PWA-sovellukset modernissa ohjelmistotuotannossa

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Toukokuu 2024
Lauri Lahtinen

TURUN YLIOPISTO
Tietotekniikan laitos

Lauri Lahtinen: PWA-sovellukset modernissa ohjelmistotuotannossa

TkK-tutkielma, 21 s.
Tietotekniikka
Toukokuu 2024

Mobiilisovelluskehitys on pitkään keskittynyt natiivisovelluskehitykseen, jossa sovellus kehitetään kohdealustan kanssa yhteensopivaksi alustan ohjelmointikieliä ja työkaluja käyttäen. Tämän takia eri käyttäjäkuntien saavuttamiseksi natiivisovellukset on tullut kehittää useampaan kertaan eri alustoja varten. Alustariippumattomat, selainpohjaiset web-sovellukset saavuttavat kaikki mobiilikäyttäjien laitteet yhden ja saman koodipohjan kanssa. Niiltä kuitenkin puuttuu natiivisovellukselle tyypilliset ominaisuudet, kuten asennettavuus tai kyky toimia ilman verkkoyhteyttä. Web-teknologioiden kehittymisen seurauksena on kehittynyt uudenlainen sovellustyyppi, joka pystyy tarjoamaan nämä natiivisovelluksen kanssa tunnetut ominaisuudet: progressiivinen web-sovellus, eli PWA. PWA pyrkii yhdistämään natiivisovelluksen ja tavallisen web-sovelluksen parhaat ominaisuudet, mikä tekee siitä mahdollisen kilpailijan niiden rinnalle.
Tässä tutkielmassa tarkastellaan progressiivisia web-sovelluksia sekä niiden eroja tavallisiin web-sovelluksiin ja natiivisovelluksiin mobiilisovelluskehityksen näkökulmasta. Tutkielman ensimmäisenä tavoitteena on kuvailla progressiivisen web-sovelluksen keskeisimmät kyvyt ja sen tekniset ominaisuudet sekä selvittää, mikä erottaa sen tavallisista web-sovelluksista. Toisena tavoitteena on löytää progressiivisen web-sovelluksen ja natiivisovelluksen merkittävimmät eroavaisuudet ja niistä mahdollisesti aiheutuvat hyödyt ja haitat niin sovelluskehittäjän että käyttäjän näkökulmasta. Tutkielma on toteutettu kirjallisuuskatsauksena. Selville saadaan kaksi web-teknologiaa, jotka erottavat tavalliset web-sovellukset progressiivisista web-sovelluksista. Tulokset myös osoittavat, että PWA mobiilisovellustyyppinä on varteenotettava vaihtoehto natiivisovelluksen korvaajaksi, vaikka sen kyvyissä ja hakukoneyhteensopivuudessa on vielä joitakin selkeitä puutteita.

Asiasanat: progressiivinen web-sovellus, PWA, web-sovellus, natiivisovellus, ohjelmistotuotanto

Sisällys

1 Johdanto **1**

2 Tausta **3**

2.1 Natiivisovellukset 3

2.2 Alustariippumattomat sovellukset 4

2.3 Progressiiviset web-sovellukset 6

2.3.1 Palvelunvälittäjä ja välimuisti 8

2.3.2 Asennus ja web-sovellus-manifest 11

3 PWA ja muut sovellusarkkitehtuurit **14**

3.1 Ohjemistokehitys 14

3.2 Alustariippumattomuus 15

3.3 Ylläpito ja löydettävyys 16

3.4 Suorituskyky 18

3.5 Virrankulutus 19

4 Yhteenveto **20**

Lähdeluettelo **22**

1 Johdanto

Mobiililaitteet ovat yleistyneet räjähdysmäisesti viime vuosikymmenenä. Vuoden 2015 alussa internetin käytöstä 31,16% tapahtui mobiililaitteiden kautta ja vuoden 2023 lopussa mobiililaitteden osuus internetin käytössä oli noussut jo jopa 54.67% [1]. Tämän vuoksi web-sovellusten kehitys on nopeaa ja niiden on tarvinnut kyetä vastaamaan yhä enemmän mobiilikäyttäjien tarpeisiin.

Perinteiset mobiilisovellukset ovat olleet alun perin natiivisovelluksia. Natiivisovellukset ovat mobiilisovelluksia, jotka on kehitetty yksinomaan tietylle mobiilialustalle, kuten Androidille tai iOS:lle, ja niiden käyttöönotto tapahtuu kyseisten alustojen omien sovelluskauppojen kautta. Natiivien mobiilisovellusten kehittäminen ja jakaminen on kuitenkin kallista ja aikaavievää, mikä on johtanut monet yritykset etsimään vaihtoehtoisia ratkaisuja mobiililaitteiden käytön ja sovellusten saatavuusvaatimusten kasvaessa.

Viime vuosina yhdeksi tällaiseksi ratkaisuksi on noussut Googlen kehittämä sovellustyyppi progressiiviset verkkosovellukset (PWA) [2]. PWA:t ovat web-sovelluksia, jotka pyrkivät tarjoamaan moderneja web-teknologioita käyttäen mahdollisimman samankaltaisen käyttökokemuksen kuin natiivisovellukset. Vaikka natiivisovellukset pysyvät edelleen varteenotettavana vaihtoehtona niiden tarjoamien vahvuuksien takia, PWA:n tarjoamat edut ja mahdollisuudet sen verkkopohjaisuuden vuoksi, kuten alustariippumattomuus, tekevät siitä natiivisovellusten mahdollisen kilpailijan mobiilikehityksessä.

Tässä tutkielmassa pyritään selvittämään, mikä erottaa tavalliset web-sovellukset ja PWA:t toisistaan, sekä muodostamaan kuvaus PWA:n kyvyistä ja teknisistä ominaisuuksista. Sen lisäksi tarkastellaan PWA:n ja natiivisovellusten merkittävimpiä eroavaisuuksia, sekä mahdollisia esille tulevia etuja niin kehittäjän että käyttäjän näkökulmasta. Tästä on johdettu seuraavat tutkimuskysymykset:

TK1: Mitkä ominaisuudet tekevät tavallisesta web-sovelluksesta progressiivisen web-sovelluksen?

TK2: Mitä vaikutuksia progressiivisen web-sovelluksen ominaisuuksilla on sovelluksen toimintaan verrattuna muun tyyppisiin sovelluksiin?

Työn tutkimusmateriaalia on haettu IEEE:sta, Voltterista, Web of Science:sta ja Google Scholarista. Alustava haku tehtiin Google Scholarista eri hakusanojen avulla, ja lopuksi materiaalia haettiin hakusanoilla: *("PWA" OR "progressive web application") AND ("Native" OR "native application"), ("Progressive web app" OR "Progressive web application") AND (challenges OR issues OR comparison).*

Tutkielman luvussa 2 käydään läpi pohjustavaa tietoa natiivi-, web- ja hybridisovelluksista, mitä ne ovat ja miten ne toimivat. Progressiivisista web-sovelluksista käydään samanlainen taustan läpikäynti sen teknisistä ominaisuuksista, ja tarkastellaan tarkemmin sen yksittäisiä ominaisuuksia, kuten palvelunvälittäjää, sen asentamiseen tarvittuja vaatimuksia ja offline-ominaisuuksia. Luvussa 3 pyritään vastaamaan annettuihin tutkimuskysymyksiin tutkimalla niiden ohjelmistokehitykseen tarvittavia vaatimuksia, asennusta ja päivitystä, niiden suorituskykyä, löydettävyyttä ja myös niiden virrankulutusta. Lopuksi luvussa 4 tutkimuksesta annetaan yhteenveto tuloksista ja johtopäätöksistä.

2 Tausta

2.1 Natiivisovellukset

Mobiilisovellukset voidaan jakaa eri sovellustyyppeihin niiden kehitystavan perusteella, joista perinteisin on ollut natiivikehitys. Web-teknologioiden kehittyessä viime vuosien aikana on kuitenkin nostanut suosiota alustariippumattomien web-sovellusten kehittämiseen. Mobiilisovellukset voidaan siis jakaa alustariippumattomiin ja natiiveihin sovelluksiin.

Natiivisovellus on sovellusohjelma, joka on kehitetty tietylle alustalle tai käyttöjärjestelmälle käyttäen virallisesti tuettuja ohjelmointikieliä ja työkaluja [3]. Esimerkiksi Android-laitteelle tehty sovellus on kirjoitettu Java- tai Kotlin-ohjelmointikielellä, kun taas Applen iOS-käyttöjärjestelmää hyödyntäville iPhoneille tehdyt natiivisovellukset on kirjoitettu Objective C- tai Swift-ohjelmointikielellä. Näillä ohjelmointikielillä on alustakehittäjän laajin tuki. [4]

Koska natiivisovellukset ovat suunniteltu tietyille mobiililaitteille, ne pystyvät käyttämään hyväksi laitteiden tarjoamia sisäisiä työkaluja ja ohjelmistoja toiminnassaan. Natiivisovellus pääsee ohjelmistorajapintojen (engl. application programming interface, lyh. API) avulla laajalti käsiksi muun muassa laitteen kameraan, GPS:ään, sensoreihin ja tallennustilaan. Natiivisovelluksen ongelmaton integraatio laitteen käyttöjärjestelmän kanssa mahdollistaa käyttäjälle paremman käyttökokemuksen ja erinomaisen suorituskyvyn. Sen lisäksi laitteelle asennettu natiivisovellus

pystyy toimimaan ilman internet-yhteyttä. [5]

Vaikka natiivisovellukset ovat olleet pääasiallinen vaihtoehto mobiilikehityksessä, ne vaativat alustalle dedikoituja kehittäjätiimejä sekä alustaspesifiä osaamista. Koska natiivisovellukset kehitetään käyttöjärjestelmälle sopivaksi, niille saatetaan vaatia useampia työtiimejä kehittämään erilliset sovellukset jokaiselle vaaditulle laitteelle, vaikka niiden toimintatavat olisivatkin käyttäjänäkökulmasta samanlaiset. Tämä myös johtaa kaikkien sovellusten erilliseen ylläpitoon, ja sitä kautta myös mahdollisten bugikorjausten ja päivitysten erilliseen julkaisemiseen. Työn jakaminen, sekä monen sovelluksen kehittäminen ja ylläpito ovat aikaa vievää ja kallista. Käyttäjän tulee myös ladata sovellus alustan omasta sovelluskaupasta, kuten Androidin Play Storesta tai Apple App Storesta, sekä asentaa uudet päivitykset saman palvelun kautta. [5][6]

2.2 Alustariippumattomat sovellukset

Alustariippumattomat sovellukset ovat sovelluksia, jotka toimivat yhdellä ja samalla koodipohjalla monella eri alustalla. Näitä sovelluksia ei siis tarvitse kehittää tietyille alustoille, kuten iOS tai Android. Tämä helpottaa sovelluksen kehitysprosessia ja pienentää kehitystyön kuluja. Alustariippumaton sovellus ei pysty kuitenkaan hyödyntämään laitteen sisäisiä rajapintoja tai työkaluja yhtä laajasti kuin natiivisovellus. Alustariippumattomat sovellukset voidaan jakaa web-, PWA- ja hybridisovelluksiin, joista seuraavaksi käsitellään web- ja hybridisovellukset.

Web-sovellus on sovellusohjelma, joka ei vaadi asentamista ja voidaan sen sijaan ottaa käyttöön verkkoselaimen kautta. Tätä ei kuulu kuitenkaan sekoittaa verkkosivuston kanssa: verkkosivustot määritellään joukoksi tietoa välittäviä sivuja, jotka liittyvät toisiinsa ja joihin pääsee käsiksi verkkoselaimen kautta. Web-sovellus puolestaan toimii verkkopalvelimella, jota käyttäjä käyttää verkkoselaimen kautta. web-sovellukset ovat huomattavasti monimutkaisempia kuin staattiset verkkosivus-

tot sekä arkkitehtuuriltaan että ominaisuuksiltaan, koska toisin kuin verkkosivustot, web-sovelluksen tulee tarjota interaktiivinen käyttökokemus ja se vaatii palvelimen käsittelemään käyttäjien pyyntöjä. [7]

Web-sovellusten suurin vahvuus on niiden selainpohjaisuuden mahdollistava alustariippumattomuus ja löydettävyys. Yhdellä koodipohjalla kehitetty sovellus toimii millä tahansa verkkoselaimella varustetulla laitteella ja on helposti löydettävissä selaimen kautta. Web-sovellukset käyttävät yleensä palvelunpuolen ohjelmointikieliä, kuten PHP:ta ja ASP:ta, tietojen tallennustapahtuman toteuttamiseen ja tietojen haun käsittelyyn. Useat sovellukset käyttävät myös asiakaspuolen ohjelmia, kuten JavaScriptiä, esittämään tietoa käyttäjille. Web-sovellusten raskaat prosessit suoritetaan siis verkkopalvelimen puolella ja vain käyttöliittymä renderöidään käyttäjän näytölle. Koska sovellukset toimivat web-palvelimella, kaikki käyttäjät, riippumatta laitteen käyttöjärjestelmästä, pääsevät käsiksi samaan sovellusversioon. Ja jos laitteella on selain, tukee se silloin automaattisesti selaimen tukemia tekniikoita ja tarjoamia rajapintoja. Tämä vähentää tuottajan ja ylläpitäjän kuluja, sillä muutokset web-sovelluksen ominaisuuksiin tapahtuvat suoraan palvelimen kautta. Käyttäjän ei myöskään tarvitse asentaa sovellusta käyttämälleen laitteelleen, joten se poistaa tilanrajoitukseen liittyvät ongelmat. [8]

Hybridisovelluksilla tarkoitetaan sovelluksia, jotka ovat yhdistelmä natiivi- ja web-sovellusten periaatteita. Hybiridisovelluksen ydin kehitetään web-sovelluksen tapaan eri web-teknologioiden avulla, kuten CSS, JavaScript tai HTML. Tämä ydin kuitenkin koteloidaan natiivisovelluksen sisälle, joka voidaan asentaa laitteelle niin kuin mikä tahansa mobiilisovellus. Kun sovellus avataan, se renderöi sovelluksen sisällön käyttämällä natiivisovelluksen sisälle rakennettua verkkoselainta (WebView), joka näyttää verkkosivustot käyttäjän näkymään sopivaksi. Tavallisten natiivisovellusten tapaan hybridisovellukset pääsevät käsiksi laitteen natiiveihin ominaisuuksiin ja ovat ladattavissa laitteen mobiilikaupasta. [9]

2.3 Progressiiviset web-sovellukset

Progressiiviset web-sovellukset (engl. Progressive Web Application, lyh. PWA) ovat web-sovelluksia, jotka pyrkivät tavoittelemaan natiivi- ja web-sovellusten parhaimpia ominaisuuksia. Kuten hybridisovellukset, PWA:t pystyvät käyttämään natiivisovellusten kaltaisia ominaisuuksia, kuten pääsemään käsiksi laitteen sisäisiin työkaluihin. Mutta siinä missä hybridisovellukset ovat käytännössä web-sovelluksia natiivisovelluskuoressa, PWA:t ovat pääasiassa verkkosovelluksia, jotka käyttävät nykyaikaisia verkkoteknologioita ja suunnittelumalleja antamaan käyttäjille mahdollisimman samankaltaisen käyttökokemuksen kuin natiivisovellukset. [10]

Alex Russell, senioriohjelmistosuunnittelija Google Chrome -tiimissä, kertoo blogikirjoituksessaan *"Progressive Web apps: Escaping Tabs Without Losing Our Soul"* [2], kuinka hän ja Frances Berriman olivat vuonna 2015 havainneet uudentyyppisiä verkkosivustoja, jotka tarjosivat huomattavasti paremmat käyttökokemukset kuin sen ajan perinteiset verkkosovellukset. Berriman kutsui niitä nimellä "Progressive Open Web Apps", jonka jälkeen Berriman ja Russell päätyivät käyttämään nykyään tunnettua nimeä "Progressive Web Apps". Russell määrittelee blogikirjoituksessaan PWA:n web-sovellukseksi, jolla on seuraavat yhdeksän ominaisuutta:

Responsiivisuus: Hyödyntää responsiivisia web-suunnittelutekniikoita skaalautuakseen ja toimiakseen kaiken kokoisilla ja muotoisilla laiteilla, kuten mobiililaiteen, tablettin tai tietokoneen näytöllä.

Verkkoyhteydestä riippumaton: Sovellusta on palvelunvälittäjän avulla mahdollista käyttää myös ilman verkkoyhteyttä.

Sovelluksenkaltainen vuorovaikutus: Hyödyntää mallia, joka koostuu sovelluskuoresta ja sisällöstä luodakseen natiivisovelluksen kaltaisen navigoinnin ja vuorovaikutuksen.

Tuoreus: On aina ajan tasalla, koska uudet päivitykset tulevat aina automaattisesti verkkopalvelimelta palvelunvälittäjän avulla.

Turvallisuus: Sovellus toimii vain salatun HTTPS yhteyden yli vakoilun estämiseksi.

Löydettävyys: On löydettävissä hakukoneiden avulla, ja on luokiteltu sovellukseksi web-sovellus-manifest-tiedoston avulla.

Sitouttavuus: Pystyy lähettämään käyttäjälle push-ilmoituksia laitteen oman käyttöjärjestelmän avulla.

Asennettavuus: Sovelluksen pystyy asentamaan ja lisäämään laitteen kotivalikkoon hakukoneesta.

Linkitettävyys: Uniikin URL-osoitteen ansiosta sovellus on mahdollista jakaa helposti muille käyttäjille ja siihen pääsee käsiksi selaimen kautta.

Alex Russell tarkentaa reilu vuosi myöhemmin PWA:n nimeämisen jälkeen blogikirjoituksessaan "*What, Exactly, Makes Something A Progressive Web App?*" progressiivisen web-sovelluksen vähimmäisvaatimukset [11]. Jotta selain tunnistaisi verkkosivun PWA:ksi, tulee sen täyttää tietyt tekniset vaatimukset. Jeremy Keith listaa blogissaan "*What is a Progressive Web App?*" [12] nämä kolme ydinvaatimusta:

HTTPS: Sovelluksen tulee toimia suojatun yhteyden yli. Palvelunvälittäjä mahdollistaa monet progressiivisen web-sovelluksen tarjoamista ominaisuuksista, mutta vaatii toimiakseen HTTPS-yhteyden

Palvelunvälittäjä: Sovelluksen täytyy rekistetöidä palvelunvälittäjä, joka mahdollistaa sen verkkopyyntöjen ja välimuistin täsmällisen kontrolloinnin. Tämän avulla verkkosivu saadaan toimimaan nopeasti ja luotettavasti, sekä latautumaan myös ilman toimivaa verkkoyhteyttä.

Web-sovellus-manifesti: Sovelluksen täytyy sisältää manifest-tiedosto, joka antaa tarvittavan tiedon selaimelle PWA:n asentamisen mahdollistamiseksi käyttäjän laitteelle. Se sisältää muun muassa tiedon sovelluksen kuvakkeesta ja nimestä.

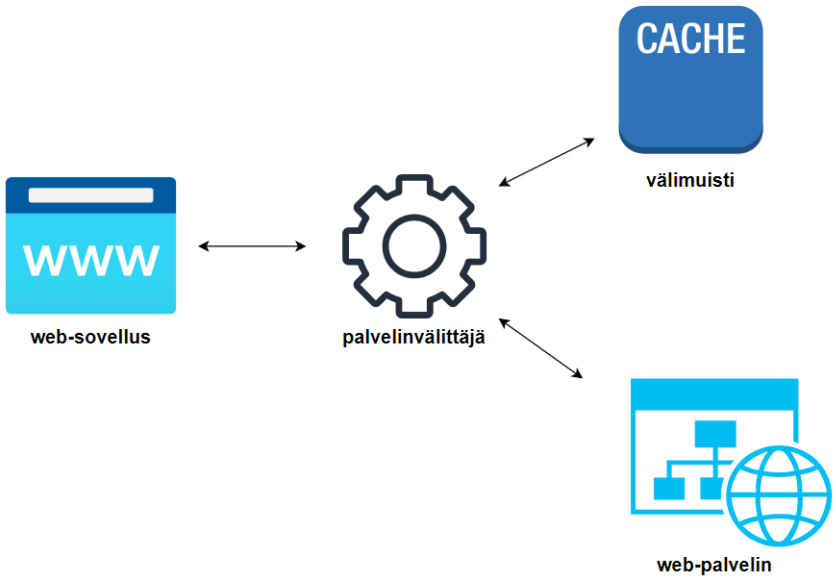
Progressiivisella web-sovelluksella voi olla monia muita ominaisuuksia, mutta ilman näitä kolmea teknistä vaatimusta sitä ei voi kutsua PWA:ksi.

2.3.1 Palvelunvälittäjä ja välimuisti

Palvelunvälittäjä (engl. service worker) toimii virtuaalisena välityspalvelimena verkolle ja selaimelle. Sen avulla on mahdollista tallentaa verkkosivuston sisältö välimuistiin ja antaa ne käyttäjälle saataville, kun käyttäjän laite on offline-tilassa. Se myös mahdollistaa osan ominaisuuksista, jotka erottavat progressiivisen web-sovelluksen tavallisesta web-sovelluksesta. Se pystyy mukaan lukien käsittelemään ilmoituksia ja suorittamaan raskaita laskelmia samanaikaisesti. [13]

Palvelunvälittäjä on JavaScript-sovellus, joka suorittaa taustaoperaatioita erillään muista web-sovelluksen osista. Se toimii eri säikeessä kuin missä web-sovellusta pyörittävä JavaScript-sovellus toimii, minkä takia sillä ei ole pääsyä verkkosivun dokumenttioliomalliin (DOM). Tämä tarjoaa uuden lähestymistavan web-ohjelmoinnissa; API on ei-estävä, eli varmistaa ettei koodin suoritus pysähdy odottaessa muiden toimintojen suorittamista, ja pystyy lähettämään sekä vastaanottamaan viestejä eri kontekstien välillä. Palvelunvälittäjän keskeisin tehtävä on siepata ja uudelleenohjata tarpeen mukaan sen kontrollialueen sisälle kuuluvien resurssien verkkopyyntöjä. Yksi yleisin käyttötarkoitus tälle on antaa palvelunvälittäjän tarkistaa, onko selaimella verkkoyhteys. Tällöin palvelunvälittäjä saattaa ohjata noutamaan sivun välimuistiversion, jos internet yhteyden muodostaminen epäonnistuu. Ja koska PWA-sovellukset käyttävät sovelluskuorta, kaikki resurssit tallennetaan automaattisesti välimuistiin. Palvelunvälittäjät toimivat ilman verkkoyhteyttä ja mah-

dollistavat välimuistin avulla toimivien verkkoyhteydettömien kokemusten luomisen. Kaikki suosituimmat verkkoselaimet ja niiden mobiiliversiot tukevat tänä päivänä palvelunvälittäjä-API:a [14]. [13][15]



Kuva 2.1: Palvelunvälittäjän rooli välipalvelimena

Resurssipyyntöjen sieppaaminen ja uudelleenohjaaminen, sekä resurssien yksityiskohtainen tallentaminen välimuistiin antavat PWA:n kehittäjille täyden hallinnan siitä, miten sovelluksen tulisi toimia tietyissä olosuhteissa [16]. Kuva 2.1 havainnollistaa palvelunvälittäjän roolia selaimessa toimivan web-sovelluksen, web-palvelimen ja välimuistin välissä toimivana kerroksena. Kun web-sovellus pyytää palvelunvälittäjän näkyvyysalueelle kuuluvaa resurssia, palvelunvälittäjä sieppaa pyynnön. Sen jälkeen palvelunvälittäjä päättää ohjaako se pyynnön laitteen paikalliseen välimuistiin, vai perinteiseen tapaan verkkoyhteyden yli web-palvelimelle. Vaikka siis web-sovelluksella ei olisi pääsyä verkkoon, palvelunvälittäjä pystyy tarjoamaan sille ainakin osan sen resursseista ja toiminnallisuuksista välimuistiohjelmointirajapinnan avulla. Näin sovelluksen käyttäjille voidaan tarjota natiivisovelluksen kaltainen käyttökokemus verkkoyhteydestä riippumatta. [17]

Turvallisuussyistä palvelunvälittäjät toimivat suojatun HTTPS-yhteyden läpi. Suurin syy tälle on se, etteivät HTTPS-yhteydet ole alttiita väliintulohyökkäyksille ja niiden syöttämille haitallisilla ja ilkimielisille koodeille. Tällaiset hyökkäykset olisivat vielä entistä pahempia, jos niille annettaisiin pääsy PWA:n tehokkaisiin ohjelmointirajapintoihin. [16]

Progressiivisen web sovelluksen tärkeimpiä ominaisuuksiin lukeutuvat sen alustariippumattomuus ja sen nopea sivujen latautuminen. **Välimuistin** oikeaoppisen käytön avulla web-sovellus on käytettävissä offline-tilassa ja se tarjoaa sisältönsä mahdollisimman nopeasti, verkkoyhteyden laadusta riippumatta. Tavoitteena on, että kaikki PWA:n toiminnallisuudet toimisivat mahdollisimman samalla tavalla niin verkkoyhteydessä kuin ilman sitä. [18]

Palvelunvälittäjä-rajapinnassa toimiva CacheStorage-API on välimuistin hallintaan tarkoitettu rajapinta. CacheStorage on matalan tason API, joka tarjoaa kehittäjille täyden kontrollin siitä, mitä välimuistissa tulee milloinkin olla ja mitä sieltä tulisi milloinkin noutaa. Välimuistin käytöksen yksityiskohtainen muokattavuus tarjoaa kehittäjälle mahdollisuuden luoda ja hyödyntää sovelluksessa useita erilaisia välimuististrategioita. [19]

Jotta välimuistin käyttö voidaan aloittaa, sovelluksen kehittäjän tulee ensin päättää, mitä resursseja välimuistiin tallennetaan. Vaikka tähän ei ole suoraa vastausta, kannattaa aloittaa resursseista, joita tarvitaan käyttöliittymän hahmottamiseen. Tällaisia tiedostoja ovat esimerkiksi pääsivun HTML-dokumentti, CSS-tiedostot, käyttöliittymän kuvat, tarvitut JavaScript-tiedostot käyttöliittymän renderöimiseen ja web-fontit. [19] Nämä tiedostot luovat kokonaisuuden jota kutsutaan sovelluskuoreksi (engl. Application shell) [20]. Välimuistiin tallennettu sovelluskuori latautuu käyttäjän mobiililaitteen näytölle nopeasti riippumatta verkkoyhteydestä.

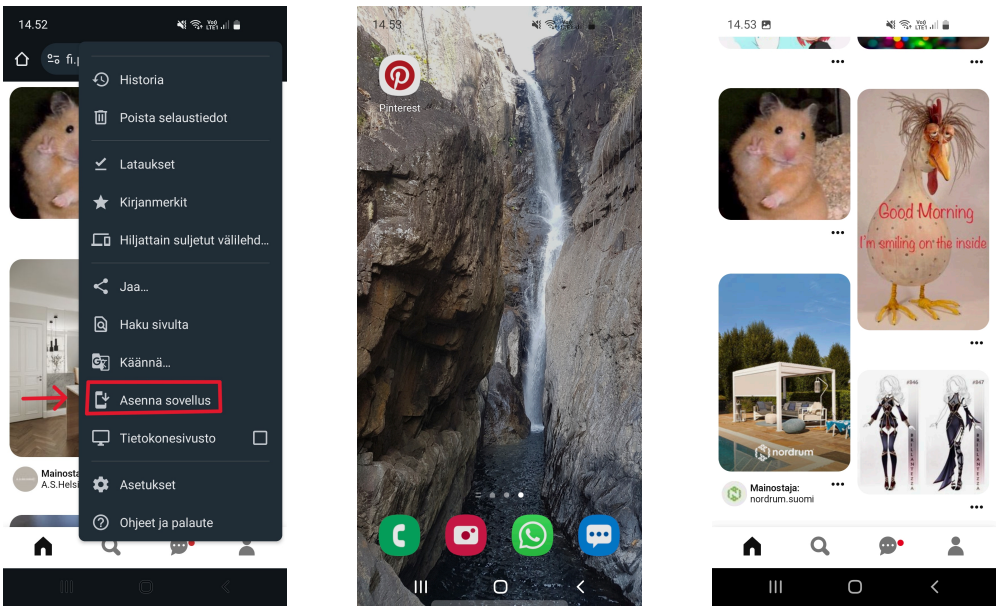
Kehittäjän vastuulla on myös päättää, milloin nämä resurssit tallennetaan välimuistiin. Vaikka yksi strategia olisi tallentaa mahdollisimman paljon dataa heti

alussa, se ei ole yleensä paras strategia. Tarpeettomien resurssien tallentaminen haaskaa osan yhteyden kaistaleveydestä ja tallennustilasta, ja voi johtaa tulevaisuudessa sovelluksen tarjoamaan käyttäjälle vanhentuneita tiedostoja. Yksi yleisimmistä strategioista on varastoida vain elintärkeät resurssit palvelunvälittäjän asennusvaiheen yhteydessä. Resurssien tallentaminen voidaan myös suorittaa ensimmäisen sivun latauksen jälkeen, kun käyttäjä navigoi sovelluksen toiselle sivulle tai kun verkkoyhteydessä ei ole muuta liikennettä.

2.3.2 Asennus ja web-sovellus-manifest

Samankaltaisesti kuin natiivisovellukset, progressiiviset web-sovellukset voidaan ladata käyttäjän laitteelle. Asennettu PWA näkyy käyttäjälle natiivisovelluksen kaltaisena sovelluksena, jonka voi käynnistää laitteen kotivalikosta natiivisovelluksen tapaan. Käyttäjän ei siis tarvitse avata verkkoselainta ja navigoida web-sovelluksen sivuille kirjanmerkin tai URL-osoitteen avulla. Verkkoselaimen tavallinen käyttöliittymä, kuten yläreunassa sijaitseva hakukenttä, voidaan poistaa kokonaan näkyvistä (kuva 2.2c). Nämä nopeuttavat sovelluksen avaamista, sekä antavat natiivisovelluksen kaltaisen käyttökokemuksen.

Kuvat 2.2a, 2.2b ja 2.2c havainnollistavat PWA-sovelluksen mahdollista asennusprosessia käyttäjän laitteelle selaimesta käsin. Kuvassa 2.2a Android-laitteen Chrome-selain tarjoaa omassa käyttöliittymässsään, päävalikossa ehdotuksen asentaa web-sovellus, jos verkkosivu tarjoaa eräänlaista manifest-tiedostoa. Sovellus löytyy asennuksen jälkeen käyttäjän laitteen kotivalikosta, kuten kuvassa 2.2b PWA-sovellus Pinterestin kanssa on tapahtunut. Kotivalikosta sovelluksen voi käynnistää natiivisovelluksen tapaan, ja sovellus näyttäytyy ilman selaimen kontrolleja, kuten kuvassa 2.2c on esitetty. Kriteerit PWA:n asennettavuudelle vaihtelevat hieman selaimen mukaan. Android-alustan Chrome- ja Firefox-selainten kriteerien yhdistelmäksi saadaan seuraavanlainen listaus [21][22]:



(a) PWA:n asennus Chromen valikon kautta

(b) PWA Android-laitteen kotivalikossa

(c) Kotivalikosta käynnistetty PWA

Kuva 2.2: PWA:n asennus laitteelle. Näyttökuvat ovat otettu Galaxy S10e-älypuhelimella Chromen versiossa 123

- Sovellusta ei ole vielä asennettu kyseiselle laitteelle.
- Sovellus toimii HTTPS-yhteyden yli.
- Sovelluksella on web-sovellus-manifesti, joka sisältää vähintään kaikki koodilistauksessa 1 näkyvät kentät.
- Sovellus rekisteröi palvelunvälittäjän, jolla on fetch-tapahtumakäsittejilä.

Web-sovellus-manifesti (engl. Web Application Manifest) on JSON-tekstitiedosto, joka sisältää PWA:han liittyvää metadataa. Se mahdollistaa PWA:n asentamisen käyttäjän laitteelle. Ilman Manifest-tiedostoa selain ei voi tarjota käyttäjälle web-sovelluksen lataamismahdollisuutta. Tiedoston sisältämä metadata kertoo verkkoselaimelle, miten PWA:n tulee käynnistyä, käyttäytyä ja miltä sen pitää näyttää. Tiedosto voi muun muassa sisältää web-sovelluksen nimen, linkit sovelluksessa käytettäviin kuvakkeisiin sekä ensisijaisen URL-osoitteen, jonka web-sovellus avaa käynnistyessään [23]. Näiden metatiedostojen avulla verkkoselaimet tarjoavat

kehittäjille keinoja luoda käyttökokemuksia, jotka ovat verrattavissa natiivisovelluksiin. Koodilistauksessa 1 on koodilistausesimerkki hyvin yksinkertaisesta web-sovellus-manifest-tiedostosta. Manifest-tiedoston käyttönotto tapahtuu sen viittauksella HTML-dokumentin `<head>`-osiossa koodilistauksen 2 mukaisesti. [24]

```
{
    "name": "Esimerkki PWA",
    "icons": [
        {
            "src": "kuvakkeet/kuvake1.png",
            "sizes": "48x48",
            "type": "image/png"
        },
        {
            "src": "kuvakkeet/kuvake2.png",
            "sizes": "72x72",
            "type": "image/png"
        }
    ],
    "background_color": "grey",
    "display": "fullscreen",
    "start_url": "https://esimerkki_url/index.html"
}
```

Ohjelmalistaus 1: Web-sovellus-manifesti esimerkki

```
<html>
   <head>
      <link rel="manifest" href="manifest.json" />
   </head>
   <body>
       ...
   </body>
</html>
```

Ohjelmalistaus 2: Manifest-tiedoston linkittäminen HTML-dokumenttiin

3 PWA ja muut sovellusarkkitehtuurit

3.1 Ohjemistokehitys

Progressiiviset web-sovellukset ovat uusi tulokas mobiilisovelluskehityksessä. Se on yhdistänyt aikaisempien mobiilisovellusarkkitehtuurien parhaimmat ominaisuudet, kuten natiivisovellusten laiteintegraation ja web-sovellusten alustariippumattomuuden. Alustariippumattomuuden takia PWA:ta ei tarvitse kehittää monelle eri alustalle, minkä vuoksi sen kehittäminen vaatii vähemmän vaivaa verrattuna natiivisovelluksiin. Artikkelissa ”Progressive Web Application Assessment Using AHP” [25] Khan, Al-Badi ja Al-Kindi mainitsevat, että monet teknologiayritykset, kuten Google ja Microsoft, ovat tukeneet vaihtoa tavallisista mobiilisovelluksista PWA-sovellusten kehittämiseen.

Web-teknologioiden osaaminen on kehittäjien keskuudessa paljon yleisempää kuin Androidin ja iOS:n natiivien ohjelmointikielien osaaminen. Vuonna 2022 JavaScript oli yleisin käytetty ohjelmointikieli koko maailmassa, kattaen 65 % kaikista kehittäjistä. Web-kehityksessä käytetyt HTML ja CSS olivat heti toisella sijalla JavaScriptin jälkeen. Kummatkin kielet ovat edelleen tilaston huipulla kehittäjien käytetyimpinä ohjelmointikielinä. Sen sijaan natiivisovelluskehityksessä usein käytetyt kielet, kuten Java, Kotlin ja Swift jäivät kauemmas kärkisijoilta noin 30,55 %:n,

9,06 %:n ja 4,65 %:n osuuksilla. Usein natiivisovelluskehityksessä tarvitaan erikoistuneempaa osaamista ja useampi työntekijä vastaamaan eri alustojen sovelluskehityksestä. [26]

Mobiilisovellusominaisuudet, kuten offline-tila, sovelluksen koko ja usean alustan tuki ovat olleet tärkeimmät huolenaiheet kehittäjille, toimittajille ja loppukäyttäjille. Khanin ja muiden tutkimuksen mukaan ohjelmistokehittäjät käyttävät kahdesta kolmeen kertaan enemmän resursseja natiiviarkkitehtuuriin perustuvien mobiilisovellusten kehittämiseen verrattuna muihin alustariippumattomiin sovelluksiin. AHP-tekniikalla (Analytical Hierarchy Process) tehdyn tutkimuksen mukaan PWA:lla on enemmän painoarvoa verrattuna natiivi-, hybridi ja web-sovelluksiin ohjelmistokoon, usean alustan tuen ja offline-käytettävyyden kriteerien kannalta. Tulevaisuudessa tarvittaisiin enemmän palautetta mobiilikäyttäjiltä PWA:n toiminnasta ja kehitysehdotuksia. [25]

3.2 Alustariippumattomuus

Yksi suuri PWA:n vahvuuksia natiivisovelluksiin verrattuna on sen alustariippumattomuus. Sen sijaan että sama sovellus tulisi kehittää useaan kertaan monelle eri alustalle eri ohjelmistokielillä, hyvin toteutettu PWA toimii yhden ja saman koodipohjan avulla kaikilla eri laitteilla [27]. Uuden sovelluksen kehittäminen on siis PWA:n tapauksessa usein huomattavasti nopeampaa ja kustannustehokkaampaa, kun sovellus halutaan saatavaksi kaikille mobiilialustoille. PWA:n kehitysprosessin pohjalla voidaan myös helposti käyttää jo olemassa olevaa verkkosivua.

PWA:n kanssa samalle tasolla alustariippumattomuudessa tulevat myös muut web-sovellukset. Kummatkaan sovellukset eivät ole rajoitettu tietylle järjestelmäalustalle ja ne ovat saataville verkkoyhteydellä monella eri laitteella. Web- ja hybridisovellukset ovat kuitenkin rajoitettu tarjottuun verkkoyhteyteen tai käytetyn verkkoselaimen kykyihin. Verkkoyhteyden katkeaminen on melko yleistä varsinkin

maaseudulla, mikä vaikuttaa saatavan internetin laatuun. Paljon verkkoa hyödyntävät web-sovellukset hidastuvat käytettäessä hitaan verkkoyhteyden yli, minkä takia ne useasti häviävät natiivisovelluksille suorituskyvyssä niiden verkkoyhteysriippuvuuden vuoksi. [8][28]

Toisin kuin muut web-sovellukset, PWA:t pystyvät toimimaan alhaisella verkkoyhteydellä, jopa ilman yhteyttä. PWA käyttää eduksi käyttäjän laitteen välimuistia tarjoamaan sovelluksen ominaisuudet niin verkkoyhteydellä kuin ilman sitä. Vaikka verkkoselaimet tarjoavat web-sovelluksille alustariippumattomuuden, se myös tuo vastapainoksi omat ongelmansa. Kaikki PWA:n ominaisuudet eivät ole välttämättä tuettu käyttäjän käyttämällä selaimella, mikä tulee vaikuttamaan sovelluksen toimintaan. Esimerkiksi Applen Safari-selain ei tue kaikkia PWA:n käyttämiseen tarvittavia API-rajapintoja [29]. PWA:ta voidaan siis käyttää vain sellaisten selainten läpi, joka tukee sen toimintaa.

3.3 Ylläpito ja löydettävyys

Verrattuna muihin web-sovelluksiin, PWA on mahdollista asentaa natiivisovelluksen tapaan käyttäjän laitteelle. Se ei ole kuitenkaan välttämätöntä PWA-sovelluksen käyttämiseksi, koska se voidaan yhtä hyvin avata suoraan verkkoselaimelta kuten tavallinen verkkosivu. Sen sijaan natiivisovellus on aina asennettava käyttäjän laitteelle ennen kuin sen pystyy käynnistämään. Asentamisen vapaaehtoisuus tekee progressiivisesta web-sovelluksesta tutustuttamisessa käyttäjän näkökulmasta nopean ja vaivattoman.

PWA:n ja natiivisovelluksen julkaisu- ja asennusprosessi ovat hyvin erilaisia toisistaan. Natiivisovellusten julkaisu ja asentaminen tapahtuu aina sovelluskaupasta, kuten Google Play:n ja Apple Appstoren kautta. PWA:t sen sijaan eivät ole sidottuja sovelluskauppoihin, sillä niiden asennus tapahtuu suoraan verkkoselaimelta. Tämä tekee niiden asentamisesta nopeampaa ja vaivattomampaa. Sama pätee myös päivi-

tyksiin, jotka PWA:n tapauksessa tapahtuvat automaattisesti ilman välissä toimivaa sovelluskauppaa. Chromen versiosta 72 lähtien PWA:t on ollut mahdollista lisätä Google Play sovelluskauppaan Trusted Web Activities -ominaisuuden avulla [30]. Android käyttäjillä on siis mahdollisuus asentaa tarpeen mukaan PWA myös alustan omasta sovelluskaupasta. Apple App Storessa ei ole samanlaista ominaisuutta.

Suurin syy sovellusten poistamiseen on yleensä laitteen tallennustila. Asennettu natiivisovellus tuo mukanaan kaikki sovelluksen tiedostot, joka vie laitteen tallennustilaa. Koska PWA:t toimivat pääosin verkossa ja ne voidaan suunnitella kevyemmäksi käyttämiensä resurssien suhteen, ne pystyvät käyttämään vähemmän laitteen tallennustilaa natiivisovellukseen verrattuna ja myös operoimaan ilman asennusta verkossa. Eräässä tutkimuksessa vertailtiin sovelluskokoja PWA:n ja natiivisovellusten välillä asennuksen yhteydessä, missä sovellusten PWA versiot osoittautuivat viemään vähemmän laitteen tallennustilaa kuin saman sovelluksen natiiviversio. Esimerkiksi Facebook-PWA-versio vei 33 KB tallennustilaa kun taas sovelluskaupasta ladattu Facebook-sovellus vei 69 MB tallennustilaa. [31]

PWA:t ovat verkkosivuja, joten hakukoneet indeksoivat ne omiin tietokantoihinsa ja niihin on mahdollista hyödyntää hakukoneoptimointia (engl. search engine optimization, lyh. SEO). [32] Progressiiviset web-sovellukset ovat siis helposti löydettävissä hakukoneen, kuten Googlen avulla. PWA:n URL-osoitteen avulla sen jakaminen on hyvin helppoa. Jaetun osoitteen vastaanottanut osapuoli pääsee heti linkin kautta kokeilemaan PWA:ta ilman minkäänlaista asennusprosessia.

PWA:t ovat kuitenkin vielä melko uusi ja tuntematon konsepti suuren yleisön keskuudessa. On siis mahdollista, että hakukoneen sijaan ensisijainen paikka uusien mobiilisovellusten etsimiseen on usein yhä alustan oma sovelluskauppa. Toisin kuin tavalliset web-sovellukset, PWA:t voivat kuitenkin olla löydettävissä ja niitä voidaan myös mainostaa sovelluskaupassa, kuten Google Play Storessa samalla tavalla kuin natiivisovelluksia [31].

3.4 Suorituskyky

Correia, Fernando, Ribeiro et al. tutkimuksessa [33] vertailtiin PWA:n ja tavallisten web-sovellusten suorituskykyä Googlen Lighthouse -työkalulla, joka on avoimen lähdekoodin automatisoitu työkalu verkkosivujen laadun mittaamiseen. Tätä varten he kehittivät kaksi sovellusta, toinen PWA ja toinen web-sovellus. Jotta pystyttäisiin tekemään vertailtava tutkimus, kummatkin sovellukset pyrittiin tekemään mahdollisimman samankaltaisiksi. Vaikka PWA sai tutkimuksessa joitakin suotuisia mittareita, tuloksia ei voita sanoa täysin vakuuttaviksi. Kummatkin sovellukset olivat hyvin yksinkertaisia, minkä takia testien suoritus loppui nopeasti ja suuria eroja ei pystytty havaitsemaan.

Toisessa tutkimuksessa [34] Rochim, Rahmatulloh, El-Akbar et al. testasivat sivujen latausnopeutta eri mobiilisovelluksilla käyttämällä GTXMetriksiä. Vaikka jokaisen sovelluksen suorituskyky oli noin samaa luokkaa, PWA:n latausnopeus hävisi natiivi- ja web-sovellukselle ajoilla 2,1 s, 1,4 s ja 1,6 s. Koska PWA:n tulee suorittaa ylimääräisiä lisäprosesseja, kuten rekisteröidä palvelunvälittäjä ja tallentaa dataa välimuistiin, sen suorituskyky ei ole yhtä nopea kuin muiden sovellusten, vaikka ero ei ole merkittävä.

Samassa tutkimuksessa oli huomattu, että PWA käynnistäminen ensimmäistä kertaa kotinäytöltä oli huomattavasti hitaampaa kuin toisella käynnistyskerralla. Ensimmäisellä käynnistyksellä PWA käytti 77 000 kilotavua dataa verkosta, kun taas toisella käynnistyksellä se käytti vain 41 700 kilotavua. Tämä johtuu siitä, kun sivu ladataan ensimmäisen kerran, välimuistitiedot tallennetaan palvelunvälittäjään. Toisella kerralla dataa kutsutaan palvelunvälittäjästä, ei verkosta. PWA:n suorituskyky siis kasvaa ensimmäisestä käyttökerrasta, koska data päivittyy verkon kautta palvelunvälittäjään käytön jälkeen ja on seuraavalla kerralla saatavilla samasta paikasta. PWA:t ovat siis kannattava valinta sovelluksille, jotka ovat useasti käyttäjän käytössä, kuten sosiaalisen median sovellus tai jokin vastaava. [34]

3.5 Virrankulutus

Mhatre, Aatmaj, Mali et al. mainitsevat tutkimuksessaan [31] PWA:n kuluttavan saman verran tai jopa vähemmän energiaa käytettäessä kuin natiivisovellukset ja tavalliset web-sovellukset. Fauzan, Krisnahati, Nurwibawa et al. huomasivat omassa tutkimuksessaan [29] PWA:n kuluttavan paljon vähemmän energiaa kuin React-natiivisovellus, kun PWA pyörii Chromen kautta. PWA:n energiajalanjälki oli kuitenkin tällöin suurempi kuin natiivisovelluksen.

Huber, Stefan, Demetz et al. vertailevat tutkimuksessaan [35] eri mobiilisovellustyyppien virrankulutusta ja toteavat myös Chromessa avattavan PWA:n kuluttavan vähemmän energiaa kuin React natiivisovellus, mutta enemmän kun vertaillaan Android-natiivisovellukseen. Tutkimuksessa paljastui myös, että PWA:n energiankulutukseen vaikuttaa merkittävästi käytettävä verkkoselain; Firefox kulutti huomattavasti enemmän energiaa kuin Chrome. Tutkimuksen kokonaisvertailussa osoittautui, että PWA:t kuluttavat kokonaisuudessa enemmän virtaa kuin natiivisovellukset.

4 Yhteenveto

Asennettavuus on yksi PWA:n keskeisimpiä ominaispiirteitä ja kriteeri, jota ilman verkkoselaimet, kuten Google Chrome, eivät myönnä web-sovellukselle PWA-nimikettä. Täten voidaan todeta, että tavallista web-sovellusta ei voi kutsua PWA:ksi ilman, että sitä voisi asentaa verkkoselaimesta käyttäjän laitteelle. Verkkoselainten asettamia teknisiä vaatimuksia PWA:n asennettavuudelle voidaan siis pitää kaikkein yksinkertaisimpana kuvauksena ominaisuuksista, jotka tekevät tavallisesta web-sovelluksesta progressiivisen web-sovelluksen.

Jotta PWA olisi asennettavissa, tulee sen käyttää suojattua HTTPS-yhteyttä, rekisteröidä toimiva palvelunvälittäjä ja sisältää web-sovellus-manifesti-tiedosto kaikkina vaadittuine kenttineen. Suojatun yhteyden käyttö voidaan kuitenkin jättää erikseen mainitsematta, koska se sisältyy jo palvelunvälittäjän omiin vaatimuksiin. Niinpä tutkimuksen ensimmäisen tutkimuskysymyksen TK1 ”Mitkä ominaisuudet tekevät tavallisesta web-sovelluksesta progressiivisen web-sovelluksen?” vastaukseksi voidaan saada tiivistettyä kaksi ominaisuutta: palvelunvälittäjä ja web-sovellus-manifesti. Ne, sekä niiden mahdollistamat asennettavuus ja verkkoyhteydestä riippumattomuus muodostavat kokonaisuuden, joka yhdistää kaikkia progressiivisia web-sovelluksia ja erottaa ne tavallisista web-sovelluksista. Kuten tutkielman luvussa 2.3 todettiin, PWA:lla on myös muita ominaispiirteitä. Nämä piirteet ovat kuitenkin vain tavoitteita joihin hyvän PWA:n tulisi pyrkiä, tai ominaisuuksia, joita tavallisella web-sovelluksella voi olla.

PWA:lla ja natiivisovelluksilla on molemmilla hyvät ja huonot puolensa. Luvussa 3 toteutetun vertailujen perusteella voidaan vastata tutkielman toiseen tutkimuskysymykseen TK2: ”Mitä vaikutuksia progressiivisen web-sovelluksen ominaisuuksilla on sovelluksen toimintaan verrattuna natiivisovelluksiin?”. Eroavaisuudet voidaan jakaa selvennyksen vuoksi vahvuuksiin ja heikkouksiin. PWA:n merkittävimpiä vahvuuksia natiivisovelluksiin verrattuna ovat niiden edullisempi ohjelmistokehitys, alustariippumattomuus, jakelun helppous, asennusprosessin vapaaehtoisuus ja sen vaatima pieni tallennustila, löydettävyys selainten hakukoneiden kautta ja niiden linkitettävyys. Heikkouksia ovat puolestaan natiivisovellusta heikompi suorituskyky ja suurempi virrankulutus, sekä rajatumpi integraatio mobiililaitteiden komponenttien kanssa. Tutkimuksessa huomattiin myös PWA:n suorituskykyyn ja virrankulutukseen vaikuttavan merkittävästi käytössä oleva selain ja sen tarjoamat API-rajapinnat.

Kumpikaan sovellustyyppi ei tarjoa täydellistä ratkaisua kaikkiin mobiilisovelluskehityksen haasteisiin, mutta oikean lähestymistavan valitseminen voi tarjota kehitysprosessille merkittäviä etuja. Ennen sovelluksen kehitystyön aloittamista kannattaakin siis tarkkaan harkita sopivin sovellustyyppi käytössä olevien resurssien sekä sovelluksen käyttötarkoituksen ja vaatimusten mukaan. Tulevaisuudessa tutkielman näkökulma voitaisiin laajentaa pöytätietokoneisiin ja niiden työpöytäsovelluksiin. Jatkotutkimusaiheena voisi lisäksi olla PWA:n turvallisuus, progressiivisten web-sovellusten suosio ja määrä tavallisiin web-sovelluksiin verrattuna, sekä PWA:n käyttäjäkokemus natiivisovelluksiin verrattuna.

Lähdeluettelo

[1] StatCounter, *Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2023*, tammikuu 2024. url: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (viitattu 26. 02. 2024).

[2] A. Russell, *Progressive Web Apps: Escaping Tabs Losing Our Soul*, kesäkuu 2015. url: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/ (viitattu 22. 03. 2024).

[3] haltu. "Natiivisovellus vai verkkosovellus -vertailu". (joulukuu 2022), url: https://www.haltu.fi/blogi/natiivisovellus-vai-verkkosovellus-vertailu (viitattu 26. 02. 2024).

[4] Z. Tarif, *The Best Programming Languages For Android And iOS Apps*, 2022. url: https://blog.back4app.com/android-and-ios-programming-languages/ (viitattu 16. 04. 2024).

[5] P. Koffer. "What is a Native Mobile App Development?" (Syyskuu 2023), url: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development- (viitattu 28. 02. 2024).

[6] opti.ro. "Advantages and Disvantages of Native Mobile Development". (lokakuu 2023), url: https://www.opti.ro/post/native-mobile-development (viitattu 28. 02. 2024).

[7] GeeksforGeeks, *Difference Between Web application and Website*. url: https://www.geeksforgeeks.org/difference-between-web-application-and-website/ (viitattu 10. 05. 2024).

[8] Javatpoint, *What is a Web Application?* Url: https://www.javatpoint.com/web-application (viitattu 08. 03. 2024).

[9] StarDust, *Hybrid Apps: The Benefits, Limitations and Consequences for your Testing Phases*. url: https://www2.stardust-testing.com/en/blog-en/hybrid-apps (viitattu 11. 03. 2024).

[10] Mozilla, "Progressive web app", lokakuu 2023. url: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (viitattu 22. 03. 2024).

[11] A. Russell, *What, Exactly, Makes Something A Progressive Web App?*, syyskuu 2016. url: https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/ (viitattu 25. 03. 2024).

[12] J. Keith, *What is a Progressive Web App?*, marraskuu 2017. url: https://adactio.com/journal/13098 (viitattu 25. 03. 2024).

[13] Mozilla, *Making PWAs work offline with Service workers*. url: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Offline_Service_workers (viitattu 30. 03. 2024).

[14] Mozilla, *ServiceWorker*. url: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker (viitattu 02. 04. 2024).

[15] M. Pedersen, *How to use service workers in progressive web apps*. url: https://techbeacon.com/app-dev-testing/how-use-service-workers-progressive-web-apps (viitattu 01. 04. 2024).

[16] Mozilla, *ServiceWorker*. url: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API (viitattu 13. 05. 2024).

[17] Google, *Service workers*. url: https://web.dev/learn/pwa/service-workers (viitattu 02. 04. 2024).

[18] Mozilla, *Caching*. url: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Caching (viitattu 25. 04. 2024).

[19] Google, *Caching*. url: https://web.dev/learn/pwa/caching (viitattu 25. 04. 2024).

[20] A. Osmani, *Instant Loading Web Apps With An Application Shell Architecture*, marraskuu 2015. url: https://medium.com/google-developers/instant-loading-web-apps-with-an-application-shell-architecture-7c0c2f10c73 (viitattu 25. 04. 2024).

[21] P. LePage, *What does it take to be installable?*, helmikuu 2020. url: https://web.dev/articles/install-criteria (viitattu 24. 04. 2024).

[22] Mozilla, *How to make PWAs installable*. url: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Installable_PWAs (viitattu 24. 04. 2024).

[23] M. Cáceres, K. R. Christiansen, M. Giuca et al., *Web Application Manifest*, marraskuu 2023. url: https://w3c.github.io/manifest/ (viitattu 04. 04. 2024).

[24] Mozilla, *Web app manifest*. url: https://developer.mozilla.org/en-US/docs/Web/Manifest (viitattu 04. 04. 2024).

[25] A. I. Khan, A. Al-Badi ja M. Al-Kindi, ”Progressive Web Application Assessment Using AHP”, *Procedia Computer Science*, vol. 155, s. 289–294, 2019, The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sus-

tainable Energy Information Technology, ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.08.041.

[26] S. Overflow, *Most used programming languages among developers worldwide as of 2023*, 2023. url: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (viitattu 16. 04. 2024).

[27] Adetunji, Oluwatofunmi, Ajaegbu et al., "Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development", *American Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 68, nro 1, s. 85–99, toukokuu 2020. url: https://asrjetsjournal.org/index.php/American_Scientific_Journal/article/view/5812.

[28] iTrobes, *What are the Advantages and Disadvantages of Web Applications?* Url: https://www.itrobes.com/what-are-the-advantages-and-disadvantages-of-web-applications/ (viitattu 08. 03. 2024).

[29] R. Fauzan, I. Krisnahati, B. D. Nurwibawa ja D. A. Wibowo, "A systematic Literature Review on Progressive Web Application Practice and Challenges", *IPTEK The Journal of Technology and Science*, vol. 33, nro 1, s. 43–58, 2022.

[30] P. Conn, *Trusted Web Activities: Quick Start*, elokuu 2019. url: https://developer.chrome.com/docs/android/trusted-web-activity/quick-start/ (viitattu 03. 05. 2024).

[31] Mhatre, Aatmaj, Mali ja Swati, "Progressive Web Applications, a New Way for Faster Testing of Mobile Application Products", teoksessa *2023 3rd Asian Conference on Innovation in Technology (ASIANCON)*, 2023, s. 1–6. DOI: 10.1109/ASIANCON58793.2023.10269806.

[32] Z. Lee, *PWA SEO: How To Boost Progressive Web Apps on Search*, marraskuu 2023. url: https://www.tigren.com/blog/pwa-seo/ (viitattu 12. 05. 2024).

[33] Correia, Fernando, Ribeiro, Óscar, Silva ja J. C., ”Progressive Web Apps Development: Study of Caching Mechanisms”, teoksessa *2021 21st International Conference on Computational Science and Its Applications (ICCSA)*, 2021, s. 181–187. DOI: 10.1109/ICCSA54496.2021.00033.

[34] R. V. Rochim, A. Rahmatulloh, R. R. El-Akbar ja R. Rizal, ”Performance Comparison of Response Time Native, Mobile and Progressive Web Application Technology”, *Innovation in Research of Informatics*, nro 1, s. 36–43, 2023, ISSN: 2656-8993. DOI: 10.37058/innovatics.v5i1.7045.

[35] Huber, Stefan, Demetz, Lukas, Felderer ja Michael, ”PWA vs the Others: A Comparative Study on the UI Energy-Efficiency of Progressive Web Apps”, teoksessa *Web Engineering*, Brambilla, Marco, Chbeir et al., toim., Cham: Springer International Publishing, 2021, s. 464–479, ISBN: 978-3-030-74296-6.