

Antti Sirkiä

Modernin verkkosovelluksen kehittäminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

3.5.2015

Tekijä Otsikko	Antti Sirkiä Modernin verkkosovelluksen kehittäminen
Sivumäärä Aika	27 sivua 3.5.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Lehtori Olli Alm
<p>Insinööriyössä selvitettiin, kuinka ja minkä työkalujen avulla moderni verkkosovellus voidaan kehittää. Lisäksi työssä selvitettiin modernin ja perinteisen verkkosovelluksen toiminnallisuuden eroavaisuuksia ja mitä hyötyä sovelluskehittäjän kannalta moderni verkkosovellus ja sen mahdollistavat teknologiat tarjoavat. Työssä selvitettiin myös, mitä lisäarvoa modernin verkkosovelluksen kehittämisessä käytettävät työkalut tuovat sovelluksen kehittämisprosessiin.</p> <p>Työssä kehitettiin esimerkisovelluksena moderni verkkosovellus, jonka avulla käyttäjä pystyy luomaan verkossa kyselyitä, joihin loppukäyttäjät pystyvät vastaamaan anonyymisti reaaliajassa. Kehitetyn sovelluksen teknologioiksi valittiin moderneihin verkkosovelluksiin soveltuvat uusimmat teknologiat työn osana kehitetyn verkkosovelluksen vaatimusmääritelmien perusteella. Työssä kehitetyssä verkkosovelluksessa pyrittiin ohjelmistokehittäjän näkökulmasta mahdollisimman yksinkertaiseen ja prosessointitehovaatimuksiltaan kevyeen ratkaisuun. Sovelluksen kehitysprosessia pyrittiin automatisoimaan ja helpottamaan mahdollisimman paljon uusimpien sovelluskehitystyökalujen avulla.</p> <p>Kehitetyn sovelluksen kokemukset osoittivat modernin verkkosovelluksen kehityksessä käytettävien työkalujen helpottavan ja nopeuttavan sovelluskehittäjän työtä. Modernin verkkosovelluksen käyttämien teknologioiden edut todettiin hyödyllisiksi, ja niiden todettiin olevan sovelluksen ylläpitäjän ja omistajan kannalta kustannustehokkaita. Työssä todettujen tulosten mittausta perustuu sovelluskehittäjän kokemuksiin, eivätkä ne ole tarkkaan mitattavissa.</p>	
Avainsanat	Single-Page Application, moderni verkkosovellus, JavaScript, HTML5

Author Title	Antti Sirkiä Creating a modern web application
Number of Pages Date	27 pages 3 May 2015
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Olli Alm, Lecturer
<p>This thesis explored how and with what tools a modern web application can be developed. This thesis also describes the differences between a modern and a traditional web application and what kind of benefits a modern web application provides and how its technologies benefit a software developer. The added value of the tools used in modern web application development are evaluated and demonstrated.</p> <p>A working example of a modern web application was developed as a part of this thesis. The application enables a user to create online polls that the end users can answer anonymously in real-time. The newest modern web application technologies used in the application were chosen based on the application's requirements specification. The application in question aimed to be as simple as possible and to use as little processing power as possible. The tools used in the example application aimed to automate the development process and to make the software developer's easier and faster.</p> <p>The experience gained from developing the example application showed the use of modern technologies and tools to make the software developer's job easier and faster. The technologies provided by a modern web application were found to be valuable and cost effective from both the developer's and the product owner's perspective. The results found in this thesis are based on the software developer's experience and are not measurable.</p>	
Keywords	Single-Page Application, modern web application, JavaScript, HTML5

Sisällys

1	Johdanto	1
2	Modernit ja perinteiset verkkosovellukset	2
2.1	SPA-sovellus	2
2.2	Verkkosovellukset ilman SPA-sovellusarkkitehtuuria	3
2.3	SPA-sovelluksena toteutetut verkkosovellukset	5
3	SPA-sovelluksen kehittämisessä käytettävät työkalut	6
3.1	Käytettävät työkalut	6
3.1.1	Kehityspalvelin	7
3.1.2	Paketinhallinta	10
3.1.3	Työlienhallinta	12
3.1.4	Automaattinen lähdekoodin tarkistus	13
3.1.5	Tuotantoon vieminen	15
3.2	Kehitysympäristö	16
3.3	Ohjelmistokehykset	17
4	SPA-sovellus käytännössä	19
4.1	Sovelluksen vaatimukset ja toteutus	19
4.2	Ongelmakohdat	24
4.3	Mahdollisuudet jatkon kannalta	25
5	Yhteenveto	27
	Lähteet	28

1 Johdanto

Insinööriyön tavoitteena on selvittää ohjelmistokehitykseen soveltuva kokonaisuus aputyökaluista, joiden avulla voidaan luoda SPA-sovellus (engl. Single Page Application), ja se, mitä hyötyä sillä saavutetaan. SPA-termiä käytetään viittaamaan moderniin verkkosovellukseen. Insinööriyön osana kehitetään kyselytyökalu, jonka avulla voidaan luoda kyselyjä, joihin käyttäjät pystyvät anonyymisti vastaamaan reaaliajassa.

Verkkosovellukset ja varsinkin niiden kehittämisessä käytetyt työkalut, teknologiat ja ohjelmointikielet kehittyvät jatkuvasti (1). Internetin kautta voi nykyään ostaa lähes mitä tahansa. On mahdollista tilata autovakuutus tai vaikkapa kotiin sähkö internetistä, ja tämä kaikki onnistuu jonkin päätelaitteen avulla, jossa on internetyhteys. Interaktioita ihmisten kanssa tai jonottamista liiketiloissa ei tarvita, vaan palvelut toimivat mutkattomasti verkossa. Modernien verkkosovellusten ja asiakkaiden kasvavien tarpeiden sekä heidän odottamansa palveluntason nousun vuoksi SPA-sovellusten suosio on kasvanut (2). SPA-sovelluksen avulla käyttökokemus on mahdollista luoda hyvin sulavaksi ja intuitiiviseksi, oli päätelaite sitten tietokone tai mobiililaitte (3). SPA-sovelluskehitys on myös poikinnut lukuisia aputyökaluja, joiden lyhenteitä työpaikkailmoitukset vilisevät. Tällä alati muuttuvalla ja kehittyvällä alalla on siis ainakin tällä hetkellä runsaasti työmahdollisuuksia. Insinööriyössä perehdytään siihen, mitä lisäarvoa SPA-sovelluskehityksessä käytettävät aputyökalut tuovat sovelluskehitykseen.

Tuotteiden ja palvelujen myynnin yleistyttyä verkossa on luontaista, että myös verkkosovellusten laatu paranee kilpailun mukana. Internet-pohjaisessakin myynnissä on tärkeää luoda helppo ja sulava käyttökokemus, jolloin verkkosovellusten laadun on oltava hyvä ja palvelun on toimittava virheettömästi. SPA-sovellusten nopean ja ketterän kehityskaaren ansiosta asiakaslähtöisen palvelun kehittäminen on mahdollista, etenkin huomioiden SPA-sovelluksen tarjoamat mahdollisuudet sovelluksen luomiseen, ylläpitämiseen ja jatkokehitykseen.

2 Modernit ja perinteiset verkkosovellukset

2.1 SPA-sovellus

SPA:lla tarkoitetaan modernia verkkosovellusta, jossa käyttökokemus muistuttaa natiivisovellusta (4, s. 3). Natiivisovelluksella tarkoitetaan ohjelmistoa, joka on asennettuna esimerkiksi tietokoneelle ja toimii yhtenä itsenäisenä kokonaisuutena (4, s. 10). Perinteisen verkkosovelluksen käyttökokemus on eronnut natiivisovelluksen käytöstä siten, että perinteinen verkkosovellus joutuu tekemään palvelimelle HTTP-pyyntöjä, joiden avulla verkkosovelluksen näkymä tai tiedot päivittyvät (4, s. 4, 7). SPA-sovelluksessa kokonaisuuden toiminnan kannalta olemassa olevia tiloja käyttäjä ei huomaa, vaan hän pystyy yksinkertaisesti aloittamaan sovelluksen käyttämisen ja lopettamaan sen käyttämisen. Koska JavaScript-ohjelmointikielen ylläpidettävyys laajassa sovelluksessa käy nopeasti haastavaksi, SPA-sovellukset tyypillisesti käyttävät sovelluskehityksessä jotain ohjelmistokehystä, joka luo sovelluksen lähdekoodille jonkin tietynlaisen rakenteen. SPA-sovellus voi terminä olla monelle internetissä liikkuvalla tuntematon, mutta hyvin todennäköisesti suurin osa internetin käyttäjistä on törmännyt SPA-sovellusarkkitehtuurilla toteutettuun verkkosovellukseen.

SPA-sovelluksessa kaikki sisältö voidaan ladata käyttäjän päätölaitteelle yhdellä kertaa tai mahdollisesti paloina. Palojen tulee kuitenkin muodostaa yksi kokonaisuus sen sijaan, että sovellus koostuisi useammasta kokonaisuudesta tai aliohjelmasta. SPA-sovellus sisältää kaikki mahdolliset näkymät, eikä niitä tarvitse erikseen generoida palvelimella. Nämä näkymät kuitenkin usein käytännössä luodaan reaaliajassa, mutta erillisiä HTTP-pyyntöjä palvelimelle ei tarvitse tehdä. SPA-sovelluksena toteutetuissa sovelluksissa on aina olemassa jonkinlainen tila, mutta tämä tila ei välttämättä ilmene loppukäyttäjälle millään tavalla. Sovelluksen tilalla tarkoitetaan jonkin tapahtumaketjun välivaihetta. Ohjelmistokehityksen kannalta tämä tila on oleellinen tieto, jonka avulla tehdään loogisia operaatioita käyttäjän tekemien toimintojen perusteella. Tunnetuista SPA-sovelluksista mainittakoon esimerkiksi Gmail, joka oli aikansa suunnannäyttävä SPA-sovellusten suhteen (3).

SPA-sovelluksena toteutetun verkkosovelluksen sisällön ja datan synkronisointi palvelimen kanssa ei välttämättä ole asynkronista, kuten vanhemmissa verkkosovelluksissa oli tapana, vaan sisältö ja data voidaan synkronisoida dynaamisesti käyttäjän tarpeiden ja

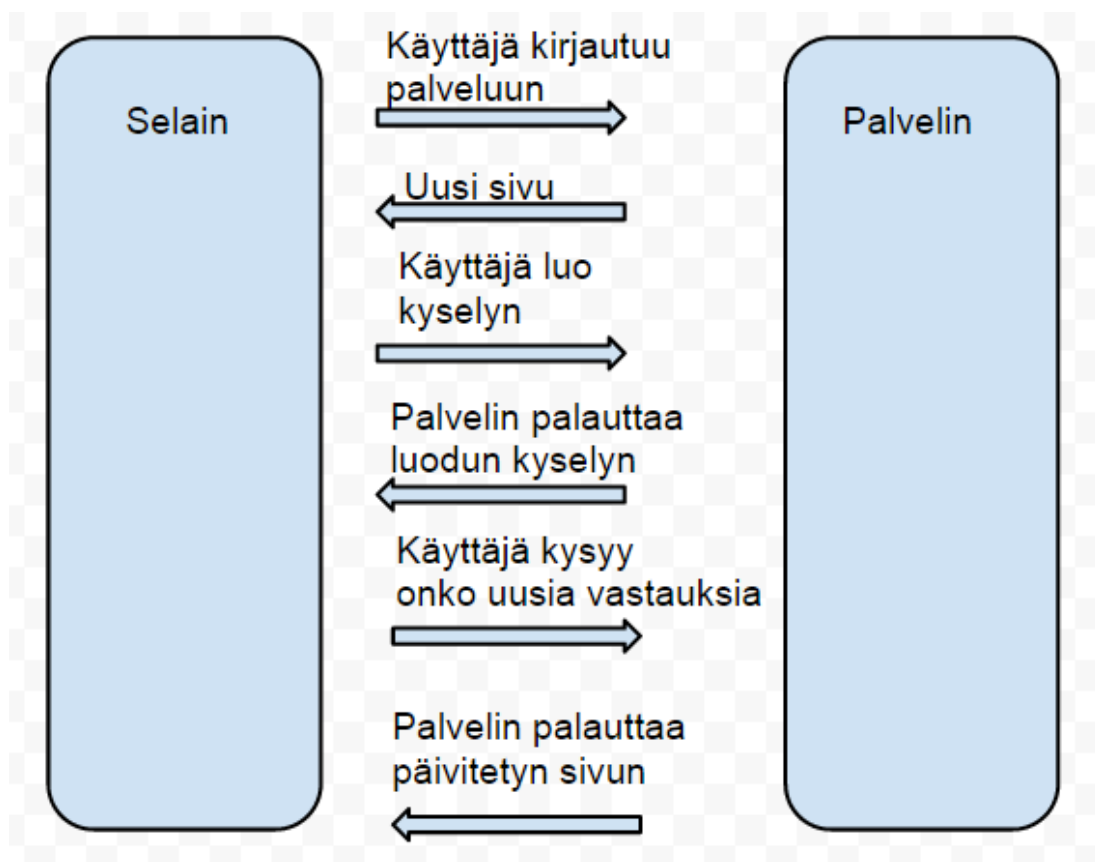
sovelluksen tilan mukaan. Tämä tekee käyttökokemuksesta sulavan, eikä käyttäjä tunne siirtyvänsä verkkosivulta toiselle sivulle, vaikka todellisuudessa käytettävä sovellus tai sivusto olisi monimutkainen ja sisältäisi lukuisia näkymiä tai tiloja (3). SPA-sovellus voi käytännössä sisältää äärettömän määrän erilaisia näkymiä ja tiloja, jotka luodaan dynaamisesti käytön aikana, eli SPA-sovelluksen käyttötarkoitus ei rajoitu pelkästään verkkosivuihin tai -sovelluksiin, jotka kirjaimellisesti koostuvat yhdestä sivusta tai näkymästä, kuten nimi antaa ymmärtää. Esimerkiksi SPA-sovelluksessa jokin verkkolomake voidaan generoida dynaamisesti käyttäjän ennalta syöttämien tietojen perusteella, kun perinteisessä verkkosovelluksessa palvelin joutuisi generoimaan lomakkeen ja lähettämään sen staattisena HTML:nä loppukäyttäjälle.

Uusien SPA-sovelluksena toteutettujen verkkosovellusten määrä on lisääntynyt nopeammin kuin perinteisten verkkosovellusten määrä (2). SPA-sovelluksen etuihin voidaan laskea käyttäjän mahdollisuus käyttää verkkosovellusta ilman internet-yhteyttä ja sen natiivisovellusta muistuttava käyttökokemus. Lisäksi on huomioitava SPA-sovelluksen kuormittavan palvelinta huomattavasti vähemmän perinteiseen verkkosovellukseen verrattuna. Ohjelmistokehittäjän kannalta etuihin voidaan laskea ketterä kehitys ja hyvä ylläpidettävyys. SPA-sovelluksena kehitetty verkkosovellus pystyy synkronisoimaan datansa palvelimen kanssa, kun internetyhteys on käytettävissä, tai vaihtoehtoisesti tallentamaan tietonsa loppukäyttäjän laitteelle (4, s. 12). Käytännössä tämä johtuu siitä, että koko MVC (Model, View, Controller)- tai jokin muu ohjelmistoarkkitehtuurimalli on käyttäjän internetselaimessa eikä osittain palvelimella (4, s. 49, 51). Perinteisesti kehitetyissä verkkosivuissa tai -sovelluksissa toimiva internetyhteys on käytön kannalta ehdoton kriteeri. Tunnetuista ja yleisesti käytössä olevista verkkosovelluksista, jotka eivät toimi ilman internetyhteyttä, mainittakoon esimerkiksi Facebook ja Twitter. SPA-sovelluksena toteutettu Gmail-sähköpostisovellus sen sijaan toimii ainakin osittain ilman internetyhteyttä. Uusien sähköpostien vastaanottaminen tai lähettäminen ei Gmailissa ilman internetyhteyttä onnistu, mutta siinä voi esimerkiksi kirjoittaa sähköpostin, tallentaa sen ja lähettää myöhemmin internetyhteyden palattua.

2.2 Verkkosovellukset ilman SPA-sovellusarkkitehtuuria

Ilman SPA-sovellusarkkitehtuuria verkkosovellukset ovat täysin internetyhteyden avulla käytettävissä olevan palvelimen varassa. Jokaisen näkymän ja tilan muodostaminen ja

ylläpitäminen on täysin riippuvainen palvelimesta. Kuvassa 1 on havainnollistettu käyttäjän interaktio verkkosivun tai -sovelluksen datan tai tilan kanssa. Käyttäjän internetse-lain luo uuden pyynnön palvelimelle, ja palvelin vastaa käyttäjälle senhetkisen kuormi-tuksen mahdollistaman nopeuden mukaan päivitetyllä tilalla tai näkymällä (4, s. 8). Täl-laisessa tapauksessa kaikki laskenta tapahtuu palvelimella ja loppukäyttäjän päätelaite ainoastaan esittää käyttäjälle palvelimen lähettämää dataa, jolloin palvelin hoitaa kaiken prosessoinnin käyttäjän päätölaitteen ainoastaan odottaessa vastausta. Tämä aiheut-taa luonnollisesti enemmän prosessointia palvelimella verrattuna SPA-sovellukseen. Tämä mahdollisesti johtaa hitaaseen vasteaikaan ja suurempiin palvelinkustannuksiin, mikä johtuu suuremmasta laskentaprosessoinnista palvelimen osalta.



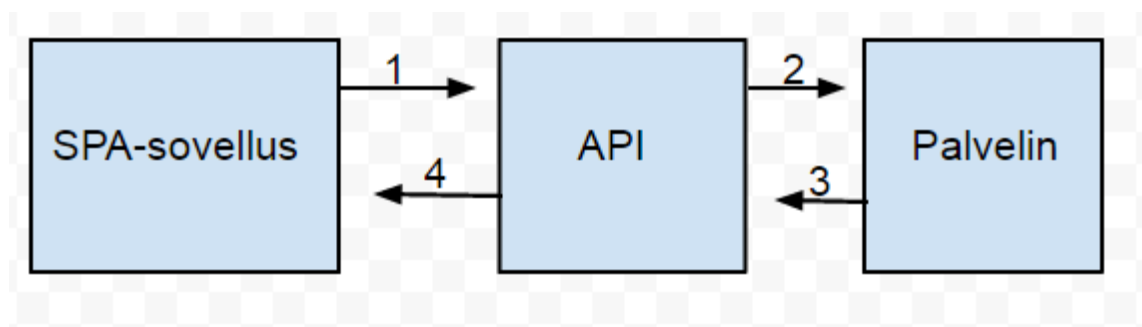
Kuva 1. Perinteisen verkkosovelluksen HTTP-pyyntö palvelimelle.

Perinteisessä verkkosovelluksessa toimimattoman internetyhteyden kanssa käyttökoke-mus on rajallinen, sillä sovelluksen näkymiin tai tiloihin ei pysty saamaan päivityksiä. Kun tieto ei liiku loppukäyttäjän laitteen ja palvelimen välillä, ei verkkosovellus voi toimia lain-kaan. Kun internetyhteys on huono tai pätkivä, käyttökokemus kärsii, sillä käyttäjän nä-kemä käyttöliittymä ja data ovat täysin riippuvaisia palvelimen lähettämistä vastauksista.

Ilman internetyhteyttä käyttäjä ei myöskään pysty halutessaan tallentamaan senhetkistä tilannetta verkkosivulla tai -sovelluksessa. Esimerkiksi Gmail-sähköpostisovelluksessa lähettämättömän sähköpostin tallentaminen olisi mahdotonta ilman SPA-sovellusarkkitehtuuria, ja tämän vuoksi käyttäjä on täysin riippuvainen toimivasta internetyhteydestä. Lähettämättömän sähköpostin tallentaminen tässä esimerkissä onnistuu siksi, että SPA-sovelluksessa käyttäjän internetselain ylläpitää tietoa siitä, missä tilassa sovelluksen malli (engl. model) on.

2.3 SPA-sovelluksena toteutetut verkkosovellukset

Verkkosovelluksissa käytetyt tekniikat ja teknologiat, kuten esimerkiksi ohjelmistokehykset ja ohjelmistokirjastot, kehittyvät ja uusiutuvat nopeasti (1). Ilman SPA-sovellusarkkitehtuuria verkkosivut ja -sovellukset toimivat palvelimen varassa; käyttäjälle lähetetään aina yksi staattinen näkymä verkkosovelluksen tai -sivun senhetkisestä näkymästä tai tilasta. Jos verkkosivu tai -sovellus on monimutkainen, palvelin joutuu prosessoimaan jokaisen näkymän erikseen ja sen jälkeen lähettämään sen käyttäjälle. Tällaisessa käyttötapauksessa käyttäjän päätelaite ei tehnyt minkäänlaista prosessointia, vaan ainoastaan vastaanotti tietoa palvelimelta, ja jonkinlaisen interaktion kautta pyysi palvelimelta päivitetyn näkymän verkkosovelluksen tai -sivun senaikaiseen tilaan. SPA-sovelluksessa käyttäjän päätelaite vastaa käyttöliittymän ja sovelluksen logiikan prosessoinnista ja ylläpitämisestä. Kuvassa 2 on esitelty tapahtumaketju SPA-sovelluksen pyytäessä tai tallentaessa dataa palvelimelle. Ilman SPA-sovellusarkkitehtuuria toteutetuissa verkkosivuissa ja -sovelluksissa ohjelmointikielinä ovat esimerkiksi PHP, Java tai Ruby on Rails. Nämä ohjelmointikieliset ovat kuitenkin edelleen suosittuja SPA-sovellusarkkitehtuurilla toteutettujen sovellusten palvelimella toimivien ohjelmistorajapintojen toteuttajina.



Kuva 2. SPA-sovelluksen tekemän HTTP-pyyynnön aiheuttama tapahtumaketju.

Nykyiset päätelaitteet, mukaan lukien mobiililaitteet, ovat enemmän kuin tarpeeksi suorituskykyisiä prosessoimaan moderneja verkkosivuja tai -sovelluksia. Tällöin palvelimen kuorma pienenee, mikä laskee kustannuksia ja vapauttaa resursseja palvelimella. Palvelimen ylläpitäminen ja useassa tapauksessa myös käyttökuorma vaikuttavat palvelimen kustannuksiin, jolloin käyttäjän päätölaitteen prosessointitehon hyödyntäminen vaikuttaa ylläpitokustannuksiin myönteisesti verkkosovelluksen tai -sivuston ylläpitäjän kannalta.

SPA-sovelluksessa palvelimen rooli tyypillisesti on pelkästään käsitellä dataa, jonka SPA-sovellus sille lähettää, jolloin palvelimen suorittamat prosessointioperaatiot ovat suhteellisen kevyitä. Tyypillinen operaatio palvelimella voi olla vaikkapa kaikkien uutisaiheiden lataaminen tietokannasta. Data palvelimen ja käyttäjän päätölaitteen välillä kulkee niin sanotun ohjelmointirajapinnan kautta. Ohjelmointirajapinnan avulla on mahdollista eriyttää palvelinpuolen sovelluksen ja käyttäjäpuolen sovelluksen tilan hallinta täysin. Ohjelmointirajapinta voi tällöin olla tilaton, eikä sen tarvitse olla millään tavalla tietoinen SPA-sovelluksen tilasta. Tilaton ohjelmointirajapinta mahdollistaa myös erityyppisten sovellusten rakentamisen samaan ohjelmointirajapintaan perustuen. Palvelinpuolen sovelluksen ei tarvitse olla millään tavalla tietoinen siitä, minkälainen sovellus käyttäjäpuolella on. Ohjelmointirajapinta mahdollistaa myös sen, että palvelinta voidaan käyttää monella tapaa. Sitä voidaan hyödyntää useassa eri sovelluksessa, tai sen dataa voidaan käyttää monella eri tavalla, riippuen käyttötapauksesta. Loppukäyttäjän käyttämä SPA-sovellus taas ei ole tietoinen siitä, minkälainen sovellus palvelimella on toiminnassa, kunhan se vastaa ohjelmointirajapintaan määriteltyjen vaatimusten mukaisesti.

3 SPA-sovelluksen kehittämisessä käytettävät työkalut

3.1 Käytettävät työkalut

SPA-sovelluksen kehittämiseen on olemassa lukuisia työkaluja, jotka helpottavat ohjelmistokehittäjän työtä. Useista työkaluista on olemassa useita eri toteutuksia, mutta kaikki variaatiot toteuttavat käytännössä saman asian. Kun ohjelmistokehitysprosessia automatisoidaan työkalujen avulla, ohjelmistokehittäjä pystyy keskittymään paremmin itse SPA-sovelluksen kehittämiseen, mikä tekee koko ohjelmistokehitysprosessista nopeamman ja kustannustehokkaamman. On tärkeää myös huomioida, että SPA-sovelluksessa

palvelimella toimivan sovelluksen kehittäminen voidaan eriyttää loppukäyttäjän näkemästä sovelluksesta kokonaan, jolloin työskentelyprosessi on helppo hajauttaa tehokkaasti esimerkiksi kahdelle tai useammalle ohjelmistokehittäjälle.

Käytettävät työkalut pystyvät toimimaan esimerkiksi palvelimena, automatisoimaan tiedostojen julkaisun, tarkistamaan kirjoitetun lähdekoodin tyylin ja oikeaoppisuuden sekä päivittämään sovelluksen senhetkisen näkymän tai tilan. Lähdekoodin tyyllillä tarkoitetaan esimerkiksi, kuinka erilaiset ohjelmistokielen operaatioita ilmaisevat rivit sisennetään, eli miltä koodi visuaalisesti näyttää. Tästä on hyötyä erityisesti, kun ohjelmistokehittäjiä on useita. Kun on yhteiset pelisäännöt, kaikkien tuottama lähdekoodi näyttää samalta, jolloin sen lukeminen ja tulkitseminen yksittäiselle ohjelmistokehittäjälle on helpompaa ja etenkin virheiden havaitseminen on tehokkaampaa ja nopeampaa. Nämä työkalut pystyvät myös havaitsemaan virheitä SPA-sovelluksen lähdekoodissa, mikä säästää ohjelmistokehittäjältä aikaa ja vaivaa hänen etsiessään esimerkiksi puuttuvaa puolipistettä mahdollisesti tuhansia rivejä sisältävästä lähdekooditiedostosta.

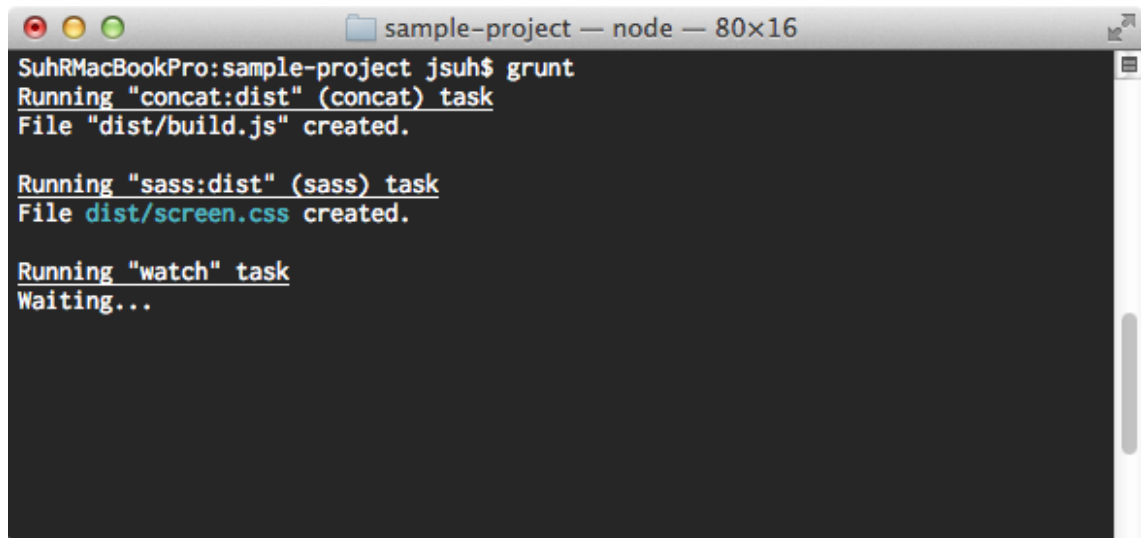
SPA-sovellusta kehitettäessä erilaisten työkalujen avulla voidaan automatisoida erilaisia kehitysprosessin osia ja jättää ohjelmistokehittäjälle enemmän aikaa keskittyä itse sovellukseen, eikä sen kehitysprosessin kehittämiseen tai ylläpitämiseen. Seuraavaksi tarkastellaan tarkemmin SPA-sovelluksen kehittämisessä käytettäviä työkaluja, niiden tuomia etuja ja niiden toimintaa. Kaikista SPA-sovelluskehityksen osa-alueista esitellään yleisimmät tai tässä insinööriyössä käytetyt sovellukset.

3.1.1 Kehityspalvelin

SPA-sovellus, kuten kaikki muutkin internetissä toimivat sovellukset ja sivut, tarvitsee toimiakseen palvelimen, joka lähettää loppukäyttäjälle prosessoidun lähdekoodin, joka näkyy käyttäjälle vaikkapa jonkinlaisena verkkosovelluksena. Loppukäyttäjän päätelaite prosessoi tämän lähdekoodin ja muodostaa siitä toimivan näkymän internet-sivuna hyödyntäen käyttäjän päätelaitteen prosessointitehoa lähdekoodin tulkitsemiseen ja erilaisten loogisten operaatioiden ratkaisemiseen. Jos SPA-sovellus käyttää jotain ohjelmistokehystä, palvelimen tulee muuntaa ohjelmistokehityksen muodostama lähdekoodi loppukäyttäjän päätelaitteen tulkittavaksi lähdekoodiksi. Tämä käytännössä tarkoittaa sitä, että ohjelmistokehityksen luoma lähdekoodi sellaisenaan ei ole internetselaimen tulkittavissa, vaan se täytyy ensin kääntää selainten tukemaan muotoon. Lisäksi on mahdollista, että ohjelmistokehityksessä on käytetty jonkin ohjelmointikielen päälle rakennettua

kieltä, joka tulee ensin kääntää alkuperäisen kielen standardin mukaiseen muotoon. Esimerkiksi CoffeeScript on ohjelmointikieli, joka täytyy ensin kääntää JavaScriptiksi, jotta se on internetselaimen tulkittavissa. Sovelluskehitystä tehtäessä sovelluspalvelimeksi on olemassa muutama vaihtoehto, kuten esimerkiksi Gulp ja Grunt. Grunt simuloi Apache-palvelinta kehitystyötä tehdessä ja pystyy suorittamaan automatisoituja toimintoja kehitysprosessin eri vaiheissa. Se on ilmainen sovellus ja saatavilla kaikille käyttöjärjestelmille (5).

Kun ohjelmistokehittäjä tekee muutoksia JavaScript-lähdekoodiinsa ja tallentaa ne, Grunt tunnistaa, mikä tai mitkä tiedostot ovat muuttuneet, ja automaattisesti konvertoi sen muotoon, jota loppukäyttäjän (tai sovelluskehityksen tapauksessa ohjelmistokehittäjän) internetselain ymmärtää. Kuvassa 3 Grunt on käynnistetyssä tilassa ja valmiina reagoimaan ohjelmistokehittäjän tekemiin muutoksiin lähdekoodissa. Grunt tarjoaa myös mahdollisuuden määritellä, mitä kaikkia erillisiä toimintoja suoritetaan ohjelmistokehittäjän tallentaessa tiedoston. Tämä mahdollistaa sen, että jokaisen tallennuskerran jälkeen esimerkiksi tarkistetaan lähdekoodin sisennykset ja suoritetaan automatisoidut testit, jotka kokeilevat, toimivatko lähdekoodin sisällä sijaitsevat yksittäiset toiminnot niin, kuin niiden on tarkoitus toimia. Tämä poistaa tilanteen, jossa ohjelmistokehittäjän tekemät muutokset tuovat kehitettävään sovellukseen uuden ominaisuuden, mutta joka rikkoo jonkin aiemmin toimineen ominaisuuden. Gruntin suorittamat automaattiset toiminnot ovat täysin käyttäjän valittavissa. Automaattisten toimintojen määrittäminen tapahtuu muokkaamalla Gruntin asetustiedostoa.

A screenshot of a terminal window on a Mac. The window title is "sample-project — node — 80x16". The terminal shows the following output:

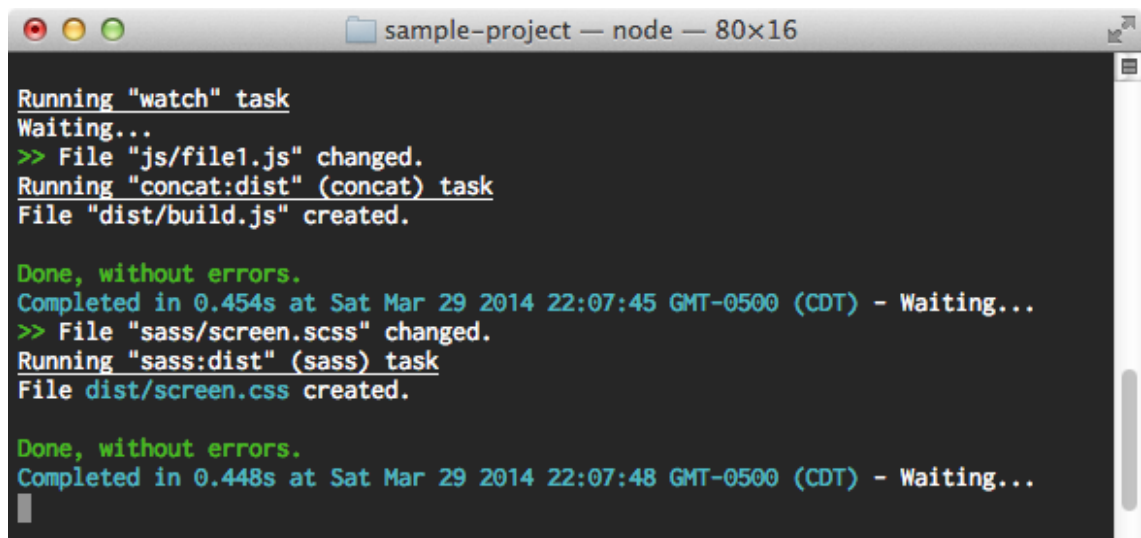
```
SuhRMacBookPro:sample-project jsuh$ grunt
Running "concat:dist" (concat) task
File "dist/build.js" created.

Running "sass:dist" (sass) task
File dist/screen.css created.

Running "watch" task
Waiting...
```

Kuva 3. Grunt käynnistetyssä tilassa (6).

Grunt tarjoaa mahdollisuuden tehdä kehitettävästä SPA-sovelluksesta julkisen ohjelmistokehittäjän lähiverkossa. Kun SPA-sovellus on ohjelmistokehittäjän lähiverkossa vapaasti käytettävissä, on mahdollista kokeilla sen käyttämistä erilaisilla päätelaitteilla, kuten esimerkiksi iPadilla tai jollain muulla mobiililaitteella. Jos palvelin ei sijaitisi ohjelmistokehittäjän omalla tietokoneella, hän joutuisi jokaisen tallennuskerran jälkeen siirtämään tehdyt muutokset jollekin palvelimelle odottamaan niiden siirtymistä ja vasta sitten tarkastelemaan tuoreita muutoksia (4, s. 287). Tässä kuluisi ylimääräistä aikaa, ja kehitystyö ilman toimivaa internetyhteyttä olisi käytännössä turhauttavan hidasta. Kuvassa 4 näkyy, kuinka Grunt on reagoinut muutokseen lähdekoodissa ja luonut päivitettyt tiedostot. Grunt oletusarvoisesti myös päivittää näkymän kaikkiin niihin päätelaitteisiin, joilla on internetselaimella navigoitu kehityspalvelimen asetuksissa määriteltyyn IP-osoitteeseen ja porttiin, jossa kehitettävä SPA-sovellus toimii.



```

sample-project — node — 80x16

Running "watch" task
Waiting...
>> File "js/file1.js" changed.
Running "concat:dist" (concat) task
File "dist/build.js" created.

Done, without errors.
Completed in 0.454s at Sat Mar 29 2014 22:07:45 GMT-0500 (CDT) - Waiting...
>> File "sass/screen.scss" changed.
Running "sass:dist" (sass) task
File dist/screen.css created.

Done, without errors.
Completed in 0.448s at Sat Mar 29 2014 22:07:48 GMT-0500 (CDT) - Waiting...

```

Kuva 4. Gruntin reaktio ohjelmistoprojektin lähdekoodin muutokseen (6).

Jos SPA-sovellus käyttää jotakin ohjelmointirajapintaa esimerkiksi datan tallentamiseen tai hakemiseen, ohjelmistokehittäjä saattaa kohdata niin sanottuun CORS-ongelmaan. CORS-tapauksessa SPA-sovellus ja sen tavoittelema ohjelmointirajapinta sijaitsevat eri aliverkoissa, jolloin SPA-sovellus pyytää tai lähettää tietoa toiseen aliverkkoon, mikä on oletusarvoisesti aina estetty tietoturvallisuussyistä. Jos CORS-pyyntö olisi oletusarvoisesti sallittu, kuka tahansa pystyisi käyttämään jokaista ohjelmointirajapintaa internetissä tai ainakin tekemään niihin HTTP-pyyntöjä. Tämä voidaan kuitenkin kiertää muokkaamalla SPA-sovelluksen lähettämien HTTP-pyyntöjen metatietoja. Esimerkiksi palvelu,

joka sijaitsee osoitteessa <http://www.example1.com>, ei pysty lähettämään HTTP-pyyntöjä osoitteeseen <http://www.example2.com> ja saamaan toivottua vastausta ilman, että CORS-pyyntö on erikseen sallittu. Kun SPA-sovellus julkaistaan, on mahdollista määrittellä ohjelmointirajapinnan sijaitsevan samassa aliverkossa kuin SPA-sovellus. Tällöin CORS-pyyntöt voidaan pitää kiellettyinä, jolloin ulkopuoliset tahot eivät ohjelmointirajapinnan kautta pääse vaikuttamaan SPA-sovellukseen tilaan tai dataan.

3.1.2 Paketinhallinta

Automaattinen paketinhallinta mahdollistaa ulkopuolisten ohjelmistokirjastojen lataamisen ja automaattisen hallinnoinnin. Kaksi suosittua vaihtoehtoa paketinhallintaan ovat Bower ja npm (7; 8). Kuvassa 5 Bowerin avulla asennetaan jQuery-niminen ohjelmistokirjasto. Tämän jälkeen kirjasto on sovelluksessa käytettävissä, mutta siitä käytettävää versionumeroa pystytään muokkaamaan Bowerin konfiguraatitiedostosta.

```
C:\dev\Bower\BowerInVS2013>bower install jquery --save
bower cached      git://github.com/jquery/jquery.git#2.1.3
bower validate    2.1.3 against git://github.com/jquery/jqu
bower install     jquery#2.1.3

jquery#2.1.3 bower_components\jquery
C:\dev\Bower\BowerInVS2013>_
```

Kuva 5. Ulkopuolisen ohjelmistokirjaston lisääminen projektiin (9).

Automaattisen paketinhallinnan käyttämisellä tarkoitetaan sitä, että sovellus on riippuvainen jostakin olemassa olevasta ulkopuolisesta ohjelmistokirjastosta tai ohjelmistokehyksestä, joka ladataan automaattisesti SPA-sovelluksen käytettäväksi. Tällaisessa tapauksessa tuotantovalmis SPA-sovellus sisältää nämä ulkopuoliset riippuvuudet ja mahdollistaa SPA-sovelluksen käytön ilman internetyhteyttä, jos se on kertaalleen ladattu käyttäjän päätölaitteelle. Ohjelmistokehittäjä pystyy myös määrittelemään tarkat versiot ulkoisista riippuvuuksista tai käyttämään ulkoisen riippuvuuden uusinta versiota. Monet SPA-sovellukset määrittelevät tarkat versiot tai minimiversiot haluttavista ulkoisista riippuvuuksista, jolloin ohjelmistokehittäjä välttyy käytettävien ohjelmistokirjastoiden tai ohjelmistokehysten välisiltä konflikteilta. Konflikteilla tarkoitetaan sitä, että jokin ulkoisen paketin versio ei toimi jonkin toisen ulkoisen riippuvuuden version kanssa, mikä johtuu muutoksista kyseisen riippuvuuden lähdekoodiin. Kuvassa 6 on avattuna ulkopuolisten

riippuvuuksien konfiguraatiotiedosto, jossa on määritelty kunkin riippuvuuden käytettävä versionumero. On myös mahdollista määrittellä käytettävän riippuvuuden minimiversio, joka on ilmaistu ~-merkillä versionumeron edessä.

```
1  {
2    "name": "qtool",
3    "version": "0.0.0",
4    "dependencies": {
5      "angular": "1.2.15",
6      "json3": "~3.2.6",
7      "es5-shim": "~2.1.0",
8      "jquery": "~1.11.0",
9      "bootstrap-sass-official": "~3.1.0",
10     "angular-resource": "1.2.15",
11     "angular-cookies": "1.2.15",
12     "angular-sanitize": "1.2.15",
13     "angular-route": "1.2.15",
14     "font-awesome": "~4.1.0"
15   },
16   "devDependencies": {
17     "angular-mocks": "1.2.15",
18     "angular-scenario": "1.2.15"
19   }
20 }
21 |
```

Kuva 6. Ohjelmistoprojektin ulkopuoliset riippuvuudet.

Ilman paketinhallintaa SPA-sovelluksessa ohjelmistokehittäjän täytyisi määrittellä jokaisen ulkopuolisen kirjaston ja ohjelmistokehyksen URL-osoite ja mahdollinen versionumero sovelluksen metatietoihin. Mahdolliset muutokset käytettäviin ulkopuolisiin kirjastoihin tai niiden URL-osoitteisiin ohjelmistokehittäjän täytyisi aina tehdä manuaalisesti. Monessa tapauksessa tämä johtaisi ainakin ajoittaiseen käyttökatkokseen sovelluksessa ulkopuolisten riippuvuuksien konfliktien vuoksi. Jos jostain syystä jonkin tarpeellisen ulkopuolisen ohjelmistokirjaston URL-osoite muuttuisi, olisi SPA-sovellus todennäköisesti toimimaton, kunnes ohjelmistokehittäjä korjaisi oikean URL-osoitteen sovelluksen metatietoihin. Ennalta määritellyt versionumerot tekevät myös SPA-sovelluksen toiminnasta ennakoitavaa ohjelmistokehittäjän näkökulmasta. Jos käytettäisiin internetin kautta ladattua uusinta versiota, jokin osa SPA-sovelluksesta tai sen ulkopuolisista ohjelmistokirjastoista tai -kehyksistä saattaisi olla toimimaton tämän ulkopuolisen ohjelmistokirjaston

tai -kehiksen kanssa, mikä johtaa mahdollisesti ainoastaan osittain toimimattomaan SPA-sovellukseen.

3.1.3 Tyylienhallinta

SPA-sovellukset, kuten kaikki muutkin internetsivut ja -sovellukset, hyödyntävät tyylitiedostoja, jotka määrittelevät, miltä kunkin elementin tulee näyttää ja miten sen tulee toimia toisten elementtien suhteen. Elementillä tarkoitetaan mitä tahansa HTML-tagia, esimerkiksi <h1>-tagia, jonka avulla voidaan merkitä otsikko internetsivulla. Internetsivuilla ja -sovelluksissa, sisältäen kaikki SPA-sovellukset, elementtien tyylit määritellään CSS:llä. CSS on lyhenne sanasta cascading style sheets ja se on standardoitu tyylien merkintäkieli (10).

CSS-merkintäkielessä käytetään valitsimia (engl. selector), joilla ensin valitaan jokin elementti ja määritellään niille tyyliminuisuuksia ja niille attribuutteja. On olemassa kehittyneempiä merkintäkieliä tyylejä varten, kuten esimerkiksi SASS (ja LESS, jotka ovat esiprosessoituja CSS-merkintäkieliä, jotka ennen julkaisua käännetään aina CSS:ksi. Nämä esiprosessoidut merkintäkielet voidaan mieltää jatkeina CSS:lle, ja ne mahdollistavat muun muassa muuttujien, funktioiden ja mixinien käytön.

Muuttujat, funktiot ja mixinit muodostavat osittain esiprosessoidun CSS:n vahvuuden ja mahdollistavat jo kirjoitetun lähdekoodin uudelleenkäytön. Perinteinen CSS-merkintäkieli ei tue edellä mainittuja ominaisuuksia. Muuttujien avulla voidaan tallentaa johonkin muuttujaan esimerkiksi jokin haluttu väri, jota voidaan sitten käyttää monessa tyyliattribuutissa. Jos käytössä ei olisi esiprosessoitua CSS:ää, jouduttaisiin jokaiselle tyyliattribuutille määrittelemään ominaisuudeksi kyseinen väri. Tämä voisi koskea satoja tyyliattribuutteja, jolloin värin vaihtamiseen pitäisi muokata kaikkia tyyliattribuutteja erikseen. Muuttujia käytettäessä kaikki tyyliattribuutit voivat viitata tähän muuttujaan, ja väriä vaihdettaessa voitaisiin muokata pelkästään yhtä riviä eli kyseisen muuttujan arvoa. Muuttujan nimi, vaikkapa \$tummanHarmaa1, on myös todennäköisesti kuvaavampi ohjelmistokehittäjälle kuin esimerkiksi hex-väriarvo #FFCC00.

Mixinit tekevät mahdolliseksi ohjelmistokehittäjälle käyttää yhtä tyyliminaisuutta, vaikka todellisuudessa hän käyttää useaa tyyliminaisuutta, esimerkiksi erilaisia internetse-laimia varten. Sen sijaan, että ohjelmistokehittäjä määritteli jokaiselle eri internetse-

laimelle oman tyyliominaisuuden, hän voi käyttää määrittelemäänsä mixiniä, joka esiprosessin jälkeen muuttaa tyylitiedoston sisältämään tarvittavan lähdekoodin jokaista internetselainta varten. Kuvassa 7 on esimerkki yksinkertaisesta mixinistä, jonka avulla voidaan määritellä jollekin HTML-elementille transition-ominaisuus. Tämä mixin sisältää tyylimäärittelyt jokaiselle eri internetselaimelle erikseen, jolloin mixiniä käytettäessä voidaan vain viitata sen nimeen, kuten kuvan 7 esimerkissä tehdään rivillä 12. Tämä tarvittava CSS-merkintäkielen lähdekoodi on määritelty mixinin sisälle, mikä tekee esiprosessoimattoman tyylitiedoston lukemisesta ja mixiniä käyttävän tyyli-ominaisuuden käytettävyydestä helpompaa. Eri internetselaimet, esimerkiksi Google Chrome ja Internet Explorer, tulkitsevat joitakin tyyliattribuutteja eri tavoin ja käyttävät erilaisia etuliitteitä jollekin tyyliattribuuteille, jolloin mixinin avulla ohjelmistokehittäjä pystyy liittämään nämä eroavaisuudet yhteen mixiniin, mikä tekee tyylitiedoston kirjoittamisesta selainriippumattonta.

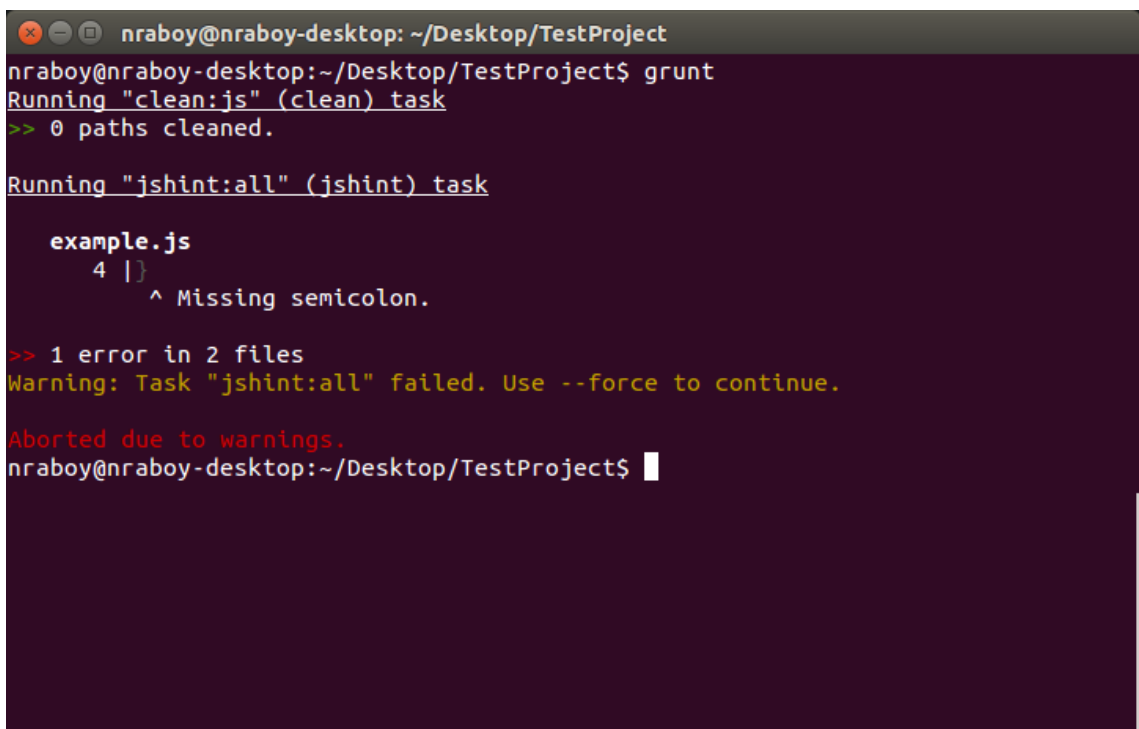
```
1  @mixin transition($args...) {
2      -webkit-transition: $args;
3      -moz-transition: $args;
4      -ms-transition: $args;
5      -o-transition: $args;
6      transition: $args;
7  }
8
9  // käyttöesimerkki
10 a {
11     color: #FFF;
12     @include transition(color .3s ease);
13     &:hover {
14         color: #000;
15     }
16 }
```

Kuva 7. SASS-merkintäkieli käytännössä.

3.1.4 Automaattinen lähdekoodin tarkistus

Automaattisella lähdekoodin tarkistuksella voidaan koneellisesti tarkistaa, että koko sovelluksen lähdekoodi noudattaa tiettyjä ennalta määriteltyjä sääntöjä. Ohjelmistokehittäjä pystyy vapaasti määrittelemään nämä säännöt, ja ne voivat esimerkiksi kertoa, kuinka monta merkkiä muuttujan nimessä on vähintään oltava tai sallitaanko lähdekoo-

dissa olevan muuttujia, joita ei käytetä lainkaan. Yksi suosittuja automaattisia koodintarkastajia on JSLint, joka on ilmainen työkalu ja sisältää joitakin oletusarvoisia tarkistuksia (11). Kun jonkinlaiset säännöt lähdekoodin ulkoasulle ja konventioille on määritelty, sovelluksen lähdekoodi pysyy samankaltaisena, mikä tekee siitä yhdenmukaista ja helposti tulkittavaa. Tämän merkitys korostuu etenkin silloin, kun ohjelmistokehittäjiä on useita. Automaattinen lähdekoodin tarkistus pystyy myös silmänräpäyksessä löytämään ohjelmistokielellisiä virheitä lähdekoodista, kuten esimerkiksi puuttuvan ;-merkin koodirivin lopusta, kuten kuvassa 8 on tapahtunut. Mikäli tällaista ongelmaa ei välittömästi löydetä, se saattaa kostautua vasta jossain myöhemmässä vaiheessa ja sen löytäminen saattaa olla hyvinkin työlästä.



```

nraboy@nraboy-desktop: ~/Desktop/TestProject
nraboy@nraboy-desktop:~/Desktop/TestProject$ grunt
Running "clean:js" (clean) task
>> 0 paths cleaned.

Running "jshint:all" (jshint) task

  example.js
    4 |}
      ^ Missing semicolon.

>> 1 error in 2 files
Warning: Task "jshint:all" failed. Use --force to continue.
Aborted due to warnings.
nraboy@nraboy-desktop:~/Desktop/TestProject$

```

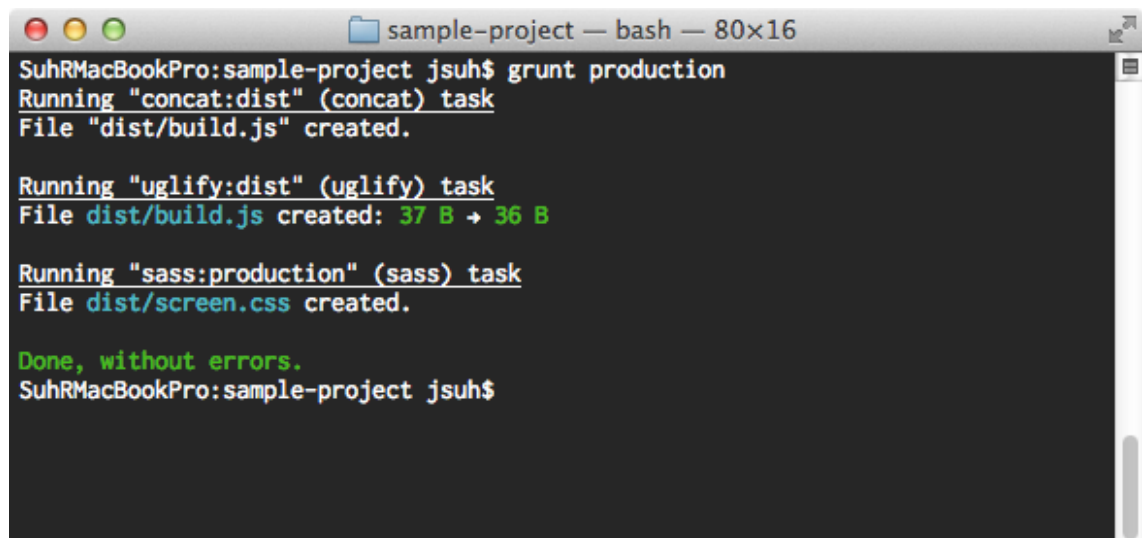
Kuva 8. Automaattinen lähdekoodin tarkistus (12).

Ohjelmistokehityksessä vuoden mittainen projekti ei ole pitkä. Kun kehitystyötä jatketaan pitkään, automaattinen lähdekoodin tarkistus pitää huolen, että lähdekoodin ulkoasu ja konventiot pysyvät yhtenäisenä. Ilman automaattista tarkistusta on mahdollista, että lähdekoodi on ulkonäöllisesti ja konventioiltaan erilaista eri kohdissa lähdekoodia. Tämä korostuu ohjelmistoprojekteissa, joissa on useita ohjelmistokehittäjiä ja etenkin kun projektiin liittyy myöhemmässä vaiheessa uusia ohjelmistokehittäjiä. Ulkonäöllisesti tarkistettavia seikkoja voisi olla esimerkiksi se, että rivin sisennys on aina jokin tietty määrä

välilyöntejä tai että ohjelmointikielen kirjoitus on konsistenttia. Tällä tarkoitetaan, että esimerkiksi ennen jokaista { -merkkiä on välilyönti, vaikka se ei lähdekoodin toiminnan kannalta olisikaan olennaista.

3.1.5 Tuotantoon vieminen

Kun ohjelmistoprojekti on siinä vaiheessa, että se on julkaisukelpoinen, siitä voidaan muodostaa kompressoitu versio, joka siirretään julkiseen jakeluun internetiin. Kompresoidussa versiossa voidaan poistaa lähdekoodista kaikki turhat välilyönit ja rivinvaihdot sekä muuttaa muuttujien nimet mahdollisimman lyhyiksi, jolloin saadaan aikaan huomattavasti pienempi tiedosto. Mitä pienempi tämä kompressoitu versio SPA-sovelluksesta on, sitä nopeammin sen loppukäyttäjä saa ladattua omalle päätelaitteelleen. Jos ohjelmistokehittäjä käytti kehitysprosessissaan vaikkapa SASS-esiprosessoitua CSS:ää, tuotantoon viedyssä versiossa on mukana ainoastaan yksi valmis CSS-tiedosto, johon on koottu ja esiprosessoitu kaikki SASS-tiedostot. Kuvassa 9 on luotu kompressoitu versio projektista. Siinä on luotu dist-nimiseen kansioon kaikista sovelluksen tiedostoista yksi HTML-, CSS- ja JavaScript-tiedosto.



```

sample-project — bash — 80x16
SuhRMacBookPro:sample-project jsuh$ grunt production
Running "concat:dist" (concat) task
File "dist/build.js" created.

Running "uglify:dist" (uglify) task
File dist/build.js created: 37 B → 36 B

Running "sass:production" (sass) task
File dist/screen.css created.

Done, without errors.
SuhRMacBookPro:sample-project jsuh$

```

Kuva 9. Grunt luo projektista julkaistavan version (6).

Se, mitä tarkalleen ottaen kehitysversion kompressoinnin aikana tapahtuu, on ohjelmistokehittäjän konfiguroitavissa. Ohjelmistokehittäjä voi esimerkiksi määritellä, että jotkin tiedostot tai kansiot jätetään kompressoimatta. Konfigurointitiedostossa määritellään,

mitkä kaikki tiedostot tulee kompressoida, mihin ne siirretään ja tarvitseeko joitakin tiedostoja esiprosessoida jollakin tapaa. Aiemmin esitellyistä aputyökaluista Grunt mahdollistaa ohjelmistoprojektin kompressoinnin. Oletusarvoisesti, jos käyttäjä ei muokkaa Gruntin julkaisuasetuksia, tuotantoversiota luotaessa kirjoitetaan komento 'Grunt build' tai 'Grunt production'. Tällöin automaattisesti luodaan projektin sisälle 'dist'-niminen kansio, joka sisältää ainoastaan yhden HTML-tiedoston, yhden CSS-tiedoston ja yhden JavaScript-tiedoston. Halutessaan ohjelmistokehittäjä voi jatkokehittää sovellustaan ja samalla julkaisuprosessilla luoda siitä uusia jakeloversioita.

3.2 Kehitysympäristö

SPA-sovelluksen kehittämisessä itse lähdekoodin kirjoittaminen on tietenkin olennaista, kuten kaikessa muussakin ohjelmistokehityksessä. SPA-sovellukset koostuvat HTML- ja CSS-merkintäkielistä ja JavaScriptistä, jolloin ohjelmistokehittäjän käyttämä kehitysympäristö on pienessä roolissa kokonaiskuvassa. JavaScript on tulkittava kieli, mikä tarkoittaa sitä, että se prosessoidaan samalla, kun se suoritetaan. JavaScript-kieli siis prosessoidaan jokaisen suorituskerran yhteydessä. Tämä johtaa siihen, että millä tahansa tekstieditorilla voidaan tehokkaasti kehittää SPA-sovelluksia.

Eri kehitysympäristöissä kuitenkin on omia etuja, mutta yhteisenä tekijänä voidaan pitää lähdekoodin väriä. Muita mahdollisesti hyödyllisiä ominaisuuksia ovat ohjelmointikielen sisäänrakennettujen funktioiden automaattinen ehdottaminen ja erilaisten hakujen tekeminen ohjelmistoprojektin sisällä. Erilaiset ohjelmointikielen liittyvät asiat voidaan värjätä jollain tietyllä värillä, jolloin lähdekoodin silmäily on helppoa ja miellyttävää. Hyvänä esimerkkinä kehitysympäristön merkitsemättömyydestä voidaan pitää sitä, että nyt 30 vuotta vanhalla Emacs-editorilla voidaan kehittää ohjelmistoa aivan samalla tapaa kuin modernilla Sublime Text-editorilla. Kuvassa 10 on Sublime Text 3-editori, jossa on oletusasetukset päällä. Kuvasta huomaa, kuinka tämä editori värittää lähdekoodin osia eri väreillä ja tekee siitä helposti luettavaa.

```

5  qtoolControllers.controller('MainCtrl', function ($scope, PollService, $cookieStore) {
6
7      var source = new EventSource("//localhost/qtool-api/poller.php");
8      PollService.getLatestPoll($scope, source)
9      $scope.hasVoted = false;
10
11     $scope.votes = $cookieStore.get('votes') || [];
12
13     $scope.vote = function(id) {
14         $scope.votes.push($scope.livePoll.ID);
15         $cookieStore.put('votes', $scope.votes); // store voted ID:s in a cookie
16         PollService.vote(id);
17     }
18
19 });
20
21 qtoolControllers.controller('LoginCtrl', function ($scope, $rootScope, AuthService) {
22     $scope.login = function(user) {
23         AuthService.auth(user);
24     }
25 });

```

Kuva 10. Lähdekoodin väritys ja sisennys esiteltyinä Sublime Text 3-tekstieditorissa.

3.3 Ohjelmistokehykset

Ohjelmistokehys (engl. framework) on ohjelmistokehityksessä käytettävä kokonaisuus, jonka tarkoitus on helpottaa ja suoraviivaistaa ohjelmistokehitysprosessia. Ohjelmistokehyksiä on useita erilaisia, mutta yhteisenä piirteenä niille kaikille on se, että ne ovat yksinään toimivia kokonaisuuksia ja niiden avulla ohjelmistokehittäjän ei tarvitse kehitysprosessinsa aikana ratkaista jokaista pulmaa itse, vaan ohjelmistokehyksen ympärille rakentunut yhteisö tai ohjelmistokehyksen kehittäjät ovat ratkaisseet ne aiemmin. Ohjelmistokehyksissä keskenään on kuitenkin paljon eroavaisuuksia. Jotkin vaativat todella tarkkaan määritellyn kehitysprosessin ja ohjelmointitavan, kun taas jotkin tarjoavat lähinnä ohjeita ja pieniä aputyökaluja ohjelmistokehitykseen. Osa ohjelmistokehyksistä ratkaisee todella tarkkaan määriteltyjä ongelmia, esimerkiksi internetsivuilla olevien lomakkeiden käsittelyn, kun jotkin ohjelmistokehykset taas tarjoavat työkalut kaikkeen ohjelmistokehitykseen rajaten kuitenkin alueen siihen ohjelmistokehitykseen, johon ohjelmistokehys on suunnattu, esimerkiksi verkkosivujen ja -sovellusten kehitykseen.

Ohjelmistokehys mahdollisesti myös tarjoaa ohjelmistokehittäjälle tarkan rungon ja rakenteen, jonka avulla hän pystyy kehitysprosessinsa aikana pitämään yhtenäisen linjan tuotetun lähdekoodin ja sovelluksen visuaalisen ja toiminnallisen lopputuloksen kanalta. Tämä helpottaa mahdollista jatkokehitystä huomattavasti, sillä esimerkiksi ulkopuolinen ohjelmistokehittäjä, jolla on kokemusta käytetystä ohjelmistokehityksestä, pystyy

helposti ymmärtämään jo toteutetun sovelluksen ja jatkamaan sen kehitystä. Lisäksi virheiden löytäminen ja korjaaminen ja avun saaminen muilta ohjelmistokehittäjiltä on helppoa, koska he pystyvät kokemuksensa perusteella hahmottamaan, minkälainen ongelmatilanne apua etsivällä ohjelmistokehittäjällä mahdollisesti on, jos ongelmatilanne liittyy ohjelmistokehityskeskiseen ongelmaan.

Kaksi hyvin suosittua ohjelmistokehitystä verkkosivujen ja -sovellusten kehittämiseen ovat AngularJS ja Backbone.js (13). Nämä kaksi ohjelmistokehitystä tarjoavat paljon samantaisia asioita ohjelmistokehittäjälle, mutta omalla tavallaan, eikä luotu lähdekoodi näiden kahden ohjelmistokehityksen välillä ole yhteensopivaa. On myös paljon asioita ja ominaisuuksia, jotka puuttuvat toisesta ohjelmistokehityksestä kokonaan, mutta ovat helposti toteutettavissa toisessa. Yhtenä esimerkkinä voidaan pitää AngularJS:n two-way data binding -ominaisuutta. Two-way data bindingissä sovelluksen käyttämä tietomalli on automaattisesti liitetty näkymässä näkyvään tietoon, esimerkiksi lomakkeeseen verkkosivulla (14). Käyttäjän syöttäessä tietoa lomakkeeseen AngularJS sisäisesti ylläpitää tämän tietomallin yhteneväisyyttä käyttöliittymän ja itse sovelluksen välillä. Vastaavassa esimerkissä Backbone.js:llä toteutettuna ohjelmistokehittäjän tulee itse hallita tietomallin yhteneväisyys asettamalla sovellukseen niin sanottuja kuuntelijoita, jotka kuuntelevat käyttäjän syötettä ja tallentavat tiedon sovelluksen tietomalliin. Molemmissa tapauksissa on hyvät ja huonot puolensa, mutta tämä esimerkki on huomattavasti nopeampi toteuttaa AngularJS-ohjelmistokehityksellä.

Ohjelmistokehitysten etuna voidaan pitää myös sitä, että ohjelmistokehittäjä mahdollisesti välttää tarpeen käyttää useita eri ulkopuolisia ohjelmistokirjastoja sovelluskehityksessä, mikä johtaa pienempään määrään ristiriitoja erilaisten ohjelmistokirjastoiden välillä. Ohjelmistokirjastojen päivitykset voivat myös aiheuttaa ongelmatilanteita, eikä niiden jatkuvasta ylläpidosta ole mitään takeita ohjelmistokehittäjälle. Tällöin sovelluksen kehittäjä säästää aikaa ja vaivaa, jos mahdollisimman monta ongelmaa on ratkaistu itse ohjelmistokehityksessä, eikä tarvetta ulkopuolisille ohjelmistokirjastoille ole.

4 SPA-sovellus käytännössä

4.1 Sovelluksen vaatimukset ja toteutus

Insinööriyön osana toteutettiin verkkosovellus, jonka avulla sovelluksen käyttäjä pystyy luomaan kyselyn, johon toiset käyttäjät pystyvät vastaamaan anonymiminä yhden kerran äänestystä kohti. Kysely toimii reaaliajassa, jolloin kyselyitä voi tehdä peräjälkeen useita ilman, että verkkosivua tarvitsee ladata uudestaan, sillä se on toteutettu SPA-sovelluksena. Kyselyn luonut taho pystyy tarkastelemaan annettujen vastausten lukumäärää, mutta vastannut taho näkee ainoastaan kaikkien annettujen vastausten prosentuaalisen jakauman. Vastaajat ja kyselyn luonut taho näkevät reaaliajassa, paljonko asetetusta vastausajasta on jäljellä. Vastausajan loputtua kyselyyn ei pysty enää vastaamaan, mutta tulokset ovat näkyvissä olettaen, että uutta kyselyä ei ole luotu. Uusi kysely korvaa aina vanhan kyselyn, vaikka äänestysaikaa olisikin jäljellä.

Tämänkaltainen kysely soveltuu monenlaisiin käyttötapauksiin, mutta yhtenä käyttötapauksena voidaan pitää opetustilannetta, jossa opettaja toimii kyselyjen luovana tahona ja oppilaat vastaajina. Opetustilanteessa opiskelijat useasti eivät ole halukkaita vastaamaan suullisiin kysymyksiin muiden opiskelijoiden läsnä ollessa, mutta anonymieihin inter- tai intranetin kautta toteutettuihin kyselyihin vastauskynnys on matalampi. Tilanteessa, jossa opettaja saa mahdollisimman monelta opiskelijalta vastauksen kyselyyn, hän pystyy paremmin seuraamaan opiskelijoiden osaamista ja oppimista. Käyttötapauksia on lukemattomia, mutta tässä insinööriyössä keskitytään tämän sovelluksen tekniseen toteutukseen.

Työn teknisessä suunnittelussa rajattiin sovelluksen tarpeet ja vaatimukset. Sovelluksen vaatimuksiksi rajattiin seuraavat seikat:

- Kyselyt ja vastaukset voidaan tallentaa.
- Kyselyn luomiseen vaaditaan käyttäjätunnus ja salasana.
- Kyselyistä ja annetuista vastauksista voidaan tehdä hakuja.
- Vastaaminen kyselyyn onnistuu anonymisti.
- Vastata voi kerran kyselyä kohti.
- Sovelluksen käyttäminen on mahdollista ja sulavaa myös mobiililaitteilla.

- Sovellus luo mahdollisimman vähän verkkoliikennettä.
- Kaikki sovelluksen toiminnot tapahtuvat reaaliajassa.

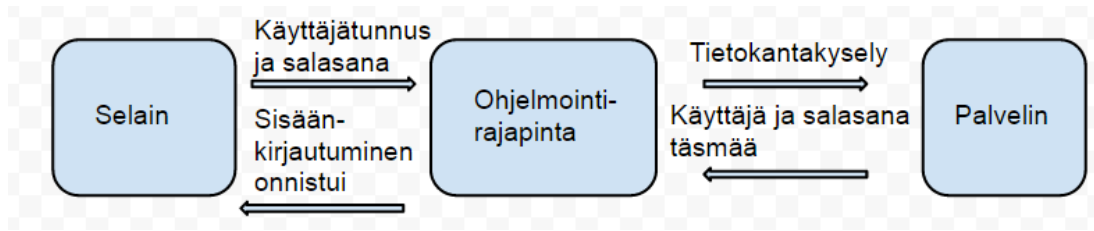
Insinööriyön tarkoituksena oli käyttää moderneja sovelluskehitystekniikoita, joten tässä tapauksessa päätettiin SPA-sovelluksen sopivan tähän erinomaisesti. Se mahdollisti sovelluksen näkyvän osan ja palvelimella tapahtuvan osan täydellisen eriyttämisen. Tämän vuoksi sovelluskehittäjä pystyi keskittymään pääosin sovelluksen näkyvän osan kehittämiseen moderneilla tekniikoilla. Sovelluksen näkyvä osa ja palvelimella sijaitseva sovellus eivät siis ole millään tavalla tietoisia toisistaan tai toisen sovelluksen sisäisestä tilasta. Sovelluksen näkyvällä osalla tarkoitetaan sitä osaa, joka loppukäyttäjällä on nähtävillä internetselaimessaan. Sovelluksen näkyvän osan ja palvelimella toimivan osan väliin suunniteltiin ohjelmointirajapinta, jonka avulla nämä sovelluksen kaksi eri osa-aluetta pystyvät kommunikoimaan keskenään.

Teknisessä suunnittelussa todettiin, että palvelinpuolen sovellus voi olla mahdollisimman yksinkertainen, mikä jättää suuremman painoarvon sovelluksen näkyvän osan kehitykseen. Yksinkertaisella tarkoitetaan tässä tapauksessa sitä, että ohjelmointirajapinnan tai sovelluksen käsittelijöiden ei tarvitse olla robusteja tai proseduraalisia. Palvelinpuolen ohjelmointikieleksi valittiin PHP ja siihen ohjelmistokehykseksi FlightPHP, joka mahdollistaa RESTFUL-ohjelmointirajapinnan luomisen (15). RESTFUL-ohjelmointirajapinnat tyypillisesti kommunikoivat HTTP-protokollan kautta ja sallivat neljä erilaista kutsua. Näitä kutsuja nimitetään CRUD-kutsuiksi, eli Create (luo), Read (lue), Update (päivitä) ja Delete (poista/tuhoa). Palvelinpuolen ohjelmointikieleksi olisi aivan yhtä hyvin sopinut jokin toinen ohjelmointikieli, mutta valintaa tehtäessä otettiin huomioon sovelluskehittäjän aiempi kokemus PHP-ohjelmointikielestä. Koska sovelluksen vaatimuksissa määriteltiin, että tietoa pitää pystyä tallentamaan johonkin, valittiin käytettäväksi tiedon tallentamisjärjestelmäksi MySQL-tietokanta. MySQL on yksi suosituimmista relaatiotietokannoista, ja se tukee SQL-kyselykieltä (16). Tietokantasovellusta valittaessa otettiin myös huomioon sovelluskehittäjän aiempi kokemus erilaisista tietokantasovelluksista. Tässäkin tapauksessa valinta olisi hyvinkin voinut osua johonkin toiseen ohjelmointikieleen tai tietokantasovellukseen, mutta sitä ei pidetty merkittävänä seikkana, eikä sen nähty tuovan mitään lisäarvoa lopulliseen sovellukseen.

Sovelluksen näkyvän osan ohjelmointikieleksi valittiin JavaScript ja ohjelmistokehykseksi AngularJS. Koska kyseessä on verkkosovellus, HTML- ja CSS-merkintäkielet ovat luontainen osa kokonaisuutta sovelluksen rakenteessa. AngularJS:n valintaa tuki

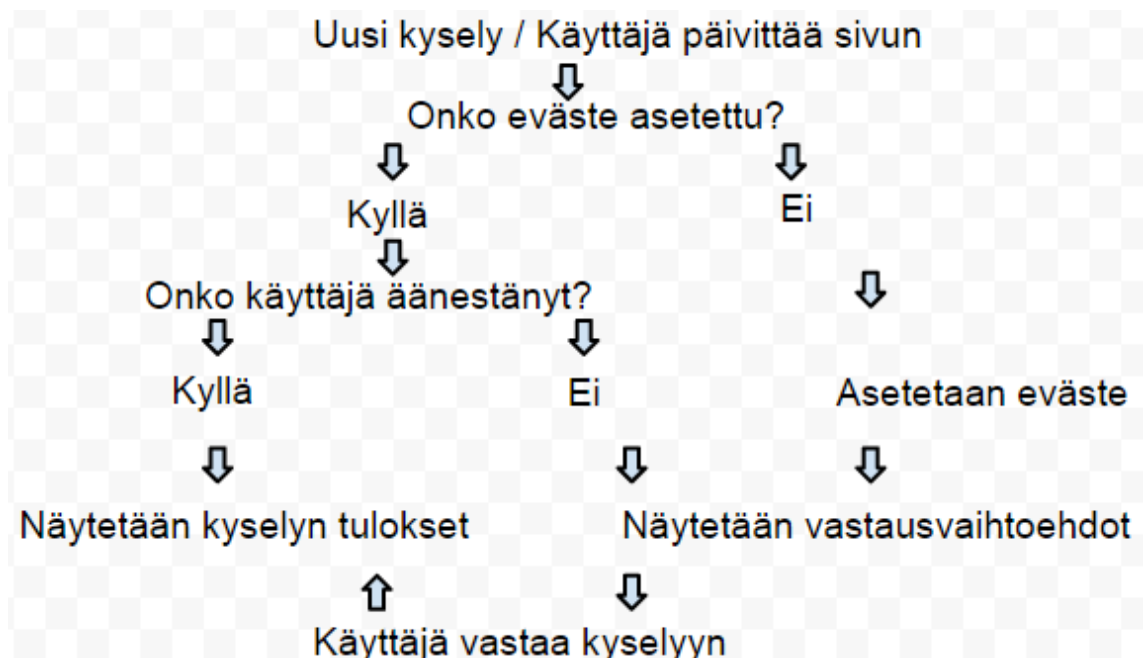
ohjelmistokehittäjän aiempi kokemus ja ohjelmistokehityksen helppokäyttöisyys sovelluksessa, jossa käsitellään paljon lomakkeita. AngularJS:n two-way data binding -ominaisuus oli myös yksi valintakriteeri, sillä sen avulla sovelluksessa voidaan helposti ja nopeasti toteuttaa lomakkeen käsittelyä (14). Lomakkeen käsittelyssä two-way data binding on sovelluskehityksen kannalta hyvä ominaisuus, sillä mallin tieto päivittyy automaattisesti käyttäjän syöttäessä tietoa lomakkeeseen. Automaattiseen lähdekoodin tarkistukseen valittiin JShint, kehitysympäristöksi Sublime Text-tekstieditori ja versionhallintaa varten Git. Git on ilmainen versionhallinta työkalu, jonka avulla ohjelmistokehittäjä pystyy muun muassa tallentamaan eri versioita sovelluksestaan ja päivittämään sitä osa kerrallaan säilyttäen aina mahdollisuuden palata edelliseen toimivaan versioon. CSS-esiprocessoitavaksi kieleksi valittiin SASS ja CSS-ohjelmistokehitykseksi Twitterin kehittämä, avoimeen lähdekoodiin perustuva Bootstrap. Yhtenä tärkeänä kriteerinä kehitystyökaluja valittaessa oli niiden saatavuus ja kustannukset. Kaikki valitut kehitystyökalut ovat ilmaisia ja saatavilla kaikille eri käyttöjärjestelmille. Kehitystyötä tehtiinkin OS X-, Windows- ja Linux-käyttöjärjestelmillä.

Jotta kyselyitä luova taho pystyy kirjautumaan sovellukseen omilla tunnuksillaan, sovellukseen tarvitaan käyttäjänhallinta. Käyttäjänhallinnalla tarkoitetaan sitä, että sovellukseen pystyy kirjautumaan sisään, siellä voi tehdä joitakin toimintoja omalla käyttäjätunnuksellaan ja lopuksi pystyy lopettamaan session kirjautumalla ulos. Kuvassa 11 on esitelty tapahtumaketju, jossa sovelluksen käyttäjä yrittää kirjautua sisään sovellukseen. PHP-ohjelmointikielessä sessioiden hallitseminen on varsin suoraviivaista. Sessio alkaa käyttäjän alkaessa käyttää sovellusta, ja se päättyy joko käyttäjän kirjautuessa ulos tai jonkin määritellyn ajan kuluttua. Tässä sovelluksessa tuo aika määriteltiin 30 minuuttiin, jos käyttäjä ei sinä aikana tehnyt yhtäkään toimintoa. Sessio tässä tapauksessa asetti käyttäjälle uniikin tunnisteen, jonka avulla sovellus oli tietoinen, että jonkin toiminnon tai kyselyn tehnyt käyttäjä oli juuri tämä yksi ja sama käyttäjä. Koska sovelluksessa on näkyvä osa, palvelinpuoli ja niitä yhdistävä ohjelmointirajapinta, tarvitsi jokaista kyselyä tehdessä tarkistaa käyttäjän käyttöoikeus palvelimelta.



Kuva 11. Yksittäisen toiminnon tapahtumaketju sovelluksessa.

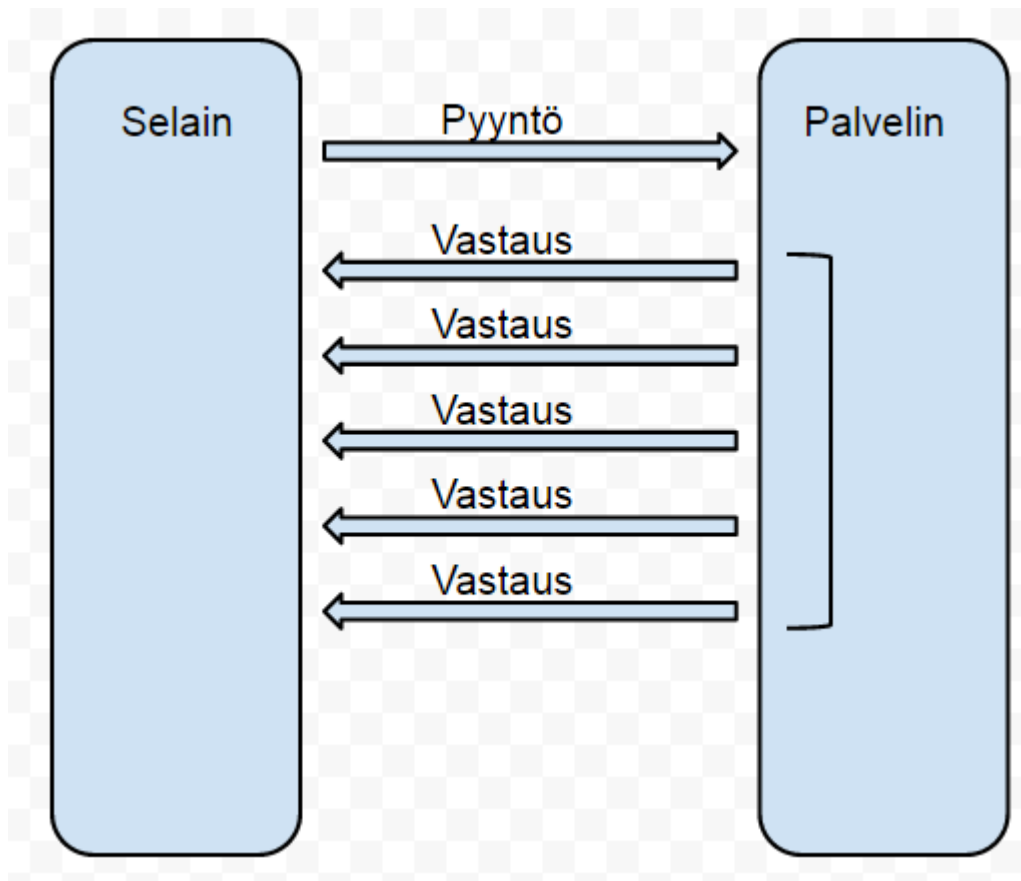
Session avulla pystyttiin myös varmentamaan sovelluksessa kyselyihin vastanneiden käyttäjien pystyvän vastamaan vain yhden kerran jokaiseen kyselyyn. Käyttäjän aloittaessa sovelluksen käytön aloitettiin sessio, joka asetti käyttäjän päätelaitteelle uniikin tunnisteen, joka tallennettiin käyttäjän internetselaimen evästeisiin. Vastauksia antaneiden käyttäjien päätelaitteilta sessioita ei tarvitse lopettaa, koska heidän saamaansa uniikkia tunnistetta voidaan käyttää jatkossa, jos käyttäjä palaa käyttämään sovellusta. Sessio voitaisiin kuitenkin lopettaa tuhoamalla annettu eväste käyttäjän päätölaitteelta, mutta se ei ollut tarpeellista. Evästeen tallentaminen ei aiheuta käyttäjälle minkäänlaista haittaa, eikä se vie tilaa päätölaitteen tietojärjestelmästä kuin aivan olemattoman osan. Kuvasssa 12 on kyselyyn vastaavan käyttäjän käyttötapauksessa tapahtuva tapahtumaketju, jonka avulla koetetaan varmentaa sitä, että käyttäjä pystyy vastaamaan kyselyyn ainoastaan yhden kerran.



Kuva 12. Sovelluksen logiikka käyttäjän antaessa vastausta kyselyyn.

Käyttäjän vastatessa kyselyyn sovellus lähettää sovelluksen ohjelmointirajapintaan uniikin tunnisteen kyselystä ja hänen valitsemastaan vastausvaihtoehdosta. Nämä tiedot tallennetaan tietokantaan, josta ne ovat muidenkin käyttösessioiden käytettävissä.

Verkkoliikenteen minimoimiseen ja moderneihin SPA-sovelluksiin soveltuvista ohjelmistokehitystekniikoista valittiin Server-sent events. Se on osa HTML5-standardia, ja se toimii ainoastaan uusimmissa internet-selaimissa. Ilman tätä teknologiaa, jotta sovelluksen käyttäjä saisi tiedon vaikkapa annettujen vastausten prosentuaalisesta jakaumasta, sovelluksen tulisi jonkin tietyn ajan välein kysyä ohjelmointirajapinnalta nykyistä vastaustilannetta. Kuvassa 13 esitellyssä tapauksessa Server-sent events mahdollistaa sen, että käyttäjän internetselain tekee yhden HTTP-pyyntön saadakseen kyselyn nykyisen vastaus tilanteen, mutta jättää HTTP-yhteyden päätelaitteen ja HTTP-pyyntön kohteen kanssa. Pyyntöön vastaava taho pystyy itse määrittelemänsä ajan välein lähettämään päivitettyä informaatiota käyttäjälle. Ilman Server-sent eventsiä käyttäjän tarvitessa kerran sekunnissa päivitetyn tilan kyselystä, hänen tulisi tehdä 60 erillistä HTTP-pyyntöä sekunnin välein. Server-sent eventsin avulla HTTP-pyyntöjen määrä vähenee yhteen, mutta käyttäjän päätelaite voi silti saada sekunnin välein päivitettyä informaatiota. Jokaisessa HTTP-pyyntössä ja vastauksessa liikkuu tietty määrä metadatan, joka kuvailee esimerkiksi, minkälainen pyyntö on kyseessä, miltä internetselaimelta se tulee, mikä versio internetselaimesta käyttäjällä on ja lukuisia muita tietoja. Kun pyyntöjen määrää voidaan vähentää, vähenee verkkoliikenteen ja varsinkin tämän metadatan edestakaisen lähettelyn määrä huomattavasti. Tämä myös vähentää palvelimen kuormitusta, sillä sen ei tarvitse käsitellä niin montaa HTTP-pyyntöä. Tässä insinööriyössä toteutetussa sovelluksessa palvelin vastasi käyttäjille kerran sekunnissa. Pidempi aikaväli haittaisi sovelluksen reaaliaikaisuutta, ja lyhyempi aikaväli taas kuormittaisi palvelinta tuomatta minikäänlaista lisäarvoa loppukäyttäjille.



Kuva 13. Server-sent events-tekniikkaa käytettäessä tapahtuvat HTTP-pyyntöt.

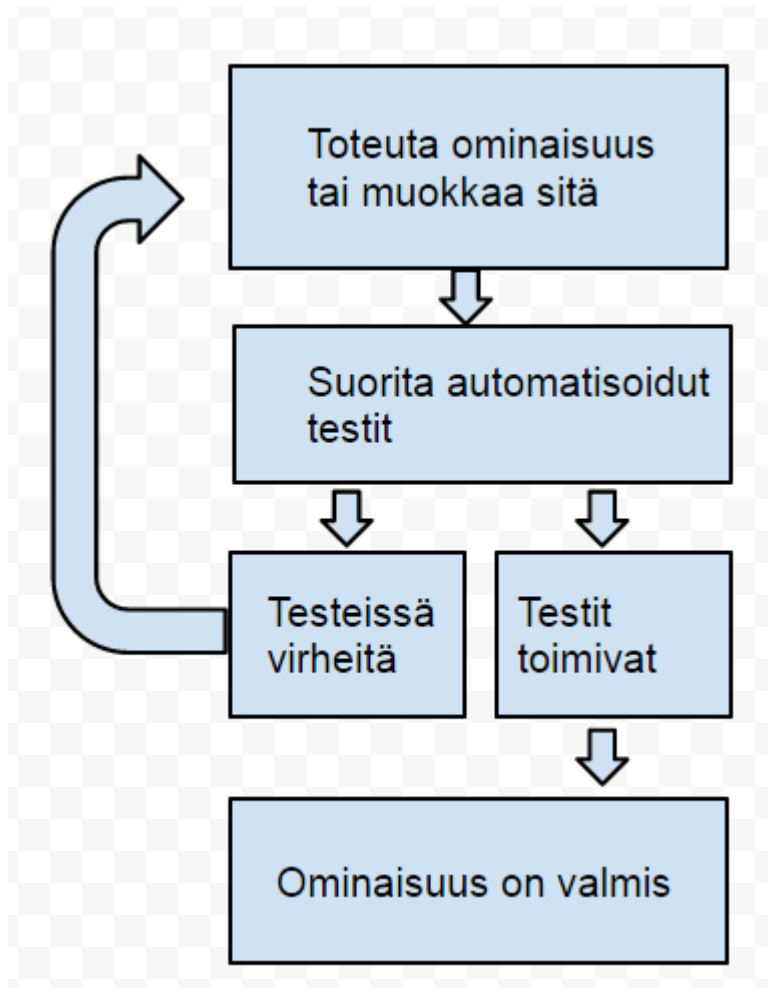
4.2 Ongelmakohtat

Insinöörityön toteutuksen vaatimusmääritelmien perusteella tehtyjen ratkaisujen ja ulkopuolisten seikkojen takia sovelluksessa kuitenkin on ongelmakohtia. Käytettyjen SPA-sovellusarkkitehtuuriin sopivien modernien teknologioiden ongelmiksi muodostuvat loppukäyttäjien käyttämät internetselaimet. Sovellus toimii oikein ainoastaan aivan uusimilla internetselaimilla; vanhemmilla selaimilla se ei toimi lainkaan. Internet-sovelluskehityksessä käytettävät tekniikat ja teknologiat uudistuvat ja kehittyvät nopeasti, mutta käyttäjien käyttämät internetselaimet eivät niinkään nopeasti. Usein ongelmaksi myös muodostuu se, että käyttäjä ei vain ole päivittänyt käyttämäänsä internetselainta uusimpaan versioon. Voidaan kuitenkin olettaa, että jonkin ajan kuluttua käyttäjien internetselaimet ovat kuitenkin päivittyneet sellaisiin versioihin, jotka tukevat tässä insinöörityössä kehitettyä sovellusta.

Sovellus yritti myös ratkaista internetsovellusten kehittämisen klassista ongelmaa, jossa käyttäjä, joka ei ole kirjautunut sovellukseen omilla tunnuksillaan, pystytään silti identifioimaan uniikiksi sekä yhdeksi ja samaksi käyttäjäksi. Tämän insinööriyön sovelluksessa kyselyyn vastaava käyttäjä pystyisi yksinkertaisesti vastauksen annettuaan tuhoamaan evästeen käyttölaitteeltaan ja sen takia vastaamaan kyselyyn uudestaan. Tähän ei nykyisellä teknologialla ole saumatonta ratkaisua, mutta tässä sovelluksessa sääntöjen kiertämistä kuitenkin onnistuttiin vaikeuttamaan huomattavasti.

4.3 Mahdollisuudet jatkon kannalta

Valitut teknologiat ja aputyökalut tarjoavat jatkokehityksen mahdollisuuden kenelle tahansa, jolle sovelluksen lähdekoodi annetaan. Sovelluksen ohjelmistokehyksenä käytetty AngularJS on yksi maailman suosituimpia JavaScript-ohjelmistokehyksiä, ja sovelluksen lähdekoodi noudattaa AngularJS:n määrittelemiä parhaita käytäntöjä. Jatkokehityksessä lähdekoodia ajatellen voitaisiin ottaa käyttöön automaattisen koodintarkistuksen lisäksi automaattinen testaus (4, s. 207). Kuvassa 14 esitelty tapahtumaketju havainnollistaa sovelluskehittäjän kannalta tilannetta, jossa automaattinen testaus varmentaa sovellukseen tehdyn muutoksen toimivuuden. Automaattisella testauksella tarkoitetaan sitä, että jollakin testaamiseen soveltuvalla ohjelmointi- tai merkintäkielellä määriteltäisiin tietyt ehtoja, joita sovelluksen lähdekoodin tiettyjen osien tulisi täyttää. Käytännössä tämä tarkoittaa, että jatkokehitystä tehdessä voisi helposti varmistua siitä, että uudet muutokset eivät aiheuttaisi ongelmia tai virhetilanteita jo aiemmin tuotetussa lähdekoodissa. Testauksessa voisi myös testata sovelluksen käyttäytymistä eri internetse-laimissa, jolloin ohjelmistokehittäjän ei tarvitsisi kokeilla sovelluksen jokaista toiminnallisuutta erikseen jokaisella erilaisella selaimella.



Kuva 14. Automatisoitu testausprosessi.

Tässä insinööriyössä kehitetyn sovelluksen lähdekoodin ylläpidettävyys on siis ohjelmistokehityksen kannalta standardeja noudattava, mutta sen lisäksi sovelluksen kehityksen historia on saatavilla versionhallinnasta niille käyttäjille, joille annetaan tarvittavat käyttöoikeudet tarkistella tätä versionhallinnan historiaa. Mahdolliset uudet jatkokehittäjät voisivat siis työskennellä keskenään ja keskittää kaikki muutokset sovelluksen lähdekoodiin yhteiseen versionhallintaan. Tämä estäisi lähdekoodin pirstaloitumisen lukuisiin enemmän tai vähemmän toimiviin versioihin. Lisäksi voitaisiin pitää kirjaa siitä, ketkä vaikuttavat eniten sovelluksen jatkokehitykseen ja minkälainen panos heillä on ollut sovelluksen jatkokehityksen kannalta.

Sovellukseen voisi myös luoda lisää sisältöä, etenkin kyselyitä luoville käyttäjille. Tällaista sisältöä voisi olla muun muassa tilastoja luoduista kyselyistä ja niihin annetuista

vastauksista. Kaikki sovellukseen luodut kyselyt ja niihin annetut vastaukset ovat tallennettuna tietokantaan, joten niistä voisi johtaa erilaisia kuvaajia ja tilastoja ilman, että jo aiemmin tallennettuun tietoon tarvitsisi tehdä minkäänlaisia muutoksia.

5 Yhteenveto

Insinööriyössä toteutettiin internetsivulla toimiva kyselytyökalu, jonka avulla käyttäjä voi luoda kyselyitä ja saada niihin anonyymeiltä käyttäjiltä vastauksia. Kyselytyökalun käyttötapauksia on useita, kuten aiemmin mainittu opetustilaisuus, sillä työssä toteutettu sovellus voi hyvinkin toimia jonkin suuremman ohjelmistokokonaisuuden osana tuomassa jonkinlaista lisäarvoa. Tämänkaltaisia työkaluja on olemassa: esimerkiksi <http://www.polleverywhere.com> on kaupallinen tuote, joka toteuttaa tämän saman asian useammilla ominaisuuksilla ja sisältää esimerkiksi tuen SMS-viesteille. Tässä insinööriyössä ei kuitenkaan ollut tarkoitus kehittää mullistavaa kyselytyökalua, vaan tarkastella, kuinka SPA-sovelluksen kehittäminen käytännössä tapahtuu. Kyselytyökalu on kuitenkin erinomainen esimerkki siitä, mihin SPA-sovellus soveltuu. SPA-sovellusarkkitehtuurin suomalla nopealla ohjelmistokehityksellä ja aiemmin esiteltyjen aputyökalujen avulla on mahdollista luoda hyvin ylläpidettävä sovellus nopeassa aikataulussa. Näiden aputyökalujen avulla mitä tahansa SPA-sovellusta on helpompi ylläpitää ja jatkokehittää, sillä ne helpottavat rakentamaan toimivan rungon sovellukselle, jonka päälle voi jatkaa kehittämistä.

Työstä tuli onnistunut, sillä sovellus on toimiva, se vastaa teknisessä suunnittelussa todettuja määritelmävaatimuksia ja se on ollut Metropolia Ammattikorkeakoulussa käytössä yliopettaja Kari Salolla. Työssä käytettiin myös kaikista uusimpia SPA-sovellukseen sopivista kehitysteknologioista, ja ne todettiin toimiviksi. Käytetyt ohjelmat, aputyökalut ja ohjelmistokehitykset toivat sovellukseen lisäarvoa ja mahdollistavat sen ylläpidettävyyden ja jatkokehityksen ja ennen kaikkea helpottavat näitä kahta esimerkiksi automaattisella lähdekoodin tarkistuksella. Ongelmana uusimpien teknologioiden käytössä internetsovelluksissa on huono internetselaintuki, sillä sovellus toimii tällä hetkellä ainoastaan selainten lähes uusimmilla versioilla. Työllä on kuitenkin konkreettista käyttöarvoa, sillä sellaisenaan se tekee käyttäjälle mahdolliseksi luoda verkossa kyselyitä, joihin muut käyttäjät voivat vastata. Yksinkertaiseen ongelmaan yksinkertainen ratkaisu, vaikka aputyökalujen määrä saattaa kuulostaa suurelta.

Lähteet

- 1 Pike, Allen. 2015. A JS framework on every table. Verkkodokumentti. <<http://www.allenpike.com/2015/javascript-framework-fatigue/>>. Luettu 4.4.2015.
- 2 Pupius, Daniel F. 2013. Rise of the SPA. Verkkodokumentti. <<https://medium.com/@dpup/rise-of-the-spa-fb44da86dc1f>>. Luettu 1.4.2015.
- 3 Beletsky, Alexander. 2013. Building Single Pages Applications. Verkkodokumentti. <<http://beletsky.net/2013/04/building-single-pages-applications.html>>. Luettu 1.4.2015.
- 4 Fink, Gil & Flatow, Ido. 2014. Pro Single Page Application development. Apress.
- 5 Grunt, The JavaScript Task Runner. 2015. Verkkodokumentti. Bocup. <<http://www.gruntjs.com>>. Luettu 2.4.2015.
- 6 Suh, Jonathan. 2014. Get Started with Grunt. Verkkodokumentti. <<https://jonsuh.com/blog/get-started-with-grunt>>. Luettu 2.4.2015.
- 7 A Package management for the web. 2015. Verkkodokumentti. Bower. <<http://www.bower.io>>. Luettu 2.4.2015.
- 8 npm. 2015. Verkkodokumentti. npm, Inc. <<https://www.npmjs.com>>. Luettu 2.4.2015.
- 9 Paquette, Dave. 2014. Using Bower with Visual Studio 2013. Verkkodokumentti. <<http://www.davepaquette.com/archive/2014/12/27/using-bower-with-visual-studio-2013.aspx>>. Luettu 2.4.2015.
- 10 What is CSS? 2015. Verkkodokumentti. W3C. <<http://www.w3.org/Style/CSS/>>. Luettu 5.4.2015.
- 11 JSLint. 2015. Verkkodokumentti. <<http://www.jshint.com>>. Luettu 2.4.2015.
- 12 Raboy, Nic. 2014. Use Grunt To Lint And Uglify Your JavaScript Project. Verkkodokumentti. Nic Raboy. <<https://blog.nraboy.com/2014/12/use-grunt-lint-uglify-javascript-project/>>. Luettu 3.4.2015.
- 13 Shaked, Uri. Javascript Framework Comparison. Verkkodokumentti. <<https://www.airpair.com/js/javascript-framework-comparison>>. Luettu 6.4.2015.
- 14 Herskovits, Itay. 2015. AngularJS Two-Way Data Biding. Verkkodokumentti. <<http://java.dzone.com/articles/angularjs-two-way-data-binding>>. Luettu 3.4.2015.

- 15 Cao, Mike. 2015. FlightPHP. Verkkodokumentti. <<http://flightphp.com/>>. Luettu 1.4.2015.
- 16 MySQL. 2015. Verkkodokumentti. Oracle. <<https://www.mysql.com/>>. Luettu 6.4.2015.