

Sampo Kataja

# JavaScript-sisällönhallintajärjestelmät

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinööriytyö

3.4.2016

Tekijä Otsikko	Sampo Kataja JavaScript-sisällönhallintajärjestelmät
Sivumäärä Aika	21 sivua 3.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Lehtori Ilkka Kylmäniemi
<p>Insinööriyön tarkoituksena oli tutkia JavaScript-ohjelmointikieleen pohjautuvia sisällönhallintajärjestelmiä ja toteuttaa sovellus yhdellä käytetyimmistä JavaScript-pohjaisista hallintajärjestelmistä. Tämän lisäksi pyrkimyksenä oli käydä läpi kaikki pääteknologiat, joita JS-julkaisujärjestelmät yleensä vaativat toimiakseen.</p> <p>Esimerkkisovelluksen tarkoituksena oli pitkälti muodostaa kuva siitä, ovatko JS-pohjaiset hallintajärjestelmät käytettäviä ja onko niillä mahdollisuutta saada jalansijaa markkinoilla tulevaisuudessa.</p> <p>Esimerkkisovelluksesta kehittäessä selvisi, etteivät JS-pohjaiset hallintajärjestelmät ole vielä tarpeeksi pitkälle vietyjä ja kehittyneitä, jotta ne voisi tosissaan haastaa PHP-pohjaiset markkinoilla.</p> <p>Esimerkkisovelluksessa käytettiin seuraavia teknologioita: Node.js, Express, MongoDB, Jade, JavaScript. Lopputuloksena oli palvelu, jonka pääfunktionaalisuus on käyttäjälle lähimpien avoimien työpaikkojen listaus.</p> <p>Insinööriyö opetti työn tekijälle paljon uutta käytetyistä teknologioista ja harjaannutti MVC-arkkitehtuurin käyttöä. Työssä havaittiin JavaScript-pohjaisten sisällönhallintajärjestelmien olevan vielä liian kehittymättömiä kaupalliselle puolelle.</p>	
Avainsanat	JavaScript, Node.js, KeystoneJS, MongoDB, sisällönhallintajärjestelmä

Author Title	Sampo Kataja JavaScript content management systems
Number of Pages Date	21 pages 3 April 2016
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Ilkka Kylmäniemi, Senior Lecturer
<p>The purpose of this final year project was to study JavaScript content management systems and create an example application using one of the most used JavaScript content management systems. Another purpose was to go through all main technologies which the systems are based.</p> <p>The main purpose of the example application was to outline if JS content management systems are usable and if they have any possibility to gain foothold in the industry of WEB development.</p> <p>The following technologies were used in the example application: Node.js, Express, MongoDB, Jade and JavaScript. The outcome was a service, the main functionality of which is listing the nearest open positions to its user.</p> <p>The thesis provides valuable new information about the theories on which the used technologies are based. It also offers information on the use of MVC architecture. The thesis concludes that JavaScript content management systems are still too elementary to be used in the commercial industry.</p>	
Keywords	JavaScript, Node.js, KeystoneJS, MongoDB, content management system

## Sisällys

### Lyhenteet

1	Johdanto	1
2	JavaScript-sisällönhallintajärjestelmät	2
2.1	KeystoneJS-julkaisualusta	2
2.2	Ghost-blogialusta	3
2.3	JavaScript- ja PHP-sisällönhallintajärjestelmät	3
3	Käytettävät tekniikat ja teknologiat	5
3.1	Node.js-kehitysympäristö	5
3.2	Express-palvelinohjelmistokehys	7
3.3	NoSQL-tietokanta	7
3.4	MVC-arkkitehtuuri	11
3.5	Jade-verkkomallinemoottori	12
4	Työnhakupalvelusovellus	13
4.1	Lähtökohdat ja suunnittelu	13
4.2	KeystoneJS:n asennus	14
4.3	Käyttäjien luonti KeystoneJS:llä	15
4.4	Tarvittavan datan mallinnus tietokannassa	16
4.5	Algoritmi lähimmän avoimen työpaikan löytämiseen	19
5	Yhteenveto	21
	Lähteet	22

## Lyhenteet

JS	JavaScript, ohjelmointikieli, jota sovelletaan pääasiassa verkko-ympäristöissä.
PHP	Hypertext Preprocessor, ohjelmointikieli, jota sovelletaan pääasiassa verkkosivujen palvelinpäässä.
NPM	Node Package Manager: vakio pakkaustenhallintajärjestelmä JavaScriptille.
MVC	Model-View-Controller, ohjelmistoarkkitehtuurityyppi, joka erottaa mallin, näkymän ja käsittelijän logiikat toisistaan.
SSH	Secure Shell: salattuun tietoliikenteeseen tarkoitettu protokolla.
Python	Monipuolinen tulkattava ohjelmointikieli, soveltuu mm. palvelinpään ohjelmointikieleksi.
Ruby	Tulkattava, dynaaminen ja pelkästään olio-ohjelmointiin perustuva ohjelmointikieli.
HTML	Hypertext Markup Language, verkkosivujen luomiseen tarkoitettu merkin-täkieli.
CSS	Cascading Style Sheets, verkkosivuille kehitetty merkin-täkieli, joka määrittää dokumentin tyylin.
HTTP	Hypertext transfer protocol: protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.
JSON	JavaScript Object Notation, tiedonvälitykseen tarkoitettu tallennusmuoto, jota käytetään lähinnä palvelimen ja sovelluksen väliseen tiedonkulkuun.
SQL	Structured query language, kieli relaatiotietokantojen hallintaan ja muok-kaamiseen.

URL Uniform Resource locator, käytetään osoittamaan WWW-sivuja.

## 1 Johdanto

Insinööriyössä perehdytään JavaScript-ohjelmointikieleen pohjautuviin sisällönhallintajärjestelmiin ja toteutetaan työnhakupalvelu käyttäen KeystoneJS-sisällönhallintajärjestelmää. Sisällönhallintajärjestelmällä tarkoitetaan järjestelmää, jonka avulla käyttäjä voi hallinnoida verkkopalvelunsa koko sisältöä lisäten, poistaen sekä muokaten sitä. [2.]

PHP-ohjelmointikielellä rakennetut sisällönhallintajärjestelmät ovat pitkään hallinneet alan markkinoita. JavaScript on kuitenkin ohjelmointikielenä nosteessa; se on juurruttanut itsensä moderneihin selaimiin, monimutkaisiin verkkosovelluksiin, palvelinpuolen ohjelmointiin sekä laitteisiin ja koneisiin. Erilaisia JavaScript-rajapalveluita kehitetään jatkuvasti lisää, ja näin se saa enemmän jalansijaa verkkokehityksen maailmassa. Puhtaasti JavaScript-pohjaiset sisällönhallintajärjestelmät eivät ole kuitenkaan vielä kovin käytettyjä PHP:n suosion vuoksi. On kuitenkin huomioitava, että PHP-pohjaiset sisällönhallintajärjestelmät ovat nykyään lähes yhtä riippuvaisia JavaScriptistä kuin ne ovat PHP:stä. [3.]

Insinööriyön tavoitteena on tutkia JavaScript-sisällönhallintajärjestelmissä käytettyjä tekniikoita ja teknologioita lähteiden ja käytännön toteutuksen kautta ja täten muodostaa kuva, onko puhtaasti JavaScriptiin pohjautuvilla sisällönhallintajärjestelmillä tulevaisuutta.

Työn alussa tutustutaan kahteen käytetyimpään JavaScript-sisällönhallintajärjestelmään ja verrataan PHP- ja JavaScript-pohjaisia sisällönhallintajärjestelmiä keskenään. Tämän jälkeen perehdytään yleisimpiin tekniikkoihin ja teknologioihin, jotka ovat myös käytössä työnhakupalvelusovelluksessa. Ennen yhteenvetoa käydään käytännön toteutus läpi pääkohdiltaan ja analysoidaan lopullinen toteutus.

## 2 JavaScript-sisällönhallintajärjestelmät

### 2.1 KeystoneJS-julkaisualusta

KeystoneJS on avoimen lähdekoodin julkaisujärjestelmä, joka on rakennettu lähinnä verkkokehittäjien käyttöä varten. Toisin sanoen se vaatii käyttäjältään jonkin verran JavaScript-kielen osaamista. KeystoneJS käyttää Express- ja MongoDB-teknologioita, joten niiden asentaminen on ennakkoon pakollista [5.]

Uuden KeystoneJS-projektin aloittaminen vaatii yhden datamallin lisäämistä ja säätämistä. Datamallilla tarkoitetaan jäsennellyä tietorakennetta. Datamalleja voidaan kuvata monella eri tapaa. Ne voivat olla esimerkiksi graafisia diagrammeja tai tietojenkäsittelyssä käytettyjä tietokantarakenteita. Niillä on kaksi päätarkoitusta: Kuvastaa missä muodossa tietoa tallennetaan tai käsitellään, tai toimia itse rakenteena. KeystoneJS-projektin aloittamisen yhteydessä datamallilla tarkoitetaan JSON-objektia, joka määrittää, mitkä tiedot ja minkälaisella rakenteella tieto tulee tallentaa tietokantaan. Tämän jälkeen pääkäyttäjän hallintapaneeli on käytettävissä. KeystoneJS:a ei pysty asentamaan selaimen kautta, vaan se vaatii komentorivin käyttöä. Tähän vaaditaan Yeomanilla tehtyä KeystoneJS-generaattoria. Yeoman on verkkotyökalu, joka on kirjoitettu Node.js:lla. Yeomanin avulla pystytään muun muassa luomaan uuden projektin kansiorakenteet yhdellä komentorivikomennolla. [1;4.]

Hallintapaneelin toiminnallisuuksien lisäksi KeystoneJS tarjoaa seuraavat tekniikat:

- mukautetut sisältötyypit
- sessioiden hallinta ja todennus.

Se tarjoaa integraatiot seuraaviin palveluihin:

- Cloudinary – mahdollistaa kuvien syötön, varastoimisen ja kuvakokojen muuttamisen
- Mandrill – mahdollistaa sähköpostien lähettämisen
- Google Place – mahdollistaa lokaatiokenttien käytön
- Embedly – mahdollistaa videoiden ja Rich Media-ratkaisujen upottamisen.



KeystoneJS on suhteellisen tuore sovellusalusta, ensimmäinen julkaisu (versio 0.0.1) on kesäkuulta vuonna 2013. [4.]

## 2.2 Ghost-blogialusta

Ghost on avoimen lähdekoodin blogialusta, joka on tällä hetkellä käytetyin Node.js:iin pohjautuva sisällönhallintajärjestelmä. Ghost tarjoaa kaksi eri versiota, yksi kaikille ja yksi kehittäjille. Ghost sai alkunsa onnistuneen Kickstarter-kampanjan ansiosta. Tämä kampanja järjestettiin keväällä 2013. Ghostin tarjoaa voittoa tavoittelematon säätiö (Ghost foundation), joka pitää liikevaihtokirjanpitoa avoimena kaikille. Rahoituksensa säätiö saa lahjoittajilta. [7.]

Ghost on tarkoitettu lähinnä vain blogialustaksi, joten monimutkaisempia sivustoja ja selainsovelluksia sillä ei kannata lähteä toteuttamaan. Poiketen monista muista JavaScript-julkaisualustoista Ghost tarjoaa käyttäjien tekemiä teemoja ostettavaksi. Joitain ilmaisia-kin on. [6.]

Ghost suosittelee kolmea vaihtoehtoa palveluntarjoajaksi siinä vaiheessa, kun sivuston julkaiseminen on ajankohtaista:

- Bitnami
- Rackspace deployments
- DigitalOcean Droplet.

Ghost on myös mahdollista asentaa manuaalisesti käyttäjän itse valitsemaa palveluntarjoajaa käyttäen. Valitettavan moni Node.js:a tukevista palveluntarjoajista ei kuitenkaan ole yhteensopiva Ghostin kanssa, joten tästä syystä kannattaa valita jokin kolmesta suositellusta vaihtoehdosta. [7.]

## 2.3 JavaScript- ja PHP-sisällönhallintajärjestelmät

PHP-sisällönhallintajärjestelmät ovat tällä hetkellä vielä ylivoimaisesti käytetympiä kuin JavaScript-pohjaiset. Tämän vastakkainasettelun tarkoituksena on lähinnä kartoittaa

syyt, miksi JavaScript-pohjaiset julkaisujärjestelmät saattavat (tai eivät saata) nousta tulevaisuudessa PHP:n rinnalle käyttäjämäärissä.

PHP-sisällönhallintajärjestelmät pohjautuvat palvelinpuolella nimensä mukaisesti PHP:hen, JavaScript järjestelmät taas Node.js:iin. PHP on Node.js:a paljon vanhempi, ja siksi mahdollisissa ongelmatilanteissa PHP tarjoaa huomattavasti paremman tuen ja avun. Molemmat toki tarjoavat omat dokumentaatiot, mutta 20 vuoden PHP:n olemassaolon aikana tämä dokumentaatio on paljon Node.js:n dokumentaatiota kattavampi.

JavaScript-pohjaisissa järjestelmissä on se huomattava etu, ettei kehittäjän tarvitse vaihtaa kieltä palvelinpuolen ja loppukäyttäjäpuolen välillä – tarvitsee vain osata JavaScriptia. PHP-pohjaisissa järjestelmissä taas JavaScript hyvin yleisesti hoitaa loppukäyttäjäpuolen, joten kehittäjän on vaihdettava tapaansa ajatella, sillä näissä kielissä on eronsa.

JavaScriptille kehitetään jatkuvasti enemmän työkaluja, jotka parantavat työnkulkua. Yksi parhaista Node.js-työkaluista on NPM-pakkaushallintajärjestelmä (engl. the Node Package Manager). NPM:n avulla kehittäjä pystyy nopeasti asentamaan ja hallitsemaan kehityksessä vaadittavia paketteja. PHP:lle on vastikään rakennettu samankaltainen työkalu nimeltä Composer, mutta se ei tule PHP:n mukana automaattisesti, eikä sen käyttö ole kovin yleistä.

Tällä hetkellä integraatioiden rakentaminen PHP-pohjaisilla sisällönhallintajärjestelmillä on helpompaa ja rajattomampaa kuin Node.js:lla. Node.js kehittyy integraatioiden saralla jatkuvasti, mutta on vielä PHP:ta jäljessä. Vanhempiin rajapintapalveluihin integrointi saattaa olla vielä mahdotonta Node.js:lla, sillä ne eivät tarjoa tukea Node.js:lle.

PHP vie myös voiton palvelintarjoajaa etsiessä. Suurin osa palvelintarjoajista tarjoaa tukea PHP:lle. Node.js vaatii huomattavasti enemmän konfigurointia toimiakseen. Tästä syystä palveluntarjoajat jättävät usein Node.js-tuen tarjoamatta. Node.js vaatii myös SSH-oikeuden palvelinkansiorakenteen juureen. Tällä oikeudella ulkopuolinen käyttäjä pystyy kaatamaan koko palvelun. Tämän vuoksi palveluntarjoajat eivät myöskään kovin herkästi Node.js:lle tukeaan tarjoa.

PHP-pohjaisten julkaisujärjestelmien suuresta suosiosta johtuen niille rakennetaan jatkuvasti enemmän lisäosia, jotka helpottavat sivuston kehittämistä. Tämä on suurin syy, miksi PHP-pohjaiset järjestelmät ovat niin suosittuja. [9.]

### 3 Käytettävät tekniikat ja teknologiat

#### 3.1 Node.js-kehitysympäristö

Node.js on avoimen lähdekoodin alustariippumaton palvelinpuolen kehitysympäristö, joka on kirjoitettu JavaScriptilla. Node.js:n kehitti Ryan Dahl vuonna 2009. Se poikkeaa perinteisimmistä palvelinpuolen kehitysympäristöistä (kuten PHP, Python, Ruby jne.) siinä, että se on täysin asynkroninen. Tämä tarkoittaa, että kaikki tapahtumat tapahtuvat taustalla eivätkä vaadi sivun uudelleenlatausta. [10; 11.]

Perinteisesti selainsovellukset toteutetaan siten, että palvelinpuolen ohjelmointikieli toteuttaa taustalla tapahtuvat toiminnollisuudet (esim. PHP) synkronisesti, eli yksi kerrallaan. Kun HTML-rakenteesta ja CSS-tyylimäärittelyistä koostuvasta loppukäyttäjäpuolesta tehdään palvelinkysely, on sivuston latauduttava joka kerta uudelleen. Tästä johtuen seuraavaa tapahtumaa ei voida kutsua, ennen kuin edellinen on täysin suoritettu. Node.js käyttää Google Chromelle rakennettua V8 JavaScript-suoritusmoottoria, joka toimii yksisäikeisesti (engl. single thread) ja asynkronisesti. [10.]

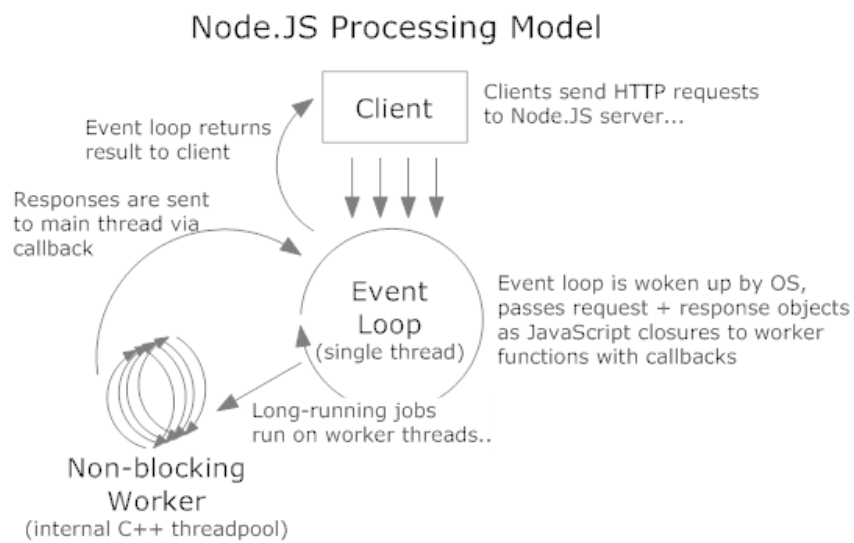
Asynkronisia prosesseja käytettäessä on kehittäjän tarkennettava palautusfunktio (engl. callback function) jokaisen funktion suorittamisen päätteeksi. Tätä kutsutaan tapahtumaohjatuksi ohjelmoinniksi (engl. event-driven). Tapahtumaohjattu ohjelmointi mahdollistaa sen, että sovellus pystyy suorittamaan kaikki pyynnöt samassa tapahtumajonossa (engl. event loop). Seuraava palautusfunktio kutsutaan vasta, kun edellinen on suoritettu. Mikäli tapahtumia on paljon, tai ne ovat hyvin raskaita, lähettää tapahtumajono nämä vielä erilliseen tapahtumakäsittelyyn. Node.js sisältää Libuv -nimisen ohjelmistokirjaston, joka suorittaa nämä erilliset tapahtumakäsittelyt. [10, 12.] Kuva 1 kuvastaa asynkronista prosessia.

Node.js ei tee itsessään yhtään mitään, se on vain ympäristö. Kehittäjän on tehtävä kaikki itse. Jos kehittäjä haluaa käyttää esim. http-protokollaa, se täytyy itse määrittää koodissa (koodiesimerkki 1). Node.js:n mukana tulee valmiina moduuleita, jotka helpottavat ja nopeuttavat kehitystä. Moduuleita voi myös ladata tarpeen vaatiessa lisää NPM-pakkaushallintajärjestelmällä. Esimerkiksi http-protokollamoduuli tulee oletuksena Node.js:n mukana.

```
var http = require("http");
```

Koodiesimerkki 1. http-protokollan sisällyttäminen koodiin.

Yksittäisten moduulien lisäksi Node.js:iin voidaan lisätä laajempia ohjelmistokehyksiä. Näitä kehyksiä kutsutaan väliohjelmistoiksi (engl. middleware). Väliohjelma tarjoaa sovellukselle palveluita, joita käyttöjärjestelmällä ei pystytä tai kannata tarjota. Väliohjelmat nimensä mukaisesti operoivat sovelluksen ja käyttöjärjestelmän välissä. Node.js:n kuu-  
luisin väliohjelma on Express. [13; 14; 15.]



Kuva 1. Node.js:n asynkroninen prosessointimalli [15].

### 3.2 Express-palvelinohjelmistokehys

Express.js on Node.js:lle rakennettu palvelinohjelmistokehys. Sen päätarkoituksena on helpottaa osoitteiden reititysten ja palvelinpyyntöjen luontia ja hallintaa. Express sisältää hyvin vähän toiminnallisuuksia itsessään. Se on pohjimmiltaan sarja väliohjelmafunktioita, joita pystytään lisäämään tarpeen vaatiessa. Niiden päätehtävä on nopeuttaa ja helpottaa kehitystä. Express-sovellus pystyy käyttämään seuraavanlaisia väliohjelmitojoja:

- sovellustason väliohjelmitojoja
- osoite reititystason väliohjelmitojoja
- virhekäsittelyn väliohjelmitojoja
- kolmannen osapuolen väliohjelmitojoja.

Express hoitaa taustahallinnan puolen yhdessä MongoDB:n kanssa MEAN-kokoonpanossa (engl. stack). MEAN on avoimen lähdekoodin JavaScript-kokoonpano, joka on myös nosteessa tällä hetkellä. Siihen kuuluvat MongoDB, Express, AngularJS ja Node.js. [16; 17.]

### 3.3 NoSQL-tietokanta

NoSQL (alkuperäisesti ”Not Only SQL”) kattaa laajan valikoiman erilaisia tietokantatekniikoita, joiden päätarkoituksena on käsitellä suuria datamassoja (Big Data). Tämänkaltaiset tietokannat ovat olleet olemassa jo 1960-luvulta asti, mutta niitä alettiin kutsua ”NoSQL”-tietokannoiksi vasta 2000-luvulla suurten verkkosovellusten synnyttyä.

NoSQL-tietokannat voidaan jakaa neljään päämalliin:

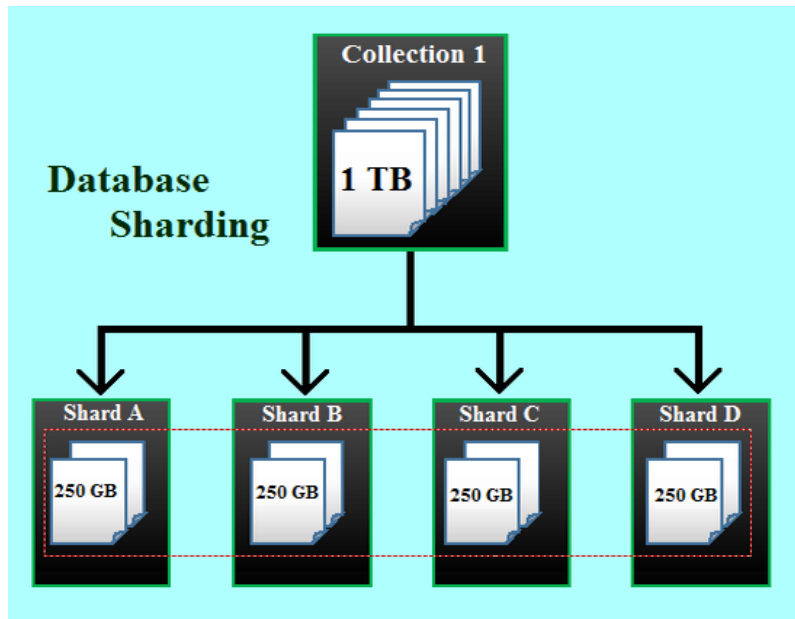
- Dokumenttitietokannat: Varastoivat jokaiselle avaimelle arvon sijasta tietorakenteen jota kutsutaan dokumentiksi. Dokumentit voivat sisältää monia erilaisia avain-arvo pareja, tai avain-taulukko pareja(engl. array), tai jopa sisäkkäisiä dokumentteja.
- Kaaviomaiset tietokannat: Varastoivat dataa joka voidaan hyvin esittää kaavioina. Esimerkkinä sosiaaliset suhteet tai julkisen liikenteen reitit.

- Avain-arvotietokannat: Yksinkertaisin NoSQL-malli. Jokainen tietokanta-alkio varastoidaan avaimena sen arvon kanssa. Jotkut avain-arvokannat sallivat arvolle tyyppiluokituksen (esim. kokonaisluku).
- Laaja-kolumniset tietokannat: Optimoitu suurten aineistojen tiedusteluihin ja varastoivat tietokolumnit kolumneittain rivien sijaan.

NoSQL-kantoja kannattaa käyttää silloin, kun on tarvetta helposti skaalautuville tietokannoille. NoSQL-kannat ovat myös paljon nopeampia käsittelemään suurta määrää tietoa kuin relaatiokannat. Relaatiokantoja on vaikea skaalata, sillä niitä ei ole suunniteltu käsittelemään suuria määriä jäsenelyä, semi-jäsenelyä tai jäsentymätöntä tietoa (engl. structured, semi-structured, unstructured).

Relaatiokannat vaativat kantakaavioiden (engl. schema) määrittämisen, ennen kuin tietoa voidaan syöttää. Jos halutaan luoda verkkokauppa, johon syötetään tuotteet ja niiden hinnat, tarvitsee relaatiokantaan rakentaa kaavio, josta löytyy paikat syötettäville tiedoille. Suuria muutoksia tehtäessä on päivittäminen todella hidasta, sillä joka kerta pitää kantaan erikseen luoda rakenne tulevia muutoksia varten. NoSQL-kannat on rakennettu siten, että tietoa voidaan syöttää ilman etukäteen rakennettua kaaviota. Täten suuret muutokset sovelluksessa eivät vaadi aikaa vievää muutosta tietokannassa. Kehittäminen on nopeampaa eikä vaadi juurikaan tietokannan hallinnointia.

Relaatiokannat skaalautuvat yleensä vain vertikaalisesti. Tämä tarkoittaa, että yhden sovelluksen tietokanta pitää olla isännöitynä yhdellä palvelimella. Jos tila loppuu, sitä pitää ostaa lisää. Tämä tulee nopeasti isoja tietokantoja käsiteltäessä kalliiksi. NoSQL-kannat tarjoavat tähän ratkaisun. Sen sijaan, että lisättäisiin kapasiteettia yhdellä palvelimella, jaetaankin tietoa muille vapaille palvelimille (ns. horisontaalinen skaalaus). Tämä toimenpide on myös mahdollista relaatiokannoilla, mutta se vaatii monimutkaisia ja aikaa vieviä toimenpiteitä kannoissa ja koodissa. NoSQL-kannat suurimmaksi osaksi tukevat horisontaalista skaalautumista. Ne jakavat tietoa automaattisesti muille palvelimille tilan loppuessa, eikä siihen vaadita sovelluksen päästä erillistä lupaa. [20.] Kuvassa 2 esitetään, miten horisontaalinen skaalautuvuus tapahtuu.



Kuva 2. Horisontaalinen skaalautuvuus (auto-sharding) [19].

## MongoDB

MongoDB on alustariippumaton dokumenttipainotteinen (engl. a document-oriented) tietokanta. Dokumenttipainotteinen tietokanta on suunniteltu varastoimaan, vastaanottamaan ja hallitsemaan dokumenttipainotteista informaatiota. Tätä informaatiota kutsutaan myös semi-strukturoiduksi tiedoksi (engl. semi-structured data). Dokumenttipainotteiset tietokannat ovat yksi pääkategorioista NoSQL-tietokannoissa. MongoDB on tällä hetkellä tunnetuin NoSQL-tietokanta. Se kehitettiin vuonna 2007 sovellusyhtiö MongoDB:ssä. [18; 20.]

Verrattaessa perinteiseen relaatiotietokantaan, jotka koostuvat tietokantatauluista, MongoDB kerää dokumentteja, jotka koostuvat avain-arvoattribuuteista (kuvassa 3 on esimerkki relaatiotietokantataulusta ja kuvassa 4 avain-arvorakenteesta). Yhtä dokumenttia voidaan pitää taulun rivin vastikkeena. MongoDB (NoSQL)-dokumentin avain taas on samankaltainen kuin sarakkeen nimi ja avaimen arvo taas samankaltainen kuin rivin arvo. Suurin ero relaatiokannan ja MongoDB-kannan välillä on se, että dokumentti ei ole sidoksissa tiettyyn malliin tai sarakkeen taulun sisällä. Kahdella dokumentilla voi olla samankaltaiset elementit (esim. ID-kentät) tai täysin eri elementit. Käytetään esimerkkinä verkkokauppaa: Kaikilla tuotteilla on hinta, mutta vaatetuotteiden lisäattribuutit (koko,

miesten / naisten, väri, merkki) eroavat täysin esimerkiksi kirjojen lisäattribuuteista (kustantaja, kirjoittaja, sivumäärä). MongoDB-dokumentit ovat JSON-muodossa, mikä on myös suuri ero perinteiseen relaatiokantaan verrattuna (SQL). [20.]

**Column**

ID	NAME	EXTENSION	MANAGER
1	John S	54213	Y
2	Susan P	59867	N
4	Andrew J	<del>55935</del>	N
5	Michael B	52137	Y
6	Jeremy W	50603	Y
7	Leah E	58963	N

**Table** (left side label), **Row** (right side label)

Kuva 3. Relaatiotietokannan taulu [18].

key	value
firstName	Bugs
lastName	Bunny
location	Earth

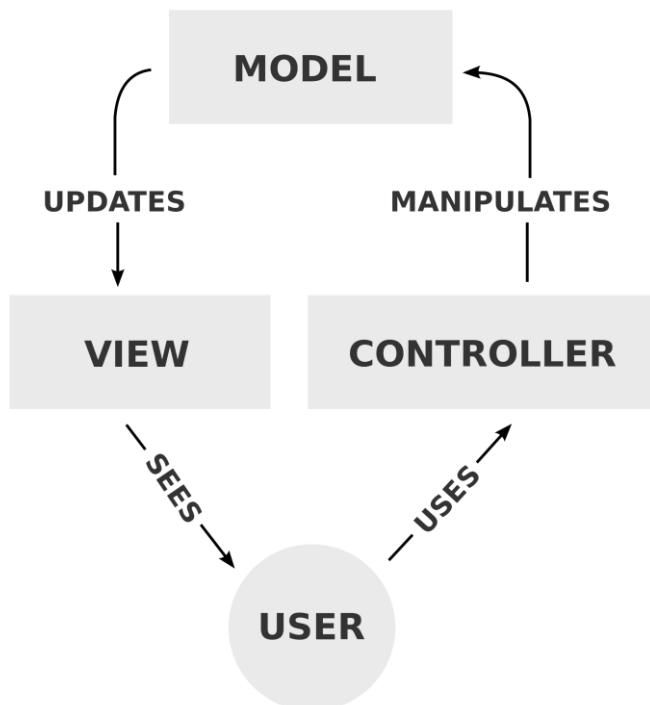
Kuva 4. Avain-arvodokumentin perusrakenne [18].



### 3.4 MVC-arkkitehtuuri

MVC-arkkitehtuuri (Model, View, Controller) on sovelluskehityksessä käytettävä sovellusarkkitehtuurityyppi, jota käytetään käyttöliittymän rakentamisessa (kuvassa 5 graafinen selitys MVC-kierrosta). Käytännössä MVC-arkkitehtuuri tarkoittaa koodin jakamista kolmeen eri osaan:

- malli
- näkymä
- kontrolleri.



Kuva 5. MVC-arkkitehtuurin osien interaktiomalli [23].

Mallin päätarkoituksena on hoitaa tietokantakäsittelyt ja muut taustalogiikat. Näkymä vastaa taas ulkoasusta, eli hoitaa merkintäkieli liittyvän koodin (HTML) ja kontrolleri käyttäjän syöttämistä komennoista ja niiden vastaanotosta sekä tiedon välittämisestä näkymään. [22.]

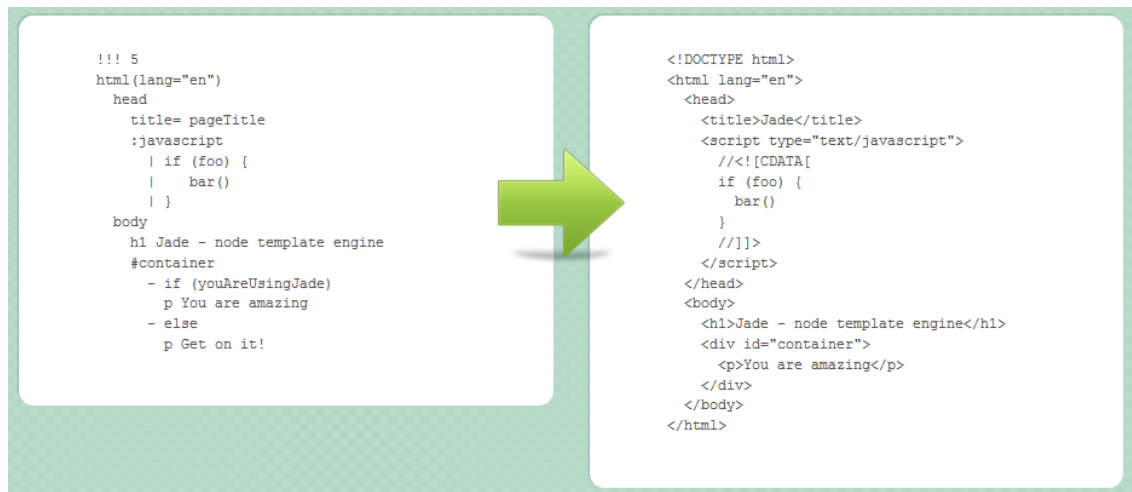
MVC-arkkitehtuuri on ollut tunnettu jo pidempään perinteisessä sovelluskehityksessä. Viime vuosina siitä on tullut suosittu myös verkkosovelluskehityksessä. Varsinkin monet

suositut JavaScript-sovelluskehitykset perustuvat MVC-arkkitehtuuriin, kuten AngularJS, Backbone ja ReactJS. [23.]

MVC-arkkitehtuurin suurimpia hyötyjä ovat luokkien uudelleenkäytettävyys ilman suurempia muutoksia. Jos halutaan käyttää samaa jo rakennettua osaa (esim. kuvagalleriaa) eri osiossa sovellusta, mutta pienin muutoksin, ei tarvitse koko galleriaa rakentaa uudelleen.

### 3.5 Jade-verkkomallinemoottori

Jade on JavaScriptilla toteutettu korkean suorituskyvyn verkkomallinemoottori (engl. WEB template engine) sekä merkintäkieli. Jade on ottanut suuresti mallia Haml-merkintäkielestä. Jadea käytetään Node.js ympäristöissä. Jade-moottori kääntää kirjoitetun Jade-syntaksin lopulta HTML:ksi loppukäyttäjäpuolella (tästä esimerkki kuvassa 6). Jadesta on ladattava versio, jonka voi linkittää suoraan sovellukseen. Tämä ei ole kuitenkaan suositeltavaa, sillä kirjastolla on vain uusimpien selainten tuki. Paras tapa käyttää Jadea on kääntää Jade-mallineet JavaScriptiksi ja käyttää niitä loppukäyttäjäpuolella.



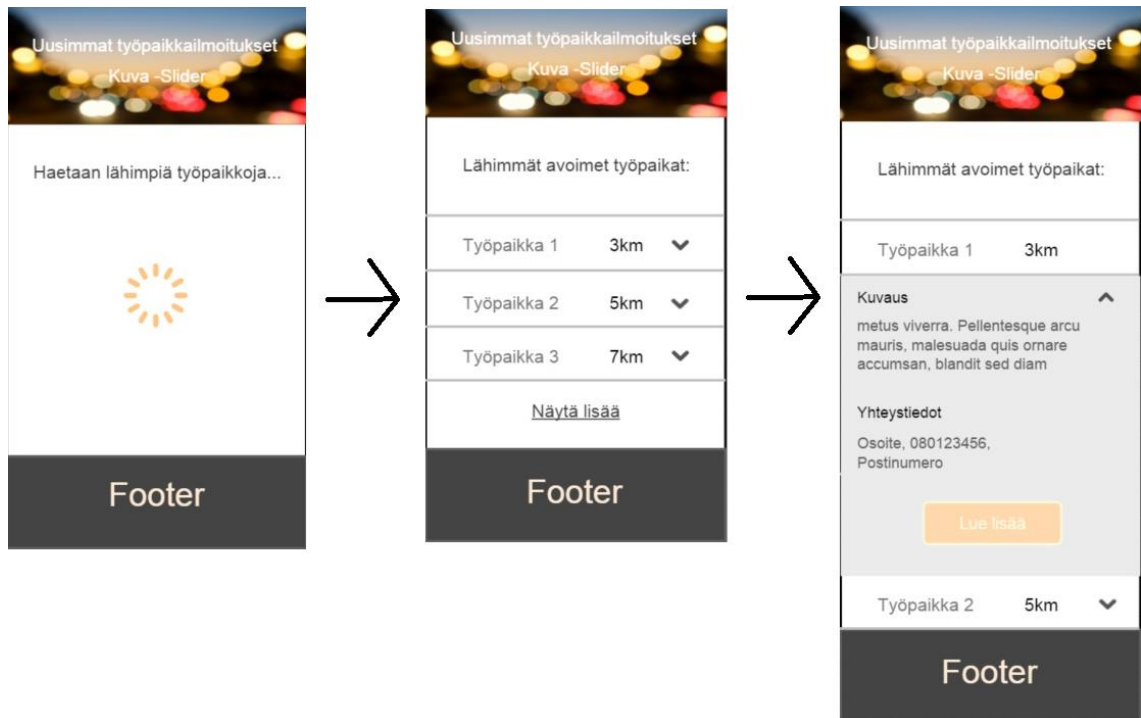
Suurin syy sille, minkä takia Jadea kannattaa käyttää Node.js-ympäristöissä HTML:n sijaan, on syntaksin selkeys ja puhtaus. Tästä johtuen sen kirjoittaminen on nopeampaa ja tuottavampaa ja hallinnointi helpompaa. [24; 25.]

## 4 Työnhakupalvelusovellus

### 4.1 Lähtökohdat ja suunnittelu

Työnhakupalvelusovelluksen tarkoituksena on listata sovelluksen käyttäjälle lähimmät avoimet työpaikat. Sovellusta ei tehdä yritykselle tai säätiölle, vaan se on täysin itse keksitty. Monet työnhakupalvelut ovat hyvin vaikeasti lähestyttäviä, ja tästä syystä moni jättää hakemukset tekemättä. Palvelun tarkoitus ei ole tarjota korkean osaamisen huippu-työpaikkoja, sillä näihin työpaikkoihin työnantajat varmasti odottavat ahkeria ja aikaan saavia työntekijöitä, jotka eivät raskaimmista työnhakuprosesseista säikähdä. Nykypäivänä tapailupalveluiden yleistyttyä perustöiden hakeminen voisi olla yhtä helppoa. Palvelu kehitetään lähtökohtaisesti mobiililaittekokoon.

Palvelun ei ainakaan tässä vaiheessa ole tarkoitus tulla julkaistuksi oikeille käyttäjille. Tällä hetkellä sen kehittämisen päätarkoituksena on KeystoneJS:iin syventyminen ja sen selvittäminen, kuinka paljon tämä sisällönhallintajärjestelmä on PHP-pohjaisia jäljessä. Mikäli palvelu onnistuu ja sillä on potentiaalisesti käyttöä, voidaan harkita jälkikäteen graafikon ottamista mukaan ja tehdä palvelusta julkaisukelpoinen. Kuvassa 7 on hahmotelma loppunäkymästä.



Kuva 7. Sovelluksen käyttöliittymän perusnäkyvät.

#### 4.2 KeystoneJS:n asennus

KeyStoneJS:n asennus onnistuu erittäin helposti käyttämällä Yeoman-generaattoria. Tarvitsee vain syöttää muutama komento komentorivillä ja uusi sovellus on valmiina kehitettäväksi. Generaattori kysyy käyttäjältä uutta projektia asennettaessa muutamien kysymyksen liittyen asennettavaan projektiin: Mikä projektin nimeksi tulee? Mitä lisäosia halutaan asentaa? Uuden projektin perusasetukseen ei siis pitäisi mennä monta minuuttia.

Kun asennus on valmis, käyttäjällä on valmis kansiorakenne olemassa määrittämässään hakemistossa. Mikäli käyttäjä haluaa muuttaa sivunäkymää, tähän ei vaadita muuta kuin tyyli- ja näkymätiedoston muokkaamista ja tallentamista, niin muutokset tulevat näkyviin. Taustalogiikan muutosten näkyminen sivulla (olivat kyseessä sitten URL-muutokset, uuden sivun luonti koodilla tai mitä tahansa) vaativat nämä muutokset http-palvelimen ja sovelluksen uudelleen käynnistämistä. Tämä prosessi on hidas ja turhauttava, joten sitä varten on onneksi rakennettu Node.js-moduuli nimeltä Nodemon. Nodemon tarkkailee kaikkia tiedostoja projektin kansiorakenteessa. Mikäli se havaitsee mitään muutoksia, se

käynnistää palvelimen ja sovelluksen uudelleen. Tämä säästää aikaa kehittäessä. Riippuen täysin Node.js:n ja MongoDB:n versioista joutuu käyttäjä mahdollisesti tekemään muutoksia KeystoneJS-ydintiedostoihin. Itse jouduin näin tekemään, sillä vaadittavat moduulitiedostot eivät linkittyneet oikein, jotta Nodemon toimisi moitteettomasti. Kun Nodemon on saatu asennettua, se päivittää sovelluksen aina tiedostoa tallennettaessa. Tästä eteenpäin peruskehitys pitäisi olla mahdollista. Mikäli sovellus tarvitsee paljon lisätoiminnallisuuksia, pystytään asentamaan lisää moduuleita, joiden konfigurointi saattaa tuottaa ongelmakohtia. Toisin sanoen KeyStoneJS:n asentaminen siihen pisteeseen, että sillä pystytään kehittämään sekä back-endiä että front-endiä, onnistuu yllättävänkin kivuttomasti.

### 4.3 Käyttäjien luonti KeystoneJS:llä

Uutta KeystoneJS-projektia luotaessa määritetään pääkäyttäjä admin-oikeuksilla. Pääkäyttäjä pystyy lisäämään uusia käyttäjiä taustahallintapaneelista. Tämä tarkoittaa käytännössä sitä, kun uusi työnantaja haluaa päästä lisäämään avoimen työpaikan kantaan, hänen täytyy ensiksi lähettää pyyntö pääkäyttäjälle, joka luo käyttäjän hänelle. KeystoneJS:ssä ei vielä tällä hetkellä ole siis rekisteröintimahdollisuutta loppukäyttäjäpuolella. Jos haluaa rekisteröintimahdollisuuden, se täytyy luoda itse. Kuvassa 8 on koodiesimerkki, miten rekisteröintimahdollisuus luodaan JavaScriptillä.

```

38
39 view.on('post', { action: 'user.create' }, function(next) {
40     var newUser = new User.model({
41         name: {
42             first: locals.formData.first,
43             last: locals.formData.last
44         }
45     });
46
47     var updater = newUser.getUpdateHandler(req);
48
49     updater.process(req.body, {
50         fields: 'email, password',
51         flashErrors: true,
52         logErrors: true
53     }, function(err, result) {
54         if (err) {
55             data.validationErrors = err.errors;
56         } else {
57             req.flash('success', 'Account created. Please sign in.');
```

Kuva 8. Koodiesimerkki rekisteröinnin kontrollerista.

#### 4.4 Tarvittavan datan mallinnus tietokannassa

KeystoneJS tarjoaa vakiona vain yhtä syötemallia, nimeltä "post". Tätä syötemallia luottaessa pystyy käyttäjä syöttämään perustiedot julkaisusta:

- otsikko
- tila (julkaistu, luonnos)
- kirjoittaja
- päivämäärä
- kuva
- sisältö
- jatkettu sisältö.

Koska sivuston tarkoituksena on listata lähimpiä avoimia työpaikkoja, täytyy työpaikoille luoda oma syötemalli nimeltä "työpaikat". Perussyötekenttien lisäksi "työpaikat"-julkaisu

vaatii paikkatiedot, jotta pystytään näyttämään käyttäjälle lähimmät työpaikat sivulla. Uusi räätälöity syötemalli saadaan luomalla Keystone List -objekti. Tästä on esimerkki kuvassa 9.

```

var keystone = require('keystone');
var Types = keystone.Field.Types;

/**
 * Job Model
 * =====
 */

var Job = new keystone.List('Job', {
  map: { name: 'title' },
  autokey: { path: 'slug', from: 'title', unique: true }
});

Job.add({
  title: { type: String, required: true },
  state: { type: Types.Select, options: 'draft, published, archived', default: 'draft', index: true },
  author: { type: Types.Relationship, ref: 'User', index: true },
  publishedDate: { type: Types.Date, index: true, dependsOn: { state: 'published' } },
  location: { type: Types.Location, defaults: { country: 'Finland' } },
  image: { type: Types.CloudinaryImage },
  content: {
    brief: { type: Types.Html, wysiwyg: true, height: 150 },
    extended: { type: Types.Html, wysiwyg: true, height: 400 }
  },
  categories: { type: Types.Relationship, ref: 'PostCategory', many: true }
});

Job.schema.virtual('content.full').get(function() {
  return this.content.extended || this.content.brief;
});

Job.defaultColumns = 'title, state|20%, author|20%, publishedDate|20%';
Job.register();

```

Kuva 9. Koodiesimerkki ”työpaikat” List-objektin luonnista.

Tärkeimmät kohdat uutta List-objektia rakennettaessa ovat objektin luonti, kenttien lisäys objektiin ja objektin rekisteröinti. Nämä tehdään syntakseilla

- `new keystone.List(key[ 'objektin_nimi', options ] );` (luonti)
- `Objekti.add` (kenttien lisäys)
- `Objekti.register();` (rekisteröinti).

Rekisteröinnin jälkeen tietokannassa on uusi malli, joka perustuu avain-arvodokumentti-rakenteeseen.

Koska KeystoneJS-kansiorakenne perustuu MVC-arkkitehtuuriin, kuuluu uudet syötemallit tallentaa models-kansioon. Routes-kansioon taas tallennetaan tietokantahakuihin liittyvät tiedostot, tässä tapauksessa tiedosto `job.js`, joka hakee syötemallin syötteet ja palauttaa sen käsiteltäväksi loppukäyttäjänäkymälle, jonka tiedostot löytyvät `views`-kansiossa. Tästä on JavaScript esimerkki kuvassa 10.

```

1 var keystone = require('keystone');
2
3 exports = module.exports = function(req, res) {
4
5     var view = new keystone.View(req, res);
6     var locals = res.locals;
7
8     // Set locals
9     locals.section = 'blog';
10    locals.filters = {
11        job: req.params.job
12    };
13    locals.data = {
14        jobs: []
15    };
16
17    // Load the current job
18    view.on('init', function(next) {
19
20        var q = keystone.list('Job').model.findOne({
21            state: 'published',
22            slug: locals.filters.job
23        }).populate('author categories');
24
25        q.exec(function(err, result) {
26            locals.data.job = result;
27            next(err);
28        });
29
30    });
31
32    // Load other jobs
33    view.on('init', function(next) {
34
35        var q = keystone.list('Job').model.find().where('state', 'published').sort('-publishedDate').populate('author').limit('1');
36
37        q.exec(function(err, results) {
38            locals.data.jobs = results;
39            next(err);
40        });
41
42    });
43
44    // Render the view
45    view.render('job');
46
47 };
48

```

Kuva 10. Koodiesimerkki työpaikkasyötteiden hausta.

Tärkeimmät kohdat job.js-kontrolleritiedostossa ovat moduulien vienti, uuden näkymä-objektin luonti, itse haku ja lopuksi näkymän renderöinti. Nämä tapahtuvat syntakseilla

- `module.exports = fuction(request, results){};` (Moduulien eksportointi)
- `new keystone.View();` (Uuden näkymä-objektin luonti)
- `keystone.list('Objektin_nimi').model.find().where(array('argumentit'));` (Haku)
- `view.render('objektin_nimi');` (Näkymän renderointi).

Kun tarvittavat tiedot on mallinnettu tietokantaan ja tietokantahaut suoritettu, voidaan kirjoittaa loppukäyttäjänäkymän tiedosto.



#### 4.5 Algoritmi lähimmän avoimen työpaikan löytämiseen

Lähimpien työpaikkojen listaaminen on sovelluksen tärkein toiminnallisuus, joten logiikka sen toteuttamiseen täytyy toimia moitteettomasti. Ennen kuin syvennyttään listauksen mahdollistavaan logiikkaan tarkemmin, tarvitaan perustiedot joilla päästä alkuun.

Jotta pystytään laskemaan kahden sijainnin etäisyys toisistaan, tarvitaan näiden kahden sijainnin paikkatiedot. Tämän projektin tapauksessa tarvitaan siis työnhakijan ja listattavien työpaikkojen leveys- ja pituusasteet. Työpaikat pysyvät paikallaan, joten niiden tiedot saadaan helposti. Ne syötetään samalla, kun luodaan taustahallinnassa uusi työpaikka. Työnhakijan paikkatiedon saamiseen tarvitaan HTML5-rajapinnan tarjoamaa geolokaatio-palvelua. Geolokaatiopalvelusta on koodiesimerkki kuvassa 11.

```
var user_pos_lat, user_pos_long;

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    alert("Geolocation is not supported by this browser.");
  }
}

function showPosition(position) {
  user_pos_lat = position.coords.latitude;
  user_pos_long = position.coords.longitude;
}
```

Kuva 11. Koodiesimerkki työnhakijan paikkatietojen saamisesta.

Jotta pystytään vertaamaan yhden hakijan paikkatietoja kaikkiin palvelun sisältämiin avoimiin työpaikkoihin, täytyy avointen työpaikkojen paikkatiedot asettaa yhteen ja samaan objektiin. Jotta saadaan jokaisen työpaikan sijaintitiedot samaan muuttujaan, täytyy objekti iteroida läpi. Kun on tiedossa hakijan ja jokaisen julkaistun työpaikan paikkatiedot, ne voidaan lähettää funktioon, jossa itse sijaintivertailu lasketaan. Näiden tapahtumien koodiesimerkki on kuvassa 12.

```

var jobs = [];

jobs[] = array(
    "ID" : data.job.id,
    "latitude" : data.job.Location.lat,
    "longitude" : data.job.Location.lng
);

jQuery.each(jobs, function(i, val) {
    var job_pos_lat = jobs[i]["lat"];
    var job_pos_long = jobs[i]["long"];
    var distance = distance( user_pos_at, user_pos_long, job_pos_lat, job_pos_long );
});

```

Kuva 12. Koodiesimerkki työpaikkojen läpikäynnistä ja funktion kutsusta tarvittavilla muuttujilla.

Lopuksi lasketaan etäisyys käyttämällä Haversine-kaavaa. Sitä käytetään yleensä kahden pisteen välisen etäisyyden laskemiseen maapallon pinnalla. Haversine-kaavan antamaa tulosta kutsutaan myös isoympyräetäisyydeksi (engl. Great-circle distance). [26.] Kuvan 13 koodiesimerkissä on käytetty Haversine-kaavaa.

```

//Function to calculate distance
function distance(lat1, lon1, lat2, lon2) {

    var theta = lon1 - lon2;
    var dist = Math.sin(toRad(parseFloat(lat1))) * Math.sin(toRad(parseFloat(lat2)))
    + Math.cos(toRad(parseFloat(lat1)))
    * Math.cos(toRad(parseFloat(lat2))) * Math.cos(toRad(parseFloat(theta)));
    dist = Math.acos(dist);
    dist = toRad(dist);
    //Distance as SM
    var miles = dist * 60 * 1.1515;

    //Return distance as km
    return ($miles * 1.609344);
}

function toRad(Value) {
    /** Converts numeric degrees to radians */
    return Value * Math.PI / 180;
}

```

Kuva 13. Koodiesimerkki Haversine-kaavan käytöstä.

Kun kaikkien työpaikkojen etäisyydet ovat tiedossa, ne tallennetaan vielä yhteen ja samaan tietokenttään (engl. array), joka lopulta järjestetään pienimmästä isompaan. Kun tätä tietokenttää käydään läpi, pystytään tulostamaan lähimmät etäisyydet.

## 5 Yhteenveto

Insinööriyön tarkoituksena oli tutkia JavaScript-pohjaisia sisällönhallintajärjestelmiä ja teknologioita niiden taustalla sekä toteuttaa esimerkkisovellus käyttäen KeyStoneJS-sisällönhallintajärjestelmää. Esimerkkisovelluksessa oli tarkoituksena käyttää kaikkia perustoiminnallisuuksia, jotka sisällönhallintajärjestelmiin yleensä kuuluvat ja näin muodostaa kuva siitä, eroavatko JavaScript-pohjaiset järjestelmät edukseen tarpeeksi, niin että ne haastaisivat PHP-pohjaiset tulevaisuudessa.

Sovelluksen toiminnallisuudet ovat valmiit. Kunnollista ulkoasua sovelluksella ei vielä ole. Tärkeintä on kuitenkin tässä vaiheessa, että tärkein ominaisuus eli lähimpien työpaikkojen listaus toimii. Sovellus ei ainakaan toistaiseksi ole julkisena missään. Seuraava askel olisi ulkoasun suunnitteleminen ja toteuttaminen.

Vanhimmat JavaScript-sisällönhallintajärjestelmät ovat noin kolmen vuoden takaa, kun taas vanhimmat PHP-pohjaiset ovat reilusti yli 10-vuotiaita. Tällä hetkellä ei JavaScript-pohjaisista ole haastamaan PHP:ta kaupallisissa sovelluksissa. JavaScript vaatii aivan erilaisen palvelinympäristön toimiakseen, mikä sen heikkous on kaupallisella puolella. Myös dokumentaation suppeus varmastikin saa monen kehittäjän edelleen valitsemaan PHP-pohjaisen järjestelmän työkalukseen. JavaScript palvelinpuolen kielenä havaittiin todella erilaiseksi verrattaessa PHP:hen. Tämä eroavuus voi olla myös suuri kynnys monelle kehittäjälle siirtyä JavaScriptin pariin.

JavaScriptin käyttö varmasti lisääntyy verkkosovelluksissa. Nähtäväksi jää, vahvistuuko sen käyttö PHP:n vierellä, vai onnistuvatko puhtaasti JavaScriptiin pohjautuvat järjestelmät saamaan jalansijaa markkinoilla.

## Lähteet

- 1 What's Yeoman? Verkkodokumentti. Yeoman. <<http://yeoman.io/>> Luettu 29.10.2015.
- 2 Tolvanen Perttu. 2009. Käsitesekamelskaa: julkaisujärjestelmä, CMS, portaali, sisällönhallintajärjestelmä. Verkkodokumentti. <<http://vierityspalkki.fi/2009/11/03/kasitesekamelskaa-julkaisujarjestelma-cms-portaali-sisallönhallintajarjestelma/>> Luettu 29.10.2015.
- 3 JavaScript: The Perfect Language for the Internet of Things. Verkkodokumentti. JScribler.<<https://blog.jscrambler.com/javascript-the-perfect-language-for-the-internet-of-things-iot/>> Luettu 29.10.2015.
- 4 KeystoneJS general. Verkkodokumentti. KeystoneJS. <<https://github.com/keystonejs/keystone/blob/master/README.md>> Luettu 2.11.2015.
- 5 KeystoneJS getting started. Verkkodokumentti. KeystoneJS. <<http://keystonejs.com/getting-started/>> Luettu 3.11.2015.
- 6 Ghost getting started. Verkkodokumentti. GhostJS. <<https://github.com/TryGhost/Ghost>> Luettu 3.11.2015.
- 7 Kiss Marcell. 2015. Node.js cms framework comparison. Verkkodokumentti. <<http://blog.budacode.com/2015/05/08/node-js-cms-framework-comparison/>> Luettu 6.11.2015.
- 8 Suda Paul. A Review of Node.js Content Management Systems. Verkkodokumentti. <<http://waitingfortheelevator.com/a-review-of-node-js-content-management-systems/>> Luettu 6.11.2015.
- 9 Buckler Craig. 2015. Verkkodokumentti. Sitepoint smackdown: PHP vs. Node.js. <<http://www.sitepoint.com/sitepoint-smackdown-php-vs-node-js/>> Luettu 6.11.2015.

- 10 Sopylo Maciej 2012. Node.js for Beginners. Verkkodokumentti. <<http://code.tutsplus.com/tutorials/nodejs-for-beginners--net-26314>> Luettu 15.11.2015.
- 11 About node.js. Verkkodokumentti. Node.js. <<https://nodejs.org/en/about>> Luettu 19.11.2015.
- 12 Introduction to libuv. Verkkodokumentti. Libuv. <<http://nikhilm.github.io/uv-book/introduction.html#background>> Luettu 22.11.2015.
- 13 Understanding the node.js event loop. 2011. Verkkodokumentti. Mixu.net. <<http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/>> Luettu 25.11.2015.
- 14 Merrit Todd. 2012. Verkkodokumentti. Multi-threaded application vs. single threaded application. Performance Zone. <<https://dzone.com/articles/multi-threaded-application-vs>> Luettu 4.12.2015.
- 15 Asynchronous IO. Verkkodokumentti. Stackoverflow. <<http://stackoverflow.com/questions/14795145/how-the-single-threaded-non-blocking-io-model-works-in-node-js>> Luettu 7.12.2015.
- 16 Using middleware. Verkkodokumentti. Express.js. <<http://expressjs.com/en/guide/using-middleware.html>> Luettu 12.12.2015.
- 17 Express.js Middleware Demystified. Verkkodokumentti. Express.js <<https://www.safaribooksonline.com/blog/2014/03/10/express-js-middleware-demystified/>> Luettu 5.1.2016.
- 18 Wilson Michael. 2014. Verkkodokumentti. MongoDB explained in 5 min or less. <<https://www.credera.com/blog/technology-insights/java/mongodb-explained-5-minutes-less/>> Luettu 9.2.2016.
- 19 MongoDB – Day 16 (Sharding). Verkkodokumentti. C-sharpcorner. <<http://www.c-sharpcorner.com/UploadFile/f0b2ed/mongodb-day-16-sharding/>> Luettu 10.2.2016.

- 20 DB-Engines Ranking. Verkkodokumentti. Db-engines. <<http://db-engines.com/en/ranking>> Luettu 10.2.2016.
- 21 What is NoSQL. Verkkodokumentti. MongoDB. <<https://www.mongodb.com/nosql-explained>> Luettu 15.2.2016.
- 22 How Graph Databases Relate To Other NoSQL Data Models. Verkkodokumentti. Neo4j. <<http://neo4j.com/developer/graph-db-vs-nosql/>> Luettu 16.2.2016.
- 23 What is MVC? Verkkodokumentti. Tutorialspoint. <[http://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](http://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)> Luettu 17.2.2016.
- 24 Sayar Rami. 2015. Verkkodokumentti. Top JavaScript frameworks, libraries, and tools and when to use them. <<http://www.sitepoint.com/top-javascript-frameworks-libraries-tools-use/>> Luettu 17.2.2016.
- 25 About jade. Verkkodokumentti. Jade. <<https://github.com/pugjs/pug>> Luettu 17.2.2016.
- 26 Introduction to Jade. Verkkodokumentti. Jade. <<http://jade-lang.com/tutorial/>> Luettu 17.2.2016.
- 27 About haversine formula. Verkkodokumentti. Movabletype.co. <<http://www.movable-type.co.uk/scripts/latlong.html>> Luettu 14.3.2016.

