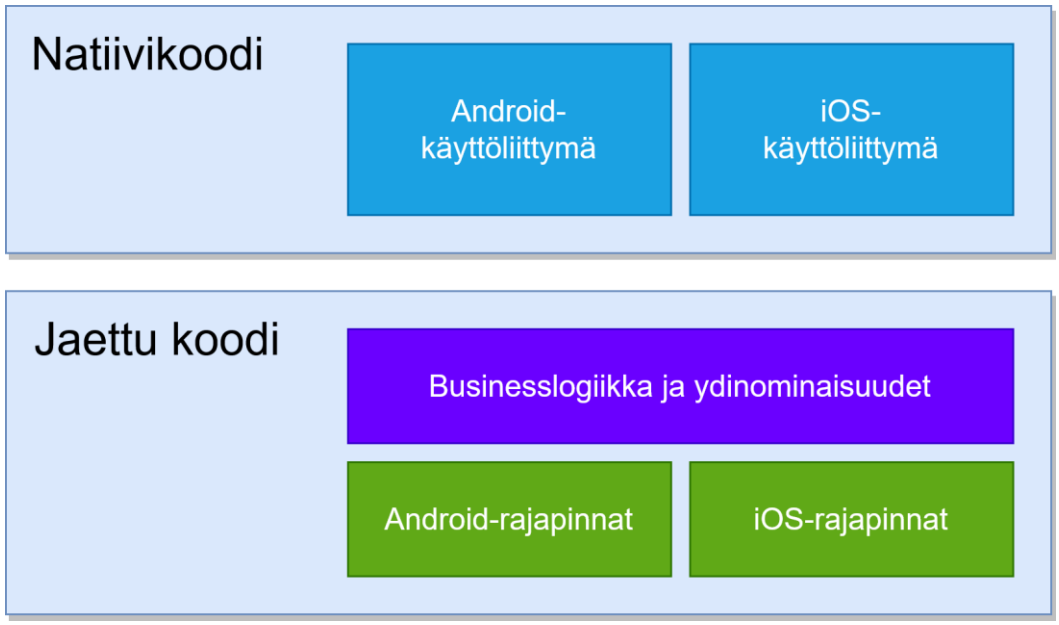
2.2 Alustariippumaton sovelluskehitys

Vaihtoehto puhtaasti natiiville sovelluskehitykselle on käyttää samaa koodipohjaa, tai sen osaa, usealla alustalla toimivassa sovelluksessa. Tähän on kehitetty useita eri ratkaisuja, jotka lähestyvät ongelmaa eri tavoin.

Esimerkiksi Googlen Androidille suosittelema ohjelmointikieli Kotlin on itsessään suunniteltu alustariippumattomaksi. Niin sanottu ”yleinen Kotlin” sisältää itse kielen, siihen liittyvät ydinkirjastot ja perustyökalut. Tämä osa koodista toimii kaikilla Kotlinia tukevilla alustoilla. Näiden lisäksi on olemassa Kotlin-monialustakirjastoja, jotka auttavat koodin

uudelleenkäytössä eri alustojen välillä esimerkiksi HTTP-protokollan osalta. Kotlinin alustakohtaiset versiot sisältävät valmiiksi näitä laajennuksia. [5]

Androidin ja iOS:n yhteiseen sovelluskehitykseen on olemassa Kotlin Multiplatform Mobile (KMM) -kehitysympäristö. KMM mahdollistaa yhteisen koodikannan sovelluksen pohjimmaiselle logiikalle (engl. business logic) ja muille ydinominaisuuksille, mutta alustakohtaiset rajapinnat ja käyttöliittymä vaativat oman toteutuksensa kullekin alustalle erikseen. [5] Kuvassa 1 havainnollistetaan Kotlin Multiplatform Mobile -sovelluksen koodin uudelleenkäytettävyyttä Androidin ja iOS:n välillä. Käyttöliittymä täytyy KMM-sovelluksessa kirjoittaa alustan natiivikeinoilla.

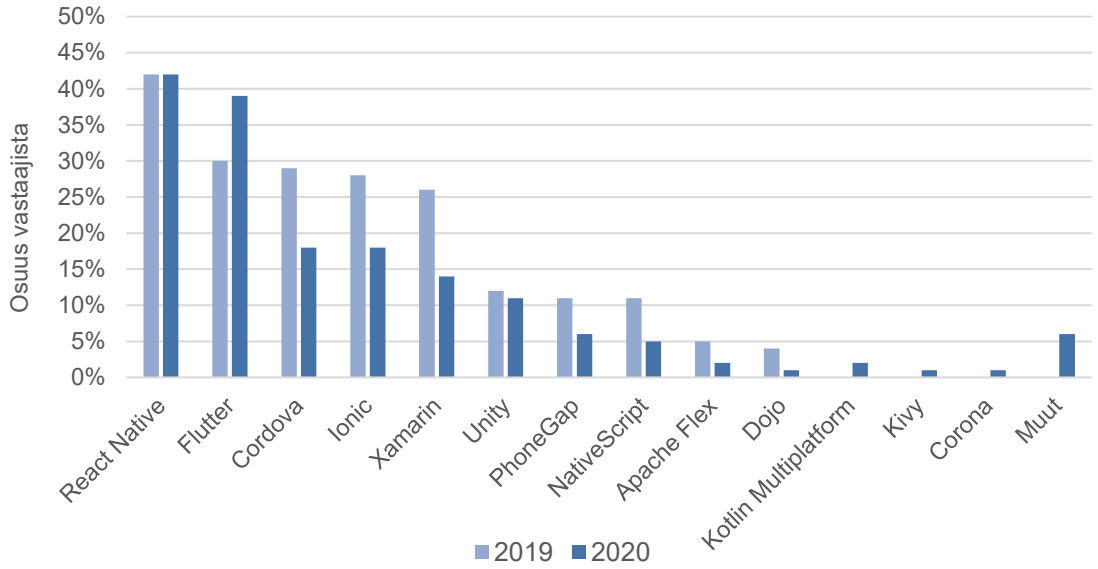


Kuva 1. *Havainnollistus Kotlin Multiplatform Mobile -sovelluksen arkkitehtuurista. [6]*

Natiivin sovelluskehityksen rinnalle on pitkään ennustettu myös web-teknologioiden nousua niiden yleisen alustariippumattomuuden vuoksi. Jo vuonna 2007 ensimmäisen iPhonen julkaisun yhteydessä Applen edesmennyt toimitusjohtaja Steve Jobs kertoi [7], että iPhonen sovelluskehitys ei tulisi vaatimaan erillistä SDK:ta. Sen sijaan se hyödyntäisi kehittäjille jo valmiiksi tuttuja web-teknologioita, kuten AJAXia (Asynchronous JavaScript and XML).

Myöhemmin Apple perui tämän lähestymistavan, jolloin päädyttiin nykytilanteeseen, jossa iOS-järjestelmälle kehitettävät sovellukset käyttävät niille kehitettyjä omia teknologioitaan. Erilaisia web-pohjaisia ratkaisuja on kuitenkin ollut olemassa lähes yhtä kauan kuin mobiililaitteilla on voinut selata internetiä. Uusimpina tulokkaina ovat muun muassa progressiiviset web-sovellukset, jotka muistuttavat ominaisuuksiltaan hyvin paljon Applen vuoden 2007 visiota älypuhelimen sovelluksista.

JetBrainsin vuonna 2020 suorittaman kyselyn [8] mukaan joka kolmas sovelluskehittäjä käyttää joitakin alustariippumattomia teknologioita tai sovelluskehyksiä. Kaksi kolmannesta taas on pidättäytynyt natiiviteknologioissa. Kuvassa 2 on esitetty kyselyn tulokset graafisessa muodossa. Tuloksista käyvät ilmi käytetyimmät sovelluskehykset vuosina 2019 ja 2020.



Kuva 2. *Sovelluskehittäjien käyttämät alustariippumattomat sovelluskehykset vuosina 2019 ja 2020. [8]*

Käytetyimmät alustariippumattomat sovelluskehykset vuonna 2020 olivat React Native ja Flutter. Sovelluskehyksiä käyttävistä kehittäjistä 42 prosenttia ilmoitti käyttävänsä React Nativea ja 39 prosenttia Flutteria. Muita suosittuja vaihtoehtoja olivat muun muassa Apache Cordova (18 %), Ionic (18 %) ja Xamarin (14 %). Niiden suosio oli kuitenkin laskussa edellisvuodesta noin 10 prosenttiyksikköä. Flutter puolestaan kasvatti suosiotaan eniten. Listattuna olivat myös muun muassa pelimoottori Unity (11 %), Apache Cordovan suljetun lähdekoodin versio PhoneGap (6 %) ja aiemmin mainittu Kotlin Multiplatform Mobile (2 %). [8]

3. SOVELLUSKEHYSTEN KATEGORIAT

Android-sovelluskehitykseen soveltuvat alustariippumattomat sovelluskehykset jakautuvat karkeasti neljään eri kategoriaan: web-sovelluksiin, hybridisovelluksiin, tulkattaviin sovelluksiin ja käännettäviin sovelluksiin. [9, s. 2] Jako ei kuitenkaan ole yksiselitteinen, ja näitä käsitteitä käytetään joskus eri tavalla eri yhteyksissä. Etenkin hybridisovelluksella tarkoitetaan usein laajemmin alustariippumattomia teknologioita hyödyntävää sovellusta. Tässä työssä hybridisovellus tarkoittaa web-teknologioita hyödyntävää sovellusta, jota suoritetaan natiivissa ohjelmistosäiliössä. Tämän lisäksi progressiiviset web-sovellukset (PWA) esitetään web-sovelluksista erillään omassa kategoriassaan niiden eroavaisuuksien vuoksi.

3.1 Web-sovellukset

Yksi yksinkertaisimmista tavoista tehdä usealla alustalla ja laitteella toimiva sovellus on selainpohjainen ratkaisu. Web-sovellukset ovat web-sivustoja, jotka suoritetaan esimerkiksi sovellukseksi naamioidussa internet-selaimessa, tai suoraan käyttäjän valitsemassa selaimessa. Täten niissä myös hyödynnetään yleisiä web-teknologioita, kuten HTML:ää, CSS:ää sekä JavaScriptiä. Web-sovelluksien kehityksessä on yleistä käyttää myös web-ohjelmistokehystä, kuten Angularia, Reactia tai Vue.js:ää. [9, s. 2–3]

Nykyaikaiset web-sivustot muotoutuvat käyttäjän näyttökoon ja -muodon mukaan, jolloin samaan lähdekoodiin pohjautuva sivusto on käytettävissä helposti erityyppisillä laitteilla. Web-sovelluksia voidaan suorittaa lähes millä tahansa alustalla, joka tukee moderneja verkkoselainteknologioita. Koska kaikki web-sovelluksen esittämä sisältö haetaan erillisiltä palvelimilta, niitä ei myöskään tarvitse asentaa erikseen käyttäjälaitteelle. [9, s. 2–3]

3.2 Hybridisovellukset

Hybridisovellukset käyttävät samoja teknologioita kuin web-sovelluksetkin, mutta niitä suoritetaan yksittäiselle alustalle natiivissa ohjelmistosäiliössä (engl. software container). Kategorian nimi tulee alustalle natiivin sovellusosan ja sisällön esittävän web-osan yhdistelmästä. Web-osa suoritetaan Androidilla WebView-komponentin avulla, joka on natiivisovellukseen upotettava verkkoselain. Sen käyttö poistaa tarpeen käyttää erillistä verkkoselainta web-sisällön näyttämiseen. [10]

Tämänkaltainen arkkitehtuuri mahdollistaa yhden koodipohjan uudelleenkäytön kaikilla tuetuilla alustoilla tavallisten web-sovellusten tapaan. WebView:ssä suoritettavat komponentit ovat luonnostaan alustariippumattomia ja hybridisovelluskehys tarjoaa yhteyden alustakohtaisiin ominaisuuksiin. [9, s. 3–4] Koska hybridisovelluskehys esittää web-sisällön alustakohtaisessa kevyessä natiivisovelluksessa, hybridisovellusten jakelu hoidetaan yleensä alustakohtaisten sovelluskauppojen välityksellä natiivisovellusten tapaan.

Yksi tunnetuimmista klassisista hybridisovelluskehyksistä on vuodesta 2009 kehitetty Apache Cordova. Se tunnettiin aiemmin Adoben omistuksessa nimellä PhoneGap, mutta julkaistiin myöhemmin erikseen myös avoimena lähdekoodina. [9, s. 3]

Apache Cordovalla kehitetty sovellus koostuu käyttöliittymän esittävästä WebView:stä ja liitännäisistä (engl. plugins), jotka yhdistävät Cordova-sovelluksen käyttäjälaitteen tarjoamiin ominaisuuksiin, kuten kameraan tai kiihtyvyysanturiin. Liitännäisten avulla on mahdollista hyödyntää järjestelmän omia rajapintoja suoraan JavaScriptillä. Cordovan liitännäiset muuttavat JavaScriptillä tehdyt rajapintakutsut jokaiselle alustalle ominaisiksi kutsuiksi. [9, s. 4]

Kuten tavallisia web-sovelluksia kehitettäessä, myös hybridisovellusten web-teknologioihin perustuvan osan kehittämisessä voidaan hyödyntää erillisiä web-ohjelmistokehyksiä. Esimerkiksi aiemmin mainittu Ionic-ohjelmistokehys perustuu osittain Apache Cordovaan. Se tarjosi alun perin lähinnä CSS- ja JavaScript-kirjastoja hybridisovelluksien käyttöliittymien luomiseen. Nykyään Ionic on itsenäinen hybridisovellusten ohjelmistokehityspaketti, jonka kanssa voidaan käyttää muita web-ohjelmistokehyksiä. [11]

3.3 Progressiiviset web-sovellukset

Progressiiviset web-sovellukset (engl. Progressive Web Applications, PWA) ovat web-sovelluksia, jotka hyödyntävät uuden sukupolven selainrajapintoja ja ovat niiden ansiosta hybridisovellusten tapaan ominaisuuksiltaan tavallisia web-sovelluksia kattavampia. Progressiivisilla web-sovelluksilla on pääsy useisiin laitteisto-ominaisuuksiin, ja ne voivat toimia myös ilman aktiivista verkkoyhteyttä. [12]

Jotta sovellus voisi toimia itsenäisesti taustalla, progressiiviset web-sovellukset sisältävät niin sanotun *service workerin* eli taustalla suoritettavan JavaScript-tiedoston. Sen avulla sovellus voi esimerkiksi lähettää push-ilmoituksia sovelluksen käyttäjälle näkyvissä olevan osan ollessa suljettuna. Progressiivisista web-sovelluksista on myös haluttu tehdä käyttäjälaitteisiin asennettavia, jotta ne muistuttaisivat käytökseltään mahdollisimman paljon perinteisiä sovelluksia. PWA voidaan kiinnittää laitteen kotinäkymään ja niitä voidaan käsitellä moniajonäkymässä natiivi- ja hybridisovellusten tapaan. [13]

Mozillan mukaan [13] progressiivisten web-sovellusten tavoitteisiin kuuluvat muun muassa löydettävyys ja linkitettävyys. Löydettävyystavoite viittaa sen näkyvyyteen hakukoneissa, eli siten helppoon löydettävyyteen. Linkitettävyydellä puolestaan tarkoitetaan PWA:n saatavuutta tietystä URL-osoitteesta perinteisten web-sivustojen tapaan. Tällöin PWA:ta ei tarvitse etsiä erillisestä sovelluskaupasta. Progressiivisten web-sovellusten suosion lisääntyessä myös sovelluskaupat ovat kuitenkin alkaneet hyväksyä PWA-formaatin niissä julkaistavissa sovelluksissa. Esimerkiksi Microsoft on kertonut tukevansa progressiivisia web-sovelluksia Microsoft Store -sovelluskaupassaan ja nostanut ne samanarvoiseen asemaan omien Universal Windows Platform (UWP) -sovellustensa kanssa. [14]

Yleensä sovelluskaupassa julkaistava (progressiivinen) web-sovellus pakataan hybridisovelluksista tuttuun tapaan ohjelmistosäiliönä toimivaan natiivisovellukseen. Googlen ratkaisu tähän on Trusted Web Activity (TWA) -ominaisuus. TWA on Chrome-selaimen mukautetun välilehden (engl. Chrome Custom Tab) erikoismuoto. Chromen mukautetut välilehdet mahdollistavat Chrome-selaimen avaamisen Android-natiivisovelluksessa. Idea on samantyylinen kuin hybridisovelluksissa tutussa WebView:ssä. TWA eroaa kuitenkin yleisesti käytetystä WebView:stä sen tarjoamien ominaisuuksien suhteen. [15]

Mukautettu välilehti antaa sovelluskehittäjälle enemmän sananvaltaa sovelluksen ulkonäköön, mutta on huomattavasti perinteistä WebView:tä suorituskykyisempi. Trusted Web Activitiesia hyödyntävä sovellus on mahdollista julkaista Google Play -sovelluskaupassa. TWA:lla on myös ominaisuuksia, joita ilman hybridisovelluskehystä käytetty WebView ei tarjoa, kuten push-ilmoitukset, taustasynkronointi ja Androidin natiivin jakamisnäkymän käyttömahdollisuus. [15]

3.4 Tulkatut sovellukset

Tulkatut sovellukset pyrkivät parantamaan käyttäjäkokemusta edellisistä kategorioista käyttämällä käyttöjärjestelmän omia, alustakohtaisia käyttöliittymäkomponentteja. Niiden on tarkoitus mahdollistaa parempi suorituskyky ja yhtenäisempi kokemus natiivisovellusten kanssa. Kategorian nimi tulee tavasta tulkata sovelluksen lähdekoodi käyttäjälaitteessa käyttäen JavaScript-moottoria. [9, s. 4]

Vuoden 2015 julkaisunsa jälkeen suosituin tulkkaukseen perustuva sovelluskehys on ollut Facebookin kehittämä React Native [8]. Se on JavaScript-kirjasto ReactJS:ään pohjautuva avoimen lähdekoodin sovelluskehys. React Nativella voidaan kehittää sovelluk-

sia kaikille ReactJS:n tukemille alustoille kuten verkkoselaimille, älytelevisioihin, Windowsille sekä niiden lisäksi Androidille ja iOS:lle, joiden tapauksessa käytetään hyväksi kehitysalustojen natiiveja käyttöliittymäkomponentteja. [16]

React Nativen toimintaperiaate on hyvin samanlainen kuin ReactJS:n. Molempien pääasiallinen ohjelmointikieli on JavaScript ja perinteisen dokumenttioliomallin (DOM) sijaan React Native käyttää ReactJS:n tapaan virtuaalista dokumenttioliomallia. Virtuaalisessa DOM:ssa dokumenttioliomalli tallennetaan muistiin, ja siihen päivitetään vain muutetut osat koko DOM:in uudelleenlataamisen sijaan. Tästä saadaan huomattava nopeushyöty käytännön sovelluskehityksessä. [9, s. 4]

Olennaisin ero arkkitehtuurissa on niin sanottu *Silta* (engl. Bridge), joka yhdistää React Native -sovelluksen alustakohtaisiin rajapintoihin. Siinä missä ReactJS:llä rakennettu web-sovellus käyttää selaimen DOM:ia käyttöliittymän rakentamiseen, React Native -sovellus kutsuu Androidilla sen natiiveja Java- tai Kotlin-käyttöliittymärajapintoja. Reactin komponentit piilottavat natiivirajapintojen kutsut sovelluskehittäjältä, joka kasaa sovelluksen React-komponenteista deklaratiivisesti JavaScriptillä. JavaScript-moottoria käytetään sillan rinnalla käyttöliittymäkomponenttien kanssa vuorovaikuttamiseen. [9, s. 4–5]

Sovelluksen jaettu lähdekoodi voi sisältää alustakohtaisesti konfiguroituja komponentteja, mutta lähtökohtaisesti kaikki koodi on yhä tällöinkin jaettua. Androidia varten React Native sisältää esimerkiksi takaisin-painikkeen käsittelyyn liittyvän komponentin. Suurin osa sovelluksen rakentamiseen liittyvistä komponenteista on ReactJS:stä kehittäjille tuttuja rakenteita. [16]

React Nativella kehitetyt sovellukset on mahdollista julkaista alustakohtaisissa sovelluskaupoissa tai vaihtoehtoisesti web-alustalle kehitettäessä verkossa. Muita tulkattavia mobiilikehitykseen soveltuvia sovelluskehyksiä ovat muun muassa Microsoftin kehittämä Xamarin, NativeScript sekä omaa QML-merkintäkieltään hyödyntävä Qt.

3.5 Käännetyt sovellukset

Tässä työssä mainituista kategorioista uusimpana tulokkaana ovat käännöspohjaiset sovelluskehykset. Ne käyttävät omaa grafiikkamoottoriaan natiivin käyttöliittymän imitointiin. Kaikki käyttöliittymäkomponentit ovat omia pienoisohjelmiaan (engl. widget), joista käyttöliittymä rakennetaan. [9, s. 5]

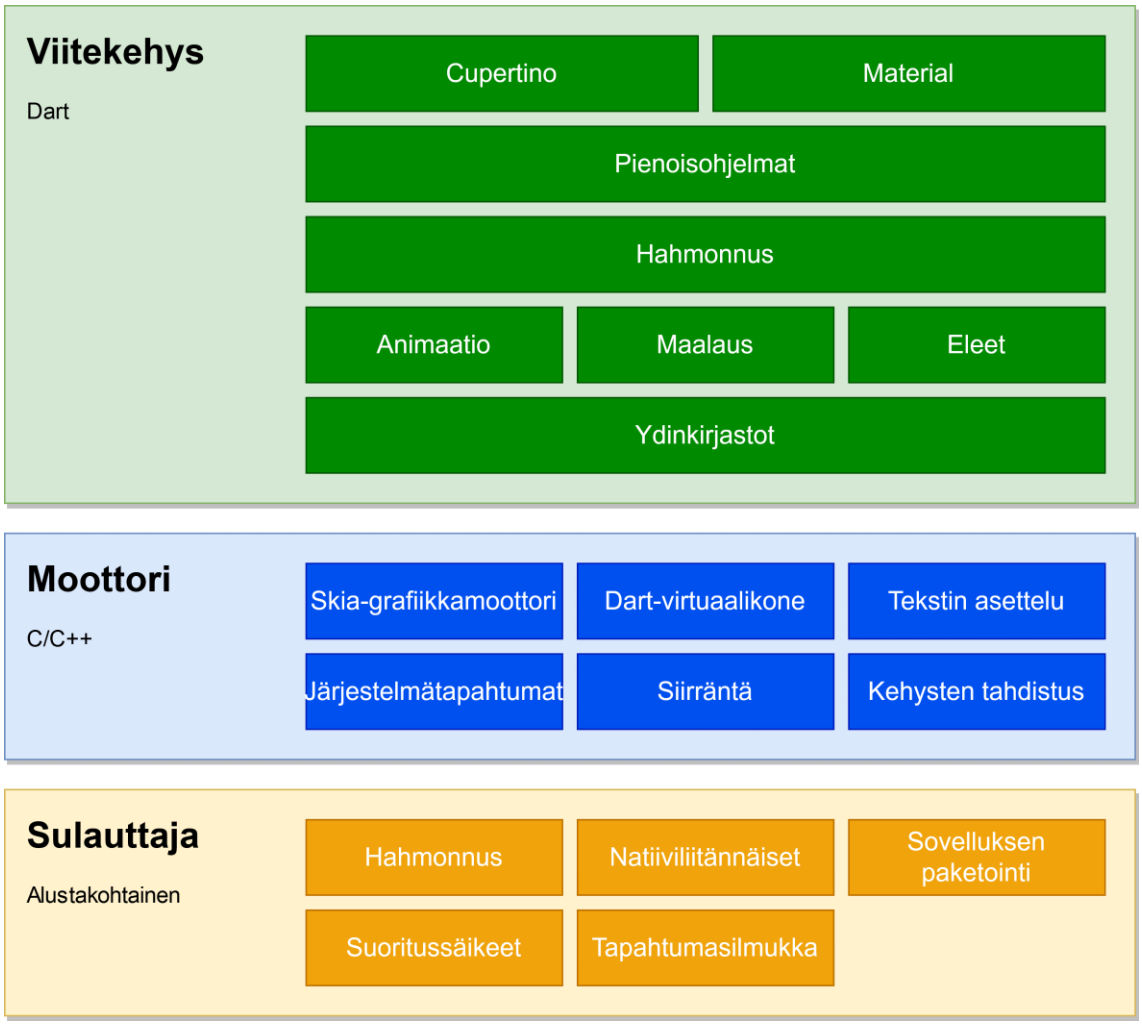
Googlen toukokuussa 2017 julkaisema avoimen lähdekoodin sovelluskehys Flutter on ponnistanut nopeasti suosituimpien sovelluskehysten joukkoon. Flutter-projekti on kerännyt kirjoitushetkellä GitHubin käyttäjiltä yli 117 000 tähtimerkintää [17], ohittaen tällä

mittarilla suosiossa jopa aikaisemmin julkaistun React Native -projektin, jolle tähden on antanut 94 200 käyttäjää [18]. Flutter-sovellukset hyödyntävät pienoisohjelmapohjaista rakennetta, joka luodaan tarkoitukseen kehitetyllä Dart-ohjelmointikielellä sekä Skia 2D -grafiikkamoottorilla. [9, s. 5]

Flutter koostuu käytännössä kahdesta erillisestä moduulista: itse kehittäjille suunnatusta Flutter-viitekehyksestä ja alla piilossa toimivasta Flutter-moottorista. Päällimmäinen viitekehysosa perustuu Dart-kieleen. Se tarjoaa sovelluskehittäjille Androidin suunnitteluohjeiden mukaan tyylitellyt Material- ja iOS-tyyliset Cupertino-pienoisohjelmakirjastot sekä animaatiot, asettelun (engl. layout) ja muut käyttöliittymäorientoituneet ominaisuudet. [9, s. 5]

Dart-luokat yhdistetään Flutter-moottoriin dart:ui-kirjaston avulla. Taustalla toimiva moottori tarjoaa matalan tason toteutuksen Flutterin ydinrajapinnoille, kuten grafiikkahahmonnuksen Skia 2D:llä, tekstinkäsittelyn, asettelun, siirräntäominaisuudet (engl. input-output) sekä Dart-virtuaalikoneen, jossa ylemmän tason viitekehyksen koodi suoritetaan. Moottori perustuu C++ -kieleen, jolloin sen suoritus käännetään Android-sovelluksessa alustan natiivikielelle Native Development Kitin (NDK) avulla. [19]

Kuvassa 3 havainnollistetaan Flutterin arkkitehtuurillisen rakenteen jakautumista kolmeen pääosaan. Ylimpänä on esitetty kehittäjälle näkyvissä oleva osa, eli Dart-kieleen pohjautuva viitekehys. Viitekehyksen alla on taustalla toimiva Flutter-moottori. Alin taso on niin sanottu sulauttaja (engl. embedder).



Kuva 3. *Flutterin arkkitehtuurilliset tasot. [9,19]*

Flutter-sovellus näyttäytyy muiden sovellusten tapaan järjestelmälle tavallisena natiivisovelluksena. Sovelluksen natiivimuotoon paketoinnin hoitaa sulauttaja, joka toimii Flutter-moottorin yhteytenä käyttöjärjestelmään. Se myös mahdollistaa pääsyn useisiin järjestelmäominaisuuksiin, kuten tapahtumasilmukkaan (engl. message event loop), käyttäjäsyötteeseen ja hahmonnuspintoihin (engl. rendering surfaces). Sulauttaja on toteutettu aina alustalle ominaisella ohjelmointikielellä, Androidin tapauksessa Javalla ja C++:lla. [19]

4. SOVELLUSKEHYSTEN EDUT JA HAITAT

Vuonna 2012 Facebookin toimitusjohtaja Mark Zuckerberg totesi haastattelussa [20] yhtiön suurimman silloisen virheen olleen liiallinen luotto HTML5-teknologiaan natiivikehityksen kustannuksella. Tavoitellut edut HTML5-teknologiaan panostamisessa olivat samoja kuin kaikessa alustariippumattomassa kehityksessä nykyäänkin. Sama koodipohja skaalautuisi hyvin usealle alustalle ja näyttökoolle, jolloin kustannustehokkuus usean alustan natiivikehitykseen verrattuna olisi ilmeinen. Odotus käyttäjäkokemuksen vertautumisesta natiivisovelluksiin oli kuitenkin virheellinen. Zuckerbergin mukaan HTML5 ei yksinkertaisesti ollut valmis. [20]

Facebook lupasi parantaa mobiilisovellusten käyttäjäkokemusta [20], ja vain vuotta myöhemmin yhtiö julkaisi ReactJS-kirjastonsa ensimmäisen version, mobiiliympäristöihin suunnatun React Nativen julkaisun seuratessa vuonna 2015. React Nativen ja muiden alustariippumattomien sovelluskehysten suosion jatkaessa tasaista kasvua on selvää, että monissa tapauksissa alustariippumattomat kehykset nähdään vakavana vaihtoehtona myös mobiilialustojen sovelluskehityksessä.

Alustariippumattomiin sovelluskehyksiin liittyy monenlaisia etuja ja haittapuolia. Itse sovelluksissa korostuvat etenkin käyttäjäkokemus ja käytettävissä olevat laitekohtaiset ominaisuudet. Huomioon otettavia näkökulmia löytyy kuitenkin myös sovelluskehitysprosessista sekä sovelluksen jakelusta.

4.1 Sovelluskehityksen kustannukset

Alustariippumattomat sovelluskehykset käyttävät useimmiten yleisesti tunnettuja teknologioita ja ohjelmointikieliä, kuten JavaScriptiä. WORA-sloganin (Write Once, Run Anywhere) mukaisesti sama koodipohja on hyvin laajalti käytössä sovelluksen kaikissa eri alustojen versioissa. Natiivisovelluksia kehitettäessä on tärkeää valita tuettavat käyttöjärjestelmät, sillä eri järjestelmien sovelluskehitys vaatii hyvin erilaista osaamista. Yleisesti taitava alustariippumattoman sovelluskehyksen osaaja voi taas kehittää yhtä aikaa sekä Android, että iOS versiota samasta sovelluksesta, mikä tuo mahdollisesti huomattavia säästöjä. [9, s. 1]

Luonnollisesti alustariippumattomia ohjelmistoja kehitettäessä myös sovelluksen ylläpito helpottuu, sillä sama korjauspäivitys voidaan tuoda sellaisenaan kaikille alustoille [9, s. 2]. Sovelluskehysten välisiä erojakin kuitenkin on. Web-teknologioihin ja tulkkaukseen

perustuvia sovelluskehyksiä käytettäessä voidaan tietyissä tapauksissa ohittaa alustakohtaiset sovelluskaupat ja hyödyntää Microsoft CodePushin kaltaisia työkaluja JavaScript koodin puskemiseen suoraan käyttäjille. Kääntämiseen perustuvat sovelluskehykset, kuten Flutter, eivät tähän kuitenkaan pysty. [21, s. 3003]

Oikean teknologian valinta on tärkeää, sillä sovelluskehyksestä toiseen vaihtaminen voi pahimmillaan tarkoittaa käyttöliittymän ja pohjimmaisen logiikankin kirjoittamista uudelleen. [21, s. 3002–3003] Esimerkiksi React Nativen JavaScriptillä ohjelmoiduilla käyttöliittymillä ei ole juuri mitään yhteistä Qt:n QML-kieleen perustuvan käyttöliittymäratkaisun kanssa. Sama ongelma ei ole yhtä laaja web-teknologioihin perustuvissa ratkaisuissa, jolloin niissä pysyttäytymien voi tuoda ohjelmistokehitysprosessiin joustavuutta.

Web-teknologioihin nojaavat ratkaisut, eli web- ja hybridisovellukset ovat yleensä ratkaisuista halvimpia. Muut alustariippumattomat vaihtoehdot ovat hieman monimutkaisempia toteuttaa, mutta natiivisovelluksiin verrattuna kustannusetuna on yhä usean alustan yhtäaikainen tuki. [9, s. 6]

4.2 Suorituskyky

Kustannusten vastapainona toisella puolella vaakakupissa painavat käyttäjien kokema laatu ja suorituskyky. Sovelluksia kehitetään monenlaisiin tarpeisiin, ja jotkin niistä esittävät suurempia vaatimuksia sovelluksen suorituskyvylle kuin toiset.

Tammikuussa 2019 julkaistun Biørn-Hansen et al. järjestämän kyselytutkimuksen [22] (N = 101) mukaan yleisimmät alustariippumattomien sovelluskehysten käyttöön liitetyt ongelmat olivat yleinen suorituskyvyn heikkeneminen natiivisovelluksiin verrattuna (63 % vastaajista), epäoptimaalinen käyttäjäkokemus (58 %), sovelluskehyksen epäkypsyys (55 %) ja epäoptimaaliset vaihtoehdot käyttöliittymien luomiseen (50 %). Muita mainittuja ongelmia olivat muun muassa liian pienet yhteisöt teknologioiden ympärillä (35 %), käyttäjälaitteiden rajapintojen käytön vaikeus (32 %) ja virheenjäljityksen vaikeus (31 %). Näistä ongelmista kärsi kuitenkin alle puolet vastaajista. [22, s. 6]

Kyselytutkimuksessa [22] sovelluksen suorituskyky oli yleisin huolenaihe, ja se onkin yksi tutkituimmista eroista natiivisovellusten ja alustariippumattomien ratkaisujen välillä. Alustariippumattomien sovelluskehysten suorituskykyä tutkittiin toisessa, kesäkuussa 2020 julkaistussa Biørn-Hansen et al. tutkimuksessa [21]. Esimerkiksi sovellusten Android-pakettitiedostojen (APK) tiedostoko'oissa havaittiin suuria eroja eri teknologioiden välillä. Taulukossa 1 on esitetty sovelluskehysten väliset tulokset esimerkkisovelluksen avulla suoritetussa vertailussa.

Taulukko 1. *Pakettitiedoston kokovertailua eri sovelluskehysten välillä. [21]*

Sovelluskehys	Lähestymistapa	Ohjelmointikieli	Pakettitiedoston koko (Mt)
Ionic	Hybridi (Cordova-pohjainen)	TypeScript	10,3
React Native	Tulkattu	JavaScript	9,7
NativeScript	Tulkattu	JavaScript	30,2
Flutter	Käännetty	Dart	32,8
Natiivisovellus	Natiivi	Java	2,7

Esimerkkisovelluksen pakettitiedoston kokoa vertaillessa kääntämiseen ja omaan grafiikkamoottoriin perustuvan Flutterin APK:n tiedostokoko oli suurin (32,8 Mt), tulkkaukseen perustuvan NativeScriptin seuratessa sen jäljessä (30,2 Mt). NativeScriptin tapaan tulkkaukseen perustuva React Native tuotti kuitenkin pienemmän pakettitiedoston (9,7 Mt), joka oli samaa tasoa hybridisovelluksia tutkimuksessa edustaneen Ionicin kanssa (10,3 Mt). Parhaiten kokeesta suoriutui Javalla kirjoitettu natiivisovellus. [21 s. 3008] Vertailussa ei ollut mukana progressiivisia web-sovelluksia, mutta niiden tiedostokoko on yleisesti pieni, ja todennäköisesti verrattavissa hybridisovelluksiin.

Erilaisten tehtävien suoritusaikaa ja keskusmuistin (RAM) käyttöä mitattaessa yleinen havainto on, että kaikki sovelluskehykset suoriutuvat keskimäärin natiivisovellusta heikommin. Hybridisovelluksia edustava Ionic oli pahimmillaan suoritusaikakokeissa yli kolme kertaa natiivisovellusta hitaampi. Tietyissä tapauksissa kuitenkin yksittäiset sovelluskehykset nousivat vertailun kärkeen: NativeScript alitti suoritusajassa ja suorittimen käyttöajassa jopa natiivisovelluksen, kun taas Flutter piti muistinkulutuksen hyvin tasaisena muihin ratkaisuihin verrattuna. [21]

Biørn-Hansen et al. tutkimus [21] ei julista mitään yksittäistä ratkaisua parhaaksi, vaan painottaa projektin vaatimusten huomiointia teknologian valinnassa. Suorituskykyä vaativissa ratkaisuissa turvallisin vaihtoehto on kuitenkin natiivisovellus, ja heikoimmilla ovat todennäköisesti web-pohjaiset ratkaisut.

4.3 Käyttäjäkokemus ja käyttökohteet

Suorituskyvyn lisäksi käyttäjäkokemukseen vaikuttavat vahvasti esimerkiksi käyttöliittymän toimivuus ja johdonmukaisuus, sekä sovelluksessa tarjolla olevat ominaisuudet. Natiivisovelluksiin verrattuna alustariippumattomuus luo haasteita molempien suhteen.

Ominaisuuksien puolesta web-sovellusten suurin rajoittava tekijä on niiden puhdas pohjautuminen web-teknologioihin. Vaikka progressiivisella web-sovelluksella onkin pääsy useisiin käyttäjälaitteen ominaisuuksiin, sen on silti tyydyttävä selainrajapintoihin, jotka eivät ole yhtä kattavia kuin natiivisovelluksille tarjolla olevat. [21] Selaimet ja niiden tuki

web-rajapinnoille myös vaihtelevat alustan mukaan, jolloin kokemus voi vaihdella eri alustojen välillä. [23]

Progressiiviset web-sovellukset pystyvät hyödyntämään esimerkiksi laitteen sijaintia ja kameraa, mutta laitteen alemman tason rajapinnat ja muun muassa puhelutoiminnot eivät tällä hetkellä ole käytettävissä [24]. Muut sovellustyypit, kuten tulkattavat-, käännettävät- ja hybridisovellukset voivat valitun sovelluskehyksen mukaan hyödyntää lähes tai täysin samoja laiteominaisuuksia kuin natiivisovelluksetkin.

Hybridisovelluksia ja web-sovelluksia yhdistää käyttöliittymään liittyvät käytettävyysongelmat. HTML:ään perustuvat käyttöliittymät saattavat käyttäytyä eri tavalla kuin natiivikehitykseen perustuvat vastaavat, vaikka ne olisivatkin tyylitelty vastaamaan järjestelmän omia käyttöliittymäelementtejä. [21] Natiivisovelluksiin verrattuna web-elementeistä puuttuvat usein esimerkiksi visuaaliset tai haptiset vihjeet niitä kosketettaessa. Web- ja hybridisovellusten yleisesti heikompi suorituskyky saattaa myös luoda halvan vaikutelman. Natiivikomponentteja hyödyntävät tulkattavat sovellukset sekä käännettävät pienoisohjelmapohjaiset sovellukset muistuttavat huomattavasti enemmän natiivisovellusta, jolloin niihin liittyvät käyttäjäkokemuksen puutteet liittyvät useimmiten lähinnä suorituskykyyn.

Erillisiä natiivisovelluksia kehitettäessä voidaan ottaa paremmin huomioon eri alustojen erot käyttöliittymässä tai muissa käyttäjäkokemukseen liittyvissä suunnittelukysymyksissä. Etenkin käyttöliittymissä yhteisen koodipohjan käyttö monessa eri versiossa johtaa helposti sovelluksen ulkonäön yhtenäistymiseen eri alustojen välillä. Alustan suunnitteluohjeista riippuen se voi olla joko hyvä tai huono asia. Mikäli suurin osa natiivisovelluksista noudattaa alustalle ominaista tyyliä, mutta sovelluskehyksiä hyödyntävät sovellukset eivät, voi käyttäjien kokema laatuvaikutelma kärsiä.

Merkkinä Androidin ja iOS:n käyttöliittymien lähentymisestä voidaan pitää esimerkiksi sovelluksien navigaatioiden samankaltaistumista. Näytön alareunaan sijoitetut navigaatiopalkit sisällytettiin Googlen Material Design -suunnitteluohjeisiin vuonna 2016 [25]. Siihen asti Android-sovelluksissa suosittiin enemmän esimerkiksi sivupalkkiin sijoitettua navigaatiota. Nykyään muun muassa näytön reunoja hyödyntävät navigointieleet tekevät sivupalkkinavigaatioista käytettävyydeltään kömpelömpiä, joten paine siirtyä niistä pois on suurentunut myös tästä syystä. Kuvassa 4 on havainnollistettu Androidin ja iOS:n perinteisten navigaatiopalkkien tyyliä.

Kuva 4. *Androidin (vasemmalla) ja iOS:n perinteisten sovellusnavigaatioiden havainnollistus.*

Kuvan 4 Android-havainnollistuksessa sovelluksen navigaatioelementit on piilotettu vasemman reunan valikkoon. Applen iOS:lle kehitetyt sovellukset ovat kuitenkin käyttäneet sovelluksen alareunaan sijoitettua navigaatiopalkkia jo sen vuoden 2007 julkaisusta lähtien.

4.4 Sovelluskehysten tietoturva

Alustariippumattomien sovelluskehysten tietoturvaa käsitellään useissa tutkimuksissa. Etenkin web-teknologiat ja samalla hybridisovellusten hyödyntämä WebView-komponentti on todettu tietoturvaltaan helposti haavoittuvaiseksi, ellei asianmukaisia ohjelmointikäytäntöjä noudateta. Tyypillisesti tietoturvatutkimuksissa WebView:stä on paljastunut useita vakavia turvallisuusriskejä, jotka voisivat vaarantaa käyttäjän arkaluontoisia tietoja. [22]

WebView-komponentin ohjelmointirajapinnat mahdollistavat Android-sovelluksen ja WebView:ssä esitettävän web-sivuston välisen vuorovaikutuksen. Näiden rajapintojen kautta oli pitkään mahdollista suorittaa hyökkäyksiä, joilla saatiin pääsy käyttäjän yksityisiin tietoihin. Hyökkäyksiä voitiin suorittaa web-sivuston sisältämän JavaScript-koodin

avulla. Sen mahdollisti WebViewiin rekisteröity Java-olio, jolla oli pääsy Android-järjestelmän sisältämiin resursseihin. [26] WebView:n suoritus muunnettiin muusta järjestelmästä eristetyksi prosessiksi Android 8.0 -version julkaisun yhteydessä vuonna 2017.

Hybridisovelluksiin verrattuna WebView:n hyödyntämistä välttävät natiivisovellukset ovat tietoturvaltaan historiallisesti varmempia. Mobiilisovellusten usein laiminlyöty tietoturva on kuitenkin alustariippumattomia sovelluskehyksiä laajempi ongelma, ja koskee kaikkea sovelluskehitystä. Käytetystä teknologiasta riippumatta käyttäjädatan heikko suojaaminen on yleistä. Käyttämällä salaamattomia HTTP-pyyntöjä ja jättämällä noudattamatta hyviä ohjelmointikäytäntöjä, sovelluskehittäjät tekevät sovelluksistaan alttiita hyökkäyksille ja väärinkäytöksille. [22]

Sovelluskehysten turvallisuusongelmiin Biørn-Hansen et al. kyselytutkimuksessa [22] kiinnitti huomiota 13 vastaajaa 101:stä. Koska julkaisun data kerättiin kyselytutkimuksella, tutkimuksessa nostetaan esiin myös huoli turvallisuusnäkökohtien täydellisestä sivuuttamisesta. Toisessa toukokuussa 2017 julkaistussa mobiilisovellusten kehitystä käsittelevässä tutkimuksessa [27] todettiin 17 prosentin osuuden sovelluskehittäjistä jättävän tietoturvatestauksen kokonaan suorittamatta. Manuaalisesti testauksen ilmoitti suorittavansa 51 prosenttia vastaajista, ja loput ainakin osittain automatisoidusti.

4.5 Sovelluksen jakelu

Sovellukselle halutaan yleensä mahdollisimman suuri käyttäjäyleisö. Etenkin mobiilialustoilla sovelluskauppojen rooli korostuu. Käyttäjät ovat jo tottuneet etsimään ratkaisuja ongelmiinsa sovelluskaupoista. Esimerkiksi Amazonin ja eBayn verkkokaupoissa vietetystä ajasta 85 % tapahtuu sovelluksen kautta, kun verkkosivujen osuus on vain 15 %. [28]

Kaikki vertaillut sovelluskehykset ja teknologiat mahdollistavat sovelluksen julkaisun myös suurimmissa sovelluskaupoissa, mikäli alustan tyyliohjeistusta on noudatettu tarpeeksi tarkasti. [9] Natiiviin sovellukseen WebView:llä upotettu web-sivusto ei välttämättä täytä sovelluskauppojen vaatimuksia. Sovelluskauppojen tiedetään myös nostavan alustariippumattomia sovelluksia harvemmin niin sanotuille esittelyssä-sivuille [9], johon pääsy yleensä tuo sovellukselle suuren määrän uusia käyttäjiä.

Sovelluskaupat ovat monelle sovellukselle tärkeä mahdollisuus etenkin näkyvyyden kannalta, mutta mikäli sovelluksen voi julkaista myös sovelluskauppojen ulkopuolella, potentiaalinen käyttäjäkunta ei rajoitu mobiilialustoihin. Esimerkiksi progressiiviset web-sovellukset soveltuvat jakeluun sekä sovelluskauppojen kautta, että web-sivustona, ja siten

niillä on mahdollisuus saavuttaa laajin käyttäjäyleisö ilman koodipohjan uudelleenkirjoitusta. [12] Web-teknologioita hyödyntämällä saadaan kehitettyä yleispätevä ratkaisu lähes kaikille moderneille alustoille, kuten Androidin ja iOS:n lisäksi esimerkiksi pöytäkoneen selaimille.

Progressiivisten web-sovellusten saavutettavuustavoitteen hyödyistäkin on esitetty väitteitä. Googlen mukaan [12] esimerkiksi Twitterin web-versiosta twiitattiin 75 % enemmän, sivuja avattiin 65 % enemmän istuntoa kohden ja välittömät poistumiset vähenivät 20 %, kun PWA-sovellus korvasi vanhan mobiilisivuston. Samalla sovelluksen tiedostokoko pieneni natiivisovellukseen verrattuna 97 %.

5. YHTEENVETO

Yleinen trendi eri sovelluskehyksiä tutkiessa on, että ne eivät yllä natiivisovellusten kanssa samalle laatutasolle niin suorituskyvyn kuin käyttäjäkokemuksenkaan osalta. Etenkin web-teknologioihin liittyvissä sovelluksissa on useammin myös tietoturvaongelmia. Tämä tulos on kuitenkin hyvin odotettava, sillä alustariippumattoman kehityksen motiivina onkin monen alustan yhtäaikainen tuki, jolloin luonnollisesti sovelluksen toimintaa ei voida optimoida alustakohtaisesti kovin tarkasti.

Alustariippumattomien teknologioiden edut ovat ensisijaisesti sovelluskehitykseen liittyvissä nopeus- ja ketteryyseduissa sekä kustannussäästöissä. Sovelluskehityksen nopeutuessa itse tuote saadaan markkinoille nopeammin. Sovelluksen kehitysprosessin nopeutumisen lisäksi myös ylläpito on helpompaa yhtä yhteistä koodipohjaa kehitettäessä. Monet alustariippumattomat teknologiat mahdollistavat koodin uudelleenkäytön myös mobiilialustojen ulkopuolella, jolloin samaa koodipohjaa hyödyntäen voidaan kehittää sovelluksia myös esimerkiksi älytelevisioihin tai verkkoselaimilla hyödynnettäväksi.

Viime vuosina suurimpaan suosioon nousseet sovelluskehykset ovat kuitenkin vielä tuoreita ja jatkuvassa muutoksessa, mikä voi aiheuttaa sovelluskehittäjille odottamattomia ongelmia kehitysprosessissa. Sovelluskehysten usein heikentynyt suorituskyky ja vaihteleva tuki laitteisto-ominaisuuksille ovat joitakin syitä miksi natiivisovelluksen kehittämistä kannattaa yhä harkita.

Alustariippumattoman ratkaisun käyttäjäkokemus saattaa heikentyä sekä laskeneen suorituskyvyn, että käyttöliittymäsuunnittelun rajoitteiden kautta. Kun sovelluksia kehitetään useammalle alustalle, on vaikea ottaa huomioon kaikkien alustojen suunnitteluohjeistukset.

Alustariippumattomien sovellusten eri tyypeistä perinteisemmät hybridisovellukset vaikuttavat tekevän tilaa uusille tulkkaus- ja käännöspohjaisille ratkaisuille, kuten React Nativelle ja Flutterille. Progressiiviset web-sovellukset saattavat kuitenkin tuoda uutta nousua myös web-teknologioihin pohjautuvien sovellusten suosioon.

Stack Overflow:n vuoden 2020 kehittäjäkyselyssä [29] pidetyimpien ohjelmistokehysten listalla nousee esille useita tässäkin työssä käsiteltyjä sovelluskehyksiä. Etenkin Flutter-kehittäjät ovat käyttämäänsä sovelluskehykseen yleensä tyytyväisiä, vastaajista 68,6 prosenttia kertoo mielenkiinnosta jatkaa teknologian käyttöä. React Native ja Xamarin ovat listan keskivaiheilla 57,9 ja 45,4 prosentilla. Sen sijaan hybridisovelluskehys

Apache Cordovan käytöstä haluaisi luopua peräti 71,3 prosenttia sitä käyttävistä kehittäjistä. React Native ja Flutter herättävät mielenkiintoa myös niitä vielä kokeilemattomien kehittäjien joukossa. [29]

Alustariippumattomien sovelluskehysten käyttöä harkitessa kannattaa tutustua kyseisen teknologian tunnettuihin vahvuuksiin sekä mahdollisiin kompastuskiviin ja verrata niitä ohjelmistoprojektin vaatimuksiin. Mikään yksittäinen ratkaisu ei ole kaikissa tapauksissa paras, mutta natiivikehityksen rinnalle on tullut useita teknologioita, jotka tarjoavat suuren osan natiivisovelluksen hyödyistä pienemmillä kustannuksilla.

LÄHTEET

[1] StatCounter. Mobile operating systems' market share worldwide from January 2012 to January 2021. Statista. (2021) Luettu 6.3.2021. Saatavissa: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[2] Application Fundamentals. Android Developers. Luettu 23.2.2021. Saatavissa: https://developer.android.com/guide/components/fundamentals

[3] Get started with the NDK. Android Developers. Luettu 30.3.2021. Saatavissa: https://developer.android.com/ndk/guides

[4] Sensor Tower. Annual number of app downloads from the Google Play Store worldwide from 2016 to 2020 (in billions). Statista. 2021. Luettu 15.4.2021. Saatavissa: https://www.statista.com/statistics/734332/google-play-app-installs-per-year/

[5] Multiplatform programming. Kotlin docs. Luettu 13.3.2021. Saatavissa: https://kotlinlang.org/docs/multiplatform.html

[6] Kotlin Multiplatform Mobile. Kotlin. Luettu 13.3.2021. Saatavissa: https://kotlinlang.org/lp/mobile/

[7] Sam Dods. When Will The Web Replace Native Mobile Apps? (Part I). Luettu 15.4.2021. Saatavissa: https://medium.com/kinandcartacreated/when-will-the-web-replace-native-mobile-apps-5408c895c8a6

[8] JetBrains. Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. Statista. 2020. Luettu 11.2.2021. Saatavissa: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[9] K. Shah, H. Sinha, P. Mishra. Analysis of Cross-Platform Mobile App Development Tools. 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Mumbai, Intia, 2019.

[10] WebView. Android Developers. Luettu 15.3.2021. Saatavissa: https://developer.android.com/reference/android/webkit/WebView

[11] Ionic Framework. Luettu 30.3.2021. Saatavissa: https://ionicframework.com/docs

[12] What are Progressive Web Apps? web.dev. Luettu 18.3.2021. Saatavissa: https://web.dev/what-are-pwas/

[13] Introduction to progressive web apps. MDN Web Docs. Luettu: 15.4.2021. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction

[14] Welcoming Progressive Web Apps to Microsoft Edge and Windows 10. Windows Blogs. Luettu 15.4.2021. Saatavissa: https://blogs.windows.com/msedgedev/2018/02/06/welcoming-progressive-web-apps-edge-windows-10/

[15] Introducing a Trusted Web Activity for Android. Chromium Blog. Luettu 15.4.2021. Saatavissa: https://blog.chromium.org/2019/02/introducing-trusted-web-activity-for.html

[16] React Native. Luettu 23.2.2021. Saatavissa: https://reactnative.dev/

[17] Flutter project. GitHub. Luettu 22.3.2021. Saatavissa: https://github.com/flutter/flutter

[18] React Native project. GitHub. Luettu 22.3.2021. Saatavissa: https://github.com/facebook/react-native

[19] Flutter architectural overview. Flutter. Luettu 8.3.2021. Saatavissa: https://flutter.dev/docs/resources/architectural-overview

[20] Zuckerberg's Biggest Mistake? 'Betting on HTML5'. Mashable. Luettu: 12.3.2021. Saatavissa: https://mashable.com/2012/09/11/html5-biggest-mistake

[21] Biørn-Hansen A, Rieger C, Grønli T-M, Majchrzak TA, Ghinea G. An empirical investigation of performance overhead in cross-platform mobile development frameworks. Empirical software engineering: an international journal, 2020. 25:2997–3040.

[22] Biørn-Hansen A, Tor-Morten Grønli, Ghinea G, Alouneh S. An Empirical Study of Cross-Platform Mobile Development in Industry. Wireless Communications & Mobile Computing (Online) 2019.

[23] What are Progressive Web Apps? Ionic Blog. Luettu 16.4.2021. Saatavissa: https://blog.ionicframework.com/what-is-a-progressive-web-app/

[24] Progressive Web App (PWA) and Hardware Access. SimiCart Blog. Luettu 3.4.2021. Saatavissa: https://www.simicart.com/blog/pwa-hardware-access/

[25] Google updates Material Design spec with animated bottom bars for navigation. 9to5Google. Luettu 15.4.2021. Saatavissa: https://9to5google.com/2016/03/15/materia-design-apps-bottom-navigation-bars/

[26] J. Yu and T. Yamauchi. Access Control to Prevent Attacks Exploiting Vulnerabilities of WebView in Android OS. 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 1628-1633.

[27] R. Francese, C. Gravino, M. Risi, G. Scanniello, G. Tortora. Mobile App Development and Management: Results from a Qualitative Investigation. 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Buenos Aires, Argentina, 2017, pp. 133-143.

[28] eMarketer. Share of time spent on retailers' mobile apps vs. websites among internet users in the United States as of March 2019. Statista. 2019. Luettu 31.3.2021. Saatavissa: https://www.statista.com/statistics/1019768/us-retailers-apps-vs-websites-time-distribution/

[29] 2020 Developer Survey. Stack Overflow. Luettu 22.4.2021. Saatavissa: https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-other-frameworks-libraries-and-tools-dreaded3