Wolf Wikgren

Lukiovertailusovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

15.1.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Wolf Wikgren Lukiovertailusovellus 61 sivua + 2 liitettä 15.1.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Uutispäätoimittaja Minna Holopainen Lehtori Olli Alm

Insinöörityön tavoitteena oli kehittää datajournalistinen lukiovertailusovellus tilaajayritykselle. Pyrkimyksenä oli verkkosovelluksen avulla visualisoida tilaajayrityksen lukiovertailua varten keräämää data-aineistoa ja luoda sovelluksesta datajournalistinen tuote tilaajayrityksen asiakkaille muun aiheeseen liittyvän journalistisen sisällön oheen.

Toteutuksen pohjustamiseksi tutkittiin moderneja tapoja toteuttaa verkkosovelluksia pyrkien tarkastelemaan niitä uutistoimitusympäristön tarpeiden ja vaatimusten näkökulmasta. Tämän tutkimuksen perusteella valittiin teknistä toteutusta varten yhdistelmä moderneja verkkoteknologioita, jotka mahdollistavat uutistoimistoympäristöön sopivan nopean verkkosovelluskehityksen. Lisäksi tutkittiin ja tarkasteltiin datajournalismin ja datavisualisoinnin käsitteitä ja niiden merkitystä sovelluskehityksen kannalta journalistisessa ympäristössä. Tämä tutkimus antoi kuvan informaatiomuotoilun hyviksi todetuista periaatteista, joita hyödynnettiin sovelluksen suunnittelussa ja arvioinnissa.

Sovellus toteutettiin tutkimuksen pohjalta hyödyntäen MongoDB-NOSQL-tietokantaa, Node.js-palvelinympäristöä ja Javascript-pohjaisia datavisualisointikirjastoja. Palvelinsovelluksen ja asiakassovelluksen välinen tiedonsiirto toteutettiin REST-rajapinnan avulla niin, että asiakas- ja palvelinsovellus pidettiin erillisinä toisistaan SPA-arkkitehtuurimallin mukaisesti. Sovellus visualisoi lukiovertailusta saatavan aineiston karttapohjaisessa näkymässä ja mahdollistaa aineiston vuosikohtaisen vertailun jokaisen lukion kohdalla. Loppuasiakas pystyy myös paikallistamaan sovelluksen maakuntatasolla.

Sovellus julkaistiin kevään 2014 ylioppilaskirjoitustulosten julkaisun yhteydessä 31.5.2014. Tilaajayrityksen loppuasiakkaista yksi julkaisi sovelluksen sivullaan. Sovellus on tilaajayrityksen jatkokehityksessä ja sitä käytetään jatkossa ylioppilaskoetulosten julkaisuun: sisällöltään päivitetty versio julkaistiin 26.12.2014 syksyn ylioppilaskirjoitusten yhteydessä ja se keräsi noin 7 500 käyttäjää päivässä.

Avainsanat	datajournalismi, node.js, datavisualisointi, javascript

Författare Titel Sidnummer Datum	Wolf Wikgren Applikation för gymnasiejämförelse 61 sidor + 2 bilagor 15.1.2015
Examen	Ingenjör (YH)
Utbildningsprogram	Medieteknik
Specialiseringsområde	Digitala medier
Handledare	Redaktionschef Minna Holopainen Lektor Olli Alm

Ändamålet för detta examensarbete var att utveckla en datajournalistisk webbapplikation för uppdragsgivande företaget STT-Lehtikuva (FNB). Applikationens syfte var att visualisera data ifrån en nationell gymnasiejämförelse som gjorts av uppdragsgivaren och fungera som en digital produkt för beställarens kunder utöver det traditionella journalistiska utbjudet.

För att bygga applikationen, forskades moderna metoder för utveckling av webapplikationer, med särskild fokus på funktionalitet som krävs i en nyhetsredaktionsomgivning. På basis av denna forskning valden en kombination webbteknologier för förverkligandet av applikationen, som möjliggör snabb webbapplikationsutveckling. Förutom det, forskades begreppen datajournalistik och datavisualisering med avsikt att undersöka deras innebörd för mjukvaruutveckling. Detta gav som resultat en bild om vad som anses forma bra informationsdesign för journalistik, vilket togs i beaktan under formgivningen av applikationens visuella användargränssnitt.

För förverkligandet av applikationen användes på basis av den gjorda forskningen Node.js-programsystemet, en MongoDB-NOSQL-databas och Javascriptbaserade datavisualiseringsbibliotek. Överföringen av data mellan klient- och serverapplikationen utfördes med i formen av ett REST-gränssnitt, för att hålla de två applikationsdelarna separata enligt SPA-arkitekturmodellen. Applikationen visualiserar gymnasierna i Finland på en karta och möjliggör noggrannare begränskning av datat för enskilda gymnasium.

Applikationen publicerades i samband med publiceringen av vårens studentexamensresultat 31.5.2014. Applikationen utvecklas vidare och används också i framtiden för publicerandet av gymnasiejämförelseresultat: En innehållsvis uppdaterad version publicerades 26.11.2014 i samband med höstens studentexamensresultat, och samlade runt 7 500 unika tittare.

nyckelord	datajournalistik, node.js, datavisualisering, javascript

Author Title Number of Pages Date	Wolf Wikgren High school comparison application 61 pages + 2 appendices 15 January 2015
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Editor-in-Chief Minna Holopainen Lecturer Olli Alm

The purpose of this thesis project was to develop a data journalistic web application based on data acquired from a national high school comparison made by the ordering company, STT-Lehtikuva. The aim was to create digital product to accompany the traditional journalistic content regarding the high school comparison.

In order to build the application, research was made into modern methods of web application development, with a focus on the aspect of functionality in a newsroom environment. This research was used to choose a combination of web application development tools and technologies that are suitable for rapid development and publishing. Furthermore, the concepts of data journalism and data visualisation were explored, inspecting their meaning for software development in a journalistic environment. This research was used to form an informed view about best practices of visualizing information in a journalistic context, which was utilized in the visual design of the application.

The application was built using a MongoDB- NOSQL -database, the Node.js server runtime environment and Javascript-based data visualisation libraries. Communication between the client and the server application was conducted via a REST-API, keeping the client and the server separate as per the SPA-architecture model. The application visualises all Finnish high schools on a map, enabling the user to select specific high schools and view more detailed information about each school.

The application was published after the matriculation examination results in the spring of 2014 were released, 31.5.2014. The application is under further development, and will be used for publishing high school comparison results in the future as well. A content-wise updated version was published along with the results of the autumn matriculation examination results on the 26th of November, 2014, and it gathered around 7 500 unique visitors in a day.

Keywords	data journalism, node.js, data visualization, javascript

Sisällys

Lyhenteet & käsitteet

Ajax	Asynchronous Javascript and XML. Kokoelma tekniikoita ja teknologioita, jotka mahdollistavat asynkronisen tiedonsiirron verkkosovelluksissa.
BSON	Binary JSON. Binäärimuotoinen JSON.
CRUD	Create, Read, Update, Delete. Viittaa REST-rajapinnan perustoimintoihin.
CSS	Cascading Style Sheets. Verkkosivuilla käytetty tyylimäärittelykieli. Määrittelee verkkosivun ulkoasun.
DOM	Document Object Model. Konventioksi muodostunut malli rakenteisten dokumenttien muokkaamiseen.
HTML	Hypertext Markup Language. Verkkokehityksen runkona käytettävä merkintäkieli. Määrittelee verkkosivun rakenteen.
HTTP	HyperText Transfer Protocol. Selainten ja verkkopalvelimien väliseen kommunikaatioon käytettävä tiedonsiirtoprotokolla.
JS	Javascript. Pääasiassa web-ympäristössä käytettävä ohjelmointikieli. Lisää verkkosivustolle toiminnallisuuksia.
JSON	Javascript Object Notation. Javascript-objektisyntaksin mukainen tiedostoformaatti.
MVC	Model-View-Controller. Sovelluskehityksessä käytetty arkkitehtuurimalli.
MV*	Model-View-*. Sovelluskehityksessä käytetty MVC-arkkitehtuurin mukainen arkkitehtuurimalli, jossa Controller-osa on korvattu muulla toiminnallisuudella.
NoSQL	Yhteisnimitys relaatiotietokantamallista eroaville tietokantaratkaisuille.
PHP	PHP: Hypertext Preprocessor. Lähinnä verkkosovellusten palvelinympäristöissä käytettävä ohjelmointikieli.

REST	Representational State Transfer. Rajapinta-arkkitehtuurimalli.
RESTFUL	REST-rajapintatoteutus, joka ei täytä kaikkia REST-määritelmän mukaisia ominaisuuksia.
SVG	Scalable Vector Graphics. Tiedostoformaatti vektorigrafiikan esittämiseen.
SQL	Structured Query Language. Relaatiotietokantojen käyttämä kyselykieli.

1 Johdanto

Insinöörityön tarkoituksena on kehittää verkkoon julkaistava datajournalistinen lukiovertailusovellus uutistoimisto STT-Lehtikuvalle. STT-Lehtikuva on Suomen johtava uutis- ja kuvatoimisto, ja tämä projekti on osa laajempaa kokonaisuutta, jossa se pyrkii rakentamaan digitaalista sisältöä asiakkailleen. Insinöörityön tarkoituksena on suunnitella ja toteuttaa STT-Lehtikuvalle pilottisovellus datajournalismin julkaisutarpeisiin ja tutkia mahdollisuuksia hyödyntää uusimpia verkkoteknologioita. Sovellus toimii interaktiivisena käyttäjärajapintana esitettävään lukiodata-aineistoon visualisoimalla dataa loppukäyttäjän kannalta helpommin esitettävään ja ymmärrettävään muotoon. Valmistuttuaan sovelluksen on tarkoitus toimia pohjana jatkokehittelyä varten, joko saman projektin suorana jatkona tai uusissa datajournalistisissa hankkeissa. Sovelluksen kehitystä tarkastellaan etenkin sen vaatimien teknisten ratkaisuiden näkökulmasta tutkimalla projektin kulun aikana mahdollisuuksia toteuttaa toiminnallisuuksia mahdollisimman tehokkaasti, niin toiminnallisesta kuin kustannusnäkökulmasta.

STT-Lehtikuvan julkaisuympäristö on lähtökohdiltaan haastava verkkojulkaisua ajatellen, sillä uutistoimistona sillä ei ole omaa julkaisua, vaan se myy uutisvirtaansa palveluna asiakkailleen. Materiaali siis toimitetaan loppuasiakkaille julkaisualustan kautta, minkä jälkeen loppuasiakkaat halutessaan voivat julkaista sen omissa julkaisuissaan. Tämä asettaa projektille huomattavia teknisiä haasteita ja vaatimuksia, sillä lopullinen verkkojulkaisualusta ja sen vaatimukset saattavat vaihdella asiakkaasta toiseen ja siihen asiakkaan päässä kohdistuvien sivulatausten määrää, ja sen myötä myös sivulatausten käsittelyyn vaadittavien resurssien määrää, on vaikea ennakoida. Sovelluksen kehityksessä tulee siis erityisesti huomioida skaalautuvuus yllättävien verkkoliikennepiikkien varalta ja runsas vanhojen selainten tuki, kuitenkaan unohtamatta visuaalisuutta ja esitettävän aineiston viestinnällistä merkitystä.

Sovelluksessa esitetään STT:n lukiovertailua varten kerättyä ja koostettua aineistoa, joka koostuu pääosaltaan opetushallituksen tilastoista lukioiden sisäänpääsy- ja ylioppilaskoekeskiarvoista vuosilta 2012, 2013 ja 2014. Sisäänpääsy- ja ylioppilaskoekeskiarvoista on laskettu vertailulukuja, joiden avulla voidaan arvioida lukion opiskelijoiden keskiarvon kehittymistä lukio-opiskelun alkuvaiheesta sen loppuun. Tämän avulla pyritään antamaan kuva lukion vaikutuksesta oppilaiden keskiarvoihin. Tämä poikkeaa niin sanotusta perinteisestä tavasta vertailla lukioiden tasoa, jossa tarkastellaan vain yliop-

pilaskokeen pakollisten aineiden puoltoäänten kokonaissumman keskiarvoa. Esitettävä data sisältää myös tiedon lukioiden sijainnista ja erilaisista taustamuuttujista, kuten lukion koosta ja hylättyjen ylioppilaskoekirjoittajien määrästä. Sovelluksen tarkoituksena on esittää koostettu data selkeällä tavalla ja mahdollistaa yksittäisten lukioiden ja niiden arvojen kehityksen tarkastelun.

Insinöörityö ei käsitä käytettävän data-aineiston keräämistä, käsittelyä ja analysointia. Journalistisen näkökulman kannalta käydään läpi datajournalismin peruskäsitteitä, koska datajournalismin yleistyminen on merkittävä syy verkkosovellusten nousuun osaksi tavallista uutistoimitustyötä. Työssä pyritään kuitenkin keskittymään sovelluksen vaatimiin teknisiin, toiminnallisiin ja visuaalisiin ominaisuuksiin niin palvelinpuolella kuin käyttäjärajapinnassa, perustelemalla ratkaisuja etenkin teknisestä ja informaatiomuotoilullisesta näkökulmasta.

2 Datajournalismi

2.1 Verkkosovellukset osana uutisvirtaa

Perinteisen uutismedian voidaan nähdä elävän parhaillaan voimakasta murroskautta. Niin kotimainen kuin kansainvälinen uutistarjonta on viime vuosien aikana digitalisoitunut ja mobilisoitunut voimakkaasti lukijoiden siirtyessä perinteisistä painetuista julkaisuista käyttämään yhä suuremmassa määrin verkko- ja mobiilipäätelaitteita mediakulutuksessaan. Suomen suurimmat sanomalehdet tavoittavat jo enemmän yleisöä verkkopalveluiden kuin paperilehtien kautta [KTM Lehtien kokonaistavoittavuus 2013 2014] ja etenkin matkapuhelin- ja tablettilukijoiden määrä on kasvanut voimakkaasti vain muutamassa vuodessa; matkapuhelimella uutisia viikoittain lukevien kuluttajien määrä on kahdessa ja puolessa vuodessa yli kaksinkertaistunut ja tablettilukijoiden määrä lähes kymmenkertaistunut painetun lehden lukijoiden osuuden hitaasti mutta tasaisesti pienentyessä [Kansallinen Mediatutkimus KMT 2013 2014]. Yhdysvaltalaisen, puolueettoman Pew Research Center -ajatushautomon State of The Media -raportin osana koottujen tilastojen perusteella digitaaliset kanavat ovat yhteenlaskettuna jo lähes saavuttaneet television tärkeimpänä uutislähteenä Yhdysvalloissa [Where Americans Get News 2012].

Tämän murroksen myötä mediayhtiöt ovat joutuneet panostamaan entistä enemmän digitaalisen sisällön luomiseen. Mediatalot ovat kehittäneet uusia digitaalisia jakelukanavia; Suomessa suurilla sanomalehdillä on tyypillisesti jopa useita eri mobiilisovelluksia eri sisällöille ja eri kohdeyleisöille, minkä myötä verkkokehitys ja ohjelmistotuotanto on kasvanut selkeäksi ja tärkeäksi osaksi etenkin suurien mediakonsernien toimintaa [Sanoman vuositulos 2013 2014: 3]. Nämä uudet digitaaliset kanavat ovat tuoneet mukanaan mahdollisuuden luoda aivan uudenlaista uutissisältöä. Verkkoon ja mobiililaitteille luodun sisällön ei tarvitse koostua perinteisestä yksisuuntaisesta koko lukijakunnalle suunnatusta tekstin ja kuvituksen yhdistelmästä, vaan lukijan on laitteensa avulla mahdollista päästä vuorovaikutukseen sisällön kanssa, sisällön niin salliessa.

Vuorovaikutteisen sisällön luominen vaatii kuitenkin huomattavan erilaista osaamista perinteiseen toimittajantyöhön verrattuna. Näiden uusien digitaalisten kanavien moninaisuuden ja päätelaitteiden vaihtelevuuden myötä yksinkertaiset verkkosovellukset ovat muodostuneet tyypilliseksi tavaksi lisätä interaktiivisuutta uutisvirtaan. Verkossa yleisesti käytössä olevat teknologiat ja standardit, kuten Javascript (JS), HTML ja CSS, ovat kehittyneet erinomaisiksi skaalautumaan erikokoisille näytöille ja toimimaan vaihtelevantehoisilla päätelaitteilla, samanaikaisesti mahdollistaen täysimittaisten sovellusten kehittämisen. Tämän lisäksi erittäin tärkeää kehitykselle on myös päätelaitevalmistajien melko yhdenmukainen tuki yleisimmille näistä teknologioista ja tekniikoista. Toisin sanoen, verkkosovellukset toimivat oikein toteutettuna suurimmalla todennäköisyydellä suurimmassa määrässä laitteita. [Standards for Web Applications… 2014.]

Verkkoteknologioiden avulla voidaan siis luoda niin sanottua rikasta sisältöä aina yksinkertaisesta grafiikasta tai animaatioista täysimittaisiin sovelluksiin julkaisualustojen sisällä tehostamaan viestinnällistä sisältöä tavoilla, jotka ovat painetussa mediassa mahdottomia. Tällainen interaktiiviseksi journalismiksi kutsuttava lisätoiminnallisuus pyrkii mahdollistamaan lukijalle jonkinasteisen vapauden perinteisestä lineaarisesta tavasta käsitellä sisältöä [Nip 2006].

2.2 Datajournalismi

Datajournalismi on noussut kuvaamaan suurta osaa tuotetusta uudenlaisesta uutissisällöstä. Interaktiivisuuden lisäksi datajournalismin käsitteelle on olennaista sen informaatiosisältö eli data. Yksinkertaisimman määritelmän mukaan datajournalismi on

journalismia, joka on tuotettu käyttäen hyväksi laajempia tietoaineistoja kuin tavanomaisessa journalistisessa prosessissa. Toisin sanoen, kun journalistisen jutun taustaaineistona käytetään asiantuntijoiden ja valmiiksi analysoitujen julkaisujen sijaan tilastoaineistoa, verkossa jaettavaa avointa dataa tai muuta raakadataa, voidaan puhua datajournalismista. Englannin kielen termi data-driven journalism, eli datalähtöinen journalismi, kuvaakin ehkä prosessia paremmin, kun viitataan datajournalismin rooliin ennen kaikkea jutun taustamateriaalina ja sen pohjan muodostustapana. Tätä numeerista, usein tilastopohjaista aineistoa käsittelemällä voidaan asioita tarkastella tilastollisesta näkökulmasta ja analysoimalla sitä vetää johtopäätöksiä, joiden pohjalta mahdollisesti rakentaa journalistista sisältöä. Datajournalismi nähdäänkin useimmiten osana tutkivan journalismin perinnettä. Sen ominaispiirteenä voidaan pitää datasta niin sanotun tarinan, eli varsinaisen viestinnällisen sisällön, löytämistä ja esittämistä [Gray ym. 2013: 93–94].

Datajournalismi liitetään terminä kuitenkin myös huomattavasti laajempaan kokonaisuuteen käytäntöjä, tekniikoita ja strategioita kuin yksinkertaisesti datalähtöisyyteen. Termiä käytetään myös kattamaan tietokantajournalismin (engl. database journalism tai structured journalism) ja tietokoneavusteisen tiedottamisen (engl. computer-assisted reporting) menetelmiä ja prosesseja. Tietokantajournalismilla tarkoitetaan toimitusten tapaa järjestelmällisesti tallentaa tietoa [Chua 2010] ja tietokoneavusteisella tiedottamisella kaikkea tietojenkäsittelytieteen ja tietokoneiden käyttöä uutistoimituksissa 1950-luvulta lähtien [Cox 2000]. Datajournalismi yhdistetään myös vahvasti informaatiomuotoilun ja infografiikan käsitteisiin [Gray ym. 2013: 33–34], koska prosessin lopputuote on usein datavisualisointi.

Datajournalismi voidaan käsittää kattotermiksi, joka itsessään viittaa monipuoliseen kokonaisuuteen ominaisuuksia, tekniikoita ja menetelmiä journalistisen prosessin eri vaiheissa. Termillä datajournalismi halutaan ennen kaikkea viitata koko dataa sisältävään journalistiseen prosessiin eikä vain tiettyyn osaan kokonaisuutta [Poikola 2011]. Teknisestä näkökulmasta datajournalismi viittaa tapoihin ja teknologioihin, joilla puhdistaa, käsitellä, analysoida, yhdistellä ja esittää data-aineistoja. Käytännössä tämä viittaa työkaluihin, esimerkiksi taulukkolaskentaohjelmistoihin, ohjelmointikirjastoihin, tietokanta- ja paikkatietojärjestelmiin sekä visualisointityökaluihin. Konkreettisia esimerkkejä ammattilaisten parissa suosituista ohjelmista ja teknologioista ovat taulukkolaskentaohjelmat Google Spreadsheet ja Microsoft Excel, paikkatietojärjestelmät ArcGIS ja QGIS sekä ohjelmointikielet Javascript ja Python [Gray ym. 2012: 405–414]. Työkalut ovat

lähes poikkeuksetta suhteellisen uusia, ja samoja toimintoja täyttäviä ohjelmistoja on tarjolla useampia. Tämä tarjonnan runsaus yhdistettynä koko datajournalismin käsitteen uutuuteen aiheuttaa suurta vaihtelua käytettävissä työkaluissa. Koska ala on vielä voimakkaasti kehitysvaiheessa, ei voimakkaasti vakiintuneita yleisiä käytäntöjä käytettävien teknologioiden ja ohjelmistojen suhteen vielä ole muodostunut, vaan voidaan nähdä, että eri toimitukset hyödyntävät aivan erilaisia teknisiä kokonaisuuksia.

Suomessa datajournalismi on vakiintumassa osaksi toimitusten työtapaa. Esimerkiksi Helsingin Sanomat vakinaisti vuonna 2013 datadeskin, joka keskittyy nimenomaan datajournalistisen sisällön tuottamiseen. Helsingin Sanomilla onkin sivuillaan säännöllisesti datajournalistisia uutisia, jotka perustuvat usein kyselyihin, testeihin, aikajanoihin tai karttoihin. Näiden tapaiset yksinkertaiset visualisoinnit ovat useimmin näkyvät datajournalistisen lopputuloksen muodot: pieniä verkkosovelluksia, jotka esittävät havainnollisesti joko niihin syötettyä informaatiota (kyselyt, testit) tai jostain taustajärjestelmästä koottua aineistoa (kartat, aikajanat, ym.).

2.3 Informaatiomuotoilu ja datan visualisointi

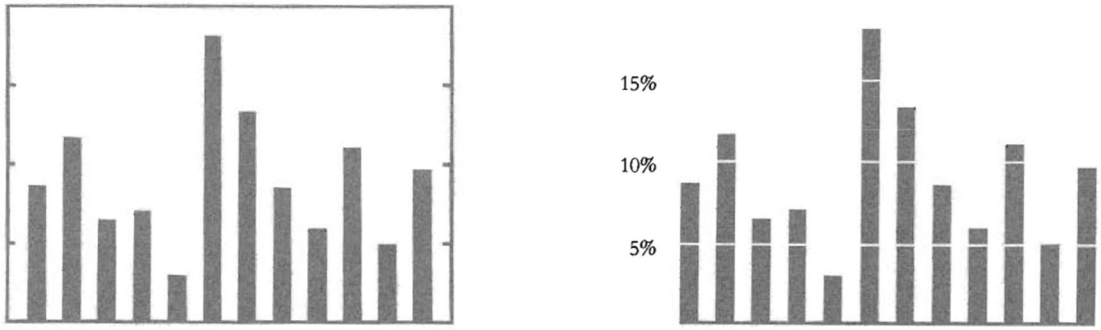
Olennainen osa datajournalistisen sovelluksen toteutusta on kyseessä olevan datan esittäminen havainnollistavalla tavalla ja sen viestinnällisen merkityksen välittäminen lukijalle. Niin kutsutulla datavisualisoinnilla pyritään luomaan grafiikkaa suoraan dataaineistosta ja toteuttamaan helpommin käsiteltävä ja yksinkertaisempi käyttöliittymä esitettävälle aineistolle.

Tilastotieteilijä ja informaatiomuotoilun pioneeri Edward R. Tufte pyrkii kirjassaan The Visual Display of Quantitative Information määrittelemään, mistä ainesosista hyvä datan visualisointi rakentuu. Tuften mukaan hyvän graafisen esityksen tulee ennen kaikkea esittää dataa vääristelemättä ja koherentisti samalla johdatellen lukija ajattelemaan itse dataa sen keräämiseen käytetyn metodologian sijaan [Tufte 2001: 13]. Visualisointien laadun arvioinnissa Tufte esittää neljä termiä erityisesti huomioon otettaville yksityiskohdille: *valekerroin* (lie factor), *data-mustesuhde* (data-ink-ratio), *kuvioroina* (chart junk) ja *informaatiotiheys* (data density).

Valekertoimella Tufte mittaa visualisoinnin kykyä edustaa todellisuutta. Yksinkertaisimmillaan valekerroin toimii suhteena kuviossa esitettävän arvon ja todellisuuden välil-

lä, eli jos grafiikassa esitetään kahden esineen kokojen suhdetta ja esine A on todellisuudessa kaksi kertaa esinettä B suurempi, tulee myös esinettä A esittävän kuvion olla kaksi kertaa esineen B kuviota suurempi. Konsepti on yksinkertainen, mutta visualisoinneissa usein joko tahallisesti tai tahattomasti rikotaan esitettävien arvojen todellisia suhteita toisiinsa. Esimerkiksi yhdistettäessä perspektiivi kuvioon saatetaan helposti tahattomasti johtaa lukijaa harhaan esittämällä kuvion suhteet virheellisesti. Mahdollisimman totuudenmukainen visualisointi varmistaa, että mahdollisimman moni lukija tulkitsee datan yhdenmukaisesti. [Tufte 2001: 56–57.]

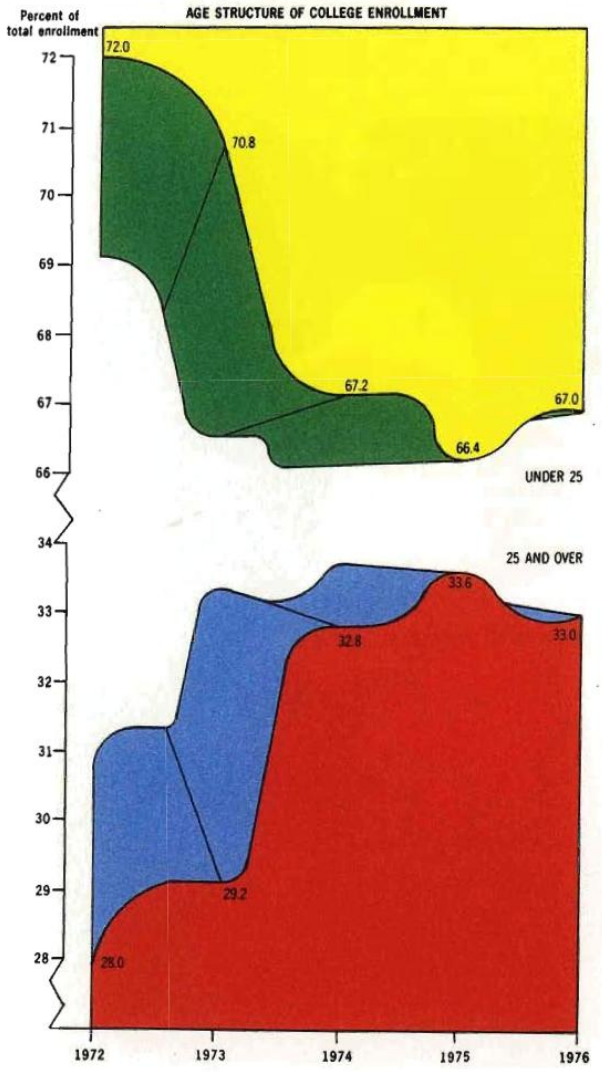
Data-mustesuhteella Tufte viittaa itse datan visualisointiin käytetyn musteen, datamusteen (data-ink), määrään suhteessa muuhun visualisointiin käytettyyn musteeseen; visualisointi on sitä parempi, mitä tehokkaampi tämä suhde on. Visualisoinnin tulisi siis esittää mahdollisimman paljon dataa mahdollisimman vähällä ylimääräisellä musteella. Optimaalinen tila on löydetty silloin, kun mitään ei enää voi poistaa vääristämättä dataa, toisin sanoen kun jäljellä on vain datamustetta [Tufte 2001: 91–107]. Vaikka muste ei materiaalina enää ole käytössä digitaalisessa ympäristössä, voidaan tätä periaatetta ajatella digitaalisella näytöllä vaikkapa datapikseleinä, dataa esittävien pikselien suhteena dataa esittämättömiin pikseleihin. Tuften kirjassaan esittämä esimerkki pylväsgraafin datamusteen optimoinnissa esitetään kuvassa 1.



Kuva 1. Tuften ajatus pylväsgrafiikan datamustemäärän optimoinnista. Vasemmalla tyypillinen pylväsgrafiikka, oikealla Tuften ehdotus paremmasta mallista. [Tufte 2001: 127.]

Kuvioroinalla viitataan kaikkeen datavisualisoinnissa ylimääräiseen, dataa esittämättömään oheisgrafiikkaan. Kuvituselementit grafiikan osana saattavat helposti vääristää dataa tai tehdä visualisoinnista vaikeaselkoisemman, ja Tuften mukaan niitä olisi parasta välttää kokonaan. Tyypillisiä esimerkkejä kuvioroinasta ovat tahattomat optiset illuusiot (yleisiä etenkin mustavalkotulosteissa), turha tai liian voimakas ruudukon tai asteikon käyttäminen ja erilaiset koriste-elementit tai turha koristeellisuus itse kuviossa

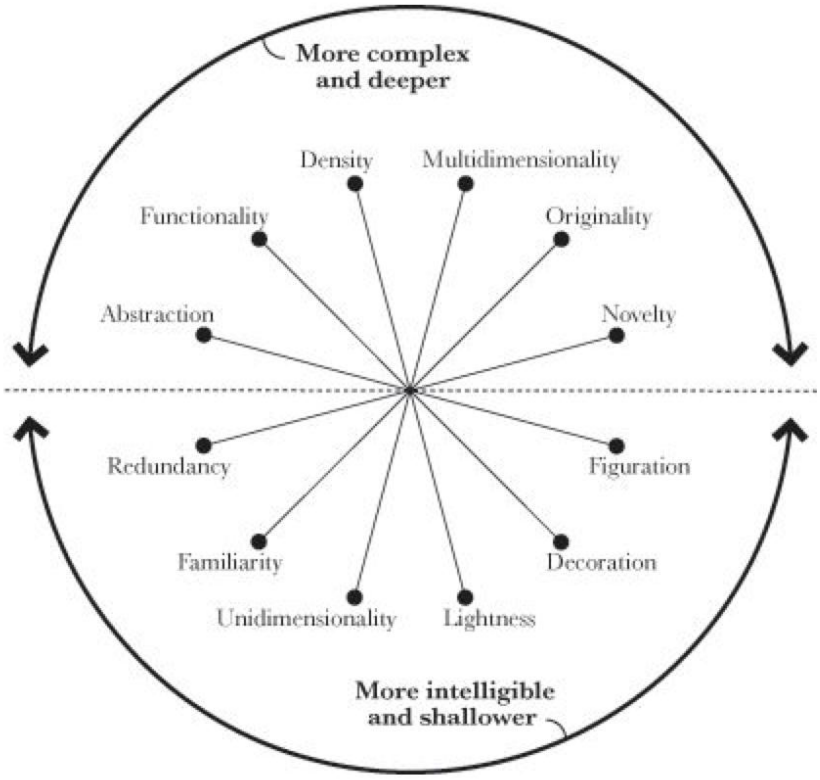
(kuva 2). Tällainen pyrkimys visuaaliseen näyttävyyteen on tehnyt esimerkiksi kuvassa 1 esitettävässä American Education -lehdessä 1970-luvulla julkaistusta infografiikasta erittäin vaikealukuisen. [Tufte 2001: 107–123.]



Kuva 2. American Education -lehden yliopisto-opiskelijainfografiikka. Tuften kirjassaan esittämä esimerkki kuvioroinan ja turhan koristeellisuuden haitoista. [Tufte 2001: 118]

Informaatiotiheys tarkoittaa esitettävän datan määrää sen viemän pinta-alan suhteen. Tufte argumentoi, että ihmiset ovat erinomaisia hahmottamaan pieniäkin yksityiskohtia heille esitetystä aineistosta, joten datavisualisoinneissa tulisi pyrkiä maksimoimaan esitetyn datan määrä pinta-alan suhteen. Jotta luettavuus varmistetaan, on kuitenkin käytettävä harkintaa. Yksi tapa parantaa visualisoinnin informaatiotiheyttä on pienentää esitettävän kuvion kokoa. [Tufte 2001: 161–169.]

Tuften argumentoinnissa voidaan havaita graafiselle modernismille tyypillinen vähemmän on enemmän -asenne. Tufte painottaa voimakkaasti selkeyden tärkeyttä: mikään ei saa astua datan edelle tai ylitse visualisoinnissa. Vaikka nämä periaatteet pohjautuvat Tuften alkuperäisteokseen 1970-luvulta, ne ovat erittäin käyttökelpoisia myös tänä päivänä. Tätä voimakkaasti tilastotieteisiin pohjautuvaa, funktionalistista näkökulmaa on kuitenkin myöhemmin tutkittu, siitä on keskusteltu ja sitä on myös kritisoitu [Few 2011: 1]. Esimerkiksi niin sanotulla kuvioroinalla voidaan tutkitusti parantaa informaation mieleenpainuvuutta: Saskatchewanin yliopistossa toteutettu tutkimus havaitsi, että koristeellisemman kuvion esittämä informaatio jäi huomattavasti paremmin tutkittavien mieleen kolmen viikon kontrollijakson lopulla. Heikompi data-mustesuhde, ylimääräinen koristeellisuus ja näiden mahdollisesti kasvattama vale-kerroin ei kuitenkaan haitannut tutkittavien kykyä hahmottaa kuvion sisältämää dataa [Bateman et. al. 2010: 6–10]. Etenkin journalistisessa käyttötarkoituksessa on paras mahdollinen visualisointi jossain kultaisella keskitiellä funktionaalisen ja koristeellisen esitystavan välillä. University of Miamin viestinnän apulaisprofessori Alberto Cairo onkin kehittänyt Tuften käsitystä kattavamman työkalun datavisualisoinnin hyvyyden arviointiin, visualisointiympyräkuviomallin (kuva 3) (engl. visualization wheel). Tämän ympyrämallin akseleiden avulla voidaan subjektiivisesti arvioida datavisualisoinnin tasapainoa sen informaation ja informatiivisuuden suhteen korostaen Cairon tärkeänä pitämiä vastapareja [Cairo 2013: 125–138].



Kuva 3. Alberto Cairon ympyrämalli visualisoinnin laadun arviointiin [Cairo 2013: 126].

Cairo avaa mallin akseleiden merkityksiä kirjassaan seuraavasti:

- Abstraktio – esittävyys (engl. abstraction – figuration) kuvaa, kuinka visuaalisesti lähellä tai kaukana realismia visualisointi on. Esimerkiksi tekninen piirros esineestä esitettäessä sen kokoa on voimakkaasti esittävä, kun taas kokoa havainnollistava geometrinen kuvio on erittäin abstrakti. [Cairo 2013: 128–131.]

- Funktionaalisuus – koristeellisuus (engl. functionality – decoration) kuvaa niin sanotun kuvioroinan eli dataa esittämättömän koristeellisuuden määrän suhdetta muuhun sisältöön [Cairo 2013: 131–132].

- Tiheys – keveys (engl. density – lightness) kuvaa esitettävän datan määrää tiheyttä käytettävissä olevalla pinta-alalla [Cairo 2013: 132].

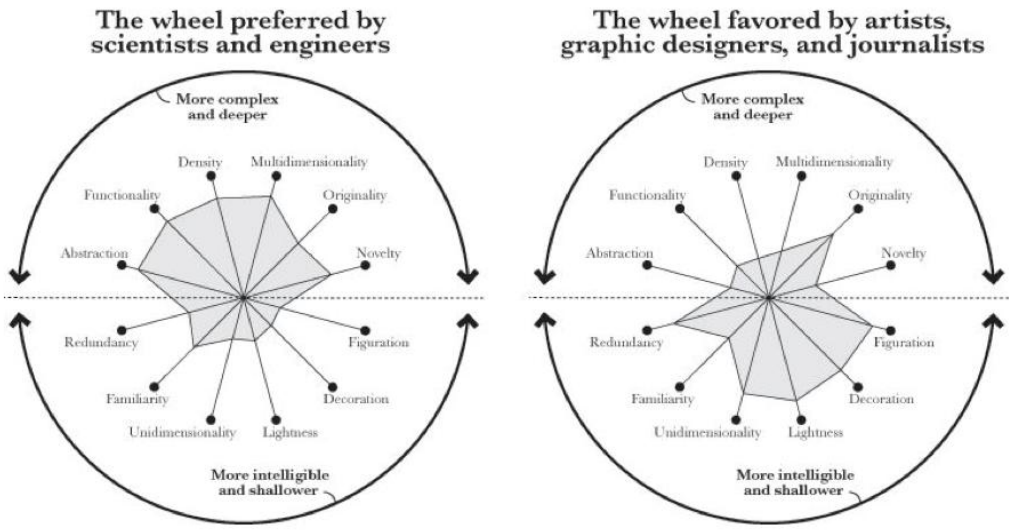
- Moniulotteisuus – yksiulotteisuus (engl. multidimensionality - unidimensionality) kuvaa, kuinka monipuolisesti ja kuinka monelta kannalta esitettävää dataa voidaan tarkastella visualisoinnin avulla. Erittäin yksiulotteinen

visualisointi antaa varsin yksipuolisen tai yksiselitteisen kuvan datasta, kun taas moniulotteinen visualisointi tarjoaa runsaasti informaatiota mahdollistaen useita näkökulmia dataan. [Cairo 2013: 134–135.]

- Ainutlaatuisuus – tuttuus (engl. originality – familiarity) kuvaa esitystavan tuttuutta tai yleisyyttä eli sitä, kuinka tuttuja lukijat todennäköisesti olevat esitystavan kanssa. Esimerkiksi piirakka-, palkki- ja viivadiagrammit ovat erittäin yleisiä ja siksi tuttuja sekä helposti ymmärrettäviä suurelle osalle ihmisistä, kun taas circo- tai sankeydiagrammit ovat harvinaisempia ja sitä myötä vähemmän tuttuja. [Cairo 2013: 136–137.]

- Uutuus – toisto (engl. novelty – redundancy) kuvaa esitettyjen uusien asioiden suhdetta jo esitettyjen asioiden toistoon. Toistamalla informaatiota voidaan parantaa visualisoinnin ymmärrettävyyttä, mutta uusi informaatio pitää sen kiinnostavana lukijoille. [Cairo 2013: 137–138.]

Cairo painottaa, että hänen ympyrämallinsa toiminta perustuu erittäin subjektiiviseen arviointiin, joten tarkka tai objektiivisuuteen pyrkivä visualisoinnin hyvyyden arviointi on tällä työkalulla mahdotonta. Kuitenkin visualisoinnin suunnittelussa käytännössä voi malli auttaa paremmin hahmottamaan visualisoinnin tyyliä, toimivuutta ja tasapainoa, jotta sitä voisi kehittää haluttuun suuntaan. Tulee kuitenkin muistaa, että visualisoinnin hyvyys ja sen oikea tasapaino edellä mainituilla mallin akseleilla riippuu aina soveltuvuudesta tavoitellulle yleisölle, eikä suoraa kaavaa hyvän visualisoinnin luomiseen voi Caironkaan työkalulla esittää [Cairo 2013: 175–201]. Eri kohdeyleisöt arvostavat erilaisia ominaisuuksia visualisoinneissa; Cairo arvioi itse kahden suurimman infografiikkaa ja visualisointeja tuottavan ryhmän, teknisen tai taiteellisen taustan omaavien visualisointien tuottajien välisiä lähestymistapaeroja infografiikan ja visualisointien luomiseen ja arvostamiseen kuvan 4 mukaisilla kuviomalleilla. Cairo arvioi kuviomalleissaan teknis- ja tieteellistaustaisten tuottajien arvioivan laadukkaaksi moniulotteisempia ja tietorikkaampia visualisointeja, taiteellis-, suunnittelu- ja journalististaustaisten tuottajien arvostaessa enemmän selkeyttä ja yksinkertaisuutta [Cairo 2013: 144–148].



Kuva 4. Cairon arvio teknisesti orientoituneiden ja taiteellisesti orientoituneiden datavisualisointien tuottajien näkökulmaeroja datavisualisointien laadun arvioinnissa [Cairo 2013: 147.]

2.4 Digitaalinen datavisualisointi

Digitaalisen median työkalut ja tekniikat sopivat erinomaisesti datavisualisoinnin tarkoitukseen tarjoamalla mahdollisuuden rikkaampaan ja voimakkaampaan esitystapaan kuin perinteiset mediat ja grafiikkaesitykset. Digitaaliset esitystavat voivat tarjota uudenlaista vuorovaikutteisuutta lukijan ja median välillä, yhdistää erilaisia mediasisältöjä kuten kuvaa, videota ja ääntä kokonaisvaltaisiksi kokonaisuuksiksi ja esittää laajoja tietoaineistoja intuitiivisilla tavoilla. Datajournalistisissa hankkeissa onkin yleisesti päädytty käyttämään voimakkaasti vaihtelevia uusia teknologioita luomaan grafiikkaa datasta, sen sijaan että dataa tarvitsisi erikseen tulkita uudelleen grafiikkaohjelmassa [Gray ym. 2012: 72–161].

Käyttämällä verkon vakiintuneita teknologioita, kuten HTML-, CSS- ja Javascript -kieliä mahdollistetaan lisäksi myös grafiikkaesitysten luontainen sopivuus verkkojulkaisuun. Digitaalisen datavisualisoinnin luomiseen voidaan käyttää joko yleisluontoisia visualisoinnin ja grafiikan teknologioita tai varta vasten tiedon esittämiseen suunniteltuja työkaluja. Koska teknologioiden, ohjelmistojen ja visualisointikirjastojen kirjo on erittäin laaja, on niiden käyttö toimituksissa myös vaihtelevaa. Yleisin työnkulku vaikuttaa kuitenkin toistaiseksi perustuvan avoimiin Javascript-pohjaisiin kirjastoihin [Mäkinen 2013] tai valmiisiin, esimerkiksi Googlen tarjoamiin, verkkopohjaisiin visualisointialustoihin

[Poikola 2013]. Tässä työssä keskitytään avoimiin Javascript-ratkaisuihin, joista esimerkkejä käydään tarkemmin läpi osiossa Datan visualisointi Javascriptillä.

3 Datajournalistisen verkkosovelluskehityksen tekniikat ja teknologiat

Verkkosovelluskehitys on edistynyt runsaasti viime vuosien aikana parempien verkkoselainten, tehokkaampien mobiililaitteiden ja uusien standardien myötä. Tässä luvussa pyritään aluksi määrittelemään nykyisen, Javascript-pohjaisen, verkkokehityksen eroja perinteisiin tapoihin rakentaa verkkosovelluksia ominaisuus ominaisuudelta ja tutkimaan modernien teknologioiden ja tekniikoiden etuja perinteisiin menetelmiin nähden. Lisäksi perehdytään tarkemmin verkkosovelluskehityksen eri osiin ja tutkitaan eri vaihtoehtoja toteuttaa toiminnallisuuksia datajournalistisen sovelluskehityksen kannalta.

Javascript end-to-end eli Javascript päästä päähän tai Javascript alusta loppuun on käsite, joka viittaa tässä luvussa esiteltävien modernien verkkosovellusteknologioiden mahdollistamaan Javascriptin tai Javascript-pohjaisten teknologioiden käyttöön verkkosovelluksen kaikissa ydintoiminnallisuuksissa. Esiteltyjen teknologioiden ja tekniikoiden avulla on mahdollista käyttää Javascriptiä palvelinpuolen, tietokannan ja käyttäjärajapinnan toteutuksissa, mihin perinteisillä menetelmillä on vaadittu usean ohjelmointikielen osaamista. [Mikowski & Powell 2014: 3–4.]

Luvussa esiteltävät tekniikat ja teknologiat mahdollistavat monipuolisten interaktiivisten verkkosovellusten kehityksen ja ovat näin olennaisia datajournalistisessa sovelluskehityksessä. Interaktiivisuus, vuorovaikutteisuus ja sovelluskehityksen nopeus nousevat datajournalistisessa sovelluskehityksessä olennaiseen rooliin. Interaktiivisuus erottaa datajournalistiset sovellukset perinteisistä tavoista toteuttaa visuaalista journalismia: ilman näitä teknologioita esitykset koostuisivat staattisista näkymistä.

3.1 Asynkroniset verkkosovellukset

Perinteisesti verkkosovellusten voidaan sanoa koostuvan kahdesta osasta: asiakasnäkymästä ja palvelinsovelluksesta. Asiakasnäkymä esittää sovelluksen käyttäjän näkemän sisällön ja palvelinsovellus suorittaa varsinaisen sovelluksen toiminnallisuuksiin

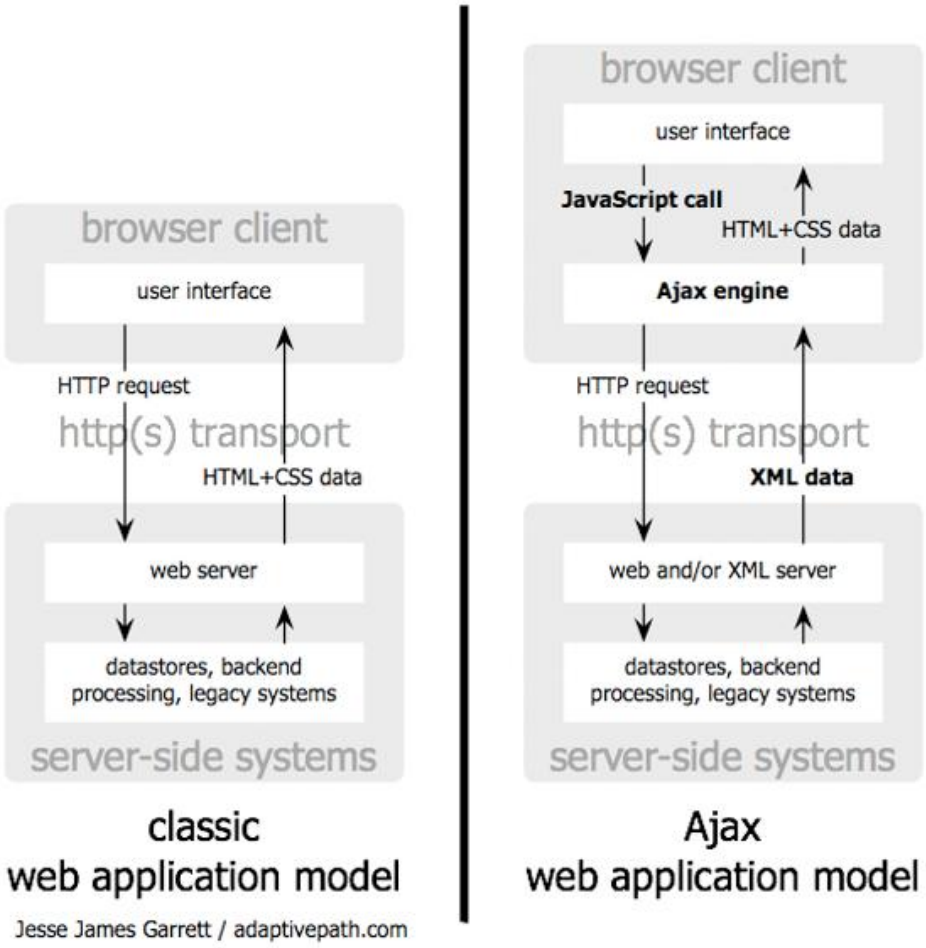
liittyvän logiikan. Yksinkertaisimmillaan verkkosovelluksen tai sivuston asiakasnäkymä koostuu HTML-rakenteesta ja CSS-tyylimäärittelystä sekä mahdollisesti Javascript-ohjelmointikielellä toteutettavasta toiminnallisuudesta. Palvelinsovellus toteutetaan erillisellä ohjelmointikielellä, joista yleisin on PHP [Historical yearly trends… 2014].

Tätä kahtiajakautunutta rakennetta, jossa näkymä ja toiminnallisuus erotellaan toisistaan, voi yhä pitää verkkosovellusten suunnittelun perustana. HTML- ja CSS-koodi määrittelevät sovelluksen ulkoasun ja rakenteen, ja palvelinteknologia mahdollistaa sisällön dynaamisen päivittämisen, tiedon tallentamisen ja tallennetun tiedon hakemisen tietokannasta. Kommunikaatio käyttäjänäkymän ja palvelimella suoritettavan toimintalogiikan välillä toteutetaan HTTP-pyynnöillä palvelimelle. Käyttäjäinteraktion perusteella asiakasnäkymä lähettää pyynnön palvelimelle. Pyynnön perusteella palvelinsovellus toteuttaa käyttäjän pyytämän toiminnallisuuden, kuten päivittää sivuston sisältöä, käsittelee sille syötettyä dataa tai tekee tietokantahakuja ja palauttaa HTML- ja CSS-koodista muodostuvan asiakasnäkymän käyttäjälle.

Perinteisesti nämä pyynnöt ja niiden käsittely toteutettiin synkronisesti eli pyyntö kerrallaan jättää asiakassovelluksen ja käyttäjän odottamaan palvelinsovelluksen vastausta sen käsitellessä pyyntöä [Garrett 2005]. Tämä ongelma on pyritty ratkaisemaan kehittämällä teknologioita, jotka mahdollistavat asiakas- ja palvelinsovellusten kommunikoinnin asynkronisesti, eli ilman koko sivun uudelleenlatausta, minkä pohjalta syntyi Ajax. Ajax ei ole yksittäinen teknologia, vaan kokoelma teknologioita ja tekniikoita, jotka mahdollistavat toiminnallisuuksillaan työpöytäsovellusmaisten verkkosovellusten luomisen [Garrett 2005]. Ajax ja asynkronisuus mahdollistavat siis verkkosovelluksen asiakassovelluksen käyttöliittymän osien päivittymisen reaaliaikaisesti ilman sivulatauksia. Tämä toiminnallisuus voidaan nähdä olennaisena datajournalististen sovellusten kannalta; interaktiivisuudella voidaan erottaa datajournalismi perinteisistä visuaalisen journalismin muodoista.

Ajax perustuu verkkoselaimissa tulkattavaan Javascript-ohjelmointikieleen, jonka tulkkaus on ollut osa verkkoselainten toiminnallisuutta jo 1990-luvulla [Flanagan 2006: 1–4]. Kokonaisuuden kannalta tärkeä teknologia on kuitenkin 2000-luvun puolenvälin jälkeen verkkoselaimissa yleistynyt XMLHttpRequest-rajapinta (XHR), joka antoi Javascriptille mahdollisuuden tehdä HTTP-pyyntöjä suoraan palvelimille [McLellan 2005]. Termin Ajax tunnetuksi tehnyt suunnittelija Jesse James Garrett kuvailee Ajax-pohjaisen verkkosovelluksen eroa perinteiseen verkkosovellukseen kuvan 5 mukaises-

ti. Perinteisessä verkkosovelluksessa asiakassovellus tekee pyyntöjä suoraan palvelinsovellukselle ja saa palautteena kokonaisia uusia HTML- ja CSS-kokonaisuuksia. Ajax-pohjainen asiakassovellus taas kommunikoi palvelimen kanssa ainoastaan Ajax-moottoriksi kutsuttavan Javascript-sovelluksen välityksellä. JS-sovellus lähettää palvelinsovellukselle pyyntöjä, jotka palauttavat dataa valmiin HTML- ja CSS-koodin sijaan. Tätä dataa käsittelemällä JS-sovellus esittää käyttäjänäkymässä HTML- ja CSS-koodia: osa sovelluslogiikasta on siis siirretty asiakassovellukseen, ja palvelimelta siirtyy asiakassovellukseen vain dataa eikä valmista koodia [Garrett 2005].



Kuva 5. Garrettin kuvaus perinteisen ja modernin Ajax-pohjaisen verkkosovelluksen toiminnallisista eroista [Garrett 2005].

3.2 Sovellusarkkitehtuuri: SPA-sovellukset

Viitatattaessa asynkroniseen tiedonvaihtoon perustuviin dynaamisiin yhden sivun sovelluksiin puhutaan usein englanninkieliseen termiin single-page application perustu-

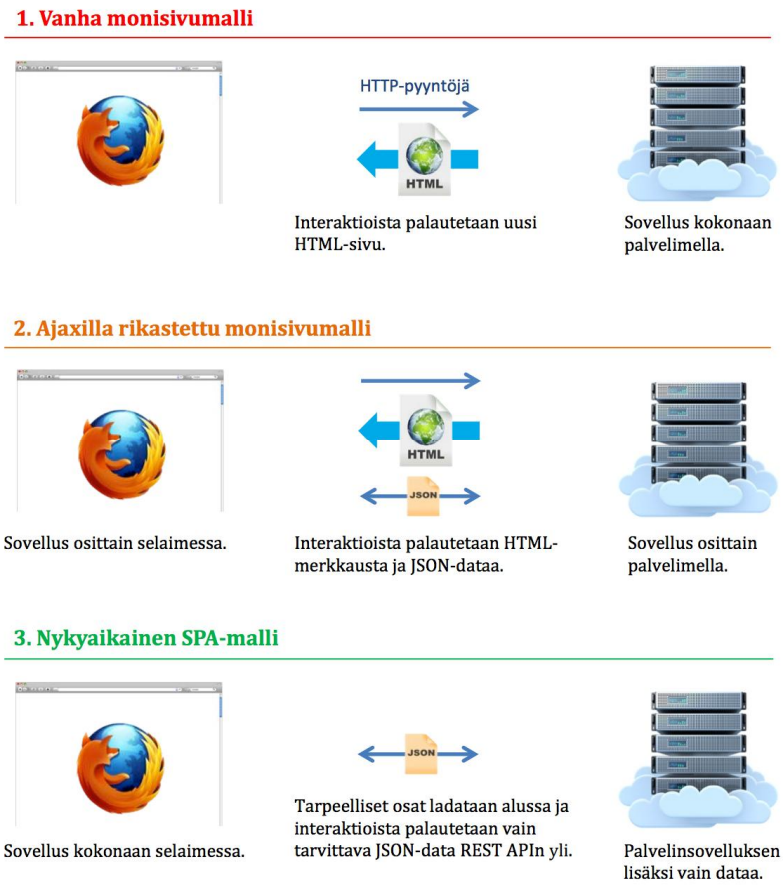
van lyhenteen mukaisesti SPA-mallista ja SPA-sovelluksista. SPA-mallia, eli yhden sivun verkkosovellusarkkitehtuuria, voidaan pitää modernina tapana kehittää sovelluksia verkkoon [Mielonen 2013: 26].

Asynkronisuus mahdollisti Javascript-pohjaisten verkkosovellusten rakentamisen, mikä antoi alkusysäyksen uudentapaiselle verkkokehitykselle. Ajax mahdollistaa käyttäjänäkökulmasta poikkeamisen perinteisestä pyyntö-vastausmallista. Pyyntöjä voidaan tehdä käyttäjän niitä havaitsematta ja esittää sovellus käyttäjälle yhtenä sivuna Javascript-teknologioiden avulla. Toisin sanoen, sovellus käynnistetään käyttäjänäkökulmasta kertaalleen, eikä sovelluksen kanssa interaktio vaadi tämän jälkeen sivun uudelleenlatausta. SPA-sovelluksilla on aiemmin myös viitattu esimerkiksi Flash- tai Java-pohjaisiin verkkosovelluksiin, jotka Javascript-sovelluksista poiketen vaativat erillisen selainliitännäisen toimiakseen [Mikowski & Powell 2014: 3–9].

SPA-malli mahdollistaa pöytätietokonesovellusten tyyppisten verkkosovellusten toteuttamisen verkkoselaimessa. Käytännössä tällä tarkoitetaan vain yhden täyden sivulatauksen vaativia sovelluksia. Ensimmäisen sivulatauksen jälkeen sovelluksen sisällön päivittyvät osat voidaan Ajax-teknologioita hyödyntäen ladata asynkronisesti osa kerrallaan, mikä poistaa sivun uudelleenlatauksen tarpeen. Myös sovelluksen rakenteesta saadaan modulaarisempi: palvelimella toimiva taustajärjestelmä on toiminnallisesti irrallinen selainsovelluksesta, toisin sanoen osat ovat helpommin vaihdettavissa tai kehitettävissä. Olennaista tässä modulaarisuudessa on taustajärjestelmän suunnittelussa käytettävä rajapinta-ajattelu. Taustajärjestelmän tulee toimia rajapintana varsinaiseen dataan, jota selain- tai muu käyttäjäsovellus esittää. SPA-mallissa kaikki sovelluksen toiminnallinen logiikka pyritään pitämään Javascriptillä toteutettuna selaimessa, mikä poikkeaa voimakkaasti niin sanotusta klassisesta verkkosovelluskehityksestä, jossa toimintalogiikka on joko täysin tai osittain palvelinsovellukseen rakennettu.

Taustajärjestelmä ja käyttäjäsovellus keskustelevat toistensa kanssa rajapinnan välityksellä, minkä avulla data siirtyy sovelluksien eri osien välillä. Tyypillinen tiedonsiirtoformaatti on JS-objektien tallennusmuotoa mukaileva JSON, joka mahdollistaa datan yksinkertaisen käsittelyn käyttäjäsovelluksen päässä. Perinteisen ja modernin mallin eroa esitetään kuvassa 6: SPA-mallin mukaisessa sovelluksessa sovelluksen asiakas- ja palvelinsovelluksen välillä siirtyy vain tarvittava määrä JSON-muotoista dataa, esimerkiksi REST-rajapinnan välityksellä, mikä tekee siitä soveltuvan arkkitehtuurimallin datapohjaisten sovellusten suunnitteluun, jossa muuttuva tekijä käyttäjänäkymässä

koostuu data-aineistosta. SPA-sovelluksille määrittävää on siis se, että asiakassovellus on kokonaan selaimessa ja palvelinsovellus lähettää tälle vain tietoa, ei uusia HTML-sivuja: kaikki HTML-rakenteen muokkaaminen toteutetaan asiakassovelluksessa. [Mielonen 2013: 26–29.]



Kuva 6. Perinteisen verkkosovelluskehityksen monisivumallin, Ajaxilla rikastetun hybridimallin ja SPA-mallin väliset erot [Mielonen 2013: 28].

SPA-mallia käyttämällä voidaan nopeuttaa verkkosovelluskehitystä, mikä on tärkeää etenkin STT-Lehtikuvan kaltaisessa uutistoimitusympäristössä. Datajournalistiset sovellustoteutukset ovat osa tavanomaista uutistarjontaa, jolloin päivän uutisaiheista sovellusten toteuttamiseen saattaa olla vain tunteja aikaa. Yksinkertainen ja selkeä arkkitehtuurimalli nopeuttaa sovelluksen rakenteen suunnittelua ja toimintojen kehitystä erottamalla palvelin- ja asiakassovelluksen käsitteellisesti toisistaan. Pelkän datan siirtäminen sovellusten osien välillä voi myös olennaisesti vähentää sovelluksen tarvitsemaa tiedonsiirron määrää, lyhentäen latausaikoja ja siten nopeuttaa sovellusta myös käyttäjän näkökulmasta.

3.3 Ohjelmointirajapinnat: REST

Tavallinen tapa toteuttaa kommunikaatio taustajärjestelmän ja selainsovelluksen välillä perustuu niin sanottuun REST-malliin (Representational State Transfer). REST on arkkitehtuurimalli, jonka tarkoituksena on välittää resursseja, kuten järjestelmän tila ja toiminnallisuudet, taustajärjestelmältä käyttäjänäkymälle. Rajapinta on täysin REST-mallin mukainen, kun se täyttää kaikki REST-määritelmän mukaiset vaatimukset [Fielding 2000: 76–86]:

- Rajapinnan tulee toimia asiakas-palvelinmallin mukaisesti, eli käyttäjänäkymän tulee olla täysin riippumaton taustajärjestelmästä [Fielding 2000: 78].
- Rajapinnan tulee olla tilaton, eli jokaisen asiakaspyynnön tulee sisältää tarvittava sovelluksen tilainformaatio taustajärjestelmän toimintaa varten. Toisin sanoen, sovelluksen tila tallennetaan käyttäjänäkymässä, ei taustajärjestelmässä [Fielding 2000: 78–79].
- Rajapinnan tulee mahdollistaa välipalvelimen käyttö. Jokaisen rajapinnan vastauksen tulee sisältää tieto siitä, onko vastauksen sisältämä data tallennettavissa välimuistiin vai ei [Fielding 2000: 79–81].
- Rajapinnan tulee olla kauttaaltaan yhdenmukainen, eli kaikkien REST-resurssien tulee käyttää samanmuotoista rajapintapyyntöjä [Fielding 2000: 81–82].
- Rajapinnan tulee mahdollistaa kerroksittaisuus. Kerroksittaisuudella tarkoitetaan, että rajapinnan tulee abstrahoida mahdolliset taustajärjestelmät ja toimia yhdenmukaisesti riippumatta siitä, tuleeko vastaus esimerkiksi välimuistista tai usealta palvelimelta [Fielding 2000: 82–84].
- Rajapinnan tulee mahdollistaa Code on demand -toiminnallisuus. Code on demand -toiminnallisuudella tarkoitetaan rajapinnan mahdollisuutta laajentaa asiakasohjelmiston tai käyttäjänäkymän toiminnallisuutta siirtämällä sille rajapintaan tallennettua ohjelmakoodia [Fielding 2000: 84–86].

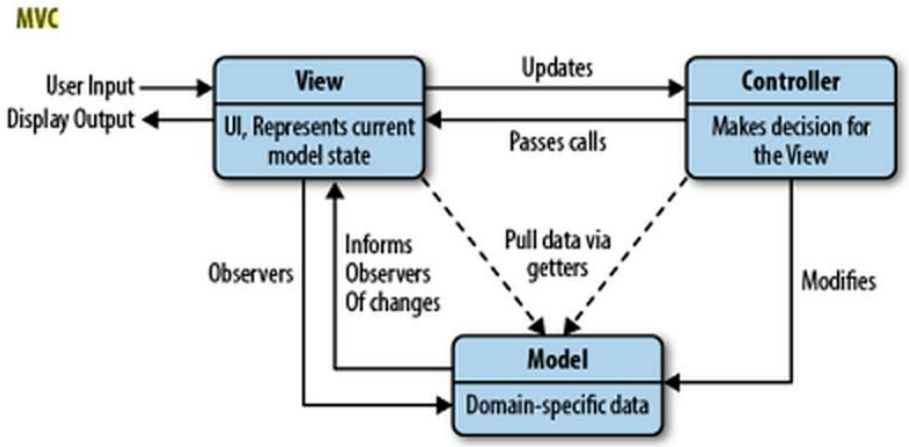
REST-arkkitehtuurimallin mukaiseen rajapintaan viitataan termillä RESTFUL eli RESTin kaltainen tai RESTin mukainen. Koska REST ei ole standardi vaan teoreettinen malli, eivät käytännön rajapintatoteutukset välttämättä seuraa kaikkia REST-periaatteita; ominaisuuksia voidaan pragmaattisuuden vuoksi jättää toteuttamatta. Tällaisia ei täysin RESTin mukaisia palveluita kutsutaan kuitenkin RESTin kaltaisiksi eli niin sanotuiksi RESTFUL-rajapinnoiksi. [Webber et. al 2010: 12–20.]

REST-rajapinta siis toimii asiakassovelluksen esittämää dataa hallitsevana käyttöliittymänä ja mahdollistaa tarvittavan CRUD-toiminnallisuuden (Create, Read, Update, Delete). REST-rajapinnan avulla asiakassovellus voi siis käsitellä tietokantaan tallennettua dataa vaatimatta suoraa tietokantayhteyttä. SPA-sovellukset eivät vaadi REST-rajapintaa toimiakseen, mutta REST-mallin mukainen rajapintatoteutus on muodostunut lähes standardiksi. [Mielonen 2013: 27]

Datajournalistisen sovelluskehityksen kannalta REST-rajapinnat luovat mahdollisuuden käyttää uudelleen ja yhdistää eri sovelluksia varten koottua dataa ilman datan uudelleenkäsittelyä. Koska data siirretään erillisestä palvelinsovelluksesta rajapintaa hyödyntäen, voidaan samasta aineistosta saman palvelinsovelluksen avulla rakentaa useita visualisointeja vaatimatta aineiston käsittelyä tai kopiointia uuteen sovellukseen. Rajapinnasta saatavaa aineistoa voidaan myös yhdistää jostain toisesta rajapinnasta saatavaan dataan ja luoda täysin uusia käyttäjäsovelluksia rakentamatta uutta palvelinsovellusta.

3.4 Käyttöliittymäarkkitehtuuri: MVC ja MV* -arkkitehtuuri

MVC (Model, View, Controller) eli malli-näkymä-ohjainarkkitehtuuri on 1970-luvulla alkunsa saanut sovelluskehityksessä käytettävä käyttöliittymäsuunnitteluun liittyvä sovellusarkkitehtuurityyli, jossa sovellus jaetaan kolmeen toiminnalliseen osaan: malliin, näkymään ja ohjaimeen. Yksinkertaisimmillaan malli vastaa sovelluksen taustalogiikasta ja sisällöstä, näkymä ulkoasusta ja ohjain käyttäjän syöttämien komentojen vastaanottamisesta ja tulkitsemisesta. Käyttäjän syöttäessä sovellukseen komennon ohjain ottaa komennon vastaan näkymän kautta ja tulkitsee sen mallille. Malli suorittaa sen ohjaimelta saamat pyynnöt, jolloin sen tila muuttuu. Näkymä esittää tämän muuttuneen tilan, jolloin käyttäjä näkee komentonsa tuloksen käyttöliittymässä. Tätä osien vuorovaikutusta esitetään kuvassa 7. [Laine 2008: 17–22; Osmani 2012: 111–113.]



Kuva 7. MVC-arkkitehtuurin mukaan suunnitellun sovelluksen toiminta ja mallin osien interaktiomalli [Osmani 2012: 113].

MVC-arkkitehtuurin mukainen rakenne on noussut myös JS- ja verkkosovelluskehityksessä suureen suosioon; tyylin käyttöä varten on kehitetty useita suosittuja sovelluskehyksiä, kuten Backbone, Ember ja AngularJS. Verkkosovellusten kehittyessä yhä monipuolisemmiksi ja sitä myöten monimutkaisemmiksi on ohjelmistotuotannon järjestelmällinen ylläpito enenevässä määrin tärkeää myös verkkosovelluskehityksessä. MVC-sovelluskehykset auttavat ylläpitämään ohjelmistokoodin modulaarisuutta pitämällä sovelluksen osat erillisinä toisistaan. Verkkosovelluksissa MVC-tyyliä ei kuinkaan aina käytetä kauttaaltaan; usein esimerkiksi ohjaimen toiminnallisuus yhdistyy näkymän kanssa. Tästä syystä puhutaankin verkkosovelluskehityksessä usein MV*-arkkitehtuurista. Muita MV*-arkkitehtuurityyliin pohjautuvia malleja ovat esimerkiksi MVP (Model, View, Presenter), jossa ohjain on korvattu esittäjällä, joka on toiminnaltaan samankaltainen mutta toimii näkymän kanssa samassa tasossa, tai Microsoftin MVVM (Model View ViewModel), jossa ohjaimen korvaa näkymän tilaa abstrahoiva näkymämalli. [Osmani 2012: 111–138.]

MV*-arkkitehtuurin käyttö verkkosovelluskehityksessä mahdollistaa datan ja toiminnallisuuden erottamisen ulkoasusta, minkä mukaista lähestymistapaa myös WWW:n standardisointiorganisaatio W3C:n (World Wide Web Consortium) WCAG-ohjeistuksessa (Web Content Accessibility Guidelines) suosittaa kaiken rakenteen erottamiseksi ulkoasumäärittelystä [G140: Separating information and structure… 2014]. Käyttäjäsovelluksen käsiteltävissä olevaa dataa voi säilyttää palvelimella ja sovelluksen malli sisältää ohjeet datan käsittelyyn ja ohjain ohjeet käyttäjäinteraktion tulkitsemiseen [Mielonen 2013: 31].

MV*-mallin mukaisella mallin, näkymän ja ohjaimen erottamisella voidaan helpottaa monimutkaisten verkkosovellusten rakenteen hallintaa. Datajournalististen sovellusten kehittyessä yhä monipuolisemmiksi voidaan MV*-mallia käyttämällä selkeyttää sovelluksen rakennetta pitämällä toiminnallisuudet erossa toisistaan.

3.5 Tietokantajärjestelmät: NoSQL

Toimiakseen tietoa käsittelevä ja esittävä verkkosovellus tarvitsee mahdollisuuden tallentaa ja säilyttää tätä sen toiminnalle olennaista tietoa jonkinlaisessa tietokannassa. Verkkosovelluskehityksessä yleisin tapa toteuttaa tiedonhallintaa on niin sanottu relaatiotietokanta. Solid IT -konsulttiyrityksen tarjoaman tietokantajärjestelmien suosittuutta tarkkailevan DB-Engines-sivuston mukaan suosituimmat tietokantajärjestelmät ovat Oraclen Oracle RDBMS ja MySQL sekä Microsoftin SQL Server [DB-Engines Ranking 2014]. Relaatiomallin mukaisessa tietokannassa tieto esitetään tietueina, missä tietyn tyyppiset asiat esitetään samassa taulussa, jossa jokainen tietue on yksi taulun rivi. Taulun sarakkeet muodostuvat joukosta ominaisuuksia, joiden avulla kuvataan sen sisältämän tiedon piirteitä. Taulut ja niiden sisältämät tietotyypit on määriteltävä etukäteen ennen tiedon syöttämistä tauluun, ja taulujen välille määritellään yhteyksiä eli relaatioita taulukohtaisten tunnisteiden avulla. Relaatiotietokantojen sisältämää tietoa käsitellään standardisoidulla SQL-kielellä (Structured Query Language), jonka avulla voidaan poistaa, lisätä ja muokata tietokannan sisältöä. [Codd 1982; Hovi ym. 2005: 7–11; Silberschatz ym. 2011: 39–104.]

Relaatiotietokantamallin rinnalle on verkkosovellusten monimuotoistumisen ja käsiteltävien tietomäärien kasvaessa syntynyt vaihtoehtoinen NoSQL, eli nimensä mukaan SQL:lle vaihtoehtoisia ratkaisuja käyttävä tietokanta-arkkitehtuurimalli. NoSQL on terminä kuitenkin jokseenkin harhaanjohtava, koska sillä käytännössä viitataan uusimpaan aaltoon tietokantaratkaisuja, jotka eivät ole relaatiomallin mukaisia. SQL on olennainen osa relaatiotietokantojen toimintaa, muttei osa varsinaista tietokantarakennetta josta termillä NoSQL pyritään erottautumaan. Termin NoSQL nykymerkityksessään ja koko ajatuksen moderneista hajautetuista ei-relaatiotietokannoista voidaan nähdä lähteneen Googlen ja Amazonin palvelujen tietomäärä hallitsevien tietokantajärjestelmien, BigTablen ja Dynamon, kehityksestä. Google ja Amazon julkaisivat tietokantajärjestelmiänsä kuvaavat artikkelit [Chang ym. 2008; DeCandia ym. 2007], mikä kasvatti myös

pienempien toimijoiden kiinnostusta uudentyyppistä tietokanta-arkkitehtuuria kohtaan. [Sandborg 2012: 23–24; Tiwari 2011: 4–7.]

Nämä uudentyyppiset tietokantatoteutukset perustuvat useimmiten huomattavasti relaatiomallia yksinkertaisempaan tietomalliin, kuten avain-arvo- (engl. key-value-model), dokumentti-, graafi- tai sarakemalliin, ja ne on suunniteltu paremmin skaalautuviksi ja helpommin hajautettaviksi kuin perinteiset teknologiat. NoSQL-tietokantajärjestelmät mahdollistavat tiedon automaattisen toisintamisen ja hajauttamisen usealle eri fyysiselle palvelimelle, eli niin sanotun horisontaalisen skaalautumisen. Tällä pystytään takaamaan kokonaisjärjestelmän saatavuus ja skaalautuvuus: resursseja voidaan lisätä tarvittaessa ilman, että yksittäisten palvelinten tehoa tarvitsisi parantaa eli skaalata järjestelmää vertikaalisesti. NoSQL-tietokannat tukevat myös usein skeemattomia tietorakenteita: Skeemattomuus viittaa siihen, ettei tietokannan käyttämää tietorakennetta tarvitse määritellä tarkkaan etukäteen kuten relaatiotietokantatoteutuksissa, vaan sitä voidaan muokata sovelluskehityksen aikana tarvittaessa. Esimerkiksi dokumenttitietomallin mukaiset niin sanotut dokumenttiorientoituneet tietokannat ovat skeemattomia [Ritchie 2010; Sandborg 2012: 5–11, 25–31; Tiwari 2011: 4–10, 137–148]

Ei-relaatiomallisten tietokantojen etuina voidaan siis pitää etenkin horisontaalista skaalautuvuutta ja skeemattomuuden tuomaa tietomallin vapautta. Tietomallin vapaus nopeuttaa NoSQL-tietokantajärjestelmän käyttöönottoa, mutta saattaa myös muodostua ongelmaksi sen myötä, ettei tiedostorakennetta tarvitse suunnitella yhtä pitkälle etukäteen kuin relaatiotietokantojen kanssa työskenneltäessä. Horisontaalinen skaalautuvuus mahdollistaa NoSQL-järjestelmien tehokkaan toteuttamisen pilvipalveluympäristöissä [Konstantinou ym. 2011]. Lisäämällä virtuaaliresurssi-instanssien lukumäärää voidaan suoraan lisätä tietokannan pyynnönkäsittelykapasiteettia. Suurin hidaste NoSQL-järjestelmien käytännön käyttöönotossa lienee järjestelmien suhteellinen nuoruus: vanhimmatkaan NoSQL-järjestelmät eivät ole vielä olleet käytössä tai kehityksessä kymmentä vuotta. Tähän verrattuna perinteinen relaatiotietokanta tuo läpikotaisin testattua ja standardoitua toiminnallisuutta.

Suosituimmaksi NoSQL-tietokantajärjestelmäksi on kasvanut avoimen lähdekoodin MongoDB, jonka lisäksi suosittuja NoSQL-paradigman mukaisia tietokantajärjestelmiä ovat niin ikään avoimet Apache Cassandra ja Redis [DB-Engines Ranking 2014]. MongoDB on automaattista skaalausta tukeva, hyvää suorituskykyä ja saatavuutta varten

suunniteltu dokumenttiorientoitunut tietokantajärjestelmä [Introduction to MongoDB 2014]. Dokumenttiorientoituneisuudella tarkoitetaan tietokannan noudattavan tietomallia, jossa jokainen tallennettava tietue ja siihen sisältyvä data voidaan käsittää dokumenttina, joka sisältää kaiken itseensä liittyvän tiedon. Dokumentit ovat skeemattomia, eli niiden sisältöä ei tarvitse tarkkaan määritellä ennen niiden syöttämistä. MongoDB:n käyttämä dokumenttisyntaksi on JSONin kaltainen BSON. BSON-dokumenttirakenne on binäärinen representaatio JSON-dokumentista [MongoDB Architecture Guide 2014]. Toisin sanoen, JSON-muotoinen data soveltuu vähällä muokkauksella suoraan MongoDB-tietokantaan. [Ritchie 2010.]

NoSQL-tietokantajärjestelmät antavat sovelluskehittäjälle helposti käyttöönotettavan rungon datapohjaisille sovelluksille ja nopeuttava näin datajournalististen sovellusten toteuttamista. Nopeuden tärkeys korostuu uutistoimitusympäristössä, jossa sovellukset tulee toteuttaa osana päivän uutisvirtaa, usein hyvin nopealla aikataululla. Skaalautuvat NoSQL-tietokantapalvelut helpottavat sovelluksen palvelinarkkitehtuurin suunnittelua ja mahdollistavat sovelluksen resurssien lisäämisen käytön aikana. Tämä on erityisen tärkeää STT-Lehtikuvan jakelumallissa, jossa sovellukseen kohdistuvan käytön määrää voi olla vaikeata ennakoida.

3.6 Palvelinteknologiat: Node.js

Kuten luvussa 3.1 esitettiin, on PHP yhä verkkosovellusten palvelinpuolen taustajärjestelmien rakentamisessa yleisin käytetty ohjelmointikieli. PHP:n lisäksi yleisiä palvelinpuolen teknologioita ovat Microsoftin palvelinjärjestelmissä käytetty ASP.NET ja Oraclen Java [Historical yearly trends… 2014]. Yleistä palvelinpuolen ohjelmistokokonaisuutta, joka koostuu PHP:n lisäksi laajasti käytetystä palvelinpuolella käytettävästä Unix-pohjaisesta Linux-käyttöjärjestelmästä [Usage statistics and market share of Unix for Websites 2014], Apachen HTTP-palvelimesta [June 2014 Web Server Survey 2014] ja MySQL-tietokantajärjestelmästä, kutsutaan akronyymillä LAMP. P:llä voidaan viitata myös Perl- tai Python-ohjelmointikieleen, mutta PHP on näistä ylivoimaisesti yleisin. LAMP ei ole varsinainen palvelinarkkitehtuuri tai sovellus, vaan nimitys yleisimmin käytetylle kokoonpanolle avoimen lähdekoodin palvelinsovelluksia. LAMP koostuu siis useasta eri sovelluksesta, jotka yhdistettynä tarjoavat verkkopalvelinkokonaisuuden, joka soveltuu verkkosovellusten rakentamiseen: HTTP-palvelin ohjaa käyttäjän palvelimelle kohdistuvia pyyntöjä ja suorittaa PHP-skriptejä pyyntöjen mukaisesti. PHP-

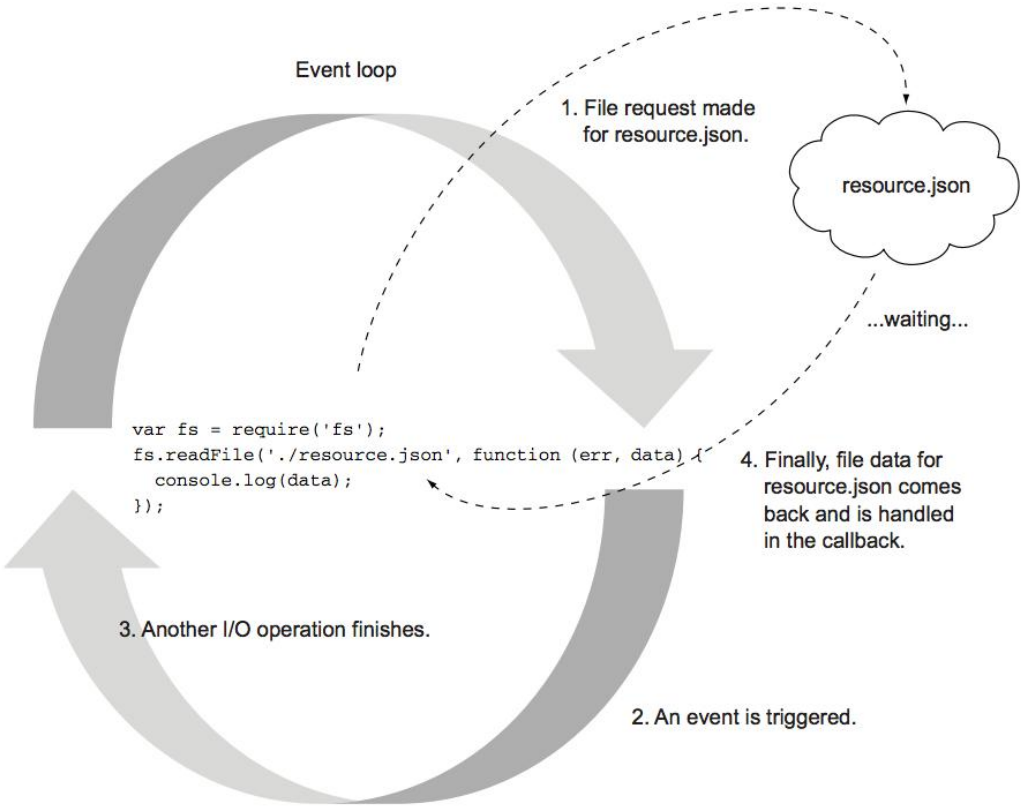
skriptit puolestaan palauttavat dataa, usein HTML-rakennetta, käyttäjänäkymään esitettäväksi [Lawton 2005; Nixon 2012: 1–14].

LAMP vaatii sovelluskehittäjältä useiden eri sovellusten ja ohjelmointikielisten laaja-alaista osaamista. Verkkosovelluksen toteuttamista varten vaaditaan yleensä valitun käyttöjärjestelmän toiminnallisuuksien hallitsemisen lisäksi ainakin palvelinpuolen ohjelmointikielen, palvelinohjelmiston, Javascriptin, HTML:n ja CSS:n hallintaa. Lisäksi useasti tarvitaan palvelinohjelmiston lisäosien, kuten Apachen mod_rewrite, ja SQL-tietokantakyselykielen osaamista. Kaikkien näiden teknologioiden osaamisen kartuttaminen ja ylläpitäminen on sovelluskehittäjän näkökulmasta usein työlästä ja työnantajanäkökulmasta kallista. Tämä on yksi syy Javascript-pohjaisen node.js-palvelinteknologian kehittämiseen. Vuonna 2009 julkistettu Node.js on Googlen kehittämän V8 Javascript -tulkin ja virtuaalikoneen pohjalle rakennettu Javascriptiin perustuva palvelinympäristö, joka antaa mahdollisuuden välttää erillisen palvelinpuolen ohjelmointikielen opettelemisen, jos sovelluskehittäjä osaa Javascriptiä. [Mikowski & Powell 2014: 229–232.]

Node.js:lle ominaista on sen tapa käsitellä sille kohdistuvia pyyntöjä ja suorittaa ohjelmistokoodia. Perinteisessä Apache/PHP-toteutuksessa jokaista pyyntöä varten käynnistetään uusi Apache-prosessi tai säie ja PHP-ohjelmistokoodin suoritukseen PHP-prosessi. Node.js:n käyttämä V8-teknologia puolestaan toimii yksisäikeisesti: Sovellus suorittaa kaikki sille suunnatut pyynnöt yhdessä ja samassa tapahtumajonossa jakaen suoritettavia toimintoja pienempiin osatoimintoihin. HTTP-pyynnön saapuessa palvelimelle Node.js suorittaa pyynnölle määritellyn tapahtuman (engl. event) ja rekisteröi sen tapahtumasilmukkaan (engl. event loop). Tapahtumalle määritellään yksi tai useampia palautusfunktioita (engl. callback functions), jotka palvelin suorittaa järjestyksessä, kun tapahtuma on suoritettu. Tapahtumasilmukka käy rekisteröityjen tapahtumien jonoa läpi niin sanottuna ikuisena silmukkana ja kutsuu tapahtumien palautusfunktioita sitä mukaa, kuin tapahtumat suoritetaan. Palautusfunktioihin voidaan määrittää lisää tapahtumia, jolloin kaikki toiminnallisuus saadaan suoritettua oikeassa järjestyksessä. Silmukka käy läpi jonotettuja tapahtumia kutsuen niiden palautusfunktioita sitä myöden, kuin ne on suoritettu valmiiksi: jos valmiita tapahtumia ei ole, sovellus jatkaa silmukan läpikäyntiä kunnes niitä esiintyy.

Tämän silmukkarakenteen tarkoituksena on poistaa suoritettavien tapahtumien lukkiutuminen (engl. blocking). Lukkiutumisella tarkoitetaan sovellusprosessin pysähtymistä

sen odottaessa jonkin laskennallisesti raskaan tai muuten hitaan tehtävän suorittamista. Esimerkiksi PHP-funktion jäädessä odottamaan raskaan tietokantakyselyn tulosta lukkiutuu sovellus täksi ajaksi, sillä ohjelmakoodi suoritetaan täysin synkronisesti, eli yksi kerrallaan. Node.js:n lähestymistavassa ohjelmistokoodi suoritetaan synkronisesti, mutta hitaat I/O-operaatiot, kuten tietokantahaut ja kirjoitus- tai lukuoperaatiot, voidaan suorittaa asynkronisesti (kuva 8). Koska I/O-operaatiot eivät lukitse sovellusprosessia, Node.js soveltuu erityisen hyvin dataintensiivisten sovellusten toteuttamiseen. Myös ohjelmistokoodia voidaan suorittaa asynkronisesti erottamalla se omaan prosessiinsa. [Cois 2013; Cantelon ym. 2014: 7–11; Rauch 2012: 27–32, 69–88; Takada 2012.]



Kuva 8. Asynkroninen I/O-operaatio silmukkarakenteessa [Cantelon ym. 2014: 11].

Node.js tarjoaa sovellusten ja lisämoduulien hallintaan NPM-pakkaushallintajärjestelmän. NPM mahdollistaa Node.js-sovellusten toiminnallisuuden yksinkertaisen laajentamisen: suoritettavan sovelluksen asetuksiin voidaan määritellä pakkausvaatimuksia, jotka automaattisesti noudetaan pakkaushallintajärjestelmästä ennen sovelluksen suorittamista [npm-faq 2014]. Laajennukset tarjoavat laaja-alaisesti valmiita toiminnallisuuksia Node.js:n pohjalle, kuten valmiita tietokantakäsittelymoduuleja tai Javascript-ohjelmointia edistäviä moduuleja [Node packaged modules 2014].

Laajennukset voivat huomattavasti nopeuttaa sovelluskehitystä ja vähentää erikoistuneiden yleisten toiminnallisuuksien ohjelmoinnin tarvetta käyttövalmiiden yhteisön hyväksi toteamien toiminnallisuuksien muodossa. Samalla laajennusten käyttö ylläpitää niin ikään yhteisön hyväksi toteamia käytäntöjä sekä sovellusrakenteissa että yksittäisten toiminnallisuuksien toteutuksissa.

Yksittäisten toiminnallisuuksien lisäämisen lisäksi laajennuksia on myös koostettuna laajempiin kokonaisuuksiin: laajennukset voivat vaatia toimiakseen muita laajennuksia ja muodostaa ohjelmistokehyksiä, jotka ohjaavat koko sovellusarkkitehtuuria. Node.js-pohjaisia laajennuksia kutsutaan väliohjelmistoiksi (engl. middleware) niiden suoritustavan takia: ne suorittavat erikoistuneita, yksittäisiä toiminnallisuuksia pääohjelmiston silmukkarakenteen sisällä, muiden toiminnallisuuksien välissä. Kirjoitushetkellä suosituin Node.js-sovelluskehys on NPM-pakkaushallintajärjestelmän npmjs-sivuston seuraajamäärän mukaan Express [Node packaged modules 2014]. Se on kevyt Node.js:n palvelintoiminnallisuutta tehostamaan pyrkivä ohjelmointikehys. Expressin kehitys alkoi pian Node.js:n ensimmäisen julkaisun jälkeen, tarkoituksena yksinkertaistaa verkkokehityksessä yleisiä toiminnallisuuksia, jotka Noden rajapinnan kautta voivat muodostua vaikeasti käytettäväksi. Express-kehys määrittelee sovelluksen rakenteen ja mahdollistaa osoitteiden reitityksen, staattisten sivujen ja resurssien jakamisen palvelininstanssilta, MV*-arkkitehtuurin mukaisen näkymän erottelun toiminnallisuudesta ja paljon valmiita verkkoyhteystoiminnallisuuksia, kuten sessio- tai keksipohjaisen todennuksen. [Yaapa 2013: 41–48.]

Node.js:n avulla voidaan nopeuttaa datajournalististen sovellusten kehitystä ja täysin uusien sovellusten kehityksen aloittamista. Uusien sovellusten ja ajatusten kehittäminen nopeasti on arvokasta nopeita toteutuksia vaativassa uutistoimitusympäristössä. Jos aikaa saadaan säästettyä sovelluskehityksestä, voidaan käyttää enemmän aikaa sovelluksen loppukäyttäjälle näkyvään osaan eli varsinaiseen visualisointiin. Node.js-ympäristö tarjoaa NPM-järjestelmän kautta valmiita sovellusrunkoja, jotka edesauttavat hyväksi todettuja periaatteita noudattavien sovellusten toteuttamista ja vähentävät sovellusrakenteen ennakkoon suunnittelun tarvetta. Node.js on myös yksisäikeisen rakenteensa myötä kevyempi kuin perinteiset verkkopalvelinympäristöt ja vaatii siksi vähemmän resursseja palvelinarkkitehtuurilta. Datajournalistisisten sovellusten tapauksessa sovelluskehitysryhmät ovat usein pieniä, jolloin Node.js:n tuoma mahdollisuus toteuttaa kaikki toiminnallisuus yhdellä ohjelmointikielellä voi myös vähentää tarvittavien henkilöstöresurssien määrää.

3.7 Javascript käyttäjärajapinnassa: jQuery

Javascript, kuten luvussa 3.1 mainitaan, on jo pitkään ollut tärkeä osa verkkosovellusten käyttäjärajapinnassa käytettäviä teknologioita. Sen alkuperäinen tarkoitettu toiminnallisuus liittyy käyttäjälle näkyvän sivuston manipuloimiseen: HTML-rakennetta voidaan muuttaa ja CSS-määrityksiä muokata. Tämän ja aiemmin esitellyn Ajax-toiminnallisuuden lisäksi tärkeimpiä Javascriptin tarjoamia toiminnallisuuksia on tapahtumanhallinta (engl. event handling). Koska Javascript toimii käyttäjärajapinnassa, voidaan sen avulla toteuttaa käyttäjäinteraktioita, kuten klikkauksen tai hiirenliikutuksen havainnointia. Nämä toiminnallisuudet yhdessä mahdollistavat monipuolisten ja dynaamisten käyttöliittymien rakentamisen verkkosovelluksille. [Flanagan 2011: 307–311.]

Verkkosovellusten kasvaessa monimutkaisemmiksi tarvitaan kuitenkin panostamista etenkin vanhojen verkkoselainten toiminnan varmistamiseen. Monimutkaisempaa JS-pohjaista sovelluskehitystä helpottamaan on kehitetty useita Javascript-kirjastoja. Yleisesti Javascript-kirjastot yksinkertaistavat Javascriptin syntaksia ja helpottavat yleisimpien toiminnallisuuksien toteuttamista ottamalla huomioon mahdolliset selainten väliset erot [Flanagan 2011: 523].

Yhdeksi yleisimmistä ja laajimmalle levinneistä käyttäjärajapinnan Javascript-kirjastoista on kasvanut jQuery. Muita yleisiä kirjastoja ovat esimerkiksi dojo, Prototype ja YUI. Nimensä mukaisesti jQuery toimii hakujen pohjalta. Sen syntaksi koostuu DOM-elementteihin kohdistuvista hauista, jotka haussa määrätyn elementin löydyttyä mahdollistavat sen muokaamisen. jQuery mahdollistaa kaikki samat toiminnallisuudet kuin puhdas Javascript: DOM-elementtien muokkaus, CSS-tyylien muuttaminen, Ajax ja käyttäjäinteraktioon reagointi voidaan kaikki toteuttaa jQueryn Javascriptiä yksinkertaisemman hakupohjaisen syntaksin avulla. Lisäksi jQuery tarjoaa yksinkertaistavia toimintoja elementtien animointia ja muita yleisiä Javascript-toimintoja, kuten iteroinnin toteuttamista, varten. JQuery pyrkii mahdollistamaan yhdenmukaisen toiminnallisuuden selaimesta riippumatta: se abstrahoi tehokkaasti vaadittavan selainriippuvaisen ohjelmakoodin ja tarjoaa sovelluskehittäjälle rajapinnan niitä vaativien toiminnallisuuksien käyttöön. Monimutkaisempia tai erikoislaatuisempia käyttötapauksia mahdollistavat toiminnot eristetään itse kehyksestä erillisiin lisäosiin, mikä pitää ydinkokonaisuuden mahdollisimman pienenä ja kevyenä. [Flanagan 2011:523–525; Chaffer & Swedberg 2013: 8–10.]

3.8 Datan visualisointi Javascriptillä

Modernit verkkoselaimet mahdollistavat näyttävien grafiikkaesitysten luomisen verkkosovelluksissa, mutta niiden toteuttaminen puhtaassa Javascriptissä voi olla työlästä. Visualisoinnin prosessia yksinkertaistavia Javascript-kirjastoja on kuitenkin kehitetty runsaasti. Datajournalismissa kirjastot, jotka mahdollistavat informaation visualisoinnin suoraan datalähteestä, ovat laajasti käytettyjä: niiden avulla voidaan luoda käyttöliittymiä suoraan data-aineistoon ja mahdollistaa aineiston interaktiivinen tarkastelu ja käsittely. Usein esiintyviä Javascriptillä toteutettuja visualisointeja ovat kartat, interaktiiviset graafit, aikajanat ja interaktiiviset kuvaesitykset.

Karttakirjastot tarjoavat Javascript-rajapinnan karttapalvelun luomiseen verkkosovelluksen käyttäjänäkymässä. Karttakirjastoja on useita, mutta kaksi suosittua avointa kirjastoa ovat avoimen karttapalvelu OpenStreetMap-yhteisön suosittelemat OpenLayers ja Leaflet.js [Switch2OSM: The Basics 2013]. Sekä OpenLayers että Leaflet perustuvat karttatasoihin: kartta esitetään tasona, jonka päälle voidaan lisätä toisia tasoja. Tasot voivat sisältää karttoja, osakarttoja tai kartalle piirrettäviä muita elementtejä, kuten merkkejä tai aluetta kuvaavia polygoneja. Yleensä perustasona käytetään jonkin karttapalvelun tarjoamaa pohjakarttaa. Kun kartan koko ja mitta-asteikko sivulla on määritelty, karttakirjasto lataa karttapalvelusta tarvittavan kokoisen kartan määrätynkokoisina neliönmuotoisina osina. Näitä kartan osia kutsutaan karttatiiliksi, ja ne ovat tyypillisesti joko 256 tai 512 pikseliä leveitä rasteri- tai vektorikuvia. Kirjastot pystyvät koordinaattien perusteella pyytämään palveluntarjoajalta halutun osan karttaa ja hallitsevat kartan kanssa vuorovaikuttamista, kuten skaalan muutosta tai näkymän siirtelyä. Kartalla esitettävä muu data tulee geokoodata, eli tiedolle tulee määrittää koordinaatit, jotta sen voi esittää kartalla halutussa kohdassa. [Hazzard 2011: 7–13; Maclean 2014; MacWright 2012.]

Graafien ja muun yleisen datan visualisoinnin mahdollistavia Javascript-kirjastoja on runsaasti. Datavisualisoinnin tietolähteeksi itseään kutsuva datavisualization.ch-sivusto listaa kirjoitushetkellä valikoimassaan yli kolmekymmentä hyväksi havaitsemaansa interaktiivisten graafien toteuttamisen mahdollistavaa sovellusta, joista valtaosa on Javascript-ohjelmistokirjastoja [Datavisualization.ch selected tools 2014]. Ohjelmistoresurssi GitHub listaa hakusanalla ”graphing”, ”visualization” tai ”charts” lähes 8 000 Javascript-ohjelmistoprojektia, joista moni on visualisoinnin mahdollistavia Javascript-

kirjastoja [GitHub 2014]. GitHubin listauksessa suosituin kirjasto on D3.js, joka esiintyy myös Datavisualization.ch:n valikoimassa.

D3, eli koko nimeltään Data-Driven Documents, on Mike Bostockin alun perin kehittämä ilmaisuvoimainen datan visualisointityökalu. D3 mahdollistaa datan lataamisen ja liittämisen verkkosovelluksen elementteihin ja näiden elementtien muotoilun ja muokkaamisen käyttäjäinteraktion perusteella. D3 ei varsinaisesti sisällä valmiita visualisointeja, vaan tarjoaa työkalut visualisointien luomiseen Javascriptillä, mutta vaatii hyvää ymmärrystä Javascript-ohjelmoinnista. D3 perustuu aiempaan Protovis-kirjastoon, joka tarjosi yksinkertaisemmat, mutta huomattavasti rajoittuneemmat työkalut visualisointien luomiseen. D3 käyttää grafiikan piirtämiseen SVG-vektorigrafiikkaa. Modernit verkkoselaimet tukevat tätä vektorigrafiikan muotoa hyvin, mutta Microsoftin Internet Explorerin versio 8 ja sitä aiemmat versiot eivät tue SVG-grafiikkaa ollenkaan, mikä käytännössä rajaa kirjaston toiminnallisuuden näitä uudemmille selaimille. [Murray 2013: 7–10]

Muita GitHubin listauksen mukaan erittäin suosittuja (yli 4 000 projektia seuraavaa jäsentä) kirjastoja ovat muun muassa Chart.js, rickshaw, morris.js ja sigma.js. Näistä Chart.js perustuu HTML5-standardin uuteen canvas-elementtiin. Canvas mahdollistaa kaksiulotteisen grafiikan renderöinnin ja muokkaamisen selaimessa, ja se on hyvin tuettu Internet Explorer 8:aa uudemmissa selaimissa, mutta ei toimi Microsoftin vanhoissa selaimissa ilman tuen mahdollistavia lisäkirjastoja [Downie 2014].

Rickshaw on Shutterstock-valokuvapalvelun kehittämä D3-kirjaston pohjalta rakennettu kaaviotyökalu. Se rakentaa D3-kirjaston toiminnallisuuden päälle valmiita malleja ja toiminnallisuuksia erimuotoisten graafien ja kaavioiden rakentamiselle ja käyttää samaa SVG-teknologiaa myös vanhempien selainten tuen rajoitteiden kannalta [Rickshaw 2014].

Morris.js on Raphaël-nimisen vektorigrafiikkakirjaston pohjalle rakennettu yksinkertainen kaaviotyökalu, joka tarjoaa valmiita malleja yleisimpien kaaviotyyppien luomiseen. Morris.js tukee pohjakirjastonsa myötä SVG-formaatin lisäksi Microsoftin VML-vektoriformaattia: tämä takaa tuen myös Microsoftin selaimille Internet Explorerin versiosta 6 lähtien [Smith 2014].

Sigma.js on toiminnallisuudeltaan erikoistuneempi kirjasto verkostografiikan luomiseen. Sigma käyttää grafiikan piirtämiseen canvas-elementtiä, mikä rajoittaa selaintuen Internet Explorer 8:aa uudempiin selaimiin ilman lisäkirjastoja [Jacomy ym. 2014].

Esitellyt kirjastot tarjoavat valmiita työkaluja datan, eli myös datajournalistisen sisällön, esittämiseen. Käyttämällä visualisointikirjastoa voidaan datajournalistisessa sovelluksessa käyttää hyväksi todettuja datan esitystapoja ja nopeuttaa sovelluksen kehitystä välttämällä ajan käyttämistä esitystavan suunnitteluun ja toteuttamiseen. Projektin koostuessa esitettävästä datasta nousee datan visualisointi olennaiseksi osaksi asiakassovelluksen toteutusta: jotta numeerisesta datasta saisi enemmän irti, se voidaan piirtää kartaksi ja kuvioiksi, jotka kertovat numeerisesta datasta ilmenevän viestin, tai tarinan, jolloin visualisointi voidaan nähdä myös journalistisena sisältönä datajournalistisen prosessin lopputuotteena.

3.9 Verkkosovelluskehitys osana uutistoimitusta

Luvussa 3 esitellyt verkkoteknologiat ja tekniikat mahdollistavat entistä paremmin verkkosovelluskehityksen tuomisen osaksi jokapäiväistä uutistoimitustyötä. Ympäristössä, jossa sovelluksen määräaika ajatuksesta julkaisuun usein lasketaan tunneissa tai päivissä viikkojen sijaan, on jokainen sovelluskehitystä nopeuttava tai tehostava keino kallisarvoinen.

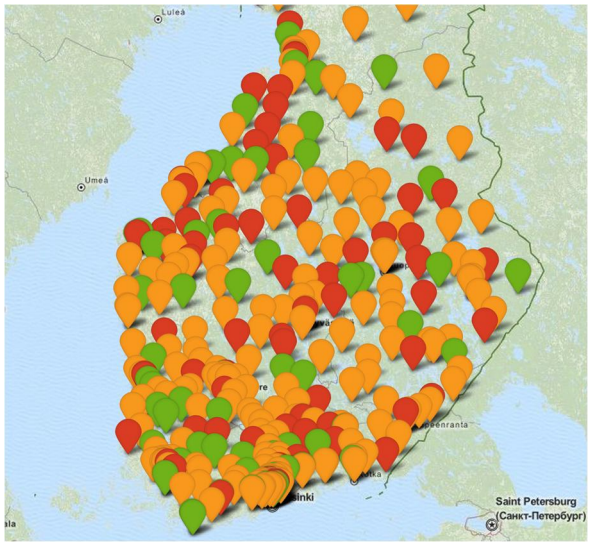
Verrattaessa näitä esiteltyjä verkkosovellustekniikoita ja teknologioita tyypillisiin niin sanottuihin perinteisiin vastaaviin, voidaan väittää, että modernit teknologiat antavat oikein toteutettuina mahdollisuuden näyttävämpien sovellusten ja visualisointien luomisen lyhyemmässä ajassa kuin perinteiset tavat ja teknologiat. Node.js-palvelinympäristö ja NoSQL-tietokannat mahdollistavat perinteiseen LAMP-kokonaisuuteen verrattuna nopeamman sovellusrungon toteuttamisen, ja ne vaativat käytännössä vain yhden ohjelmointikielen hallitsemista sovelluksen toteutukseen ja sen myötä pienentävät mahdollisesti vaadittujen henkilöresurssien määrää. Toteuttamalla toimivia REST-rajapintoja aineistoon, pystytään varmistamaan kootun data-aineiston helppo uudelleenkäytettävyys tulevissa sovelluksissa tai yhdistäminen toisiin koottuihin aineistoihin sovelluksessa. SPA-arkkitehtuurimalli yhdessä Ajax-teknologioiden ja jQueryn kanssa mahdollistaa loppukäyttäjälle, eli lukijalle, paremman käyttökokemuksen: Sovellus toimii pöytäkonesovellusmaisesti, eli interaktio on sovelluksen kanssa

sulavaa ja toimii ilman sivuston uudelleenlatauksia. Esitettävä grafiikka voidaan visualisointikirjastojen avulla toteuttaa niin ikään interaktiivisena ilman erillisiä selainlisäosia ja valmiiden, kertaalleen toteutettujen pohjien myötä myös nopeasti. SPA- ja MV*-arkkitehtuurimallit auttavat toteutettujen sovellusten ylläpidossa ja kehityksessä: usein toteutettu sovellus ei ole tarkoitettu kertakäyttöiseksi, vaan jää jatkokehitykseen. Selkeä modulaarinen rakenne ja sovellusarkkitehtuuri helpottaa jatkokehitystä pitämällä sovelluksen toiminnalliset osat erillisinä, jolloin yksittäisiä osia voidaan vaivattomammin muuttaa tai käyttää uudelleen.

4 Lukiovertailusovelluksen toteutus

4.1 Sovelluksen lähtökohdat ja suunnittelu

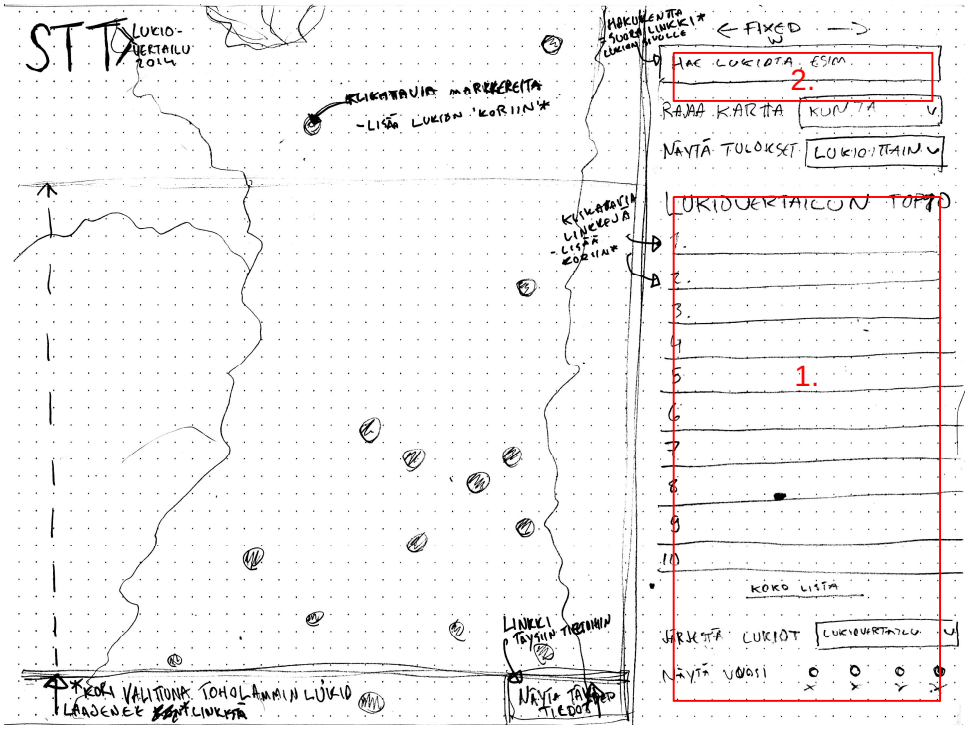
Insinöörityöprojektin tavoitteena oli suunnitella verkkosovellus lukiovertailuaineiston esittämiseen. Sovelluksella avulla voidaan tarkastella STT-Lehtikuvan toteuttaman lukiovertailun tuloksia ja vertailussa huomioon otettuja taustamuuttujia yksittäisten lukioiden tasolla. Asiakas oli aiempien lukiovertailutulosten esittämiseen toteuttanut yksinkertaisia karttavisualisointeja, kuten kuvassa 9 esitetty syksyn 2013 toteutus. Uuden sovelluksen käyttöliittymän perustoiminnallisuus sovittiin toteutettavan edellisiä toteutuksia mukaillen karttapohjaisena, kuitenkin lisäten ominaisuuksia ja parantaen karttanäkymää.



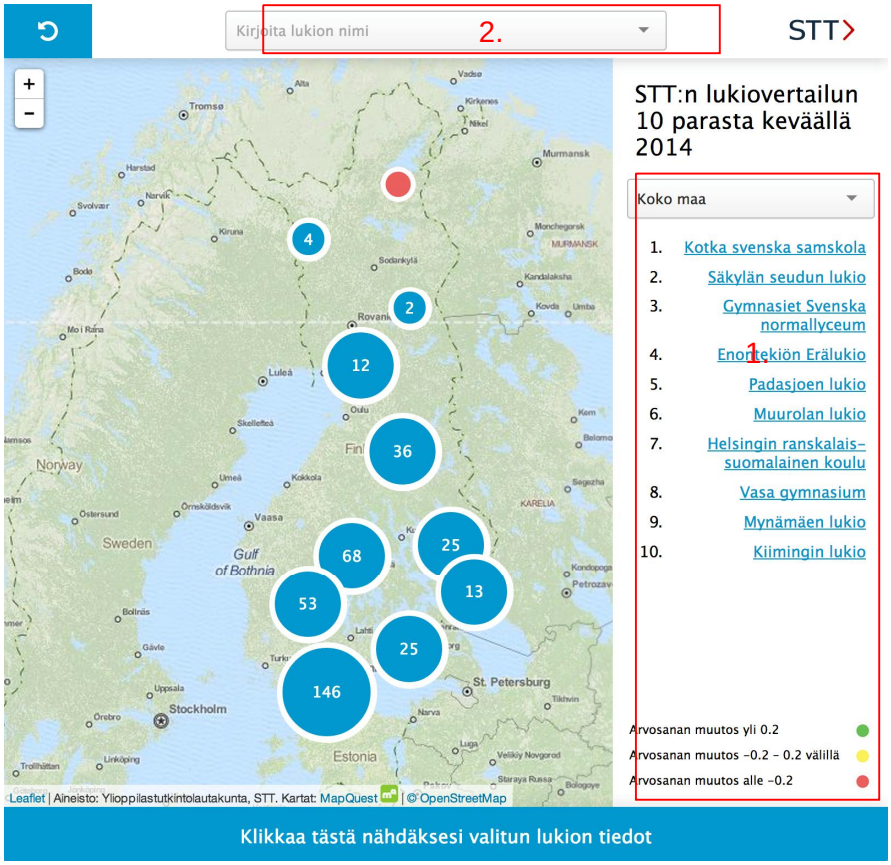
Kuva 9. STT-Lehtikuvan aiempi lukioiden tulossovellus.

Sovellus koostuu kolmesta osasta: palvelinsovelluksesta, tietokannasta ja asiakassovelluksesta. Asiakassovellus, eli varsinainen käyttöliittymä, koostuu kahdesta karttanäkymästä: karttanäkymästä ja yksityiskohtanäkymästä. Näiden kahden näkymän rautalankamallit ja lopulliset versiot esitellään kuvissa 10, 11, 12 ja 13 ja toiminnallisuus tarkemmin luvussa 4.1. Rautalankamallien ja lopullisten näkymien välillä voidaan nähdä eroavaisuudet projektin toteutuksen ensimmäisen vaiheen ja viimeisen vaiheen välillä: lopulta erot rautalankamallien ja lopullisten näkymien välillä kuitenkin jäivät pieniksi. Sovellus on kirjoitushetkellä julkaistuna osoitteessa lukiovertailu.herokuapp.com.

Kuvissa 10 ja 11 esitetään sovelluksen karttanäkymän suunnitteluvaiheen rautalankamalli ja lopullisen sovelluksen karttanäkymä. Korostettuna kuvissa näkyy kaksi navigaatioelementtiä, hakukenttä ja kärkilista- ja aineiston suodatuskenttä, hallitsevana elementtinä kuitenkin vasemmalla näkyvä kartta. Muutokset rautalankamallista lopulliseen versioon eivät ole suuria: muutama suodatinvaihtoehto jätettiin lopullisessa versiossa toteutuksesta.

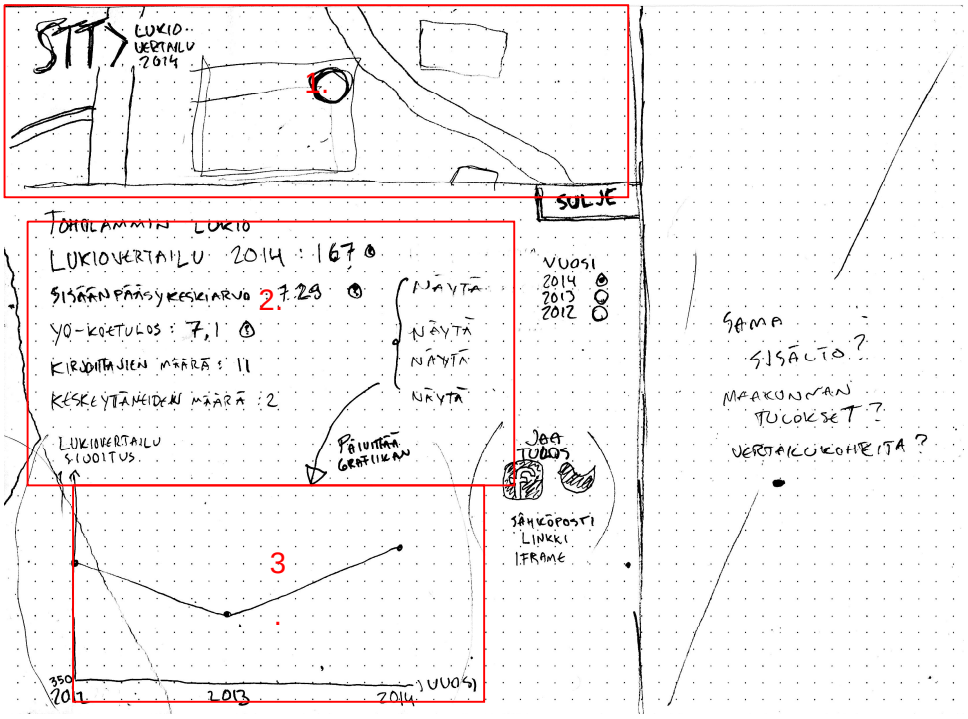


Kuva 10. Karttanäkymän rautalankamalli: 1. kärkilista ja aineiston suodatus, 2. hakukenttä.

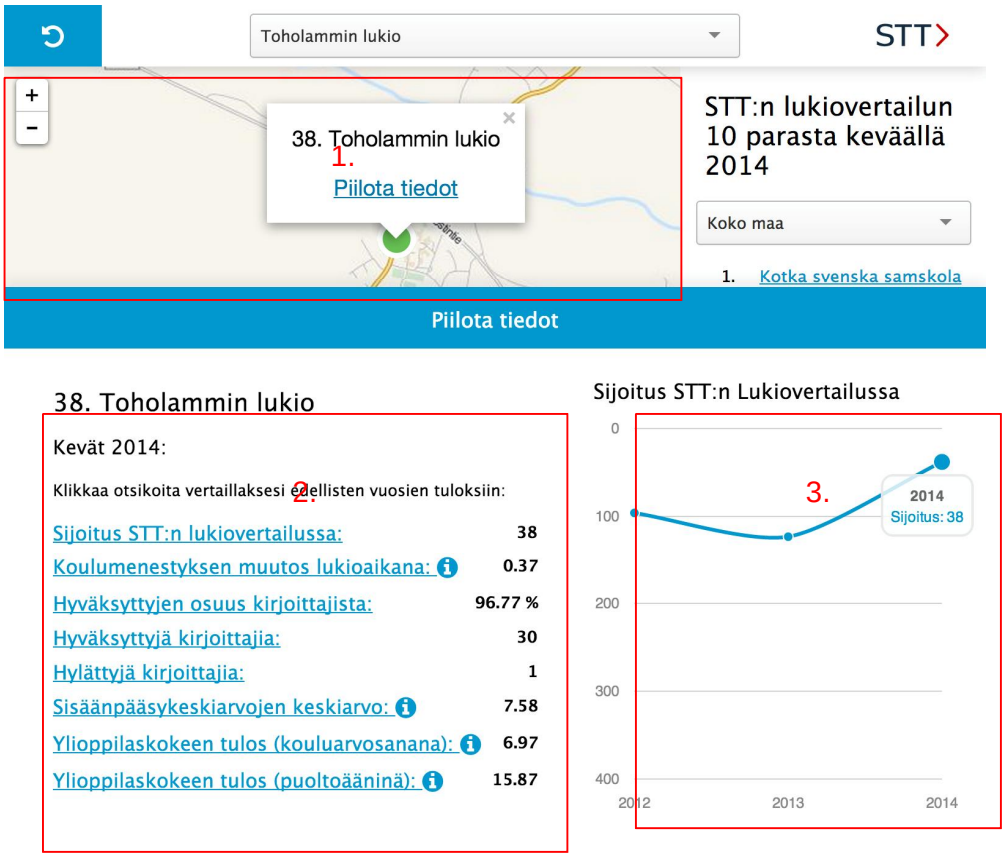


Kuva 11. Karttanäkymä: 1. kärkilista, aineiston suodatus ja kartan selite, 2. hakukenttä.

Kuvissa 12 ja 13 esitetään yksityiskohtanäkymän rautalankamalli ja lopullinen näkymä. Korostettuna kuvassa esitetään yksityiskohtanäkymän pääelementit: tarkennettu karttanäkymä, esitettävän aineiston listanäkymä ja aineiston visualisointi. Suurimmat muutokset rautalankamallin ja lopullisen version välillä liittyvät elementtien sijoitteluun: kaavailtu oikea elementtipalkki jätettiin pois tilan antamiseksi visualisoinnille. Asiakassovellusta, sen osia ja interaktiota esitetään tarkemmin luvussa 4.2.



Kuva 12. Yksityiskohtanäkymän rautalankamalli: 1. tarkennettu kartta, 2. esitettävä aineisto, 3. graafi aineistosta.



Kuva 13. Yksityiskohtanäkymä: 1. tarkennettu kartta, 2. esitettävä aineisto, 3. graafi aineistosta.

Karttanäkymän avulla voidaan selata lukioita sijainnin perusteella, ja valitsemalla lukio voidaan nähdä kyseisen lukion tulos vertailussa. Yksittäisen lukion tuloksia voidaan verrata edellisten keväiden vastaaviin tuloksiin yksityiskohtanäkymässä: sovellus esittää kehityksen graafina. Sovellus mahdollistaa myös lukioiden tekstipohjaisen hakemisen, mikä antaa käyttäjille oikotien suoraan heitä kiinnostaviin lukioihin. Sovellus voidaan kohdentaa julkaistaessa joko maakunta- tai lukiotasolla: asiakas voi julkaistessaan määritellä alkunäkymään esitettäväksi tietyn koordinaatein määritellyn pisteen tai lukion. Eri maakunnissa pääosin toimivat STT-Lehtikuvan asiakkaat voivat näin kohdentaa sovelluksen omia lukijoitaan kiinnostavalle alueelle. Näkymä antaa myös käyttäjälle mahdollisuuden vaihtaa näkymän kohdennusta maakuntavalikon avulla. Kohdennus vaikuttaa myös esitettävään sisältöön: sovellus esittää kärkilistan parhaiten vertailussa pärjänneistä lukioista, mikä toimii maakuntakohtaisesti valittaessa esitettäväksi tietty maakunta. Kärkilistauksessa pyrittiin olemaan korostamatta tarkkoja lukuarvoeroja lukioiden välillä, sillä ne ovat varsin pieniä: kärkilistan lukioiden arvosanojen erot ovat vain arvosanan kymmenyksiä tai sadasosia ja koko vertailuaineisto mahtuu noin yhden arvosanan muutoksen sisälle. Heikoiten pärjänneitä lukioita ei haluttu erityisesti korostaa, joten niitä esittävää listavaihtoehtoa ei toteutettu. Kahden tai useamman lukion välisen vertailukorin toteuttaminen jätettiin myös tämän insinöörityöprojektin ulkopuolelle, mutta se olisi mahdollista toteuttaa tulevaisuudessa.

Lukiovertailuaineisto koostuu paikannetusta, eli geokoodatusta, lukiokohtaisesta tilastodatasta, joka on koostettu ja käsitelty taulukkolaskentaohjelmassa ja koottu käsittelyn jälkeen JSON-muotoon. Käsitelty tilastodata sisältää jokaisen Suomen lukion sijainnin, nimen, lukiovertailun tuloksen, sijoituksen tehdyssä vertailussa, hyväksyttyjen ja hylättyjen ylioppilaskoekirjoittajien määrän sekä lukiovertailun pohja-aineistona käytetyn datan sisäänpääsy- ja yo-koekeskiarvosta vuosilta 2012, 2013 ja 2014. Toteutettavan sovelluksen suunnittelun näkökulmasta data toimitetaan valmiiksi käsiteltynä JSON-aineistona, eikä aineistoa tarvitse jatkokäsitellä sovelluksessa.

Karttanäkymässä pyrittiin ensin esittämään kartalla kaikki pisteet, eli lukiot, yhdessä näkymässä, mikä antaa käyttäjälle kokonaiskuvan aineistosta kuten aiemmissa toteutuksissa. Lopullisessa toteutuksessa nämä pisteet päädyttiin yhdistämään pisteryppäiksi, joita klikkaamalla voi tarkentaa karttaa, jolloin yksittäisen pisteet esitetään. Karttanäkymän oheen toteutettiin muita tietoa avaavia toiminnallisuuksia, kuten kärkilista parhaista tuloksista. Lukiot esitetään kartalla geokoodauksen perusteella kuvakkeina vasemmanpuoleisessa näkymässä. Oikean reunan näkymään kootaan sovelluksen

muuta toiminnallisuutta: datan rajausvaihtoehdot, kärkilista ja karttanäkymän selite (kuvat 10 ja 11, 1). Hakukentälle saatiin enemmän tilaa siirtämällä se suunnitellulta paikaltaan oikeassa reunassa (kuvat 10 ja 11, 2) keskelle näkymää.

Käyttäjä voi valita lukion kartalta klikkaamalla, minkä jälkeen esitetään yksityiskohtainen näkymä lukion tiedoista. Lukiokohtaiset esitettävät tiedot ovat

- lukion nimi
- sijoitus vertailussa
- laskettu tulos vertailussa
- ylioppilaskokeen kirjoittajien määrä
- ylioppilaskokeessa hyväksyttyjen määrä
- ylioppilaskokeessa hylättyjen määrä
- lukioonpääsykeskiarvon keskiarvo
- neljän pakollisen aineen yo-koetulos puoltoääninä
- neljän pakollisen aineen yo-koetulos kouluarvosanana
- lukion sijainti karttakoordinaatteina
- lukion sijaintikunta
- lukion sijaintimaakunta.

Lukiokohtaista tietoa on koottu esitettäväksi vuosilta 2012–2014, jotta vuosien välistä kehitystä pystytään esittämään. Yksityiskohtanäkymään näkyviin jäävä kartta (kuvat 12 ja 13, 1) tarkennetaan valitun lukion kohdalle, ja avautuvaan näkymään ladataan rajapinnasta valitun lukion esitettävä data. Näkymässä esitetään oletusarvoisesti viimeisimmän vertailun kohdalta ajankohtaiset numeeriset arvot kaikista dataan tallennetuista aiemmin esitetyistä tietueista (kuvat 12 ja 13, 2). Näkymään piirretään myös graafi, joka kuvaa valitun arvon kehitystä data-aineiston sisältämän kolmen vuoden ajalta (kuvat 12 ja 13, 3). Käyttöliittymän suunnittelussa pyritään Cairon esittämän visualisoinnin arviointimallin mukaiseen balanssiin, sillä käyttöliittymä ja sen toiminta on olennainen osa visualisoinnin kokonaisuutta.

Toiminnallisuudeltaan sovelluksen tuli olla mahdollisimman yksinkertainen, mutta sen tuli kuitenkin tarjota mahdollisuus taustadatan merkitykselliseen ja monipuoliseen tar-

kasteluun. Jakelu toteutettiin upotuksena asiakkaiden verkkopalveluihin, joten käyttöliittymän tuli olla tarpeeksi yksinkertainen ja toiminnallisuudeltaan huomiota herättämätön, niin että se toimii monenkaltaisella alustalla. Niin pitkälti kuin mahdollista, tuli varmistaa sovelluksen toiminta myös vanhemmilla selaimilla ja vähemmän tehokkailla alustoilla: mobiilitoiminnallisuus ei kuitenkaan ollut prioriteetti tämän projektin toteutuksen osalta.

Itse sovellusrakenne suunniteltiin niin, että saman rakenteen uudelleenkäyttö muissa projekteissa on mahdollista. Luvussa 3 esitellyt teknologiat mahdollistavat modulaarisen ja skaalautuvan järjestelmäarkkitehtuurin: yksittäisiä osia voidaan kehittää toisista riippumattomina. Tietokantajärjestelmä, palvelinohjelmisto ja asiakassovellus toteutettiin toistaan riippumattomina, mikä mahdollistaa minkä tahansa yksittäisen osan vaihtamisen jälkeenpäin rikkomatta sovelluksen toiminnallisuutta niin kauan kuin rajapintamäärittelyt pysyvät yhdenmukaisina. Etenkin asiakassovelluksen toteutuksen kannalta toteutettu REST-rajapintatoteutus tarjoaa vapautta: asiakassovellus ei ole riippuvainen palvelinsovelluksen toteutuksesta, vaan ainoastaan rajapinnan tarjoamista toiminnallisuuksista. Saman rajapinnan päälle voisi samasta data-aineistosta rakentaa aivan erilaisen käyttöliittymän.

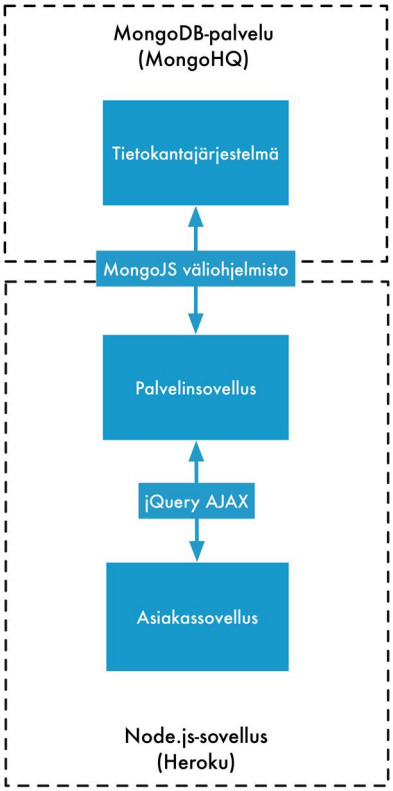
Sovelluksen tekninen toteutus ja rakenne suunniteltiin luvussa 3 esiteltyjen tekniikoiden ja teknologioiden pohjalta korostaen etenkin sovelluskehityksen soveltuvuutta uutistoimitusympäristöön: käytännössä tämä näkyy voimakkaimmin sovelluskehityksen nopeuden painottamisena.

SPA-malli, eli yhden sivun sovellusrakennemalli, mahdollisti pöytäkonesovellusmaisen toiminnallisuuden toteuttamisen verkkoselaimessa. Malli antoi myös selkeän rakenteen sovellukselle, mikä nopeutti sovelluskehitystä. Javascript-pohjainen SPA-sovellus koostuu tietokannasta, palvelinpohjaisesta taustajärjestelmästä ja toiminnallisuuden sisältävästä käyttäjänäkymästä tai asiakassovelluksesta. Sovelluksen esittämä data tallennettiin MongoDB-tietokantapalveluun, joka toimii erillisenä muusta sovelluksesta ylläpitäen sovelluksen modulaarisuuden myös tiedon tallennuksen osalta: tietokantapalvelun vaihtaminen toiseen on tulevaisuudessa yksinkertaista. MongoDB:n dokumenttiorientoituneisuus mahdollisti data-aineiston viemisen tietokantaan lähes muokkauksetta, ja skeemattomuus antoi vapauden muokata tietorakennetta tarpeen mukaan projektin edetessä.

Asiakassovelluksen ja palvelinsovelluksen välillä liikkuu ainoastaan JSON-muotoinen data, eli palvelinsovellus ei tuota esimerkiksi HTML-koodia asiakassovellukselle, mikä vähentää tiedonsiirron määrää ja nopeuttaa loppukäyttäjien latausaikoja. Asiakassovelluksen ja palvelinsovelluksen kehitystä voidaan erottelun myötä jatkaa tulevaisuudessa toisistaan itsenäisinä. Asiakassovellus ja palvelinsovellus sijaitsevat samassa virtuaalikoneyksikössä, mutta ne keskustelevat keskenään vain Ajax-teknologioilla siirretyn datan välitse RESTFUL-rajapinnan avulla. Toteuttamalla dataan rajapinta voidaan samaa aineistoa yksinkertaisesti hyödyntää myös muissa sovelluksissa: voidaan rakentaa esimerkiksi niin sanottuja mashup-sovelluksia yhdistellen aineistoja eri rajapinnoista ja luoda uusia journalistisia sisältöjä samasta aineistosta.

Sovelluksen tietokantajärjestelmä, palvelinsovellus ja asiakassovellus toimivat toisistaan erillisinä osina, jotka vain välittävät dataa: yksittäinen osa voidaan kokonaan vaihtaa muuttamatta muita, niin kauan kuin siirrettävä data pysyy samanlaisena. Koska sovelluksen osat ovat toisistaan erillisiä, niitä voidaan paremmin jatkossa hyödyntää muissa projekteissa, mikä säästää aikaa niiden pohjalle rakennettavien uusien sovellusten kehityksessä. Käyttöliittymää suunniteltaessa harkittiin MV*-arkkitehtuurin täysimuotoista hyödyntämistä, mutta tämä todettiin sovelluksen kehityksen alussa ylimitoitetuksi: Kun malli on käytännössä staattinen, toteuttavat staattiset resurssit sekä mallin että näkymän roolin. Koska käyttöliittymän rakenne on yksinkertainen, olisi mallin ja näkymän erottamisessa erillisiksi toiminnallisuuksiksi paljon töitä pientä hyötyä varten ja se jätettiin, projektin aikataulu huomioon ottaen, pois lopputoteutuksesta.

Sovelluksen rakenne toteutettiin kuvan 14 mukaisesti: tietokantajärjestelmä, palvelinsovellus ja asiakassovellus toimivat erillisinä osina, joiden välillä liikkuu vain data tietokantaväliohjelmiston ja Ajax-teknologioiden avulla.



Kuva 14. Lukiovertailusovelluksen rakennekaavio.

4.2 Asiakassovellus

Asiakassovellus on lukijan käyttöliittymä esitettävään aineistoon. Se vastaa tiedon esittämisestä ja sen visualisoimisesta ja pyrkii havainnollistamaan datan informaatiosisältöä.

Mukaillen esiteltyjä Cairon ja Tuften periaatteita visualisoinnin tulee toiminnallisuudellaan antaa mahdollisimman tarkka näkemys taustadatan informaatiosisältöön vääristelemättä sitä ja antaen tilaa lukijoiden omille tulkinnoille ja johtopäätöksille asiasisällöstä. Varsinaisen numeerisen tulosinformaation visualisoinnissa suunniteltiin kokonaisuus Tuften esittämän minimalismin perusteella: minimoimalla koristeellisuus vältetään virheitä tiedon esityksessä. Käyttöliittymäkokonaisuuden suunnittelun tukena käytettiin kuvassa 2 esiteltyä Alberto Cairon ympyrämallia visualisoinnin laadun arvioimiseen. Sovelluskehityksen aikana itsearvioimalla toteutusta mallin avulla pyrittiin pitämään kokonaisuus balanssissa, päästämättä sitä liian lähelle kumpaakaan kuvassa 4 esitettyä ääritapausta. Käyttöliittymän muotoilussa pyrittiin mallin avulla välttämään liikaa

teknisyyttä ja monimutkaisuutta samalla kuitenkaan yksinkertaistamatta esitystä liikaa ja luomaan helpommin lähestyttävä kokonaisuus kuin puhtaasti käytännöllinen toteutus vastapainona numeerisen datan minimalistiselle esitystavalle.

Esitettävä lukiokohtainen data on maantieteellisesti määriteltyä (geokoodattua), vuosittain eroteltua ja pääosin numeerista aineistoa, jota siirretään taustajärjestelmästä JSON-muodossa. Yhden vuoden tietue sisältää seuraavat arvot:

- result, tulos
- testTakers, ylioppilaskokeen kirjoittajien määrä lukiossa
- entryGradeAvg, lukioonpääsykeskiarvon keskiarvo
- yoGradeConversion, ylioppilaskoetulos kouluarvosanana
- unconvertedGrade, ylioppilaskoetulos puoltoääninä
- rank, sijoitus
- passedTests, hyväksyttyjen kirjoittajien määrä
- failedTests, hylättyjen kirjoittajien määrä.

Jokainen vuosittainen tietue kuuluu yhteen lukiotietueeseen, jossa on tallennettuna tiedot lukiosta ja sen sijainnista:

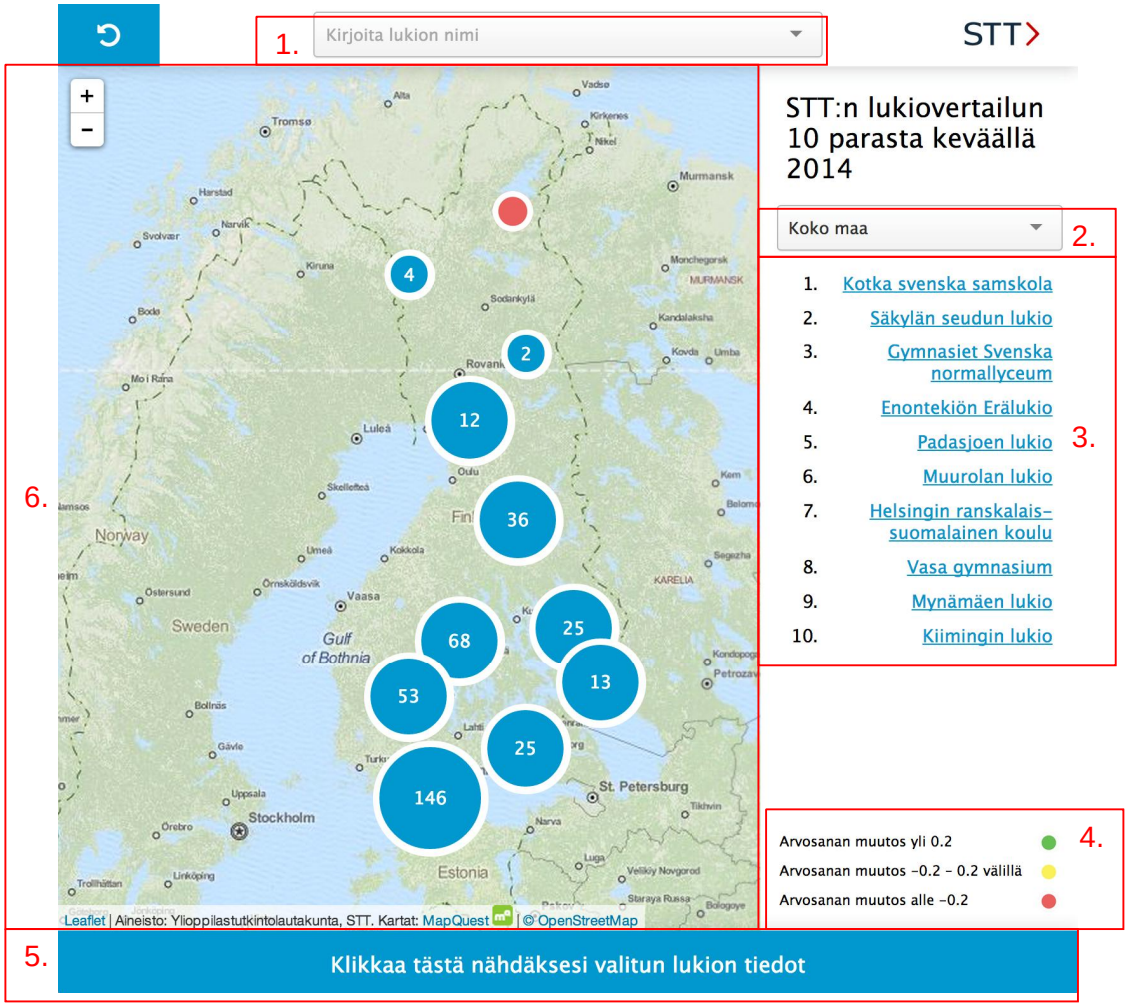
- _id eli tunnistenumero
- lat eli lukion sijainnin latitudiarvo
- lon eli lukion sijainnin longitudiarvo
- municipality eli lukion kunta
- region eli lukion maakunta
- schoolName eli lukion nimi.

Geokoodatulle informaatiolle sopii luontevasti karttaan pohjautuva esitystapa ja pienehkön numeerisen data-aineiston tarkkojen arvojen esittämiseen taulukko [Tufte 2001: 178]. Esitettävää numeerisessa datassa tapahtuvaa vuosittaista vaihtelua on kuitenkin helpompi havainnollistaa esimerkiksi viivadiagrammista kuin taulukosta: antamalla numeroarvoille muoto helpottuu niiden merkityksen hahmottaminen [Cairo

2013: 37]. Kokonaisuuden tulee siis pystyä esittämään datan kolme pääpiirrettä: maantieteellinen sijainti, tarkat arvot sekä arvojen kehitys.

Datan visualisointiin valittiin Javascript-kirjastot Leaflet ja Morris.js. Valinta perustui kummankin kirjaston kohdalla mataliin kustannuksiin, laajaan käyttäjäyhteisöön, hyvään selaintukeen ja jossain määrin ohjelmistojen keveyteen. Molemmat kirjastot ovat avoimia ohjelmistoja, joilla on aktiivisesti lisäominaisuuksia ja tukea tarjoava GitHub-yhteisö. Kumpikin tukevat selaimia aina Internet Explorerin versioon 8 asti, mikä vastaa projektille asetettua hyvän selaintuen vaatimusta. Lisäksi kumpikin kirjasto on kooltaan hyvin pieniä, mikä tekee niiden lataamisesta sivuilla kevyttä. Leafletin pohjakarttana käytetään karttapalveluntarjoaja MapQuestin avointa, Open Street Map -dataan perustuvaa MapQuest-OSM Tiles-palvelua [MapQuest Developers 2014].

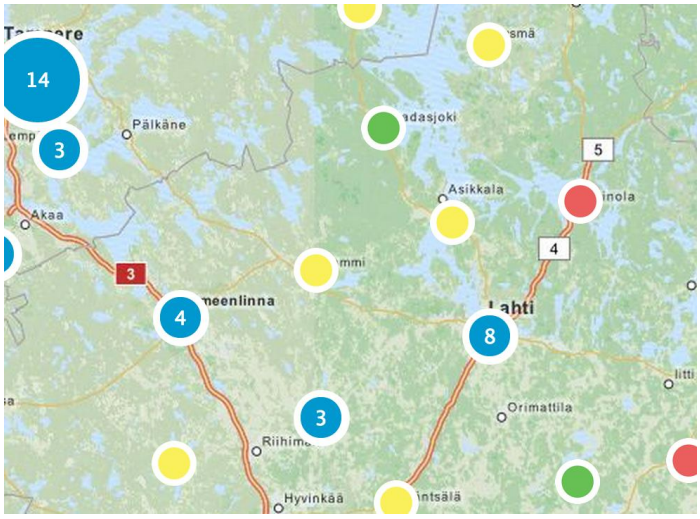
Asiakassovelluksen toiminnallisuus toteutettiin Javascriptilla. Sivuston HTML-merkinnässä määritellään vain staattisia elementtejä: HTML:n perusrakenteen lisäksi näitä ovat projektisovelluksessa valikoiden ja taulukoiden otsikot ja lisätietoikkunoiden sisällöt. Kaikki ulkoasumääritelmät tehdään CSS-merkinnässä. Sovelluksen rakentaminen käydään tässä raportissa läpi perusperiaatteiltaan. Varsinainen asiakassovelluksen kommentoitu lähdekoodi on luettavissa liitteissä 1 ja 2, ja kuvissa 15 ja 19 kuvataan lopullisen asiakassovelluksen pää- ja yksityiskohtanäkymää.



Kuva 15. Lopullinen karttanäkymä: 1. hakukenttä, 2. maakuntavalitsin, 3. kärkilista: linkit avaavat esitettyyn lukioon kohdistetun näkymän, 4. kartan selite, 5. yksityiskohtanäkymän avaava painike, 6. kartta, jossa lukiot on esitetty merkkiryppäinä.

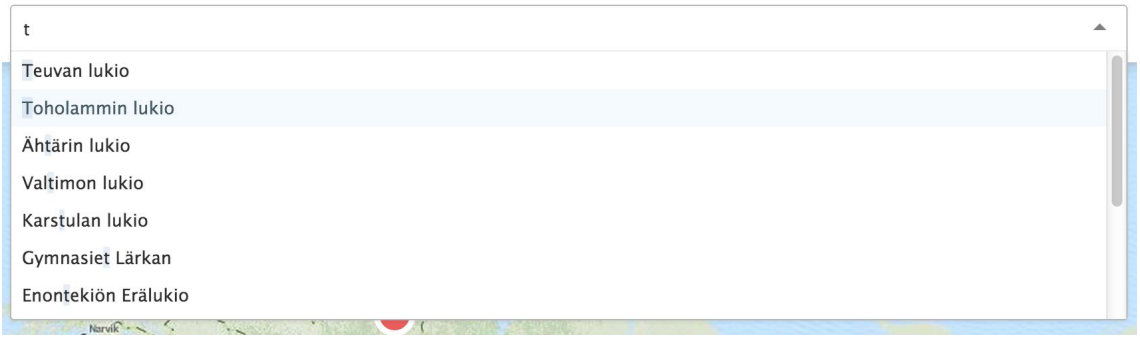
Karttanäkymän suurin visuaalinen elementti on vasemman puolen kartta. Sijaintinsa perusteella lukiot esitetään kartalla pyöreinä, jotka on alkunäkymässä yhdistetty merkkiryppäiksi, joita klikkaamalla saa esiin yksittäiset merkit. Koska jokaisen lukion tietue on geokoodattu, pystyy valittu Leaflet-karttakirjasto piirtämään lukioiden kuvakkeet tasona suoraan karttatiilien päälle oikeille maantieteellisille paikoilleen. Jokaiselle kartalle piirrettävälle kuvakkeelle määritellään vihreä, keltainen tai punainen väri lukiovertailussa saadun tuloksen perusteella: lukiot jaetaan erikseen määriteltyjen raja-arvojen perusteella parhaiten, keskiarvon mukaisesti ja heikoiten pärjänneisiin ryhmiin. Väriarvojen raja-arvot esitetään karttanäkymässä.

Koska datapisteitä on projektisovelluksen käyttämässä aineistossa lähes 400, ne täyttäisivät yksittäin koko kartan, jolloin siitä tulisi käytettävyydeltään kelvoton etenkin tiheämmin lukioita sisältäviltä alueilta. Tästä syystä päädyttiin käyttämään hyväksi Leaflet-lisäosaa MarkerCluster. MarkerCluster on lisäosa, joka muodostaa piirrettävistä merkeistä merkkiryppäitä. MarkerCluster-taso näyttää yksittäisten merkkien sijaan rypäsmerkkejä, joissa ilmoitetaan kyseisen rypäsmerkin alle sijoittuvien merkkien määrä. Klikattaessa rypäsmerkkiä tai siirrettäessä kartan tarkennustasoa lähemmäksi näytetään rypäsmerkkien sijaan yksittäisiä merkkejä sitä mukaa, kuin ne mahtuvat tarpeeksi kauas toisistaan näkymässä (kuva 16). Näin karttanäkymä (kuva 15) ei täyty merkeistä, mutta säilyttää kuitenkin viittauksen kaikkiin datapisteisiin.



Kuva 16. MarkerCluster-merkkiryppäiden toiminta. Siniset merkit ovat merkkiryppäitä, keltaiset, punaiset ja vihreät yksittäisiä lukioita värjättynä vertailun tuloksen mukaan.

Niin ikään tilansäästön vuoksi päätettiin toteuttaa lukijaa kiinnostavan lukion löytämistä helpottava hakutoiminto. Usein lukija tietää itseään kiinnostavimmat lukiot nimeltä, jolloin on luonnollista tarjota mahdollisuus hakea näitä lukioita suoraan, kartan tarkentamisen ja selaamisen sijaan. Hakutoiminto toteutettiin Selectize-jQuery-lisäosalla, joka mahdollistaa yhdistetyn hakukentän ja alasvetovalikon luomisen suoraan rajapinnasta saatavan datan perusteella [Reavis 2014]. Tämän valikon toimintaa esitetään kuvassa 17. Selectize hakee rajapinnan tarjoaman nimihaun perusteella listan lukioista, joilla se täydentää pudotusvalikkoa reaaliajassa käyttäjän syöttäessä hakuehdot hakukenttään, mikä antaa käyttäjälle käytännössä itsetäyttyvän hakukentän, josta voi poimia haluamansa hakuehdon jo ennen täyden hakusanan kirjoittamista, mikä nopeuttaa haun käyttöä.



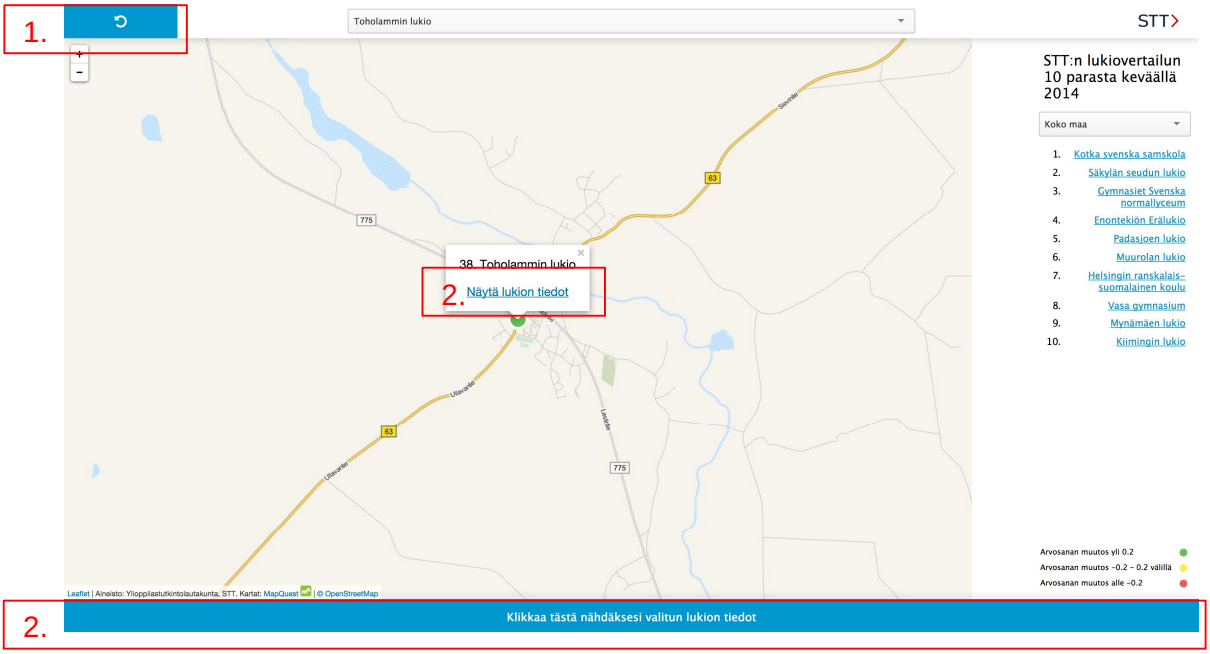
Kuva 17. Selectize-kirjastolla toteutettu hakukenttä.

Karttanäkymää on mahdollista tarkentaa tiettyyn maakuntaan alasvetovalikosta (kuva 18). Oletusarvoisesti kartalla esitetään koko maa. Maakunnan vaihtaminen valikosta muuttaa myös kärkilistan sisältämään vain kyseisen maakunnan lukioita. Asiakkaan on myös mahdollista karttaa käyttäessään sivullaan määritellä selainparametrinä kartan keskipiste, tarkennustaso tai tietty lukio. Tämä antaa maakuntalehtiasiakkaille mahdollisuuden kohdentaa visualisoinnin informaatiosisältöä omille lukijoilleen.

Kuva 18. Maakuntavalikko.

Kärkilista toimii yksinkertaisena listana linkkejä siinä esitettyjen lukioiden tarkempiin tietoihin. Käyttäjällä on siis kolme tapaa päästä kiinni yksittäisen, häntä kiinnostavan lukion tietoihin. Tämä antaa mahdollisuuden erilaisiin tapoihin käyttää sovellusta, mutta esitettävä informaatiosisältö pysyy samana käyttötavasta riippumatta. Tällä pyritään luomaan monipuolinen mutta helppokäyttöinen käyttöliittymä data-aineistoon luvussa 2.3 esitettyjen informaatiomuotoilun periaatteiden mukaisesti.

Käyttäjän valittua häntä kiinnostavan lukion näkymä kohdistuu kyseiseen lukioon kuvan 19 mukaisesti. Tämä näkymä toimii hyväksyntäikkunan tavoin: se esittää käyttäjälle valitun kohteen ja antaa mahdollisuuden vaihtaa sitä ennen seuraavaan näkymään siirtymistä. Näkymästä on kaksi linkkiä tarkempiin tietoihin, lukion kohdalla kartalla ja alareunan painikkeessa. Lukion kohdalla esitettävä linkki nähtiin tarpeelliseksi lisätä käyttäjätestauksen myötä: pelkkä alareunan painike ei yksin ollut tarpeeksi havainnollinen tehdäkseen selväksi sen toiminnallisuuden. Kahdella painikkeella on kuitenkin sama toiminnallisuus, ja niitä voi käyttää ristiin: kaksi painiketta antaa jälleen useamman polun samaan asiaan päätymiseksi, minkä avulla vähennetään mahdollisten erehdysten määrää.

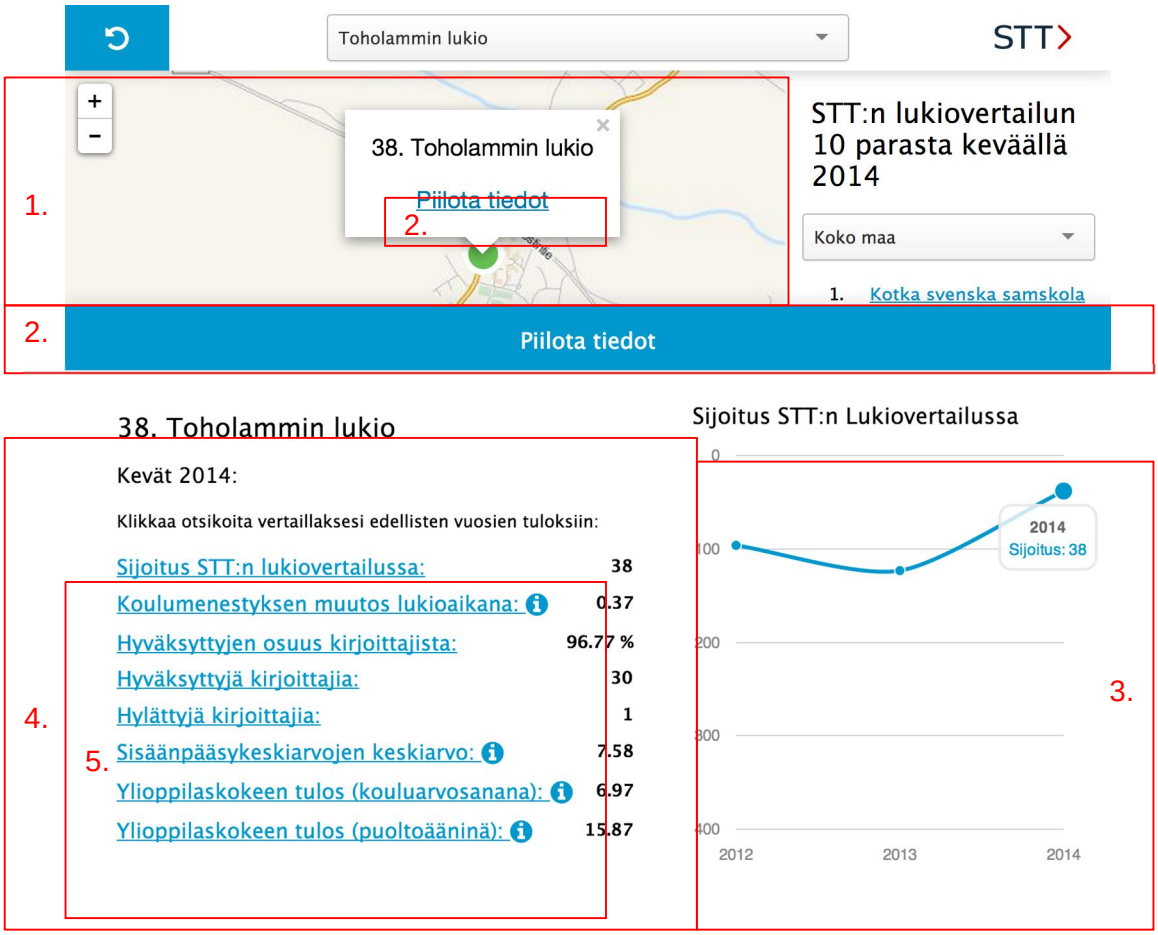


Kuva 19. Yksittäiseen lukioon tarkennettu näkymä. 1. painike perusnäkymän palauttamiseen, 2. painikkeet yksityisnäkymän avaamiseen.

Kummatkin painikkeet johtavat kuvan 20 mukaiseen yksityiskohtanäkymään, jossa avautuu palvelinsovelluksen rajapinnalta saatavat kyseisen lukion tarkemmat tiedot. Tässä näkymässä esitetään viimeisimmän lukiovertailun tulokset, numerot joista tulos on laskettu sekä taustatietoja lukiosta. Itse tulos esitetään otsikolla *Koulumenestyksen muutos lukioaikana.* Käyttäjälle yksiselitteisempi sijoitusluku on kuitenkin nostettu tämän yli arvojärjestyksessä ja esitetään listassa ensimmäisenä. Näkymän oikeaan reunaan piirretään valitusta datasta sen kehitystä kuvaava graafi. Graafissa esitetään listasta valitun arvon vastaavat arvot myös vuosilta 2012 ja 2013, ja niiden perusteella

voidaan nähdä arvon kehitys, jonka perusteella käyttäjä voi arvioida lukion tulosta myös edellisvuosien kontekstissa.

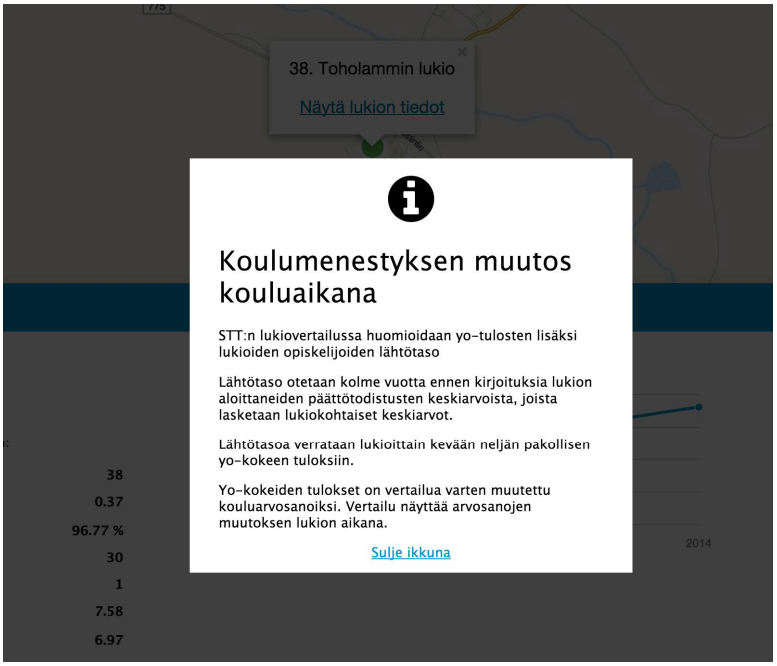
Yksityiskohtanäkymä on informaatioltaan rikkain näkymä sovelluksessa, ja tiedon muotoilussa on pidetty kiinni tuftemaisista periaatteista: pitäytymällä visuaalisesti yksinkertaisessa esitystavassa pidetään valekerroin mahdollisimman pienenä ja vältetään niin sanottua kuvioroinaa. Näkymässä ei ole kuvituskuvaa, vaan kaikki näkymän osat esittävät jotain informaatiota: kohdennettu kartta maantieteellistä sijaintia, graafi arvojen vuosittaista vaihtelua ja lista tarkkoja numeerisia arvoja. Data-mustesuhde ja informaatiotiheys pidetään näin korkeana. Lukija saa selkeän kuvan aineiston sisällöstä, eikä esitystapa johdata lukijaa tiettyyn tulkintaan, vaan pitäytyy mahdollisimman neutraalina. Tuften periaatteisiin nojaaminen soveltuu hyvin esitettävänkaltaisen, yksinkertaisen informaation esittämiseen. Mahdollisten monimutkaisempien korrelaatioiden osoittaminen tällä minimalistisella esitystavalla voisi muodostua lukijan kannalta vaikeaselkoiseksi, jolloin visuaalisesti rikkaampi esitystapa voisi olla tehokkaampi.



Kuva 20. Asiakassovelluksen yksityiskohtanäkymä: 1. valittuun lukioon tarkennettu karttanäkymä, 2. linkit näkymän sulkemiseen, 3. graafi esitettäväksi valitusta aineistosta, 4. vii-

meisimmän vertailun tulokset ja tiedot, 5. linkit graafin esittämän aineiston vaihtamiseen.

Koska osa esitettyjen arvojen merkityksistä saattaa olla vaikeasti tulkittavissa pelkästä arvolle annetusta otsikosta, päätettiin näille abstrakteille arvoille antaa erilliset esitystä tukevat selitteet. Lisäinformaatiota kuvastava i-kuvake otsikon vierellä toimii linkkinä lisätietoon kyseisestä arvosta. Lisätieto avautuu ikkunaksi yksityiskohtanäkymän päälle kuvan 21 mukaisesti, häivyttäen näkymän taustaltaan.



Kuva 21. Lisäinformaatioikkuna.

Yksityiskohtanäkymä myös mukautuu eri näyttöko'oille. Koko sovellus on responsiivinen, eli se mukautuu käytettävän näyttökoon mukaan sopivaksi, mutta esitettävän informaation runsauden vuoksi tämä näkyy selkeimmin yksityiskohtanäkymässä. Kuvassa 22 esitetään miten yksityiskohtanäkymä käyttäytyy kapeassa, niin kutsutussa potrettikoossa. Näkymä ylettyy sivun yläreunaan asti antaen graafille tilaa sivun alareunassa oikean palstan sijaan.

Kuva 22. Yksityiskohtanäkymä kapeassa selainikkunassa.

Visuaalisen kokonaisuuden arvioinnissa Cairon mallin avulla korostettiin mallin esittämiä vastapareja. Näitä vastapareja esiteltiin luvussa 2.3.

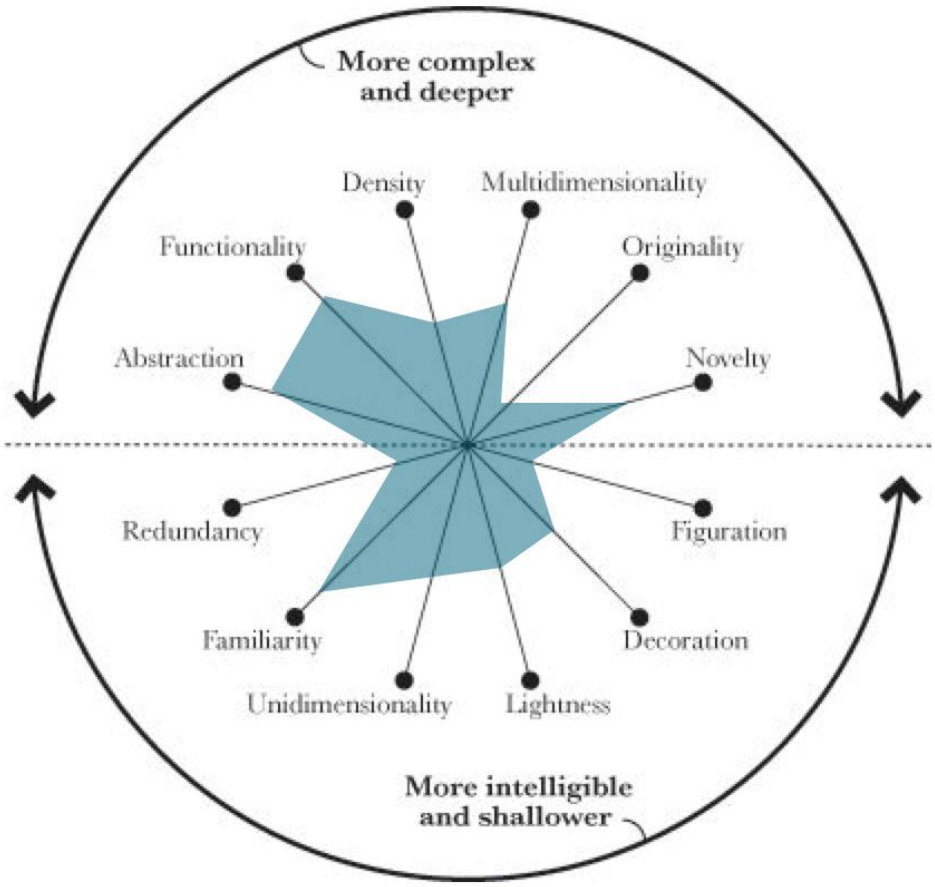
Abstraktio–esittävyys-akselilla visualisointi sijoittuu abstraktimpaan päähän: varsinaista aineistoa esittämätöntä grafiikkaa ei ole, ja voisi materiaalin numeerisuuden takia olla vaikeata tai harhaanjohtavaa lisätä sitä. Esimerkiksi lukioiden esittämistä kartalla lukiokuvakkeina kuitenkin harkittiin, muttei toteutettu aikataulun takia. Karttanäkymää voidaan sinänsä pitää esittävänä itsessään, sillä lukioiden sijainnit ja etäisyydet esitetään realistisesti ja tarkasti.

Funktionaalisuus–koristeellisuus-akselilla esitystä voidaan pitää voimakkaasti funktionaalisena, mutta karttanäkymää ainakin osittain koristeellisena. Vaikka kartta esittääkin aineistoa, sen realismi on selkeästi koristeellista datavisualisoinnin kannalta. Aineiston kannalta voidaan myös kyseenalaistaa kartan tarpeellisuus varsinaisen tuloksen esittämiseen: vertailun tulokset voitaisiin esittää ilman paikkatietoa. Tällöin kartta voidaan nähdä puhtaasti koristeellisena käyttöliittymäelementtinä.

Tiheys–keveys-akselilla sovellus toimii sekä varsin keveänä päänäkymässä että tiheänä yksityiskohtanäkymässä. Päänäkymässä on hyvin vähän varsinaista esitettävää aineistoa pinta-alaan nähden, koska informaation välitys on jätetty yksityiskohtanäkymään: kiinnostuneet käyttäjät pääsevät tarkastelemaan tiheämpää datavisualisointia.

Moniulotteisuus–yksiulotteisuus-akselilla visualisointia voidaan pitää lukijan kannalta varsin moniulotteisena. Data pyritään esittämään neutraalilla tavalla etenkin yksityiskohtanäkymässä, mikä antaa tilaa lukijan omille tulkinnoille. Päänäkymässä yksiulotteisuutta tuovina elementteinä voidaan nähdä kärkilista, jossa näytetään yksipuolisesti parhaiten pärjänneet, ja karttaelementtien väritys, joka on asetettu yksiselitteisesti tulkittaviin kategorioihin menestyksen mukaan. Päänäkymän visualisoinnin suodatinvaihtoehtojen vähyys voidaan myös nähdä yksiulotteisena: käyttäjälle ei anneta mahdollisuutta muuttaa kärkilistan sisältöä tai karttavisualisoinnin väritystä omien kriteeriensä mukaan.

Ainutlaatuisuus–tuttuus-akselilla voidaan visualisoinnin katsoa asettuvan tukevasti tutun puolelle: visualisoinnissa ei ole käytetty uusia tai epätavallisia tapoja esittää aineistoa. Uutuus–toisto-akselilla visualisoinnin voidaan nähdä kallistuvan jonkin verran uuden puolelle. Yksittäisen lukion osalta informaatiota toistetaan vähän: karttakäyttöliittymän värityksestä voidaan tulkita osittain kokonaisuustulosta. Samoin kärkilistaan mahtuvien lukioiden kannalta listassa esiintyminen kertoo hyvästä tuloksesta vertailussa, mutta varsinaista aineistoa ei toisteta yksittäisen lukion kannalta. Koko visualisoinnin arvioinnissa mallin avulla sen voidaan nähdä sijoittuvan akseleille kuvaa 23 mukaillen. Arvioinnissa funktionaalisuus ja abstraktiotaso korostuvat, mutta kokonaisuus onnistuttiin pitämään itsearvioinnin perusteella kohtuullisessa tasapainossa kahden Cairon esittämän ääripään, teknisen ja taiteellisen, välillä.



Kuva 23. Lukiovertailuovelluksen käyttöliittymän ja ulkoasun itsearviointi Cairon mallin mukaisesti.

4.3 Lukiovertailudatan mallinnus tietokannassa

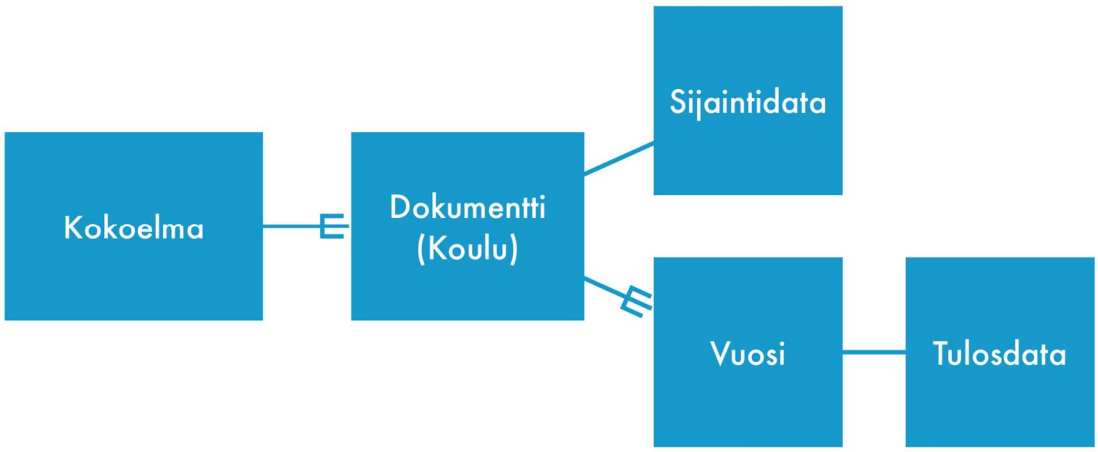
Datajournalistisen sovelluksen tarkoituksena on esittää dataa ja toimia käyttöliittymänä siihen. Jotta tämä tavoite pystyttiin toteuttamaan, piti data ensin tallentaa tavoitettavaan paikkaan. Lukiovertailusovelluksen osalta päädyttiin jo suunnitteluvaiheessa erottamaan data sovelluksesta erilliseen tietokantaratkaisuun.

Tietokantaan tallennettava aineisto koostui JSON-muotoisesta numeerisesta ja geokoodatusta informaatiosta lukioiden sijainneista, lukiovertailun tuloksista ja lukioiden taustatiedoista, kuten oppilasmääristä. Sovelluksen kehitystä voidaan nopeuttaa minimoimalla tämän jo koneluettavassa muodossa toimitettavan datan käsittelyä. Tämän perusteella pyritään projektin toteutukseen valitsemaan tekninen ratkaisu, joka pystyy hyödyntämään JSON-muotoista informaatiota mahdollisimman muokkaamattomana. Käytännössä tämä tarkoittaa luvussa 3.5 käsiteltyjä NoSQL-järjestelmiä, joiden skaa-

lautuvuutta ja datankäsittelyä ajatellen suunniteltu rakenne tekee ne myös soveltuviksi datapohjaiselle sovellukselle. Toteutettavaksi järjestelmäksi valittiin kirjoitushetkellä suosituin NoSQL-järjestelmä MongoDB [DB-Engines Ranking 2014]. Se mahdollistaa BSON-dokumenttimuotonsa ansiosta JSON-muotoisen aineiston lähes muokkaamattomana tallentamisen tietokantaan, mikä yksinkertaistaa käyttöönottoprosessia verrattuna perinteiseen relaatiotietokantatoteutukseen. MongoDB on myös hyvin tuettu Node.js-ympäristössä [Node.js MongoDB Driver 2014].

Käytettäväksi tietokantajärjestelmäpalveluntarjoajaksi valittiin MongoHQ sen tarjoaman elastisen järjestelmävaihtoehdon perusteella. Elastinen tietokantajärjestelmä skaalautuu siihen kohdistuvien hakujen vaatimien resurssien mukaisesti, mikä varmistaa mahdollisimman hyvän saatavuuden ja nopeat vaste-ajat myös yllättävien liikennepiikkien aikana. Tietokantajärjestelmän erottaminen sovelluksen muusta toiminnallisuudesta mahdollistaa tietokantapalvelimen hankinnan ulkopuoliselta taholta, mikä varmistaa sen vaihdettavuuden tulevaisuudessa muokkaamatta asiakassovellusta.

Tietokantaan vietävä JSON-muotoinen data-aineisto sisältää tiedon yksittäisten koulujen ylioppilaskirjoitusten tuloksista kirjoituksittain, keväästä 2012 lähtien. Jokaiselle koululle on lisäksi datassa määritelty tarvittava staattinen maantieteellinen lisäinformaatio, eli sijainti, kunta ja maakunta. JSON-datan voi viedä MongoHQ-palveluun BSON-dokumentiksi MongoDB:n tarjoaman mongoimport-toiminnallisuuden avulla. Mongoimport luo tietokantaan kokoelman, jonka dokumenteiksi asetetaan yksittäiset koulukohtaiset aineistot. MongoDB luo jokaiselle dokumentille dokumenttitunnisteen _id, ellei sitä ole erikseen luotu. Tietokantarakenne muodostuu kuvan 24 mukaiseksi.



Kuva 24. Lukiovertailusovelluksen tietokantarakenne. Kokoelma sisältää kaikki dokumentit (koulut), jokainen dokumentti sisältää kyseessä olevan koulun sijaintidatan ja kaikkien vuosien tulosdatan.

4.4 Lukiovertailudatan hakurajapinta

Asiakassovelluksen ja tietokannan välille toteutettiin palvelinsovellus, joka hakee tietokantaan tallennettua tietoa ja siirtää sitä asiakassovellukselle sen pyyntöjen perusteella. Palvelinsovellus toimii siis tiedonvälittäjänä asiakassovelluksen ja tietokannan välillä. Sen tärkeimpänä toiminnallisuutena on mahdollistaa datan välittäminen asiakassovellukselle REST-rajapinnan avulla. Täyden CRUD-toiminnallisuuden toteuttaminen ei kuitenkaan ole vaatimus tämän projektin osalta; data saadaan valmiiksi tietokantaan vietävässä muodossa, eikä sitä tarvitse muokata tai poistaa sovelluksen kautta tässä toteutuksessa. Rajapinnan laajentaminen täyteen CRUD-toiminnallisuuteen kuitenkin mahdollistaisi tulevaisuudessa erillisen sovelluksen rakentamisen datan lisäämistä, muokkausta ja käsittelyä varten, eli se on huomioitava potentiaalisena jatkokehityksen kohteena. Tämän sovelluksen kannalta tarpeelliset reitit määritellään taulukossa 1.

Taulukko 1. Sovelluksen RESTFUL-rajapinnan GET-reittien kuvaus

Reitti	**GET-pyyntö**
api/id/:id	Näytä lukio annetun id-tunnisteen perusteella
api/name/:name	Näytä lukiot joiden nimestä löytyy haettu teksti
api/all/latestlocations	Näytä kaikkien lukioiden sijaintitiedot ja viimeisin tulos

api/all/latestresults	Näytä kaikkien lukioiden viimeisimmät tulostiedot
api/all/year/:year	Näytä kaikkien lukioiden pyydetyn vuoden tulostiedot
api/:region/top	Näytä pyydetyn maakunnan parhaiten menestyneet lukiot

Node.js-palvelin yhdistettynä Express-ohjelmistokehykseen antaa valmiin pohjan palvelinsovelluksen rakenteelle. Express mahdollistaa REST-rajapinta-arkkitehtuuria mukailevien RESTFUL-verkkosovellusten rakentamisen [Cantelon ym. 2014: 76–81]. Reitittämällä saapuvia HTTP-pyyntöjä tietokannasta haettaviin resursseihin pystytään Expressin avulla yksinkertaisesti rakentamaan CRUD-toiminnallisuuksia mukaileva REST-rajapinta suoraan dataan, jota voidaan tarvittaessa käsitellä palvelinohjelmistossa [Mikowski & Powell 2014: 277–280].

Express-pohjainen node.js-sovellus voidaan oletuksena jakaa toiminnallisesti neljään osaan: varsinainen palvelinsovellus, reitit (engl. routes), näkymät (engl. views) ja julkiset staattiset resurssit. Express tarjoaa mahdollisuuden luoda automaattisesti tämän rakenteen mukaisen sovelluksen ja nopeuttaa näin sovelluskehityksen aloittamista [Yaapa 2013: 129–136]. Itse palvelinsovellus on Javascript-tiedosto, jonka sijainti määrittelee sovelluksen tiedostorakenteen juuren; Expressin standardinimitys tälle sovellukselle on app.js. App.js:ään määritellään koko palvelinsovelluksen perustoiminnallisuus HTTP-pyyntöihin reagoitaessa, kuten virheilmoitukset ja reitit staattisiin resursseihin.

Sovellukseen lisätään toimintoja hyödyntämällä erilaisia väliohjelmistoja, jotka suoritetaan omista tiedostoistaan; näin app.js-tiedostossa tarvitsee pitää vain sovelluksen ydintoiminnallisuus väliohjelmistojen suorittaessa erikoistuneemmat pyynnöt, kuten palvelinohjelmistolle kohdistuvien pyyntöjen reititys [Yaapa 2013: 87–94]. Reitityksellä tarkoitetaan tiettyyn URL-osoitteeseen kohdistuvan HTTP-pyynnön ohjaamista oikeaan näkymään. Kun Express-sovellukseen kohdistuu HTTP-pyyntö, sovellus luo pyyntöobjektin *req* (engl. request) ja vastausobjektin *res* (engl. response) pyynnön käsittelyä varten. Req-objekti sisältää tietoa pyynnöstä, kuten tiedon pyydetystä URL-osoitteesta ja mahdollisen pyynnön mukana siirretyn käyttäjän syöttämän datan [Yaapa 2013: 59–65]. Res-objektissa määritellään pyynnöstä aiheutuva palaute, yleensä ainakin HTTP-vastausobjekti ja HTTP-tilakoodi [Yaapa 2013: 65–70]. Näiden objektien sisältämän informaation perusteella reititin voi ohjata tietylle URL:lle osoitetun pyynnön sitä vastaavaan resurssiin palvelimella. Ohjaamalla tällä tavalla HTTP GET-, PUT-, POST- ja DELETE-metodeja merkityksiään vastaaviin tietokantakäskyihin voidaan yksinkertai-

sesti toteuttaa RESTFUL-rajapinta, joka tukee tarvittaessa täysimittaista CRUD-toiminnallisuutta.

Rajapintareititys määritellään omassa api.js-väliohjelmistossaan, jossa jokainen pyyntö liitetään tietokantahakufunktioon URL-rakenteen perusteella. Nämä hakufunktiot poimivat req-objektista tiedon halutuista hakutermeistä, jotka saadaan pyydetystä URL-osoitteesta. Varsinainen tietokantayhteys muodostetaan erillisellä tietokantaväliohjelmalla; tässä toteutuksessa tietokantaväliohjelmaksi valikoitui MongoJS. MongoJS hakee ja palauttaa MongoDB-tietokannasta hakufunktion hakutermien perusteella sisältöä, joka tallennetaan res-objektiin. Palvelinsovellus lähettää tämän jälkeen res-objektin vastauksena siihen kohdistuneeseen pyyntöön. Esimerkiksi kun asiakassovellus tarvitsee yksittäisen lukion tiedot, se lähettää Ajaxilla HTTP-pyynnön palvelinsovellukselle URL-osoitteeseen http://lukiovertailun.osoite.com/id/1234. Tässä numero 1234, eli id, on ennalta määritelty jokaisen lukion yksilöivä numero ja /id on pyydetty reitti. Palvelinsovellus vastaanottaa pyynnön ja käsittelee sen api.js-väliohjelmassa, jossa reititin tunnistaa annetun URL-rakenteen ja ohjaa pyynnön sen perusteella oikean tietokantahakufunktion käsiteltäväksi req-objektina. Tietokantahakufunktio poimii pyynnöstä eli req-objektista annetun muuttujan 1234 ja tekee sen perusteella MongoJS-väliohjelmistolla tietokantahaun. MongoJS palauttaa tehdyn tietokantahaun tuloksen, joka onnistuessaan sisältää tunnisteeseen 1234 liitetyn koulun tiedot, res-objektina. Palvelinsovellus palauttaa tämän jälkeen asiakassovellukselle res-objektin sisällön, joka tässä vaiheessa sisältää tietokannasta saatua JSON-muotoista aineistoa. Aineisto voidaan tämän jälkeen esittää halutulla tavalla asiakassovelluksessa. Hakufunktiot määritellään kaikille rajapinnan reiteille niin, että rajapinta vastaa kuvauksensa (taulukko 1) mukaisesti sille kohdistettuihin pyyntöihin.

Express mahdollistaa oletuskokoonpanossaan myös näkymien käytön. Näkymällä viitataan dynaamisiin sivumalleihin, jotka vastaavat MVC-arkkitehtuurimallin näkymää sovelluksessa, jossa reititin voidaan nähdä ohjaimena ja malli voitaisiin toteuttaa erillisenä tietokantakutsuja käsittelevänä väliohjelmistona [Tsonev 2013]. Express käyttää mallimoottorinaan (engl. template engine) oletusarvoisesti Jadea, jonka kuitenkin voi halutessaan korvata jollain muulla soveltuvalla sovelluksella. Jade mahdollistaa näkymien dynaamisen luomisen: Nämä dynaamiset näkymät luodaan Jaden omaa, HTML:ää syntaktisesti muistuttavaa, merkintäkieltä tulkitsemalla. Jade-merkintäkieli voi rakenteessaan sisältää funktioita, joiden avulla voidaan määritellä dynaamista sisältöä sivumalliin [Lindesay 2014]. Projektisovelluksen yksinkertaisuuteen pyrkivä asiakassovellus

ei kuitenkaan vaadi sivumallien käyttöä, vaan interaktiivinen toiminnallisuus toteutetaan jQueryllä.

5 Yhteenveto

Insinöörityönä toteutettiin STT-Lehtikuvalle datajournalistinen lukiovertailuaineistoa esittävä verkkosovellus moderneilla Javascript-pohjaisilla verkkosovellusteknologioilla. Työn paino asettui etenkin uusien teknisten ratkaisumahdollisuuksien tutkimiseen ja datan esittämisen toteutukseen sovelluksessa. Sovellus rakennettiin MongoDB-tietokantajärjestelmästä, Node.js-palvelinsovelluksesta ja pääosin Leaflet-, Morris.js- ja selectize-Javascript-kirjastojen pohjalta rakennetusta asiakassovelluksesta. Sovelluksen tarkoituksena on esittää STT:n tekemissä lukiovertailuissa kerättyä dataa.

Insinöörityön projektiosa toteutettiin tehdyn, raportissa esiteltävän taustatutkimuksen perusteella: Tutkimuksen teoreettinen osuus koostui datajournalismin käsitteen ja informaatiomuotoilun yleisimpien periaatteiden tutkimisesta. Käytännön tutkimusta tehtiin sovelluksen toteutukseen vaadittujen teknologioiden ja tekniikoiden osalta. Lisäksi vertailtiin myös niin sanottuja moderneja ja perinteisiä teknisiä ratkaisuja. Tämän tutkimuksen perusteella pyrittiin toteutukseen löytämään ja valitsemaan sovellukseen parhaiten soveltuvat tekniset, journalistiset ja visuaaliset ratkaisut.

Sovellus toteutettiin tutkitun SPA-verkkosovellusmallin mukaisesti yhtenä verkkosivuna ja sovelluksen toiminnalliset osat erotettiin toisistaan. Tietokantaan tallennettu data abstrahoitiin palvelinsovelluksessa RESTFUL-rajapinnaksi Express-ohjelmointikehyksen päälle rakennetulla Node.js-sovelluksella, jonka välityksellä dataa siirrettiin Ajax-teknologioilla palvelinsovelluksesta asiakassovellukseen. Asiakassovelluksen toiminnallisuus rakennettiin jQuery-Javascript-kirjastolla perinteisen verkkosivun HTML- ja CSS-rakenteen päälle. Leaflet-karttakirjastolla visualisoitiin paikkadataa kartalle ja Morris.js-visualisointityökalulla numeerista dataa diagrammeina. Visualisoinnissa pyrittiin Tuften ja Cairon esittämiin infografiikan hyväksi todettuihin käytäntöihin.

Sovellus toteuttaa lähtövaatimukset vähintäänkin kohtuullisesti. Javascript-pohjaisten sovelluskehitysteknologioiden perusajatus pohjautuu skaalattavuuteen ja saatavuuteen, joten tämä vaatimus täyttyy. Sekä palvelinsovellus että tietokantajärjestelmä voidaan valitun teknologiayhdistelmän ansiosta suorittaa skaalautuvissa pilvipalveluissa,

josta tarvittaessa voidaan lohkaista lisää suorituskykyä. Toteutettu asiakassovellus täyttää myös asetetut vaatimukset: se tukee Internet Explorer 8:aa uudempia selaimia, ja käyttäjäkokemus on hyvin yhteneväinen Internet Explorer 9:ssä ja sitä uudemmissa selaimissa. Sovellus on myös tarpeeksi kevyt suunniteltuun iframe-jakeluun.

Kokonaisuutena sovelluksen kehitysprosessi sujui pitkälti ongelmitta; sovellus valmistui aikataulussa ja julkaistiin 31.5.2014, mutta lopulta kierto loppuasiakkaille oli vähäistä. Loppuasiakkailta sovelluksesta ei saatu palautetta, mutta sisäisessä käytössä ja testausvaiheessa saatu palaute asiakkaalta oli pääosin myönteistä. STT:llä on tarkoituksena jatkaa sovelluksen kehitystä ja käyttää sitä lukiovertailuaineiston julkaisuun myös jatkossa. Mahdollisiksi kehityskohteiksi muodostuvatkin REST-rajapinnan CRUD-toiminnallisuuden täysimittainen toteuttaminen tai vaihtoehtoiseesti Excel-tiedostoja tietokantaan tuovan suoran toiminnallisuuden rakentaminen uuden tiedon syöttämisen yksinkertaistamiseksi. Sovellus on suunniteltu toimimaan käytännössä responsiivisesti, mutta mobiililaitetukea voisi jatkokehittää, sillä sitä ei tässä toteutuksessa priorisoitu.

Lähteet

Bateman, Scott; Mandryk, Regan L.; Gutwin, Carl; Genest Aaron; McDine David; Brooks, Christopher. 2010. Useful Junk? The Effects of Visual Embellishment on Comprehension and Memorability of Charts. University of Saskatchewan. Department of Computer Science.

Cantelon, Mike; Harter, Marc; Holowaychuk, T.J.; Rajlich, Nathan. 2014. Node.js in Action. Shelter Island: Manning Publications.

Cairo, Alberto. 2013. The Functional Art: An introduction to information graphics and visualization. Berkeley: New Riders.

Chaffer, Jonathan; Swedberg, Karl. 2013. Learning jQuery. Fourth Edition. Birmingham: Packt Publishing.

Downie, Nick. 2014. Chart.js Documentation. Verkkodokumentti. <http://www.chartjs.org/docs/>. Luettu 23.7.2014.

Chua, Reg. 2010. Structured Journalism. Verkkodokumentti. <http://structureofnews.wordpress.com/structured-journalism/>. Luettu 2.7.2014

Codd, E. F. 1982. Relational Database: A Practical Foundation for Productivity. Communications of the ACM. Volume 25 Issue 2, Feb 1982, s. 109–117.

Cois, Constantine Aaron. 2013. Why Should You Learn Node.js Today. Verkkodokumentti. <https://www.udemy.com/blog/learn-node-js/>. Päivitetty 9.6.2013. Luettu 21.7.2014.

Cox, Melisma, 2000. The Development of Computer-Assisted Reporting. A paper presented to the Newspaper Division, Association for Education in Journalism and Mass Communication. University of Miami.

DB-Engines Ranking. 2014. Verkkodokumentti. Solid IT. <http://db-engines.com/en/ranking>. Päivitetty 7.2014. Luettu 19.7.2014.

Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson; Wallach, Deborah A.; Burrows, Michael; Chandra, Tushar; Fikes, Andrew; Gruber, Robert. 2008. Bigtable: a distributed storage system for structured data. ACM Transactions on Computer Systems, 26, 2 (2008), s. 4:1-4:26.

Datavisualisation.ch selected tools. 2014. Verkkodokumentti. Interactive Things. <http://selection.datavisualization.ch/>. Luettu 23.7.2014.

DeCandia, Giuseppe; Hastorun, Deniz; Jampani, Madan; Kakulapati, Gunavardhan; Lakshman, Avinash; Pilchin, Alex; Sivasubramanian, Swaminathan; Vosshall, Peter; Vogels, Werner. 2007. Dynamo: Amazon's Highly Available Key-value Store. Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, s. 205–220.

Few, Stephen. 2011. The Chartjunk Debate. Visual Business Intelligence Newsletter, April, May and June: 2–10.

Fielding, Roy Thomas. 2000. Architectural Styles and the Design of Network-based Software Architectures. Doctoral Dissertation. University of California.

Flanagan, David. 2006. Javascript: The Definitive Guide. Fifth Edition. Sebastopol: O'Reilly Media.

Flanagan, David. 2011. Javascript: The Definitive Guide. 6th Edition. Sebastopol: O'Reilly Media.

G140: Separating information and structure from presentation to enable different presentations. 2014. Verkkodokumentti. W3C.org. <http://www.w3.org/TR/WCAG20-TECHS/G140.html>. Luettu 15.7.2014.

Garrett, Jesse James. 2005. Ajax: A New Approach to Web Applications. Verkkodokumentti. Adaptive Path. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/> Päivitetty 18.2.2005. Luettu 9.7.2014.

GitHub. 2014. Verkkodokumentti. Github, Inc. <https://github.com/search?o=desc&q=%22charts%22+OR+%22graphing%22+OR+%22visualization%22&ref=searchresults&s=stars&type=Repositories>. Päivitetty 23.7.2014. Luettu 23.7.2014.

Gray, Jonathan; Chambers, Lucy; Bounegru, Liliana (eds.). 2012. The Data Journalism Handbook. Sebastopol: O'Reilly Media.

Hazzard, Eric. 2011. OpenLayers 2.10. Brimingham: Packt Publishing.

Historical yearly trends in the usage of server-side programming languages for websites. 2014. Verkkodokumentti. Q-Success. <http://w3techs.com/technologies/history_overview/programming_language/ms/y>. Päivitetty 9.7.2014. Luettu 9.7.2014.

Hovi, Ari; Huotari, Joni; Lahdenmäki, Tapio. 2005. Tietokantojen suunnittelu & indeksointi. Jyväskylä: Docendo.

Introduction to MongoDB. 2014. Verkkodokumentti. MongoDB, Inc. <http://docs.mongodb.org/manual/core/introduction/>. Luettu 25.7.2014.

Jacomy, Alexis; Plique, Guillaume. 2014. Sigmajs. Verkkodokumentti. <http://sigmajs.org/>. Luettu 23.7.2014.

Lindesay, Forbes. 2014. Jade. Verkkodokumentti. <https://github.com/visionmedia/jade>. Päivitetty 24.7.2014. Luettu 28.7.2014.

June 2014 Web Server Survey. 2014. Verkkodokumentti. Netcraft. <http://news.netcraft.com/archives/2014/06/06/june-2014-web-server-survey.html>. Päivitetty 6.7.2014. Luettu 20.7.2014.

Kansallinen Mediatutkimus KMT 2013. 2014. Verkkodokumentti. MediaAuditFinland Oy, TNS Gallup. <http://mediaauditfinland.fi/wp-content/uploads/2014/04/KMT_lukijatiedote_helmikuu_2014.pdf>. Päivitetty 28.2.2014. Luettu 1.7.2014.

Konstantinou, Ioannis; Angelou, Evangelos; Boumpouka, Christina; Tsoumakos, Dimitrios; Koziris, Nectarios. 2011. On the elasticity of nosql databases over cloud management platforms. Proceedings of the 20th ACM international conference on Information and knowledge management, s. 2358–2388.

KTM Lehtien kokonaistavoittavuus 2013. 2014. Verkkodokumentti. MediaAuditFinland Oy.<http://mediaauditfinland.fi/wp-content/uploads/2014/05/kokonaistavoittavuus_2013.pdf>. Luettu 1.7.2014.

Laine, Harri. 2008. Ohjelmistojen mallinnus: Ohjelmistoarkkitehtuuri 2. Verkkodokumentti. <https://www.cs.helsinki.fi/u/laine/malli/s08/pdf/arkkitehtuuri3_c.pdf>. Helsingin yliopisto.

Lawton, George. 2005. LAMP Lights Enterprise Development Efforts. Computer. Volume 38, Issue 9, s. 18–20.

MacWright, Tom. 2012. How Do Web Maps Work?. Verkkodokumentti. <http://www.macwright.org/2012/05/15/how-web-maps-work.html>. Päivitetty 15.5.2012. Luettu 23.7.2014.

Maclean, Malcolm. 2014. Leaflet Tips and Tricks. Verkkodokumentti. <https://leanpub.com/leaflet-tips-and-tricks/read>. Päivitetty 28.2.2014. Luettu 22.7.2014.

MapQuest-OSM Tiles + MapQuest Open Aerial Tiles. 2014. Verkkodokumentti. MapQuest, Inc. <http://developer.mapquest.com/web/products/open/map>. Luettu 28.7.2014.

McLellan, Drew. 2005. Very Dynamic Web Interfaces. Verkkodokumentti. XML.com <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>. Päivitetty 9.2.2005. Luettu 9.7.2014.

Mielonen, Mika. 2013. HTML5-pohjainen sovelluskehitys ratkaisuna laitekannan pirstoutumiseen. Pro gradu -tutkielma. Helsingin yliopisto.

Mikowski, Michael S.; Powell, Josh C. 2014. Single Page Web Applications: Javascript End-to-End. Shelter Island: Manning Publications Co.

MongoDB Architecture Guide. 2014. Verkkodokumentti. MongoDb, Inc. <http://info.mongodb.com/rs/mongodb/images/MongoDB_Architecture_Guide.pdf>. Julkaistu 3.2014. Luettu 25.7.2014.

MongoDB Manual. 2014. Verkkodokumentti. MongoDB, Inc.<http://docs.mongodb.org/manual/>. Luettu 28.7.2014.

Mongoose Guide. 2014. Verkkodokumentti. Learnboost. <http://mongoosejs.com/docs/guide.html>. Päivitetty 27.7.2014.

Smith, Olly. 2014. Morris.js. Verkkodokumentti. <https://github.com/morrisjs/morris.js/>. Päivitetty 15.6.2014. Luettu 23.7.2014.

Murray, Scott. 2013. Interactive Data Visualization for the Web. Sebastopol: O'Reilly Media.

Mäkinen, Esa. 2013. Lista datajournalistin työkaluista. Verkkodokumentti. <http://www.esamakinen.fi/e/lista-datajournalistin-tyokaluista/>. Päivitetty 4.2.2013. Luettu 3.2.2014.

Nip, Joyce Y. M. 2006. Exploring the Second Phase of Public Journalism. Journalism Studies, Vol. 7, No 2, 2006: 212–236.

Nixon, Robin. 2012. Learning PHP, MySQL, Javascript and CSS. Sebastopol: O'Reilly Media.

Node.js MongoDB Driver. 2014. Verkkodokumentti. MongoDB, Inc. <http://docs.mongodb.org/ecosystem/drivers/node-js/>. Luettu 25.7.2014.

Node packaged modules. 2014. Verkkodokumentti. Npmjs.org. <https://www.npmjs.org/>. Luettu 25.7.2014.

npm-faq. 2014. Verkkodokumentti. Npmjs.org. <https://www.npmjs.org/doc/faq.html>. Luettu 22.7.2014.

Osmani, Addy. 2012. Learning Javascript Design Patterns. Sebastopol: O'Reilly Media.

Poikola, Antti. 2011. Mitä on datajournalismi? Verkkodokumentti. <http://www.slideshare.net/apoikola/mit-on-datajournalismi/>. 14.10.2011. Luettu 3.2.2014.

Poikola, Antti. 2013. Tutoriaaleja ja luentomateriaaleja. Verkkodokumentti. <http://datajournalismi.fi/tutoriaaleja/>. Luettu 3.2.2014.

Rauch, Guillermo. 2012. Smashing Node.js: Javascript Everywhere. Chichester: John Wiley & Sons.

Reavis, Brian. 2014. Selectize.js. Verkkodokumentti. <http://brianreavis.github.io/selectize.js/>. Luettu 28.7.2014.

Rickshaw. 2014. Verkkodokumentti. Shutterstock, Inc. <http://code.shutterstock.com/rickshaw/>. Luettu 23.7.2014.

Ritchie, Brian. 2010. An Introduction to Document Databases. Verkkodokumentti. <http://weblogs.asp.net/britchie/document-databases>. Päivitetty 13.8.2010. Luettu 11.11.2014.

Sanoman vuositulos 2013. 2014. Verkkodokumentti. Sanoma Oyj. <http://sanoma.com/sites/default/files/reports/sanoma_vuositulos_2013.pdf>. Päivitetty 7.2.2014. Luettu 2.7.2014.

Sandborg, Joel. 2012. Laajan mittakaavan Internet-sovelluksia varten kehitetyt hajautetut tietokannat. Pro gradu -tutkielma. Helsingin yliopisto.

Silberschatz, Abraham; Korth, F. Henry; Sudarshan, S. 2011. Database system concepts. Sixth Edition. New York: McGraw-Hill.

Standards for Web Applications on Mobile: Current state and roadmap. 2014. Verkkodokumentti. W3C. <http://www.w3.org/Mobile/mobile-web-app-state/>. Päivitetty 5.5.2014. Luettu 2.7.2014.

Switch2OSM: The Basics. 2013. Verkkodokumentti. OpenStreetMap. <http://switch2osm.org/the-basics/>. Luettu 24.8.2014.

Takada, Mikido. 2012. Mixu’s Node book: 2. What is Node.js? Verkkodokumentti. <http://book.mixu.net/node/ch2.html>. Luettu 21.7.2014.

Tiwari, Shashank. 2011. Professional NoSQL. Indianapolis: Wiley & Sons.

Tsonev, Krasimir. 2013. Build a Complete MVC Website With Express.js. Verkkodokumentti. <http://code.tutsplus.com/tutorials/build-a-complete-mvc-website-with-expressjs--net-34168>. Päivitetty 15.8.2013. Luettu 18.7.2014.

Tufte, Edward R. 2001 The Visual Display of Quantitative Information. Second Edition. Cheshire, Connecticut: Graphics Press.

Webber, Jim; Parastatidis, Savas; Robertson, Ian. 2010. REST in Practise. Sebastopol: O'Reilly Media.

Where Americans Get News. 2014. Verkkodokumentti. Pew Research Center. <http://www.journalism.org/media-indicators/where-americans-get-news/>. Päivitetty 27.9.2014. Luettu 1.7.2014.

Usage statistics and market share of Unix for Websites. 2014. Verkkodokumentti. Q-Success. <http://w3techs.com/technologies/details/os-unix/all/all>. Päivitetty 20.7.2014. Luettu 20.7.2014.

Yaapa, Hage. 2013. Express Web Application Development. Birmingham: Packt Publishing.

Map-script.js

```
1. /* map-script.js sisältää lähinnä visualisointeihin liittyvää toiminnalli-
   suutta */
2.
3. // Määritellään noudettavat vuodet
4.
5. var latestYear = "14"; //Viimeisin tietue
6. var yearIndex = [["2012", "12"], ["2013", "13"], ["2014", "14"]]; // Kaikki
   noudettavat tietueet & niiden selkokieliset nimet
7.
8. // luetaan kartan muuttjat urlista (pituuspiiri, leveyspiiri, zoomausaste)
9. function getUrlVars() {
10.     var vars = {};
11.     var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi,
   function (m, key, value) {
12.         vars[key] = value;
13.     });
14.     return vars;
15. }
16.
17. var markers = [];
18. var prevMarker;
19. var prevMarkerIcon;
20. var currentMarker;
21.
22. var isSchool = getUrlVars()["lukio"];
23. var alZoom = getUrlVars()["zoom"];
24. var alLat = getUrlVars()["lat"];
25. var alLong = getUrlVars()["lon"];
26.
27. // jos muuttujia ei ole määritelty urlissa, käytetään oletusarvoja
28.
29. if (alZoom == undefined || alZoom.length <= 0) {
30.     var alZoom = 5;
31. }
32. if (alLat == undefined || alLat.length <= 0) {
33.     var alLat = 64.741222;
34. }
35. if (alLong == undefined || alLong.length <= 0) {
36.     var alLong = 25.942411;
37. }
38.
39.
40. // Luodaan kartta #map-diviin ja asetetaan alkulat -long ja -zoom
41. var map = L.map('map', {
42.     attributionControl: false
43. }).setView([
44.     alLat, alLong
45. ], alZoom);
46.
47.
48. // Lisätään MapQuest karttakerros; footeriin attribuutio käyttöehtojen mu-
   kaisesti
```

```
49. L.tileLayer(/*'http://1.base.maps.cit.api.here.com/maptile/2.1/maptile/newest/normal.day/{z}/{x}/{y}/256/png8?app_id=hKC3ALJu42sX7E4ymlGD&app_code=Q3b0QyPOjtXK2ub6QtYEug'*/
    'http://otile4.mqcdn.com/tiles/1.0.0/map/{z}/{x}/{y}.png'/*'http://{s}.tiles.mapbox.com/v3/kodhage.i5h55cih/{z}/{x}/{y}.png'*/, {
50.     attribution: 'Aineisto: Ylioppilastutkintolautakunta, STT. Kartat: <a href="http://www.mapquest.com/" target="_blank">MapQuest</a> <img src="http://developer.mapquest.com/content/osm/mq_logo.png"> | <a href="http://www.openstreetmap.org/copyright">© OpenStreetMap</a>',
51.     detectretina: true,
52.     attributionControl:false
53. }).addTo(map);
54. L.control.attribution({position: 'bottomleft'}).addTo(map);
55.
56. var redMarker = L.icon({
57.     iconUrl: '../images/red-icon.png',
58.     iconRetinaUrl: '../images/red-icon@2x.png',
59.     iconSize: [32, 32],
60.     iconAnchor: [16, 16],
61.     labelAnchor: [16, 3]
62. });
63.
64. var greenMarker = L.icon({
65.     iconUrl: '../images/green-icon.png',
66.     iconRetinaUrl: '../images/green-icon@2x.png',
67.     iconSize: [32, 32],
68.     iconAnchor: [16, 16],
69.     labelAnchor: [16, 3]
70. });
71.
72. var orangeMarker = L.icon({
73.     iconUrl: '../images/yellow-icon.png',
74.     iconRetinaUrl: '../images/yellow-icon@2x.png',
75.     iconSize: [32, 32],
76.     iconAnchor: [16, 16],
77.     labelAnchor: [16, 3]
78. });
79.
80. var markerClusters = new L.MarkerClusterGroup({
81.     showCoverageOnHover: false,
82.     maxClusterRadius:70,
83.     disableClusteringAtZoom: 13
84. });
85.
86. $.getJSON("api/all/latestlocations/", function (data) {
87.     $.each(data, function (i, item) {
88.         var val = parseFloat(item[latestYear].result);
89.         var markerType;
90.
91.         if (val >= 0.2) {
92.             markerType = greenMarker;
93.         } else if (val >= -0.2 && val <= 0.2) {
94.             markerType = orangeMarker;
95.         } else if (val < -0.2) {
96.             markerType = redMarker;
97.         }
98.
```

```
99.
100.      //Luodaan markkeri, annetaan sille arvot noudetusta datasta ja sido-
   taan onClick-ominaisuudet & muut markkerikohtaiset parametrit.
101.
102.      //ifib() tarkistaa onko lukion kohdalla huomoitu IB-linja (määritel-
   lään arrayssa)
103.      function ifib(id) {
104.        var ibid = {"534bcccd4ab7b06f8e3ac08f": 7 ,
   "534bcccd4ab7b06f8e3ac087": 24, "534bcccd4ab7b06f8e3ac1a0": 17,
   "534bcccd4ab7b06f8e3ac0f7": 131, "534bcccd4ab7b06f8e3ac132" : 301}
105.        var val = ibid[id];
106.        if(val) {
107.          $("#ifib").html("Vertailussa on huomioitu myös IB-linjan opiskeli-
   joiden sisäänpääsykeskiarvojen keskiarvot. Ilman IB-linjan opiskelijoita lu-
   kio sijoittuisi vertailussa sijalle "+val+".");
108.      } else {
109.          $("#ifib").html("");
110.        }
111.      }
112.
113.          var marker = L.marker([item.lat, item.lon], {
114.                  icon: markerType,
115.                  riseOnHover: true,
116.                  id: item._id,
117.                  title: item.schoolName
118.              })  .on('click', function () {
119.                  $("#schoolName").html(item[latestYear].rank+".
   "+item.schoolName);
120.                  currentMarker = item._id; // Klikattu marker 'tämänhet-
   kiseksi'
121.                  putData(item); // Pusketaan data liukufooteriin
122.                  redrawChart(item._id, "rank"); // Piirre-
   tään/uudelleenpiirretään graafi morris.js:llä
123.                  ifib(item._id);
124.                  $("#graphHeader").html("Sijoitus STT:n Lukiovertailussa");
125.                  marker.options.zIndexOffset = 250; // nostetaan klikattu
   marker ylimmäiseksi
126.                  setHash(item._id); // asetetaan risuaitanavigaatio
127.              }).bindPopup("<h2>"+item[latestYear].rank+".
   "+item.schoolName+"<h2><a id='lukioinfo' href='#'>Näytä lukion tiedot</a>")
128.              markers.push(marker); // Lisätään marker listaan
129.              markerClusters.addLayer(marker); // Käsitellään markkeri mar-
   kerClusters-pluginilla
130.          });
131.          map.addLayer(markerClusters); // Lisätään markkeriklusterit kar-
   talle
132.          checkHash(hash); // Tarkistetaan onko risuaitanavigaatiota urlissa
133.      });
134.
135.      $("body").on('click', '#lukioinfo', function(){
136.        $("#footerSlideButton").trigger('click')
137.        if($(this).html() === "Näytä lukion tiedot")
138.        {
139.          $(this).html("Piilota tiedot");
140.        } else {
141.          $(this).html("Näytä lukion tiedot");
142.        }
```

```
143.      }
144.      );
145.
146.      /*
147.          redrawChart-funktio piirtää morris.js:llä graafin yearIndexissä
   määritellyiltä vuosilta, poimien targetDatassa määritellyn aineiston id:n
   mukaan rajapinnasta
148.      */
149.      function redrawChart(id, targetData) {
150.
151.          if(!id){$("#graphHeader").html("Valitse ensin lukio");}else{
152.          // Pyydä dataa APIlta
153.          $.getJSON("api/id/" + id, function (data) {
154.
155.              // Tyhejennetään graafi-div
156.              $("#footerGraph").empty();
157.
158.
159.              var chartData = [];
160.              var max;
161.              var min;
162.
163.              // Käydään läpi graafiin piirrettävä data
164.              for (var i = 0; i < yearIndex.length; i++) {
165.                  // Vuosi-indeksin i:nnen arvon arvo
166.                  var index = yearIndex[i][1];
167.                  chartData[i] = {};
168.                  if (!data[0][index][targetData]) {
169.                      chartData[i].a = null;
170.                  } else {
171.                      chartData[i].a = data[0][index][targetData];
172.                  }
173.                  chartData[i].y = yearIndex[i][0];
174.
175.                  // max- ja min-arvojen asetus haetun datan mukaan
176.                  // tarpeellista pisterajoja varten (esim. arvosana ei voi
   olla yli 10)
177.                  // Muut arvot automaattisesti
178.
179.
180.                  if (targetData == "entryGradeAvg") {
181.                      max = 10;
182.                      min = 7;
183.                      chartData[i].y = chartData[i].y-3;
184.                  } else if (targetData == "passedTests" && chartData[i].a >
   max) {
185.                      min = max;
186.                      max = chartData[i].a + 2;
187.                  } else if (targetData == "failedTests" && chartData[i].a >
   max) {
188.                      min = max;
189.                      max = chartData[i].a + 2;
190.                  } else if (targetData == "result") {
191.                      max = 1;
192.                      min = -1;
193.                  } else if (targetData == "yoGradeConverstion") {
194.                      max = 10;
```

```
                    min = 7;
                }
            }

            // Piirretään graafi
            // Donitsi
            if (targetData == "failedTestsPercentage") {
                var z1 = parseInt(data[0][index]["passedTests"]);
                var z2 = parseInt(data[0][index]["failedTests"]);
                Morris.Donut({
                    element: 'footerGraph',
                    data: [{
                        label: "Hylätyt",
                        value: z2
                    }, {
                        label: "Hyväksytyt",
                        value: z1
                    }, ],
                    formatter: function (y, data) {
                        var ret = (y / (z1 + z2) * 100).toFixed(2);
                        return ret + ' %'
                    },
                    colors: ["#e86061", "#6abe5a"]

                });
                // Sijoitus (käänteinen akseli)
            } else if (targetData == "rank") {
                chartData = chartData.map(function (x) {
                    x.a = -x.a;
                    return x;
                });

                var format = function (y) {
                    return Morris.commas(-y);
                }
                Morris.Line({
                    element: 'footerGraph',
                    data: chartData,
                    xkey: 'y',
                    ykeys: ['a'],
                    labels: ['Sijoitus'],
                    parseTime: false,
                    continuousLine: false,
                    yLabelFormat: format,
                    lineColors: ['#0099cc'],
                    hideHover: false,
                    ymax: 0,
                    ymin: -400,
                    format: function(y) {return Morris.commas(y)},
                    gridIntegers: true
                });
            } else {
                // Viiva
                var chart = Morris.Line({
                    gridIntegers: true,
                    element: 'footerGraph',
                    xkey: 'y',
```

```
252.                     ykeys: ['a'],
253.                     labels: [''],
254.                     lineColors: ['#0099cc'],
255.                     hideHover: false,
256.                     ymax: max,
257.                     ymin: min,
258.                     parseTime: false,
259.                     continuousLine: false,
260.                     format: function(y) {return Morris.commas(y)},
261.                     //xLabelFormat: function(y){ return "" },
262.                     resize: true
263.                 });
264.                 chart.setData(chartData);
265.             }
266.         });
267.     }
268.     }
269.     /*
270.         putData asettaa arvot liukukenttään, kutsutaan klikkauksesta
271.     */
272.     function putData(item) {
273.         $("#maakunta").html(item.maakunta);
274.         $("#kunta").html(item.kunta);
275.         $("#hyvaksyttyja").html(item[latestYear].passedTests);
276.         $("#reputtaneita").html(item[latestYear].failedTests);
277.         $("#reputtaneita-
    pros").html(((parseInt(item[latestYear].passedTests)/(parseInt(item[latestYe
    ar].failedTests)+parseInt(item[latestYear].passedTests)))*100).toFixed(2)+"
    %");
278.         $("#kanka").html(item[latestYear].entryGradeAvg);
279.         $("#sijoitus").html(item[latestYear].rank);
280.         $("#tu-
    los").html(parseFloat((item[latestYear].result)).toFixed(2));
281.         $("#yokoetu-
    los").html(parseFloat((item[latestYear].yoGradeConverstion)).toFixed(2));
282.         $("#tulo-
    spuoltoaani").html(parseFloat((item[latestYear].unconvertedGrade)).toFixed(2
    ));
283.
284.         map.setView([item.lat, item.lon],13);
285.     }
286.
287.     /*
288.         Kuuntelee klikkauksia liukukentässä
289.     */
290.     $(".link").click(function(){
291.         var value = $(this).attr("value");
292.         var graphHeaders = {"entryGradeAvg":"Sisäänpääsykeskiarvojen kes-
    kiarvo 3 v. ennen kirjoituksia","result":"Opiskelijoiden koulumenestyksen
    muutos lukioaikana kouluarvosanoina"};
293.         if (graphHeaders[value]){
294.           $("#graphHeader").html(graphHeaders[value]);
295.         } else {
296.         $("#graphHeader").html($(this).html());
297.         }
298.         redrawChart(currentMarker, value);
299.     });
```

```
300.
301.      /*
302.          markerFunction laukaisee määritellyn markkerin (id) onClick-
   funktion. onClick määritelty markeria luotaessa.
303.      */
304.      function markerFunction(id) {
305.          for (var i in markers) {
306.              var markerID = markers[i].options.id;
307.              if (markerID == id) {
308.                  markers[i].fire("click");
309.              };
310.          }
311.      }
312.
313.      /*
314.          Risuaitanavigaation ominaisuuksia:
315.          hash: selaimesta poimittu #-arvo
316.          checkHash: tarkistus että #-arvo on validi mongo-id (regxHash)
317.          Jos totta, iskee click-eventin markerFunction kautta
318.          setHash: asettaa saadun id:n URLiin.
319.      */
320.
321.      var hash = window.location.hash.substr(1);
322.      var regxHash = new RegExp("^[0-9a-fA-F]{24}$");
323.      //console.log(hash);
324.
325.      function checkHash (hash) {
326.          if  (hash.match(regxHash)) {
327.              markerFunction(hash);
328.          } else if (hash == ""){
329.          } else {
330.                  alert("Osoitteen mukaista koulua ei löytynyt. Näppäilyvir-
   he?");
331.          }
332.      };
333.
334.      function setHash (id){
335.          window.location.hash = id;
336.      };
337.
338.      $("#clearMap").click(function(){map.setView([alLat, alLong], al-
   Zoom);})
```

Site-functions.js

```
1. /*   site-functions.js sisältää sivun toiminnallisuudet jotka eivät liity
   suoraan kartan toimintaan.
2. */
3.
4. $(function () {
5.
6. /*  Selectize.js -pluginin määrittely
7.     Selectize hakee rajapinnan findByName-toiminnon kautta tietokannasta
   hakutuloksia käyttäjän syötteen mukaan.
8. */
9.
10.  $("#get").click(function () {
11.    markerFunction($(".item").attr('data-value'));
12.});
13.
14.    getTop("all");
15.
16.    function setMapRegion(region) {
17.      var regionLatlng = {"all":[65.241222,25.742411],"etela-karjala":
   [61.060003, 28.195694], "etela-pohjanmaa":[62.794671, 22.831903], "etela-
   savo":[61.699468, 27.282986], "kainuu":[64.224183, 27.721739], "kanta-
   hame":[60.994626, 24.451394], "keski-pohjanmaa":[63.838506, 23.130268],
   "keski-suomi":[62.249189, 25.738331], "kymenlaakso":[60.866758, 26.698344],
   "Lappi":[66.572375, 25.623507], "pirkanmaa":[61.493701, 23.748436], "pohjan-
   maa":[63.116398, 21.620444],"pohjois-karjala":[62.638636, 29.737118], "poh-
   jois-pohjanmaa":[65.029503, 25.459067], "pohjois-savo":[62.904459,
   27.688106], "paijat-hame":[60.989022, 25.628060],"satakunta":[61.496017,
   21.813007], "uusimaa":[60.171918, 24.935979], "varsinais-suomi":[60.461033,
   22.254091]};
18.      var setLatlng = regionLatlng[region];
19.
   if(region==="all"){map.setView(setLatlng,5);}else{map.setView(setLatlng,8);}
20.    }
21.
22.    function getTop(region) {
23.      $.getJSON('api/'+region+'/top/', function (data){
24.        var items = [];
25.      $.each( data, function( key, val ) {
26.        items.push( "<li> <a class='toplist' href='#' value='"+val._id+"'>"+
   val.schoolName +"</a></li>" );
27.    });
28.    $("#top-schools").html(items);
29.    $('#region-select').selectize({onChange: function(){
30.      setMapRegion(this.getValue());
31.      getTop(this.getValue());
32.    }});
33.    $("#region-container").find("input").hide();
34.    });
35.  }
36.
37.  $("#top-schools").on('click', '.toplist', function() {
38.    var id = $(this).attr('value');
39.    markerFunction(id);
40.  })
```

```
41.
42.     $('#select-school').selectize({
43.         valueField: '_id',
44.         labelField: 'schoolName',
45.         searchField: 'schoolName',
46.         create: false,
47.         render: {
48.             option: function (item, escape) {
49.                 //console.log(item);
50.                 return '<div>' +
51.                     '<span class="title">' +
52.                     '<span class="name"></i>' + escape(item.schoolName) +
    '</span>' +
53.                     '</span>' +
54.                     '</div>';
55.             }
56.         },
57.         load: function (query, callback) {
58.             $.ajax({
59.                 url: 'api/name/' + encodeURIComponent(query),
60.                 type: 'GET',
61.                 error: function () {
62.                     callback();
63.                 },
64.                 success: function (res) {
65.                     callback(res.slice(0, 10));
66.                 }
67.             });
68.         },
69.         onChange: function() {
70.             markerFunction($("#search-container").find(".item").attr('data-
    value'));
71.             hideKeyboard();
72.         }
73.
74.     });
75.
76. /* 'Liukuva' footer-kenttä
77.     Tarkistukset responsiivisuutta varten (checkMobile()), vaihtelevat koot
    ruudun leveyden mukaan
78. */
79.
80. var isBig;
81. var open = false;
82.
83.     function checkMobile() {
84.
85.     // Suuri näyttö
86.
87.     var wW = $(window).width();
88.     if (wW > 800) {
89.
90.     if (/*isBig === false && */open === true) {
91.     $("#footerSlideButton").trigger('click');
92.     $("#footerSlideButton").unbind('click');
93.     isBig = true;
94.     } else if(isBig === false){
```

```
95.           $("#footerSlideButton").unbind('click');
96.         isBig = true;
97.         checkMobile();
98.     }else {
99.     open = false;
100.          isBig = true;
101.          $("#footerSlideButton").unbind('click');
102.          $('#footerSlideButton').click(function () {
103.              if (open === false) {
104.                  $('#footerSlideContent').animate({
105.                      height: '500px'
106.                  },250);
107.                  /*$('#map').animate({
108.                      'bottom': '200px'
109.                  },250);*/
110.                  $('.leaflet-bottom').hide();
111.
112.                  $(this).css('backgroundPosition', 'bottom left');
113.                  map.panBy([0, 200]);
114.                  //map.setZoom(8);
115.                  $("#openClose").html("Piilota tiedot");
116.                  open = true;
117.
118.              } else {
119.                  $('#footerSlideContent').animate({
120.                      height: '0px'
121.                  },250);
122.                  /*$('#map').animate({
123.                    'bottom': '0px'
124.                  },250);*/
125.                  $('.leaflet-bottom').show();
126.
127.                  $(this).css('backgroundPosition', 'top left');
128.                  //map.setView();
129.                  map.panBy([0, -200]);
130.                  $("#openClose").html("Klikkaa tästä nähdäksesi valitun
    lukion tiedot");
131.                  open = false;
132.              }
133.          });}
134.
135.          // Pieni näyttö
136.
137.          } else if (wW < 800) {
138.            if (open === true) {
139.            $("#footerSlideButton").trigger('click');
140.            $("#footerSlideButton").unbind('click');
141.              isBig = false;
142.                //open = false;
143.            } else if(isBig === true){
144.                $("#footerSlideButton").unbind('click');
145.                isBig = false;
146.                checkMobile();
147.      } else {
148.                open = false;
149.                isBig = false;
150.              $("#footerSlideButton").unbind('click');
```

```
151.          $('#footerSlideButton').click(function () {
152.              if (open === false) {
153.                  $('#footerSlideContent').animate({
154.                      height: $(window).height() - 100
155.                  },250);
156.                  $('.leaflet-bottom').hide();
157.                  $('#map').animate({
158.                      'bottom': $(window).height() - 100
159.                  },250);
160.                  $('.leaflet-top').hide();
161.                  $(this).css('backgroundPosition', 'bottom left');
162.                  map.panBy([0, 300]);
163.                  $("#openClose").html("Piilota tiedot");
164.                  open = true;
165.
166.              } else {
167.                  $('#footerSlideContent').animate({
168.                  height: '0px'
169.                  },250);
170.                  $('#map').animate({
171.                      'bottom': '0px'
172.                  },250);
173.                  $('.leaflet-bottom').show();
174.                  $('.leaflet-top').show();
175.                  $(this).css('backgroundPosition', 'top left');
176.                  map.panBy([0, -300]);
177.                  $("#openClose").html("Klikkaa tästä nähdäksesi valitun
    lukion tiedot");
178.                  open = false;
179.              }
180.          });
181.          }}
182.          }
183.
184.          checkMobile();
185.          $(window).resize(function(){checkMobile();});
186.
187.          function hideKeyboard () {
188.       document.activeElement.blur();
189.       var inputs = document.querySelectorAll('input');
190.       for(var i=0; i < inputs.length; i++) {
191.        inputs[i].blur();
192.       }
193.      };
194.      });
```