Reactored-ohjelmiston offline-versiointi

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikan koulutus.

HAMK-Riihimäki

syksy 2020

Juuso Helenius

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka Tiivistelmä

Riihimäki

Tekijä	Juuso Helenius	Vuosi 2020
Työn nimi	Reactored-ohjelmiston offline-versiointi	
Ohjaajat	Toni Laitinen	

TIIVISTELMÄ

Reactored on verkkopohjainen kielenoppimiseen tarkoitettu ohjelma. Tämä ohjelma tukee sekä koulujen, yksityisten kuin yritysten kielenoppimista. Reactored erottuu monista kilpailijoistaan, sillä ettei materiaali tule yritykseltä, vaan kielenoppimis materiaalin pääsee luomaan itse koulujen opettajat tai materiaalin tekijät.

Opinnäytetyön tavoitteena oli luoda jo valmiiksi JavaScriptiä käyttävälle verkkosivulle offline-versiointi, joka mahdollistaisi sivuston käyttöä ilman verkkoyhteyttä. Tämä on hyödyksi maissa, joissa verkkoyhteyttä ei ole aina saatavilla, tai verkkoyhteyden omistaminen olisi todella kallista. Sivusto käyttää JavaScript pohjaista angularJS kehikkoa, joka mahdollisti helposti yksisivuisen rakenteen sivustolle.

Valmiin tuloksen saavuttamiseksi, tarvittiin kaivaa tieto, onko käyttäjällä verkkoyhteyttä, Tämän perusteella voitiin peittää tiedot, joita ei haluta näyttää, kun verkkoyhteyttä ei ole. Tämän lisäksi tarvittiin tieto siitä, mitä haluttiin ladata selaimen välimuistiin, jotta oppitunnit voitiin näyttää. Tämä pystyttiin tekemään latauspainikkeella, joka lataa selaimen välimuistiin oppitunnin kuva- sekä äänitiedostot. Tarvittiin myös tieto siitä, jos verkkoyhteys muodostetaan, jotta oppitunnista saadut pisteet voitaisiin lähettää palvelimen tietokantaan.

Projektia varten tarvittiin kattava verkkosivun offline-versiointi, jonka saavuttamiseen vaadittiin useita komponentteja. Onnistuneeseen lopputulokseen vaadittiin toimiva verkkosivu verkkoyhteydettömässä tilassa.

Avainsanat offline, Reactored, verkkosivu, JavaScript, angularJS

Sivut 26 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communications technology ABSTRACT

Riihimäki

Author Juuso Helenius Year 2020

Subject Offline access to Reactored software

Supervisors Toni Laitinen

ABSTRACT

Reactored is a web-based language learning application. This application supports language learning for schools, private or organizations. Reactored differs from its many competitors by not creating its own material but allowing schools or material creators to create their own material.

The goal for this thesis project was to create an offline version for a website that uses JavaScript as a coding language. The offline version was to allow customers to operate the website without any internet connection. This would be useful in countries where internet connection is rarely available or using that connection would be costly. As a framework the site uses JavaScript based angularJS, that allows coders to create one page only solutions.

For a working solution, information on customers internet connections is needed. With this information we can hide information, not wanted to be shown without a connection. Information on what is needed to be saved to browser's cache is needed, so we can use HTML, CSS and JS files even without a connection. This can be achieved for example by creating a download button that saves all image and audio files to cache from every study session. Also, information on when to send data to server again is needed.

To reach our goal, a website with working offline version requiring many components is needed. For a working solution, a website that can be operated without internet connection is required.

Keywords offline, Reactored, website, JavaScript, angularJS.

Pages 26 pages

Sisällys

1 JOHDANTO

Kouluissa eripuolilla maailmaa, oppilaat opettelevat toisen kielen oman äidinkielensä lisäksi, jotkut jopa kaksi. Maailmalle suosioon on noussut kielenoppimisalustat kuten Dualingo, joiden tarkoituksena on edistää oppilaan kielenoppimista tekemällä siitä mieluisampaa pelillisyyttä hyödyntäen. Tämä konsepti sai Reactored:in luojat miettimään, miten kielenoppimista voisi edelleen parantaa kouluissa tai jopa ihan yksityiselämässä. Tarkoituksena oli lähteä tekemään konseptia, joka mullistaisi kielenoppimisen hyödyntäen vapaan materiaalin taktiikkaa. Tämä taktiikka tarkoittaa sitä, että opettajat voisivat itse luoda materiaalia, jota oppilaat lukisivat ja opettelisivat. Tämä tarkoittaa, ettei asiakkaan tarvitsisi enää lukea ulkomaanmatkaa varten sanoja koulumaailmasta, vaan hän voisi aloittaa tärkeämmistä, kuten ravintolassa asioimiseen liittyvistä sanoista. Näitä materiaaleja voisi luoda opettajat oppilailleen sekä materiaalin luojat yksityisasiakkaille.

Opinnäytetyön tavoitteena on luoda Reactored:iin verkkoyhteydetön versiointi, jolla oppilaat eripuolilla maailmaa voisivat käyttää sivustoa, myös ilman verkkoyhteyttä. Tämä parantaisi tuotteen käyttäjäläheisyyttä ja toisi täten lisää asiakkaita.

2 KEHITYKSESTÄ TAVOITTEESEEN

Tässä luvussa kerron, mistä vaiheista on päädytty siihen, että opinnäytetyön aiheena on verkkoyhteydetön versio. Aluksi kerrotaan hieman siitä, mitä kielenoppimisen ongelmat ovat ja miten niitä voidaan helpottaa. Tästä päästään verkkoyhteyden ongelmiin ja siihen, miten opinnäytetyön tavoite avittaisi asiakasta.

2.1 Kielenoppimisen ongelmat

Kielenopiskelua ajatellen törmäämme useisiin vaikeuksiin. Näitä vaikeuksia voivat olla elämäntilanteet, oppimisvaikeudet tai puuttuva halu oppia. Elämäntilanteita voivat olla

esimerkiksi vaikeudet kotona tai maailmanlaajuisesti koskettavat tilanteet. Oppimisvaikeuksia puolestaan ovat lukihäiriö tai vieraan kielen oppimisvaikeus, joka esiintyy suurella osaa suomalaisista. (Elisabet Service, 2017)

Elisabet Service (2017) muotoilee asian seuraavasti: ”Kyky ymmärtää ja tuottaa ainakin yhtä vierasta kieltä vuorovaikutuksen mahdollistavalla tasolla ei ole Suomessa ylellisyys vaan oleellinen kansalaistaito.”

Keväällä vuonna 2020, maailma koki ison pandemian COVID-19, jonka seurauksena useat koulut siirtyivät etäopetukseen. Etäopetuksen voimaanastuminen loi aivan uuden ongelman kouluille, sillä useilla kouluilla ei ollut vaadittavia resursseja etäopetuksen suorittamiseen. Resursseja olisivat esimerkiksi ryhmäkeskusteluun vaadittavat ohjelmat, joiden avulla opettajat voisivat pitää normaalia vuoropuhelua oppilaiden kanssa, ja jotka vaatisivat myös ajallaan osallistumista oppilailta. Näitä resursseja kaivattiin ja nopeasti!

Kielenoppiminen on ollut myös usealle oppilaalle vaikeuksia tuottava haaste. Näitä haasteita on Suomessa kuvattu nimellä oppimisvaikeudet, joiden takia usealla koululla on käytössä apuryhmiä, joissa oppilaat saavat pienemmissä ryhmissä yksilöllisempää tukea opiskeluun. Yhtenä oppimisvaikeutena pidettyä lukihäiriötä voidaan tukea kouluissa esimerkiksi antamalla oppilaan tehdä kokeensa suullisesti kirjoittamisen sijaan. Tämä helpottaa oppilaan etenemistä, sillä lukihäiriöisellä oppilaalla voi mennä tehtävien lukemiseen liian pitkä aika, jolloin itse vastauksille ei jää aikaa riittävästi.

Myös yhtenä ongelmana voidaan pitää oppilaan motivaation häviämistä, kun kieltä ei opi välittömästi, vaan oppilas vaatii enemmän aikaa opiskeluun. Tämä saattaa käydä pitkäveteiseksi perinteisellä kirjasta oppimisella, kun muu maailma on siirtynyt huomattavasti räväkämpään teknologiamaiseen suuntaan. Joillekin oppilaille kirja sopii paremmaksi työkaluksi opetteluun, kun taas toiset vaativat pelillisempää lähestymistä.

2.2 **Tutkimuksen tekeminen parempaan lopputulokseen**

Reactored:in tiimi on alusta asti tehnyt tutkimusta siitä, miten uusien ominaisuuksien lisääminen vaikuttaa opiskelun laatuun. Tätä edistetään pitkillä testausvaiheilla, joiden aikana Reactored:in tiimi pääsee käymään läpi ohjelmoijien uusimmat aikaansaannokset niin sanotussa esivaiheessa, joka ei vielä näy asiakkaille. Tämän tarkoituksena on pyrkiä vähentämään mahdollisten ongelmien olemassaoloa, eli toisin sanottujen ”bugien” päätymistä loppusivulle, jota asiakkaat käyttävät.

Toisena tutkimuksen aiheena Reactored:in tiimi on käynyt kehityskeskusteluja koulujen oppilaiden ja opettajien kanssa siitä, mitä uutta Reactored:iin halutaan ja miltä uudet ominaisuudet ovat vaikuttaneet. Tämä mahdollistaa sen, että tuote voisi olla mahdollisimman joustava ja asiakasläheinen. Tällä luodaan myös jälkitestausverkko, sillä uusimpien päivityksien ongelmat tulevat heti yrityksen tietoon ja ne voidaan ratkoa mahdollisimman pian.

Kolmantena tutkimuksena Reactored mahdollistaa opettajille työkaluja parantaa oman materiaalin laatua. Näitä työkaluja ovat esimerkiksi pisteytys systeemi kokeissa, joilla opettaja näkee, onko materiaali ollut puutteellista tai kokeet liian vaativia. Myös oppilaille on annettu mahdollisuus antaa korjausehdotus nappulaa painamalla, joka myös vähentää riskiä pysyviin materiaalivirheisiin.

Näillä tiedoilla tuotteesta voidaan luoda entistä monipuolisempi, asiakasystävällisempi ja kokonaisuudessaan kattavampi. Yhtenä kehitysehdotuksena onkin välimuistiin tuleva tiedon tallennus, joka mahdollistaa tuotteen käytön maissa, jossa verkkoyhteyttä ei ole aina saatavilla tai sen käyttäminen on kallista. Tähän aiheeseen pohjustuu opinnäytetyön aihe.

2.3 **Verkkoyhteyden käyttö ulkomailla**

Aiemmin todettu verkkoyhteys voi olla kehnosti saatavilla ulkomailla tai sen omistaminen voi olla todella kallista. Suomessa tilanne on erittäin hyvä, sillä lähes jokaisella on pääsy verkkoyhteyden ja tietokoneen pariin vähintään kirjastossa asioidessa. Edistyksellisissä maissa noin 81% asukkaista on internetin käyttäjiä ja globaalisti noin 48%. Tämä siis tarkoittaa, että maailman 7,8 miljardista asukkaasta noin 3,7 miljardilla on käytössä jokin

verkkoyhteyttä käyttävä laite. Tämä luku on kasvanut vuosi vuodelta yhä korkeammaksi ja onkin oletettavaa, että jonain päivänä luku lähentyisi lähes samaksi kuin maailman väkiluku. (Wikipedia, n.d)

Kuten aiemmin on todettu, joissain maissa verkkoyhteyden omistaminen voi olla kallista. Otetaan esimerkiksi Saksa, joka sijaitsee vain 1348 kilometrin päässä Suomesta. Otetaan esimerkkiin mobiiliyhteys, jota käyttää esimerkiksi puhelimet ja tabletit. Alle kootussa listassa on hintoja kahdelta eri palveluntarjoajalta, jotka tarjoavat palveluaan Lyypekin alueella. (Google, n.d)

Taulukko 1 Hintoja eri palveluntarjoajilta Saksassa (O2, n.d.; Vodafone, n.d.).

Palveluntarjoaja	Hinta halvin mahdollinen	Hinta paras liittymä
O2	34,99€/kk/20GB	44,99€/kk/rajaton
Vodafone	29,99€/kk/4GB	49,99/kk/30GB

Tätä seuraten voimme todeta, että verkkoyhteyden omistaminen ei välttämättä ole aina mahdollista, tai sitä joudutaan säästämään, koska sen määrä on rajattua. Tätä ongelmaa ajatellen päättötyön aihe, eli välimuistiin tallennus auttaa tuotetta olemaan joustavampi myös maissa, joissa tämä ongelma tulee vastaan.

2.4 **Miten välimuistiin tallennus auttaa asiakasta?**

Välimuistia on erilaista, selainperäistä tai tietokanta peräinen. Kun selaimen välimuistiin tallennetaan tietoa, voidaan verkkoyhteys ottaa pois käytöstä, mutta sivu on silti käytettävissä. Tämä tarkoittaa sitä, että HTML, JavaScript ja CSS tiedostot, kuten myös tietokannasta haetut kuvat, fontit ja käyttäjätiedot tallentuvat laitteen muistiin.

Asiakasta tämä hyödyntää sillä, että esimerkiksi oppilas voisi ladata verkkosivun tiedot välimuistiin koulun verkkoyhteyttä hyödyntäen ja kotona käyttää välimuistiin tallennettua tietoa ilman verkkoyhteyttä. Täten oppilaan ei tarvitse käyttää kallista verkkoyhteyttään tai hakeutua tilaan opiskelemaan, jossa verkkoyhteyttä on. Tämän johdosta käyttäjä voi hyödyntää tuotetta esimerkiksi katvealueilla, ilman että hänen opiskelunsa keskeytyisi. Tämä toisin sanoen antaa mahdollisuuden entistä joustavampaan tuotekäyttöön.

3 SUOMALAINEN KEHITTÄJÄTIIMI ROUHIA

Reactored niminen tuote sai korkean suosion kouluympäristöön tarkoitetulla sivullaan. Tämä sivu mahdollistaa kokeiden tekemisen, opettajan materiaalinluonnin, ryhmien muodostamisen, palautteenannon sekä useita muita kouluympäristössä tärkeänä pidettäviä työkaluja.

Reactored sai alkunsa hämeenlinnalaisen insinööriopiskelija Jaakko Vartiaisen toimesta. Jaakko opiskeli englantia verkkopohjaisilla kielenoppimis sovelluksilla, muttei löytänyt mieleistään ohjelmaa, joka tarjoaisi välitöntä palautetta. Hän päätti ruveta opettelemaan ohjelmoimaan ja rakensi oikeinkirjoitus tarkastajan, joka myöhemmin tunnetaan Reactoredina.

Saatuaan positiivista palautetta kouluilta, Jaakko ryhtyi yhteistyöhön Kari Savolaisen kanssa, jolla oli kokemusta myynnistä ja markkinoinnista. Jaakko hoiti teknisen puolen yrityksestä, kun taas Kari hoiti myyntityöt. He tekivät projektia eteenpäin muiden töidensä ohella ja viimein perustivat yrityksen vuonna 2015 nimeltä Rouhia, jonka päätuotteena toimi Reactored.

4 TYÖKALUT JA SUUNNITTELU

Tässä osassa käymme läpi projektiin liittyvät työkalut, sekä suunnittelun.

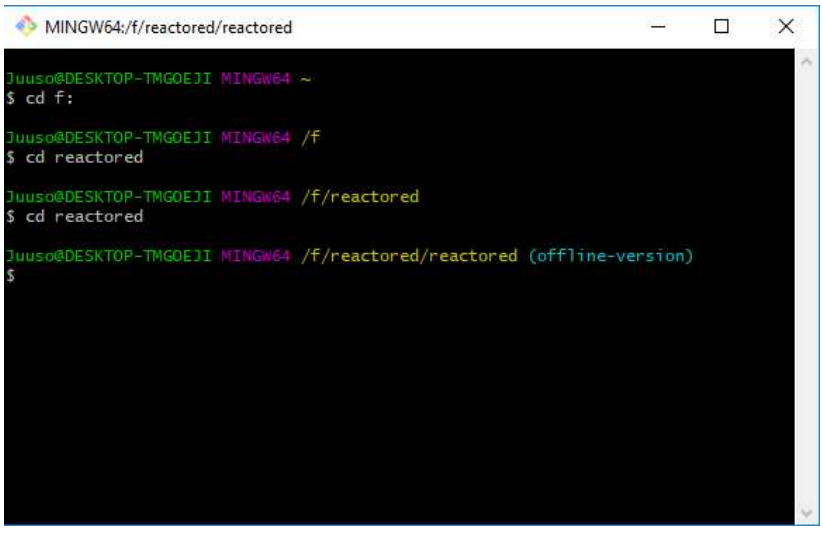
4.1 **Työkalut**

Projektin aloittaessa, on tärkeää tiedostaa, mitä työkaluja tulisi käyttää. Näitä työkaluja tässä projektissa ovat ohjelmointieditori, versiohallinta, kehitystiimin kommunikointityökalu sekä mahdolliset tiedonlähteet uuden oppimiseen.

Ohjelmointieditoreina käytämme Visual Studio Code sekä Sublime Text. Nämä ohjelmat ovat siitä hyvä valinta, että niiden toiminnallisuus, sekä ulkoasu ovat hyvin lähellä toisiaan. Näiden kahden pikanäppäimet ovat identtiset, esimerkiksi etsi- ja avaa toiminnot toimivat CTRL+F ja CTRL+P pikanäppäimillä. Syy kahteen eri ohjelmointieditorin käyttöön on kaksi eri työtietokonetta, joista toinen toimii Mac ja toinen Windows ympäristöillä, sillä työpaikka tarjoaa mahdollisuuden myös etätyöskentelyyn omalta tietokoneelta.

Versiohallintaan käytetään koulustakin opetettua GitHub-versiohallintaa, jota ohjataan Git Bash-ohjelmalla. Mac ympäristössä saman tarkoituksen ajaa Terminal. Tämä versiohallinta mahdollistaa useiden eri projektien tekemisen samanaikaisesti samassa kehitysympäristössä, ilman muiden haarojen vaikuttamista toiseen. Kun projekti on viimeistelty, voidaan se siirtää haaraan, joka siirtyy niin sanottuun testausympäristöön.

Kuva 1 Git Bash versiohallintaan



Työyhteisen yleiseen kommunikointiin on myös tärkeää pitää jonkinlaista ohjelmaa, jolla työn tilannetta voi seurata ja toivotuista muutoksista saadaan tieto reaaliaikaisesti. Tähän tarkoitukseen valittiin Slack, joka oli työpaikalla entuudestaan tuttu ja käytetty alusta. Tätä alustaa käyttäen, voidaan lähettää yksityisviestiä esimiehelle, jotta työn tulos olisi odotusten kaltainen. Tätä alustaa käyttäen, työn edetessä pidettiin kirjallisia palavereita päivittäin ja katsottiin yhdessä, missä tilanteessa työn eteneminen on jokaisen työpäivän jälkeen.

4.2 **Suunnittelu**

Työn aloittaessa, tärkein osuus on suunnittelu ja opiskelu. Tämä aihe pitää sisällään lukuisia asioita, joita tarvitsee opiskella etukäteen. Näitä asioita ovat esimerkiksi se, miten tieto tallentuu välimuistiin ja miten se käytännössä tehdään. Aika arviolta, tähän olisi hyvä käyttää ainakin 1/3 itse työn kestosta. Tämä edesauttaa sitä, että työ etenisi mahdollisimman nopeasti ja myös vähentäisi riskien määrää. Hyvä esimerkki asiasta on juurikin miten tieto välimuistiin tallentuu.

Tärkeä suunnittelun aihe verkkoyhteydettömään verkkosivuun on tutkia, miten selain tukee tätä ominaisuutta. Verkkosivu tukee tätä ominaisuutta tallentamalla paikallisesti ladatun tiedon laitteelle selaimen tiedostoihin. Esimerkiksi lataa paikallisesti HTML sivun, joka on tarkoitus ladata verkkoyhteydettömässä tilassa. Täten laitteen ei tarvitse hakea tietoa serveriltä, vaan se voi hakea jo aiemmin ladatun tiedon omasta muististaan. Tämä tarkoittaa toki sitä, että uusimmat muutokset, jotka verkkosivulle tehdään eivät tule näkyviin ennen verkkoyhteyden palaamista ja välimuistin päivitystä. Täten on hyvä suunnitella myös se, miten isoa tallennustilaa verkkosivun tallennukseen tarvitaan, jotta se voitaisiin tallentaa mahdollisimman pienelle määrälle muistia. Myös hyvä huomio tässä, on se, ettei jokainen selain tue välimuistiin tallennusta. (MDN web docs, n.d)

ServiceWorker on erilaisiin tapahtumiin pohjautuva työkalu. Sitä voidaan ohjata **Download**, **Install** ja **Activate** kutsuilla. Näitä kutsuja hyödyntäen voidaan käskeä ja muovata verkkosivuston sisällä tapahtuvia sivustomuutoksia, eli verkkosivuston sisällä liikkumista,

jolloin HTML tiedosto muuttuu toiseen. ServiceWorkeria hyödyntäen voimme hallita, miten sivusto käyttäytyy erilaisissa tilanteissa. ServiceWorker myös käyttäytyy normaalin JavaScript tiedoston tavoin. (MDN web docs, n.d)

On myös olemassa erilaisia JavaScript workereita, joista yksi on ServiceWorker. Näitä ovat esimerkiksi ServiceWorker ja Web Worker. Nämä jakavat joitain ominaisuuksia keskenään, mutta työhön valittiin ServiceWorker. (Bitsofcode, 2018)

5 TYÖN KULKU

Ennen varsinaista työn aloittamista, tarvitsee versiohallintaan lisätä haara työtä varten. Tämä onnistuu Git Bash sovellusta käyttäen, siirtymällä ensin kansioon, joka pitää sisällään sivuston tiedot. Tässä esimerkissä siirrymme ensin levyosioon, josta kansion haemme käyttäen **cd f:** komentoa. Tämän jälkeen olemme F: levyosiossa, jolloin voimme suorittaa **cd Reactored** komennon. Tämä siirtää meidät F: levyosion alaiseen kansioon Reactored. Tämän jälkeen Git versiohallinta näyttää nykyisen haaran reitin oikealla puolella, tässä tapauksessa Master, joka on versiohallinnan päähaara. Voimme luoda uuden haaran käyttäen **git checkout -b offline-version**, jossa **git** tarkoittaa Git versionhallinnalle osoitettua käskyä, **checkout** antaa käskyn vierailla toisessa haarassa ja **-b offline-version** antaa käskyn luoda uuden haaran nimeltä offline-version.

5.1 Nettiyhteyden tilan tarkistus

Voimme avata Visual Studio Code:n, jolla päästään luomaan ensimmäiset muutokset itse koodiin. Aloitamme luomalla palvelun Angulariin, joka on JavaScript pohjainen kehikko. Tämän palvelun tarkoituksena on tarkistaa, onko käyttäjällä verkkoyhteyttä, jonka pohjalta voimme piilottaa sivustolta verkkoyhteyttömään tilaan kuulumattomat asiat. Tähän tarkoitukseen saadaan tieto **window.navigator.online** käyttäen, joka palauttaa tiedon siitä, onko verkkoyhteyttä. Kutsumme tätä palvelua nimellä isOnline.

Kuva 2 isOnline palvelu

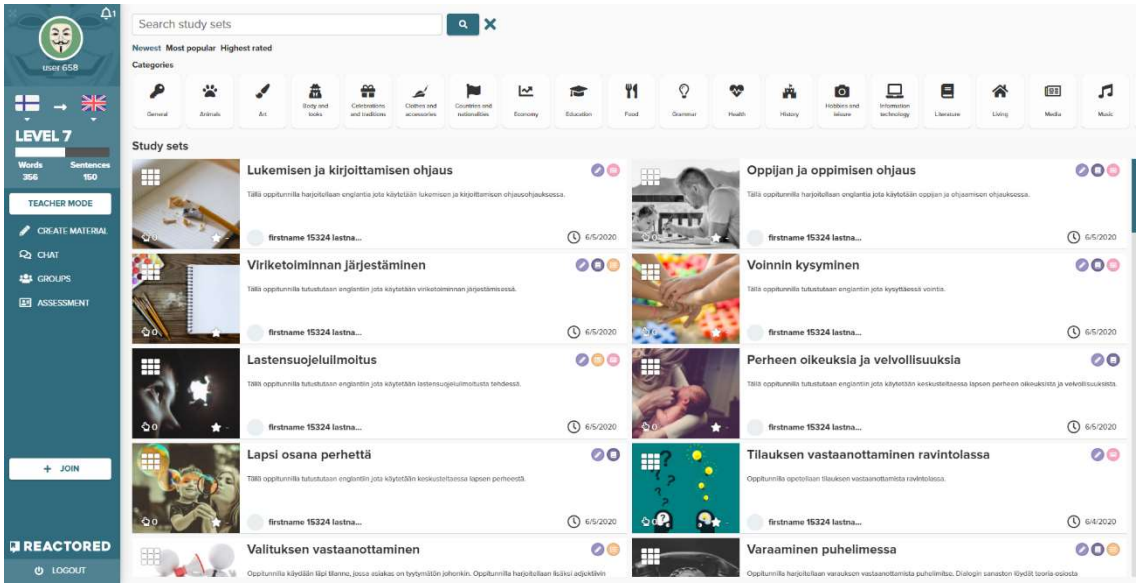
```
// Check the online status
"use strict";

angular.module("reactored.ui").service("isOnline", function() {

this.checkOnlineStatus = async () => {
    let ifConnected = window.navigator.onLine;
        return ifConnected;
    };
})
```

Tätä palvelua voidaan kutsua JavaScript tiedostoissa, jonka pohjalta voidaan tehdä päätös siitä, näytetäänkö tietoa silloin, kun verkkoyhteyttä ei ole. Tämä on hyödyllinen keino piilottamaan sivupalkissa näkyviä pikalinkkejä, sillä keskustelupalsta ei ole verkkoyhteyttömässä tilassa tarvittava.

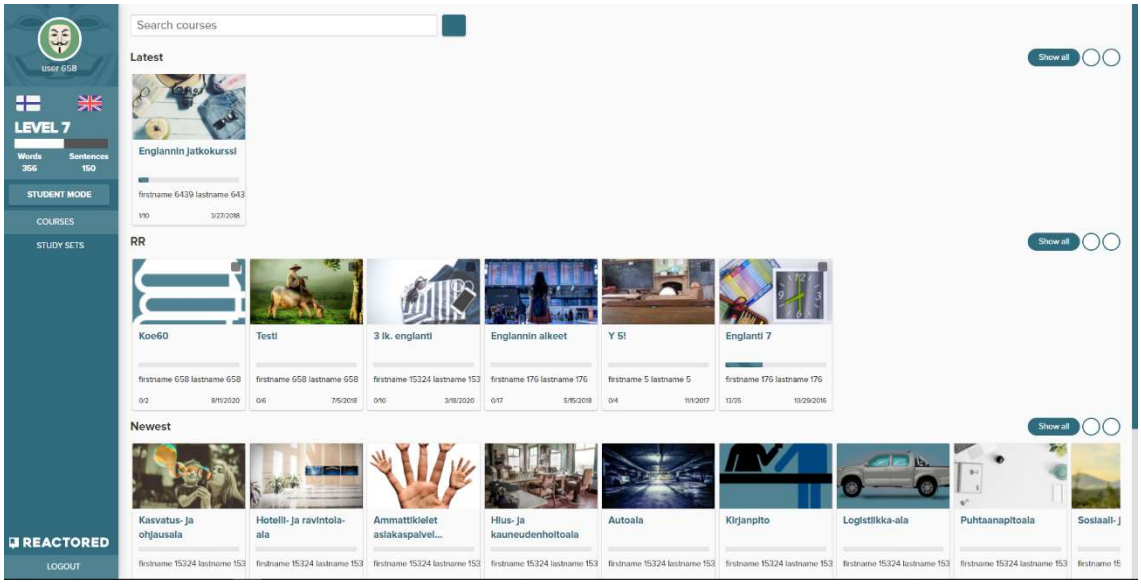
Kuva 3 isOnline palvelun kutsu JavaScript tiedostossa

```
// check internet connection
this.connection = await isOnline.checkOnlineStatus();
if (this.connection) {
    console.log("HI");
}
```

Kuva 4 Sivusto verkkoyhteydellä



Kuva 5 Sivusto ilman verkkoyhteyttä



5.2 **Tiedon tallennus välimuistiin**

Vaikka isOnline palvelu tarkistaakin verkkoyhteyden tilan, sivusto näyttää vielä ”ei verkkoyhteyttä” viestin, silloin kun verkkoyhteys katkeaa. Tämä voidaan todeta esimerkiksi Chromen offline toiminnolla. Sivuston tiedot täytyy tallentaa selaimen välimuistiin, joka säilöö HTML, CSS, JavaScript sekä mediatiedostoja käyttäjän laitteelle. Tämä mahdollistaa sen, että kun nettiyhteys katkeaa, sivusto voi vielä ladata tiedot välimuistista, sekä näyttää pyydetyt tiedot näytöllä. Tätä varten tarvitsemme kahta tiedostoa, ServiceWorker palvelua itse palvelun rekisteröimiseen, sekä sille osoitettua välimuistiin tallentamiseen tarkoitettua sw_cached_pages tiedostoa, jonka tarkoituksena on kerätä tarvittavat HMTL, CSS, JavaScript sekä mediatiedostot. Nämä tiedostot voidaan samassa sw_cached_pages tiedostossa lähettää selaimelle, sekä nettiyhteyden katketessa hakea välimuistista.

ServiceWorkerin tarkoituksena on siis ladata sille annetut tiedot selaimen välimuistiin sille osoitetun nimen perusteella. Tämä tapahtuu niin, että kun asiakas vierailee sivustolla, ServiceWorker lähettää sille ennalta määritellyt tiedot. Välimuisti pitää tietoja tallessa ennaltamääritellyn ajan, jonka pystyy tarvittaessa muuttamaan. Tässä tapauksessa jätämme tiedot välimuistiin päivän ajaksi. ServiceWorker käyttää tietojen lähettämiseen **install** komentoa, jonka alle syötetään lähetettävät tiedostot, sekä määritellään lähetettävän päämäärän nimi. Kun nettiyhteys katkeaa voidaan tiedot puolestaan hakea välimuistista käyttäen **fetch** komentoa, jonka alle voidaan erikseen määritellä mitä tietoa palautetaan. Tässä esimerkissä palautettiin pyydettyyn päämäärään liitetyt tiedostot. Viimeisenä silauksena käytetään **activate** komentoa, jolla putsaamme vanhat tiedot välimuistista, jos tiedostopäätteen nimi muuttuu. Tällä voimme luoda eri versioita välimuistiin, jos halutaan tehdä muutoksia, jotka eivät astuisi heti voimaan.

ServiceWorker palvelun tarkoituksena on aktivoida ServiceWorker toimimaan selaimessa. Tämä siis tarkoittaa sitä, että se lisää palvelun esimerkiksi Chrome selaimeen, kun taas sw_cached_pages antaa sille tarvittavia tietoja. **Navigator.serviceWorker** palauttaa tiedon siitä, onko ServiceWorker tuettu selaimessa, esimerkiksi Chrome-selain tukee tätä mahdollisuutta, mutta Safari-selain ei tue. Tätä tietoa verraten, voidaan antaa käsky **load,** jonka tehtävänä on ladata ServiceWorker. Kun ServiceWorker on ladattu, voidaan käyttää **navigator.serviceWorker.register(”sw_cached_pages.js”)**, joka asettaa ServiceWorker:in käyttämään tietonaan sw_cached_pages tiedostoa.

Kuva 6 sw_cached_pages

```
// Install Service Worker
self.addEventListener("install", function(event) {
    console.log("Service Worker: Installing....");

    event.waitUntil(
        // Open the Cache
        caches.open(cacheName).then(function(cache) {
            console.log("Service Worker: Caching App Shell at the moment......");

            // Add Files to the Cache
            cache
                .addAll(filesToCache)
                .then(() => console.log("Assets added to cache"))
                .catch(err => console.log("Error while fetching assets", err));

            return;
        })
    );
});

// Fired when the Service Worker starts up
self.addEventListener("activate", function(event) {
    console.log("Service Worker: Activating....");

    event.waitUntil(
        caches.keys().then(function(cacheNames) {
            return Promise.all(
                cacheNames.map(function(key) {
                    if (key !== cacheName) {
                        console.log("Service Worker: Removing Old Cache", key);
                        return caches.delete(key);
                    }
                })
            );
        })
    );
    return self.clients.claim();
});

self.clients.claim();

self.addEventListener("fetch", function(event) {
    console.log("Service Worker: Fetch", event.request.url);

    console.log("Url", event.request.url);

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

Kuva 7 SeviceWorker

```
// Show course info service.
"use strict";

angular.module("reactored.ui").service("serviceWorker", function(modal) {
    this.serviceWorker = () => {
        if (navigator.serviceWorker) {
            window.addEventListener("load", () => {
                navigator.serviceWorker
                    .register("/sw_cached_pages.js")
                    .then(reg => console.log("serviceWorker: registered"))
                    .catch(err => console.error(`ServiceWorker: error: ${err}`));
            });
        }
    };
});
```

5.3 **Erikoisfonttien lisäys**

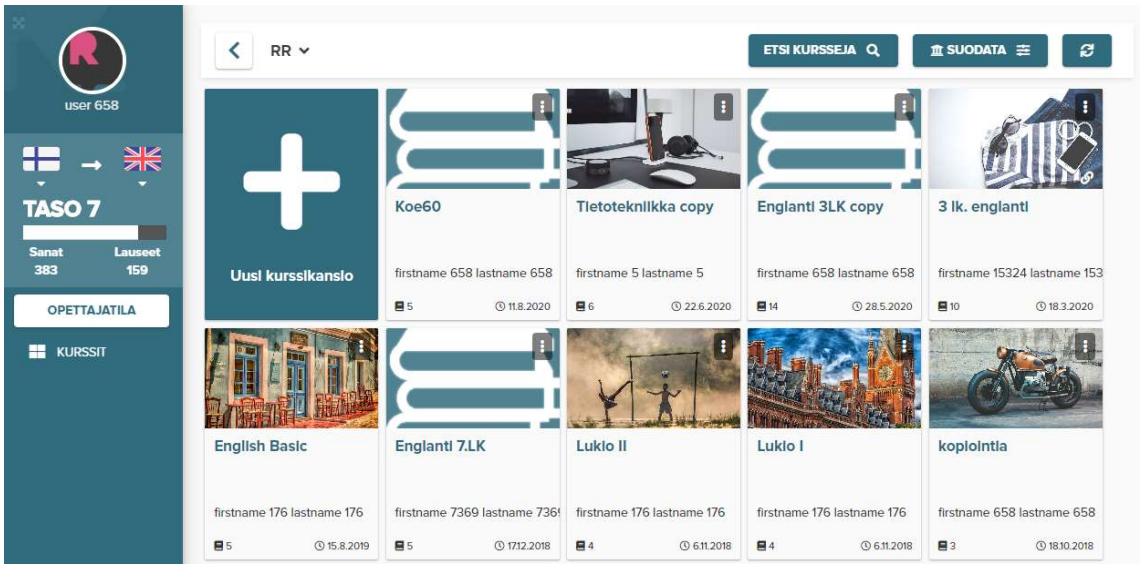
Erikoisfontit ovat siitä hankalia, että ne eivät lataudu ilman verkkoyhteyttä, sillä ne vaativat yhteyden verkkoon hakeakseen fontin tiedot. Tätä voidaan kuitenkin kiertää lataamalla fontit paikallisesti. Tässä esimerkissä käytämme fonttikirjastoa FontAwesome, joka käyttää tyypillisten tekstifonttien sijaan kuvakkeita. Tällä saadaan aikaan hienoja painikkeita, jotka käyttävät kuvakkeita.

Jotta päästään alkuun, täytyy fontit ensin ladata. Tämä voidaan FontAwesomin tapauksessa tehdä sivustolta **https://fontawesome.com/.** Kun fontit ovat latautuneet, täytyy kansio vielä purkaa, sillä se on pakatussa tilassa. Tämä voidaan tehdä esimerkiksi Winrar nimisellä ohjelmalla. Fonttien lisäysten jälkeen, halutaan kansiosta kopioida css- ja webfonts alikansiot, jotka voidaan sijoittaa projektin tyylittely kansion alle. (FontAwesome, n.d)

Kun fontit ovat projektissa, täytyy ne vielä ladata välimuistiin. Tämä voidaan tehdä siten, että nimeämme niiden sijainnin **sw_cached_pages** tiedoston **filesToCache** listan sisään. Tämä lista käydään läpi välimuisti tiedostojen latautuessa välimuistiin, kun asiakas vierailee sivustolla. Kaikkia tiedostoja ei ole syytä nimetä erikseen, vaan pelkästään **css/fontawesome-all.min.css** lisääminen on riittävää. Tämä tiedosto sisältää kaikki FontAwesome fonttikirjastoon kuuluvat kuvakkeet.

Nyt kun fontit ovat saatu lisättyä välimuistiin, voidaan sivu päivittää. Välimuistin päivitys on vielä nopeaa, joten voimmekin katkaista verkkoyhteyden lopputuloksen näkemiseksi. Kun verkkoyhteyttä ei ole, voimmekin nähdä, että fontit ovat latautuneet ja sivustolta tutut kuvakkeet näkyvät verkkoyhteydestä riippumatta.

Kuva 8 FontAwesome

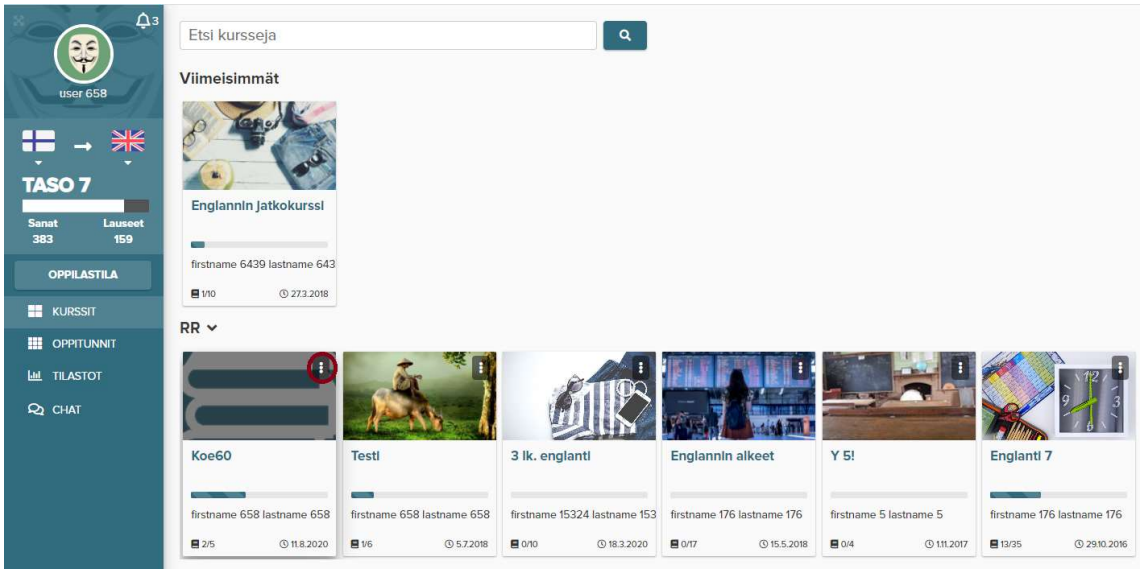


5.4 **Kurssien lisäys**

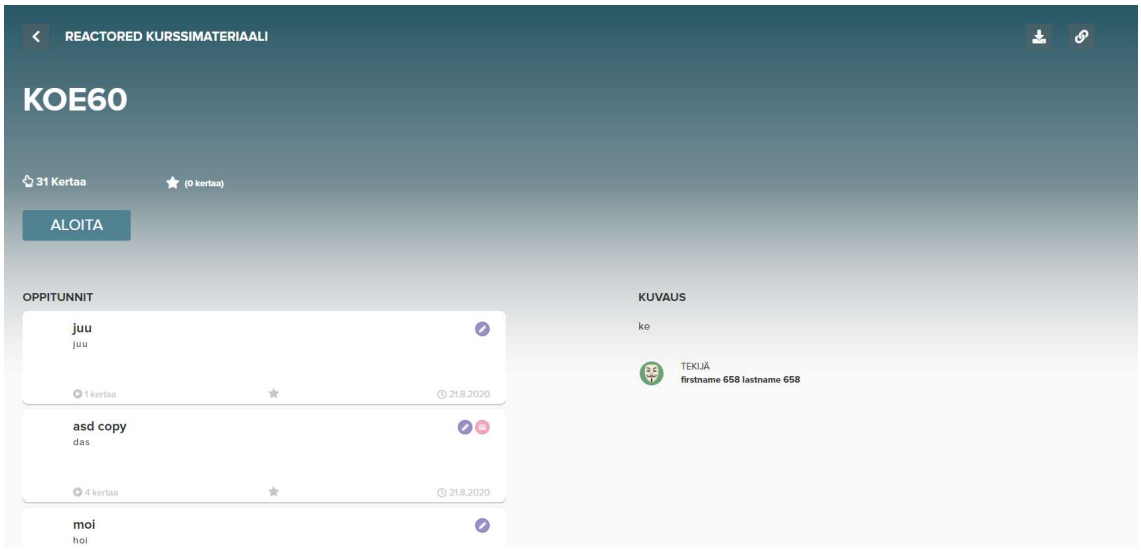
Nyt kun sivusto toimii yhteydettömässä tilassa, on myös tärkeää saada kurssit säilöttyä, jotta niitä voitaisiin opiskella. Kurssien sisältä löydämme oppitunnit, joten niiden lataus on myös välttämätöntä, kuten myös kuvien. Reactored oppitunti perustuu visuaaliseen oppimiseen ja oppitunnin kysymyksenä voikin olla ”mitä näet kuvassa”. Ilman kuvia edeltävään kysymykseen on mahdotonta vastata.

Aloittaaksemme kurssien lisäämisen, tarvitaan verkkoyhteydelliseen tilaan jonkinlainen painike, jota painamalla kurssi voitaisiin ladata. Tämä voidaan piilottaa kurssikortista kolmea pistettä painamalla avautuvaan välilehteen. Samalla kannattaa myös luoda painike, joka näkyy silloin, kun kurssi on jo ladattuna. Tämä näyttää sen, milloin kurssi on ladattu, mutta myös mahdollistaa kurssin poiston latauslistalta.

Kuva 9 Kurssikortin välinäkymän avaus



Kuva 10 Kurssikortin välinäkymä ja latauspainike



Voimme lisätä latauspainikkeen käyttäen HTML button ominaisuutta. Tämä ominaisuus rekisteröi painikkeen painalluksen, ja suorittaa siihen sidotun komennon. Näitä komentoja voidaan ohjata jokaiselle HTML sivulle erikseen niihin sidotuilla JavaScript tiedostoilla. Kun kyseessä on AngularJS kehys, käytämme käskyn kutsuen **ng-click** ominaisuutta. Voimme myös piilottaa latauspainikkeen silloin, kun kurssi on jo ladattuna ja poistopainikkeen, kun kurssi ei ole ladattuna. Tämä voidaan suorittaa käyttäen **ng-if** ominaisuutta ja lisätä vertailu

siihen, onko käyttäjällä kurssi ladattuna myöhemmin määriteltävää **checkDownloadedStatus** käskyä hyödyntäen.

Kuva 11 Kurssien lataus ja poistopainikkeet

```
<button class="course-view-close-btn" on-click="infoCtrl.downloadCourse()" ng-if="infoCtrl.connection && !infoCtrl.checkDownloadedStatus">
    <i class="fa fa-download"></i>
</button>
<button class="course-view-close-btn" on-click="infoCtrl.removeCourse()" ng-if="infoCtrl.connection && infoCtrl.checkDownloadedStatus">
    <i class="fas fa-times"></i>
</button>
<button class="course-view-close-btn" on-click="infoCtrl.toggleLinkList()">
    <i class="fas fa-link"></i>
</button>
```

Nyt kun kurssikorteissa on määritelty painikkeet poistoon ja lataukseen, täytyy niille asettaa komennot, eli käsketään niiden tehdä jotain silloin kun painiketta painetaan. Tätä varten luomme JavaScript tiedostoon käskyt **removeCourse** ja **downloadCourse**.

Kuva 12 Kurssien poisto ja lataus käskyt

```
/**
 * Download course to cache
 */
this.downloadCourse = async () => {
    console.log("Donlowad study set", this.course); // debug

    if (!this.course.id) return;

    offlineList.write(this.course);

    console.log("Downloading", this.course); // debug

    $scope.$apply();
};

/**
 * Remove course from cache
 */
this.removeCourse = async () => {
    console.log("Deleted study set", this.course); // debug

    if (!this.course.id) return;

    offlineList.deleteCurrent(this.course);

    console.log("Deleted", this.course); // debug

    $scope.$apply();
};
```

downloadCourse käskyyn määrittelemme käskyn **offlineList.write(this.course)**, joka antaa käskyn seuraavassa osiossa luotavaa **offlineList** palvelua varten. **This.course** on aiemmin luotu tieto siitä, mikä kurssi on kyseessä. Tämä kurssi halutaankin lähettää **offlineList** palvelulle **write** käskyä hyödyntäen, joka myöskin luodaan seuraavassa osiossa. Selkeyden vuoksi voidaan vielä kirjata kyseinen kurssi, jotta voidaan varmistaa kyseessä olevan oikea kurssi. **removeCourse** on käytännössä suora kopio edellisestä, sillä erolla, että **write** käskyn sijaan ajetaankin **deleteCurrent** käsky.

Nyt kun olemme määritelleet käskyt kurssien poistoon ja lataukseen, voidaankin siirtyä **offlineList** palvelun luontiin. Tämän palvelun tarkoitus on tallentaa, poistaa ja tarkistaa olemassa olevat kurssit ja myös siirtää ne selaimen paikalliseen tallennukseen, jottei tiedot häviäisi selaimen sulkeutuessa.

Kuva 13 Ensimmäinen osa offlineList palvelusta

```
// Storing data to offline list
"use strict";

angular.module("reactored.ui").service("offlineList", function() {
    this.list = JSON.parse(localStorage.getItem("local_list"));
    this.image_urls = JSON.parse(localStorage.getItem("image_list"));
    this.database_list = JSON.parse(localStorage.getItem("database_list"));
    if (!this.list) {
        this.list = [];
    }
    if (!this.image_urls) {
        this.image_urls = [];
    }
    if (!this.database_list) {
        this.database_list = [];
    }
    (this.write = async downloaded_course => {
        this.course = downloaded_course;

        const existing_course = this.list.find(ss => ss.id === this.course.id);

        console.log("existing", this.course, existing_course);

        if (this.list.length >= 3) {
            this.list.splice(2);
            alert("List full");
        }

        if (existing_course) {
            alert("Already exist");
        } else {
            this.course.study_sets.map(study_set => {
                study_set.study_item_groups.map(study_item_group => {
                    study_item_group.study_items.map(study_item => {
                        if (study_item.image) {
                            const image_url = study_item.image.bucket_url;
                            this.image_urls.push(image_url);
                        }
                    });
                });
            });
            this.list.push(this.course);
        }

        localStorage.setItem("local_list", JSON.stringify(this.list)); // store to local storage
        localStorage.setItem("image_list", JSON.stringify(this.image_urls)); // store to local storage
        let localList = JSON.parse(localStorage.getItem("local_list")); // Fetch list from local storage
        let imageList = JSON.parse(localStorage.getItem("image_list")); // Fetch list from local storage
        console.log("stored items", localList, imageList);
    }),
        (this.read = () => {
            let localList = JSON.parse(localStorage.getItem("local_list")); // Fetch list from local storage
            console.log(localList);
            return JSON.parse(localStorage.getItem("local_list"));
        }),
```

Yllä olevassa esimerkissä luomme ensin **write** ominaisuuden, jonka tarkoitus on tallentaa kurssit paikalliseen tallennukseen. Ennen kyseistä ominaisuutta asetamme kuitenkin **this.list** sekä **this.image_urls** joista **this.list** pitää sisällään listan kurssien tiedoista ja **this.image_urls** tiedon kurssiin liittyvistä kuvista, jotka siirrämme paikalliseen muistiin verkkolinkeistä ladattuina kuvina. Ennen virallista tallennusta halutaan varmistaa, onko kyseinen kurssi jo tallennuslistalla käyttäen **existing_course**, jossa haemme tulevasta kurssista **id** tiedon, ja vertaamme sitä listassa olevien kurssien **id** tietoon. Jos kurssia ei löydy, palauttaa se epätotuuden ja jos kurssi puolestaan löytyy listasta, palautetaan totuus. Tätä ominaisuutta hyödyntäen voimme tallentaa kurssin vain siinä tapauksessa, jos kurssi ei jo entuudestaan löydy listasta. Tämän jälkeen voimme myös määritellä halutessamme listalle pituudeksi esimerkiksi kolme kurssia, jolloin ladattavien kurssien määrä on rajallinen, joka puolestaan säästää hiukan tallennustilan kanssa. Tämä voidaan suorittaa käyttäen **this.list.length** käskyä käyttäen. Tämä käsky on normaalia JavaScript ominaisuutta, joka palauttaa tiedon kyseisen listan pituudesta. Esimerkissä käytämme maksimipituutena kolmea tietoa. Annamme myös ilmoituksen silloin, jos kurssi jo löytyy listasta. Tämän jälkeen pääsemme itse tallennukseen, joka suoritetaan **push** ominaisuutta käyttäen, joka siirtää tulevan kurssin listan viimeiseksi kohteeksi. Myös tämä ominaisuus on normaalia JavaScriptiä. Sama toimenpide halutaankin suorittaa myös **this.image_urls** listalle. Tämän jälkeen tietomme ovat paikallisissa listoissa **this.image_urls** ja **this.list**, mutta tiedot halutaan tämän jälkeen siirtää paikalliseen tallennukseen, jolloin tiedot säilyvät esimerkiksi välilehden tallennuksen jälkeen. Tämä voidaan suorittaa **localStorage** kohteeseen kohdistuvalla **setItem** käskyllä, jolle voimme antaa nimeksi **local_list**, johon tallennamme **this.list** tiedot ja **image_list**, johon puolestaan tallennamme **this.image_urls** listan tiedot. Joudumme myös määrittelemään seuraavaksi tiedoksi kyseisten listojen nimet. Tämä ominaisuus tallentaa täten kyseisten listojen tiedot paikalliseen muistiin ja viimeistelee **write** ominaisuuden luonnin.

Seuraavaksi ominaisuudeksi haluamme luoda **read** ominaisuuden, jonka tarkoituksena on tarkistaa, onko kyseinen kurssi jo tallennettuna ja tällä tiedolla voimme myöhemmin näyttää poisto tai lataus ikonit kurssin välinäkymässä. Tätä ominaisuutta varten halutaan käyttää kohteeseen kohdistavaa **getItem** käskyä. Tämä on käytännössä vastakohta aiemmin käytetylle **setItem** käskylle. Kun **setItem** lähettää tiedon paikalliseen muistiin, **getItem** hakee muistista tiedon.

Kuva 14 Toinen osa offlineList palvelusta

```
(this.deleteCurrent = async deleted_course => {
    this.course = deleted_course;
    const existing_course = this.list.findIndex(ss => ss.id === deleted_course.id);
    console.log("Index of currect", existing_course);
    this.list.splice(existing_course, 1);
    localStorage.setItem("local_list", JSON.stringify(this.list)); // store to local storage

    this.course.study_sets.map(study_set => {
        study_set.study_item_groups.map(study_item_group => {
            study_item_group.study_items.map(study_item => {
                if (study_item.image) {
                    const image_url = study_item.image.bucket_url;
                    this.image_urls.splice(image_url, 1);
                }
            });
        });
    });

    localStorage.setItem("image_list", JSON.stringify(this.image_urls));

    var localList = JSON.parse(localStorage.getItem("local_list")); // Fetch list from local storage
    let imageList = JSON.parse(localStorage.getItem("image_list")); // Fetch list from local storage
    console.log("stored items", localList, imageList);
}),
```

Nyt kun kurssit voidaan lisätä paikalliseen tallennukseen ja myös tarkastaa, onko kurssi tallennettuna, halutaan myös luoda poisto ominaisuus. Tämä ominaisuus kutsutaan, kun poista nappia painetaan kurssin välinäkymässä. Ominaisuus toimii käytännössä samalla tavalla kuin kurssin tallennus, sillä erolla, että **this.list** listasta poistetaan kyseinen kurssi. Tämä voidaan suorittaa käyttäen **this.list.splice** ominaisuutta, joka pilkkoo listasta siitä määritellyn kohdan. Tämä kohta voidaan määrätä hakemalla kurssin **id** tiedon perusteella kohta, jossa kurssi on listassa. Käytämme tähän **existing_course** muuttujaa, joka on nimeltään sama kuin aiemmin käytetty, mutta eroavaisuutena palauttaa kurssin kohdan listasta totuuden tai epätotuuden sijaan. Näin voimme antaa **splice** ominaisuudelle kohdan, josta lista täytyy pilkkoa. Kun lista on pilkottu, voimme taas tallentaa nykyisen listan paikalliseen muistiin. Nyt meillä on ominaisuus poistoon, tallennukseen sekä hakuun. Voimme käyttää hakuominaisuutta nyt JavaScript tiedostossa, jossa saamme piilotettua, tallenna napin, kun kurssi on jo ladattuna.

Kuva 15 Kurssin haku

```
this.checkDownloadedStatus = offlineList.find(this.course);
console.log("Is downloaded", this.checkDownloadedStatus);
```

5.5 **Valmiin tuloksen tarkistus**

Nyt kun kurssit, fontit ja kuvat ovat ladattuina sekä latauspainikkeet luotuna, voimme tarkistaa onko sivusto oletetun kaltainen. Oletuksena tällä hetkellä on, että sivupalkissa näkyy vain ne ominaisuudet, jotka eivät ole piilotettuina **isOnline** palvelua hyödyntäen. Tämä voidaan tarkistaa katsomalla sivupaneelia ja tarkastamalla näkyykö vain ”kurssit” painike.

Kuva 16 Sivupaneeli

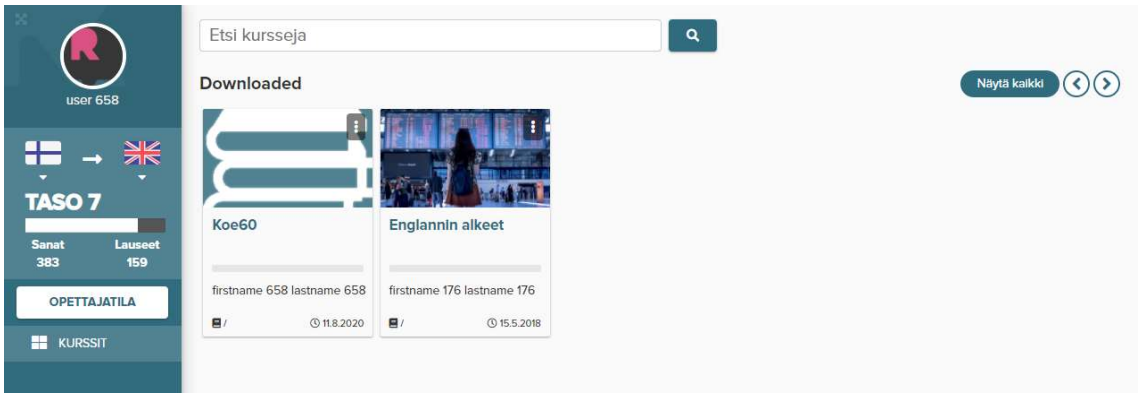
Yllä olevassa kuvassa näemme, että **isOnline** palvelumme on toimiva ja saamme piilotettua ylimääräiset tavarat sitä hyödyntäen. Tämän jälkeen on hyvä tarkastaa kurssien lataus ja poisto. Painamalla lataus painiketta, kurssin pitäisi latautua ja painikkeen muuttua poista napiksi.

Kuva 17 Latauspainike

Kuva 18 Poistopainike

Nähdään myös lataustilan tarkistuksen toimivan, sillä latauspainike muuttuu poistopainikkeeksi sitä klikkaamalla. Voidaan vielä painaa latauspainiketta muutamalle kurssille, nähdäksemme onko kurssien lataus todella toiminut. Otamme nettiyhteyden pois ja nyt kurssit näkymässä pitäisi näkyä vain ladatut kurssit.

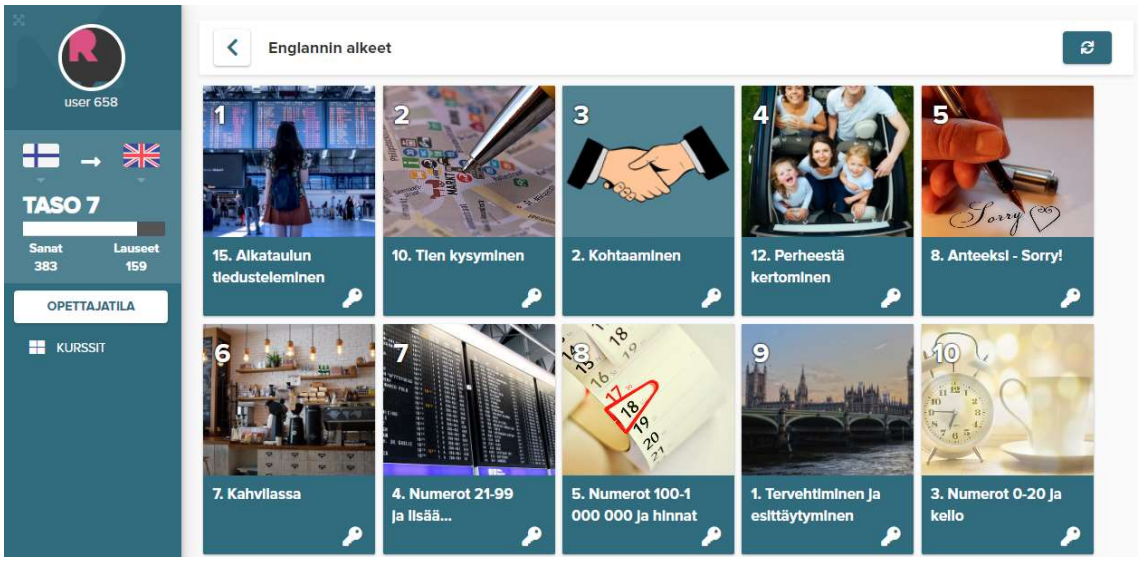
Kuva 19 Ladatut kurssit näkymä



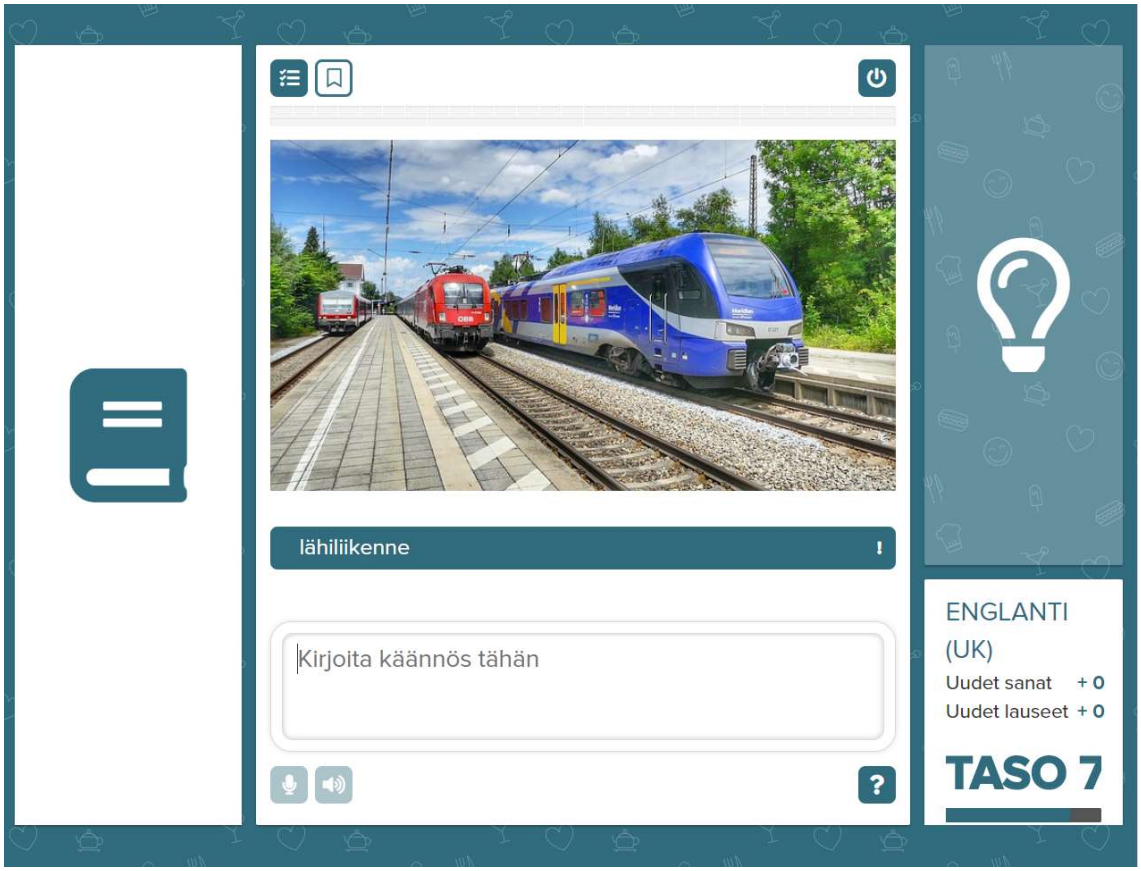
Voidaan todeta kurssien latauksen sekä yhteyden tilan tarkistuksen toimivan. Voidaan myös tarkistaa poistonappia painamalla kurssin poiston onnistuvan, joka esimerkissä toimi moitteettomasti. Nyt jäljelle jääkin tarkistus oppitunnista ja siihen liittyvistä kuvista. Avataan oppitunti painamalla kurssikorttia ja sitten valitsemalla jonkin oppitunnin avautuvasta

oppitunti näkymästä. Tämä avaa kyseisen oppitunnin ja optimaalisessa tilanteessa näkyviin pitäisi tulla kuva, kysymys ja vastauslaatikko tai vastausvaihtoehtoja.

Kuva 20 Oppitunti näkymä



Kuva 21 Oppitunti



Jos lopputulos on kuin yllä olevissa kuvissa, voimme todeta kurssien ja siihen liittyvien kuvien latauksen onnistuneen. Vastauskenttään vastauksen kirjoittamalla pitäisi päästä seuraavaan kysymykseen ja kuvan pitäisi vaihtua. Tämä toimi esimerkissä oletetusti.

6 LOPPUPOHDINTA

Onnistunut lopputulos on asia mitä työssä lähdettiin hakemaan ja myös sen suoritus onnistui oletetusti. Oppitunteja voidaan ladata ja niiden suorituskin onnistuu.

Alun perin kuvien lisääminen oppituntiin verkkoyhteyttömässä tilassa katsottiin lisäominaisuutena, mutta se lisättiinkin nopean toiminnan johdosta jo työn aikana. Tämän lisäksi kyseisinä lisäominaisuuksina katsotaan **responsivevoice** lisäpalikkaa, jonka tarkoitus on lukea käyttäjän kirjoittama oikea vastaus konelukijalla. Tämän lisäys vaatii uuden lisäpalikan löytämistä, sillä vanha lisäpalikka ei tue verkkoyhteydetöntä tilaa. Sama pätee myös mikrofoonin lisäykseen. Tämän ominaisuuden tulisi kirjata käyttäjän puhuma asia tekstinä vastauskenttään.

Tämä on ollut todella opettava työ, sillä allekirjoittaneella ei ollut aiempaa kokemusta **serviceWorker** ominaisuudesta. Tämä ominaisuus on kuitenkin hyvin dokumentoitu ja siitä löytyykin paljon tietoa Googlea hyödyntäen. Lopputulos onkin oletetun kaltainen ja odottaa nyt testausvaihetta testaustiimiltä, ennen sen siirtymistä asiakaskäyttöön.

Lähteet

Bitsofcode. (2018). Web workers vs Service workers vs Worklets. Haettu 19.12.2020 osoitteesta https://bitsofco.de/web-workers-vs-service-workers-vs-worklets/

Elisabet Service. (2017). Vieraan kielen oppimisen vaikeudet. Haettu 01.09.2020 osoitteesta https://oppimisvaikeus.fi/tietoa/teemat/tietoa-oppimisvaikeuksista/artikkeli-vieraan-kielen-oppimisen-vaikeudet/

FontAwesome. (n.d.). Hosting Font Awesome Yourself. Haettu 09.09.2020 osoitteesta https://fontawesome.com/how-to-use/on-the-web/setup/hosting-font-awesome-yourself

Google. (n.d.). Maps. Haettu 19.8.2020 osoitteesta https://www.google.fi/maps

Heroku Dev Center. (n.d.). Increasing Application Performance with HTTP Cache Headers. Haettu 17.08.2020 osoitteesta https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers#http-cache-headers

MDN web docs. (n.d.). Service Worker API. Haettu 09.09.2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

MDN web docs. (n.d.). HTTP caching. Haettu 17.12.2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

MDN web docs. (n.d.). Window.localStorage. Haettu 09.09.2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

O2. (n.d.). Handyvertrag. Haettu 20.8.2020 osoitteesta https://www.o2online.de/tarife/handyvertrag

Vodafone. (n.d.). Mobilfunk. Haettu 20.8.2020 osoitteesta https://www.vodafone.de/privat/mobilfunk.html

Wikipedia. (n.d.). Internet. Haettu 19.8.2020 osoitteesta https://fi.wikipedia.org/wiki/Internet