Asko Lahti

Datan visualisointi HTML5-sovelluksessa

Insinööri (AMK)

Tietotekniikka

Syksy 2016

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä: Lahti Asko

Työn nimi: Datan visualisointi HTML5-sovelluksessa

Tutkintonimike: Insinööri (AMK) Tietotekniikka

Asiasanat: Datan visualisointi, HTML5, JavaScript, Web-ohjelmointi

Opinnäytetyön aiheena oli korjata ja parantaa toimeksiantajan aiempaa datan visualisointisovellusta. Tavoitteena oli tehdä sovelluksesta aiempaa suorituskykyisempi ja selkeämpi sekä muokata koodia selkeämmäksi jatkokehitystä varten. Toimeksiantajana oli Collapick Company Oy.

Työssä käsitellään muotojen piirtämistä HTML5-sovellukseen käyttämällä grafiikkakirjastoja ja kertoa miten alkuperäisen sovelluksen ongelmia ratkaistiin ja mistä koodia piti muokata selkeämmäksi.

Alkuperäisen sovelluksen ratkaistavat ongelmat liittyivät kuvaajaan ja sen rajoitetusta piirtämislogiikasta, jolloin uusien ominaisuuksien ja parannusten kehittäminen oli vaikeaa. Lisäksi kuvaajan piirtämiseen käytetty grafiikkakirjasto Kinetic.js oli vanhentunut. Koodia muokattiin ja vaihdettiin grafiikkakirjasto Konva.js:ään, jotta pystyttiin lisäämään kuvaajaan uusia ominaisuuksia.

Kuvaajaan lisättiin työn aikana mahdollisuus skaalata tarkasteltavaa aluetta ja valitsemaan sen sisällä olevia dataelementtejä. Kuvaajan piirtämislogiikassa oli myös pari suorituskykyongelmaa, jotka saatiin ratkaistua.

Lopputuloksesta tuli apuväline ohjelmoinnin jatkamiselle, jotta ohjelmoijilta säästyisi aikaa ja he voisivat jatkokehityksen aikana muokata siitä tarkoituksenmukaisemman.

ABSTRACT

Author: Lahti Asko

Title of the Publication: Data visualization in HTML5 application

Degree Title: Engineer, Information Technology

Keywords: Data visualisation, HTML5, JavaScript, Web-programming

The subject of the thesis was to fix and improve the commissioner's earlier data visualization application. The aim was to make the application more efficient and distinct than before, as well as edit the code clearer for further development. The work was commissioned by Collapick Company Oy.

The work focuses on drawing shapes to the HTML5-application using specific javascript framework and tells how problems from the original application were solved and where the code had to be edited to be clearer.

The problems of the original application were related to the canvas and its limited drawing logic when creating new features and fixes became difficult. Additionally, the graphic library Kinetic.js used to draw into the canvas was outdated. The code was modified and the graphic library was changed to Konva.js so that it was possible to add new features to the canvas.

The visible area scaling was added to the canvas and also the possibility to select data elements inside it. There were also a few performance issues in the draw logic of the canvas that were solved.

The final result of the thesis was a tool for continuing programming so time should be saved from the programmers and they can further develop it to be more appropriate.

SISÄLLYS

TERMISTÖ

Bitmap

Kuvatiedostoformaatti, joka varastoi bittejä. Varastoidut bitit sisältävät kuvissa pikseleiden tiedot. [1].

Canvas

HTML-elementti. Luo piirtoalueen, johon voidaan piirtää JavaScriptillä Canvas API:n kautta. [2,s. 395.]

Git

Versionhallintatyökalu, jolla voi haaroittaa projekteja eri osiin ja yhdistää niitä. Käytetty erityisesti ohjelmointiprojekteissa. [3.]

HTML5

Merkintäkieli, jolla voidaan luoda nettisivuja ja nettisovelluksia. HTML on lyhenne Hypertext markup language-nimityksestä. [1.]

HTML-elementti

HTML5-sovelluksen yksi itsenäinen osa, joka näkyy HTML-sivussa. Esimerkiksi otsikko tai kuva. [1.]

JavaScript

JavaScript on HTML:n yhteydessä pääasiassa selainskriptien tekemiseen käytetty kieli. [2,s. 55]

Konva.js ja Kinetic.js

Javascriptin luokkakirjastoja, joiden avulla HTML5-sovelluksiin voidaan hallita grafiikan tekoa [4].

Luokkakirjasto

Sisältää tiettyjä valmiita ohjelmistokoodikomponentteja, joka voidaan halutessa lisätä ohjelmistoon ilman, että sitä tehdään itse. [4.]

Selainskripti

Tarkoittaa ohjelmakoodia, jonka selain suorittaa HTML-sivun näyttämisen yhteydessä [2,s 55].

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimi Collapick Company Oy, joka on joensuulainen ohjelmistoyritys. Sillä on toimipisteet Kajaanissa ja Joensuussa. Päätuotteina yritys kehittää mobiili- ja web-sovelluksia. Yritys kehittää myös suunnitteluun ja raportointiin liittyviä työkaluja, joita se myy lisenssipohjaisesti asiakkaille. Yrityksen asiakkaita ovat toiset yritykset, eivätkä ne keskity tietylle toimialalle. Tämä opinnäytetyö pohjautuu erääseen näistä tuotteista, jota ei ole vielä julkaistu, joten siitä ei tässä opinnäytetyössä käytetä tarkempaa nimeä kuin HTML5-sovellus.

Kehittämistehtävänä oli parantaa ja korjata yrityksen sovellusta. Alkuperäisessä sovelluksessa esitettiin dataa asettamalla se graafiseen kuvaajaan. Käyttäjä pystyi halutessaan lisäämään ja poistamaan dataa, ja valitsemaan, millä tarkastelualueella data näkyi kerrallaan. Kehittämistarve syntyi, kun sovelluksessa huomattiin jatkokehityksen olevan hidasta ja vaikeaa. Lisäksi sovelluksessa oli suorituskykyongelmia.

Toimeksiantaja määritteli tavoitteiksi korjata ongelmat, lisätä ominaisuuksia ja tehdä jatkokehityksestä sujuvampaa. Näistä kriteereistä viisi oli tarpeeksi laajoja, joita käyn tässä työssä läpi tarkemmin. Työn ohjelmistoarkkitehtuurin tulisi olla joustava, jotta yritys voisi jatkokehittää sovellusta tehokkaasti. Lisäksi muutokset kuvaajaan tulee käyttäjänäkökulmasta olla selkeät.

2 DATAN ESITTÄMINEN

Kerätty tieto tutkittavasta asiasta tallennetaan usein numerokokoelmaksi, ja käsittelemättömänä sitä kutsutaan raakadataksi. Silloin se on asiantuntijoiden ja tietokoneiden ymmärtämässä, mutta ei ulkopuolisille tarkastelijoille esitettävässä muodossa. Jos halutaan esittää raakadatan tietoa muille, täytyy se jalostaa ymmärrettävään ja käytännöllisempään muotoon eli informaatioksi. [5.]

Informaation esitysmenetelmiä ovat tekstuaalinen, numeerinen ja graafinen menetelmä. Numeerisesta informaatiosta kuten taulukoista on tyypillisesti kovin hidasta ja työlästä saada kokonaiskuvaa, jos dataa on runsaasti. Tarkkoja tuloksia hakeva lukija voi saada haluamansa tiedon paremmin kuin graafisesta esityksestä. [6.]

Visuaalisella menetelmällä esitetään informaatio lukijalle kuvaajan muodossa. Visualisaation kaksi pääasiallista tehtävää ovat (1) tiedon tiivistäminen sekä (2) tiedon esittäminen sellaisessa muodossa, että ihminen pystyy kontekstualisoimaan sen oikein eli assosioimaan saamansa informaation muuhun kokemukseensa [5].

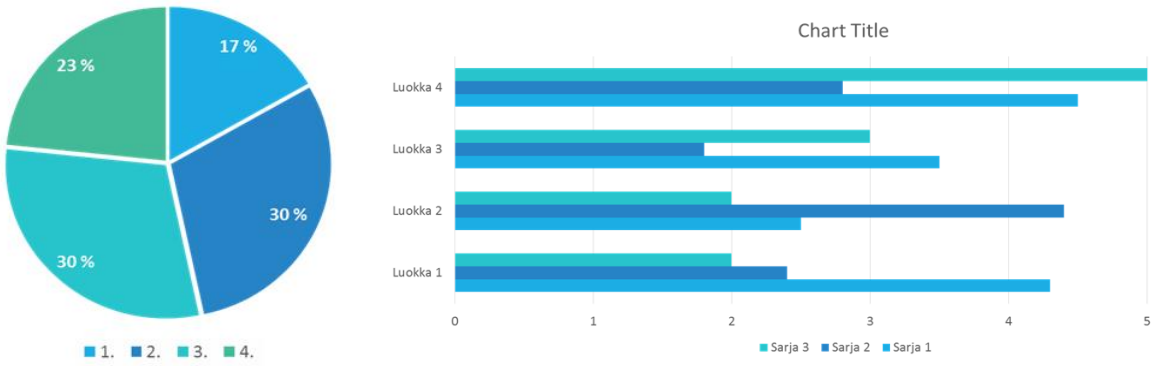
Kuvaaja lähes poikkeuksetta tiivistää dataa - ts. hukkaa dataa - ja siksi kuvaaja onkin syytä suunnitella huolella ennen kuin sen tekee. Keskeinen kysymys onkin: mitä haluan kuvaajalla esittää. Kun tästä on selkeä kuva, niin on syytä pohtia, millaista dataa on tarjolla ja miten siitä saadaan kaivettua päämäärän kannalta keskeinen data. Kolmanneksi on mietittävä, millainen kuvaajatyyppi havainnollistaa dataa parhaiten. Graafisia kuvaajia onkin useita erilaisia moniin eri tarkoituksiin; onkin syytä harkita tarkkaan, mitä kuvaajaa käyttää missäkin tilanteessa. [6.]

Kuvaaja koostuu neljästä osasta: ensinnäkin datan välittäjästä, joka voi tyypistä riippuen olla esim. viivoja, palkkeja, pisteitä, aloja tai lohkoja. Toiseksi on koordinaatisto, jonka suhteen datan välittäjä tulkitaan.

Hyvään kaavioiden käyttöön kuuluu myös lyhyt, mutta kuvaava otsikko, sekä selite. Kuvassa 1 on esimerkki yksinkertaisesta pistekaaviosta, jossa havainnollistetaan datan esittämistä koordinaatistossa. [6.]

Kuva 1. Yksinkertainen pistekaavio

Hyvässä kuvaajassa voidaan verrata dataa toisiinsa. Siksi kaksiulotteiset kuvaajatyypit ovat yleisimmin käytettyjä. Ne ovat lukijalle helposti ymmärrettävissä eivätkä vääristä dataa. Kuvassa 2 on ympyräkaavio ja palkkikaavio. Ympyräkaavio eli piirakkakaavio on hyvä osa-alueiden prosentuaaliseen esittämiseen, ja palkkikaavio on numeraalisten tuloksien vertailuun tätä selkeämpi ratkaisu. [6.]



Kuva 2. Esimerkki piirakkakaaviosta ja palkkikaaviosta

3 TYÖKALUJEN ESITTELY

3.1 HTML5-Canvas

HTML5-sovelluksissa piirtäminen toteutetaan HTML5-Canvas-elementeillä eli piirtoalueilla. Ne tuottavat ohjelmakoodit resoluutioriippuvaisella bitmap-piirtoalueella, ja niitä voidaan käyttää grafiikan, peligrafiikan, taiteen ja muiden visuaalisten kuvien piirtämiseen käytön aikana [1, kohta 4.11.4]. Piirtoalueelle voidaan muodostaa kaksiulotteista tai kolmiulotteista grafiikkaa.

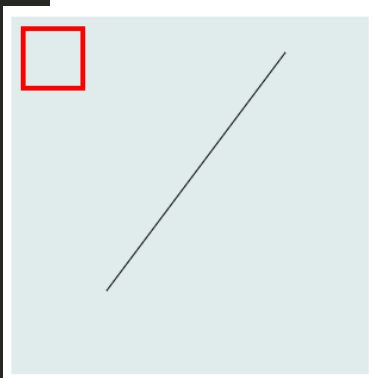
Piirtäminen piirtoalueelle toteutetaan komentojen avulla käyttäen Javascript-ohjelmointikieltä. Ennen piirtämisen aloittamista täytyy piirtoalue-elementille liittää kaksiulotteinen piirtokonteksti. Tämän avulla saadaan käyttöön kaikki kaksiulotteiseen piirtämiseen tarvittavat funktiot. On mahdollista liittää myös kolmiulotteinen konteksti, mutta ei molempia samaan aikaan. [2. s.395]

Muotojen piirtäminen piirtoalueelle toimii vaiheittain. Piirtävälle virtuaaliselle siveltimelle määritellään sen tyyppi, aloituspiste, reitti ja piirtokutsu. Piirrettävä muoto voidaan muodostaa viiva kerrallaan tai käyttäen kaksiulotteisen kontekstin valmiita perusmuotoja. Muodostusvaiheessa ei vielä aseta muotoa piirtoalueelle, vaan sillä kerrotaan, minkälaista ollaan piirtämässä. Lopuksi lähetetään piirtokomento piirtoalueelle, jolloin ruudulle ilmestyy haluttu muoto. [2.]

Muotojen piirtämisessä on tärkeää muistaa kaikki vaiheet, koska voi tapahtua helposti virheitä. Yksi niistä on tilanne, jos unohtaa aloittaa polun tekemisen jokaiselle piirrettävälle muodolle erikseen. Silloin polkua ei resetoida, ja kaikki virtuaalisen siveltimen asetukset tulevat voimaan viimeisimmässä piirtokutsussa, ellei asetuksia vaihdeta jokaiselle muodolle erikseen.

Kuvassa 3 on esimerkkikoodi siitä, miten Javascript ohjelmointikielen avulla voidaan luoda näytölle muotoja ja näyttää esimerkkiä piirtämisen eri vaiheista.

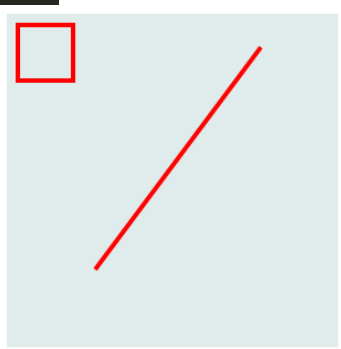
```
// Ilmoitetaan, että piirtäminen voi alkaa
context.beginPath();

// Siirrytään piirtoalueen koordinaatistossa
// kohtaan X: 100 Y: 150
context.moveTo(100, 250);

// Määritetään piirrettävä viiva.
context.lineTo(250, 50);

// Piirrä viiva piirtoalueelle.
context.stroke();



// Ilmoitetaan, että piirtäminen voi alkaa
context.beginPath();

// Määritä pensselin paksuus
context.lineWidth = "4";

// Määritä pensselin väri punaiseksi
context.strokeStyle = "red";

// Määritä neliön sijainti ja koko.
// (X, Y, Leveys, Korkeus)
context.rect(30, 30, 50, 50);

// Piirrä neliö piirtoalueelle.
context.stroke();
```



Kuva 3. Esimerkkikoodi vaiheittain piirtämisestä Javascript-komennoilla ja lopputulos

Yleinen virhe tapahtuu silloin, jos unohtaa stroke()- ja beginPath()-kutsumisen ennen toisen muodon piirtämistä. Kuvassa 4 on havainnollistettu esimerkkikoodilla tilannetta kommentoimalla kyseiset komennot pois koodista.

```
// Ilmoitetaan, että piirtäminen voi alkaa
context.beginPath();

// Siirrytään piirtoalueen koordinaatistossa
// kohtaan X: 100 Y: 150
context.moveTo(100, 250);

// Määritetään piirrettävä viiva.
context.lineTo(250, 50);


// context.stroke();


// context.beginPath();


// Määritä pensselin paksuus
context.lineWidth = "4";

// Määritä pensselin väri punaiseksi
context.strokeStyle = "red";

// Määritä neliön sijainti ja koko.
// (X, Y, Leveys, Korkeus)
context.rect(30, 30, 50, 50);

// Piirrä neliö piirtoalueelle.
context.stroke();
```



Kuva 4. Varoittava esimerkkikoodi, jossa näytetään virhetilanne

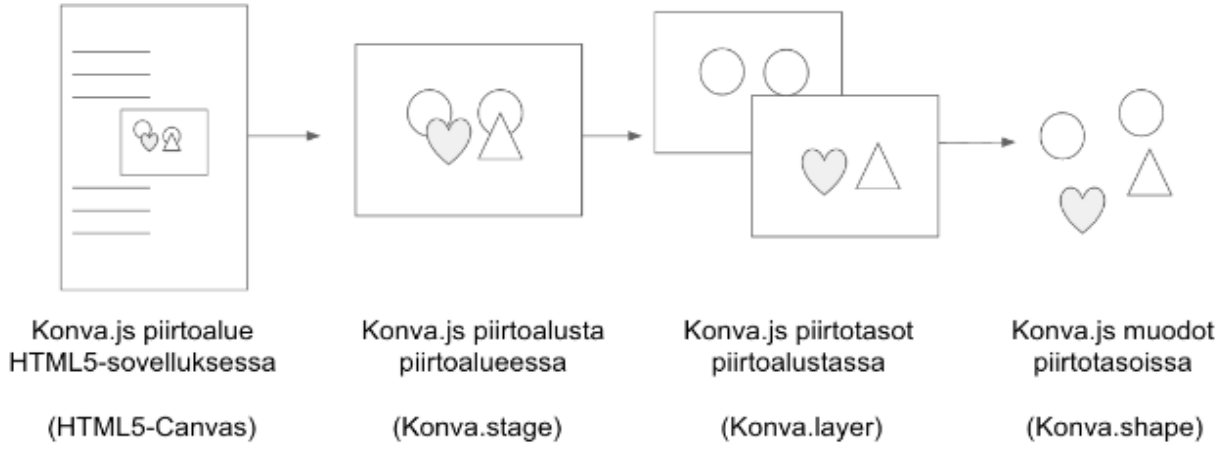
3.2 Grafiikkakirjastot

Grafiikkakirjastoja ovat esimerkiksi Kinetic.js ja Paper.js. Ne ovat HTML5 Canvas Javascriptin luokkakirjastoja, jotka laajentavat 2d ympäristöä mahdollistamalla piirtoalueen vuorovaikutteisuutta tietokone- ja mobiilisovelluksissa [4]. Tämän avulla myös piirrettävät objektit ovat olioita, joihin voi lisätä skaalaus-, liikkumis-, ja animointifunktioita.

Piirtäminen käy vaikeaksi ohjelmoijalle ja on raskasta sovellukselle, jos piirrettäviä asioita on paljon. Piirtämistä voidaan helpottaa ja tehostaa käyttämällä grafiikkaluokkakirjastoja. Ne sisältävät piirtämiselle olennaiset funktiot ja tekevät piirrettävistä muodoista itsenäisiä olioita. Ohjelmoija pystyy piirtämään haluamiansa muotoja antamalla vain parametrit ja asettamalla muoto ruudulle piirrettäväksi. Koodin luettavuus paranee, ja sovellus suoriutuu tehokkaammin, kun grafiikkakirjasto hallinnoi kuvien piirtämistä ruudulle. [4.]

3.3 Konva.js

Konva.js:n piirtoalue (kuva 5) koostuu alustoista (Konva.stage) ja sen piirtotasoista (Konva.layer). Toisin kuin HTML5:ssä piirtoalue voidaan Konva.js:n piirtoalueella piirtää kerroksittain piirtotasoilla. Alustat eli Konva.Stage-objektit sisältävät yhden tai useampia Konva.Layer-piirtotasoja. Tämän avulla ei tarvitse erikseen määrätä jokaiselle piirrettävälle muodolle piirtämisjärjestystä, vaan se voidaan määritellä piirtotasoilla. Esimerkiksi sijoittamalla pohjimmaiseen tasoon taustakuva ja päällimmäiseen tasoon huomiotekstit. [4.]



Kuva 5. Havainnollistava kuva Konva.js-piirtoalueen rakenteesta

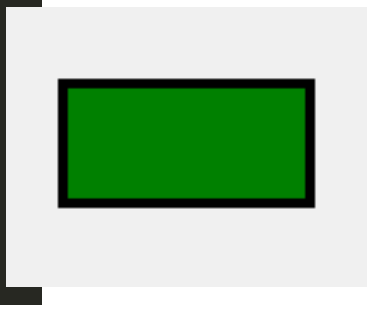
Piirtäminen alustalla ei ole jatkuvaa, mutta piirtämistä voi kutsua mistä vain. Esimerkiksi alustan piirtotasolla olevaa muotoa klikattaessa muutetaan muodon väri punaiseksi ja

kutsutaan piirtämiskutsua alustasta, sillä muoto ei vaihdu punaiseksi ilman uudelleenpiirtoa. [4.]

Ohjelmoimalla piirtäminen Konva.js:llä on yksinkertaisempaa kuin suoraan piirtämällä piirtoalueelle. Se sisältää laajan valikoiman erilaisia valmiita muotoja, joille voi antaa haluamansa koon, värin ja muita ominaisuuksia. Muodosta tulee automaattisesti objekti, jota voi kopioida, muokata ja poistaa helpommin. Muodon piirtäminen vaatii alustan ja piirtotason luomisen, minkä jälkeen muoto voidaan asettaa piirrettävälle tasolle. Tason pitää olla olemassa alustassa, että piirtäminen voidaan toteuttaa. [4.]

Kuvassa 6 on esitetty Konva.js:llä piirtämistä ohjelmoimalla. Samanlainen muotojen luomislogiikka on käytössä monessa eri grafiikkakirjastoissa, mutta Konva.js:ssä on erityisesti haluttu tehdä kuvan tasojen tekemisestä ja hallinnoimisesta mahdollisimman yksinkertaista. Siksi tasojen ja alustojen luominen toimii samanlailla kuin muotojen, eli niistä tehdään objekteja. [4.]

```
// Luodaan alusta piirtämiselle.
var stage = new Konva.Stage({
  container: 'container', // HTML5-Canvas piirtoalueen nimi
  width: width,
  height: height
});

// Luodaan piirtotaso.
var layer = new Konva.Layer();

// Luodaan Konva neliö objekti.
var rect = new Konva.Rect({
  x: 50,
  y: 50,
  width: 100,
  height: 50,
  fill: 'green',
  stroke: 'black',
  strokeWidth: 4
});

// Lisätään neliö piirtotasoon
layer.add(rect);

// Lisätään piirtotaso alustaan.
stage.add(layer);
```



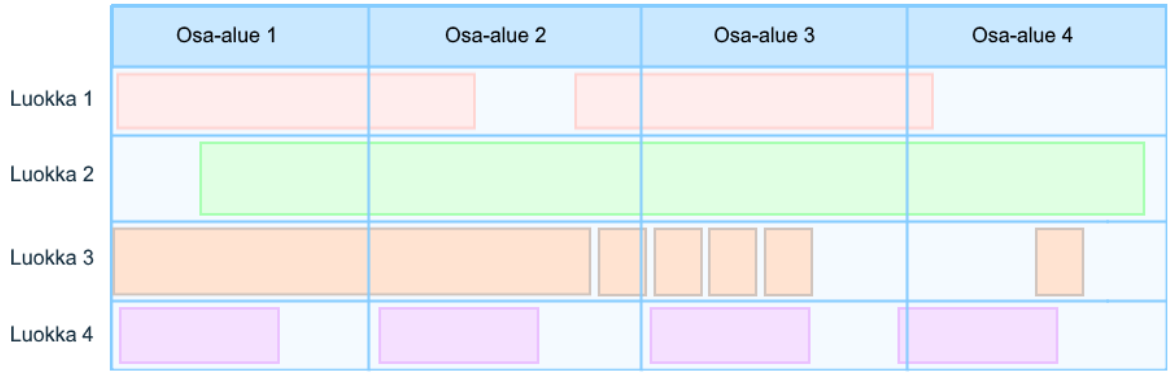
Kuva 6. Esimerkkikoodi Konva.js-grafiikkakirjaston käytöstä

Alusta, piirtotaso ja muoto voidaan käyttää objektien luomisen jälkeen milloin vain, kunhan niihin viitataan koodissa oikein. Piirtokutsua ei tarvitse tehdä erikseen. Koodi on hallittavammassa muodossa ja näin syntyy vähemmän virheitä, kuin tehtäisiin piirrettävät muodot pelkällä javascriptillä.

4 TYÖN TOTEUTUS

4.1 Lähtökohta

Työn kohteena oleva HTML5-sovellus esitti tietokannasta haettua dataa kaksiulotteisessa kuvaajassa. Siinä näkyi määrätty määrä osa-alueita, joita käyttäjä pystyi hiirellä osoittamalla korostamaan. Osa-alueiden sisällä piirretyt dataelementit kuvasivat datan määrää, ja ne oli sijoitettu lineaarisesti vasemmalta oikealle. Dataelementit sijoitettiin kuvaajaan niille annettujen luokkien mukaan omiin luokkariveihinsä. Kuvassa 7 on havainnollistettu kuvaajaa, jossa näkyy dataelementtien sijoitus osa-alueissa ja luokkariveissä.

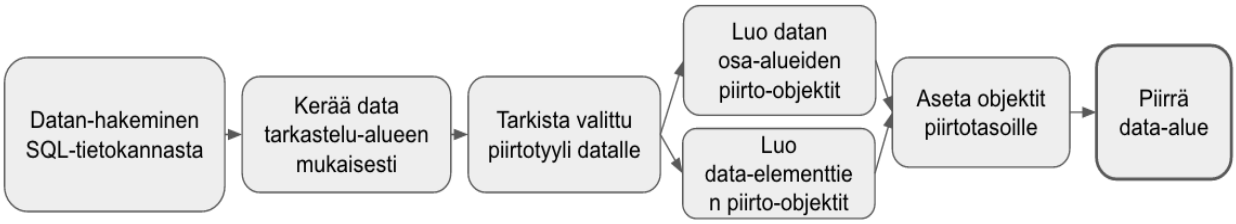


Kuva 7. Digitaalinen hahmotelma sovelluksen kuvaajasta

Kuvaajan yleiset tiedot ja data-elementtien tiedot näkyivät kuvaajan vieressä olevassa hallintapaneelissa. Siellä näkyi data-elementeistä tarkempaa tietoa ja muuta informaatiota. Sieltä pystyi lisäämään ja poistamaan dataa ja vaikuttamaan kuvaajaan suodattamalla luokkia pois näkyvistä.

Kuvaajan piirtäminen oli toteutettu Kinetic.js-luokkakirjastoa käyttäen. Kuvaajassa oli mahdollista vaihtaa kahden eri näkymän välillä, joissa sama data esitettiin eri tavalla.

Piirrettävä data saatiin SQL-tietokannasta, ja dataa muokatessa sovellus päivittää muokatun datan kuvaajaan reaaliaikaisesti. Kuva 8 havainnollistaa tietokannasta ladatun datan muodostamista data-alueeksi.



Kuva 8. Havainnollistava prosessikaavio data-alueen piirtämisestä

4.2 Ratkaistavat ongelmat ja tehtävät muutokset

Ennen opinnäytetyön aloittamista käytiin toimeksiantajan kanssa sovellusta läpi, määritettiin, mitä isoja ongelmia sovelluksessa havaittiin ja mitä ominaisuuksia se kaipasi. Kerättiin viisi isoa ongelmaa ja muutosta, jotka tässä opinnäytetyössä pitäisi suorittaa.

Grafiikkakirjaston vaihtaminen

Sovelluksen piirtämiseen käytettiin Eric Dowellin luomaa Kinetic.js-luokkakirjastoa. Se on vanhentunut, ja tekijä on poistanut dokumentaatiot internetistä. Dowell lopetti sen jatkokehityksen, kun hänellä ei ollut aikaa ja dokumentaatioiden ylläpitäminen internetissä maksoi liikaa. Kinetic.js:lle täytyy löytää korvaava grafiikkakirjasto, joka olisi mahdollisimman samankaltainen, ettei koodin rakenteeseen tarvitsisi tehdä isoja muutoksia ja koodi pysyisi mahdollisimman samankaltaisena.

Osa-alueiden koko ja määrä

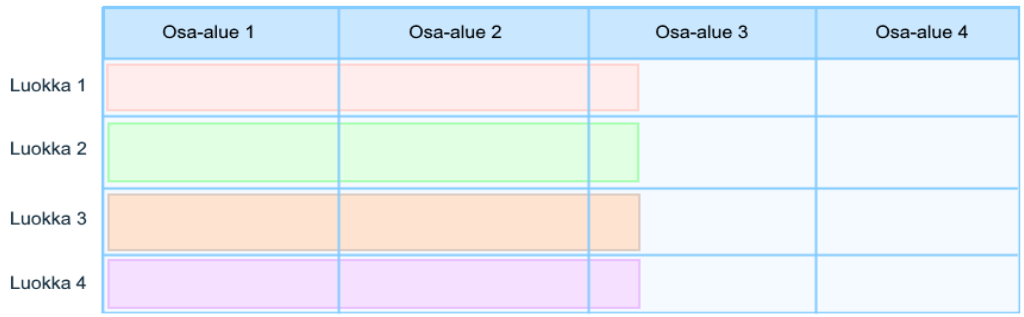
Alkuperäisen sovelluksen esitystavassa ei voinut esimerkiksi suurentaa eikä pienentää osa-alueiden kokoa ja määrää, koska ne oli valmiiksi määrätty. Käyttäjä pystyi siirtymään

eteenpäin ja taaksepäin tarkastelualueessa, mutta vain tietyn määrän kerrallaan. Tehtäväksi saatiin lisätä mahdollisuus lisätä ja vähentää osa-alueita. Pitäisi olla myös mahdollista pilkkoa osa-alueita pienempiin osiin tarkempaa tarkastelua varten.

Dataelementtien selkeyttäminen

Kuvaajassa olevia dataelementtejä ei voinut alkuperäisessä sovelluksessa valita hiirellä. Haluttiin lisätä mahdollisuus valita niitä hiirellä, joka näyttäisi tarkempaa tietoa valitusta dataelementistä. Lisäksi dataelementtiin pitää lisätä korostusgrafiikka, kun se on valittu tai sitä osoitetaan hiirellä.

Suunnitteluvaiheessa huomattiin, että kun listadatalle annetaan monta luokkaa, sovellus loi kuvaajaan saman data-elementin kaikille luokkariveille. Tämä ominaisuus oli vahingossa tehty alkuperäiseen sovellukseen, mutta sitä ei ollut korjattu. Esimerkkinä on kuva 9, jossa on dataelementille annettu luokat ja kuvaaja monistaa saman datan jokaiselle luokkariville.



Kuva 9. Havainnollistus dataelementtien monistamisesta

Esitysnäkymien muokkaus ja lisääminen

Alkuperäisen sovelluksen esitystapa oli alkujaan rajattu vain yhteen tarkoitukseen, jolloin uusien ja erilaisten esitysnäkymien tekeminen oli haastavaa. Ohjelmistorakennetta pitää

muuttaa, jotta sovelluksessa voisi pystyä luomaan uusia esitysmenetelmiä eri tarkoituksiin.

Suorituskykyongelmat

Alkuperäisessä sovelluksessa oli suorituskykyongelmia. Tehtävänä oli etsiä hidastuksia aiheuttavat ongelmat ja korjata ne. Suorituskykyongelmien etsiminen rajattiin kuitenkin sovelluksen kuvaajaan.

4.3 Suunnittelu

Grafiikkakirjaston vaihtaminen

Monien grafiikkaluokkakirjastojen joukosta löydettiin Konva.js, joka on Anton Lavrenovin muokkaama ja itsenäiseksi haarautunut projekti Kinetic.js:tä. Tämän kirjaston avulla ajateltiin, että muutokset alkuperäiseen koodiin tulee olemaan minimaaliset. Piirtologiikkaan ei pitäisi tulla kovin isoja muutoksia ja vaihtaminen pitäisi sujua mahdollisimman nopeasti.

Osa-alueiden koko ja määrä

Osa-alueiden ja dataelementtien piirtäminen oli toteutettu erikseen, jolloin niiden piirtäminen ei ollut osa-alueiden määrän eli tarkastelualueen koon mukaan toteutettu, vaan piirtämislogiikka oletti niiden olevan vakio. Logiikka täytyy muuttaa ottamaan osa-alueiden määrää huomioon. Niiden määrän muuttuessa kuvaajan koko pitää pysyä samana ja kuvaajan sisällä olevat osa-alueet ja data-elementit skaalautua.

Dataelementtien selkeyttäminen

Suunniteltiin, että kuvaajalle pitäisi muodostaa ja piirtää listadatasta vain yksi dataelementti, vaikka sillä olisi monta luokkaa. Moniluokkainen dataelementti näyttäisi kuvaa-

jassa sisältämät luokkansa visuaalisesti. Sitten käyttäjälle näkyisi vähemmän dataelementtejä, jolloin kuvaaja olisi selkeämpi, mutta sisältäisi saman määrän informaatiota. Ongelmana on saada dataelementtien indikaattorit tarpeeksi selkeiksi.

Dataelementin valitsemiseen hiirellä käytetään Konva.js-ominaisuutta liittää tapahtumia objekteihin. Niihin pitää lisätä hiiren valitsemistapahtuma ja animaatiotapahtuma korostusta varten. Korostusgrafiikka haluttiin olevan animoitu ja dataelementin tiedot pitäisi näkyä sovelluksen hallintopaneelissa.

Esitysnäkymien muokkaus ja lisääminen

Erilaisia esitysmenetelmiä oli alkuperäisessä sovelluksessa kaksi, ja ne oli käsin räätälöity omaan tarkoitukseensa. Toimeksiannon vaatimuksissa haluttiin mahdollisuus lisätä niitä useita. Uusien esitysmenetelmien tekeminen oli vaikeaa, jolloin sovellukselle täytyy kehittää uusi menetelmä tai parantaa vanhoja näkymiä yksinkertaisemmaksi. Pitää olla mahdollista räätälöidä haluttu näkymä tiettyyn tarkoitukseen, ja sen lisääminen sovellukseen ei saisi olla kovin vaikeaa.

Päätettiin yksinkertaistaa alkuperäisten näkymien luontia, mikä johtaisi siihen, että voidaan luoda näkymästä olio. Sillä voidaan jatkossa luoda uusia näkymiä tehokkaammin, nopeammin ja vähennetään virheiden tapahtumista sekä ylimääräistä toistoa koodissa. Arvioitiin, että tämä tehdään viimeiseksi muiden parannusten jälkeen, mutta se olisi silti yksi tärkeimmistä ominaisuuksista.

Suorituskykyongelmat

Tutkimalla sovelluksen suorituskykyongelmia huomattiin niiden tapahtuvan silloin, jos dataelementtejä oli kuvaajassa satoja. Osa-alueiden korostaminen, datan lisääminen ja datan päivittäminen aiheutti huomattavaa suorituskyvyn heikentymistä hetkeksi, sovelluksen vasteaika suureni, ja grafiikoiden päivittyminen hidastui. Piirtämislogiikkaa pitää päivittää tehokkaammaksi, koska sovellusta käyttämällä dataelementtien määrä nousee useihin satoihin.

4.4 Kuvaus muutosten toteuttamisesta

Grafiikkakirjaston vaihtaminen

Grafiikkakirjaston vaihtaminen Kinetic.js:stä Konva.js:ään tapahtui ohjelmakoodissa korvaamalla Kinetic-kutsut Konvaksi. Se ei vaatinut muita toimenpiteitä. Esimerkiksi neliötä luodessa kutsuttiin ennen Kinetic.Rect-luokkaa, mutta muuttamalla se Konva.Rect-luokaksi sai sovelluksen tekemään saman asian. Tässä opinnäytetyössä muutos auttoi graafisten ominaisuuksien lisäämisessä, koska Konva.js-luokkakirjastolla oli vielä olemassa dokumentaatiot internetissä.

Osa-alueiden koko ja määrä

Kehittämisen aikana tehtiin myös ominaisuus, että osa-alueita pystyi pilkkomaan pienempiin osiin. Tämä aiheutti myös sovelluksen hidastumista, ja tutkimalla löydettiin osa-alueista ylimääräisiä piirrettäviä osia. Osa-alueilla oli erikseen taustakuva, reunat ja hiirellä valittava alue, ja kaikki olivat eri tasoilla mutta jokainen osa sisälsi muidenkin osat, eli piirrettiin kolme kertaa sama kuva ruudulle. Ongelma ratkaistiin yksinkertaistamalla muotoja ja tekemällä hiiren osoittaminen osa-alueeseen tehokkaammaksi, jolloin osa-alueiden määrä ei enää vaikuttanut sovelluksen suorituskykyyn.

Tarkastelualueen koon eli osa-alueiden määrän muuttaminen oli suurin ja vaikein muutos sovelluksessa. Vähentämällä ja lisäämällä osa-alueita piti skaalata kuvaajassa olevat osa-alueet ja dataelementit oikean kokoiseksi. Alkuperäinen piirroslogiikka, joka oletti osa-alueita olevan tietty määrä, aiheutti eniten ongelmia korjauksen aikana. Dataelementtien ja osa-alueiden piirtämislogiikka piti uusia melkein kokonaan. Tarkastelualueen koon pienentämisellä pystyttiin tutkimaan tarkemmin dataelementtien välisiä eroja ja laajentamalla näki paremmin laajemman informaatiookonaisuuden kerralla.

Dataelementtien selkeyttäminen

Kehitettiin keino esittää yhdellä dataelementillä monta luokkaa piirtämällä sen sisältämien luokkien indikaattorit muodon sisälle. Näin vähennetään dataelementtien määrää kuvaajassa ja saadaan jo yhtä tarkastelemalla selville enemmän informaatiota, jos luokat on asianmukaisesti tehty.

Esitysnäkymien muokkaus ja lisääminen

Näiden ominaisuuksien ja parannusten tekemisessä oli paljon työtä, jolloin uusien näkymien lisäyksen helpottamiseen ei jäänyt paljon aikaa. Olemassa olevia näkymiä saatiin tehokkaammaksi ja selkeämmäksi, mutta ne tarvitsevat vielä työtä, että olisivat valmiita olemaan pohjia uusille näkymille. Näkymien lisääminen pitäisi toimia pohjalla, johon lisätään asetukset, piirtämislogiikka ja mahdolliset lisämuutokset. Näkymät ovat nyt tilanteessa, jossa piirtämislogiikka olisi muutettavissa haluttuun muotoon, mutta asetukset ja muut muutokset täytyy tehdä erikseen.

Suorituskykyongelmat

Animaation lisääminen ja useamman osa-alueen korostaminen aiheutti huomattavaa hidastusta, joka johti siihen, että kehityksen aikana löydettiin vika piirtämislogiikassa. Asiaa tutkimalla huomattiin, että alkuperäisessä kuvaajassa oli kolme piirtotasoa: pohjataso, keskitaso ja päällimmäinen tekstitaso, ja keskitasossa olivat kaikki dataelementit ja osa-alueet. Sovellus piirsi tason uudelleen, jos sen sisältämissä muodoissa tapahtui muutos. Eli jokainen muutos osa-alueissa tai data-elementeissä sai keskitason päivittymään, mikä aiheutti hidastumista.

Ongelma ratkaistiin luomalla omat tasot osa-alueille, dataelementeille ja animaatioille. Jos korostettiin osa-alue, niin se ei enää päivitä turhaan kaikkia dataelementtejä. Lisäksi animaatiolle piti tehdä oma taso, jotta saadaan animaatiossa oleva objekti väliaikaisesti pois muiden elementtien joukosta. Dataelementin korostusanimaatio olisi muuten päivittänyt kaikkia dataelementtejä turhaan.

4.5 Testaaminen

Testaaminen toteutettiin yrityksen käytössä olevilla menetelmeillä. Sovellusta testattiin kehityksen aikana ohjelmoinnin- ja käyttäjän näkökulmasta. Sovellusta testattiin ennen ja jälkeen sovelluksen viemistä serverille, jolloin saatiin mahdollisimman paljon informaatiota sovelluksen toimivuudesta.

4.5.1 Ohjelman testaus

Sovelluksen kehityksen aikana testattiin sovellusta käymällä yrityksen kehittämää testauslistaa läpi. Listassa oli jokainen sovelluksen toiminto, ja testaajan pitää käydä kaikki ominaisuudet läpi ja todeta niiden toimivuus. Toiminnot olivat esimerkiksi datan lisääminen ja osa-alueen korostaminen kuvaajassa. Listan alkuun merkittiin testin päivämäärä, sovelluksen versio, käytettävä selain ja laite millä testataan. Jos toiminto ei toimi, voidaan päätellä johtuuko se esimerkiksi selaimesta vai sovelluksen päivityksestä.

Virheen tapahtuessa merkitään, mitä tapahtui ja mikä olisi voinut aiheuttaa ongelman mahdollisimman tarkasti selitettynä. Listaan merkityt merkinnät näkyvät kaikille kehittäjille korostettuna, jolloin kehittäjät pystyvät katsomaan missä on tapahtunut virheitä. Kehittäjät merkitsevät virheilmoituksiin miten virhe voidaan korjata tai miten se on korjattu.

4.5.2 Käytettävyystestaus

Sovellusta testattiin myös käytettävyystestausmenetelmällä. Siinä henkilö, joka ei ole sovellusta vielä käyttänyt, testaa sitä samalla kun kehittäjät seuraavat hänen suoritustaan. Tässä sovelluksessa valittiin vuorotellen kollegoista se, joka ei ollut vielä päässyt kokeilemaan sovellusta. Pyydettiin testihenkilöä myös puhumaan ääneen tekemistään ja havaintojaan sovelluksessa. Tämän aikana seuraajat eivät kommentoineet tai antaneet ohjeita, ellei sovelluksessa tapahtunut isoa virhettä tai testaaja ei keksinyt, mitä seuraavaksi olisi pitänyt tehdä. Kirjuri kirjoitti listaan kaikki huomiot.

Testaamisen jälkeen pyydettiin testaajalta kommentteja ja mielipiteitä, joita tuli mieleen käyttökokemuksesta ja sovelluksesta. Testaajalta kysyttiin myös lisäkommentteja ongelmista, joita tuli testauksen aikana ja miten se voidaan korjata. Keskusteluun käytettiin noin 15 minuuttia. Sen jälkeen pidettiin kehittäjien kesken palaveri, jossa keskusteltiin testauksesta ja suunniteltiin mahdollisia parannuksia sovellukseen.

Testaamisesta saatiin paljon hyvää palautetta ja hyviä ideoita. Testaajat olivat yrityksen muita kehittäjiä, jolloin kommentit keskittyivät kehittäjän ja asiakkaan näkökulmiin. He osasivat kertoa kattavasti uusista kommenteista ja niille heränneistä ideoista testauksen aikana. Joistakin ideoista tuli tähän toimeksiantoon toteuttavia, mutta jotkut ehdotukset eivät liittyneet tähän opinnäytetyöhön, vaan sovelluksen jatkokehitykseen.

5 TYÖN TULOKSET

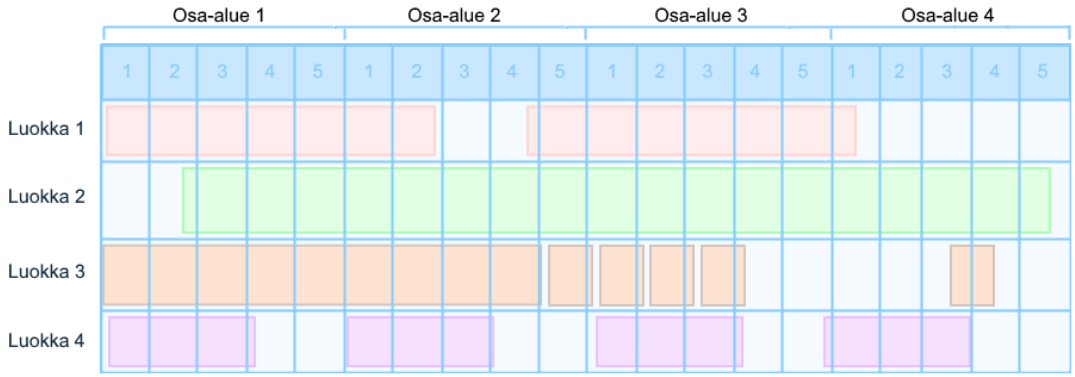
Sovelluksesta saatiin tehokkaampi ja dynaamisempi kuin ennen, mitä toimeksiannon aikana haluttiin saavuttaa. Koodia on muokattu ongelma-alueiden kohdista ja edistämällä koodin luettavuutta. Suurimmaksi osaksi saatiin kaikki kriteerit toteutettua.

Grafiikkakirjaston vaihtaminen

Kinetic.js:stä Konva.js:ään ei vaikuttanut sovelluksen toimintaan, mutta testattiin varmuuden vuoksi, oliko vaihtamisella suorituskykyyn tai graafiseen toimivuuteen vaikutusta. Ei havaittu minkään muuttuneen. Konva.js:n dokumentaatiot on vielä olemassa, mikä tulee auttamaan jatkokehityksessä.

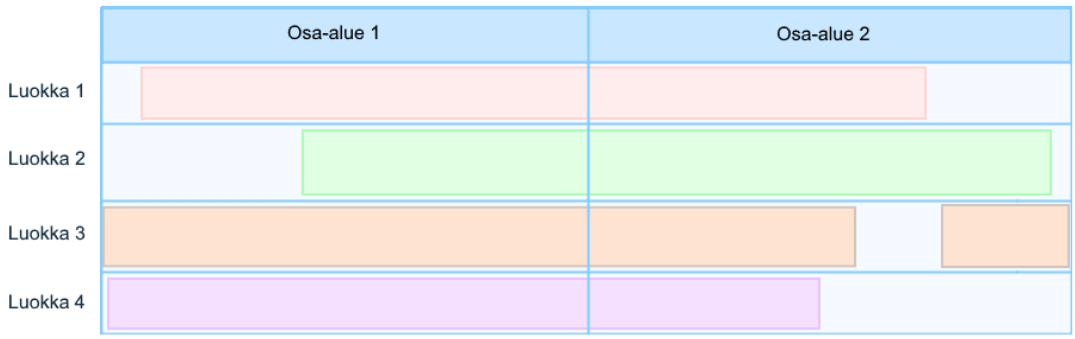
Osa-alueiden koko ja määrä

Sovelluksen kuvaajaa voi säätää, kuinka monta osa-aluetta se näyttää, ja voi määrätä osa-alueet pienempiin osiin halutessaan. Kuvassa 10 on havainnollistettu osa-alueiden pilkkomisen toimivuutta.



Kuva 10. Osa-alueiden pilkkominen pienempiin osiin

Tarkasteluväliä voidaan myös suurentaa ja pienentää poistamalla ja lisäämällä osa-alueita. Toisin kuin pilkkominen osiin, tämä myös vaikuttaa osa-alueiden ja dataelementtien pituuteen kuvaajassa, koska kuvaajan koko pysyy samana. Tämä on havainnollistettu kuvassa 11, jossa on alkuperäisestä kuvaajasta poistettu kaksi osa-aluetta.

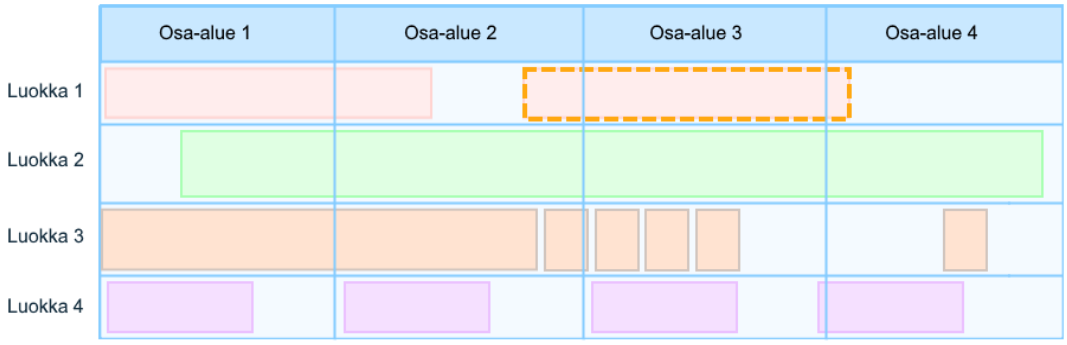


Kuva 11. Osa-alueiden vähentäminen ja data-elementtien skaalaus

Dataelementtien selkeyttäminen

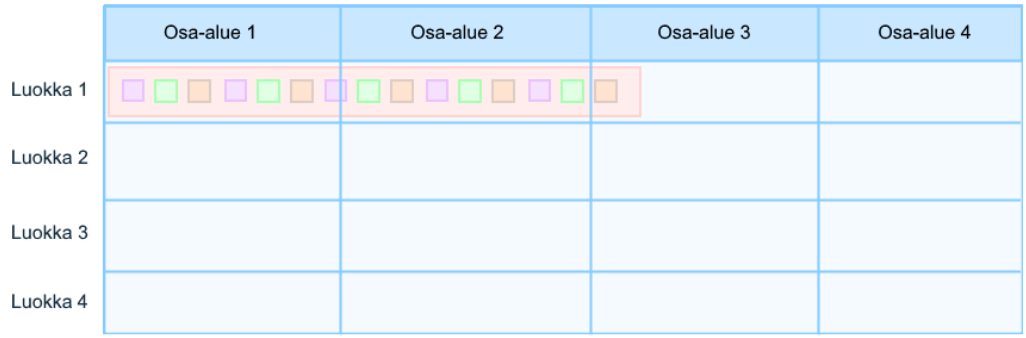
Kuvaajan data-elementtejä pystyy nyt valitsemaan ja korostamaan ne kuvaajasta pienellä animaatiolla. Tätä ei pystynyt ennen tekemään, ja se on ollut monien käytettävyystestaajien kommenteissa myönteisenä parannuksena. Valitseminen onnistuu valitsemalla joko kuvaajasta haluttua dataelementtiä tai valitsemalla datan sovelluksen hallintapaneelista. Valitsemalla dataelementin kuvaajasta avaa sen datan ikkunan hallintapaneelissa, jolla voi päästä tarkastelemaan datan tietoja nopeammin kuin ennen. Alkuperäisessä sovelluksessa datan tarkastelu piti hoitaa selaamalla hallintapaneelin datalistaa läpi yksitellen.

Korostamisanimaatio vaihtaa valitun dataelementin väriä vaaleammasta tummempaan 1 sekunnin välein 5 sekunnin ajan. Animaation voi pysäyttää sulkemalla hallintapaneelista dataan liittyvän tarkasteluikkunan. Kuvassa 12 havainnollistetaan dataelementin korostusta kuvaajassa.



Kuva 12. Dataelementin korostaminen kuvaajassa

Moniluokkainen dataelementti ei enää näytä samaa dataa eri luokkariveillä, vaan pääluokan omistama dataelementti sisältää muiden luokkien väriset indikaattorit sisällänsä. Kuvassa 13 data-elementin pääluokkana on luokka 1, ja esittää luokat 2 - 4 erivärisinä indikaattoreina.



Kuva 13. Dataelementin luokkien indikaattorit

Esitysnäkymien muokkaus ja lisääminen

Uusien piirtotyylien lisäämisen helpottaminen jäi kesken. Sovelluksessa oli alun perin kaksi erilaista kuvaajaa, joista ensimmäistä muokattiin koodirakenteeltaan selkeämmäksi ja tehokkaammaksi. Sitä saatiin tarpeeksi muokattua, jotta sitä voidaan käyttää muiden piirtotyylien pohjana. Niiden lisääminen sovellukseen on silti haastavaa, koska niistä pitäisi tehdä omia luokkia. Sen tekeminen tosin vaatii tietynlaista apuluokkaa koodissa, jota ei ehditty tekemään.

Suorituskykyongelmat

Piirtäminen saatiin tehtyä paljon tehokkaammaksi alkuperäiseen verrattuna. Kuvaaja ei hidastu, vaikka sillä olisi yli kaksituhatta dataelementtiä. Ainoastaan dataelementtien luominen on hidasta, kun ne haetaan tietokannasta ja piirretään ensimmäistä kertaa ruudulle.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli kehittää alkuperäisen sovelluksen kuvaajaa, korjata ongelmia ja parantaa koodin rakennetta jatkokehittäjiä varten. Lopputuloksena saatiin kaikinpuolin tehostettua ja parannettua sitä. Se on suorituskykyisempi ja koodiltaan selkeämpi kuin ennen työn aloittamista. Sovellus on jatkokehitykseen kykenevä kokonaisuus.

Työn aikana huomattiin kuvaajan värien harmonian vaikutus visualisointiin. Yrityksen graafikko alkoi jatkamaan sen tutkimista omassa opinnäytetyössään.

Kaikkia tavoitteita ei saatu suoritettua, kun yksi tärkeimmistä eli esitysnäkymien helppo lisäämisominaisuus olikin arviota isompi työ. Siihen saatiin tehtyä alustavat korjaukset, mutta työtä vielä riittää. Opinnäytetyön suuruus oli muuten asetettu tarpeellisen laajaksi.

Yritys pystyy tästä jatkamaan kehitystä eteenpäin. Kuvaaja ei ole enää niin rajoittunut yhteen menetelmään, jolloin uusien ominaisuuksien lisääminen ei ole niin vaikeaa kuin ennen.

LÄHTEET

WWW-julkaisuihin viitattu 15.05.2016.

1. World Wide Web Consortium: A vocabulary and associated APIs for HTML and XHTML. [WWW-julkaisu] <https://www.w3.org/TR/html5>
2. Korpela Jukka: HTML5-Käsikirja. Docendo Oy 2014. ISBN: 978-952-291-031-8
3. Software Freedom Conservancy: Git documentation. [WWW-julkaisu] <https://git-scm.com/doc/>
4. Lavrenov Anton: Konva.js [WWW-julkaisu] <http://konvajs.github.io/>
5. Cybercom: Big data suomessa - visualisaatio. Kirjoittanut Tero Keski-Valkama [WWW-julkaisu] <http://www.cybercom.com/fi/Suomi/Yritys/Blogit/Blogit/big-data-suomessa---visualisaatio/>
6. Karjalainen Leila, Karjalainen Juha: Tilastojen graafinen esittäminen. Otavan Kirjapaino Oy 2009. ISBN: 978-952-9776-31-3
7. Dowell Eric: Kinetic.js. [WWW-julkaisu] <https://github.com/ericdrowell/KineticJS/>
8. Bucephalus.org: The HTML5-Canvas handbook. [WWW-julkaisu] <http://bucephalus.org/text/CanvasHandbook/CanvasHandbook.html/>