Tallinna Ülikool

Digitehnoloogiate instituut

JavaScript Service Workeri kasutamine veebirakenduse näitel

Seminaritöö

Autor: Andre Post

Juhendaja: Romil Rõbtšenkov

Autor:.....”.....”2017

Juhendaja:.....”.....”2017

Instituudi direktor:.....“.....“2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda pole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, seisukohad, kirjandusallikatest ning mujalt pärinevad andmed on viidatud.

....(kuupäev)(autor)

Sisukord

Sissejuhatus

Mitmed töölaua rakendused on liikumas veebi, võttes kasutusele veebipõhiseid lahendusi funktsionaalsuse säilitamiseks. Nii interneti kiirused, kui ka interneti kasutusmaht on siiani piiratud. Selleks, et tagada rakenduse toimimine on mõistlik salvestada enim kasutatud failid vahemällu. See muudab rakendused kiiremaks ja annab võimaluse ka rakendust kasutada võrguühenduseta. Mälus hoidmist võimaldav tehnoloogia ei ole veel kindel ja on alles implementeerimis järgus, kuid kasutajate arv on pidevalt kasvamas. Eestis on näiteks paljud e-poed võtnud kasutusele tõuketeavitused (ingl push notifications), mis toetuvad vastavale taustale ja võrguta olekus töötavale tehnoloogiale.

Seminaritöö eesmärgiks on anda ülevaade JavaScript Service Worker'i kasutamisest. Selleks tutvustatakse veebirakenduse toimimist, Service Worker'i eelkäijat AppCache'i ja Service Worker'it ennast. Samuti antakse ülevaade näidisrakenduse arendamisest, mis kasutab Service Worker'it.

Seminaritöö teema valik tulenes sellest, et veebilehitsejate loojad on lõpetanud toetuse Service Worker'i eelkäija AppCache'ile ja üritavad veenda veebiarendajaid valima AppCache'i asemel Service Worker'i tehnoloogiat. Seminaritöö pakuks huvi Eesti veebiarendajatele, kellel on plaanis teha veebirakendusi, mida oleks võimalik ka võrguühenduseta kasutada. Lisaks ka neile, kellel on rakendusse jäänud AppCache, aga soovivad seda välja vahetada uuema tehnoloogia vastu.

1 Veebirakenduse kasutamine võrguta olekus

Veebirakendus koosneb klientidest (ingl *clients*) ja serveritest (ingl *servers*). Kliendid on tavalised internetti ühendatud seadmed. Serverid on arvutid mis talletavad veebilehti, andmeid ja rakendusi. Kui klient soovib ühendust saada veebilehega, laetakse talle serverist veebilehe koopia (Kammardi, 2017). Minnes veebilehele juhtuvad järgmised sündmused (Kammardi, 2017):

1. Veebilehitseja võtab ühendust nimeserveriga ja leiab päris serveri aadressi kus veebileht asetseb.
2. Veebilehitseja saadab HTTP päringu serverisse, millega küsib koopiat veebilehest kliendile.
3. Kui server kinnitab heaks kliendi päringu, saadab server kliendile „200 OK“ vastuse, mis tähendab, et kliendil on luba saada koopia veebilehest. Server saadab kliendile veebilehe failid andme pakettidena.
4. Veebilehitseja paneb andme paketid kokku terviklikuks veebileheks ja kuvab seda kliendile.

Algusest peale on veebilehitsejad andnud kasutajatele lihtsa lahenduse, kuidas veebilehti salvestada kasutaja masinasse ja hiljem neid avada isegi ilma võrguühenduseta. Nupp või menüüriba valik „Save Page As...“ salvestab veebilehe kasutaja masinasse HTML dokumendina. Serveris andmete uuenemisel ei uuendata kasutaja arvutis html dokumenti, mis ei tee sellest ideaalset lahendust. (Zajdband, 2016)

Chrome for Android’il on ka võimalus laadida leht mällu. Lingil näppu hoides avaneb menüü, kust saab valida „Lae link“ (ingl *download link*), mis laeb lingil oleva lehe alla. Lisaks, kui külastada mõnda lehte võrguühenduseta, on lehel nupp „Lae leht hiljem“ (ingl *download page later*). Võrguühenduse saamisel laetakse kohe leht alla mällu. Uue akna avamisel kuvab Chrome allalaetud lehed mälust uuesti kasutamiseks. (Oppenheimer, 2017)

Veebilehitsejad on hakanud ka toetama sellised tehnoloogiaid nagu AppCache ja Service Worker, millest antakse ülevaade järgmistes alapeatükkides.

1.1 AppCache

HTML5-ga tuli kasutusele võimalus kasutada veebirakenduse vahemälu läbi AppCache tehnoloogia. See andis võimaluse kuvada kliendile veebilehte isegi siis, kui kliendil puudus võrguühendus. Lisaks sellele, et sai veebilehte ilma võrguühenduseta kasutada, tegi AppCache ka lehed kiiremaks, kuna andmeid loeti vahemälust (Mozilla, 2017).

AppCache tekitas veebiarendajatele segadusi ja tõstis esile mõningaid tähtsaid probleeme. Kõige suurem probleem manifest failidega ja AppCache'iga on see, et kui lehe andmeid uuendatakse, siis kasutaja näeb ikka vanu cachitud andmeid kuni ta lehe uuendab (ingl *refresh*). Alles siis laetakse cachi uued failid ja kasutaja saab neid näha. Üksikut faili ei saa uuendada ning, kõik failid peab uuesti salvestama vahemällu isegi need, mida ei ole uuendatud. Kui viide manifest'i failile on lisatud veebirakenduse HTML koodi, siis kõik ressursid laetakse sünkroonselt kohe, kui manifest'i fail on laetud veebilehitseja poolt. See tähendab, et ressursid, mida ei ole veel hetkel vaja, laetakse kohe alguses AppCache'ist. Raskem on kontrollida seda, mis järjekorras ressursse laetakse (Bloom, 2012). See viis lõpuks AppCache tehnoloogia asendamisele Service Worker'iga. Võttes kokku põhilised raskused, mis tekkisid kasutades AppCache'i, saab välja tuua järgmiselt (Zajdband, 2016):

1. vahemällu salvestatud failid loetakse alati, isegi siis, kui kasutajal on võrguühendus, mis tegi vahemälu uuendamise raskeks,
2. failid vahemälus uuendatakse ainult siis, kui manifest'i uuendatakse.

Autoril ei õnnestunud leida töö koostamise ajal ühtegi head näidet rakendusest, mis kasutaks AppCache'i. Nii veebilehitsejate loojad, kui ka rakenduste arendajad, suruvad AppCache'i maha ja üritavad veenda kõiki üle minema uuemale tehnoloogiale, nagu tagaplaanil töötav script Service Worker'ile. Sellest annab ülevaate järgmine peatükk.

1.2 Service Worker

Service Worker on tehnoloogia, mida veebilehitseja käivitab tagaplaanil ja on lehest eraldatud. Service Worker võimaldab kasutada funktsionaalsust, mille jaoks ei ole vaja kasutaja interaktsiooni või võrguühendust (Gaunt, 2017).

Veebirakendused eeldavad tavapäraselt, et võrguühendus on alati saavutatav. See eeldus läbistab platvormi. HTML dokumendid laetakse üle HTTP protokolli ja traditsiooniliselt hangitakse kõik nende alamressursid järgnevate HTTP päringute abil. See seab veebisisu ebasoodsamasse olukorda võrreldes muude tehnoloogiakogustega (W3C, 2016).

Service Worker töötab kõigepealt selle tasakaalu parandamiseks, pakkudes veebitöötaja (ingl *web worker*) konteksti, mida saab käivitada alguses enne navigeerimise toimumist. See sündmustepõhine töötaja on registreeritud päritolu ja teega või mustriga. Mis tähendab, et sellega saab suhelda, kui navigeeritakse vastavale veebilehele. Sündmused, mis vastavad võrgupäringutele, saadetakse Service Worker'ile ja vastused päringutele võivad ülistada vakimisi võrgu käitumise. See paneb Service Worker'i kontseptuaalselt võrgu ja dokumendihalduri vahele, võimaldades veebisisu kuvada isegi siis, kui võrguühendus puudub (W3C, 2016).

Võrguühenduseta veebisisu kuvamine ei ole ainuke funktsioon. Notifications API[1] ja Push API[2] kasutavad Service Worker'it, et küsida serverist andmeid isegi siis, kui rakendus ei ole avatud. Mõlemad API'd on loodud Service Worker API peale, mis võimaldab saata kasutajatele tõuketeavitusi (ingl *push notifications*) (Google, 2017). Näiteks Google Calendar[3] kasutab tõuketeavitusi Service Worker'iga, saates kasutajale sõnumi isegi siis, kui Google Calender ei ole veebilehitsejas lahti.

Service Worker on hetkel veel alles töös olev projekt ja ei ole lõplikult W3C poolt jõustunud (W3C, 2016). Lisaks ei toeta Service Worker'eid mitmed veebilehitsejad ja Androidi versioonid. Android v4.4.4 ja vanemad, millel on suurim kasutajaskond, ei toeta Service Worker'eid, kuna kasutavad vanemaid veebilehitsejaid. Internet Explorer, Edge ja Safari ei toeta töö kirjtamise hetkel Service Worker'it (Gup, 2016).

[1] Notifications API - https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API
[2] Push API - https://developer.mozilla.org/en-US/docs/Web/API/Push_API
[3] Google Calendar - https://calendar.google.com

Service Worker'i tugi on veebilehitsejatel vastavalt (Deveria, 2017):

- Firefox versioon 55+,
- Chrome versioon 49+,
- Opera versioon 47+,
- Chrome for Android versioon 61+.

AppCache oli, selle välja tulles, parim idee vahemälu kasutamiseks, kuid jättis arendajad, kui ka kasutajad nõudma enamat, kui AppCache pakkuda sai. Hetkel mõlemad suurimad veebilehitsejate tootjad, Mozilla ja Google, toetavad enda veebilehitsejates Service Worker'it ja soovitavad isegi lõpetada AppCache'i kasutamist arendajatel ja minna üle Service Worker'ile. Uuemad veebilehitsejad kuvavad konsoolis ka teate, kui veebirakendus kasutab ikka veel AppCache'i ja soovitab arendajal vahetada see välja Service Worker'i vastu. Nii Mozilla kui ka Google pakuvad arendajatele suunatud veebilehtedel detailseid ja arusaadavaid õpetusi[4][5] Service Worker'i kasutamisest.

Igal kasutajal on võimalik näha veebilehitsejast, millised külastatud lehtedest kasutavad Service Worker'it. Google Chrome'il on selleks aadress chrome://serviceworker-internals/, mille kirjutamisel aadressi reale näeb kogu loetelu ja vajadusel saab kustutada Service Worker'eid. Service Worker'i kustutamisel jääb vahemälu alles ja lehele uuesti navigeerimisel kontrollitakse, kas on vaja uuendada vahemälu või mitte.

Huvi korral on võimalik ka vaadata Service Worker'i faili koodi. Google Chrome'is selleks tuleb avada arendaja tööriistad (ingl *developer tools*) vajutades F12 klahvi klaviatuuril. Uuelt ilmunud menüüribalt valida allikad (ingl *sources*) ja vasakul valikutes on näha sellist faili nagu service-worker.js või sw.js. Mõningatel rakendustel võib olla seal failis viide välisele Service Worker'i hoiustajale, kust laetakse fail rakendusse.

[4] Google Service Worker õpetus - https://developers.google.com/web/fundamentals/primers/service-workers/
[5] Mozilla Service Worker õpetus - https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Näiteks kasutatavad Service Worker'it Google Drive[6] ja teised Google rakendused. Lisaks kasutab Google ka AppCache'i veebilehitsejate jaoks, mis ei toeta veel Service Worker'it. Google kasutab enda rakendustes ka tõuketeavitusi, et kasutajatele saata teavitusi rakendustest. Lisaks saab tuua näitena 1A.EE[7], mis on Lätis tegutseva veebipoe tütarettevõte ning kasutab tõuketeavitusi, et saata kasutajatele teavitusi tellimuste, või poe toodete kohta. Photopoint[8], Eesti üks suurimaid e-poode, ei kasuta küll Service Worker'it, kuid kasutab Push API't kasutajatele teatiste saatmiseks. Lisaks Push API'le kasutab Photopoint andmete salvestamiseks veebilehitsejas IndexedDB'd.

1.3 Service Workeri kasutuselevõtt veebirakenduses

Kui kasutada Visual Studio Code-i koodiredaktoriks, siis on võimalik paigaldada redaktorist selline teek nagu PWA Tools[9]. Lisab koodiredaktorisse käsud, millega saab kiirelt ja lihtsalt genereerida Service Worker'i faile ja käsklusi.

Eesrakenduste raamistikele Angular ja React on võimalik installeerida teek sw-toolbox[10]. Sw-toolbox lihtsustab Service Worker'i kasutamist ja lisab funktsionaalsust juurde. Üks huvitavamaid lisandeid on veebirakenduse raamistiku ExpressJS stiilis andmete pärimine vahemälust ja võimalus muuta seda, kuidas igat päringut päritakse. Näiteks saab määrata, et teatud päringud küsitakse algselt võrgust ja seejärel alles vahemälust, samas mingid päringud küsitakse kohe vahemälust.

NodeJS-i teek offline-plugin[11], mis on JavaScript moodulite komplekteerija, Webpack[12] projektidele. Lihtsustab Webpack projektides Service Worker'i kasutamist.

[6] Google Drive - https://drive.google.com
[7] 1A.EE – https://www.1a.ee
[8] Photopoint – https://photopoint.ee/
[9] PWA Tools - https://marketplace.visualstudio.com/items?itemName=johnpapa.pwa-tools
[10] Sw-toolbox - https://googlechromelabs.github.io/sw-toolbox/
[11] Offline-plugin - https://github.com/NekR/offline-plugin
[12] Webpack - https://webpack.js.org/

1.4 Andmete salvestamine Service Workeri abil

Service Worker kasutab andmete salvestamiseks ja pärimiseks veebilehitseja vahemälu Cache Storage'it. Andmed salvestatakse päringutena, mille kaudu neile algselt ligi on saadud. See tähendab, et näiteks tavalisi muutujaid ei saa Service Worker'iga salvestada, kui neile ei pääse ligi läbi mingi HTTP päringu.

Rakenduse enda andmete salvestamiseks on võimalik kasutada veebilehitsejate poolt kasutusel olevaid võimalusi näiteks Local Storage, Session Storage ja IndexedDB

Erinevalt Service Worker'ist lubavad veebisirvijates olev Local Storage ja Session Storage salvestada ka muutujaid ja objekte stringi kujul veebilehitseja vahemällu. Erinevus Local Storage'i ja Session Storage'i vahel on see, et Session Storage-is on andmetel kehtivuse aeg ja andmed kustutatakse kui veebilehitseja sessioon lõppeb.

IndexedDB on aga tehingute põhiline andmebaasi süsteem, nagu SQL-põhised andmebaasid. Erinevalt SQL-põhistest andmebaasidest, mis kasutavad fikseeritud veeru tabeleid, on IndexedDB JavaScript-põhine objektorienteeritud andmebaas. IndexedDB võimaldab salvestada ja hankida võtmeid, indekseeritavaid objekte, mis on määratud andmebaasi skeemi poolt (Mozilla, 2017).

2 Näidisrakenduse arendamine

Näidisrakenduse arendamise eesmärk on tutvustada Service Worker'i ülesehitust ja kasutamist. Rakendus ehitatakse üles nii, et seda oleks võimalik kasutada ka ilma võrguühenduseta ja kasutada Service Worker'i funktsionaalsust.

Arendatav rakendus kuvab kasutajale Eesti linna ilmaprognoosi ja kasutajal on võimalik valida, mis linna ilma ta soovib näha. Kõige tähtsam rakenduse nõue on HTTPS protokolli tugi, sest Service Worker'it ei saa kasutada, kui ei ole HTTPS tuge veebilehele.

Rakenduse arendamiseks kasutab autor tehnoloogiaid:

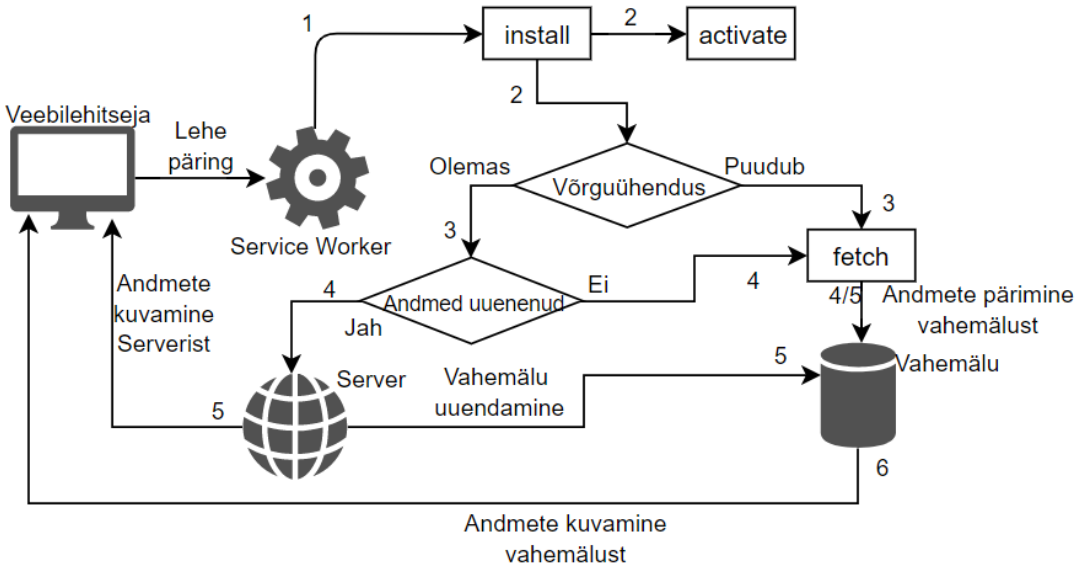
- Nodejs – veebiserver, mille peale näidisrakendus on ülesehitatud. Nodejs on asünkroonne sündmustepõhine JavaScripti runtime, mille abil saab luua veebiservereid (Node.js Foundation, 2017). See sai valitud kuna võimaldab lihtsalt ja kiirelt üles seada veebiserverit, mida saab enda vajadustele konfigureerida erinevate teekidega.
- ExpressJS – Nodejs veebirakenduse raamistik, mis võimaldab lahutada serveri poole kliendi poolest ja lisada juurde funktsionaalsust Nodejs'ile (StrongLoop, 2017). Andmetega, millega rakendus tegeleb on JSON formaadis ja selle tõttu sai valitud serveri pooleks ExpressJS, et saaks lihtsalt JavaScriptiga andmeid vajadusel töödelda.
- AngularJS – Kliendi poolne rakendus, mis lisab HTML dokumentidele dünaamilisi funktsionaalsusi. Näidisrakenduses on kasutusel AngularJS versioon 1.6.6 (Google, 2017). NodeJS'i kasutades oli võmalik ehitada nii serveri pool, kui ka kliendi pool ülesse kasutades JavaScripti ja tänu sellele sai valitud kliendi poolele AngularJS. Kuigi on väljas Angular'i uuemad versioonid, siis selleks, et hoida põhirõhk Service Worker'il valiti vanem Angular'i versioon, mis on vähem modulaarsem ja millele on lihtsam väikest rakendust üles ehitada.

Koodiredaktoriks kasutati Visual Studio Code-i[13]. Lisa teeke mida sai kasutatud, et arendus lihtsamaks teha, olid Angular ES6 snips[14]ja npm[15]. Angular ES6 snips võimaldas lihtsamalt ja kiiremalt luua Angulari funktsioone. JavaScripti paketihaldussüsteemi npm'i abil sai koodi redaktorist käivitada ja testida veebiserverit.

Koodi talletamiseks ja versiooni halduseks kasutati Githubi[16]. Näidisrakenduse kood on olemas avalikult repositooriumis aadressil https://github.com/AndreTLU/seminaritoo. Visual Studio Code-is on sisseehitatud Githubi tugi, mis võimaldab otse koodiredaktorist koodi salvestada ning lähetada Githubi.

2.1 Rakenduse ülesehitus ja toimimine

Järgnevalt tutvustatakse näidisrakendust, mis sai seminaritöö käigus arendatud. Rakenduse ülesehitust iseloomustab Joonis 1.



Joonis 1. Rakenduse joonis

Veebilehitsejas navigeerimisel rakenduse lehele tehakse lehe päring Service Worker'ile. Edasi käivitatakse Service Worker'i funktsiooni install, mis loeb vastavad andmed vahemällu. Installeerimisele järgneb käsk activate ja samal ajal kontrollitakse,

[13] Visual Studio Code - https://code.visualstudio.com/
[14] Angular ES6 snips - https://marketplace.visualstudio.com/items?itemName=kasperkeso.es6-angular-snips
[15] Npm - https://marketplace.visualstudio.com/items?itemName=eg2.vscode-npm-script
[16] Github - https://github.com/

kas võrguühendus on olemas ,või mitte. Võrguühenduse puudumisel käivitatakse käsk fetch ja päritakse andmed vahemälust. Võrguühenduse olemasolul kontrollib Service Worker, kas andmeid on vaja uuendada, või mitte. Kui andmeid ei ole vaja uuendada, käivitatakse käsk fetch ,mis pärib andmed vahemälust. Vajadusel, kui andmed on uuenenud laetakse andmed serverist vahemällu ja kuvatakse andmed veebilehitsejale. Kasutajal on võimalik valida Eesti linn, mille ilmaprognoosi ta soovib näha. Kasutaja valik salvestatakse Service Worker'i abil vahemällu ja järgmisel külastusel kuvatakse kasutajale tema viimasena valitud linna andmed vahemälust. Veebirakendust on võimalik külastada ka ilma võrguühenduseta ja siis kuvatakse ka vahemälus oleva viimase valitud linna ilma andmed.

Service Worker töötab tagaplaanil veebilehitsejas. Service Worker'i kasutamiseks veebilehel tuleb see esmalt registreerida. Selleks on vaja, et veebilehitseja tuvastab Service Worker'i registreerimise käsu ja käivitab selle tagaplaanil (vt Koodinäide 1).

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function () {
    navigator.serviceWorker.register('sw.js')
      .then(function (registration) {
        console.log(
          'ServiceWorker registration successful',
          registration.scope
        );
      }, function (err) {
        console.log('ServiceWorker registration failed: ', err);
      });
  });
}
```

Koodinäide 1. Service Worker'i registreerimine

Üleval toodud kood kontrollib kas Service Worker API on veebilehitseja poolt olemas, või mitte ja kui on olemas, siis Service Worker failis sw.js registreeritakse.

Service Worker'i fail koosneb kolmest osast, milleks on install, activate ja fetch. Koodinäide 2 olev käsk install käivitatakse kohe peale Service Worker'i registreerimist ja alustab Service Worker'i installeerimist. Avatakse või luuakse uus vastav kirje vahemälus ja lisatakse andmed sinna (Gaunt, 2017). Antud rakenduse puhul on seal välja toodud kõik 7 faili, mida rakendus kasutab. Järjestus on oluline.

```
var cacheName = 'Seminaritoo-cache-v1';
var dataCacheName = 'data-v1';
var urlsToCache = [
  '/js/home/home.js',
  '/js/home/home.html’,
  '/js/app.js',
  '/js/sw.js',
  '/index.html',
  '/’
];
self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open(cacheName)
      .then(function(cache) {
        return cache.addAll(urlsToCache);
      })
  );
});
```

Koodinäide 2. Service Worker'i installeerimine

Kui kõik failid on edukalt laetud vahemällu, siis Service Worker on installeeritud. Kui vähemalt ühte faili ei saa laadida vahemällu, siis terve Service Worker’i installeerimine peatatakse (Gaunt, 2017).

Kui failid on edukalt salvestatud vahemällu ja klient, kas uuendab lehte või navigeerib teisele lehele, saadetakse veebilehitseja poolt Service Worker’ile fetch käsk (vt Koodinäide 3).

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

Koodinäide 3. Fetch käsk andmete saamiseks

Fetch käsuga avatakse kõik vahemälud ja kontrollitakse, kas on vahemälus Service Worker’ile teada olevad failid päringutena või mitte. Kui on vahemälus vastavad

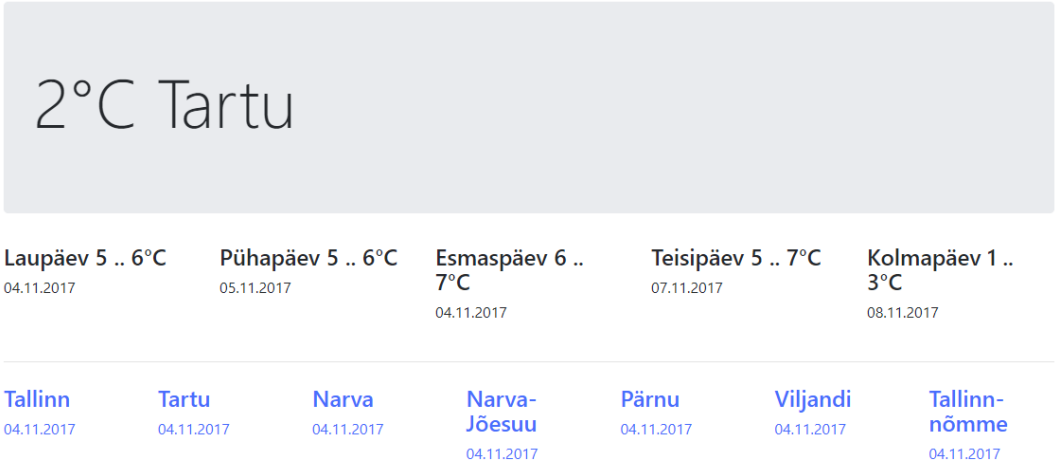
päringud olemas, laetakse need vahemälust. Päringu puudumisel vahemälus tehakse päring failile võrku.

Viimaseks Service Worker peab uuendama vahemälu, kui on vajadust. Veebilehitseja proovib navigeerimisel uuesti laadida faile ja kontrollib, kas on erinevusi, kui on isegi 1 baidine erinevus loetakse fail uueks ja tehakse uus Service Worker. Service Worker'i installeerimine käivitatakse uuesti ja vana Service Worker lülitatakse välja. Uue Service Worker'i tööle minekul käivitatakse käsk activate (Gaunt, 2017) (vt Koodinäide 4).

```
self.addEventListener('activate', function (event) {
   console.log('Activating Service Worker ' + cacheName);
   event.waitUntil(
      caches.keys().then(function (keyList) {
        return Promise.all(keyList.map(function (key) {
          if (key != cacheName && key !== dataCacheName) {
            return caches.delete(key);
          }
        }))
      })
   ); return self.clients.claim();
});
```

Koodinäide 4. Activate käsk andmete uuendamiseks

Tutvustades rakenduse serveri poolt, saab välja tuua olulisemana ilma andmete pärimise ja edastamine kliendile. Ilm.ee-st andmete kättesaamine toimub veebirakenduse raamistiku ExpressJS'i abil ilm.js failis. Ilma andmed näidatakse välja ühe nädala päevade lõikes (vt Joonis 2)



Joonis 2. Ekraanitõmmis rakendusest

Serveri poolele saadetakse eesrakendusest päring koos muutujaga city, mis on linna number. Esimesena tehakse päring ühele Ilm.ee lehele, kust saadakse muutuja abil teada linna ilmaprognoos ja linna nimi. Linna nime kasutatakse järgmises päringus, et saada teada linnas oleva tuule kiirus ja sademete hulk, mis on eraldi andmestikus. Andmed pannakse kokku ja edastatakse kliendile.

2.2 Arendusprotsessi kokkuvõte

Arendusprotsessi käigus õnnestus autoril luua veebirakendus, mis pärib ilmaprognoosi andmed eraldiseisvast serverirakendusest. Rakendus kuvab kasutajale andmed ja laseb kasutajal valida, millise linna andmeid ta näha soovib. Kasutaja valik salvestatakse, et kasutaja saaks lihtsalt ja mugavalt uuesti lehele navigeerimisel näha andmeid. Lisaks kõigele sellele on kasutajal võimalik rakendust kasutada ka peale esimest külastust võrguühenduseta. Andmed, mis sai päritud Ilm.ee rakendusest, tulid kahest erinevast kohast ja liideti tervikuks kokku enne kliendile edastamist. Service Worker salvestab andmed vahemällu. Võrguühenduse puudumisel pärib andmed vahemälust ja edastab need kasutajale.

HTTPSiga Nodejs serveri majutamiseks kasutas autor Microsofti Azure[17] keskkonda, mis on üliõpilastele õpingute ajal tasuta. Võimalik on kasutada ka Heroku't[18], mis lubab tasuta jooksutada ühte Nodejs'i serverit.

Vahemällu salvestatavate failide asukohtadega oli arenduse käigus probleeme. Kuna Service Worker otsib ette antud faile kliendi poolse rakenduse kaustast, siis Service Worker ei saanud korralikult salvestada ette antud faile. Eesrakendus oli eraldi kaustas public, mis on Angular-iga ülesehitatud ja autoril tuli vastavalt muuta failide asukohad Service Worker'i failis, et neid kätte saada public kaustast.

Service Worker'it implementeerida rakendustesse on üpris kerge. Rakenduses tuleks luua vastav Service Worker'i fail, kus saab määrata milliseid faile vahemällu salvestada. Aluseks võib võtta näidisrakenduse sw.js faili. Kui on soov küsida andmeid

[17] Azure - https://azure.microsoft.com/en-us/
[18] Heroku - https://www.heroku.com/

ka mõnest muust rakendusest, nagu näidisrakenduses küsitakse andmeid serverist, siis tuleks ka selle päringu aadress lisada faili.

Kokkuvõte

Seminaritöö eesmärgiks oli anda ülevaade Service Worker'ist ja selle eelkäijast AppCache'ist koos kasutuse tutvustusega näidisrakenduse näitel.

Töö käigus tutvustati Service Worker'i eelkäijat AppCache'i, selle probleeme ja selle kasutuselevõtuga arendajatele tekkinud raskusi. Näiteks oli probleemiks see, et serveris olevate failide uuendused ei jõudnud kasutajani ja ta pidi uuenduste saamiseks veebilehitsejas lehte uuendama. Service Worker lahendab eelnimetatud probleemid ja pakub võimaluse kuvada veebisisu võrguühenduseta. Töötades veebisirvijas taustal haldab ta failide ja andmete päringuid ning võrguühenduse puudumisel laeb need vahemälust. Eestis kasutatakse Service Worker'it e-poodides tõuketeatiste saatmiseks klientidele.

Autor arendas seminaritöö käigus näidisrakenduse, et ise põhjalikumalt tutvuda ning anda ka teistele ülevaade Service Worker'i kasutamisest ja sellest, kuidas see tagaplaanil andmetega töötab. Loodud näidisrakendus kuvab Eesti linnade ilmaprognoosi ning kasutajal on võimalik valida mis linna andmeid ta soovib näha. Service Worker salvestab enim kasutatavad failid vahemällu. Ilmaprognoosi andmed päritakse serverist ning lisatakse ka vahemällu. Järgneval külastusel laeb Service Worker vahemälust nii failid kui ka andmed. Failidest loob veebilehitseja eesrakenduse liidese kasutajale ja kuvab selles saadud andmed. Võrguühenduse puhul peale päringut vahemällu kontrollitakse ka serverist failide uudsust. Vajadusel faile vahemälus uuendatakse. Võrguühenduse puudumisel kontrolli serveris ei tehta.

Arenduse käigus välja tulnud raskuseks oli failide asukohtade saamine JavaScriptis ning nende asukohtade kasutamine andmete salvestamisel ei ole ühtne. Eeltöö ning näidisrakenduse loomine andis autorile oskuse kasutada Service Worker'it oma järgmise rakenduse arendamisel.

Service Worker on hetkel alles arendamise faasis ja veebisirvijate tugi on osaline. Tulevikus võib näha veel rohkem võimalusi selle kasutamiseks, mitte ainult võrguühenduseta rakendustes vahemälu haldamiseks ning veebilehitsejatele tõuketeavituste saatmiseks.

Kasutatud kirjandus

Bloom, J. D. (24. juuni 2012. a.). *Problems with Application Cache*. Allikas: James D Bloom - Blog: http://blog.jamesdbloom.com/ProblemsWithApplicationCache.html

Deveria, A. (21. oktoober 2017. a.). *Service Workers*. Allikas: Can I use: http://caniuse.com/#feat=serviceworkers

Gaunt, M. (26. september 2017. a.). *Service Workers: an Introduction*. Allikas: Web Fundamentals | Google Developers: https://developers.google.com/web/fundamentals/primers/service-workers/

Google. (23. oktoober 2017. a.). *AngularJS*. Allikas: AngularJS: https://angularjs.org/

Google. (11. oktoober 2017. a.). *Introduction to Push Notifications*. Allikas: Progessive Web Apps Training: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications

Gup, A. (2. veebruar 2016. a.). *Application Cache is not gone, oh my! Or, is it?* Allikas: The Page Not Found Blog: http://www.andygup.net/application-cache-is-not-gone-oh-my-or-is-it/

Kammardi, P. K. (11. august 2017. a.). *How the Web works*. Allikas: Learn web developmend | MDN: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works

Microsoft. (kuupäev puudub). *Visual Studio Code*. Allikas: Visual Studio Code: https://code.visualstudio.com/

Mozilla. (9. oktoober 2017. a.). *IndexedDB API*. Allikas: MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

Mozilla. (18. oktoober 2017. a.). *Using Service Workers*. Allikas: MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

Mozilla. (7. august 2017. a.). *Using the application cache*. Allikas: MDN web docs: https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache

Mozilla. (11. september 2017. a.). *Window.localStorage*. Allikas: MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Node.js Foundation. (23. oktoober 2017. a.). *node.js*. Allikas: About Node.js: https://nodejs.org/en/about/

Oppenheimer, T. (8. mai 2017. a.). *Read web pages offline with Chrome on Android*. Allikas: The Keyword: https://blog.google/products/chrome/read-web-pages-offline-chrome-android/

StrongLoop. (23. oktoober 2017. a.). *Express*. Allikas: Express: https://expressjs.com/

Zajdband, D. (12. september 2016. a.). *Enabling Offline First Experiences on the Web with Service Workers*. Allikas: Medium: https://medium.com/offline-camp/enabling-offline-first-experiences-on-the-web-with-service-workers-e4bc8c773dae

W3C. (11. oktoober 2016. a.). *Service Workes 1*. Allikas: W3C Working Draft: https://www.w3.org/TR/service-workers-1