Tampereen yliopisto

Martti Grönholm

PROGRESSIIVISTEN WEB-SOVELLUSTEN HYÖDYT JA TOTEUTUS

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintutkielma
Syyskuu 2021

TIIVISTELMÄ

Martti Grönholm: Progressiivisten web-sovellusten hyödyt ja toteutus
Kandidaatintutkielma
Tampereen yliopisto
Tieto- ja sähkötekniikan kandidaatin tutkinto-ohjelma, tietotekniikka
Syyskuu 2021

Tässä kandidaatintutkielmassa tutkittiin progressiivisia web-sovelluksia (PWA, Progressive Web Application). PWA on selaimessa pyörivä web-sovellus, joka mahdollistaa natiivin käyttökokemuksen. PWA:n voi asentaa suoraan laitteen kotinäytölle, se tukee push-ilmoituksia eikä se ole täysin verkkoyhteydestä riippuvainen. PWA:n kehityksessä hyödynnetään web-ohjelmointia ja -tekniikoita, kuten JavaScript-ohjelmointikieltä, HTML-merkintäkieltä sekä CSS-tyyliohjeita.

PWA vaatii toimiakseen service workerin, web app manifestin sekä HTTPS:n eli turvallisen tiedonsiirron. Service worker on JavaScript tiedosto, joka pyörii web-sovelluksen taustalla ja hoitaa taustasynkronoinnin ja push-ilmoitukset. Web app manifest on JSON-muotoinen tekstitiedosto, joka määrittää web-sovelluksen ja mahdollistaa sen asentamisen.

Työssä käsiteltiin natiivisovellusten ja web-sovellusten eroavaisuuksia, PWA:n määrittelyä ja teknisiä vaatimuksia sekä PWA:n hyötyjä ja haasteita nykypäivänä. Lisäksi työssä toteutettiin yksinkertainen progressiivinen web-sovellus Oracle Application Express -kehitysympäristössä. Toteutettu PWA analysoitiin avoimeen lähdekoodiin perustuvalla Lighthouse-työkalulla, joka on käytettävissä osana Google Chrome -selaimen laajaa web-kehittäjän työkaluvalikoimaa. Lopuksi tutkittiin vielä tarkemmin PWA:n asemaa nykypäivänä ja otettiin katsaus sen tulevaisuuteen.

Tutkimuksesta kävi ilmi, että ihmiset palaavat progressiiviseen web-sovellukseen todennäköisemmin kuin natiiviin sovellukseen. PWA on kevyt sovellus, ja sillä on hyvä suorituskyky ja nopeat latausajat.

Avainsanat: PWA, sovelluskehitys, web-ohjelmointi, progressiivisuus

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

APEX	Oracle Application Express, Oraclen ohjelmistokehitysympäristö
API	Application Programming Interface, ohjelmointirajapinta
ATP	Autonomous Transaction Processing, Oraclen tarjoama pilvitietokanta
CAGR	Compound Annual Growth Rate, kertyvä vuotuinen kasvuprosentti
CSS	Cascading Style Sheets, kaskadiset tyyliohjeet
HTML	Hypertext Markup Language, hypertekstin merkintäkieli
HTTP	Hypertext Transfer Protocol, hypertekstin siirtoprotokolla
HTTPS	Hypertext Transfer Protocol Secure, HTTP:n suojattu versio
JSON	JavaScript Object Notation, merkintäkieli
LCDP	Low Code Development Platform, vähäisen koodin kehitysalusta
OCI	Oracle Cloud Infrastructure, Oraclen tarjoama pilvipalvelu
ORDS	Oracle REST Data Services, Oraclen REST-datapalvelu
PWA	Progressive Web Application, progressiivinen web-sovellus
REST	Representational state transfer, arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen
RESTful	REST-arkkitehtuurimallia käyttävä verkkopalvelu
SDK	Software Development Kit, ohjelmistokehityspaketti
URL	Uniform Resource Locator, verkko-osoite

1. JOHDANTO

Ohjelmointitekniikat ovat kehittyneet merkittävästi vuosikymmenien aikana. Yhä useampi käyttää mobiililaitteita ja niiden tarjoamia sovelluksia. Niinpä sovelluskehityksestä on tullut varsin merkittävä ohjelmoinnin suuntaus nykypäivänä. Aikaisemmin mobiilisovellukset ovat olleet pelkästään natiiveja. Natiivit sovellukset kehitetään suoraan tietylle alustalle, kuten Googlen kehittämälle Android-käyttöjärjestelmälle. Nykyajan tekniikoilla voidaan kuitenkin hyödyntää myös suosittua web-ohjelmointia sovelluskehityksessä. Tällaisia web-ohjelmoinnilla kehitettyjä sovelluksia kutsutaan web-sovelluksiksi.

Progressiiviset web-sovellukset (PWA, Progressive Web Application) puhuttavat paljon ohjelmointialalla tällä hetkellä. PWA:t ovat web-sovelluksia, jotka mahdollistavat sovelluksen natiivin käyttökokemuksen runsailla toiminnallisuuksillaan hyödyntämällä web-ohjelmointia ja -tekniikoita. PWA mahdollistaa muun muassa uutena web-sovelluksille push-notifikaatiot ja verkkoyhteydestä riippumattomuuden.

Tässä työssä käsitellään web-sovellusten suosion nousua natiivien sovellusten rinnalle, progressiivisten web-sovellusten määrittelyä ja teknisiä vaatimuksia sekä PWA:n hyötyjä ja haasteita. Etenkin hyötyjä, haasteita ja PWA:n toteuttamisen kannattavuutta analysoidaan yrityksen näkökulmasta. Työssä myös tutkitaan PWA:n asemaa tämän päivän sovelluskehityksessä sekä otetaan katsaus sen tulevaisuuteen. Lisäksi työssä toteutetaan yksinkertainen PWA käyttämällä Oracle Application Express (APEX) -ohjelmistokehitysympäristöä. APEX on LCDP (Low Code Development Platform), mikä mahdollistaa nopean sovelluskehityksen vähäisellä koodimäärällä. Lopuksi analysoidaan toteutettua PWA:ta Google Chrome -selaimen Lighthouse-työkalulla.

2. NATIIVIT SOVELLUKSET JA WEB-SOVELLUKSET

Sovelluskehitys ei ole lainkaan uusi ohjelmistokehityksen suuntaus. Voidaan ajatella, että ensimmäinen mobiilisovellus on kehitetty jo viime vuosituhannella, kun Hagenuk MT-2000 -matkapuhelin julkaistiin vuonna 1994 Tetris-pelillä varustettuna. Myöhemmin vuonna 1997 Nokia julkaisi Snake-pelin suositussa Nokia 6610 -matkapuhelimessa. Toisaalta ensimmäinen kosketusnäytöllä varustettu älypuhelin tuli markkinoille vuonna 2009, kun Apple julkaisi iPhone-älypuhelimen. [1] Ensimmäisen iPhonen sovellukset ja niiden kehitys vastaavat paremmin nykypäivän mobiilisovelluksia ja -kehitystä kuin aiemmat matkapuhelinten yksinkertaiset pelit ja sovellukset.

Sovelluskehitys on ollut kovasti nouseva ohjelmointikehityksen osa-alue yli vuosikymmenen ajan. Ihmiset viettävät aikaa mobiililaitteiden parissa yhä enemmän, minkä seurauksena sovelluskehityksestä on tullut hyvinkin vakiintunut ohjelmistokehityksen suuntaus.

Mobiilisovellukset voidaan jakaa kahteen pääkategoriaan: natiiveihin sovelluksiin sekä web-sovelluksiin.

2.1 Natiivit sovellukset

Natiivit sovellukset ovat perinteisin mobiilisovellusmuoto. Natiivisovellukset kehitetään suoraan sovelluskauppoihin, kuten Android-käyttöjärjestelmän Google Playhin ja Applen kehittämän iOS-käyttöjärjestelmän App Storeen. Natiivisovellusten ohjelmointikielet riippuvat täysin alustasta. Muun muassa Androidiin sovelluskehitys tehdään Java-ohjelmointikielellä hyödyntäen Eclipseen pohjautuvaa Androidin ohjelmistokehityspakettia (SDK, Software Development Kit). Applen mobiililaitteille kehitys tehdään puolestaan joko Objective-C -kielellä tai Applen omalla Swift-kielellä XCode-kehitysympäristössä. [2]

Koska sovelluksen kehittäminen usealle alustalle vaatii eri ohjelmointikielien käyttämistä, se on näin ollen aikaa vievää ja työlästä. Tästä johtuen useat natiivit mobiilisovellukset ovat käytettävissä vain yhdellä alustalla, kuten Androidissa. Tähän ongelmaan on kehitetty ratkaisuksi natiivien sovellusten rinnalle web-sovellukset, jotka mahdollistavat saman koodin hyötykäytön eri alustoilla ja laitteilla. Seuraavaksi alaluvussa 2.2 käsitellään tarkemmin web-sovelluksia.

2.2 Web-sovellukset

Web-sovellukset ovat yleistyneet vuosien aikana merkittävästi. Niiden perusideana on natiivin käyttökokemuksen saavuttaminen hyödyntäen web-ohjelmointia ja -tekniikoita. Web-sovelluksien kehittäminen mahdollistaa suositun JavaScript-kielen käytön sekä muiden web-ohjelmoinnissa käytettyjen tekniikoiden, kuten Hypertext Markup Languagen (HTML) ja Cascade Style Sheetin (CSS), hyödyntämisen.

Web-sovellukset voidaan jakaa kolmeen kehitysstrategiaan: mobiiliweb-sovelluksiin, web-pohjaisiin hybridimobiilisovelluksiin ja progressiivisiin web-sovelluksiin. [2]

Mobiiliweb-sovellukset hostataan etäpalvelimella, ne palvelevat standardiprotokollilla, kuten Hypertext Transfer Protocollilla (HTTP), ja ne ovat saatavilla uniikilla Uniform Resource Locatorilla (URL) eli URL-osoitteella. Ne ovat mobiilille optimoituja verkkosivuja, joihin pääsee käsiksi selainsovelluksilla, kuten Google Chromella. Mobiiliweb-sovelluksilla on kuitenkin vaikeuksia käsitellä raskaita grafiikoita eikä niitä voi jakaa sovelluskauppoihin. [2]

Web-pohjaiset hybridimobiilisovellukset yhdistävät parhaat puolet natiivisovelluksista ja mobiiliweb-sovelluksista. Niitä kehitetään standardeilla web-tekniikoilla, ja ne ovat käytettävissä jokaisella tuetulla mobiililaitteella, kuten Androidilla ja iOS:llä. Hybridimobiilisovelluksien suorituskyky voi kuitenkin osoittautua ongelmaksi, ja sovelluksen integrointi eri alustoille on työlästä. [2]

Uusimpana web-sovellusmuotona on PWA, joka mahdollistaa sovelluksen toimimisen ilman verkkoyhteyttä, sovelluksen pyörimisen taustalla, push-ilmoituksien tuen sekä turvallisuuden. [2]

Luvussa 3 käsitellään PWA:ta tarkemmin ja määritellään se teknisten vaatimusten avulla.

3. PROGRESSIIVISEN WEB-SOVELLUKSEN MÄÄRITTELY

Google mainitsee progressiiviselle web-sovellukselle kymmenen määrittelevää käsitettä. Nämä ovat progressiivinen, reagoiva, yhteydestä riippumaton, sovellusmainen, tuore, turvallinen, löydettävä, uudelleen sitoutuva, asennettava sekä linkitettävä. [3]

Kuitenkaan kaikki PWA:t eivät täytä näitä määritelmiä eivätkä ne siksi ole vaatimuksia. Varsinaiset tekniset vaatimukset määrittävät PWA:n ja mahdollistavat web-sovelluksen olevan aiempien määrittelyiden mukaiset.

Yksinkertaisuudessaan PWA vaatii ainoastaan kolme teknistä ominaisuutta. Nämä ovat Hypertext Transfer Protocol Secure (HTTPS), service worker sekä web app manifest. [4] Kun nämä kolme teknistä vaatimusta täyttyvät oikeaoppisesti, voidaan sovellusta pitää PWA:na.

Progressiivisilla web-sovelluksilla voi edellisten lisäksi olla muitakin ominaisuuksia, mutta edeltävät ovat ehdottomat vaatimukset PWA:lle.

Tutkitaan seuraavaksi tarkemmin edellä mainittuja kolmea teknistä vaatimusta.

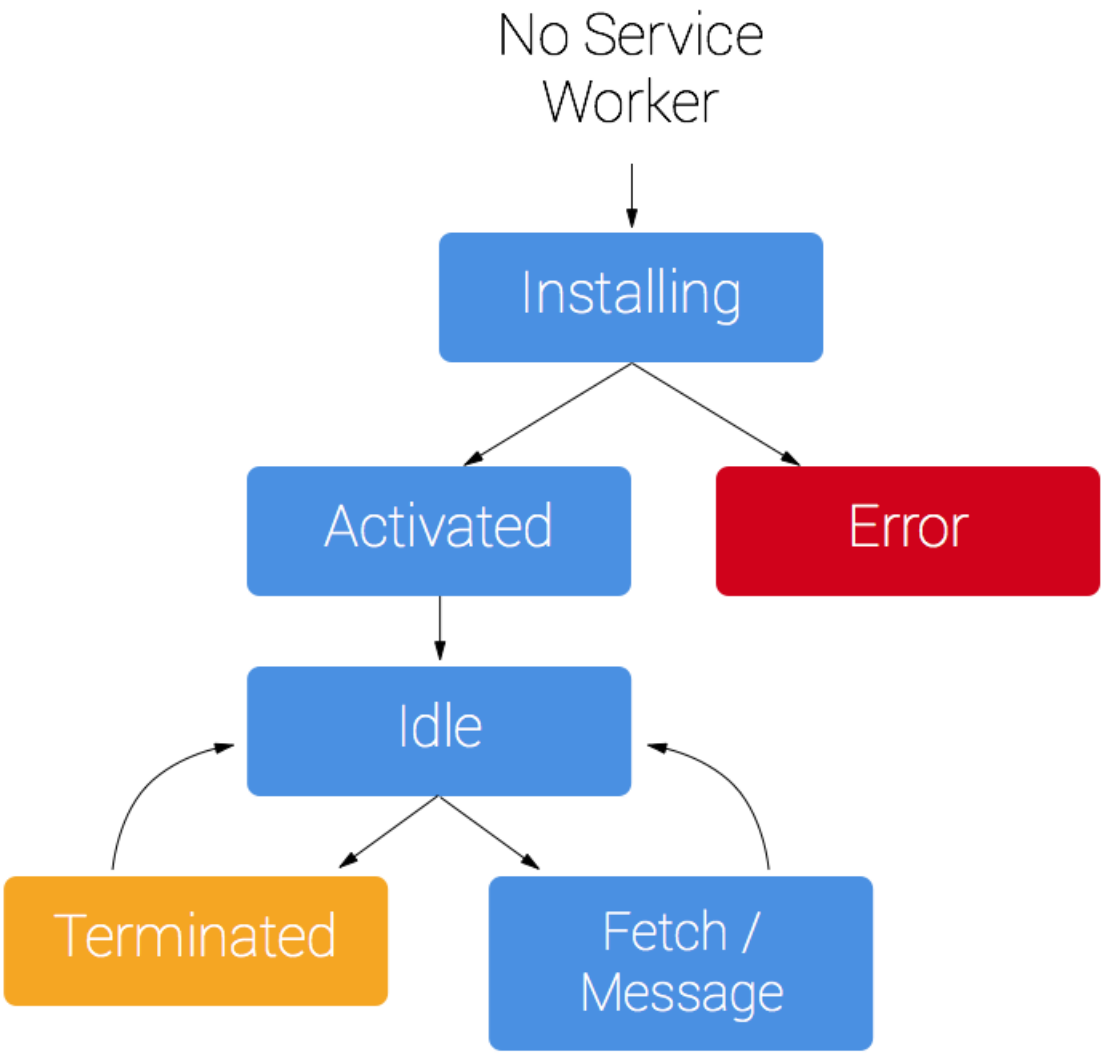
3.1 HTTPS

HTTPS on HTTP:n laajennus, joka mahdollistaa turvallisen tiedonsiirron verkossa. HTTPS suojaa tietojen eheyttä ja luottamuksellisuutta käyttäjän tietokoneen ja sivuston välillä [5]. Sillä HTTPS on vaatimus progressiivisille web-sovelluksille, se tekee niiden käyttämisestä hyvinkin turvallista. HTTPS on vaatimus PWA:lle, sillä PWA:lle edellyttävän service workerin käyttäminen sovelluksessa vaati turvallisen tiedonsiirron verkossa.

3.2 Service worker

Service worker on JavaScript-tiedosto, jota selain pyörittää web-sovelluksen taustalla. Se mahdollistaa muun muassa push-ilmoitukset, taustasynkronoinnin sekä verkkoyhteydestä riippumattomuuden välimuistin avulla. Service workerin voi rekisteröidä vain HTTPS-yhteyden omaavilla sivuilla. Tällä pyritään estämään yhteyden kaappauksia. [6]

Service workerin elinkaari on esitetty kuvassa 1.



Kuva 1. *Service workerin elinkaari [6]*

Rekisteröidyttyään service workerin onnistuneesti, se asentuu sivulle. Jos asennus tapahtuu onnistuneesti, se aktivoituu, jonka jälkeen service worker pyörii taustalla ja odottaa seuraavaa tapahtumaa. Lopuksi service worker joko lopettaa toimintansa tai se hakee tapahtumaa saatuaan verkkopyynnön. Service worker lopettaa toimintansa, jos se ei ole käytössä. Se käynnistyy myöhemmin taas uudelleen, kun sitä seuraavan kerran tarvitaan. [6]

3.3 Web app manifest

Web app manifest on JavaScript Object Notation (JSON) -muotoinen tekstitiedosto, joka määrittää web-sovelluksen sekä mahdollistaa web-sovelluksen asentamisen mobiililaitteelle [7]. Yksinkertaisuudessaan web app manifest vaatii web-sovelluksen asentami-

seen vain name ja icons -jäsenet sisältäen ainakin yhden oikeaoppisesti määritetyn ikonin [8]. On kuitenkin suotavaa määritellä web-sovellus mahdollisimman tarkasti, jotta sitä on miellyttävää käyttää eri laitteilla ja alustoilla.

4. PWA:N HYÖDYT JA HAASTEET

Web-sovellukset mahdollistavat uusien tekniikoiden käytön sovelluskehityksessä. Progressiiviset web-sovellukset taas mahdollistavat uusia ominaisuuksia web-sovelluksille. Toisaalta PWA-kehitys on vielä sen verran uutta, että ratkaistavia haasteitakin vielä on. Seuraavaksi tutkitaan tarkemmin PWA:n hyötyjä sekä haasteita.

4.1 Hyödyt

PWA mahdollistaa natiivin käyttökokemuksen hyödyntäen web-ohjelmointia ja -tekniikoita. Sillä PWA on web-sovellus, se mahdollistaa saman koodin hyötykäytön kaikilla alustoilla ja laitteilla, mikä ei ole natiiveille sovelluksille mahdollista.

Progressiivisuus on PWA:lle nimensä mukaisesti avainsana. Se tarkoittaa sitä, että mahdollisimman moni käyttäjä pystyy käyttämään web-sovellusta perustasolla. Toisin sanoen käyttäessä PWA:ta vanhalla selaimella tai heikolla internet-yhteydellä, PWA tarjoaa silti välttämättömät toiminnallisuudet. Modernia selainta käyttävät käyttäjät nopealla internet-yhteydellä puolestaan pääsevät nauttimaan PWA:n tarjoamasta täydellistä käyttökokemusta. Progressiivisuudella tarkoitetaan myös sitä, että käyttökokemus paranee progressiivisesti sitä mukaan, kun sovellus ja sen komponentit latautuvat. [9] Jo ladattuja komponentteja ei tarvitse välimuistin ansiosta enää jatkossa ladata uudelleen.

Yhtiöiden on syytä pohtia PWA:n mahdollisuutta uuden sovelluksen suunnittelussa ja kehityksessä. Jos yhtiö omistaa jo nettisivun valmiiksi, on sen muuttaminen PWA:ksi helppoa ja todennäköisesti kannattavaa. Etenkin yhtiöille, jotka tekevät rahaa verkkosivulla verkkokaupan ja mainonnan avulla, PWA:lla voi olla suurikin merkitys taloudellisesti. Yhtiöt, jotka ovat ottaneet käyttöön PWA:n, ovat nähneet kasvua konversiossa, käyttäjien sitoutumisessa, myynnissä sekä mainostuloissa. [4]

Muun muassa Pinterest-sovellus muutettuaan PWA:ksi nosti käyttäjien käyttöaikaa 40 %, käyttäjien tuottama mainostuloa 44 %, mainosten napsautusmääriä 50 % sekä ydintoimintoja 60 %. Lisäksi sovelluksen latausajat pienenivät ja tehokkuus kasvoi. [10]

Progressiiviset web-sovellukset ovat yhtä helposti löydettävissä kuin verkkosivu, sillä PWA on verkkosivu. Kaikki sovelluksen ominaisuudet ovat heti käytettävissä, eikä käyttäjän tarvitse tehdä ylimääräistä käyntiä sovelluskaupassa. Toisaalta sovellusta ei tarvitse välttämättä asentaa ollenkaan, sillä PWA toimii yhtä lailla verkkosivulla. PWA:n voi kuitenkin halutessaan asentaa laitteen kotinäytölle vain yhdellä napsautuksella suoraan verkkosivulta. [4]

Niin kuin jo alaluvussa 3.1 todettiin, on HTTPS vaatimus PWA:lle. Tämä takaa PWA:n käytöstä turvallisen. Turvallinen verkkosivu on edellytys etenkin, jos verkkosivu sisältää yksityisiä ja arkaluonteisia tietoja. Nykyään HTTPS:n käyttöönotto on myös helppoa ja edullista. Lisäksi Chrome varoittaa nykyään verkkosivuista, jotka eivät ole turvattu HTTPS:llä. Selaimet alkavat myös jatkossa vaatimaan verkkosivulta HTTPS:ää, jotta sivu pystyy käyttämään uusia web-tekniikoita. [4]

Service worker on ydinosa PWA:ta. Se mahdollistaa web-sovelluksen nopean käyttökokemuksen, sillä service workerissa määritellään mitä tiedostoja tallennetaan välimuistiin. Välimuistissa oleviin tiedostoihin pääsee käsiksi paljon nopeammin kuin tiedostoihin, jotka haetaan verkosta. Service worker mahdollistaa myös PWA:n käytön ilman verkkoyhteyttä, mihin web-sovellukset eivät ole aiemmin pystyneet. [4] Verkkoyhteydestä riippumattomuus on erityisen tärkeää alueilla, joissa verkkoyhteys on heikko. Tämä on välttämätöntä huomioida, jos web-sovelluksen käyttäjiä kohdentuu tällaisiin alueisiin, ja jos sovelluksesta haluaa mahdollisimman maailmanlaajuisen.

Toinen web-sovelluksille uusi PWA:n tukema ominaisuus on push-ilmoitukset, joka on kriittinen ominaisuus pitämään käyttäjät sitoutuneena sovellukseen. Esimerkiksi OLX, joka on yksi suurimmista verkkokohteista luokitelluille mainoksille Intiassa, nosti uudelleen sitoutumista 250 % käyttäen push-ilmoituksia, kun se julkaisi sovelluksensa PWA:na vuonna 2016. Lisäksi mainosten klikkausprosentti (CTR, Click-through rate) nousi 146 %, aika, jolloin sivusta tuli interaktiivinen laski 23 %, ja välitön poistumisprosentti, eli osuus kävijöistä, jotka poistuvat sivustolta heti ensimmäisen sivun jälkeen, laski 80 %. [11]

4.2 Haasteet

Progressiivisista web-sovelluksista on tutkitusti paljon hyötyä. Ja vaikka progressiiviset web-sovellukset mahdollistavat paljon sellaista, mihin natiivit sovellukset ja ei-progressiiviset web-sovellukset eivät tällä hetkellä pysty, on uudessa tekniikassa haasteitakin.

Etenkin natiivin sovelluksen muuttaminen PWA:ksi vaati paljon aikaa ja resursseja. Koodia pitäisi uudelleenkirjoittaa ja uusia tekniikoita opetella. Sovelluksen muuttaminen PWA:ksi ei siis ole itsestäänselvyys eikä aina välttämättä kannattavaa.

Huomionarvoista on myös se, että vanhat selaimet eivät täysin tue PWA-teknologiaa, joten PWA:n käyttökokemus joillakin selaimilla voi olla heikko.

Vaikka service worker tehostaa PWA:ta ja pienentää latausaikoja, joudutaan tiedostot ladata välimuistiin ensimmäisellä käyntikerralla. Verkkosivun tiedostojen pitkä latausaika

käyttäjän ensimmäisellä käyntikerralla suurentaa käyttäjän poistumisen todennäköisyyttä. [9] Täten onkin keskeistä saada PWA:n tiedostojen lataukset mahdollisimman nopeaksi jo heti käyttäjän ensimmäisellä vierailulla.

5. YKSINKERTAISEN PWA:N TOTEUTUS

Tässä luvussa toteutetaan yksinkertainen progressiivinen web-sovellus. Tällöin jokainen luvussa 3 määritetty tekninen vaatimus täyttyy sovellukselle, eli web-sovellus on suojattu HTTPS:llä, sillä on rekisteröity service worker ja oikein määritelty web app manifest. Lisäksi toteutettu PWA asennetaan mobiililaitteelle ja siihen toteutetaan välimuistiin tallentaminen ja välimuistista hakeminen, jotta PWA toimii ilman verkkoyhteyttä. Lopuksi analysoimme toteutettua PWA:ta Google Chrome DevToolsin Lighthouse-työkalulla.

5.1 Kehitysympäristö ja ohjelmistokehys

Kehitysympäristön ja ohjelmistokehyksen valitseminen on merkittävä osa kaikenlaista ohjelmistokehitysprojektia ja sen aloitusta. Kehitysympäristössä toteutetaan itse ohjelman kehitys, ja ohjelmistokehys muodostaa rungon ohjelmalle.

Valitsemalla kehitettävään ongelmaan sopivat työkalut, voi säästää kehitykseen käytettyä aikaa sekä rahaa. Asianmukaisella kehitysympäristöllä ja ohjelmistokehyksellä kehittäminen on sujuvaa ja lopputulos todennäköisesti halutunlainen.

Myös selaimen valintaa kannattaa pohtia. Toisaalta web-sovellusta on tärkeää testata useilla eri moderneilla selaimilla, jotta se varmasti toimii mahdollisimman laajasti, mikä on PWA:lle olennaista.

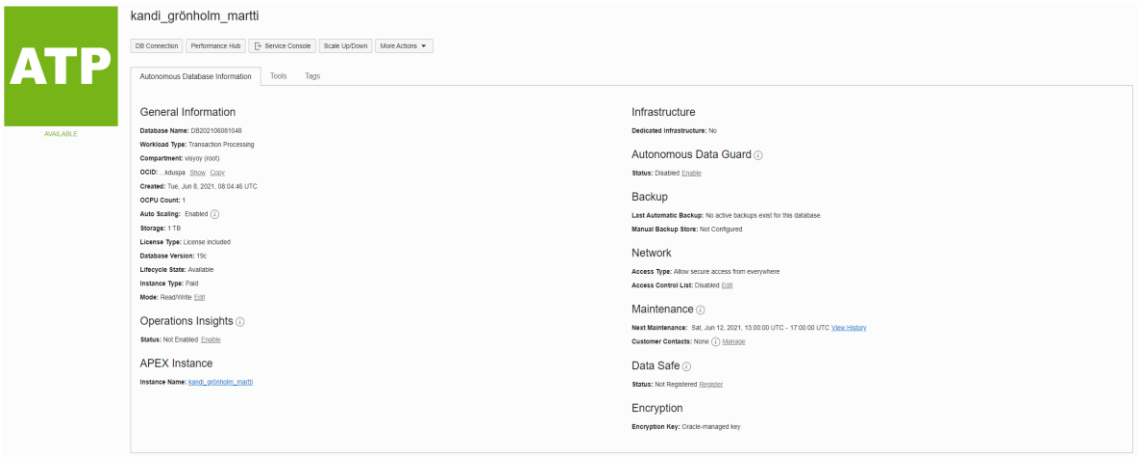
Tässä toteutuksessa käytetään Oraclen tietokannassa pyörivää APEX-ohjelmistokehitysympäristöä. APEX on LCDP, jonka ansiosta kehitys on nopeaa vähäisellä koodimäärällä. APEX ei vaadi erillistä ohjelmistokehystä, sillä se muodostaa sovellukselle rungon automaattisesti. Yksinkertaisuudessaan APEX koostuu App Builderistä sekä SQL Workshopista. App Builderissä tapahtuu itse web-sovelluksen kehittäminen, kuten sivujen luonti, komponenttien määrittäminen sekä tiedostojen ylläpito. SQL Workshopissa voidaan tehdä ja muokata tietokantatauluja käyttämällä nimensä mukaisesti SQL-kyselykieltä. SQL Workshop sisältää myös RESTful Servicen, joka kommunikoi sovelluksen kanssa. Tietokantataulut tallentuvat automaattisesti Oraclen tietokantaan. Lisäksi APEX sisältää Team Development -osion, johon sovelluskehittäjät voivat listata havaittuja ongelmia, sovellukseen haluttuja muutoksia, ja määrittää niiden kehittäjät, mikä mahdollistaa sujuvan tiimityöskentelyn. Tässä toteutuksessa keskitytään kuitenkin App Builderiin ja hieman SQL Workshoppiin.

Selaimena tässä PWA-toteutuksessa käytetään Google Chrome -selaimen DevTool-versiota, joka laajentaa normaalia Chromea sisältämällä paljon hyödyllisiä web-kehitystyö-

kaluja. Chrome DevTools sisältää muun muassa Lighthouse-työkalun, joka antaa yksityiskohtaisia analyysejä kehitettävästä web-sovelluksesta. Lighthouse kertoo lisäksi, onko tarkasteltava web-sovellus PWA, mikä on erityisen tärkeä varmistus tässä toteutuksessa.

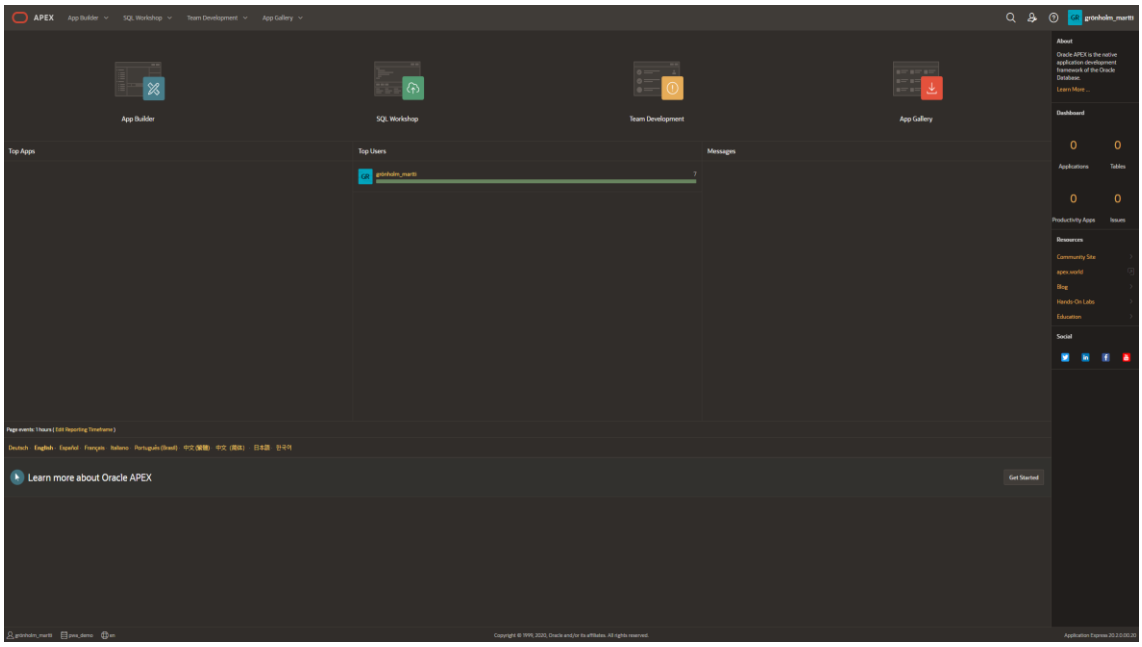
5.2 Tietokanta ja käyttöliittymä

Aloittaakseen PWA-kehityksen, täytyy ensiksi Oraclen pilvipalveluun (OCI, Oracle Cloud Infrastructure) luoda uusi tietokanta. Oracle tarjoaa useita erilaisia tietokantoja eri ongelmien ratkaisuun. Osa tietokannoista vaatii maksullisen lisenssin. Tässä toteutuksessa käytetään Oraclen tarjoamaa Autonomous Transaction Processing (ATP) -pilvitietokantaa, joka helpottaa korkeasuorituskykyisten tietokantojen operoimisen ja turvaamisen [12]. Lisäksi se nimensä mukaisesti automatisoi erilaisia toimintoja helpottaen tietokannan hallitsemista. Näkymä luodusta tietokannasta OCI:ssa on esitetty kuvassa 2.



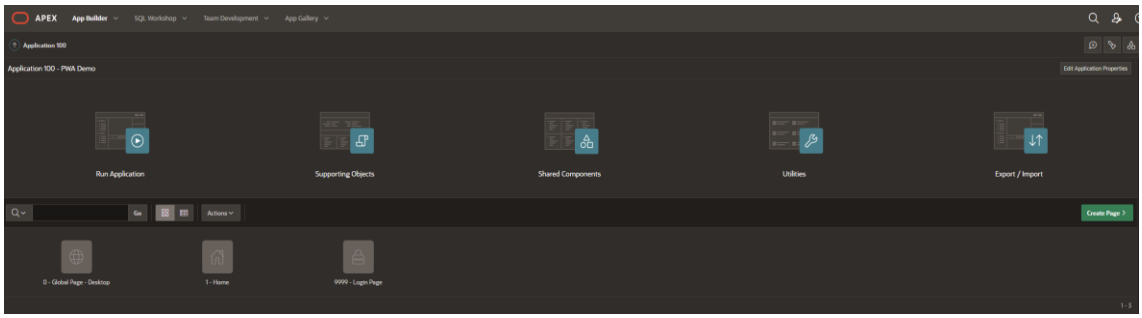
Kuva 2. *ATP:n näkymä OCI:ssa*

Tools-välilehdestä voidaan avata pilvessä toimiva APEX-kehitysympäristö, joka avaa kuvan 3 mukaisen näkymän tehtyään sinne oman käyttäjän.



Kuva 3. *Oracle APEX -aloitusnäkymä*

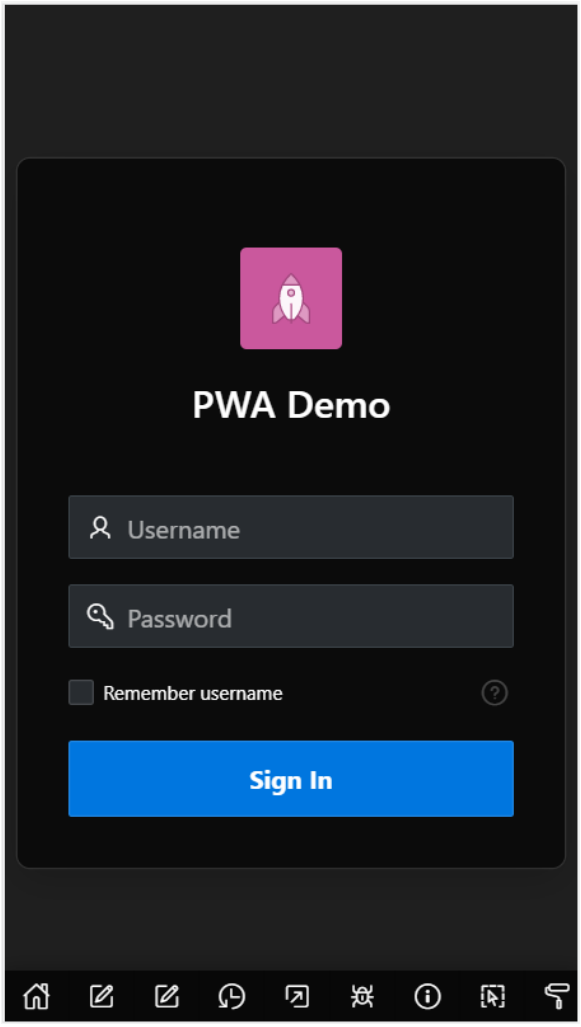
Tehdään seuraavaksi uusi sovellus App Builderistä. Kun uusi sovellus on luotu, on APEX-näkymä App Builderissä kuvan 4 mukainen.



Kuva 4. *Uusi sovellus App Builderissä*

APEX luo automaattisesti globaalin sivun, kotisivun sekä kirjautumissivun. Globaalilla sivulla voi määrittää kaikkia sovelluksen sivuja koskevia seikkoja, kuten nappeja, CSS-tyylejä tai JavaScript-koodia. Kotisivulle pääsee kirjautumisen jälkeen, joka toimii tässä sovelluksessa omilla APEX-tunnuksilla.

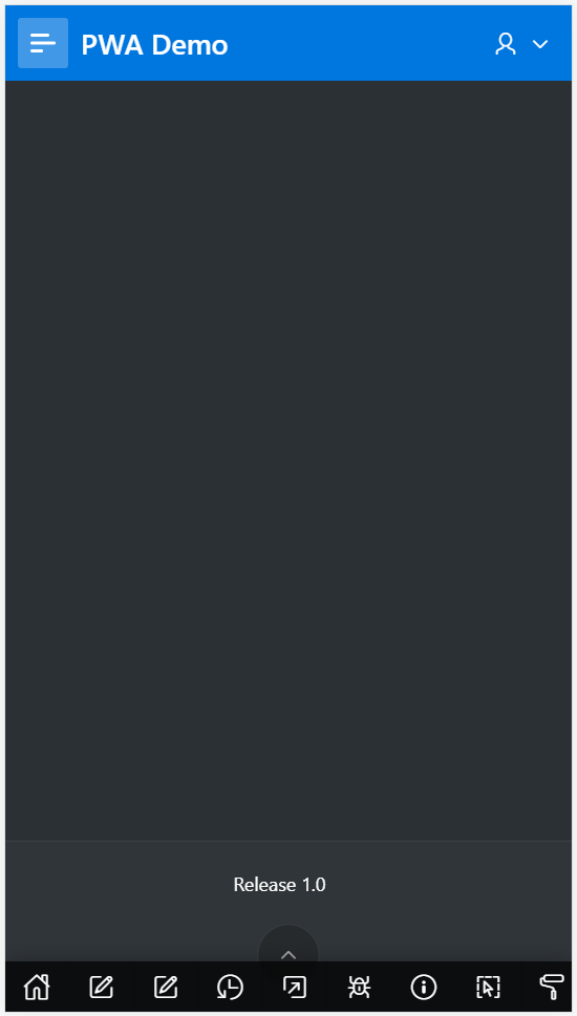
Sovelluksen kirjautumissivu on esitetty kuvassa 5.



Kuva 5. *Sovelluksen kirjautumissivu*

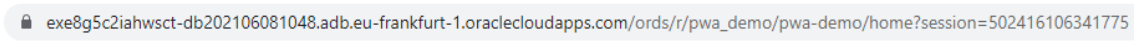
Tässä on hyvä huomata, että tämä on kehittäjän näkymä, eli alapalkin työkalut eivät ilmenny sovelluksen käyttäjille. Lisäksi näkymä on saatu mobiiliystävälliseksi hyödyntämällä Chrome DevToolsin mobiilisimulointia. Tässä toteutuksessa simuloidaan Moto G4 -mobiililaitetta. Web-sovellus toimii myös täysin tietokoneen selaimella, mutta keskitymme toteutuksessa mobiilinäkymään.

Sovelluksen kotisivu yksinkertaisuudessaan on esitetty kuvassa 6.



Kuva 6. *Sovelluksen kotisivunäkymä*

Jo nyt sovelluksesta täyttyy yksi kolmesta teknisestä vaatimuksesta, sillä APEX automaattisesti suojaa sivun HTTPS:llä. Tämä voidaan myös itse todentaa, sillä Chromen osoitepalkissa havaitaan suojattua yhteyttä esittävä lukkosymboli. Kuvassa 7 on esitetty sovelluksen verkko-osoite ja HTTPS-yhteys.



Kuva 7. *Verkko-osoite ja suojattu HTTPS-yhteys*

Lisäksi huomataan, että APEX tekee web-sovelluksen verkko-osoitteesta melko pitkän ja epäselkeän. Osoite voidaan toki sovelluksen julkistaessa muuttaa halutunlaiseksi eri keinoin, mutta se ei ole oleellista tässä toteutuksessa.

5.3 Vaadittavat tiedostot

Jotta web-sovelluksesta saadaan PWA, täytyy seuraavaksi sillä olla toimiva service worker sekä web app manifest. Yksinkertaisuudessaan web app manifest on ohjelman 1 mukainen.

```
{
    "name": "PWA Demo",
    "short_name": "PWA Demo",
    "start_url": "https://exe8g5c2iahwsct-db202106081048.adb.eu-frankfurt-1.oraclecloudapps.com/ords/r/pwa_demo/pwa-demo/home",
    "scope": "/ords/",
    "display": "standalone",
    "orientation": "portrait-primary",
    "theme_color": "#ffffff",
    "background_color":"#ffffff",
    "gcm_sender_id": "103953800507",
    "icons": [
    {
        "src": "icon-512.png",
        "sizes": "512x512",
        "type": "image/png"
    }]
}
```

Ohjelma 1. *manifest.json*

Icons-osion *icon-512.png* on tässä tapauksessa APEXin valmiskuva 512x512 pikselikoossa. On hyvä käytäntö, esittää sama kuva eri kokoisina, jotta sitä voi käyttää skaalautuvasti eri paikoissa. Tässä yksinkertaisessa PWA-toteutuksessa käytetään vain yhtä kuvan kokoa.

Service worker puolestaan yksinkertaisimmillaan ainoastaan kuuntelee *activate*-tapahtumaa:

```
self.addEventlistener('activate', event => {
    return self.clients.claim();
});
```

Nimetään tämä tiedosto *sw.js*:ksi. Service worker aktivoituu, kun erillinen JavaScript-tiedosto kutsuu sitä. Tehdään vielä erillinen *app.js*-tiedosto tätä varten, joka on esitetty ohjelmassa 2.

```
if ("serviceWorker" in navigator) {
    if (navigator.serviceWorker.controller) {
        console.log("Active service worker found, no need to register");
    } else {
        // Register the service worker
        navigator.serviceWorker
        .register("https://exe8g5c2iahwsct-db202106081048.adb.eu-frankfurt-1.oraclecloudapps.com/ords/r/pwa_demo/pwa-demo/home",
            scope: "/ords/"
        })
        .then(function (reg) {
            console.log("Service worker has been registered for scope: " + reg.scope);
        });
    }
}
```

Ohjelma 2. *app.js*

Ohjelmassa 2 *navigator.serviceWorker.register* rekisteröi service workerin, jos sitä ei ole vielä rekisteröity. Tämän jälkeen service workerin *activate*-tapahtuma toteutuu.

Jotta web-sovellus voi käyttää näitä tiedostoja, täytyy ne ladata APEXin Shared Components Static Application Files -osioon. Ladataan *icon512.png*, *manifest.json*, *sw.js* sekä *app.js* -tiedostot tänne.

Tämän lisäksi web app manifest, service worker sekä app.js vaativat pientä lisäkäsittelyä, jotta ne toimivat halutunlaisesti APEXissa.

Sekä web app manifest että app.js täytyy erikseen viitata APEXissa. Viittaukset tehdään Shared Componentsin User Inferface Attributes -osiossa. Viitataan *manifest.json* Favicon HTML -koodikentässä:

<link rel="manifest" href="#APP_IMAGES#manifest.json">

Link on HTML-attribuutti, jossa *rel* määrittää nykyisen kohteen ja viitatun resurssin yhteyden, ja *href* on URL-osoite viittauksen kohteeseen [13], eli tässä toteutuksessa APEXin Static Filesiin.

Lisätään myös samaiseen Favicon HTML-kenttään koodit:

<link rel="apple-touch-icon" href="#APP_IMAGES#icon-512.png" sizes="512x512">

<meta name="theme-color" content="#ffffff">

Lighthouse-työkalu analysoi näitä kahta attribuuttia myöhemmin, joten on hyvä käytäntö lisätä ne Faviconiin. *Apple-touch-icon* -attribuutti mahdollistaa Applen iOS-mobiilikäyttöjärjestelmässä ikonin järkevän sommittelun, ja *theme-color* -attribuutilla määritetään osoitepalkin väri.

Tämän jälkeen viitataan *app.js* JavaScript File URLs -osiossa:

    #APP_IMAGES#app.js

Service worker vaatii vielä vähän erityisempää kohtelua. Se täytyy APEXissa erikseen hakea ORDS:n (Oracle REST Data Services) kautta. Tätä varteen pitää APEXissa mennä RESTful Services -välilehteen ja tehdä sinne uusi module, template sekä GET-handler, johon lähdekoodiksi kirjoitetaan Ohjelma 3:en mukainen PL/SQL koodipätkä.

PL/SQL on Oraclen kehittämä laajennus SQL-kyselykielelle, joka lisää SQL:ään proseduraalisuutta. PL/SQL mahdollistaa muun muassa silmukoinnin sekä ehdollisia lauseita, mitä SQL ei tue.

```
declare
    v_clob blob;
begin
    apex_session.create_session(
        p_app_id => 100,
        p_page_id => 1,
        p_username => 'GRÖNHOLM_MARTTI'
    );
    OWA_UTIL.mime_header('text/javascript', FALSE);
    HTP.p('Service-Worker-Allowed: /');
    OWA_UTIL.http_header_close;
    select blob_content
    into v_clob
    from wwv_flow_files
    where flow_id=100 and filename='sw.js';
    WPG_DOCLOAD.download_file(v_clob);
end;
```

Ohjelma 3. *GET-Handlerin lähdekoodi*

Nyt APEX pystyy hakemaan ja rekisteröimään service workerin web-sovellukselle.

5.4 Asentaminen laitteelle

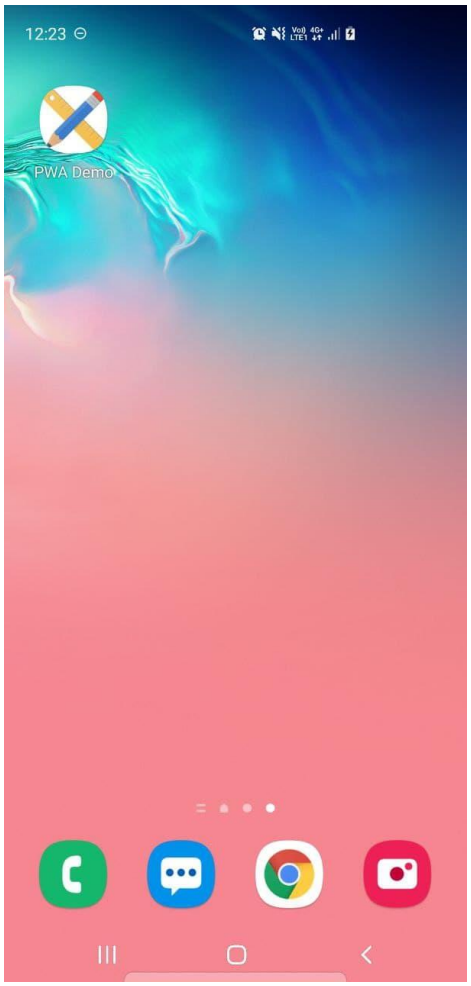
Vaikka progressiivisen web-sovelluksen asentaminen ei ole välttämätöntä, sillä se toimii täysin myös selaimella, on asentaminen yksi PWA:n ydinosa.

Aikaisemmin toteutettu web app manifest mahdollistaa toteutetun web-sovelluksen asentamisen halutulle selainta tukevan laitteen kotinäytölle.

Käynnistämällä PWA:n, laite kysyy käyttäjältä web-sovelluksen kotinäytölle lisäämisestä kuvan 8 mukaisella ponnahduspainikkeella.

Kuva 8. *Lisää PWA kotinäytölle -ponnahduspainike*

Jos käyttäjä painaa ponnahduspainiketta, asentuu web-sovellus laitteelle. Kuvassa 9 on esitetty PWA asennettuna Android-käyttöjärjestelmällä varustetun Samsung Galaxy S10+ -mobiililaitteen kotinäytöllä.



Kuva 9. *PWA asennettuna Android-mobiililaitteen kotinäytöllä*

Huomataan, että PWA asentuu laitteen kotinäyttöön kuin natiivisovellus.

5.5 Välimuisti ja verkkoyhteydestä riippumattomuus

Myös web-sovelluksen käyttäminen ilman verkkoyhteyttä on yksi olennainen PWA:n ominaisuus. Tämä saadaan lisättyä sovellukseen lisäämällä välimuisti osaksi service workeria. Välimuistiin tallennetaan sovelluksen tietoja ja komponentteja, kun verkkoyhteys on käytössä. Verkkoyhteyden menetettyään, hakee sovellus viimeisimmät tiedot välimuistista ja näyttää ne käyttäjälle.

Käytetään kahta erilaista välimuistiin tallentamisstrategiaa. Nämä ovat staattinen- ja dynaaminen välimuisti.

Staattinen välimuisti toteutetaan service workerin *install*-osioon:

```
const cacheStatic = await caches.open(cacheStaticName);

cacheStatic.addAll(apexPagesUrl)
```

Lisätään seuraavaksi service workeriin kokonaan uusi *fetch*-tapahtuma:

```
self.addEventlistener('fetch', event => {
        event.respondWith(fetchSW(event));
});
```

Lisätään *fetch*-tapahtuman asynkroniseen funktioon dynaaminen välimuisti, joka on esitetty ohjelmassa 4.

```
68  async function fetchSW(event) {
69      try {
70          const serverResponse = await fetch(event.request);
71
72          if (serverResponse) {
73              const cacheResponse = await caches.match(event.request);
74
75          if (cacheResponse) {
76              console.log('Fetching from server. No need to cache:', event.request.url);
77          } else {
78              console.log('Fetching from server, then caching request:', event.request.url);
79              const cacheDynamic = await caches.open(cacheDynamicName);
80              cacheDynamic.put(event.request.url, serverResponse.clone());
81          }
82      }
83
84      return serverResponse;
85
86      } catch (serverErr) {
87          const cacheResponse = await caches.match(event.request);
88
89          if (cacheResponse) {
90              console.log('Fetching from server failed. Fetching from cache:', event.request.url);
91              return cacheResponse;
92          }
93      }
94  }
```

Ohjelma 4. *Fetch-tapahtuma ja dynaaminen välimuisti*

Asynkronisuus on modernissa JavaScriptissä laajasti käytetty funktion kutsutyyli. Asynkronisuus JavaScriptissä mahdollistaa useamman operaation samanaikaisen suorituksen, eikä ohjelman tämän johdosta tarvitse odottaa aiemmin määritetyn koodin loppuun suorittamista. [14] Näin ollen myöhemmin määritetty operaatio voi suoriutua ennen edeltävää operaatiota.

Nyt PWA-toteutuksen service worker on valmis, ja web-sovellus pystyy tallentamaan resursseja välimuistiin ja hakemaan niitä tarvittaessa.

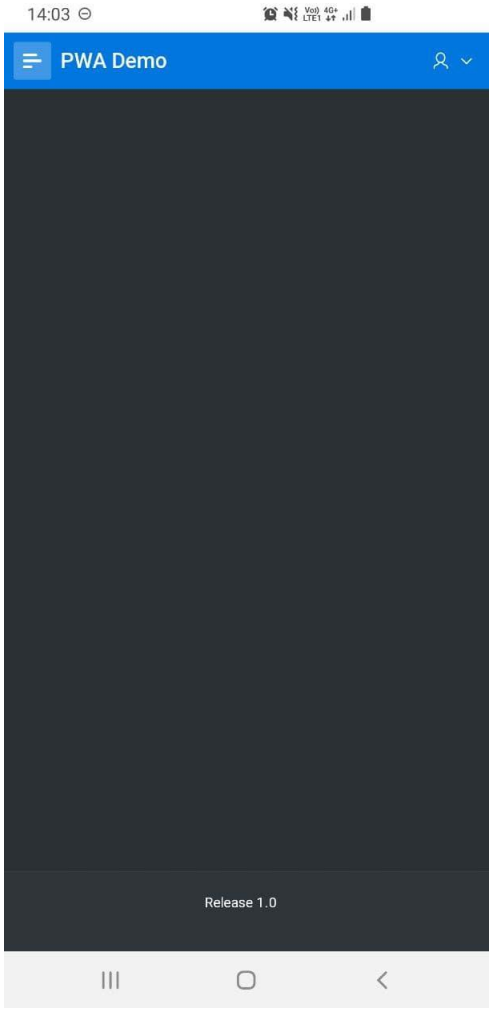
Koko service worker on esitetty vielä ohjelmassa 5.

```
const apexAppId = 100;                    // App ID
const apexPages = [0, 1, 9999];           // Array of app page IDs
const apexPagesUrl = [];                  // Array for app page URLs
const cacheStaticName = 'static-cache';   // Static cache
const cacheDynamicName = 'dynamic-cache'; // Dynamic cache

/**
 * All service worker events
 **/
self.addEventListener('activate', event => {
    console.log('Activating service worker:', event);
    return self.clients.claim();
});

self.addEventListener('install', event => {
    console.log('Installing service worker:', event);
      event.waitUntil(installSW());
});

self.addEventListener('fetch', event => {
      event.respondWith(fetchSW(event));
});

/**
 * @function installSW
 * Installs all static resources for the app shell
 **/
async function installSW() {
    let clientUrl;

      // Getting the current page URL
      await self.clients.matchAll({
            includeUncontrolled: true
      }).then(clients => {
            for (const client of clients) {
                  if (new URL(client.url).search.split(':')[0] === '?p=' + apexAppId) {
                        clientUrl = new URL(client.url);
                  }
            }
      });

    if (clientUrl) {
        console.log('Install Event processing');
        // Apply the current page URL to the array of static pages to cache
        for (const apexPage of apexPages) {
            let queryString = clientUrl.search.split(':');
            queryString[1] = apexPage;
            queryString = queryString.join(':');
            apexPagesUrl.push(clientUrl.origin + clientUrl.pathname + queryString);
        }

        // Store all static pages in the static cache
        const cacheStatic = await caches.open(cacheStaticName);
        cacheStatic.addAll(apexPagesUrl)
            .then(function () {
                console.log('Caching static files', apexPagesUrl);
            })
                .catch(function (err) {
                console.error(err);
            });
    }
}

/**
 * @function fetchSW
 * Intercepts resources, caches resources, serves resources
 **/
async function fetchSW(event) {
    try {
        const serverResponse = await fetch(event.request);

        if (serverResponse) {
            const cacheResponse = await caches.match(event.request);

        if (cacheResponse) {
            console.log('Fetching from server. No need to cache:', event.request.url);
        } else {
            console.log('Fetching from server, then caching request:', event.request.url);
            const cacheDynamic = await caches.open(cacheDynamicName);
            cacheDynamic.put(event.request.url, serverResponse.clone());
        }
    }

    return serverResponse;

    } catch (serverErr) {
        const cacheResponse = await caches.match(event.request);

        if (cacheResponse) {
            console.log('Fetching from server failed. Fetching from cache:', event.request.url);
            return cacheResponse;
        }
    }
}
```

Ohjelma 5. *sw.js*

5.6 Valmis PWA ja analyysi

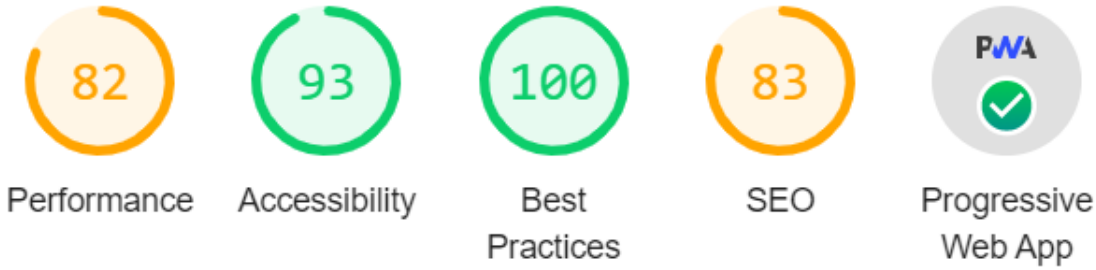
Toteutettu web-sovellus täyttää nyt kaikki PWA:n ehdot. Se on lisäksi asennettavissa käyttäjän laitteelle ja toimii ilman verkkoyhteyttä. Valmiin PWA:n kotisivunäkymä on esitetty kuvassa 10 pienten CSS-tyylimuokkausten jälkeen.



Kuva 10. *Valmis PWA*

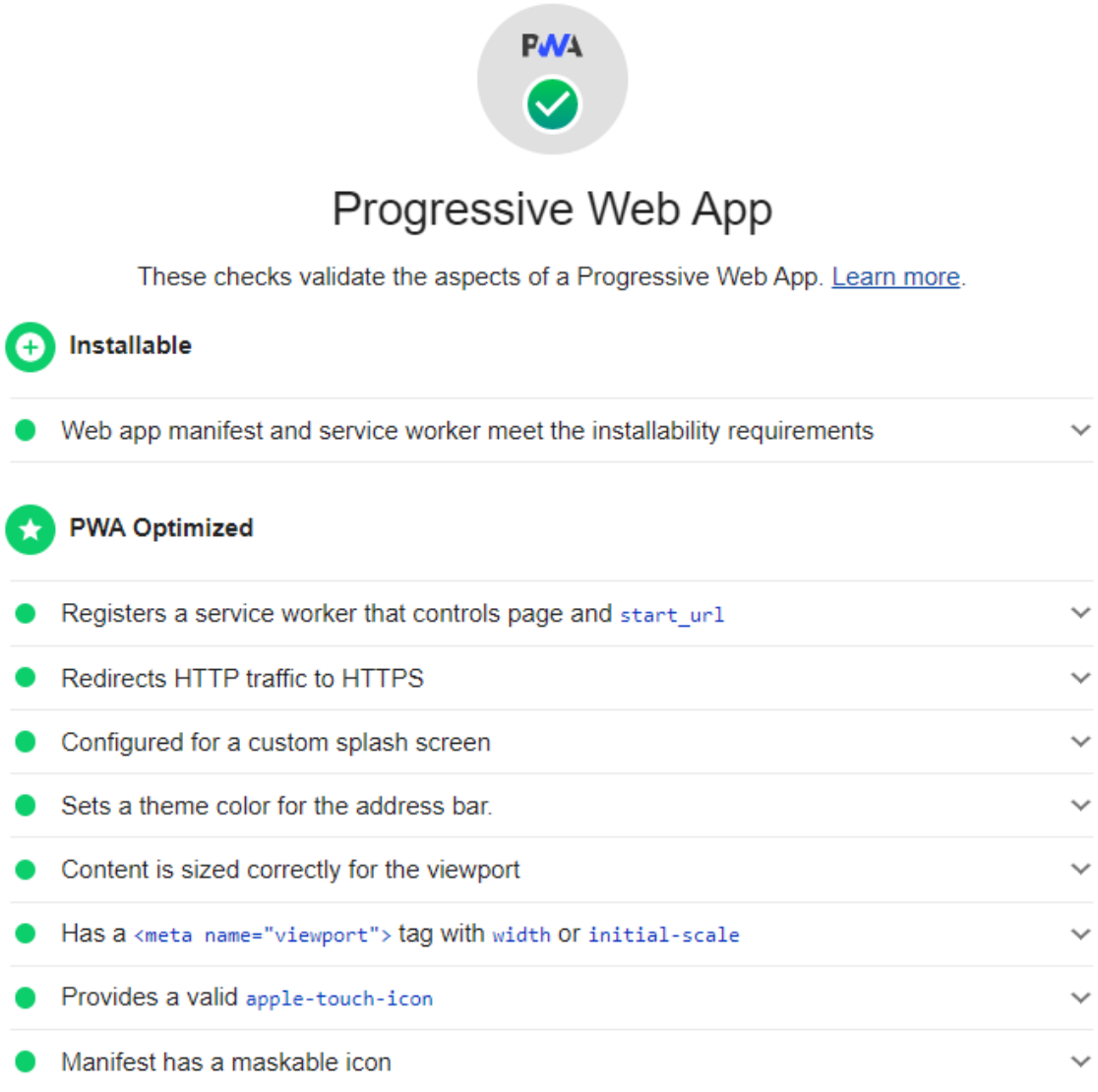
Kuva 10 on otettu kuvankaappauksena Samsung Galaxy S10+ -mobiililaitteella.

Lopuksi voidaan vielä avoimeen lähdekoodiin perustuvalla Lighthouse-työkalulla tarkistaa, että web-sovellus on varmasti PWA. Lighthouse analysoi myös sovelluksen suorituskykyä ja sen laadukkuutta yleisesti, mitkä antavat tärkeätä tietoa kehittäessä tehokasta web-sovellusta. Lighthouse antaa kuvan 11 mukaisen analyysin sovelluksesta.



Kuva 11. *Lighthouse-analyysi*

Lighthouse-analyysin mukaan PWA:n suorituskyky on kohtalainen, se on hyvin saavutettavissa eri käyttäjille, sillä on erinomainen turvallisuus, ja se on kohtalaisesti optimoitu hakukoneille. Lighthouse antaa myös tälle toteutukselle analyysin tärkeimpänä kategoriana varmistuksen sille, että web-sovellus täyttää PWA:n ehdot. Kuvassa 12 on esitettynä vielä PWA-analyysi tarkemmin.



Kuva 12. *Lighthousen PWA-analyysi*

Havaitaan, että Lighthousen PWA-analyysi ei anna ollenkaan varoituksia. Analyysin perusteella toteutettuun progressiiviseen web-sovellukseen voidaan olla tyytyväisiä.

6. PWA NYKYHETKEN JA TULEVAISUUDEN SOVELLUSKEHITYKSESSÄ

Progressiiviset web-sovellukset ovat nousseet merkittäväksi mobiilisovelluksen muodoksi viime vuosien aikana. PWA:n markkinoiden koko oli peräti 1,13 miljardia Yhdysvaltain dollaria vuonna 2019 [15].

Joitakin merkittäviä progressiivisia web-sovelluksia nykyään ovat Twitter, Pinterest, Trivago ja Starbucks [16]. Yhtiöt ovat saaneet positiivisia tuloksia vaihdettuaan sovelluksia PWA:ksi, ja yhä useampi sovellus nykyään on PWA.

Vaikka progressiivisista web-sovelluksista on kehittynyt vuosien aikana jo hyvinkin merkittävä sovellustyyppi, on niiden kehitys jatkuvaa.

On arvioitu, että PWA:n globaali markkina saavuttaa 10,44 miljardin Yhdysvaltain dollarin arvon vuoteen 2027 mennessä 31,9 %:n kertyvällä vuotuisella kasvuprosentilla (CAGR, Compound Annual Growth Rate). [15]

Varsinkin tietotekniikka-alan jättiläinen Google on toiminut PWA:n kehityksen edelläkävijänä ja liikkeellepanevana voimana. Se on lisännyt Chrome-selaimeensa jatkuvasti uusia ominaisuuksia, joita PWA:t tukevat. Tästä tuoreena esimerkkinä on tuleva ominaisuus, joka tulee mahdollistamaan uuden ohjelmointirajapinnan (API, Application Programming Interface) avulla PWA:n lukemaan tiedostoja käyttäen käyttöjärjestelmän omaa tiedostojärjestelmää. Google mainitsee uuden ominaisuuden dokumentaatiossa, että projektin tavoite on parantaa avoimuutta web-sovellusten ja natiivien sovellusten välillä, ja toimittaa johdonmukaisempaa käyttäjäkokemusta. [17]

7. YHTEENVETO

Tässä tutkimuksessa määriteltiin progressiivinen web-sovellus, tutkittiin sen hyötyjä ja haasteita etenkin yritysnäkökulmasta, ja toteutettiin yksinkertainen PWA. Lisäksi analysoitiin PWA:n nykypäivän asemaa hyödyntäen tutkimuksia, ja otettiin katsaus sen tulevaisuuteen.

PWA:sta on selkeitä hyötyjä verrattuna natiiviin sovelluksiin ja muihin ei-progressiivisiin web-sovelluksiin. PWA on kooltaan kevyt sovellus ja sillä on hyvä suorituskyky ja lyhyet sivujen latausnopeudet.

Tutkituista hyödyistä huolimatta, PWA:t eivät korvaa natiiveja sovelluksia. Etenkin natiivien sovellusten muuttaminen PWA:ksi vie aikaa ja resursseja. Varsinkin pienille ohjelmistoyrityksille ei välttämättä ole kannattavaa lähteä tähän muutokseen. Toisaalta valmiin nettisivun muuttaminen PWA:ksi on helppoa ja todennäköisesti kannattavaa.

Voidaan todeta, että PWA ei ole ainoastaan hetken muotisana sovelluskehityksessä. PWA on varteenotettava sovellusmuoto, ja se on täällä pysyäkseen.

LÄHTEET

[1] Chaichitwanidchakol, Pitsanu: Feungchan, Withca (October 2018). “Exploring Mobile Game Interactions”. Internation Journal of Electrical and Computer Edngineering. 8 (5): 3954-3965.

[2] I. Malavolta, Beyond native apps: web technologies to the rescue!(keynote), in: Proceedings of the 1st International Workshop on Mobile Development, ACM, 2016. Saatavissa (viitattu 7.6.2021): http://www.ivanomalavolta.com/files/papers/splash_2016_workshop_keynote.pdf.

[3] Addy Osmani, Getting Started with Progressive Web Apps, Google Developers, 2015. Saatavissa (viitattu 7.6.2021): https://developers.google.com/web/updates/2015/12/getting-started-pwa.

[4] Grigsby, Jason. Progressive Web App. 1st edition. A Book Apart, 2018. Print. Saatavissa (viitattu 7.6.2021): https://learning.oreilly.com/library/view/progressive-web-app/9781492018001/.

[5] Secure your site with HTTPS, Google Developers, n.d. Saatavissa (viitattu 24.6.2021): https://developers.google.com/search/docs/advanced/security/https?hl=en&visit_id=637601182612177218-2952084827&rd=1.

[6] Matt Gaunt, Service Workers: an Introduction, Google Developers, n.d. Saatavissa (viitattu: 21.6.2021): https://developers.google.com/web/fundamentals/primers/service-workers.

[7] Web app manifest, MDN Web Docs, n.d. Saatavissa (viitattu 21.6.2021): https://developer.mozilla.org/en-US/docs/Web/Manifest.

[8] How to make PWAs installable, MDN Web Docs, n.d. Saatavissa (viitattu 24.6.2021): https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs.

[9] Domes, Scott. Progressive Web Apps with React. Packt Publishing, 2017. Print.. Saatavissa (viitattu 24.6.2021): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_askewsholts_vlebooks_9781788296137.

[10] Addy Osmani, A pinterest progressive web app performance case study, 2017. Saatavissa: https://medium.com/dev-channel/ a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154.

[11] OLX boosts re-engagement on the mobile web by 250% with a Progressive Web App, Google Developers, 2017. Saatavissa (viitattu 24.6.2021): https://developers.google.com/web/showcase/2017/olx.

[12] Oracle Autonomous Transaction Processing, Oracle, n.d. Saatavissa (viitattu 8.6.2021): https://www.oracle.com/autonomous-database/autonomous-transaction-processing/.

[13] HTML <link> Tag, W3Schools, n.d. Saatavissa (viitattu 9.6.2021): https://www.w3schools.com/tags/tag_link.asp.

[14] General asynchronous programming concepts, MDN Web Docs. Saatavissa (viitattu 3.9.2021): https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Concepts.

[15] Emergen Research, Progressive Web Application Market . . ., 2020. Saatavissa (viitattu 7.6.2021): https://www.emergenresearch.com/industry-report/progressive-web-application-market.

[16] Tatsiana Tsiukhai, 9 Successful PWA Examples That May Insiper You, Dizzain, 2020. Saatavissa (viitattu 21.6.2021): https://www.dizzain.com/blog/insights/pwa-examples/.

[17] Mayank Parmar, Google Chrome is getting a new Progressive Web App feature, Bleeping Computer, 2021. Saatavissa (viitattu 21.6.2021): https://www.bleepingcomputer.com/news/google/google-chrome-is-getting-a-new-progressive-web-app-feature/.