Opinnäytetyö (AMK)

Tietojenkäsittely

2021

Matti Wallenius

WEB-KÄYTTÖLIITTYMÄ AUTOMAATIODATAN VISUALISOINTIIN

– Case Green Automation Group Oy

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittely

2021 | 39 sivua, 4 liitesivua

Matti Wallenius

Web-käyttöliittymä automaatiodatan visualisointiin

- Case Green Automation Group Oy

Automaation juuret juontavat tuhansien vuosien taakse. Todelliseen nosteeseen se kuitenkin pääsi teollisen vallankumouksen seurauksena ja nykyään automaatio vaikuttaa lähes kaikissa teknisissä järjestelmissä ja laitteissa.

Tämän opinnäytetyön tarkoituksena oli vertailla tuotantodatan visualisoinnissa hyödynnettäviä visualisointikirjastoja ja kehittää vertailun perusteella valittua Chart.js kirjastoa hyödyntävä web-käyttöliittymä. Käyttöliittymän avulla toimeksiantajayrityksen asiakkaat voivat seurata automatisoitua salaattien ja yrttien kasvatusprosessia. Kehityksen loppuvaiheessa käyttöliittymän resurssit optimoitiin tiedostokokojen ja suorituskyvyn parantamiseksi.

Tuloksena saatiin toimiva käyttöliittymä, joka onnistuttiin asentamaan paikallisesti asiakkaan laitteelle siten, että siihen pääsi käsiksi kaikilla asiakkaan verkossa toimivilla laitteilla selaimen välityksellä käyttämällä palvelimen IP-osoitetta. Optimointi osoittautui tehokkaaksi tavaksi vähentää palvelimen ja asiakaslaitteiden välistä ylimääräistä liikennettä, mutta testiympäristössä suoritetuissa suorituskykymittauksissa ei havaittu merkittäviä parannuksia latausajoissa.

Asiasanat:

automaatio, datavisualisointi, käyttöliittymä

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Bachelor’s degree of Business Information Technology

2021 | 39 pages, 4 pages in appendices

Matti Wallenius

Web user interface for visualization of automation data

- Case Green Automation Group Ltd

Automation has been with us for thousands of years. Its full potential was reached during industrialization and today automation affects almost all technical systems and devices.

The purpose of this thesis was to compare visualization libraries meeting the requirements of the client and to develop a web user interface that utilized the selected Chart.js library. Using the user interface customers of the client company could follow the automated growing process of salads and herbs. At the end of development, the user interface resources were optimized to reduce the size of them and to improve performance.

The user interface was successfully installed locally on the customers’ server in such a way that it was accessible on all devices in customer network with a browser and the IP of the server. Optimization was found to be very effective in reducing the amount of traffic between server and client devices, but no significant performance improvements were detected in test the environment.

Keywords:

automation, data visualization, user interface

Sisältö

Liitteet

Kuvat

Taulukot

Sanasto

CSS	Cascading Style Sheets on mekanismi, jonka avulla muokataan elementtien ulkoasua ja asettumista verkkosivuilla (MDN Web Docs 2021d).
JS	JavaScript on skriptikieli, jonka avulla voidaan suorittaa monimutkaisia ominaisuuksia verkkosivustoilla (MDN Web Docs 2021a).
JSON	JavaScript Object Notation on kevyt tekstipohjainen tietotyyppi tiedon säilytykseen, joka noudattaa JS-oliosyntaksia (MDN Web Docs 2021b).
Minifiointi	Minifioinnilla tarkoitetaan tekstipohjaisten tiedostojen koon pienentämistä poistamalla koodista sen suorituksen kannalta turhaa koodia (Imperva 2021).
MIT	Lisenssi, joka sallii koodin rajattomat käytön, kunhan tekijänoikeustiedot säilytetään lähdekoodissa (Goldstein 2021).
PHP	PHP: Hypertext Preprocessor on avoimen lähdekoodin skriptikieli, jonka koodi suoritetaan palvelimella (MDN Web Docs 2021c).

1 Johdanto

Automaation juuret juontavat yli 2000 vuoden taakse. Todelliseen nosteeseen automaatio pääsi teollisen vallankumouksen seurauksena, kun teolliset laitteet loivat tarpeen toiminnallisuuksien automatisoimiseksi. Nykyään automaatio vaikuttaa lähes kaikissa teknisissä järjestelmissä ja laitteissa. (Koskinen 2018.)

Tämän opinnäytetyön toimeksiantaja Green Automation Group Oy suunnittelee ja toimittaa kasvatusautomaatiojärjestelmiä salaattien ja yrttien vesiviljelyyn kasvihuoneympäristössä. Kasvatusprosessin aikana kerätään dataa mm. tuotanto- ja kylvömääristä, järjestelmän veden laadusta ja määrästä, järjestelmävirheistä ja tapahtumista. Data tallentuu paikalliseen tietokantaan, josta se voidaan hakea aikaisemmin kehitetyn rajapinnan avulla. Tämän datan hyödynnettävyys on kuitenkin rajallista, sillä käytössä on vain raakadata.

Opinnäytetyön tavoitteena on vertailla tuotantodatan visualisoinnissa hyödynnettäviä visualisointikirjastoja, sekä kehittää valittua kirjastoa hyödyntävä web-käyttöliittymä. Käyttöliittymän avulla toimeksiantajayrityksen asiakkaat voivat seurata kasvatusprosessin kehittymistä ja tilaa. Käyttöliittymän avulla visualisoidaan myös tapahtumadataa, joka esitetään taulukkomuodossa ja sen käsiteltävyyteen tulee kiinnittää erityishuomiota.

Käyttöliittymän kehityksessä hyödynnettävät teknologiat ovat vapaasti valittavia, kunhan ne täyttävät kehitykselle asetetut minimivaatimukset. Tutkimusmenetelmänä toimii toiminnallinen tutkimus.

Teoriaosuudessa esitellään visualisoinnin kannalta oleelliset taustalla vaikuttavat teknologiat, vertailtavat datavisualisointikirjastot sekä optimoinnissa hyödynnettävät työkalut.

Toteutus on jaettu kolmeen lukuun. Ensimmäisessä luvussa käydään läpi kehitykselle asetetut vaatimukset sekä kehitysympäristö, minkä jälkeen siirrytään käyttöliittymän suunnitteluun ja tekniseen toteutukseen sekä suoritetaan datavisualisoinnissa hyödynnettävien kirjastojen vertailu. Viimeinen toteutusluku keskittyy käyttöliittymän optimointiin suorituskyvyn kannalta.

Optimoinnissa hyödynnetään avoimen lähdekoodin työkaluja ja sen päätavoitteena on vähentää turhaa resurssien lataamista palvelinkuormituksen minimoimiseksi ja käyttäjän näkökulmasta latausaikojen pienentämiseksi.

Lopetusluvussa kerrataan opinnäytetyön tavoitteet ja katsotaan, miten niissä onnistuttiin. Teknologioiden toimivuus keskenään ja mahdolliset ongelmatilanteet kehityksessä käydään läpi. Luvun lopussa suunnataan katse tulevaan ja pohditaan, mihin käyttöliittymän kehitys voitaisiin suunnata tulevaisuudessa.

2 Käytetyt teknologiat

2.1 Bootstrap

CSS-kehykset ovat työkaluja, jotka kokoavat lukuisia tyylimuokkauksia yhden paketin alle. Niiden avulla selainpuolen kehitys nopeutuu ja tiimityöskentely helpottuu. Sen sijaan että aikaa käytettäisiin toistuviin, lähes kaikilla sivustoilla esiintyvien tyylisäätöjen tekemiseen voidaan keskittyä varsinaiseen kehitystyöhön. (Lawrence 2018.)

Bootstrap on Mark Oton vuonna 2011 julkaisema ilmainen CSS-kehys, joka tyylimuokkausten lisäksi tarjoaa useita modernia web-kehitystä tukevia ominaisuuksia kuten selainriippuvaisten erojen alustus, mobiililaitteiden asettaminen etusijalle, responsiivisuus ja selainikkunan jakaminen ruudukko- ja rivijärjestelmään. Se on noussut suosituimmaksi CSS-kehykseksi vuosien varrella ja uusin pääversio 5 julkaistiin toukokuussa 2021. (Bootstrap 2021a.)

2.1.1 Selainriippuvaisten erojen alustus

Selaimet asettavat normaalisti HTML-dokumenttien elementeille tiettyjä tyylimuokkauksia. Nämä muokkaukset on koottu ”Useragent stylesheet” tiedostoon, jonka sisältö vaihtelee selainmoottorin ja selaimen version mukaan. (Lachman 2016.)

Ratkaisuna selainten välisiin eroihin on kehitetty useita erilaisia CSS-alustustiedostoja, jotka asettavat elementeille nollatut arvot, jotta lopullinen sivusto näyttäisi kaikille käyttäjille samalta selaimesta riippumatta.

Bootstrapin mukana tuleva Reboot.css toimii hiukan eri tavalla kuin monet vanhemmat alustustiedostot. Sen sijaan, että se alustaisi kaikki mahdolliset elementit, keskittyy Reboot asettamaan selainten tyypillisesti muokkaamia elementtejä järkevään tilaan, josta niitä on helppo muuttaa kehyksen tarpeiden mukaan. (Bootstrap 2021c.)

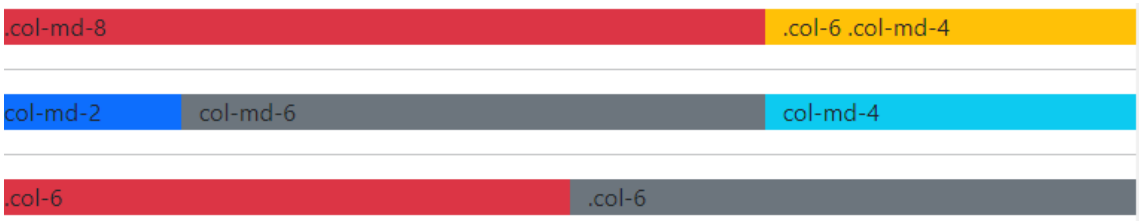
2.1.2 Responsiivisuus ja ”Mobile-first” periaate

Bootstrapin kehityksessä on otettu versiosta 3 lähtien käyttöön ”mobile first” periaate (Bootstrap 2021a). Käytännössä tämä tarkoittaa sitä, että html-elementtien CSS-tyylit on ensin asetettu mobiililaitteille sopiville perusarvoille ja vasta tämän jälkeen ne skaalataan tarpeen vaatiessa mediakyselyiden avulla suuremmille selainikkunaleveyksille sopiviksi. (Flores 2021.)

2.1.3 Ruudukkojärjestelmä

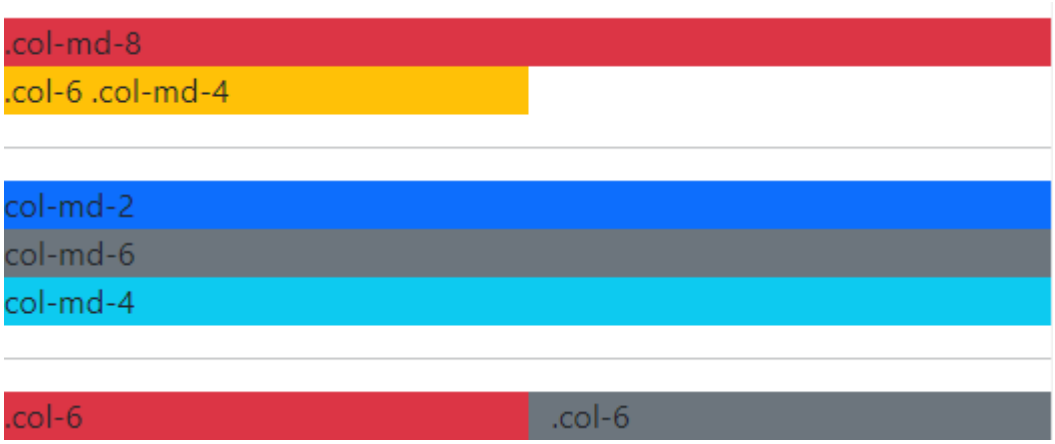
Responsiivisen kehityksen kannalta ehkä tärkein ominaisuus on Bootstrapin tarjoama ruudukkojärjestelmä (engl. Grid layout). Perusperiaate on, että selainikkuna jaetaan riveihin sekä 12 osasta koostuvaan ruudukkoon.

Ruudukon osan koko on kehittäjän päätettävissä ja erikokoisia osia voidaan yhdistää vapaasti, kunhan yhdellä rivillä on enintään 12 osaa (Kuva 1). Osille voidaan asettaa myös selainikkunan leveyden mukaan vaihtuva koko asettamalla elementille haluttua rajapistettä kuvaava luokkanimi.



Kuva 1 Ruudukkojärjestelmä, leveä selainikkuna.

Jokaiselle rivin osalle voidaan asettaa oma rajapiste (engl. Breakpoint) jolla ohjataan kohtaa, jossa siirrytään mobiilinäkymään. Mobiilinäkymässä samassa rivissä olevat osat asettuvat päällekkäin, kunhan osille on asetettu rajapiste (Kuva 2). Mikäli rajapiste puuttuu, samalla rivillä olevat osat näkyvät vierekkäin kaikilla selainikkunaleveyksillä. (Bootstrap 2021b.)



Kuva 2 Ruudukkojärjestelmä, kapea selainikkuna.

2.2 DataTables

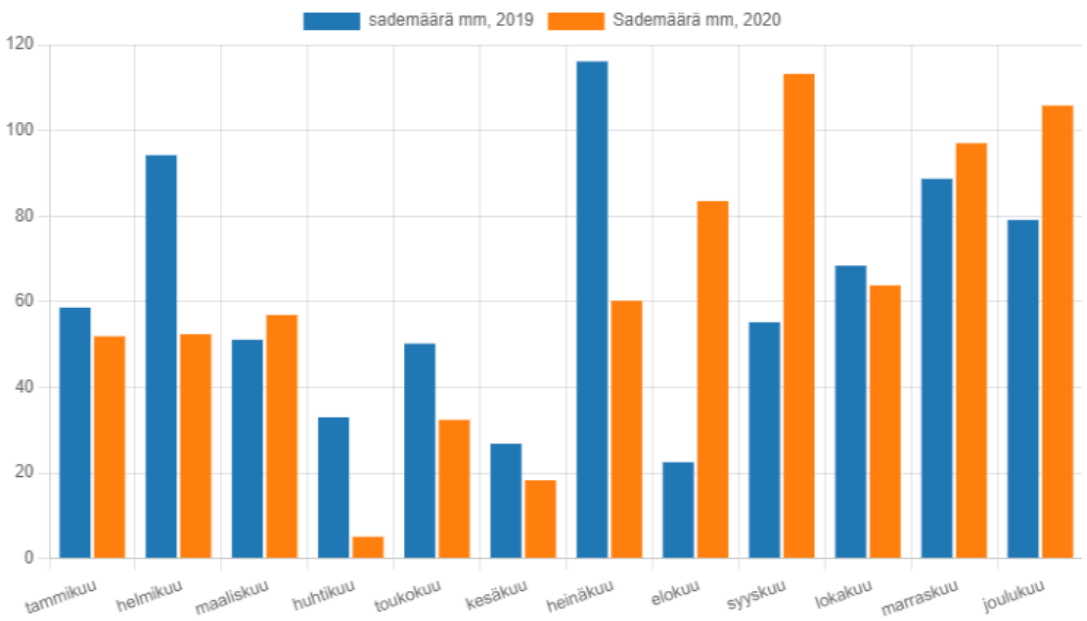
DataTables on SpryMedian kehittämä JQuery-kirjaston liitännäinen, joka lisää normaaleihin html-taulukoihin useita datan käsittelyä parantavia ominaisuuksia. Sen virallinen tavoite on parantaa tiedon saatavuutta HTML-taulukoissa. Alustamalla normaali html-taulukko DataTables -taulukoksi, saadaan tuki taulukon jakamiseen sivuihin, joiden pituutta käyttäjä pystyy itse säätämään. Datan järjestely onnistuu sarakekohtaisesti ja käytössä on myös kaikkien sarakkeiden yhteinen hakupalkki. Koska prosessointi tapahtuu käyttäjän selaimella, tulokset näkyvät heti ilman palvelimen ja selaimen välistä liikennettä. (DataTables 2021a.)

Tuettuna on myös virallisia perusominaisuuksia parantavia lisäosia, joiden avulla pystyy mm. suodattamaan tuloksia sarakekohtaisesti ja lataamaan taulukossa esiintyvää dataa CSV tai Excel muodoissa tai tulostamaan. (DataTables 2021b.)

2.3 Vertailtavat visualisointikirjastot

2.3.1 Chart.js

Chart.js (Kuva 3) on Nick Downien vuonna 2013 julkaisema avoimen lähdekoodin JS-kirjasto. Tuettuna on kahdeksan eri kaaviotyyppiä, joita voidaan sekoittaa keskenään yhdistelmäkaavioiden luomiseksi. Toisista vertailtavista kirjastoista poiketen kaavioiden luonnissa käytetään html5 canvas-elementtiä.

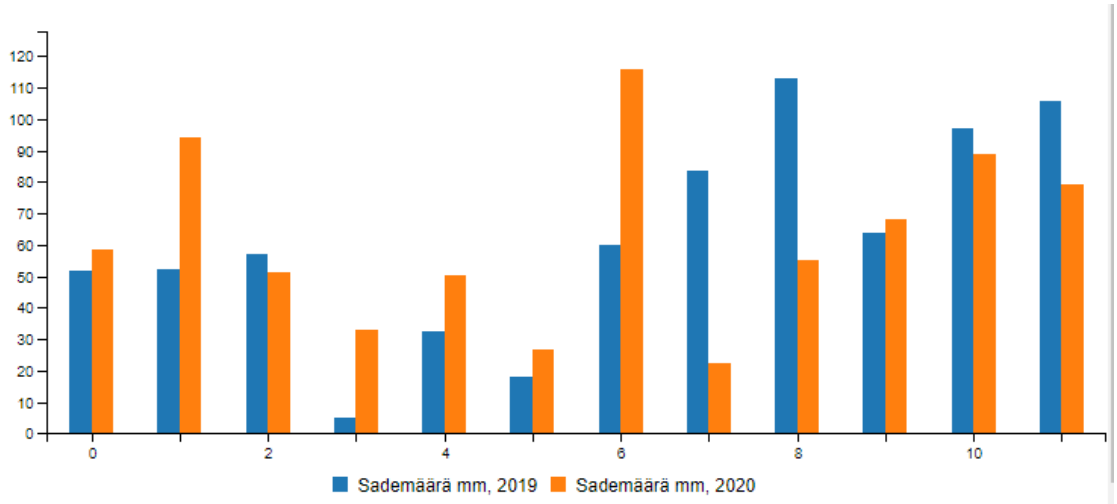


Kuva 3 Chart.js pylväsdiagrammi peruskonfiguroinnilla (Liite 1).

Kirjasto on edelleen aktiivisessa kehityksessä ja sen uusin versio 3.6.0 on julkaistu lokakuussa 2021. Chart.js tarvitsee toimiakseen ainoastaan kirjaston oman JS-tiedoston, muokkaukset kaavioiden tyyleihin tehdään suoraan JavaScriptin avulla. (Chart.js 2021.)

2.3.2 Billboard.js

Billboard.js (Kuva 4) on julkaistu vuonna 2017. Julkaisun jälkeen se on saanut yli 100 uutta versiopäivitystä vuosien varrella. Kirjasto on rakennettu D3-kaaviokirjaston pohjalle ja vaatii oman kirjastonsa lisäksi sen lataamisen.



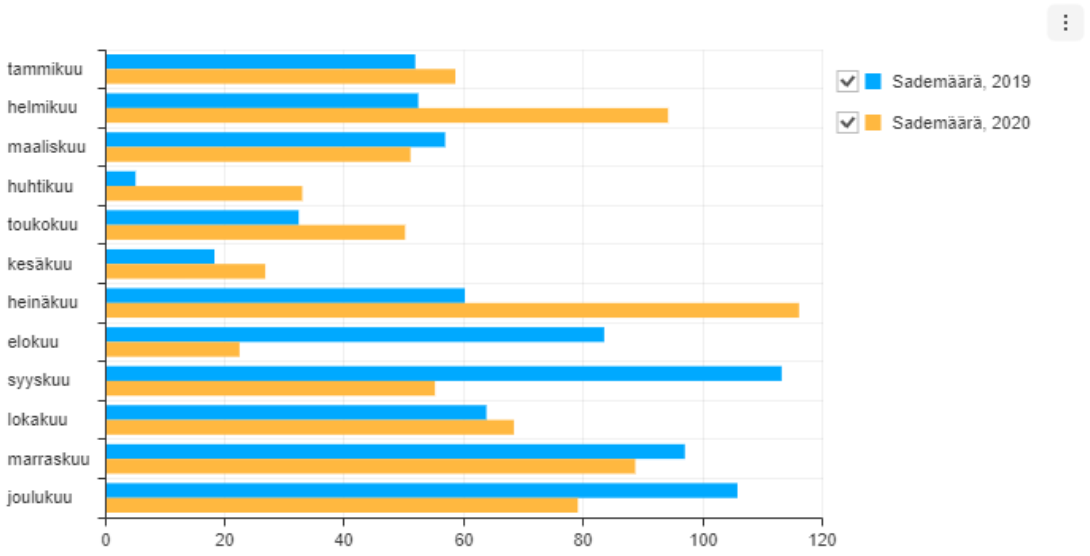
Kuva 4 Billboard.js pylväsdiagrammi peruskonfiguroinnilla (Liite 1).

Tuettuina on yli 20 erilaista kaaviota ja esimerkiksi viiva- ja pylväskaavioita voidaan yhdistää yhdeksi. (Billboard.js 2021.)

2.3.3 TOAST UI Chart

TOAST UI Chart (Kuva 5) on julkaistu vuonna 2015 ja on vertailun kirjastoista suurikokoisin 600 kilotavun tiedostokoollaan. Kirjasto on osa suurempaa Toast UI kokoelmaa, joka sisältää selaimessa toimivia työkaluja kuten kuvaeditorin ja dynaaminen kalenterin. Kaavioita voidaan kuitenkin käyttää itsenäisesti ilman

muita osia. Myös tämä kirjasto on edelleen kehityksessä



Kuva 5 TOAST UI Chart pylväsdiagrammi peruskonfiguroinnilla (Liite 1).

Itse JS kirjastotiedoston lisäksi tulee asentaa CSS-tyylitiedosto, jonka avulla visualisointia säädetään. Tuettuina on 18 eri kaaviotyyppiä ja perinteisen kaaviotyyppien lisäksi mukana on mm. bubble, scatter, radar ja heatmap kaavioita. Tuettuna on myös datan vieminen joko kuvana, CSV tiedostona tai suoraan Excel muodossa. (TOAST UI chart 2021.)

2.4 Suorituskyvyn optimointi

Kehityksen loppuvaiheessa käyttöliittymä optimoidaan tuotantokäyttöä varten parhaan suorituskyvyn takaamiseksi. Optimoinnin tarkempi toteutus käsitellään luvussa **Error! Reference source not found.**. Keskeisimpänä työkaluna on Googlen kehittämä Lighthouse, jonka raportin avulla suoritetaan suorituskyvyn optimointi.

2.4.1 Google Lighthouse

Lighthouse on Googlen kehittämä avoimen lähdekoodin työkalu, joka tarkastaa web-sivuston suorituskyvyn, saavutettavuuden ja hakukoneoptimoinnin

kannalta, ja luo raportin sen pohjalta. Raportissa on mukana korjausehdotuksia sekä arviot niiden vaikutuksista suorituskykyyn. Lighthouse on esiasennettuna Googlen Chrome-selaimen kehittäjänasetuksissa. (Google 2021.)

2.4.2 Minify

Minify on komentoriviltä suoritettava työkalu, jonka avulla voidaan minifioida CSS ja JS -tiedostoja. (Minify 2021).

Minifioinnilla tarkoitetaan menetelmiä, joilla koodin viemää tilaa pienennetään. Html, CSS ja JS koodia kirjoittaessa käytetään yleisesti välilyöntejä, rivivaihtoja, kommentteja sekä JS-tiedostojen kohdalla kuvaavia funktio- ja muuttujanimiä. Koodi on ihmissilmälle helppolukuista ja jo tehdyn muokkaaminen on helppoa. Selaimen suorituksen kannalta näistä luettavuutta parantavista tekijöistä ei kuitenkaan ole mitään hyötyä. Koodi vie enemmän tilaa, mikä pidentää latausaikoja. (Imperva 2021.)

Minifioidussa JS tiedostossa kaikki koodi on yhdellä rivillä, välilyönnit ja kommentit on poistettu ja funktioiden muuttujanimet ovat korvattu yksittäisillä kirjaimilla (Kuva 6).

```
!function(t,e){"object"==typeof exports&&"undefined"!=typeof module?module.exports=e():"function
```
…

Kuva 6 Katkelma minifioitua Bootstrap_bundle_min.js tiedostoa.

2.4.3 PurgeCSS

PurgeCSS on työkalu, jonka avulla voidaan poistaa sivuston käyttämättömiä CSS-tyylisäätöjä. PurgeCSS analysoi html-tiedostot ja CSS-tyylitiedostot ja vertailemalla niissä esiintyviä valitsimia poistaa tarpeettomat tyylimuokkaukset ja luo uuden tyylitiedoston. (PurgeCSS 2021.)

Kun kehityksessä otetaan käyttöön ulkoisia kirjastoja ja tyylimuokkauksia, jää niiden tarjoamista ominaisuuksista helposti osa käyttämättä. Tämän seurauksena asiakaslaitteella joudutaan lataamaan ulkoasuun vaikuttamattomia

CSS-tyylimuokkauksia, mikä hidastaa turhaan latausnopeutta ja lisää palvelinkuormitusta. Siksi niiden puhdistaminen on kannattavaa. (Arsenault 2019.)

2.4.4 Gzip

Gzip on deflate-algoritmiin perustuva tiedostonpakkausmenetelmä, jonka avulla voidaan pakata palvelimelta lähetettävät resurssit. Modernit selaimet tukevat sitä laajasti. (MDN 2021f.)

Tekstipohjaisten resurssien pakkaaminen on yksinkertainen, mutta erittäin tehokas tapa nopeuttaa resurssien ja sivuston latautumista kokonaisuudessaan. Mikäli käyttäjän selain ei tue palvelimen pakkausalgoritmia palauttaa palvelin resurssit pakkaamattomina. Näin voidaan käyttää useampia teknologioita ja esimerkiksi uudemmat algoritmit eivät aiheuta ongelmia. Loppukäyttäjän kannalta ei ole muuta havaittavaa vaikutusta kuin nopeammat sivulataukset verrattuna pakkaamattomaan koodiin sekä hiukan kohonnut prosessointitehon tarve ensilatauksella, koska resurssit täytyy purkaa ennen kuin niitä voidaan hyödyntää. (MDN 2021e.)

3 Toimeksianto ja vaatimukset

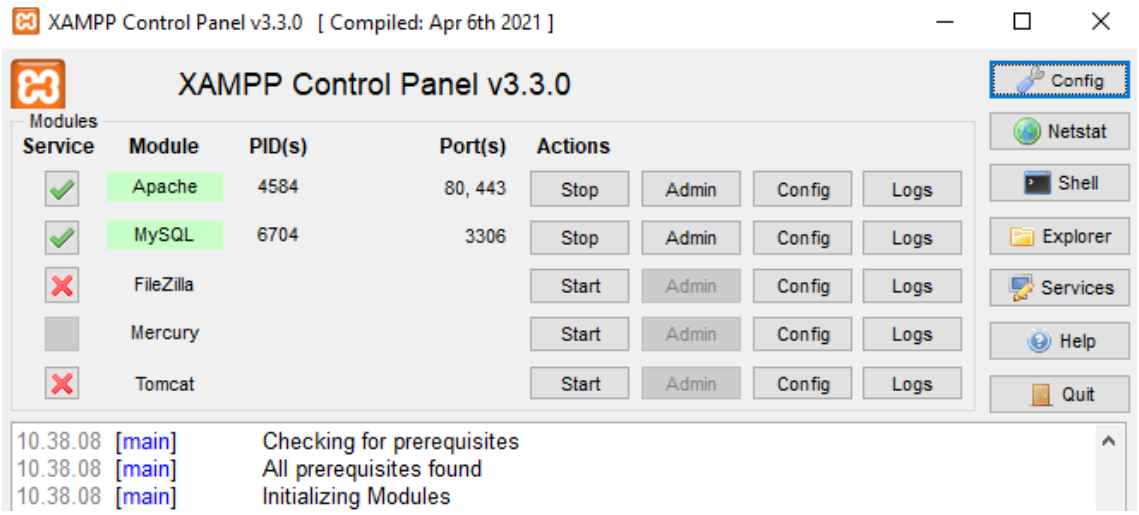
Toimeksiantona oli vertailla tuotantodatan visualisoinnissa hyödynnettäviä visualisointikirjastoja kehittää web-käyttöliittymä. Kehitykselle on asetettu tässä luvussa mainitut reunaehdot, mutta muuten kehityksen suunta ja valitut teknologiat ovat vapaasti valittavia.

3.1 Kehitysympäristö

Käyttöliittymän suunnittelu, kehitys ja testaus suoritetaan Windows-ympäristössä käyttämällä Google Chrome -selainta. Visualisoitava, tietokannassa sijaitseva data haetaan aiemmin kehitetyn PHP-rajapinnan kautta, joka palauttaa halutun datan JSON-muodossa. Kehityksessä hyödynnettävien teknologioiden tulee täyttää luvuissa 3.2 ja 3.3 asetetut vaatimukset.

Kehitysympäristössä käytetään XAMPP-ohjelmistoa, joka tarjoaa vaaditun web-palvelimen sekä tietokantapalvelimen yhdessä paketissa kehitysympäristöön optimoidulla asetuksilla. Web-palvelin sekä tietokanta voidaan käynnistää ohjelman avulla helposti ”start” painiketta painamalla tai ne voidaan asentaa käynnistymään Windows-palveluina tietokoneen käynnistyessä (Kuva 7).

Tuotantokäytön testaus tapahtuu niin ikään Windows ympäristössä Chrome-selaimella.



Kuva 7 XAMPP-käyttöliittymä.

3.2 Käyttöliittymän vaatimukset

Koska käyttöliittymä on tarkoitus ottaa kaupalliseen käyttöön, tulee kaikkien kehityksessä hyödynnettävien teknologioiden ja työkalujen lisenssien sallia kaupallinen käyttö. Mikäli mahdollista, hyödynnetään kehityksessä avoimen lähdekoodin ohjelmistoja. Kehitykselle on annettu kokonaisbudjetti, 1 000 €, joka kattaa kaikki kehityksen osa-alueet suunnittelusta käyttöönottoon. Ulkoasun suunnittelussa käytetään pohjana yrityksen kotisivujen värejä ja tyylejä. Käyttöliittymän tulee toimia kaikilla moderneilla selaimilla niiden tuetuilla versioilla Windows, Linux ja Mac OS ympäristöissä sekä Android ja IOS mobiililaitteilla.

3.3 Datavisualisoinnin vaatimukset

Visualisoinnissa hyödynnettävän JS-kirjaston valinta on perusteltava. Sen valinnassa tulee huomioida tuetut kaaviotyypit: Tuettuna tulee olla ainakin pylväs- ja viivadiagrammeja sekä mahdollisesti molempia hyödyntäviä yhdistelmäkaavioita. Pieni kirjaston koko sekä suorituskyvyn säilyminen sujuvana tilanteissa, jossa on paljon yhtäaikaisesti visualisoitavaa dataa,

huomioidaan positiivisina ominaisuuksina. Kirjaston tulee lisäksi tukea dynaamisesti muuttuvia datasettejä.

Kun tapahtumadatan visualisointia lähdetään kehittämään, on otettava huomioon suuri visualisoitavan datan määrä. Uusia tapahtumia voi tulla kymmeniä tuhansia muutaman tunnin sisällä, joten datan tehokas hyödyntäminen vaatii sarakekohtaisen datan järjestelyn, hakuominaisuuden sekä suodatusmahdollisuuden.

3.4 Ympäristömuuttujien vaikutukset suorituskykymittauksiin

Jotta suorituskykymittaukset olisivat mahdollisimman vertailukelpoisia keskenään, pyritään ympäristöstä johtuvia suorituskykyyn vaikuttavia tekijöitä torjumaan. Suorituskykymittaukset suoritetaan käyttämällä selaimen ”yksityinen selaus” tilaa. Tämä poistaa selaimeen asennettujen lisäosien vaikutuksen, sekä mahdolliset välimuistissa pyörivät ylimääräiset resurssit. Kaikki testaamisen kannalta tarpeettomat ohjelmat ja selainikkunat suljetaan mittausten ajaksi. Tämän lisäksi mittaukset optimoinnin jälkeen suoritetaan välittömästi ennen optimointia vedettyjen mittausten jälkeen.

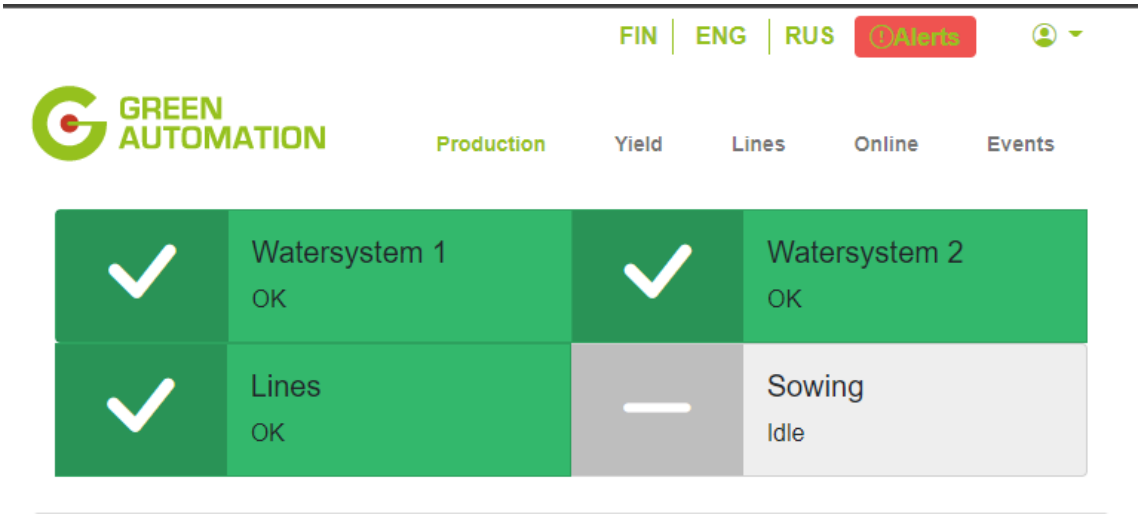
4 Käyttöliittymän kehitys

4.1 Suunnittelu ja toteutus

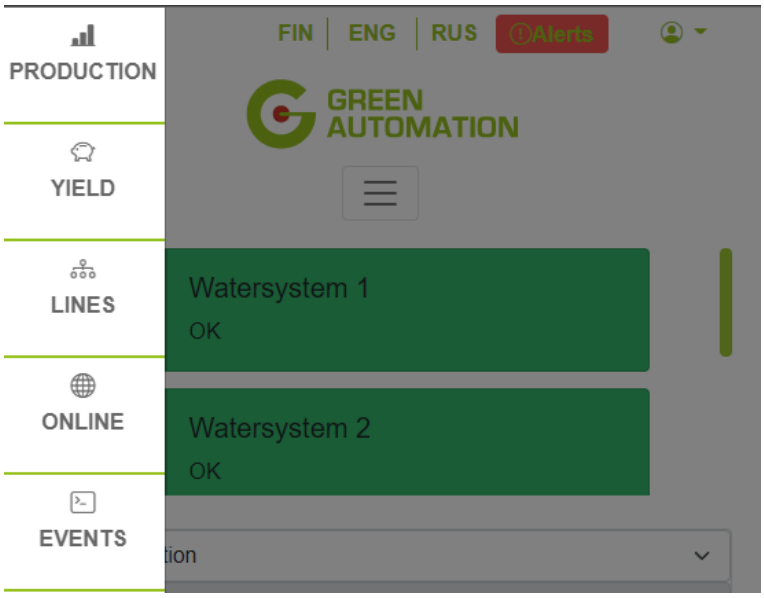
Teknologioiden valinnoissa lähdettiin ensisijaisesti hakemaan avoimen lähdekoodin ratkaisuja, jotka on julkaistu MIT-lisenssin alla. Näin voidaan varmistaa sujuva kehitystyö ilman lisenssiehtojen rikkomisen vaaraa. Bootstrapin ruudukkojärjestelmä, ”mobile first” periaate sekä valmiit ja muokattavat komponentit navigoinnille luovat hyvän pohjan responsiiviselle kehitykselle. Sen avulla kehityksessä voidaan keskittyä varsinaiseen käyttöliittymän ominaisuuksien kehittämiseen ja käyttöliittymän testaamiseen menee vähemmän aikaa.

Tavoitteena oli luoda yksinkertainen ja helppokäyttöinen, yrityksen kotisivuja mukaileva käyttöliittymä. Navigointi sekä asetusten hallinta päätettiin jakaa kahteen osaan. Ylimpänä osana toimii hallintapalkki, joka on näkyvissä niin mobiililaitteilla kuin laajakuvanäytöillä. Hallintapalkissa sijaitsevat kielivalinnat, järjestelmähälytysten avauspainike sekä tässä kehitysvaiheessa hyödyntämätön käyttäjävalikko. Hallintapalkin alapuolella on navigointipalkki, joka mukautuu selainikkunan leveyden mukaan (Kuva 8). Mobiililaitteilla navigointipalkki korvataan painikkeella, jota painamalla aukeaa Bootstrapin offcanvas-elementillä toimiva sivupalkki sivun vasemmasta reunasta (Kuva 9). Palkki sulkeutuu automaattisesti, jos palkin ulkopuolista tilaa painetaan tai jos sivua vaihdetaan palkin painikkeista.

Seuraavaksi vuorossa on statuspalkki. Statuspalkin sisältö vaihtelee sivun mukaan, mutta sen päätarkoituksena on tarjota keskeisin tieto -tai hallintaominaisuuksia käyttäjälle. Esimerkiksi etusivulla siinä sijaitsevat eri järjestelmien tilaa ilmaisevat kortit (kuva 8) kun taas tuottosivulla palkista löytyy päivän sadon ilmaisevat kortit.



Kuva 8 hallintapalkki, navigointi ja statuskortit.



Kuva 9: Bootstrap offcanvas palkkina toteutettu mobiilinavigointi.

Varsinainen sisältö on div-elementin sisällä, ja sen asettumisessa hyödynnettiin Bootstrap-ruudukkojärjestelmää. Sisällön jälkeen näkymän pohjalla sijaitseva alatunniste, josta löytyy mm. linkkejä käyttöehtoihin, tukeen sekä yrityksen muihin palveluihin.

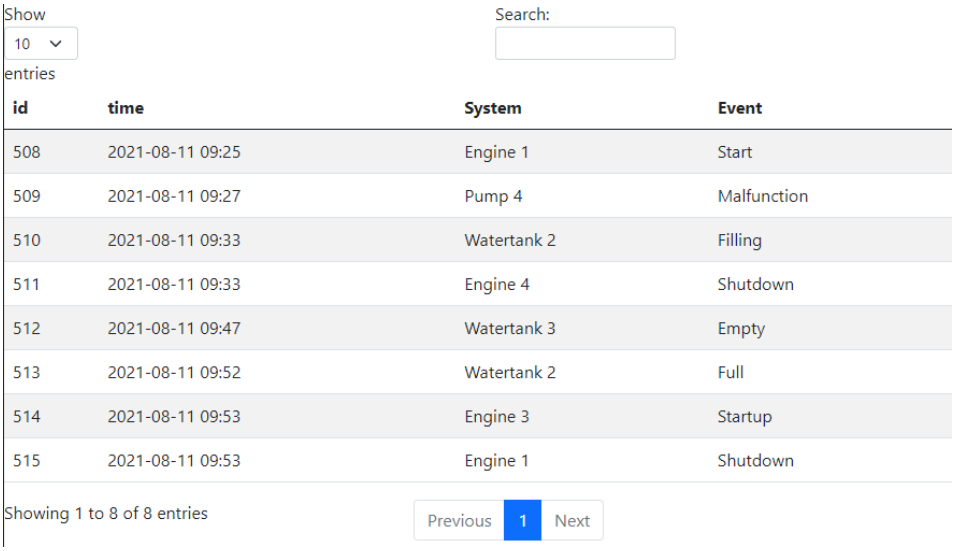
4.2 Tapahtumadatan visualisointi: DataTables

Tapahtumadatan visualisoinnissa tuli vastaan haasteita. Muutaman tunnin aikana tuli jopa kymmeniä tuhansia uusia tapahtumia riippuen kasvatusjärjestelmän laajuudesta. Käsittelemättömillä html-taulukoilla ei voida tehdä käytännössä mitään muuta kuin näyttää tuloksia riveittäin (Kuva 10), ja omien ratkaisujen kehittäminen ei olisi mielekästä ajankäytön kannalta.

id	time	System	Event
515	2021-08-11 09:53	Engine 1	Shutdown
514	2021-08-11 09:53	Engine 3	Startup
513	2021-08-11 09:52	Watertank 2	Full
512	2021-08-11 09:47	Watertank 3	Empty
511	2021-08-11 09:33	Engine 4	Shutdown
510	2021-08-11 09:33	Watertank 2	Filling
509	2021-08-11 09:27	Pump 4	Malfunction
508	2021-08-11 09:25	Engine 1	Start

Kuva 10 Tavallinen html-taulukko.

Tämän vuoksi haettiin mahdollisia valmiita ratkaisuja. Koska DataTables oli jo ennestään tuttu ja toimivaksi todettu, valikoitui se tähän mukaan. DataTables tukee peruskonfiguroinnilla sivunumerointia sekä sivukohtaista rivimäärää ja sarakekohtaista lajittelua. Tämän lisäksi käytössä on kaikkien sarakkeiden yhteinen hakupalkki, josta voidaan hakea yhdistämällä sanoja useasta sarakkeesta halutunlainen lopputulos (DataTables 2021d.) (Kuva 11)


Show 10 entries

Search:

id	time	System	Event
508	2021-08-11 09:25	Engine 1	Start
509	2021-08-11 09:27	Pump 4	Malfunction
510	2021-08-11 09:33	Watertank 2	Filling
511	2021-08-11 09:33	Engine 4	Shutdown
512	2021-08-11 09:47	Watertank 3	Empty
513	2021-08-11 09:52	Watertank 2	Full
514	2021-08-11 09:53	Engine 3	Startup
515	2021-08-11 09:53	Engine 1	Shutdown

Showing 1 to 8 of 8 entries

Previous 1 Next

Kuva 11 DataTables html-taulukko.

Vaikka tavallisiin html-taulukoihin nähden uudet ominaisuudet parantavatkin tietojen saatavuutta, on tapahtumadatan virrasta edelleen vaikea löytää spesifejä tuloksia. DataTables tukee virallisesti lisäosia, jotka parantavat sen olemassa olevia ominaisuuksia tai tuovat kokonaan uutta toiminnallisuutta (DataTables 2021b). Suuren datamäärän takia haettiin tiedon suodattamiseen soveltuvia lisäosia.

Käyttämällä DataTables kehittäjän luomaa livekoodiesimerkkiä saatiin käyttöön sarakekohtainen suodatus, jonka avulla pystyttiin valitsemaan pudotusvalikosta kustakin sarakkeesta haluttu arvo ja muiden sarakkeiden pudotusvalikoiden lista päivittyy vastaamaan asetetun suodattimen kanssa yhteneviä arvoja (DataTables 2020.) Kuvassa 12 on valittu järjestelmän 2(Unit_id) mikseri (Line/ID), jonka vuoksi tapahtumalistassa (Event) näkyy kyseisiä arvoja vastaavat ”on/off” tapahtumat. Tämän lisäksi otetiin käyttöön Buttons-liitännäinen, joka tarjoaa taulukon latauksen CSV,Excel ja PDF muodoissa.

Kuva 12 pudotusvalikot sarakekohtaiselle suodatukselle.

Tapahtumadatan kanssa ilmeni merkittävää suorituskyvyn heikkenemistä tapahtumamäärän noustessa 100 tuhannen rivin tuntumaan. Datan lataaminen oli edelleen nopeaa, mutta sen prosessoinnissa oli havaittavissa merkittäviä viiveitä. Jokainen sivunvaihto kesti useamman sekunnin, vaikka jokaisella sivulla oli vain 10 riviä. Sarakekohtaisen järjestyksen muuttaminen aiheutti vielä pidemmän viiveen, ollen lähes käyttökelvoton yli kymmenen sekunnin latausajalla.

Ratkaisuna otettiin käyttöön DataTables asetus ”defer render”. Kun asetus on käytössä, ainoastaan sen hetkisellä taulukon sivulla näkyvä data ladataan muistiin ja seuraavat tulokset ladataan dynaamisesti tarpeen mukaan. (DataTables 2021c.)

4.3 JS-kirjastovertailu

Kirjastojen vertailuperusteina toimivat luvussa 3.1 määriteltyjen vaatimusten lisäksi toimivien kaavioiden vaatima määrittelyn määrä, kirjaston oma dokumentointi sekä saatavilla oleva tuki verkossa ja Stack Overflow sivustolla.

Projektin kannalta optimaalinen kirjasto on julkaistu MIT-lisenssin alla, on pienikokoinen yleisen suorituskyvyn kannalta ja on edelleen tuettuna ja

kehityksessä, jotta kirjastossa esiintyvät ohjelmavirheet korjataan ja mahdollisia uusia ominaisuuksia voidaan hyödyntää tulevaisuudessa.

Potentiaalisia visualikirjastoja kartoittaessa yleisimmät syyt kirjastojen karsimiseen olivat tiukat lisenssiehdot ja niihin liittyvät korkeat hinnat sekä vaatimukset ylittävät ominaisuudet, jotka paisuttivat kirjaston koon turhan suureksi tai lisäsivät riippuvuuksia ulkoisiin kirjastoihin. Rajauksen jälkeen valikoitui kolme kirjastoa, jotka otettiin tarkempaan vertailuun mukaan. Kaikki kirjastot ovat julkaistu MIT-lisenssin alla, eli ne tarjoavat erittäin vapaan käytön, kunhan lisenssitiedot ovat mukana kirjaston koodissa.

Kaavioiden luonnissa ja visuaalista ilmettä hahmottavissa kuvissa on käytetty Turun Artukaisista olevaa sadekertymädataa vuosilta 2019-2020 (Kilotavu 2021).

Taulukko 1 Vertailtavat JS-kirjastot.

	Chart.js	Billboard.js	Toast UI Chart
Paketin koko^1	157kt	490kt	600kt
GitHub tähtiä^2	55000+	5000+	5000+
Lisenssi	MIT	MIT	MIT
Versio^3	3.6.0 [23.10.2021]	3.2.1[22.10.2021]	4.4.1 [22.10.2021]

^1 Kaikki vaaditut resurssit, minifioituna ^2 Tilanne 25.10.2021 Github.com palvelussa ^3 Uusin versio kirjoitushetkellä, Tilanne 25.10.2021

Taulukon 1 pohjalta nähdään, että vertailtavien kirjastojen koko vaihtelee melko paljon, mutta Chart.js oli selvästi pienikokoisin 157kt tiedostokoollaan. Se oli myös selvästi suosituin ja sillä oli yli kymmenenkertainen määrä tähtiä GitHub-palvelussa. Kaikki kirjastot ovat edelleen aktiivisessa kehityksessä ja koska ne on julkaistu MIT-lisenssin alla tarjoavat ne käytännössä rajattoman käytön.

Vaikka kaaviot vaikuttavat päällisin puolin samanlaisilta (Liite 1), niin pienet erot nostivat TOAST UI Chart ja Chart.js kirjastot Billboard.js kirjaston edelle. Chart.js toimii suoraan pelkästään omalla JS-tiedostollaan, eikä erillistä tyylitiedostoa vaadita tyylien muokkaamiseen. Billboard.js vaati oman

kirjastotiedoston ja tyylitiedoston lataamisen lisäksi erillisen D3-kirjaston lataamisen toimiakseen.

Kaavioiden luonti toimii hyvinkin samalla tavalla kaikissa kirjastoissa. Chart.js käyttää html canvas-elementtejä, kun taas Billboard.js ja TOAST UI Chart käyttävät perinteistä div-elementtiä. Canvas-elementin käyttö on visualisoinnin kannalta perusteltua, sillä se on luotu dynaamista grafiikkaa ja ajatellen (Adobe 2021).

Yksikään vertailluista kirjastoista ei kohonnut selvästi ylitse muiden. Kaikissa oli kuitenkin jokin asia toteutettu paremmin kuin muissa. TOAST UI Chart tarjosi toisaalta kattavimmat ominaisuudet, mutta samalla niiden kustomoitavuus oli hankalaa. Se oli vertailtavista kirjastoista ainoa, joka tuki kaavioiden datan lataamista. Billboard.js oli kirjastoista helppokäyttöisin ja perusolomuodoltaan yksinkertaisin, mutta silti suurikokoisin tiedostokooltaan.

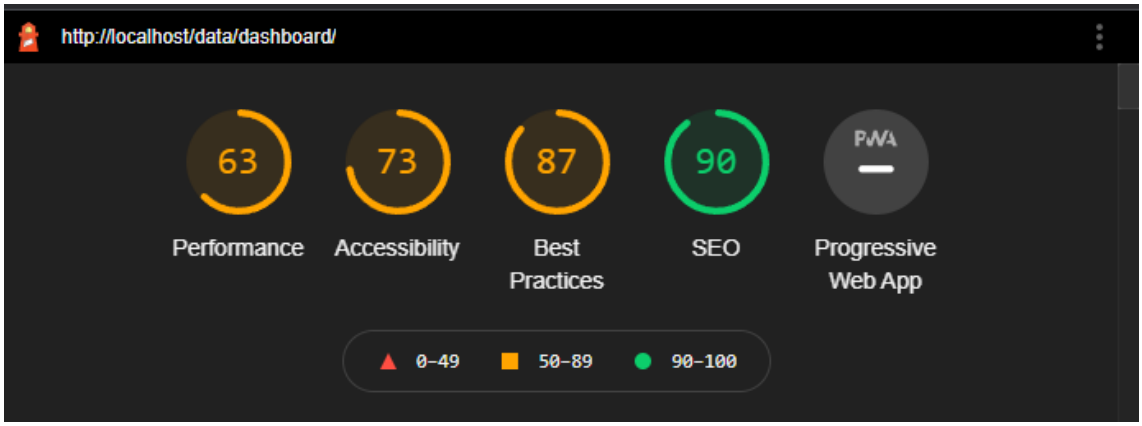
Valinta kohdistui lopulta Chart.js-kirjastoon, joka oli vertailtavista kirjastoista selvästi suosituin latausmäärinen perusteella. Toast UI Chart oli pitkään ensisijaisena valittavana kirjastona. Sen tarjoama datan latausominaisuus ei kuitenkaan ollut riittävän laaja ja käyttöliittymän vaatimusten perusteella sitä käytettäessä olisi tarvinnut kehittää joka tapauksessa erillisen latausominaisuus. Netistä löytyvää apua kehityksen ongelmatilanteisiin oli myös selvästi vähemmän saatavilla kuin Chart.js kirjastossa, jonka laajasta dokumentoinnista ja yhteisön tuesta voi olla erityisesti tulevaisuudessa hyötyä, mikäli visualisointia laajennetaan. Chart.js oli myös vertailtavista kirjastoista kevyin niin tiedostokooltaan kuin suorituskyvyltään. Tästä oli selvää hyötyä, kun käyttöliittymä asennettiin asiakasyrityksen heikommille laitteille.

5 Suorituskyvyn optimointi

Kehityksen loppuvaiheessa käyttöliittymä optimoidaan suorituskyvyn näkökulmasta. Käytännössä tämä tarkoittaa toimia, joilla pienennetään palvelimen asiakaslaitteille lähettämien resurssien kokoa, jolloin hitailla yhteyksillä saavutetaan käyttöliittymännopeampi latausaika. Optimoinnin tukena hyödynnetään selaimeen sisäänrakennettuja kehittäjäntyökaluja.

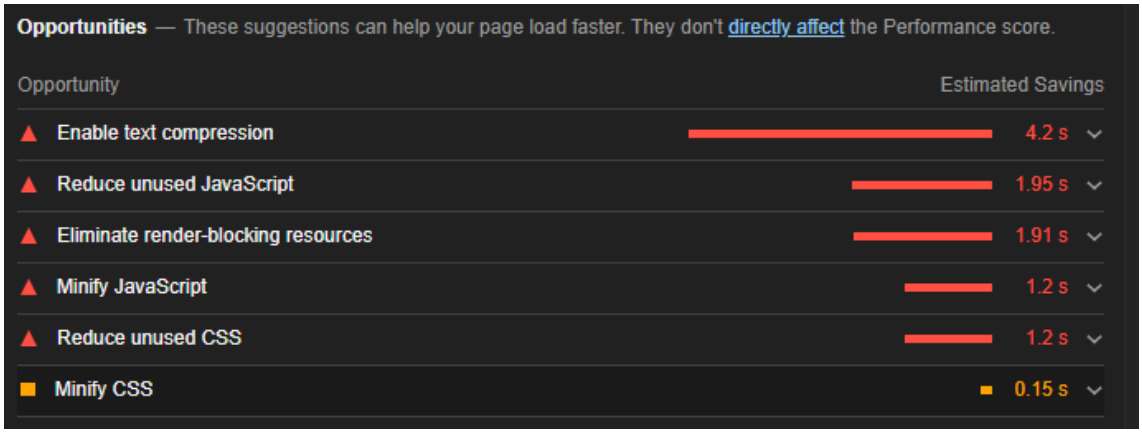
5.1 Lighthouse raportti

Optimoinnin lähtötilanne kartoitetaan Lighthousen raportointitoiminnalla. Tämä saadaan suoritettua avaamalla Chromen kehittäjänasetukset painamalla F12 ja valitsemalla valikosta Lighthouse ja ”Generate report”. Lighthouse raportin perusteella nähdään, että suorituskyvyssä on paljon optimoitavaa (Kuva 13). Muut optimointikohteet käsitellään tämän opinnäytetyön ulkopuolella.



Kuva 13 Lighthouse raportin yleiskatsaus.

Kun raportista avataan suorituskykyvälilehti (engl. Performance), saadaan lista tärkeimmistä suorituskykyyn vaikuttavista tekijöistä (Kuva 14), jotka käydään seuraavaksi läpi.



Kuva 14 Suorituskyvyn optimointikohteet.

5.2 Tekstipohjaisten resurssien pakkaaminen (Enable text compression)

Tekstipohjaisten resurssien pakkaaminen saadaan käyttöön eri tavoilla, riippuen käytössä olevasta web-palvelimesta. Perusperiaate on kuitenkin sama, eli muokkaamalla palvelinasetuksia tai ottamalla jokin moduuli käyttöön.

Apache palvelimella pakkaaminen saadaan käyttöön ottamalla ensin deflate-moduuli käyttöön konfigurointitiedostossa httpd.conf poistamalla rivin edestä kommenttia ilmaiseva merkki # (Tutorials24x7 2019.)

LoadModule deflate_module modules/mod_deflate.so

Tämän jälkeen voidaan ottaa pakkaaminen erikseen käyttöön halutuille tiedostotyypeille lisäämällä konfigurointitiedoston loppuun moduulin asetukset (Tutorials24x7 2019.)

<IfModule mod_deflate.c>

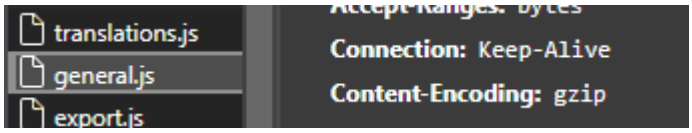
AddOutputFilterByType DEFLATE application/javascript

AddOutputFilterByType DEFLATE text/css

</IfModule>

Palvelinasetusten muokkaamisen jälkeen käynnistetään muutosten aktivoimiseksi palvelin uudestaan. Tulokset voidaan tarkastaa Chromen

kehittäjänasetusten verkkovälilehdestä valitsemalla jokin tekstipohjainen resurssi. Katsomalla kohtaa ”Content-Encoding” nähdään, että resurssien pakkaaminen on käytössä gzip-algoritmilla (Kuva 15).



Kuva 15 General.js on pakattu Gzip-menetelmällä.

5.3 JS/CSS Minifiointi (Minify JavaScript/CSS)

Minifioinnissa käytetty NPM-moduuli Minify asennetaan NPM-palvelusta komennolla:

```
npm install minify -g
```

Tämän jälkeen suoritetaan komentoriviltä minify-komento ja valita haluttu tiedosto sekä minifioidun tiedoston nimi:

```
minify style.css > style.css.min
```

(Minify 2021.)

style.css tiedoston koko laski minifioinnin avulla 38 %. Komento suoritettiin kaikille CSS/JS tiedostoille, joita ei ollut pakattu. Monet kehityksessä hyödynnetyt kirjastot tarjosivat jo valmiiksi minifioidut tiedostot.

5.4 CSS-puhdistus (Reduce unused CSS)

CSS-tiedostojen puhdistus suoritetaan komentoriviltä käyttämällä purgeCSS työkalun komentoa. PurgeCSS asennetaan NPM-palvelusta komennolla: (PurgeCSS 2021.)

```
npm install purgecss -g
```

Suorittamalla komentorivillä seuraava koodipätkä, puhdistetaan Bootstrap Icons-tiedosto:

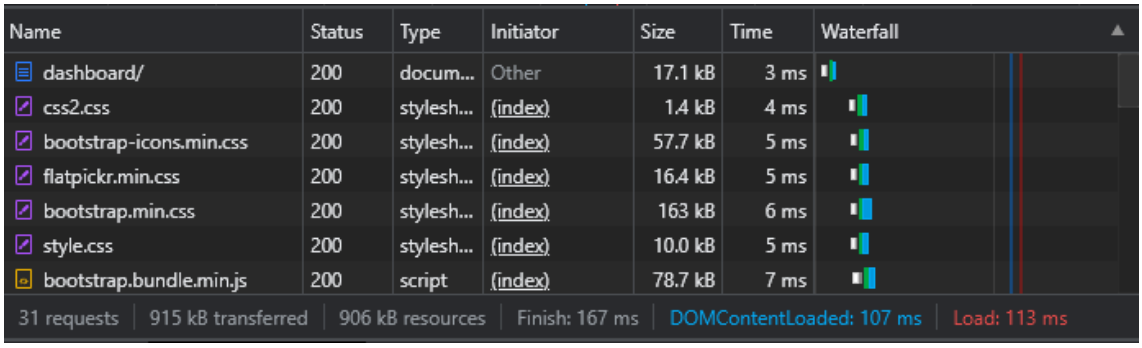
```
purgecss --css .\MINIFIED\bootstrap-icons.min.css --content
.\index.html ".\**/*.js" --output .\PURGED\
```

Bootstrap Icons tarjoaa lukuisia vapaasti käytettäviä ikoneita. Niistä kuitenkin vain pieni osa on käytössä tämän käyttöliittymän kehityksessä, eli suurin osana mukana toimitettavista ikoneista vie turhaa tilaa. PurgeCSS puhdistuksen jälkeen tiedostokoko putoaa 1.57 kilotavuun eli 2.8 prosenttiin alkuperäisestä 57.5kt tiedostokoosta. Sama toimenpide suoritettiin muille CSS-tiedostoille.

5.5 Optimoinnin tulokset

Suorituskyvyn mittaukset otetaan välittömästi ennen optimointia ja optimoinnin jälkeen käyttöliittymän etusivulta. Mittauksissa käytettään Chrome-selaimen kehittäjän asetusten verkkovälilehteä (engl. Network), josta nähdään sivulatauksen aikana tehdyt resurssikyselyt (engl. Requests), siirrettyjen resurssien koko, resurssien todellinen koko sekä kokonaislatausaika (Kuva 16).

Etusivulla ladataan suurin osa käyttöliittymän tarvitsemista staattisista resursseista, joka antaa parhaan kuvan optimoinnista.



Name	Status	Type	Initiator	Size	Time	Waterfall
dashboard/	200	docum...	Other	17.1 kB	3 ms	
css2.css	200	stylesh...	(index)	1.4 kB	4 ms	
bootstrap-icons.min.css	200	stylesh...	(index)	57.7 kB	5 ms	
flatpickr.min.css	200	stylesh...	(index)	16.4 kB	5 ms	
bootstrap.min.css	200	stylesh...	(index)	163 kB	6 ms	
style.css	200	stylesh...	(index)	10.0 kB	5 ms	
bootstrap.bundle.min.js	200	script	(index)	78.7 kB	7 ms	

31 requests | 915 kB transferred | 906 kB resources | Finish: 167 ms | DOMContentLoaded: 107 ms | Load: 113 ms

Kuva 16 Kehittäjänasetukset: resurssien koko ja latausajat.

Optimoinnista oli hyötyä resurssien kulutuksessa välimuistia käytettäessä. Ennen optimointia resursseja ladattiin 30.7kt kun taas optimoinnin jälkeen resursseja siirtyi puolet vähemmän, 15.3kt. (Taulukko 2) Keskimäärin latausajat

laskivat kuitenkin vain 7.25 % (Taulukot 2–3). Kun käytössä ei ollut selaimen välimuisti siirrettävien resurssien koko väheni vielä enemmän. Resursseja siirrettiin 64 % vähemmän kuin ennen optimointia ja niiden kokonaiskokokin oli vähentynyt 26 %.

Taulukko 2 Suorituskyky ilman välimuistia (Liitteet 2 ja 3).

	Kaikki valmiina	Ulkoasu valmis	Resursseja ladattu	Resurssit yhteensä
Lähtötaso	231,9ms	111,7ms	928,00kt	928,00kt
Optimointi	201,7ms	111,1ms	336,00kt	688,00kt
Muutos	-13 %	-1 %	-64 %	-26 %

Taulukko 3 Suorituskyky, välimuistin kanssa (Liitteet 2 ja 3).

	Kaikki valmiina	Ulkoasu valmis	Resursseja ladattu	Resurssit yhteensä
Lähtötaso	192,2ms	91,3ms	30,70kt	928kt
Optimoitu	180,3ms	83,1ms	16,30kt	688kt
Muutos	-6 %	-9 %	-47 %	-26 %

Vaikka testiympäristössä latausaika parani parhaimmillaan vain 13 % ilman välimuistia (Taulukko 2) ja 9 % välimuistin kanssa (Taulukko 3), voidaan yleisluonteisesti kuitenkin todeta, että suorituskykyparannukset olisivat todellisessa käyttöympäristössä selvästi suuremmat. Siirrettäessä resursseja verkon yli vaikuttaa tulokseen mm. palvelimen kuormitus, päätelaitteen verkon latausnopeus sekä palvelimen ja päätelaitteen välisen etäisyyden aiheuttama viive. Tällöin siirrettävien resurssien koko ja määrä ovat huomattavasti tärkeämmässä asemassa. testiympäristössä käyttöliittymän latausnopeuteen vaikutti lähinnä käytetyn laitteen nopeus.

6 Tulokset ja jatkokehitys

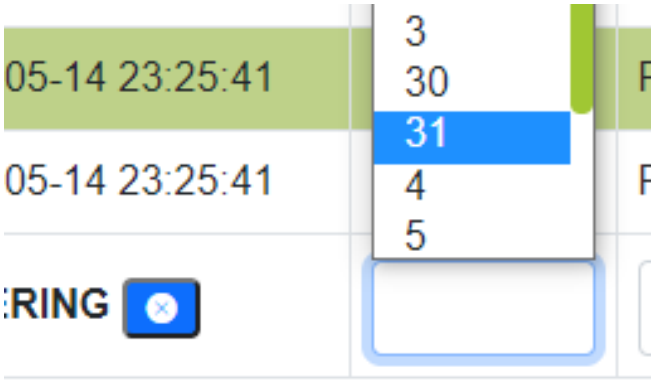
Opinnäytetyön tavoitteina oli vertailla tuotantodatan visualisoinnissa hyödynnettäviä visualisointikirjastoja ja kehittää vertailun perusteella valittua kirjastoa hyödyntävä web-käyttöliittymä. Kehityksen loppuvaiheessa käyttöliittymän resurssien käyttö oli tarkoitus optimoida suorituskyvyn maksimoimiseksi ja palvelimen resurssikuorman pienentämiseksi.

Lopputulos vastasi hyvin tavoitteita ja kehitykselle asetettuja vaatimuksia. Käyttöliittymä asennettiin paikallisena asennuksena asiakaslaitteelle ja muuttamalla yhteen tiedostoon koottuja ympäristömuuttujia se saatiin toimimaan ilman ongelmia. Käyttöliittymää pystyttiin käyttämään asiakkaan sisäverkossa olevilla laitteilla selaimen ja palvelimena toimivan laitteen IP-osoitteen avulla.

Käyttöliittymässä hyödynnetyt teknologiat toimivat hyvin keskenään. Bootstrapin avulla saatiin pienellä työmäärällä responsiivisesti skaalautuva käyttöliittymä ilmaan suurempia CSS-muokkauksia. Ainoastaan kaavioiden näkyvyyden kanssa tuli ongelmia mobiililaitteilla, sillä kaavioiden datapisteitä oli vaikea erottaa toisistaan pienellä ruudulla. Ongelmaa yritettiin korjata käyttämällä Chart.js kaaviokirjaston lisäosaa Chart.js zoom, mutta sen eleohjauksissa ilmeni ongelmia. Näkymä hyppi pienilläkin eleohjauksilla minimizoomauksesta maksimiin sekä vasemmasta laidasta oikeaan. Liitännäistä ei saatu toimimaan netistä löytyvillä ohjeilla. Vaihtoehtoisena ratkaisuna kaavioille asetettiin CSS min-width säädöllä minimileveys, jonka jälkeen kaavioita voidaan vierittää, jolloin data on helppolukuisempaa. Kirjasto vastasi toiminnaltaan hyvin vaatimuksia ja kaavioiden luonnissa ja muuttuvien datasettien kanssa työskentelyssä ei ilmennyt ongelmia. Laaja dokumentointi sekä hyvä yhteisön tuki vauhdittivat kehitystä. Kehityksessä käytetty kirjaston 3.5.1 versio toi kuitenkin lisähaasteita. Huhtikuussa 2021 julkaistu pääversio 3.0 toi lukuisia muutoksia kirjaston päätoimintoihin ja siirsi asetusfunktioiden sijaintia. Suurin osa netistä löytyvistä ohjeista on kuitenkin tehty vanhemmalle 2.X versioille, jonka alkuperäisestä julkaisusta on kulunut kirjoitushetkellä jo yli viisi vuotta.

Tämän seurauksena kirjaston dokumentointia jouduttiin selaamaan hyvinkin tarkkaan, sillä läheskään kaikkien muutosten kohdalla ei löytynyt perusdokumentoinnista selvää vastausta, vaan vasta edistyneiden käyttäjien api-dokumentoinnista.

Tapahtumadatan visualisoinnissa hyödynnetty DataTables saatiin toimimaan pienellä säädöllä hyvin. Suurilla datamäärillä latausajat kävivät kuitenkin pitkiksi, ja taulukon rivimäärän noustessa yli 100 tuhanteen suodattamisesta tuli käyttökelvottoman hidasta. Ratkaisuna kokeiltiin kirjaston oman dokumentoinnin ohjeistuksen mukaan siirtää prosessointi palvelinpuolelle. Tämä ei kuitenkaan ollut toimiva ratkaisu, sillä rajapinta ei tukenut datan käsittelyä. Suorituskyky saatiin lopulta järkevälle tasolle ottamalla kirjaston deferRender-asetus käyttöön, jonka avulla ainoastaan näkyvä taulukkodata piirrettiin. Tapahtumien suodatuksessa ilmeni myös ongelma, kun järjestelmä ei suostunut suodattamaan pelkästään numeroita sisältävää saraketta numeraalisesti, vaan korjausyrityksistä huolimatta suodatus tapahtui aakkosjärjestyksessä (Kuva 17). Tämän vuoksi numerot järjestyivät aina ensimmäisen numeron mukaan. Suodatus toimi siis oikein ainoastaan, kun tuloksissa oli käytössä numerot 1-9.



Kuva 17 Ohjelmavirhe numeroita sisältävässä suodatuslistassa.

Rajallisen kehitysajan vuoksi ongelmaa ei ehditty korjaamaan tämän opinnäytetyön kirjoittamisen aikana. Erittäin suurilla datamäärillä taulukon prosessointiajat kävivät myös sangen pitkiksi. Ratkaisuna tähän kokeiltiin prosessoinnin siirtämistä palvelinpuolelle. Käytetyn rajapinnan rajoittuneen takia

menetettiin samalla taulukon sarakekohtainen uudelleenjärjestys sekä suodatusominaisuudet.

Optimoinnilla onnistuttiin vähentämään palvelimen lähettämien resurssien kokoa ensilatauksella 64% ja välimuistin ollessa käytössä 47%. Latausajoissa ei havaittu merkittävää parannusta testiympäristössä, mutta jos käyttöliittymä otetaan joskus käyttöön verkossa toimivana palveluna, tulee pienentyneestä ladattavien resurssien määrästä olemaan hyötyä niin palvelimella vähentyneenä kuormituksena kuin loppukäyttäjälle sujuvampana käyttönä.

Tällä hetkellä käyttöliittymän sivua vaihdettaessa data joudutaan lataamaan uudestaan. Käytännössä tällä ei ole suurempaa merkitystä staattisten resurssien kanssa, jotka ladataan selaimen välimuistista. Rajapinnalta haettava data taas joudutaan lataamaan joka kerta sivua vaihdettaessa uudestaan. Esimerkiksi joka sivulla näkyvät järjestelmähälytykset joudutaan hakemaan uudestaan, riippumatta siitä, onko uusia hälytyksiä tullut. Tämä aiheuttaa turhia palvelinkutsuja ja lisää palvelimen kuormitusta. Paikallisesti asennettuna tästä ei ole paljonkaan haittaa, sillä sisäverkossa data liikkuu nopeasti, eikä liikenteestä kerry ylimääräisiä maksuja tai rasitteita. Tehokas tapa vähentää turhaa resurssien hakemista voisi olla käyttöliittymän muuttaminen yhden sivun sovellukseksi (Engl. Single Page Application).

Seuraavana kehitysaskeleena on käyttöliittymän siirtäminen palvelimelle. Tämä tuo kuitenkin mukanaan useita vaatimuksia mm. tietoturvan kannalta. Ensimmäisenä pitää kuitenkin valita käytettävä palvelinpuolen rajapinta. Käyttöliittymän data tulee PHP-rajapinnan kautta, eli varsinainen käyttöliittymä voitaisiin toteuttaa helposti samalla kielellä. Tällöin ei tarvittaisi ylimääräisiä teknologioita ja ylläpito olisi tulevaisuudessa yksinkertaisempaa. Paikallisen tietokantadatan siirtäminen verkon yli etäpalvelimelle vaatisi myös todennäköisesti VPN-tunnelin muodostamisen, jotta tiedot pysyisivät varmemmin salattuna.

Jotta optimoinnista olisi pitkällä tähtäimellä hyötyä, tulisi niiden toteutus automatisoida. Manuaalisesti toteutettu optimointi on aikaa vievää ja

vikaherkkää. Samalla pienten muutoksen tekemisen jälkeen resurssit on optimoitava aina uudestaan. Muutosten testaaminen ja uusien ominaisuuksien sovittaminen vanhaan toiminnallisuuteen vaatii myös paljon työtä. Tähän projektiin liittyen potentiaalinen automatisointityökalu on esimerkiksi Grunt. Grunt on julkaistu saman MIT-lisenssin alla kuin suurin osa kehityksen aikana käytetyistä työkaluista ja kirjastoista. Gruntin avulla luodaan tehtäviä (engl. Tasks), joiden avulla voidaan mm. yhdistävät eri tyylitiedostoja yhdeksi ja minifioivat CSS ja JS tiedostot. Grunt voi suorittaa myös erilaisia testejä, joiden avulla voidaan varmistaa, etteivät edellisen suorituksen jälkeen tehdyt muutokset ole rikkoneet mitään, ja että kirjoitettu koodi on validia.

Lähteet

Adobe 2021. Create HTML5 Canvas document in Animate. Viitattu 14.12.2021 https://helpx.adobe.com/fi/animate/using/creating-publishing-html5-canvas-document.html

Arsenaut, C. 2019. How to Remove Unused CSS for Leaner CSS Files. Viitattu 1.11.2021 https://www.keycdn.com/blog/remove-unused-css

Billboard.js 2021. Billboard.js Docs. Viitattu 16.10.2021 https://naver.github.io/billboard.js/release/latest/doc/index.html

Bootstrap 2021a. About Bootstrap v5.1. Viitattu 24.9.2021 https://getbootstrap.com/docs/5.1/about/overview/

Bootstrap 2021b. Bootstrap Grid system. Viitattu 22.11.2021 https://getbootstrap.com/docs/5.1/layout/grid/

Bootstrap 2021c. Reboot. Viitattu 22.11.2021 https://getbootstrap.com/docs/5.1/content/reboot/

Chart.js 2021. Chart.js Docs. Viitattu 24.10.2021 https://www.chartjs.org/docs/3.6.0/

DataTables 2021a. DataTables Manual. Viitattu 15.10.2021 https://datatables.net/manual/

DataTables 2021b. DataTables Plug-ins. Viitattu 16.11.2021 https://datatables.net/plug-ins/index

DataTables 2021c. Deferred rendering for speed. Viitattu 27.11.2021 https://datatables.net/examples/ajax/defer_render.html

DataTables 2021d. Installation: Initialising DataTables. Viitattu 13.12.2021 https://datatables.net/manual/installation

DataTables 2020. DataTables JS Bin. Viitattu 13.12.2021 http://live.datatables.net/gejojiqu/1/edit

Flores, R. 2021. Utilizing Bootstrap for ”mobile first” Development. Viitattu 22.11.2021 https://medium.com/swlh/utilizing-bootstrap-for-mobile-first-development-f85c8b65118a

Goldstein, A, 2021.Open Source Licenses explained. Viitattu 1.12.2021 https://www.whitesourcesoftware.com/resources/blog/open-source-licenses-explained

Google 2021. Lighthouse. Viitattu 6.10.2021 https://developers.google.com/web/tools/lighthouse

Imperva 2021.What is minification. Viitattu 1.12.2021 https://www.imperva.com/learn/performance/minification/

Kilotavu 2021. Turku Artukainen – Taulukkotilastot. Viitattu 29.11.2021 https://kilotavu.com/asema-taulukko.php?asema=100949

Koskinen, K. 2018. Automaatio ennen, nyt ja tulevaisuudessa. Viitattu 7.10.2021 https://www.automaatioseura.fi/site/assets/files/1380/automaatio_ennen_nyt_ja_tulevaisuudessa_av_artikkelisarja_2018.pdf

Lachman, D. 2016. A wordy history of default browser styles and CSS resets. Viitattu 13.10.2021 https://medium.com/@tinydinosaur/a-wordy-history-of-default-browser-styles-and-css-resets-befdd614d93b

Lawrence, M. 2019. What is a css framework. Viitattu 18.10.2021 https://medium.com/html-all-the-things/what-is-a-css-framework-f758ef0b1a11

MDN Web Docs 2021a. What is JavaScript. Viitattu 6.11.2021 https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

MDN Web Docs 2021b. Working with JSON. Viitattu 6.11.2021 https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

MDN Web Docs 2021c. PHP. Viitattu 6.11.2021 https://developer.mozilla.org/en-US/docs/Glossary/PHP

MDN Web Docs 2021d. CSS. Viitattu 6.11.2021 https://developer.mozilla.org/en-US/docs/Glossary/CSS

MDN Web Docs 2021e. Compression in http. Viitattu 6.11.2021 https://developer.mozilla.org/en-US/docs/Web/HTTP/Compression

MDN Web Docs 2021f. gzip compression. Viitattu 13.12.2021 https://developer.mozilla.org/en-US/docs/Glossary/GZip_compression

Minify 2021. NPM Minify. Viitattu 12.11.2021 https://www.npmjs.com/package/minify

Purgecss 2021. PurgeCSS- remove unused CSS. Viitattu 16.11.2021 https://purgecss.com/

Toast UI Chart 2021. Toast UI Chart Documentation Viitattu 21.10.2021 https://nhn.github.io/tui.chart/latest/

Tutorials24x7 2019. Enable Gzip Compression on Apache HTTP Server Viitattu 13.12.2021 https://apache.tutorials24x7.com/blog/enable-gzip-compression-on-apache-http-server

Vertailtavien JS-kirjastojen koodi

```
// Jaetut muuttujat
let Title = "Sadekertymä, Turku, Artukainen vuosina 2019-2020"
let ekaDatasetLabel = "Sademäärä mm 2019"
let tokaDatasetLabel = "Sademäärä mm 2020"
let ekaData = [58.6, 94.2, 51.1, 33.0, 50.2, 26.8, 116.1, 22.5, 55.2,
68.4, 88.7, 79.1]
let tokaData = [51.9, 52.4, 56.9, 5.1, 32.4, 18.3, 60.2, 83.5, 113.2,
63.8, 97.0, 105.8]
let labels = ["tammikuu", "helmikuu", "maaliskuu", "huhtikuu",
"toukokuu", "kesäkuu", "heinäkuu", "elokuu", "syyskuu", "lokakuu",
"marraskuu", "joulukuu"]
// chart.js
new Chart(document.getElementById("chartjsChart"), {
  type: 'bar',
  data: {
    labels: labels,
    datasets: [{
        type: "bar",
        label: ekaDatasetLabel,
        backgroundColor: ["#1f77b4"],
        data: ekaData
      },
      {
        type: "bar",
        label: tokaDatasetLabel,
        backgroundColor: ["#ff7f0e"],
        data: tokaData
      }
    ]
  },
  options: {
    legend: {
      display: false
    },
    title: {
      display: true,
      text: 'Kuuk.'
    }
  }
});
// Billboard.js
var chart = bb.generate({
  data: {
```

```
    columns: [
      [ekaDatasetLabel, ekaData],
      [tokaDatasetLabel, tokaData],
    ],
    type: "bar", // for ESM specify as: bar()
  },
  bar: {
    width: {
      ratio: 0.5
    }
  },
  bindto: "#barChart"
});
// TOAST UI Chart
const toastChart = toastui.Chart;
const el = document.getElementById('toastChart');
const data = {
  categories: labels,
  series: [{
      name: ekaDatasetLabel,
      data: ekaData,
    },
    {
      name: tokaDatasetLabel,
      data: tokaData,
    }
  ],
};
const options = {
  chart: {
    width: 700,
    height: 400
  },
};
toastChart.barChart({
  el,
  data,
  options
});
```

Suorituskykymittaukset ennen optimointia, index.html

Finish = kaikkien resurssien lataus valmis

DOMContent = käyttöliittymän näkyvän osan piirto valmis

Ilman välimuistia			Resurssien koko	928kt
	Finish	DOMContent	Ladattuja resursseja	928kt
	249	128	Prosentteina	100 %
	267	117		
	224	100		
	225	111		
	230	116		
	226	108		
	228	116		
	228	107		
	230	113		
	212	101		
Keskiarvo	231,9	111,7	ms	
Välimuistin kanssa			Resurssien koko	928kt
	Finish	DOMContent	Ladattuja resursseja	30,70kt
	177	86	Prosentteina	3,31 %
	192	85		
	177	70		
	181	91		
	209	108		
	201	101		
	189	86		
	204	98		
	177	82		
	215	106		
Keskiarvo	192,2	91,3	ms	

Suorituskykymittaukset optimoinnin jälkeen, index.html

Finish = kaikkien resurssien lataus valmis

DOMContent = käyttöliittymän näkyvän osan piirto valmis

Ilman välimuistia			Resurssien koko:	688kt
	Finish	DOMContent	Ladattuja resursseja	336kt
	183	108	Prosentteina	48,84 %
	194	100		
	200	115		
	206	95		
	185	102		
	205	114		
	209	121		
	210	124		
	204	111		
	221	121		
Keskiarvo	201,7	111,1	ms	
Välimuistin kanssa			Resurssien koko:	688kt
	Finish	DOMContent	Ladattuja resursseja:	16.03kt
	167	71	Prosentteina:	9,72 %
	200	99		
	160	78		
	198	74		
	179	91		
	191	87		
	196	87		
	166	74		
	182	91		
	164	79		
Keskiarvo	180,3	83,1	ms	