Joel Hietala

Natiivisovellusten ja progressiivisten web-sovellusten erot ja yhtäläisyydet sovellustuotannon ja käyttäjän näkökulmista

Tietotekniikan kandidaatintutkielma

14. toukokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joel Hietala

Yhteystiedot: jokrhiet@student.jyu.fi

Ohjaaja: Tuomo Rossi

Työn nimi: Natiivisovellusten ja progressiivisten web-sovellusten erot ja yhtäläisyydet sovellustuotannon ja käyttäjän näkökulmista

Title in English: The differences and similarities between native apps and progressive web apps in software development and user experience

Työ: Kandidaatintutkielma

Sivumäärä: 28+0

Tiivistelmä: Tutkielmassa vertaillaan natiivisovelluksia ja progressiivisia web-sovelluksia sovellustuotannon ja käyttäjän näkökulmista iOS-mobiililaitteilla. Kirjallisuuskatsauksen tavoitteena on käsitellä erilaisia syitä valita natiivisovelluksen ja web-sovelluksen väliltä sovelluksen suunnitteluvaiheessa. Tutkimuksen pohjalta voidaan todeta, että natiivisovellukset ovat pääsääntöisesti tehokkaampia ja monipuolisempia, kun taas progressiiviset web-sovellukset ovat joustavampia ja kustannustehokkaampia.

Avainsanat: natiivisovellus, progressiivinen web-sovellus, PWA, sovelluskehitys, käyttökokemus

Abstract: The thesis examines native applications and progressive web applications on the iOS mobile platform from both the developers' and the users' points of view. The literature review aims to explore the various reasons for a developer to decide on one of the aforementioned ways to develop an app. The research suggests that native apps are usually faster and more capable, whereas progressive web apps are more flexible and cost-effective.

Keywords: native app, progressive web app, PWA, software development, user experience

Joel Hietala

Kuviot

Sisällys

1 Johdanto

Mobiilisovellusten ala on valtava. CNBC (2021) arvioi, että Applen App Storen bruttomyynti ylitti 64 miljardia dollaria vuonna 2020, ja että määrä olisi yhä kasvussa. Sovellusten tekniseen toteutukseen on kuitenkin useita tapoja. Tässä tutkielmassa käsitellään natiivisovellusten ja progressiivisten web-sovellusten välisiä eroja iOS-mobiilikäyttöjärjestelmällä.

Natiivisovelluksilla viitataan sovelluksiin, jotka pohjautuvat jollekin tietylle käyttöjärjestelmälle suunnattuun käännettyyn (engl. *compiled*) koodiin, kun taas progressiivisilla web-sovelluksilla tarkoitetaan web-teknologioilla toteutettuja ja verkkoselaimen sisällä suoritettuja sovelluksia (Sharma 2017; Charland ja Leroux 2011).

Tämän tutkielman päähypoteesi on, että natiivisovellukset ovat käyttökokemukseltaan, ominaisuuksiltaan ja suorituskyvyltään parempia iOS-käyttöjärjestelmällä, kun taas progressiiviset web-sovellukset ovat kehittäjän näkökulmasta edullisempia ja nopeampia toteuttaa.

Tutkielmassa tarkastellaan ensin natiivisovellusten historiaa, toteutusta ja ominaispiirteitä iOS-käyttöjärjestelmällä. Seuraavaksi käsitellään progressiivisia web-sovelluksia samoista näkökulmista. Luvuissa 4 ja 5 pohditaan toteutustapojen eroja kehittäjän ja käyttäjän näkökulmista, minkä jälkeen käsitellään muita vaihtoehtoja. Lopuksi luvussa 7 on yhteenveto tutkimuksen tuloksista ja lyhyt pohdinta jatkotutkimuksen mahdollisuuksista.

2 Natiivisovellukset

Kaikki App Storesta löytyvät sovellukset pohjautuvat natiivikoodiin, eli joko Objective-C:hen tai Swiftiin. Osa sovelluksista, kuten niin kutsutut hybridisovellukset, on toteutettu myös muita kieliä käyttämällä; niitä käsitellään tarkemmin luvussa 6.1. Tässä luvussa käsitellään yksinomaan natiivikoodilla rakennettuja sovelluksia, kuten Firefoxin (Github 2015) ja WordPressin (Github 2008) iOS-versiot.

2.1 Historiaa

Kun ensimmäinen iPhone ja iOS[1] julkistettiin vuonna 2007, kolmannen osapuolen kehittäjien sovellukset eivät olleet minkäänlainen myyntivaltti, eikä ulkopuolisia sovelluksia edes mainittu lehdistötiedotteessa (Apple Inc. 2007). Kehittäjille kerrottiin julkistustilaisuudessa, ettei erillistä ohjelmistokehityspakettia (SDK) tarvita, vaan sovellukset toteutetaan web-teknologioilla verkkoselaimen kautta käytettäviksi – eli alun perin iPhonelle oli tarkoituksenakin tehdä vain web-sovelluksia, joskin silloiset web-sovellukset olivat hyvin alkeellisia nykypäivän progressiivisiin web-sovelluksiin verrattuna. Päätös herätti paljon keskustelua, ja web-sovellusten eron natiivisovelluksiin huomasi selvästi: ”Useimmat web-sovellukset ovat melko vaatimattomia verrattuina natiivin Google Mapsin ja yksinkertaisen mutta tyydyttävän sääwidgetin kaltaisiin sovelluksiin. Ja mitä tukevampia näistä web-sovelluksista tulee, sen raskaampia ne ovat iPhonen yrittäessä ladata ne EDGE-verkon kautta kun Wi-Fiä ei ole saatavilla” (Lam 2007, käännös tekijän). Jo seuraavana vuonna Apple pyörsi päätöksensä ja julkisti ohjelmistokehityspaketin iPhonelle sekä App Store -sovelluskauppansa, joka avattiin heinäkuussa 2008. Ensimmäisenä viikonloppunaan App Storen 800 saatavilla ollutta natiivisovellusta ladattiin yli 10 miljoonaa kertaa (Apple Inc. 2008).

2.2 Tekninen toteutus

Natiivisovellukset iOS-käyttöjärjestelmälle kirjoitettiin alun perin yksinomaan 1980-luvulla kehitetyllä Objective-C-kielellä. Vuonna 2014 Apple esitteli Swift-kielen, ”voimakkaan ja

1. vuoteen 2010 asti iPhone OS

intuitiivisen ohjelmointikielen” jonka syntaksi on ”ytimekäs mutta ilmaisuvoimainen” (Apple Inc. 2014b, käännös tekijän). Nykyään natiivin iOS-sovelluksen voi kirjoittaa joko Objective-C:llä, Swiftillä tai niiden yhdistelmällä. Swift-ohjelmat ovat kuitenkin Applen mukaan nopeampia ja turvallisempia, ja kielessä on lisäksi hyvä muistinhallinta automaattisen viitteidenlaskun ansiosta (Apple Inc. 2018a). Swift on myös jatkuvan kehitystyön kohde, sillä siihen on julkaistu suuria päivityksiä lähes jokaisena vuonna sen julkistuksen jälkeen (Apple Inc. 2021).

Xcode-sovellus on Applen kehittämä ohjelmointiympäristö, jolla voi toteuttaa muun muassa MacOS- ja iOS-sovelluksia. Xcode on saatavilla ainostaan MacOSille, joten natiiveja iOS-sovelluksia ei ole mahdollista kehittää ilman Mac-tietokonetta; vaikka koodia voikin toki kirjoittaa muilla käyttöjärjestelmillä kuin MacOSilla, koodia ei voi kääntää tai allekirjoittaa ilman Xcode-sovellusta. Xcoden mukana tulevat myös iOS-simulaattorit ja kokoelma hyödyllisiä virheenkorjauksen ja suorituskyvyn analysoinnin työkaluja.

2.3 Saatavuus

Jotta iOS-sovelluksen voi julkaista App Storessa, kehittäjä tarvitsee Apple Developer -jäsenyyden, joka maksaa 99 euroa vuodessa. Tämäkään ei vielä takaa sovelluskauppaan pääsyä, sillä Apple tarkistaa jokaisen sovelluksen ja tekee päätöksen siitä, soveltuuko se App Storeen vai ei. Tätä varten on joukko erilaisia arviointikriteereitä, jotka on kerätty App Store Review Guidelines -oppaaseen (Apple Inc. 2014a). Tietyntyyppiselle sisällölle, kuten graafiselle seksuaaliselle sisällölle, väkivallalle ja vihapuheelle, sovelluskaupassa on nollatoleranssi, kun taas toisia tarkastellaan tapauskohtaisesti.

Sovelluksen sisältö ei ole ainoa arviointikriteeri, vaan myös muun muassa sen käyttöliittymää arvioidaan. Applen *Human Interface Guidelines* -oppaassa eritellään käyttöliittymää ja käyttökokemusta koskevat vaatimukset ja suositukset (Apple Inc. 2018b). Oppaassa käsitellään muun muassa sisällön tärkeyttä, fonttien ja värien käyttöä, sovelluksen sisäistä arkkitehtuuria, erilaisia näkymiä, animaatioita, laajennuksia ja paljon muuta.

2.4 Käyttäjän näkökulmasta

Useimmille käyttäjille ainoa virallinen[2] tapa asentaa natiivisovelluksia on App Storen kautta.

Yhtäältä tämä on hyvin käyttäjäystävällinen ratkaisu. Kun kaikki sovellukset löytyvät yhdestä paikasta, on niiden lataaminen helppoa. Tämä voi olla myös brändikysymys – Applen logolla varustetut *Download on the App Store* -merkit ovat tuttuja monesta mainoksesta.

Toisaalta taas tämä rajoittaa käyttäjän valinnanvaraa. Laitteeseen saa ladattua vain ne sovellukset, jotka Apple sallii, ja kuten mainittu, Applella on hyvin tarkat kriteerit. Esimerkiksi Android-laitteilla on suurempi valikoima saatavilla olevia sovelluksia, mutta löyhempien laatuvaatimusten vaikutukset saattavat myös näkyä sovellusten käyttökokemuksessa ja tietoturvassa.

Applen *Human Interface Guidelines* -ohje kertoo, miten sovellukset täyttävät korkeat laatua ja toiminnallisuutta koskevat odotukset (Apple Inc. 2018b). Tärkeää on muun muassa selkeys: tekstin pitää olla luettavaa, kuvakkeiden selkeitä ja koristeiden hienovaraisia. Ohjeessa korostetaan myös muun muassa käyttöliittymän yhdenmukaisuutta, palautetta, liikkeen sulavuutta ja käyttäjän kontrollia, ja sovellusten käyttöliittymienkin toteutukseen on hyvin konkreettisia ohjeita. Näiden suuntaviivojen ansiosta iOS-sovellukset ovat monella tapaa yhdenmukaisia ja käyttäjän kannalta helppokäyttöisiä.

Lisäksi hyvin toteutettujen natiivisovellusten suorituskyky iOS-laitteilla on erinomainen, mikä edesauttaa miellyttävää käyttökokemusta.

2. Joihinkin laitteisiin ja ohjelmistoversioihin on teknisesti mahdollista asentaa sovelluksia myös epäviralisesti sivukuormituksen (engl. *sideloading*) tai ns. *jailbreakauksen* kautta, mutta nämä tavat ovat iOSin käyttösopimuksen vastaisia (Apple Inc. 2020).

3 Progressiiviset web-sovellukset

Natiivisovellukset ovat tehokkaita ja oikein toteutettuna hyvin miellyttäviä käyttää. Niiden kehittäminen vaatii kuitenkin erityisosaamista, aikaa ja usein myös rahoitusta. Tässä luvussa käsitellään varteenotettavaa vaihtoehtoa niille, progressiivisia web-sovelluksia.

Progressiiviset web-sovellukset ovat sovelluksen kaltaisia verkkosivustoja, jotka voivat hyödyntää laitteen ominaisuuksia huomattavasti monipuolisemmin kuin tavalliset sivustot. Osmani (2015, käännös tekijän) kuvailee tällaisten web-sovellusten olevan progressiivisia, responsiivisia, internet-yhteydestä riippumattomia, sovelluksen kaltaisia, tuoreita, turvallisia, löydettävissä olevia, uudelleenkäytettäviä, asennettavia ja linkitettäviä. Esimerkiksi Starbucks, Pinterest ja Uber tarjoavat progressiivisen web-sovelluksen, kuten myös 2048-peli.

3.1 Historiaa

Alkuperäisen iPhonen julkistustilaisuudessa tammikuussa 2007 kerrottiin, että kolmannen osapuolen sovellukset iPhonelle toteutettaisiin web-sovellusten muodossa. Web-sovellukset olivat kuitenkin natiivisovelluksiin verrattuna toiminnallisuudeltaan hyvin rajoitettuja (Puder, Tillmann ja Moskal 2014), joten jo saman vuoden syksyllä Apple ilmoitti ohjelmistokehityspaketin (SDK) natiivisovellusten kehittämiseksi iPhonelle olevan tulossa.

Kun älypuhelimet alkoivat yleistyä, myös verkkosivustojen täytyi mukautua pienemmille näytöille. Web-teknologia kehittyi natiivisovellusten rinnalla, ja sivustot muuttuivat entistä dynaamisemmiksi, interaktiivisemmiksi ja responsiivisemmiksi. Ei kuitenkaan ollut vielä mitään yleistä määrittelyä sille, millaisia web-sovellusten tulisi olla.

Vuonna 2015 eräs Google Chromen kehittäjistä kirjoitti blog-julkaisun, jossa esitteli progressiivisten web-sovellusten käsitteen ja kertoi ajatuksiaan konseptiin liittyen (Russell 2015, käännös tekijän): ”Näitä sovelluksia ei paketoida tai jaeta sovelluskaupoissa, ne ovat vain verkkosivustoja jotka söivät juuri oikeita vitamiineja. Ne lisäävät uusia toimintoja, kuten ovat päällimmäisenä moniajonäkymässä, kotinäytöllä ja ilmoituskeskuksessa. Käyttäjien ei tarvitse tehdä suuria valintoja etukäteen, eivätkä he suostu mihinkään vaaralliseen vain klik-

kaamalla linkkiä.”

Tämän myötä progressiivisten web-sovellusten suosio ja tunnettuus on noussut tasaisesti. Google ja Microsoft ovat alkaneet tukea näiden uusia ominaisuuksia nopeasti, osin jopa yhteistyötä tehden (Himango 2020), kun taas Applen tahti on ollut hitaampi. Vuonna 2018 Apple toi iOS 11.3-päivityksessä tuen *Service Worker* -rajapinnalle, joka mahdollistaa progressiivisten web-sovellusten päivityksen taustalla, mutta rajoituksia kuitenkin on yhä; näistä lisää luvussa 4.3.

3.2 Tekninen toteutus

Progressiiviset web-sovellukset toteutetaan web-teknologioilla, kuten JavaScript-pohjaista React-sovelluskehystä käyttäen (Biørn-Hansen, Majchrzak ja Grønli 2017). Laitteen muistiin tallennettu runko latautuu välittömästi sovelluksen avautuessa, ja siihen ladataan sisältö dynaamisesti verkosta (Osmani 2015).

Pohjimmiltaan progressiiviset web-sovellukset ovat siis vain verkkosivustoja, jotka on toteutettu tietyllä tavalla ja mobiiliyhteensopivasti. Niihin kuitenkin kohdistuu kolme konkreettista vaatimusta, jotta selaimet tunnistavat ne progressiivisiksi web-sovelluksiksi, kuten kyseisiä vaatimuksia Google Chromeen suunnitellut Russell (2016) selittää blog-julkaisussaan:

1. Tietoturvasyistä sovelluksen täytyy toimia TLS-salausprotokollalla, eli kaikkien sovelluksen osien tulee latautua suojatuista HTTPS-lähteistä ilman sekoitettua aktiivista sisältöä (engl. *mixed active content*). Tällä viitataan suojaamattomaan sisältöön, jolla on pääsy sivun dokumenttioliomalliin (DOM) – käytännössä esimerkiksi JavaScript-tiedostoon, joka ladataan HTTP-yhteyden kautta – minkä avulla hyökkääjä voisi muokata salatun sivun sisältöä tai päästä käsiksi käyttäjän dataan (MDN 2021a).
2. Sovelluksen täytyy olla ladattavissa ilman verkkoyhteyttä, vaikka saatavilla oleva sisältö olisikin vain sivu, joka ilmoittaa verkkoyhteyden puutteesta. Tämän vuoksi progressiiviset web-sovellukset vaativat *Service Workerin*, offline-toiminnallisuudesta ja sovelluksen taustasynkronoinnista vastaavan JavaScript-ohjelman. Joillakin käyttöjärjestelmillä tämä mahdollistaa myös esimerkiksi push-ilmoitusten käytön, mutta iOSilla tämä ei ole ainakaan vielä mahdollista.

3. Sovelluksen tulee viitata *Web App Manifestiin*, eli JSON-tiedostoon, jossa määritellään muun muassa sovelluksen nimi, sen käynnistysosoite sekä kuvake (MDN 2021b, 2021c).

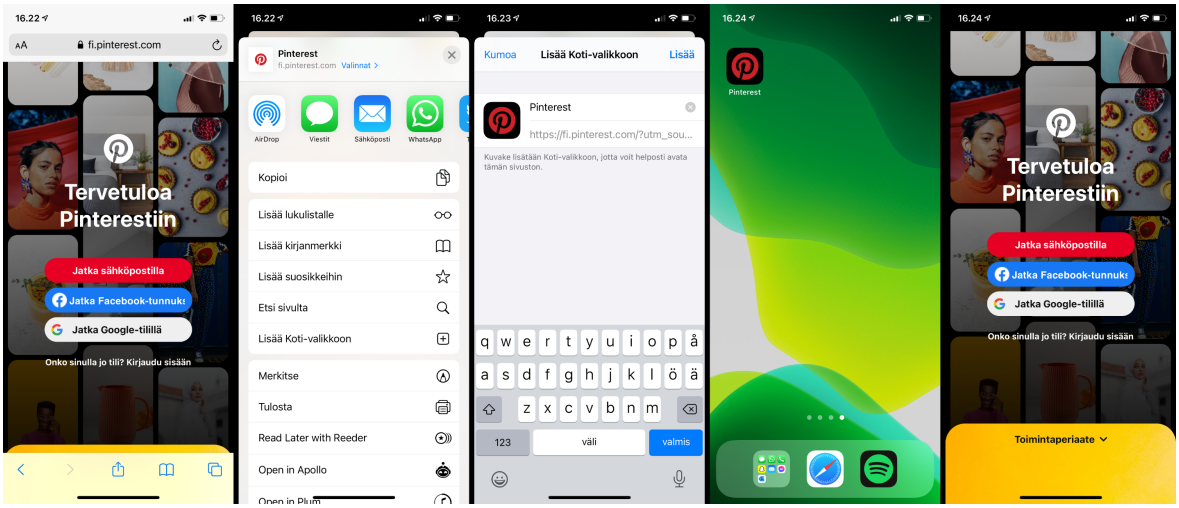
Näiden ehdottomien teknisten vaatimusten lisäksi Russell (2016) kirjoittaa, että progressiivisten web-sovelluksien olisi hyvä myös olla suunniteltu mobiiliystävällisesti, latautua lähes välittömästi, toimia laajasti eri laitteilla ja selaimilla, ja niiden animaatioiden tulisi olla sulavia. Kaikkein parhaiden progressiivisten web-sovellusten hän määrittelee myös toimivan mobiililaitteiden lisäksi muillakin laitteilla, kuten tietokoneilla; omaavan korkealaatuisen käyttöliittymän ja -kokemuksen, jotka saavat sovelluksen käyttämisen tuntumaan hyvin vuorovaikutteiselta; kertovan tehokkaasti, mitä käyttäjä voi tehdä ilman verkkoyhteyttä; harkitsevan push-ilmoitusten ja taustasynkronoinnin käyttöä; ja kysyvän lupaa esimerkiksi sijainnin käyttämiseen oikeassa kontekstissa.

3.3 Käyttäjän näkökulmasta

Progressiiviset web-sovellukset eivät ole saatavilla sovelluskaupoista, vaan ne asennetaan laitteeseen verkkosivuston kautta. Luvussa 5.1 käsitellään tarkemmin, millaisia vaikutuksia tällä on käyttökokemukseen.

Käytännössä iOS-laitteella progressiivisen web-sovelluksen asentaminen tapahtuu[1] avaamalla progressiivisen web-sovelluksen verkkosivu, koskettamalla selaimen alalaidassa olevaa jakopainiketta, rullaamalla alaspäin, koskettamalla *Lisää Koti-valikkoon* -painiketta ja koskettamalla Lisää-painiketta (kuvio 1). Tämän jälkeen laitteen Koti-valikosta löytyy kyseisen progressiivisen web-sovelluksen kuvake, jota koskettamalla sovellus avautuu natiivisovelluksen kaltaisesti koko näytölle ilman verkkoselaimen käyttöliittymäelementtejä.

1. iOS 14.4.2:ssa, uusimmassa tämän tutkielman kirjoittamisen aikana saatavilla olevassa ohjelmistoversiossa



Kuvio 1. Progressiivisen web-sovelluksen asentaminen ja avaaminen iOS-laitteella

4 Kehittäjän näkökulmasta

Natiivisovellukset ja progressiiviset web-sovellukset ovat luonnollisesti teknisesti hyvin erilaiset. Tässä luvussa käsitellään näitä eroja ja niiden merkitystä kehittäjien kannalta.

4.1 Tekninen toteutus ja alustariippumattomuus

Suurin ero natiivisovellusten ja progressiivisten web-sovellusten välillä on eittämättä teknisessä toteutustavassa.

Natiivisovellukset vaativat jokaisella käyttöjärjestelmällä oman kielensä, ohjelmistokehyksensä ja jopa ohjelmointiparadigmansa. Siinä missä iOSilla sovellus toteutetaan Objective-C:llä tai Swiftillä ja käyttöliittymä esimerkiksi Xcoden *storyboards*-näkymillä tai SwiftUI:lla, Androidilla kirjoitetaan Javaa tai Kotlinia eikä iOSia vastaavaa ominaisuutta käyttöliittymän luomiseen ole (Lamhaddab, Lachgar ja Elbaamrani 2019).

Progressiiviset web-sovellukset puolestaan toimivat sellaisenaan useimmilla käyttöjärjestelmillä, jolloin käytännössä tarvitaan vain yksi koodikanta. Tällainen alustariippumattomuus on luonnollisesti suuri etu kehittäjän kannalta.

Toinen näkökulma on kehittäjän omat taidot. Kokeneella web-kehittäjällä on varmasti paremmat valmiudet ryhtyä progressiivisen web-sovelluksen tekoon, kun taas harjaantuneelle olio-ohjelmoijalle Objective-C tai Swift saattaa olla helpompi vaihtoehto oppia syntaksin samantapaisuuden vuoksi.

Natiivisovelluksiin on myös luonnollisesti helpompi toteuttaa käyttöliittymä, joka on samankaltainen muiden käyttöjärjestelmälle suunnattujen natiivisovellusten kanssa. Etenkin Applen vuonna 2019 esittelemä SwiftUI-sovelluskehys helpottaa iOSille sopivan käyttöliittymän tekemistä huomattavasti. Progressiivisilla web-sovelluksilla sen sijaan ei ole samaan tapaan natiivin käyttöliittymän elementtejä käytettävissä, joten ne voivat erottua kielteisesti usein yhdenmukaisista natiivisovelluksista.

4.2 Ylläpito

Progressiivisten web-sovellusten ylläpito on kiistämättä helpompaa kuin natiivisovellusten. Ensinnäkin web-sovelluksen pystyy päivittämään välittömästi, kun taas natiivisovelluksella se vaatii useimmiten[1] uuden version lähettämisen Applen hyväksyttäväksi, mihin voi mennä muutama päiväkin. Toisekseen iOSin kaltaisella käyttöjärjestelmällä, jonka ohjelmistokehyksiin ja Swiftin tapauksessa jopa ohjelmointikieleen tulee vuosittain muutoksia, voi sovellus vaatia suuriakin muutoksia, jotta se edes toimisi uudella käyttöjärjestelmäversiolla tai uusilla laitteilla. Etenkin yksinkertaisemmat progressiiviset web-sovellukset puolestaan toimivat usein vähäiselläkin ylläpidolla, vaikka tietoturva- ja käytettävyyssyistä onkin toki tärkeää pitää myös progressiivinen web-sovellus teknisesti ajan tasalla.

4.3 Progressiivisten web-sovellusten rajoitukset

Progressiivisten web-sovellusten tuki iOSilla on alusta asti ollut heikompi kuin muilla alustoilla. Nykyään käytettävissä on suuri osa natiivisovellusten tukemista ominaisuuksista, kuten sijaintipalvelut, kamera, gyroskooppi ja kiihtyvyysanturi, mutta rajoituksia kuitenkin on yhä – muun muassa paikallista tallennustilaa on käytettävissä vain 50 megatavua, push-ilmoitukset eivät ole käytettävissä, eikä tukea Face ID:lle tai Sirille ole. Progressiiviset web-sovellukset eivät myöskään välttämättä pysty hyödyntämään samoja kolmannen osapuolen rajapintoja kuin natiivisovellukset.

4.4 Saatavuus

Natiivisovelluksia voi iOSilla ladata vain App Storen kautta, ja App Storen kautta voi ladata vain natiivisovelluksia[2]. Kehittäjän kannalta tämä voi rajoittaa merkittävästi sovelluksen löydettävyyttä, kuten Instapaper- ja Overcast-sovellusten tekijä, tunnettu iOS-kehittäjä Marco Arment, kommentoi: ”[Progressiivisen web-sovelluksen] suurin ongelma olisi, ettei kukaan löytäisi sitä. Kun ihmiset etsivät uutta sovellusta, josta ovat kuulleet, he eivät etsi sitä verkosta – he etsivät sitä App Storesta. He hakisivat hakusanalla "overcast", eivät löytäisi

1. sovelluksen lataamaa etäsisältöä pystyy toki muokkaamaan ilman sovelluksen päivittämistä
2. poikkeuksena tähän sääntöön hybridisovellukset, joista lisää luvussa 6.1

sitä, eivätkä koskaan etsisi muualta.” (Arment 2021, käännös tekijän)

Kehittäjien on nykyään mahdollista saada progressiiviset web-sovelluksensa joihinkin sovelluskauppoihin, kuten Google Playhin (McLachlan ja Buckley 2020). Tämä voi auttaa niitä saavuttamaan suuremman yleisön. App Storessa tätä mahdollisuutta ei kuitenkaan ainakaan vielä ole.

4.4.1 App Storen väitetty monopoliasema

Apple on vastaanottanut kritiikkiä päätöksestään rajoittaa natiivien iOS-sovellusten saatavuus yksinomaan App Storeen siinä, missä esimerkiksi Android-laitteilla on tarjolla Google Play -sovelluskaupan lisäksi myös muun muassa Amazon Appstore ja Samsungin puhelimilla Galaxy Store. Jos kehittäjä haluaa julkaista natiivin iOS-sovelluksen, sen täytyy täyttää App Storen vaatimukset.

Viimeisin ja kenties tunnetuin aiheeseen liittyvä kiista kärjistyi elokuussa 2020, kun Epic Games -peliyhtiö haastoi Applen oikeuteen väittäen App Storea monopoliksi (Statt 2020). Apple on kommentoinut väitettä Australian kilpailu- ja kuluttajakomissiolle argumentoiden käyttäjillä olevan muitakin väyliä digitaalisen sisällön ostamiseen ja kuluttamiseen kuin App Store, esimerkiksi tavalliset verkkosivut ja progressiiviset web-sovellukset (Apple Pty Ltd 2021).

4.5 Sisältö

Kuten mainittu, Apple tarkistaa kaikki App Storeen lähetettävät sovellukset. Näin ollen natiivisovelluksia koskee huomattavasti suurempi määrä ulkoasullisia ja sisällöllisiä rajoituksia kuin progressiivisia web-sovelluksia. Esimerkiksi väkivaltainen tai seksuaalinen sisältö, jotka ovat kiellettyjä App Storesta, voidaan kuitenkin julkaista progressiivisen web-sovelluksen muodossa.

4.6 Monetisointi

Koska progressiiviset web-sovellukset eivät ole App Storessa, käyttäjältä ei voi pyytää maksua sovelluksesta samalla tavalla. Myöskään sovelluksen sisäiset ostokset (IAP, *In-App Purchase*) eivät Applen protokollia käyttäen onnistu. Kehittäjän kannalta tämä on suuri takaisku, sillä App Store tarjoaa niin kehittäjän kuin käyttäjänkin kannalta hyvin helpon tavan pyytää ja vastaanottaa rahaa vastineeksi sovelluksesta ja sen sisällöstä. Toisaalta Apple myös veloittaa 30 prosentin[3] suuruisen provision App Storen kautta tehtävistä myynneistä, mikä ei tietenkään koske progressiivisia web-sovelluksia. On kuitenkin syytä olettaa, että käyttäjät ovat taipuvaisempia ostamaan natiivisovelluksen tai sen sisäisen ostoksen kuin vastaavan progressiivisen web-sovelluksen, sillä monen App Store -käyttäjän tiliin on jo yhdistetty maksutiedot, jolloin maksaminen onnistuu parissa sekunnissa.

4.7 Kustannukset

Natiivisovelluksen ensimmäisen version toteutuuttaminen maksaa Clutch-analytiikkayhtiön kyselytutkimuksen mukaan keskimäärin noin 25 000 dollarista pitkälti yli 100 000 dollarin (Craigmile 2015). Progressiivisesta web-sovelluksesta ei löytynyt vastaavaa tutkimusta, mutta muutamien muiden lähteiden mukaan sen toteuttaminen olisi merkittävästi edullisempaa. Erityisen ymmärrettävää tämä on sovelluksissa, jotka toteutetaan usealle eri käyttöjärjestelmälle, koska tällaiset natiivisovellukset vaativat luonnollisesti monta eri koodikantaa.

Kuten mainittu, Applen kehittäjäjäsenyyden hinta on 99 euroa vuodessa, joten natiivisovelluksen pitäminen App Storessa maksaa vähintään sen verran myös itsenäisille kehittäjille. Progressiivisilla web-sovelluksilla ei ole vastaavia kuluja, mutta käytännössä kuitenkin ainakin domain ja palvelin voivat helposti maksaa vuositasolla yhtä paljon.

3. 15 prosentin, jos kehittäjä kuuluu *App Store Small Business Programiin* eivätkä kehittäjän tulot App Storessa ylitä miljoonaa dollaria vuodessa

5 Käyttökokemus

Seuraavaksi käsitellään natiivisovellusten ja progressiivisten web-sovellusten eroja käyttäjän kannalta.

5.1 Asentaminen

Natiivisovellukset asennetaan iOS-käyttöjärjestelmällä App Storen kautta. Tämä on koskenut kaikkia natiivisovelluksia niin kauan kuin natiivisovelluksia on ollut saatavilla iOSilla, pitkälti yli 12 vuotta. Voidaan olettaa, että useimmilla käyttäjillä on tässä vaiheessa selvä käsitys siitä, kuinka sovelluksen voi ladata App Storesta. App Storen valttikorttina voidaan pitää myös sitä, kuinka helppoa sen kautta on löytää uusia sovelluksia – heti sen etusivulla on suuria, houkuttelevia mainoksia Applen valikoimista sovelluksista ja peleistä, ja muilta välilehdiltä löytyy vielä lisää. Kaupasta löytyy myös suosituimpien ilmaisten ja maksullisten sovellusten ja pelien listat, ja hakutoiminnolla löytää loput App Storen valtavasta sovellusvalikoimasta.

Progressiiviset web-sovellukset eivät voi nojata samalla tavalla App Storen infrastruktuuriin. Koska sovellus ei ole App Storessa, sen löytäminen voi olla hankalampaa: osa käyttäjistä etsii sovelluksia vain App Storen haun avulla, eivätkä välttämättä edes tiedä progressiivisten web-sovellusten mahdollisuuksista.

Toisaalta sovelluksen asentaminen suoraan verkkoselaimesta voi olla helpompi vaihtoehto joillekin käyttäjille. Etenkin käyttäjät, jotka vierailevat muutenkin palvelun verkkosivustolla, voivat yksinkertaisesti lisätä sovelluksen laitteeseensa Koti-valikkoon suoraan verkkosivustolta; heille tämä lienee nopeampaa kuin erillinen App Store -vierailu.

5.2 Suorituskyky

Teorian tasolla on selvää, että natiivikoodi on JavaScriptin kaltaista tulkittua koodia nopeampaa. JavaScriptiä hyödyntävän sovelluksen suorituskyky kuitenkin riippuu sen toteutustavasta (Selakovic ja Pradel 2016), joten käytännössä natiivisovellusten ja progressiivisten web-

sovellusten väliset suorituskykyerot voivat vaihdella merkittävästikin.

Empiirisiä tutkimuksia natiivisovellusten ja progressiivisten web-sovellusten suorituskykyeroista on valitettavan vähän, ja niistäkin suurin osa on toteutettu Android-käyttöjärjestelmällä, jolloin ne eivät varsinaisesti kuulu tämän tutkielman aihepiiriin – käyttöjärjestelmien välillä voi olla suuria eroja suorituskyvyn optimoinnissa. Etenkin progressiivisten web-sovellusten teknologia myös kehittyy jatkuvasti, ja eri käyttöjärjestelmien tuki sille vaihtelee ja muuttuu. Tämän vuoksi myös olemassaolevien tutkimusten tulokset eivät enää välttämättä ole täysin valideja.

Willocx, Vossaert ja Naessens (2016) tutkivat erilaisten sovelluskehitysstrategioiden suorituskykyä vertailemalla natiivisovelluksen ja kymmenellä eri työkalulla toteutetun vastaavanlaisen web- tai hybridisovelluksen suorituskykyä kahdella eri iOS-laitteella, tehokkaammalla ja edullisemmalla mallilla. Kaikilla JavaScript-ohjelmistokehyksillä toteutetut sovellukset käynnistyivät vähintään kaksi ja enintään lähes viisi kertaa hitaammin kuin natiivisovellus. Sovelluksen sisäisen navigaation nopeudessa oli mielenkiintoisia eroja: vaikka natiivisovellus olikin nopeampi sekä uuden sivun avaamisessa että aloitussivulle palaamisessa kuin useimmat web-sovellukset, kaksi JavaScript-ohjelmistokehystä suoriutui molemmista testeistä nopeammin. Testien välillä kyseessä oli kuitenkin kaksi eri kehystä, joten keskimäärin natiivisovellus suoriutui selvästi parhaiten. Muistinkulutus oli kuitenkin erikoinen poikkeus, koska tehokkaampi malli kulutti enemmän muistia kuin mikään JavaScript-kehyksistä ja edullisempi malli puolestaan vähemmän. Tämä ei kuitenkaan välttämättä kieli suorituskykyerosta, vaan siitä, että iOS yksinkertaisesti allokoi natiivisovellukselle enemmän muistia tehokkaammalla laitteella. Ainakaan tällaisessa mittakaavassa suurempi muistinkulutus ei myöskään ole ongelma, etenkin kun iOSin moniajokyky on yhä varsin rajoitettu.

5.3 Ulkoasu

Progressiivisten web-sovellusten käyttöliittymä on yleensä sama joka laitteella, kun taas natiivisovellusten tulisi seurata kunkin käyttöjärjestelmän omaa suunnitteluparadigmaa. Natiivisovellukset ovat siis ihannetapauksessa keskenään käyttöliittymältään yhdenmukaisia, minkä ansiosta niiden käyttökokemusta voi pitää progressiivisiin web-sovelluksiin verrattu-

na parempana. Tämä ei kuitenkaan tarkoita sitä, että kaikissa natiivisovelluksissa olisi hyvä tai intuitiivinen käyttöliittymä, tai kaikissa progressiivisissa web-sovelluksissa olisi huono, vaan se riippuu ensisijaisesti kehittäjästä.

6 Muita vaihtoehtoja

Vaikka natiivisovellukset ja progressiiviset web-sovellukset ovatkin tässä tutkielmassa keskiössä, ne eivät ole ainoat vaihtoehdot. Tässä luvussa käsitellään kahta muuta tapaa toteuttaa sovellus tai tarjota käyttäjille tietoa.

6.1 Hybridisovellukset

Hybridisovellukset toimivat natiivisovellusten ja progressiivisten web-sovellusten välimaastossa. Ne asennetaan natiivisovellusten tavoin sovelluskaupasta, mutta toteutetaan käyttämällä sekä natiivikoodia että – progressiivisten web-sovellusten tavoin – web-kieliä[1]. Tämä mahdollistaa sovelluksen helpon kääntämisen monelle eri käyttöjärjestelmälle.

Hybridisovelluksia on monenlaisia. Osa niistä on käytännössä vain koko näytön kattava selainikkuna ilman osoitepalkkia tai muita ohjaimia; iOSin tapauksessa nykyään yleensä *WKWebView*-objekti. Sen sisällä näytetään sovelluksen varsinainen sisältö, joka on toteutettu web-kielillä. Hybridisovellukseen, toisin kuin web-sovellukseen, voi kuitenkin useimmiten upottaa lisäksi natiivikoodia, mikä mahdollistaa myös joidenkin laiterajapintojen hyödyntämisen.

Etenkin teknisesti taitamattomalle loppukäyttäjälle hyvin toteutettu hybridisovellus ei välttämättä eroa ominaisuuksiltaan tai käyttökokemukseltaan natiivisovelluksesta juurikaan. Sharma (2017) korostaa natiivisovellusten ja hybridisovellusten eroissa etenkin hybridisovellusten heikompaa suorituskykyä, mutta kaikissa sovelluksissa ja kaikille käyttäjille tällaiset erot eivät välttämättä ole selkeitä. Samoin hän mainitsee kehnon käyttökokemuksen, mutta esimerkiksi React Nativen ja vähemmän tunnetun Framework7:in kaltaiset ohjelmistokehykset mahdollistavat käyttökokemuksen, joka on hyvin lähellä natiivisovellusta – natiiveja käyttöliittymäelementtejä ja suunnitteluparadigmoja myöten. React Native ei riipu selainnäkymistä, vaan yhdistelee JavaScript-koodia ja natiivielementtejä, kun taas Framework7 vain imitoi natiiveja käyttöliittymäelementtejä muotoilemalla elementit samannäköisiksi (Biørn-Hansen, Majchrzak ja Grønli 2017). Lopputulos on hyvin samannäköinen, mutta React Na-

1. useimmiten HTML5, CSS ja JavaScript, sekä erilaiset JavaScript-kehykset kuten Vue.js ja Svelte

tiven toteutustapaa voidaan pitää parempana.

Esimerkkejä tunnetuista hybridisovelluksista ovat React Nativella toteutetut Facebook, Instagram ja Discord (Facebook 2020). Kaksi jälkimmäistä on saanut erittäin hyvät arvostelut App Storessa – viiden tähden asteikolla Instagram-sovelluksella on 117 tuhannen arvion jälkeen keskimäärin 4,5 tähteä ja Discord-sovelluksella 7,4 tuhannen arvion jälkeen keskimäärin 4,6 tähteä, kun taas Facebook-sovelluksella on 3 tuhannen arvion jälkeen keskimäärin vain 2,3 tähteä (Suomen App Store, tilanne 27.3.2021). On vaikea sanoa, kuinka suuri painoarvo arvosteluissa on itse sovelluksella ja kuinka suuri palvelulla, mutta Discord- ja Instagram-sovelluksen suosion pohjalta voidaan todeta, että hybridisovelluksetkin voivat olla suosittuja.

Tässäkin tapauksessa kehittäjän on siis tehtävä valinta tapauskohtaisesti. Kehittäjän taidoista riippuen hybridisovelluksen toteuttaminen voi olla hyvinkin helppoa tuttuja web-teknologioita käyttäen. Tällöin sovelluksen toteuttaminen usealle eri käyttöjärjestelmälle voi olla todella nopeaa ja kustannustehokasta, jolloin mahdolliset pienet miinukset käyttökokemuksessa ja suorituskyvyssä voivat olla hyväksyttäviä. Hybridisovellusten toteutustapa siis voi olla jopa varsin lähellä progressiivisten web-sovellusten toteutustapaa, mutta ensin mainitut ovat natiivisovellusten tapaan saatavilla App Storesta.

6.1.1 Hybridisovellusten suorituskyvystä

Kuten mainittu, hybridisovellusten suorituskyky on luonnollisesti teorian tasolla heikompi kuin natiivisovellusten, ja käytännön tulokset vaikuttavat tukevan tätä. Sekä Eskola (2018) että Johansson ja Söderberg (2018) tutkivat yleisesti käytetyn hybridiohjelmistokehyksen, React Nativen, suorituskykyä Android-laitteilla, ja totesivat sen olevan merkittävästi heikompi kuin natiivikoodin. Ensin mainittu painotti tutkimuksessaan laitteen ikää – uudemmilla laitteilla sovelluksen hitaus tuntuu vähemmän, kun taas ”etenkin vanhemmilla laitteilla yleiset toiminnot kuten sovelluksen käynnistyminen ja komponenttien renderointi ovat huomattavasti hitaampia ja latenssi saattaa olla jopa 10 kertaa pidempi kuin natiivivastineessa” (Eskola 2018, käännös tekijän). Johansson ja Söderberg (2018) puolestaan korostivat sovelluksen tyyppiä – React Native soveltuu useimpiin käyttökohteisiin, mutta monimutkaisem-

paa käyttäjän laitteessa tapahtuvaa prosessointia vaativissa sovelluksissa suorituskyky voisi olla ongelma. iOS-käyttöjärjestelmällä vastaavia empiirisiä tutkimuksia on tehty vähemmän, mutta ainakin Bilberg (2018) tuntuu puoltavan vastaavanlaisia tuloksia etenkin silloin, kun näytölle joudutaan renderoimaan suuria määriä kohteita.

6.2 Verkkosivut

Mobiilisovelluksen – sen paremmin natiivi- tai hybridisovelluksen kuin progressiivisen web-sovelluksenkaan – olemassaolo ei ole itseisarvo. Todella monet sovellukset olisivat korvattavissa yksinkertaisella verkkosivustolla, jonka toteuttaminen, päivittäminen ja ylläpitäminen olisi niin edullisempaa kuin helpompaakin. Käyttäjät eivät välttämättä halua täyttää laitteensa valikoita tai tallennustilaa sovelluksilla, joita käyttävät vain harvoin. Toisaalta Liu ym. (2019) tutkivat aihetta, ja totesivat, että vähittäiskauppiaiden kannattaa sijoittaa mobiilisovelluksen kehittämiseen, sillä se voi lisätä asiakasuskollisuutta. Siksi valinta verkkosivujen ja sovelluksen väliltä, tai päätös toteuttaa molemmat, onkin syytä tehdä harkiten.

7 Yhteenveto

7.1 Tulokset

Tutkielman päähypoteesi oli, että natiivisovellukset ovat käyttökokemukseltaan, ominaisuuksiltaan ja suorituskyvyltään parempia, kun taas progressiiviset web-sovellukset ovat kehittäjän näkökulmasta edullisempia ja nopeampia toteuttaa. Hypoteesi piti osittain paikkansa, sillä progressiiviset web-sovellukset eivät ainakaan vielä pysty täysin vastaamaan natiivisovellusten ominaisuuksiin ja suorituskykyyn, mutta ne ovat monille kehittäjille varmasti helpompi ja edullisempi vaihtoehto etenkin, jos sovellus täytyy kehittää useammalle käyttöjärjestelmälle.

Tutkielman pohjalta voidaan todeta, että valinta natiivisovelluksen ja progressiivisen websovelluksen väliltä on syytä tehdä tapauskohtaisesti. Jos sovellus toteutetaan vain yhdelle käyttöjärjestelmälle, tai se vaatii esimerkiksi push-ilmoituksia, on natiivisovellus luultavasti parempi valinta. Jos sovellus toteutetaan useammalle käyttöjärjestelmälle tai se on suhteellisen yksinkertainen, on progressiivinen web-sovellus puolestaan hyvin varteenotettava vaihtoehto. Myös tavallista verkkosivustoa tai hybridisovellusta voidaan harkita. Tietyt erityistapaukset, kuten vaativat 3D-pelit, on kuitenkin varmasti syytä toteuttaa natiivisovelluksena sekä käytettävissä olevien rajapintojen, kuten iOSin tapauksessa Applen Metal-grafiikkarajapinnan, että suorituskyvyn vuoksi.

7.2 Jatkotutkimuksesta

Progressiivisten web-sovellusten ja natiivisovellusten välisiä suorituskykyeroja on tutkittu harmillisen vähän etenkin iOSilla. Tässäkin tutkielmassa hyödynnetty tutkimus, jonka Willocx, Vossaert ja Naessens (2016) toteuttivat, on jo tässä vaiheessa puolen vuosikymmenen takaa, missä ajassa teknologia on ehtinyt kehittyä. Laitteiden nopeutuessa ja progressiivisten web-sovellusten teknologioiden kypsyessä erot natiivisovellusten ja progressiivisten websovellusten välillä voivat muuttua entistä pienemmiksi. Suorituskykyerojen tutkimisen lisäksi myös toteutustapojen välisten kustannuserojen tutkiminen voisi olla hyödyllistä.

Lähteet

Apple Inc. 2007. “Apple Reinvents the Phone with iPhone”. Viitattu 8. maaliskuuta 2021. https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/.

———. 2008. “iPhone App Store Downloads Top 10 Million in First Weekend”. Viitattu 10. maaliskuuta 2021. https://www.apple.com/fi/newsroom/2008/07/14iPhone-App-Store-Downloads-Top-10-Million-in-First-Weekend/.

———. 2014a. “App Store Review Guidelines – Apple Developer”. Viitattu 5. maaliskuuta 2021. https://developer.apple.com/app-store/review/guidelines/.

———. 2014b. “Swift – Apple Developer”. Viitattu 5. maaliskuuta 2021. https://developer.apple.com/swift.

———. 2018a. “Automatic Reference Counting – The Swift Programming Language (Swift 5.4)”. Viitattu 5. maaliskuuta 2021. https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html.

———. 2018b. “iOS – Human Interface Guidelines – Apple Developer”. Viitattu 5. maaliskuuta 2021. https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.

———. 2020. “iOS and iPadOS Software License Agreement”. Viitattu 8. maaliskuuta 2021. https://www.apple.com/legal/sla/docs/iOS14%5C%5FiPadOS14.pdf.

———. 2021. “Swift / Changelog”. Viitattu 5. maaliskuuta 2021. https://github.com/apple/swift/blob/main/CHANGELOG.md.

Apple Pty Ltd. 2021. “Further submission in response to the Digital Platform Services Inquiry into App Marketplaces”, helmikuu. Viitattu 29. maaliskuuta 2021. https://www.accc.gov.au/system/files/Apple%20Pty%20Limited%20(10%20February%202021).pdf.

Arment, Marco. 2021. “PWAs close some of the technical barriers, but the biggest is that nobody would ever find it. When people seek out a new app they’ve heard about, they don’t search the web for it — they search the App Store. They’d search for “overcast”, not find it, and never look elsewhere.” Viitattu 26. maaliskuuta 2021. https://twitter.com/marcoarment/status/1375097438772592640.

Bilberg, Dennis. 2018. *Comparing performance between react native and natively developed smartphone applications in swift : A comparative analysis and evaluation of the React Native framework.* University of Skövde, School of Informatics.

Biørn-Hansen, Andreas, Tim A Majchrzak ja Tor-Morten Grønli. 2017. “Progressive web apps: The possible web-native unifier for mobile development”. Teoksessa *International Conference on Web Information Systems and Technologies,* 2:344–351. Scitepress.

Charland, Andre, ja Brian Leroux. 2011. “Mobile Application Development: Web vs. Native”. *Commun. ACM* (New York, NY, USA) 54, numero 5 (toukokuu): 49–53. ISSN: 0001-0782. https://doi.org/10.1145/1941487.1941504. https://doi-org.ezproxy.jyu.fi/10.1145/1941487.1941504.

CNBC. 2021. “Apple’s App Store had gross sales around $64 billion last year and it’s growing strongly again”. Viitattu 22. helmikuuta 2021. https://www.cnbc.com/2021/01/08/apples-app-store-had-gross-sales-around-64-billion-in-2020.html.

Craigmile, Natalie. 2015. “Cost to Build a Mobile App: A Survey”. Viitattu 10. maaliskuuta 2021. https://clutch.co/app-developers/resources/cost-build-mobile-app-survey-2015.

Eskola, Rasmus. 2018. “React Native Performance Evaluation”. Tutkielma. https://aaltodoc.aalto.fi/bitstream/handle/123456789/32475/master%5C%5FEskola%5C%5FRasmus%5C%5F2018.pdf.

Facebook, Inc. 2020. “Who’s using React Native?” Viitattu 24. maaliskuuta 2021. https://reactnative.dev/showcase.

Github. 2008. “WordPress for iOS”. Viitattu 24. maaliskuuta 2021. https://github.com/wordpress-mobile/WordPress-iOS.

Github. 2015. “Firefox for iOS”. Viitattu 24. maaliskuuta 2021. https://github.com/mozilla-mobile/firefox-ios.

Himango, J. 2020. “Microsoft and Google team up to make PWAs better in the Play Store”, heinäkuu. Viitattu 29. maaliskuuta 2021. https://medium.com/pwabuilder/microsoft-and-google-team-up-to-make-pwas-better-in-the-play-store-b59710e487.

Johansson, Erik, ja Jesper Söderberg. 2018. “Evaluating performance of a React Native feature set”. Tohtorinväitöskirja. http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-148323.

Lam, Brian. 2007. “Apple iPhone Review”. Viitattu 8. maaliskuuta 2021. https://gizmodo.com/apple-iphone-review-276116.

Lamhaddab, Khalid, Mohamed Lachgar ja Khalid Elbaamrani. 2019. “Porting Mobile Apps from iOS to Android: A Practical Experience”. *Mobile Information Systems* 2019 (syyskuu): 4324871. ISSN: 1574-017x. https://doi.org/10.1155/2019/4324871. https://doi.org/10.1155/2019/4324871.

Liu, Huan, Lara Lobschat, Peter C. Verhoef ja Hong Zhao. 2019. “App Adoption: The Effect on Purchasing of Customers Who Have Used a Mobile Website Previously”. *Journal of Interactive Marketing* 47:16–34. ISSN: 1094-9968. https://doi.org/https://doi.org/10.1016/j.intmar.2018.12.001. https://www.sciencedirect.com/science/article/pii/S1094996818300719.

McLachlan, PJ, ja Tom Buckley. 2020. “Easy to build, monetize, and discover: List your web app on Google Play”, joulukuu. Viitattu 29. maaliskuuta 2021. https://blog.chromium.org/2020/12/pwa-play-billing-support.html.

MDN. 2021a. “Mixed content”, maaliskuu. Viitattu 29. maaliskuuta 2021. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

———. 2021b. “Progressive web apps (PWAs)”, helmikuu. Viitattu 29. maaliskuuta 2021. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

———. 2021c. “Web app manifests”, helmikuu. Viitattu 29. maaliskuuta 2021. https://developer.mozilla.org/en-US/docs/Web/Manifest.

Osmani, Addy. 2015. “Getting started with Progressive Web Apps”. Viitattu 10. maaliskuuta 2021. https://addyosmani.com/blog/getting-started-with-progressive-web-apps/.

Puder, Arno, Nikolai Tillmann ja Michał Moskal. 2014. “Exposing Native Device APIs to Web Apps”. Teoksessa *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems,* 18–26. MOBILESoft 2014. Hyderabad, India: Association for Computing Machinery. ISBN: 9781450328784. https://doi.org/10.1145/2593902.2593908. https://doi.org/10.1145/2593902.2593908.

Russell, Alex. 2015. “Progressive Web Apps: Escaping Tabs Without Losing Our Soul”. Viitattu 27. maaliskuuta 2021. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

———. 2016. “What, Exactly, Makes Something A Progressive Web App?” Viitattu 29. maaliskuuta 2021. https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/.

Selakovic, Marija, ja Michael Pradel. 2016. “Performance Issues and Optimizations in JavaScript: An Empirical Study”. Teoksessa *Proceedings of the 38th International Conference on Software Engineering,* 61–72. ICSE ’16. Austin, Texas: Association for Computing Machinery. ISBN: 9781450339001. https://doi.org/10.1145/2884781.2884829. https://doi.org/10.1145/2884781.2884829.

Sharma, Ashok. 2017. “Native vs hybrid apps: Key points to help you decide the best way forward”. *Open Source for You* (elokuu). https://www-proquest-com.ezproxy.jyu.fi/magazines/native-vs-hybrid-apps-key-points-help-you-decide/docview/1934962494/se-2?accountid=11774.

Statt, Nick. 2020. “Epic Games is suing Apple”, elokuu. Viitattu 29. maaliskuuta 2021. https://www.theverge.com/2020/8/13/21367963/epic-fortnite-legal-complaint-apple-ios-app-store-removal-injunctive-relief.

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.