# Varautuminen yheyskatkoksiin Single-Page –web-sovelluksissa

## Web-teknologioiden kehittyminen

Viimeisen vuosikymmenen aikana koko ohjelmistokehitysala on ollut murroksessa. Siinä missä aikaisemmin ainoa varteenotettava toteutustekniikka oli alustakohtainen natiivisovellus, on web-teknologioiden nopea kehitys mahdollistanut nyt myös liiketoimintakriittisten sovelluksien toteuttamisen selainympäristössä. Tänä päivänä käyttäen HTML-, CSS- ja JavaScript-teknologioita voidaan selaimessa toteuttaa niin taulukkolaskentasovelluksia kuin CAD-sovelluksia.

Ennen yllämainittujen teknologioiden kehittymistä Internet-sivuissa paino oli sanalla *sivu*, eikä varsinaisista *sovelluksista* voitu vielä puhua. Käyttäjän selain pyysi palvelimelta sivun, ja palvelimen vastauksen perusteella selain muodosti sivusta näkymän. Tässä näkymässä saattoi olla jotain kenttiä käyttäjän täytettäväksi. Tällöin kenttien tiedot lähetettiin palvelimelle, palvelin teki laskentaa tietojen perusteella, ja selaimelle lähetettiin vastaukseksi jälleen kokonainen sivu. Tässä mallissa selain toimii *tyhmänä päätteenä*, ja käyttäjä on täysin palvelimen toiminnan armoilla.

Tämän paradigman rikkomisen on mahdollistanut erityisesti JavaScript-kielen kehittyminen ja toteutettu tuki yleisimmin käytössä olevissa selaimissa. Käyttäen JavaScriptiä on Internet-sivujen toteuttamiseen vaadittavaa logiikkaa ja toiminnallisuutta saatu siirrettyä palvelimelta myös selaimelle, joka on mahdollistanut rikkaamman käyttökokemuksen toteuttamisen ja samalla luonut pohjan varsinaisten web*-applikaatioiden* toteuttamiselle.

Nykyaikainen *web-applikaatio* on toteutettu niin kutsutulla *single page* –paradigmalla, missä käyttäjä lataa palvelimelta ainoastaan yhden sivun (siihen liittyvine skripteineen), ja tämän jälkeen JavaScriptillä toteutettu toiminnallisuus tekee taustalla asynkronisesti pyyntöjä erinäisiin API:n, muokaten pieniä osia selainnäkymästä yksitellen. Näin toimien sivua ei tarvitse ladata ja piirtää aina kokonaisuudessaan uudestaan, vaan näin täytyy tehdä ainoastaan sille osalle sivua, mikä oikeasti muuttuu.

## Ongelmakohta: liiketoimintakriittinen sovellus ja heikko Internet-yhteys

Vaikka selaimeen saadaan lisää älyä, ja osa liiketoimintalogiikasta saadaan siirrettyä sinne, on web-sovellus kuitenkin aina pohjimmiltaan riippuvainen palvelimesta. Selainympäristön muodostuessa houkuttelevammaksi alustaksi sovelluksille, toteutetaan sille yhä liiketoimintakriittisimpiä applikaatioita. Mitä merkittävämpi sovelluksen käyttötarkoitus on, sitä tärkeämpää sen toiminnan varmistaminen on myös tilanteissa, missä Internet-yhteyden laatu ei ole optimaalinen. Joillekin sovelluskohteille tämän ongelman ratkaiseminen ja toiminnan takaaminen myös yhteyskatkon aikana voi olla pakollista, eikä pelkästään oleellista käyttökokemuksen kannalta.

Tämä on ongelma, minkä ratkaisuun diplomityöni keskittyy. “Kuinka voimme selainympäristössä varautua Internet-yhteyden huonoon laatuun tai hetkittäiseen katkeamiseen?”

## Offline-tuki single-page –web-applikaatioon

Diplomityössä implementoitiin osittainen offline-tuki jo käytössä olevaan web-sovellukseen. Kyseinen sovellus on työkalu päiväkodin hoitajille. Sovelluksen avulla hoitajat näkevät aina ajantasalla olevan läsnäolotilanteen päiväkodissa, he pystyvät selaamaan lapsien tietoja sekä merkitsemään lapset saapuneiksi ja myöhemmin lähteneiksi.

Kaikille sovelluksen ominaisuuksile ei toteutettu offline-tukea. Ainoastaan sellaisille toiminnallisuuksille, jotka ovat kaikista kriittisimpiä, mahdollistettiin käyttö myös yhteyskatkon aikana.

Offline-tuen ydinominaisuudet toteutettiin käyttäen seuraavia tekniikoita:

**HTTP Cache Headers & Application Cache**

Kaikki staattiset resurssit, kuten kuvaikonit ja tyylitiedostot, talletettiin selaimen välimuistiin. Riippuen resurssin tyypistä, se tallennetaan joko ennaltamäärätyksi ajaksi käyttäen *HTTP Cache Headeria*, tai sitten toistaiseksi käyttäen *Application Cachea*. Tämän avulla selain pystyy näyttämään kerran jo ladattuja resursseja, vaikka verkkoyhteys katkeaisi.

**Local Storage**

*Local Storage* on selaimessa oleva domain-kohtainen avain-arvo –varasto, jonne sovellus tallentaa kaikki API:sta saamansa vastaukset. Kun yhteyskatko havaitaan, haetaan sovelluksen tarvitsemaa tietoa API:n sijasta local storagesta, jos kyseinen data oli jo kertaalleen haettu. Tämä toteutus on mahdollisimman yksinkertainen; niin kauan kuin Internet-yhteys toimii, local storageen ainoastaan kirjoitetaan tietoa. Kun yhteys katkeaa, AJAX-pyyntöjen sijasta tietoja pyritään hakemaan local storagesta. Jos kyseisiä tietoja ei ole talletettu sinne, ilmoitetaan käyttäjälle että tietoa ei ole saatavilla.

*Local Storagen* avulla API-funktioiden vastaukset saadaan talletettua selaimeen, ja käyttäjä pystyy selaamaan yhteyskatkonkin aikana sellaisia näkymiä, joissa hän on jo vieraillut.

***Promise*-jono kirjauksia varten**

Sovelluksen tärkein toiminnalisuus, jonka tulee olla saatavilla käyttäjille kaikissa olosuhteissa, on lapsien sisään- ja uloskirjaaminen. Tämän mahdollistamiseksi myös offline-tilassa sovellukseen toteutettiin *Promise*-*jono*, johon kaikki kirjaukset lisätään. Tietyllä intervallilla jonon vanhinta *Promisea* (eli “lupausta tulevaisuuden arvosta”) pyritään lähettämään palvelimelle. Jos yhteys on poikki, eikä kirjauksen lähettäminen onnistu, yritetään sen lähettämistä palvelimelle hetken päästä uudelleen. Tämän ansiosta käyttäjä voi tehdä kirjauksia, oli yhteys toiminnassa tai ei, ja käyttäjän jatkaessa tehtäviään merkinnät talletetaan palvelimelle kun yhteys on jälleen aktiivinen.

## Tulokset – mitä opittiin?

Käyttäjäkokemus paranee, kun sovellus ei jumiudu välittömästi yhteyden hävitessä. Kaikkia sovelluksen toiminnallisuuksia ei tarvitse tukea offline-tilassa, vain kriittisten toiminnallisuuksien mahdollistaminen helpottaa käyttäjiä jo merkittävästi. Jo osittainen offline-tuki mahdollistaa sovelluksien käytön sijainneissa, joissa Internet-yhteys laatu on heikko.

Vaikka sovellukseen ei aiota / ei tarvitse toteuttaa offline-tukea, voi offline-tuen mahdollistavien tekniikoiden, kuten selaimen välimuistin tehokasta hyödyntämistä, käyttää paremman käyttäjäkokemuksen saavuttamiseen. Tämä realisoituu esimerkiksi sovelluksen lyhyempinä käynnistysaikoina, sekä pienempänä tiedonsiirron määränä.

Monenlaiset sovellukset voivat hyötyä offline-tuesta. Tämän vuoksi sen implementoinnin mahdollisuus kannattaa ottaa huomioon sovellusarkkitehtuuria suunnitelteaessa, vaikka sitä ei olisikaan ensimmäisellä tiekartalla mukana. Jo pienillä ratkaisuilla, kuten keskittämällä API-pyyntöjen tekeminen yhteen paikkaan, voidaan helpoittaa mahdollista offline-tuen toteuttamista merkittävästi.

Koska web-sovellukset ovat offline-tuesta huolimatta aina tietyllä tavalla riippuvaisia palvelimesta, tulee käyttäjien jollain tasolla olla tietoisia asiakas-palvelin –mallin toimintaperiaatteesta. Päikky-sovelluksessa käyttäjän tulee huolehtia, että päivän lopuksi *Promise-jono* on kokonaisuudessaan lähetetty palvelimelle; muutoin seuraavan päivän tiedot ovat lähtökohtaisesti jo pielessä. Jotta tämän tehtävän varsinainen tarkoitus sisäistetään, tulee käyttäjän ymmärtää, että datan varsinainen tallennuspaikka on muualla sijaitseva palvelin, eikä käyttäjän puhelin. Tämän asian hahmottamisessa oli monella diplomityötä varten haastatellulla päiväkodin hoitajalla ongelmia.

## 