Joona Haavisto

Web-sovelluskehityksen modernit tekniikat

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

22.11.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Joona Haavisto Web-sovelluskehityksen modernit tekniikat 35 sivua 22.11.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Outi Grotenfelt

Web-sovelluskehitys elää murrosvaihetta. JavaScript-kielen käyttö on lisääntynyt ja perinteisten tekniikoiden rinnalle on noussut uusia menetelmiä. Uusia JavaScript-pohjaisia komponentteja julkaistaan kiihtyvällä tahdilla ja kieltä on mahdollista nykyään käyttää sekä selainpuolen että palvelinpuolen sovelluksissa.

AngularJS on laaja ja käytännöllinen ohjelmistokehys selainpuolen käyttöliittymäsovelluksen rakentamiseen ja Node.js on laajalti integroitavissa oleva alusta palvelinsovelluksen rakentamiseen. Molemmat näistä hyödyntävät JavaScript-kieltä. Hajautetun tietokanta-arkkitehtuurimallin mukaisen NoSQL-tietokannat ovat yleistyneet, eli kehittäjän ei tarvitse enää lukkiutua tiettyyn rakenteeseen. Esimerkiksi erilaisten web-sovellusten prototypointi on nopeutunut merkittävästi.

Opinnäytetyössä tutkittiin web-sovelluksen rakennetta ja tekniikoita niin selain- kuin palvelinpuolen logiikan suhteen. Lisäksi käydään läpi hyväksi todettua kehitysympäristöä ja erilaisia web-sovelluskehityksen työkaluja. Työn tarkoituksena oli keskittyä erityisesti uusimpiin moderneihin tekniikoihin, joissa on ratkaisevassa roolissa yhteys JavaScript-kielen käyttöön tavalla tai toisella. Samalla on tuotu näkökulmaa web-sovelluskehityksen historian eri vaiheiden kautta.

Työtä on havainnollistettu käytössä olevan web-sovelluksen kehityksessä käytettyjen tekniikoiden ja työkalujen avulla. Työssä käytettyjen tekniikoiden avulla, erikseen tai yhdessä, rakennetaan sekä yksinkertaisia web-sovelluksia että monipuolisia digitaalisia palveluita.

Avainsanat	Web-sovelluskehityksen työnkulku, AngularJS, Node.js, Express.js, CouchDB, CoffeeScript, JavaScript, ElasticSearch

Author(s) Title Number of Pages Date	Joona Haavisto Modern technologies used in web software development 35 pages 22nd November 2016
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Outi Grotenfelt, Senior Lecturer

Web software development is constantly changing. The usage of the JavaScript programming language has increased and new methods have risen alongside traditional techniques. New JavaScript based components are published at an accelerating rate and the JavaScript programming language can nowadays be used in the client side, as well as the server side.

AngularJS is a versatile and useful framework for building client side web application user interface solutions and Node.js is a server side platform for a building widely integrated server software. Both of these utilize the JavaScript language. NoSQL-databases built by following distributed architecture has become more popular. Developers are no longer tied to a certain structure. For example, building web software prototypes has become more efficient.

In the thesis the structure and techniques used in the client side, as well as the server side of software logic were studied. Also the thesis goes through different development tools and the development environment. The purpose of the thesis was to focus on the newest most modern techniques, especially related to JavaScript language in way or another. At the same time the perspective is brought in through the history of web software development.

The thesis is demonstrated with the web software that is in use and techniques and tools used within the development of that software. These techniques and tools, combined or separately, can be used to build both simple web software and versatile digital services.

Keywords	Web-development workflow, AngularJS, Node.js, Express.js, CouchDB, CoffeeScript, JavaScript, ElasticSearch

Sisällys

Lyhenteet

CSS	Cascading Style Sheets, esityskieli verkkosivujen ulkoasun määrittelyyn.
DOM	Document Object Model, ohjelmointirajapinta HTML-dokumenttien muokkausta varten.
Git	Hajautetun versionhallintamallin mukaan toimiva versionhallintajärjestelmä.
GitHub	Git-versionhallintajärjestelmää hyödyntävä web-palvelu.
HTML	HyperText Markup Language, merkintäkieli verkkosivujen määrittelyyn.
JS	JavaScript, selaimen tulkkaama ohjelmointikieli.
JSON	JavaScript Object Notation, standardoitu tiedostomuoto tiedonvälitykseen, nimestään huolimatta ei ole sidottu JavaScriptiin.
Node.js	Alusta, jonka avulla voi rakentaa palvelinsovelluksia JavaScript kieltä hyödyntäen.
NoSQL	Hajautettu tietokanta-arkkitehtuuri, jossa tiedon rakenteella on vähäisempi merkitys kuin SQL-tietokannoissa.
SPA	Single-page Application, web-sovellus joka toimii käytännössä yhtä web-dokumenttia käyttäen.
REST	Representational State Transfer, arkkitehtuuri http-protokollaan perustuvien ohjelmointirajapintojen luomiseen.

1 Johdanto

Insinöörityön tarkoituksena on tutkia web-ohjelmistojen sovelluskehityksessä nykyisin käytettyjä uusia tekniikoita ja erilaisia ohjelmointikieliä sekä sovelluksia. Näihin perehdytään erilaisten lähteiden ja kirjallisuuden avulla sekä tuomalla esimerkkejä Siili Solutions Oyj:n käyttämän sovelluksen logiikasta ja toiminnallisuudesta. Siili Solutions on kehittänyt omaan sisäiseen käyttöönsä konsulttiensa osaamisen ja projektihistorian raportoinnin sekä hallinnan sovelluksen, jossa konsultti voi ylläpitää omia henkilötietoja, ansioluetteloaan sekä projektihistoriaansa. Lisäksi sovellus toimii yrityksen myynnin tukena tarjoten mahdollisuuden hakea konsultteja osaamisalueen tai muiden tietojen perusteella. Tähän sovellukseen viitataan tässä opinnäytetyössä useaan kertaan.

Web-ohjelmistojen sovelluskehityksessä käytettäviä tekniikoita on monia, ala kehittyy jatkuvasti nopealla tahdilla. Nykyään puhutaan paljon Full-Stack -osaajista, eli kehittäjistä, joilla on ymmärrys ja kyky kehittää sovellusta kokonaisuutena. Luonnollisesti se ei tarkoita, että kyseinen kehittäjä tietäisi kaikesta kaiken, mutta ohjelmoijan pitäisi pystyä tiedostamaan, mitä tapahtuu selainpuolen käyttöliittymäsovelluksessa ja mitä tapahtuu taustalla palvelimella. Tällaiset kehittäjät pystyvät luomaan monipuolisia sovelluksia yhdessä tai yksin, nopeasti ja sujuvasti.

Insinöörityö tarkastelee web-sovelluksen jatkuvaan kehitykseen käytettyjä tekniikoita nimenomaan Full-Stack -kehittäjän näkökulmasta. Työ käy läpi sekä selainpuolen käyttöliittymäsovelluksen että taustalla pyörivän palvelinsovelluksen tekniikoita. Lisäksi tutustutaan hyviin käytäntöihin ja työn edistymisen kannalta toimiviin sovellutuksiin.

Työn ensimmäisessä osioissa perehdytään web-sivustojen selaimelle näkyvän käyttöliittymäpuolen sovelluksen tekniikoihin. Työn viittaamassa sovelluksessa web-selaimessa näytettävät sivut ja tyylimääritykset muodostetaan tiettyjen esikääntäjien avulla. Varsinaisen sovelluslogiikan mahdollistaa JavaScript-kieli ja sitä käyttävät ohjelmakirjastot ja ohjelmakehykset. Samalla tutustaan digitaalisten palvelujen ohjelmistokehityksen historiaan ja elinkaareen.

Työn toinen osio tarjoaa puolestaan katsauksen palvelinpuolen sovelluksen tekniikoihin. Erityisesti keskitytään palvelinsovelluksen toiminnan mahdollistavaan alustaan ja ohjelmakehykseen sekä käytetyn tietokantaratkaisuun ja laajempaan tietojen haut

mahdollistavaan sovellutukseen. Palvelinpuolen sovelluksessa olennaista on muodostaa rajapintoja, joiden avulla käyttöliittymäsovellus saa tarvitsemansa tiedot.

Työn kolmannessa osiossa tuodaan esille tiettyjä työkaluja ja tekniikoita, joita Siili Solutions hyödyntää web-sovelluksensa ohjelmistokehityksessä. Projektissa on mukana useita kehittäjiä, joten on tärkeää, että tietyt osat toimivat yhteen ja toimintatavat ovat tuttuja kaikille. Sovelluksen ohjelmakoodin rakenne pyritään muodostamaan mahdollisimman modulaariseksi ja toimivaksi todettuja kolmannen osapuolen laajennuksia käytetään mahdollisuuksien mukaan. Versiohistorian käyttö on tärkeää. Tarvittaessa voidaan palata aiempaan toimivaan versioon. Hyvän kehitysympäristön muodostamiseen kannattaa panostaa ja sen asennus sekä määrittäminen voidaan tehdä nopeaksi tietyillä työkaluilla.

2 Web-sovelluksen front end -tekniikat

Web-sovelluksen selaimen avulla esitettyyn käyttöliittymään ja ulkoasuun liittyy monenlaisia ohjelmointikieliä ja -tekniikoita. Harvemmin web-sivu on pelkästään staattista HTML-rakennetta ja tyylisääntöjä CSS-kielellä kirjoitettuna. Usein käyttöliittymä jo itsessään muodostaa oman sovelluksen, joka logiikan ja toiminnallisuuksien osalta sisältää paljon muutakin kuin sen miltä sovellus näyttää selaimessa. Lisäksi käyttöliittymän sovelluslogiikan rakentamiseen on hyvä ottaa käyttöön erilaisia suunnittelua helpottavia ohjelmakehyksiä ja työkaluja. Tätä kokonaisuutta kutsutaan web-sovelluksen front end -puoleksi.

Opinnäytetyön sovelluksessa selaimen tulkitsemat HTML- ja CSS-määritykset muodostetaan esikääntäjien tulkkaamista koodikielistä. Käyttöliittymäpuolen sovelluslogiikka on toteutettu CoffeeScript-koodikielellä, joka käännetään esikääntäjän avulla JavaScriptiksi. Sovelluslogiikka hyödyntää JavaScriptillä toteutettua suosittua AngularJS -ohjelmakehystä. Tässä osiossa käydään läpi sovelluksen tekniikoita tutustuen ensin HTML:n ja CSS:n sekä JavaScriptillä toteutettujen web-sovellusten historiaan.

2.1 Merkintä- ja esityskielet HTML ja CSS

Internetin alkuajoista lähtien on internetin dokumentit, eli sivut määritelty HTML-merkintäkielellä (HyperText Markup Language). HTML on yhdistävä kieli World Wide Webissä. Yksinkertaisuudessaan se sisältää erilaisten elementtien merkkejä ja attribuutteja luoden näin määrittelyt web-sivun näyttämiseen selaimessa. Määrittelyjä ovat esimerkiksi tekstin asettelu, sivulla näytettävät kuvat ja erilaiset elementit tietojen syöttämiseen. Nykyään HTML5-version myötä on entistä yksinkertaisempaa määrittää sivuille videokuvaa ja ääntä. Yksinkertaisen sivun määrittelyyn riittää pelkkä HTML, mutta hiemankin laajemman toiminnallisuuden ja ulkonäön rakentamiseen tarvitaan jo esitys- ja skriptikieliä.

Edellä mainittu HTML5 on merkintäkielen viimeisin versio. HTML:n ensimmäinen julkinen versio 2.0 julkaistiin 1994 (1, s. 2) ja kielen kehitys jatkui 1990-luvulla aina versioon 4. Vuonna 1998 W3C, the World Wide Web Consortium, päätti, etteivät he jatka HTML:n kehittämistä ja merkintäkieli jämähti pitkäksi aikaa versioon 4.01 [2, s. introduction xi]. W3C päätti aloittaa työn XML-pohjaisen web-dokumenttien merkintäkielen

kehityksen parissa. Tätä he kutsuivat nimellä XHTML (eXtensible HyperText Markup Language). Tämä merkintäkieli oli erittäin tarkka merkkien sulkemisesta ja oikeanlaisesta määrittelystä. Kieli ei sallinut juurikaan vapauksia eikä tuonut varsinaisesti uusia mahdollisuuksiakaan. [1, s. 3-4.]

W3C:n jatkama XHTML:n kehityssuunta oli edelleen tiukempi ja XHTML 2 version oli määrä rikkoa yhteensopivuus aikaisempien versioiden kanssa. Vastareaktiona 2000-luvulla muodostui WHATWG-ryhmä (the Web Hypertext Application Technology Working Group) Applen, Mozillan ja Operan perustamana. Ryhmä lähti laajentamaan HTML:ää kohti entistä rikkaampaa merkintäkieltä, joka antaisi web-kehittäjille enemmän mahdollisuuksia ja toiminnallisuuksia. [1, s. 3-4.]

HTML:n yleistymisen myötä mukaan tuli myös CSS-tyylikieli (Cascading Style Sheets), joka mahdollistaa HTML-elementtien ulkoasun muovaamisen erillisillä määrityksillä ja luokka-attribuuteilla. W3C julkisti CSS:n vuonna 1996 ja jatkoi edelleen kielen kehittämistä. CSS:ää käytetään esimerkiksi joidenkin sivulla olevien elementtien värin, tekstin tyylin, tekstikoon, sijoittelun, reunusten ja esitysjärjestyksen määrittelyyn. CSS helpottaa merkittävästi web-sivun elementtien ulkoasun määrittelyjä ja tuo mahdollisuuden tehdä tiettyjä yleisiä tyylejä koko sivustolle käytettäväksi.

CSS:n kehittyessä selainten erilaiset kyvyt ja tavat esittää elementtien tyylimäärityksiä saattoivat muodostaa haasteita sovelluksen halutulle toteutukselle. Ohjelmistokehittäjät keksivät usein kiertoteitä ja jakoivat uusia tekniikoita netin keskustelupalstoilla ja muilla sivustoilla. Osa määrityksistä toimi lähes kaikilla selaimilla, osa vain tietyssä selaimessa. Internet Explorer -selaimen bugit loivat erityisesti päänvaivaa ja saivat kehittäjät varpailleen. 2010 -luvun lähestyessä selainten kehitystahti kiihtyi ja CSS-bugeja korjattiin. Samalla kehittyi uusia mahdollisuuksia ja ominaisuuksia CSS-kieltä varten. Kuitenkin aika ajoin tulee edelleen vastaan tilanteita, joissa CSS-määrityksen näyttävät erilaisilta eri selaimissa. [3, s. 1-2.]

Yhdessä HTML-merkintäkieli ja CSS-esityskieli mahdollistavat monenlaisten websivustojen rakentamisen. CSS-kielen monipuolistuessa on huomattu, että yhä harvempaan ulkoasuun liittyvään toiminnallisuuteen tarvitaan erillistä ohjelmakoodia. Uusimpien versioiden myötä CSS:llä on mahdollista lisätä varjoa elementeille, asettaa läpinäkyvyyttä ja rakentaa animaatioita tyylin muutoksesta. HTML5 puolestaan on kehittyessään tuonut mukanaan uusia mahdollisuuksia videoiden ja äänen toistamiseen.

Lomakkeita ja muita kenttiä varten on uusia oikeellisen syötteen tarkistamisen keinoja. Sijaintitietojen hyödyntäminen on nykyään mahdollista ja selaimen muistia voidaan hyödyntää laajemmin. Semanttiset elementit helpottavat erilaisten ohjelmien työtä, esimerkiksi näkökyvyltään rajoittuneille tarkoitettuja lisäosia. HTML5:n ilmestymisen myötä interaktiivisten digitaalisten palvelujen rakentaminen selaimessa ajettavaksi on mahdollista ja nykypäivää.

2.2 HTML- ja CSS-esikääntäjät

Web-sivuston luominen HTML:n ja CSS:n avulla vaatii lopulta aikaa ja työtä. Yksinkertaisenkin sivun luomiseen ja tyylittelyyn vaadittu koodi voi kertyä varsin pitkäksi. Lisäksi HTML- ja CSS-tiedostot ovat staattisia, joten niissä ei ole mahdollisuutta käyttää dynaamisia ominaisuuksia kuten esimerkiksi funktioita tai toistolauseita.

Ohjelmistokehittäjät ovat luonnollisesti pyrkineet vähentämään uudelleenkirjoittamisen tarvetta eri vaiheissa ja halunneet lisätä toiminnallisuutta staattisiin tiedostoihin. Tähän ratkaisun ovat tuoneet erilaiset esikääntäjät, eräänlaiset johdannaiset kyseisistä kielistä, jotka sitten käännetään HTML:ksi tai CSS:ksi ennen sovelluksen julkaisua. Olennaista on yksinkertaistaa ja nopeuttaa HTML:n tai CSS:n kirjoittamista. Lisäksi esikääntäjät tarjoavat yleensä mahdollisuuden luoda funktioita ja tuovat muita dynaamisia ominaisuuksia kyseessä olevaan kieleen.

HTML:n osalta on opinnäytetyön sovelluksessa käytetty Pug-nimistä esikääntäjää (ennen tunnettu nimellä Jade). Pugille ominaista on yksinkertaistettu HTML:n määritysten kirjoittaminen ilman turhaa toistamista. Lisäksi on mahdollista käyttää funktioita, liittää osioita, iteroida ja käyttää muuttujia. Esimerkkikoodit 1 ja 2 kuvastavat lyhyesti HTML:n ja Pug-kielen eroavaisuudet.

```
1           <!DOCTYPE html>
2           <html>
3             <head>
4             <title>KnoMe @Siili</title>
5             ...
6             </head>
7           </html>
```

Esimerkkikoodi 1. Sivun määritys HTML-kielellä

```
1           doctype html
2           html
3             head
4               title KnoMe @Siili
5           ...
```

Esimerkkikoodi 2. Edellisen esimerkkikoodin määritys Pug-esikääntäjän kielellä. Tekstin sisennyksellä kerrotaan kääntäjälle, minkä elementin sisälle alemman tason elementit luodaan

CSS:n osalta on usein käytetty esimerkiksi SASS, LESS tai Stylus -esikääntäjiä. Opinnäytetyön sovelluksessa on käytetty Stylusta. Tavallinen CSS on havainnollistettu esimerkkikoodissa 3.

```
#deliveriesView .link:hover {
    text-decoration: none !important;
    opacity: 1 !important;
}

#deliveriesView .table {
    width: 100%;
    table-layout: fixed;
}

.smallBorderRadius {
    -webkit-border-radius: 3px !important;
    -moz-border-radius: 3px !important;
    border-radius: 3px !important;
}
```

Esimerkkikoodi 3. Tavallinen CSS-koodi

Stylus-kielessä tekstin sisennyksillä kerrotaan, mitä elementtiä määritykset koskevat. Lisäksi Stylus-kielessä voi käyttää tavallisista ohjelmointikielistä tuttuja toimintoja kuten funktioita ja muuttujia. Lisäksi Styluksella voi liittää muita Stylus-tiedostoja yhteen, tämä mahdollistaa tyylitiedostojen modulaarisuuden ja näin pystytään välttämään massiivisia tyylitiedostoja. Esimerkkikoodissa 4 on esitelty Styluksella kirjoitettu versio esimerkkikoodin 3 sisällöstä.

```
1           #deliveriesView
2             .link
3               &:hover
4                 text-decoration none !important
5                 opacity 1 !important
6                 color #ff500a !important
7             .table
8               width 100%
9               table-layout fixed
10              white-space nowrap
11
12          .smallBorderRadius
13            border-radius 3px !important
```

Esimerkkikoodi 4. Stylus-kielellä kirjoitettua CSS-määrittelyä. Reunan pyöristyksen (border-radius) on käytetty funktiota, joka on esitelty aikaisemmin stylus-tiedostossa. Se käytännössä kirjoittaa kolme eri selainten tukemaa border-radius -määritystä funktiolle annetuilla argumenteilla.

2.3 JavaScript

Erilaisten web-sovellusten kehityksessä tulee hyvin pian vastaan tilanne, jossa tarvitaan laajempaa toiminnallisuutta, mitä ei pystytä pelkällä HTML-kielellä luomaan. Tässä vaiheessa JavaScript (usein lyhennetty JS) tulee mukaan. Se on verkkosivuilla käytetty tulkattu, kevyt olio-pohjainen skriptauskieli. JavaScriptin oliopohjaisuus perustuu prototyyppipohjaiseen ohjelmointiin, jossa vahvat tyypitykset ja luokkamääritykset puuttuvat, mutta muuten on olio-ohjelmoinnin piirteet tuettuna. JavaScript toimii myös proseduraalisena ohjelmointikielenä. Kaikki metodit, muuttujat ja oliot luodaan ajon aikana. [4]

JavaScriptiä ei tule sekoittaa Java-ohjelmointikieleen. 1990-luvun loppupuolella JavaScript alkoi yleistyä selaimissa. Niihin otettiin mukaan JavaScript-moottorit ECMAScript-standardin mukaisesti [5]. Nykyisissä selaimissa on käytössä erilaisia JavaScript-moottoreita. Googlen Chrome-selaimessa käytetään V8 JavaScript -moottoria. Mozillan FireFox-selaimessa on käytössä C++-ohjelmointikielellä toteutettu Spider-

Monkey ja Java-ohjelmointikielellä toteutettu Rhino. Applen Safari-selaimesta löytyy JavaScriptCore-moottori. [4]

JavaScriptin historian aikana suuret yritykset kuten Sun, Microsoft, Adobe ja Google ovat yrittäneet syrjäyttää kielen omilla toteutuksillaan, mutta JavaScript jatkoi kehittymistään avoimena. Kielen kehittämisen ympärille muodostui avoimen lähdekoodin kehittäjien yhteisö. Vuonna 2005 julkistettiin Ajax-termi ja sen mukana tekniikoita, joissa tietoa ladataan ja ohjataan eteenpäin palvelimelle dynaamisesti ilman sivulatauksia [5]. Oltiin askelta lähempänä nykyaikaisia moderneja digitaalisia palveluja, joissa parhaimmillaan ei sivulatauksia tarvita ollenkaan ja käytetään termiä Single-page Application (SPA). JavaScriptistä on tullut yleisesti käytetty web-ohjelmointikieli, koska se on avoin, standardoitu ja ominaisuuksiltaan kattava.

JavaScriptin ominaisuudet ja mahdollisuudet ovat merkittäviä. JavaScriptillä on pääsy tiettyihin selaimen tarjoamiin ominaisuuksiin ja ladatun sivun kaikkiin elementteihin. JavaScriptin avulla voi lähettää http-pyyntöjä (Ajax), muodostaa erilaisia tietomalleja ja varastoida tietoa selaimen muistiin tai sivun evästeisiin. JavaScript voi lukea ja muokata HTML-dokumentin elementtejä selaimen luotua DOM-puun. Tämä puu on JavaScriptin luettavissa ja muovattavissa. JavaScriptilla voi esimerkiksi vaihtaa elementin sisältöä tai tyyliä ja lisätä uusia elementtejä sivulle. Tätä kutsutaan DOM-manipuloinniksi. [6]

2.3.1 JavaScript-ohjelmistokehykset ja -kirjastot

JavaScriptin yleistymistä ovat auttaneet sen ympärille kehitetyt erilaiset ohjelmistokehykset ja kirjastot. 2000-luvun puoliväliin saakka oli käytännössä tarjolla vain pelkkä JavaScript. Selainten tarjoamien DOM-rajapintojen välillä oli eroavaisuuksia, ne haittasivat ja hidastivat ohjelmistokehitystä. Ensimmäiset kirjastot ja ohjelmistokehykset ilmestyivät vuosikymmenen puolivälin jälkeen. [6]

Ohjelmistokehyksissä ja kirjastoissa on monissa pyritty selainriippumattomuuteen, eli ne toimivat yhtenäisesti ja samalla tavalla riippumatta, millä selaimella JavaScriptia suoritetaan. Ohjelmistokehykset ja kirjastot nopeuttavat kehitystyötä ja auttavat websovellusten toteutuksessa. Niiden haasteena on kuitenkin niiden valtava määrä ja kehityksen tahti. JavaScriptin maailmassa kirjasto tai ohjelmistokehys voi syntyä ja kuolla

samalla tahdilla. Riskinä on, että sivuston kehitys aloitetaan jotain kehystä apuna käyttäen ja seuraavana vuonna kehyksen tuki on loppunut eikä sitä enää kehitetä.

jQuery oli ensimmäisiä merkittäviä kirjastoja JavaScriptille. Vuonna 2006 julkaistu kirjasto toi mukanaan riskittömän ja selainriippumattoman mahdollisuuden DOM-manipulointiin, tapahtumien hallintaan ja Ajax-kutsuihin. Nämä molemmat liitettiin yhteen rajapintaan kehittäjän käytettäväksi. Käytännössä jQuery-kirjasto on suuri kokoelma JavaScript-funktioita. Näitä funktioita kutsutaan esimerkiksi CSS-valitsimella ja näin kehittäjällä on suora hallinta kyseisen valitsimen viittamaan elementtiin. Elementtejä voi valita myös CSS-luokan avulla. Tällöin valittuna on yleensä useampi elementti, joille voi lisätä attribuutin tai uuden sisällön. Ajax-kutsuihin tarjottu rajapinta on helppokäyttöinen ja yksinkertainen: määritellään vain osoite ja data sekä funktio, joka ajetaan onnistuneen kutsun jälkeen.

jQuery-kirjasto on edelleen laajalti käytössä ja sen ympärille on luotu monenlaisia lisäosia ja laajennuksia. Avoimen lähdekoodin yhteisö kehittää kirjastoa jatkuvasti tuoden parannuksia ja uusia ominaisuuksia. Haittapuolena vaikea hallittavuus, koodista tulee helposti sekavaa ja vaikeaselkoista. [7]

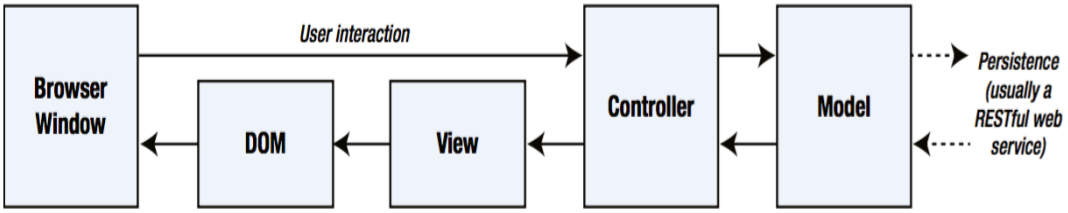
Vaatimus helpommasta hallittavuudesta loi pohjaa ohjelmistokehysten synnylle. Backbone.js, Knockout ja AngularJS julkaistiin vuonna 2010. Nämä ovat edelleen yleisimpiä JavaScript-ohjelmistokehyksiä. Ohjelmistokehykset toivat JavaScript-ohjelmistokehitykseen suunnittelumalleja, jossa käyttöliittymä- ja datan logiikka eriytettiin selkeästi toisistaan. Knockout ja AngularJS tarjosivat jo datan liittämistä näkymään kaksisuuntaisesti, joka ovat ominaisuuksia; joihin kehittäjät olivat tottuneet työpöytäsovelluskehityksessä. [7]

AngularJS on onnistunut saavuttamaan merkittävää huomiota web-sovellusten kehittäjien keskuudessa. AngularJS on onnistunut ratkaisemaan monia yksisivuisten web-ohjelmistojen kehitykseen liittyviä haasteita. [8, s. 155] Näitä toiminnallisuuksia ja ominaisuuksia olivat kaksisuuntainen tiedon liittäminen, MVC-suunnittelumalli (Model-View-Controller), riippuvuuksien hallinta, uudelleenkäytettävyys ja testattavuus.

MVC-suunnittelumallia on käytetty web-sovellusten kehittämisessä jo pitkään ennen AngularJS:n mukaan tuloa. Ensin MVC-mallia toteuttivat palvelinpuolella ajettavat web-sovellukset, joita ajettiin ASP.NET ja Ruby on Rails -ohjelmistokehyksien avulla. Viime

vuosina MVC-malli on lisännyt hallittavuutta selainpuolen nopeasti kasvavassa ja monimutkaistuvassa kehitysympäristössä. MVC-malli eriyttää selkeästi datan ja sitä käsittelevän logiikan sekä näkymätason, jossa hyödynnetään HTML-elementtejä tiedon näyttämiseen. [9]

AngularJS:n tapa toteuttaa MVC-mallia on luonnollisesti hieman erilainen, koska ohjelmakoodia ajetaan selaimen avulla. Kuvio 1 havainnollistaa AngularJS:n tapaa toteuttaa MVC-mallia. Selainpuolen MVC-mallia toteuttavaan web-sovellukseen data saadaan yleensä palvelinpuolen komponenteista, esimerkiksi REST-rajapinnan kautta. Kontrolleri- ja näkymä-tason logiikat käsittelevät saatua dataa ja tuovat halutut tiedot näkyviin DOM manipuloinnin avulla. Tämä puolestaan saa aikaan halutun näkymän selaimen ikkunaan. Käyttäjän interaktiot selaimen näkymässä saavat kierron alkamaan alusta. Esimerkiksi napin painallus viedään kontrolleriin ja näin kierto jatkuu edellä kuvatun mukaisesti tehden sovelluksesta interaktiivisen. Näkymä ja kontrolleri on liitetty toisiinsa $scope-muuttujalla ja siihen määritetyillä ominaisuuksilla ja funktioilla. [9]



Kuvio 1. AngularJS:n tapa toteuttaa MVC-mallia [9]

On myös muita yleisistä ohjelmointikielistä ja ohjelmistokehyksistä tuttuja tekniikoita, jotka AngularJS mahdollistaa JavaScript-websovelluksien toteutuksessa. Esimerkiksi riippuvuuksien määrittelyn ja selvittelyn logiikka vaatii aluksi hieman tutustumista, mutta helpottaa suuresti ohjelmointia. AngularJS:ssä ei ole tarvetta käyttää globaaleja muuttujia ja silti kontrolleriin saadaan automaattisesti näkymään liittyvä $scope-muuttuja. Riittää, että $scope mainitaan kontrollerin määrityksessä. Samanlainen logiikka toimii myös kaikkien kontrollerien, factoryjen ja muiden objektien kanssa. Määritellään uniikki nimi, joka on liitettävissä mihin tahansa toiseen objektiin. Tämä mahdollistaa monenlaisia yleisemmistä ohjelmointikielistä tuttuja suunnittelumalleja. Helppo ja käytännöllinen tapa käyttää riippuvuuksia tukee sovelluksen modulaarisuutta. Kontrollerit ja tietyt toiminnot suorittavat palvelut, factoryt tuleekin sijoitettua selkeästi erilleen, kun ne on joka tapauksessa helppo liittää toisiinsa tarvittaessa. [9]

AngularJSn deklaratiivinen lähestymistapa on ollut uutta web-ohjelmistojen kehityksessä [8, s. 157]. Ideana on, että halutun lopputuloksen saavuttamiseksi kaikkia vaiheita ei tarvitse ohjelmoida, vaan AngluarJS sisältää monia sisäänrakennettuja toimintoja ja tekniikoita. AngularJS:n merkintöjä voidaan sijoittaa suoraan näkymän HTML-koodiin ja esimerkiksi kontrollerissa $scope-muuttujaan määritetty taulukko voidaan iteroida läpi tulostaen tietyt arvot taulukko-elementtiin. Esimerkkikoodi 5 kuvastaa AngularJS:ssä käytettyä tapaa toteuttaa web-ohjelmisto.

```
1   <!DOCTYPE html>
2   <html lang="en" ng-app="app">
3     <head>
4       <title>Declarative App</title>
5     </head>
6     <body>
7       <div ng-controller="BodyController">
8         <ul>
9           <li ng-repeat="animal in animals">{{animal}}</li>
10        </ul>
11      </div>
12      <script src="/bower_components/angularjs/angular.js"></script>
13      <script>
14        var app = angular.module('app', []);
15        app.controller('BodyController', function($scope) {
16          $scope.animals = ['cats', 'dogs', 'hamsters'];
17        });
18      </script>
19    </body>
20  </html>
```

Esimerkkikoodi 5. Yksinkertainen deklaratiivinen web-ohjelmisto toteutettuna AngularJS-ohjelmistokehyksen avulla. [8]

Aivan toisenlaisen lähestymistavan JavaScript-webohjelmistokehitykseen tarjoaa Facebookin kehittämä ja ylläpitämä React-kirjasto. Sovellusten monimutkaistuessa jopa AngularJS voi käydä sekavaksi ja suorityskyky laskea. React on yksinkertainen, mutta monipuolinen JavaScript-kirjasto käyttöliittymätason logiikkaa varten. Facebook kehitti Reactin ja siihen liittyviä tekniikoita itseään varten, Facebook- ja Instagram-sovelluksien kehityksessä käytettäväksi. Sen jälkeen esimerkiksi Netflix ja Airbnb on osoittanut kiinnostusta Reactiin. Perusajatukseltaan React on komponenttipohjainen ja deklaratiivinen. React-komponenttien määrittelyyn käytetään JSX-skriptikieltä, joka on käytännössä laajennus JavaScriptiin ja käännetään JavaScriptiksi erillisellä ohjelmalla.

[10] React on kuitenkin käytännössä vain käyttöliittymätason logiikkaa varten luotu. Luonnollisesti täysin toimiva dynaaminen sovellus vaatii muutakin kuin näkymän logiikan. Siihen puoleen ratkaisun tarjoaa Redux-ohjelmistokehys. Toki muitakin ohjelmistokehyksiä löytyy tähän tarkoitukseen, mutta React ja Redux toimivat parhaiten yhteen. [11]

Web-sovelluksen elinkaaren kannalta on olennaista, millä teknologioilla lähdetään sovelluskehitystä toteuttamaan. JavaScript-ohjelmistokehysten ja kirjastojen avulla nopeuttaa ja helpottaa kehitystä, mutta haasteena on oikeiden kirjastojen ja ohjelmistokehysten valinta. Sovelluskehittäjän on punnittava, voiko luottaa siihen, että kirjaston kehitystä jatketaan ja mahdolliset bugit korjataan.

2.3.2 JavaScript-esikääntäjät

Samoin kuin HTML- ja CSS-kielille löytyy esikääntäjänsä, löytyy myös JavaScriptille. CoffeeScript-kielen avulla JavaScript-koodista saa luettavampaa, selkeämpää ja lisäksi se korjaa muutamia tunnettuja virheitä. CoffeeScript-kielessä ei käytetä puolipisteitä eikä aaltosulkeita ja osiot erotetaan sisennyksillä. Muuttujien kohdalla CoffeeScript karsii yleisen virheen tehdä muuttujasta globaali ilman var-määritystä, CoffeeScript lisää käännösvaiheessa var-määrityksen muuttujien esittelyyn. Funktiot määritellään pelkällä ohuella viivalla ja nuolella, ja niitä voi kutsua ilman sulkeita. Objektien määrittely onnistuu ilman aaltosulkeita ja taulukot ilman hakasulkeita. Esimerkkikoodi 6 esittää muutaman esimerkin avulla CoffeeScriptin-kielen ja esimerkkikoodi 7 näyttää siitä käännetyn JavaScriptin.

```
1          number = 42
2          opposite = true
3          # Conditions:
4          number = -42 if opposite
5          # Functions:
6          square = (x) -> x * x
```

Esimerkkikoodi 6. CoffeeScript-kielen avulla esitelty pari muuttujaa ja funktio

```
1           var number, opposite, square;
2           number = 42;
3
4           opposite = true;
5           if (opposite) {
6             number = -42;
7           }
8
9           square = function(x) {
10            return x * x;
11          };
```

Esimerkkikoodi 7. Edellisen esimerkkikoodin mukainen koodi käännettynä JavaScript-kieleksi

CoffeeScript on laajalti käytetty esikääntäjä JavaScriptille, mutta nykyään JavaScriptin kehitys on johtanut ECMAScript (ES) versioon 6, joka uusien ominaisuuksiensa vuoksi tekee monien kehittäjien mielestä CoffeeScriptistä tarpeettoman. ES 6:n ainoa heikkous on vielä se, että kovinkaan moni selain ei sitä vielä tue, mutta toisaalta se voidaan erillisellä työkalulla kääntää laajemmin tuettuun ES 5 -versioon.

3 Palvelinpuolen back end -tekniikat

Harvemmin laajan web-sovelluksen toteuttamiseen riittää pelkästään selainpuolen front end -tekniikat. Tarvitaan myös palvelinpuolen sovellus, joka tarkoittaa, että mukaan saadaan tietokantoja, tausta-ajoja ja nopeampaa suorituskykyä sekä skaalautuvuutta. Lähtökohta usein on, että front end -puoli on tietyn rajapinnan kautta vahvasti yhteydessä back endiin. Rajapinta tietyillä päätepisteillään mahdollistaa esimerkiksi tietokantakutsut, käyttäjien autentikoinnin, yhteydet muihin sovellusrajapintoihin ja niin edelleen. Käyttöliittymäpuolen sovellus kutsuu näitä rajapintoja hakiessaan tietoja tai suorittaessaan muita toimintoja. Opinnäytetyön projektissa rajapinta rakennetaan tietyn ohjelmakehyksen avulla, joka pyörii palvelinsovellusta hyödyntäen. Esimerkiksi konsulttien henkilötietojen, projektihistorian ja ansioluettelon tietojen tallennusta hakemista varten on rakennettu tietyt rajapinnat. Tässä osiossa tutustutaan erityisesti palvelinsovelluksen logiikkaan ja toiminnallisuuksiin, sovelluksessa käytettyyn tietokantaratkaisuun sekä hakutoiminnot mahdollistavan sovellukseen.

3.1 Node.js ja Express

Front end -puolella ohjelmistosuunnittelijat keskittyvät usein käyttöliittymäsovelluksen kirjoittamiseen ja back end -puolella taas järjestelmän rakentamiseen. Tämän eron lisäksi molemmilla puolilla on yleensä ollut omat ohjelmointikielensä. Node.js (kutsutaan myös Node) tuo kuitenkin poikkeuksen tähän. Noden avulla voidaan käyttää samaa kieltä, eli JavaScriptiä, molemmilla puolin. Tämä mahdollistaa huomattavasti helpommin ns. full-stack kehityksen, jossa sama kehittäjä voi tehdä sekä fronttiin että backiin haluttuja ominaisuuksia. [12]

Node perustuu Google Chrome -selaimessakin käytettyyn V8 JavaScript -moottoriin. Node tarjoaa alhaisen tason rajapinnat esimerkiksi tiedostojen lukemiseen ja http-toiminnallisuuden, mutta erilaisten laajennusten ja moduulien eli pakettien myötä Nodella rakentaa toiminnallisuuksiltaan laajoja palvelinsovelluksia. Noden npm-pakettienhallintasovelluksen toimintaa esitellään tarkemmin tämän työn neljännessä osiossa. Nämä paketit voivat esimerkiksi mahdollistaa tietokantayhteyden, reititystä ja välittää JSON-muodossa tietoa web-sivulle näytettäväksi.

Ensimmäinen versio Nodesta julkaistiin vuonna 2009 ja seuraavana vuonna sen ympärille kehittyi useita moduuleita ja laajennuksia. Tällä vuosikymmenellä Node on yleistynyt ja vakiinnuttanut asemansa laajojen web-sovellusten ja digitaalisten palveluiden palvelinpuolen sovellusalustana. Esimerkiksi Netflix, Paypal, LinkdedIn, Uber ja eBay sekä monet muut isot yritykset käyttävät Nodea eri sovelluksissaan. Useimmiten Nodea näkee käytettävän selaimella ajettavien digitaalisten palveluiden yhteydessä, mutta aivan samalla tavalla Node voi tarjota tietoa esimerkiksi mobiilisovelluksille. Noden yhteydessä on nykyisin puhuttu myös jonkin verran isomorfisesta sovelluskehityksestä, joka karkeasti tarkoittaa sitä, että samaa JavaScript-koodia voidaan ajaa sekä palvelin- että front end -puolella. Näin palvelin voisi ensin generoida sivun, sen jälkeen toiminnallisuudet tarjotaan selainpuolella.

Noden ympärille on muodostunut myös tavallisia moduuleja laajempia ohjelmistokehyksiä, kuten Express.js (kutsutaan myös Express). Express helpottaa websovelluksen palvelinpuolen toimintojen rakentamista monilla ominaisuuksillaan ja määrityksillään. Se tarjoaa työkaluja http-pyyntöjen käsittelyyn mahdollistaen REST-rajapintojen (Representational State Transfer) päätepisteiden ja takaisinkutsufunktioiden määrittelyn. Expressin tarjoamiin http-pyyntöihin liittyviin palveluihin kuuluu myös sovelluksen reitityksen hallinta ja se pystyy tarjoamaan rajapinnan http-protokollan ominaisuuksille kuten tilakoodeille ja tietyn sisältötyypin käsittelylle. Esimerkkikoodi 8 kuvastaa Expressillä rakennetun palvelinpuolen sovelluksen käynnistämisen Node-ympäristössä.

```
1           express  = require('express')
2           app      = express()
3           ...
4           app.use(bodyParser.json())
5           app.use(cookieParser())
6           ...
7           app.use acl.controlAccess()
8           ...
9           app.listen expressPort, ->
10            logger.info "Express server listening on port " + expres-
sPort
```

Esimerkkikoodi 8. Ensimmäiset rivit alustavat Express-ohjelmistokehyksen ja myöhemmin koodissa käynnistetään palvelin listen-funktiolla. Opinnäytetyössä esitellyssä sovelluksessa on käytetty CoffeeScript-kieltä myös palvelinpuolella.

Express mahdollistaa myös erilaisten moduulien yhteiskäytön, mitkä hoitavat esimerkiksi varsinaisen tietokantayhteyden, tiedon haun tietokannasta tai JSON-muotoisen tiedon parsimisen eteenpäin web-sovelluksen käyttöliittymäsovelluksen käyttöä varten. Express mahdollistaa myös kustomoitujen omien moduulien käyttämisen. Käyttäjän autentikointia ja roolienhallintaa varten opinnäytetyön käsittelemässä sovelluksessa on luotu omat moduulit, jotka liitetään Express-ohjelmistokehyksen instanssiin sovelluksen alustuksen yhteydessä use-metodia hyödyntäen. Esimerkkikoodi 8 rivien 2 ja 9 välissä sijaitsevat kaikki ohjelmistokehykseen liittyvien integrointien alustukset ja määritykset.

3.2 CouchDB- ja NoSQL-tietokantasovellukset

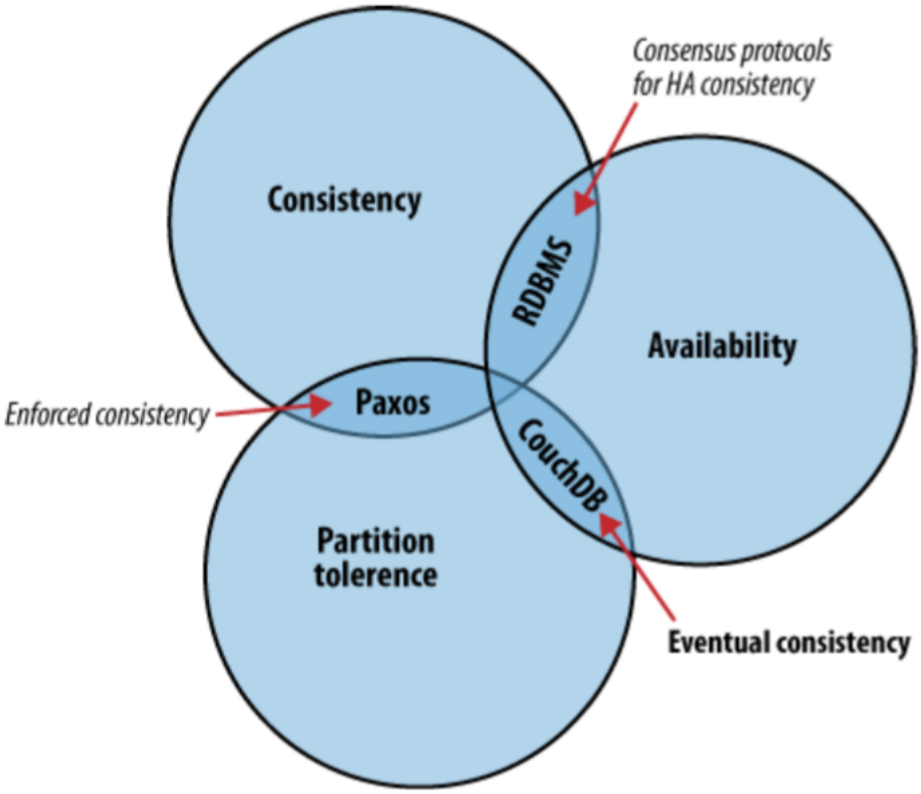
Useimmiten web-sovellus käyttää jonkinlaista tietokantaa ja sen hallintaan käytettävää tietokantasovellusta. Käytännössä erilaiset SQL-pohjaiset relaatiotietokannat ovat pitkään hallinneet web-sovelluksien tietokantaratkaisuna. Nykyään puhutaan kuitenkin yhä enenevissä määrin NoSQL (Not only SQL) -käsitteestä, jossa tiedon taulukoinnissa ja rakenteella on vähäisempi merkitys ja tietomalli perustuu yksilöityihin dokumentteihin. Nykypäivän sovellukset vaativat usein huomattavasti joustavampia tietokantaratkaisuja, puhtaan suurista tietomassoista ja vaatimuksena on nopea suorituskyky. Tähän NoSQL-tietokannat vastaavat horisontaalisella skaalautuvuudella ja joustavuudella, ne voivat tarjota hyvän vaihtoehdon relaatiotietokannalle silloin, kun tiedon rakenteelle ei ole asetettu tiukkoja vaatimuksia. Tiedon rakenteen joustavuus helpottaa ohjelmointia ja mahdollisten muutosten tekoa.

Apachen kehittämä avoimen lähdekoodin tietokantajärjestelmä CouchDB on puhtaasti web-sovellusten käyttöä varten suunniteltu tietokanta. Tiedot kirjataan järjestelmään JSON-muotoisena ja niitä pystytään hakemaan http-rajapinnan yli JavaScriptia hyödyntäen. Inkrementaalisen replikoinnin, eli monistamisen, vuoksi CouchDB on nopea ja tiedoista säilyy omat versionsa loogisessa järjestyksessä. Usean käyttäjän yhtäaikainen tietojen muokkaaminen onnistuu älykkään konfliktien selvittelyn ansiosta. Konfliktit selvitetään automaattisesti ja järjestelmään tallentuu myös selvittelyssä hävinnyt versio. Näin ohjelmatasolla voidaan päättää mitä tehdään tiedostojen konfliktitilanteessa.

CouchDB:n sisäänrakennettu replikointiominaisuus voi olla erittäin hyödyllinen, kun rakennetaan korkean saavutettavuuden sovelluksia. Idea on, että tietoja synkronoidaan useampien CouchDB-tietokantojen välillä. Tämä tarkoittaa myös sitä, että tietokannan

tietoja voidaan säilyttää esimerkiksi puhelimessa ja synkronoida sitten, kun internetyhteys toimii taas kunnolla.

Hajautettujen tietojärjestelmien kohdalla puhutaan CAP-lauseesta. Se on professori Eric A. Brewerin esittämä teoreettinen lause, jonka mukaan järjestelmä ei voi samanaikaisesti taata yhtäpitävyyttä (consistency), saavutettavuus (availability) ja vikasietoisuutta (partition tolerance). Yhtäpitävyydellä tarkoitetaan, että kaikki sovellukset näkisivät saman tiedon. Saavutettavuudella tarkoitetaan sitä, että kaikki sovellukset pääsisivät käsiksi tietoon, vaikka se ei olisikaan uusinta. Vikasietoisuus sallii tietokannan hajauttamisen useammalle laitteelle ilman tiedon hävikkiä. Sovellusta määritellessä on tärkeää valita oikea työkalu oikeaan tehtävään, kaikkia kolmea edellä mainittua ei voi huomioida yhtä aikaa. Kuvio 2 havainnollistaa näiden seikkojen keskinäistä suhdetta. [13]



Kuvio 2. Brewerin lause

CouchDB vastaa tähän keskittymällä olemaan vikasietoinen ja korkeaan saavutettavuuteen. Kaikilla laitteilla on pääsy tietoon, vaikka se ei olisikaan täysin ajan tasalla,

vikasietoisuus taataan automaattisella konfliktien hallinnalla. Näin CouchDB:sta voidaan puhua, että se on lopullisesti yhtäpitävä (eventually consistent). [13: 14.]

Toinen suosittu NoSQL-tietokantasovellus on MongoDB, jossa tiedot vastaavasti tallennetaan JSON-muodossa dokumentteihin. MongoDB:n käyttämästä dokumenttiformaatista käytetään nimitystä BSON. BSON on JSON:in binäärimuotoinen esitys, johon on lisätty ylimääräisiä datatyyppien määrittelyjä. MongoDB on erityisen suosittu kokeiluluontoisissa projekteissa, joissa halutaan ottaa tietokanta helposti ja nopeasti käyttöön eikä haluta pohtia tietokannan rakennetta sen kummemmin. Ohjelmistokehittäjän näkökulmasta MongoDB:n käyttämät dokumentit ovat yhteneviä useiden oliopohjaisten ohjelmointikielien käyttämien tietorakenteiden kanssa. MongoDB sisältää SQL-relaatiotietokannoista tuttuja ominaisuuksia kuten dynaamisia kyselyitä ja indeksejä. MongoDB eroaa CouchDB:stä siinä, että se ei tarjoa korkeaa tiedon saavutettavuutta, vaan se keskittyy ennen kaikkea tiedon yhtäpitävyyteen ja vikasietoisuuteen.

Opinnäytetyön sovelluksessa on käytetty CouchDB-tietokantaa henkilöiden osaamisen ja muiden yleisten tietojen sekä tiedostojen tallennukseen. Yhteys CouchDB-palvelimeen hoidetaan Nano-moduulin avulla. Nano on yksinkertainen lisäosa Nodelle ja sen voi hakea Noden pakettienhallintasovelluksen avulla. Nano käynnistetään tietokantapalvelimen osoitteen avulla ja sille kerrotaan, mitä tietokantaa käytetään. Tietokantatoimintojen kutsuihin annetaan aina takaisinkutsu-funktio, jolla parametreina mahdollinen virhe (error), sisältö JSON-muodossa (body) ja otsikkotiedot (header).

Projektissa Nano on sisällytetty omaan tietokantayhteyden tarjoavaan luokkaan, joka tarjoaa funktiot tietueiden listaamiseen, lisäämiseen, muokkaamiseen ja poistamiseen. Näitä funktioita kutsutaan Expressin avulla määritettyjen päätepisteiden kautta ja näin saadaan sovelluksen front end -puolelle haluttua tietoa.

3.3 Tietojen haku ElasticSearchin avulla

Sovelluksessa, jossa tiedon hakeminen on merkittävässä roolissa, hakutoimintoihin on syytä kiinnittää huomiota. ElasticSearch on avoimen lähdekoodin ohjelma rakennettuna Apachen Lucene -hakumoottorin päälle. Lucene on skaalautuva ja kevyt, mutta samalla tehokas ja suorituskykyinen Java-ohjelmointikielellä toteutettu vapaan tekstihaun hakukone. Käytännössä Lucene toimii ohjelmakirjastona. Sellaisenaan se olisi vaikea-

selkoinen ja hankala käyttää. ElasticSearch hyödyntää Lucenea monin tavoin tiedon indeksoimiseen ja hakemiseen. Samalla ElasticSearch pyrkii tekemään tekstihaun toiminnoista helppokäyttöisiä ja tarjoaa kattavan REST-rajapinnan haku- ja muita pyyntöjä varten. [21, s. 3]

Shay Banon julkaisi ElasticSearch projektinsa vuonna 2010. Julkaisun jälkeen ElasticSearch on kasvattanut suosiotaan räjähdysmäisesti. Nykyisin ohjelmistoa kehittää Elastic-yhtiö, jonka sovelluksia käyttää mm. Facebook, Wikipedia ja eBay.

ElasticSearch on käytännössä toiminnaltaan dokumenttipohjainen, eli se tallentaa ja indeksoi dokumentteja. Lisäksi se käsittelee tietoa JSON-muodossa. Dokumenttipohjaisuuden ja tietojen tallennuksen muodon vuoksi ElasticSearch toimii hyvin yhteen CouchDB-dokumenttipohjaisen tietokannan kanssa.

ElasticSearch palvelinsovelluksen kanssa kommunikointi onnistuu joko Java tai REST-rajapinnan kautta. Opinnäytetyön sovelluksessa on hyödynnetty ElasticSearch-sovelluksen REST-rajapintaa. Yleisin päätepiste rajapintaan on luonnollisesti *_search*, johon voi liittää hakuparametrilla tai JSON-määrityksillä haun rajauksia. JSON-määritykset mahdollistavat monia kompleksisia hakutapoja, kuten loogisten operaattorien tai suodattimien hyödyntämisen. Tekstihaussa hakutuloksiin liitetään relevanssiin viittaava arvo, joka on sitä suurempi, mitä paremmin sen sisältö vastaa haettuja sanoja. Hakutuloksissa täsmäävät sanat voidaan myös korostaa automaattisesti, tämä helpottaa suuresti hakutulosten näyttämistä web-sovelluksessa. [21, s. 13-20]

ElasticSearchin ominaisuuksiin kuuluu myös analytiikkaa, geolokaation määrittämistä, ehdotusten antamista ja paljon muuta. Opinnäytetyön sovelluksessa hakutoiminnot ovat suurimmassa roolissa, muiden ominaisuuksien käyttö on jäänyt vähemmälle.

4 Sovelluksen kehityksen työnkulku

Sovelluksen kehityksen työnkulku sisältää useita vaiheita ja työkaluja. Tässä osiossa tutustutaan opinnäytetyön sovelluksen kehityksessä käytettyihin tekniikoihin. Sovelluksen kehityksessä mukana useita kehittäjiä, joten on tärkeää, että käytetään samoja ratkaisuja ja tietyt vaiheet tehdään sovitulla tavalla.

Opinnäytetyön sovelluksen ohjelmistokehitykseen liittyvä kehittäjien välinen kommunikointi ja tehtävien kuvaus sekä jakaminen on hoidettu tietyillä työkaluilla. Slack on kätevä työkalu tiimin viestintään, se muistuttaa toiminnoiltaan ja tyyliltään IRC-keskustelusovellusta sekä edistää integroinneillaan projektin sujuvuutta. Projektin tehtävien kuvaukset ja suunnitelmat sijaitsevat Trello-sovelluksessa, joka ominaisuuksillaan mahdollistaa projektitiimin ketterän työskentelyn. Näiden työkalujen avulla projektissa mukana olevat kehittäjät pystyvät työskentelemään etänä projektin parissa silloin kun parhaiten ehtivät eikä sovelluksen kehitys ole sidottu tiettyyn paikkaan. Sovelluksen parissa työskentelevät kehittäjät saattavat vaihtua yllättäen asiakastilanteen mukaan.

4.1 Sovelluksen modulaarinen rakenne

Ohjelmistotuotannon hyvät käytännöt suosittelevat sovelluksen logiikan jakamista komponentteihin ja moduuleihin. Isot projektit paisuvat helposti, joten erilliset komponentit ja moduulit helpottavat hallintaa ja mahdollistavat toteutusta monin eri tavoin.

Opinnäytetyön sovellus on rakenteeltaan jaettu kolmeen eri projektiin, joista jokainen sijaitsee versionhallintajärjestelmässä omassa säilytyspaikassaan (repository). Varsinaisen client-sovelluksen logiikan lisäksi on sovelluksen infrastruktuuriin ja pilvipalveluun liittyvät projektit. Client-puolen rakenne noudattaa web-sovelluksista tuttua ja usein käytettyä modulaarista rakennetta, jossa eri resurssit sijaitsevat omissa kansiossaan. Käyttöliittymän sovelluksen resurssit ja ohjelmakoodit sijaitsevat client-kansiossa ja palvelinsovelluksen server-kansiossa. Infrastruktuuriin liittyvä projekti helpottaa kehittäjän työtä oman kehitysympäristön muodostamisessa, pilvipalvelun projekti puolestaan sisältää varsinaisen julkaistun sovelluksen hallintaan liittyvää koodia.

4.2 Versionhallintajärjestelmät

Minkä tahansa sovelluksen kehitysprosessin kasvaessa versionhallinnan käyttäminen muodostuu lähes väistämättömäksi. Versionhallintajärjestelmien etuna on, että ne helpottavat useampien kehittäjien samanaikaista työskentelyä ohjelmakoodin parissa. Järjestelmä tallentaa automaattisesti muutoksista versioita joita sitten verrataan toisen kehittäjän tekemiin muutoksiin. Opinnäytetyön sovelluksen kehityksessä on käytetty Git-versionhallintaohjelmistoa. Git julkaistiin vuonna 2005 [15], se on nykyään yleisin versionhallintajärjestelmä. Git on avoimen lähdekoodin hajautetun versionhallintamallin mukainen ohjelmisto.

4.2.1 Versionhallintamallit ja Gitin toiminta

Versionhallintamalleja on hajautetun lisäksi myös paikallinen ja keskitetty. Paikallisesta versionhallintamallista puhutaan nykyään harvoin, mutta käytännössä se voisi tarkoittaa tiedostojen ja kansioiden nimeämistä esimerkiksi päivämäärän mukaan. Näin muodostuisi eri versiot tiedostoista ja kansiosta, mutta tämä tapa osoittautuu pian hankalaksi ja vahinkoalttiiksi. Luonnollisesti tähän ongelmaan on kehitetty paikallisen versionhallinnan sovelluksia. [15]

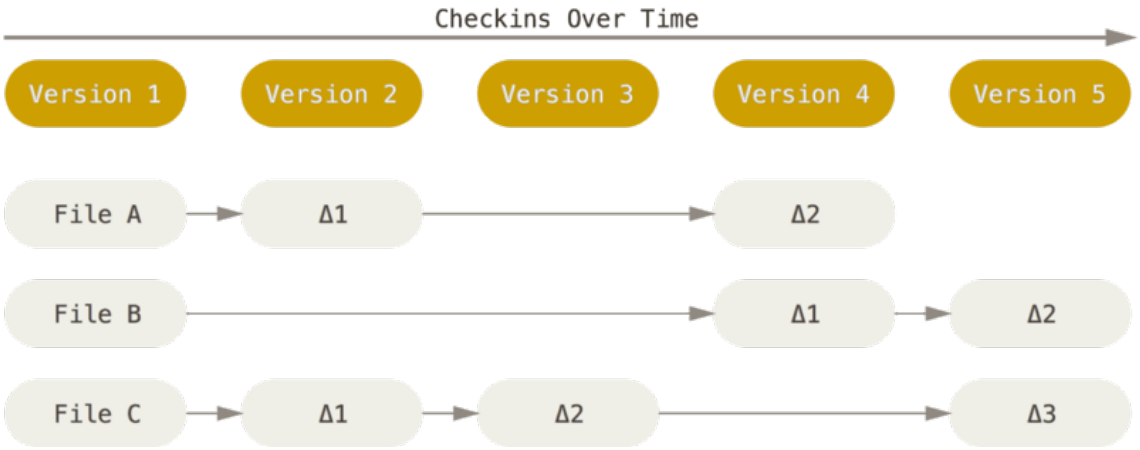
Keskitetyssä versionhallintamallissa projektin versiotiedot löytyvät keskitetysti palvelimelta. Ohjelmistokehittäjät tallentavat ja lähettävät muutoksensa projektin tiedostoihin ja kansioihin tälle kyseiselle versionhallintapalvelimelle. He voivat myös hakea muutoksia ja versionhallintasovellus automaattisesti päivittää tiedostot ja kansiot paikallisesti. Haasteita muodostuu, mikäli yhteys palvelimeen ei onnistu. Tällöin voi projektin kehitys pysähtyä kokonaan. [15]

Käytännössä niin kauan kuin ohjelmakoodeja on kirjoitettu tekstitiedostoihin, on ollut jonkinlaista versionhallintaa. 1980-luvun loppupuolella julkaistiin CVS-ohjelmisto, joka oli siis keskitetyn versionhallintamallin mukainen järjestelmä. Versionhallintajärjestelmien elinkaari onkin liikkunut paikallisesta hallinnasta keskitettyyn hallintaan ja nykyisin varsinkaan isoissa projekteissa harvoin enää käytetään muuta kuin hajautetun mallin mukaista versionhallintaohjelmistoa.

Molemmissa sekä paikallisessa että keskitetyssä versionhallintamalleissa on ongelmansa joihin hajautettu versionhallintamalli vastaa. Mallin mukaisesta järjestelmästä

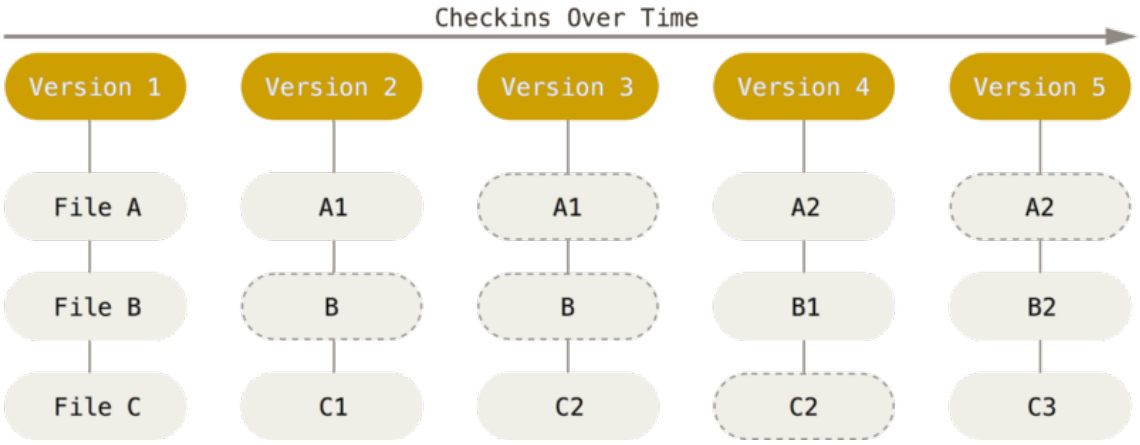
löytyy yhteys versionhallintapalvelimeen ja lisäksi koko tietolähde versiohistoria mukaan lukien on kopioitu paikallisesti kehittäjän tietokoneelle. Tässä yhteydessä puhutaan kloonauksesta. Kehitystä voidaan jatkaa versiohistorian mistä tahansa kohdasta ja kehitystä voidaan myös haarauttaa eli branchata, jos kehittäjä haluaa esimerkiksi kokeilla jotain uutta tapaa toteuttaa jokin toiminnallisuus. Mikäli palvelinyhteys ei toimi, kehittäjä pystyy kuitenkin jatkamaan työtä ja versioida muutoksia tilannevedoksina. Jokainen kloonattu paikallinen tietolähde on käytännössä täydellinen kopio palvelimella olevasta tietolähteestä. Kun kehittäjä haluaa lähettää tiedot palvelimelle, esimerkiksi muiden kehittäjien nähtäväksi tai varmuuskopiointia varten, muutokset synkronoidaan automaattisesti ja mahdollisten konfliktien hallintaan löytyy ratkaisuja. Git-ohjelmisto sisältää kaikki nämä toiminnot. [15]

Gitin merkittävin ero muihin versionhallintajärjestelmiin on sen tavassa hallita tiedostoja ja dataa. Kuviossa 3 on havainnollistettu useimpien muiden versionhallintajärjestelmien tapaa käsitellä tietojen muutoksia aina suhteessa ensimmäiseen versioon. [15]



Kuvio 3. Uusia versioita tallennetaan muutoksina suhteessa tiedoston ensimmäiseen pohjaversioon.

Git puolestaan muodostaa jokaisen kommitoinnin yhteydessä tilannevedoksia projektissa olevista tiedostoista ja kansioista. Yleisesti kun puhutaan kommitoinnista, se tarkoittaa, että tiedosto, kansiot tai muutokset tiedostoissa lähetetään versioitavaksi versionhallintajärjestelmään. Mikäli jokin tiedosto lähteessä ei ole muuttunut, sitä ei tietenkään tallenneta uudelleen, vaan käytännössä luodaan viittaus aikaisempaan tilannevedokseen. Näin Git pysyy myös suorituskyvyltään tehokkaana. Kuvio 4 havainnollistaa Gitin tavan hallita versiointia, katkoviivoin merkityt muutokset pysyvät tilannevedoksissa mukana, mutta viittaavat aikaisempaan versioon. [15]



Kuvio 4. Tiedostojen muutoksien tallentaminen projektin tilannevedoksina.

Lähes jokainen Gitin toiminnallisuus suoritetaan paikallisesti. Se on suuri ero muihin versionhallintajärjestelmiin nähden ja nopeuttaa kehitystyötä merkittävästi. Kehittäjän ei tarvitse odottaa vastausta palvelimelta, kun halutaan versioida muutoksia. Versiohistoria tallennetaan paikalliseen versiotietokantaan, kunnes muutokset työnnetään repositorioon eli säilytyspaikkaan palvelimella. Eri toimintoja ajettaessa versioihin viitataan 40-merkkisen heksadesimaaleja sisältävän SHA-1-tarkistussumman mukaan, tätä voidaan kutsua version tunnisteeksi. Gitin toiminnallisuuksia voidaan suorittaa komentopäätteellä tai graafisilla ohjelmistoilla. Monesti kehittäjät saattavat hyödyntää molempia, yksinkertaiset kommitoinnit ja versiomuutosten työntämiset nopeasti komentopäätteellä. Toisaalta eri versioiden muutosten hallinta ja mahdolliset konfliktit on helpompi tehdä visuaalisesti edistyneempien ohjelmistojen avulla.

Gitissä tiedostolla voi olla kolme tilaa. Nämä ovat kommitoitu, muokattu ja vaiheistettu. Kommitoitu tiedosto on turvallisesti tallennettu versiohistorian paikalliseen tietokantaan tilannevedokseen. Muokattu-tilassa tiedoston tietoja on muokattu, mutta niitä ei ole kommitoitu eli tallennettu tietokantaan. Vaiheistettu-tila tarkoittaa, että muokattu tiedosto on merkitty kommitoitavaksi seuraavaan paikallisen tietokannan tilannevedokseen. [15]

4.2.2 Gitin kehitysmallit

Versionhallinnan vaiheet muodostavat projektille puumaisen kehityshistorian, josta löytyy jokainen kommitointi omassa haarassaan. Projekti voi sisältää useita yhtäaikaisia aktiivisia haaroja, jotka syntyvät toisista haaroista ja jotka voivat yhdistyä takaisin toisiin

haaroihin. Tällaisen kehityshistorian hallinta voi olla hankalaa ilman, että sovitaan yhteisistä työskentely- ja merkitsemistavoista. Gitin vahvuutena on nopea ja kevyt haarojen luonti, joten tämän kynnystä ei kannata korottaa, mutta tietyt hyvät käytännöt pitävät kehityksen eri vaiheet selkeinä.

Gitin omissa oppaissa avataan neljän eri toimintamallin eroja. Monissa isoissa projekteissa käytetään Gitflow-kehitysmallia. Opinnäytetyön projektissa on käytössä Gitflow-kehitysmalli. Tämä kehitysmalli perustuu kahteen jatkuvasti säilytettävään kehityshaaraan, eli aktiivinen kehityshaara (development branch) ja päähaara (master). Näiden lisäksi kehityksen aikana luodaan useita tilapäisiä haaroja. Uusia ominaisuuksia kehitetään luomalla kehityshaarasta uusi ominaisuushaara (feature branch). Näille sovitaan yleensä joitain yhteisiä nimeämiskäytäntöjä. Julkaisut puolestaan viimeistellään omassa julkaisuhaarassa (release branch). Päähaarassa sijaitseva tilanne pyritään pitämään mahdollisimman vakaana ja valmiina julkaisua eli tuotantoon panoa varten. Toimintamallin mukaan päähaaraan ei koskaan tehdä muutoksia suoraan, vaan mahdollisille kiireellisille korjauksillekin luodaan omat haaransa. [16]

Kehityshaarassa sijaitsee aina tuorein kehitysversio projektista. Kun ominaisuuden kehitys on valmis, se yhdistetään historioineen takaisin aktiiviseen kehityshaaraan. Tietysti Git mahdollistaa usean tällaisen haaran kehittämisen samanaikaisesti. Aktiivisen kehityshaaran lähestyessä julkaisukelpoisuutta luodaan erillinen tulevan versionumeron mukainen julkaisuhaara (release branch), johon ei enää uusia ominaisuuksia lisätä vaan ainoastaan korjaukset sallitaan. Kun julkaisuhaara todetaan valmiiksi, yhdistetään se vakaaseen päähaaraan (master) ja samalla tehdään versionumeron mukainen historiamerkintä (tag). [16]

4.2.3 Versionhallinnan säilytyspaikka

Opinnäytetyön projektissa versionhallinnan säilytyspaikan tarjoaa Gitlab-sovellus. Gitlab on versionhallintapalvelimelle asennettu Git-versionhallintaa käyttävä säilytyspaikkojen hallintasovellus. Suurimman osa Gitlabin toiminnoista suoritetaan sen selainsovelluksen kautta, sieltä löytyy esimerkiksi kooditiedostojen historia, muutokset ja versiohallinnan haarat. Vastaava julkinen yleisesti tunnettu sovellus on Github. [16]

Gitlab tarjoaa mahdollisuuden kloonata projekti omaan kehitysympäristöön ja jatkaa kehitystä Git-versionhallinnan toimintojen mukaisesti. Monet yritykset käyttävät usein

Gitlabiä kehitystyössään, sillä se on erityisesti yrityksen omaan ympäristöön suunnattu toisin kuin Github on julkinen. Samalla Gitlab tarjoaa laajempia ominaisuuksia esimerkiksi käyttäjienhallintaan ja on luotettavampi suurien projektien hallinnassa. [17]

4.3 Pakettien hallinta Node-ympäristössä

Usein tutuissa kehitysympäristöissä ja ohjelmakehyksissä on jonkinlainen hallintatyökalu erilaisten laajennusten, komponenttien tai moduulien etsimiseen ja jakamiseen. Työmäärä näiden pakettien etsimiseen voisi olla suuri ilman minkäänlaista pakettienhallinnan työkalua.

Node-ympäristöissä moduulien hallintaan käytetään npm (Node Package Manager) pakettienhallintatyökalua. Npm on ollut mukana Noden julkaisuissa vuodesta 2010 lähtien. Tarkoituksena on tehdä erilaisten komponenttien ja moduulien eli pakettien julkaisusta ja jakamisesta helpompaa. Moduulien asentaminen ja tarvittaessa myös poistaminen on yksinkertaista ja nopeaa. Moduulit ovat käytännössä avoimen lähdekoodin moduuleita, sillä npm asentaa ne ohjelmakoodeineen kehitysympäristöön ajettavaksi. Npm:stä on tullut suosittu työkalu, rekisteristä löytyy tällä hetkellä lähes 340 000 pakettia. Useimmat niistä tuovat jonkin suhteellisen yksinkertaisen toiminnon tai työkalun projektiin, mutta löytyy myös laajempia kokonaisuuksia kuten edellä insinöörityössä mainittu Express-moduuli. Erilaiset paketit mahdollistavat monimutkaisenkin sovelluksen rakentamisen fiksusti omia ja kolmannen osapuolien toteuttamia moduuleita yhdistämällä. [18]

Paketin asentaminen npm:n avulla on yksinkertaista ja nopeaa. Käytännössä kehittäjä ajaa komennon *npm install <nimi> --save* -komennon, jossa annetaan paketin nimi parametrina ja tieto tallennuksesta attribuuttina. Työkalu etsii rekisteristä nimeä vastaavan paketin ja asentaa uusimman version. Lisäparametreilla voidaan määrittää tietty versionumero paketista ja tarvittaessa jopa tarkka osoite git-säilytyspaikkaan, mahdollistaen näin myös yksityisten pakettien käytön. Julkisia paketteja voi etsiä *npm search* -komennolla ja rekisteri löytyy myös internetistä npm:n omilta sivuilta.

Npm:n avulla asennettavia paketteja ei ole tarvetta laittaa mukaan versionhallintaan. Ainoastaan pakettien hallinnan käyttämä package.json -tiedosto kannattaa versioida.

Konfigurointitiedosto sisältää tiedot sovelluksessa käytetyistä paketeista sekä sovellukseen liittyviä tietoja kuten sovelluksen nimen, version ja käytetyt kehitysympäristöt.

Uuden kehittäjän tullessa mukaan projektiin hän ajaa kloonauksen jälkeen *npm install* -komennon ja näin kaikki projektissa käytössä olevat paketit asentuvat mukaan kehittäjän ympäristöön. Mikäli paketin käytöstä halutaan luopua, kehittäjä kirjoittaa komennon *npm uninstall <nimi> --save*. Näin paketti poistuu kehittäjän ympäristöstä ja samalla muutos tallennetaan myös työkalun käyttämään package.jsoniin. Tämän jälkeen muutos kannattaa viedä versionhallintaan, näin muut kehittäjät saavat tiedon paketin poistamisesta ja projektin kehityshistoriaan jää tieto. [19]

Asennettaessa pakettia tallennukseen viittaava *--save* -attribuutti huolehtii paketin nimen ja versionumeron tallentamisesta työkalun käyttämään package.json -tiedostoon. Yksittäinen pakettikin voi sisältää omia viittauksia toisiin paketteihin, aivan samalla tavalla package.json -tiedostoon konfiguroituna. Nämä viittaukset voidaan havainnollistaa puunäkymänä *npm ls* -komennolla.

Useimmiten paketit noudattavat internetissä julkaistun semanttisen versionumeroinnin ohjeistusta, joka määrittää tiettyjä käytäntöjä paketin päivityksille. Järjestelmän, joka nojaa moniin moduuleihin, päivitys ja jatkokehitys voi helposti muodostua haastavaksi prosessiksi. Tätä helpottamaan on luotu ohjeistus sovelluksen osien rajapintayhteensopivuuden säilyttämisen käytäntöihin versionumeroa kasvatettaessa. Semanttisen versionumeroinnin ohjeistuksen mukaan versionumeroon sisältyy kolme pisteellä eroteltua osaa, esimerkiksi 3.12.5. Nämä ovat pää- (major), ala- (minor) ja paikkaustason (patch) versionumerot. Paikkaustason versionumeroa kasvattava julkaisu ei odoteta tekevän muutoksia rajapintaan. Alatason versionumeroa kasvattava julkaisu saattaa toteuttaa lisäyksiä rajapintaan, mutta sen odotetaan kuitenkin säilyttävän yhteensopivuuden aikaisempiin versioihin. Päätason versionumeroa kasvattavan julkaisun voi jo odottaa tuovan merkittäviä muutoksia rajapintaan, se saattaa myös rikkoa yhteensopivuutta aikaisempiin versioihin. [20]

Ulkopuolisia moduuleja asennettaessa ohjelmistokehittäjän kannattaa seurata moduulin päivityshistoriaa ja tutustua ainakin pintapuolisesti moduulin tarjoaman rajapinnan eri versioihin. On syytä huomata esimerkiksi, että yleensä 1.0.0 versiota aikaisemmat versiot ovat kehitysvaiheen versioita. Myöhemmin kehitysvaiheessa tulee todennäköisesti eteen tilanne, jossa moduuli on päivittynyt, tällöin versionumeron tasojen muutok-

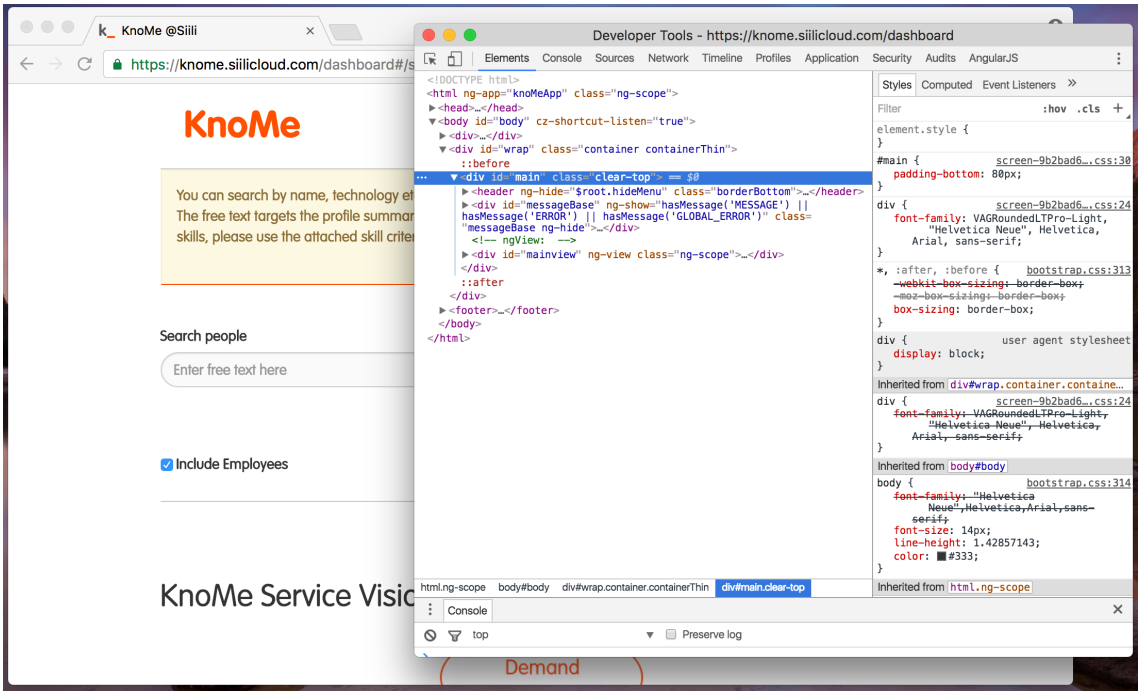
sia tarkastelemalla ohjelmistokehittäjä pystyy arvioimaan päivityksen vaativuutta. On suositeltavaa, että myös toteutettava sovellus noudattaa semanttisen versioinnin ohjeistusta. Se pitää sovelluksen kehityksen selkeänä ja helpottaa ulkopuolisten tutustumista ja jatkokehittämistä sovelluksen suhteen. [20]

Npm asentaa automaattisesti paketin uusimman julkaistun versionumeron mukaisen version. Asennuskomennossa voi myös määrittää versionumeron halutulla tarkkuudella. Sovelluskehittäjä voi esimerkiksi vertailuoperaattorien ja muiden merkkien avulla määrittää halutun tai sitä suuremman tai pienemmän versionumeron paketista. Esimerkiksi merkinnällä *>=1.2.0* paketista saadaan 1.2.0 versio tai sitä versionumeroa suuremmat versiot. Tähti- (*), hattu- (^) ja tilde-merkillä (~) ja x-kirjaimella voidaan määrittää, minkä versionumeroinnin tason päivitykset voidaan asentaa ympäristöön. [19]

Opinnäytetyön projektissa on esimerkiksi Express-moduulista *4.13.x* -merkinnällä määritetty sallittavan ainoastaan 4.13 versioon paikkaustason päivitykset. Tällä tavalla on varmistettu, että moduuli päivittyy uudempaan paikkaustason versioon, kun niin halutaan. Pää- tai alatason versionumeron mukaiset päivitykset eivät kuitenkaan asennu yllättäen.

4.4 Selaimen kehittäjätyökalut

Selaimen tarjoamat kehittäjätyökalut ovat hyödyllisiä, kun halutaan selvittää websovelluksen ominaisuuksia, elementtejä, sovelluskoodia, evästeitä ja ajon aikaisia tapahtumia sekä suorituskykyä. Joihinkin selaimiin saa erilaisia laajempia työkaluja, jotka helpottavat kehittämisprosessia monin eri tavoin. Googlen Chrome-selaimesta löytyy valmiiksi asennettuna kattavat ja helposti käytettävät kehittäjätyökalut. Kuviossa 5 on Chrome-selaimen kehittäjätyökalun Elements-näkymä avattuna omaan ikkunaansa.



Kuvio 5. Google Chrome -selaimen kehittäjätyökalun Elements-näkymä oikealla omaan ikkunaan avattuna.

Elements-näkymä tarjoaa kehittäjälle sivun lähdökoodin alkaen juuresta, josta kehittäjä pystyy porautumaan syvemmälle sivun rakenteeseen. Kehittäjä voi näin arvioida, onko sivu toteutettu rakenteeltaan oikein. Oikean elementin löytämistä helpottaa se, että sivun renderoidun näkymään muodostuva korostus valitun elementin kohdalle. Renderoidusta näkymästä voi myös valita tietyn elementin tarkistettavaksi, se avautuu suoraan kehittäjätyökalun elementit-näkymään. Sovelluksen toiminnallisuuden ja käytettävyyden kannalta oikean ulkoasun viilaukseen tämä näkymä soveltuu erittäin hyvin, kehittäjälle tarjotaan mahdollisuus muokata ja lisätä elementtien tyylimäärityksiä ja nämä muutokset näkyvät heti sivun renderoidussa näkymässä. Jos kehittäjä huomaa esimerkiksi jonkun painikkeen tai kuvan olevan väärässä kohdassa, voi hän kokeilla kehittäjätyökalulla erilaisia tyylimäärityksiä. Tämän jälkeen hän voi toteuttaa muutoksen oikealla tavalla sovelluksen tyylitiedostoihin.

Kehittäjätyökalu tarjoaa myös mahdollisuuden tarkastella ja muokata sovelluksen javascript-koodia. Sovelluksen koodiin voi merkitä pysähtymispisteitä, jotka saavat suorituksen pysähtymään ajonaikaisesti ja näin kehittäjä voi tarkastella muuttujia sekä koodin toimivuutta. Javascript-koodiin voi tehdä myös muokkauksia, joka voi olla toiminnallisuuden muutoksen tai korjauksen testaamisen kannalta hyödyllistä.

AngularJS:n avulla rakennettu web-sovellus voi kasvaa hyvinkin laajaksi sovellukseksi ja kaksisuuntaisten sitomisten (binding) määrittely voi muodostua todella raskaaksi. Google Chrome -selaimen kehittäjätyökalu tarjoaa suoraan ilman lisäosia työkaluja AngularJS:n käytön tarkasteluun. AngularJS luo kaksisuuntaisen sidonnan määrittelyn yhteydessä aina tarkastelijan (watcher) kyseessä olevaa arvoa vasten, näiden lukumäärän kasvaessa sovelluksen suorituskyky voi kärsiä. Sovelluksen ajonaikaisesti luotujen tarkastelijoiden määrää, suoritusaikaa ja muita tietoja voi tarkastella kehittäjätyökalulla, näin kehittäjä voi arvioida, löytyykö toista tapaa toteuttaa sitominen. Tämä voidaan AngularJS:ssä ratkaista esimerkiksi yksisuuntaisella kertaluontoisella arvon tai tiedon esittämisellä. Näin ei jäädä tarkastelemaan, tuleeko arvoon muutoksia.

Usein työkalut tarjoavat myös mahdollisuuden tarkastella sovellusta erilaisten laitteiden mahdollistamien ikkunakoon mukaisesti, esimerkiksi iPhone tai Android puhelimen näytön koon mukaan. Näin voidaan varmistaa sovelluksen responsiivisuus eri laitteilla ja ikkunan koon mukaan. Yleisesti selaimen kehittäjätyökalut ovat käytännöllisiä sovelluksen toimivuuden ja ominaisuuksien tarkastelussa ja tarvittaessa myös korjauksen etsimisessä.

4.5 Kehitysympäristön hallinta

Minkä tahansa sovelluksen laajetessa useamman kehittäjän projektiksi, kehitysympäristössä käytettyjen komponenttien ja ohjelmistojen sekä määritysten asettaminen ja hallinnointi käyvät yhä vaativammiksi. Opinnäytetyön sovelluksen kehitysympäristö sisältää useita toimintoja, joilla sovellus saadaan kehittäjän tietokoneella toimimaan niin kuin sovellusta ajettaisiin oikeasti palvelimelta. Tällaisen kehitysympäristön rakentamiseen uudelle kehittäjälle voisi käydä työlääksi ilman prosessia helpottavaa sovellusta. Vagrant tarjoaa ratkaisun kehitysympäristön luomiseen ja hallintaan.

Vagrant on avoimen lähdekoodin sovellus siirrettävien virtuaalisten kehitysympäristöjen luomiseen ja hallinnointiin. Sovelluksen ensimmäinen versio julkaistiin keväällä 2010 sivuprojektina. Kaksi vuotta myöhemmin sovelluksesta julkaistiin ensimmäinen stabiili versio, sovelluksen ympärille muodostettiin yritys ja kehittäjäyhteisö kasvoi suuremmaksi. Sovelluksen tarkoituksena on kehitysympäristön hallinnan helpottaminen hyödyntäen virtuaalista ympäristöjä ja muita toimintoja. Ongelmana on, että ympäristöjen hallinta kasvaa suurissa projekteissa hankalammaksi ja hankalammaksi, prosessit si-

sältävät useita teknisiä suoritteita. Vagrant helpottaa tässä keräten kaikki asentamiseen ja hallinointiin vaadittavat määritykset tiettyjen suoritteiden sisään. [22]

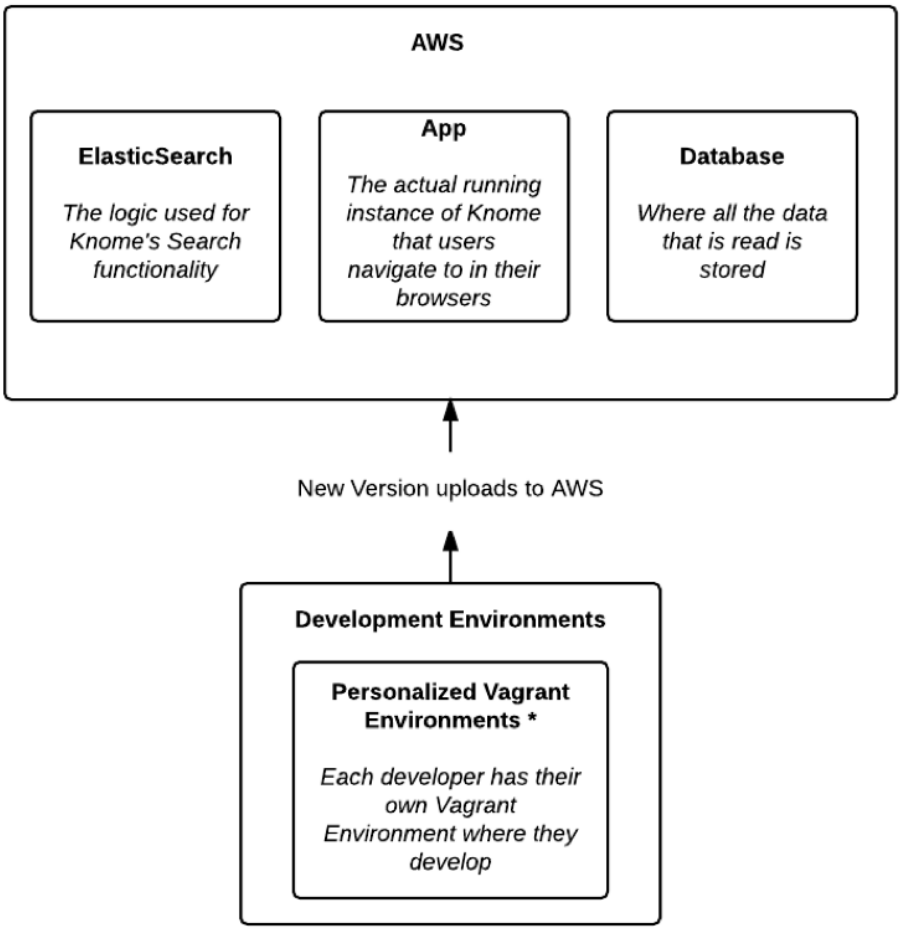
Opinnäytetyön ympäristössä Vagrantin suoritteeseen on kerätty virtuaalisen palvelimen käynnistämiseen liittyvät komennot, CouchDB-tietokannan määritykset ja testidatan luomiset, Noden pakettien asentamiset ja ympäristömuuttujien asettamiset. Opinnäytetyön sovellus asennetaan palveluna ja käynnistetään. Lisäksi Vagrant asentaa ja käynnistää ElasticSearch-sovelluksen ja muita palveluita. Nämä kaikki suoritteet on kerätty Vagrantfile-tiedostoon ja riittää, että kehittäjä vain suorittaa *vagrant up* -komennon.

4.6 Sovelluksen julkaisu

Opinnäytetyön sovelluksen julkaisussa ja tuotannon version ajossa hyödynnetään Amazon Web Services -alustan palveluita (AWS). AWS on kokoelma tietojenkäsittelyresursseihin liittyviä palveluja, jotka ovat tarjolla Amazonin pilvessä. Amazon tarjoamista pilvipalveluista keskeisin ja tunnetuin on Elastic Computer Cloud (EC2), joka tarjoaa palvelininstanssin käyttöjärjestelmineen ja sovelluksineen. Lisäksi projektissa on käytössä Route 53 -palvelu, joka tarjoaa nimipalvelujen hallinnan. Kaikkia Amazonin tarjoamia pilvipalveluita yhdistää skaalautuvuus, korkea saavutettavuus ja suuren datamäärän mahdollistava laskentakapasiteetti sekä helppokäyttöinen web-käyttöliittymä.

Amazonin pilvialustan palvelut tulivat markkinoille vuonna 2006. Ensin julkaistiin Simple Storage Service (S3) -palvelu, joka tarjoaa tietojen tallentamiseen ja jakamiseen suunnatun skaalautuvan tallennustilapalvelun. Samana vuonna palvelutarjontaan tuli myös EC2. Amazonin pilvipalveluita käyttää monet suuret yritykset kuten Netflix, Slack ja Airbnb. Tuotteiden skaalautuvuus, helppokäyttöisyys ja selkeä hinnoittelu ovat olleet merkittävimpiä syitä suosion kasvuun. EC2-palvelininstanssi voidaan esimerkiksi valita hinnoiteltavaksi toteutuneen käytön mukaan. [23]

Käytännössä sovelluksen uuden julkaisukelpoisen version käyttöönotto tapahtuu Amazonin web-käyttöliittymässä EC2-hallintapaneelin avulla. Samasta paneelista löytyy myös tietokannan ja ElasticSearch-sovelluksen hallintaan liittyviä työkaluja. Kuvio 6 havainnollistaa sovelluksen arkkitehtuurin, sovellus pyörii varsinaisesti Amazonin pilvipalvelussa, mutta kehittäjällä on myös oma ympäristö.



Kuvio 6. Opinnäytetyön sovelluksen arkkitehtuuri. Julkaistava versio sovelluksesta viedään Amazonin pilvipalveluun, jossa pyörii myös tietokanta ja ElasticSearch-sovellus. Kehittäjien omissa kehitysympäristöissä on Vagrant-sovelluksen avulla luotu ympäristö, jossa sovelluskehitys onnistuu kuin sovellusta ajettaisiin palvelimella.

Amazonin kaltaiset skaalautuvat ja helppokäyttöiset palvelinalustat ovat tuoneet Serverless-ajattelutavan. Aikaa ennen pilvipalveluita palvelimien hallintaan käytettiin paljon resursseja, yrityksen täytyi esimerkiksi huolehtia oikeanlaisten palvelimien hankinnasta, kaapeloinnista, konesalista ja monesta muusta asiasta tarjotakseen laajoja digitaalisia palveluita käyttäjilleen. Pilvipalvelujen ensimmäiset ratkaisut tarjosivat infrastruktuuria palveluina. Sovelluskehittäjien on kuitenkin hankala suhteuttaa rakentamiaan ratkaisuina palvelinten määrään, ehkä siksi modernit pilvipalvelut eivät välttämättä tarjoa enää pelkästään yksittäisiä palvelimia vaan tietojenkäsittelyyn liittyviä resursseja skaalautuvasti sovelluksen tarpeen mukaan.

Nimensä mukaisesti ”Serverless” ei tarkoita, ettei palvelimia enää käytetä tai tarvita. Yksinkertaisesti ajattelutavalla tarkoitetaan, ettei kehittäjän tarvitse paljoa välittää pal-

velimista. Tietojenkäsittelyn resurssit ostetaan palveluna tarpeen mukaan ilman, että kehittäjän tarvitsee miettiä rajoitteita. Pilvipalvelut tarjoavat skaalautuvuuden, suuren datamäärän käsittelyn, suorituskykyistä laskentatehoa ja niin edelleen. Palveluntarjoaja puolestaan ottaa vastuun palvelimien hallinnasta, tietolähteistä ja muusta infrastruktuurista.

5 Yhteenveto

Kaiken kaikkiaan tässä opinnäytetyössä on tutustuttu tekniikoihin, joilla rakentaa niin pienimuotoisia web-sovelluksia kuin monipuolisia digitaalisia palveluita. Työssä läpikäytyihin tekniikoihin on tuotu esimerkkiä ja vertailupohjaa Siili Solutionsin käyttämän ja kehittämän web-sovelluksen kautta.

Esimerkkinä käytetty sovellus ja sen kehitysympäristö ovat monella tapaa tyypillisiä web-sovelluskehityksessä. Tässä yhteydessä puhutaan termistä MEAN-stack, eli yhdistelmästä, jossa ovat käytössä MongoDB, Express.js, AngularJS ja Node.js. Opinnäytetyön sovelluksessa MongoDB:n sijaan käytetään toista NoSQL-pohjaista tietokantaa, CouchDB:tä, mutta idea on joka tapauksessa sama. MEAN-stack on Full-Stack kehittäjän työkalupakki, joka kattaa tekniikat sekä front end että back end -puolelta ja mukana myös tietokannan sovellutus. MEAN-stack työkalupakin tekniikat ovat myös mahdollistaneet erilaisten sovellusten prototypoinnin selaimessa suoritettuna. Kyseessä voi olla selain- tai työpöytäsovellus, tai jopa sulautettu järjestelmä.

Sovelluksen testaus on puoli, joka jätettiin opinnäytetyössä käsittelemättä. Web-sovellusten ohjelmistokehityksessä testaus ei välttämättä ole aikaisemmin ollut kovin suuressa roolissa. Nykyään on testaukseen kuitenkin tarjolla erilaisia työkaluja. Parhaimmillaan testaussovellukset käyvät läpi myös selaimessa suoritettavaa käyttöliittymäsovellusta. Testauksessa olennainen osa on myös rajapintojen ja taustalogiikan yksikkötestaus, palvelinsovelluksen ympärille voidaan rakentaa yksikkötestejä vähällä vaivalla erilaisia työkaluja hyödyntäen. Web-sovellusten testaamisessa tunnetuimmat työkalut ovat Karma ja Jasmine. Jälkimmäinen on etenkin AngularJS sovellusten testaamisessa usein käytetty. Molemmissa testien logiikan kirjoittaminen tapahtuu JavaScript-koodikielellä.

Työn tekemisen yhteydessä myös web-sovelluskehityksen historian läpikäynti muodostui mielenkiintoiseksi tarkastelun kohteeksi. Web-sovelluskehityksessä käytetään nykyään monia muun tyyppisten sovellusten kehityksestä tuttuja suunnittelumalleja ja työkaluja. On tärkeää huomata, miten nämä tekniikat ovat kehittyneet vuosien varrella palvelemaan web-sovellusten tarpeita. Versiohistoriamalli on kehittynyt hajautetuksi, jossa kehittäjät vievät sovelluksensa GitHubiin tai muuhun pilvessä sijaitseviin versiohistoriasovellutuksiin ja samalla versiohistorian avulla työskentely on yhä varmempaa, nope-

ampaa ja helpompaa. Tietokannat ovat kehittyneet myös hajautetun mallin suuntaan palvelemaan suuria käyttäjämassoja ja datamääriä.

Opinnäytetyön esimerkkinä käytetty sovellus on jatkuvassa kehityksessä ja ketteriä menetelmiä hyödynnetään. Opinnäytetyöstä saa hyvää pohjaa web-sovelluskehityksen ratkaisuihin ja esimerkkejä yleisesti käytetyistä tekniikoista.

Lähteet

1. Keith, Jeremy. 2010. HTML5 for Web Designers. A Book Apart.

2. Lawson, Bruce & Sharp, Remy. 2011. Introducing HTML5. New Riders.

3. Cederholm, Dan. 2010. CSS3 for Web Designers. A Book Apart.

4. About JavaScript - JavaScript | MDN. Verkkodokumentti. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>. Luettu 28.9.2016.

5. Champeon, Steve. 2001. JavaScript: How Did We Get Here? Verkkodokumenti. <http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html>. 6.4.2001. Luettu 1.10.2016.

6. JavaScript HTML DOM. Verkkodokumentti. W3C. <http://www.w3schools.com/js/js_htmldom.asp>. Luettu 1.10.2016.

7. Osmani, Addy. 2015. Learning JavaScript Design Patterns. Verkkodokumentti. <https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html>. Luettu 1.10.2016.

8. Ambler, Tim & Cloud, Nicholas. 2015. JavaScript Frameworks for Modern Web Dev. Appress.

9. Freeman Adam. 2014. Pro AnguarJS. Appress.

10. A JavaScript library for building user interfaces | React. Verkkodokumentti. <https://facebook.github.io/react/>. Luettu 2.10.2016.

11. Read Me - Redux. Verkkodokumentti. <http://redux.js.org>. Luettu 2.10.2016.

12. Inc. Joyent. About Node.js. Verkkodokumentti. <https://nodejs.org/en/about>. Luettu 8.10.2016.

13. Anderson, J. Chris & Lehnardt, Jan & Slater, Noah. 2010. CouchDB - the definitive guide. O'Reilly.

14. Thompson, Mick. 2011. Getting started with GEO, CouchDB & Node.js. O'Reilly.

15. Chacon, Scott & Straub, Ben. 2014. Pro Git. Apress. <https://git-scm.com/book/en/v2>. Luettu 15.10.2016.

16. Git. Git Flow. <https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>. Luettu 22.10.2016.

17. Features | Gitlab. Verkkodokumentti. <https://about.gitlab.com/features/>. Luettu 22.10.2016.

18. Kennedy, Hugh & DeVay, Paul. Understanding npm. Verkkodokumentti. <https://unpm.nodesource.com>. Luettu 23.10.2016.

19. npm Documentation. Verkkodokumentti. <https://docs.npmjs.com>. Luettu 23.10.2016.

20. Preston-Werner, Tom. Semantic Versioning. Verkkodokumentti. <http://semver.org>. Luettu 25.10.2016.

21. Gormley, Clinton & Tong, Zachary. ElasticSearch: Definitive Guide. O'Reilly.

22. About - Vagrant by HashiCorp. Verkkodokumentti. <https://www.vagrantup.com/about.html>. Luettu 2.11.2016.

23. Cloud Products - Amazon Web Services. Verkkodokumentti. <https://aws.amazon.com/products>. Luettu 2.11.2016.