TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SAMI AALTO

# MOBIILISOVELLUKSIEN TEKEMINEN JAVASCRIPTILLÄ

Kandidaatintyö

Tarkastaja: Mikko Nurminen
22. huhtikuuta 2018

# TIIVISTELMÄ

**Sami Aalto**: Mobiilisovelluksien tekeminen JavaScriptillä
Tampereen teknillinen yliopisto
Kandidaatintyö, 21 sivua
Huhtikuu 2018
Pääaine: Ohjelmistotekniikka
Tarkastaja: Mikko Nurminen

Avainsanat: JavaScript, hybridisovellus, PWA

JavaScript on tuonut mobiilisovelluskehitystä helpottavia menetelmiä, joiden avulla voidaan tehdä sovelluksia paljon helpommin ja nopeammin eri alustoille. Tässä kandidaatintyössä tarkastellaan näitä JavaScriptiin pohjautuvia lähestymistapoja, niiden toimintaperiaatteita sekä mitä heikkouksia ja vahvuuksia niissä on. Näissä tarkastellaan sekä ohjelmistokehittäjän että käyttäjän näkökulmia. Lähestymistavat on ryhmitelty hybridisovelluksiin, natiivikomponentteja käyttäviin sovelluksiin sekä progressiivisiin websovelluksiin.

Kandidaatintyössä saadaan selville, että kullakin lähestymistavalla on erilaiset vahvuudet ja heikkoudet, minkä vuoksi ne voivat soveltua eri tilanteisiin. JavaScriptillä on mahdollista tehdä mobiilisovelluksia, jotka ovat hyvin lähellä tai jopa joiltain osin parempia kuin natiivit sovellukset. JavaScript on siis oikein toimiva vaihtoehto mobiilisovelluksien tekemiseen.

# SISÄLLYSLUETTELO

# LYHENTEET JA MERKINNÄT

| | |
|---|---|
| Cross-platform | Useilla alustoilla toimiva. Alustoja voivat olla esimerkiksi Android, iOS ja Windows. |
| CSS | Cascading Style Sheets. Ohjelmointikieli, jota käytetään web-sivujen tyylillisen ulkoasun määrittämiseksi. |
| HTML | Hypertext Markup Language. Ohjelmointikieli, jota käytetään web-sivujen rakenteen ja sisällön määrittämiseen. |
| HTTPS | HTTP-protokolla, jossa kyselyt ja niiden vastaukset salataan TLS/SSL-protokollan avulla ennen lähettämistä. |
| Input/Output | Tiedonsiirto ohjelmistosta sisään ja ulos. Esimerkiksi näppäimistön syötteet. |
| JavaScript | Perinteisesti selaimissa ajettava ohjelmointikieli, jolla yleensä ajetaan selaimissa web-sivujen käyttäytymislogiikkaa. |
| Push-notifikaatio | Sovelluksen lähettämä viesti, joka syntyy usein jostain sovellukseen liittyvästä tapahtumasta. Voivat usein toimia myös silloin, kun sovellus ei ole aktiivisesti käytössä. |
| PWA | Progressive Web App. Lähestymistapa web-sovelluskehitykseen, jossa yritetään ottaa webin ja perinteisten sovelluksien parhaat puolet. |
| React | Web-käyttöliittymien tekoa auttava JavaScript-kirjasto. |
| React Native | Reactiin pohjautuva sovelluskehys, jonka avulla voi tehdä sovelluksia muun muassa mobiilialustoille. React Native tarjoaa React-käyttöliittymäkomponentteja, jotka voidaan kääntää alustan natiiveiksi komponenteiksi. |
| SDK | Software development kit. Mobiilialustoilla tarjoavat ohjelmistorajapinnan alustan ominaisuuksiin. |
| Service worker | Progressiivisiin web-sovelluksiin liittyvä web-sivun päälogiikasta erillinen JavaScript-prosessi, joka voi esimerkiksi hallinnoida selaimen tekemiä kyselyitä ja välimuistia. |
| SPA | Single Page Application. Lähestymistapa web-sivujen tekemiseen, jossa JavaScriptin avulla muodostetaan dynaamisesti sivun sisältö. Tämä lähestymistapa mahdollistaa sen, että koko sivua ei tarvitse ladata uudestaan, kun vaihtaa sivua. |
| WebView | Selaimen minimaalinen versio. Mahdollistavat web-sivujen näyttämisen mobiilisovelluksien sisällä. |
| Wrapper | Ohjelmistoissa käytetty rakenne, jossa jokin toiminnallisuus laitetaan uuden rajapinnan taakse. Soveltuu esimerkiksi erilaisten rajapintojen yhtenäistämiseen ja toteutusyksityiskohtien piilottamiseen. |

.

# 1. JOHDANTO

Mobiililaitteet ovat tulleet viimeisen vuosikymmenen aikana yhä suositummiksi ja ne ovat yhä isompi osa käyttäjien arkipäivää. Vuonna 2016 jopa yli puolet digitaalisen median käytöstä kohdistui Yhdysvalloissa mobiilisovelluksiin [1]. Mobiilisovelluksien suuri käyttöaste ja kasvavat tarpeet korostavat myös mobiilisovelluskehityksen tärkeyttä.

Mobiilialustat ovat hyvin erilaisia, minkä seurauksena sovelluksien kehitys usealle alustalle voi olla haastavaa. Android-, iOS- ja Windows-sovellukset käyttävät tyypillisesti eri ohjelmointikieliä ja niillä on erilaiset SDK-ohjelmointirajapinnat (Software Development Kit) alustakohtaisiin ominaisuuksiin. Tämän vuoksi näiden alustojen tukeminen on perinteisesti vaatinut kattavaa osaamista ja mahdollisesti moninkertaisen työn, koska samaa koodia ei pysty jakamaan alustojen välillä.

Natiivit ratkaisut olivat pitkän aikaa ainoa keino mobiilisovelluksien kehitykseen. Viime vuosien aikana on kuitenkin alkanut ilmestyä JavaScript-pohjaisia ratkaisuja, jotka yrittävät ratkoa näitä perinteisten natiivisovelluksien ongelmia.

Tässä kandidaatintyössä tutkimuskysymyksenä on, mitä erilaisia JavaScript-pohjaisia lähestymistapoja on perinteisen mobiilikehityksen ongelmien ratkaisemiseksi ja mitä hyvää ja huonoa niissä on. Hyvyyttä ja huonoutta tarkastellaan sekä sovelluskehittäjien että loppukäyttäjien näkökulmasta. Lähestymistapoina tarkastellaan erityisesti hybridisovelluksia, natiivikomponentteja hyödyntäviä sovelluksia ja uudempana ilmiönä tulleita progressiivisia web-sovelluksia (PWA, Progressive Web App).

Luvussa 2 tarkastellaan yleisellä tasolla, mitä keinoja on mobiilisovelluksien kehitykseen JavaScriptillä. Luvuissa 3 ja 4 tarkastellaan, miten näiden JavaScript-pohjaisten ratkaisujen käyttö voi näkyä sovelluskehittäjien ja käyttäjien näkökulmista. Lopuksi tehdään yhteenveto JavaScript-pohjaisten ratkaisujen ominaisuuksista.

# 2. JAVASCRIPT MOBIILISOVELLUKSISSA

JavaScript tunnetaan parhaiten selaimissa käytettynä ohjelmointikielenä, mutta sen käyttö on viime vuosien aikana laajentunut moniin muihin käyttökohteisiin. Esimerkiksi vuonna 2009 julkaistu Node.js on tuonut JavaScriptin myös palvelimien puolelle [2] ja vuonna 2013 julkaistu Electron-sovelluskehys mahdollistaa desktop-sovelluksien tekemisen JavaScriptillä [3]. Vastaavasti mobiilisovellukset ovat saaneet viime vuosina JavaScript-pohjaisia ratkaisuja, mutta JavaScriptiä on käytetty mobiilialustoilla aikaisemminkin.

Mobiilioptimoituja web-sivuja voi teknisesti ajatella ensimmäisinä JavaScript-pohjaisina mobiilisovelluksina. Tässä lähestymistavassa tehtiin pöytäkoneversioista täysin erilliset sivut, joiden rakenne oli suunnattu pienemmille näytöille sopiviksi. Tämä lähestymistapa kuitenkin vaati erillisten sivujen toteutuksen pöytäkoneille ja mobiililaitteille, mikä aiheutti lisätyötä. [4]

HTML5- ja CSS3-teknologioiden kehityttyä mobiilioptimoitujen sivujen paikalle on tullut RWD:tä (Responsive Web Design) hyödyntäviä web-sivuja, jotka voivat muuttaa sisältöään dynaamisesti näytön koon ja laitteen mukaan [4]. Tämä on nykyisin vallitseva lähestymistapa web-sovelluksien tekemiseen. Tämän lisäksi kehittyneempien sovelluskehyksien myötä SPA:t (Single Page Application) ovat yleistyneet, mikä on tehnyt web-sivuista yhä enemmän sovelluksien kaltaisia.

Webin jatkuvasta kehityksestä huolimatta web-sovelluksilla on silti joitain heikkouksia verrattuina perinteisiin mobiilisovelluksiin. Web-sovelluksien kehittäjät eivät esimerkiksi pysty hyödyntämään mobiililaitteiden natiiveja ominaisuuksia eivätkä web-sovellukset yleensä toimi ilman nettiyhteyttä. Tämä ei kuitenkaan ole ongelma uudemmille JavaScript-pohjaisille ratkaisuille.

JavaScript-pohjaiset mobiilisovellusratkaisut voidaan jakaa karkeasti kolmeen kategoriaan: WebView-komponentteihin pohjautuviin hybridisovelluksiin, natiivikomponentteja hyödyntäviin sovelluksiin ja progressiivisiin web-sovelluksiin. Nämä lähestymistavat mahdollistavat sovelluksen lataamisen mobiililaitteelle, ja kukin tarjoaa pääsyn natiiveihin ominaisuuksiin. Eräs näiden menetelmien isoimmista eduista on, että ne voivat jaettujen web-standardien avulla toteuttaa sovelluksia useille alustoille samalla tai lähes samalla koodipohjalla.

Seuraavissa alaluvuissa tutustutaan paremmin hybridisovelluksien, natiivikomponenttisten sovelluksien ja progressiivisten web-sovelluksien toimintaperiaatteisiin. Näistä tarkastellaan sekä niiden teknisiä lähestymistapoja että yleispiirteitä.

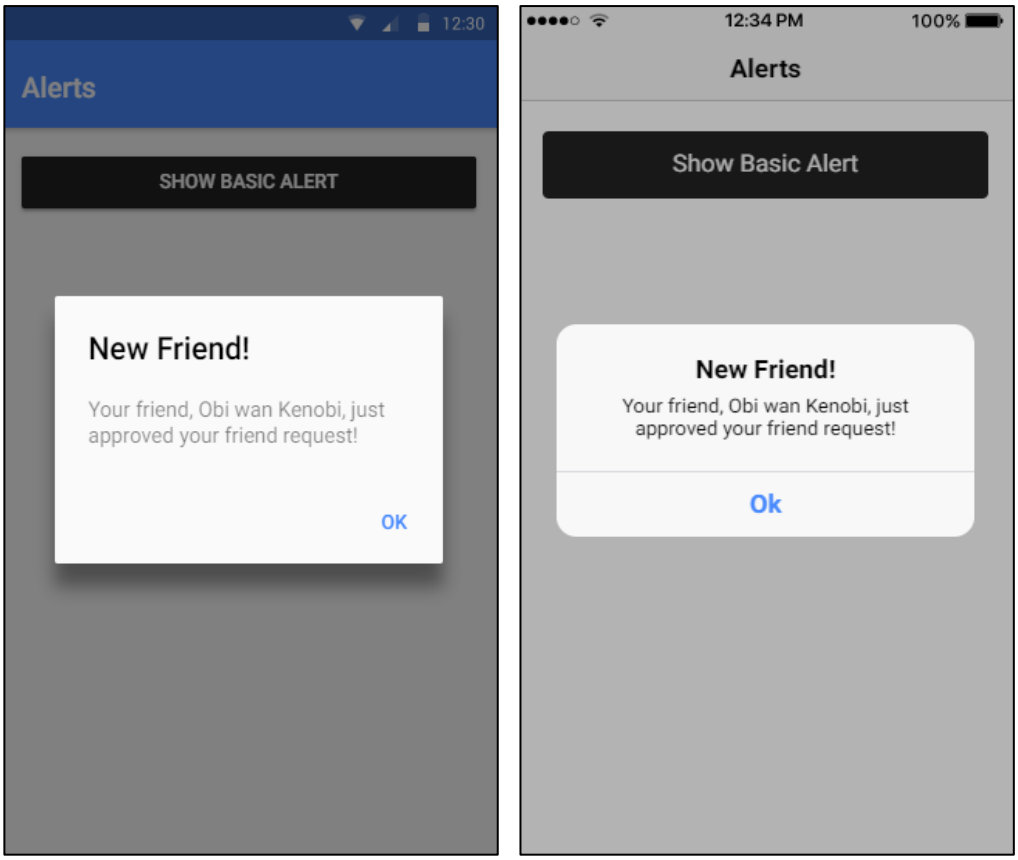
## 2.1 Hybridisovellukset

Hybridisovelluksilla on sekä natiivien että web-sovelluksien ominaisuuksia. Hybridisovelluksilla tarkoitetaan tyypillisesti sellaisia sovelluksia, jotka pohjautuvat WebView-komponentteihin ja antavat lisäksi pääsyn mobiililaitteen natiiveihin ominaisuuksiin. Natiiveilla ominaisuuksilla tarkoitetaan ominaisuuksia, joihin pääsee perinteisesti käsiksi alustakohtaisen natiivin ohjelmointikielen SDK-rajapinnoilla. Näihin kuuluvat esimerkiksi kamera, yhteystiedot, GPS, Bluetooth ja puhelimen kiihtyvyyssensorit.

WebView-komponentit ovat mobiililaitteissa sovelluksien sisäisiä selaimia. Tämä tarkoittaa, että niissä pystyy ajamaan selaimissa perinteisesti ajettavaa koodia, joka pohjautuu HTML-, CSS- ja JavaScript-ohjelmointikieliin. Erona hybridisovelluksissa on, että niiden tiedostot on tallennettu paikallisesti käyttäjän omalle laitteelle eikä niitä tarvitse hakea internetin välityksellä. WebView-komponentit eivät yleensä sisällä selaimien yleisiä osia, kuten hakupalkkia tai kirjanmerkkejä, minkä vuoksi sovellus voi näyttää loppukäyttäjälle täysin tavalliselta sovellukselta, jos sovellus tyyliltään on muuten perinteisen sovelluksen näköinen. [5]

Hybridisovellukset pääsevät mobiililaitteen natiiveihin ominaisuuksiin tyypillisesti wrapperien kautta, jotka tarjoavat JavaScript-rajapinnan natiivi ominaisuuksien käyttämiseksi. JavaScript-kutsut voivat siis ajaa natiivia koodia, jonka toteutusyksityiskohdat on piilotettu sovelluksen kehittäjältä. Tämä on hyödyllistä, sillä wrapper voi kulissien takana kutsua erilaisia rajapintoja eri alustoilla, vaikka hybridisovelluksen koodi olisi muuten sama. Eräs esimerkki tällaisesta wrapperista on Cordova-kirjasto [6].

Hybridisovelluksissa käyttökokemuksen olennaisin ero perinteisiin natiivisovelluksiin on se, että hybridisovelluksissa näkymät pohjautuvat webin käyttöliittymäkomponentteihin, jotka eivät ole täysin samanlaisia kuin natiivit. Useat hybridisovelluksien teon mahdollistavat sovelluskehykset kuitenkin tarjoavat tyyliteltyjä komponentteja, joiden avulla voi jäljitellä natiivien käyttöliittymäkomponenttien ulkoasua ja toiminnallisuutta. Tyylit voivat myös olla alustakohtaisia, sillä eri alustoilla on hieman erilaiset käyttöliittymät. Ionic on eräs esimerkki tällaisesta sovelluskehyksestä [7]. Kuva 1 sisältää esimerkin siitä, miten Ionic asettaa Alert-komponentin tyylit kohdealustan tyylin mukaiseksi.




***Kuva 1.*** *Ionic-sovelluskehyksen Alert-komponentti. Vasemmalla on Android-versio ja oikealla iOS. [7]*

Eräs hybridisovelluksien heikkous on, että natiivimaisista tyyleistä huolimatta ulkoasu ei välttämättä ole aina täysin identtinen ja käyttäytymisessä voi olla pieniä eroja. Tulevat käyttöjärjestelmäpäivitykset voivat myös muuttaa natiivien komponenttien ulkoasuja, mitä ei määritellyissä tyyleissä välttämättä huomioida. Eroavat tyylit voivat joskus olla myös tarkoituksellisia, jos halutaan saada yhtenäiset sovellusversiot kaikille alustoille, joissa ei haluta käyttää perinteisiä natiiveja ulkoasuja.

Hybridisovellukset eivät ole aivan yhtä tehokkaita kuin natiivit sovellukset, sillä JavaScript ohjelmointikielenä ei ole suorituskyvyltään aivan yhtä tehokas ja hyödyntää vain yhtä säiettä. Toisaalta hybridisovellukset voivat ainakin vuonna 2016 julkaistun tutkimuksen mukaan olla joissain tilanteissa energiatehokkaampia kuin vastaavat natiivit versiot. [8] Huonomman tehokkuuden vuoksi hybridisovellukset eivät välttämättä sovellu kovin hyvin suorituskykyintensiivisiin sovelluksiin, kuten peleihin. Vuonna 2015 tehdyn Google Play Storeen kohdistuneen tutkimuksen mukaan suuri osa hybridisovelluksista onkin enimmäkseen datan näyttämiseen tarkoitettuja sovelluksia, joissa ei yleensä vaadita kovin raskaita operaatioita [9].

Hybridisovelluksien vahvuutena on, että ne pohjautuvat usein ”Write once, run anywhere” -periaatteeseen, jossa samaa koodia voi käyttää kaikille alustoille. Käyttöliittymät pystyy toteuttamaan alustojen jakamilla web-standardeilla, ja alustakohtaiset erot

saadaan piilotettua wrapperien avulla. Alustakohtaisia erojakin tosin tarvitsee aina välillä ottaa huomioon. Esimerkiksi Android-laitteilla on käytössä erillinen paluunappi, jota iOS-laitteilla ei ole, mikä täytyy mahdollisesti ottaa sovelluksen suunnittelussa huomioon.

Hybridisovellukset voi laittaa perinteisiin sovelluskauppoihin samoin kuten natiivit sovellukset, eikä peruskäyttäjä välttämättä edes huomaa eroa hybridin ja puhtaan natiivin välillä. Tämän lisäksi hybridisovellukset voivat toimia myös tavallisina web-sivuina melkein sellaisinaan, koska ne käyttävät web-standardeja. Mobiililaitteiden natiiveja ominaisuuksia ei tosin voi kyseisissä web-versioissa yleensä käyttää.

## 2.2 Natiivikomponentteja hyödyntävät mobiilisovellukset

JavaScriptiin pohjautuvat natiivikomponentteja hyödyntävät mobiilisovellukset ovat vielä yhden askeleen lähempänä natiivia kuin hybridisovellukset. Kuten hybridisovelluksissa, sovelluksien toimintalogiikka on toteutettu pääasiallisesti JavaScriptillä. Erona on, että käyttöliittymä käyttää kulissien takana natiiveja käyttöliittymäkomponentteja, jotka on toteutettu kohdealustan natiivilla kielellä. Koska natiivikomponenttipohjaiset ratkaisut ovat ominaisuuksiltaan WebView-hybridisovelluksien ja puhtaiden natiivisovelluksien välissä, niihin on eri lähteissä viitattu sekä hybridi- että natiivisovelluksina.

Natiivikomponenttien käyttö tarkoittaa sitä, että mobiilisovellukset ovat ulkoasultaan aivan kuten puhtailla natiivikielillä toteutetut mobiilisovellukset. Natiivikomponenteilla on lisäksi natiivituntuma, koska niiden ydintoteutuksessa käytetään natiiveja kieliä. Nämä natiivit toteutukset on kytketty JavaScript-koodiin hieman vastaavalla tavalla kuin hybridisovelluksien wrappereissa. JavaScript-koodi voi siis kutsua natiivin koodin funktioita ja natiivin koodin tapahtumat voivat aktivoida JavaScript-funktioita.

Vaikka sovellus käyttääkin täysin erillisiä käyttöliittymäkomponentteja eri alustoille, JavaScriptin sovelluskehykset mahdollistavat usein saman käyttöliittymäkoodin käytön. Tämä on toteutettu määrittämällä komponenttiabstraktioita, jotka viittaavat erilaisiin toteutuksiin eri alustoilla. Näissä toteutuksissa yritetään varmistaa mahdollisimman samankaltainen toiminnallisuus eri alustoille. Aina tämä ei kuitenkaan ole mahdollista, sillä kaikille ominaisuuksille ei ole välttämättä tukea kaikilla alustoilla. Tämä johtaa usein tilanteisiin, joissa pitää tehdä alustakohtaisia ratkaisuja. Tämän vuoksi natiivikomponenttipohjaisissa ratkaisuissa onkin periaatteena yleisemmin ”Learn once, write anywhere”, jonka mukaan samalla toimintaperiaatteella voidaan kirjoittaa alustakohtaista koodia [10]. Tästä tunnetuimpana esimerkkinä on React Native -sovelluskehys.

Koska käyttöliittymien toteutus yleensä rajoittuu sovelluskehyksen ja kirjastojen tarjoamiin komponentteihin, voi tulla helpommin vastaan tilanteita, joissa ei ole valmiina omaan käyttötarkoitukseen sopivaa komponenttia. Tässä tapauksessa ainoa ratkaisu on toteuttaa kyseiset komponentit itse kullekin alustalle natiivilla koodilla.

Koska sovelluksissa käytetään natiivikomponentteja HTML-elementtien sijaan, käyttöliittymäkoodia ei yleisesti voi käyttää myös selaimessa. Tähän on kuitenkin viime vuosina tullut kirjastoja esimerkiksi React Nativelle. Nämä kirjastot muuntavat komponentit natiivin sijaan niitä vastaaviksi HTML-elementeillä toteutetuiksi React-komponenteiksi. [11]

Natiivikomponenttisilla ratkaisuilla pystyy hyödyntämään tehokkaammin alustan natiivia tehokkuutta kuin hybridisovelluksilla, mutta nekään eivät yllä puhtaan natiivin tehokkuuksiin. Eräs tehokkuutta haittaava seikka on, että JavaScriptin ja natiivikoodin välinen kommunikaatio aiheuttaa myös ylimääräistä kuormaa. [10]

## 2.3 Progressiiviset web-sovellukset

Web-sivustot ovat uusien SPA-sovelluskehyksien myötä tulleet yhä sovellusmaisemmiksi, ja progressiiviset web-sovellukset (PWA) vievät tämän vielä yhden askeleen pidemmälle. Progressiivisten web-sovelluksien ominaispiirteitä on web-kehityksessä ollut jo jonkin aikaa, mutta termi Progressive Web App tuli käyttöön vasta vuonna 2015. Progressiivisille web-sovelluksille ei ole yhtä yksikäsitteistä määritelmää, mutta niillä on useita ominaispiirteitä. Näihin piirteisiin sisältyvät esimerkiksi responsiivisuus, turvallisuus, yhteysriippumattomuus, sovellusmaisuus, asennettavuus ja linkitettävyys. Progressiivisten web-sovelluksien tavoitteena on tuoda esiin webin ja perinteisten sovelluksien parhaat puolet. [12]

Kuten useat modernit web-sivustot, progressiiviset sovellukset ovat responsiivisia, eli ne mukauttavat sisältöjensä asettelua ja skaalausta näytön koon mukaan. Tämä mahdollistaa hyvän käyttökokemuksen sekä pöytäkoneilla että mobiililaitteilla. Turvallisuusominaisuus puolestaan viittaa siihen, että kaikki progressiiviset web-sovellukset välitetään HTTPS:n välityksellä. Isoin ero perinteisiin moderneihin web-sivuihin tulee kuitenkin asennettavuus-ominaisuuden kanssa.

Aluksi progressiiviset web-sovellukset käyttäytyvät aivan kuten perinteiset web-sovellukset. Niihin päästään käsiksi tavallisten selaimien kautta, eikä niissä ole toiminnallisuuden osalta olennaisia poikkeavuuksia. Progressiiviset web-sovellukset voivat kuitenkin tarjota käyttäjille mahdollisuuden ”ylentää” web-sovelluksen progressiiviseksi, jolloin sovellus ladataan käyttäjän omalle laitteelle ja laitetaan muiden sovelluksien joukkoon. [12]

Jokaiselle progressiiviselle web-sovellukselle on määritelty Web App Manifest -tiedosto, jossa määritetään useita mobiilisovelluksille tyypillisiä piirteitä. Näitä ovat esimerkiksi sovelluksen nimi, sovellusikoni, sovelluksen aloitussivu, splash screen ja sovelluksen väriteema. [13] Näiden ominaisuuksien avulla sovellus saadaan käynnistyksen osalta käyttäytymään päällisin puolin aivan kuten natiivit.

Progressiiviset web-sovellukset ajetaan asennettuinakin selaimessa, mutta usein ilman selaimen omaa käyttöliittymää, kuten esimerkiksi hakupalkkia. Tältä osalta progressiiviset web-sovellukset toimivat siis samankaltaisesti kuin hybridisovellukset.

Sovelluksen yhteydettömyys tarkoittaa sitä, että se voi toimia myös ilman nettiyhteyttä. Koska sovelluksen tärkeimmät tiedostot ovat asennettuna laitteelle, niitä ei tarvitse erikseen hakea joka kerta, mikä nopeuttaa sovelluksen sisällön lataamista.

Toiminnallisuuden osalta progressiiviset web-sovellukset eroavat tavallisista web-sovelluksista siinä, että ne voivat käyttää service workereitä. Service workerit ovat JavaScript-ohjelmia, joita selain voi ajaa taustalla web-sivusta erillään. Ne voivat siis olla päällä myös silloin, kun sovellus ei ole auki. Tämä mahdollistaa eräiden natiivimaisten ominaisuuksien toteuttamisen, kuten esimerkiksi push-notifikaatiot, joiden avulla käyttäjälle voi lähettää tiedotuksia esimerkiksi sovelluksen tapahtumista. [14]

Eräs service workerien tärkein käyttökohde on välimuistin hallinta. Service workerit voivat tallentaa laitteelle tärkeitä resursseja, jolloin ne ovat heti valmiina ja saatavilla myös silloin, kun laitteella ei ole nettiyhteyttä. Esimerkiksi elokuvalippuja myyvässä sovelluksessa service worker voisi tallentaa käyttäjän ostamien lippujen tiedot paikallisesti. Tällöin käyttäjä voisi myöhemmin esimerkiksi tarkistaa lipun tunnistenumeron, vaikka nettiyhteyttä ei olisi. Service workerit voivat tulevaisuudessa ladata tietoja myös silloin, kun sovellus ei ole päällä. Esimerkiksi uutissovellus voisi ladata joka aamu päivän 10 merkittävintä uutista, jolloin ne olisivat heti saatavilla, kun sovellus avataan.

Toisin kuin hybridi- ja natiivikomponenttiratkaisuissa, progressiivisia websovelluksia ei voi laittaa mobiilialustojen omiin sovelluskauppoihin. Progressiivisiin web-sovelluksiin pääsee käsiksi vain sovelluksen vastaavalta web-sivulta. Tällä lähestymistavalla on muutama etu. Ensiksi, progressiiviset mobiilisovellukset voivat perinteisten web-sivujen tavoin hyödyntää hakukoneiden indeksointeja, minkä ansiota käyttäjät voivat löytää ne helpommin. Toiseksi, progressiivisen web-sovelluksen ylläpito on helpompaa, koska kaikki alustat käyttävät samaa versiota. Sovelluspäivitykset ovat paljon helpompia, sillä ne tarvitsee tehdä vain yhteen paikkaan, ja muutokset näkyvät kaikille lähes välittömästi. [15]

Progressiiviset web-sovellukset eivät ole pelkästään mobiililaitteita varten, sillä ne voivat parantaa käyttökokemusta myös pöytäkoneilla. Esimerkiksi service workerit ja push-notifikaatiot toimivat myös tavallisessa selaimessa. [16]. Lisäksi Microsoft aikoo antaa PWA-sovelluksille täyden tuen Windowsilla asennettavuuden osalta, ja ne tulevat saataville myös Microsoft Storeen Universal Windows Appien rinnalle [17].

Eräs progressiivisten web-sovelluksien heikkous tällä hetkellä on se, että niitä ei tueta vielä kaikilla alustoilla. Erityisesti iOS-alustoilla ei ole vielä tukea useille PWA-ominaisuuksille, kuten service workereille. Service worker -tuki on kuitenkin tulossa Safari-selaimeen iOS-versiossa 11.3, jonka odotetaan tulevan vuoden 2018 kevään aikana [18, 19]. iOS-tuessa on julkisen betan perusteella vielä joitain ongelmia ja eroavaisuuksia muiden

alustojen toteutuksiin verrattuna, mikä pitää mahdollisesti ottaa huomioon iOS-alustan kanssa [20].

***Taulukko 1.*** *PWA-ominaisuuksien tukitaulukko mobiilialustoille [16, 21].*

| | **App Manifest** | **Service workerit** | **Push-notifikaatiot** |
|---|---|---|---|
| **Chrome for Android** | 2014 | 2015 | 2016 |
| **Firefox for Android** | 2016<br>2018 täysi tuki [22] | 2016 | 2016 |
| **Microsoft Edge for Android** | Ei tukea, mutta on tulossa | 2017, oletuksena ei käytössä<br>2018 tulossa oletukseksi | Tulossa 2018 |
| **Opera for Android** | 2016 | 2016 | 2016 |
| **iOS Safari** | Tulossa 2018 | Tulossa 2018 | Ei tukea |
| **Samsung Internet** | 2016 | 2016 | 2016 |

Alustojen lisäksi myös selaimissa on pieniä eroja PWA-ominaisuustuen kannalta. Esimerkiksi Microsoft Edge -selaimen nykyinen versio (v16) ei tue push-notifikaatioita ja service workerit eivät ole oletuksena käytössä. Taulukossa 1 näkyy tarkemmin mobiiliselaimien tukia PWA-sovelluksien pääominaisuuksille. iOS-selaimet pohjautuvat pääosin Safarin WebViewiin [20], minkä vuoksi muita iOS-selaimia ei ole otettu vertailuun mukaan.

# 3. OHJELMISTOKEHITTÄJÄN NÄKÖKULMA

Yksi tärkeimmistä syistä JavaScript-pohjaisen ratkaisun valinnalle mobiilisovelluskehityksessä on se, että se helpottaa ja nopeuttaa ohjelmistokehittäjien työtä. Seuraavissa alaluvuissa katsotaan tarkemmin, miten erilaiset JavaScript-lähestymistavat näkyvät mobiilisovelluskehityksessä.

## 3.1 Koodin uusiokäyttö ja ylläpito

JavaScriptin avulla kaikki alustat voivat käyttää samaa koodipohjaa sovelluslogiikan toteuttamiseen. Tämä sekä nopeuttaa sovelluskehitystä että vähentää sen kokonaiskustannuksia. Saman koodipohjan käyttö helpottaa myös koodin ylläpitoa koko sovelluksen elinkaaren aikana. Esitetyissä lähestymistavoissa on kuitenkin eroavaisuuksia sen suhteen, miten paljon uusiokäyttö käytännössä onnistuu.

Hybridisovelluksissa yleisen ”Write once, run anywhere” -periaatteen ja web-standardien avulla lähes kaiken koodin voi jakaa alustojen välillä, ellei erikseen haluta alustakohtaisia eroavaisuuksia. Sovelluskehykset ovat yleensä vastuussa alustakohtaisista eroista, joten kehittäjän ei tarvitse kiinnittää niihin läheskään niin paljoa huomioita kuin suoraan natiivin kanssa työskennellessä.

Natiivikomponentteihin pohjautuvat sovellukset voivat usein jakaa hyvin suuren osan koodista alustojen välillä. Tämän lähestymistavan alustaläheisyys kuitenkin tarkoittaa sitä, että kaikille ominaisuuksille ei välttämättä löydy vastinetta muilta alustoilta. Tämän vuoksi alustakohtaiset hienosäädöt ovat yleisempiä kuin hybrideissä. Natiivikomponenttiset lähestymistavat eivät yleensä toimi täysin suoraan myöskään selaimessa, joten mahdollinen selainversio saattaa vaatia myös erillistä työtä.

Progressiiviset web-sovellukset mahdollistavat parhaimmin koodin uusiokäytön, koska kaikki alustat pyörivät samassa ympäristössä: selaimessa. Kaikki selaimet noudattavat ainakin teoriassa yhteneviä web-standardeja, joten toiminnallisuuteen voi yleensä luottaa oikein hyvin. Käytännössä selaimilla on kuitenkin joitain eroavaisuuksia, eivätkä ne toteuta kaikkia standardin ominaisuuksia. Nykyisin selaineroavaisuudet ovat kuitenkin paljon pienempiä kuin ennen, joten tämä ei ole yleensä iso ongelma, ellei halua käyttää kaikista uusimpia ominaisuuksia.

Saman koodipohjan käytön lisäksi progressiiviset web-sovellukset ovat muutenkin helpommin ylläpidettäviä. Sekä hybridi- että natiivikomponenttilähestymistavoissa täytyy ottaa huomioon kohdealustan sovellusmääritykset, sovelluksien rakentamisprosessit ja sovelluskauppaan laittaminen. Sovelluspäivitykset ovat usein myös monivaiheisia prosesseja. Progressiivisilla web-sovelluksilla tätä ongelmaa ei ole, koska niitä ei tarvitse

laittaa sovelluskauppoihin: ne voi julkaista suoraan webbiin, jossa ne ovat kaikkien saatavilla. Päivityksiäkin voi julkaista suoraan ilman, että niiden pitäisi mennä sovelluskaupan hyväksynnän läpi.

Hybridisovelluksista ja progressiivisista web-sovelluksista on hyvä huomioida, että ne eivät ole toisiaan poissulkevia ratkaisuja. Ne ovat monilta osin hyvin samankaltaisia. Olennaisimmat erot niiden välillä ovat julkaisupaikka ja hybridien pääsy natiivimaisempiin toiminnallisuuksiin. Esimerkiksi Ionic-sovelluskehys mahdollistaa saman koodipohjan käytön hybridisovelluksissa ja progressiivisissa web-sovelluksissa [23].

Koodin uusiokäytön ja ylläpidon osalta progressiiviset web-sovellukset ovat selkeästi näistä paras ratkaisu. Tämä näkyy myös sovelluksen kustannuksissa, sillä progressiivisten web-sovelluksien sovelluskehitys ja ylläpito voi maksaa keskimäärin vain 25–33 % siitä, mitä natiivi sovellus maksaisi [24].

## 3.2 Vaadittu osaaminen

Natiivit alustat ovat perinteisesti vaatineet useita ohjelmointikieliä ja vastaavasti useita eri osaamistaustan ohjelmistokehittäjiä. Esimerkiksi Android-sovelluksissa ensisijaisena kielenä on Java ja iOS-sovelluksissa on vastaavasti Swift ja Objective-C. Nämä eivät toimi selaimissa, joten mahdollinen selainversio vaatisi vielä web-osaamistakin.

JavaScriptin kanssa tätä useiden alustojen ongelmaa ei ole, koska kaikki alustat voivat ajaa JavaScriptiä. Jos sovelluskehittäjillä on jo entuudestaan web-osaamista moderneilla sovelluskehyksillä, siirtyminen modernista web-kehityksestä hybridisovelluksiin tai progressiivisiin web-sovelluksiin onnistuu varsin helposti. Hybridisovelluksien osalta tarvitsee vain opetella wrapper-kirjaston tarjoamat uudet rajapinnat ja progressiivisissa web-sovelluksissa service workerien käyttöä. Muutoin sovelluskehitys on hyvin samankaltaista perinteisempään web-ohjelmointiin verrattuna.

Myös natiivikomponenttisissa ratkaisuissa on hyvin paljon samankaltaisuuksia web-ohjelmoinnin kanssa. Vaikka ne eivät käytäkään HTML:n käyttöliittymäkomponentteja tai CSS:n tyylejä, niiden sovelluskehykset käyttävät usein hyvin samankaltaista syntaksia. Esimerkiksi React Nativen tapauksessa HTML-elementtitagien sijaan käytetään natiiveja elementtejä kuvaavia React-komponentteja, jotka käyttäytyvät ohjelmoijan näkökulmasta hyvin samalla tavalla.

Ohjelma 1 ja Ohjelma 2 sisältävät hyvin yksinkertaisen esimerkin React- ja React Native -komponenteista. Ohjelmissa on yksinkertainen funktionaalinen komponentti, jolle annetaan counter-lukuarvo ja add-funktio, joka kasvattaa counterin arvoa nappia painettaessa. add-funktion toteutus tulee tässä ulommalta komponentilta, jota ei esimerkissä näytetä. Komponentille on myös määritelty yksinkertaiset tyylit. React esimerkissä on käytetty CSS-in-JS-lähestymistapaa, jossa CSS-tyylit määritellään JavaScript-koodissa. React

Nativessa ei ole erillistä otsikkoelementtiä, minkä vuoksi esimerkissä on otettu ylimääräinen fontWeight-tyyliasetus yhtenäisemmän tyylin saamiseksi. Ohjelmia vertailemalla voidaan havaita, että ne ovat syntaksiltaan lähes identtiset. Tämän ansiota React Nativen kirjoittamisen pitäisi sujua web-kehittäjältä melkein yhtä helposti kuin perinteisen webin Reactin.

```
1  import { StyleSheet } from 'aphrodite';
2
3  const styles = StyleSheet.create({
4    title: {
5      color: 'blue',
6      fontSize: 24
7    },
8    counter: {
9      fontSize: 16
10   }
11 });
12
13 const ExampleCounter = ({ counter, add }) => (
14   <div>
15     <h1 style={styles.title}>Counter</h1>
16     <p style={styles.counter}>Value: {counter}</p>
17     <button onClick={add}>Add</button>
18   </div>
19 );
20 export default ExampleCounter
```

***Ohjelma 1.*** *Yksinkertainen esimerkki React-komponentista.*

```
1  import { Button, StyleSheet, Text, View } from 'react-native';
2
3  const styles = StyleSheet.create({
4    title: {
5      color: 'blue',
6      fontWeight: 'bold',
7      fontSize: 24
8    },
9    text: {
10     fontSize: 16
11   }
12 });
13
14 const ExampleCounter = ({ counter, add }) => (
15   <View>
16     <Text style={styles.title}>Counter</Text>
17     <Text style={styles.counter}>Value: {counter}</Text>
18     <Button onPress={add} title="Add" />
19   </View>
20 );
21 export default ExampleCounter
```

***Ohjelma 2.*** *Yksinkertainen esimerkki React Native -komponentista.*

Natiivikomponenttisilla JavaScript-ratkaisuilla ei kuitenkaan ole takanaan yhtä pitkää historiaa kuin webillä, minkä vuoksi kaikkia ominaisuuksia ei välttämättä ole valmiina.

Koska käyttöliittymäkomponentit on sisäisesti toteutettu kunkin alustan natiivilla koodilla, uuden toiminnallisuuden lisääminen edellyttää sen toteuttamista itse. Tämä siirtymä voi olla tavalliselle web-koodarille hieman hankalampaa, jos mobiilialustojen natiivit rajapinnat ja kielet eivät ole muutoin tuttuja. Tämän vuoksi natiivikomponenttisissa ratkaisuissa voi olla hyvä, että tiimissä on myös perinteiseen mobiiliohjelmointiin perehtyneitä sovelluskehittäjiä.

On hyvä huomioida, että JavaScript ei suinkaan ole ainoa kieli, jolla pystyy tekemään cross-platform-mobiilisovelluksia. Esimerkiksi C#-ohjelmointikieleen pohjautuva Xamarin-sovelluskehys mahdollistaa myös cross-platform-sovelluksien tekemisen [25]. JavaScriptin etuna on kuitenkin sen saama suosio webin puolelta, mikä voi myös mahdollistaa mobiilisovelluksien koodin käytön selaimessa, jos sille on tarvetta.

## 3.3 Sovelluksen testaus

JavaScript-vaihtoehtojen välillä testauksen osalta isoimpana erona on koodin ajoympäristö. Hybridi- ja progressiivisia web-sovelluksia voi ajaa selaimessa, kun taas natiivikomponenttiset ratkaisut edellyttävät aina joko fyysisen mobiililaitteen tai emulaattorin.

Selainympäristö on yleisesti hieman helpommin testattava, koska se ei tarvitse erillistä laitetta tai mobiilisovelluksien rakennusprosessia, joka voi hidastaa sovelluskehitystä. Kaikki perinteisen web-kehityksen testaustyökalut toimivat, joista Google Chromen kehittäjätyökalut (dev tools) ovat hyvä esimerkki. JavaScriptin ja käyttöliittymärakenteen yleisen tarkastelun lisäksi Chromen työkaluilla avulla pystyy simuloimaan mobiililaitteiden käyttäytymistä esimerkiksi muuntamalla selainikkunan kokoa pienemmäksi tai rajoittamalla verkon latausnopeutta. [26]

Hybridi- ja progressiivisten web-sovelluksien testaus on muuten identtistä, paitsi että kaikki hybridisovelluksien ominaisuudet eivät välttämättä toimi selaimessa. Tämä johtuu siitä, että tavallisissa selaimissa ei voi ajaa mobiililaitteiden natiivien rajapintojen ominaisuuksia. Nämä ominaisuudet pitää testata oikeilla laitteilla erikseen. Hybrideissä pitää myös mahdollisesti ottaa huomioon eri alustoille suunnatut tyylit, jotka voivat olla hieman erilaiset.

Vaikka natiivikomponenttiset sovellukset ajetaankin natiivien sovelluksien tavoin joko oikealla laitteella tai emulaattorissa, ne voivat siitä huolimatta käyttää selaimien kehittäjätyökaluja. Esimerkiksi React Native -mobiilisovellukset voidaan kytkeä ulkoisen tietokoneen selaimeen siten, että selaimessa voi tarkastella sen tilaa. Tämä soveltuu erityisesti JavaScript-logiikan, konsolitulostuksien ja käyttöliittymärakenteen tarkasteluun. Jos halutaan tarkastella enemmän natiivin koodin toimintaa, se täytyy tehdä perinteisemmillä natiivimenetelmillä, kuten esimerkiksi Android Studiolla ja Xcodella. Näitä voi käyttää sovelluksien natiivien osien tarkasteluun samoin kuin puhtaasti natiiveissa sovelluksissa. [27] Selaimen työkalut todennäköisesti riittävät useimpien ongelmien löytämiseen, mutta

jos ongelma on peräisin natiivista koodista, sovelluskehittäjä voi joutua käyttämään useita eri työkaluja ongelman löytämiseksi.

Koska natiivikomponenttisissa sovelluksissa ajetaan eri koodia eri alustoilla, niissä on tärkeämpää testata usealla alustalla. Hybrideissä ja progressiivisissa voi luottaa hieman enemmän selaintoteutuksien yhtenäisyyteen. Selaintestaus tai yleisesti testaus vain yhdellä alustalla ei tietenkään yksin riitä sovelluksen toimivuuden varmistamiseksi, mutta varsinkin sovelluksen kehitysvaiheessa se voi olla suuri apu.

Kaikille vaihtoehdoille on olemassa myös työkaluja, joiden avulla sovelluksille pystyy tekemään automaattitestejä, joita voidaan ajaa erilaisissa virtuaaliympäristöissä. Testejä pystyy tekemään eri alustoille esimerkiksi appium-sovelluskehyksellä, joka toimii kaikille esitetyille lähestymistavoille [28].

# 4. KÄYTTÄJÄN NÄKÖKULMA

Sovellukset tehdään loppujen lopuksi loppukäyttäjiä varten, minkä vuoksi on myös tärkeää huomioida vaikutukset käyttäjäkokemukseen teknologiaa valittaessa. Seuraavissa alaluvuissa tarkastellaan tarkemmin, miten eri JavaScript-lähestymistavat voivat vaikuttaa sovellukseen käyttäjien näkökulmasta.

## 4.1 Käyttökokemus

Sekä hybridi- että natiivikomponenttiset sovellukset ovat saatavilla sovelluskaupoista aivan kuten puhtaat natiivisovellukset. Tämän vuoksi on hyvin mahdollista, että käyttäjä ei edes tiedosta, että sovellus ei ole puhtaasti natiivi. Natiivikomponenttiset ovat ulkoasullisesti täysin identtisiä puhtaisiin natiivisovelluksiin, ja hybridisovellukset voivat päästä hyvin lähelle sitä. Vuonna 2015 tehdyn Google Play Storen arvosteluihin kohdistuneen tutkimuksen mukaan käyttäjät antoivat hybridisovelluksille keskimäärin samankaltaisia arvosteluita kuin natiivisovelluksille eri sovelluskategorioissa [29]. Tästä voidaan päätellä, että hybridivaihtoehdon valinta ei suoraan vaikuta käyttäjien kokemaan käyttökokemukseen.

Hybridi- ja natiivikomponenttiset sovellukset ovat ladattavissa sovelluskaupasta, minkä vuoksi niiden asennusprosessi on käyttäjille varsin tuttu. Toisaalta vaikka käyttäjät käyttävätkin suuren osan ajastaan mobiililaitteilla, he lataavat uusia sovelluksia harvoin. Suurin osa käyttäjistä ei asenna sovelluksia kuukausittain [30]. Progressiiviset web-sovellukset voivat tuoda käyttäjille matalamman asennuskynnyksen, koska käyttäjät voivat helposti testata sovellusta selaimessa ennen sovelluksen kunnollista asentamista omalle laitteelle. Progressiiviset web-sovellukset ovatkin moninkertaistaneet kävijöiden ja sovellusasentajien lukumäärän sekä sovelluksen käyttöaikoja useille yrityksille, verrattuna sekä natiivi- että web-sovelluksiin [31].

Käyttökokemuksen parantamisessa eräs hyvä esimerkki on Afrikassa toimiva Jumia-verkkokauppa, joka on saanut progressiivisella web-sovelluksella 12 kertaa enemmän käyttäjiä kuin natiiveilla Android- ja iOS-sovelluksilla. Sovelluksen suuri suosio pohjautuu pitkälti siihen, että se vie 25 kertaa vähemmän muistia ja käyttää 80 % vähemmän mobiiliverkkodataa. Tämä on Afrikassa erityisen tärkeää, koska siellä on käytössä paljon hitaampi 2G-verkko ja niissä on usein tiukat datarajoitukset. [32] Progressiiviset web-sovellukset voivat siis olla oikein hyvä vaihtoehto varsinkin alueille, joissa tietoliikenne on rajoitetumpaa.

Eräs erottava tekijä erityisesti natiivikomponenttisten ja progressiivisten sovelluksien kanssa on sovelluksen ulkoasun natiivimaisuus. Koska natiivikomponentit ovat alustan

natiiveja komponentteja, ne oletuskohtaisesti noudattavat kunkin alustan yleisiä tyylisääntöjä. Progressiivisissa web-sovelluksissa sovelluksen ulkoasu on useammin sama kaikilla alustoilla, mikä tuo samalla enemmän yhtenäisyyttä eri alustojen välille. Yhtenäisyys alustojen välillä tai eroavaisuudet natiivien alustojen yleisistä periaatteista voivat olla tarkoituksenmukaisiakin varsinkin, jos sovelluksen tekijä haluaa käyttää omanlaista tyyliään. Tämä lähestyminen on mahdollista myös hybridisovelluksissa, kuten luvussa 2.1 todettiin.

## 4.2 Ominaisuudet

Natiivikomponenttisella lähestymistavalla on mahdollista toteuttaa kaikki natiivit ominaisuudet, koska ne voivat ajaa mitä tahansa natiivia koodia. Hybridisovellukset voivat usein käyttää lähes kaikkia natiiveja ominaisuuksia, mutta käyttöliittymä ei yleensä koostu natiivikomponenteista.

Progressiivisten web-sovelluksien tapauksessa sovelluksen rajoitteet kohdistuvat selaimeen. Selaimet ovat kuitenkin viime vuosina saaneet yhä enemmän ominaisuuksia, jotka ovat perinteisemmin olleet natiiviominaisuuksia, kuten esimerkiksi kamera ja gps. Taulukko 2 näyttää esimerkkeinä eräiden natiivimaisten ominaisuuksien tukia eri menetelmillä. Selaimista mukaan on otettu vain Chrome, Firefox, Edge ja Safari.

***Taulukko 2.*** *Natiivimaisten ominaisuuksien tuki eri lähestymistavoilla [21, 33].*

| **Ominaisuus** | **Natiivikomponenttinen** | **Hybridi** | **Progressiivinen web-sovellus** |
|---|---|---|---|
| Akun määrä | Kyllä | Kyllä | Chrome |
| Apple pay | Kyllä | Kyllä | Chrome, Safari |
| Bluetooth | Kyllä | Kyllä | Chrome, osittainen tuki |
| Geometrinen sijainti | Kyllä | Kyllä | Chrome, Firefox, Edge, Safari |
| Kamera / mikrofoni | Kyllä | Kyllä | Chrome, Firefox, Edge, Safari |
| Liikesensorit | Kylä | Kyllä | Chrome, Firefox, Edge |
| Natiivit käyttöliittymäkomponentit | Kyllä | Ei | Ei |
| Nettiyhteyden tunnistaminen | Kyllä | Kyllä | Chrome, Firefox, Edge, Safari |
| Touch-tapahtumat | Kyllä | Kyllä | Chrome, Firefox, iOS Safari |
| Värähtely | Kyllä | Kyllä | Chrome, Firefox |
| Yhteystiedot / tekstiviestit | Kyllä | Kyllä | Ei |

Taulukosta voidaan havaita, että varsin moni ominaisuus on jo saatavilla myös selaimille, mutta selaintuki on vielä hieman vaihtelevaa. Webin nopean kehityksen myötä on odotettavissa, että selaimet tulevat tulevaisuudessa saamaan yhä enemmän ominaisuuksia.

## 4.3 Tehokkuus ja sovelluskoko

JavaScript ei ole ohjelmointikielenä yleensä yhtä suorituskykyinen kuin mobiilialustoilla käytetyt natiivit kielet, minkä seurauksena sovellus voi olla hieman hitaampi kuin natiivit. Esimerkiksi JavaScriptin yksisäikeinen luonne rajoittaa sen tehokkuutta. Huonommasta suorituskyvystään huolimatta JavaScript voi vuonna 2016 tehdyn tutkimuksen mukaan olla noin 2 kertaa energiatehokkaampi Android-hybridisovelluksissa kuin vastaava Javalla tehty toteutus, ainakin tietyissä CPU-intensiivisissä operaatioissa [34]. Kyseisessä tutkimuksessa ei kuitenkaan ajettu sovelluksille tyypillisiä Input/Output- ja käyttöliittymäoperaatioita, minkä vuoksi tutkimuksen tulos ei välttämättä päde suoraan kokonaisiin sovelluksiin.

Natiivikomponenttisissa lähestymistavoissa osan sovelluksesta voi kirjoittaa kokonaan natiivilla koodilla, minkä vuoksi sillä voi saavuttaa parhaimpia suorituskykyjä. Hybridisovelluksissa ja progressiivisissa web-sovelluksissa tämän tason optimointi ei käytännössä ole mahdollista. Toisaalta kommunikointi JavaScriptin ja natiivin koodin välillä aiheuttaa myös työkuormaa [35], minkä vuoksi liian tiheät siirtymät näiden välillä voivat aiheuttaa huonomman suorituskyvyn.

Progressiivisissa web-sovelluksissa on yleisesti mainostettu, että service workerit toisivat energiasäästöjä. Vuonna 2017 tehdyssä tutkimuksessa ei kuitenkaan havaittu, että service workerit olisivat tuoneet tilastollisesti havaittavia energiasäästöjä. Joissain testitapauksissa ne jopa pahensivat energiankulutusta, sillä service workerit kuluttavat myös hieman energiaa. Tutkimus toisaalta myös havaitsi, että tulokset vaihtelivat hieman sovelluksesta riippuen, joten tämä voi riippua siitä, miten service workereita käytetään. [36] Service workerien avulla on kuitenkin mahdollista nopeuttaa sivujen latautumista merkittävästi ja vähentää datansiirron aiheuttamaa kuormaa, kuten luvussa 4.1 jo todettiin.

Muistin osalta hybridit ja natiivit sovellukset vievät suurin piirtein yhtä paljon muistia [29]. Progressiiviset web-sovellukset voivat puolestaan olla paljon kevyempiä, koska niiden ei tarvitse kuljettaa ollenkaan natiivia koodia, ja niiden ei tarvitse ladata kaikkea kerralla. Esimerkiksi Twitterin progressiivinen web-sovellus Twitter Lite vie vain alle 3 % vastaavan natiivin Android-sovelluksen koosta [37] ja intialaisen Ola-taksisovelluksen progressiivinen versio vie yli 500 kertaa vähemmän muistia kuin sovelluksen iOS-versio [38]. Näin isot säästöt voivat olla hyvin merkittäviä mobiililaitteilla, joiden muistinkäyttö on rajoitettua.

# 5. YHTEENVETO

JavaScript tarjoaa kolme lähestymistapaa mobiilisovelluksien tekemiseen, joiden avulla on helpompi toteuttaa cross-platform-mobiilisovelluksia. Kaikki vaihtoehdot mahdollistavat koodin jakamisen eri alustoille ja siten koodin helpomman ylläpidon, mutta jokaisessa vaihtoehdossa on sekä hyvät että huonot puolensa. Taulukko 3 sisältää tiivistettynä erilaisten JavaScript-lähestymistapojen yleispiirteet.

***Taulukko 3.*** *JavaScript-mobiilisovellusmenetelmien ominaisuuksia.*

| **Ominaisuus** | **Natiivikomponenttinen** | **Hybridisovellus** | **Progressiivinen web-sovellus** |
|---|---|---|---|
| Alustatuki | Vahva | Hyvin vahva | Vahva, mutta täysi tuki vaihteleva |
| Koodin jaettavuus | Suuri osa | Lähes kaikki | Kaikki |
| Käyttöliittymä | Natiivi | Web natiivimaisilla tyyleillä | Web, voi olla myös natiivimainen tyyli |
| Natiivit ominaisuudet | Kaikki, mutta voi vaatia natiivikoodia | Lähes kaikki | Rajoittuneesti |
| Sovelluskauppa | Kyllä | Kyllä | Ei |
| Sovelluskoko | Keskikokoinen | Keskikokoinen | Pieni |
| Suorituskyky | Hyvä | Kohtalainen | Kohtalainen |
| Testattavuus | Vaativa | Melko helppo | Helppo |
| Toimii webissä | Kyllä, mutta ei suoraan | Kyllä, pienillä hienosäädöillä | Kyllä |
| Vaadittu osaaminen | Korkea<br>(Web + mobiilialustat) | Pieni<br>(Web) | Keskitaso<br>(Syvä web-tuntemus) |

Natiivikomponentteja käyttävät menetelmät mahdollistavat käyttäjäkokemuksen, joka on lähimpänä puhtaasti natiiveja sovelluksia. Osan koodista voi kirjoittaa puhtaasti natiivilla koodilla, mitä voidaan käyttää esimerkiksi sovelluksen tehokkuuskriittisten osien optimointiin. Tämän kääntöpuolena on, että kaikkea koodia ei voi jakaa yhtä tehokkaasti, ja natiivit optimoinnit voivat vaatia perinteisempää mobiilikehitysosaamista. Natiivikomponenttiset menetelmät sopivat oikein hyvin sellaisiin sovelluksiin, joissa halutaan tehdä alustakohtaisia optimointeja.

WebVieweihin pohjautuvat hybridisovellukset tarjoavat natiivikomponentteja helpomman tavan useamman alustan tukemiseksi, sillä lähes kaiken koodin pystyy jakamaan alustojen välillä. Tämän lisäksi hybridisovellukset käyttävät yleensä pelkästään webin ohjelmointikieliä, minkä vuoksi ei tarvitse olla kokemusta natiiveista mobiilisovelluksista. Tämä helppous tulee kuitenkin käyttäjäkokemuksen hinnalla, sillä hybridisovellukset eivät usein yllä aivan yhtä isoihin tehokkuuksiin. Tämän lisäksi HTML-pohjainen käyttöliittymä ei tarjoa aivan samanlaista tuntumaa kuin natiivi, vaikka lähelle voidaan

päästäkin. Hybridiratkaisut sopivat oikein hyvin sellaisiin sovelluksiin, joissa ei ole suorituskyvyltään tehokkuuskriittisiä ominaisuuksia.

Progressiiviset web-sovellukset pyrkivät tarjoamaan uusien web-standardien ja service workerien avulla optimaalisia käyttökokemuksia niin pöytäkone- kuin mobiilikäyttäjillekin. Vaikka ne eivät olekaan mobiilisovelluskaupoissa kuten muut ratkaisut, ne ovat helposti löydettävissä esimerkiksi hakukoneiden indeksointien ansiota. Mobiilisovelluskaupan välttäminen helpottaa suuresti progressiivisten web-sovelluksien ylläpidettävyyttä, sillä päivitykset tarvitsee tehdä vain yhteen paikkaan. Natiivimaisten ominaisuuksien suhteen ne rajoittuvat selaimien tarjoamiin ominaisuuksiin, mikä on nykyisin kuitenkin varsin kattava ja varmasti kasvaa tulevaisuudessa. Isoin este progressiivisten web-sovelluksien tiellä on toistaiseksi hyvin vaihteleva selaintuki. Erityisesti iOS-alustan heikko tuki PWA-ominaisuuksille hankaloittaa vielä sen kunnollista käyttöä, mutta tilanne menee varmasti parempaan päin vuoden 2018 edetessä.

JavaScript ei välttämättä sovellu hieman huonomman suorituskyvyn vuoksi aivan kaikkiin käyttökohteisiin, kuten visuaalisesti intensiivisiin peleihin. Sen tarjoama suorituskyky kuitenkin riittää useimpiin käyttökohteisiin. Jokaisella lähestymistavalla on vahvuutensa, minkä vuoksi oikean menetelmän valinta riippuu suuresti kehitettävästä sovelluksesta. Web on jatkuvassa kehityksessä, minkä vuoksi voidaan olettaa web-ratkaisujen tulevan jatkossa yhä paremmiksi ja yleisimmiksi, kun halutaan tehdä sovelluksia useille alustoille.

# LÄHTEET

[1] A. Lella, Smartphone Apps Are Now 50% of All U.S. Digital Media Time Spent, comScore.com, 2016, Saatavilla (viitattu 11.02.2018): https://www.comscore.com/Insights/Blog/Smartphone-Apps-Are-Now-50-of-All-US-Digital-Media-Time-Spent.

[2] Node.js, verkkosivu. Saatavissa (viitattu 18.03.2018): https://nodejs.org/en/.

[3] Electron, verkkosivu. Saatavissa (viitattu 18.03.2018): https://electronjs.org/.

[4] Minh Q. Huynh, Prashant Ghimire, Donny Truong, Hybrid App Approach: Could It Mark the End of Native App Domination? Issues in Informing Science and Information Technology, Vol. 14, 2017, pp. 49–65.

[5] Building Web Apps in WebView, verkkosivu. Saatavissa (viitattu 18.03.2018): https://developer.android.com/guide/webapps/webview.html.

[6] Apache Cordova, verkkosivu. Saatavissa (viitattu 11.3.2018): https://cordova.apache.org/.

[7] Ionic framework, verkkosivu. Saatavissa (viitattu 18.03.2018): https://ionicframework.com/framework.

[8] W. Oliveira, W. Torres, F. Castor, B.H. Ximenes, Native or Web? A Preliminary Study on the Energy Consumption of Android Development Models, 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, pp. 589–593.

[9] I. Malavolta, S. Ruberto, T. Soru, V. Terragni, End Users' Perception of Hybrid Mobile Apps in the Google Play Store, Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference, 2015, pp. 56–59. http://ieeexplore.ieee.org.libproxy.tut.fi/stamp/stamp.jsp?arnumber=7283028.

[10] T. Ottochino, React Native: Bringing Modern Techniques To Mobile, 2015, Saatavilla (viitattu 4.3.2018): https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/.

[11] React Native Web, verkkosivu. Saatavissa (viitattu 4.3.2018): https://github.com/necolas/react-native-web.

[12] A. Russell, Progressive Web Apps: Escaping Tabs Without Losing Our Soul, 2015, Saatavilla (viitattu 17.03.2018): https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

[13] M. Gaunt, P. Kinlan, Wep App Manifest, verkkosivu. Saatavissa (viitattu 17.03.2018): https://developers.google.com/web/fundamentals/web-app-manifest/.

[14] M. Gaunt Service Workers: an Introduction, verkkosivu. Saatavissa (viitattu 17.03.2018): https://developers.google.com/web/fundamentals/primers/service-workers/.

[15] J. Campos, When Less is More: The Case For Progressive Web Apps, MediaPost, 2018, Saatavilla (viitattu 17.03.2018): https://www.mediapost.com/publications/article/314194/when-less-is-more-the-case-for-progressive-web-ap.html.

[16] M. Santoni, Progressive Web Apps : Feature compatibility based on the browser, 2018, Saatavilla (viitattu 18.03.2018): https://blog.goodbarber.com/Progressive-Web-Apps-Feature-compatibility-based-on-the-browser_a883.html.

[17] K. Pflug, K. Chinnathambi, A. Gustafson, I. Shahid, Welcoming Progressive Web Apps to Microsoft Edge and Windows 10, 2018, Saatavilla (viitattu 18.03.2018): https://blogs.windows.com/msedgedev/2018/02/06/welcoming-progressive-web-apps-edge-windows-10/#hsh86bBfkc84HweD.97.

[18] What's New in Safari 11.1, verkkosivu. Saatavissa (viitattu 17.03.2018): https://developer.apple.com/library/content/releasenotes/General/WhatsNewInSafari/Articles/Safari_11_1.html.

[19] J. Cross, iOS 11.3 Features, release date and how to install, Macworld, 2018, Saatavilla (viitattu 17.03.2018): https://www.macworld.com/article/3250650/ios/ios-11-3-features-release-date-and-how-to-install.html.

[20] M. Firtman, PWAs are coming to iOS 11.3: Cupertino, we have a problem, 2018, Saatavilla (viitattu 17.03.2018): https://medium.com/@firt/pwas-are-coming-to-ios-11-3-cupertino-we-have-a-problem-2ff49fd7d6ea.

[21] Can I Use, verkkosivu. Saatavissa (viitattu 18.03.2018): https://caniuse.com/.

[22] A. Bovens, Add Progressive Web Apps to your Home screen in Firefox for Android, 2017, Saatavilla (viitattu 18.03.2018): https://hacks.mozilla.org/2017/10/progressive-web-apps-firefox-android/.

[23] Ionic PWA, verkkosivu. Saatavissa (viitattu 18.03.2018): https://ionicframework.com/pwa.

[24] Does Your Business Need Progressive Web App? [Infographic], verkkosivu. Saatavissa (viitattu 18.03.2018): https://skilled.co/resources/business-need-progressive-web-app-infographic/.

[25] Xamarin, verkkosivu. Saatavissa (viitattu 18.03.2018): https://www.xamarin.com/.

[26] Chrome dev tools, verkkosivu. Saatavissa (viitattu 22.04.2018): https://developer.chrome.com/devtools.

[27] React Native Debugging, verkkosivu. Saatavissa (viitattu 22.04.2018): https://facebook.github.io/react-native/docs/debugging.html.

[28] Appium, verkkosivu. Saatavissa (viitattu 01.04.2018): http://appium.io/.

[29] I. Malavolta, S. Ruberto, T. Soru, V. Terragni, Hybrid mobile apps in the Google play store: an exploratory investigation, Proceedings of the Second ACM International Conference on mobile software engineering and systems, IEEE Press, pp. 56–59.

[30] Perezm Sarah, Majority of U.S. consumers still download zero apps per month, says comScore, 2017, Saatavilla (viitattu 07.04.2018): https://techcrunch.com/2017/08/25/majority-of-u-s-consumers-still-download-zero-apps-per-month-says-comscore/.

[31] Google web case studies, verkkosivu. Saatavissa (viitattu 07.04.2018): https://developers.google.com/web/showcase/2017/.

[32] Jumia sees 33% increase in conversion rate, 12X more users on PWA, 2017, Saatavilla (viitattu 07.04.2018): https://developers.google.com/web/showcase/2017/jumia.

[33] What web can do today, verkkosivu. Saatavissa (viitattu 07.04.2018): https://whatwebcando.today/.

[34] W. Oliveira, R. Oliveira, F. Castor, A Study on the Energy Consumption of Android App Development Approaches, 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE, pp. 42-52.

[35] I. Malavolta, Beyond native apps: web technologies to the rescue! (keynote), Proceedings of the 1st International Workshop on mobile development, ACM, pp. 1-2.

[36] I. Malavolta, G. Procaccianti, P. Noorland, P. Vukmirovic, Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps, 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), IEEE, pp. 35-45.

[37] Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage, 2017, Saatavilla (viitattu 07.04.2018): https://developers.google.com/web/showcase/2017/twitter.

[38] Ola drives mobility for a billion Indians with Progressive Web App, 2017, Saatavilla (viitattu 19.04.2018): https://developers.google.com/web/showcase/2017/ola.