jamk.fi

Ei-relaatiotietokanta web-sovelluksessa

Case: JUHA

Tommi Kaura-Aho

Opinnäytetyö
Marraskuu 2013

Tietojenkäsittelyn koulutusohjelma
Luonnontieteiden ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) Kaura-Aho, Tommi	Julkaisun laji Opinnäytetyö	Päivämäärä 12.11.2013
	Sivumäärä 49	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
Ei-relaatiotietokanta web-sovelluksessa
Case: JUHA

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
Tuikka, Tommi

Toimeksiantaja(t)
Protacon Solutions Oy

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi jyväskyläläinen IT-alan yritys Protacon Solutions Oy. Yritys suunnittelee, toteuttaa ja ylläpitää erilaisia IT-sovelluksia. Yksi kanava, jota kautta yritys etsii potentiaalisia asiakkaita, on julkisten hankintojen ilmoituskanava HILMA. Tämän järjestelmän selaamista varten yritys tarvitsi työkalun, joka mahdollistaa hankintailmoitusten selaamisen, filtteroinnin ja merkinnän.

Opinnäytetyön toissijainen tavoite oli saada yritykselle kokemusta ja tietoa uusista teknologioista erityisesti ei-relaatiotietokannoista. Ei-relaatiotietokannat ovat uusi tulokas tietokantamarkkinoilla ja haastavat suorituskyvyllään ja skaalautuvuudellaan useita vuosikymmeniä vallassa olevat relaatiotietokannat.

Työn tuloksena saatiin aikaan toimiva tietojärjestelmä ja vertailupohjaista tietoa eri ei-relaatiotietokannoista. Lisäksi saatiin tietoa siitä, miten ei-relaatiotietokannat käyttäytyvät kehitysprojektissa. Työn perusteella muodostettiin myös hypoteesi siitä, että todennäköisesti ei-relaatiotietokannat eivät tule syrjäyttämään relaatiotietokantoja lähitulevaisuudessa, vaan elämään näiden rinnalla ratkaisemaan spesifejä ongelmia.

Avainsanat (asiasanat)
HILMA, Julkinen hankinta, NoSQL, Ei-relaatiotietokanta

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Kaura-Aho, Tommi	Bachelor´s Thesis	12.11.2013
	Pages 49	Language
		Permission for web publication (X)

Title
NoSQL-database in web-application
Case: JUHA

Degree Programme
Business Information Systems

Tutor(s)
Tuikka, Tommi

Assigned by
Protacon Solutions Ltd.

Abstract

This thesis was assigned by an IT-company from Jyväskylä named Protacon Solutions Ltd. The company plans, implements and maintains different IT-applications. One avenue, where the company seeks out potential customers, is a public procurement notice system called HILMA. In order to browse, filter and mark potential notices, the company required an application.

The secondary objective of the thesis was to gain information about new technologies, especially non-relational databases. Non-relational databases (or NoSQL) are a new product on the market, which challenges in performance and scalability relational databases, which have been in power for several decades.

The results of this project provided the company with a working application and comparable knowledge about non-relational databases. In addition, information was received on how non-relational databases function in development. Based on the project, a hypothesis was formed on the future of non-relational databases, which specified that non-relational databases are not going to surpass relational databases, but are instead working alongside them to resolve specific problems.

Keywords
HILMA, Public procurement, NoSQL, non-relational database

Miscellaneous

SISÄLTÖ

KUVIOT

Käsitteet

Ei-relaatiotietokanta

Uusi tietokantamalli, joka poikkeaa relaatiotietokantamallista.

backend

Palvelimen päähän asennettu kokonaisuus, jonka tarkoitus on suorittaa laskutoimituksia ja sovelluksen toimintalogiikkaa taustalla.

single-page-application

Kuvaa web-sovellusta, jossa kaikki toiminta tapahtuu yhdellä html-sivulla.

HTML

Hyperlinkkejä sisältävänä tekstin kuvailukieli, joka kuvailee tekstin eri rakenneosio tageilla.

Word Wide Web

CERN:n vuonna 1991 julkaisema hypertekstijärjestelmästandardi.

http

Hypertekstin siirtoprotokolla

php

Palvelinpuolen ohjelmointikieli, jota käytetään dynaamisen web-sivujen luonnissa.

AJAX

Teknologiakokonaisuus, jonka tarkoitus on nopeuttaa web-sovellusten toimintaa suorittamalla palvelinkutsuja asynkronisesti.

Relaatietietokanta

Relaatiomalliin ja joukko-oppiin perustuva tietokantaratkaisu, jossa tieto tallennetaan tauluihin ja sarakkeisiin.

JSON

JavaScript-perusteinen yksinkertainen tiedonsiirtomuoto, joka seuraa JavaScriptin olionotaatiota.

ORM

Object Relational Mapping kuvaa backend-sovelluskehysten tapaa abstraktoida tietokannoille tehdyt kyselyt helpommin ymmärrettävämpään muotoon.

1 Johdanto

HILMA on julkisten hankkeiden ilmoitukseen tarkoitettu ilmoitusjärjestelmä. Ei-relaatiotietokannat (tai NoSQL) on uusi teknologia, jonka tarkoituksena on näkökannasta riippuen joko syrjäyttää tai täydentää jo yli 30 vuotta vallalla ollutta relaatiotietokantamallia. Mutta miten nämä kaksi liittyvät toisiinsa?

Protacon Solutions Oy pyrkii jatkuvasti seuraamaan HILMA:ssa olevia ilmoituksia ja tekemään niistä tarjouksia. Ilmoitusten selausprosessi on kuitenkin jäänyt viime vuosituhannen tasolle, joten siihen tarvitaan muutosta tietojärjestelmän avulla.

Tietojärjestelmän lisäksi yritys haluaa saada tietoa uusista teknologioista (tässä tapauksessa ei-relaatiotietokannoista) ja nähdä, miten ne käyttäytyvät niin tuotannossa kuin kehityksessäkin.

Tämän työn tarkoituksena on tarjota kyseinen tietojärjestelmä ja toimittaa tutkimus ei-relaatiotietokantojen toiminnasta tämän järjestelmän kehityksen raameissa.

2 Työn lähtökohdat

2.1 Toimeksiantaja ja tutkimuksen tausta

Protacon Solutions Oy on jyväskyläläinen ICT-palveluihin erikoistunut yritys, joka tarjoaa mm. ohjelmistojen suunnittelua ja toteuttamista sekä valmiiden ohjelmistojen ylläpitoa ja konesalipalvelua. Protacon työllistää noin 200 eri alojen ammattilaista ja toimii kahdeksalla paikkakunnalla. (Yritys n.d.)

Valtion, kuntien ja viranomaisten on lain mukaan julkisesti kilpailutettava hankintansa. Tällä lailla ja siihen liittyvillä asetuksilla on tarkoitus tehostaa julkisten varojen käyttöä ja turvata yritysten mahdollisuudet tarjota palveluitaan ja tuotteitaan julkisissa hankintatilanteissa. (L 348/2007, 1§.)

Julkisia hankintoja varten on luotu oma järjestelmänsä (joka tunnetaan nimellä HILMA) ja järjestelmän yhteyteen www-sivusto *hankintailmoitukset.fi,* joka tarjoaa maksuttoman sähköisen ilmoituskanavan julkisia hankkeita varten (HILMA:n etusivu). Protacon pyrkii seuraamaan HILMA:n ilmoituksia ja tekemään niistä tarjouksia, joista se on useita voittanutkin. HILMA:n seuraamiseen ei Protaconilla kuitenkaan ole valmista järjestelmää, vaan sihteerit tekevät potentiaalisista hankkeista listan viikoittain, ja se toimitetaan eteenpäin. Käytäntö aiheuttaa turhaa manuaalista työtä, joten tätä tarkoitusta varten tarvitaan oma tietojärjestelmänsä.

2.2 Tutkimuksen tavoite ja tutkimuskysymykset

Työn tavoitteena on saada aikaiseksi toimiva tietojärjestelmä, joka listaa Protaconia kiinnostavia hankkeita ja mahdollistaa näiden selailun, kommentoinnin ja hyväksynnän yrityksen toiminnanohjausjärjestelmään.

Toissijaisena tavoitteena on saada välillisesti uutta tietoa siitä, miten **ei-relaatiotietokannat** toimivat valittujen järjestelmien kanssa yhteistyössä. Yhtenä tavoitteena on myös minimoida **backend-**toiminta järjestelmässä ja keskittää mah-

dollisimman paljon sovelluksen toimintalogiikasta käyttäjän päähän eli tehdä sovelluksesta selainpainotteinen **single-page-application.**

Projektin taustalla rajoittamassa ja suuntaamassa näkökulmaa toimivat seuraavat tutkimuskysymykset:

- Miten ei-relaatiotietokanta käyttäytyy valittujen sovelluskehysten ja kirjastojen kanssa?
- Miksi kannattaa käyttää ei-relaatiotietokantaa eikä relaatiotietokantaa?
- Mikä ei-relaatiotietokanta on paras valinta?

3 Web-sovelluksista

3.1 Internetin ja WWW:n lyhyt historia

Vuonna 1957 Neuvostoliitto järkytti läntisiä maita laukaisemalla ensimmäisenä onnistuneen satelliitin Maan kiertoradalle. Sputnik I:sen onnistuneen lennon vuoksi Yhdysvaltojen presidentti Eisenhower näki tarpeelliseksi perustaa *Advanced Research Projects Agencyn* eli ARPAn (Gromov 1995, 2). ARPA suoritti alkuperäisen tehtävänsä (satelliitin rakentaminen) 18 kuukaudessa ja keskittyi sen jälkeen tietoverkkoihin ja kommunikaatiomenetelmiin (Gromov 1995,3).

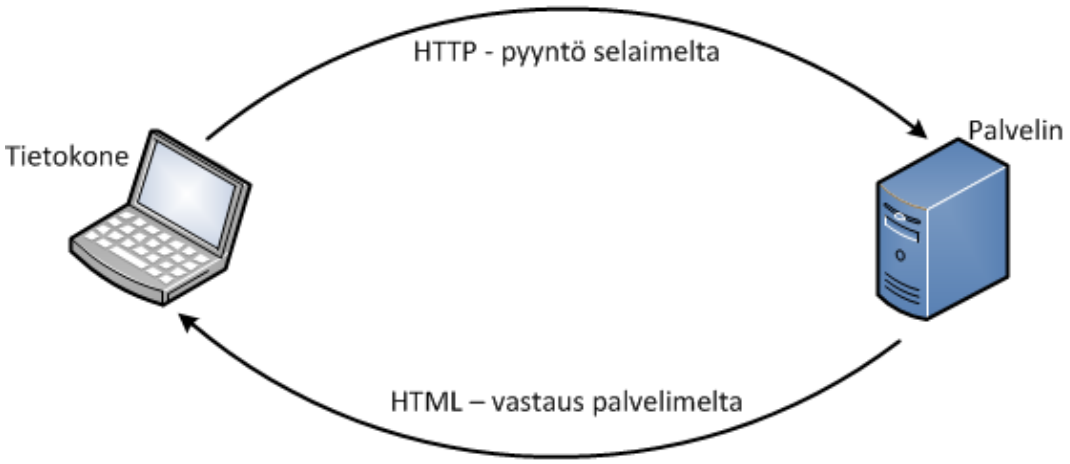
Vuonna 1969 onnistuttiin ensimmäisen kerran ottamaan yhteys toiseen tietokoneeseen. Neljä yliopistoa, UCLA Los Angelesissa, Stanfordin tutkimuskeskus (SRI), Santa Barbata (UCSB) ja yliopisto Utahissa Salt Lake cityssä, yhdistettiin toisiinsa neljäsolmuiseen verkkoon. Kokeilun tarkoitus oli yhdistää tietokoneet ja onnistuneesti välittää dataa UCLA:n ja SRI:n tietokoneiden välillä kirjoittamalla ”login” UCLA:n päässä ja katsomalla, näkyykö kirjoitus SRI:n tietokoneella. Vaikka järjestelmä kaatuikin o-kirjaimeen päästyään, kyseessä oli silti vallankumouksellinen tapahtuma. (Gromov 1995, 4–5.)

Tulevina vuosina syntyivät teknologiat ja protokollat, joita vielä nykyäänkin käytetään Internetin toiminnassa. Internet ja Web nykyisessä muodossaan saivat kuitenkin alkunsa vasta vuonna 1989. Euroopan hiukkastutkimuslaitoksella CERNissä Tim Berners-Lee haaveili järjestelmästä, joka hypertekstiä käyttäen yhdistäisi tietovarantoja. Hänen ideoinnistaan syntyi **HTML**-kielenä tunnettu hypertekstimalli, ja vuonna 1991 CERN julkaisi **World Wide Web**-standardin. Moderni Web oli syntynyt. (Suomen internetopas n.d.)

3.2 Web-sovelluksen toiminta

World Wide Web (tai pelkästään Web) on hypertekstijärjestelmä, jota käytetään internetissä lähettämällä pyyntö palvelimelle Webbiä varten nimenomaan luodun standardin, **http**:n (Hyper Text Transfer Protocol) avulla, joka palauttaa takaisin selaimeen html-sivun. (Vainionpää 2005, 8.)

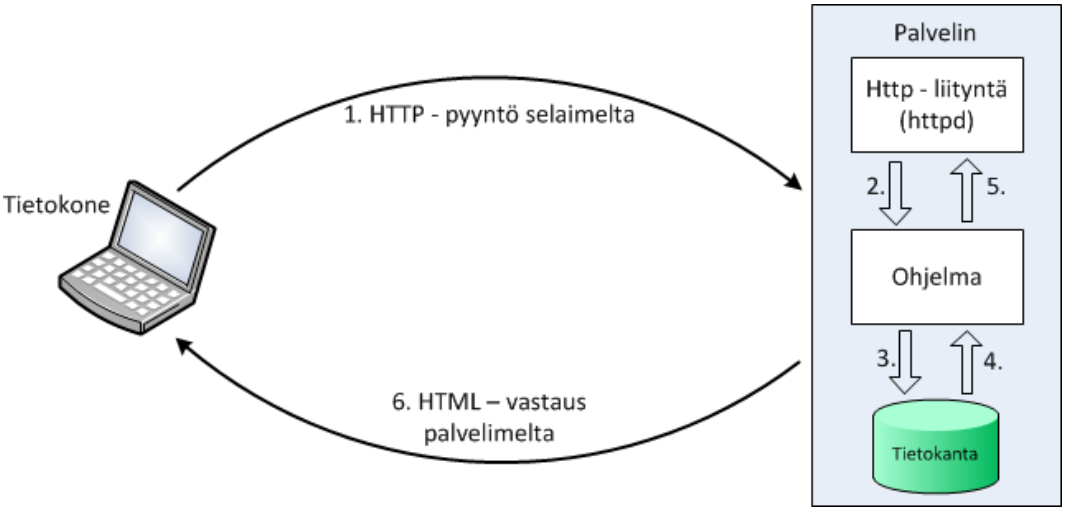
Perinteisen web-sovelluksen toiminta on hyvin pelkistettyä. Selain lähettää palvelimelle pyynnön, ja palvelin palauttaa dokumentin, jonka selain sitten tulostaa näytölle (Kuvio 1). HTML ei ole ohjelmointikieli, vaan sen tarkoituksena on kuvailla sivuston rakennetta tagien avulla kertomalla selaimelle, mikä on esim. otsikko ja mikä on leipätekstiä. Tällaisia muuttumattomia sivuja, jotka palautetaan sellaisinaan palvelimelta, kutsutaan staattisiksi sivuiksi. Koska HTML on kuvailukieli eikä ohjelmointikieli, sen avulla ei voida toteuttaa interaktiivisuutta. (Vainionpää 2005, 8–9.)



Kuvio 1. Staattinen HTML-dokumentti (Mukaillen Vainionpää 2005, 9)

Alkuaikoina staattiset sivut toimivat. HTML-kielen helppouden takia WWW-sivujen määrä ja ulkoasu alkoivat kasvaa ja muuttua. Pian HTML-kielen rajoitteet kuitenkin tulivat vastaan, kun ensimmäiset web-sivut alkoivat kaivata tietojen päivittämistä. Valtaosa sivuista tehtiin uutuuden viehätyksestä, ja niiden päivittämiseen ei oltu valmiita käyttämään resursseja. (Vainionpää 2005, 9–10.)

Toiminnallisuuksia web-sivuihin lähdettiin luomaan palvelinpuolen ohjelmoinnilla. Käytännössä palvelin ottaa vastaan pyynnön, jossa on tietyt käyttäjän valitsemat parametrit. Näiden parametrien perusteella palautetaan vastaus selaimelle (Kuvio 2). Vastauksena tuleva tiedosto on edelleen HTML-tiedosto, mutta tällä kertaa sen sisältöön vaikuttavat käyttäjän tekemät valinnat. (Vainionpää 2005, 10.)



Kuvio 2. Dynaaminen web-sivu (mukaillen Rantala 2005, 4)

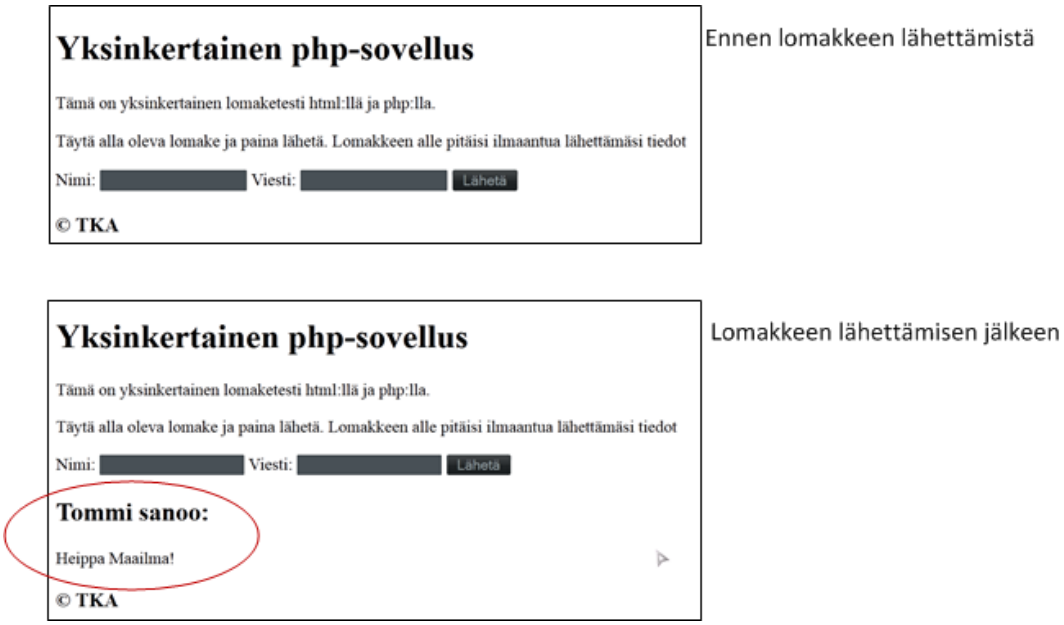
Dynaamisia web-sovelluksia luodaan monilla eri ohjelmointikielillä ja tulkeilla (C ja sen johdannaiset, PHP, Python, Perl, Ruby, JavaScript jne.). Erityisesti suureen suosioon on helppokäyttöisyytensä, ilmaisuutensa ja avoimen lähdekoodin luonteensa takia noussut **PHP**, joka on alustasta riippumaton ja sisältää tuen useimmille käytössä oleville tietokantaratkaisuille, kuten MySQL:lle ja Oraclelle (Vainionpää 2005, 26–29).

Dynaamisten sivujen avulla pystytään luomaan käyttäjien ja sovellusten välistä vuorovaikutusta. Nykyään web-sivujen kautta pystyy tekemään lukuisia jokapäiväisiä toimenpiteitä, kuten hoitamaan pankkiasioita, käydä kauppaa tai lukemaan lehteä. (Vainionpää 2005, 11.)

3.3 Moderni web-sovellus

Vaikka web-sovellus olisikin ohjelmointikielellä tehty dynaaminen sivusto, niin se sisältää silti paljon ongelmia niin käyttäjäystävällisyyden kuin tehokkuudenkin suhteen. Esimerkiksi jokaiselle Webin käyttäjälle on tuttua lomakkeen täyttäminen ja sen lähettäminen. Normaalisti käyttäjä tietää, että hän voi muokata lomaketta niin paljon kuin haluaa ja että sitä ei käsitellä, ennen kuin käyttäjä on painanut ”Lähetä”-nappia. Tämän jälkeen uusi pyyntö lähetetään palvelimelle, joka palauttaa kokonaan uuden sivun käyttäjän selaimeen. (Ballard & Moncur 2008, 41.)

Yllä mainittu käyttöliittymä sisältää kuitenkin joukon ongelmia. Käyttöliittymä, joka on rakennettu kyseisellä mallilla, viivästää sovelluksen toimintaa ja täten käyttäjän omaa ”rytmiä”. Lisäksi kokonainen sivu täytyy ladata uudestaan joka kerta, kun sovellukseen tulee muutos. Usein suurin osa sivun sisällöstä on lähes identtistä edellisen sivun kanssa, joten tämä toimenpide vie turhia resursseja. Kuvio 3 illustroi, kuinka paljon samaa dataa voidaan ladata turhaan. (Ballard ym. 2008, 42.)



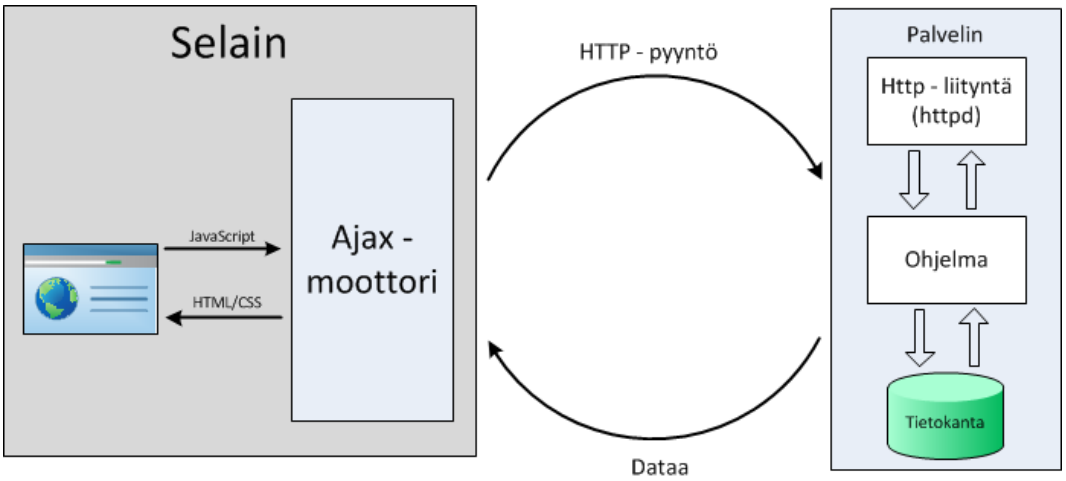
Kuvio 3. Vain punaisella merkitty osio on uutta, ja silti palvelin lähetti kokonaan uuden dokumentin

Web-sivujen toimintaa tehostamaan keksittiin **AJAX**:na tunnettu teknologiakokonaisuus. AJAX lisää uuden kerroksen käyttöliittymään tarjoamaan samanlaisia toiminnallisuuksia kuin työpöytäsovellus, jossa käyttöliittymän osien oletetaan olevan jatkuvasti näkyvillä ja kaikki laskentatoimet jäävät taustalle. (Ballard ym. 2008, 42.)

Ajax ei ole uusi teknologia vaan käyttää vanhaa teknologiaa uudella tavalla ja on yksi Web 2.0-arkkitehtuurin avainkomponenteista. Termi Web 2.0 viittaa muuttuneeseen ajatusmalliin, jossa Web-sovellukset nähdään paljon potentiaalisempina sovelluksina kuin ennen. Vasta viime aikoina WWW-tallennustilan kasvun ja internetyhteyksien nopeutumisen myötä osaa vanhoista teknologioista voidaan käyttää kokonaan uudella tavalla. (Viitanen 2007, 22–24.)

Ajaxin etu on siinä, että se reagoi käyttäjän syötteeseen lähes saman tien, jolloin käyttäjää ei häiritä turhilla ja ärsyttävillä sivunlatauksilla (Viitanen 2007, 26). Esimerkiksi lomakkeiden tietojen tarkistus tapahtuu taustalla ja uutta sivua ei tarvitse ladata kokonaan. Ajax toimii siis taustalla ja mahdollistaa selaimen ja palvelimen välisen kommunikaation asynkronisesti.

Moderni Web-sovellus ei ole enää Kuvio 2 kaltainen palvelimelta hallittava sovellus. Asiakaspään sovellukset mahdollistavat nopeamman ja käyttäjäystävällisemmän käyttöliittymän, ja Ajaxin kaltaisilla teknologioilla voidaan sovelluksen toimintalogiikka siirtää asiakkaan päähän selaimeen (ks. Kuvio 4). Seurauksena palvelimen päähän ei jää kuin tietokantarajapintana toimiminen ja sovelluksen lopullisen tilan tallentaminen. (Kumar 2012.)



Kuvio 4. Palvelimen ja selaimen välinen kommunikointi tapahtuu taustalla

3.4 Sovelluskehykset

Web-ohjelmoinnin perusteiden oppiminen ei ole vaikeaa. Internet on täynnä ohjeita, kirjoja ja tutoriaaleja, joilla ihmiset oppivat tekemään Web-sivuja ja -sovelluksia. Kehittäjien määrän kasvaessa myös koodaustavat ja käytänteet lisääntyvät. Lisäksi uudet kehittäjät oppivat nopeasti parempia keinoja tehdä asioita ja päivittävät vanhoja sivujaan näillä keinoilla. Useimmiten koodaaja ottaa valmista koodia tietystä lähteestä ja lisää sen omaan sovellukseensa. Tämän tapainen kopioi-liitä-toiminta aiheuttaa usein sen, että sovellus muuttuu lähdekoodiltaan sekavaksi ja vaikeaksi käyttää ja lukea. Ohjelmoijien kesken tällainen tilanne tunnetaan nimityksellä spagettikoodi.

Spagettikoodia voi syntyä, jos vanhaa koodia päivitetään usein ja eri paikoista aiheuttaen koodin määrän kasvamista ja hyppelyä eri osaan koodia. Lisäksi tämän tapainen koodaus on erittäin epäluotettava, sillä muutos johonkin kohtaan huonosti organisoitua koodia saattaa aiheuttaa muutoksia muuhun koodiin, mitä ei ennakoitu. Kuvio 5 illustroi potentiaalista spagettikoodia. (Rouse 2008.)

```
<?php
    $viesti = "Moikka maailma";
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Spaghettikoodia</title>
    <script>
        window.onload = function () {
            var otsikko = document.getElementById('otsikko');
            otsikko.style.color = "red";
        }
    </script>
</head>
<body>
    <div class="testi">
        <h1 id="otsikko">Otsikko</h1>
        <p> <?php echo $viesti;  ?> </p>
    </div>
</body>
</html>
```

Kuvio 5. Eri ohjelmointikoodia html:n seassa on tyypillinen spagettikooditilanne.

Ohjelmoinnin tehostamiseksi on kehitetty joukko käytänteitä ja valmista optimoitua koodia, jonka tarkoituksena on tehostaa ohjelmistokehitystä sekä parantaa koodin luettavuutta ja luotettavuutta. Toisin kuin uudelleenkäytettävä koodi tai koodikirjasto, sovelluskehys määrittelee myös sovelluksen kontrollivirran. Samalla se antaa hiukan ohjausta koodauskäytänteistä. Vaikka sovelluskehyksissä on paljon toiminnallisuuksia ja hyvää koodia valmiina, niin useimmat tukevat myös tarvittaessa sitä, että käyttäjä voi soveltaa omaa koodiaan sovelluskehityksessä. (Baker 2009.)

Esimerkkejä web-kehityksessä käytetyistä sovelluskehyksistä ovat mm. Codeigniter, Zend Framework, Yii ja Laravel. Jokainen näistä seuraa tavalla tai toisella MVC:nä tunnettua sääntöä.

MVC on akronyymi sanoista *Model, View ja Controller* (suomeksi vastaavia termejä kutsutaan Malliksi, Näkymäksi ja Ohjaimeksi). Kyseessä on malli, jossa tekninen arkkitehtuuri jaotellaan toimintansa mukaan kolmeen osaan. (Vahtolam 2010.)

Malli on ainoa osa ohjelmasta, joka on yhteydessä tietokantaan. Muut osiot ovat tietokannan kanssa tekemisissä mallin kautta, ja riippuen sovelluksen laajuudesta ja suunnittelusta malleja voi olla useita. Näkymä huolehtii tietojen esittämisestä. Riippuen ratkaisusta näkymä saa näytettävät tiedot joko ohjaimelta tai mallilta ja luo niistä representaation. Ohjain ottaa vastaan sovelluksen pyynnöt ja muokkaa sovelluksen tilaa ja valitsee asiakkaalle näkyvän näkymän. (Vahtolam 2010.)

Web-kehityksen puitteissa voidaan karkeasti jakaa MVC-malli kuviota neljä hyväksi käyttäen seuraavasti:

- Html/css-osio on näkymä (view).
- Ajax-moottori ja osa palvelimen ohjelmaa ovat ohjain (controller).
- Loppuosa ohjelmaa ja tietokanta ovat malli (model).

Web-ympäristössä MVC-malli kulminoituu näkyvästi siihen, mitä kieltä tai teknologiaa kussakin osiossa käytetään.

4 Tietokantaratkaisut

4.1 Mikä on tietokanta?

Kuvio 2 kuvaa dynaamisen Web-sovelluksen toimintaa pelkistetysti. Kuviosta näkee, kuinka sovellus ottaa yhteyden palvelimeen ja palvelimelta otetaan yhteys tietokantaan, joka on yksi tärkeimmistä Web-sovelluksen osista.

Tietokantana voidaan pitää lähes mitä tahansa loogisesti yhteenkuuluvaa ja tallennettavaa tietokokoelmaa. Tietokannalta vaaditaan edellisen määritelmän lisäksi myös kykyä tietojen suojaamiseen, varmistukseen ja tehokkaaseen hallintaan. (Rantala 2005, 252.)

Tiedot tallennetaan tietokantaan lisäämään muutosjoustavuutta, turvaamaan tietojen eheyttä ja helpottamaan sovellusohjelmointia (Hovi, Huotari & Lahdenmäki 2005, 4).

Nykyiset tietokantajärjestelmät ovat valtaosin sql-pohjaisia **relaatiotietokantoja**. Perinteiset tietokantamallit, kuten verkko- tai hierarkiamalliset tietokannat, ovat vaikeampia käyttää, ja tästä syystä relaatiotietokannoilla toteutetaan sekä operatiivisia sovelluksia että tietovarastoja. (Hovi ym. 2005, 5.)

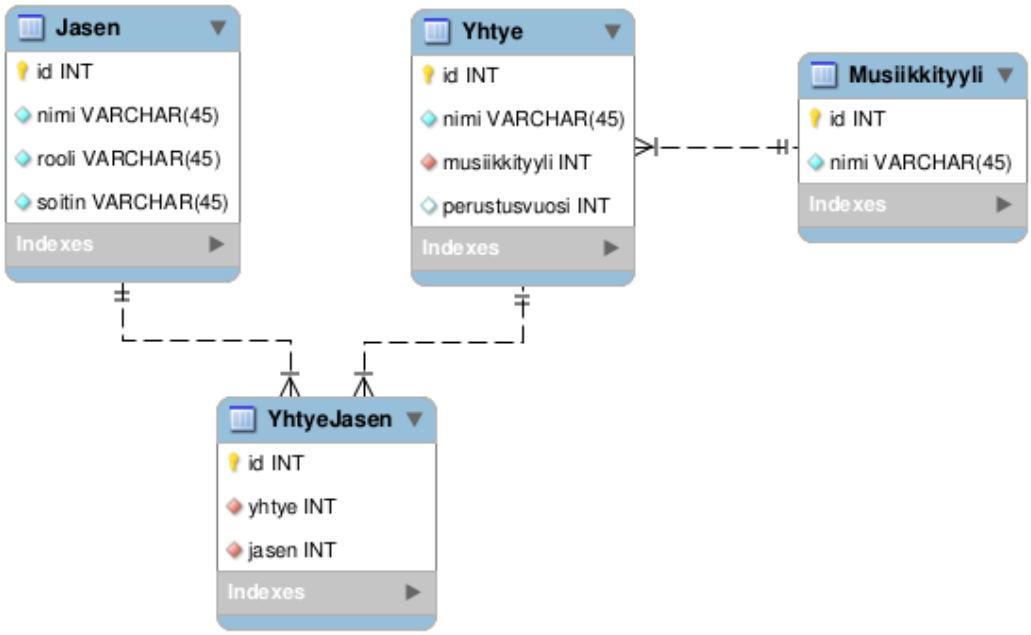
Viime aikoina valtavirtaan ovat nousseet myös ei-relaatiotietokannat, jotka eivät seuraa perinteistä relaatiotietokantamallia (Sierla 2013).

4.2 Relaatiomalli

Relaatiotietokantamalli perustuu joukko-oppiin, matematiikkaan ja predikaattilogiikkaan. Relaatiomalli, jonka IBM:n tutkija E. F. Codd määritteli vuonna 1970, aiheutti vallankumouksen tietokantamaailmassa siihen perustuneiden tietokantatuotteiden syrjäytettyä aiemmin käytetyt verkko- ja hierarkiamallit. (Hovi ym. 2005, 7.)

Relaatiotietokannassa tiedot ryhmitellään omiin yksikköihinsä, joista käytetään termiä taulu. Taulun sisällä tietoja jaotellaan sarakkeilla, joihin tallennetut tiedot kuuluvat omaan arvojoukkoonsa ja tietotyyppiinsä. Itse tiedot, jotka taulujen sarakkeisiin laitetaan, tunnetaan termillä **rivi**. (Hovi ym. 2005, 8.)

Jokaisessa taulussa on uniikki kyseisen taulun tunnistava perusavaimena tunnettu sarake, jonka perusteella muut taulut ja tietojärjestelmät tunnistavat ja viittaavat kyseiseen tauluun. Jos tauluja halutaan yhdistää toisiinsa, niin tällöin käytetään myös viiteavaimia. (Hovi ym. 2005, 9.)



Kuvio 6. Esimerkki relaatiotietokannan taulurakenteesta

Relaatiomallin nerokkuus perustuu joukko-oppiin, jonka mukaan tietoja käsitellään joukko-opillisesti. Taulun riveistä muodostuneisiin joukkoihin voidaan kohdistaa joukko-operaatioita, kuten tietyn kaupungin asukkaiden osoitetietojen haku. Näitä joukko-operaatioita relaatiotietokannoissa suoritetaan SQL-kielellä. (Hovi ym. 2005, 10.)

4.3 Ei-relaatiotietokanta

Tietojärjestelmien käsiteltävän tiedon määrä kasvaa jatkuvasti, ja tämä tiedon määrän kasvu luo uusia haasteita tiedon käsittelyyn, arkistointiin ja analysointiin. Lisäksi tiedon määrän kasvaessa myös sen rakenne muuttuu hajaantuneemmaksi. Tämä tarkoittaa sitä, että relaatiomallin tapaiset perinteiset tietokantaratkaisut eivät enää riitä. Ongelman ratkaisuksi on noussut uusi tietokantamalli, joka keskittyy mm. avain–arvo-, pari-, sarake- ja dokumenttipohjaisiin tietokantaratkaisuihin relaatioiden sijaan. Nämä ratkaisut tunnetaan yhteiseltä nimeltään NoSQL tai ei-relaatiotietokantana. (Tiwari 2011, Introduction.)

Terminä NoSQL ei ole mitenkään uusi keksintö. Päinvastoin NoSQL on vanhempi tietokantaratkaisu kuin relaatiotietokannat, jotka nousivat vallitsevaan asemaansa 1970-luvun jälkeen. Kuten HTML5, NoSQL:kin kasvoi sateenvarjotermiksi sisältämään useita eri teknologioita ja tekniikoita. Nykykontekstissaan NoSQL kuvaa tietokantaratkaisuja, jotka eivät seuraa relaatiotietokantojen asettamaa rakennemallia. (Tiwari 2011, 4.)

Relaatiotietokantojen aiheuttaessa ongelmia suurten tietomäärien (erityisesti skaalautuvuuden) kanssa pari yritystä päätti saaneensa tarpeeksi relaatiotietokantojen ongelmista ja lähti kehittämään omia sovelluksiaan, joiden tarkoitus oli ottaa huomioon suuret tietomäärät ja tietorakenteiden muutokset. Google kehitti valtavalle infrastruktuurilleen (hakukone, google maps, gmail jne.) *Bigtable*-tietokantaratkaisun, ja Amazon kehitti omaan toimintaansa *Dynamon*. Pian Googlen ja Amazonin kehittäjät alkoivat keskustella ja julkaista artikkeleita näistä uusista tietokantaratkaisuista, ja aiheen ympärille syntyi hyvin nopeasti liike. (Fowler 2013.)

NoSQL-tietokantoja on valtava määrä ottaen huomioon teknologian lyhyen iän. Termiä NoSQL on myös erittäin vaikea määritellä, koska se toimii sateenvarjoterminä usealle eri teknologialle. Martin Fowler (2013) määritteli NoSQL-tietokannoille ominaiset piirteet Goto-konferenssissa pitämällään luennollaan seuraavasti:

- Ei-relaatioluontoinen

- Taipumus klusteriystävällisyyteen
- Useimmat avoimen lähdekoodin sovelluksia
- 21 vuosisadan web-teknologiaa
- Skeematon/rakenteeton

NoSQL-tietokannoilla on omanlaisensa tietorakenne, joka pyrkii olemaan vastavirrassa relaatiotietokantojen rakenteiden kanssa.

Avain-Arvo-tietokanta

Yksinkertaisin NoSQL–tietorakenteista on Avain-Arvo-rakenne. Käytännössä se tarkoittaa sitä, että arvo voi olla mitä vain aina yksinkertaisesta numeroarvosta monimutkaiseen dokumenttiin (tietokanta ei tiedä eikä välitä tietää), ja avain toimii yksinkertaisena id-arvona, jonka perusteella tieto haetaan levyltä. (Fowler 2013.)

Dokumentti

Dokumenttimallissa tietokanta olettaa tiedon olevan joukko dokumentteja, jotka sisältävät erilaista tietoa erilaisissa tietorakenteissa. Nykyään tämä tietorakenne on useimmiten **JSON**-muotoa. Dokumenttimuotoisesta tietokannasta voidaan hakea ja muokata kokonaisia tai osia tiedoista, kun taas Avain-Arvo-tietokannassa tietoja käsitellään kokonaisina yksiköinä. (Fowler 2013.)

Sarake

Sarakepohjaisessa tietokannassa tiedot on tallennettu kaksiosaisesti. Ensimmäinen osa on riviavain, joka toimii tavallaan sarakkeen etuliitteenä, ja sen jälkeen annetaan itse sarake, josta halutaan tietoa. Sarake itsessään voi olla millaista tahansa dataa. (Fowler 2013.)

Graafi/Kaavio

Kaaviotietokannat keskittyvät entiteettien välisten suhteiden optimointiin ja hallintaan. Relaatiotietokannat eivät loppujen lopuksi osaa tätä toimenpidettä kovin hyvin (relaatio-sanasta huolimatta). Kaaviotietokanta toimii päinvastoin kuin muut tieto-

kantatyypit NoSQL-perheessä. Muut tietokantatyypit pyrkivät kasaamaan hajanaista dataa yhdeksi helposti hallittavaksi yksiköksi, kun taas kaaviotietokanta pyrkii hajottamaan tiedot entistä pienempiin yksiköihin. (Fowler 2013.)

4.4 Relaatio vai ei-relaatio?

Kuten aiemmista luvuista on selvinnyt, relaatiotietokannat ovat olleet vallitseva tietokantaratkaisu useita vuosikymmeniä. Miksi järjestelmän käyttämiseen kannattaa siis käyttää uutta ei-relaatiotietokantaratkaisua perinteisen relaatiotietokannan sijaan ja millaisissa tilanteissa ei-relaatiotietokanta on parempi ratkaisu?

Ali Gangji vertaili ei-relaatiotietokantoja ja relaatiotietokantoja lyhyesti käyttäen esimerkkinä MySQL:ää ja MongoDB:ta.

Tiedon esitys

MySQL esittää tiedot tauluina ja riveinä. MongoDB taas esittää tiedot kokoelmina JSON-dokumentteja. Tarkemmin tarkasteltuna JSON on syntaksi, jonka kanssa työskennellään sovellustasolla. (Gangji n.d.)

Kyselyt

SQL-kieli (Structured Query Language) on MySQL:n käyttämä kyselykieli, jonka avulla kyselyitä suoritetaan. MongoDB käyttää kyselyissään objektikyselyitä, mikä käytännössä tarkoittaa, että tietokannalle annetaan dokumentti, joka selostaa mitä tietoja halutaan. (Gangji n.d.)

Suhteet

JOIN on relaatiotietokannoissa oleva yksi parhaimmista ominaisuuksista, joka mahdollistaa kyselyiden suorittamisen useaan tauluun samaan aikaan. MongoDB ei tue JOINin tapausta toiminnallisuutta, mutta se tukee usean asteen tietotyyppejä. (Gangji n.d.)

Transaktiot

MySQL tukee atomisia transaktioita, mikä mahdollistaa useiden toimenpiteiden kohtelemisen yhtenä toimenpiteenä. MongoDB ei tue tätä mutta kaikki yksittäiset operaatiot ovat atomisia. (Gangji n.d.)

Skeema

MySQL vaatii taulujen ja sarakkeiden määrittämisen ennen varastointia ja jokaisella rivillä täytyy olla samat sarakkeet. MongoDB:ssä ei määritellä skeemaa, vaan dokumentit voi pudottaa tietokantaan sellaisinaan, ja kahden dokumentin ei edes tarvitse sisältää samaa rakennetta. (Gangji n.d.)

Skeeman suunnittelu ja normalisointi

MySQL:ssä ei ole suuremmin joustavuutta, jos halutaan seurata normalisointistandardeja. MongoDB:ssä skeema tulee optimoida sen mukaisesti miten sovellus hallitsee dataa. (Gangji n.d.)

Suorituskyky

MySQL:ssä suorituskyky riippuu suuresti tietokannan suunnittelusta ja siitä, miten sitä käytetään. Yksinkertainen käsittely (ei ORM-mallia) ja oikeanlainen indeksointi antaa hyvän suorituskyvyn. MongoDB:ssä suorituskyky on relaatiotietokantoja parempi, koska se uhraa JOINin tapaiset toiminnallisuudet. Indeksointi on silti edelleen tarpeen mutta valtaosa sovelluksista eivät luo tarpeeksi dataa, jotta eroa huomaisi. (Gangji n.d.)

Relaatiotietokantojen ja ei-relaatiotietokantojen toiminnassa ja käytössä on siis pieniä ja suuria eroja, kuten myös paljon samankaltaisuuksia.

Ei-relaatiotietokannoilla on monia etuja ylitse relaatiotietokantojen (Gajani 2013):

1. **Ne ovat tauluttomia**

   Ei-relaatiotietokannat tarjoavat helpompaa hallittavuutta ja joustavuutta uudempien tietorakenteiden kanssa.

2. **Avoimen lähdekoodin sovelluksia**

   Valtaosa ei-relaatiotietokannoista on avoimen lähdekoodin sovelluksia, mikä tarkoittaa sitä, että ne ovat ilmaisia käyttää ja täten laskevat dramaattisesti kustannuksia.

3. **Helpompi skaalautuvuus**

   Map Reduce-teknologian avulla ei-relaatiotietokannat tarjoavat elastisen skaalautuvuuden ja tietokantaratkaisut on suunniteltu toimimaan optimaalisesti myös pienemmän kapasiteetin koneissa.

4. **Ei tarvetta yksityiskohtaiselle skeemalle**

   Tietokantojen luonti ei-relaatiotietokannoilla on helppoa ja nopeaa, koska ne eivät vaadi tarkasti määriteltyjä ja hiottuja tietokantamalleja. Tämä säästää kehitysajassa huomattavasti.

Toisaalta ei-relaatiotietokannoilla on myös haittapuolensa (Gajani 2013):

1. **Yhteisön puute**

   Ei-relaatiotietokannat ovat vielä suhteellisen tuoreita, ja niiden ympärille ei ole ehtinyt kehittyä tervettä ja vahvaa yhteisöä.

2. **Raportointityökalujen puute**

   Suorituskyvyn testaukseen ja analyyseihin tarvittavien työkalujen puute on ei-relaatiotietokantojen suurin ongelma.

3. **Ei seuraa ACID-sääntöjä**

   Ei-relaatiotietokannat eivät seuraa tietokannoille yleisesti määriteltyä ACID-standardia eli atomisuuden, eheyden, eristyneisyyden ja pysyvyyden sääntöjä (eng. Atomicity, Consistency, Isolation, Durability).

4. **Standardien puute**

   Ei-relaatietokannoilta puuttuu standardi kyselykieli, joka on tärkeä osa tietokantateollisuuden yhtenäisyyden takia. Voidakseen kasvaa ei-relaatiotietokannat tarvitsevat SQL:n tapaisen standardin kyselykielen.

Loppujen lopuksi ei-relaatiotietokannat eivät ole vielä yhtä kypsiä kuin relaatiotietokannat. Jos sovellus ei ole Googlen, Yahoon, Facebookin tai Wikipedian kaltainen sovellus, niin relaatiotietokanta voi olla parempi ratkaisu.

4.5 Ei-relaatiotietokantojen vertailu

4.5.1 Vertailukohteet

Ei-relaatiotietokantoja on tämän raportin kirjoittamisen aikana listattuna noin. 150 kappaletta (Etusivu N.d nosqldatabases.org). Kattavan vertailun tekeminen on tämän tutkimuksen skaalan ulottumattomissa, joten vertailuun otettiin viiden tietokannan otanta perustuen osittain sattumanvaraisuuteen, suosioon ja arvioihin. Pääsääntöisesti suosio arvioidaan sen perusteella miten tietokanta on arvioitu eri listauksissa ja selaten ns. top ten-listoja.

Listat, joiden perusteella valinnat tehtiin:

- Findthebest lista: http://nosql.findthebest.com/
- 10 standout NoSQL databases to try: http://www.infoworld.com/slideshow/67904/10-standout-nosql-databases-try-203989
- 10 best NoSQL databases: http://www.siliconindia.com/news/enterpriseit/10-Best-NoSQL-Databases-nid-131254-cid-7.html

Näiden perusteella valituiksi tietokannoiksi päätyivät:

- Redis

- Cassandra
- MongoDB
- CouchDB
- Voldemort

Vertailun tarkoitus on antaa korkean tason kuvaus tietokannoista. Vertailussa otetaan kantaa mm. tietokannan luonteeseen (avain-arvo, dokumentti jne.) ja mahdollisiin käyttötapauksiin.

4.5.2 Vertailudata

Suoritetaan käytännön vertailu eri tietokantojen välillä hyvin yksinkertaisella esimerkillä. Esimerkissä luodaan yksittäinen tietokantatietue, käyttäjä, jolla on seuraavat tiedot:

- Id: 1
- Käyttäjätunnus: tommi
- Nimi: Tommi Kaura-Aho
- Ikä: 23
- Salasana: t0mm1

Lisäksi näytetään, miten kyseinen tietue voidaan hakea tietokannasta.

Relaatiotietokannassa (käyttäen esimerkkinä MySQL:ää) edellisessä kappaleessa esitellyt tiedot tarvitsisivat seuraavan skeeman:

```
1 CREATE TABLE users(
2    id INT AUTO_INCREMENT,
3    tunnus VARCHAR(100),
4    nimi VARCHAR(100),
5    ika INT,
6    salasana VARCHAR(255),
7    PRIMARY KEY(id)
8 );
```

Kuvio 7. Tietokantataulun luonti MySQL:ssä

Yllä oleva lause siis luo tietokantaan taulun *users* ja sinne sarakkeet *id, tunnus, nimi, ika* ja *salasana*. Lisäksi rajoitetaan, mitä jokaiseen sarakkeeseen voi tallentaa, kuten

id-sarakkeen INT, joka kertoo, että kyseisessä sarakkeessa voi olla vain kokonaislukuja. PRIMARY KEY kertoo, mikä tieto jokaisessa tietueessa tekee siitä uniikin, ja erottaa sen muista tietueista. Tässä tapauksessa kyseinen tieto on numeerinen kokonaisluku, ja sen lisääminen jokaiseen tietueeseen annetaan tietokannan itsensä tehtäväksi AUTO_INCREMENT-määrityksellä.

Seuraavaksi lisätään uuteen tietokantatauluun esimerkkidataa:

```
INSERT INTO users (id, tunnus, nimi, ika, salasana) VALUES(
    "",
    "tommi",
    "Tommi Kaura-Aho",
    23,
    "t0mm1"
);
```

Kuvio 8. Uuden käyttäjän lisääminen tauluun *users* MySQL:ssä

Huom! Salasanoja ei koskaan kannata tallentaa tietokantaan sellaisenaan, vaan on tärkeää suojata salasanat kryptauksella. Tässä tapauksessa salasana on vain demonstraatiotarkoituksessa, joten kryptausta ei ole käytetty.

Seuraavaksi tietojen haku tietokannasta:

```
SELECT * FROM users WHERE tunnus="tommi";
```

Kuvio 9. Käyttäjän ”tommi” tietojen haku MyQSL:llä

Em. tietokantaskeema ja testidata toimivat siis vertailun pohjana, mutta jokaista tietokantaa ei suoraan verrata tähän pohjaan, vaan kyseessä on enemmänkin pienimuotoinen esittely siitä, miten kyseiset tietokannat saavat saman toiminnallisuuden aikaan.

4.5.3 Vertailu

Redis

Redis on Linuxilla, Macilla ja Windowsilla toimiva Salvatore Sanfilippon vuonna 2009 kehittämä avain-arvo-tietokanta. Se on C:llä ja C++:lla implementoitu nopeasti vaih-

tuvan tiedon käsittelyyn tarkoitettu tietokanta, ja sen suurimpien käyttäjien joukossa on mm. ohjelmoijien yhteisösivusto StackOverflow. Esimerkkikohteita tietokannan käyttöön ovat reaaliaikaisen tiedon hallinta, osakehinnastot ja eri analytics-sovellukset. (NoSQL Find The Best n.d.)

Redisissä ei tarvitse luoda erikseen tietokantaskeemaa kuten relaatiotietokannoista on tuttua, vaan tietokantajärjestelmä osaa tehdä ”taulun” dynaamisesti tiedon laittamisvaiheessa, jos tällaista konstruktiota ei ole vielä olemassa. Tiedon lisääminen Redis-tietokantaan onnistuu seuraavasti:

```
redis 127.0.0.1:6379> set users:tommi "{tunnus:tommi, nimi: Tommi Kaura-Aho, ika:23, salasana: t0mmi}"
OK
redis 127.0.0.1:6379>
```

Kuvio 10. Tietojen lisäys Redis-tietokantaan

Käyttäjän *tommi* tietojen haku Redis tietokannasta:

```
redis 127.0.0.1:6379> get users:tommi
"{tunnus:tommi, nimi: Tommi Kaura-Aho, ika:23, salasana: t0mmi}"
redis 127.0.0.1:6379>
```

Kuvio 11. Käyttäjän *tommi* tietojen haku Redis tietokannasta

Redisin käyttö on siis hyvin yksinkertaista verrattuna relaatiotietokantaan ja sql-kieleen, koska valmiiksi määriteltyä skeemaa ei tarvita ja käyttöliittymän kieli on yksinkertaista. Esimerkissä tietokantaan asetettiin JSON-objekti kuvaamaan tietuetta käyttäjästä, mutta Redis ei vaadi sitä, vaan tietue voi hyvin olla millaista dataa vain.

Cassandra

Cassandra on Apache:n vuonna 2008 kehittämä sarakepohjainen tietokantaratkaisu. Se on yhteensopiva Linuxin, Macin ja Windowsin kanssa ja on kirjoitettu Javalla. Esi-

merkkikäyttökohteita ovat mm. lokien kirjoittaminen, pankki- ja rahoitusjärjestelmät. Suurimpia käyttäjiä ovat Facebook, Twitter ja Digg. (NoSQL Find The Best n.d.)

Aluksi Cassandrassa tulee luoda *keyspace*-niminen osio, jonka vertailun vuoksi voidaan ajatella olevan yksittäinen tietokanta.

```
tommi@tommi-VirtualBox:/opt/cassandra/bin$ ./cassandra-cli
Connected to: "Test Cluster" on 127.0.0.1/9160
Welcome to Cassandra CLI version 2.0.2

Please consider using the more convenient cqlsh instead of CLI
CQL3 is fully backwards compatible with Thrift data; see http://www.datastax.com/dev/blog/thrift-to-cql3

Type 'help;' or '?' for help.
Type 'quit;' or 'exit;' to quit.

[default@unknown] CREATE KEYSPACE demo;
40da6a05-0d29-3155-984f-d74d70693bf7
[default@unknown] USE demo;
Authenticated to keyspace: demo
[default@demo]
```

Kuvio 12. Tietokannan luonti Cassandralla

Cassandra seuraa hyvin vahvasti SQL-kieltä toiminnassaan, eli taulujen luonti tietokantaan onnistuu lähes samalla syntaksilla.

```
cqlsh:demo> CREATE TABLE users(
        ... id int PRIMARY KEY,
        ... tunnus text,
        ... nimi text,
        ... ika int,
        ... salasana text);
cqlsh:demo>
```

Kuvio 13. Cassandra tuo SQL-kielen NoSQL-tietokantaan.

Myös tietojen tallennus ja haku taulusta on tismalleen samaa syntaksia kuin SQL-kielessä.

```
cqlsh:demo> INSERT INTO users(id, tunnus, nimi, ika, salasana)
        ... VALUES(1, 'tommi', 'Tommi Kaura-Aho', 23, 't0mmi');
cqlsh:demo>
cqlsh:demo> SELECT * FROM users;

 id | ika | nimi            | salasana | tunnus
----+-----+-----------------+----------+--------
  1 |  23 | Tommi Kaura-Aho |    t0mmi |  tommi

(1 rows)

cqlsh:demo>
```

Kuvio 14. Tietojen lisäys ja haku Cassandralla

Toisin kuin moni muu ei-relaatietokanta, kuten Redis, Cassandra ei ole luonut omaa syntaksia tietokannan hallitsemiselle, vaan on pyrkinyt tuomaan SQL-kielen osaksi itseään. Relaatiotietokannoista ei-relaatiotietokantoihin tulevalle henkilölle tämä on sekä etu että haitta, koska syntaksin samantapaisuus, vaikka auttaakin omaksumaan tietokannan nopeasti, myös mahdollistaa riskin kohdella Cassandraa kuin relaatiotietokantaa.

MongoDB

Linuxilla, Macilla ja Windowsilla toimiva, MongoDB on 10genin luoma C++:lla kirjoitettu kommenttien varastointiin, CMS-järjestelmiin ja äänestysjärjestelmiin tähdätty dokumenttipohjainen tietokanta. Vuonna 2009 kehitetty MongoDB sisältää jotain relaatiotietokannoista tuttuja piirteitä, kuten indeksoinnin ja kyselyt. Suurimpien käyttäjien joukossa ovat mm. Craigslist, Shutterfly ja Intuit. (NoSQL Find The Best n.d.)

MongoDB:N syntaksi ja toiminta ovat ottaneet vahvoja vaikutteita JavaScriptista, ja tiedot tallennetaankin JSON-muodossa. Mongossa ei ole tarvetta luoda erikseen tietokantoja, tauluja tai muutakaan skeemaa, vaan tietoa voi tallentaa dynaamisesti, ja se voi olla lähes millaista muotoa tahansa. Esimerkiksi uuden käyttäjän tallentaminen tietokantaan tapahtuu *insert()*-funktion avulla.

```
MongoDB shell version: 2.0.4
connecting to: test
> db.users.insert({tunnus: "tommi", nimi: "Tommi Kaura-Aho", ika: 23, salasana: "t0mmi"})
>
```

Kuvio 15. MongoDB:ssä JSON ja JavaScript syntaksi ovat vahvassa asemassa.

Kuten kuviossa 15 näkyy, tiedon tallennus tietokantaan onnistuu hyvin yksinkertaisesti funktioparametrina. Tässä tapauksessa kokoelma (MongoDB:n versio taulusta) users luodaan, jos sitä ei ole vielä olemassa. MongoDB ei kuitenkaan tästä erikseen ilmoita.

Tiedon haku onnistuu lähes yhtä helposti ja samalla periaatteella.

```
> db.users.find({tunnus:"tommi"})
{ "_id" : ObjectId("5277609808b380cb7a8ebfe0"),
"tunnus" : "tommi", "nimi" : "Tommi Kaura-Aho",
"ika" : 23, "salasana" : "t0mmi" }
>
```

Kuvio 16. Tietojen haku MongoDB:ssä on yksinkertainen funktioparametri.

Jos kuvion 16 kyselyä muutettaisiin niin, että ei määriteltäisi tunnusta ehtona, niin find()-funktio hakisi kaikki tiedot users-kokoelmasta.

MongoDB:n syntaksi on yksinkertainen käyttää. Toisinaan jopa liian yksinkertainen, koska koko tietokannan tyhjennyksen voi tehdä yhdellä funktiokutsulla. Samalla periaatteella, jos remove()-funktion kutsuu ilman parametreja, MongoDB tulkitsee sen tarkoittavan kaikkia kokoelman tietoja ja poistaa ne. Käyttäjän tulee olla siis tarkkana sen suhteen, mitä komentoja MongoDB:ssä annetaan.

CouchDB

Toinen Apachen kehityksessä oleva tietokantaratkaisu, CouchDB, on dokumenttipohjainen Erlangilla implementoitu Linuxilla, Macilla ja Windowsilla toimiva tietokanta.

CouchDB on kehitetty vuonna 2005, sen suurin käyttäjä on LostOfWords.com, ja se on tähdätty CRM- ja CMS-järjestelmien käyttöön. (NoSQL Find The Best n.d.)

CouchDB:tä hallinnoidaan suoraan selaimella http:n avulla, ja tieto tallennetaan JSON-dokumentteihin (Apache n.d.). Toisin sanoen CouchDB:stä ei ole terminaalissa ajettavaa käyttöliittymää, vaan tietokantaan otetaan yhteys suoraan eri ohjelmointikielillä.

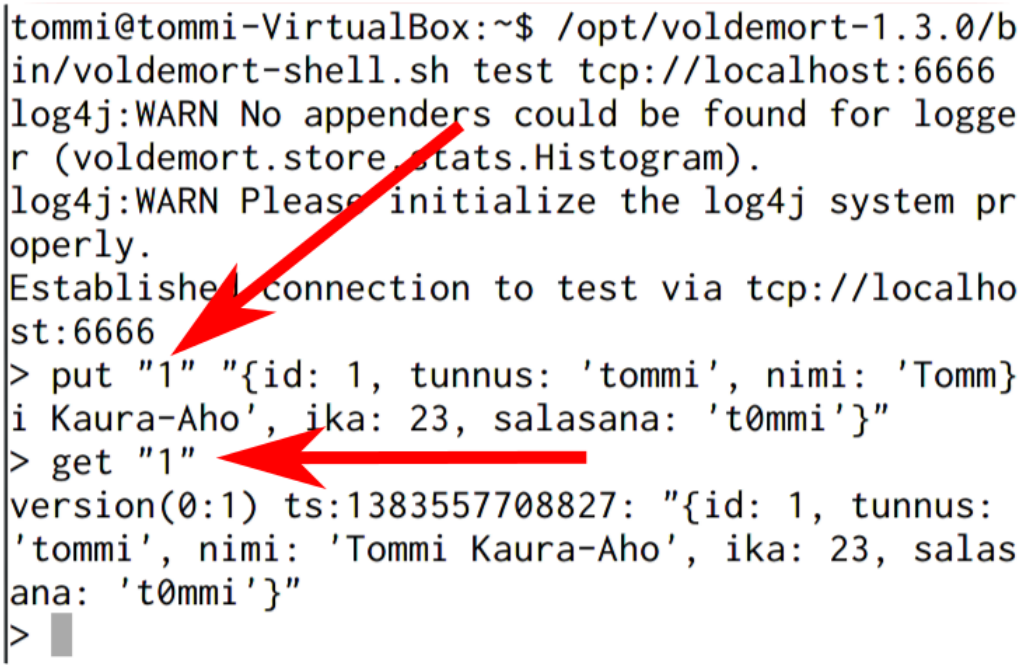
CouchDB:n voisi ajatella olevan frontend-kehittäjän unelmatietokanta, koska se mahdollistaa yhteydenoton tietokantaan suoraan ilman välikäsiä, kuten palvelinpuolen ohjelmointia. On kuitenkin kysymysmerkki, miten tämä vaikuttaa tietoturvaan, jos autentikoinnin tapaiset toiminnallisuudet implementoitaisiin suoraan JavaScriptillä, kielellä, jonka lähdekoodia kuka tahansa sivulla käyvä henkilö voi selaimella lukea.

CouchDB tarjoaa siis luonnostaan sen, mitä tässä projektissa sovelletaan Laravel 4:n avulla eli yhteyden tietokantaan REST-pohjaisen rajapinnan avulla. Kutsut, tallennukset, päivitykset ja poistot tehdään siis käyttäen http:sta tuttuja toiminnallisuuksia kuten GET, POST, PUT ja DELETE-toimintoja.

Voldemort

Voldemort on avoimen lähdekoodin versio Amazonin Dynamo-tietokannasta. Javalla vuonna 2009 kehitetyn reaaliaikaisen tiedon käsittelyyn tarkoitetun tietokannan, Voldemortin, suurimpana käyttäjänä on yhteisöpalvelu LinkedIn. LinkedInin itsensä kehittämä tietokanta sisältää tuen Linuxille, Macille ja Windowsille. (NoSQL Find The Best n.d.)

Voldemort seuraa syntaksiltaan http:tä, vaikka ei CouchDB:n tapaan tarjoakaan suoraa rajapintaa tietokannan käsittelyyn. Koska Voldemort on Redisin tapaan avain-arvo-tietokanta, tietoja voidaan tallentaa lähes missä muodossa tahansa, kunhan avain on määritelty. On kuitenkin luontevaa asettaa tiedolle jonkinasteinen rakenne, vaikka tietokanta ei skeemaa tarvitsekaan, ja JSON on siihen kätevä ja helppolukuinen keino.



Kuvio 17. Voldemortin syntaksi seuraa vahvasti http:ta.

Periaatteessa tallennettu tieto tietokantaan on puhdasta tekstiä, mutta JSON antaa mahdollisuuden luoda rakenteen omaista tietoa. Aivan yhtä hyvin tiedot voisi tallentaa erikseen avaimiin ”tunnus” tai ”nimi”.

5 Toteutus

5.1 Valitut teknologiat ja kirjastot

5.1.1 Backend

Backend koostuu palvelimella ajettavasta sovelluskehyksestä. Koska kyseessä on single-page-application ja selain-pohjainen sovellus, niin backendin tarkoitus on toimia rajapintana tietokannan ja frontendin välillä. Ohjelmointikieleksi valittiin php. Aluksi sovelluskehyksenä käytettiin Yii-nimistä sovelluskehystä, mutta uutuutensa ja selkeämmän koodinsa takia otettiin käyttöön Laravel 4. Lisäksi Laravel 4 sisältää laajemman tuen valitulle NoSQL-tietokannalle kuin Yii.

Koska sovelluksella ei ole tarvetta autentikoinnille tai käyttäjien hallinnalle, backendin rooli rajapintana tietokannan ja frontendin välillä korostuu entisestään. Tässä tapauksessa päätettiin backendistä tehdä **REST**-rajapinta, joka minimoi backendin vaatimuksia ja koodin määrää huomattavasti etenkin Laravelin kaltaisen sovelluskehyksen kanssa (Laravel abstraktoi paljon php-koodia taustalle ja jättää käyttäjän vastuulle vain kaikkein pinnallisimmat toimenpiteet).

REST on kokoelma periaatteita, joiden mukaan http:tä ja URI-osoitteita pitäisi käyttää (Tilkov 2007). Käytännössä se tarkoittaa sitä, että yhteydenpito backendin ja frontendin välillä voidaan käydä käyttäen http-protokollassa jo olevia tekniikoita. Toisin sanoen teknologiaa, jolla haetaan tavallisia html-dokumentteja, voidaan käyttää dynaamisen tiedon hakemiseen ja välittämiseen selaimen ja palvelimen välillä.

5.1.2 Frontend

Frontend koostuu kahdesta osasta: html-koodilla luodusta käyttöliittymästä ja JavaScriptilla kirjoitetusta toimintalogiikasta. Frontend-sovelluskehykset pyrkivät yhdistämään nämä kaksi osiota toisiinsa mutta pitäen ne kuitenkin erillään itse koditasolla. Käytännössä tämä tarkoittaa sitä että käyttöliittymä pyrkii olemaan mahdollisimman paljon html-kieltä.

Käytetyksi sovelluskehykseksi valittiin Angularjs. Angularjs yhdistää html-kieltä ja JavaScript koodia, mutta ei tee näistä spagettikoodia. Yhdistäminen tapahtuu tagien, attribuuttien ja luokkien avulla. Itse toimintalogiikka kirjoitetaan JavaScriptillä käyttäen Angularjs:n moduuleja vähentämään kirjoitetun koodin määrää. Yhteys backendiin hoidetaan REST-kutsuilla.

5.1.3 Ulkoasu

Sovelluksen ulkoasu koostuu css-koodista, jolla myös tavallisten html-sivujen ulkoasu luodaan. Lisäksi selainyhteensopivuuden ja **responsiivisuuden** aikaansaamiseksi käytettiin Bootstrap 3 nimistä css-sovelluskehystä, joka sisältää suuren määrän css- ja JavaScript-koodia. Sen perimmäinen tarkoitus on tarjota valmista ulkoasua, yhdenmukaistaa ulkoasu useille eri selaimille ja tarjota helppo tapa tehdä sovelluksesta responsiivinen eli kykenevä toimimaan ja näkymään myös mobiililaitteilla.

5.1.4 Tietokanta - MongoDB

MongoDB valittiin, koska se oli yksinkertaista asentaa, sen syntaksi on helppolukuista ja sen tietorakenne ja käyttöliittymä seuraavat JSONia ja JavaScriptia.

Tuki Laravel 4-sovelluskehykselle (erillisen lisäosan avulla) mahdollistaa sen, että backendin toteuttamisessa voidaan käyttää Laravelin omaa tapaa hoitaa tietokantayhteydet (ns. **ORM**-malli). Tämä mahdollistaa sen, että parhaimmassa tapauksessa tietokantaratkaisua voidaan vaihtaa tarvittaessa toiseen tietokantaan ilman tarvetta vaihtaa sovelluskehystä.

CouchDB olisi eliminoinut tarpeen Laravel 4:lle, mutta tässä tapauksessa oli luontevampaa ja jatkokehitystä silmällä pitäen joustavampaa käyttää palvelinpuolen sovelluskehystä. Lisäksi sovelluksen yksi vaatimus, integrointi Protaconin HILMA-rajapinnan kanssa, vaati tietynlaista palvelinpuolen toimintaa ja toiminnallisuutta, joka toimii taustalla käyttäjän huomaamatta.

Lisäarvoa MongoDB:n valintaan antoi sen suosio. MongoDB oli ainoa ei-relaatiotietokanta, joka sijoittui kymmenen parhaan joukkoon suosituimpien tietokantojen listalla tämän raportin kirjoittamisen aikana (DB-Engines Ranking 2013).

5.2 Järjestelmän koodaus

Järjestelmän koodaus aloitettiin tutustumalla valittuihin teknologioihin. Laravel 4:n tapauksessa pääpaino oli REST-rajapinnan teon opettelussa, johon löytyivät kattavat ohjeet Laravelin dokumentaatiosta. Lisäksi kokeiltiin tietokantaratkaisujen käyttämistä Laravelin ORM-syntaksilla. Ensimmäisissä kokeiluissa käytettiin tietokantana MySQL-relaatiotietokantaa siihen asti, kunnes MongoDB saatiin integroitua Laraveliin. Tämä toiminto suoritettiin erillisellä kirjastolla, joka tarjoaa Laravelin ORM-mallin käytön MongoDB-ympäristössä.

Kun perehdytys sovelluskehyksiin ja tietokantaan oli saatu tarpeeksi hyvälle tasolle, lähdettiin koodaamaan itse sovellusta. Käytännössä sovelluksesta pyrittiin koodaamaan toiminnallisuudet yksi kerrallaan lähtien liikkeelle käyttöliittymästä ja siirtyen pätkä kerrallaan lähemmäksi tietokantaa.

Käytännössä kaikki toimintalogiikka tapahtui JavaScriptillä, ja backendin ainoa tavoite oli saada tietokannasta data lähetettyä selaimeen ja otettua päivitykset vastaan. Lisäksi backendissa yksi tärkeimmistä toiminnallisuuksista oli tietokannan päivitys järjestelmän ulkopuolisesta rajapinnasta, joka tapahtui taustalla.

5.3 Aikataulutus

Projektin aikataulu oli hyvin korkealla tasolla määritelty. Jokaista yksittäistä tuntia ei esikirjattu ja resursoitu, vaan asetettiin etukäteen takarajat, joiden perusteella määriteltiin eri projektin vaiheet.

Käytännössä projekti aikataulutettiin kuukausittain niin, että elo-syyskuu omistettiin suunnittelulle ja taustatutkimukselle, syys-lokakuu koodaukselle, ja raporttia kirjoitettiin jokaisen vaiheen lomassa. Aikataulutusta määriteltiin, tarkennettiin ja muokattiin perustuen viikoittaiseen palaveriin toimeksiantajan kanssa.

5.4 Kehitysympäristö

Sovelluksen kehitysympäristönä toimi ns. LAMP-nimellä käyvä web-kehityksen paikallinen asennus. LAMP on akronyymi sanoista Linux Apache Mysql Php ja sisältää kyseiset kirjastot ja moduulit. Luonnollisesti joukkoon lisättiin myös MongoDB. Kehitys tapahtui paikallisesti oman tietokoneen sisällä, johon oli Asennettu Arch Linux.

Alkuperäisenä tavoitteena oli myös saada käyttöön viimeistään projektin loppuvaiheilla tuotantoon tarkoitettu palvelinympäristö, johon lopullinen tuotos asennettaisiin ja jossa se testattaisiin. Aikataulun puitteissa tämä suunnitelma hylättiin, ja sovelluksen asennus tuotantoon jätettiin projektista.

Noin puolessavälissä kehitystä otettiin käyttöön työkaluja, joiden tarkoitus oli tehdä kehitystyöstä sujuvampaa. Frontend-kehitykseen tarkoitettu Yeoman, Gruntjs ja Bower ovat kokoelma erilaisia kirjastoja ja työkaluja, jotka tarjoavat mm. seuraavia ominaisuuksia:

- Nopea sovelluksen asennus perustilaan (ns. scaffolding)
- Kolmannen osapuolten kirjastojen lataus ja asennus (esim. Bootstrap)
- Minimointi (uglify) ja virheiden jäljitys (jshint)

Gruntjs on tarkoitettu hoitamaan kehityksessä tiettyjä toistuvia toimenpiteitä, kuten koodin minimoinnin, käyttäjän puolesta. Bower on tarkoitettu asentamaan helposti kolmannen osapuolen kirjastoja, kuten Jquery tai Bootstrap, ilman tarvetta metsästää jokaista kirjastoa erikseen netistä. Yeoman sitoo nämä kaksi yhteen tarjoten työkalun, jolla voidaan luoda sovellusten runkoja.

5.5 Tuotantoympäristöön

Sovelluksen siirto tuotantoympäristöön sisältää useita toimenpiteitä, jotka perustuvat kehityksessä käytettäviin resursseihin. Tässä tapauksessa sovelluksen saaminen tuotantotilaan vaati seuraavia toimenpiteitä:

- JavaScript koodin minimointi ja kommenttien poisto
- CSS-tiedostojen minimointi ja kommenttien poisto
- Kolmannen osapuolen kirjastojen integrointi
- Backend- ja frontend-kirjastojen integrointi

Nämä toimenpiteet suoritettiin Gruntjs:n ja muiden em. työkalujen avulla. Backendin ja frontendin integrointi suoritettiin käsin. Sovellus luovutettiin toimeksiantajalle tuotantovalmiina.

6 Pohdinta

6.1 Ei-relaatiotietokannat

Tämän tutkimuksen alussa tietämykseni ei-relaatiotietokannoista oli rajoittunut pariin oppituntiin yhdellä opintojaksolla ja omiin pieniin kokeiluihin. NoSQL oli termi, josta kuuli ajoittain mutta, joka ei iskostunut tajuntaan minään vallankumouksellisena uutena teknologiana. Tutkimuksen aikana selvisi, että ei-relaatiotietokantoja on valtava määrä. Miten näistä pitäisi sitten valita kaikkein sopivin? Miten valittu relaatiotietokanta käyttäytyy ja miksi valita ei-relaatiotietokanta relaatiotietokannan sijaan?

Relaatiotietokannat ovat olleet vallitseva tietokantajärjestelmä lähemmäs 40 vuotta. Omassa tapauksessani NoSQL oli rajattu pariin oppituntiin ja 2–3 tehtävään yhden opintojakson puitteissa, joka on vapaaehtoinen. Luonnollisesti tämä johtuu relaatiotietokantojen vallitsevasta markkina-asemasta yritysmaailmassa ja olemassa olevissa tietojärjestelmissä.

Ei-relaatiotietokantojen luonne eroaa huomattavasti relaatiotietokannoista. Relaatiotietokannoilla on pitkät perinteet, syvään juurtuneet toimintatavat ja lähes universaali käyttökieli SQL:n muodossa. Esimerkiksi HTML5 tulkitaan sateenvarjotermiksi käsittämään useita eri teknologiakokonaisuuksia. Silti sen on aina koettu olevan termi, joka jatkaa jo olemassa olevia käytänteitä eteenpäin ja ei täten tunnu ”vieraalta”. Tämän tuntuista jatkuvuutta ei vielä ole ei-relaatiotietokantojen muodossa, vaan jo pelkästään nimi NoSQL viittaa siihen, että teknologia on radikaalisti erilainen tuttuun ja turvalliseen relaatiotietokantaan verrattuna.

Ovatko ei-relaatiotietokannat sitten syrjäyttämässä relaatiotietokannat? On totta, että Google:n, Facebookin ja Twitterin kaltaiset valtavat sovellukset tarvitsevat tietokantaratkaisuja, jotka menevät eri suuntaan kuin relaatiotietokannat. Erityisesti skaalautuvuus on aina koettu ongelmaksi relaatiotietokannoissa. Epäilen kuitenkin relaatiotietokantojen totaalista katoamista.

Relaatiotietokannoissa skaalautuvuus on lähes poikkeuksetta tarkoitettu skaalautuvan ylöspäin. Tämä tarkoittaa sitä, että tietokanta toimii yhdessä tietokoneessa, johon asennetaan lisää resursseja tarpeen vaatiessa. Luonnollisesti tämä nostaa esiin ongelmia. Ensinnäkin tietokanta on yhdessä keskitetyssä paikassa, ja täten ongelmatapauksissa koko järjestelmä voi lakata toimimasta. Lisäksi ennemmin tai myöhemmin raja tulee vastaan sen suhteen, miten paljon samaan tietokoneeseen voi lisätä resursseja. Myös kustannukset nousevat ylöspäin skaalattaessa: lisäresurssien osto on aina investointi, joka avaa kukkaron nyörejä.

Ei-relaatiotietokannat on tarkoitettu skaalautumaan myös sivusuunnassa. Tämä tarkoittaa sitä, että yhden suuren koneen sijaan tietokanta ripotellaan pienten koneiden kesken ns. klusteriin. Tämä mahdollistaa sen, että yhden koneen rikkoutuessa koko tietokanta ei ole vaarassa, ja käyttäjä ei käytännössä huomaa vikaa palvelukatkoksena tms.

Relaatiotietokannoilla on kuitenkin etuja, joita ei-relaatiotietokannat eivät ole vielä omaksuneet. Selkeästi suurin etu relaatiotietokannoilla on ikä ja sen tuoma kypsyys. Relaatiotietokannat ovat olleet täällä pitkään. Niitä on käytetty pitkään, ja niillä on laaja käyttäjäryhmä. Lisäksi käyttäjien määrä on mahdollistanut sen, että relaatiotietokannoille on syntynyt standardointi, ja organisaatiot pyrkivät pitämään huolen siitä, että relaatiotietokannat seuraavat tiettyjä yhtäläisiä sääntöjä.

Ei-relaatiotietokannoilla ei ole vielä keskitettyä auktoriteettia, joka voisi hallinnoida ja päättää yhteisistä käytänteistä. Käytännössä ei-relaatiotietokannat ovat samassa tilassa tällä hetkellä kuin eri selaimet olivat muutama vuosi sitten: jokainen tekee, mitä haluaa, ja kukaan ei seuraa standardeja. Jos ei-relaatiotietokannat haluavat samantasoista hyväksyntää kuin relaatiotietokannat niiden täytyy lähteä organisoitumaan ja standardoimaan itseään, sopimaan yhteisistä pelisäännöistä ja pitämään niistä kiinni.

Standardoinnissa ja organisoitumisessa ei-relaatiotietokantojen tapauksessa on kuitenkin ongelmansa. Siinä, missä relaatiotietokannat seuraavat samoja ajatusmalleja ja toiminnallisuuksia joukko-opin ja relaatioiden suhteen, ei-relaatiotietokannat luo-

tiin alun perin ratkaisemaan yksi tietty ongelma, joka oli spesifi tietokannan kehittäneeseen organisaatioon. Se on sekä ei-relaatiotietokantojen voima että heikkous. Luvun 4 vertailussa nähtiin, miten selkeästi eri ei-relaatiotietokannat eroavat toisistaan niin toimintalogiikkansa, käyttökohteidensa kuin syntaksinsakin puolesta. Ainoa syy, miksi ei-relaatiotietokannat nousivat tietoisuuteen, johtui siitä, että yksittäiset kehittäjät lähtivät kirjoittamaan blogeja ja jakamaan ajatuksia omista tietokannoistaan. Tulevaisuudessa voi hyvinkin käydä niin, että ei-relaatiotietokantojen määrä tuotantoympäristöissä pienenee ja aatteen ympärille nousee organisaatioita, jotka pyrkivät standardisoimaan ja yhdenmukaistamaan ei-relaatiotietokantojen käytön.

Koska ei-relaatiotietokannat on tarkoitettu ratkaisemaan spesifejä ongelmia, ne eivät tule syrjäyttämään relaatiotietokantoja. Mitä todennäköisesti tapahtuu, on, että ei-relaatiotietokannat tulevat elämään relaatiotietokantojen rinnalla, ja niitä otetaan käyttöön, kun tietynlainen ongelma syntyy.

Kehittäjän näkökulmasta ei-relaatiotietokannat ovat osuma ja huti. Toisaalta MongoDB:llä kehittäminen oli helppoa, mutta toisaalta se oli liiankin helppoa. Siirtyminen ajatusmaailmassa tilaan, jossa tietokantaan ei käytetä enää kolmasosaa projektin elinkaaresta, on vieras. Relaatiotietokantoihin kouluttautuneena ennalta määrätyn skeeman puuttuminen antaa tietyssä määrin turvattoman tunteen. Toisaalta jos kehittäjää ei kiinnosta, miten tietokanta toimii, kunhan se toimii, niin ei-relaatiotietokannat ovat erinomainen vaihtoehto saada nopeasti kehitys liikkeelle.

MongoDB:n tapauksessa erityisen kiinnostavaa oli Laravel 4:n lisäosien määrä. Useimmiten vähän käytetylle teknologialle tai tekniikalle löytyy yksi tai kaksi lisäosaa, mutta MongoDB:lle löytyi nopealla etsinnällä kolme. Tämä kertoo siitä, että tietokannan suosio on suoraan verrannollinen resurssien määrään ja että sovelluskehyksien kanssa töitä tekevät (oli kyseessä sitten itse sovelluskehyksen tekijät tai ulkopuolinen kehittäjä) haluavat tarjota tuen myös ei-relaatiotietokannoille tai ainakin kaikkein suosituimmille.

Tässä projektissa MongoDB toimi erittäin hyvin. JSON-muotoinen tiedon tallennus mahdollisti integraation useiden rajapintojen välillä huomattavasti pienemmällä

muokkauksella ja esivalmistelulla. MongoDB:n dynaaminen luonne mahdollisti myös poikkeamisen ennalta määrätystä skeemasta tarvittaessa. Silti käytännössä yhtä hyvin toimivan tietokantaratkaisun olisi saanut aikaan myös relaatiotietokannalla ja loppukäyttäjä ei huomaisi eroa. MongoDB:n voima näkyi siinä, että kun tietokanta oli asennettu ja yhteys backendin kanssa hoidettu niin tietokannasta ei tämän jälkeen tarvinnut huolehtia, vaan Laravelin lisäosa abstraktoi paljon manuaalista työtä pois. Sinällään lisäosan käyttö ei olisi ollut välttämätöntä, koska MongoDB:n php-kirjasto on hyvin kattava ja tarjoaa helpon syntaksin, mutta se tarjosi erinomaisen tilaisuuden nähdä, miten hyvin ei-relaatiotietokanta on onnistuttu integroimaan relaatiotietokantoja yleensä käsittelemään tarkoitetun sovelluskehyksen ORM-malliin.

Milloin sitten ei-relaatiotietokanta olisi parempi vaihtoehto? Selkeästi ei-relaatiotietokantojen suurin etu on skaalautuvuudessa. Täten ehkä parhain käyttökohde ei-relaatiotietokannoille on big data ja muut tilanteet, joissa käsiteltävän tiedon määrä on poikkeuksellisen suuri tai poikkeuksellisen monimuotoinen. Esimerkkisovelluksia maailmalta olisivat Googlen valtava sovellusinfrastruktuuri, Twitter ja Facebook. Jos tarve ajaa tietokantaa useissa eri palvelinkokonaisuuksissa ja klustereissa on avainasemassa, ei-relaatiotietokannat toimivat paremmin.

Jos käyttäjäryhmä on pieni ja tietoa ei liiku kummoisesti, relaatiotietokanta on edelleen realistinen vaihtoehto. Relaatiotietokanta osaa myös käsitellä suuria tietomääriä ilman huomattavia hidastuksia, mutta ylläpito mielessä pitäen relaatiotietokannan pitäminen yhdessä koneessa vähentää harmaita hiuksia huomattavasti. Jos tiedetään tiedon rakenne ja se, että sovellus ei tästä tietorakenteesta poikkea, niin relaatiotietokanta on hyvä vaihtoehto, koska se osittain pakottaa käyttäjän ajattelemaan tiedon rakennetta ja määrittelemään sen tarkasti. Tämä tekee sovelluksesta loppujen lopuksi vakaamman.

Käytännössä ei-relaatiotietokannat ovat vielä nuoressa ikäluokassa. Ajan ja käyttökokemusten myötä käyttäjämäärät tulevat lisääntymään ja täten tietokannoilla tulee myös lisää näkyvyyttä. Toisaalta ei-relaatiotietokantojen suhteen näkyvyys koskee vain murto-osaa kaikista 150:stä. Tällä hetkellä ei-relaatiotietokantojen tarjonta on ylittänyt huomattavasti kysynnän ja on suhteellisen pienen piirin tietoisuudessa.

Miten tämä lopulta näkyy markkina-asemassa, jää nähtäväksi, mutta on realistinen hypoteesi todeta, että jos kysyntä ei-relaatiotietokannoille nousee, niin tarjonta supistuu keskittymään muutamaan tietokantaan, jotka voivat täyttää kysynnän vaatimukset.

Tällä hetkellä ei-relaatiotietokannat nähdään luultavasti joko äärimmäisten isojen organisaatioiden projekteina tai pienten piirien kokeiluina. Mainstream NoSQL-hype on vielä alkamatta.

6.2 Laravel ja AngularJS

Laravel 4 php-sovelluskehys, jonka tarkoitus on abstraktoida valtaosa php-koodista, jota backendissä käytetään. Tässä tapauksessa Laravelilla tehtiin REST-rajapinta, johon otettiin yhteys frontendista. Sovelluskehys itsessään on suhteellisen helppolukuinen. Koodin luettavuus on helppoa, ja monimutkaisiakin toiminnallisuuksia voi saada aikaan muutamalla koodirivillä.

AngularJS on frontend-sovelluskehys, jonka tarkoitus on tehdä JavaScriptilla kehittämisestä helpompaa. Se on tarkoitettu single-page-application-sovellusten tekoon ja onnistuu siinä erinomaisesti. Angularin luonne modulaarisena sovelluskehyksenä mahdollistaa erillisten lisätoiminnallisuuksien käytön suhteellisen helposti. Erityisen vaikuttavaa Angularissa on DOM-elementtien suoran manipuloinnin tarpeettomuus valtaosassa toiminnallisuuksia ja sovelluskehyksen tehokkuus. Angularissa on suhteellisen matala oppimiskynnys tiettyyn pisteeseen asti. Tähän pisteeseen johtavat toiminnallisuudet kuitenkin riittävät hyvinkin monimutkaisten sovellusten luontiin.

6.3 Muut työkalut ja kirjastot

Muita työkaluja ja kirjastoja sovelluksen kehityksessä käytettiin toistamaan tiettyjä kehityksen toimintoja. Yeoman-nimisellä työkalulla luotiin sovelluksen templaatti, mihin kuuluivat erilaiset konfigurointitiedostot ja kansiorakenteen luominen. Bowerin avulla sovellukseen lisättiin muita kolmannen osapuolen kirjastoja, kuten Jquery,

Bootstrap ja Angularin eri moduuleja. GruntJS:llä hoidettiin koko sovelluksen pakkaaminen tuotantoympäristöön ja kehityksen aikana mm. selaimen automaattinen päivitys.

6.4 Projektin kulku ja aikataulutus

Projektin kulku seurasi suhteellisen hyvin alkuperäistä suunnitelmaa. Loppupäässä tapahtui pientä venymistä, mutta tämä otettiin huomioon jo alkuperäisessä aikataulutuksessa.

Kaikkia toiminnallisuuksia projektissa ei ajan ja resurssien puutteessa saatu aikaan. Hankkeiden listaus, korvamerkintä ja filtterointi, jotka olivat tärkeimmät toiminnallisuudet, saatiin kuitenkin aikaan, kuten myös integrointi HILMA-rajapinnan kanssa.

Projekti kulki tasaiseen tahtiin, kiitos viikoittaisten palaverien ja demotilaisuuksien. Vaikka palaverit olivatkin lyhyitä (keskimäärin puoli tuntia kestoltaan), niin joka kerta saatiin projektia eteenpäin ja toiminnallisuuksia hiottua. Viikkopalaverit mahdollistivat myös sen, että projekti ei lähtenyt missään vaiheessa sivuraiteille.

Käytännössä projekti lähti suunnitelman ja tiettyjen byrokraattisten toimintojen jälkeen liikkeelle esitutkimuksesta, joka käytännössä käsittää tämän raportin neljä ensimmäistä lukua. Tutkimuksen perusteella valittiin työkalut ja teknologiat, ja näitä lähdettiin testailemaan ja kokeilemaan. Tietyssä kohtaa projektia vaihdettiin yksi teknologia toiseen. Toimeksiantaja antoi kehittäjälle suhteellisen vapaat kädet sen suhteen, mitä sovelluksessa käytettiin mihinkin toiminnallisuuksiin. Pääpaino oli toimivan sovelluksen saamisesta aikaan ja se, että taustalla toimi ei-relaatiotietokanta.

Koodaus tapahtui pyrähdyksissä. Kun yksi vaihe oli saatu valmiiksi, niin etsittiin eri tutoriaaleista ja esimerkkikohteista inspiraatiota seuraavan vaiheen rakentamiseen. Mitään tarkkaa kaaviota tai ennalta määrättyä suunnitelmaa alkuperäisten vaatimusten lisäksi ei ollut.

Testaus jäi vähäisemmälle huomiolle tekniseltä osaltaan (testitapausten kirjaaminen ja luominen, testiympäristö jne.), mutta käytännössä jokaista toiminnallisuutta kokeiltiin useaan kertaan eri parametreilla. Sovellus oli toiminnaltaan pieni, joten end-to-end-testien tekemättä jättäminen ohjelmallisesti ei riskeerannut sovelluksen toimintaa radikaalisti.

6.5 Haasteet, ongelmat ja ratkaisut

Suurimmat haasteet projektissa esiintyivät tuntemattomien teknologioiden muodossa. MongoDB oli täysin uusi teknologia, ja vaikka kokemusta Laravelin tapaisesta sovelluskehyksestä olikin, niin uusi sovelluskehys tarkoittaa uusia toimintatapoja. Laravel on onneksi helppolukuinen ja -käyttöinen sovelluskehys.

Tietokannan integrointi backendin kanssa herätti kysymyksiä, koska suurin osa sovelluskehyksistä ei tue ei-relaatiotietokantoja natiivisti. Tämä ongelma ratkesi erillisellä lisäosalla, joka tarjosi Laravelin ORM-kartoituksen tietokannan kanssa, ja mahdollisti MongoDB-spesifin syntaksin abstraktoinnin.

AngularJS ja frontend-kehitys olivat uusia konsepteja, ja meni hetki totuttautua ajatusmalliin, että kaikki tapahtuu yhdessä tiedostossa ja että sovelluksen päätoimintalogiikka implementoitiin JavaScriptillä, joka on aina vaikuttanut tietoturvallisesti kyseenalaiselta kieleltä, koska kenellä tahansa on pääsy sovelluksen lähdekoodiin.

Erityisen ongelman nosti esiin Angularin ja Laravelin integrointi keskenään. Laravelin tarkoitus oli toimia REST-rajapintana, ja täyden sovelluskehyksen käyttäminen tähän vaikutti vieraalta konseptilta. Lisäksi ongelmia tuotti frontend-koodin mahdollinen sijoituspaikka. Loppujen lopuksi tämä ratkaistiin pyrkimällä pitämään frontend-koodi samassa paikassa huolimatta siitä ovatko kyseessä tyylit tai JavaScript-koodi.

Frontend-kehitys itsessään oli moderniudessaan vieras. Enää ei kirjoitettu pitkää rimpsua koodia, lähetetty sitä palvelimelle ja päivitetty selainta, vaan koodia muokattiin reaaliajassa, ajettiin minimointi ja syntaksi tarkistustyökalujen läpi ja selain päivittyi automaattisesti.

Loppujen lopuksi projekti saatiin sellaiseen vaiheeseen, että sen pystyi luovuttamaan toimeksiantajalle sellaisessa muodossa, että siitä on toimeksiantajalle hyötyä. Sovelluksen lopullinen siirto tuotantoympäristöön jäi toimeksiantajan vastuulle.

6.6 Jatkokehitys

Luodussa sovelluksessa on potentiaalia jatkokehitykselle. Yksi toiminnallinen idea on laajentaa sovellusta käsittämään käyttäjien hallinnan. Suurempi valinnanvapaus sen suhteen, mitä tietoja yksittäisistä hankkeista käyttäjä haluaa nähdä, olisi erinomainen toiminto.

Ulkoasun suhteen muutoksia pitäisi saada admin-puolen nappien määrään. Lisäksi sovellus ei ole täysin responsiivinen, vaan tuki mobiililaitteille tuli puolivahingossa. Siirtymät eri osa-alueista toiseen kaipaisivat värikkyyttä mm. animaatioiden muodossa. Tämä auttaisi myös käytettävyyttä, kun sovelluksessa olisi selkeä siirtymä paikasta toiseen ja käyttäjän ajatus ei katkeaisi näkymän muuttuessa yht äkkiä toiseksi.

Luvun 4 vertailu ei-relaatiotietokannoista oli hyvin pelkistetty ja pinnallinen. Syvällinen käytännön vertailu suuremmalla otoksella, ja yksityiskohtaisemmalla vertailupatteristolla, mahdollistaa huomattavasti monipuolisemman analyysin ja antaa tarkempia viitteitä siihen, mikä ei-relaatiotietokanta sopii mihinkin ympäristöön.

6.7 Mielipide projektista

Projekti oli luonteeltaan kiinnostava ja antoi tilaisuuden opetella useita eri kehitysmenetelmiä, kirjastoja ja sovelluskehyksiä. Projektin aikana oma kehitysprosessi hioutui tehokkaammaksi ja kynnys ottaa käyttöön monimutkaisiakin kirjastoja tehostamaan frontend-kehitystä madaltui.

Vaikka sovellus ei ollut teknisiltä vaatimuksiltaan kovin laaja, se antoi mukavasti haastetta eri sovelluskehyksen osa-alueista. Kokemusta tuli koodaamisen ja siihen

liittyvien kehitystyökalujen ja koodikirjastojen käytön lisäksi käyttöliittymän rakentamisesta, eri teknologioiden integraatiosta ja priorisoinnista.

Suurimmat muutokset projekti vaati sen suhteen, miten tietokantaa ajateltiin. Tietokantaskeema tuli tässä tapauksessa Protaconin HILMA-rajapinnasta saapuneen datan mukaan. Koska MongoDB seurasi samaa syntaksia, kuin haettava data niin tietojen muokkaus oli minimissään. Tämä tietokantaskeeman puute oli aluksi vaikea ymmärtää, ja oli vaikea hyväksyä, että tietokannan voisi saada toimintakuntoon käytännössä murto-osassa sitä aikaa, mikä relaatiotietokannan tekoon olisi kulunut. Relaatiomalliin tottuneelle ei-relaatiotietokanta on vaikea käsittää, koska se on suunniteltu syntaksiltaan ja toiminnaltaan huomattavasti käyttäjäystävällisemmäksi. Tämä on toisaalta myös heikkous, koska se mahdollistaa laiskemman kehityksen ja tietokannan roolin aliarvioimisen.

Loppujen lopuksi tilaisuus tutkia ei-relaatiotietokantoja tuotantoympäristöön tarkoitetun projektin puitteissa oli palkitseva, ja tieto siitä, että tehty työ tulee hyödyksi, on aina positiivista. Lisäksi tämä projekti antoi myös korvaamatonta kokemusta frontend-kehityksestä, integraatiosta ja henkilökohtaisista projektinhallintamenetelmistä.

Lähteet

Apache. N.d. A Database for the Web. Viitattu 3.11.2013.
http://couchdb.apache.org/

Baker, M. 2009. What is a Software Framework? And why should you like 'em? Viitattu 2.9.2013.
http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em

Ballard, P. & Moncur, M. 2008. Sams Teach Yourself: Ajax, JavaScript and PHP. USA: Sams Publishing.

DB-Engines Ranking. N.d. DB-Engines. Viitattu 3.11.2013.
http://db-engines.com/en/ranking

Dorsey, P & Rosenblum M. N.d. Knowing Just Enough about Relational Databases. Viitattu 3.9.2013.
http://www.dummies.com/how-to/content/knowing-just-enough-about-relational-databases.html

Etusivu. N.d. HILMA:n etusivu. Viitattu 21.8.2013.
http://www.hankintailmoitukset.fi/fi/

Etusivu. N.d. nosql-databases.org. Viitattu 1.11.2013.
http://nosql-database.org/

Fowler, M. 2013. Introduction to NoSQL. Goto-konferenssiluento. Youtube. Viitattu 3.9.2013.
http://www.youtube.com/watch?v=qI_g07C_Q5I

Gajani, A. 2013., The key differences between MySQL and NoSQL DBs. Viitattu 15.10.2013.
http://blog.monitor.us/2013/05/cc-in-review-the-key-differences-between-mysql-and-nosql-dbs/

Gangji, A. N.d. MySQL vs. MongoDB: Relational and Non-Relational Databases. Viitattu 15.10.2013.
http://www.neonrain.com/blog/mysql-vs-mongodb-relational-and-non-relational-databases

Gromov, G. 1995. Viimeisin muokkaus 2012. Roads And Crossroads Of the Internet History. Viitattu 28.8.2013.
http://history-of-internet.com/history_of_internet.pdf

Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu & indeksointi. Docendo.

Kumar, K. S. 2012. Architecture of a Modern Web Application. Viitattu 2.9.2013.
http://www.techiekernel.com/2012/12/architecture-of-modern-web-application.html

Laki Julkisista hankinnoista 348/2007. Viitattu 19.8.2013.
http://www.finlex.fi/fi/laki/ajantasa/2007/20070348

NoSQL Find The Best. N.d. Viitattu 1.11.2013.
http://nosql.findthebest.com/

Suomen Internetopas. N.d. WWW-standardin synty. Viitattu 28.8.2013.
http://www.internetopas.com/historia/www/

Rantala, A. 2005. Web-ohjelmointi. Docendo.

Rouse, M. 2008. Spaghetti code. Viitattu 2.9.2013.
http://searchcio-midmarket.techtarget.com/definition/spaghetti-code

Sequin, K. 2012. The Little Redis Book. Viitattu 2.11.2013.
http://openmymind.net/redis.pdf

Sierla, A. 2013. Tieto talteen tarkoituksenmukaisesti. Viitattu 2.9.2013.
http://gofore.com/tag/nosql-tietokanta/

Tilkov, S. 2007. A Brief Introduction to REST
http://www.infoq.com/articles/rest-introduction

Tiwari, S. 2011. Professional NoSQL. Wrox. USA. Viitattu 3.9.2013.
http://site.ebrary.com.ezproxy.jamk.fi:2048/lib/jypoly/docDetail.action?docID=10494715

Vahtolam. 2010. MVC-Malli, peruskauraa frameworkkien käyttäjille. Viitattu 21.10.2013.

http://vahtolam.wordpress.com/2010/09/23/mvc-malli-peruskauraa-frameworkkien-kayttajille/

Vainionpää, V. 2005. Näkökohtia tehokkaan web-sovelluksen suunnitteluun. Viitattu 31.8.2013.
http://www.metropolia.fi/fileadmin/user_upload/Julkaisutoiminta/Julkaisusarjat/Opinnaytetyot/PDF/Opinnaytetyoet_5_verkkojulkaisu.pdf

Viitanen, E. 2007. Ajax-tekniikka. Opinnäytetyö. Lahden ammattikorkeakoulu. Viitattu 2.9.2013.
http://publications.theseus.fi/bitstream/handle/10024/11932/2008-03-17-04.pdf?sequence=1

Yritys. N.d. Protaconin sivut. Viitattu 29.10.2013
http://www.protacon.com/protacon/