Opinnäytetyö (AMK)

Tietojenkäsittely

Yrityksen tietojärjestelmät

2015

Ville Kohonen

REAALIAIKAISEN SYÖTESOVELLUKSEN TOTEUTUS METEORJS:LLÄ

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma | Yrityksen tietojärjestelmät

2015 | 28

Tuomo Helo

Ville Kohonen

REAALIAIKAISEN SYÖTESOVELLUKSEN TOTEUTUS METEORJS:LLÄ

Opinnäytetyön tavoitteina on luoda reaaliaikainen syötesovellus Meteor-sovelluskehyksellä ja selvittää, mitä hyötyjä ja mahdollisia haasteita sen käyttöönotossa on sovelluskehittäjälle. Meteor on uusi JavaScript-verkkosovelluskehys, joka tekee reaaliaikaisten sovelluksien luomisesta nopeaa ja vaivatonta. Opinnäytetyön toimeksiantajana on WorldSome Oy.

Teoriaosuudessa käsitellään projektissa käytettyjä teknologioita sekä myös uutta vuorovaikutteista datavisualisointia, sen tärkeyttä ja käyttöönottoa.

Empiirinen osa koostuu syötesovelluksen suunnittelusta ja toteutuksesta. Työssä esitellään toteutusvaiheessa aiheutuneita haasteita ja kuinka ne on ratkaistu Meteoriin lisättävillä lisäosan tapaisilla paketeilla. Opinnäytetyön tuloksena syntyi reaaliaikainen syötesovellus verkkoselaimelle, ja näin ollen projektille asetettuihin tavoitteisiin päästiin.

ASIASANAT:

MeteorJS, datavisualisointi, JavaScript, sovelluskehys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Business Information Systems

2015 | 28

Tuomo Helo

Ville Kohonen

IMPLEMENTING A REAL-TIME FEED APPLICATION WITH METEORJS

The objectives of this thesis were to create a real-time feed application using a Meteor framework and to determine what benefits and possible challenges there are in using it. Meteor is a new JavaScript web application framework, which assists in developing real-time applications swiftly and effortlessly. The client of the thesis was WorldSome Oy.

The theoretical part of the thesis reviews different technologies used in creating the real-time feed application, and it also introduces interactive data visualization, its importance, and implementation.

The empirical part consists of designing and implementing the feed application. The thesis presents challenges occurred during the development stage and how they were solved using plugin-like packages in Meteor. As a final result, the real-time feed application for a web browser was created and so the objectives set for the project were achieved.

KEYWORDS:

MeteorJS, data visualization, JavaScript, application framework

SISÄLTÖ

KUVAT

KUVIOT

TAULUKOT

1 JOHDANTO

Tässä opinnäytetyössä perehdytään projektiin, jossa suunniteltiin ja toteutettiin reaaliaikainen syötesovellus WorldSome Oy:lle. Projektin tavoitteena oli toteuttaa reaaliaikainen ja helppokäyttöinen syötesovellus selaimelle, mikä toimisi palvelujen toimituskanavana asiakkaille. Sovelluksen tuli hakea dataa tietokannasta ja näyttää se asiakkaalle reaaliajassa. Asiakkaan tuli myös pystyä tuomaan dataa ulos sovelluksesta. Sen lisäksi sovelluksessa tuli olla datan parempaa ymmärtämistä varten helposti ymmärrettäviä ja vuorovaikutteisia kaavioita.

Työ toteutettiin Meteor-sovelluskehyksellä (framework), koska se on oletukseltaan reaaliaikainen. Tietokannaksi valittiin MongoDB, koska Meteor käyttää sitä oletuksena ja koska se sopi toimeksiantajan sovellusvaatimuksiin. Sivuston materialistinen ulkoasu toteutettiin Bootstrap-kehyksellä.

Opinnäytetyön tavoitteena on selvittää, mitä hyötyjä verkkosovelluskehityksessä on saavutettavissa Meteor-sovelluskehyksen käyttöönotolla ja mitä mahdollisia haasteita sen käyttöönotto asettaa sovelluskehittäjälle. Opinnäytetyössä käsitellään myös uutta vuorovaikutteista visualisointia, mitä se on, miksi se on tärkeää ja miten sitä pystyy toteuttamaan.

Opinnäytetyön toisessa luvussa kuvataan toimeksiantajan yritystä, toimialaa ja toimeksiantoa. Luvut 3-5 käsittelevät projektissa käytettyjä teknologioita ja luvussa 6 käsitellään syötesovelluksen toteutusta. Luvussa 7 on pohdintaa koko projektiin liittyen.

2 WORLDSOME OY

Opinnäytetyön toimeksiantajana toimiva WorldSome Oy on suomalainen start-tup-yritys, joka tarjoaa analysoituja reaaliaikaisia tietoja asiakkaidensa yrityksistä, tuotteista, markkinoista ja kilpailijoista runsaiden sosiaalisten medioiden ja uutislähteiden kautta.

WorldSomen palveluilla asiakas voi

- parantaa yrityksensä imagoa
- keskittyä olennaisiin asioihin yrityksen parantamisessa
- tunnistaa markkinapotentiaaleja
- olla parempi kuin kilpailijansa.

Toimeksiantaja otti yhteyttä 2015 vuoden maaliskuussa. He kysyivät, olisinko halukas toteuttamaan reaaliaikaisen syötesovelluksen, joka hakee tietokannasta dataa ja julkaisee sen automaattisesti asiakkaan syötteeseen, vähän niin kuin Facebookissa tulee uusia tilapäivityksiä käyttäjän aikajanalle.

Toimeksiantajan kanssa tavattiin kasvotusten muutamia kertoja keväällä 2015 ennen töiden aloittamista.

3 SOVELLUSKEHYKSET JA JAVASCRIPT

JavaScript

Nykyään lähes kaikki interaktiivisuus mitä käyttäjä kokee verkossa on JavaScriptin ansiota. JavaScript luotiinkin tekemään verkkosivuista elävämpiä (Rauschmayer 2015). Interaktiivisten käyttöliittymien luomisen helpottamiseksi ilmestyi JavaScript-kirjastoja kuten Prototype tai jQuery.

JavaScript-kirjastojen ansiosta web-suunnittelijoilla meni käyttöliittymien tekemiseen vähemmän aikaa kuin ennen ja näin he pystyivät keskittymään paremmin palvelinpuolen koodin kirjoittamiseen. Kun pikakelataan ajassa eteenpäin, JavaScriptin tehokkuus sekä suosio ovat kasvaneet huomattavasti ja JavaScriptistä on tullut selainten de facto skriptauskieli (W3Techs 2015).

Koska kokonaisten verkkosivujen tai sovellusten luominen JavaScript-kirjastolla, kuten jQuery, muuttuisi ennen pitkää sekavaksi ja vaikeasti ylläpidettäväksi, on ongelman ratkaisemiseen kehitetty JavaScript-sovelluskehyksiä. (Osmani 2012.)

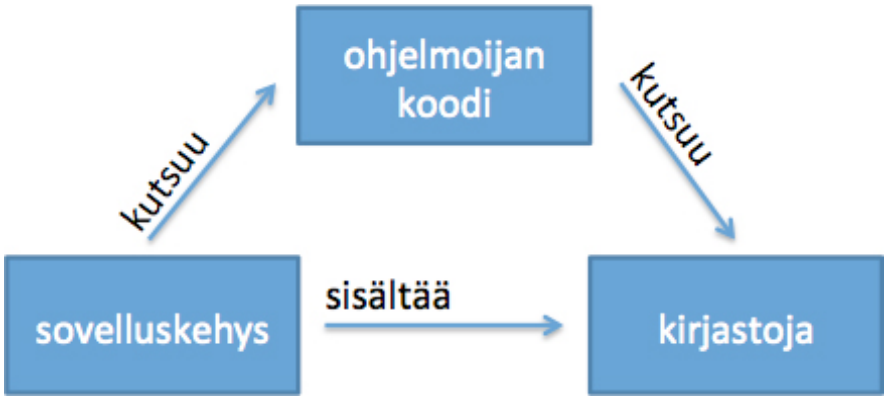
JavaScript-sovelluskehykset ja -kirjastot

JavaScriptin kasvavan suosion myötä selainpuolen sovellukset ovat tulleet paljon monimutkaisemmiksi kuin ennen. Sovelluskehitys vaatii useiden koodareiden välistä yhteistyötä. Tästä syystä ylläpidettävän ja uudelleenkäytettävän koodin kirjoittaminen on hyvin tärkeää modernille webille. (Google Chrome Developers 2015.)

Suunnittelumallit ovat tärkeitä ylläpidettävän ja uudelleenkäytettävän koodin kirjoittamiseen. Viime vuosina on putkahtanut esiin runsas määrä jonkin muotoista MVC-mallia hyödyntäviä JavaScript-sovelluskehyksiä, kuten Backbone.js, Ember.js ja AngularJS. (Google Chrome Developers 2015.)

Sovelluskehys on kokoelma kirjastoja, jotka yhdessä luovat huolellisesti määritellyn rajapinnan sovellusohjelmointiin, esimerkiksi verkkosovellusten luomiseen.

Kirjastot pitävät sisällään hyödyllisiä funktioita, joita kehittäjä voi tarvittaessa kutsua omassa sovelluksessaan. Käyttämällä kirjastoja kehittäjä säästää aikaa, koska hänen ei tarvitse kirjoittaa kirjastoissa olevia funktioita itse. (Mysore 2015.)



Kuvio 1. Käänteinen kontrolli.

Sovelluskehykset ja kirjastot eroavat yksityiskohtaisin piirtein, kuten kuviossa 1 nähtävällä käänteisellä kontrollilla (inversion of control). Kirjastojen ollessa pelkkiä funktioita, kehittäjä on täydessä kontrollissa sovelluksessaan kutsuttavista funktioista. Käänteisellä kontrollilla tarkoitetaan ilmiötä, jossa funktioita kutsutaan ja ajetaan sovelluskehyksen sisällä, milloin sovelluskehys onkin kontrollissa funktioista sovelluskehittäjän sijaan. (Fowler 2005.)

4 METEOR

Meteor, toiselta nimeltään MeteorJS, on reaktiivinen, yksinkertainen ja tehokas sovelluskehys, jolla voidaan tuottaa kehittyneitä ja tehokkaita verkkosovelluksia vain muutamilla riveillä koodia. (Strack 2012, 1.)

Meteor on vuonna 2011 perustettu startup-yritys, joka on menestyksekkään yritys-hautomo YCombinatorin suojassa. Vuonna 2012 Meteor sai 11.2 miljoonan dollarin sijoituksen arvostetulta pääomasijoitusyhtiöltä nimeltä Andressen Horowitz. Meteorin perustajiin ja ensimmäisiin työntekijöihin lukeutuu ihmisiä Googlelta, Asanalta sekä Etherpadin, nykyisin Titanpadinä tunnetun sovelluksen luoja. (Dascalescu 2015.)

Meteorin valitsemista opinnäytetyössä käytettäväksi sovelluskehykseksi puoltaa sen yhteisön laajuus ja viime vuosien kehitysvauhti. Meteorin 1.0 versio julkaistiin talvella 2014. Kolme päivää myöhemmin, Meteor oli saanut jo yli 21 tuhatta tähteä eli tykkäystä Githubissa ja sijoittui näin eniten tähtiä saaneiden projektien listalle yhdenneksitoista. Viikkoa myöhemmin yli 4000 sovelluskehittäjää 130 eri kaupungista ja 40 eri maasta kokoontui henkilökohtaisesti viettämään Meteorin päivää. (Dascalescu 2015.)

Opinnäytetyön kirjoitushetkellä syksyllä 2015, Meteorilla oli

- 434 000 asennuskertaa
- 129 tulevaa tapahtumaa
- 28611 tähteä Githubissa.

Meteor hoitaa kaikki vaikeat ja arkipäiväiset verkkosovelluskehitykseen kuuluvat askareet sovelluskehittäjän puolesta, käyttäen vakiintuneita ja oikeaksi todettuja sovelluskehityksen suunnittelumalleja. Tämän ansiosta kehittäjän ei tarvitse käyttää aikaansa esimerkiksi jälleen kerran uuden tietokantarajapinnan kirjoittamiseen tai uuden sivupohjamoottorin opiskeluun, vaan hän pystyy keskittymään itse sovelluksen tekemiseen käyttäen Meteorin tarjoamia viimeisimpiä innovaatioita. Näitä ovat esimerkiksi

- reaktiivinen ohjelmointi
- sivupohjat
- lisäosat
- datan välimuistittaminen (caching) ja synkronoiminen selaimessa (Strack 2012, 1).

Node.js:llä luotu Meteor on suunniteltu alusta asti isomorfiseksi. Tällä tarkoitetaan sitä, että kaiken sovelluskoodin tulisi toimia niin palvelimella kuin myös selaimella. Isomorfisuudella on tietenkin rajansa, sillä eivät esimerkiksi selaimet itsessään voi lähettää sähköposteja, kuten eivät myöskään palvelimet voi napata käyttäjien hiirenliikkeitä. (Meteor.com 2015a.)

Meteor pitää sisällään seitsemän keskeistä periaatetta:

1. Dataa langalla (Data on the Wire). Meteor ei lähetä HTML-koodia verkon yli. Palvelin lähettää datan ja antaa selaimen renderöidä sen.
2. Yksi kieli (One Language). Meteor antaa sovelluskehittäjän kirjoittaa niin palvelin- kuin selainpuolen osat sovelluksestaan JavaScriptillä.
3. Tietokanta kaikkialla (Database Everywhere). Sovelluskehittäjä voi hyödyntää samoja metodeja niin palvelin- kuin selainpuolella käyttääkseen sovelluksensa tietokantaa.
4. Viiveen kompensointi (Latency Compensation). Meteor ei koskaan odota palvelimelta vastauksia jatkaakseen, vaan yrittää ennustaa palvelimen vastauksen käyttämällä saatavissa olevia tietoja.
5. Full Stack -reaktiivisuus (Full Stack Reactivity). Meteorissa kaikki toimii oletukseltaan reaaliaikaisesti. Kaikki tasot tietokannasta mallipohjiin päivittävät itse itsensä automaattisesti.
6. Ekosysteemiä syleillen (Embrace the Ecosystem). Meteor on avoimen lähdekoodin projekti ja on integroitavissa muiden avoimen lähdekoodin työkalujen ja sovelluskehysten kanssa.
7. Yksinkertaisuus yhtä kuin tuottavuus (Simplicity Equals Productivity) (Meteor.com 2015b).

4.1 Meteorin arkkitehtuuri ja rakenne

Meteor-sovellusta rakennettaessa Meteor etsii ja pakkaa oletuksena kaikki projektiin kuuluvat JavaScript-tiedostot yhdeksi nipuksi ja lähettää ne eteenpäin palvelimelle ja selaimelle. Meteor-sovelluksen hakemistorakenteen luomiseen ei ole yhtä ja oikeaa tapaa, eikä Meteor myöskään pakota kehittäjää noudattamaan mitään erityissääntöjä sitä luodessa, mutta tiedostojen nimet ja kansiot vaikuttavat tiedostojen pakkausjärjestykseen. (Meteor.com 2015c.)

Taulukossa 1 on esitetty lista kansioista, joita Meteor käsittelee erityisellä tavalla.

Taulukko 1. Kansiot joita Meteor käsittelee erityisellä tavalla (Meteor.com 2015c).

Kansio	Kansiossa olevat tiedostot
client	Ajetaan vain selaimella.
server	Ajetaan vain palvelimella.
public	Tarjotaan sellaisenaan selaimelle. Hyvä paikka tiedostoille, kuten favicon.ico tai robots.txt.
private	Ovat saatavissa ainoastaan Assets-rajapinnan kautta. Tiedostoihin on mahdotonta päästä käsiksi muulla tavalla.
compability	Ajetaan ennen muita JS-tiedostoja uuteen viittausalueeseen (variable scope) pakkaamatta.
tests	Pysyvät koskemattomina. Kansio tarkoitettu lokaalisti suoritettavia testejä varten.
node_modules	Pysyvät koskemattomina. Kansio tarkoitettu Meteorin rinnalla ajettaville Node.js-sovelluksille.

Taulukossa 1 esiteltyjen kansioiden ulkopuolelle jäävät JavaScript-tiedostot ajetaan sekä selaimella että palvelimella. Ulosjääviin tiedostoihin kuuluu Meteor-sovelluksessa käytettävien mallien (models) määrittelyt ja muut funktiot. Ulosjäävät HTML- ja CSS-tiedostot ladataan vain selaimelle, eikä niitä voida käyttää palvelinpuolen koodissa. Meteorissa on Meteor.isClient- ja Meteor.isServer-muuttujat, joiden avulla sovelluskehittäjä voi määritellä, mitä osia koodista ajetaan selaimella ja mitä palvelimella. (Meteor.com 2015c.)

Tiedostojen latausjärjestys

Meteor-sovellus tulisi kirjoittaa siten, ettei se välitä, missä järjestyksessä sen tiedostot ladataan. Tämä on mahdollista toteuttaa esimerkiksi Meteor.startup-funktiolla, joka ajetaan aina Meteor-sovelluksen käynnistyessä, tai sitten tekemällä paketteja (packages), joiden latausjärjestystä kehittäjä pystyy yksityiskohtaisesti hallitsemaan. (Meteor.com 2015c.)

Meteorilla on useita sääntöjä projektissa olevien tiedostojen lataamiseen ja pakkaamiseen, ja niitä sovelletaan kaikkiin sovellettaviin tiedostoihin seuraavanlaisessa järjestyksessä:

1. HTML-sivupohjat (templates) ladataan aina ensimmäisenä.
2. Kaikki main-nimiset tiedostot ladataan viimeisenä.
3. Tiedostot lib-kansiossa ladataan seuraavaksi.
4. Tiedostot syvimmällä kansiorakenteessa ladataan seuraavaksi.

Esimerkki näistä säännöistä on esitetty alla. Tiedostot on järjestetty siihen järjestykseen, missä Meteor lataa ne. Tiedostoihin client/lib/style.js ja lib/feature/styles.js pätee sama sääntö numero neljä, mutta client-kansio ladataan ennen lib-kansiota koska se on aakkosissa ennen libiä. Esimerkki tiedostojen latausjärjestyksestä:

1. nav.html
2. main.html
3. client/lib/methods.js

4. client/lib/styles.js
5. lib/feature/style.js
6. lib/collections.js
7. client/feature-y.js
8. feature-x.js
9. client/main.js

4.2 Live HTML-sivupohjat

HTML-sivupohjat ovat keskeinen asia verkkosovelluksille. Meteorin Blaze-niminen reaktiivinen käyttöliittymäkirjasto (UI library) mahdollistaa HTML:n reaktiivisen renderöinnin. Reaktiivinen renderöinti tarkoittaa sitä, että kun sivupohjaan tehdään muutoksia, niin Blaze huomaa muutokset ja renderöi sivupohjan automaattisesti uudelleen. Sovelluskehittäjän ei siis tarvitse itse päivittää sovellusta nähdäkseen muutokset, vaan Meteor hoitaa sen hänen puolestaan. (Meteor.com 2015d.)

Meteorin mukana tulee sivupohjakieli (templating language) nimeltä Spacebars, joka on saanut vaikutteita minimaaliselta ja logiikattomalta Handlebars-sivupohjakieleltä, joka pitää koodin erossa sivupohjista. Spacebars on ajatukseltaan ja syntaksiltaan samankaltainen kuin Handlebars, mutta se on räätälöity tuottamaan reaktiivisia sivupohjia Meteorissa. (Meteor.com 2015d.)

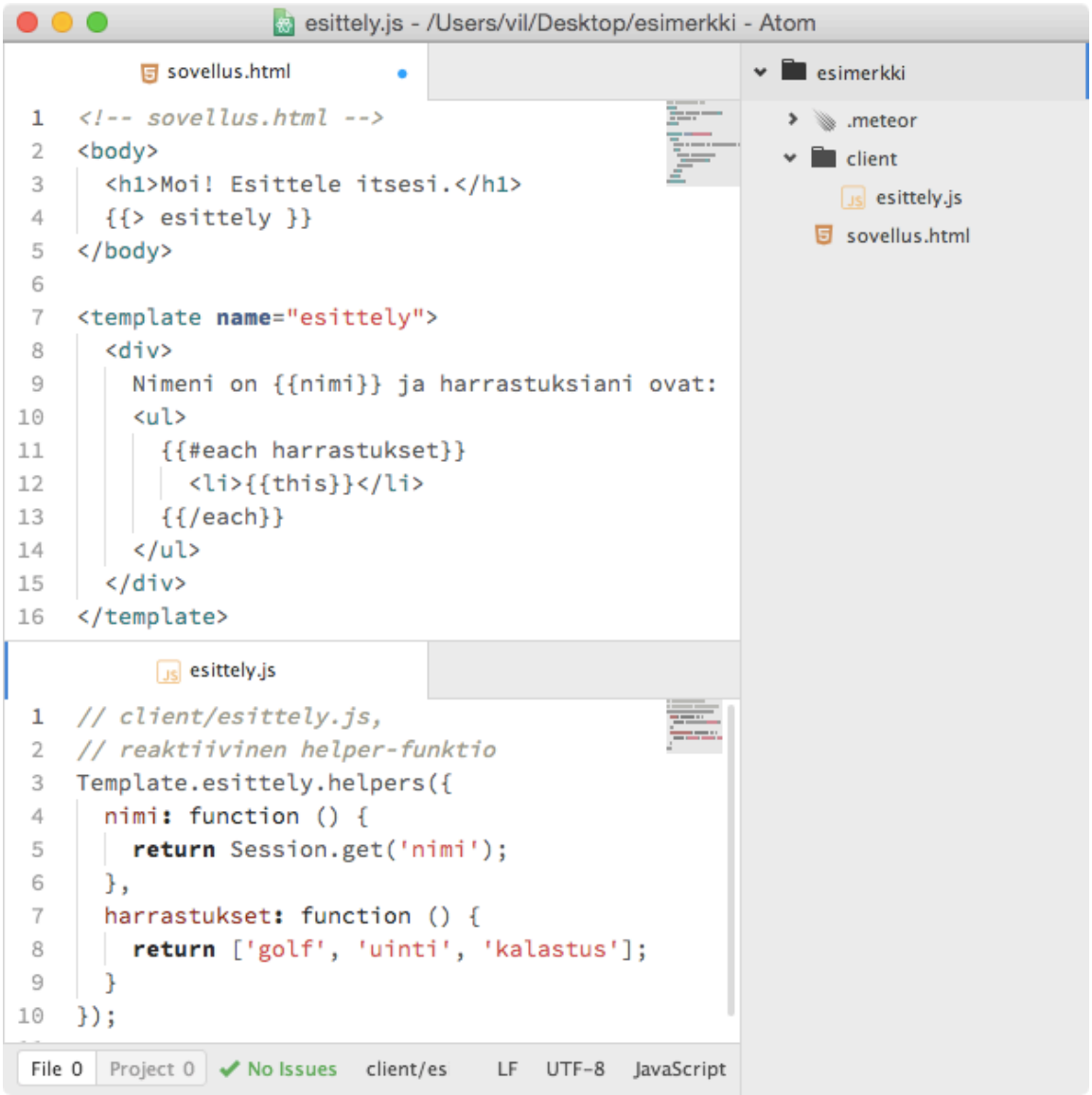
Opinnäytetyön kirjoitushetkellä syksyllä 2015 Meteorin mukana ei tullut muita sivupohjakieliä Spacebarsin kanssa, mutta Meteorin kehittäjäyhteisö oli luonut paketteja toisille sivupohjakielille, kuten Jade. (Meteor.com 2015d.)

Sivupohja tehdään luomalla .html-päätteinen tiedosto, jonka sisälle luodaan template-elementti ja sen name-attribuuttiin määritellään sivupohjan nimi. Sivupohjan sisältö laitetaan template-elementin sisään. Kun tiedosto tallennetaan, Meteor lähettää sen automaattisesti selaimelle. (Meteor.com 2015d.)

Kun Meteor-sovellus avataan, se automaattisesti renderöi body-nimisen erikois-sivupohjan, jonka sisältö on kirjoitettu normaalista sivupohjasta poiketen nimen-

sä mukaan body-elementin sisään. Sivupohjiin lisätään toisia sivupohjia käyttämällä Spacebarsin {{> toisenSivupohjanNimi }}-operaattoria. Sen lisäksi että sovelluskehittäjä voi luoda myös omia operaattoreita, Spacebarsin mukana tulevat {{#each}}-, {{#if}}-, {{#with}}- ja {{#unless}}-operaattorit (Meteor Development Group 2015).

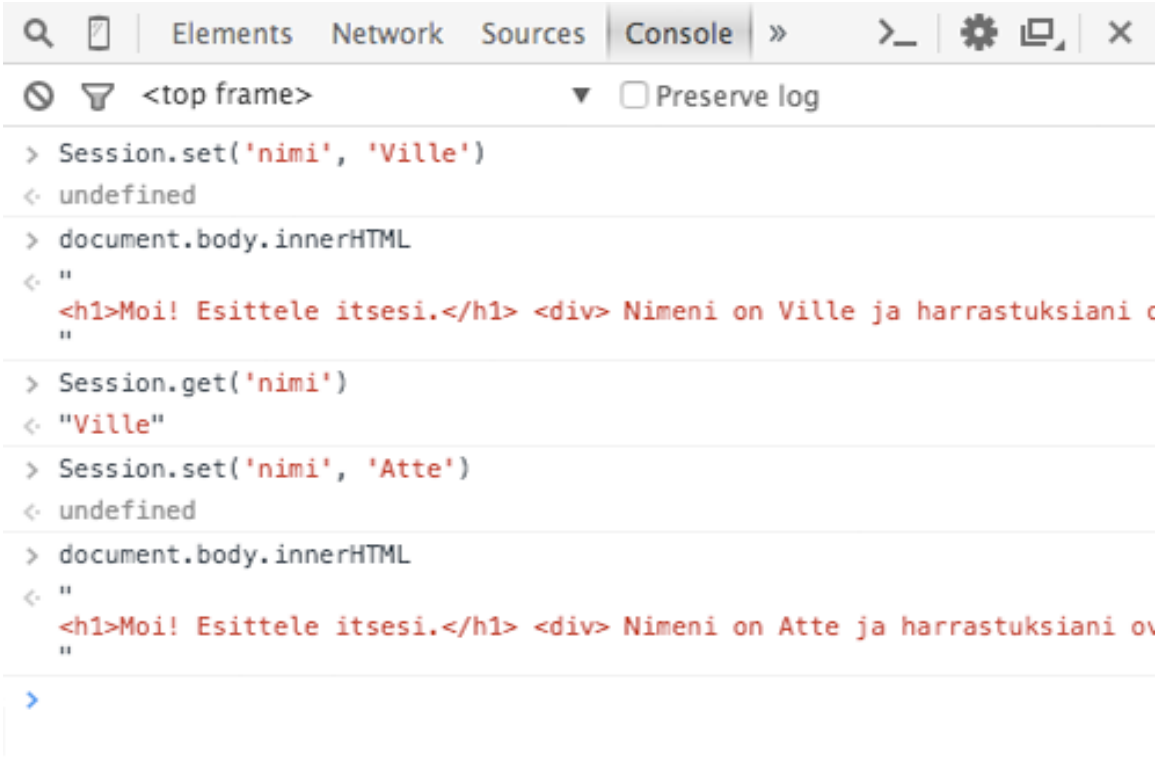
Sivupohjiin saadaan helpoiten dataa kirjoittamalla helper-funktioita JavaScriptillä. Helper-funktiot määritellään Template.sivupohjanNimi.helpers-funktiossa.



Kuva 1. Esimerkkisovellus.

Kuvassa 1 on yksinkertainen esimerkki Meteor-sovelluksesta, missä sivulle renderöidään henkilön nimi ja harrastukset. Sovellus.html-tiedoston rivillä 4 lisätään esittely-sivupohja sivulle. Esittely-sivupohjassa haetaan dataa esittely.js-tiedostossa sijaitsevista helper-funktioista käyttämällä {{ helperFunktionNimi }} -operaattoria. Taulukoita (array) ja tietokannan kursoreita (cursor) iteroidaan käyttäen {{#each}}-operaattoria. Taulukoita iteroitaessa taulukon soluun viitataan operaattorilla {{this}}.

Kun kuvan 1 esimerkkisovellus ajetaan (sivu ladataan), henkilön nimi on tyhjä. Se johtuu siitä, että esittely.js:n rivillä 5 nähtävän Session-objektin "nimi"-avaimeen ei ole asetettu arvoa. Session on reaktiivinen globaali objekti, johon voidaan tallettaa avain-arvo-pareja (Meteor.com 2015e). Session-objekti tyhjenee, jos sivun lataa uudelleen. Session-objektiin syötetään arvoja Session.set-funktiolla. Esimerkkisovellukseen hyvä parannus olisi ollut nimensyöttökenttä, mutta nimen syöttäminen onnistuu myös JavaScript-konsolissa kuvan 2 mukaisesti.



Kuva 2. Esimerkki Session.set-funktion käytöstä JavaScript-konsolissa.

4.3 Kokoelmat

Kokoelmat (Collections) ovat Meteorin tapa varastoida pysyvää dataa. Erikoista kokoelmista Meteorissa tekee se, että niihin voi päästä käsiksi palvelinpuolen lisäksi myös selainpuolella. Tämä mahdollistaa sovelluskehittäjälle helpon tavan kirjoittaa näkymä-logiikkaa (view logic), ilman että hänen tarvitsisi kirjoittaa paljon palvelinpuolen koodia. Kokoelmat myös päivittävät itsensä automaattisesti, joten kokoelmalla varustettu sivupohja näyttää aina uusinta dataa, ei koskaan vanhentunutta. (Meteor.com 2015f.)

Kokoelma luodaan kuvan 3 rivillä 1 olevan koodin mukaisesti. Palvelimella samainen koodi luo "harrastukset"-nimisen MongoDB-kokoelman. Selaimella se luo välimuistissa pidetyn yhteyden kokoelmaan. Rivillä 3 varmistetaan, että sen sisällä oleva koodi suoritetaan selaimella. Ehtolausekkeen sisään on kirjoitettu helper-funktio, joka hakee kaikki dokumentit harrastukset-kokoelmasta. (Meteor.com 2015f.)

Kuva 3. Kokoelman luominen.

Sen lisäksi että dataa voidaan hakea kokoelmista find-funktiolla, dataa voidaan myös

- lisätä insert-funktiolla
- poistaa remove-funtiolla
- päivittää update-funktiolla (Meteor.com 2015g).

Kokoelmien tietoturva

Ajatus siitä, että kuka tahansa voisi tehdä muutoksia sovelluksen tietokantaan ei kuulosta kovin hyvältä tietoturvallisuuden kannalta. Miksi sellainen toiminnallisuus on edes tehty mahdolliseksi? Vastaus löytyy insecure-nimisestä paketista, joka tulee aina uuden Meteor-projektin mukana (Meteor.com 2015h).

Insecure-paketti mahdollistaa nopean prototyyppien luomisen Meteorilla, koska sen avulla pystytään muokkaamaan tietokantaa selainpuolella. Kun prototyypit on luotu ja kun paketti halutaan poistaa, mennään komentorivillä Meteor-projektiin ja ajetaan "meteor remove insecure"-komento. Kun insecure-paketti poistetaan, poistuvat myös kaikki selainpuolen tietokantaoikeudet. (Meteor.com 2015h.)

Ilman oikeuksia tietokantaan mitkään tietokantakyselyt eivät luonnollisesti toimi. Asia korjataan kirjoittamalla Meteor.methods-funktioon metodi jokaisesta kyselystä, mitä halutaan suorittaa. Metodeja kutsutaan Meteor.call-funktiolla, jonka ensimmäiseksi parametriksi tulee kutsuttavan metodin nimi, sitä seuraten muut mahdolliset metodille annettavat parametrit. (Meteor.com 2015h.)

Metodien tulisi sijaita koodissa, joka on suoritettavissa niin palvelimella kuin myös selaimella, koska se mahdollistaa Meteorin ominaisuuden nimeltä optimistinen käyttöliittymä (Optimistic UI). Optimistisella käyttöliittymällä tarkoitetaan sitä, kun metodia kutsutaan Meteor.call-funktiolla. Tämän jälkeen tapahtuu kaksi asiaa samanaikaisesti:

1. Selain lähettää pyynnön palvelimelle turvallisen ympäristön läpi.

2. Simulaatio metodista ajetaan välittömästi selaimella, pyrkimyksenä ennustaa metodin palauttama vastaus palvelimelta käyttäen saatavissa olevia tietoja. (Meteor.com 2015h.)

Tämä siis tarkoittaa sitä, että kutsutun metodin vastaus näkyy sovelluksessa *ennen* kuin oikea vastaus on ehtinyt tulla palvelimelta. Jos palvelimelta tullut vastaus on sama kuin ennustettu, kaikki on kunnossa ja mikään ei muutu. Jos vastaus taas jostakin syystä eroaa ennustetusta, optimistinen käyttöliittymä -ominaisuus korjaa pieleen menneen ennustuksensa oikealla vastauksella. Optimistisella käyttöliittymällä saavutetaan palvelinpuolen koodin turvallisuus ja lähes olemattomalta tuntuva viive. (Meteor.com 2015h.)

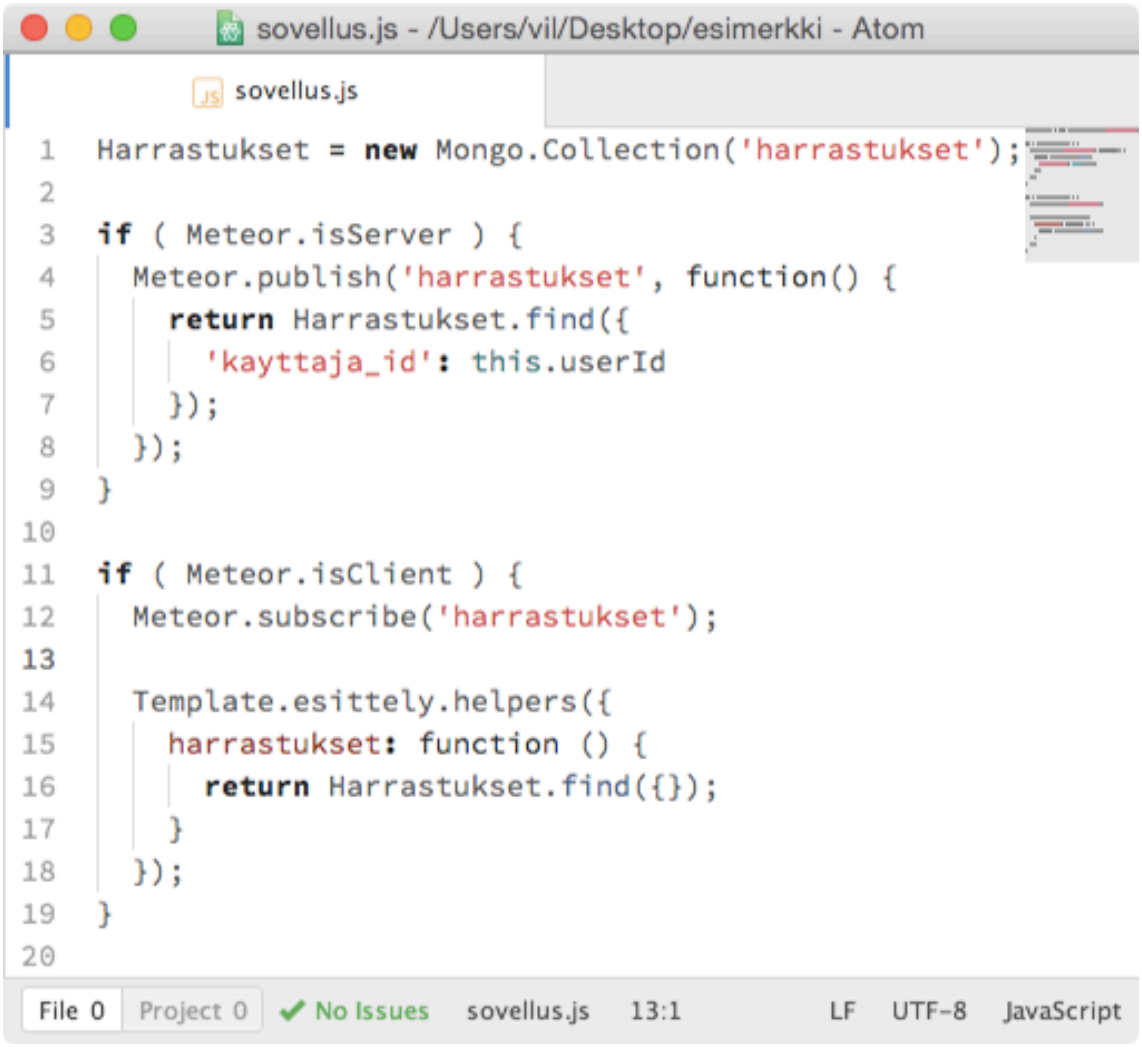
Datan julkaiseminen ja tilaaminen

Voisi kuvitella, että Meteor-sovelluksesta tulisi turvallisempi jos aikaisemmin mainittu insecure-paketti poistettaisiin. Vielä on kuitenkin yksi asia huomioitavana ja se on Meteorin kanssa oletuksena tuleva autopublish-paketti. Autopublish julkaisee automaattisesti kaiken mahdollisen datan tietokannasta kaikille käyttäjille, tarkoittaen siis, että jos esimerkiksi kuvan 3 mukaisesti haettaisiin käyttäjän kaikkia harrastuksia tietokannasta Harrastukset.find-kyselyfunktiolla, palauttaisi kysely tietokannassa olevien kaikkien käyttäjien harrastukset. Ei taaskaan kuulosta kovin hyvältä tietoturvan kannalta. (Meteor.com 2015i.)

Autopublish-paketti poistetaan samalla tavalla kuin insecure, eli mennään komentorivillä Meteor-projektiin ja ajetaan "meteor remove autopublish"-komento. Kun autopublish ei ole enää käytettävissä, sovelluskehittäjän on itse määriteltävä erikseen mitä dataa palvelimen tulisi lähettää selaimelle. Tämän tekemiseen Meteorista löytyy Meteor.publish- ja Meteor.subscribe-funktiot. (Meteor.com 2015i.)

Kuvassa 4 on kuvan 3 esimerkkisovellus johon on implementoitu Meteor.publish- ja Meteor.subscribe-funktiot niin, että Harrastukset-kokoelman find-funktio palauttaa vain sisäänkirjautuneelle käyttäjälle kuuluvat harrastukset. Rivillä 6 Meteor.find-funktion sisällä määritellään tietokannasta haettavan datan

ehdoksi sisäänkirjautuneen käyttäjän yksilöllinen tunnus, joka saadaan Meteorin tarjoamasta this.userId-muuttujasta. (Meteor.com 2015j.)



Kuva 4. Tietokannasta saatavien tulosten rajaus käyttäjätunnuksella.

4.4 Meteor-sovelluksen asentaminen toiselle palvelimelle

Meteor-sovelluksen asentaminen (deploying) Meteorin tarjoamalle ilmaiselle palvelimelle on tehty helpoksi "meteor deploy"-komennolla. Meteor-sovellus asennetaan menemällä komentorivillä Meteor-projektiin ja ajamalla "meteor deploy esimerkki.meteor.com"-komento. Meteorille on tunnistauduttava ennen kuin se voi aloittaa Meteor-sovelluksen asentamisen palvelimille. Tunnistautu-

minen tehdään Meteorin sivuilla luotavilla käyttäjätunnuksilla. Jos äsken mainittu komento "meteor deploy esimerkki.meteor.com" ajettaisiin, asentaisi Meteor sovelluksen osoitteeseen http://esimerkki.meteor.com, minkä jälkeen kyseinen sivu olisi kaikkien nähtävillä. (Meteor.com 2015k.)

Meteor-sovelluksen asentaminen omalle palvelimelle "meteor deploy"-komennolla on toki myös mahdollista, tosin vaivalloiseksi todettua. Meteorin kehittäjäyhteisö näyttää suosivan monipuolista "Meteor Up"-pakettia, jolla Meteor-sovelluksen asennus ja hallinta on tehty yksinkertaisemmaksi. (Meteor.com 2015l.)

Meteor Up (lyhennettynä mup) on komentorivityökalu, joka opinnäytetyön kirjoitushetkellä syksyllä 2015 tuki Debian-, Ubuntu- ja Open Solaris -käyttöjärjestelmiä. Mup käyttää näiden käyttöjärjestelmien tukemia forever- ja upstart-komentoja, joiden ansiosta Meteor-sovellus käynnistyy heti uudelleen, mikäli Meteor-sovellus kaatuu tai palvelin käynnistyy uudelleen. Mupilla myös hallitaan sillä asennettuja Meteor-sovelluksia komennoilla mup stop (pysäytys), mup start (käynnistys) ja mup restart (uudelleenkäynnistys). Sovelluksen lokitiedostoja voidaan tarkastella komennolla "mup logs". Mupilla Meteor-sovelluksen asennus hoituu lyhykäisyydessään näin:

1. Asennetaan Meteor Up komennolla "npm install -g mup".
2. Aloitetaan Meteor Up -projekti komennolla "mup init", joka luo tiedostot mup.json ja settings.json.
3. Mup.json-tiedostoon täytetään asennukseen tarvittavat tiedot, kuten oman palvelimen ip-osoite ja käyttäjätunnukset.
4. Valmistellaan oma palvelin valmiiksi asennukselle "mup setup"-komennolla.
5. Asennetaan Meteor-sovellus "mup deploy"-komennolla. (Arunoda 2015.)

5 MODERNI DATAVISUALISOINTI

Maailmassa olevan datan määrä kasvaa kasvamistaan päivä päivältä. Big datan nopean kasvun takia sen ymmärtäminen ja analysoiminen tulee jatkuvasti yhä vain vaikeammaksi. Kuinka tästä big data -jätistä saadaan ote otettua? Vastaus löytyy datavisualisoinnista.

Datavisualisointi on datan esittämistä visuaalisella tavalla, eli siis datan visualisointia. Suurten datamäärien visualisoimiseen on luotu visualisointisovelluksia, jotka mahdollistavat

- visuaalisesti esitettyjen analyyttisten tulosten tarkastelemisen nopeasti
- yhtäläisyyksien löytämisen suurten muuttujamäärien seasta
- konseptien ja olettamuksien välittämisen muille
- ehkä jopa tulevan ennustamisen. (SAS Institute Inc. 2014.)

Näiden asioiden lisäksi datavisualisointi auttaa myös ymmärtämään datasta asioita, jotka eivät ole välttämättä olleet itsestäänselvyyksiä aikaisemmin. Visualisoinnit välittävät tietoa ihmisille universaalilla tavalla ja tekevät ideoiden jakamisesta muiden kanssa helppoa. (SAS Institute Inc. 2014.)

Ihmisten kyky ymmärtää ja visualisoida suuria datamääriä on astumassa evoluutiossa samalle tasolle 1600-luvun astronomian kanssa. Ihan kuten vuosisatoja sitten, ihmisillä on nyt alkeelliset versiot työkaluista, joilla on hyvät mahdollisuudet tulla tehokkaiksi tulevaisuudessa. Datavisualisoinnin muuttuessa yhä enemmän ja enemmän digitaaliseksi, vastaan tulee yhä useammin visualisointeja, jotka eivät ole täysin teoreettisia. (Hidalgo & Almossawi 2014.)

Sen sijaan, että suunnittelija valitsisi jäykän kokoelman muotoja, asetelmia ja värejä, hän pystyy nyt valitsemaan sääntöjä, joilla saadaan data hengittämään geometrisiin abstraktioihin. Näiden uusien ominaisuuksien avulla pystytään rikastuttamaan visualisointien ja visualisointeja katsovien välejä, sillä katselija onkin nyt tavallisen katselijan sijasta pikemminkin tutkimusmatkailija. (Hidalgo & Almossawi 2014.)

Visualisointeja löytyy monenlaisia, vaikkakin kaikki niistä eivät ole hyödyllisiä, saati samanarvoisiksi luotuja. Yhden aineiston (dataset) esittämiseen löytyy monta eri tapaa. Parhaat datavisualisoinnit paljastavat jotakin uutta ja hyödyllistä datasta, jotakin, mitä ei ole koskaan ennen huomattu. Uusien löytöjen ymmärtäminen ja niiden huomioiminen on olennaista päätöksiä tehtäessä. (Steele 2012.)

Datavisualisointeja voidaan tehdä datan tutkimiseen tai sen selittämiseen. Tutkimiseen tarkoitetut visualisoinnit voivat ja saavatkin olla epätarkkoja. Epätarkkuus on hyödyllistä silloin, kun ei tiedetä, mitä datalla on kerrottavanaan ja kun siitä yritetään löytää mahdollisia yhteyksiä ja kuvioita (patterns) ensimmäistä kertaa. Oikean tavan löytäminen datan lähestymiseen tai siitä selvän saamiseen saattaa kestää jonkin aikaa. Visualisoinnit tulisi pystyä iteroimaan ja testaamaan nopeasti, jotta signaali löytyisi kohinasta mahdollisimman nopeasti. (Steele 2012.)

Datan selittämiseen tarkoitetut visualisoinnit ovat parhaita silloin, kun ne ovat selkeitä ja helposti ymmärrettäviä. Kohinan täydellinen poistaminen, eli tiedon rutistaminen yksinkertaisimpaan muotoonsa, kasvattaa visualisoinnin tehokkuutta kuinka vastaanottaja ymmärtää sen. Selittämiseen tarkoitettu visualisointi on parasta tehdä silloin kun datan sisältö ymmärretään ja tieto halutaan välittää toiselle henkilölle ymmärrettävässä muodossa. (Steele 2012.)

Datavisualisointiin on monia työkaluja, joista suuri osa näyttää olevan tehty selaimille. Syynä tähän saattaa olla se, että visualisoinnit ovat hyödyllisiä siellä missä niille on katselijoita, eli netissä. Github teki vuonna 2014 listan webille tarkoitetuista datavisualisointi-työkaluista, joista suurin osa oli JavaScript-kirjastoja. Kolme suosituinta eli eniten tähtiä saanutta kirjastoa listalla oli

1. d3.js
2. chart.js
3. leaflet. (Github.com 2014.)

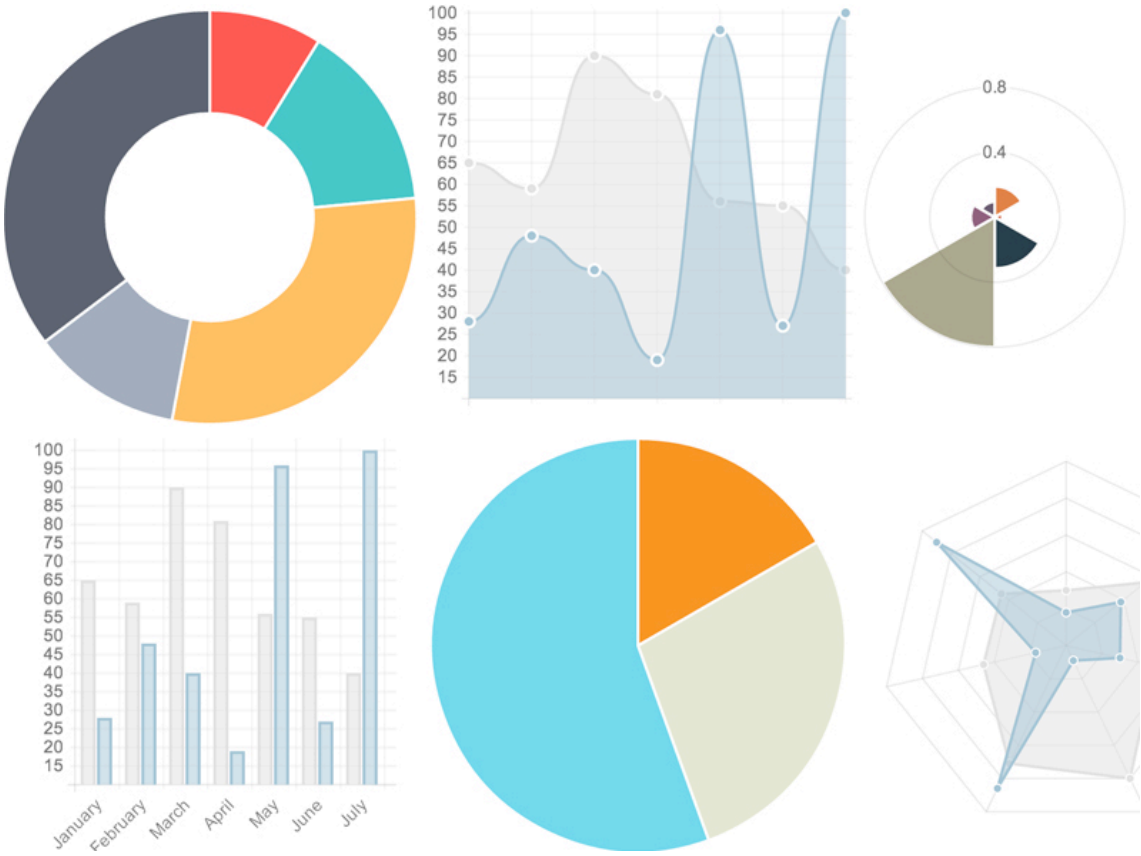
D3 (Data Driven Documents) herättää datan henkiin käyttäen HTML, SVG ja CSS apunaan. D3 ei tunne rajoituksia, minkä ansioista suunnittelijan on mah-

dollista tehdä minkälaisia visualisointeja tahansa. D3 on täynnä erilaisia ominaisuuksia, se on vuorovaikutteinen ja todella joustava, kaunis kirjasto. Haittapuolena siinä on jyrkkä oppimiskäyrä.

Chart.js on pieni mutta tehokas kirjasto, joka tukee kuutta eri kaaviotyyppiä:

1. käyrä (line)
2. palkki (bar)
3. tutka (radar)
4. polaari (polar area)
5. piirakka (pie)
6. donitsi (doughnut).

Chart.js on hyödyllinen kaikille tuttujen kuvassa 5 nähtävien kaavioiden esittämiseen, ja se onnistuu työssään erinomaisen hyvin. Kaavioiden ulkoasua on mahdollista muuttaa ja myös ihan omia kaavioita on mahdollista tehdä.



Kuva 5. Chart.js:n tukemat kaaviotyypit (Downie 2013).

Leaflet on JavaScript-kirjasto erilaisten interaktiivisien ja mobiiliystävällisten karttojen tekemiseen. Leaflet käyttää kartoissaan pelkästään HTML:ää ja CSS:ää.

Kuten tässä kappaleessa aiemmin mainittiin, ei ole yhtä ja ainutta oikeaa tapaa visualisoida dataa. Ihan samalla tavalla kun ei ole yhtä ja ainutta tapaa toteuttaa sovellustakaan JavaScript-kehyksellä. Paras kirjasto tai kehys niin visualisointiin kuin myös sovelluskehitykseen on sellainen, mikä sopii projektin vaatimuksiin, millä asiat saa yksinkertaisimmin hoidettua ja mistä itse pitää. Kannattaa siis kokeilla eri kirjastoja ja kehyksiä, todeta mitkä tuntuvat hyvältä ja sitten pysyä niissä hetken aikaa aloillaan.

6 SYÖTESOVELLUKSEN TOTEUTUS (SALATTU)

7 POHDINTA

Projektin tavoitteena oli suunnitella ja toteuttaa reaaliaikainen syötesovellus kesän 2015 aikana. Aikamääreissä pysyttiin ja sovellus julkaistiin saman vuoden syksynä.

Opinnäytetyön lopputuloksena syntyi selaimelle tehty reaaliaikainen ja helppokäyttöinen syötesovellus, joka toimii palvelujen toimituskanavana WorldSomen asiakkaille. Työ toteutettiin Meteor-sovelluskehyksellä, millä oli todella vaivatonta toteuttaa reaaliaikaisia toiminnallisuuksia. Työssä hyödynnettiin paljon Meteorille tehtyjä lisäosia, joilla ratkaistiin toteutusvaiheessa ilmestyneitä ongelmia. Ainoa asia mitä syötesovelluksessa ei saatu toimimaan reaaliajassa oli vuorovaikutteiset kaaviot, mutta toisaalta reaaliaikaisuutta kaavioissa ei toimeksiantaja vaatinutkaan. Kaavioiden reaaliaikaistaminen voisi kuitenkin olla jatkokehitysidea.

Meteorin parissa työskentely oli helppoa ja nopeaa. Henkilöillä kenellä on vähänkään mielenkiintoa JavaScript-sovelluskehyksiin tai moderneihin verkko- ja älypuhelinsovelluskehitys-tekniikoihin kannattaa ehdottomasti kokeilla Meteoria. Meteor sopii niin pieniin kuin myös suuriin projekteihin.

LÄHTEET

Alethes 2015. Meteor Pages. Viitattu 19.11.2015 https://github.com/alethes/meteor-pages.

Arunoda 2015. Meteor Up. Viitattu 3.10.2015 https://github.com/arunoda/meteor-up.

Coleman, T. & Greif, S. 2013. DISCOVER METEOR. Viitattu 7.6.2015 https://www.discovermeteor.com.

Dascalescu, D. 2015. Why Meteor. Viitattu 29.9.2015 http://wiki.dandascalescu.com/essays/why_meteor.

Downie, N. 2013. Chart.js. Viitattu 3.12.2015 https://dribbble.com/shots/989123-Chart-js.

Fowler, M. 2005. InversionOfControl. Viitattu 25.9.2015 http://martinfowler.com/bliki/InversionOfControl.html.

Github.com 2014. Data visualization. Viitattu 5.10.2015 https://github.com/showcases/data-visualization.

Google Chrome Developers 2015. MVC Architecture. Viitattu 25.9.2015 https://developer.chrome.com/apps/app_frameworks.

Hidalgo, C.A. & Almossawi, A. 2014. The Data-Visualization Revolution. Viitattu 4.10.2015 http://www.scientificamerican.com/article/the-data-visualization-revolution.

Meteor Development Group 2015. Spacebars. Viitattu 1.10.2015 https://atmospherejs.com/meteor/spacebars.

Meteor.com 2015a. What is Meteor? Viitattu 30.9.2015 http://docs.meteor.com/#/full/whatismeteor.

Meteor.com 2015b. Principles of Meteor. Viitattu 30.9.2015 http://docs.meteor.com/#/full/sevenprinciples.

Meteor.com 2015c. Structuring your application. Viitattu 30.9.2015 http://docs.meteor.com/#/full/structuringyourapp.

Meteor.com 2015d. Live HTML templates.http://docs.meteor.com/#/full/livehtmltemplates.

Meteor.com 2015e. Session. Viitattu 2.10.2015 http://docs.meteor.com/#/full/session.

Meteor.com 2015f. Storing tasks in a collection. Viitattu 2.10.2015 https://www.meteor.com/tutorials/blaze/collections.

Meteor.com 2015g. Collections. Viitattu 2.10.2015 http://docs.meteor.com/#/full/mongo_collection.

Meteor.com 2015h. Security with methods. Viitattu 2.10.2015 https://www.meteor.com/tutorials/blaze/security-with-methods.

Meteor.com 2015i. Filtering data with publish and subscribe. Viitattu 2.10.2015 https://www.meteor.com/tutorials/blaze/publish-and-subscribe.

Meteor.com 2015j. Publish and subscribe. Viitattu 2.10.2015 http://docs.meteor.com/#/full/publish_userId.

Meteor.com 2015k. Deploying your app. Viitattu
3.10.2015 https://www.meteor.com/tutorials/blaze/deploying-your-app.

Meteor.com 2015l. Why is deploying meteor apps so "hard"? Viitattu
3.10.2015 https://forums.meteor.com/t/why-is-deploying-meteor-apps-so-hard/1080.

Mysore, A. 2015. 10 Best JavaScript Frameworks and Libraries of 2015., 2015 Viitattu
22.9.2015 http://beebom.com/2015/04/best-javascript-frameworks-and-libraries.

Osmani, A. 2012. Journey Through The JavaScript MVC Jungle. Viitattu
7.6.2015 http://www.smashingmagazine.com/2012/07/journey-through-the-javascript-mvc-jungle.

Rauschmayer, A. 2015. Speaking JavaScript. Viitattu
7.6.2015 http://speakingjs.com/es5/ch04.html#ftn.id212981.

SAS Institute Inc. 2014. Data Visualization: What it is and why it's important. Viitattu
4.10.2015 http://www.sas.com/en_us/insights/big-data/data-visualization.html.

Steele, J. 2012. Why data visualization matters. Viitattu
4.10.2015 http://radar.oreilly.com/2012/02/why-data-visualization-matters.html.

Strack, I. 2012. Getting Started with Meteor.js JavaScript Framework. Birmingham: Packt Publishing Ltd. Saatavissa
myös http://site.ebrary.com/lib/turkuamk/docDetail.action?docID=10642559&ppg=1.

W3Techs 2015. Usage of client-side programming languages for websites. Viitattu
22.9.2015 http://w3techs.com/technologies/overview/client_side_language/all