TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SAKARIAS JÄRVINEN

NORMAALIN WEB-SOVELLUKSEN MUUTTAMINEN PROGRESSIIVISEKSI WEB-SOVELLUKSEKSI JA MUUTOKSEN VAIKUTUS SOVELLUKSEN KÄYTTÖKOKEMUKSEEN

Kandidaatintyö

Tarkastaja: Maria Törhönen.
Jätetty tarkastettavaksi 9. joulukuuta 2018

TIIVISTELMÄ

SAKARIAS JÄRVINEN: Normaalin web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi ja muutoksen vaikutus sovelluksen käyttökokemukseen
Tampereen teknillinen yliopisto
Kandidaatintyö, 22 sivua
Joulukuu 2018
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Maria Törhönen
Avainsanat: Progressiivinen web-sovellus, web-sovellus, service workers, pwa

Tässä työssä kuvataan olemassa olevan Angular-web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi ja tähän muutokseen kuuluvat vaiheet. Lisäksi työssä vertaillaan progressiivisen web-sovelluksen ja normaalin web-sovelluksen käyttökokemusta.

Progressiivinen web-sovellus pyrkii tekemään web-sovelluksesta mahdollisimman natiivin web-sovelluksen (Android, iOS) kaltaisen. Normaalin web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi vaatii muutosten tekemistä ja lisäkirjastojen asentamista sovellukseen. Tässä työssä käydään läpi nämä askeleet, joita muutokseen tarvitaan Angular-sovelluksen osalta.

Lisäksi työssä vertaillaan normaalin web-sovelluksen ja progressiivisen web-sovelluksen käyttökokemusta empiirisesti sekä toteuttamalla mittauksia. Työssä tutkitaan minkälainen vaikutus progressiivisella toteutuksella on sovelluksen tehokkuuteen ja sitä kautta käyttökokemukseen. Työssä hyödynnetään olemassa olevaa Angular-sovellusta, joka muutetaan progressiiviseksi, ja vertaillaan näin syntyvää progressiivista Angular-sovellusta ja alkuperäistä Angular-sovellusta.

Lopuksi tehdään yhteenveto tuloksista ja pohditaan progressiivisen web-sovelluksen hyötyjä ja progressiivisen toteutuksen kannattavuutta eri tyyppisille sovelluksille. Progressiivisen web-teknologian avulla saavutetaan huomattava parannus sovelluksen käyttökokemukseen ja progressiivinen toteutus suoriutuu sovelluksille toteutettavista testeistä huomattavasti paremmin.

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. PROGRESSIIVINEN WEB-SOVELLUS 2
2.1 Toimintaperiaate 2
2.2 Service workers -ohjelmakoodi 3
2.3 Ohjelmistotuki 4
3. MUUNNETTAVA JA TESTATTAVA SOVELLUS 5
3.1 Sovelluksen toimintaperiaate 5
3.2 Sovelluksen rakenne 5
3.3 Angular-web-sovellus 6
3.3.1 Angular-framework lyhyesti 6
3.3.2 Angularin peruskomponentit ja toimintaperiaate 7
3.3.3 Angular CLI -konsolikäyttöliittymä 7
3.4 Sovelluksen toiminta 7
3.5 Sovelluksen ajonaikainen ympäristö 8
4. SOVELLUKSEN MUUTTAMINEN 9
4.1 Kirjastojen asentaminen 9
4.2 Konfigurointi 9
4.2.1 Manifest.json-tiedosto 9
4.2.2 Ngsw.json-tiedosto 10
4.2.3 Sovelluksen ikonien lisääminen sovellukseen 10
4.3 Muutokset sovellukseen 11
5. TESTIEN SUORITTAMINEN 12
5.1 Testattavat suureet 12
5.1.1 Latausaika 12
5.1.2 Tiedonsiirron määrä 12
5.1.3 Palvelinkyselyiden määrä 13
5.2 Testausten toteutus 13
5.2.1 Sivun latausajan testaaminen 13
5.2.2 Sovelluksen käytön simulointi 13
5.2.3 Siirtymän mittaus 14
6. TESTAUSTULOKSET 15
6.1 Yleinen tuntuma 15
6.2 Sovelluksen latausaika 15
6.3 Sovelluksen käytön simulointi 16
6.4 Siirtymän mittaus 18
7. YHTEENVETO 19
7.1 Normaalin web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi 19
7.2 Sovellusten käyttökokemuksen vertailu 19
7.3 Progressiivinen web-teknologia 20

LÄHTEET 21

LYHENTEET JA MERKINNÄT

API	Ohjelmointirajapinta
Array	Tietotyyppi, joka sisältää kokoelman elementtejä
Backend	Sovelluksen palvelimella ajettava ohjelmakoodi
Framework	Sovelluskehitysympäristö
Frontend	Sovelluksen verkkoselaimessa ajettava ohjelmakoodi
Oauth	Sovelluksessa käytettävä autentikointiprotokolla
Push-notifikaatio	Käyttöjärjestelmän tarjoama palvelu sovelluksen käyttäjille annetaville ilmoituksille.
PWA	Progressiivinen web-sovellus
SPA (Single Page Application)	Sovellus, joka ajetaan yhdellä verkkosivulla ja jonka sisältöä muokataan ohjelmallisesti.
Teknologiapino	Sovelluksen käyttämät teknologiat

1. JOHDANTO

Modernin web-sovelluskehityksen uusimpia teknologioita ovat progressiiviset web-sovellukset. Progressiiviset web-sovellukset pyrkivät saavuttamaan natiivien mobiilisovellusten (Android [1], iOS [2]) kaltaisen käyttökokemuksen perinteisessä web-selaimessa pyörivällä sovelluksella.

Suositut JavaScript frameworkit kuten Angular [3], React [4] ja Vue.js [5] mahdollistavat progressiivisen web-teknologian tehokkaan hyödyntämisen ja tarjoavat progressiivisen web-teknologian käyttöönottoon suoraviivaiset työkalut. Näiden frameworkien käytön kasvun [6] myötä myös progressiiviset web-sovellukset ovat kasvattaneet suosiotaan.

Tässä työssä kuvataan progressiivisen web-teknologian perusteet ja miten progressiivinen web-sovellus on luotavissa olemassa olevasta web-sovelluksesta. Työssä käytetään olemassa olevaa Angular-sovellusta ja tämän avulla kuvataan progressiivisen web-sovelluksen luominen olemassa olevasta web-sovelluksesta.

Työssä pyritään selvittämään, kuinka paljon hyötyä tästä muutoksesta on sovelluksen tehokkuuteen ja sitä kautta käyttökokemukseen. Asiaa tutkitaan vertailemalla saman sovelluksen normaalia web-versiota sekä progressiivista versiota.

Työssä kuvataan ensin progressiivinen web-sovelluksen toimintaperiaate. Tämän jälkeen kuvataan työssä käytettävän web-sovelluksen toiminta ja rakenne. Seuraavaksi kuvataan normaalin web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi. Tämän jälkeen esitellään sovelluksille toteutettavat testit ja näiden testien tulokset. Lopuksi tehdään vielä yhteenveto sovelluksen muuttamisprosessista ja sovellusten vertailusta sekä analysoidaan lyhyesti progressiivista web-teknologiaa.

2. PROGRESSIIVINEN WEB-SOVELLUS

Progressiivinen web-sovellus on sovellus, joka on ajossa käyttäjän selaimessa, mutta käyttökokemukseltaan ja ominaisuuksiltaan pyrkii mahdollisimman lähelle natiiveja mobiilisovelluksia (iOS, Android). Progressiivisen web-teknologian perustana ovat seuraavat kolme sovellukselle asetettua vaatimusta: luotettavuus, nopeus ja käyttäjän sitouttaminen [7]. Progressiivinen web-sovellus pyrkii progressiivista web-teknologiaa hyödyntämällä täyttämään nämä kolme vaatimusta.

2.1 Toimintaperiaate

Osa natiivia käyttökokemusta on sovelluksen toiminta offline-tilassa ja silloin kun verkon signaali on heikko. Progressiivinen web-teknologia tarjoaa mahdollisuuden hyödyntää selaimen välimuistia sekä tallentaa ja hyödyntää välimuistista löytyvää dataa silloin kun tämä on tarpeellista. Tällaisia tilanteita ovat esimerkiksi, kun datan lataaminen palvelimelta ei onnistu, lataaminen olisi verkon kautta kohtuuttoman hidasta tai kun data on luoneeltaan sellaista, että sen lataaminen jatkuvasti uudestaan on sovelluksen käyttökokemuksen kannalta raskasta ja merkityksetöntä.

Edellä mainitun toiminnallisuuden toteuttamiseen progressiivinen web-teknologia tarjoaa web-sovelluksen taustalla ajettavan ohjelmakoodin, joka mahdollistaa tehokkaan ja älykkään selaimen välimuistin hyödyntämisen sekä ohjelman toiminnan myös offline-tilassa. Tätä sovelluksen taustalla toimivaa ohjelmakoodia kutsutaan nimellä service workers [8].

Edellä mainittujen ominaisuuksien lisäksi progressiivinen web-sovellus käyttäytyy mobiililaitteissa kuten natiivi sovellus. Progressiivinen web-sovellus voidaan ladata mobiililaitteeseen omana sovelluksenaan ja avata omasta ikonistaan. Tällöin progressiivinen sovellus käynnistyessään on ajossa verkkoselaimessa, mutta selaimen osoiterivi ja valikot ovat piilotettuna ja sovelluksen sisältö täyttää koko ruudun ja näyttäytyy käyttäjälle aivan kuten natiivi sovellus.

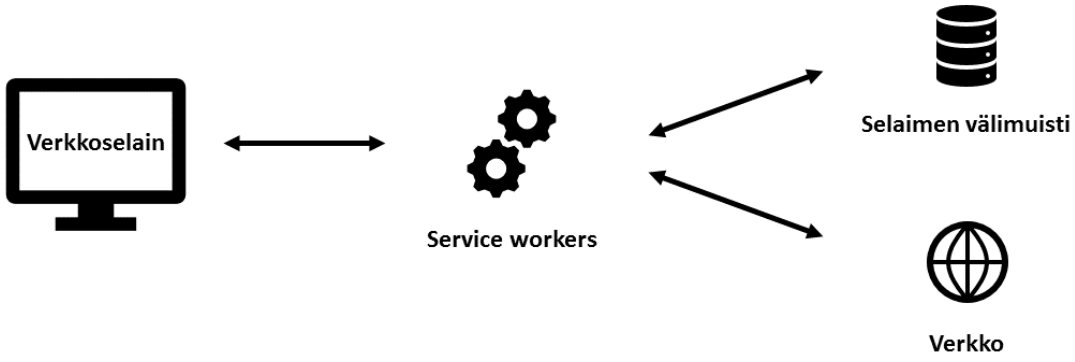
Progressiiviseen web-teknologiaan luetaan kuuluvaksi myös erilaiset rajapinnat, joilla web-sovellukset pystyvät hyödyntämään mobiililaitteiden laitteistorajapintoja, kuten kameraa, GPS-paikanninta ja mobiilikäyttöliittymän push-notifikaatioita. Nämä rajapinnat jätetään tässä työssä tarkastelun ulkopuolelle.

2.2 Service workers -ohjelmakoodi

Progressiivisen web-sovelluksen perustan rakentaa service workers -ohjelmakoodi. Service workers on ohjelmakoodi, joka toimii käyttäjän selaimeen ladattavan sovelluksen taustalla. Service workers -ohjelmakoodi on siis osa sovelluksen frontend-ohjelmakoodia.

Service workers -ohjelmakoodi toteuttaa ohjelmoijan määrittelemän välimuistin käytön automaattisesti. Ohjelmoijan ei tarvitse siis sovelluksen tasolla kommunikoida service workers -ohjelmakoodin kanssa vaan service workers -ohjelmakoodi osaa itse lukea sovelluksen ja palvelimen kommunikaation. Näin ollen sovelluksen olemassa olevat API-kutsut toimivat myös service workers -ohjelmakoodin kanssa ilman muutoksia.

Käytännön tasolla service workers -ohjelmakoodi toimii sovelluksen taustalla proxyn eli välityspalvelimen kaltaisesti. Service workers -ohjelmakoodin toiminta on havainnollistettuna kuvassa 1.



Kuva 1: Service workers-ohjelmakoodin toiminta välityspalvelimena [9]

Service workers -ohjelmakoodi toimii välikätenä kaikessa sovelluksen kommunikaatiossa palvelimelle. Ohjelmakoodi päättelee asetustensa ja verkon ominaisuuksien mukaan parhaan tavan käsitellä sovelluksen tekemät palvelinkyselyt.

2.3 Ohjelmistotuki

Progressiivisen web-sovelluksen mahdollistava service workers -ohjelmakoodi ei ole tuettuna kaikissa selaimissa. Tuki ohjelmakoodille on tullut viimeisen kahden vuoden aikana käytetyimpien selaimien uusimpiin versioihin.

Service workers -ohjelmakoodi on tällä hetkellä tuettuna seuraavien selaimien viimeisimmissä versioissa: Chrome, Firefox, Edge, Safari, Opera, UC Browser (Android-versio) ja Samsung Internet [10].

3. MUUNNETTAVA JA TESTATTAVA SOVELLUS

Tässä työssä muutos normaalista web-sovelluksesta progressiiviseksi web-sovellukseksi toteutetaan olemassa olevalle normaalille web-sovellukselle. Sovelluksia vertailevat testit toteutetaan alkuperäiselle web-sovellukselle sekä tästä luodulle progressiiviselle web-sovellukselle.

Edellä mainittu sovellus hyödyntää ulkopuolista API (Application Programming Interface) -rajapintaa eli ohjelmointirajapintaa käyttäjään liittyvän datan hakemiseen, analysoi dataa ja esittelee analysoidun datan käyttäjälle. Sovellus on niin kutsuttu SPA (Single page application), joka on toteutettu Angular-frameworkillä.

3.1 Sovelluksen toimintaperiaate

Työssä käytettävä sovellus on suunnattu käyttäjille, jotka tekevät työtään mobiililaitteilla. Tämä on otettu huomioon sovelluksen suunnittelussa ja toteutuksessa. Sovellus on suunniteltu käyttäjälähtöisesti sekä mobiilioptimoidusti, ja sovelluksen toiminta on optimoitu käyttökokemuksen mukaan.

Sovellus ei ole reaaliaikainen. Sovelluksen käyttämä data päivitetään kerran vuorokaudessa. Edellä mainituista syistä progressiivinen web-teknologia on tässä työssä käytettävälle sovellukselle varteenotettava vaihtoehto. Välimuistia voidaan hyödyntää tehokkaasti datalle, joka muuttuu vain kerran vuorokaudessa, ja mobiilia käyttökokemusta pyritään kehittämään progressiivisen web-teknologian avulla.

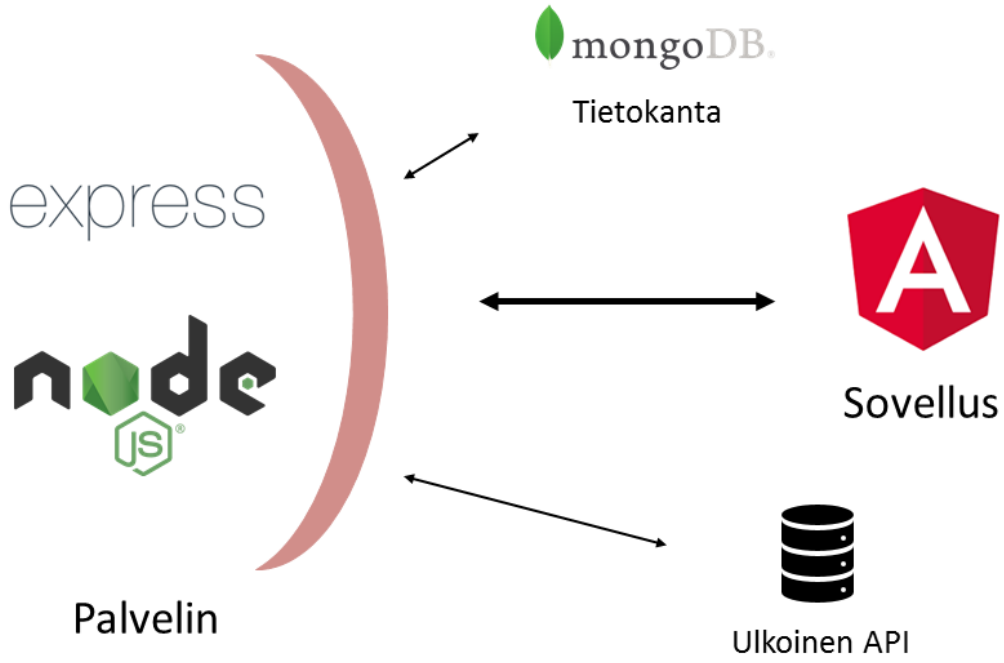
3.2 Sovelluksen rakenne

Työssä käytettävä sovellus rakentuu kahdesta eri syötteestä, joista molemmat esittelevät käyttäjälle ulkoisesta API-rajapinnasta haettua ja analysoitua dataa. Nämä syötteet ovat sovelluksen tärkeimmät osat, sillä ne sisältävät runsaasti käyttäjälle lisäarvoa antavaa dataa, jota sovellus on hakenut ja analysoinut.

Sovelluksessa on kirjautumiselle olemassa oma osionsa, joka sisältää myös kirjautumisen jälkeisen käyttäjän tervehtimisen. Lisäksi tähän osioon on toteutettu ensimmäisen käyttökerran aikana käyttäjältä vaadittavat toimet ja lisätiedot. Edellä mainittujen osien lisäksi sovelluksessa on omat sivunsa käyttäjän asetuksille sekä käyttäjän profiilille.

Työssä käytettävä sovellus on toteutettu MEAN-teknologiapinolla [11, 12] käyttäen AngularJS:n [13] tilalla Angularia. MEAN-nimi tulee sanoista Mongo DB [14], Express.js [15], Node.js [16] ja Angular/AngularJS.

Sovelluksen palvelimena toimii Node.js-ympäristö Express.js-kirjastolla. Sovelluksen tietokanta on Mongo DB -tietokanta. Sovelluksen frontend-teknologiana on Angular 6. Sovelluksen rakenne ja toiminta on havainnollistettu kuvassa 2.



Kuva 2: Sovelluksen rakenne ja toiminta

Sovelluksen backend toimii Angular-frontendille API-rajapintana, josta Angular-sovellus pyytää resursseja. Sovelluksen backend kommunikoi myös tietokannan sekä ulkoisen API-rajapinnan kanssa.

3.3 Angular-web-sovellus

3.3.1 Angular-framework lyhyesti

Angular-framework on JavaScript [17] -framework, joka mahdollistaa nopean, tehokkaan ja modulaarisen web-sovelluskehityksen. Angular-framework julkaistiin Googlen toimesta vuonna 2010 alun perin nimellä AngularJS. AngularJS oli täysin JavaScriptillä kirjoitettu. AngularJS:ssä oli puutteita, josta syystä frameworkille tehtiin täydellinen uudelleenkirjoitus.

Frameworkin uusi versio oli täysin uudelleenkirjoitettu Angular, joka perustuu TypeScript-ohjelmointikieleen [18]. Angular paikkasi AngularJS:n puutteita ja on olemassaolonsa aika kehittynyt jatkuvasti ja uusia versiopäivityksiä on julkaistu tiheällä aikataululla. AngularJS:ää seurannut Angular 2 julkaistiin vuonna 2016. Tällä hetkellä uusin versio Angularista on syksyllä 2018 julkaistu Angular 7.

3.3.2 Angularin peruskomponentit ja toimintaperiaate

Angular-sovellus on SPA (Single Page Application) -sovellus, joka on ajossa selaimessa ja jonka perustana toimii Javascript-ohjelmakoodi. Angular-sovellus koostuu lukuisista pienemmistä osista, jotka kommunikoivat toistensa kanssa ja näin yhdessä muodostavat Angular-sovelluksen.

Angular-sovelluksen perustana toimii ng-moduulit. Angular-sovellus koostuu vähintään yhdestä tai yleensä useammasta ng-moduulista, joiden tarkoituksena on paketoida tietyt toiminnallisuudet omaan moduuliinsa. Nämä moduulit yhdessä muodostavat sovelluksen. Moduulien avulla sovellukseen saadaan modulaarisuutta ja sovellus saadaan jaettua toisistaan riippumattomiin, uudelleenkäytettäviin moduuleihin.

Angularin moduulit koostuvat pienemmistä osista, kuten komponenteista ja palveluista. Komponentit ovat ohjelman osia, jotka sisältävät logiikan, näkymän sekä tyylimerkkauksen. Palvelut ovat ohjelman osia, joita sovelluksen komponentit hyödyntävät, ja joilla on usein oma vastuualueensa.

Edellä mainitut käsitteet ovat Angular-sovelluksen tärkeimmät osat. Näiden lisäksi Angular sisältää monia kirjastoja, jotka tarjoavat rajapintoja erilaisiin web-sovelluksen toimintoihin.

3.3.3 Angular CLI -konsolikäyttöliittymä

Angular-frameworkille julkaistiin vuonna 2017 Angular CLI [19] -konsolikäyttöliittymä, joka tekee Angularin käytöstä helppoa ja tehokasta. Angular CLI esimerkiksi luo valmiin aloitussovelluksen, hoitaa sovelluksen pakkaamisen ja muut nopeusoptimointiasetukset sekä tekee komponenttien luonnin helpoksi.

Tässä työssä hyödynnetään Angular CLI:tä työssä käytettävän sovelluksen kääntämiseen, pakkaamiseen, optimointiin sekä progressiivisen web-sovelluksen toteuttamiseen.

3.4 Sovelluksen toiminta

Tässä työssä käytettävä sovellus käyttää Facebookin Oauth 2.0 -kirjautumista [20] sovellukseen kirjautumiseen ja käyttäjän autentikointiin. Sovellus hakee käyttäjään liittyvää dataa ulkoisesta API-rajapinnasta automaattisesti tausta-ajona, ja tallentaa datan myöhempää analysointia ja tiedon käyttäjälle esittämistä varten. Datan hakeminen ulkoisesta API-rajapinnasta tapahtuu palvelimelta http-kutsuilla ja ulkoinen API-rajapinta palauttaa tiedon JSON-muodossa palvelimelle. Palvelin käsittelee palautetun data ja tallentaa datan tietokantaan.

Ohjelma analysoi käyttäjän dataa tausta-ajona ja tallentaa analysoidun datan tietokantaan. Ohjelma esittää analysoitua dataa käyttäjälle tämän käyttäessä ohjelmaa. Angular-sovellus pyytää dataa palvelimelta sen mukaan, kun käyttäjä navigoi sovelluksessa.

3.5 Sovelluksen ajonaikainen ympäristö

Testien aikana molemmat sovellusversiot ovat ajossa samankaltaisessa Herokun [21] palvelininstanssissa. Näin toimimalla saadaan ajonaikaisen ympäristön aiheuttamat vaikutukset mittaustuloksiin minimoitua, mutta sovelluksia pystytään silti testaamaan rinnakkain molempien sovellusten ollessa samaan aikaan ajossa.

Palvelimen infrastruktuurina toimii Amazon AWS-pilvipalvelin [22], jossa Herokun palvelininstanssit ovat ajossa. Sovelluksen käyttämä Mongo DB-tietokanta sijaitsee mLab-palvelussa [23], joka myös käyttää Amazon AWS-pilvipalvelimia.

4. SOVELLUKSEN MUUTTAMINEN

Normaalin Angular-web-sovelluksen muuttaminen progressiiviseksi Angular-web-sovellukseksi vaatii muutosten tekemistä, konfigurointia ja lisäkirjastojen asentamista sovellukseen. Tässä luvussa on käyty läpi muuttamiseen vaadittavat toimet tässä työssä käytettävän web-sovelluksen osalta.

4.1 Kirjastojen asentaminen

Angular CLI:n avulla progressiivisen web-sovelluksen mahdollistavien lisäkirjastojen asentaminen on tehty suoraviivaiseksi prosessiksi, jossa Angular CLI tekee asennukset ohjelmoijan puolesta. Lisäksi Angular CLI asentaa sovellukseen esitäytetyt konfiguraatiotiedostot oletusasetuksilla.

Angular CLI asentaa progressiivisen web-sovelluksen vaatimat kirjastot ja tiedostot seuraavalla komennolla:

ng add @angular/pwa

Mikäli komento toimii oikein eikä komennon suorituksessa tapahdu virheitä, sovellukseen on nyt asennettu kaikki tarvittava ja seuraavaksi service workers -ohjelmakoodi tulee konfiguroida toimimaan halutulla tavalla.

4.2 Konfigurointi

Service workers -ohjelmakoodin lopulliset, selaimeen ladattavaan sovelluspakettiin liitettävät, konfiguraatiotiedostot tehdään tässä työssä hyödyntäen Angular CLI:tä.

Angular CLI hyödyntää seuraavissa luvuissa esiteltyjä tiedostoja, joissa ohjelmoija määrittelee service workers -ohjelmakoodille haluamansa asetukset. Angular CLI luo näiden tiedostojen perusteella lopulliseen sovelluspakettiin erilliset tiedostot, jotka ladataan verkkoselaimeen sovellukseen liitettyinä.

4.2.1 Manifest.json-tiedosto

Manifest.json-tiedosto asennetaan Angular-sovelluksen index.html-tiedoston HEAD html-kenttään [24]. Manifest.json-tiedoston tarkoituksena on antaa selaimelle tieto siitä, että sovellusta tulee käsitellä progressiivisena web-sovelluksena. Tässä tiedostossa myös määritellään perustietoja, joiden pohjalta selaimet osaavat paketoida sovelluksen käyttöjärjestelmään asennettavaksi mobiilisovellukseksi.

Lisäksi Manifest.json-tiedosto sisältää kaiken tarvittavan tiedon ja kuvamateriaalin, jonka avulla selain muodostaa päätelaitteeseen asennettavan sovelluksen käyttöjärjestelmään sidotut elementit. Tällaisia elementtejä ovat esimerkiksi käynnistysruutu, sovellusikoni ja sovelluksen metatiedot.

4.2.2 Ngsw.json-tiedosto

Ngsw.json-konfiguraatiotiedosto sisältää käytännössä kaiken sen tiedon, jonka perusteella service workers -ohjelmakoodi osaa toimia ohjelmoijan määrittelemällä tavalla. Tässä tiedostossa määritellään mitä halutaan tallentaa välimuistiin, kuinka kauan tämä data tulee välimuistissa säilyttää, missä vaiheessa tietty data haetaan palvelimelta sekä muita vastaavia välimuistin hyödyntämiseen ja verkon käyttöön liittyviä asetuksia.

Yksi tärkeimmistä ngsw.json-tiedoston asetuksista on sen määrittäminen, miten sovellus käyttäytyy uusien palvelinkyselyiden ja välimuistin hyödyntämisen välillä. Tässä vaiheessa on tärkeää ymmärtää sovelluksen luonne, ja näin ollen tehdä ratkaisu sen välillä mikä on kyseiselle sovellukselle paras strategia.

Asetukset tallennetaan Ngsw.json-tiedostoon json-muodossa. Asetukset tallennetaan useana eri asetusryhmänä omiin objekteihinsa, jotka taas lisätään asetuksille varattuun Array-rakenteeseen. Tämä tarkoittaa sitä, että ohjelmoija voi halutessaan määrittää rajattomasti erilaisia asetuksia erilaiselle sisällölle, ja määritellä tarkkaan millaisilla asetuksilla tiettyä resurssia käsitellään. Ngsw.json-tiedoston AssetsGroups-Array sisältää kaikki staattisille resursseille määritetyt asetusryhmät ja DataGroups-Array sisältää kaikki palvelinkyselyille määritetyt asetusryhmät.

Tässä työssä käytettävän sovelluksen progressiiviselle versiolle määritetään sovelluksen kaikki assets-tiedostot hyödyntämään välimuistia ja tälle datalla määritetään olemassaoloajaksi yksi päivä. Kaikki käyttäjälle esiteltävä data asetetaan hyödyntämään välimuistia ja tähän dataan liittyvien API-kutsujen kohdalle määritetään voimassaoloajaksi esimerkiksi 60min. Edellä mainituilla asetuksilla Ngsw.json-tiedoston molemmat asetus-Arrayt sisältävät nyt yhden objektin eli asetusryhmän.

4.2.3 Sovelluksen ikonien lisääminen sovellukseen

Angular CLI lisäsi oletuksena käytettävät Angular-ikonit progressiivisen web-sovelluksen manifest.json-tiedostoon sekä tämän sovelluksen assets/icons-kansioon. Ohjelmoijan tulee vaihtaa nämä ikonit oman sovelluksen ikoneiksi sekä päivittää manifest.json-tiedosto oikeilla tiedosto-osoitteilla.

Service workers -ohjelmakoodi käyttää PNG-muotoisia ikoneita ja sovellukselle tulee määritellä ikonit useassa eri pikselikoossa. Sovelluksen vaatimat pikselikoot ovat nähtävissä Angular CLI:n luomista oletusikoneista.

4.3 Muutokset sovellukseen

Progressiivinen web-sovellus hyödyntää välimuistia sekä toimii myös offline-tilassa. Tämä aiheuttaa sovelluksen toiminnalle tiettyjä vaatimuksia. Sovelluksen vaatimien muutosten määrä vaihtelee sovelluskohtaisesti. Tässä työssä tehdään muutokset, joilla tässä työssä käytettävä sovellus toimii optimaalisesti ja halutulla tavalla myös hitaassa verkossa sekä offline-tilassa.

Progressiivisen sovelluksen tulisi toimia rajoitetusti, mutta kuitenkin moitteettomasti myös offline-tilassa. Tätä varten sovellukseen tulee tehdä sellaiset muutokset, että sovellus osaa kommunikoida käyttäjälleen, että tietyt toiminnot eivät ole käytettävissä offline-tilassa. Lisäksi sovellusta tulee muokata niin, että sovellus ei tuota virhesivua, mikäli käyttäjä yrittää tällaista vain online-tilassa käytettävissä olevaa toimintoa toteuttaa.

Tässä työssä käytettävän sovelluksen osalta tämä toteutettiin lisäämällä sovellukseen notifikaatioita sekä estoja niille toimille, joita sovellus ei voi toteuttaa offline-tilassa.

Usein sovelluksessa on tarvetta toiminnolle, jolla pakotetaan sovellus lataamaan palvelimelta tuoreet tiedot. Tämä voidaan service workers -ohjelmakoodin osalta toteuttaa service worker API-rajapinnan avulla. Tämän API-rajapinnan cache.delete-metodi tyhjentää service workers -ohjelmakoodin välimuistiin tallentaman datan, jolloin seuraavat datakyselyt ohjataan suoraan palvelimelle.

Tässä työssä käytettävän sovelluksen osalta tällaista toteutusta tarvitaan esimerkiksi silloin, kun käyttäjä muuttaa asetuksia. Näiden asetusten aiheuttamat muutokset sovelluksen toiminnassa ja sisällössä vaativat datan uudelleenhakemisen palvelimelta.

5. TESTIEN SUORITTAMINEN

Suoritettavien testien tarkoituksena on testata sovellusten tehokkuutta ja verrata tuloksia toisiinsa. Tässä luvussa esitetyt testit suoritetaan sekä normaalille että progressiiviselle sovellukselle.

Testattavat sovellukset ovat luonteeltaan erilaisia ja käyttäytyvät eri tavalla erilaisissa käyttöön liittyvissä skenaarioissa. Näin ollen sovelluksia ei voi verrata suoraan yksinkertaisilla testeillä. Tästä syystä tässä työssä toteutetaan useampia erilaisia skenaarioita ja vertaillaan sovellusten käyttäytymistä näissä skenaarioissa.

5.1 Testattavat suureet

Sovellusten käytettävyyttä mitataan kolmella eri suureella, jotka kaikki vaikuttavat sovelluksen tehokkuuteen ja sitä kautta käytettävyyteen. Nämä suureet ovat lisäksi yksiselitteisesti mitattavissa.

5.1.1 Latausaika

Latausaika tarkoittaa näiden mittausten yhteydessä aikaa, mikä sovellukselta kuluu tietyn mitattavan toimen suorittamiseen. Latausaika on luonnollisesti yksi tärkeimmistä websovelluksen tehokkuuden mittareista.

Tässä työssä käsitellään kahta eri latausaikaa. DomContentLoaded-latausaika tarkoittaa ajankohtaa, jolloin HTML-sivu on täysin ladattu ja jäsennelty selaimeen, eikä sivu enää odota sivuston näkymään liittyviä resursseja. Load-latausaika taas tarkoittaa ajankohtaa, jolloin koko HTML-sivu sekä sen hyödyntämät muut resurssit ovat ladattu ja jäsennelty selaimeen. [25]

5.1.2 Tiedonsiirron määrä

Tiedonsiirron määrä on monesta syystä hyvä optimoida. Tiedonsiirron määrä on suoraan verrannollinen sovelluksen nopeuteen ja sitä kautta käytettävyyteen. Lisäksi tiedonsiirrosta sovelluksen käyttäjälle aiheutuvien kulujen minimoiminen on käyttäjän kannalta tärkeää mobiilidatan kulutuksen kasvaessa [26] sekä mobiilidatan hinnan noustessa jatkuvasti.

Tiedonsiirron määrää mitataan biteissä. Tämän työn mittauksissa tiedonsiirron määrällä tarkoitetaan sitä datan määrää, jonka sovellus hakee palvelimelta käyttäjän selaimeen.

5.1.3 Palvelinkyselyiden määrä

Palvelinkyselyt on syytä minimoida, sillä sovelluksen palvelimelle lähetettyjen kyselyjen määrä on suoraan verrannollinen sovelluksen nopeuteen. Tämän työn mittauksissa palvelinkyselyt ovat kyselyitä, joita sovellus tekee palvelimelle tietyn resurssin hakua tai tallentamista varten.

Progressiivinen web-sovellus tekee toimiessaan palvelinkyselyitä samalla tavalla kuin normaali web-sovellus, mutta service workers -ohjelmakoodi voi pysäyttää nämä kyselyt, ja palauttaa kysytyn resurssin sovellukselle selaimen välimuistista. Tällaista kyselyä, joka ei toteutunut verkon kautta, ei lasketa mukaan palvelinkyselyiden määrään.

5.2 Testausten toteutus

Sovellukset ovat toiminnoiltaan toistensa täydellisiä kopioita. Testit toteutetaan sovelluksiin samaan aikaan, samassa ympäristössä ja samalla verkonnopeudella. Testit toteutetaan Google Chromen [27] ja Google Chromen DevTools [28] -työkalun avulla. Chrome DevTools on Google Chrome -selaimeen sisältyvä lisäosa.

Sovelluksille toteutetaan erilaisia testejä, jotka on kuvattu alla omissa luvuissaan. Näiden testien tarkoituksena on mitata sovellusten toimintaa erilaisissa käyttöskenaarioissa.

5.2.1 Sivun latausajan testaaminen

Sivun latausaika mitataan siitä hetkestä, kun selaimen otsikkoriviltä kysytään sovellusta. Tässä mittauksessa molemmista sovelluksista mitataan toisen, laajemman datasyöte-sivun latautuminen. Progressiivinen web-sovellus on tässä mittauksessa käyttäjän selaimen välimuistissa, jolloin sovellus hyödyntää progressiivisia ominaisuuksiaan ja välimuistiin tallennettua dataa.

Tämä mittaus toistetaan kymmenen kertaa ja näiden mittausten perusteella lasketaan latausaikojen keskiarvo. Tässä mittauksessa mitataan sekä DomContentLoaded- ja Load-latausaika. Mittaus toteutetaan lisäksi kohdella eri verkon nopeudella: Nopea 3G ja hidas 3G.

5.2.2 Sovelluksen käytön simulointi

Tässä mittauksessa sovelluksilla tehdään ennalta määritellyt toimenpiteet ja simuloidaan näin sovelluksen normaalia käyttöä. Mittaus aloitetaan sovelluksen tervetuloa-ruudusta, jolloin käyttäjä on autentikoinut itsensä ja sovellus on valmiina avattavaksi.

Sovelluksella tehdään seuraavat toimet seuraavassa järjestyksessä:

1. Siirtyminen ”tervetuloa”-sivulta ensimmäiseen datasyötteeseen.
2. Siirtyminen ensimmäisestä datasyötteestä toiseen datasyötteeseen.
3. Siirtyminen toisesta datasyötteestä takaisen ensimmäiseen datasyötteeseen.
4. Siirtyminen ”asetukset”-sivulle.
5. Siirtyminen ”tili/profiili”-sivulle.
6. Siirtyminen ensimmäiseen datasyötteeseen.
7. Siirtyminen toiseen datasyötteeseen.
8. Uloskirjautuminen.

Tässä mittauksessa mittaukset toteutetaan selaimella, jonka välimuisti on tyhjennetty. Näin ollen molemmat sovellukset lataavat aluksi tarvitsemansa datan. Progressiivinen web-sovellus lataa tarvitsemansa datan vain kerran, jonka jälkeen progressiivinen web-sovellus hyödyntää välimuistiin tallennettua dataa. Näin ollen simuloidaan tilanne, jossa sovellus avataan esimerkiksi ensimmäisen kerran aamulla, jolloin myös progressiivisen web-sovelluksen välimuistiin tallentamat datakyselyt ovat vanhentuneet ja sovellus hakee ne ensin uudestaan palvelimelta.

Tässä mittauksessa mitataan sovelluksen tiedonsiirtomäärä ja kyselyiden määrä. Tämä mittaus toteutetaan kolme kertaa ja näistä tuloksista lasketaan molempien mitattavien suureiden keskiarvot.

5.2.3 Siirtymän mittaus

Progressiivisen web-sovelluksen höydyt tulevat parhaiten esiin navigoidessa sovelluksessa ja vertailemalla progressiivisen web-sovelluksen siirtymiä normaalin web-sovelluksen siirtymiin. Tässä mittauksessa mitataan yksittäinen siirtymä sovelluksen datasyötteestä toiseen datasyötteeseen.

Tässä mittauksessa mitataan siirtymän latausaika, tiedonsiirron määrä sekä palvelinkyselyiden määrä. Mittaus toistetaan kolme kertaa ja tuloksista lasketaan molempien mitattavien suureiden keskiarvot.

6. TESTAUSTULOKSET

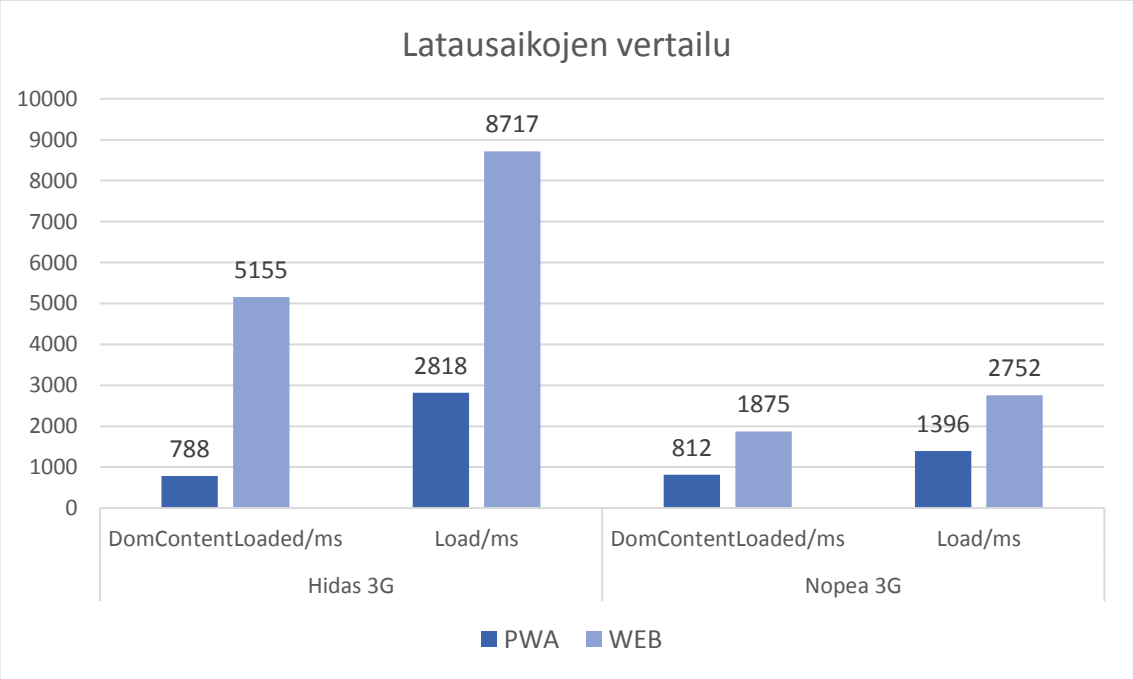
6.1 Yleinen tuntuma

Progressiivisen web-teknologian tuottaman hyödyn sovelluksen käyttökokemukselle näkee parhaiten, kun progressiivista web-sovellusta käyttää ja siirtyy sovelluksessa sivulta toiselle. Progressiivisen web-sovelluksen siirtymät ovat paljon sujuvampia ja seuraavan sivun dynaaminen data on näkyvillä lähes välittömästi.

Normaalissa web-sovelluksessa tietyn sivun dynaaminen sisältö haetaan aina palvelimelta. Tämä viive on selvästi nähtävissä sovellusta käyttäessä. Sovellusten latautuminen sovellusta avattaessa on myös huomattavasti nopeampi progressiivisella web-sovelluksella ja hyvin lähellä natiivin sovelluksen avaamista.

6.2 Sovelluksen latausaika

Mittausten tulokset ovat havainnollistettuna kuvassa 3. Vasemmalla puolella kaaviossa on sekä DomContentLoaded- että Load-latausajat verkonnopeudella Hidas 3G ja oikealla puolella on latausajat nopeudella Nopea 3G. Tämän työn kaavioissa PWA-lyhenne tarkoittaa progressiivista web-sovellusta ja WEB-lyhenne tarkoittaa normaalia web-sovellusta.



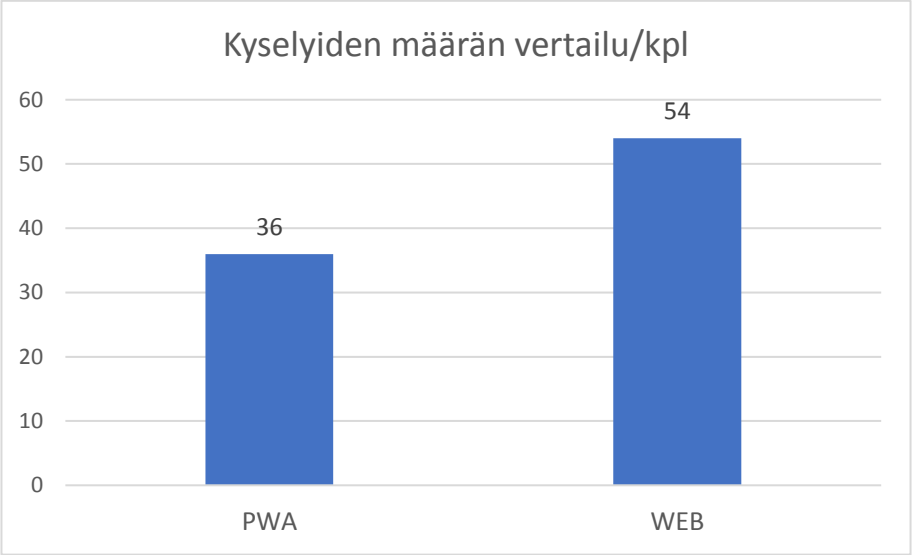
Kuva 3: Latausaikojen vertailu sivun latausajan mittauksessa

Mittausten perusteella nähdään progressiivisen web-sovelluksen latausajoissa huomattava parannus normaaliin web-sovellukseen. Verkonnopeudella nopea 3G DomContentLoaded-aika on normaalilla web-sovelluksella noin 2,3 kertainen progressiiviseen web-sovellukseen nähden. Load-aika on normaalilla web-sovelluksella noin 2 kertainen progressiiviseen web-sovellukseen nähden.

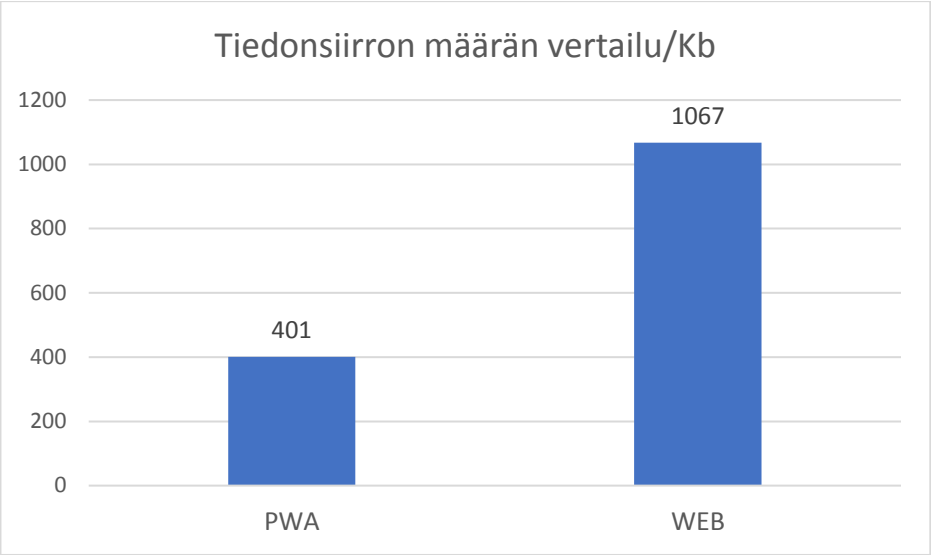
Hitaalla verkolla tehtyjen mittausten perusteella DomContentLoaded-aika on normaalilla web-sovelluksella noin 6,5 kertainen ja Load-aika noin 3,1 kertainen. Tällä verkonnopeudella sovellusten erot kasvavat paljon suuremmiksi. Tästä voidaan päätellä, että mitä hitaampi verkko on käytössä, sitä suurempi vaikutus progressiivisella web-sovelluksella on latausaikaan.

6.3 Sovelluksen käytön simulointi

Kuvassa 4 havainnollistettuna sovellusten eroavaisuus kyselyiden määrässä ja kuvassa 5 havainnollistettuna sovellusten eroavaisuus tiedonsiirron määrässä.



Kuva 4: Kyselyiden määrän vertailu käytön simuloinnin mittauksessa



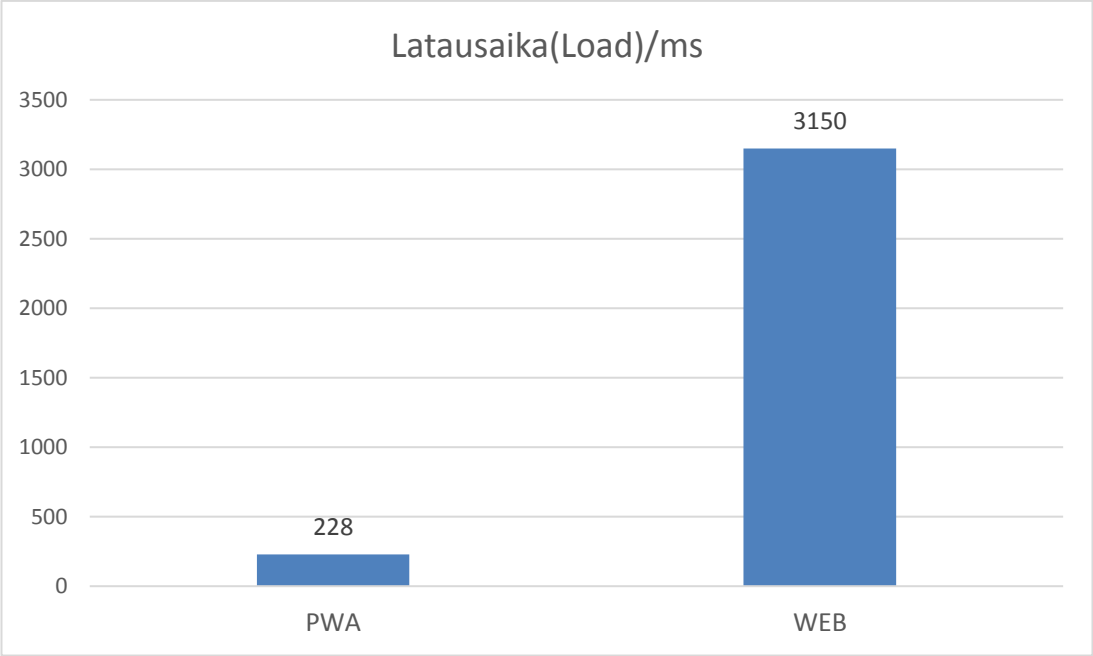
Kuva 5: Tiedonsiirron määrän vertailu käytön simuloinnin mittauksessa

Mittausten perusteella normaali web-sovellus käyttää noin 1,5 kertaisen määrän verkkokyselyitä progressiiviseen web-sovellukseen nähden. Tiedonsiirron määrä normaalilla web-sovelluksella on noin 2,7 kertainen progressiiviseen web-sovellukseen verrattuna. Tiedonsiirron määrä on tässä mittauksessa selkeästi pienempi progressiivisella web-sovelluksella.

Havainnollistetaan tiedonsiirron määrää esimerkillä. Oletetaan että käyttäjä tekee tässä testauksessa määritellyn toiminnon 10 kertaa päivässä. Tällä käyttömäärällä normaali web-sovellus käyttää kuukaudessa dataa 320100 Kb:ä. Vastaava laskelma progressiiviselle web-sovellukselle on 120300 Kb:ä. Progressiivisella web-teknologialla saavutetaan tämän yksinkertaistetun esimerkkilaskelman mukaan kuukausitasolla noin 200 000 Kb:n säästö tiedonsiirrossa. Tämä säästö on huomattava määrä dataa yhden sovelluksen kuukausittaista datan käyttömäärää ajatellessa.

6.4 Siirtymän mittaus

Siirtymän mittauksen tulokset on havainnollistettu kuvassa 6.



Kuva 6: Latausajan vertailu siirtymän mittauksessa

Tässä mittauksessa progressiivisen web-sovelluksen hyödyt ovat hyvin näkyvillä. Progressiivisen web-sovelluksen ei tarvitse hakea samaa dataa aina uudestaan. Tiedonsiirron määrää ja palvelinkyselyiden tarvetta saadaan näin karsittua.

Näiden palvelinkyselyiden jääminen pois näkyy selvästi siirtymän latausajassa. Progressiivisessa web-sovelluksessa siirtymä tapahtuu lähes välittömästi, kun taas normaalissa web-sovelluksessa sovelluksen täytyy ensin odottaa palvelinkyselyiden palauttamaa dataa. Normaalin web-sovelluksen latausaika mitatulla siirtymällä on noin 13,8 kertainen progressiiviseen web-sovellukseen nähden.

7. YHTEENVETO

7.1 Normaalin web-sovelluksen muuttaminen progressiiviseksi web-sovellukseksi

Progressiiviset web-sovellukset ovat suhteellisen uusi teknologia ja tuki selaimiin on tullut vasta viimeisten vuosien aikana. Tästä huolimatta progressiivisen web-sovelluksen luominen olemassa olevasta web-sovelluksesta on modernien web-frameworkien avulla melko suoraviivainen ja yksinkertainen prosessi.

Modernit web-frameworkit tarjoavat progressiivisen teknologian sovellukseen liittämiseen tehokkaat työkalut. Tässä työssä tarkasteltiin Angular-web-sovelluksen muuttamista progressiiviseksi web-sovellukseksi. Angular tarjoaa komentotulkkikäyttöliittymänsä Angular CLI:n avulla tähän tehokkaan ja yksinkertaisen prosessin.

Progressiivisen web-sovelluksen on määritelmänsä mukaisesti mahdollistettava sovelluksen käyttäminen myös offline-tilassa. Sovelluksen on myös osattava hyödyntää service workers -ohjelmakoodia oikealla tavalla käytettäessä sovellusta silloin, kun verkon signaali on heikko. Tämän toiminnan toteuttaminen sovelluksen käyttäjän kannalta parhaalla mahdollisella tavalla on syytä huomioida progressiiviseen web-teknologiaan siirryttäessä. Vaikka progressiivisen teknologian käyttöönotto on suhteellisen suoraviivausta ja helppoa, teknologian järkevä ja tehokas käyttö vaatii huolellista suunnittelua ja mahdollisesti sovelluksen osittaista uudelleensuunnittelua.

Tässä työssä normaalille web-sovellukselle tehtiin muutokset, joilla tässä työssä käytetty sovellus toimii oikein progressiivisena versiona. Sovelluksilla on kuitenkin aina omat erityistarpeensa, jotka tulee käydä huolella läpi muutettaessa normaali web-sovellus progressiiviseksi web-sovellukseksi.

7.2 Sovellusten käyttökokemuksen vertailu

Tärkeintä sovelluksen käyttökokemusta tarkasteltaessa on sovelluksen fyysinen, empiirinen käyttökokemus. Tässä työssä vertailtiin sovelluksia saman aikaisesti. Sovellukset olivat samaan aikaan ajossa eri palvelimilla ja sovelluksia pystyi käyttämään samaan aikaan. Näin ollen sovellusten käyttökokemusta voitiin vertailla rinnakkain.

Progressiivinen web-sovellus toimii paljon sujuvammin ja on huomattavasti lähempänä natiivin sovelluksen käyttökokemusta. Sovelluksessa navigoiminen on huomattavasti nopeampaa ja siirtymät sovelluksen eri osioista tapahtuu sujuvammin. Sujuvampi tarkoittaa

tässä yhteydessä sitä, että sovelluksen näkymän vaihtuessa uusi näkymä on valmis melkein välittömästi. Näin ollen käyttäjä ei juurikaan huomaa, että sovelluksen näkymä luodaan dynaamisesti ja näkymän sisältö haetaan käyttäjän siirtyessä uuteen näkymään.

Tässä työssä toteutetut testit tukevat käyttökokemuksen empiirisiä huomioita. Progressiivinen web-sovellus on kaikissa mittauksissa nopeampi ja vaatii vähemmän tiedonsiirtoa sekä palvelinkyselyitä. Eroavaisuudet latausajoissa sekä tiedonsiirron määrässä ovat huomattavia.

Erityisen hyvin eroavaisuus näkyi yksittäisen siirtymän mittauksessa, jossa normaalilla web-sovelluksella meni siirtymän täydelliseen latautumiseen noin 13,8 kertainen aika. Tämän eron näki hyvin myös empiirisessä kokeessa tarkasteltaessa siirtymää sovelluksen osasta toiseen.

7.3 Progressiivinen web-teknologia

Progressiivisen web-teknologian hyödyt tulevat esiin tietyn tyyppisillä sovelluksilla. Kaikille sovelluksille progressiivinen web-teknologia ei välttämättä tarjoa niin paljon hyötyjä tai lisäarvoa. Tällaisia sovelluksia ovat esimerkiksi korkeaa reaaliaikaisuutta vaativat sovellukset, jotka eivät näin ollen hyödynnä progressiivisen web-teknologian välimuistin käyttöä, muuta kuin mahdollisesti staattisille resursseille. Tällaisille sovelluksille perinteisemmät välimuistia hyödyntävät teknologiat ovat myös hyvä vaihtoehto.

Progressiivisen teknologian mahdollistama rinnakkaistaminen mobiililaitteiden natiiveihin sovelluksiin on progressiivisen teknologian mielenkiintoisimpia ominaisuuksia. Progressiivisiin web-sovelluksiin liittyy monia hyviä puolia verrattuna perinteisiin natiiveihin sovelluksiin. Käyttäjät voivat ladata sovelluksia mobiililaitteisiinsa ilman kytköstä App Storeen [29] tai Google Play -kauppaan [30]. Progressiivisen web-sovelluksen koko on usein murto-osa natiivin sovelluksen laitteistolta vaatimasta tallennustilasta. Sovelluskehittäjien näkökulmasta progressiiviset web-sovellukset tulevat mahdollisesti muuttamaan ohjelmistoalaa ja nykyistä kahden ekosysteemin (iOS ja Android) jakoa mobiilisovelluksissa ja mobiilikäyttöjärjestelmissä.

LÄHTEET

[1] Angular, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://angular.io/docs

[2] Android, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://www.android.com

[3] iOS, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://developer.apple.com/ios

[4] React, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://reactjs.org

[5] Vue.js, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://vuejs.org

[6] Npm-stat.com, verkkosivu. Viitattu: 6.12.2018. Saatavissa: https://npm-stat.com/charts.html?package=react&package=angular&package=vue&from=2013-11-23&to=2018-11-30

[7] ”Progressive web apps | web | Google Developers”, verkkosivu. Viitattu: 23.11.2018. Saatavissa: https://developers.google.com/web/progressive-web-apps

[8] Gaunt, Matt. ”Service Workers: an Introduction |Web Fundamentals | Google Developers”, verkkosivu. 21.9.2018. Viitattu: 6.12.2018. Saatavissa: https://developers.google.com/web/fundamentals/primers/service-workers

[9] ”Introduction to service worker |Web | Google Developers”, verkkosivu. 4.9.2018. Viitattu: 6.12.2018. Saatavissa: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker

[10] Caniuse.com, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://caniuse.com/#feat=serviceworkers

[11] Mean.io, verkkosivu. Viitattu: 23.11.2018. Saatavissa: http://mean.io

[12] Mean.JS, verkkosivu. Viitattu: 23.11.2018. Saatavissa: http://meanjs.org

[13] AngularJS, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://angularjs.org

[14] MongoDB, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://www.mongodb.com

[15] Express, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://expressjs.com

[16] Node.js, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://nodejs.org/en

[17] JavaScript – MDN, verkkosivu. 2018. Viitattu: 22.11.2018. Saatavissa: https://developer.mozilla.org/bm/docs/Web/JavaScript

[18] TypeScript, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://www.typescriptlang.org

[19] Angular CLI, verkkosivu. Viitattu: 23.11.2018. Saatavissa: https://cli.angular.io

[20] Facebook Login, verkkosivu. Viitattu: 22.11.2018. Saatavissa: https://developers.facebook.com/docs/facebook-login

[21] Heroku, verkkosivu: Viitattu: 24.11.2018. Saatavissa: https://heroku.com

[22] Amazon AWS, verkkosivu. Viitattu: 24.11.2018. Saatavissa: https://aws.amazon.com

[23] mLab, verkkosivu. Viitattu: 24.11.2018. Saatavissa: https://mlab.com

[24] ”<head>: The Document Metadata (Header) element – HTML: HyperText Markup Language | MDN”, verkkosivu. 19.11.2018. Viitattu: 24.11.2018. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head

[25] ”DOMContentLoaded – Event reference | MDN”, verkkosivu. 24.7.2018. Viitattu: 24.11.2018. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded

[26] ” Worldwide smartphone average monthly cellular data usage from 2016 to 2021 (in GBs per month)”, Statista.com, verkkosivu. Viitattu: 6.12.2018. Saatavissa: https://www.statista.com/statistics/752731/worldwide-average-monthly-smartphone-cellular-data-usage/

[27] Google Chrome, verkkosivu. Viitattu: 6.12.2018. Saatavissa: https://www.google.com/chrome/

[28] Google Chrome DevTools, verkkosivu. 21.9.2018. Viitattu: 24.11.2018. Saatavissa: https://developers.google.com/web/tools/chrome-devtools

[29] App Store, verkkosivu. Viitattu: 24.11.2018. Saatavissa: https://www.apple.com/lae/ios/app-store

[30] Google Play, verkkosivu. Viitattu: 24.11.2018. Saatavissa: https://play.google.com/store?hl=en