Tero Metsänen

Twitterin aihetunnisteiden visualisointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

24.4.2019

Metropolia

Tekijä Otsikko Sivumäärä Aika	Tero Metsänen Twitterin aihetunnisteiden visualisointi 21 sivua + 1 liite 24.4.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	lehtori Ilpo Kuivanen

Insinöörityön tavoitteena oli selvittää Twitterin aihetunnisteiden visualisoinnin mahdollisuuksia JavaScript-kirjasto D3:n avulla. Lopputuloksena oli tarkoitus tuottaa havainnollistava verkkograafi Twitterin rajapinnasta saatavan datan pohjalta.

Raportin teoriaosuudessa tutustuttiin datan visualisointiin D3:n avulla ja sen teknisiin ominaispiirteisiin. Raportoitavassa tutkimustyössä käytiin myös läpi demosovelluksessa käytetyt menetelmät ja työvaiheet API-kutsusta valmiiseen visualisointiin. Lopulliseksi visualisoinniksi muodostui aihetunnisteista koostuva verkkograafi, jonka data haettiin Twitteristä käyttäjän syöttämän hakusanan perusteella.

Sovellus toteutettiin paikallisesti Node.js-ympäristössä, jossa API-kutsut ja datan käsittely tehdään palvelinpuolella ja itse visualisointi selaimessa.

Lopputulos osoitti verkkograafin soveltuvan varsin hyvin Twitterin kaltaisen sosiaalisen median datan visualisointiin, mutta graafin informatiivisuus osoittautui huomattavan riippuvaiseksi aihetunnisteen suosiosta, käyttötavoista ja verkottuneisuudesta.

Avainsanat	visualisointi, d3, twitter, aihetunniste, hashtag, graafi

Author	Tero Metsänen
Title	Visualization of Twitter Hashtags
Number of Pages	21 pages + 1 appendix
Date	24 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information technology
Professional Major	Software engineering
Instructors	Ilpo Kuivanen, senior lecturer

The purpose of this thesis was to find out the possibilities of visualization of Twitter hashtags using the JavaScript library D3. The goal was to produce an interactive graph based on Twitter API data.

The theoretical part of the thesis is about visualization of data in general and D3's technology. The work also covered the methods and solutions that were used in the visualization process. For the final visualization, a network graph consisting of Twitter hashtags was created. Its data was fetched from Twitter based on user-entered search term.

The demo application was implemented locally in Node.js-environment. Api calls and data-handling were implemented on the server side and the visualization in the browser.

Final results showed that a network graph is very well suited for visualizing social media data like that of Twitter, but the informativeness of the graph seemed to be highly dependent on the hashtags connectivity, popularity and its usage.

Keywords	visualization, d3, twitter, hashtag, graph

Sisällys

1 Johdanto

Tämän insinöörityön ideana oli hankkia kokonaiskuva datan visualisoinnista webohjelmoijan näkökulmasta ja tutustua niihin työvaiheisiin, joita havainnollisen graafin tuottaminen lähdedatasta nykyaikaisilla työkaluilla käsittää. Aiheen valintaan vaikutti oma kiinnostus visualisointiin webympäristössä, ja se tarkentui Twitterin aihetunnisteiden visualisointiin johtuen sen tarjoamista ohjelmoinnillisista lisähaasteista ja nykypäivänä saatavissa olevan tiedon ja sosiaalisen median luonteesta.

Visualisoinnin työkaluna työssä tutustutaan JavaScript-kirjasto D3:een, joka on tämän hetken yksi käytetyimmistä visualisointikirjastoista. Se hyödyntää monipuolisesti eri webtekniikoita ja tarjoaa kattavan valikoiman toimintoja erityyppisten graafien ja visualisointituotteiden luomiseen.

Mikroblogipalvelu Twitter valikoitui datan lähteeksi, sillä se on yhä (2019) varsin suosittu sosiaalinen media ja antaa käyttäjilleen pääsyn sisältöönsä rajapintansa Twitter API:n kautta. Tämän työn kannalta twiittien oleellisinta informaatiota ovat niiden hashtagit eli aihetunnisteet, joiden tarkoitus on kerätä aiheeltaan samanlaiset twiitit yhteen. Yksittäisten twiittien visualisointi ei tuntunut tutkinnon ja insinöörityön kannalta tarkoituksenmukaiselta, sillä työ olisi tällöin painottunut liikaa pelkkään visualisointiin. Sen sijaan aihetunnisteiden visualisointi vaatii datan käsittelyä, jolloin raportoitava kokonaisuus on ohjelmointiteknisesti laajempi.

2 Datan visualisointi selainympäristössä D3:n avulla

2.1 Visualisointi käsitteenä

Datan visualisoinnin tarkoituksena on saattaa lähtöaineisto sellaiseen visuaaliseen muotoon, että sen sisältämät erityispiirteet, trendit ja riippuvuussuhteet ovat helpommin havaittavissa. Onnistunut visualisointi välittää tarkastelijalle hetkessä lähtöaineiston yleisvaikutelman, jonka omaksumiseen numeerista tai tekstimuotoista dataa tarkastelemalla kuluisi huomattavasti pidempi aika.

2.2 Yleistä D3:sta

D3 (*Data Driven Documents*) on yksi tämän päivän käytetyimmistä selainympäristön visualisointikirjastoista. Se hyödyntää visualisoinnissa HTML:n ja CSS:n lisäksi SVG:tä (*Scalable Vector Graphics*), vektorikuvien kuvauskieltä. D3 sisältää DOM-operaatiot esimerkiksi elementtien valitsemiseen, lisäämisen ja poistamiseen hieman jQueryn tapaan. D3 koostuu erilaisista moduuleista, joita käytetään yhdessä halutun visualisoinnin muodostamiseksi.

Koska D3 antaa valmiiden graafityyppien sijaan kehittäjille monipuoliset työkalut luoda erityyppisiä visualisointeja, monet muut visualisointikirjastot (muun muassa Taucharts ja Reactiin suunniteltu Recharts) on kehitetty D3:n toiminnallisuuden pohjalta.

2.3 SVG-kuvauskieli

Visualisointi D3:ssa koostuu yleensä SVG-elementeistä, eli XML-pohjaisesta vektorigrafiikasta. Tavallisista kuvatiedostoista poiketen SVG skaalautuu eri näyttökokoihin menettämättä terävyyttään. Responsiivisuus onkin SVG:n ehkä suurin etu verrattuna staattisiin jpg- tai png-kuvatiedostoihin. Toinen merkittävä hyöty on tekstipohjaisuus, jolloin SVG-elementeille voi antaa CSS-tyylimääreitä HTML:n tapaan.

Polku (*path*) on tärkein SVG-elementti, sillä siitä muodostetaan muut peruskuviot, kuten janat, suorakulmiot ja ympyrät. Teksti on ainoa peruselementti D3:ssa, joka ei muodostu polusta.

Elementtien sijainti dokumentissa ja ulkoasu määritetään niille annettavilla attribuuteilla. Esimerkiksi ympyräelementin (*circle*) keskikohdan sijainti määräytyy cx- ja cy -attribuuteilla ja säde r-attribuutilla. Elementin taustavärin voi asettaa fill-attribuutilla.

2.4 Valinnat ja operaatioiden ketjuttaminen

Eräs hyödyllinen D3:n operaatio on valintafunktio select, jolla voi hakea DOM:ista niin HTML- kuin SVG-elementtejä. DOM (*Document Object Model*) eli dokumenttioliomalli on HTML-elementtien ja muiden elementtien rakenteellinen kuvaus, joka mahdollistaa sen sisältämien elementtien manipuloinnin. Valinta (*selection*) koostuu taulukosta elementtejä, jotka vastaavat käytettyjä valitsimia (*selector*). Valitsin on tyypillisesti jokin luokka, id tai muu tunniste. Näitä voi myös yhdistää. Esimerkiksi kahden luokan valitsin olisi ”.class1.class2”. (Ændrew Rininsland, 2017, s. 91.)

Valinnan jälkeen valittuja elementtejä tai niiden ominaisuuksia voi muokata D3:n muilla operaatioilla. Yleensä tämä tapahtuu ketjuttamalla toiminnot select-funktion kanssa. Monet D3:n operaatiot (esimerkiksi DOM:iin lisäävä append ja DOM:ista poistava remove) palauttavat senhetkisen valinnan, jotta ketjuttaminen tai valinnan nimittäminen muuttujaksi olisi mahdollista. (Ændrew Rininsland, 2017, s. 102.)

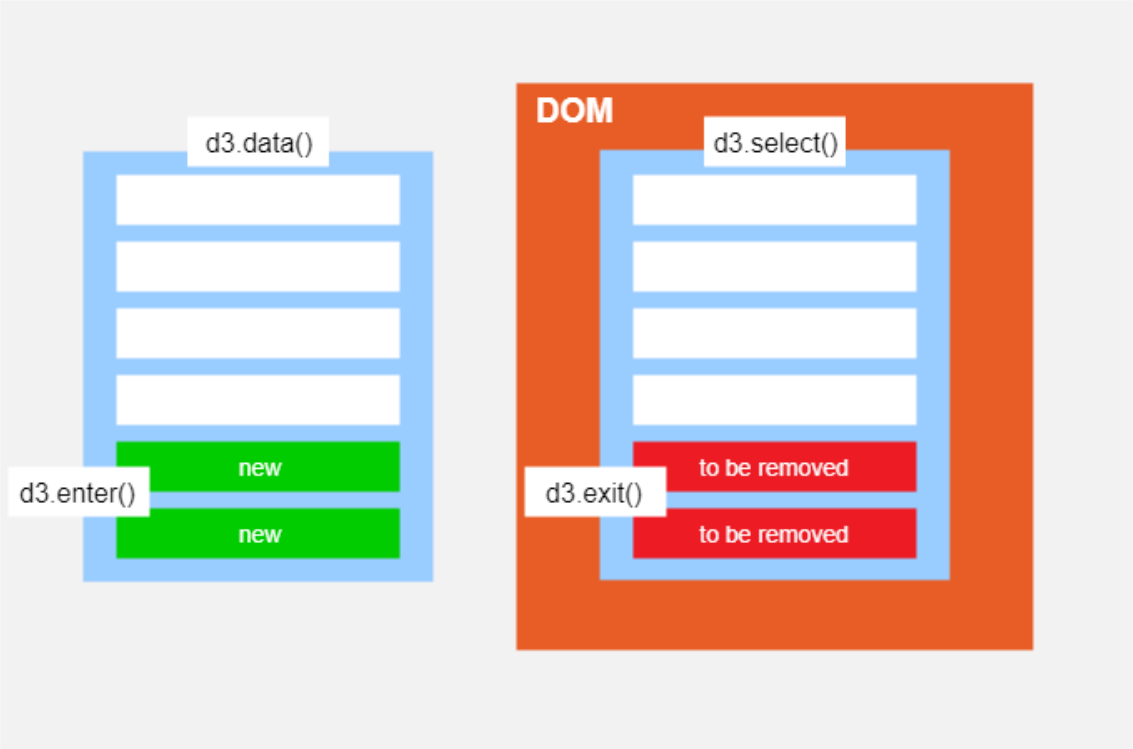
```
const newLinks = d3.selectAll('li').append('a')

newLinks.text((d,i) => { return 'Link' + (i+1) })
```

Esimerkkikoodi 1. Linkkitagin lisääminen jokaiseen li-elementtiin. Muuttujaa *newLinks* voi myöhemmin hyödyntää mm. linkkien tekstin lisäyksessä

2.5 Datan sidonta elementteihin

Koska D3 on kehitetty nimenomaan datan visualisoinnin tarpeisiin, perinteisiä DOM-operaatioita tärkeämpänä toiminnallisuutena voidaan pitää datan liittämistä DOM:in elementteihin. Tästä käytetään termiä datan sidonta *(data join*). Sidonta tapahtuu datafunktion avulla. Yksinkertaisimmillaan sidottava data voisi olla esimerkiksi kokonaisluvuista koostuva taulukko. Tuettuja datamuotoja ovat mm. CSV ja JSON.

Jotta visualisointi vastaisi siihen liitettyä dataa, täytyy taulukkomuotoisen datan alkioiden määrän vastata valittujen elementtien määrää. Mikäli näin ei ole, D3:ssa on asian korjaamiseksi enter- ja exit-operaatiot. Kuvassa 1 havainnollisestaan näiden operaatioiden toimintaperiaatetta. Enter viittaa niihin elementteihin, jotka puuttuvat valinnasta. Vastaavasti exit viittaa elementteihin, jotka löytyvät valinnasta, mutta eivät datasta. Koska enter ja exit ovat viittauksia, ne eivät luo uutta valintaa, eikä niitä täten voi ketjuttaa keskenään. (Ændrew Rininsland, 2017, s. 104.) Sen sijaan enter-operaation voi ketjuttaa merge-operaation kanssa. Tämä mahdollistaa yhtäaikaisen käsittelyn sekä uusille että nykyisille elementeille.



Kuva 1. D3:n datan valintaoperaatioiden peruslogiikka

2.6 Datan päivittäminen

D3:n luoja Mike Bostock suosittelee, että datan päivityksessä noudatetaan niin kutsuttua yleistä päivitysmallia (*general update pattern*) (Bostock, 2018). Tässä mallissa elementtien lisääminen, poistaminen ja päivittäminen tehdään samassa funktiossa. Tyypillinen suoritusjärjestys on seuraavanlainen:

1. Data sidotaan elementtiin data-funktiolla.
2. Uusiin elementteihin luodaan viittaus enter-funktiolla.
3. Uudet elementit lisätään DOM:iin append-funktiolla.
4. Merge-funktiolla laajennetaan viittaus koskemaan myös valinnan nykyisiä elementtejä.
5. Poistuviin elementteihin luodaan viittaus exit-funktiolla.
6. Poistuvat elementit poistetaan DOM:ista remove-funktiolla.

Tyypillisesti päivitysfunktiota kutsutaan aina silloin, kun visualisoitava data muuttuu. Merge-funktion avulla elementtien lisäys- ja päivitystoiminnot voidaan ketjuttaa, jolloin tarvittava koodi on lyhyempi, kuten esimerkkikoodissa 2.

```
function update(data) {
  var u = d3.select('#content')
    .selectAll('div')
    .data(data);

  u.enter()
    .append('div')
    .merge(u)
    .text(function(d) {
      return d;
    });

  u.exit().remove();
}
```

Esimerkkikoodi 2. Päivitysfunktio yleisen päivitysmallin mukaan (Cook, D3 In Depth - Enter and exit, 2018)

2.7 Transformaatiot

Transformaatiot ovat oleellinen osa D3:n toiminnallisuutta. Käytännössä ne käsittävät SVG-kuvioiden muodon ja sijainnin muuttamisen halutulla tavalla. Transformaatiot ovat matriisin monikertoja, mikä on laskennallisesti tehokas menetelmä verrattuna kuvion jokaisen pisteen laskemiseen ja päivittämiseen erikseen. Tietokone käsittelee useammankin transformaation ketjua yhtenä transformaationa. (Ændrew Rininsland, 2017, s. 126.)

Vaikka transformaatiot voidaan määritellä suoraan matriiseina matrix-funktiolla, kehittäjän näkökulmasta huomattavasti helpompaa on käyttää D3:n valmiita transformaatioita, jotka ovat translate, scale, rotate, skewX ja skewY. Transformaatiot asetetaan transform-attribuutin avulla. Esimerkkikoodissa 3 käytetään translate-transformaatiota.

```
function ticked() {
  link.attr("x1", function (d) { return d.source.x; })
    .attr("y1", function (d) { return d.source.y; })
    .attr("x2", function (d) { return d.target.x; })
    .attr("y2", function (d) { return d.target.y; });

  enterNode.attr("transform", function (d) {
      return "translate(" + d.x + "," + d.y + ")";
    })
}
```

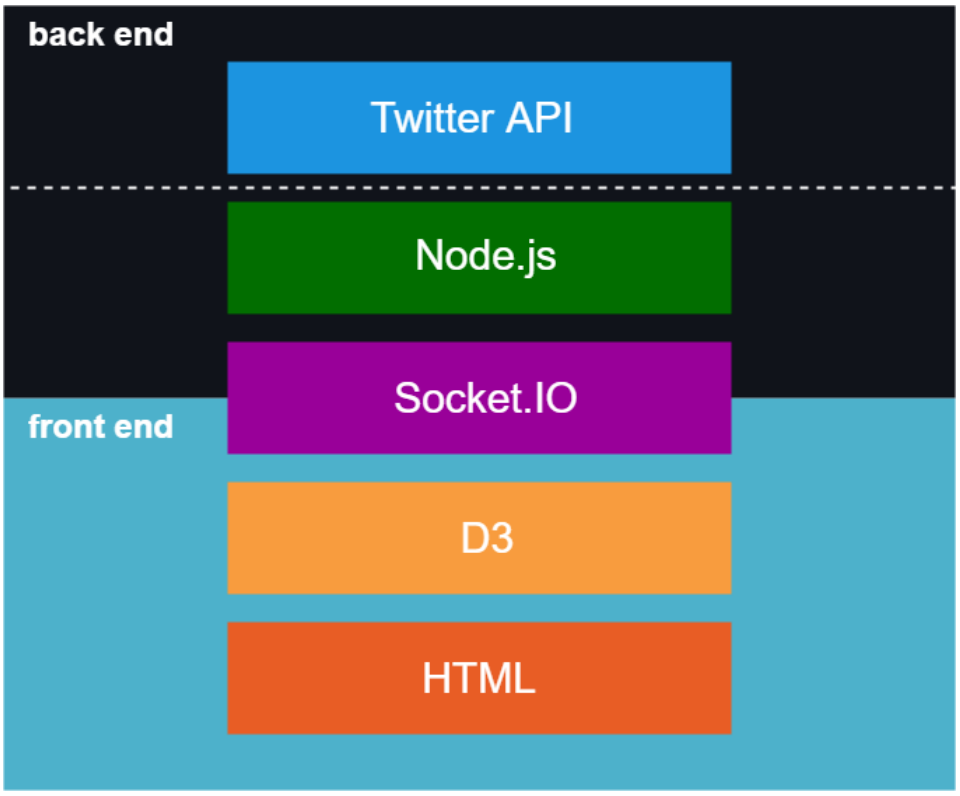
Esimerkkikoodi 3. Transformaation asettaminen ticked-funktiossa

3 Datan haku ja käsittely

3.1 Sovelluksen rakenteen kuvaus

Visualisointi toteutettiin websovelluksena Node.js-ympäristössä Express-sovelluskehyksessä. Ympäristön valintaan vaikutti muun muassa se, että tällöin sovelluksen pystyi toteuttamaan kokonaisuudessaan JavaScript-kielellä. Lisäksi Nodeen oli saatavilla kolmannen osapuolen kirjasto, joka yksinkertaisti Twitter API:n käyttöä tarjoamalla tarkoitukseen sopivat funktiot (Twitter for Node.js - Npm, 2019).

Palvelimen ja selaimen välisestä kommunikoinnista sovelluksessa vastaa tapahtumapohjainen JavaScript-kirjasto Socket.IO, joka mahdollistaa kommunikoinnin molempiin suuntiin. Socket.IO on luonteeltaan asynkroninen, eli sallii muun ohjelman samanaikaisen suorittamisen odottamisen sijaan (synkroninen viestintä). Arkkitehtuurinsa vuoksi Socket.IO soveltuu erityisen hyvin reaaliaikaista viestintää sisältäviin sovelluksiin.



Kuva 2. Sovelluksen yleisrakenne

3.2 Datan haku Twitterin rajapinnasta

Sovellus hakee twiitit Twitterin rajapinnasta Twitter-kirjaston avulla. GET-palvelupyynnön parametrina on käyttäjän sovelluksen hakukenttään syöttämä aihetunniste, sekä kerralla palautettavien twiittien määrä (100). Onnistunut kutsu palauttaa JSON-muotoisen Tweet-olioista koostuvan taulukon (esimerkkikoodi 4).

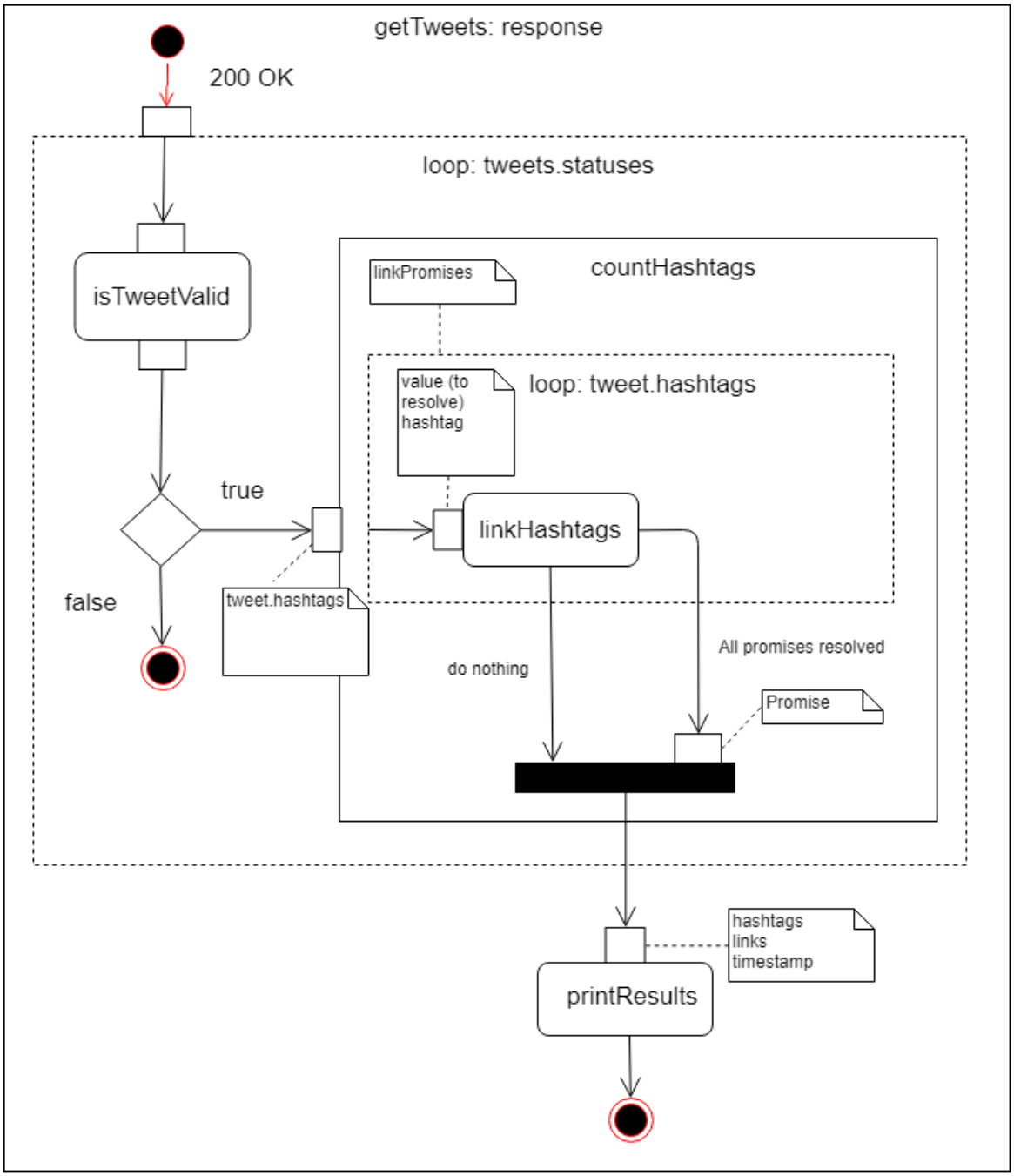
```
{
 "created_at":"Thu Apr 06 15:24:15 +0000 2017",
 "id": 850006245121695744,
 "id_str": "850006245121695744",
 "text": "1/ Today we’re sharing our vision for the future of the Twitter
API platform!nhttps://t.co/XweGngmxlP",
 "user": {},
 "entities": {}
}
```

Esimerkkikoodi 4. Tweet-olio sisältää twiitin sisällön lisäksi oleellisen metatiedon. Insinöörityön kannalta oleellisin data eli twiitin aihetunnisteet löytyvät entities-olion alta hashtags-taulukosta. (Twitter, Tweet Objects - developer.twitter.com, 2018.)

3.3 Datan käsittely Node.js-ympäristössä

Sovelluksen datan käsittely perustuu twiittien ja niiden aihetunnisteiden läpikäyntiin sisäkkäisissä silmukoissa. Twiittivirran aihetunnisteet ja niiden väliset linkit eli tulevan graafin kaaret tallennetaan taulukoiksi globaaleina muuttujina. Kuvassa 3 on esitetty käsittelyn vaiheet toimintakaaviona.

Datan käsittelyoperaatiot on tehty sovelluksessa paikoin asynkronisiksi käyttäen hyödykseen JavaScriptin Promise-olioita eli niin kutsuttuja lupauksia. Promise-oliot mahdollistavat hallitumman tulosten sekä virheiden käsittelyn (Peltomäki, 2017, s. 145). Menetelmän pääperiaatteena on, että ohjelma jatkaa tiettyyn koodiosaan vasta, kun Promise-olioon liitetty koodi on hyväksytysti suoritettu. Tämä tapahtuu resolve-metodin avulla. Vastaavasti reject-metodilla voidaan palauttaa hylätty tulos virhetilanteisiin reagoimista varten.



Kuva 3. Toimintakaavio sovelluksen datan käsittelystä. Ylemmässä silmukassa käsitellään Twitter API:n palvelupyynnön kautta saadut twiitit yksi kerrallaan. Sisemmässä silmukassa jokaisen twiitin sisältämät aihetunnisteet käydään läpi, ja ne linkitetään toisiinsa graafia varten. Kun kaikkien twiittien aihetunnisteet on lisätty taulukkoon ja linkitetty, lähetetään valmis data selaimeen visualisoitavaksi.

Ensimmäisenä Promisea hyödynnetään sovelluksessa heti datan käsittelyn alkuvaiheessa, kun koodissa käydään läpi silmukassa twitter-moduulin kautta saadut twiitit (liite 1, s. 1). Koska JavaScriptin for- ja forEach-silmukat ovat synkronisia, ne eivät odota silmukan sisällä olevan asynkronisen operaation suorittamista, vaan jatkavat suoraan silmukan seuraavaan iteraatioon. Sovelluksessa on kuitenkin tarkoituksena käsitellä twiitit järjestyksessä datan hallinnan helpottamiseksi. Tämän vuoksi silmukka on toteutettu JavaScriptin reduce-metodia hyödyntäen. Reduce "supistaa" läpikäytävän tietorakenteen alkion yhteen arvoon, jolloin edeltävän iteraation tulos on aina seuraavan käytössä. Tämä toimintaperiaate on reduce-metodin erityispiirre. (MacArthur, 2019.) Se mahdollistaa Promise-lupausten ketjun, jolloin peräkkäiskäsittely onnistuu. Menetelmää havainnollistetaan esimerkkikoodissa 5, jossa setTimeout-metodia kutsutaan satunnaisilla, maksimissaan tuhannen millisekunnin suuruisilla aikaparametreilla.

```
function sequentialLoopExample() {
    [...Array(20)].reduce((p, _, i) =>
      p.then(_ => new Promise(resolve => {
          let timeOut = Math.floor(Math.random() * 1000) + 1;
          setTimeout(() => {
              console.log('resolving ' + i);
              resolve();
          }, timeOut);
      }))
      , Promise.resolve());
}
```

Esimerkkikoodi 5. Asynkronisen koodin peräkkäiskäsittely silmukassa ketjutettujen lupausten avulla.

Aihetunnisteet käsitellään sovelluksessa twiitti kerrallaan countHashtags-funktiossa (liite 1, s. 2). Sovellus käsittelee twiittivirran aihetunnisteita yhtenä taulukkona, ja yhden twiitin sisältämät aihetunnisteet linkitetään toisiinsa tuon taulukon indeksien perusteella. Myös tässä funktiossa hyödynnetään Promise-oliota, sillä callback-funktiota kutsutaan (eli twiittejä läpikäyvään silmukkaan palataan) vasta, kun kyseisen twiitin aihetunnisteiden linkitys on tehty.

Jotta lopullinen graafi olisi mahdollisimman havainnollistava ja informatiivinen, sovellus ei laske aihetunnisteita tietyistä twiiteistä. Suodatus tapahtuu sovelluksessa ennen aihetunnisteiden laskemista. Suodatus on suurimmilta osin toteutettu tavallisia ehtolauseita käyttäen isTweetValid-funktiossa (liite 1, s. 4).

1. Retweetit. Twiittien uudelleen lähettäminen on yleinen käytäntö Twitterissä, mutta ne vääristävät visualisointia, sillä suosittujen twiittien aihetunnisteilla olisi tällöin liian suuri painoarvo graafissa. ”Retweeted”-alkuiset twiitit hylätään.

2. Enemmän kuin kuusi aihetunnistetta sisältävät twiitit. Aihetunnisteiden "spämmäys" voi myös vääristä graafia.

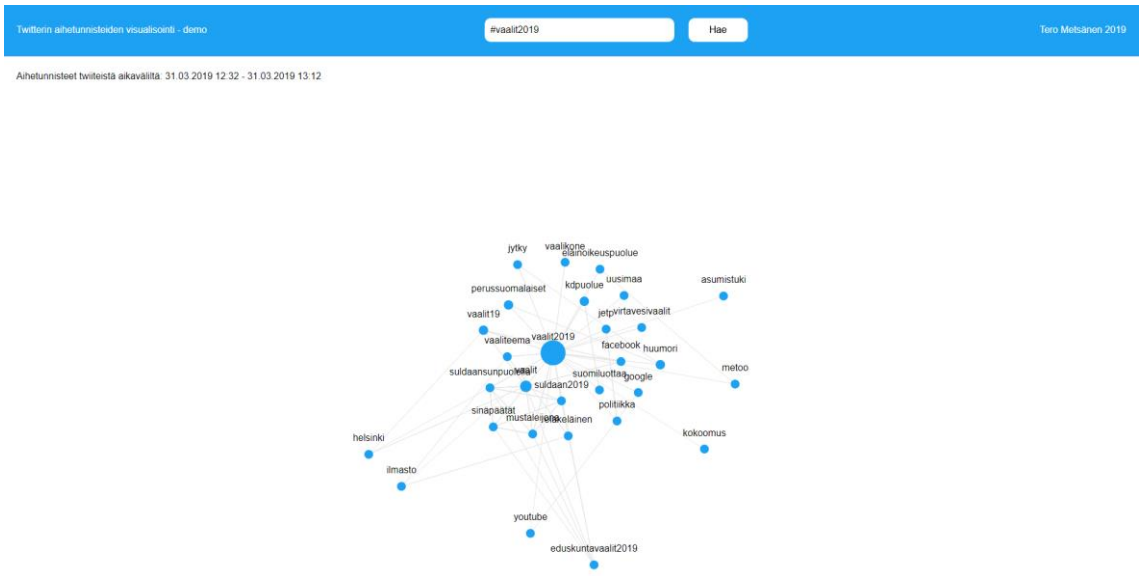
3. Aihetunniste #trending. Jonkun aiheen suosio tai ”trendaavuus” ei ole varsinainen puheenaihe itsessään, joten myös nämä twiitit suodatetaan.

4. Alatyylinen tai muuten sopimaton sisältö.

4 Visualisointi verkkograafin avulla

4.1 Sovelluksen käyttöliittymä

Sovelluksen selainpohjainen käyttöliittymä (kuva 4) on hyvin pelkistetty. Käyttöliittymä sisältää graafin lisäksi hakukentän aihetunnisteille sekä twiittien aikaleiman.



Kuva 4. Sovelluksen käyttöliittymä

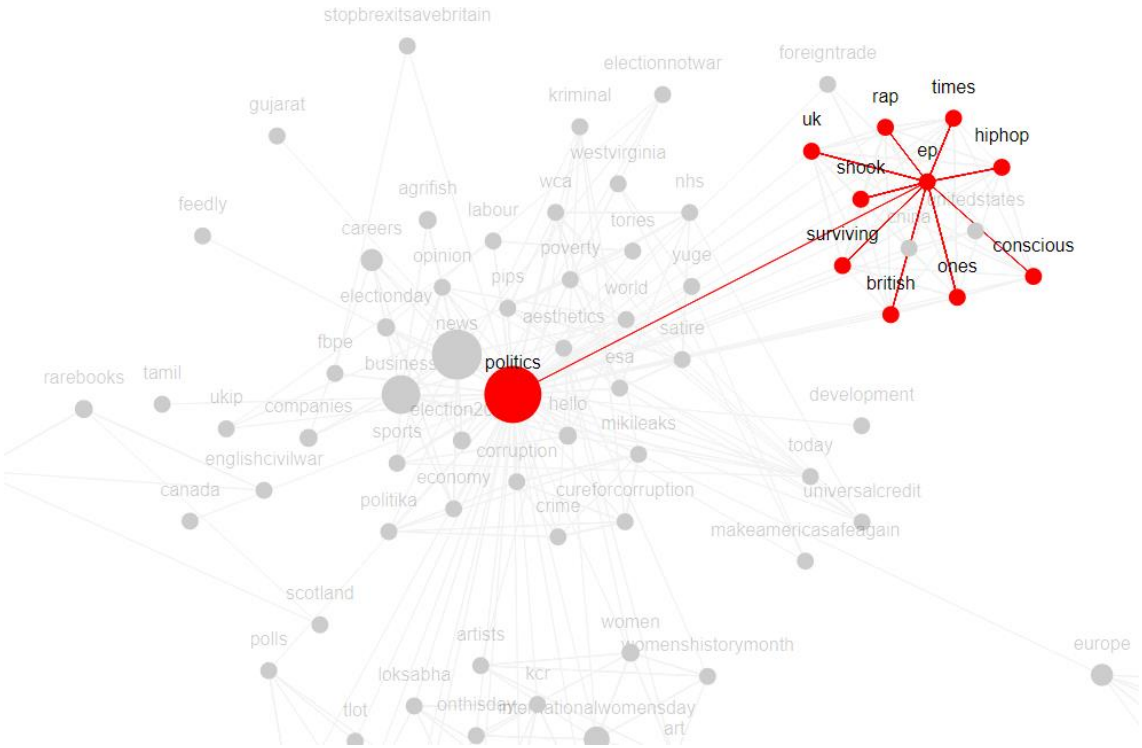
4.2 Graafin rakenne ja tekniset ominaisuudet

Sovelluksen tuottama graafi on suuntaamaton verkkograafi, jonka juurisolmuna on käyttäjän hakema aihetunniste. Twiittivirrasta poimitut aihetunnisteet ovat verkkograafin muita solmuja. Kaksi solmua ovat yhteydessä toisiinsa eli muodostavat kaaren, mikäli kyseiset aihetunnisteet on mainittu samassa twiitissä vähintään kerran. Näin graafista pystyy hahmottamaan toisiinsa liittyvät aihetunnisteet. Solmut ovat muodoltaan ympyräelementtejä, joiden pinta-ala riippuu twiittien määrästä kyseisellä aihetunnisteella (esimerkkikoodi 6). Solmujen etäisyys riippuu kahden solmun välille löydettyjen linkkien määrästä.

```
enterNode.append("circle")
      .attr("r", function (d) {
            var radius = 10 * Math.sqrt(d.count / Math.PI);
            if (radius >= 25) radius = 25;
            if (radius <= 7.5) radius = 7.5;
            return radius;
      })
```

Esimerkkikoodi 6. Ympyräelementin säteen ja sen myötä pinta-alan määräytyminen aihetunnisteen esiintymiskertojen mukaan

Graafin vuorovaikutteisuuden lisäämiseksi solmuja klikkaamalla visualisoinnissa korostuvat siihen linkittyneet muut solmut, kuten kuva 5 osoittaa. Tämä on hyödyllistä tilanteissa, joissa solmuja ja linkkejä on erityisen paljon. Vuorovaikutteisuutta parantaa myös D3:n drag-toiminto, jolloin solmujen liikuttelu onnistuu hiirellä vetämällä.



Kuva 5. Aihetunnisteeseen #ep linkittyneet solmut korostettuna

Verkkograafissa hyödynnetään D3:n Force layout -ominaisuutta. Force layoutin avulla graafin solmujen liike voidaan simuloida fysiikkapohjaisesti eri tavoin. Tämä toiminnallisuus voi saada solmut muun muassa hylkimään toisiaan tai estämään niitä menemästä päällekkäin. Simulaation toiminta pohjautuu D3:n force-funktioihin. Esimerkiksi forceCenter-funktion avulla voidaan asettaa gravitaation voimakkuus graafin keskikohtaan nähden. Solmujen päällekkäisyys voidaan estää forceCollide-funktiolla. (Cook, D3 In Depth - Force Layout, 2018) Graafin data ja linkit asetetaan samalla kuin simulaatio ja käytettävät force-funktiot määritetään (esimerkkikoodi 8).

Force layout soveltuu erityisen hyvin tilanteisiin, joissa graafin elementtien sijainti ei riipu niihin sidotusta datasta (Ændrew Rininsland, 2017, s. 330). Sovelluksessa force layout on käytössä lähinnä visualisoinnin elävöittämiseksi, vaikka päällekkäisyyden esto onkin varsin tarpeellinen ominaisuus.

Jokaisella iteraatiolla simulaatio kutsuu ticked-funktiota, jossa päivitetään graafin solmujen ja kaarien sijainnit. Mikäli graafi päivittyisi reaaliajassa, saattaisi selain kuormittua liikaa, sillä force layout on D3:n tekniikoista raskaimmasta päästä vaadittavan laskennan suhteen (solmujen sijainnin laskeminen voi kestää niiden määrästä riippuen joitain sekunteja). (Cook, D3 In Depth - Force Layout, 2018.) Sijainnin laskenta on sovelluksessa toteutettu siten, että solmut ja linkit pysyvät halutun kokoisen svg-elementin sisällä (esimerkkikoodi 7).

```
function ticked() {

    enterNode
    .attr("cx",
       function(d) {
       return d.x = Math.max(d.radius, Math.min(width - d.radius, d.x));
       })
    .attr("cy",
       function(d) {
       return d.y = Math.max(d.radius, Math.min(height - d.radius, d.y));
       })
    .attr("transform", function (d) {
        return "translate(" + d.x + "," + d.y + ")";
    });

    link.attr("x1", function (d) { return d.source.x; })
    .attr("y1", function (d) { return d.source.y; })
    .attr("x2", function (d) { return d.target.x; })
    .attr("y2", function (d) { return d.target.y; });
}
```

Esimerkkikoodi 7. Solmujen ja linkkien pakottaminen selainikkunan sisään ticked-funktiossa Tim Rothin esimerkin mukaan (Roth T. , 2019)

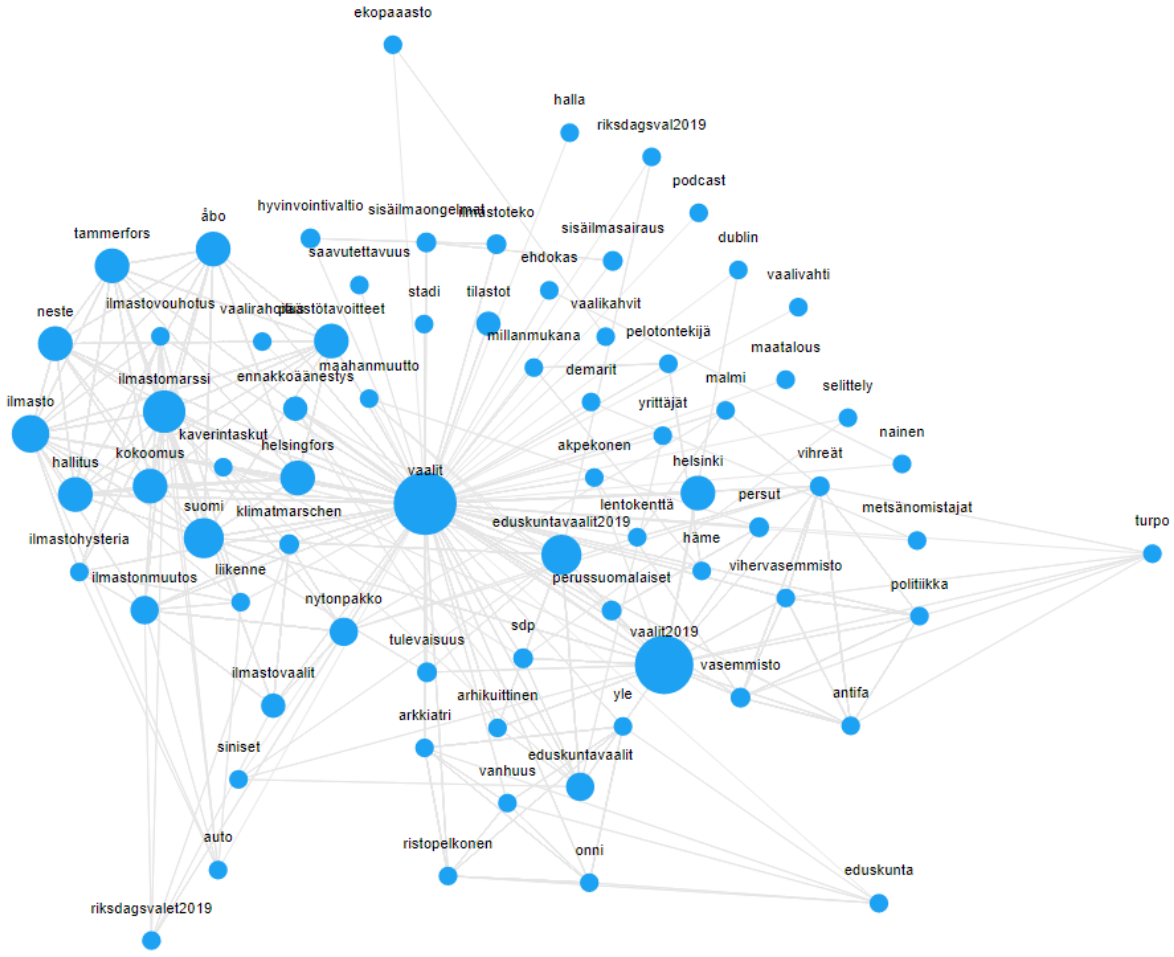
```
simulation = d3.forceSimulation(hashtags)
.force('charge', d3.forceManyBody().strength(-100))
.force('center', d3.forceCenter(width / 2, height / 2))
.force('link', d3.forceLink().links(links).distance(function (d) {
    return calculateLinkDistance(d);
}))
.on('tick', ticked)
.force('collision', d3.forceCollide().radius(function(d) {
    return d.radius;
}));
```

Esimerkkikoodi 8. Datan ja linkkien sekä force-funktioiden määrittäminen simulaation luonnin yhteydessä

4.3 Graafin analysointi

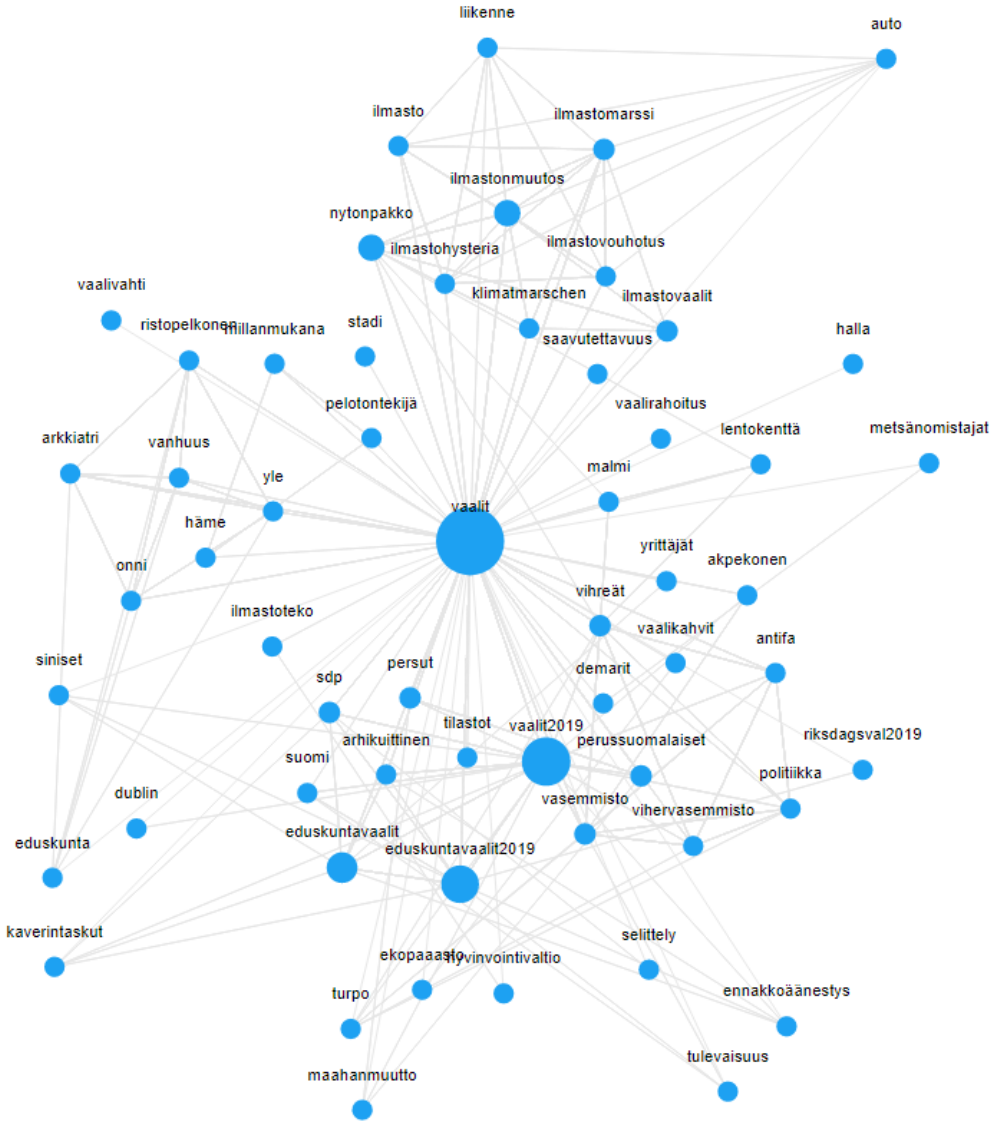
Graafin informatiivisuus riippuu pitkälti haetusta aihetunnisteesta ja sen käyttötavoista Twitterissä. Ajankohtaisiin aiheisiin liittyvät, yleiseen käyttöön vakiintuneet aihetunnisteet, kuten #yleuutiset ja #vaalit osoittautuivat varsin hyviksi hakusanoiksi, sillä näissä tapauksissa graafi välittää tarkastelijalla kokonaiskuvan päivän uutisaiheista tai vaalikeskustelun teemoista. Edellä mainitun kaltainen tilanne kuitenkin edellyttää jonkinlaista ennakkotietoa yleisesti käytetyistä aihetunnisteista, sillä summittaisella haulla sovellus harvoin tuottaa järkevää lopputulosta.

Sovellusta testattiin 6. huhtikuuta 2019 aihetunnisteella #vaalit ensin ilman twiittien suodatusta ja sen jälkeen suodatettuna. Suodattamattomasta graafista kuvassa 6 näkee, kuinka osa graafin solmuista korostuu selvästi, vaikka niiden aihetunnisteet ovat vaaliaiheeseen liittymättömiä ja spesifejä (esimerkiksi ruotsinkieliset aihetunnisteet #åbo ja #tammerfors). Ne myös ovat linkittyneet toisiinsa, mutta eivät juurikaan graafin muihin solmuihin juurisolmua lukuun ottamatta. Näiden havaintojen pohjalta voidaan todeta, että kyseiset aihetunnisteet ovat melko varmasti yhden ja saman henkilön twiiteistä peräisin ja niiden suodattamisen aihetunnisteiden määrän mukaan olevan varsin perusteltua. Aihetunnisteiden sallittu twiittikohtainen maksimimäärä sovelluksessa on kuusi.



Kuva 6. Aihetunnisteeseen #vaalit perustuvassa, suodattamattomassa graafissa vasemman puolen solmut ovat selvästi korostuneet, mutta ovat linkittyneet lähinnä toisiinsa.

Suodatettu graafi (kuva 7) on huomattavasti tasapainoisempi ja graafista pystyy nyt havaitsemaan erilaisia vaaliteemoja, ylivoimaisesti selvimpänä ilmastonmuutokseen liittyvän keskustelun. Ilmastoaihe vuoden 2019 eduskuntavaalien kynnyksellä oli odotetusti ykkösaiheena, mutta graafin luomishetkellä sen suosiota Twitterissä kasvatti entisestään kyseisenä päivänä (6.4.2019) järjestetty ilmastomarssi Helsingissä.



Kuva 7. Suodatettu graafi #vaalit-aihetunnisteella

4.4 Haasteet ja ongelmakohdat

Suurin osa insinöörityöhön liittyneistä teknisistä ongelmista liittyivät Twitterin dataan ja sen saatavuuteen. Vaikka Twitterin datan saatavuus on moniin muihin sosiaalisiin medioihin nähden varsin hyvä, on sitä Twitter API:ssa rajoitettu ilmaiskäyttäjille. Yhden API-kutsun twiittimäärän rajoitus (100) korostui erityisesti tilanteissa, joissa saatavasta datasta suodattui useita twiittejä pois, usein jopa yli puolet. Tämän lisäksi viikkoa vanhempia twiittejä ei ilmaiskäyttäjiä pysty rajapinnan kautta hakemaan. Tällöin datan määrä voi olla liian pieni verkkograafia ajatellen. API-kutsujen ketjuttaminen olisi luultavasti mahdollistanut suuremmat datamäärät suosituilla aihetunnisteilla, mutta tätä ei nähty työssä oleelliseksi toteuttaa.

Rajapinnan sadan twiitin rajoituksesta huolimatta sovelluksessa ilmeni skaalautuvuusongelmia tilanteissa, joissa aihetunnisteita oli huomattavan paljon. Esimerkiksi geneeriset, yleisesti mainontaan käytettävät aihetunnisteet, kuten #fashion, toivat graafiin aluksi huomattavan määrän solmuja ja linkkejä, sillä tämänkaltaisia aihetunnisteita käytetään Twitterissä yleensä yhdessä monen muun aihetunnisteen kanssa. Sovelluksen tapahtumapohjaisissa funktioissa (esimerkiksi solmun klikkaus) tämä aiheutti viiveen, joka heikensi selvästi käyttökokemusta. Ongelman korjaamiseksi yli sadan aihetunnisteen visualisoinneissa päätettiinkin vain huomioida ne aihetunnisteet, jotka oli mainittu lähdedatassa vähintään kaksi kertaa.

Elokuussa 2018 Twitter sulki osan API:n toiminnoista, muun muassa twiittien striimauksen (Newton, 2018). Tämä vaikutti suoraan työn tekemiseen, sillä alkuperäinen, osittain jo toteutettu idea oli visualisoida graafi reaaliajassa. Jälkikäteen ajateltuna reaaliaikaisuus ei olisi kuitenkaan tuonut graafiin varsinaista lisäarvoa, vaan olisi pikemminkin tehnyt siitä hankalamman tarkastella datan jatkuvasti päivittyessä. On myös syytä todeta, ettei reaaliaikaisen twiittivirran informaatioarvo oleellisesti eroa esimerkiksi viimeisimmästä 50 twiitistä varsinkaan suosituilla aihetunnisteilla. Sovelluksessa on kuitenkin mahdollisuus päivittää graafi uudella haulla.

Myös aihetunnisteiden käyttäjäperäisyys vaikeutti asianmukaisen visualisoinnin saavuttamista. Koska käyttäjät määrittävät itse twiittiensä aihetunnisteet, voi suosituille puheenaiheille olla useita samaa tarkoittavia aihetunnisteita. Tämä näkyy muun muassa kuvassa 7. Twitterin käyttäjien käyttäessä näitä aihetunnisteita toisinaan yhtä aikaa twiittinsä näkyvyyden maksimoimiseksi tulee graafiin tarpeetonta toistoa, ja sen informatiivisuus heikkenee. Ideaalitapauksessa visualisoinnissa olisi vain yksi aihetunniste per aihe, mutta tämänkaltaisen lopputuloksen saavuttaminen olisi vaatinut huomattavasti laajempaa datan käsittelyä.

5 Yhteenveto

Insinöörityössä pyrittiin visualisoimaan Twitterin aihetunnisteet verkkograafina ja raportoimaan visualisointiprosessin työvaiheet. Nämä tavoitteet työssä suurilta osin myös saavutettiin. Graafi osoittautui oikeantyyppiseksi visualisoinniksi aihetunnisteiden kaltaiselle verkottuneelle datalle ja D3 monipuoliseksi työkaluksi sen toteuttamiseen. Aihepiiriltään työ sivusi niin ohjelmistotekniikkaa kuin tietojenkäsittelytiedettä.

Tutustuminen visualisointikirjasto D3:een jäi työssä hieman pintaraapaisuksi tekniikan laajuudesta ja käyttömahdollisuuksien suuresta määrästä johtuen. Pääpaino raportin teoriaosuudessa olikin datan sidontaan sekä DOM-manipulointiin liittyvän toiminnallisuuden kuvaamisessa.

Ongelmia työn aikana aiheuttivat niin lähdedatan saatavuus ja luonne kuin graafin suorituskyky suurimmilla datamäärillä. Datan soveltuvuutta visualisointiin pyrittiin parantamaan muun muassa suodattamalla sitä eri ehdoilla. Suodatus paransi graafin informatiivisuutta ja luettavuutta, mutta datan käyttäjäperäisyyden vuoksi graafin onnistuneisuus riippui paljon haetusta aihetunnisteesta.

Sovelluksessa jäi kehitettävää muun muassa graafin suorituskyvyn tehostamisen sekä sen visuaalisen vaikuttavuuden parantamisen suhteen. Sovellusteknisesti datan käsittelyn voisi toteuttaa hyödyntämällä JavaScriptin async/await-ominaisuutta. Tällöin ohjelmakoodissa voitaisiin välttää takaisinkytkennät (*callbacks*), mikä todennäköisesti parantaisi koodin luettavuutta ja helpottaisi virheenkäsittelyä (Parsons, 2019). Mahdollisia keinoja visuaalisuuden parantamiseen olisivat esimerkiksi animoinnin ja värien käytön lisääminen ja graafin toteuttaminen kolmiulotteisena.

Lähteet

Ændrew Rininsland, S. T. (2017). *D3.js 4.x Data Visualization - Third Edition.* Packt.

Bostock, M. *General Update Pattern, I*. Noudettu osoitteesta bl.ocks.org: https://bl.ocks.org/mbostock/3808218. Luettu 1.10.2018.

Cook, P. *D3 In Depth - Enter and exit*. Noudettu osoitteesta https://d3indepth.com/enterexit/. Luettu 1.10.2018.

Cook, P. *D3 In Depth - Force Layout*. Noudettu osoitteesta http://d3indepth.com/force-layout/. Luettu 12.3.2018.

MacArthur, A. *Why Using reduce() to Sequentially Resolve Promises Works - CSS Tricks*. Noudettu osoitteesta https://css-tricks.com/why-using-reduce-to-sequentially-resolve-promises-works/. Luettu 24.3.2019.

Newton, C. *Twitter officially kills off key features in third-party apps*. Noudettu osoitteesta https://www.theverge.com/2018/8/16/17699626/twitter-third-party-apps-streaming-api-deprecation. Luettu 19.8.2018.

Parsons, N. *Hackernoon - JavaScript: Promises and Why Async/Await Wins the Battle*. Noudettu osoitteesta https://hackernoon.com/javascript-promises-and-why-async-await-wins-the-battle-4fc9d15d509f. Luettu 15.4.2019.

Peltomäki, J. (2017). Javascript-kieli - Uudet ominaisuudet. Books on Demand.

Roth, T. *Bounding box force directed graph I*. Noudettu osoitteesta https://bl.ocks.org/puzzler10/2531c035e8d514f125c4d15433f79d74. Luettu 6.4.2019.

Twitter. *Tweet Objects - developer.twitter.com*. Noudettu osoitteesta https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html. Luettu 8.10.2018.

Twitter for Node.js - Npm. Noudettu osoitteesta https://www.npmjs.com/package/twitter. Luettu 11.2.2019.

Liite 1. Datan käsittelyn funktiot

```
/**
 * @param {string} searchTerm. User-entered searchTerm i.e. hashtag
 */
function getTweets(searchTerm) {
    client.get('search/tweets', { q: searchTerm, count: 100 }, function (er-
ror, tweets, response) {

        if (response.statusCode == 200 && !error && tweets.statuses.length) {

            tweetCount = tweets.statuses.length;

            [...Array(tweetCount)].reduce((p, _, i) =>
                p.then(_ => new Promise(resolve => {
                    if (i == 0) { timeStampLast = getTweetDate(tweets.sta-
tuses[i]); }
                    if (i == tweetCount - 1) { timeStampFirst = getTweet-
Date(tweets.statuses[i]); }
                    if (isTweetValid(tweets.statuses[i])) {
                        const validTweet = isTweetValid(tweets.statuses[i]);
                        countHashtags(validTweet, function () {
                            resolve();
                        });
                    }
                    else {
                        resolve();
                    }
                    if (i == tweetCount - 1) {
                        printResults();
                    }
                }))
                , Promise.resolve());
        }
        else {
            io.sockets.emit('update', false);
        }
    })
}
```

```
function printResults() {

    let hashtagsFinal = [];
    let linksFinal = [];

    if (hashtags.length && links.length) {
        hashtagsFinal = hashtags;

        if (hashtags.length >= 100) {
            hashtagsFinal = hashtags.filter(hashtag => hashtag.count > 1);
        }

        linksFinal = links.filter(link => {
            return link.target < hashtagsFinal.length && link.source <
hashtagsFinal.length;
        });

        let graphData = {
            'hashtags': hashtagsFinal,
            'links': linksFinal,
            'timestamp': {
                'start': timeStampFirst,
                'end': timeStampLast
            }
        }
        io.sockets.emit('update', graphData);

    }
}


/**
 * @param {JSON} tweet. Single tweet
 * @callback. Return to getTweets() and resolve current promise in the loop
 */
function countHashtags(tweet, callback) {
    tweetHashtagIds = [];
    if (!rootAdded) {
        hashtags.push({
            "hashtag": searchTerm.split('#')[1].toLowerCase(),
            "count": 0
        });
        rootAdded = true;
    }
    var linkPromises = [];

    for (i = 0; i < tweet.hashtags.length; i++) {
        linkPromises.push(linkHashtags(i, tweet.hashtags[i]));
    }
    Promise.all(linkPromises)
        .then(() => {
            callback();
        })
        .catch((e) => {
            console.log('Error at function linkHastags: ' + e);
        });
}
```

```
/**
 * @param {number} value. An index value of linkPromises array to resolve
 * @param {string} hashtag.
 */
function linkHashtags(value, hashtag) {
    var ht = hashtag;
    return new Promise((resolve) => {
        var hashtag = ht.toLowerCase();
        var newHashtag = {
            "hashtag": hashtag,
            "count": 1
        };
        var found = hashtags.findIndex(e => e.hashtag.toLowerCase() ==
hashtag.toLowerCase());
        if (found == -1) {
            hashtags.push(newHashtag);
        }
        else {
            hashtags[found].count += 1;
        }


        findHashtagIndex(hashtags, hashtag, function (target) {
            tweetHashtagIds.push(target);

            if (tweetHashtagIds.length) {
                tweetHashtagIds.forEach(function (sourceId, index1) {
                    tweetHashtagIds.forEach(function (targetId, index2) {
                  if (index1 == 0) {
                      links.push({ source: 0, target: targetId });
                  }
                  if (sourceId != targetId && sourceId != 0 && targetId != 0) {
                      newLink = { source: sourceId, target: targetId };
                      links.push(newLink);
                  }
                    });
                });
            }
        });
        resolve(value);
    });
}

/**
 * @param {JSON} hashtags
 * @param {string} hashtag. Current hashtag in the linkPromises array
 * @callback. Proceed to linking process in nested loops
 */
function findHashtagIndex(hashtags, hashtag, callback) {
    var target = hashtags.findIndex(e => e.hashtag == hashtag);
    callback(target);
}
```

```
/**
 * @param {JSON} tweetData. JSON-formed tweet data from Twitter API
 * @returns {string} Formatted timestamp
 */
function getTweetDate(tweetData) {
    date = new Date(tweetData.created_at);
    year = date.getFullYear();
    month = date.getMonth() + 1;
    dt = date.getDate();
    hours = date.getHours();
    minutes = date.getMinutes();

    if (dt < 10) {
        dt = '0' + dt;
    }
    if (month < 10) {
        month = '0' + month;
    }
    if (minutes < 10) {
        minutes = '0' + minutes;
    }

    return (dt + '.' + month + '.' + year + ' ' + hours + ':' + minutes);
}



/**
 * @param {JSON} tweetData. JSON-formed tweet data from Twitter API
 * @returns {JSON} tweetHastags or false
 */
function isTweetValid(tweetData) {
    if (typeof tweetData !== "undefined" && tweetData.entities.hashtags.length
&& tweetData.entities.hashtags.length <= 6) {
        var tweet = tweetData.text;
        if (!isClean(tweet)) {
            return;
        } else {
            var hashtags = new Array();
            if (tweetData.entities.hashtags.length) {
                for (var i = 0; i < tweetData.entities.hashtags.length; i++) {
                    hashtags.push(tweetData.entities.hashtags[i].text);
                }
            }
            else {
                return;
            }
            var tweetHashtags = {
                hashtags: hashtags
            };
            return tweetHashtags;
        }
    }
    return;
}
```

```
/**
 * @param {string} tweet
 * @returns {boolean} true or false based on string content
 */
function isClean(tweet) {
    filter = ["porn", "nsfw", "trending"];
    for (i = 0; i < filter.length; i++) {
        if (tweet.includes(filter[i])) {
            return false;
        }
    }
    if (tweet.substring(0, 9) == "Retweeted"){
        return false
    }
    return true;
}
```