Saaremaa Ühisgümnaasium

ERALDISEISVA TUNNIPLAANI RAKENDUSE TEGEMINE

Uurimistöö

Autor: Oliver Paljak 12C

Juhendaja: Henrik Pihl

Kuressaare 2020

ANNOTATSIOON

Saaremaa Ühisgümnaasium

Töö pealkiri: Eraldiseisva tunniplaani rakenduse tegemine					
Aasta: 2020	Lehekülgede arv				
	põhitekst	tabelid	joonised	allikad	lisad
	33	0	12	58	1

Referaat

Käesoleva uurimistöö probleemiks kujunes see fakt, et SÜG-il ei olnud olemas kaasaegset tunniplaani vaatamise programmi. Seega töö põhieesmärgiks seati SÜG-ile modernse igapäeva kasutust võimaldava eraldiseisva tunniplaani rakenduse loomise. Lisaeesmärgiks seati avardada autori teadmisi olemasolevatest veebi- ja arendustehnoloogiatest.

Töös antakse ülevaade äpis kasutatud tehnoloogiatest (nii tava- kui ka arendamistehnoloogiad), disainiprintsiipidest, äpi töökorraldusest, versioonidest ja arendamisprotsessist, rakenduse privaatsuspoliitikast ja lõpptulemustest.

Vaatamata sellele, et äpi kasutamisel avastati mõningaid vigu, parandati need ning täiustati äppi vastavalt.

Uurimistöö käigus valmis modernne eraldiseisev tunniplaani rakendus, mida saab igapäevaselt koolielus kasutada. Lisaks sellele, omandas autor veebiarenduse kohta palju uusi teadmisi, mida ta saab kasutada järgmistes projektides ja töödes.

Võtmesõnad: rakenduse tegemine, äpi tegemine, veebiarendus, programmeerimine

Töö autor: Oliver Paljak allkiri:

Kaitsmisele lubatud:
Juhendaja: Henrik Pihl allkiri:

SISUKORD

LÜHENDID

AJAX – Asynchronous JavaScript And XML
AppCache – Application Cache
CSS – Cascading Style Sheets
DOM – Document Object Model
Haml – HTML abstraction markup language
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
npm – Node Package Manager
PHP – Hypertext Preprocessor
PWA – Progressive Web App
Sass – Syntactically Awesome Stylesheets
SVG – Scalable Vector Graphics
SQL – Structured Query Language
SÜG – Saaremaa Ühisgümnaasium
W3C – World Wide Web Consortium
XML – Extensible Markup Language

SISSEJUHATUS

Käesoleva uurimisöö teema valik tuli sellest, et SÜG-il oli olemas küll tunniplaani vaatamise võimalus, kuid see ei olnud eriti kasutajasõbralik arvestades tänapäeva nutivahendeid. Lisaks sellele, kasutas vana tunniplaani süsteem andmete saamiseks otsepäringuid andmebaasist, mis on serverile ressursimahukad. Seega sai uurimistöö eesmärgiks luua SÜG-ile modernne igapäeva kasutust võimaldav eraldiseisev tunniplaani rakendus. Lisaeesmärgiks seati avardada autori teadmisi olemasolevatest veebi- ja arendustehnoloogiatest.

Töö on ülesehitatud kindla loogikaga. Esimeses peatükis „Kasutatud tehnoloogiad" tuuakse välja need tehnoloogiad, mis moodustavad rakenduse enda. Teises peatükis „Kasutatud arendamistehnoloogiad" tuuakse välja need tehnoloogiad, mida kasutati äpi arendamiseks. Kolmandas peatükis „Disainiprintsiibid" antakse ülevaade nendest põhimõtetest, mida kasutati rakenduse arendamisel. Neljandas peatükis „Äpi töökorraldus" räägitakse üleüldiselt, kuidas äpp töötab ja mis turvameetmed on kasutusel. Viiendas peatükis „Versioonidest ja arendamisprotsessist" antakse ülevaade, mis probleemid ja kitsaskohad viisid järgmiste uuendusteni ning milline arendamisprotsess ise oli. Kuuendas peatükis „Privaatsusest" tuuakse välja, miks inimeste kohta internetis infot kogutakse ja kuidas loodud äpp antud valdkonnas käitub. Töö tulemused näitavad, milline rakendus valmis ja kui kaua võib ühe analoogilise äpi tegemiseks aega minna. Lisaks on töö alguses välja toodud töös kasutatud lühendid ja mõisted, mis peaksid aitama lugejal tööst paremini aru saada.

Peamisteks töös kasutatud allikateks olid kas allikad, mis olid tehnoloogiate enda veebilehtedel või mida mingi tehnoloogia autor(id) ise ametlikult soovitasid (näiteks tehnoloogia Git üheks allikaks on raamat „Pro Git", mida Giti arendajad soovitasid lugeda nendel, kes antud versioonihaldajat kasutavad) või veebiarenduses üldtuntud ja üldtunnustatud allikaid nagu näiteks Mozilla Developer Network. Kuna seletatud tehnoloogiaid on töös palju, siis otseselt ei ole olemas ühtegi põhiallikat, kuid arvatavasti on kõige rohkem viidatud erinevate tehnoloogiate erinevate Github lehekülgede peale (tegemist on veebilehega, kus avaldatakse enamus avatud lähtekoodiga tarkvarast) ja Mozilla Developer Network'i.

1. KASUTATUD TEHNOLOOGIAD

1.1. HTML

Selleks, et veebi jagada informatsiooni on vaja tehiskeelt, mis on arusaadav kõikidele veebi kasutatavatele arvutitele. HTML antud ülesannet täidabki. HTML lubab arendajal veebilehte kirja panna (märkida), mis osad tekstist on pealkirjad, lõigud, nimekirjad jne. Lisaks sellele, lubab HTML märkida ka tabelid, pilte ja muid vajalikke andmeid. (Raggett, 1997) HTML ei ole programmeerimiskeel, vaid märgistuskeel (*markup language*; mõisteid vt Lisa 1), mis paneb paika veebilehe põhilise struktuuri (Robbins, 2012).

1.2. PHP

PHP on laialdaselt kasutatud avatud lähtekoodiga üldotstarbeline skriptikeel (*scripting language*), mis on spetsiaalselt kohandatud veebi arenduseks – HTML on PHP skripti sees. See tähendab, et erinevatelt teistest klassikalisematest programmeerimiskeeltest (nagu C või Perl), mis kasutavad keerulisi funktsioone ja käske, et HTML-i väljastada, on HTML osa PHP skriptist, kus esinevad PHP koodiblokid – PHP osad algavad tähisega „<?php" ja lõpevad „?>" (joonis 1). PHP erineb ükskõik, mis kliendipoolsest skriptikeelest (näiteks tavalisest JavaScriptist) selle poolest, et PHP on ainult serveri poolel, mis tähendab, et klient näeb ainult skripti väljastatud tulemust, kuid ei näe skripti ennast (näiteks JavaScripti koodi ja ka selle tulemust on kliendil ise võimalik inspekteerida). (What is PHP, 2020)

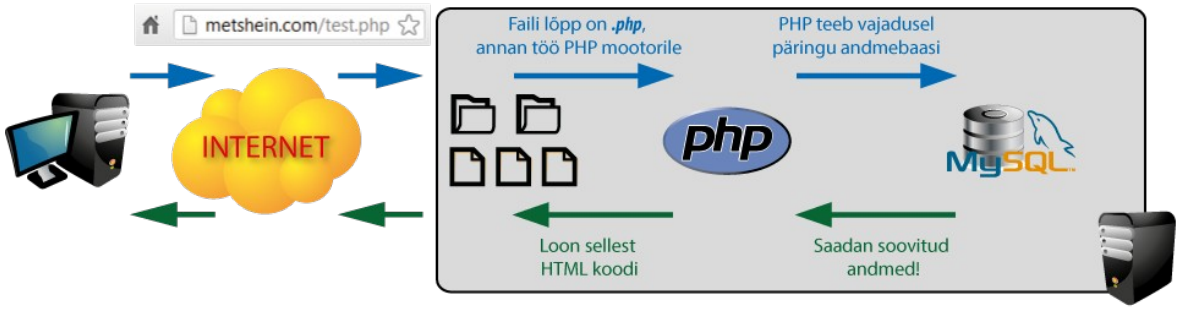
```
<section class="nav-section">
    <nav class="main-nav">
        <a href="<?php echo LOGO_LINK; ?>">
            <img class="syg-logo" alt="SÜGi logo" src="web/img/syglogo-header-min.jpg" />
        </a>

        <ul class="timetable-selects">
            <li><?php $rooms->getSelectHtml(); ?></li>
            <li><?php $classes->getSelectHtml(); ?></li>
            <li><?php $teachers->getSelectHtml(); ?></li>
        </ul>
    </nav>
</section>
```

Joonis 1. PHP koos HTML-iga (Autori foto)

PHP-d saab kasutada kolmel alal: serveripoolne skriptimine, käsurea skriptimine (*command line scripting*) ja tavaliste arvuti rakenduste tegemine. Lisaks sellele, saab PHP suhelda paljude andmebaasidega (näiteks Mongo, MySQL, PostgreSQL, SQLite, dBase jne) ja on ka olemas toetus paljudele tekstitöötlus funktsioonidele ja sortimisalgoritmidele. (What can, 2020) Tunniplaani äpis kasutati just PHP serveripoolset skriptimist, mis tähendab, et PHP väljastab HTML-i, mis on vastavuses lähtefailidega; kontrollib ja uuendab puhverdatud (*cache*) andmeid ning töötleb ka lähteandmeid, et neid oleks mugavam kasutada.

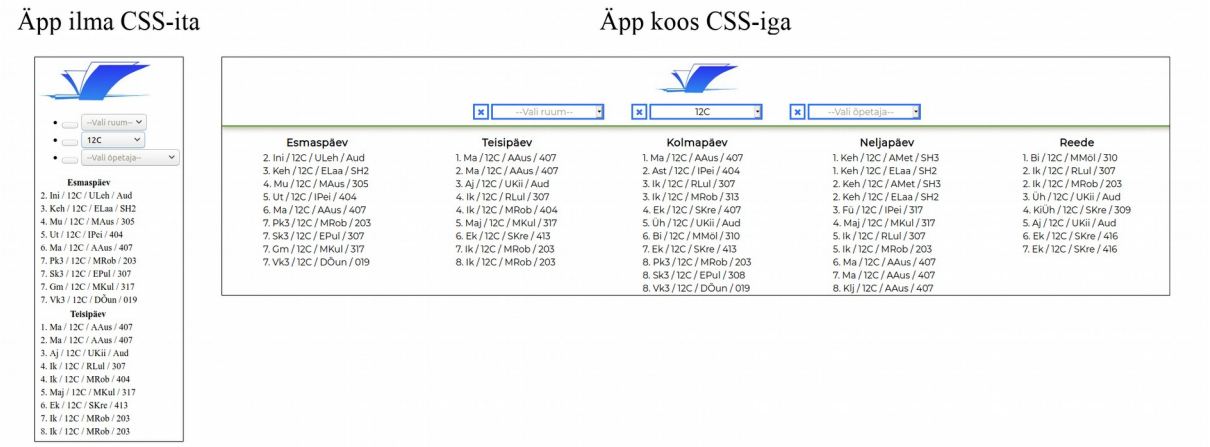
PHP veebiserveris töötab järgmiselt: kui kasutaja nõuab serverilt mingit PHP faili (näiteks test.php), siis veebiserver suunab selle PHP mootorile. See omakorda teeb järledused PHP koodist ja suhtleb vajadusel andmebaasidega. Lõpptulemusena luuakse HTML kood ja see saadetakse tagasi kasutaja veebibrauserisse (joonis 2). (Metshein, 2020)



Joonis 2. PHP töökorraldus (Metshein, 2020)

1.3. CSS

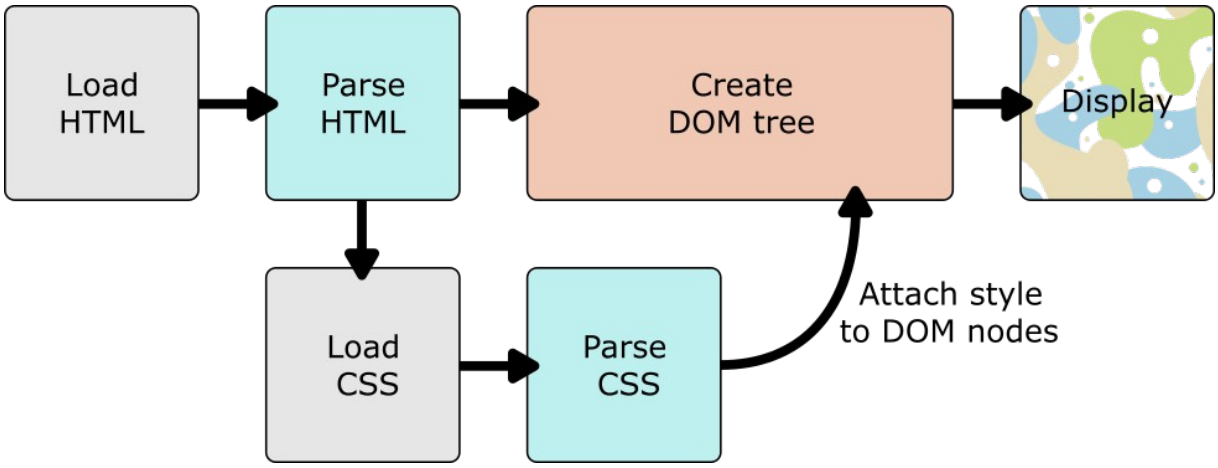
CSS on tehiskeel, mis paneb paika, kuidas visuaalselt kuvada veebilehte – missugused on veebilehe värvid, paigutus, fondid jne (joonis 3). Lisaks sellele, aitab CSS ka kuvada veebilehti korrektselt erinevate ekraani suurustega seadmetel ning on oluline komponent veebilehe kujunduse määramisel välja printimisel. CSS on HTML-ist eraldiseisev ja seda saab kasutada ka teiste märkekeelte kuvamise kirjeldamiseks. Eraldus HTML-ist muudab veebilehtede haldamise lihtsamaks ja lubab kaasata mitu laadilehte – tihtipeale nimetatakse seda meetodit „struktuuri ja sisu eraldamist kuvamise viisist”. (HTML & CSS, 2020)



Joonis 3. Äpp koos ja ilma CSS-ita (Autori foto)

CSS-i tööpõhimõtet võib laias laastus kirjeldada järgmiste sammude abil (joonis 4):

1. Brauser laeb alla HTML-i.
2. HTML muudetakse ümber dokumendiobjektide mudeliks (DOM). Tegemist on põhimõtteliselt alla laetud HTML-iga, aga rohkem struktureeritud kujul, et seda oleks lihtsam teistel veebikeeltel muuta ja ka kujundada.
3. Nüüd laeb brauser alla teised HTML-is märgitud ressursid (näiteks pildid, videod, aga ka CSS-i laadilehed).
4. Brauser sõelub (*parses*) saadud CSS-i ja sordib selektorid tüübi järgi ning lisab need vastavatele sõlmedele (*nodes*) dokumendiobjektide mudelis (antud sammu nimetatakse ka visualisatsiooni puu (*render tree*) loomiseks).
5. Visualisatsiooni puu järjestatakse selles järjekorras, et esimesena saaksid endale laadi need HTML-i elemendid, mida kasutaja näeb kõigepealt.
6. Lõpuks visualisatsiooni puu realiseeritakse ja veebileht kuvatakse kasutajale koos laadiga. (How CSS, 2020)



Joonis 4. CSS-i töökorraldus (How CSS, 2020)

1.4. JavaScript

JavaScript on mitmeplatvormiline (*cross-platform*) objektorienteeritud (*object-oriented*) skriptikeel, mida peamiselt kasutatakse veebilehtedele interaktiivsuse lisamiseks ning kasutajakogemuse tõstmiseks (näiteks animatsioonid, klikitavad nupud, hüpik menüüd jne) – seda nimetatakse ka kliendipoolseks JavaScriptiks. Lisaks sellele, kasutatakse JavaScripti ka serveri poolel ning ka arendamisprotsesside lihtsustamiseks (näiteks NodeJS). (Introduction, 2020) Tunniplaani äpis kasutatati just kliendipoolset JavaScripti.

Kliendipoolse JavaScripti tööpõhimõtteks on muuta DOM-i ehk teha muudatusi veebilehes koheselt ilma, et seda peaks värskendama – näiteks JavaScript saab lisada dünaamiliselt (*dynamically*) elemente ning reageerida kasutaja sisenditele (*inputs*). Lisaks sellele, suudab kliendipoolne JavaScript ka kontrollida teatud määral brauseri käitumist. Serveripoolne JavaScript aga täidab põhimõtteliselt samu ülesandeid, mida tunniplaani äpis täidab PHP – suhtleb andmebaasidega, töötleb lähteandmeid jne. (Introduction, 2020)

1.4.1. jQuery

jQuery on kiire, mahult väike ja võimaluste rohke JavaScripti teek (*library*). See muudab HTML-i (DOM-i) muutmise, sündmusetöötlemise (*event handling*), animatsioonid, AJAX-i ja muid tavalisi JavaScripti ülesandeid palju lihtsamaks arendajale. jQuery moto on „*write less, do more*”. (jQuery, 2020)

1.4.2. Velocity.js

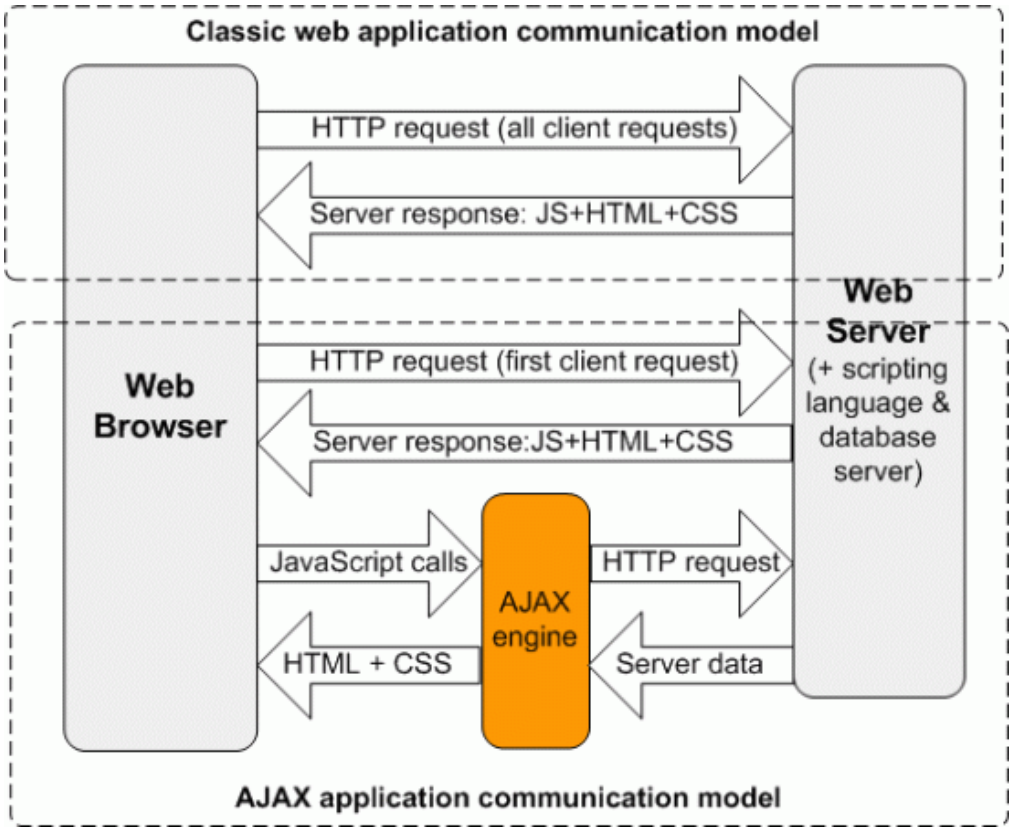
Velocity.js on animatsiooni mootor, mis sisaldab endas järgmisi võimalusi: animeerida veebilehel kerimist, värve, SVG pilte ja elementide teisendusi (*transforms*); korrata (*loop*) ja ajastada (*easing*) animatsioone. Velocity.js-i animatsiooni mootor on palju kiirem ja võimaluste rohkem jQuery animatsiooni mootorist, kuna Velocity.js kasutab erinevalt jQueryst JavaScripti oma (*native*) animatsioone, mis lubavad palju suuremat optimeerimist ja jõudlust. Lisaks sellele, Velocity.js suudab töötada koos jQueryga, mis lubab arendajal kasutada peaaegu sama kirjutamisviisi, mis jQuery animatsioonide kirjutamisel. (Shapiro, 2014a)

1.5. AJAX

AJAX ise ei ole tegelikult eraldiseisev tehnoloogia, vaid hoopis mudel, mis kasutab olemasolevaid tehnoloogiaid nagu HTML, CSS, JavaScript, DOM ja teisi tehnoloogiaid üheskoos, et oleks võimalik veebilehele teha uuendusi ilma tervet veebilehte uuesti laadimata – see muudab veebi palju kiiremaks ja kasutajasõbralikumaks. (Ajax, 2020)

Traditsiooniliselt ehk ilma AJAX-ita tehakse veebiserverile HTTP päring (tegemist on protokolliga, mida kasutatakse HTML-i ja ka teiste ressursside saatmiseks serverist kliendini). Näiteks kui kasutaja läheb veebilehele, siis server saadab kliendini terve lehekülje koos ressurssidega ning kui kasutaja täidab veebilehel vormi ja saadab selle tagasi serverile, siis peab klient laadima uuesti terve veebilehe, et näidata serverist tagasi saadud vastust. Kui aga kasutada AJAX-it, siis laetakse terve veebileht ainult ühe korra – edasi uuendatakse ainult vajalikke veebilehe osasid. Kogu suhtlust serveri ja kliendi vahel kordineerib AJAXi mootor, mis ei ole midagi muud, kui JavaScripti kood, mis on võimeline suhtlema serveriga samaaegselt – näiteks läbi XMLHttpRequest objekti (joonis 5). Tulles tagasi esialgse näite juurde, siis kui kasutaja läheb veebilehele, siis server ikka saadab kliendini terve lehekülje koos ressurssidega, kuid nüüd on nende ressursside hulgas ka eelnimetatud AJAXi mootor. Kui kasutaja täidab ära veebilehel oleva vormi, siis saadetakse vormi vastused AJAXi mootorisse, mis omakorda pärib asünkroonselt veebi serverilt vastust ning pärast vastuse saamist saadab selle tagasi kliendini. See kõik tähendab seda, et uuendama ei pea tervet veebilehte, vaid ainult osa sellest. Lisaks sellele, saab teha kõiki AJAX-i päringuid

tagataustal, ilma et kasutaja ei pruugigi aru saada, et veebileht suhtleb samaaegselt serveriga. (How does, 2020)



Joonis 5. AJAX-i töökorraldus(How does, 2020)

Äpis kasutati AJAX-it läbi jQuery funktsiooni jQuery.ajax(). Tihtipeale kasutatakse koodis sõna „jQuery” asemel märki „$” (joonis 6).

```
/**
 * Check if user is online
 *
 * @param {function} resolve Callback function for resolved/successful state
 * @param {function} reject Callback function for rejected/error state
 */
function isUserOnline(resolve, reject){
    return $.ajax({
        method: "POST",
        url: "connection-check.json",
        success: function() { resolve(); },
        error: function() { reject(); }
    });
}
```

Joonis 6. AJAX äpis (Autori foto)

1.6. Puhverdamine

Puhverdamine (*cache*) on arvutiteaduses kasutatav võte, millega salvestatakse andmed kuhugile, kust neid saab kiiresti vajadusel kätte. Puhverdamise võimalusi ja võtteid on palju, kuid veebimaailmas üks lihtsaim moodus on veebiressursside (HTML, CSS, JavaScript jne) ja serverist saadud andmete salvestamine kliendipoolele. (What is Caching, 2020) Äpis kasutati puhverdamist tunniplaani salvestamisega kliendi poolele, millega vähendati kasutaja andmeside mahtusid ja serveri koormust.

1.7. Service Workers

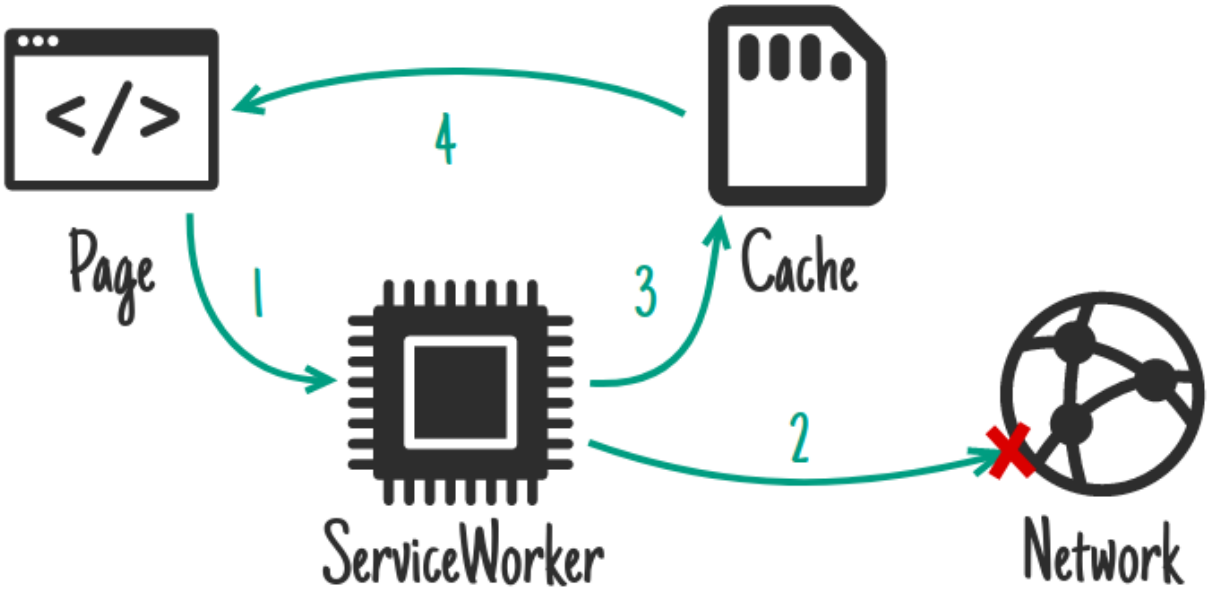
Kõigepealt, et saaks ära defineerida, mis on Service Worker, tuleks ära defineerida, mis on üleüldse JavaScriptis Worker.

Üheks suurimaks probleemiks on veebis olnud veebilehtede jõudlus ning seda eriti JavaScripti kohapealt, sest JavaScript on just see keel, mis muudab veebilehed interaktiivseks. Probleemi tuumaks aga on, et veebilehe skript on brauseris alati ühes harus (*single-threaded*), mis tähendab seda, et kõiki skripti käsklusi täidab brauser korraga – sündmusetöötlemine, kasutajaliidese muutmine (näiteks animatsioonid), AJAX-i päringud, arvutused, andmetöötlus jne. Kirjeldatud ülesanded võtavad palju resursse, aga siiski on need modernses veebilehes täielikult olemas. Siin mängivadki rolli Workerid. (Bidelman, 2010) Nimelt Workerite haru

(*thread*) on teises harus, mis tähendab, et Worker ei sega JavaScripti sündmusetöötlemisel ja kasutajaliidese muutmisel. Lisaks sellele, saab Workeritele delegeerida AJAX-i päringud, arvutused ning andmetöötluse. See kõik tähendab, et veebileht muutub palju kiiremaks. (Using Web, 2020)

Service Workerid on aga teistmoodi Workerid. Nad ikka kasutavad Workerite haru, aga nende ülesandeks ei ole enam vähendada brauseri töökoormust, vaid käituda kui proksi (*proxy*) brauseri ja serveri vahel – see tähendab, et Service Worker suudab muuta veebilehe kättesaadavaks ka siis, kui kasutaja on interneti ühenduseta (võib ka öelda, et puhverdab veebilehe); suudab vastu võtta päringu ja vastata sellele; suudab uuendada puhverdatud andmeid. Lisaks selle, on võimalik kasutada tõuketeateid (*push notifications*), tagataustal andmete sünkroniseerimist (*synchronization*) ja palju muud. (Service, 2020)

Kõige lihtsam oleks arvatavasti seletada Service Workereid läbi näite. Kuna eelpool sai mainitud, et kasutusvõimalusi Service Workeritel on palju, siis on mõistlik keskenduda ühele võimalusele – näiteks korduv kasutaja tuleb veebilehele, kuid tal puudub interneti ühendus. Kuna tegemist on kasutajaga, kes on veebilehel juba käinud, siis on tal Service Workerid juba olemas ja veebileht on ka puhverdatud. Seega, kui klient teeb päringu serverile, siis lastakse see läbi Service Workeri. Service Worker teeb omakorda päringu serverile, kuid ei saa ressursse kätte, kuna puudub interneti ühendus. Nüüd võtab Service Worker puhverdatud ressursid ja tagastab need kliendile – kasutaja näeb veebilehte ka siis, kui tal interneti ühendust ei ole (joonis 7). (Archibald, 2014b)



Joonis 7. Näide Service Workeritest (Archibald, 2014b)

Äpis kasutati Service Workereid ka selleks, et äpp oleks kätte saadav ka siis, kui kasutajal puudub interneti ühendus.

1.8. AppCache

AppCache on üks viis, kuidas tuua kasutajani veebileht, kui kasutajal ei ole interneti ühendust. Lisaks sellele, tõstab AppCache veebilehe jõudlust ja vähendab serveri koormust. (Using the, 2020)

AppCache töötab järgnevalt:

1) kui AppCache on juba kliendil olemas, siis brauser laeb kõik ressursid puhverdatud andmetest;
2) kui AppCachet ei ole, siis laeb brauser kõik päritud failid alla ja lisab need failid, mis on puhverdamise manifestis (fail, mis kontrollib puhverdamist), puhvrisse – järgmine kord kasutatakse juba AppCachet;
3) brauser kontrollib, kas manifesti on uuendatud;
4) kui see on uuendatud, siis brauser tõmbab kõik failid, mis on manifestis, alla ja uuendab vastavalt puhvrit.

Kogu protsess toimub tagataustal ehk see ei koorma kasutajaliidest. (Using the, 2020)

Nagu võib aru saada, siis AppCache täidab samasugust ülesannet (tunniplaani äpis), mis Service Workeridki – mõlemad üritavad veebilehte näidata kasutajale siis, kui kasutajal puudub interneti ühendus. Siinkohal peab kindlasti lisama, et AppCachet ei soovitata enam kasutada, kuna tegemist on vana tehnoloogiaga (Using the, 2020), mida oli raske kasutada keerulisemate veebilehtede puhul (Archibald, 2012). Äpis kasutati AppCachet Service Workerite varuplaanina, kuna Service Workerite brauseri toetus on ikka veel ünsa madal.

2. KASUTATUD ARENDAMISTEHNOLOOGIAD

2.1. Sass

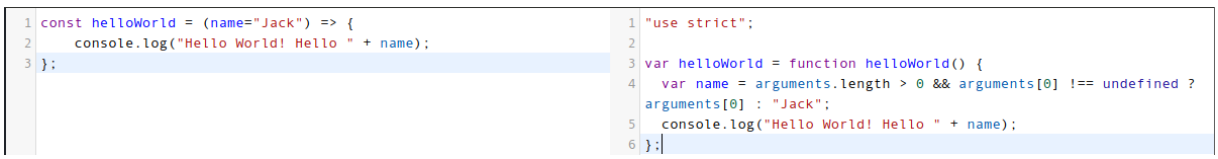
Sass on CSSi eelprotsessor (*preprocessor*), mis lisab CSS-i järgmised võimalused: muutujad, funktsioonid, *mixin*id, pärimised (*inheritance*) ja palju muud, mis muudab arendamise palju lihtsamaks. (Sass, 2020)

Sass-i tööpõhimõtteks on eelprotsessimine, mis tähendab seda, et Sass-i failid muudetakse ümber CSS-iks. Sass-i faile ei saa veebilehel lisada, kuna brauserid mõistavad ainult CSS-i. Seega lõplikult lisatakse siiski veebilehele CSS, kuid arendamine käib läbi Sass-i. Sass-i võib ka mõista kui eraldi tehiskeelt, mis muudetakse lõpuks brauserile arusaadavaks keeleks. (Giraudel, 2016)

2.2. Babel

Babel on tööriist, mida peamiselt kasutatakse uuema JavaScripti muutmiseks, et see töötaks vanemates keskkondades (nagu näiteks vanemates brauserites). Nimelt veebiarendus muutub küll kiiresti, kuid brauseritel läheb päris palju aega uute (JavaScripti) võimaluste võimaldamiseks arendajatele – seega on koguaeg uued võimalused brauseritest ees. Babel aga lahendab selle probleemi läbi süntaksi muutmise ja lisades toetusfunktsioonid (*polyfills*). (What is Babel, 2020)

Babel iseenesest on lähtekoodist-lähtekoodi kompilaator (*source-to-source compiler*), mis tähendab seda, et arendaja laseb uuema JavaScripti koodi läbi Babeli kompilaatori ning see tagastab JavaScripti koodi, mida võib kasutada vanemates keskkondades (joonis 8). (Babel, 2020)



Joonis 8. Uuem kood vasakul ja Babelist läbilastud ehk vanem kood paremal (Autori foto)

2.3. Webpack

Enne kui saab ära määratleda, mis on webpack, peab ära defineerima JavaScripti moodulid (*modules*). Nimelt programmeerimises tihtipeale jagatakse kood väiksemateks osadeks, kuna nii on lihtsam koodi hallata ja ka siluda (*debug*). Seda on võimalik teha mitut moodi, kuid üks moodus oleks jagada kood mooduliteks. Moodulid on siis väikesed programmi osad, millest lõpuks saadakse kokku üks suur programm – „kokku pakkimise" eest vastutab moodulite köitja (*module bundler*). Veebiarenduses aga on moodulid üsna halvasti toetatud – siin tulebki appi webpack. (Modules, 2020)

Webpack on moodulite köitja veebi jaoks ehk mooduleid on juba veebis võimalik kasutada täna läbi webpacki, kuid lisaks sellele suudab webpack lisada moodulina laadilehti, pilte, ja muid ressursse. Veel, mida peaks kindlasti mainima, et webpack suudab ka muuta lähtefaile läbi erinevate pluginate (näiteks läbi Babeli plugina) ehk webpacki on võimalik kasutada kui konstruktorit (*builder*), mis võtab (automaatselt) erinevad moodulid, muudab neid (vastavalt arendaja soovidele) ja lõpuks „pakib need kokku" ühte fail. (Concepts, 2020)

Webpacki tööpõhimõtteks on luua arendaja äpi moodulitest sõltuvusgraafik (*dependency graph*). Selle aluseb loob webpack ühe faili, milles on kõik moodulid ning mida on võimalik lisada veebilehele. (Dependency, 2020)

2.4. Terser

Terser on tööriist, mis minimeerib (*minify*) JavaScripti koodi (Github – terser, 2020). See tähendab, et terser eemaldab koodist kõik ebavajaliku arvuti perspektiivist – kommentaarid, tühikud, uued read jne (joonis 9). Minimeeritud kood on inimesele lugematu, kuid arvuti (brauser) saab sellest aru ja saab seda edukalt jooksutada. Minimeerimine on kasulik, sest siis muutuvad failid väiksemaks ja nende alla laadimine serverist on kiirem. Minimeerimine toetub koodi läbi sõelumisele (mida on võimalik teha mitut moodi) – sellega määratakse, mis alles jääb ja mis alles ei jää. (Code minification, 2020) Terserit kasutati äpis koos webpackiga.

```
"use strict";var helloWorld=function(){var l=arguments.length>0&&void
0!==arguments[0]?arguments[0]:"Jack";console.log("Hello World! Hello
"+l)};
```

Joonis 9. Minimeeritud kood (sama, mis joonisel 8 paremal) (Autori foto)

2.5. Node.js

Node.js on JavaScripti käituskeskkond (*runtime environment*). See tähendab, et JavaScript saab läbi Node.js keskkonna teha samu asju, mida suudab ükskõik, mis serveripoolne keel (nagu näiteks PHP). (Patel, 2018) Lisaks sellele, kasutatakse Node.js-i toel npm-i ka arendustegevuse juures (What is npm, 2020).

Node.js on ehitatud Google Chrome'i V8 JavaScripti mootori peale, mis üldse võimaldab JavaScripti jooksutamise. Käituskeskkonna ja mootori vahe on see, et mootor muudab JavaScripti koodi selliseks, millest arvuti saab aru, aga käituskeskkond võimaldab arendajale erinevaid tööriistu, mis teeb arenduse lihtsaks. (Patel, 2018) Äpis kasutatati Node.js-i koos npm-iga, mis muutis arendamisprotsessi lihtsamaks.

2.6. Npm

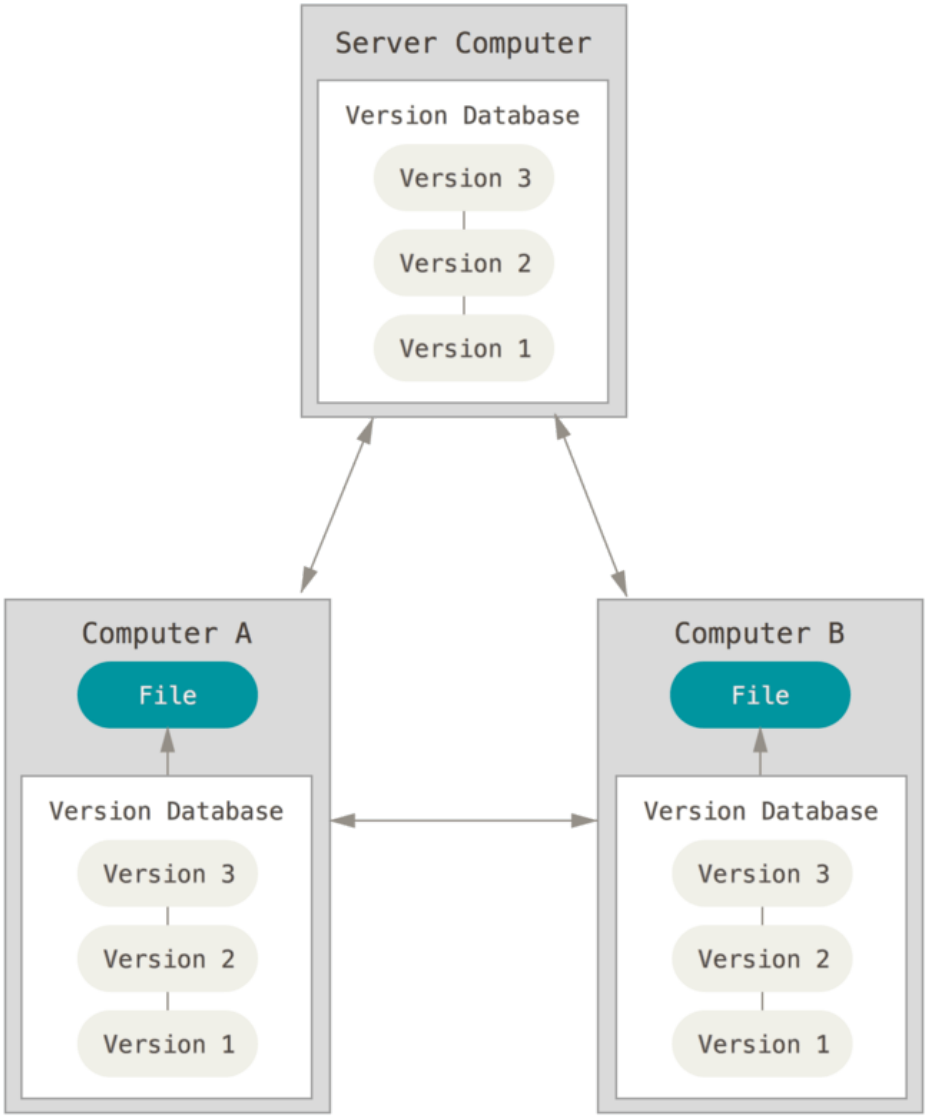
Npm on peamiselt paketihaldussüsteem (*package manager*) Node.js-ile (What is npm, 2020), kuid lisaks sellele on npm-il käsurealiides (*command line interface*), mis on väga kasulik arendusprotsessis. See kõik tähendab, et npm lubab alla tõmmata erinevaid avatud lähtekoodiga tööriistu ja tarkvara, mida saab jooksutada läbi Node.js-i. (About, 2020) Näiteks äpis kasutati webpacki ja Babelit läbi npm-i.

2.7. Git

Git on tasuta ja avatud lähtekoodiga versioonihaldaja (*version control*) (Git, 2020). See tähendab, et Git jäädvustab tehtud muudatused failides nii, et tulevikus on võimalik näha, mis muudatusi mingil ajahetkel täpselt tehti. Näiteks kui arendaja on versioonihaldamisega arendanud mingit projekti kaks aastat, siis arendajal on võimalik minna nii öelda ajas tagasi ja näha täpselt, mida ta kaks aastat tagasi tegi. Lisaks selle, kui arendajal peaks arvuti üles ütlema ning ta peaks kaotama kõik (arendamis)failid, siis versioonihaldajaga saab ta oma failid tagasi (ükskõik, mis arendamisperioodist). (Chacon, 2020)

Eristada saab kolme erinevat versioonihaldamist: lokaalne, tsentraliseeritud ja jagatud. Lokaalne tähendab seda, et kõik muudatused salvestatakse samasse arvutisse, kus muudatusi tehakse (näiteks eraldi kasuta vms), kuid see on üsna halb, sest kui arvuti kõvaketas ütleb

üles, siis on ka arendusfailid kadunud. Lisaks selle, on lokaalse versioonihaldamise puhul koodi jagamine ja programmeerijate vaheline arendus väga keeruline. Tsentraliseeritud versioonihaldamine (ehk arendusfailid on pilves) lahendab viimase, koodi jagamise, probleemi, kuid siiski jääb alles esimene probleem – failid võivad kaotsi minna serveri ülesütlemisega. Jagatud versioonihaldus (joonis 11) aga kasutab mõlemat strateegiat – failid on samaaegselt koos versioonidega nii pilves kui ka erinevate arendajate arvutis. Seega kui server peaks üles ütlema, siis arendajal on failid olemas, kui aga tööarvuti ütleb üles, on võimalik need kätte saada serverist. Kui aga peaks juhtuma mõlemat, siis saab faile küsida mõnelt teiselt arendajalt. (Chacon, 2020)



Joonis 10. Jagatud versioonihaldus (Chacon, 2020)

3. DISAINIPRINTSIIBID

3.1. Disainiprintsiibid serveri poolel

Serveri poolel kasutati ühte peamist disainiprintsiipi ja selleks on objektorienteeritavus. See tähendab, et äpp on jagatud suhteliselt sõltumatuteks kasutatavateks klassideks (joonis 12), mille põhjalt objekte luuakse. Tehniliselt on objektid alamprogrammid. (Kippar, 2018)

Lihtsamat öeldes on klassid programmi alamosad, mida on väga mugav arendamisel kasutada. Lisaks sellele, muudavad klassid koodi rohkem hallatavamaks, mis omakorda tähendab seda, et vigu on lihtsam üles leida ja programmi on lihtne edasi arendada.

```php
/**
 * Class for formating data
 *
 * @author Oliver Paljak, 10C, SÜG
 */
class DataFormatter extends DataReciever{

    /**
     * Get data as array
     *
     * @param string $file Path to the file we want data from
     * @param string $delimeter Delimeter explode the data. Optional, by default ";"
     *
     * @return array|false
     */
    public function formatAsArray($file, $delimeter=";"){
        $data = $this->getData($file);

        if($data !== false){
            $data = trim($data);
            $data = explode($delimeter, $data);

            return $data;
        }

        return false;
    }
}
```

Joonis 11. DataFormatter klass äpis (Autori foto)

Lisaks võeti äpi failistruktuuri disainimiseks eeskujuks Magento2, kus erinevad failid on jagatud erinevatesse kaustadesse. Täpset kuju ei jäljendatud, kuid näiteks kausta Blocks sees asuvad need PHP klassid, mis lisatakse kasutajaliidesse ja web kaustas on kõik veebiressursid (CSS, JavaScript ja pildid) – samamoodi on ka Magentos.

3.2. Disainiprintsiibid kliendi poolel

Üheks suurimaks põhimõtteks oli, et tehtud veebiäpp käituks või oleks väga sarnane päris äpile ehk tunniplaani veebileht peaks olema ka kättesaadav siis, kui kasutajal interneti ühendust ei ole. Seda võimaldasid nii puhverdamine (näiteks äpp salvestab koopia tunniplaanist brauserisse) kui ka Service Workerid, mille varuplaaniks on AppCache. Sellist põhimõtet, et veebiäpp oleks päris äpi sarnane nimetatakse veebiarendus maailmas ka PWA-ks. See omakorda tähendab, et kasutajad saavad selle veebilehe lisada oma seadme kodulehele ning see töötab kui eraldiseisev rakendus.

Lisaks sellele, kasutab äpp AJAX-it nii kontrollimaks, kas kasutajal on interneti ühendus kui ka tunniplaani saamiseks serverilt. See tähendab, et veebilehte ei pea uuesti tervikuna laadima, vaid ta suhtleb serveriga samaaegselt, kui kasutaja teda kasutab. Veel, mida äpp teeb, et tõsta kasutajakogemust on see, et kõik äpi CSS ja JavaScript on minimeeritud, mis vähendab laadimisaega ja kasutaja andmesidemahtusid.

Arenduse kohapealt võib üheks põhimõtteks pidada seda, et JavaScript on jagatud mooduliteks, mis tegi koodi rohkem hallatavamaks.

4. ÄPI TÖÖKORRALDUS

Kui esmakordne kasutaja tuleb äpi lehele (https://syg.edu.ee/tp), siis brauser automaatselt saadab serverile päringu, et saada äpi esileht. Server võtab päringu vastu, aga kuna äpis on kasutatud PHP-ed, siis server kõigepealt laseb päritud esilehe läbi PHP mootori ja siis alles tagastab selle.

Esilehel on kaks tähtsat kirjet – vormi võti ja kasutaja ID. Vormi võti (*form key*) on unikaalne sõne (*string*), mida kasutatakse selleks, et teha äpist AJAX päringuid. Nimelt kui vormi võtit ei oleks, siis võib igaüks maailmas, ükskõik kust, teha päringuid SÜG-i serveritele ning see ei ole optimaalne. Seega ainult need, kes kasutavad äppi, saavad teha päringuid, kuid sellest võib jääda väheseks. Igaüks maailmas võib ju tulla äpi lehele. Siin mängib rolli kasutaja ID (*user ID*), mis kaitseb selle eest, et ükski kasutaja ei tohi teha väga palju päringuid lühikese aja jooksul – näiteks praegu on piiriks seatud, et ükski kasutaja ei tohi teha 60 sekundi jooksul rohkem kui 30 päringut. Arvestades seda, et äpp teeb automaatselt ühe päringu iga 24 tunni tagant, et uuendada tunniplaani ehk kasutaja ise ei peaks põhimõtteliselt tegema ühtegi päringut serverile, siis see on igati mõistlik

Lisaks selle, kui kasutaja tuleb äpi esilehele, siis kontrollitakse üle, kas peab ka uuendama läbi Service Workeri ja AppCache puhverdatud veebiressursse (CSS, JavaScript, pildid jne). Neid uuendatakse iga 30 päeva tagant või siis kindlal kuupäeval (vaikimisi 7. septembril). See tagab, et äpp kasutab üsna värskeid ressursse ja kui on vaja, siis on võimalik ka teha manuaalseid uuendusi.

Viimasena laetakse äpi JavaScript. Kui kasutaja on esmakordne, siis laetakse ka Service Worker, mis puhverdab kõik vajalikud veebiressursid, et äppi saaks kasutada ka ilma interneti ühenduseta – see toimub ainult kõige esimesel laadimisel või ka siis, kui puhverdatud andmeid uuendatakse.

Järgmisena laetakse kogu SÜG-i tunniplaan (see tähendab kõiki kombinatsioone) kasutaja brauserisse, mis võimaldab näha ükskõik, mis tunniplaani ka siis, kui kasutajal ei ole interneti ühendust. See võib olla tundub ebamõistlik, et tõmmata brauserisse kogu tunniplaan, aga

tegelikkuses on selle maht väga väike – maksimaalselt 300 KB. Äpi kogu mahuks laadimisel võib pidada umbes 600 KB. Näiteks 2019/2020 mõlema (eraldi) poolaasta tunniplaani enda mahuks oli umbes 110 KB. Võrdluseks võib tuua delfi.ee, kus esmasel laadimise laeti vähemalt 3 MB (1 MB = 1024 KB) jagu andmeid ja seda ilma alla kerimata, kuna kerides laetakse delfi.ee's andmeid juurde.

Kui kasutaja on juba varem käinud äpis ja tunniplaani valinud, siis kuvatakse talle varem tehtud valik. Vajadusel, ehk kui viimasest uuendamisest on möödas 24 tundi ja kui kasutajal on olemas interneti ühendus, siis uuendatakse puhverdatud tunniplaani. Edasi jääb äpp juba kasutaja sisendit ootama.

Kui kasutaja valib uue tunniplaani ja kui tal on olemas puhverdatud andmed, siis pannakse soovitud tunniplaan kokku just nendest, kuid kui mingil põhjusel ei ole kasutajal puhverdatud andmeid (kasutaja saab iga kell need ära kustutada), siis saadakse soovitud tunniplaan serverist ning lisaks sellele lisatakse kasutaja brauserisse ka serverist saadud kogu SÜG-i tunniplaan, et vähendada koormust serverile. Järgmine kord saab valitud tunniplaani kokku panna just puhverdatud andmetest. Kogu protsess toimub kasutajale sujuvalt tänu animatsioonidele.

5. VERSIOONIDEST JA ARENDAMISPROTSESSIST

Kuigi äpil ametlikult ühtegi versiooni ei ole, võib siiski arendamisprotsessi jagada erinevateks etappideks. Esimene versioon valmis 2018. aasta suve alguseks ning see erines päris palju praegusest rakendusest. Nimelt selles staadiumis ei olnud võimalik äppi kasutada ka siis, kui kasutajal interneti ühendust ei olnud ning äpp sai andmeid otse SÜG-i serveri andmebaasidest kasutades MySQL-i ehk tegemist ei olnud eraldisesiva rakendusega. Tegelikult ei saaks antud lahendust nimetada äpiks või rakenduseks, kuna selles staadiumis oli see lihtsalt üks veebileht. Siiski oli see etapp äpi alguseks.

Teine rakenduse etapp valmis umbes aasta pärast – 2019. aasta kevadel. Antud uuenduse ajendiks oli see fakt, et äpp kasutas just andmete saamiseks MySQL-i. See aga ei ole optimaalne, sest andmebaasidele päringu tegemine on väga ressursimahukas. Lisaks oli esialgne idee luua ikkagi tunniplaani äpp, mitte tavaline veebileht ehk oli vaja, et tunniplaani saaks vaadata ka siis, kui kasutajal interneti ühendus puudub. Seega oli vaja sisse viia mastaapsed uuendused – nüüd tulid lähteandmed tunniplaani loomise rakendusest Untis ja tunniplaani äpp pidi olema ka siis kätte saadav, kui kasutajal interneti ühendust ei ole. Antud ülesanded ei olnud väga keerulised, kuna Untisest eksporditud failid on kõik ühesugused, mis teeb nende töötlemise lihtsaks. Äpi kättesaadavuse ka interneti ühenduseta kasutajatel võimaldas Service Workerite kasutamine ja andmete puhverdamine. Nüüd oli veebilehest saanud PWA.

Järgmised uuendused ei ole olnud enam nii mastaapsed, vaid pigem on need olnud äpi silumised, parandused ja täiustamised. Esimene selline uuendus toimus 2019. aasta suvel, kuna puhverdamisel kasutatud esileht ja ka erinevad veebiressursid võisid aeguda ja kuvada vale informatsiooni. Tegelikult veebiressurssidega ei oleks olnud nii suurt probleemi, kuid mis muret tekitas, oli puhverdatud esileht, kuna seal asuvad valikud (argumendid), millest tunniplaani saab kokku panna, võisid aeguda ja siis ei oleks olnud võimalik kuvada õigeid plaane, kuna argumendid oleksid olnud valed. See tähendas, et äpi kasutamine ilma interneti ühenduseta oleks muutunud ajapikku võimatuks. Probleem lahendati nii, et server ise iga 30 päeva tagant uuendab puhverdatud esilehte kui ka veebiressursse.

Teine silumisuuendus tehti 2019. aasta sügisel. Selle uuenduse ajendiks sai probleem, et põhiruumidel võisid olla tunniplaanis ka lisaruumid ning neid ruume ei saanud otsida päris täielikult. Näiteks põhiruumi otsimisel, millel oli olemas ka lisaruum, otsiti eelmises versioonis välja ainult sellised ainetunnid, millel oli ainult põhiruum kirjas, aga lisaruumi välja ei tulnud. Tehtud uuenduses saab ainetunde otsida nüüd iga ruumi järgi, olgu need kasvõi lisaruumid.

Viimaseks silumisuuenduseks on 2020. aasta talvel tehtud uuendus. Nimelt, kui kasutaja on juba valiku teinud minevikus, siis kasutajakogemuse tõstmiseks näidatakse talle eelmine kord valitud tunniplaani, mis on brauseris puhverdatud HTMLi kujul. Tuli aga ilmsiks, et varem valitud tunniplaani, mis on puhverdatud, tegelikult ei uuendata. See parandus lahendas selle probleemi. Nimelt kui uuendatakse kogu tunniplaani andmestiku kliendi brauseris, siis uuendatakse ka juba valitud tunniplaani, et kindlustada, et kasutajad näeksid alati kõige värskemat versiooni.

Arendamisprotsess ise oli mugav, kuna see oli ära automatiseeritud läbi erinevate arendustehnoloogiate (tööriistade), millest anti ülevaade punktis 2. Kõige olulisemaks tööriistaks võib arvatavasti pidada Git versioonihaldust, kuna kõik tehtud parandused ja etapid on ka pilves olemas, mis tähendab, et kui näiteks on soov millegipärast tagasi minna esialgse ehk MySQL-i lahendust kasutatava veebilehe peale, siis on see võimalik, kuna kogu kood selle jaoks on olemas.

6. PRIVAATSUSEST

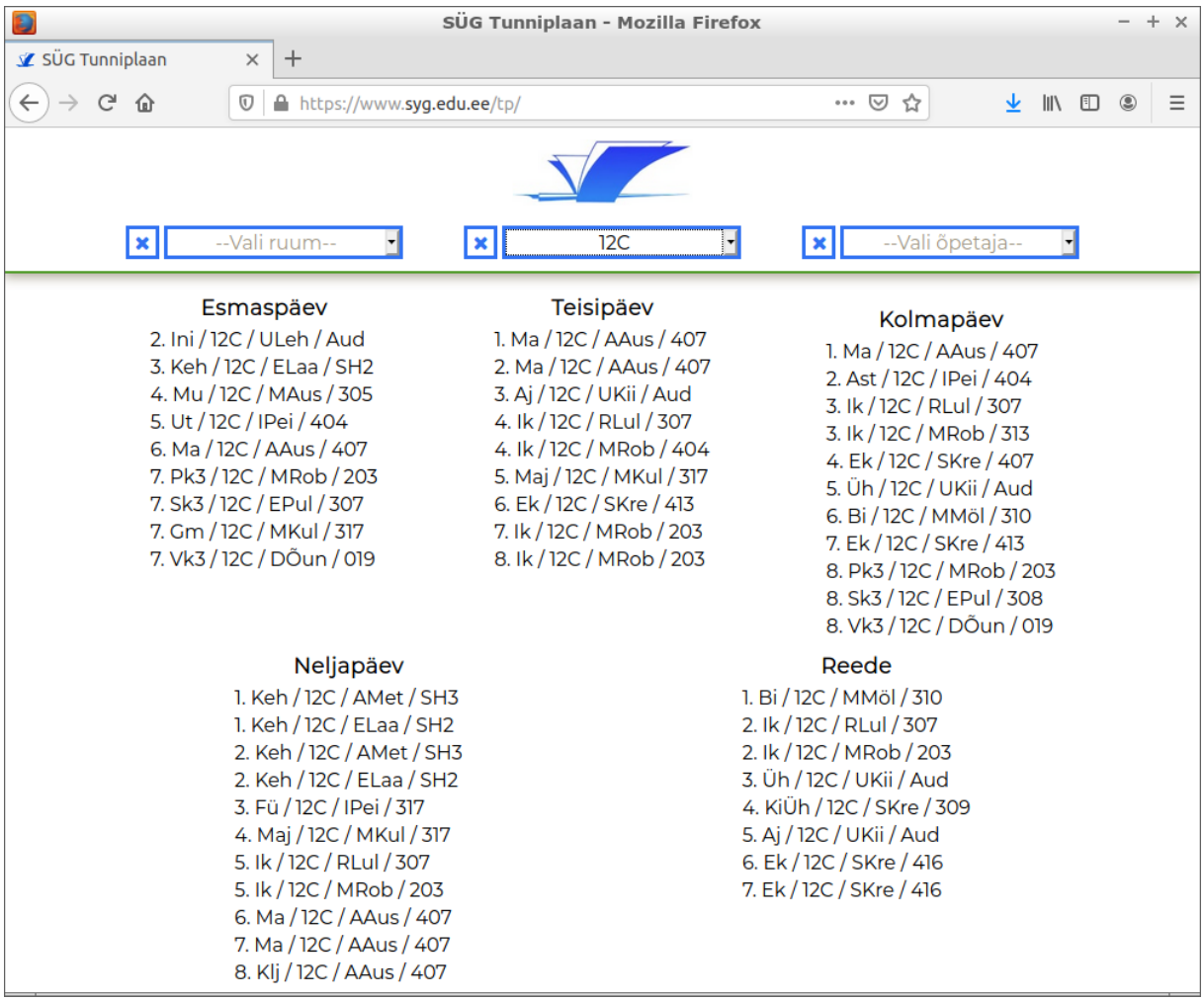
Privaatsus internetis on väga oluline teema, kuna enamus veebilehti kasutab tehnoloogiaid, mis aitavad määratleda, millised on inimese surfamisharjumused ning isegi interneti ajalugu. Arvatavasti üheks kuulsamaks võtteks on küpsiste (*cookies*) kasutamine, kuid loomulikult neid võtteid on rohkem. Siinkohal võiks luua mitu (vandenõu)teooriat, et mingi varju organisatsioon jälgib inimesi – näiteks maailma kuulsaim vilepuhuja Edward Snowden on tõestanud lekitanud dokumentidega, et riikidel on huvi inimeste jälgimise vastu, kuid enamasti on tegemist ettevõtetega, kes soovivad teada saada, mida inimesed internetis teevad. Ettevõtete eesmärgid jaguvad tavaliselt kaheks. Esimeseks on saada (statistilist) infot, kuidas nende tootel või teenusel läheb ning kuidas muuta neid veel rohkem kasutajasõbralikumaks läbi andmete kogumise. Sellisel juhul on tavaliselt andmed anonüümsed, kuid mitte alati. Teiseks eesmärgiks on koguda inimeste kohta informatsiooni, et näidata neile reklaame, mis võiksid nendele huvi pakkuda – see on üks tavalisemaid, aga ka efektiivsemaid viise, kuidas internetis raha teenida, kuna sellist ärimudelit kasutavad peaaegu kõik sotsiaalmeediad, aga ka populaarseim otsingumootor Google. Öelda, et kas andmete kogumine ja nende töötlemine on eetiline või mitte, väljub antud uurimistöö teemast, kuid siiski on oluline määratleda, kuidas käitub antud valdkonnas loodud tunniplaani rakendus.

Lühidalt öeldes, äpp kasutaja kohta andmeid ei kogu ning ei saada neid kolmandatale osapooltele. Detailidesse laskudes aga kasutab rakendus ühte küpsist, mis on PHP sessiooni küpsis, kuid seda vaid turvameetmete paika panemiseks – just vormi võti ja kasutaja ID seadistamiseks. Kuna tegemist on sessiooni küpsisega, siis see hävib sessiooni lõppedes ehk kui kasutaja paneb rakenduse kinni, siis ka antud küpsis ning kõik küpsise andmed (vormi võti ja kasutaja ID) kustutakse. Nii ei ole võimalik ka kasutajat tuvastada, kuna vormi võti ja kasutaja ID on suvaliselt genereeritud, kuid kasutaja ID genereerimise aluseks on kliendi avalik IP aadress. Lisaks sellele, need andmed, mida puhverdatakse (tunniplaan, viimane valik jne), on kõik kliendi poolel ning kasutajal on alati võimalus need ära kustutada.

7. TULEMUSED

Töö peamine eesmärk täideti ning valmis modernne eraldiseisev tunniplaani rakendus, mida saab koolielus igapäevaselt kasutada – https://syg.edu.ee/tp (joonis 12). Arendamisprotsessis omandas autor uusi teadmisi, kuidas veebirakendusi luua ja mis tehnoloogiaid ja võtteid täpsemalt veebiarenduses (nii arendusfaasis kui ka valmistootes) kasutatakse.

Ei saa täpselt määratleda, kui palju aega võttis valminud äpi tegemine, kuna arendamine käis vastavalt vajadusele, kuid võib oletada, et kokkuvõtvalt õpingute kõrvalt läks rakenduse tegemiseks umbes poolteist kuni kaks kuud.



Joonis 12. Valminud äpp (Autori foto)

KOKKUVÕTE

Töös anti ülevaade rakenduses kasutatud tehnoloogiatest (nii tava- kui ka arendamistehnoloogiad), disainiprintsiibidest, äpi töökorraldusest, versioonidest ja arendamisprotsessist ja ka rakenduse privaatsuspoliitikast.

Antud uurimistöö põhieesmärgiks oli valmistada SÜG-ile modernne igapäeva kasutust võimaldav eraldiseisev tunniplaani rakendus. Lisaeesmärgiks seati autori teadmiste pagasi suurendamine veebiarenduse valdkonnas. Mõlemad eesmärgid täideti.

Rakenduse arendus ise oli mugav, kuid äpi kasutamise käigus avastati mitmeid vigu, mis said ka lõpuks lahenduse ja uuenduse.

Kindlasti jätkab autor loodud rakenduse parandamist ja täiustamist, kui selleks peaks vajadus tekkima.

SUMMARY

The basis of this research paper was the fact that SÜG did not have a modern program to view lessons' timetable. Thus, the goal of this research paper was to create a modern separate timetable app that could be used in everday school life. Additional goal was to expand the author's knowledge of web development.

This research paper gives an overview of used technologies, design principles, how the app works, app's versions and development processes, privacy policy and end results.

Some bugs were found in the app but they were addressed accordingly and fixed.

The two goals of this research paper were accomplished: a modern separate timetable app that is ready for real world use has been developed and the author gained much knowledge in the process, which he can use in his upcoming projects.

KASUTATUD MATERJALID

About npm. URL=https://docs.npmjs.com/about-npm/. 23.02.2020.

Ajax. URL=https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX. 20.02.2020.

Archibald, J. 2012. *Application Cache is a Douchebag*. URL=https://alistapart.com/article/application-cache-is-a-douchebag/. 22.02.2020.

Archibald, J. 2014a. *Service Worker - first draft published*. URL=https://jakearchibald.com/2014/service-worker-first-draft/. 21.02.2020.

Archibald, J. 2014b. *The offline cookbook*. URL=https://jakearchibald.com/2014/offline-cookbook/#network-falling-back-to-cache. 22.02.2020.

A re-introduction to JavaScript. URL=https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript. 05.02.2020.

Babel Plugin Handbook. URL=https://github.com/jamiebuilds/babel-handbook/blob/master/translations/en/plugin-handbook.md#basics. 22.02.2020.

Bibeault, B. & Katz, Y. & Rosa, A. D. 2015. *jQuery in Action*. New York: Manning Publications Co.

Bidelman, E. 2010. *The Basics of Web Workers*. URL=https://www.html5rocks.com/en/tutorials/workers/basics/. 21.02.2020.

Breaking the CommonJS standardization impasse. URL=https://github.com/nodejs/node-v0.x-archive/issues/5132#issuecomment-15432598. 23.02.2020.

Can I use – Service Workers. URL=https://caniuse.com/#feat=serviceworkers. 21.02.2020.

Chacon, S. & Straub, B. 2020. *Pro Git*. Rugby: Apress.

Code minification. URL=https://webplatform.github.io/docs/concepts/programming/javascript/minification/. 22.02.2020.

Code Splitting. URL=https://github.com/medikoo/modules-webmake/issues/7. 22.02.2020.

Concepts – webpack. URL=https://webpack.js.org/concepts/. 22.02.2020.

Dahl, R. 2009. *Ryan Dahl: Original Node.js presentation*. URL=https://www.youtube.com/watch?v=ztspvPYybIY. 23.02.2020.

Dependency Graph - webpack. URL=https://webpack.js.org/concepts/dependency-graph/. 22.02.2020.
Giraudel, H. & Suzanne, M. 2016. *Jump Start Sass*. Collingwood: SitePoint Pty. Ltd.
Git. URL=https://git-scm.com/. 23.02.2020.
Github – Babel. URL=https://github.com/babel/babel. 22.02.2020.
Github – node. URL=https://github.com/nodejs/node. 23.02.2020.
Github – Sass. URL=https://github.com/sass/sass. 22.02.2020.
Github – terser. URL=https://github.com/terser/terser. 22.02.2020.
Github - webpack. URL=https://github.com/webpack/webpack. 22.02.2020.
Hamilton, N. 2008. *The A-Z of Programming Languages: JavaScript*. URL=https://www.computerworld.com/article/3458282/the-a-z-of-programming-languages-javascript.html. 05.02.2020.
History of PHP. URL=https://www.php.net/manual/en/history.php.php. 03.02.2020.
Hopmann, A. 2007. *The story of XMLHTTP*. URL=https://web.archive.org/web/20170424220609/http://www.alexhopmann.com/xmlhttp.htm. 21.02.2020.
How CSS works. URL=https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_works. 04.02.2020.
How does AJAX work. URL=https://web.archive.org/web/20071019003941/http://www.interaktonline.com/Support/Articles/Details/AJAX%3A+Asynchronously+Moving+Forward-How+does+AJAX+work%3F.html?id_art=36&id_asc=308. 20.02.2020.
HTML5. URL=https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. 22.02.2020.
HTML Standard. URL=https://html.spec.whatwg.org/multipage/introduction.html. 26.01.2020.
HTML & CSS. URL=https://www.w3.org/standards/webdesign/htmlcss. 03.02.2020.
Interview with Julian Shapiro. URL=https://www.awwwards.com/interview-with-julian-shapiro.html. 13.02.2020.
Introduction – JavaScript. URL=https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction. 05.02.2020.

jQuery. URL=https://jquery.com/. 05.02.2020.
Kippar, J. 2018. *Objektorienteeritud programmeerimise põhimõtted*. URL=https://eopearhiiv.edu.ee/e-kursused/eucip/arendus/342_objektorienteeritud_programmeerimise_phimtted.html. 23.02.2020.
Libby, A. 2019. *Inroducing Dart Sass*. Rugby: Apress.
Lie, H. W. & Bos, B. 1999. *Cascading Style Sheets: Designing for the Web*. Harlow: Addison Wesley.
McKenzie, S. 2016. *~2015 in review*. URL=https://medium.com/@sebmck/2015-in-review-51ac7035e272. 22.02.2020.
Metshein, M. 2020. *PHP – Mis on PHP?* URL=https://www.metshein.com/unit/php-mis-php/. 03.02.2020.
Modules – webpack. URL=https://webpack.js.org/concepts/modules/. 22.02.2020.
Patel, P. 2018. *What exactly is Node.js*. URL=https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/. 23.02.2020.
Raggett, D. & Lam, J. & Alexander, I. & Kmiec, M. 1997. *Raggett on HTML 4*. Harlow: Addison Wesley.
Robbins, J. N. 2012. *Learning Web Design*. Sebastopol: O'Reilly.
Sass Basics. URL=https://sass-lang.com/guide. 22.02.2020.
Service Worker API. URL=https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API. 21.02.2020.
Shapiro, J. 2014a. *Velocity.js*. URL=http://velocityjs.org/. 13.02.2020.
Shapiro, J. 2014b. *Initial fully-documented source release*. URL=https://github.com/julianshapiro/velocity/commit/580e01711e6d362ad8f1e9b4805e7db76673d686. 13.02.2020.
Swartz, A. 2005. *A Brief History of Ajax*. URL=http://www.aaronsw.com/weblog/ajaxhistory. 20.02.2020.
Uglify-es is stale. URL=https://github.com/mishoo/UglifyJS2/issues/3156#issuecomment-392943058. 22.02.2020.
Using the application cache. URL=https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache. 22.02.2020.

Using Web Workers.
URL=https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers. 21.02.2020.
Vallaste, H. *E-teatmik – IT ja sidetehnika seletav sõnaraamat*. URL=http://vallaste.ee/. 16.03.2020.
What can PHP do. URL=https://www.php.net/manual/en/intro-whatcando.php. 03.02.2020.
What is Babel. URL=https://babeljs.io/docs/en/. 22.02.2020.
What is Caching and How it Works. URL=https://aws.amazon.com/caching/. 08.03.2020.
What is npm. URL=https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. 23.02.2020.
What is PHP. URL=https://www.php.net/manual/en/intro-whatis.php. 03.02.2020.

LISAD

Lisa 1. Mõisted

Builder – konstruktor; programmeerimises abiprogramm, mis hõlbustab rakendusprogrammi koostamist. (Vallaste, 2020)

Cache – puhver; puhverdama; andmeid salvestama nii, et neid saaks vajadusel kiiresti kätte (näiteks programm ei pea uuesti andmeid töötema, vaid saab need juba töödeldud kujul puhvrist kätte).

Client – klient; klientprogramm; programm, mida kasutatakse serverprogrammi või teise arvutiga ühenduse võtmiseks ja sealt andmete hankimiseks. (Vallaste, 2020)

Client-side – kliendipoolne; vt *Client*.

Command line interface – käskliides; käsurealiides; operatsioonisüsteemi või rakenduse kasutajaliides, mis võtab vastu klaviatuurilt sisestatavaid käske üks rida korraga. (Vallaste, 2020)

Command line scripting – käsurea skriptimine; skriptide loomine, mida saab käsureal kasutada.

Compiler – kompilaator; kõrgkeele translaator ehk programm, mis transleerib lähtekoodi objektkoodiks (*object code*). (Vallaste, 2020)

Cookie – küpsis; andmeplokk (tekstifail), mille veebiserver saadab veebibrauserile ja mis salvestatakse klientarvuti kõvakettale. (Vallaste, 2020)

Cross-platform – mitmeplatvormiline; programmikeel, rakendusprogramm või seade, mis võib töötada mitmel platvormil. (Vallaste, 2020)

Debug – siluma; programmides vigu avastama, lokaliseerima ja kõrvaldama. (Vallaste, 2020)

Dependency graph – sõltuvusgraafik; graafik või ka andmetekogum, mis näitab, mis millest sõltub.

Document Object Model (DOM) – dokumendiobjektide mudel; eeskiri selle kohta, kuidas objekte (tekst, pildid, pealkirjad, lingid jne) veebilehel esitada. DOM määrab ära, millised atribuudid kuuluvad millise objekti juurde ning kuidas objekte ja atribuute käsitleda. (Vallaste, 2020)

Dynamic – dünaamiline; programmi töötamise ajal käigupealt vastavalt vajadusele tehtavad operatsioonid. (Vallaste, 2020)

Event handler – sündmusetöötleja; tarkvararutiin mitmesuguste sündmuste (näiteks hiire liigutamine, hiireklikk, klahvivajutus jne) töötlemiseks. (Vallaste, 2020)

Extensible Markup Language (XML) – laiendatav märgistuskeel; XML on suvaliste andmete struktureerimiseks mõeldud märgistuskeel, mis loodi eesmärgiga võtta see veebis kasutusele HTML-i asemel. (Vallaste, 2020)

HyperText Transfer Protocol (HTTP) – hüperteksti edastusprotokoll; andmevahetusprotokoll, mida kasutatakse Internetis dokumentide vahetamiseks. (Vallaste, 2020)

Inheritance – pärilus; klassi omaduste vaikimisi laiendamine alamklassile. (Vallaste, 2020)

Input – sisend

Library – teek; valmiskompileeritud alamprogramm, mida programm saab kasutada. (Vallaste, 2020)

Linker – linkur; mitut objekt- või laademoodulit täitmisprogrammiks ühendav programm.

Loop – silmus; korduvale täitmisele kuuluv käsujada. (Vallaste, 2020)

Markup language – märgistuskeel; dokumendi tekstisse selle töötlemise hõlbustamiseks manustatavatest lipikutest koosnev käsukeel. (Vallaste, 2020)

Minify – minimeerima; väiksemaks muutma.

Module – moodul; tarkvaramoodul kujutab endast programmi osa – programmid koosnevad ühest või mitmest iseseisvalt valmiskirjutatud moodulist, mida ei ühendata omavahel kokku enne, kui programmi hakatakse kokku „liimima”. (Vallaste, 2020)

Module bundler – moodulite köitja; linkur; vt *Linker*.

Native – loomulik; oma; platvormi enda oma.

Node – sõlm; dokumendiobjektide mudelis nimetatakse sõlmeks iga elementi, selle elemendi iga atribuuti ja igale atribuudile vastavat tekstilist sisu. Vt ka *Document object model*. (Vallaste, 2020)

Object code – objektkood; Käsud ja andmed translaatori väljundkujul. Programmeerija kirjutab arvutiprogrammi programmeerimiskeeles ja et arvuti seda mõistaks, transleeritakse programm objektkoodiks. (Vallaste, 2020)

Object-oriented – objektorienteeritud

Package manager – paketihaldussüsteem; programm, mis lubab hallata (installida, eemaldada, muuta jne) erinevaid pakke (programme, abiprogramme, faile jne).

Parse – sõeluma; jagama koostisosadeks, mida on võimalik analüüsida. (Vallaste, 2020)

Polyfill – toetusfunktsioon; asendusfuntksioon; teotusprogramm; funktsioon või programm, mida kasutatakse, kui klient kasutab vanemat platvormi, millel puudub oma (*native*) funktsioon, millel on sama ülesanne (näiteks vanemad brauserid ei toeta funktsiooni Object.keys(), siis arendajal on võimalik enda veebilehel antud funktsioon luua – loodud funktsioonist saabki toetusfunktsioon).

Preprocessor – eelprotsessor; tarkvara, mis töötab toorandmed põhiprogrammile sobivale kujule. (Vallaste, 2020)

Proxy – proksi; vahendaja; vt *Proxy server*.

Proxy server – proksi; puhverserver; välisliiklust vahendav tulemüüri komponent – kui klient pöördub tulemüüriga kaitstud veebiserveri poole, siis proksi asub veebilehitseja ja selle veebiserveri vahel püüdes kinni veebiserverile saadetud päringud ja kontrollib, kas ta saab neile päringuile ise vastata, kui mitte, siis edastab proksi päringu veebiserverile. (Vallaste, 2020)

Push – tõukama; rakendustes serveri algatusel andmeid serverilt kliendile saatma – kui veebis on tavaliselt kasutusel tõmbemeetod, kus server saadab veebilehti brauserile ainult viimase nõudmisel, siis tõukemeetod tähendab seda, et serverid saadavad andmeid kliendile sõltumata sellest, kas viimased on selleks soovi avaldanud või mitte (päringu teinud). Tavaliselt käib tõukemeetodi kasutamine kliendi nõusolekul. (Vallaste, 2020)

Push notifications – tõuketeated; vt *Push*.

Rendering – visualiseerimine; andmete teisendamine kuvamiseks või printimiseks sobivasse vormingusse. (Vallaste, 2020)

Render tree – visualisatsiooni puu; vt *Tree* ja *Rendering*.

Runtime environment – käituskeskkond; käitusfaasikeskkond; keskkond, milles programmi täidetatakse – põhimõtteliselt on see arvuti „platvorm”, kuhu kuuluvad keskprotsessor, opsüsteem ja süsteemiprogrammid. (Vallaste, 2020)

Scalable Vector Graphics (SVG) – mastabeeritav vektorgraafiak; veebis kasutamiseks mõeldud ja XML-keeles väljendatud vektorgraafika vorming. Erinevalt bittrastritest (näiteks JPEG, PNG jne) saab vektorjoonised mastabeerida (nende suurust muuta) ilma igasuguste moonusteta. (Vallaste, 2020)

Script – skript; vt *Scripting language*.

Scripting language – skriptikeel; skriptikeeles kirjutatud programme ei saa arvutil vahetult täita, vaid neid tuleb tõlkida arvutile mõistetavasse keelde. (Vallaste, 2020)

Server – programm, mis pakub teenuseid teistele programmidele (klientprogrammidele). (Vallaste, 2020)

Server-side – serveripoolne; vt *Server*.

Single thread – üheharuline; vt *Thread*.

String – string; sõne; arvutitermoloogias tervikuna käsitletav elemendi- või märgijada (erinevalt sõnast või arvust ei pruugi sõnel olla mingit konkreetset tähendust). (Vallaste, 2020)

Structured Query Language (SQL) – struktuurpäringukeel; enimkasutatav päringukeel. Päringukeeled kujutavad endast reeglite kogumit, mille alusel konstrueeritakse päringuid andmete otsimiseks andmebaasidest. Erinevad andmebaasihalduri toetavad erinevaid päringukeeli, kusjuures SQL on pooleldi standardiseeritud. (Vallaste, 2020)

Style sheet – laaditabel; laadileht; mall; tekstitöötluse ja elektronkirjastamise puhul määrab laadileht ära nende dokumentide küljenduse, millele ta kinnistatakse – laadilehe täitmisel määratakse ära sellised parameetrid nagu paberilehe suurus, veerised, kirjatüübid jne. (Vallaste, 2020)

Synchronization – sünkroniseerimine; sünkronisatsioon; infovahetus mobiilsete seadmete vahel. (Vallaste, 2020)

Thread – haru; lõim; iseseisev, teistest programmiosadest sõltumatult täidetav programmiosa. (Vallaste, 2020)

Transform – teisendama; kuju muutma

Tree – puu; andmestruktuur, kus iga element on ühendatud ühe või mitme temast allpool asuva elemendiga (puu hargneb ülalt alla). (Vallaste, 2020)

Version control – versioonihaldus; Tarkvaratoodete või muude dokumentide erinevate versioonide haldamine. (Vallaste, 2020)