jamk.fi

Angular-sovelluksen muuttaminen progressiiviseksi verkkosovellukseksi

Eelis Piirilä

Opinnäytetyö
Toukokuu 2018
Liiketalouden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Piirilä, Eelis	Opinnäytetyö, AMK	Toukokuu 2018
	Sivumäärä 38	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Angular-sovelluksen muuttaminen progressiiviseksi verkkosovellukseksi

Tutkinto-ohjelma

Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)

Kiviaho Niko

Toimeksiantaja(t)

Growthsetters Oy

Tiivistelmä

Progressiiviset verkkosovellukset ovat viime vuosina nousseet vaihtoehdoksi perinteiselle mobiilisovelluskehitykselle. Tutkimuksen toimeksiantaja Growthsetters Oy kehittää Game of Skills -nimistä progressiivista verkkosovellusta Angular-sovelluskehyksellä. Angular sai tuen progressiivisille verkkosovelluksille vasta marraskuussa 2017, joten toimeksiantaja halusi saada lisää tietoa aiheesta.

Tutkimuksen tavoitteena oli tutkia, kuinka progressiivinen verkkosovellus kehitetään Angularilla. Tutkimuksessa suunniteltiin myös toimeksiantajan Game of Skills -sovelluksen palvelunvälittäjän konfigurointi.

Tutkimus toteutettiin kehittämistutkimuksena, koska tutkimisen lisäksi tavoitteena oli suunnitella palvelunvälittäjän konfigurointi toimeksiantajan sovellukseen. Suunnitelma sovelluksen palvelunvälittäjän konfiguroinnista tehtiin haastattelemalla toimeksiantajaa. Haastattelussa toimeksiantaja kertoi, kuinka sovelluksen tulisi toimia ilman verkkoyhteyttä ja minkälaista dataa tallennetaan välimuistiin.

Tutkimuksen tuloksena oli suunnitelma toimeksiantajan sovelluksen palvelunvälittäjän konfigurointia varten. Toimeksiantaja voi hyödyntää suunnitelmaa sovelluksessaan.

Johtopäätös on, että Angular soveltuu hyvin progressiivisen verkkosovelluksen kehittämiseen. Angular tarjoaa vain kaksi strategiaa tallentaa sisältöä välimuistiin, jotka soveltuvat hyvin yleisempiin tilanteisiin. Jos Angularin tarjoamat strategiat eivät kuitenkaan sovi sovellukseen, oman strategian lisääminen vaatii lisätyötä. Angularin tarjoaman CLI-työkalun avulla palvelunvälittäjän voi luoda automaattisesti, mikä helpottaa sovelluksen kehittämistä.

Avainsanat (asiasanat)

PWA, progressiivinen verkkosovellus, Angular, web-sovelluskehitys

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Piirilä, Eelis	Bachelor's thesis	May 2018
		Language of publication: Finnish
	Number of pages: 38	Permission for web publication: x

Title of publication

Changing Angular application into progressive web application

Degree programme

Business Information Technology

Supervisor(s)

Kiviaho Niko

Assigned by

Growthsetters Oy

Abstract

In the recent years, progressive web applications have become an alternative to the traditional mobile application development. The assignor of this research, Growthsetters Oy develops Game of Skills named progressive web application within the Angular framework. Angular got the support for progressive web applications only in March 2017; hence, the assignor wanted to gain more information on the subject.

The aim of this research was to investigate how progressive web application is developed through the use of Angular. In the research, the service worker configuration file of the assignor's Game of Skills application was designed.

This work was conducted as development research because the aim was not solely to research but also to design the service worker configuration for the assignor's application. The design of the application's service worker was discussed while interviewing the employees of the assignor company. In the interview, the interviewees revealed how the application would work without the Internet connection and what kind of data would be saved to cache.

As a result of the research, the plan for the service worker configuration of the assignor's application was created. The assignor can make good use of the plan in their application.

The conclusion is that Angular is suitable for progressive web application development. In particular, Angular offers only two strategies for saving the content to cache, which fit well to general situations. However, if the strategies offered by Angular do not match the application, the own strategy will have to be developed. With the help of CLI tool, the service worker can be made automatically, which eases application development.

Keywords/tags (subjects)

Angular, progressive web application, PWA, web development

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Käsitteet

Angular: Googlen kehittämä sovelluskehys, jolla voidaan kehittää verkkosovelluksia.

API (Application Program Interface): Ohjelmointirajapinta. Mahdollistaa sovelluksen vuorovaikutuksen ulkoisten palveluiden kanssa.

CSS (Cascade Style Sheet): Kieli, joka kuvaa verkkosivun ulkoasua.

HTML (Hypertext Markup Language): Kieli, joka kuvaa verkkosivun rakennetta.

HTTPS (Hypertext Transfer protocol secure): Protokolla, jota käytetään suojattuun tiedonsiirtoon verkossa.

Javascript: Selaimissa toimiva ohjelmointikieli, jonka avulla verkkosovelluksiin voi lisätä toimintoja.

JSON (Javascript Object Notation): Tekstimuotoinen syntaksi, joka on usein käytössä tiedon siirrossa selaimen ja palvelimen välillä.

Mobiilisovellus: Mobiililaitteen sovelluskaupasta ladattava sovellus.

Palvelin: Tässä työssä palvelimella tarkoitetaan tietokonetta, joka jakaa verkkosovellusta asiakasohjelmille.

Palvelunvälittäjä: Progressiivisen verkkosovelluksen ja verkon välillä oleva JavaScript-koodi, jonka kautta sovelluksen tekemät verkkopyynnöt kulkevat.

REST API (Representational State Transfer): Arkkitehtuuri, joka mahdollistaa tiedon lukemisen ja kirjoittamisen palvelimelta HTTP-pyynnöillä.

Verkkosovellus: Selaimessa toimiva sovellus.

Välimuisti: Tässä työssä välimuistilla tarkoitetaan selaimessa olevaa tallennuspaikkaa, jonne progressiivisen verkkosovelluksen sisällöt tallennettaan.

1 Johdanto

Sovelluskaupasta ladattavat mobiilisovellukset ovat olleet suuressa suosiossa aina ensimmäisestä iPhonesta lähtien. Progressiiviset verkkosovellukset ovat kuitenkin viime aikoina nousseet vakavasti otettavaksi vaihtoehdoksi perinteisille mobiilisovelluksille. Progressiivisia verkkosovelluksia voi mobiilisovellusten tavoin käyttää ilman verkkoyhteyttä ja ne voidaan lisätä puhelimen kotinäyttöön. Mobiilisovelluksista poiketen niitä kehitetään kuitenkin web tekniikoilla ja niiden toimintaympäristö on selain. Sama sovellus toimii siis laajasti eri laitteilla tietokoneista älypuhelimiin.

Kaikki suurimmat selaimet tukevat jo progressiivisen verkkosovelluksen vaatimia ominaisuuksia. Myös suosituimmilla sovelluskehyksillä ja ohjelmointikirjastoilla voi kehittää progressiivisia verkkosovelluksia. Googlen kehittämä Angular-sovelluskehys lisäsi tuen progressiivisille verkkosovelluksille vasta marraskuussa 2017.

Tämän opinnäytetyön toimeksiantaja Growthsetters Oy kehittää Angularilla Game of Skills -nimistä progressiivista verkkosovellusta. Tässä työssä tutkitaan progressiivisen verkkosovelluksen kehittämistä Angular-sovelluskehyksellä ja tehdään suunnitelma Game of Skills -sovelluksen offline-toimintojen toteutuksesta.

Angularin tarjoamalla CLI-työkalulla progressiivisen verkkosovelluksen tarvitseman palvelunvälittäjän saa lisättyä sovellukseen automaattisesti. Oikein konfiguroitu palvelunvälittäjä parantaa sovelluksen käyttäjäkokemusta, sillä sen avulla sovellusta pystyy käyttämään normaalisti myös ilman verkkoyhteyttä. Angular sai tuen progressiivisille verkkosovelluksille vasta vähän aikaa sitten, joten tietoa aiheesta löytyy vähän.

2 Tutkimusasetelma

2.1 Opinnäytetyön toimeksiantaja ja tavoitteet

Opinnäytetyön toimeksiantajana on ohjelmistoja, digitaalisia palveluita ja liiketoiminnan ratkaisuja kehittävä Growthsetters Oy. Toimeksiantaja kehittää Game of Skills -

nimistä sovellusta. Sovelluksen päätehtävänä on aktivoida urheilijoita tekemään urheilusuorituksia kotona varsinaisten harjoitusten ulkopuolella. Sovellus toteutetaan Amazon Web Services -ympäristössä ja sitä kehitetään Angular-sovelluskehyksellä.

Game of Skills -sovellusta kehitetään progressiiviseksi verkkosovellukseksi (engl. progressive web app). Tässä työssä progressiiviseen verkkosovellukseen viitataan lyhenteellä PWA. PWA-sovellusten suosio on noussut viime aikoina, ja niitä pidetään varteenotettavana vaihtoehtona perinteisille mobiilisovelluksille. Angular sai PWA-tuen vasta syyskuussa 2017 versiossa 5. Kyseessä on siis uusi ja jatkuvasti kehittyvä tekniikka, jota on tutkittu vähän. Angular tarjoaa erilaisia ratkaisuja sisällön tallentamiseen ja lataamiseen PWA-sovelluksessa. Tämän opinnäytetyön tavoitteena on tutkia PWA-sovelluksen kehittämistä Angular-sovelluskehyksellä. Tutkitun tiedon ja sen kautta hankitun osaamisen perusteella tavoitteena on suunnitella, mitkä ratkaisut sisällön tallentamiseen ja lataamiseen palvelevat Game of Skills -sovelluksen toimintaa parhaiten.

2.2 Tutkimusongelma- ja kysymykset

Tutkimuksen taustalla on aina tutkimusongelma, joka muutetaan tutkimuskysymyksiksi. Tutkimusongelman määrittely ja rajaaminen ovat tärkeää, sillä tutkimusongelma ohjaa koko tutkimusprosessia. Vastaus tutkimuskysymykseen ratkaisee tutkimusongelman. (Kananen 2017, 56–60.)

Tämän opinnäytetyön tutkimusongelmaksi muodostui verkkosovelluksen kehittäminen uudella tekniikalla. Tutkimusongelman perusteella tutkimuskysymyksiksi valikoituivat seuraavat kysymykset:

- Mikä on progressiivinen verkkosovellus?
- Soveltuuko Angular-sovelluskehys PWA-sovelluksen kehitykseen?
- Kuinka Game of Skills -Angular-sovellus toteutetaan progressiiviseksi verkkosovellukseksi?

2.3 Tutkimuskysymykset ja tutkimusmenetelmä

Opinnäytetyön tutkimusmenetelmä on kehittämistutkimus. Yrityksissä ja organisaatioissa kehitetään ja parannetaan tuotteita ja palveluita jatkuvasti. Kehittämistyö lähtee aina muutostarpeesta, ja sen tavoitteena on kehitettävän asian parantaminen. Kehittämistutkimuksessa muutoksen aikaansaamiseksi kehitetään esimerkiksi tuotetta tai menetelmää. Kehittämistyötä tehdään yrityksissä jatkuvasti, mutta se ei ole kehittämistutkimusta. Kehittämistyöstä tulee kehittämistutkimus, kun siihen liitetään tutkimus sekä tutkimusprosessin ja tulosten raportointi. Kehittämistutkimus on enemmän kuin perinteinen laadullinen ja määrällinen tutkimus, sillä siihen kuuluu myös ongelman poistaminen. Ongelman poistaminen ei kuulu perinteisen tutkimukseen. (Kananen 2015, 33–40.)

Tämän opinnäytetyön tavoitteena on suunnitella, kuinka toimeksiantajan Game of Skills -sovelluksen sisällöt tallennetaan ja ladataan sovellukseen. Toimeksiantajalla on muutostarve sovellukseen. Kehittämistutkimus sopii tämän opinnäytetyön tutkimusmenetelmäksi, koska työn tavoitteena on suunnitella ratkaisu, kuinka muutos olisi parasta toteuttaa.

2.4 Opinnäytetyön tietoperusta ja rakenne

Koska PWA on pitkälti Googlen kehittämä käsite, tietoperusta koostuu pitkälti Googlen kehittäjille tarkoitettuun materiaaliin. Tal Aterin Building progressive web apps -kirja toi arvokkaan lisän tämän työn tietoperustaan, koska se on yksi harvoista PWA:ta käsittelevistä kirjoista. Angularin palvelunvälittäjää käsittelevät kappaleet perustuvat Angularin tarjoamiin ohjeisiin, koska aiheen ajankohtaisuuden takia luotettavaa tietoa ei löytynyt muualta. Kirjojen ja palveluiden virallisten ohjesivujen lisäksi tietoperusta perustuu artikkeleihin ja tutkimuksiin.

Opinnäytetyön teoriaosuuden ensimmäinen luku käsittelee PWA:ta ja toinen luku käsittelee Angular-sovelluskehystä. Teoriaosuuden kolmas luku yhdistää aiempien lukujen aiheet ja käsittelee PWA-sovelluksen kehittämistä Angular-sovelluskehyksellä. Tutkimusosuudessa esitetään toimeksiantajan PWA-sovellukseen suunniteltu ratkaisu. Tutkimustulokset ja johtopäätökset -luvussa vastataan tutkimuskysymyksiin

sekä esitetään johtopäätökset saaduista tutkimustuloksista. Viimeisessä luvussa pohditaan tutkimuksen onnistumista ja luotettavuutta.

3 Progressiivinen verkkosovellus

3.1 Mikä on progressiivinen verkkosovellus?

Älypuhelimet ja mobiililaitteet ovat muuttaneet tapaamme käyttää internetiä. Miljoonat ihmiset ympäri maailmaa käyttävät internetiä ensimmäistä kertaa mobiililaitteilla. Google tukee tätä muutosta kehittämällä PWA-menettelytapaa, jonka avulla kehittäjät voivat lisätä verkkosovelluksiinsa mobiilisovellusten ominaisuuksia. (Progressive Web App Training 2017.)

Usein sovelluskehittäjät kuluttavat enemmän rahaa sovellustensa levittämiseen eri jakelualustoilla kuin mitä he lopulta ansaitsevat sovelluksillaan. Sovellusten julkaiseminen verkkosivuilla tarjoaa ratkaisun tähän ongelmaan. Verkkosovellukset tarjoavat kuitenkin mobiilisovelluksiin verrattuna heikosti ominaisuuksia. PWA:n avulla verkkosovellukset pystyvät hyödyntämään mobiilisovellusten ominaisuuksia luopumatta kuitenkaan verkon laajasta saavutettavuudesta. (Why Build Progressive Web Apps 2018.)

PWA on uudenlainen näkökulma verkkosovelluskehitykseen. Se hyödyntää saatavilla olevia työkaluja ja tekniikoita mahdollistaakseen parhaan käyttäjäkokemuksen. PWA mahdollistaa nopeat latausajat, sovelluksen käytön ilman verkkoyhteyttä sekä toimintoja sitouttaa käyttäjä sovellukseen. (Progressive Web App Training 2017.)

PWA-termin keksi alun perin Googlen kehittäjä Alex Russel blogikirjoituksessaan 2015 (Biørn-Hansen & Majchrzak & Grønli 2017). Russelin mukaan progressiiviset verkkosovelluksen ovat vain verkkosivuja, jotka ovat syöneet oikeat vitamiinit. Verkkosivujen, jotka haluavat lähettää ilmoituksia tai olla puhelimen kotinäytöllä on ansaittava oikeus näihin toimintoihin käyttäjän käyttäessä sovellusta. Niistä tulee progressiivisesti sovelluksia. (Russel 2017.)

Googlen lisäksi myös suuret yhtiöt kuten Microsoft, Apple ja Mozilla kehittävät tuotteisiinsa PWA:n vaatimia ominaisuuksia. Syksyllä 2017 Microsoft julkisti PWA-tuen

Windows 10 -käyttöjärjestelmän Edge-selaimen uudelle versiolle. Apple on myös lisäämässä tuen PWA-sovelluksiin Safari-selaimeen. (Kinsbruner 2018)

3.2 PWA:n hyötyjä ja haittoja

Laaja saatavuus: PWA-sovellus ajetaan käyttäjän selaimessa. Se toimii siis laajasti erilaisilla laitteilla, joissa on selain. (Frankston 2018, 106.)

Toimivuus verkkoyhteydestä huolimatta: PWA-sovellusta pystyy käyttämään myös ilman verkkoyhteyttä. (Edwards 2016.)

Kotinäyttöön lisääminen: PWA-sovelluksen voi lisätä mobiililaitteen kotinäyttöön. Kotinäyttöön lisääminen sitouttaa käyttäjän sovellukseen. Sovellusta ei tarvitse lisätä sovelluskauppaan, koska sen voi asentaa kotinäyttöön selaimesta. (Edwards 2016.)

Nopeat latausajat: PWA-sovellus latautuu lähes välittömästi, vaikka käyttäjällä olisi hidas verkkoyhteys tai ei verkkoyhteyttä lainkaan. PWA on usein nopeampi kuin tavallinen mobiilisovellus. (Ater 2017, The progressive web app advantage.)

Ilmoitukset: PWA-sovellus voi lähettää käyttäjälle ilmoituksia. Ilmoitukset ovat hyvä keino saada käyttäjät palaamaan sovelluksen pariin. Ilmoituksella käyttäjälle voi ilmoittaa esimerkiksi uudesta sisällöstä. (Edwards 2016.)

Ideaalinen PWA-sovellus yhdistää mobiilisovellusten ja verkkosovellusten parhaat puolet. Mobiilisovelluksen tavoin se on nopea, toimii ilman verkkoyhteyttä ja on asennettavissa laitteen kotinäyttöön josta sitä voi käyttää kokoruututilassa. Samalla siinä on myös asiat jotka tekevät verkosta mahtavan, kuten sisällön jakamisen URL-osoitteilla. Se toimii hyvin myös useilla eri alustoilla, eikä keskity pelkästään mobiililaitteisiin. (Edwards 2016.)

PWA-sovellus ei pysty vielä pääsemään käsiksi kaikkiin laitteiden ominaisuuksiin joihin mobiilisovellukset pääsevät, koska se toimii selaimessa. PWA:ssa on kuitenkin paljon potentiaalia tulla vaihtoehdoksi natiiville mobiilisovelluskehitykselle ilman tarvetta järjestelmäriippumattomille sovelluskehyksille. PWA:t näyttävät, tuntuvat ja toimivat samalla tavalla kuin mobiilisovellukset. Vaikka PWA-sovelluksissa on mobii-

lisovelluksiin verrattuna joitakin rajoituksia laitteiston ominaisuuksien käytössä, lopulta kehitystapa määräytyy sovelluksen vaatimusmäärittelyn perusteella. (Biørn-Hansen ym. 2017.)

3.3 Palvelunvälittäjä

PWA-sovelluksen tärkein ominaisuus palvelunvälittäjä (engl. service worker). Palvelunvälittäjä on ohjelmakoodi sovelluksen ja internetin välillä, joka pystyy keskeyttämään sovelluksen tekemiä verkkopyyntöjä ja vastaamaan niihin eri tavoilla. (ks. kuvio 1). Palvelunvälittäjä on Javascriptilla kirjoitettu ohjelmakoodi, joka suoritetaan erillään sovelluksesta. Se on vuorovaikutuksessa sovelluksen kanssa tapahtumapohjaisesti ja se tarjoaa jatkuvan taustaprosessoinnin. Palvelunvälittäjä lisää sovelluksen suorituskykyä ja vähentää verkkoyhteyden käyttöä. (Malavolta & Procaccianti & Noorland & Vukmirovic 2017; Gardner 2016.)

Kuvio 1. Palvelunvälittäjä

Palvelunvälittäjä hyödyntää vahvasti Javascriptin lupausta (engl. Promise) ja selaimessa olevaa nouto-ohjelmointirajapintaa (engl. Fetch-API). Javascript on yksisäikeinen, eli kun sovellus tekee API-kutsun, se siirtyy suorittamaan seuraavaa koodiriviä eikä jää odottamaan kutsun valmistumista. Sovellus tarvitsee kuitenkin keinon käsitellä API-kutsun palauttama vastaus. Lupaus ratkaisee tämän ongelman lupauksella tehdä asynkroninen kutsu. Kutsun tehtyään se palauttaa vastauksen. Vastaus voi joko olla hyväksytty tai hylätty (ks. kuvio2). (Sheppard 2017, Promises.)

```
var promise = new Promise(function(resolve, reject) {
    //Jokin asynkroninen toiminto, esim. API-kutsu
    if(true){
        resolve("Kutsu onnistui")
    }else{
        reject(Error("Kutsu epäonnistui"))
    }
})
promise.then(function(vastaus){
    console.log(vastaus) // "Kutsu onnistui"
}).catch(function(err){
    console.log(err) // "Kutsu epäonnistui"
})
```

Kuvio 2. Lupauksen luominen ja käyttäminen

Nouto on selaimessa toimiva ohjelmointirajapinta, jonka avulla voi tehdä verkkopyyntöjä. Nouto palauttaa lupauksen, joka sisältää pyyntöön vastauksen. Kaikki sovelluksen tekemät verkkopyynnöt kulkevat palvelunvälittäjän kautta. Palvelunvälittäjä voi kaapata sovelluksen tekemän verkkopyynnön ja palauttaa pyydetyt resurssit selaimen välimuistista. (Sheppard 2017, Fetch.)

Mahdollisuus kaapata sovelluksen tekemiä verkkopyyntöjä ja muokata niiden vastauksia antaa mahdollisuuden myös palvelunvälittäjän väärinkäytölle. Käyttäjien suojaamiseksi palvelunvälittäjän väärinkäytöltä sen voi rekisteröidä vain sivuille, jotka on suojattu HTTPS-yhteydellä. (Ater 2017, HTTPS and service workers.)

Palvelunvälittäjällä on elinkaari, joka on täysin erillään sovelluksesta. Ennen palvelunvälittäjän asentamaista se pitää rekisteröidä sovelluksen Javascript-koodissa. Rekisteröinnin jälkeen selain asentaa palvelunvälittäjän. Asennuksen aikana selaimen välimuistiin tallennetaan yleensä sovelluksen staattisia resursseja, kuten Javascript-tiedostot, kuvat ja tyylit. Jos tiedostojen tallentaminen onnistuu, palvelunvälittäjä on asennettu. Asennuksen jälkeen palvelunvälittäjä on aktivoitu ja se voi aloittaa sovelluksen lähettämien verkkopyyntöjen kuuntelun. (Gaunt 2018.)

3.4 PWA-sovelluksen offline-toiminnot

Aiemmin verkkosovellukset olivat täysin riippuvaisia palvelimesta. Kaikki sovelluksen tarvitsema data, sisältö, tyylit ja logiikka olivat tallennettu palvelimelle. Asiakaskoneen tehtävänä oli ainoastaan hahmontaa HTML-koodi käyttäjälle. Verkkosovellusten kehittyessä asiakaskoneen merkitys kasvoi ja se sai enemmän tehtäviä. Toisin kuin mobiilisovellukset, verkkosovellukset pysyivät täysin riippuvaisina palvelimesta. Verkkoyhteyden kadotessa sovellus lakkasi täysin toimimasta. (Ater 2017, What is offline-first?.)

PWA-sovellusten käyttämä offline ensin -strategian mukaan verkkoyhteyden menettäminen ja hitaat verkkoyhteydet ovat väistämättömiä. Nämä tilanteet ovat PWA-sovelluksessa normaaleja, ja niiden ei pitäisi estää sovelluksen käyttöä. Niihin pitää vain valmistua huolellisesti sovelluksen kehitysvaiheessa. Vaikka jotkin sovelluksen toiminnot eivät toimisikaan ilman verkkoyhteyttä, suurin osa sovelluksesta silti toimii. Offline ensin strategialla käyttäjälle annetaan paras mahdollinen kokemus, mikä on mahdollista sen hetkisellä verkkoyhteydellä. (Ater 2017, What is offline-first?.)

Palvelunvälittäjä mahdollistaa sovelluksen offline-käytön. Se käyttää selaimen Välimuisti-ohjelmointirajapintaa (engl. Cache-API) tallentaakseen sovelluksen tarvitsemia resursseja selaimen välimuistiin. Välimuistiin tallennettuja resursseja voi käyttää ilman verkkoyhteyttä. Datan tallentaminen paikallisesti välimuistiin nopeuttaa sovellusta ja voi vähentää mobiilikäyttäjien verkon käytöstä johtuvia kustannuksia. (Offline quickstart 2018.)

PWA-sovellus voi käyttää erilaisia strategioita sisällön tallentamiseen ja lataamiseen välimuistista (Offline quickstart 2018). Tässä esitellään niistä yleisimmät:

- Vain välimuisti: Sopii muuttumattomalle sisällölle. Sisällöt tallennetaan välimuistiin, kun palvelunvälittäjä asennetaan. (Archibald 2014.)
- Vain verkkoyhteys: Sopii sisällölle, jota ei voi käyttää ilman verkkoyhteyttä. Välimuistia ei käytetä ollenkaan tässä strategiassa. (Archibald 2014.)
- Välimuisti, sitten verkkoyhteys: Sopii useasti muuttuvalle sisällölle, esimerkiksi sosiaalisen median päivityksille ja artikkeleille. Sovellus tekee kaksi verkkopyyntöä, toisen verkkoon ja toisen välimuistiin. Ensin näytetään välimuistiin

tallennettu sisältö, jonka jälkeen näytetään verkosta tullut sisältö, jos se on muuttunut. (Archibald 2014.)

- Verkkoyhteys, varasuunnitelmana välimuisti: Jos verkkopyyntö epäonnistuu, palautetaan sisältö välimuistista. Uusi sisältö näytetään käyttäjille, joilla on verkkoyhteys. Ilman verkkoyhteyttä käyttäjille näytetään välimuistiin tallennettu vanha sisältö. (Archibald 2014.)
- Välimuisti, varasuunnitelmana verkkoyhteys: Jos sisältöä ei ole välimuistissa, tehdään verkkopyyntö (ks. kuvio 3). Offline ensin -strategiassa suurin osa pyynnöistä käsitellään näin. (Archibald 2014.)

```
self.addEventListener("fetch",
    function(event) {
        event.respondWith(
            caches.match(event.request)
            .then(function(response){
                return response || fetch(event.request);
                })
            );
        });
```

Kuvio 3. Välimuisti, varasuunnitelmana verkkoyhteys -strategia

Sen lisäksi, että PWA-sovellus pystyy näyttämään sisältöä ilman verkkoyhteyttä, se pystyy myös lähettämään sisältöä ilman verkkoyhteyttä. Taustasynkronoinnin (engl. background sync) avulla käyttäjän lähettämä viesti lähetetään, kun sovellus on yhteydessä verkkoon. Taustasynkronointi osaa odottaa verkkoyhteyden palautumista, vaikka sovellus olisi suljettu. Käyttäjän ei tarvitse siis itse huolehtia verkkoyhteyden tilasta, vaan se voi luottaa, että toiminto suoritetaan verkkoyhteyden tilasta huolimatta. Taustasynkronointi toimii sovelluksen taustalla palvelunvälittäjässä. (Ater 2017, Ensuring offline functionality with background sync.)

3.5 Verkkosovelluksen manifesti

Verkkosovelluksen manifestin tarkoituksena on tuoda sovelluksen kehittäjille tiettyjä muokattavia asetuksia (Biørn-Hansen ym. 2017). Verkkosovelluksen manifesti on yksinkertainen JSON-tiedosto. Sen avulla sovellusta voi käyttää kokoruututilassa itsenäisenä sovelluksena ja siinä määritetään ikoni ja muita ulkoasuun liittyviä asetuksia. Ikoni näytetään laitteen kotinäytössä, kun sovellus on asennettu laitteelle. (Farrugia 2016.)

Jos verkkosovellus on mahdollista asentaa käyttäjän kotinäytölle, selain kysyy automaattisesti käyttäjältä sen lisäämisestä. Sovelluksen tulee täyttää seuraavat kriteerit, että se voidaan lisätä kotinäytölle:

1. Sovellus käyttää salattua HTTPS-yhteyttä
2. Palvelunvälittäjä on rekisteröity sovellukseen
3. Sovelluksessa on verkkosovelluksen manifesti, jossa on vähintään 4 pakollista kohtaa

Lisäksi selain näyttää asennusilmoituksen vain silloin, kun se tunnistaa käyttäjän pitävän sovelluksesta niin paljon, että hän haluaa sovelluksen pikakuvakkeen kotinäyttöönsä. (Ater 2017, How browsers decide when to show an app install banner.)

Kun käyttäjä avaa sovelluksen kotinäytöltä, taustalla tapahtuu useita eri asioita: Selain avautuu ja sovellus ladataan verkosta tai välimuistista. Näiden toimintojen aikana näyttö muuttuu valkoiseksi ja vaikuttaa pysähtyneeltä. Varsinkin jos sovellus ladataan verkosta, saattaa valkoinen näyttö olla esillä useita sekunteja ennen kuin sovelluksen aloitussivu tulee näkyville. Paremman käyttäjäkokemuksen luomiseksi valkoisen näytön voi korvata sovelluksen nimellä, ikonilla, värillä ja kuvilla. Nämä tiedot määritetään verkkosovelluksen manifestissa. (Gaunt & Kinlan 2018.)

3.6 Ilmoitukset

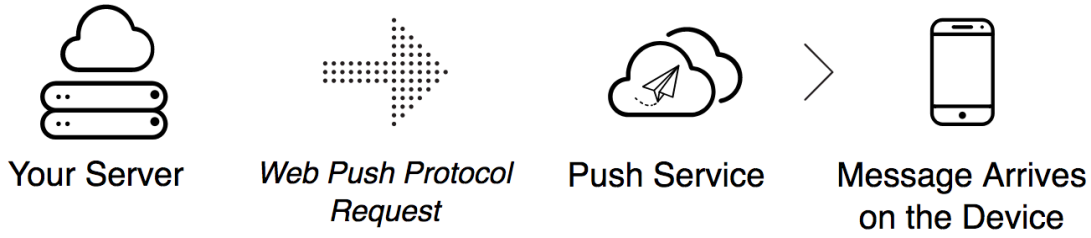
Ilmoituksia (engl. push notifications) pidettiin aiemmin tärkeimpinä mobiilisovellusten ominaisuuksina, jotka puuttuivat verkkosovelluksista. Voidaan sanoa, että ilmoi-

tukset ovat olleet yksi suurimmista tekijöistä mobiilisovellusten menestykseen. Nykyään myös PWA-sovellus voi lähettää käyttäjälle ilmoituksia, vaikka sovellus olisi suljettu. Ilmoitusten avulla käyttäjä saadaan palaamaan helposti sovelluksen pariin. Liiketoiminnan kannalta käyttäjien palaaminen sovelluksen pariin aina uudestaan ja uudestaan lisää jokaisen asennuksen arvoa. Ilmoitukset tarvitsevat toimiakseen kahta eri tekniikkaa. Viesti lähetetään käyttämällä työntö-ohjelmointirajapintaa (engl. Push-API) ja ilmoitus näytetään käyttäjälle käyttämällä ilmoitu -ohjelmointirajapintaa (Notification-API). (Ater 2017, Reach out with push notifications.)

Ilmoitus-ohjelmointirajapinta antaa palvelunvälittäjän tehdä ja hallita ilmoitusten näyttämistä. Ilmoitukset näytetään sovelluksen ulkopuolella. Ne eivät vaadi toimiakseen avointa sovellusta, joten ilmoituksia voi lähettää myös sovelluksen ollessa suljettuna. Sovelluksen pitää kysyä lupa, ennen kuin käyttäjälle voidaan näyttää ilmoituksia. (Ater 2017, The notification API.)

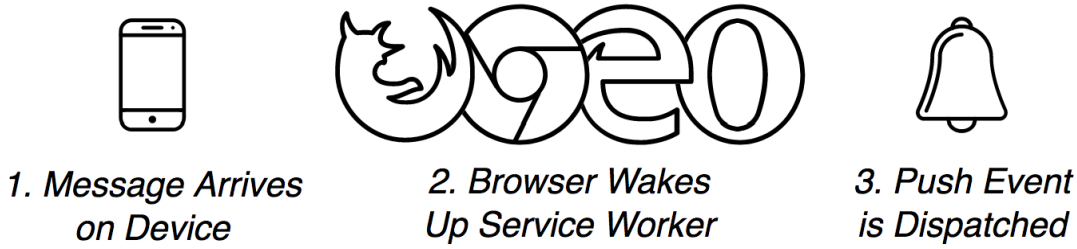
Työntö-ohjelmointirajapinta mahdollistaa viestien tilaamisen sovellukseen palvelimelta, ja antaa palvelimen lähettää viestejä selaimeen koska tahansa. Palvelunvälittäjä kuuntelee näitä viestejä, kun käyttäjä on sulkenut sovelluksen. Nämä viestit näytetään käyttäjälle ilmoituksina. Koska työntö-ohjelmointirajapinnan avulla käyttäjälle voidaan lähettää viestejä koska tahansa, voidaan sitä väärinkäyttää esimerkiksi lähettämällä käyttäjälle loputtomasti viestejä. Väärinkäytön estämiseksi kaikkien viestien pitää kulkea selaimen tarjoaman viestinvälityspalvelimen kautta. Tämä palvelin pitää kirjaa kaikista käyttäjistä jotka ovat tilanneet viestit. (Ater 2017, The push API.)

Kun käyttäjille halutaan lähettää ilmoituksia, täytyy tehdä API-kutsu viestinvälityspalvelimeen. Tämä kutsu sisältää viestin lisäksi tiedon kenelle ja miten viesti lähetetään. Tämä API-kutsu tehdään esimerkiksi sovelluksen kehittäjän omalta palvelimelta. Viestinvälityspalvelin vastaanottaa pyynnön, tarkastaa sen ja lähettää sen eteenpäin halutulle selaimille (ks. kuvio 4). Jos selaimessa ei ole verkkoyhteyttä, viesti laitetaan jonoon, kunnes selaimessa on taas verkkoyhteys. Viestin sisältämät tiedot tulee olla salattuja, jotta viestinvälityspalvelin ei pysty näkemään tietoja. (Gaunt 2018.)



Kuvio 4. Viestin reitti palvelimelta puhelimeen (Gaunt 2018)

Kun selain ottaa vastaan viestinvälityspalvelimelta viestin, se purkaa salauksen ja välittää viestin palvelunvälittäjään. Palvelunvälittäjä näyttää lähetetyn viestin käyttäjälle ilmoituksena (ks. kuvio 5). (Gaunt 2018.)



Kuvio 5. Selain välittää viestin palvelunvälittäjälle (Gaunt 2018)

4 Angular

4.1 Mikä on Angular?

Angular on Googlen kehittämä avoimen lähdekoodin Javascript sovelluskehys, jolla kehitetään verkkosovelluksia. Se on täysin uudistettu suositusta edeltäjästään AngularJS-sovelluskehyksestä. Angular on suorituskyvyltään parempi kuin edeltäjänsä. Se on myös edeltäjäänsä helpompi oppia, arkkitehtuuri on yksinkertaisempi ja sen koodia on helpompi lukea ja kirjoittaa. Angular-sovellus on komponenttipohjainen ja se kirjoitetaan Typescript-ohjelmointikielellä. (Fain & Moiseev 2017, 1–8.)

Angular-sovellus koostuu moduuleista (engl. module), komponenteista (engl. component) ja palveluista (engl. service). Ne ovat tavallisia luokkia. Luokkien alussa on @-alkuinen funktio, joka ottaa vastaan metatietoa sisältävän olion. Metatieto kertoo Angularille, kuinka luokkaa tulee käyttää. (Architecture overview n.d.)

Angular-sovellus on yhden sivun sovellus (engl. single page application). Se tarkoittaa, että sovelluksessa on vain yksi HTML-sivu, jota käytetään pohjana sovelluksen kaikille sivuille. Käyttäjän toiminnot sovelluksessa toteutetaan HTML:n, CSS:n ja Javascriptin avulla. Yhden sivun sovellus nojaa vahvasti asiakaspuoleen. Sivuja vaihdetaan reitityksen avulla, joka toteutetaan asiakaspuolella Javascriptilla, eikä perinteisten verkkosovellusten tavoin sivun uudelleen latauksella palvelimelta. (Fink & Flatow 2014, Introducing Single Page Applications.)

4.2 Typescript

Typescript on Microsoftin kehittämä avoimen lähdekoodin ohjelmointikieli. Se on vahvasti tyypitetty ohjelmointikieli, joten virheiden löytäminen sovelluksen kehitysvaiheessa on helppoa. Selaimet ymmärtävät vain Javascriptia, joten Typescript pitää kääntää Javascriptiksi ennen kuin sen voi ajaa selaimessa (ks. kuvio 6). Angular ja Typescript yksinkertaistaa keskikokoisten ja suurten verkkosovellusten kirjoittamista. Hyvät työkalut ja vahva tyypitys laskee huomattavasti virheitä ja nopeuttaa sovelluksen julkaisua. Vaikka Typescript vaatii enemmän koodia, vaatii se vähemmän testaamista, koodin muokkaamista ja virheiden tarkastusta. Vahvan tyypityksen lisäksi Typescript mahdollistaa mm. luokkien ja rajapintojen käytön. (Fain & Moiseev 2017, 388–390.)

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```

```
var Greeter = /** @class */ (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
```

Kuvio 6. Typescript-koodi käännetty Javascriptiksi

4.3 Moduulit

Angular-moduuli sisältää joukon toisiinsa liittyviä komponentteja ja palveluita. Moduulia voidaan ajatella komponenteista ja palveluista koostuvana kirjastona, joka toteuttaa jotain tiettyä toiminnallisuutta. Angular-sovelluksessa voi olla esimerkiksi oma moduuli laskutukselle ja lähetyksille. Pienen sovelluksen kaikki osat voivat sijoittua yhteen moduuliin. Sovelluksessa on pakko olla vähintään yksi moduuli, joka tarjoaa sovelluksen käynnistämiseen tarvittavan mekanismin. (Fain & Moiseev 2017, 32.)

Koodin jakaminen erillisiin moduuleihin helpottaa monimutkaisten sovellusten kehitystä ja lisää koodin uudelleen käytettävyyttä. Moduulien käyttäminen mahdollistaa myös laiskan lataamisen. Laiska lataaminen tarkoittaa, että moduuleja ladataan vain tarvittaessa. Tämä tekniikka pienentää sovelluksen käynnistämisessä ladattavan koodin määrää. (Architecture overview n.d.)

4.4 Komponentit

Angular-sovellus rakentuu komponenteista. Jokainen komponentti koostuu kahdesta osasta. Näkymä määrittää käyttöliittymän ja luokka sisältää näkymän takana olevan toimintalogiikan. Angular-sovellus esittää moduuleihin pakattuja hierarkkisia komponentteja. Sovelluksessa pitää olla moduulin lisäksi vähintään yksi komponentti. (Fain & Moiseev 2017, 33.)

Komponentti on luokka, jonka metatiedoissa määritellään valitsin (engl. selector) ja pohja (engl. template) (ks. kuvio 7). Jokainen HTML-elementti, joka vastaa komponentin metatiedoissa määritettyä valitsinta hahmonnetaan Angular-komponenttina. Pohja sisältää komponentin näkymän HTML-koodin. Jos HTML-koodi on pitkä, se voidaan sijoittaa omaan tiedostoon. (Fain & Moiseev 2017, 33–34.)

```
@Component({
  selector:  'app-component',
  template: '<h1>Hello world</h1>'
})
class HelloComponent {
/* . . . */
}
```

Kuvio 7. Komponentin metatiedot

Angular käyttää tietokytkentää (engl. data binding) käyttöliittymän ja komponentin väliseen tiedon siirtoon. Tietokytkentä on Angularissa oletuksena yksisuuntaista. Yksisuuntainen tarkoittaa, että käyttöliittymässä muuttunut tieto päivittyy komponenttiin, tai komponentista käyttöliittymään. Tietokytkentä mahdollistaa myös äiti- ja lapsikomponenttien välisen kommunikoinnin. (Intorduction to components n.d.)

4.5 Palvelut ja riippuvuusinjektio

Palvelu on sovelluksen tarvitsema arvo, funktio tai ominaisuus, joka on erillään komponentista. Palvelu on luokka, jolla on jokin tarkasti määritetty tarkoitus. Palveluiden käyttö lisää koodin uudelleenkäytettävyyttä ja modulaarisuutta. (Introduction to services and dependency injection n.d.)

Komponentin tulisi hoitaa vain näkymään liittyviä asioita, kuten tiedon esittämistä käyttäjälle. Komponentin ei ole tarkoitus hakea dataa palvelimelta tai validoida käyttäjän tietoja. Kun komponentille kuulumattoman tehtävän siirtää palveluun, voi sitä käyttää mikä tahansa komponentti. (Introduction to services and dependency injection n.d.)

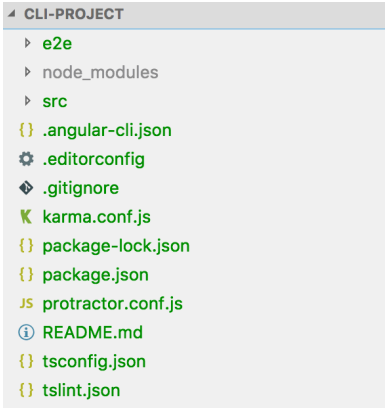
Palvelu pitää injektoida komponenttiin, jotta komponentti voisi käyttää palvelua. Angular käyttää riippuvuusinjektointia (engl. dependency injection) tarjotakseen komponenteille palveluita. Kun Angular luo uuden ilmentymän komponentti-luokasta, se määrittää komponentin tarvitsemat palvelut ja muut riippuvuudet luokan konstruktorin parametrien tyyppien perusteella (ks. kuvio 8). (Introduction to services and dependency injection n.d.)

```
constructor(private service: Service) { }
```

Kuvio 8. Komponentin tarvitsemat riippuvuudet

4.6 Angular CLI

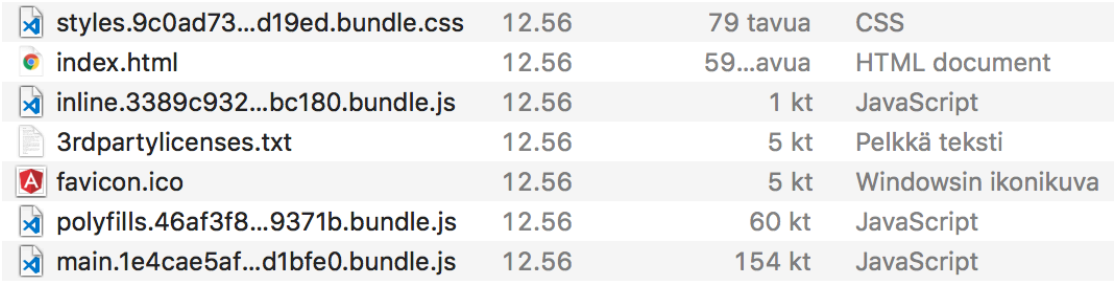
Alun perin kynnys aloittaa Angular-sovelluskehitys oli korkea, koska uusien työkalujen opettelu ja käsin konfigurointi vei aikaa. Pelkästään yksinkertaisen sovelluksen tekemiseen Angularissa tarvitaan useita eri työkaluja. Helpottaakseen sovelluksen kehittämisprosessia Angularin kehittäjät loivat työkalun nimeltä Angular CLI (command line interface). Angular CLI on komentorivissä toimiva työkalu. Yhdellä komennolla se luo uuden hakemiston, joka sisältää kaikki tarvittavat tiedostot yksinkertaiseen sovellukseen (ks. kuvio 9). Angular CLI asentaa automaattisesti kaikki vaadittavat riippuvuudet. (Fain & Moiseev 2017, 334–335.)



Kuvio 9. Angular CLI -projektin rakenne

Angular CLI -työkalulla voi ajaa sovelluksen kehitysversioita paikallisella palvelimella *ng serve* -komennolla. *ng build* -komento tuottaa Javascript -ohjelmistopaketteja,

jotka sisältävät sovelluksen koodin ja kaikki sen riippuvuudet optimoituna tuotantoon (ks. kuvio 10). Ohjelmistopaketit tallennetaan dist -kansioon. (Fain & Moiseev 2017, 335–336.)



Kuvio 10. Tuotantoversion tiedostot Dist-kansiossa

5 PWA ja Angular

5.1 PWA:n hyödyt Angular-sovelluksessa

Syyskuussa 2017 julkaistiin Angularin versio 5.0, jonka tärkein uudistus oli helpottaa palvelunvälittäjän käyttöä sovelluksissa. Aiemmin palvelunvälittäjää pystyi käyttämään Angular-sovelluksessa, mutta se vaati kehittäjältä paljon työtä. Yhden sivun sovelluksena Angular-sovellus hyötyy suuresti palvelunvälittäjästä. Palvelunvälittäjä lisää sovelluksen suorituskykyä ja luotettavuutta. Angularin palvelunvälittäjä on suunniteltu tarjoamaan paras mahdollinen käyttäjäkokemus hitaalla tai epävakaalla verkkoyhteydellä ja minimoimaan riskit vanhentuneen sisällön esittämisestä. (Krill 2017; Angular service worker introduction n.d.)

5.2 Angularin palvelunvälittäjä

Angular palvelunvälittäjän toimintaa ohjaa manifestiksi kutsuttu *ngsw.json* -tiedosto. Se sisältää tarkisteet (engl. hash) välimuistiin tallennetusta sisällöstä. Angular CLI luo manifestin *ngsw-config.json* -konfigurointitiedoston perusteella, jossa määritetään mitä tiedostoja tallennetaan välimuistiin. (Service worker configuration n.d.)

Angularin palvelunvälittäjä on suunniteltu toteuttamaan seuraavat toiminnot:

- Sovelluksen lataaminen välimuistiin on kuin mobiilisovelluksen asentaminen. Sovellus on tallennettu välimuistiin yhtenä yksikkönä, ja kaikki tiedostot päivittyvät yhdessä. (mt.)
- Päällä oleva sovellus jatkaa toimintaansa käyttäen kaikkien tiedostojen samoja versioita. Se ei yhtäkkiä ala vastaanottaa välimuistista uudempia versioita, jotka ovat todennäköisesti yhteen sopimattomia aiempien versioiden kanssa. (mt.)
- Kun käyttäjä päivittää sovelluksen, näkee hän sovelluksen uusimman välimuistista ladatun version. (mt.)
- Päivitys tapahtuu taustalla nopeasti muutosten julkaisun jälkeen. Sovelluksen aiempaa versiota käytetään, kunnes päivitys on asennettu ja valmis. (mt.)
- Palvelunvälittäjä pyrkii vähentämään verkkoyhteyden käyttöä. Resursseja ladataan vain tarvittaessa. (mt.)

Toteuttaakseen nämä toiminnot, Angularin palvelunvälittäjä lataa manifesti -tiedoston palvelimelta. Kun sovellukseen on julkaistu päivitys, manifestin sisältö muuttuu. Palvelunvälittäjä lataa ja asentaa uuden version sovelluksesta välimuistiin, kun manifestin sisältö on muuttunut. Angularin version pitää olla vähintään 5.0.0, ja Angular CLI:n version pitää olla vähintään 1.6.0, jotta sovelluksessa voidaan käyttää palvelunvälittäjää. (mt.)

Palvelunvälittäjä mahdollistaa myös ilmoitusten käytön Angular-sovelluksessa. Swpush-ohjelmointirajapinnan avulla on mahdollista tilata sovellukseen ilmoituksia ja näyttää niitä käyttäjälle. (SwPush n.d.)

Versio on joukko resursseja, jotka kuuluvat sovelluksen tietylle koontiversiolle. Kun uusi koontiversio julkaistaan, palvelunvälittäjä pitää päivitettyä koontiversiota sovelluksen uutena versiona. Versio sisältää HTML, CSS ja Javascript tiedostot. Tiedostojen kokoaminen versioihin on tärkeää sovelluksen eheyden kannalta. Tiedostot ovat usein riippuvaisia toisista tiedostoissa, joten uusien ja vanhojen tiedostojen välille syntyy helposti ristiriitoja. Tämän ongelman ratkaisemiseksi manifestin muuttuessa palvelunvälittäjä lataa kokonaan uuden koontiversion, jota se

pitää uusimpana versiona. Versioinnin avulla sovelluksessa on aina yhtenäiset tiedostot käytössä. (Service worker in production n.d.)

Aina kun käyttäjä avaa tai päivittää sovelluksen, Angularin palvelunvälittäjä tarkastaa onko manifesti muuttunut. Jos mikä tahansa välimuistiin tallennettavasta sisällöstä on muuttunut, tiedoston tarkiste muuttuu manifestissa, mikä saa Angularin palvelunvälittäjän lataamaan sovelluksen uuden version välimuistiin. Sovelluksen uusi versio näytetään käyttäjälle, kun sovellus avataan seuraavan kerran. (mt.)

Sovelluksen toiminnassa voi ilmetä ongelmia, jos sen käyttämä versio päivittyy sovelluksen ollessa päällä. Angularin palvelunvälittäjä ratkaisee ongelman jatkamalla vanhan version käyttämistä niin kauan kunnes se avataan uudestaan. Jos sama sovellus avataan selaimen toisessa välilehdessä, on silloin uudempi versio käytössä. Tämän vuoksi on mahdollista, että uudella välilehdellä voi olla eri versio sovelluksesta kuin alkuperäisellä välilehdellä. Ilman palvelunvälittäjää ei pystytä takaaman, että myöhemmin laiskasti ladattu koodi päällä olevaan sovellukseen kuuluu samaan versioon kuin sovelluksen käynnistyessä ladattu. (mt.)

5.2.1 Palvelunvälittäjän lisääminen sovellukseen

Angularin versiosta 5.0 alkaen palvelunvälittäjä-tuen on voinut lisätä helposti Angular CLI -projektiin. Palvelunvälittäjä-tuen voi lisätä uuteen projektiin lisäämällä CLI-komentoon merkinnän *--service-worker* (ks. kuvio 11). *--service-worker* -merkintä konfiguroi sovelluksen käyttämään palvelunvälittäjää lisäämällä tarvittavat tiedostot ja palvelunvälittäjä-paketin. (Service worker configuration n.d.)

```
ng new my-project --service-worker
```

Kuvio 11. Palvelunvälittäjä-tuen lisääminen uuteen projektiin (Getting started with service workers n.d.)

Palvelunvälittäjän voi lisätä myös jo olemassa olevaan sovellukseen. Silloin sovellukseen pitää lisätä manuaalisesti palvelunvälittäjä-paketti ja *ngsw-config.json* -konfigurointitiedosto. Palvelunvälittäjä pitää myös rekisteröidä manuaalisesti sovellukseen ja Angular CLI -työkaluun pitää lisätä palvelunvälittäjä-tuki. (mt.)

Palvelunvälittäjä ei toimi Angular CLI:n paikallisella palvelimella. Palvelunvälittäjän paikalliseen testaamiseen pitää asentaa erillinen http-palvelin. Kun sovelluksesta on tehty ohjelmistopaketti tuotantoon *ng build --prod* -komennolla, voidaan palvelin käynnistää dist -kansiossa. Palvelimen käynnistämisen jälkeen sovellusta voidaan testata selaimessa. (mt.)

Googlen Chrome-selaimen kehittäjän työkaluilla voi testata helposti Angularin palvelunvälittäjän toimintaa. Kehittäjän työkaluilla voi simuloida verkkoyhteyden katkeamista. Kun verkkoyhteys on poikki, Angularin palvelunvälittäjän pitäisi palauttaa tiedostot välimuistista ja sovelluksen pitäisi toimia normaalisti. (mt.)

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	(from ServiceWorker)	12 ms	
styles.9c0ad738f18adc3d19ed.bu…	200	stylesheet	(index)	(from ServiceWorker)	20 ms	
inline.c173f3aba49df529cc10.bun…	200	script	(index)	(from ServiceWorker)	15 ms	
polyfills.46af3f84a403e219371b.b…	200	script	(index)	(from ServiceWorker)	19 ms	
main.fcdd54a2a3dd8a10a50c.bun…	200	script	(index)	(from ServiceWorker)	22 ms	
data:image/svg+xml;…	200	svg+xml	main.…	(from memory cache)	0 ms	
ngsw-worker.js	200	javascript	ngsw…	(from disk cache)	2 ms	
ngsw.json?ngsw-cache-bust=…	(failed)	fetch	ngsw…	0 B	1 ms	

Kuvio 12. Tiedostot palautetaan välimuistista, kun ei ole verkkoyhteyttä

5.2.2 Palvelunvälittäjän konfigurointi

Palvelunvälittäjän konfigurointi tehdään Angular CLI:n luomaan *ngsw-config.json* -tiedostoon. JSON-muotoinen *ngsw-config.json* -konfigurointitiedosto määrittää mitä tiedostoja ja verkosta haettua dataa tallennetaan välimuistiin ja miten se päivittää välimuistissa olevia tiedostoja ja dataa. Kaikkien tiedostossa olevien polkujen on alettava / -merkillä, joka tarkoittaa dist-hakemistoa. Konfigurointitiedosto koostuu pääosin sisältöryhmistä (engl. assetsGroups) ja dataryhmistä (engl. dataGroups). (mt.)

Sisältöryhmät

Sisältöryhmät -kohta sisältää tiedostoja, jotka kuuluvat sovelluksen tiettyyn versioon ja päivittyy sovelluksen mukana. Sisältöryhmät sisältävät tiedostoja sovelluksen omalta palvelimelta. Se voi sisältää myös kolmannen osapuolen tarjoamaa sisältöä, kuten kuvia ja fontteja. Sisältöryhmään kuuluvat tiedostot saavat manifesti -tiedostossa oman tarkisteen aina version päivittyessä. Sisältöryhmät koostuvat eri ryhmistä. Jokaiselle ryhmälle määritetään välimuistiin tallennettavat tiedostot ja menettelytapa, miten tiedostoja kohdellaan välimuistissa (ks. Kuvio 13). (Service worker configuration n.d.)

```
{
 "name": "assets",
 "installMode": "lazy",
 "updateMode": "prefetch",
 "resources": {
  "files": [
   "/assets/**"
  ]
 }
}
```

Kuvio 13. Sisältöryhmä joka tallentaa kaikki assets -kansion tiedostot

Sisältöryhmä koostuu seuraavista kohdista:

name: Pakollinen kohta. Jokaiselle sisältöryhmälle annetaan nimi, joka yksilöi sisältöryhmän. (mt.)

installMode: Määrittää miten resurssit tallennetaan välimuistiin. Se voi olla jompikumpi seuraavista arvoista:

- *Prefetch* kertoo Angularin palvelunvälittäjälle, että kaikki listalla olevat tiedostot ladataan verkosta heti ja tallennetaan välimuistiin. Tämä vaatii paljon verkkoyhteydeltä, mutta takaa, että kaikki tiedostot ovat saatavilla myös ilman verkkoyhteyttä. (mt.)

- *Lazy* ei lataa mitään tiedostoa etukäteen. Angularin palvelunvälittäjä tallentaa tiedostot välimuistiin vain silloin kun se vastaanottaa pyynnön niistä. Tiedostot joita ei koskaan pyydetä ei myöskään tallenneta välimuistiin. (mt.)

updateMode: Valmiiksi välimuistiin tallennettujen tiedostojen käyttäytyminen uuden päivityksen saapuessa määritetään updateMode-kohdassa. Seuraavat arvot määrittävät ryhmän tiedostojen käyttäytymistä uuden päivityksen saapuessa:

- *Prefetch* kertoo Angularin palvelunvälittäjälle, että kaikki muuttuneet tiedostot ladataan ja tallennetaan välimuistiin välittömästi. (mt.)
- *Lazy* kieltää Angularin palvelunvälittäjää lataamasta heti välimuistiin päivittyneitä tiedostoja. Se odottaa, kunnes tiedostoja pyydetään uudestaan ennen kuin se päivittää ne välimuistiin. (mt.)

Resources: Kuvaa mitä tiedostoja tallennetaan välimuistiin. Tallennettava tiedosto voi kuulua johonkin seuraavista ryhmistä:

- *Files* listaa kaavoja, jotka vastaavat tiedostojen nimiä dist-kansiossa. Nämä voivat olla yksittäisiä tiedostoja tai kaavoja, jotka vastaavat useita tiedoston nimiä. Esimerkiksi *”/assets/**” -kaava* tarkoittaa, että kaikki tiedostot assets -kansion sisältä tallennetaan välimuistiin. (mt.)
- *Versioned files* on kuin sama kuin edellinen, mutta vain tiedostoille joiden nimessä on tarkiste. Yleensä nämä tiedostot ovat sovelluksen HTML, CSS ja Javascript -tiedostot. Angularin palvelunvälittäjä voi optimoida joitakin toimintojaan, jos se voi olettaa, että tiedostojen sisällöt eivät ole muuttuneet. (mt.)
- *Urls* sisältää URL-osoitteet, jotka tallennetaan välimuistiin. Niille ei lisätä tarkisteita, joten ladataan uudestaan aina kun sovellus päivittyy. (mt.)

Dataryhmät

Dataryhmälle määritettyjä resursseja ei versioida, eli niille ei lisätä omaa tarkistetta. Resurssit tallennetaan välimuistiin manuaalisesti määritettyjen menettelytapojen

mukaan. Dataryhmä on käytännöllinen esimerkiksi API-kutsuissa (ks. kuvio 14). (Service worker configuration n.d.)

```
"dataGroups": [{
    "name": "cached_news",
    "urls":[
      "http://apiuutiset.com/uutiset/**"
    ],
    "cacheConfig": {
      "maxSize": 10,
      "maxAge": "30m",
      "timeout": "1m",
      "strategy": "freshness"
    }}]
```

Kuvio 14. API-kutsun menettelytapojen määrittely dataryhmässä

Dataryhmä koostuu seuraavista kohdista:

Name: Jokaiselle dataryhmälle annetaan nimi joka yksilöi sen. (mt.)

Urls: Lista URL-kaavoista. URL-osoitteet jotka täsmäävät tämän listan kaavoihin tallennetaan välimuistiin tämän dataryhmän menettelytapojen perusteella. (mt.)

cacheConfig: Tässä osassa määritellään dataryhmän menettelytavat. Jos pyyntö täsmää urls-listassa olevan osoitteen kanssa, sen vastaus tallennetaan välimuistiin tässä kohdassa määritettyjen arvojen mukaan. cacheConfig-osassa määritetään alla mainitut arvot. (mt.)

- *maxSize* -kohdassa määritellään välimuistiin tallennettavien vastausten maksimimäärä. Tämä kohta on pakollinen. (mt.)
- *maxAge* -kohdassa määritetään, kuinka kauan välimuistiin tallennettu vastaus on voimassa. Vanhentunut vastaus poistetaan välimuistista. Esimerkiksi *3d12h* -merkkijono tarkoittaa, että vastausta säilytetään välimuistissa 3 päivää ja 12 tuntia. Tämä kohta on pakollinen. (mt.)
- *timeout* -kohdassa määritetään, kuinka kauan vastausta odotetaan, ennen kuin käytetään aiemmin välimuistiin tallennettua vastausta. (mt.)

- *strategy* -kohdassa määritetään, käytetäänkö *performance*- vai *freshness*-strategiaa. Performance-strategia optimoi vastaukset mahdollisimman nopeiksi. Jos resurssi on välimuistissa, sitä käytetään. Performance sopii sisällölle, joka ei vanhene nopeasti. Freshness-strategia suosii resurssien hakemista verkosta. Vain verkkopyynnön epäonnistuessa resurssit palautetaan välimuistista. Freshness sopii nopeasti muuttuville resursseille. (mt.)

6 Game of Skills -sovelluksen palvelunvälittäjän konfigurointi

6.1 Game of Skills

Game of Skills on Sanna ja Jari Mönkkösen luoma oheisharjoittelumuoto. Harjoitusmetodin tavoite on monipuolistaa urheilevien lasten harjoittelua opettamalla motorisia perustaitoja ja taidonoppimisen taitoja. Tarkoituksena on lisätä urheilijan tietoisuutta omasta kehosta ja parantaa voimatasoja sekä liikkuvuutta. Harjoitusmetodi kehittää kehonhallintaa ja ehkäisee liikuntavammoja, ja siinä korostuu kehonpainolla tehtävä lihaskuntoharjoittelu. Metodin kohderyhmänä ovat alakoululaiset urheilijat. Toisena Game of Skills haluaa tavoittaa yläkouluikäiset ja sitä vanhemmat nuoret. Ohjaajat antavat tuntien aikana yksilöllistä palautetta urheilijoille. Ohjaajan rooli on kannustaa, tarjota oivalluksia liikkumiseen, ja antaa jokaiselle mahdollisuus kehittyä omalla tasollaan. (Korpela & Palomäki & Saarinen 2017, 15.)

6.2 Game of Skills -sovellus

Game of Skills -sovellus toimii lisänä Game of Skills -harjoitusmetodiin. Sen tavoitteena on aktivoida urheilijoita tekemään urheilusuorituksia kotona varsinaisten Game of Skills -harjoitusten ulkopuolella. Sovelluksessa urheilijat suorittavat ryhmissä haasteita, jotka vaihtuvat viikoittain. Haasteet koostuvat erilaisista harjoitteista. Haaste voi olla esimerkiksi kymmenen punnerruksen tekeminen. Harjoite sisältää videon haasteen urheilusuorituksesta.

Game of Skills sovellus toteutetaan PWA:na. Käyttäjien pitää pystyä käyttämään sovellusta myös ilman verkkoyhteyttä, joten palvelunvälittäjän pitää tallentaa haasteita

ja harjoitteita välimuistiin. Seuraavissa kappaleissa suunnitellaan, miten välimuistiin tallentaminen kannattaisi toteuttaa Game of Skills -sovelluksessa.

6.3 Sovelluksen palvelinpyynnöt

Sovellus kommunikoi palvelimen kanssa REST API -rajapinnan (Representational State Transfer) kautta. REST API on arkkitehtuuri, joka mahdollistaa tiedon lukemisen ja kirjoittamisen palvelimelta HTTP-pyynnöillä (HyperText Transfer Protocol). HTTP on protokolla, jota käytetään tiedon siirtoon palvelinten ja asiakkaiden välillä. HTTP sisältää metodit, jotka tukevat kaikkia toimintoja asiakkaan ja palvelimen välillä: GET, POST, DELETE ja PUT. (Jones 2012.)

Palvelimella on kaikki sovelluksen käyttäjät, joukkueet, harjoitteet ja haasteet. Sovellus pyytää palvelimelta tarvitsemansa sisällöt (ks. taulukko 1). Sovelluksen ja palvelimen välillä on palvelunvälittäjä. Palvelunvälittäjä kaappaa sovelluksen tekemät verkkopyynnöt ja päättää päästetäänkö pyyntö palvelimelle vai palautetaanko pyydetty sisältö välimuistista.

Taulukko 1. Sovelluksen palvelinpyynnöt

Metodi	**Reitti**	**Tehtävä**
GET	/teams/123321	Palauttaa joukkoeen tiedot, jäsenet ja suoritetut viikot
GET	/teams/123321/challenges/	Palauttaa kaikki viikkojen haasteet kerralla
GET	/teams/123321/exercises/	Palauttaa kaikki harjoitteet
GET	/teams/123321/executions/	Palauttaa kaikki joukkoeen suoritetut haasteet kyseiseltä viikolta
POST	/teams/123321/executions/	Kertoo haasteen suorittamisesta

6.4 Suunnitelma palvelunvälittäjän konfiguroinnista

Angularin palvelunvälittäjä tarjoaa erilaisia strategioita sisällön tallentamiseen ja palauttamiseen välimuistista. Game of Skills -sovelluksen *ngsw-config.json* -tiedosto koostuu sisältö- ja dataryhmistä.

Sisältöryhmät

Angularin CLI -työkalu loi automaattisesti sovelluksen *ngsw-config.json* -tiedoston sisältöryhmät-osion (ks. kuvio 15). App-sisältöryhmässä välimuistiin tallennetaan sovelluksen ulkoasuun ja logiikkaan liittyvät tiedostot. Nämä tiedostot ovat CLI-työkalun luomia ohjelmistopaketteja, jotka päivittyvät vain, kun sovelluksesta julkaistaan uusi versio. Assets-sisältöryhmässä välimuistiin tallennetaan sovelluksen ikonit ja kuvat.

```
"assetGroups": [{
  "name": "app",
  "installMode": "prefetch",
  "resources": {
    "files": [
      "/favicon.ico",
      "/index.html"
    ],
    "versionedFiles": [
      "/*.bundle.css",
      "/*.bundle.js",
      "/*.chunk.js"
    ]
  }
}, {
  "name": "assets",
  "installMode": "lazy",
  "updateMode": "prefetch",
  "resources": {
    "files": [
      "/assets/**"
    ]
  }
}]
```

Kuvio 15. Game of Skills -sovelluksen sisältöryhmät Angular CLI:n luomana

Dataryhmät

Dataryhmät vaativat enemmän suunnittelua kuin automaattisesti luodut sisältöryhmät. Dataryhmiin tallennetaan palvelimelta pyydetyt muuttuvat sisällöt, kuten harjoitteet, haasteet ja joukkueet. Kävimme toimeksiantajan kanssa 26.4.2018 läpi, kuinka sovelluksen tulisi toimia ilman verkkoyhteyttä ja minkälaisissa tilanteissa verkosta yritetään hakea ensisijaisesti dataa. Suunnitelma dataryhmien toteutuksista perustuu muistiinpanoihin, jotka kirjoitettiin toimeksiantajan tapaamisen aikana.

Kaikkiin ryhmiin valittiin strategiaksi performance, joka palauttaa sisällön ensisijaisesti välimuistista. Jos sisältöä ei ole välimuistissa, haetaan se palvelimelta. Kyseessä on siis välimuisti, varasuunnitelmana verkkoyhteys -strategia. Performance ei ole kuitenkaan paras strategia, vaan sovellukseen sopii paremmin välimuisti, sitten verkkoyhteys -strategia.

Välimuisti, sitten verkkoyhteys -strategiassa käyttäjälle näytetään ensin välimuistista haettu sisältö, mutta sovellus tekee samalla pyynnön palvelimelle. Jos palvelimelta palautunut sisältö on uudempaa kuin välimuistissa oleva, päivitetään käyttäjälle uudempi sisältö. Näin käyttäjälle voidaan heti näyttää sisältöä, joka päivitetään uudempaan, jos sisältö on päivittynyt palvelimella. Riski vanhentuneen sisällön näyttämisestä vähenee, ja samalla käyttäjälle voidaan näyttää välittömästi sisältöä välimuistista.

Tässä sovelluksessa sisältö palautetaan performance-strategian mukaisesti ensisijaisesti välimuistista. Samaan aikaan lähetetään kuitenkin palvelimelle pyyntö, joka palauttaa uusimman sisällön aikaleiman kanssa. Sovelluksessa oleva ngrx-tilanhallintakirjasto vertaa aikaleimaa välimuistissa olevan sisällön ikään, ja päättää päivitetäänkö palvelimelta tullut sisältö käyttäjälle.

Palvelin palauttaa kaikki haasteet ja harjoitteet kerralla, ja niitä säilytetään välimuistissa 3 vuorokautta. Logiikka viikkohaasteiden esittämisestä käyttäjille oikeina ajankohtina on sovelluksen puolella. Joukkueiden suoritukset palautetaan vain kyseiseltä viikolta, ja niitä säilytetään välimuistissa vuorokausi (ks. kuvio 15).

```
{
    "name": "teams",
    "urls": ["/teams/"],
    "cacheConfig": {
      "strategy": "performance",
      "maxSize": 100,
      "maxAge": "1d",
      "timeout": "1m"
}
```

Kuvio 16. Joukkueiden dataryhmä

7 Tutkimustuloket ja johtopäätökset

Tämä opinnäytetyö tutki PWA-sovelluksen kehittämistä Angular-sovelluskehyksellä. Tutkitun tiedon perusteella toimeksiantajan Game of Skills-sovellukseen tehtiin suunnitelma palvelunvälittäjän toiminnasta. Tässä luvussa esitetään vastaukset tutkimuskysymyksiin sekä johtopäätökset saaduista tutkimustuloksista.

Mikä on progressiivinen verkkosovellus?

Progressiivinen verkkosovellus on yhdistelmä erilaisia tekniikoita, joiden avulla tavallinen verkkosovellus saa ominaisuuksia, jotka aiemmin olivat mahdollisia vain mobiilisovelluksille. Progressiivisen verkkosovelluksen toimintaympäristö on selain, ja sitä kehitetään tutuilla HTML, CSS ja Javascript -kielillä.

Tämän tutkimuksen perusteella progressiivinen verkkosovellus on hyvä vaihtoehto tavallisille mobiilisovelluksille, koska se toimii mobiililaitteiden lisäksi myös tietokoneilla. Offline-käyttö, kotinäyttöön asentaminen ja ilmoitukset ovat aiemmin olleet mobiilisovellusten etuoikeus, mutta nyt myös verkkosovellukset voivat käyttää niitä.

Soveltuuko Angular-sovelluskehys PWA-sovelluksen kehitykseen?

Angular-sovelluskehys on hyvä vaihtoehto PWA-sovelluksen kehittämiseen. CLI-työkalun avulla projektiin saa automaattisesti lisättyä palvelunvälittäjän, joka on PWA-sovelluksen tärkein osa. Palvelunvälittäjän toiminnan määrittäminen on tehty

hyvin yksinkertaiseksi Angular-sovelluksessa. *ngsw-config.json*-tiedostossa määritetään palvelunvälittäjän toiminta, ja CLI-työkalu luo palvelunvälittäjän automaattisesti tämän tiedoston perusteella. Kehittäjän ei tarvitse itse kirjoittaa palvelunvälittäjän asynkronisia toimintoja, mikä helpottaa sovelluksen kehittämistä ja vähentää virheitä.

Opinnäytetyön kirjoitushetkellä Angular tarjoaa vain kaksi strategiaa välimuistin käyttöön sovelluksessa: freshness ja performance. Välimuisti, sitten verkkoyhteys -strategian puuttuminen on valitettavaa. Sen avulla käyttäjälle saadaan nopeasti näytettyä dataa välimuistista, samalla kun sovellus pyytää palvelimelta tuoreempaa sisältöä. Jos palvelimelta pyydetty sisältö on tuoreempaa kuin välimuistissa oleva, näytetään se käyttäjälle.

Pienistä puutteista huolimatta Angular soveltuu hyvin PWA-sovelluksen kehitykseen. Sovelluskehyksenä komponenttipohjainen Angular on helppo oppia ja sillä voi kehittää sovelluksia, jotka toimivat mobiililaitteilla ja tietokoneilla. Palvelunvälittäjä tuo Angular-sovellukseen huomattavasti lisäarvoa mahdollistamalla offline-käytön ja ilmoitukset. Palvelunvälittäjä Angular-sovelluksessa on uusi toiminto, joka saattaa kehittyä paljon Angularin tulevien päivitysten myötä.

Kuinka Game of Skills -Angular-sovellus toteutetaan progressiiviseksi verkkosovellukseksi?

Tämän opinnäytetyön tuloksena on suunnitelma toimeksiantajan Game of Skills -sovelluksen palvelunvälittäjän konfiguroinnista. Palvelunvälittäjä on PWA-sovelluksen tärkein osa, jossa määritetään kaikki PWA:n keskeiset toiminnot. Game of Skills -sovelluksen on toimittava myös ilman verkkoyhteyttä, joten palvelunvälittäjään pitää konfiguroida, kuinka sovellus tallentaa ja palauttaa sisältöä välimuistista.

Angular CLI -työkalu loi konfigurointitiedoston automaattisesti ja lisäsi siihen sisältöryhmän, johon kuului sovelluksen tarvitsemat ohjelmistopaketit. Dataryhmien menettelytavat määritettiin toimeksiantajan toiveiden mukaisesti. REST-API:sta haetut harjoitteet, haasteet, tiimit ja suoritukset tallennetaan välimuistiin ja säilytetään siellä sisällöstä riippuen 1–2 päivää. Strategiaksi valittiin performance, eli sisältö haetaan ensisijaisesti välimuistista.

Ongelmaksi muodostui sovelluksen tarvitseman strategian puuttuminen. Performance-strategia hakee sisällön ensisijaisesti välimuistista. Jos sisältöä ei ole välimuistissa, haetaan se palvelimelta, ja säilytetään siellä 1 tai 2 päivää. Jos sisältö kuitenkin päivittyy palvelimella, haetaan se vasta kun sisältö on poistunut välimuistista. Pitkä säilytysaika palvelimella mahdollistaa sisällön käytön ilman verkkoyhteyttä, mutta nostaa kuitenkin riskiä vanhentuneen sisällön näyttämiseen.

Ongelma ratkaistiin toimeksiantajan kanssa hakemalla performance-strategian mukaan välimuistiin tallennettu sisältö, mutta samalla kuitenkin lähetetään palvelimelle pyyntö, joka palautta vastauksena uusimman sisällön. Sovellus vertaa palvelimen vastauksen aikaleimaa välimuistiin tallennetun sisällön ikään. Jos uusi sisältö on välimuistin sisältöä tuoreempaa, päivitetään se käyttäjälle. Näin käyttäjälle voidaan tarjota heti sisältöä, joka päivitetään heti uudempaan, jos se on päivittynyt palvelimella.

8 Pohdinta

8.1 Tutkimuksen onnistuminen

Tutkimuksen toteuttaminen onnistui hyvin. Tutkimuksen teoria pohjautui tutkimuksiin, kirjallisuuteen, artikkeleihin ja ohjeisiin PWA:sta ja Angularista. Vaikka PWA on melko uusi käsite, siitä löytyi hyvin tietoa. Angularista löytyi myös paljon kirjallisuutta ja ohjeita. Angularin PWA-ominaisuuksista ei löytynyt virallisen dokumentaation lisäksi muita luotettavia lähteitä, koska tuki PWA-sovelluksiin Angularissa tuli vasta marraskuussa 2017. Teoriaosuuden viimeinen kappale perustuu tästä syystä täysin Angularin viralliseen dokumentaatioon.

Tutkimuksen tuloksena on suunnitelma palvelunvälittäjän konfiguroinnista toimeksiantajan Game of Skills -sovellukseen. Suunnitelma toteutettiin toimeksiantajan toiveiden mukaisesti, ja sen avulla sovellusta voi käyttää ilman verkkoyhteyttä. Tutkimuksen aikana ilmeni, että välimuisti, sitten verkkoyhteys -strategia puuttuu tällä hetkellä Angularin palvelunvälittäjästä.

Angularin palvelunvälittäjän konfigurointi on tehty hyvin yksinkertaiseksi. Sisältö -ja dataryhmien määrittäminen tehtiin JSON-tiedostoon, joten varsinaista ohjelmointia ei tarvinnut tehdä ollenkaan. Konfiguroinnin yksinkertaisuudesta johtuen

kehittämistyö jäi melko lyhyeksi, mikä toisaalta tukee tutkimuksen johtopäätöstä Angularin soveltuvuudesta PWA-sovelluksen kehitykseen.

8.2 Tutkimuksen luotettavuus

Tutkitun aineiston ja teknisen toteutuksen perusteella tämän tutkimuksen johtopäätöksenä oli, että Angular soveltuu hyvin PWA-sovelluksen kehittämiseen. Sovelluskehitykseen valittava tekniikka on kuitenkin aina tapauskohtainen, ja siihen vaikuttaa kehitettävän sovelluksen vaatimukset. Sovelluskehittäjillä on valittavana useita eri tekniikoita PWA-sovelluksen kehittämiseen, ja Angular on niistä vain yksi hyväksi todettu vaihtoehto. Game of Skills -sovelluksen offline-toimintoihin Angular tarjosi hyvän ja helpon toteutustavan.

Tekniikat sovelluskehityksessä ovat jatkuvasti muutoksessa. Tämän opinnäytetyön kirjoittamisen aikana Angular 5 on sovelluskehyksen uusin versio, mutta kuudes versio on jo tulossa. Tässä työssä esitetyt tulokset eivät välttämättä pidä enää paikkaansa Angularin tulevissa versioissa.

8.3 Esille nousseet tutkimusaiheet

React.js ja Vue.js ovat tällä hetkellä Angularin lisäksi suosittuja komponenttipohjaisia sovelluskehyksiä tai ohjelmointikirjastoja Javascript-kehityksessä. Jokaisella näistä on hieman erilainen näkökulma sovelluskehitykseen. Tutkimuksen toteutuksen aikana esille nousi ajatus tutkimuksesta, jossa käsiteltäisiin PWA-sovelluksen kehittämisestä muilla suosituilla sovelluskehyksillä tai ohjelmointikirjastoilla. Tällä hetkellä vaikuttaa, että PWA-sovellukset tulevat yleistymään, joten tämän kaltaiselle tutkimukselle voisi olla tarvetta.

Lähteet

Angular service worker introduction. N.d. Angularin virallinen ohjesivusto. Viitattu 9.4.2018. https://angular.io/guide/service-worker-intro.

Archibald, J. 2014. The offline cookbook. Googlen kehittäjän kirjoittama ohje. Viitattu 24.4.2018. https://jakearchibald.com/2014/offline-cookbook/.

Architecture overview. N.d. Angularin virallinen ohjesivusto. Viitattu 5.4.2018. https://angular.io/guide/architecture.

Ater. T. 2017. Building progressive web apps: bringing the power of native to the browser. Kindle-lukulaitteella luettava e-kirja. Sebastopol: O'Reilly.

Biørn-Hansen A., Majchrzak T. and Grønli T. 2017. Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. Konfrenssijulkaisu. SCITEPRESS – Science and Technology Publications. Viitattu 18.4.2017. http://www.scitepress.org/Papers/2017/63537/63537.pdf

Edwards, A. 2016. The Building Blocks Of Progressive Web Apps. Artikkeli Smashing magazine -lehdessä. Viitattu 23.4.2018. https://www.smashingmagazine.com/2016/09/the-building-blocks-of-progressive-web-apps/.

Fain, Y. & Moiseev, A. 2017. Angular2 development with Typescript. New York: Manning publications.

Farrugia, K. 2016. A Beginner's Guide To Progressive Web Apps. Artikkeli InfoWorld.com sivulla. Viitattu 24.4.2017. https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/.

Fink, G & Flatow, I. 2014. Pro Single Page Application Development: Using Backbone.js and ASP.NET. Apress.

Frankston, B. 2018. Progressive web apps. IEEE, 7, 106 - 117. Viitattu 21.3.2018. https://janet.finna.fi, IEEE Xplore Digital Library.

Gardner, L. 2016. Making A Service Worker: A Case Study. Artikkeli Smashing magazine -lehdessä. Viitattu 19.4.2018. https://www.smashingmagazine.com/2016/02/making-a-service-worker/.

Gaunt, M. 2018. Service workers: an introduction. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 20.3.2018. https://developers.google.com/web/fundamentals/primers/service-workers/.

Gaunt, M. 2018. How push works. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 3.4.2018. https://developers.google.com/web/fundamentals/push-notifications/how-push-works.

Gaunt, M. & Kinlan, P. 2018. The web app manifest. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 3.4.2018. https://developers.google.com/web/fundamentals/web-app-manifest/.

Introduction to components. N.d. Angularin virallinen ohjesivusto. Viitattu 6.4.2018. https://angular.io/guide/architecture-components.

Introduction to services and dependency injection. N.d. Angularin virallinen ohjesivusto. Viitattu 6.4.2018. https://angular.io/guide/architecture-services.

Jones, K. 2012. How-to: REST Web services demystified. Artikkeli InfoWorld.com sivulla. Viitattu 27.4.2018. https://www.infoworld.com/article/2614859/application-development/how-to--rest-web-services-demystified.html.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. 2017. Laadullinen tutkimus pro graduna ja opinnäytetyönä. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kinsbruner, E. 2018. How progressive web applications (PWAs) are revolutionizing user experience. Artikkeli. Viitattu 19.4.2018. https://www.infoworld.com/article/3263655/developer/how-progressive-web-applications-pwas-are-revolutionizing-user-experience.html.

Korpela, E & Palomäki, J & Saarinen, M. 2017. Game of Skills-harjoitusmenetelmän hyötyjen arviointia 10–11 -vuotiaiden jalkapalloilijoiden iikuntataitojen kehityksessä. Viitattu 27.4.2018. http://urn.fi/URN:NBN:fi:amk-2017092815516.

Krill, P. 2017. What's new in angular 5: Easier progressive web apps. Artikkeli InfoWorld.com sivulla. Viitattu 9.4.2018. https://www.infoworld.com/article/3213244/javascript/whats-new-in-angular-5-easier-progressive-web-apps.html.

Malatova, I, Procaccianti, P, Noorland, P, Vukmirovic, P. 2017. Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps. Konfrenssijulkaisu. Viitattu 8.3.2018. https://janet.finna.fi, IEEE Xplore Digital Library.

Progressive web apps training. 2017. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 19.1.2018. https://developers.google.com/web/ilt/pwa/.

Russel, A. 2015. Progressive Web Apps: Escaping Tabs Without Losing Our Soul. Blogikirjoitus. Viitattu 19.4.2018. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

Sheppard, D. 2017. Beginning progressive web app development. iBooks-sovelluksella luettava e-kirja. Illinois: Apress.

Service worker configuration. N.d. Angularin virallinen ohjesivusto. Viitattu 11.4.2018. https://angular.io/guide/service-worker-config.

Service worker in production. N.d. Angularin virallinen ohjesivusto. Viitattu 10.4.2018. https://angular.io/guide/service-worker-devops.

SwPush. N.d. Angularin virallinen ohjesivusto. Viitattu 25.4.2018. https://angular.io/api/service-worker/SwPush.

Offline quickstart. 2018. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 27.3.2018. https://developers.google.com/web/ilt/pwa/offline-quickstart.

Why build progressive web apps. 2018. Googlen sivusto Googlen tuotteiden kehittäjille. Viitattu 20.3.2018. https://developers.google.com/web/ilt/pwa/why-build-pwa.