Petteri Aho

JAVASCRIPT – ENNEN JA NYT

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2015

TIIVISTELMÄ

Aho, Petteri
JavaScript – ennen ja nyt
Jyväskylä: Jyväskylän yliopisto, 2015, 105 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Sakkinen, Markku

JavaScript on saanut kritiikkiä hitaudesta ja lisäksi se on kärsinyt maineesta lelukielenä, joka on hyödyllinen vain suhteellisen yksinkertaisiin tehtäviin. Aikaisemmin web-sivut oli rakennettu pelkän HTML:n avulla staattisiksi sivuiksi. Ne näkyivät selaimessa aina samanlaisina ja muuttuivat vain, jos tiedostoa muokattiin palvelimella. Nykypäivänä useimmat web-sivut käyttävät JavaScriptiä, jonka tärkein tehtävä on lisätä web-sivuille vuorovaikutteisuutta ja dynaamista toiminnallisuutta. JavaScriptin käyttö on yleistynyt sen yksinkertaisuuden, joustavuuden, monipuolisten lisäosien ja tehokkaan suorituskyvyn ansiosta.

Tutkimuksessa käytettiin aihepiirin kirjallisuuteen perustuvaa teoreettis-käsitteellistä tutkimusta ja suunnittelutieteellistä, konstruktiivista tutkimusta. Sen päätavoitteena oli täydentää aikaisempia tutkimustuloksia toteuttamalla erilaisilla menetelmillä kaksi toiminnallisuudeltaan ja ulkoasultaan samanlaista web-sovellusta ja vertailemalla niiden laatuominaisuuksia. Sovellusten tarkoitus on nauhoittaa ja tallentaa käyttäjän kirjaamia erilaisia työtehtäviä muistiin. Sovellusten tehokkuutta, ylläpidettävyyttä, kytkentää ja kokoa mitattiin standardointijärjestöjen määrittelemien laatustandardien avulla. Ensimmäisessä versiossa JavaScriptin tukena käytettiin kolmansien osapuolten tekemiä kirjastoja, automaatiotyökalua ja sovelluskehystä. Toinen versio ei käyttänyt tällaisia apuvälineitä.

Tutkimus osoitti, että kirjastojen, automaatiotyökalun ja sovelluskehyksen käyttö heikensi sovelluksen tehokkuutta sivun alustusvaiheessa ja kasvatti sen kokoa. Alustuksen jälkeen näitä menetelmiä käyttävä sovellus toimi kuitenkin tehokkaammin kuin verrokkisovellus. Menetelmien käyttö parantaa sovelluksen ylläpidettävyyttä ja kytkentää. Nämä asiat vaikuttavat sovelluksen kehitys- ja ylläpitokustannuksiin. Kehittyneiden ja monipuolisten kirjastojen ja ominaisuuksien ansiosta negatiivinen käsitys JavaScriptistä vähenee jatkuvasti.

Asiasanat: JavaScript, web-sovellus, sovelluskehys, automaatiotyökalut, JavaScript-kirjastot, web-sovellusten vertailu, konstruktiivinen tutkimus

ABSTRACT

Aho, Petteri
JavaScript – Then and Now
Jyväskylä: University of Jyväskylä, 2015, 105 p.
Information Systems Science, Master's Thesis
Supervisor: Sakkinen, Markku

JavaScript has been criticized for its slowness and it has suffered from its reputation as a toy language that is useful only for relatively simple scripting tasks. Earlier web pages were constructed with plain old HTML as static pages. They looked always the same on a browser and changed only if a file was edited on a server. These days, most web pages use JavaScript because it is possible to create dynamic and interactive functionality with it. JavaScript has become a more common language because it is very simple to use, flexible, efficient and there is a wide range of add-ons.

A theoretical and conceptual study based on literature, and constructive research based on design science were used as the research methods. The main purpose of this research was to build upon previous research results by implementing two web applications with similar functionality and layout, and comparing their quality attributes. The web applications were created using different methods. The purpose of the applications is to record different tasks that the user wants to save. The efficiency, maintainability, coupling and size of the applications were measured with the help of quality standards defined by standard organizations. The first version used JavaScript libraries, a task runner and a framework created by a third party. The second version did not use such tools.

This study showed that the use of the libraries, task runner and framework decreased the efficiency of the application at the page initialization phase, and increased its size. After the initialization, however, the application built with those methods run more effectively than the other one. The maintainability and coupling of the application are improved through the use of these methods. The development and maintenance costs of the application are affected by these issues. As more advanced JavaScript libraries and applications become available, the negative perception of JavaScript will gradually improve.

Keywords: JavaScript, Web application, Software framework, JavaScript task runner, JavaScript libraries, Comparison of web applications, Constructive research

SANASTO

Tämä sanasto pitää sisällään termejä, jotka on mainittu tutkimuksessa mutta joita ei ole selitetty erikseen auki.

CSS

Cascading Style Sheets on web-sivuille kehitetty tyyliohjeiden laji.

CSS Media Queries

CSS:n mediakyselyjen avulla on mahdollista määrittää erilaisia tyylejä tietyn kokoisille näytöille. Mediakyselyjen avulla voidaan web-sivusta tehdä responsiivinen, eikä erillistä mobiiliversiota tarvita.

Java Applet

Java-sovelma on selaimen yhteydessä suoritettava Java-ohjelma, joka toimii HTML-sivun elementtinä.

Netscape Communications Corporation

Netscape Communications Corporation on web-ohjelmistoihin ja telekommunikaatioon perustuva yhtiö, joka julkaisi Netscape Navigator- ja Netscape Communicator -selaimet.

Open Source Initiative

Open Source Initiative (OSI) on järjestö, joka pyrkii edistämään avoimen lähdekoodin ohjelmistojen käyttöä.

Oracle Corporation

Oracle Corporation on tietotekniikkayhtiö, joka kehittää ja valmistaa yritysohjelmistoja ja laitteistoja.

Reflektio-mekanismi

Reflection mechanism mahdollistaa mm. vielä käännösaikana tuntemattoman tyyppisten olioiden luomisen, näiden ominaisuuksien tutkimisen ja käyttämisen.

Lively Kernel

Lively Kernel on avoimen lähdekoodin vuorovaikutteinen web-ohjelmointiympäristö, joka on toteutettu JavaScriptin avulla.

SSL/TLS

Secure Sockets Layer (SSL) on tekniikka, jossa otetaan VPN-yhteys SSL-yhteyden yli suljettuun verkkoon etätyöasemalta tai yhdistetään kaksi suljettua verkkoa keskenään. Transport Layer Security (TLS) on salausprotokolla, jolla voidaan suojata web-sovellusten tietoliikenne IP-verkkojen yli.

SOAP

SOAP on standardoitu tapa välittää dataa operaatiolta ja operaatiolle WSDL:ssä määriteltyjen porttien kautta.

TCP/IP

Transmission Control Protocol/Internet Protocol on usean Internetissä käytettävän tietoverkkoprotokollan yhdistelmä.

TIOBE Index

TIOBE Programming Community index on mittari eri ohjelmointikielten suosiosta. Listaa päivitetään kerran kuukaudessa. Luokitukset perustuvat koulutettujen insinöörien näkemyksiin ja kolmannen osapuolen toimittajiin. Lisäksi listassa otetaan huomioon ohjelmointikieliin liittyviä hakutuloksia seuraavista palveluista: Google, Bing, Yahoo!, Wikipedia, YouTube ja Baidu. TIOBE Index ei siis kuvaa parasta ohjelmointikieltä tai kieltä, jolla useimmat sovellukset on ohjelmoitu.

UDDI

Universal Description, Discovery and Integration on rekisteri, jonka avulla palvelujen tarjoajat voivat julkaista tietoja itsestään ja tarjoamistaan palveluista sekä etsiä tietoa muista palveluista.

URI

Uniform Resource Identifier on merkkijono, jonka avulla voidaan kertoa tietyn tiedon paikka tai yksittäinen nimi.

WSDL

Web Service Description Language on alun perin Microsoftin, IBM:n ja Ariban ehdottama standardi web-palveluiden liittymien määrittämiseen.

W3C

World Wide Consortium on vuonna 1994 perustettu kansainvälisten yritysten, tutkimuslaitoksien ja yhteisöjen yhteistyöhanke, jonka tavoitteena on kehittää ja ylläpitää web-standardeja tai suosituksia, kuten W3C niitä kutsuu.

XML

Extensible Markup Language on merkintäkieli tai standardi, jolla tiedon merkitys on kuvattavissa tiedon sekaan. XML:ää käytetään formaattina tiedonvälitykseen järjestelmien välillä sekä dokumenttien tallentamiseen. XML on tekstimuotoista ja muistuttaa HTML:ää.

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

> *"JavaScript on kieli, jolla on rikas historia ja vielä rikkaampi tulevaisuus."*
> Wright Tim (2012)

Web-sivut olivat melko tylsiä aikaisemmassa kehitysvaiheessaan. Ne oli rakennettu pelkän HTML:n avulla staattisiksi sivuiksi, jotka näkyivät selaimessa aina samanlaisina ja muuttuivat vain, jos tiedostoa muokattiin palvelimella. Vuorovaikutteisuus perustui lähinnä linkkien painamiseen, jolloin käyttäjälle ladattiin uusi web-sivu. (McFarland, 2011.). Nykypäivänä useimmat web-sivut käyttävät JavaScriptiä, jonka tärkein tehtävä on lisätä web-sivuille vuorovaikutteisuutta ja dynaamista toiminnallisuutta. Dynaamisen web-sivun perusidea on, että se luodaan silloin, kun selain sitä pyytää (TechTerms.com, 2009a). JavaScript voi esimerkiksi näyttää välittömästi virheviestin, jos käyttäjä yrittää lähettää weblomakkeen, josta puuttuvat tarvittavat tiedot (McFarland, 2011). JavaScriptistä on tulossa yksi suosituimmista ohjelmointikielistä, koska se on ylivoimaisesti yleisin web-sivuilla käytettävä asiakaspään dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli (Lebresne, Richards, Östlund, Wrigstad & Vitek, 2009; Flanagan, 2011).

JavaScriptin ympärille julkaistut uudet kirjastot ja sovelluskehykset ovat olleet viime aikoina paljon esillä IT-alan eri julkaisuissa. Esimerkiksi *InfoWorld*, *IDG Connect* (*ICT and the Global Community*) ja *IT Whitepapers* julkaisevat paljon JavaScriptiin liittyviä uutisia. Aikaisemmat tutkimukset keskittyvät pääasiassa JavaScript-kirjastojen ja sovelluskehysten vertailuun, jonka lisäksi on tutkittu kielen dynaamisia ominaisuuksia, käyttömahdollisuuksia, valmiiden olioiden ja funktioiden suorituskykyä, muistin käyttäytymistä, tyypillisiä ongelmatilanteita sekä tietoturvaongelmia. Tästä huolimatta aikaisemmissa tutkimuksissa ei ole keskitytty riittävästi vertaamaan kahdella erilaisella menetelmällä toteutettua web-sovellusta, joiden pääpaino on JavaScriptissä.

Tutkimuksen tavoitteena oli täydentää aikaisempia tutkimustuloksia toteuttamalla erilaisilla menetelmillä kaksi toiminnallisuudeltaan ja ulkoasultaan samanlaista web-sovellusta. Sovelluksia vertailtiin toisiinsa alaluvussa 5.3 määriteltyjen mittausmenetelmien avulla. Ensimmäisessä versiossa JavaScriptin

tukena käytettiin kolmansien osapuolten tekemiä kirjastoja, automaatiotyökalua ja sovelluskehystä. Toinen versio ei käyttänyt tällaisia apuvälineitä (ks. alaluku 3.3 ja 3.4). Tutkimuksen tutkimusongelma on:

> *Miten nykyaikaiset JavaScript-kirjastot, sovelluskehys ja automaatiotyökalu vaikuttavat asiakaspään web-sovelluksen tehokkuuteen (efficiency), ylläpidettävyyteen (maintainability), kytkentään (coupling) ja kokoon (size)?*

Tutkimusongelma on jaettavissa seuraaviin tutkimuskysymyksiin:

- Kuinka tutkimusongelmassa määriteltyjä laatuattribuutteja voidaan mitata?
- Mitä tarkoitetaan nykyaikaisella web-ohjelmoinnilla JavaScriptin avulla?
- Mitä ominaisuuksia JavaScript-sovelluskehys, erilaiset kirjastot ja automaatiotyökalu auttavat parantamaan web-ohjelmoinnissa?

Tutkimuksessa käytettiin aihepiirin kirjallisuuteen perustuvaa teoreettiskäsitteellistä tutkimusta ja suunnittelutieteellistä, konstruktiivista tutkimusta. Teoreettisen tutkimuksen avulla kohteesta pyrittiin hahmottamaan käsitteellisiä malleja, selityksiä ja rakenteita aiemman tutkimuskirjallisuuden pohjalta (Jyväskylän yliopisto, 2014). Konstruktiivisen tutkimusotteen määrittelemää työtapaa käytettiin antamaan konkreettisia tutkimustuloksia siitä, kuinka nykyaikainen web-ohjelmointi JavaScriptin avulla on vaikuttanut asiakaspään websovellusten *tehokkuuteen*, *ylläpidettävyyteen*, *kytkentään* ja *kokoon*. Ratkaisukonstruktion toimivuutta testattiin tutkimuksessa vertaamalla toisiinsa kahta toiminnallisuudeltaan sekä ulkoasultaan samanlaista JavaScript-sovellusta, jotka on toteutettu erilaisilla menetelmillä. Tuloksena saatiin tieto, kuinka erilaisilla menetelmillä toteutetut sovellukset eroavat toisistaan alaluvussa 5.3.2 määriteltyjen laatuattribuuttien osalta.

Tutkimus jakautuu kahdeksaan lukuun. Tutkielman teoreettinen osuus (luvut 2–5) vastaa kahteen ensimmäiseen tutkimuskysymykseen. Konstruktiivisen osuuden (luku 6) avulla voidaan vastata viimeiseen tutkimuskysymykseen ja tutkimusongelmaan. Luvussa 2 esitetään JavaScriptin historiaa, käsitteitä, toimintaympäristöä, käyttömahdollisuuksia ja web-sovelluksen tiedonsiirtotapoja. Luvussa 3 käsitellään JavaScriptin suosioon vaikuttavia tekijöitä, nykyaikaisen web-sovelluksen edellytyksiä ja esitellään toteutuksessa käytettävät JavaScript-kirjastot sekä sovelluskehys. Luku 4 määrittelee konstruktiiviselle tutkimukselle käytettävän tutkimusprosessin. Luvussa 5 esitellään tutkimuksessa toteutetut sovellukset, aikaisempaa tutkimusta, sovelluksen vertailuun käytetyt mittausmenetelmät ja ohjelmointivaihe. Luvussa 6 analysoidaan sovelluksia luvussa 5 määriteltyjen mittausmenetelmien avulla. Luvussa 7 esitetään tutkimuksen tulokset, verrataan niitä aiempiin tutkimuksiin, esitetään johtopäätökset sekä tunnistetaan ja analysoidaan teoreettinen kontribuutio. Tutkimus päättyy yhteenvetoon (luku 8).

2 JAVASCRIPT

Alaluvuissa 2.1 ja 2.2 luodaan katsaus JavaScriptin historiaan ja kielen määritelmään. Alaluvussa 2.3 esitellään JavaScriptiin liittyviä käsitteitä. Käsitteet auttavat ymmärtämään paremmin kielen käyttötarkoitusta ja toimintatapaa. Alaluvussa 2.4 käsitellään JavaScriptin toimintaympäristö eli ympäristö jossa JavaScriptiä käytetään. Toimintaympäristön käsittely auttaa hahmottamaan kielen keskeistä toimintaympäristöä ja sen tuomia haasteita. Alaluvussa 2.5 käydään läpi mm. JavaScriptin käyttökohteita ja olio-ominaisuuksia. Alaluvussa 2.6 käydään läpi web-sovelluksen erilaisia tiedonsiirtotapoja.

2.1 Historia

Netscape Communications Corporation palkkasi vuonna 1995 Brendan Eichin kehittämään kymmenessä päivässä toimivan ohjelmointikielen (*JavaScriptin*), jota voitaisiin ajaa Netscape-selaimessa. JavaScript -tavaramerkin on lisensoinut Sun Microsystems, joka kuuluu nykyään Oracle Corporationille (Flanagan, 2011). Uuden ohjelmointikielen kehittäminen kymmenessä päivässä olisi ollut monelle ohjelmoijalle mahdoton tehtävä. Eichillä oli kuitenkin pitkä historia uusien ohjelmointikielien rakentamisesta aina opiskeluajoista lähtien, Illinoisin yliopistossa. (Severance, 2012a.).

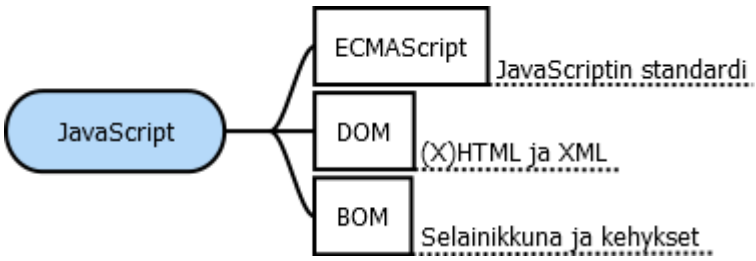
Netscape palkkasi Eichin kehittämään uuden ohjelmointikielen, koska se halusi saada Sun Microsystemsin kehittämän Java-teknologian rinnalle kevyemmän ohjelmointikielen. Uuden ohjelmointikielen piti muistuttaa Microsoftin Visual Basic-ohjelmointikieltä, ja tämän lisäksi ohjelmointikielen piti olla helposti tulkittava ja upotettavissa web-sivulle. Eich on kertonut, että vaikein tehtävä oli toteuttaa rikas ja tehokas ohjelmointikieli käyttämättä samanlaista oliosyntaksia, jota Java käytti. (Severance, 2012a.). Vaikka JavaScript muistuttaa nimeltään Java-ohjelmointikieltä, ne eivät liity millään tavalla toisiinsa. JavaScriptiä kehitettiin alun perin nimellä *Mocha*, jonka jälkeen nimi muutettiin *LiveScriptiksi*. Kaupallisuuden lisäämiseksi nimi muutettiin lopulta muistutta-

maan enemmän Java-ohjelmointikieltä. (McFarland, 2011). JavaScript tunnetaan myös nimellä ECMAScript, joka on JavaScript-kielen standardoitu versio. JavaScriptistä on tehty myös useita eri versioita ja muunnoksia, jotka perustuvat ECMAScriptiin – yksi muunnoksista on Microsoftin kehittämä JScript. (Peltomäki & Nykänen, 2006.).

2.2 JavaScriptin määritelmä

Aikaisemmin *web-sivut* (*Web pages*) oli rakennettu pelkän HTML:n avulla staattisiksi sivuiksi. Sivut näkyivät selaimessa aina samanlaisina ja muuttuivat vain, jos tiedostoa muokattiin palvelimella. Vuorovaikutteisuus perustui lähinnä linkkien painamiseen, jolloin käyttäjälle ladattiin uusi web-sivu (McFarland, 2011). Nykyään Internetin käyttäjistä yli 95 % käyttää (useimmiten tietämättään) JavaScriptiä selatessaan web-sivuja. Tämä ei ole yllätys, koska yli 99 % web-sivustoista tukee JavaScriptiä ja kielen suosio kasvaa yhä enemmän niin käyttäjien kuin kehittäjienkin käytössä (Tiwari & Solihin, 2012).

Flanagan (2011) on kuvaillut JavaScriptiä seuraavasti: JavaScript on pääasiassa asiakaspäässä eli web-ympäristössä käytettävä korkean tason dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli. JavaScriptin syntaksi perustuu löyhästi C-ohjelmointikieleen – mukaan lukien aaltosulut, puolipisteet ja varatut sanat. JavaScriptiä voidaan kuvailla C-ohjelmointikielen kevyemmäksi, ystävällisemmäksi, semanttisemmaksi ja paremmilla dynaamisilla muistiominaisuuksilla varustetuksi ohjelmointikieleksi (Severance, 2012a). JavaScriptiä käytetään yleensä parantamaan web-sivun vuorovaikutteisuutta ja dynaamista toiminnallisuutta (Yue & Wang, 2013). Dynaamisen web-sivun perusidea on, että sivu luodaan silloin, kun selain sitä pyytää (TechTerms.com, 2009a). JavaScript voi esimerkiksi näyttää välittömästi virheviestin, jos käyttäjä yrittää lähettää web-lomakkeen, josta puuttuvat tarvittavat tiedot (McFarland, 2011). Kuviossa 1 on esitelty kolme osaa, joista JavaScript muodostuu. Osat ovat ECMAScript, DOM ja BOM, jotka esitellään seuraavissa alaluvuissa.



KUVIO 1 JavaScriptin osat (Zakas, 2012, 3)

2.2.1 ECMAScript

ECMAScript (*ECMA-262*) on JavaScriptin standardoitu versio ja tavaramerkki. Kielen on standardoinut ja rekisteröinyt eurooppalaisten tietokonevalmistajien liitto *ECMA* (*European Computer Manufacturers Assosiation*). ECMAan kuuluu useita ohjelmisto- ja laitteistovalmistajia, kuten Microsoft, Netscape, IBM ja Intel. ECMAScript perustuu useisiin eri tekniikoihin, joista tunnetuimmat ovat JavaScript (Netscape) ja JScript (Microsoft). ECMAScriptin standardia noudattavat useat eri web-selainten valmistajat. ECMAScriptin kehitys aloitettiin marraskuussa 1996 ja kesäkuussa 1997 Ecma General Assembly hyväksyi ensimmäisen (1st edition) ECMA-262-standardin virallisesti. ECMA-262-standardi toimitettiin tämän jälkeen ISO/IEC JTC 1:n hyväksyttäväksi nopeutettuun menettelyyn ja se hyväksyttiin kansainväliseksi ISO/IEC 16262-standardiksi huhtikuussa 1998. ISO/IEC JTC 1 on tekninen komitea, johon kuuluu kansainvälinen standardisoimisjärjestö *ISO* (*International Organization for Standardization*) ja kansainvälinen sähköalan standardointiorganisaatio *IEC* (*International Electrotechnical Commission*). ISO/IEC JTC 1 vastaa monista kriittisistä IT-standardeista. Sen tarkoitus on kehittää, ylläpitää ja edistää erilaisia standardeja tieto- ja viestintäteknologiassa. (Peltomäki & Nykänen, 2006.; ECMA, 2011.).

ECMA-262-standardi perustuu pääosin JavaScript-kielen versioon 1.1, joka tuli Netscape Navigator 3 -selaimen mukana. ECMAScriptistä on kehitetty viisi eri versiota. Toinen laitos (2nd edition) julkaistiin elokuussa 1998, ja se pohjautui JavaScript 1.1:een ja JavaScript 1.2:een. Kolmas laitos (3rd edition) toi mukanaan mm. säännölliset lausekkeet, paremman merkkijonojen käsittelyn ja poikkeusten käsittelyn. ECMA hyväksyi kolmannen laitoksen joulukuussa 1999 ja kesäkuussa 2002 se hyväksyttiin kansainväliseksi ISO/IEC 16262 -standardiksi. Merkittävä työ ECMAScriptin kehityksessä tehtiin neljännessä laitoksessa, vaikka sitä ei saatu koskaan valmiiksi tai julkaistu. ECMAScriptin kehitystä jatkettiin viidennellä laitoksella (5th edition), joka toi mukanaan mm. paremman tietoturvan, tehostetun virheentarkistuksen ja lisää taulukoiden muokkaustoimintoja. Viides laitos on hyväksytty kansainväliseksi ISO/IEC 16262 -standardiksi. ECMAScriptin kehitys ei ole vielä valmis, vaan sitä jatketaan tulevaisuudessa. (Peltomäki & Nykänen, 2006.; ECMA, 2011.).

2.2.2 DOM

DOM (*Document Object Model*) on W3C:n määrittelemä ohjelmointirajapinta. DOM mahdollistaa (X)HTML- tai XML-dokumenttien sisällön lukemisen ja muokkauksen ajonaikaisesti. Yhdessä JavaScriptin kanssa sillä voidaan toteuttaa vuorovaikutteisia web-sivuja, jotka eivät vaadi jatkuvaa palvelinyhteyttä. DOMin tarkoitus on määrittää kuinka dokumentissa olevat elementit ymmärretään rakenteiseksi olioksi, jolla on ominaisuuksia (Korpela, 2011). DOM ei sisälly HTML-määritelmään, vaan kyseessä on oma standardi, joka on vuosia kestäneen W3C:n standardointityön tulos. (Nicol, Wood, Champion & Byrne, 2001.). Standardityön ansiosta W3C määrittelee laajan määrän suosituksia, jotka muo-

dostavat niin sanotun W3C DOM-määritelmään. Määritelmään kuuluu kolme eri tasoa, jotka Zakas (2012) on kuvaillut seuraavalla tavalla:

- *DOM Level 1* on lokakuussa 1998 hyväksytty W3C:n virallinen suositus. DOM Level 1 koostuu kahdesta eri päämoduulista: DOM Core ja DOM HTML. DOM Core määrittelee rajapinnat XML-rakenteiden käsittelyyn. Se sisältää rajapinnat määritteiden ja elementtien muokkaamiseen. DOM HTML määrittelee rajapinnat HTML-dokumenttien käsittelyyn, johon kuuluu HTML-elementtien käsittely.
- *DOM Level 2* on marraskuussa 2000 hyväksytty W3C:n virallinen suositus. DOM Level 2 on paljon laajempi kokonaisuus kuin DOM Level 1. DOM Level 2 koostuu neljästä eri päämoduulista: Views, Style, Events ja Traversal and Range. Moduuleihin kuuluvat mm. tapahtumankäsittelymalli, XML-nimiavaruustuki, näkymien luominen samaan dokumenttiin sekä CSS-tuki.
- *DOM Level 3* on huhtikuussa 2004 hyväksytty W3C:n virallinen suositus. DOM Level 3 auttaa lataamaan ja tallentamaan asiakirjoja kaikille yhtenäisellä tavalla. Laajennosta kutsutaan nimellä DOM Load and Save. Lisäksi DOM Level 3 sisältää Xpath-tuen sekä näppäimistön tapahtumien käsittelyn.

DOM koostuu *solmuista* (*nodes*), jotka sisältävät erilaista tietoa. DOMin tarkoitus on määrittää kuinka HTML-dokumentissa olevat elementit voivat välittää tietoa toisillensa ja kuinka näihin elementteihin voidaan viitata. Tämä onnistuu esittämällä dokumenttien tietoa puumaisessa rakenteessa. DOM muistuttaa hyvin pitkälti oikeaa puuta, siinä on *runko* (*body, root node*), *oksat* eli *lapsielementit* (*children, child node*) ja *lehdet* (*nodes*), joilla ei ole omia lapsielementtejä. DOMissa jokainen solmu on *olio* (*object*), joten sillä on *ominaisuuksia* (*properties*) ja *metodeja* (*methods*). Ominaisuuksia voidaan käsitellä JavaScriptillä tai muulla ohjelmointikielellä. (Nicol ym., 2001.).

2.2.3 BOM

BOM (*Browser Object Model*) on pääasiassa vuorovaikutuksessa selainikkunan ja kehysten (*frames*) kanssa. Tavallisesti kaikki selaimet määrittelevät JavaScriptin laajennokset osana BOMia (Zakas, 2012). Zakas (2012) on listannut esimerkkejä JavaScriptin laajennuksista, jotka käsitellään osana BOM-määrittelyä:

- valmius avata uusi web-selain ponnahdusikkunaan
- valmius liikuttaa, muuttaa tai sulkea selainikkuna
- *navigaattori-objekti* (*navigator object*) sisältää yksityiskohtaista tietoa selaimesta
- *sijainti-objekti* (*location object*) sisältää yksityiskohtaista tietoa sivun latauksesta

- *näyttö-objekti* (*screen object*) sisältää yksityiskohtaista tietoa näytön resoluutiosta
- *ikkuna-objekti (window object)* edustaa web-selaimen ikkunaa.
- tuki *evästeille* (*cookies*)
- *mukautetut objektit* (*custom objects*), kuten XMLHttpRequest ja Internet Explorerin ActiveXObject.

Käyttämällä BOMia ohjelmistokehittäjät pystyvät olemaan vuorovaikutuksessa selaimen muihin toiminnallisuuksiin eli ”keskustella” selaimen kanssa. Tämä auttaa paljastamaan selaimen eri toimintoja riippumatta web-sivun sisällöstä. ECMAScriptistä ja DOMista poiketen BOM ei ole standardoitu. BOMin toteutus vaihtelee eri web-selaimissa, lukuun ottamatta window- ja navigaattori-objekteja, jotka ovat standardeja. Uusi HTML5 määritys sisältää nykyään keskeisimmät asiat BOMista ja W3C pyrkii yhdenmukaistamaan HTML5:n avulla JavaScriptin eri toiminnallisuuksia ja laajennuksia web-selaimissa. (Zakas, 2012.).

2.3 JavaScriptiin liittyvät käsitteet

JavaScript on monipuolinen ohjelmointikieli ja sen toimintaan liittyy useita eri käsitteitä. Käsitteiden ymmärtäminen auttaa ymmärtämään paremmin kielen käyttötarkoitusta ja sen toimintatapaa. Alaluvun tarkoitus on esitellä eri käsitteitä ja niiden yhteyttä JavaScriptiin.

2.3.1 Asiakaspää

JavaScript on *asiakaspään* (*Client-side*) ohjelmointikieli, mikä tarkoittaa, että sovellus ajetaan web-selaimessa. Asiakaspään ohjelmointikieltä kutsutaan myös *front end* -ohjelmointikieleksi. Asiakaspään ohjelmointikieli antaa välittömästi vastauksen käyttäjän toimiin ja muuttaa sivun ulkoasua ilman ylimääräistä sivun latausta. Asiakaspään ohjelmakoodi mahdollistaa esimerkiksi dynaamisen web-sivun päivittämisen, animaatiot, lomakkeiden tarkistukset, sisällön poistamisen tai liikuttamisen sekä muita vuorovaikutteisia toiminnallisuuksia (Swiech & Dinda, 2013). JavaScript ei ole ainoa asiakaspään ohjelmointikieli. JavaScriptin lisäksi on olemassa esimerkiksi Java-sovelma. (McFarland, 2011.; Wright, 2012.).

Asiakaspäässä ajettavat ohjelmat vähentävät palvelimen kuormitusta ja luovat paremman käyttäjäkokemuksen (Hales, 2012). Asiakaspäässä voidaan käyttää apuna myös *CDN*:ää (*Content Delivery Network*). CDN on tarkoitettu staattisen eli muuttumattoman sisällön varastoksi. Sen etu verrattuna tavallisiin palvelimiin ovat noin 20 % lyhyemmät latausajat, jotka perustuvat fyysisesti lähelle sijoitettuihin palvelimiin. Selaimen pyytäessä esimerkiksi JavaScript-

kirjastoa CDN tarjoaa kirjaston käyttäjää lähimpänä olevalta palvelimelta, joka voi parhaassa tapauksessa sijaita samassa maassa. (MacCaw, 2011.).

Halesin (2012) mukaan asiakaspään ohjelmointikieli tarjoaa seuraavat edut verrattuna palvelinpäähän:

- paremman käyttäjäkokemuksen
- verkon kaistaleveyden käytön tarve pienenee
- siirrettävyyden (offline-käyttö).

Asiakaspään ohjelmointikieleen liittyy myös monia tietoturvariskejä ja ongelmia. Mikkosen ja Taivalsaaren (2007) mukaan tietoturvaongelmat ovat yleisiä tällaisissa sovelluksissa, koska nykyään 100 suosituimmasta web-sivustosta maailmassa 97 käyttää JavaScriptiä. JavaScriptin tietoturvaongelmat voivat johtua sen heikosta tyypityksestä ja mahdollisuudesta luoda uutta ohjelmakoodia ajon aikana. Web-selaimet sallivat yleensä ajon aikana tulleet JavaScript-virheet eivätkä lopeta ohjelman suoritusta virheistä huolimatta. Selain jatkaa toimivan JavaScriptin suorittamista ja antaa viallisesta ohjelmakoodista virheen selaimen konsolityökaluun. Tämä toimintamalli johtaa hienovaraisiin ongelmiin, joita on vaikea löytää testauksen aikana. (Ocariza, Pattabiraman & Zorn, 2011.; Mikkonen & Taivalsaari, 2007.). Tutkimuksessa ei käsitellä tämän laajemmin JavaScriptin käytöstä aiheutuvia tietoturvaongelmia. Tietoturvan testaus ja siihen tutustuminen vaatii laajoja testitapauksia ja tutkimuksia, jotka eivät tässä tutkimuksessa ole mahdollisia rajallisten resurssien vuoksi.

2.3.2 Palvelinpää

Palvelinpään (*Server-side*) ohjelmointikielellä tarkoitetaan ohjelmointikieltä, joka ajetaan *palvelimella* (*server*). Palvelin on tietokone, joka jakaa tietoja muihin tietokoneisiin Internetin välityksellä (TechTerms.com, 2014a). Palvelinpään ohjelmointikieltä kutsutaan myös *back end* -ohjelmointikieleksi. Palvelinpään tehtävä on pitää yllä esimerkiksi tietokanta ja liiketoimintalogiikkaa eli palvelun älykkyyttä sekä luoda liittymä muihin järjestelmiin (Kovanen, 2013). Kovanen (2013) on listannut tyypillisimmiksi palvelinpään ohjelmointikieliksi Javan, Pythonin, PHP:n ja Ruby on Railsin. Nämä ohjelmointikielet käyttävät yleensä apunaan palvelimen tietokantoja ja voivat lähettää esimerkiksi sähköpostiviestejä ympäri maailmaa. JavaScriptiä voidaan hyödyntää nykyään myös palvelinpäässä Node.js:n avulla.

Halesin (2012) mukaan palvelinpään ohjelmointikielet tarjoavat seuraavat edut verrattuna asiakaspäähän:

- paremman tietoturvan
- laajennettavuuden (palvelinkoneita lisäämällä voidaan kasvattaa kuormitusta)
- tehokkaamman tehtävien käsittelyn.

Palvelinpään ohjelmointikielien yksi ongelma on, että selain joutuu lähettämään aina HTTP-pyynnön palvelimelle. Palvelin käsittelee pyynnön ja palauttaa vastauksen käyttäjän selaimelle. Tällainen toimintamalli pakottaa käyttäjän odottamaan aina latautuvan sivun tietoja. (McFarland, 2011.).

2.3.3 Komentosarjakieli

Komentosarjakieli (*Scripting language*) eli skriptikielen avulla kirjoitetaan komentosarjoja. Komentosarjakieli tarkoittaa kaikkia kieliä, joiden avulla voidaan suorittaa käyttöjärjestelmällä erityyppisiä toimintoja ilman ohjelman kääntämistä. JavaScript luokitellaan asiakaspuolen komentosarjakieliin. Kieltä käytetään muokkaamaan, käsittelemään ja automatisoimaan toimivan järjestelmän tarjoamia palveluja (Peltomäki & Nykänen, 2006). JavaScript-sovellus suoritetaan aina selaimessa tai muussa JavaScript-tulkin sisältävässä sovelluksessa (Peltomäki & Nykänen, 2006). JavaScriptillä tehty ohjelma ei ole yleensä itsenäinen sovellus, eikä sillä voida luoda uutta järjestelmää (Mäkelä, 2010). Morin ja Brown (1999) ovat todenneet, että useimmat ihmiset luokittelevat tyypillisimmiksi komentosarjakieliksi seuraavat kielet: AppleScript, Unix-komentokielet, Visual Basic, JavaScript, Perl ja Tcl/Tk.

2.3.4 Tyyppijärjestelmä

Tyyppijärjestelmän (*Type system*) avulla määritetään, kuinka ohjelmointikieli luokittelee arvot ja muuttujat tyyppeihin. Ohjelmointikielet voidaan luokitella kahteen tyyppijärjestelmään: *staattiseen* tai *dynaamiseen*. JavaScript kuuluu dynaamisesti tyypitettyihin ohjelmointikieliin. Tyyppijärjestelmän ollessa staattinen ohjelma suorittaa tyyppien semanttisen tarkistuksen ennen ajoa, joten objektien tyypit ovat tiedossa jo käännösvaiheessa. (Meijer & Drayton, 2004.). Dynaamisesti tyypitetyssä ohjelmakoodissa objektien tyypit voivat muuttua ajon aikana. Esimerkiksi JavaScriptissä ei koskaan määritellä muuttujalle tyyppiä, kuten C:ssä tehdään. Tässä tapauksessa muuttujaan voi sijoittaa millaista dataa tahansa ja tietotyyppi voi vaihtua ajon aikana. *Vahvan staattisen* tyypityksen etuna on, että ohjelmointikielen kääntäjä voi tunnistaa ohjelmointivirheet jo käännösaikana. Liian tarkat tyyppisäännöt voivat kuitenkin laskea ohjelmistokehittäjän tehokkuutta, motivaatiota ja keskittymistä pitkällä aikavälillä. *Dynaaminen tyypitys* tarjoaa joustavuutta, mutta siinä tyyppivirheet havaitaan vasta ajon aikana. (Loui, 2008.). Tyyppijärjestelmää valittaessa Meijer ja Drayton (2004) antavat ohjeeksi käyttää staattista tyypitystä silloin, kun se on mahdollista ja dynaamista silloin, kun se on tarpeen.

2.3.5 HTML

HTML (*Hypertext Markup Language*) on W3C:n ylläpitämä hypertekstidokumenttien merkintäkieli, jonka avulla web-sivuilla kuvataan sisällön rakennetta. HTML:ää voidaan käyttää esimerkiksi tekstin tai mediatiedostojen, kuten kuvien tai äänen esittämisessä web-sivuilla. (Kasten, 1995.). HTML-dokumentteihin liitetyt JavaScript-tiedostot tarjoavat tärkeän tavan luoda dynaamista vuorovaikutusta käyttäjän kanssa (Peltomäki & Nykänen, 2006). HTML tunnetaan erityisesti kielenä, jonka avulla web-sivut on ohjelmoitu (Kasten, 1995).

HTML-tiedostot muodostuvat sisäkkäisistä ja perättäisistä elementeistä. *Aloitustunnisteita* kutsutaan myös *tageiksi* (*tag*). Elementit muodostuvat pareista, joissa on erikseen aloitustunniste ja vinoviivalla merkitty lopetustunniste sekä niiden väliin jäävä sisältö. Elementeillä voidaan esimerkiksi merkitä sivun otsikko kirjoittamalla HTML-tiedostoon <title>Hello World</title>. Aloitustunnisteeseen voi lisäksi sisältyä attribuutteja, jotka määrittävät elementin ominaisuuksia tarkemmin kuin tunnisteen nimeämä elementti. Esimerkiksi on kuva-elementti, jolla on *attribuutti* src, jonka arvoksi asetetaan näytettävän kuvan osoite. (Bouvier, 1995.).

HTML:stä on kehitetty myös *XHTML* (*Extensible Hypertext Markup Language*), joka eroaa HTML:stä tiukemmilla muotosäännöillään. Usein kuulee puhuttavan *HTML5*:stä, joka on jatkuvasti muuttuva luonnos HTML-kielen uudeksi määrittelyksi ja sen osittaisia toteutuksia. HTML5 sisältää paljon uudistuksia ja sillä viitataan nykyisin moderneihin web-tekniikoihin. (Korpela, 2011.).

2.4 JavaScriptin toimintaympäristö

Alaluvun tarkoitus on esitellä JavaScriptin toimintaympäristöä ja sen tuomia mahdollisuuksia ja haasteita. Luvussa käsitellään *selainmoottori*, *web-selain* ja *käyttöliittymä*, joita ilman JavaScriptin käyttäminen web-sivulla ei ole mahdollista. Selainmoottorin tarkoitus on toimia web-selaimen ”konehuoneena”, jota ilman web-selain ei pysty käsittelemään tietoa. Web-selaimen tarkoitus on esittää käyttäjälle web-sivuja eli sovelluksia tai tietoa Internetin yli. Web-selaimessa esitetyt tiedot tai sivut sisältävät käyttöliittymän, jonka avulla käyttäjä voi liikkua web-sivuilla.

2.4.1 Selainmoottori

Selainmoottori (*Rendering engine*) on ohjelmistokomponentti, jonka tehtävä on tulkita, jäsentää ja toteuttaa, miten esimerkiksi HTML:n sisältö esitetään käyttäjälle näkyväksi web-sivuksi. Selainmoottori on tyypillisesti upotettu web-selaimeen, sähköpostiohjelmiin tai vastaaviin sovelluksiin, joissa näytetään käyttäjälle HTML-muotoisia viestejä. Selainmoottori käyttää elementeille oletusesitystapaa, joka voi olla eri selainmoottoreissa erilainen. Selainten väliset

erot JavaScriptin esittämisessä riippuvat ennen kaikkea siitä, millainen selainmoottori niissä on. (Reis & Gribble, 2009.; Korpela, 2011.).

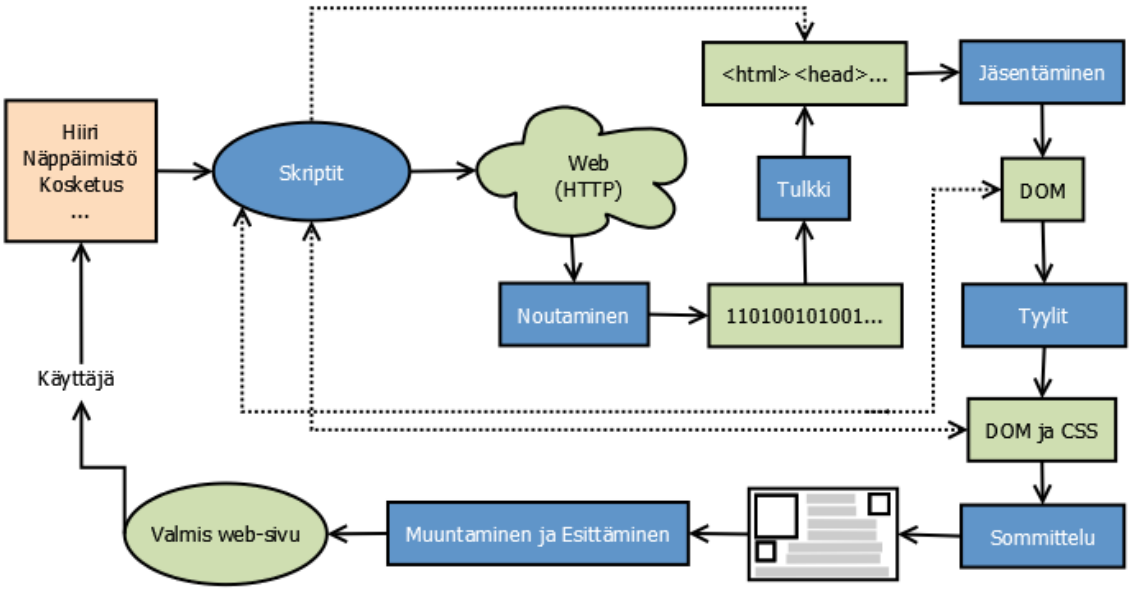
Selainmoottoreita on olemassa useita erilaisia. Taulukossa 1 on kuvattu eri selainten käyttämiä selainmoottoreita. Taulukosta 1 voidaan todeta, että kaikki selaimet eivät käytä eri selainmoottoreita. Esimerkiksi Google Chrome ja Apple Safari käyttävät samaa selainmoottoria (Webkit) ja siksi näiden selainten välillä on hyvin vähän yksityiskohtaisia eroja. Selainmoottorin lisäksi web-selaimissa käytetään *JavaScript-moottoria (JavaScript engine*). Selainmoottorista poiketen JavaScript-moottori vaikuttaa ainoastaan JavaScriptin toimivuuteen ja nopeuteen web-selaimessa. (Wright, 2012.).

TAULUKKO 1 Selaimet ja niiden selainmoottorit (Wright, 2012, 29)

Selain	Selainmoottori
Mozilla Firefox	Gecko
Microsoft Internet Explorer	Trident
Google Chrome	Webkit
Apple Safari	Webkit
iOS & Android	Webkit
Opera	Presto

2.4.2 Web-selain

Web-selain (*Web browser*) tai yksinkertaisemmin ”selain” on ohjelma, jonka avulla voidaan käyttää ja katsella web-sivuja. Yleisimmät selaimet ovat Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, Opera, iOS, Android ja Apple Safari. Selainten pääasiallinen tehtävä on muuntaa HTML-koodi käyttäjälle visuaaliseen muotoon. Aikaisemmat selaimet, kuten Mosaic ja Netscape Navigator, olivat yksinkertaisia sovelluksia, jotka käsittelivät ainoastaan lomakkeiden kenttiä sekä tuettuja kirjanmerkkejä. Nykyaikaiset web-selaimet mahdollistavat vuorovaikutteisen sisällön esittämisen web-sivulla esimerkiksi JavaScriptin ja AJAXin (ks. alaluku 2.6.1) avulla. Selaimissa on tästä huolimatta paljon yhteensopivuusongelmia. Sama sovellus ei välttämättä toimi kaikilla selaimilla oikein. Tämä johtuu siitä, että selaimet käyttävät erilaisia selainmoottoreita muuntaessaan sivuja. Kehittäjien onkin hyvä testata JavaScriptin toiminnallisuuksia useissa eri selaimissa. (TechTerms.com, 2014b.). Kuviossa 2 esitellään tyypilliset selaimen käsittelyvaiheet. Kuviossa 2 käyttäjä aktivoi web-sivulla olevan toiminnallisuuden esimerkiksi näppäimistön avulla. Tämän jälkeen palvelimelta noudetaan binäärikoodia HTTP:n avulla. Lopuksi binäärikoodi tulkataan, jäsennetään, siihen lisätään tyylejä, sommitellaan, muunnetaan ja esitetään käyttäjälle haluttuna toimintona tai web-sivuna.



KUVIO 2 Tyypilliset selaimen käsittelyvaiheet (Cascaval ym., 2013, 1, ulkoasua muutettu)

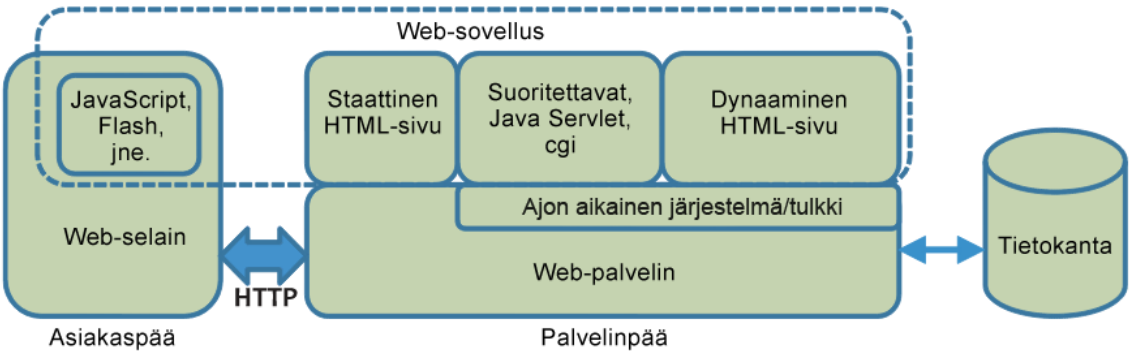
2.4.3 Käyttöliittymä

Käyttöliittymä (*User Interface, UI*) tarkoittaa laitteen, ohjelmiston tai minkä tahansa tuotteen osaa, jonka avulla käyttäjä voi käyttää sovellusta tai laitetta. Hyvä käyttöliittymä tarjoaa käyttäjäystävällisen kokemuksen, jonka avulla käyttäjä voi olla vuorovaikutuksessa ohjelmiston tai laitteiston kanssa helposti ja intuitiivisesti. Suuressa osassa ohjelmistoja on *graafinen käyttöliittymä* (*GUI*). Tämä tarkoittaa sitä, että ohjelma tai laite sisältää graafisia toimintoja, joita voidaan käyttää hiiren tai näppäimistön avulla. Graafinen käyttöliittymä sisältää tyypillisesti valikkorivin, työkalurivin, ikkunoita, painikkeita ja muita tarpeellisia lisäosia. Vaikka Windows- ja Macintosh-käyttöjärjestelmät ovat eri käyttöjärjestelmiä, ne jakavat paljon samoja elementtejä, kuten työpöydän, ikkunat, ikonit ja niin edelleen. Yhteiset elementit mahdollistavat käyttäjien käyttää erilaisia käyttöjärjestelmiä ilman, että niitä täytyy opetella kokonaan uudelleen. Asiaa voi verrata esimerkiksi eri autonvalmistajien autoihin – jokaisessa autossa on ratti, jonka avulla autoa voidaan ohjata. (TechTerms.com, 2009b.).

2.5 JavaScriptin käyttö

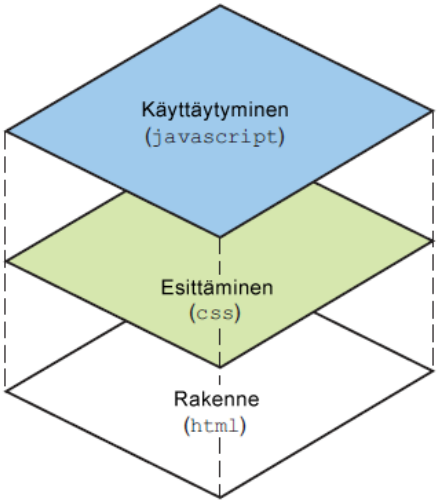
JavaScript ei ole kuten palvelinpään ohjelmointikielet, joita voidaan kontrolloida palvelimelle asennetun ympäristön avulla. Esimerkiksi Javalla toteutettu sovellus toimii aina samalla tavalla jokaisella käyttäjällä, kunhan palvelimelle on asennettu toimiva versio esimerkiksi Apache Tomcatista. Sama periaate toimii muissa palvelinpään ohjelmointikielissä, kuten Pythonissa. JavaScript ei kuitenkaan toimi samalla tavalla, koska se on asiakaspäässä ajettava ohjelmoin-

tikieli. JavaScript suoritetaan suoraan käyttäjän web-selaimessa, joten siitä ei tarvitse kääntää erillistä suoritettavaa tiedostoa. (Wright, 2012.). Asiakaspäässä ajettavan ohjelman tulee pystyä keskustelemaan palvelinpään kanssa. Otetaan esimerkiksi kuvagalleria, jossa käyttäjä pystyy arvostelemaan kuvia. Käyttäjä voi arvostella kuvan painamalla painiketta tai valitsemalla listasta haluamansa arvosanan. Tässä tapauksessa arvio pitää pystyä lähettämään HTTP:n avulla palvelinpäälle, jossa tiedon käsittelevä metodi tallentaa sen tietokantaan. Tietokannassa olevista arvosanoista voidaan nyt laskea esimerkiksi keskiarvo ja hakea se HTTP:n avulla takaisin asiakaspäälle. Kuvio 3 kuvaa tiedonsiirtoa HTTP:n yli asiakas- ja palvelinpään välillä. Kuviossa 3 web-selain hakee tarvittavat tiedot web-palvelimelta, joka pitää yllä tietokantaa ja liiketoimintalogiikkaa.



KUVIO 3 Tiedonsiirto asiakas- ja palvelinpään välillä (Li & Xue, 2014, 3)

Asiakaspään ohjelmointikieliä käyttäessä web-sovellus tulee rakentaa mahdollisimman *progressiiviseksi* (*progressive enhancement*). Yksinkertaisimmillaan progressiivinen web-kehitys tarkoittaa sovelluksen suunnittelua ja kehitystä kolmessa pääkerroksessa. Pääkerrokset ovat HTML, CSS ja JavaScript. Toimintamallin tarkoitus on tarjota käyttäjälle mahdollisimman laadukas käyttäjäkokemus, sekä varmistaa sisällön saatavuus. Progressiivinen ajattelumalli mahdollistaa sen, että sisältö ei ole sidottu teknologiaan tai esitysmuoto sisältöön. Samaa menetelmää voidaan soveltaa myös muihin kuin web-sovelluksiin. Mielenkiintoinen asia näissä kolmessa pääkerroksessa on se, että ne on käytännössä erotettu toisistaan, mutta ylempi kerros on riippuvainen edellisestä kerroksesta. Kerroksia voidaan aina poistaa ylhäältä alaspäin menettämättä web-sivun tärkeintä asiaa, sisältöä. Kuviossa 4 on esitelty kolme pääkerrosta ylhäältä alaspäin. Ylimmässä kerroksessa (*käyttäytyminen*) voidaan käyttää apuna JavaScriptiä, jonka avulla voidaan lisätä dynaamista toiminnallisuutta web-sivulle. Toinen kerros (*esittäminen*) vastaa web-sivulle lisätyistä tyyliasetuksista eli siitä, kuinka web-sivu esitetään käyttäjän selaimessa. Kolmas ja alin kerros (*rakenne*) on web-sivun tärkein kerros. Kolmannessa kerroksessa voidaan esittää käyttäjälle sisältöä, kuten kuvia tai linkkejä. (Wright, 2012.).



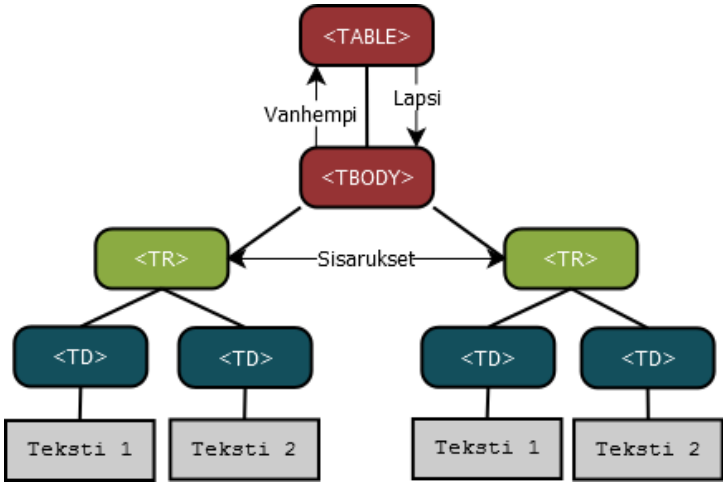
KUVIO 4 Progressiivinen web-suunnittelu (Wright, 2012, 4)

2.5.1 Sovelluksen elinkaari

Asiakaspään sovelluksen elinkaari voidaan jakaa kahteen vaiheeseen: *web-sivun alustukseen* ja *tapahtumankäsittelijöihin*. Ensimmäisessä vaiheessa, web-sivun alustuksessa, web-selaimen on tarkoitus rakentaa sivun käyttöliittymä käyttäjälle visuaalisempaan muotoon. Web-sivun rakentaminen aloitetaan jäsentämällä HTML-koodista hierarkkinen dokumenttioliomalli eli dokumenttipuu. Tarkastellaan seuraavaksi kuviota 5, joka on yksinkertainen esimerkki HTML-dokumentin taulukko-elementistä. Kuviossa 5 *table* on dokumentin runko ja sillä on yksi lapsi, *tbody*. Kaksi *tr-tagia* ovat tbodyn lapsia, jotka ovat sisaruksia keskenään. Kuviossa 5 viimeisenä on *td-tagit* eli lehdet. DOM jäsentää tällaista HTML-dokumenttia solmu kerrallaan. Jos web-selain saavuttaa JavaScriptiä sisältävän HTML-solmukohdan, niin DOMin rakentaminen keskeytetään ja siirrytään suorittamaan JavaScriptiä. Tämän jälkeen DOMin rakennusta jatketaan viimeiseen HTML-solmuun asti, jolloin käyttöliittymän rakennus on valmis. Lopuksi ohjelman suoritus siirretään tapahtumankäsittelijälle, jonka ohjelmakoodi suoritetaan vastauksena tiettyyn tapahtumaan. Tapahtumia voi olla useita erilaisia, kuten hiiren vieminen kuvan päälle tai painikkeen painaminen. (Maras, Carlson & Crnkovic, 2011.). Kuvio 6 on graafinen esitys dokumenttipuusta eli DOMista. Kuvio 6 on tehty kuvion 5 pohjalta.

```
<table>
    <tbody>
        <tr><td>Teksti 1</td><td>Teksti 2</td></tr>
        <tr><td>Teksti 1</td><td>Teksti 2</td></tr>
  </tbody>
</table>
```

KUVIO 5 HTML-dokumentin taulukko-elementti



KUVIO 6 Graafinen esitys dokumenttipuusta (DOM)

2.5.2 JavaScript-kielen olio-ominaisuudet

JavaScript on *olio-ohjelmointikieli* (*Object-oriented programming language*). Olio-ohjelmointi sopii erityisen hyvin graafisten käyttöliittymien ohjelmointiin. Suosituimmat graafisten käyttöliittymien ohjelmoinnin välineet tukevatkin luonnollisesti olio-ohjelmointia. Olio-ohjelmoinnin tarkoitus on rakentaa sovellus itsenäisistä komponenteista, joilla on tietyt ominaisuudet ja toiminnot, joita ne osaavat suorittaa. Oliota voidaan ajatella reaalimaailman esineinä, esimerkiksi kaikki näkemämme esineet vastaavat olio-ohjelmoinnissa olioita. Sovelluskehityksessä yritetään käytännössä mallintaa reaalimaailman ilmiöitä tietokoneen avulla. (Kosonen, Peltomäki & Silander, 2005.).

JavaScriptiä kirjoitettaessa voidaan käyttää valmiita olioita, luoda omia olioita tai kirjoittaa vaihtoehtoisesti funktioita. JavaScriptistä puuttuu kuitenkin kokonaan tuki luokkien käytölle, koska JavaScript on *prototyyppipohjainen* olio-ohjelmointikieli. Luokkien sijasta esimerkiksi periytyminen on toteutettu perimällä ominaisuuksia toisista olioista, prototyyppiolioista. JavaScriptin dynaamisuudesta johtuen useiden olioiden ominaisuuksia voidaan muuttaa samalla kertaa. JavaScriptissä luodaan omia olioita luomalla funktioita. JavaScriptissä luotavat yleiskäyttöiset oliot ovat kuitenkin huomattavasti rajoittuneempia kuin esimerkiksi Javassa. (Peltomäki & Nykänen, 2006.).

Kuvio 7 on esimerkki JavaScriptillä toteutetusta perintämallista. Kuviossa oleva *kulkuneuvo* perii sekä *Auto.prototype*- että *Vene.prototype-olion*. Tässä tapauksessa *kulkuneuvo* pääsee käsiksi *Veneessä* määriteltyyn funktioon *method* sekä ominaisuuteen *arvo*, jonka luotu *Vene*-olio sisältää prototyypissään. *New Auto()* ei luo uutta *Vene*-oliota vaan käyttää uudelleen sen prototyyppiin asetettua arvoa. Tässä tapauksessa kaikki *Auto*-oliot jakavat saman *arvo*-ominaisuuden.

```
function Vene() {
    this.arvo = 1;
};

Vene.prototype = {
    tulosta: function(teksti) {return teksti + " " + this.arvo; }
};

function Auto() {};

// Asettaa Vene-olion Auton prototyyppiin
Auto.prototype = new Vene();
Auto.prototype.auto = 'Olen auto';

// Huolehtii, että Auto on todellinen konstruktori
Auto.prototype.constructor = Auto;

var kulkuneuvo = new Auto(); // Luodaan uusi auto

console.log(kulkuneuvo.arvo); // Tulostaa 1
console.log(kulkuneuvo.auto); // Tulostaa "Olen auto"
console.log(kulkuneuvo.tulosta("Veneen arvo on: ")); // Tulostaa "Veneen arvo on: 1"
```

KUVIO 7 JavaScriptin perintämalli

2.5.3 JavaScriptin lisääminen web-sivuille

Web-sivuilla olevaan HTML-dokumenttiin voidaan liittää JavaScript-koodia useilla eri tavoilla. Peltomäki ja Nykänen (2006) ovat kuvanneet näistä kolme erilaista tapaa:

- JavaScriptiä voidaan lukea ulkopuolisesta tiedostosta käyttämällä *script-elementin src-ominaisuutta*. Src-määritteessä on viittaus ulkoiseen, eri tiedostossa olevaan skriptiin <script src=”kirjautuminen.js”></script>.
- JavaScriptiä voidaan sijoittaa script-elementin sisään. Funktiot ja itse määritetyt JavaScript-olioiden prototyypit sijoitetaan yleensä head-elementin sisään. Tämän toimintamallin ansiosta esimerkiksi funktiot ovat muistissa ennen dokumentin rungon lataamista. Script-elementissä ei ole src-määritettä vaan itse elementin sisältönä on suoraan suoritettava skripti <script> alert(”Esimerkki”); </script>.
- JavaScriptiä voidaan sijoittaa HTML-elementin ominaisuudeksi esimerkiksi antamalla sille tapahtumamääritteessä onclick- tai onmouseover-tapahtuma. Yleensä näihin tapahtumamääritteisiin sijoitetaan vain head-elementissä määritellyn funktionkutsu <p onclick=”tallenna();”>Tämä on esimerkkiteksti<p>.

Edellä mainituista tavoista kannattaa käyttää ensisijaisesti ensimmäistä. Silloin ei synny ongelmia esimerkiksi siitä, miten skriptissä kirjoitetaan merkit ”<” ja ”&”, joilla on HTML-merkkauksessa erikoisasema. Ulkoinen tiedosto on HTML-koodista täysin erillinen, joten kyseiset merkit voidaan kirjoittaa siihen

sellaisinaan. Mikään edellä mainituista esimerkeistä ei kuitenkaan toimi, jos selain ei tue JavaScriptiä. Jokainen JavaScriptiä tukematon selain hylkää tässä tapauksessa script-elementin, mutta tulostaa JavaScript-koodin ASCII-tekstinä, jos script-elementtiä ei piiloteta HTML-kielen kommentteihin. HTML-määritteiden sijasta voidaan käyttää myös JavaScriptissä määritettyjä funktiokutsuja. JavaScriptissä tehtävissä funktiokutsuissa voidaan HTML-koodista jättää tapahtumamäärite pois ja hakea haluttu toiminto JavaScriptillä HTML-tiedostossa määritetyn ID-elementin perusteella. (Korpela, 2011.; Peltomäki & Nykänen, 2006.).

2.5.4 Käyttökohteet

JavaScriptin avulla voidaan saavuttaa monia hyötyjä web-kehityksessä, koska se on todella dynaaminen ohjelmointikieli. JavaScriptillä voidaan esimerkiksi näyttää ponnahdusikkunoita, lisätä erilaisia *attribuutteja* elementteihin, luoda, muokata tai poistaa HTML:ää, seurata hiiren liikkeitä tai toteuttaa selaimessa toimivia pelejä. JavaScriptin ominaisuudet ovat tuettuja myös käyttäjän selaimen ollessa offline-tilassa. Lisäksi JavaScript voi kommunikoida palvelimen kanssa monilla eri tavoilla. Tarkoituksena on aina parantaa käyttäjäkokemusta. Yksi tapa parantaa käyttäjäkokemusta sovelluksissa on käyttää AJAX-kutsuja. AJAX (ks. alaluku 2.6.1) säästää käyttäjän aikaa ja hermoja ilman ylimääräisiä sivun latauksia. Menetelmän avulla sivulta voidaan päivittää haluttu määrä tietoa kerrallaan, mikä nopeuttaa sovelluksen toimintaa. (Wright, 2012.).

2.6 Web-sovelluksen tiedonsiirtotavat

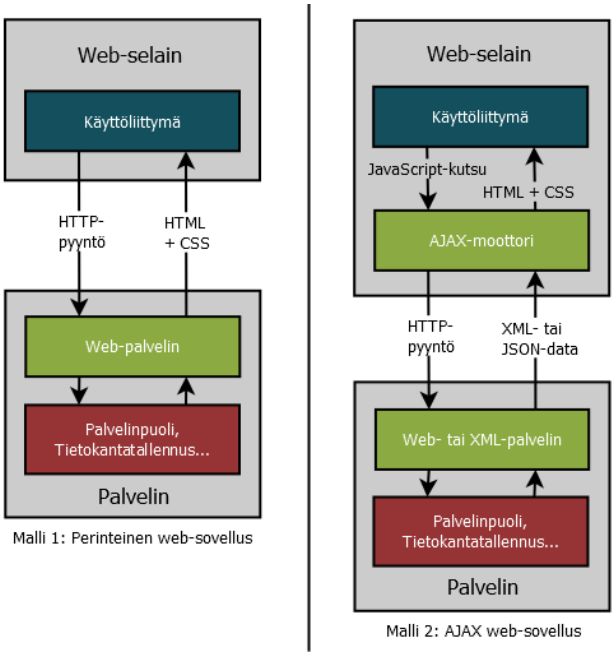
Web-sovelluksen tiedonsiirrossa voidaan käyttää useita erilaisia tekniikoita. Tekniikat on esitetty tässä alaluvussa yhdessä laajemman käsitteen yhteydessä, josta käytetään nimeä *web-palvelu*. W3C (2004) määrittelee web-palvelun (*Web service*) ohjelmistojärjestelmäksi, joka mahdollistaa keskenään yhteensopivien tietokoneiden välisen vuorovaikutuksen tietoverkon yli. Web-palvelua ei tule sekoittaa web-sivuihin, jotka tarjoavat käyttäjille palveluita. Web-palvelulla tarkoitetaan verkossa olevan palvelun ohjelmointirajapintaa. Palvelinpää tarjoaa toimiville web-sovelluksille palvelun HTTP:n tai muun web-pohjaisen protokollan yli. (W3C, 2004.). Alkuperäisen web-palvelutyöryhmän jäsen ja XML-merkintäkielen määrittelijä Adam Bosworth (2003) on kuvaillut web-palvelun olevan mikä tahansa ohjelmistojen välisen kommunikaation mahdollistava arkkitehtuuri. Bosworth (2003) tarkentaa vielä määritelmää seuraavasti: ohjelmistossa tulee olla myös tuki alustariippumattomalle formaatille ja protokollalle, joiden avulla siirrettävä data esitetään ja ohjelmistojen välinen vuorovaikutus tapahtuu.

Alaluvun 2.6 alakohdissa on esitetty tärkeimpiä web-palvelun yhteydessä käytettäviä termejä ja niiden rooleja teknisestä näkökulmasta. AJAX, JSON,

REST ja HTTP eivät kuulu *web-palvelu -protokollakentän* kolmeen XML-pohjaiseen komponenttiin, jotka ovat SOAP, WSDL ja UDDI. AJAX, JSON, REST ja HTTP katsotaan osaksi web-palvelun käsitettä, koska ne mahdollistavat ohjelmistojen välisen kommunikaation. Lisäksi ne ovat käytössä tässä tutkimuksessa toteutettavissa web-sovelluksissa, joten niiden ymmärtäminen on sovellusten toteuttamisen kannalta olennaista.

2.6.1 AJAX

AJAX (*Asynchronous JavaScript And XML*) on tekniikka, joka käyttää useita eri menetelmiä lähettämään ja palauttamaan dataa selaimen ja palvelimen välillä. AJAXin on keksinyt Jesse James Garrett. Tekniikan avulla web-sovelluksista voidaan tehdä vuorovaikutteisempia, nopeampia ja käyttäjäystävällisempiä. Monet tunnetut web-sivut, kuten Google Maps, Gmail, Google Suggest ja Flickr, käyttävät AJAX-kutsuja. Perinteinen web-sovellus lähettää datan palvelimelle, joka prosessoinnin jälkeen palauttaa uuden web-sivun käyttäjälle. Tällainen toimintamalli tekee web-sovelluksesta hitaan käyttää. AJAX-tekniikkaa käyttävät sovellukset voivat lähettää ja vastaanottaa dataa asynkronisesti lataamatta web-sivua uudelleen. Tämä tapahtuu lähettämällä HTTP-pyyntö palvelimelle ja muokkaamalla JavaScriptin avulla vain osaa web-sivusta käyttäjältä piilossa. (Peltomäki & Nykänen, 2006.; Lin, Wu, Zhang & Zhou, 2008.). Kuviossa 8 perinteinen web-sovelluksen toiminta on esitetty vasemmalla puolella ja oikealla puolella on AJAXin avulla toteutettu web-sovellus. AJAX-sovellus tekee aluksi JavaScript-kutsun AJAX-moottorille, joka suorittaa HTTP-pyynnön palvelimelle. Palvelin palauttaa datan AJAX-moottorille XML- tai JSON-muodossa. Data voidaan tämän jälkeen käsitellä ja esittää käyttäjälle halutulla tavalla.



KUVIO 8 Perinteinen web-sovellus vs. Ajax-sovellus (Lin, Wu, Zhang & Zhou, 2008, 1)

AJAX on tekniikka, joka käyttää useita eri teknologioita. Lin ym. (2008) ovat listanneet teknologiat, jotka AJAX sisältää:

- (X)HTML ja CSS
- DOM
- XML ja XSLT
- XMLHttpRequest
- JavaScript.

Peltomäki ja Nykänen (2006) ovat kertoneet, että AJAX-tekniikan keskeisimpänä ja tärkeimpänä asiana voidaan pitää XMLHttpRequest-oliota. Kyseinen olio on ensimmäinen, joka JavaScriptillä täytyy luoda (jos käytössä ei ole JavaScript-kirjastoja), jotta AJAX-tekniikkaa voidaan käyttää. XMLHttpRequest ei ole W3C:n standardi, joten se luodaan eri selaimissa eri tavalla. Internet Explorer -selaimessa luodaan ActiveX-komponentti ja muissa selaimissa, kuten Mozilla Firefox tai Safarissa, luodaan natiivi JavaScript-olio. Luomisen jälkeen XMLHttpRequest-oliota voidaan käyttää kaikissa selaimissa perusteiltaan täysin samalla tavalla. (Peltomäki & Nykänen, 2006.). Kuviossa 9 on esimerkki XMLHttpRequest-olion luonnista.

```
function luoXMLHttpRequest() {
    // Luo HTTP-pyyntö serverille
    if (window.ActiveXObject) { // IE 5+
        httpPyynto = new ActiveXObject("Microsoft.XMLHTTP");
    } else if (window.XMLHttpRequest) { // Muut: Mozilla, Opera jne.
        httpPyynto = new XMLHttpRequest();
    }
}
```

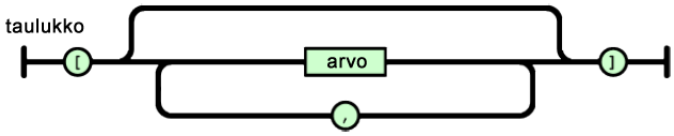
KUVIO 9 XMLHttpRequest-olio (Peltomäki & Nykänen, 2006, 299)

2.6.2 JSON

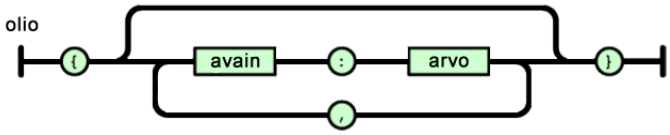
JSON (*JavaScript Object Notation*) on yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen. JSONia käytetään laajasti AJAX-sovellusten yhteydessä, koska se mahdollistaa helpon tiedonsiirron palvelimen ja asiakaspään välillä. Brendan Eich keksi JSONin kehittäessään JavaScriptiä, mutta vasta Douglas Crockford löysi ja keksi sen nykyisen käyttötarkoituksen vuonna 2001. Aikaisemmin XML oli hallitseva tiedostomuoto tiedonvälitykseen, mutta JSONista tuli nopeasti potentiaalinen vaihtoehto XML:n tilalle. Lisäksi JSON on standardoitu tiedostomuoto, jonka on standardoinut ECMA-262. (Severance, 2012b.).

JSON on määritelty ECMAScriptin standardissa, mutta se on täysin riippumaton käytetystä ohjelmointikielestä, mikä tekee siitä suositun tiedonsiirtomuodon. JSON on tekstimuotoista dataa, joka käyttää kahta erilaista rakennetta dataobjektin esittämisessä: taulukkotietorakennetta (kuvio 10) ja oliota (kuvio 11). Taulukkotietorakenne on järjestetty kokoelma, joka voi sisältää arvoja tai

olioita. Taulukkotietorakenne alkaa ja päättyy hakasulkeilla. Hakasulkeiden välissä olevat alkiot erotetaan toisistaan pilkulla. Oliolla tarkoitetaan avain-arvoparia, joka esitetään aaltosulkeiden välissä. Avain-arvopari erotetaan kaksoispisteen avulla ja kaikki avain-arvopari-yhdistelmät erotellaan toisistaan pilkun avulla. (Lin, Chen, Chen & Yu, 2012.).



KUVIO 10 JSON-taulukkotietorakenne (Lin, Chen, Chen & Yu, 2012, 1175)



KUVIO 11 JSON-olio (Lin, Chen, Chen & Yu, 2012, 1175)

JSON on kevyt ja tehokas käyttää verrattuna esimerkiksi XML:ään (Severance, 2012b). Lin ym. (2012) ovat vertailleet JSONin ja XML:n suorituskykyä tutkimuksessaan. Tutkimuksessa on perustettu testiympäristö, jonka avulla kummallakin teknologialla on pystytty lähettämään dataa ja mittaamaan datan siirrossa kulunutta aikaa. Tutkimustulokset ovat keskiarvoja, jotta kokeelliset tiedot ovat mahdollisimman tarkkoja. Taulukossa 2 on vertailtu tiedonsiirtonopeuksia XML:n ja JSONin välillä vaihtelevilla datamäärillä. Datamäärän yksikköä ei ole kerrottu Linin ym. (2012) tutkimuksessa. Testitulokset osoittavat, että JSON on tehokkaampi vaihtoehto tiedostonsiirrossa kuin XML. Tuloksista on hyvä huomioida, että datamäärän kasvaessa tiedonsiirtoeroista tulee yhä merkittävämmät. (Lin, Chen, Chen & Yu, 2012.).

TAULUKKO 2 Tiedonsiirtonopeuden vertailu XML ja JSON (Lin, Chen, Chen & Yu, 2012, 1175)

Datamäärä	Kulunut aika (ms)	
	XML	JSON
100	23,2	16,1
200	24,5	16,7
500	40,1	27,6
800	44,4	33,1
1000	69,4	47,3
2000	120,2	78,5

2.6.3 HTTP

HTTP (*Hypertext Transfer Protocol*) on sovellustason protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon (Fielding ym., 1999). HTTP-protokolla perustuu siihen, että asiakaspää avaa TCP/IP-yhteyden palvelimelle tiettyyn porttiin. HTTP-palvelin kuuntelee tätä porttia ja odottaa, että asiakaspää on lähettänyt pyyntönsä palvelimelle. HTTP-pyynnöt ovat muotoa ”lähetä minulle tiedosto /index.html”. Palvelin vastaa pyyntöön lähettämällä sopivan vastauksen, joka voi olla esimerkiksi HTML-sivu tai binääridataa, kuten kuvia, ohjelmia tai ääntä. HTTP:tä voidaan kutsua yhden yrityksen pyyntö-vastaus-malliksi (Yu, Chander, Inamura & Serikov, 2008). HTTP mahdollistaa myös muita toimintoja, kuten selaimen ohjauksen uudelle sivulle. (Ortiz, 2010.).

HTTP on ollut käytössä *WWW* (*World Wide Web*) aloitteen ansiosta vuodesta 1990 alkaen. Ensimmäinen versio HTTP:stä oli HTTP 0.9, joka oli yksinkertainen protokolla *raakadatan* (*Raw data*) lähetykseen Internetin kautta. Seuraava protokolla HTTP 1.0 paransi datan lähetystä sallimalla *MIME-tyypit* (*Multipurpose Internet Mail Extensions*) viesteissä. MIME-tyyppi kertoo sisällön tiedostomuodon, joka voi olla esimerkiksi text/html:ää eli tässä tapauksessa HTML:ää. MIME-tyyppi tunnetaan paremmin nimellä *Content-type* tai *Media-type* (Allamaraju, 2010). HTTP 1.0 -protokolla keskittyy vain itse pyyntöön, eikä ota huomioon esimerkiksi *välimuistia* (*Cache-Control*) tai *Host-saraketta*. HTTP 1.1 -protokolla on uusin protokolla, joka sisältää tiukempia vaatimuksia edelliseen protollaan verrattuna. HTTP 1.1 -protokollassa on määritettävä mm. Host-sarake, joka kertoo palvelimen tiedot, johon selain uskoo ottavansa yhteyttä. HTTP:stä on olemassa myös *HTTPS*-versio (*Hypertext Transfer Protocol Secure*), joka on HTTP-protokollan ja SSL/TLS-protokollan yhdistelmä. HTTPS:ää käytetään tiedon suojattuun siirtoon Internetissä. (Fielding ym., 1999.).

HTTP määrittelee kahdeksan erilaista standardoitua metodia resurssien käsittelyyn. Allamaraju (2010) on listannut HTTP:ssä olevat metodit ja niiden käyttötarkoitukset seuraavasti:

- *OPTIONS-metodi* palauttaa resurssiin tai palvelimen kykyihin liittyvät ominaisuudet.
- *GET-metodilla* voidaan hakea mitä tahansa tietoa, joka on tunnistettu selaimen pyytämän URI:n avulla.
- *HEAD-metodi* on muuten sama kuin GET, mutta HEAD palauttaa ainoastaan sivun ylätunnistetta koskevat tiedot.
- *POST-metodia* käytetään uusien resurssien tai aliresurssien luomiseksi palvelimelle.
- *PUT-metodin* avulla voidaan muuttaa tai päivittää tietosisältö.
- *DELETE-metodin* avulla voidaan poistaa tietosisältöä palvelimelta.
- *TRACE-metodi* palauttaa asiakkaan pyynnön sisällön sellaisenaan takaisin palvelimelta asiakkaalle. Voidaan käyttää apuna esimerkiksi järjestelmän virheentunnistamisessa.

- *CONNECT-metodia* käytetään sellaisten välityspalvelimien kanssa, jotka voivat dynaamisesti vaihtua tunneleiksi. Mahdollistaa HTTP-protokollan lisäksi esimerkiksi TLS-protokollan käyttämisen.

HTTP sisältää erilaisia statuskoodeja, joiden tarkoituksena on kertoa, kuinka palvelin suoriutui pyynnön toteuttamisesta. Statuskoodi palautetaan asiakaspäälle palvelimelta tulevan vastauksen yhteydessä. Statuskoodi koostuu kolmesta numerosta, joista ensimmäinen kertoo vastauksen luokan ja kaksi jälkimmäistä erottelevat koodin luokan sisällä. Vastausluokkia on viisi: *informatiivinen* (1xx), *onnistuminen* (2xx), *uudelleenohjaus* (3xx), *asiakasvirhe* (4xx) ja *palvelinvirhe* (5xx). Tunnetuin statuskoodi on 404, joka tarkoittaa, ettei pyydettyä tiedostoa löydy palvelimelta. (Pihlajaniemi, 2012.).

2.6.4 REST

REST (*Representational State Transfer*) on Roy Fieldingin (2000) väitöskirjassaan esittelemä sovellusarkkitehtuurityyli hajautetuille hypermediajärjestelmille. Hypermediajärjestelmä tarkoittaa järjestelmää, joka koostuu toisiinsa linkitetyistä mediadokumenteista. REST kehitettiin alun perin WWW:tä varten, mutta sitä voi hyödyntää myös muiden riippumattomien sovellusten kehityksessä, aivan kuten mitä tahansa muuta sovellusarkkitehtuurimallia. (Pihlajaniemi, 2012.).

Päämotivaatio REST-arkkitehtuurityylin kehitykselle oli tarve pystyä kommunikoimaan eri web-sovellusten välillä. Ensimmäinen versio RESTistä kehitettiin vuosien 1994–1995 välillä. Samana aikana Fielding kumppaneineen kehitti HTTP 1.0 -protokollan ja suunnitteli alustavaa versiosta HTTP 1.1:stä. REST toimii HTTP:n kehityksen pohjana, mutta samalla REST kehitettiin HTTP:n kehityksessä ilmenevien tarpeiden pohjalta. Alun perin REST tunnettiin nimellä (*HTTP Object Model*), mutta se vaihdettiin, koska vanha nimi johti usein virheelliseen tulkintaan teknologian käyttötarkoituksesta. REST pyrkii luomaan kuvan siitä, miten hyvin suunnitellun web-sovelluksen tulisi toimia: web-sovelluksen tulisi olla kokoelma toisiinsa linkitettyjä sivuja, jotka kukin *esittävät* (*represent*) käyttäjälle yhden *tilan* (*state*) sovelluksesta. Tilasta toiseen *siirrytään* (*transfer*) sivujen sisältämien linkkien kautta. REST-arkkitehtuuri ei ole standardoitu, vaan se voidaan nähdä joukkona suunnitteluperiaatteita rajapintojen toteuttamiseksi. RESTin käytön rajoitteita ja heikkouksia ovat joissain tapauksissa *asiakas-palvelin* (*client-server*), *välimuisti* (*cache*), *tilattomuus* (*statelessness*), *yhdenmukainen rajapinta* (*uniform interface*), *kerroksittainen järjestelmä* (*layered system*) sekä *ladattava koodi* (*code-on-demand*). (Fielding, 2000.; Pihlajaniemi, 2012.).

RESTin toimintaperiaate on yksinkertainen; selain tekee palvelimelle kutsun, jonka palvelin käsittelee ja palauttaa asianmukaisen vastauksen. RESTiin perustuvia web-palveluita, jotka käyttävät HTTP-protokollaa, kutsutaan *RESTful*-palveluiksi. Nämä palvelut hyödyntävät erilaisia HTTP-metodeja kutsuissa

ja vastauksissa. RESTin hyödyntämiä metodeja ovat GET, POST, PUT ja DELETE. (Valkama, 2014.).

2.7 Yhteenveto

Luvussa luotiin katsaus JavaScriptin historiaan ja kielen määritelmään. JavaScriptin todettiin olevan pääasiassa asiakaspäässä eli web-ympäristössä käytettävä korkean tason dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli. Kielen kehitti alun perin vuonna 1995 Brendan Eich. JavaScript muodostuu kolmesta eri osasta: ECMAScript, DOM ja BOM. JavaScriptin ymmärtämisen kannalta on keskeistä ymmärtää kielen toimintaympäristö ja siihen liitetyt käsitteet. Toimintaympäristö ja käsitteet auttavat luomaan kokonaiskuvan JavaScriptin ajoympäristöstä, kielen eri mahdollisuuksista ja toimintatavasta. Luvussa luotiin myös katsaus JavaScriptin käyttömahdollisuuksiin ja JavaScript-tiedostojen lisäämiseen web-sivulle. JavaScript-tiedostot kannattaa tuoda web-sivulle ulkopuolisesta lähteestä. Tällöin ei synny ongelmia esimerkiksi siitä, miten skriptissä kirjoitetaan merkit ”<” ja ”&”, joilla on HTML-merkkauksessa erikoisasema. Tiedostojen tuonnin jälkeen JavaScriptillä voidaan esimerkiksi näyttää ponnahdusikkunoita, lisätä erilaisia attribuutteja elementteihin, luoda, muokata tai poistaa HTML:ää, seurata hiiren liikkeitä tai toteuttaa selaimessa toimivia pelejä. Lisäksi on hyvä muistaa, että JavaScriptin toiminta riippuu käytettävästä selaimesta ja sen versionumerosta. Web-sivun sisällön tulee kuitenkin olla aina saatavilla käytettävästä selaimesta tai sen versionumerosta riippumatta.

3 NYKYAIKAINEN WEB-OHJELMOINTI JAVASCRIPTIN AVULLA

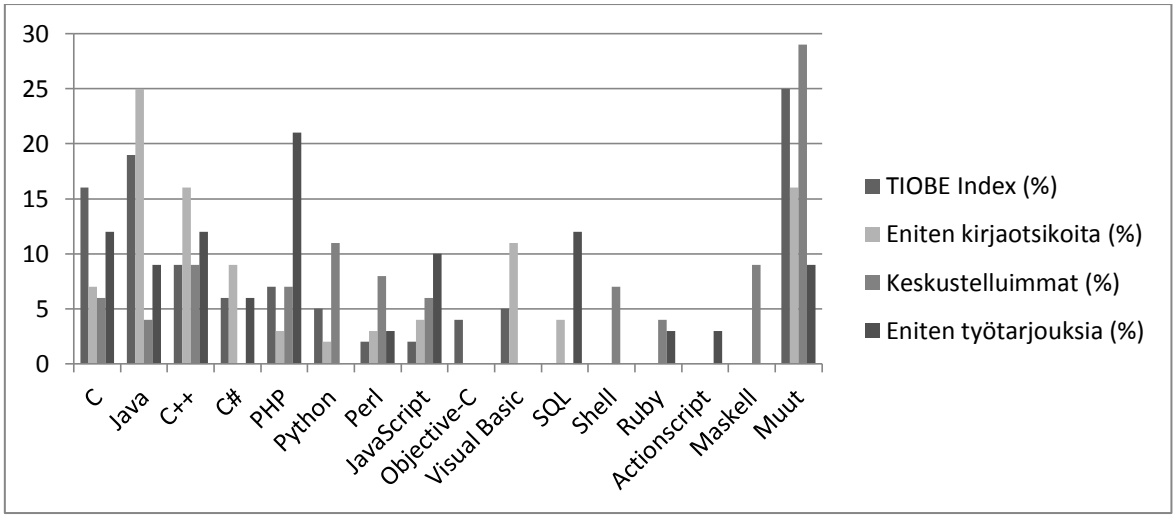
Luvussa esitetään tyypillisiä piirteitä nykyaikaisen web-sovelluksen toteuttamiseen JavaScriptin avulla ja sen tuomia haasteita. Alaluvussa 3.1 kuvataan JavaScriptin suosioon vaikuttavia tekijöitä, minkä tarkoitus on auttaa ymmärtämään kielen ominaisuuksia, jotka tekevät siitä suositun. Alaluvussa 3.2 esitetään nykyaikaisen web-sovelluksen edellytyksiä eli sovelluksen suunnittelua ja MVC-arkkitehtuuria sekä sen eri variaatioita. Alaluvussa 3.3 käydään läpi erilaisia JavaScript-kirjastoja, yksi automaatiotyökalu sekä sovelluskehys, jota käytetään tässä tutkimuksessa.

3.1 JavaScriptin suosioon vaikuttavat tekijät

Ohjelmointikielten listaaminen on helppoa, mutta niiden luokittelu kielen suosion perusteella on vaikeaa. Tämä johtuu useista erilaisista osatekijöistä. Yrityksiin ei voida esimerkiksi lähettää tutkijoita, jotka selvittävät, mitä ohjelmointikieliä yritys käyttää. Ohjelmointikielten suosio pitää selvittää epäsuoralla tiedonhankinnalla. Epäsuoralla tiedonhankinnalla tarkoitetaan tietoa, joka on kerätty suosioon vaikuttavien tekijöiden pohjalta. (King, 2011.). JavaScriptin suosioon ei ole olemassa vain yhtä tekijää vaan suosio riippuu mm. JavaScriptiin liitettyjen teknologioiden helposta saatavuudesta ja monipuolista käyttömahdollisuuksista. King (2011) on tutkinut JavaScriptin suosiota etsimällä vastaukset seuraaviin kysymyksiin ja menetelmiin:

- Missä ohjelmointikielessä on eniten avoimia työpaikkailmoituksia?
- Mitkä ohjelmointikielet ovat nousseet kuumimmaksi puheenaiheeksi erilaisilla keskustelusivustoilla?
- Mistä ohjelmointikielestä on kirjoitettu eniten kirjoja?
- TIOBE-indeksin käyttö

Kuvio 12 kuvaa viidentoista suosituimman ohjelmointikielen tilannetta vuonna 2011. Kuviossa on käytetty apuna neljää edellä mainittua tekijää. Lokakuussa 2014 JavaScriptin suosio TIOBE Softwaren (2014) -indeksin mukaan on pysynyt yhä noin 2 %:n tasossa, mutta se on noussut vuodesta 1999 alkaen seitsemän sijaa ylöspäin. Yksi mielenkiintoinen nousija TIOBE-indeksissä on Google Dart, joka on web-sovellusten tekemiseen suunniteltu ohjelmointikieli. Googlessa työskentelevän Millerin (2010) mukaan Dartin tavoite on korvata JavaScript täysin ja olla samalla verkkokehityksen *yleiskieli* (*lingua franca*) avoimessa verkkoympäristössä. Kuviosta 12 voidaan todeta, että JavaScript ei kuitenkaan tutkimuksen mukaan yllä millään osa-alueella niin korkealle, että sitä voitaisiin kutsua suosituimmaksi ohjelmointikieleksi.



KUVIO 12 Viisitoista suosituinta ohjelmointikieltä (King, 2011, 84)

Epäsuoran tiedonhankinnan lisäksi JavaScriptin suosioon vaikuttavia tekijöitä voidaan tarkastella muista näkökulmista. Näkökulmia voivat olla esimerkiksi JavaScriptiin liitetyt teknologiat, käyttömahdollisuudet ja saatavuus. Wilton ja McPeak (2010) ovat todenneet, että yksi merkittävä syy valita JavaScript on sen laaja käyttömahdollisuus ja saatavuus. Käyttömahdollisuuksilla ja saatavuudella tarkoitetaan selainten monipuolista tukea JavaScriptille. Voidaan olettaa, että käyttäjän vieraillessa web-sivulla hänellä on aina käytössä JavaScript tuki, jos sitä ei ole otettu erikseen selaimesta pois päältä. Muilla ohjelmointikielillä on yleensä suppeampi tuki web-selaimissa. (Wilton & McPeak, 2010.).

Kehittäjät voivat hyötyä JavaScriptin joustavasta syntaksista, matalasta oppimiskynnyksestä, tehokkaasta suorituskyvystä, heikosta tyypityksestä, voimakkaasta reflektio-mekanismista sekä mahdollisuudesta luoda nopeasti monipuolisia ja vuorovaikutteisia sovelluksia. Nopean kehityssyklin ja vuorovaikutuksen ansiosta web-sovellukset ovat alkaneet korvata työpöytäsovelluksia. JavaScriptiin on olemassa myös monia työkaluja, liitännäisiä ja kirjastoja, joiden tarkoituksena on helpottaa ohjelmistokehittäjien työtä. Joidenkin liitännäisten ja kirjastojen tarkoituksena on vain helpottaa JavaScriptin integraatiota

toisen ohjelmointikielen, kuten Javan, kanssa (Valkama, 2014). Toiset työkalut ja laajennukset auttavat kehittäjää tarkastelemaan ohjelmistokoodin suorituskykyä ja paikantamaan mahdollisia virheitä. Yksi työkalu on Mozilla Firefoxiin asennettava *Firebug-kehittäjätyökalu* (*web development tool*). Työkalu auttaa testaamaan ohjelmakoodin suoritusaikaa ja paikantamaan mahdollisia ohjelmistovirheitä (Mozilla Foundation, 2014). JavaScript on mielenkiintoinen sekoitus erilaisia ohjelmointikieliä ja niiden ominaisuuksia, jotka perustuvat *funktionaaliseen ohjelmointiin* (*functional programming*), *prototyyppeihin* (*prototypin*) sekä *muuttuviin objekteihin* (*mutable objects*). Muuttuvat objektit voivat muuttaa rakennetta ajon aikana. (Kienle, 2010.).

JavaScriptin kanssa työskenneltäessä kehittäjän on hyvä noudattaa tiettyä tyyliä luodessaan web-sovelluksia. Tyyliä voidaan kuvata *askel-askeleelta*-tyyliksi (*step by step*), jossa ohjelmaa kirjoitetaan vain vähän kerrallaan. Tämän jälkeen ohjelma testataan välittömästi. JavaScriptin virheitä on vaikea paikallistaa suuresta määrästä ohjelmakoodia, mutta tarkoitukseen on olemassa erilaisia testauskehyksiä, kuten QUnit ja JsUnit. JavaScriptin monipuolisia laajennuksia käyttämällä ohjelmakoodista pitäisi tulla paremmin jäsenneltyä, helppolukuisempaa ja ylläpidettävämpää verrattuna perinteiseen toteutukseen. Lively Kernelin kehittäjät uskovat, että JavaScriptin monipuolisten ominaisuuksien ansiosta negatiivinen käsitys JavaScriptistä paraneekin jatkuvasti. (Kienle, 2010.).

3.2 Nykyaikaisen web-sovelluksen edellytykset

Web-sovelluksen rakentaminen on muuttunut paljon viime vuosien aikana. Nykypäivänä web-sivut muistuttavat enemmän ohjelmia kuin yksinkertaisia dokumentteja. Tämä asettaa monia haasteita web-kehitykseen (Reis & Gribble, 2009). Käyttäjän näkökulmasta web-sivun on tarjottava monipuolinen käyttäjäkokemus käytettävästä laitteesta riippumatta. Ohjelmistokehittäjät voivat esimerkiksi määritellä erilaisia tyylejä eri kokoisille näytöille CSS:n mediakyselyjen avulla. Tätä kutsutaan responsiiviseksi suunnitteluksi ja toteutukseksi (Fielding, 2014.). Pelkkä responsiivinen suunnittelu ei kuitenkaan riitä luomaan vuorovaikutteista käyttäjäkokemusta. Vuorovaikutteinen käyttäjäkokemus perustuu myös sujuvaan web-sovelluksen käyttöön ilman ylimääräisiä sivunlatauksia. Responsiivisuuden ja vuorovaikutteisen käyttäjäkokemuksen lisäksi sovellus on suunniteltava helposti laajennettavaksi ja suorituskykyiseksi. Sovelluksen käyttöliittymä tulee pystyä erottamaan muusta ohjelmakoodista omaksi kokonaisuudeksi.

Alaluvun tarkoitus ei ole esitellä kaikkia tämänhetkisiä web-suunnittelumalleja vaan nostaa esille keskeisimpiä menetelmiä. Näitä menetelmiä käytetään apuna tutkimuksessa ohjelmoitavien web-sovellusten suunnitellussa sekä toteutuksessa. Stanley (2010) on sanonut, että sovellusten suunnittelu tulee aloittaa miettimällä työn tavoitteita, menetelmiä, aikatauluja ja resursseja.

Tutkimuksen tavoite ja käytettävät menetelmät on määritelty luvussa 1. Tutkimuksessa toteutettavia web-sovelluksia on verrattu toisiinsa luvussa 5

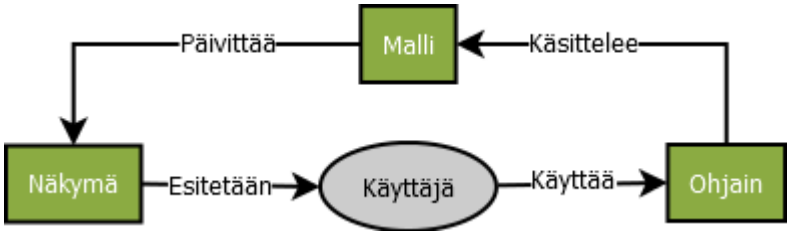
esitettyjen tietojen avulla. Tavoitteena oli myös, että sovellukset ovat valmiita vuonna 2015 ja ne ovat toteutettavissa kohtuullisessa ajassa yhden ihmisen resursseilla. Tutkimuksessa ei käsitellä palvelinpuolen teknologioita.

3.2.1 MVC-malli ja sen johdannaiset

Kehityksen ja ylläpidon helpottamiseksi web-sovelluksen eri vastuualueet on syytä erotella toisistaan. Vastuualueiden jakaminen auttaa ylläpidon lisäksi testauksessa, virheiden etsimisessä ja laajennettavuudessa sekä vähentää ohjelmistokoodin toistojen määrää. Vastuualueiden jakaminen helpottaa myös samanaikaista kehitystyötä projektin eri henkilöiden kanssa. *MVC-mallia* ja sen johdannaisia kutsutaan *esitysmalleiksi* (*presentation pattern*). Esitysmallien avulla ohjelman tilan esitys saadaan erotettua ohjelman muusta toiminnasta. Useat esitysmallit ovat muunnoksia alkuperäisestä MVC-mallista. (Kähkönen, 2014.; Jaggavarapu, 2012.). Erilaisia esitysmalleja voi toteuttaa web-sovellukseen itse tai käyttää valmiita sovelluskehyksiä, jotka noudattavat erilaisia esitysmalleja. Esitysmalleja ovat esimerkiksi *MVC*, *MVP* ja *MVVM*, joista jokainen noudattaa erilaista esitystapaa (Jaggavarapu, 2012). Usein kuulee puhuttavan myös *MV*-mallista*, jossa malli ja näkymä ovat aina mukana, mutta ohjain korvataan tilanteeseen sopivalla tavalla (Osmani, 2012a). Osmani (2012b) on sanonut, että niin kauan kuin ohjelmakoodi pidetään siistinä ja eroteltuna, voidaan jokaisella esitysmallilla saavuttaa lähes samat hyödyt.

MVC-malli

MVC-malli (*Model-View-Controller*) muodostuu kolmesta komponentista: *malli* (*model*), *näkymä* (*view*) ja *ohjain* (*controller*) (Reenskaug, 1979). Mallin tehtävä on huolehtia tiedon tallentamisesta, ylläpidosta ja käsittelystä. Malli voi olla joko passiivinen tai aktiivinen. Mallin ollessa passiivinen sen ei tarvitse kertoa tilansa muutoksista näkymälle. Passiivinen malli voi esiintyä esimerkiksi yksinkertaisessa tekstieditorissa, jossa näkymä päivittyy ainoastaan käyttäjän syöttäessä tietoa. Aktiivisen mallin tehtävä on kertoa sitä kuunteleville näkymille ja ohjaimille tilansa muutoksista suoraan kutsumalla palvelua, jonka avulla tieto välitetään. Näkymän tehtävä on näyttää käyttäjälle pyydettyjä tietoja. Näkymä rakentaa käyttöliittymän mallin tai ohjaimen välityksellä saadun tiedon perusteella. Ohjain on vuorovaikutuksessa näkymän ja mallin kanssa. Näkymä välittää tiedon ohjaimelle, joka pyytää mallia tallentamaan tai hakemaan syötettä vastaavan tiedon. Syöte voi olla painikkeen painaminen, tekstikenttään kirjoitettava hakusana tai mikä tahansa toiminto, joka pyytää mallilta tietoja. Ohjaimen tehtävä on yksinkertaisesti toimia käyttäjän ja mallin välissä. (Burbeck, 1992.; Jaggavarapu, 2012.). Kuvio 13 kuvaa tyypillisen MVC-mallin vuorovaikutuksen eri osien välillä.

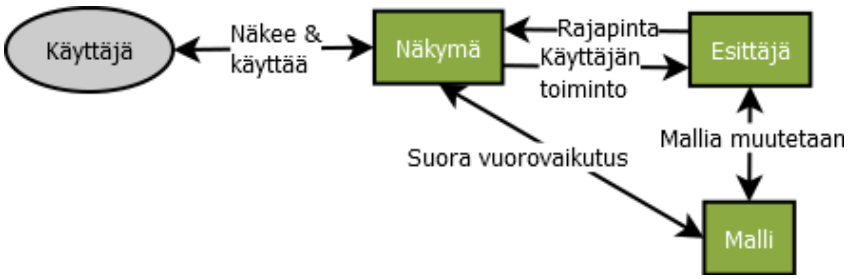


KUVIO 13 MVC-malli (Hales, 2012, 83)

MVP-malli

MVP-malli (*Model-View-Presenter*) eli malli-näkymä-esittäjä on johdannainen MVC-mallista. MVP-mallissa näkymä välittää käyttöliittymän liiketoimintalogiikan esittäjälle. Näkymän tehtävä on keskittyä parantamaan esityslogiikkaa. Esittäjän tehtävä on korvata perinteisesti ohjaimeen liittyviä tehtäviä, koska näkymä on MVP-mallissa se sovelluksen osa, joka vastaanottaa käyttäjän syötteitä. Näkymä ei itse käsittele toimintoja vaan pyytää esittäjää suorittamaan toimintoja vastaavan tehtävän. Tehtävät voivat olla esimerkiksi tiedon hakemista tai päivittämistä mallista. Näkymä keskustelee esittäjän kanssa, jonka jälkeen esittäjä päivittää näkymän tilaa rajapinnan kautta. Rajapinta mahdollistaa monia hyödyllisiä asioita, kuten helpomman testauksen. (Osmani, 2012b.).

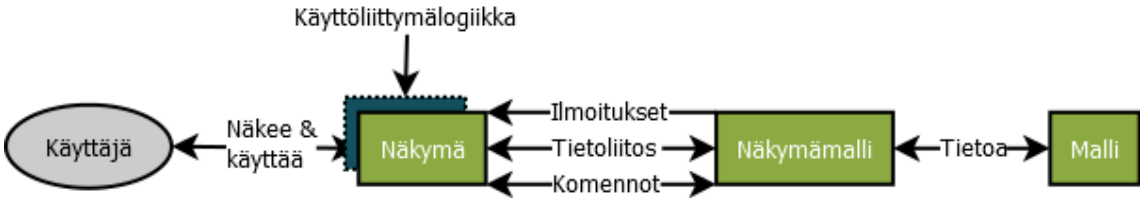
MVP-mallin tehtävä on pyrkiä ratkaisemaan kaksi keskeistä ongelmaa, jotka esiintyvät MVC-mallissa. MVC-malli joutuu perinteisesti tiedottamaan näkymälle tilansa muuttumisesta, jonka lisäksi näkymä on tietoinen mallista. Näkymän tietoisuus tarkoittaa, ettei näkymän ja mallin välillä ole erillistä rajapintaa. Tämän vuoksi näkymä ei ole enää niin passiivinen kuin sen pitäisi olla. MVP-mallin tarkoitus on mallin ja näkymän erottaminen toisistaan. MVP-mallista on olemassa passiivinen ja aktiivinen näkymä. Aktiivisen näkymän tarkoitus on helpottaa pienempien sovellusten toteutusta. Aktiivinen näkymä on muuten sama kuin passiivinen, mutta lisäksi näkymä voi olla suoraan vuorovaikutuksessa mallin kanssa. (Osmani, 2012b.; Osmani, 2012a.). Kuviossa 14 on esitetty aktiivinen MVP-malli sekä sen eri osien vuorovaikutus. Kuviosta 14 nähdään, että malli voi mm. välittää suoraan tietoa näkymälle ilman, että tieto pitää välittää esittäjän kautta.



KUVIO 14 Aktiivinen MVP-malli (Osmani, 2012a)

MVVM-malli

MVVM-malli (*Model-View-ViewModel*) eli malli-näkymä-näkymämalli on johdannainen MVC- ja MVP-mallista. MVVM-mallissa näkymämallin tehtävä on huolehtia sovelluksen toimintalogiikasta sekä tiedon muotoilusta. Näkymämalli on esimerkiksi vastuussa tietokannasta haetun päivämäärän muuttamisesta haluttuun muotoon näkymässä. Näkymän tehtäväksi ei jätetä esitettävän tiedon muotoilua, koska mallista haettava tieto muotoillaan käyttäjälle näytettävään muotoon jo näkymämallissa. Näkymän tehtäväksi jää vain käyttöliittymätapahtumien, sovelluksen toimintalogiikan sekä valmiin tiedon esittäminen. Käyttöliittymätapahtumaan liittyvä funktio voidaan suorittaa näkymämallin puolella. Tästä huolimatta näkymä vastaa sovelluksen toimintalogiikan mukaisesta tapahtuman liittämisestä oikeaan näkymämalliin (Kähkönen, 2014). MVVM-mallia noudattavassa sovelluksessa malli ei puutu sovelluksen toimintalogiikkaan eikä tiedon muotoiluun millään tavoin. Malli on siis täysin passiivinen. MVVM-malli sopii hyvin keskisuuriin ja suuriin sovelluksiin, mutta pienemmissä sovelluksissa se on liian raskas käyttää. Lisäksi dokumentaation määrä voi nousta liian suureksi. MVVM-mallin etuina ovat yksinkertaisen käyttöliittymälogiikan rakentaminen ja näkymämallin helppo testaus. (Osmani, 2012b.). Kuviossa 15 on esitelty MVVM-malli sekä sen eri osien vuorovaikutus.



KUVIO 15 MVVM-malli (Osmani, 2012a)

3.2.2 Yhden sivun sovellukset

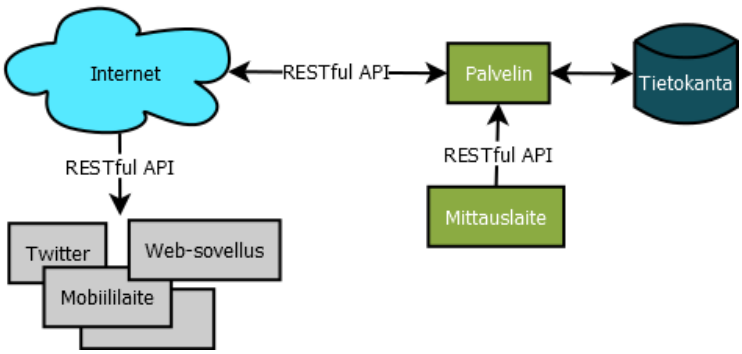
Yhden sivun sovellukset (*single page applications*) ovat tällä hetkellä web-kehityksen yksi kuumimmista trendeistä. Tämä käy ilmi lukemalla alan julkaisuja ja kuuntelemalla, mistä ihmiset puhuvat. Yhden sivun sovelluksilla tarkoitetaan sovelluksia, jotka toimivat yhdellä fyysisellä web-sivulla ilman ylimääräisiä sivun latauksia (Tesarik, Dolezal & Kollmann, 2008). Tällainen sovellus voi antaa välittömästi palautteen suoritetusta toiminnosta käyttäjälle (Tesarik ym., 2008). Tämä parantaa vuorovaikutusta ja käyttäjäkokemusta web-sivuilla. Yhden sivun sovellukset eivät ole uusi asia, vaan niitä on rakennettu aina AJAXin läpimurrosta asti. Gmail-sähköpostipalvelu on perinteinen esimerkki tällaisesta sovelluksesta. Gmailissa käyttäjä voi siirtyä esimerkiksi postilaatikosta roskapostikansioon ilman, että sivu latautuu uudelleen. Yhden sivun sovellukset tarvitsevat toimiakseen HTML:ää, JavaScriptiä ja AJAXia (Tesarik ym., 2008). Tällaisiin sovelluksiin liittyy myös erilaisia ongelmia. Yksi ongelma on selaimen takaisin-painike. Takaisin-painike vie yleensä väärälle sivulle, koska sivun osoi-

te selaimen osoiterivillä ei tavallisesti muutu tämänkaltaisissa sovelluksissa. Backbone.js-, Angular.js- ja Ember.js-sovelluskehyksissä on valmiiksi kuitenkin tuki selaimen takaisin-painikkeelle. (MacCaw, 2011.).

Yhden sivun sovellusten kehittämistä on viime vuosien aikana helpotettu merkittävästi JavaScript-sovelluskehysten avulla, jotka käyttävät MV*-mallia. Sovelluskehyksen avulla sovelluksen rakenne selkeytyy ja tarve kirjoittaa suuria määriä koodia DOMin käsittelyyn pienenee. Sopivan sovelluskehyksen käyttäminen voi parhaimmillaan parantaa tuottavuutta merkittävästi. (Gofore, 2013.).

3.2.3 REST API

REST API (*Application programming interface*) tarkoittaa *ohjelmointirajapintaa*, jonka avulla ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja eli keskustella keskenään. Ohjelmointirajapinnan avulla lähetettyä tietoja voidaan jakaa erilaisiin laitteisiin monella eri tavalla. Laitteita tai sovelluksia voivat olla mm. Twitter, mobiilisovellukset sekä web-sovellukset. Ohjelmointirajapinnan avulla sovellukset ovat helposti laajennettavissa ja sovelluksen eri vastuualueet voidaan erotella toisistaan. Nykyaikaiset sovelluskehykset ja niiden eri MV*-mallit käyttävät ohjelmointirajapintaa apuna web-sovellusten toteutuksessa. REST API voi lähettää tietoa SOAP- tai REST-rajapinnan kautta JSON- tai XML-muodossa. Vastaanottava sovellus käsittelee tiedon, minkä jälkeen tieto esitetään käyttäjälle. Avoimen tiedon lähetys mahdollistaa myös, että muut käyttäjät voivat ottaa tietoa vastaan omiin sovelluksiinsa ja esittää sitä haluamallaan tavalla. (Gao, Zhang & Sun, 2011.). Ohjelmointirajapinnan käyttö on helppoa nykyaikaisissa sovelluskehyksissä. Sovelluskehykset tarjoavat yleensä valmiit tavat tiedon lähetykseen tai vastaanottamiseen rajapinnan kautta. Kuvio 16 kuvaa ohjelmointirajapinnan toimintaa. Kuviossa 16 mittauslaite lähettää palvelimelle tietoja, jotka palvelin käsittelee haluttuun muotoon ja tallentaa ne tietokantaan. Palvelin lähettää tämän jälkeen Internetin yli halutulle sovellukselle käsitellyn tiedon, ja sovellus esittää sen käyttäjälle.



KUVIO 16 RESTful API (Gao, Zhang & Sun, 2011, 1)

3.3 Kirjastot ja automaatiotyökalut

Vaikka JavaScript on taipuisa ja muokattava ohjelmointikieli, sillä on muutama heikkous. Heikkouksiksi koetaan verkkoselainten pienet yksityiskohtaiset erot ja se, että monet ihmiset kokevat sillä ohjelmoinnin vaikeaksi. Sovelluksen optimointi eri verkkoselaimille voi vaatia paljon työtä. *JavaScript-kirjastot* ja *automaatiotyökalut* sisältävät erilaisia työkaluja ja kokoelman toimintoja kehityksen, ylläpidon ja suorituskyvyn nopeuttamiseksi. Lähes kaikki JavaScript-kirjastot on julkaistu copycenter- tai copyleft -lisenssillä. Ne mahdollistavat vapaan jakelun, käytön ja muokkauksen ei-kaupallisissa sovelluksissa. JavaScript-kirjastot toimivat hyvin web-sovelluksissa sovelluskehyksen tukena tai sellaisinaan. (McFarland, 2011.).

Alaluvun tarkoitus on esitellä kaksi JavaScript-kirjastoa ja yksi automaatiotyökalu, joita käytettiin toisen toteutettavan web-sovelluksen sekä valitun sovelluskehyksen tukena. JavaScript-kirjastojen ja automaatiotyökalujen valinta on vaikeaa, koska niitä on kymmeniä erilaisia. Monet valitsevat tarvittavat kirjastot ja työkalut tunnettavuuden, tarpeen, ominaisuuksien, dokumentaation tai vahvan yhteisön perusteella. Tutkimukseen valitut pääasialliset kirjastot ja automaatiotyökalu on esitelty alaluvuissa 3.3.1, 3.3.2 ja 3.3.3. Sovellus, jonka tueksi kirjastot ja työkalu otettiin, käytti myös muita JavaScript-kirjastoja, koska se osoittautui toteutuksen aikana tarpeelliseksi. Pääasialliset kirjastot ja automaatiotyökalu valittiin ominaisuuksien, tunnettavuuden sekä dokumentaation ja kirjallisuuden perusteella. Samanlaisen ulkoasun luomiseksi kummankin sovelluksen toteutuksessa käytettiin apuna Bootstrapin CSS-tyylitiedostoa. Tyylitiedostoa ei esitellä tässä tutkimuksessa.

3.3.1 jQuery

jQuery on vuonna 2006 julkaistu selaimille tarkoitettu ilmainen ja avoimen lähdekoodin JavaScript-kirjasto. Kirjasto kehitettiin alun perin selainten yhteensopivuusongelmien välttämiseksi. jQueryn ansiosta sovellusta ei tarvitse optimoida useille eri selaimille. W3Techs (2014) on sanonut, että jQuery on käytössä 60,9 %:ssa kaikista maailman web-sivustoista, mikä tekee siitä maailman suosituimman JavaScript-kirjaston. jQuery sisältää metodeja ja muita funktioita, joita voidaan käyttää ohjelmistokehityksen tukena. Sen avulla voidaan luoda esimerkiksi animaatioita tai yksinkertaisempia AJAX-kutsuja. Lisäksi jQuerystä on kirjoitettu useita kirjoja ja artikkeleita, ja siinä on helppolukuinen dokumentaatio. jQueryn käyttäjillä on myös ohjelmistokehittäjistä ja muista käyttäjistä muodostuva aktiivinen yhteisö tukenaan. (Wright, 2012.; jQuery, 2014.).

3.3.2 Modernizr

Modernizr on vuonna 2010 julkaistu selaimille tarkoitettu ilmainen ja avoimen lähdekoodin JavaScript-kirjasto. Kirjaston avulla pystytään tunnistamaan web-selaimen tukemat teknologiat. Web-selainten tukemien teknologioiden lisäksi kirjaston avulla HTML5-elementit saadaan käytettäväksi vanhemmilla web-selaimilla, minkä ansiosta sivustoon ei tarvitse erikseen lisätä esimerkiksi html5shim-kirjastoa. Kirjaston avulla voidaan varautua myös siihen, että web-sivun käyttäjällä ei välttämättä ole tukea JavaScriptille tai sitä ei ole asetettu web-selaimessa päälle. Tässä tapauksessa kirjaston avulla käyttäjälle voidaan näyttää haluttu virheviesti. (Modernizr, 2014.; Watson, 2012.).

3.3.3 Grunt

Grunt on JavaScriptillä kirjoitettu *automaatiotyökalu* (*task runner*), jonka avulla voidaan hallita erilaisia tehtäviä. Grunt on julkaistu vuonna 2012 ja sen avulla voidaan katsella tiedostoihin tehtyjä muutoksia, liittää CSS- ja JavaScript-tiedostoja yhteen, pakata tai pienentää tiedostoja, ajaa testejä ja katsoa, sisältääkö ohjelmakoodi virheitä. Gruntin käyttö helpottaa sovellusten ylläpidettävyyttä sekä parantaa ohjelmakoodin suorituskykyä. (Pillora, 2014.; Grunt, 2014.).

3.4 JavaScript-sovelluskehykset

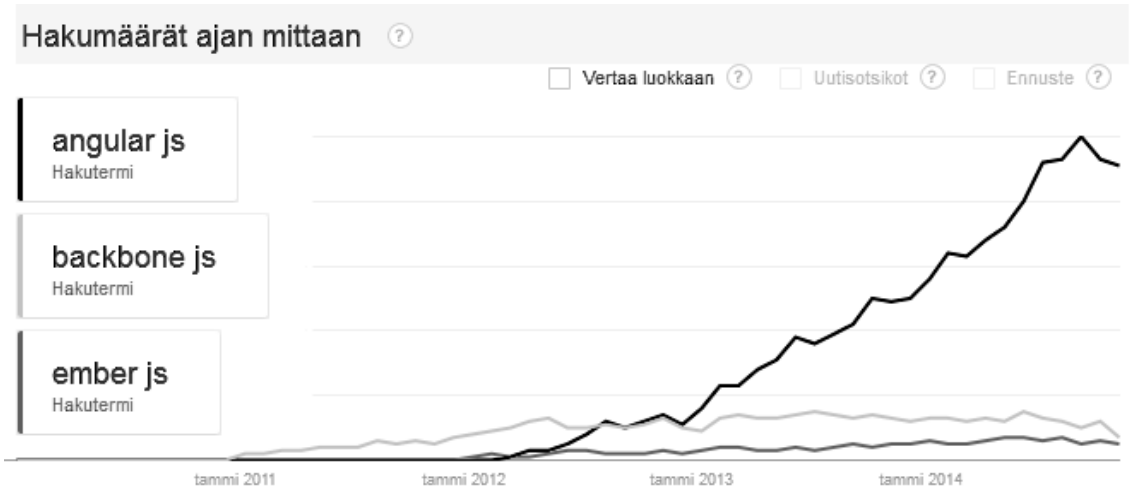
JavaScript-sovelluskehykset yhdistävät useita asiakas- ja palvelinpään teknologioita yhteen tuottamaan dynaamista sisältöä (Krithinakis, Athanasopoulos & Markatos, 2010). JavaScript-pohjaisia sovelluskehyksiä on useita erilaisia ja niiden valitsemiseen voidaan käyttää useita päiviä tai kuukausia. JavaScript-sovelluskehysten sekä kirjastojen käyttötarkoitus on pohjimmiltaan sama eli parantaa sovelluksen ylläpitoa ja suorituskykyä. Web-sovelluksen koosta riippuen vuorovaikutus käyttäjän kanssa lisääntyy ja sivuston pitää pystyä olemaan palvelinpäähän yhteydessä reaaliajassa. Ilman MVC-pohjaista sovelluskehystä ohjelmakoodista tulee helposti sekavaa ja rakenteeltaan epäjärjestelmällistä (Kerr, 2013). Sovelluskehykset ovat usein ilmaisia sekä vapaasti jaettavissa ja muokattavissa. Sovelluskehykset eroavat erilaisista kirjastoista merkittävästi. Sovelluskehysten, kuten Angular.js:n, Backbone.js:n ja Ember.js:n, tarkoitus on tarjota sovellusten tekemiseen MV*-pohjainen sovellusrunko sekä valmiita funktioita, jotka auttavat web-sovelluksen toteutuksessa. Kirjastojen tarkoitus taas on helpottaa yksittäisessä tehtävässä, kuten navigointipalkin teossa.

Sovelluskehystä valittaessa tulee ottaa huomioon, että kaikki sovelluskehykset toteuttavat JavaScript-luokkien käsittelyn omalla tavallaan. Tämä tarkoittaa, että sovellukseen tehtyjen funktioiden ohjelmakoodi on riippuvainen sovelluskehyksen rakenteesta. Tiettyä sovelluskehystä käyttävän ohjelmakoodin sovittaminen toisen sovelluskehyksen käytettäväksi vaatii luotujen funkti-

oiden muuntamista kohteena olevan projektin sovelluskehyksen mukaiseksi. (Kähkönen, 2014.).

3.4.1 Angular.js

Tutkimukseen valittiin sovelluskehykseksi *Angular.js*. Angular.js valittiin tutkimukseen, koska se tarjoaa monipuolisia ominaisuuksia dynaamisen asiakaspään web-sovelluksen tekoon. Angular.js sisältää kattavan valikoiman toimintoja, joiden avulla ohjelmakoodia voidaan jakaa erillisiin moduuleihin ja parantaa näin ohjelmakoodin uudelleenkäyttöä, ylläpidettävyyttä ja testattavuutta. Angular.js tarjoaa keskeisiä ominaisuuksia myös DOMin käsittelyyn, animaatioiden tekoon, tiedonsiirtoon ja testauksen. Sovelluskehyksen toiminta perustuu HTML-sivulle upotettaviin ng-alkuisiin attribuutteihin. Kun selain on ladannut HTML-sivun sivupohjaksi, Angular.js kirjoittaa sivun sisällön uudelleen JavaScript-mallin mukaisesti. Angular.js on julkaistu *MIT-lisenssillä*. MIT-lisenssi antaa käyttäjälle luvan käyttää, muokata, kopioida ja myydä ohjelmistoa rajoituksetta sillä ehdolla, että lisenssin teksti säilyy lähdekoodissa (*Open Source Initiative)*. (Lerner, 2013.). Angular.js valittiin lisäksi aikaisemman kokemuksen perusteella kyseisestä sovelluskehyksestä. Kuvio 17 esittää vuodesta 2011 alkaen Googlesta tietyllä hakusanalla haettujen JavaScript-sovelluskehysten suosiota. Kuviossa ei oteta huomioon Angular.js -sovelluskehyksen käyttöastetta websivuilla. Kuvion tarkoitus on olla suuntaa antava valittujen sovelluskehysten tämänhetkisestä suosiosta. Kuviosta 17 voidaan todeta, että Angular.js on vuonna 2014 ollut ylivoittoisesti haetuin sovelluskehys. Toisena on Backbone.js ja kolmantena Ember.js.



KUVIO 17 JavaScript-sovelluskehykset vertailussa (Google, 2014)

3.5 Yhteenveto

Luvussa käsiteltiin tyypillisiä piirteitä nykyaikaisen web-sovelluksen toteuttamiseen JavaScriptin avulla ja JavaScriptin mukana tuomia haasteita. Luvussa kuvattiin ensin JavaScriptin suosioon vaikuttavia tekijöitä. JavaScriptin suosioon ei vaikuta olevan vain yhtä yksilöivää tekijää vaan suosio rakentuu mm. JavaScriptiin liitettyjen teknologioiden, helpon saatavuuden ja sen monipuolisten käyttömahdollisuuksien perusteella. JavaScriptin tämän hetkinen suosio ei yllä millään osa-alueella niin korkealle, että sitä voitaisiin kutsua vielä suosituimmaksi ohjelmointikieleksi.

Kehityksen ja ylläpidon helpottamiseksi web-sovelluksen eri vastuualueet on syytä erotella toisistaan esitysmallien avulla. MVC-mallia ja sen johdannaisia kutsutaan esitysmalleiksi. Vastuualueiden jakamisella voidaan saavuttaa monia hyötyjä kehityksen ja ylläpidon lisäksi. Näitä hyötyjä ovat testaus, virheiden etsiminen, laajennettavuus ja ohjelmakoodin toistojen määrän väheneminen. Tämän lisäksi yhtäaikainen kehitystyö helpottuu, kun sovelluksen eri osa-alueet voidaan jakaa pienempiin osiin projektissa työskentelevien henkilöiden kanssa. Erilaisia esitysmalleja on mahdollista toteuttaa itse tai käyttää valmiita JavaScript-pohjaisia sovelluskehyksiä.

Esitysmallit eivät ole ainut tapa helpottaa kehitystä ja ylläpitoa. JavaScriptin tukena voidaan käyttää erilaisia kirjastoja, sovelluskehyksiä ja automaatiotyökaluja. Menetelmien käyttötarkoitus on pohjimmiltaan sama eli parantaa mm. sovelluksen ylläpitoa ja suorituskykyä. JavaScript-kirjastoja, sovelluskehyksiä ja automaatiotyökaluja on useita erilaisia ja niiden valinta riippuu toteutettavasta sovelluksesta sekä käyttötarkoituksesta. Tähän tutkimukseen valittiin Angular.js-sovelluskehyksen tueksi jQuery- ja Modernizr-kirjasto. Automaatiotyökaluksi valittiin Grunt. Ne valittiin ominaisuuksien, tunnettavuuden, aikaisemman kokemuksen sekä dokumentaation ja kirjallisuuden perusteella.

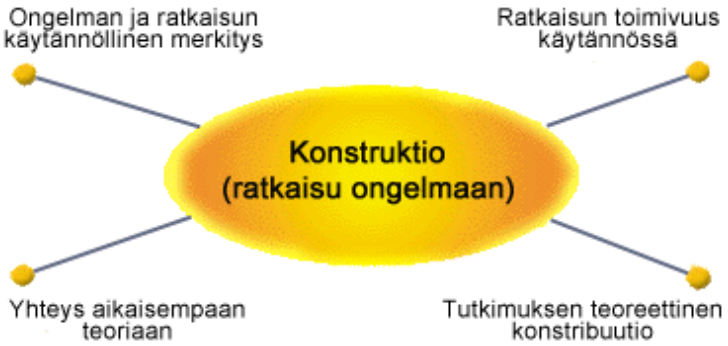
4 KONSTRUKTIIVISEN TUTKIMUKSEN TOTEUTTAMINEN

Tutkielman suunnittelutieteellisen, konstruktiivisen tutkimusotteen määrittelemää työtapaa käytetään selvittämään, miten nykyaikainen web-ohjelmointi JavaScriptin avulla on vaikuttanut asiakaspään web-sovellusten *tehokkuuteen*, *ylläpidettävyyteen*, *kytkentään* ja *kokoon*. Ratkaisukonstruktion toimivuutta testattiin tutkimuksessa vertaamalla toisiinsa kahta toiminnallisuudeltaan sekä ulkoasultaan samanlaista JavaScript-sovellusta, jotka toteutettiin eri menetelmillä. Tuloksena saatiin tieto, miten erilaisilla menetelmillä toteutetut sovellukset eroavat toisistaan edellä mainittujen laatuattribuuttien osalta.

4.1 Suunnittelutieteellinen, konstruktiivinen tutkimusote

Aken (2004) kuvaa suunnittelutieteen tarkoituksen olevan joko luoda tietämystä suunnittelua ja toteutusta varten eli konstruktio-ongelmien ratkaisemista varten, tai parantaa nykyisen systeemin suorituskykyä eli ratkaista parantamisongelma. Uuden innovaation hyödyllisyys tulee enemmin tai myöhemmin arvioida. Konstruktiivinen tutkimusote keskittyy tosielämän ongelmiin, jotka koetaan tarpeellisiksi ratkaista, eli pyritään kehittämään ongelmaan ratkaisukonstruktio. March ja Smith (1995) painottavat, että konstruoinnin tulokset voivat olla neljänlaisia (konstrukteja, malleja, metodeita ja toteutuksia) ja niissä voidaan käyttää kahta lähestymistapaa (rakentamista ja arviointia). Lukka (2001) on sanonut konstruoinnin tuloksille olevaan tunnusomaista, että ne eivät ole löydettyjä, vaan ne keksitään ja kehitetään. Kehittämällä konstruktio, joka poikkeaa kaikesta jo olemassa olevasta, luodaan jotain uutta. Uudenlaiset konstruktiot taas itsessään kehittävät uutta todellisuutta. Aken (2004) ei pidä uutta systeemiä suunnittelutieteen tuloksena, vaan hänen mukaansa suunnittelutieteen tavoitteena on tuottaa uutta suunnittelutietämystä – siis tietämystä, jota ammattilaiset voivat käyttää tulevien suunnittelu- ja konstruointiongelmien

ratkaisemisessa. (Järvinen & Järvinen, 2011.). Konstruktiivisen tutkimusotteen keskeiset elementit on esitetty kuviossa 18.



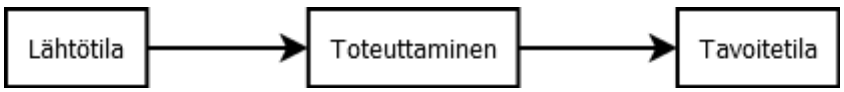
KUVIO 18 Konstruktiivisen tutkimuksen peruselementit (Lukka, 2001)

Konstruktiivisen tutkimusotteen ihanteellinen tulos on se, että tosielämän ongelma ratkaistaan implementoidulla uudella konstruktiolla. Ongelmanratkaisuprosessi tuottaa suuren kontribuution sekä käytännön että teorian näkökulmasta. Tällainen tulos olisi luultavasti tyydyttävä tutkimusprojektin kaikkien sidosryhmien kannalta. Tulee kuitenkin huomata, että akateemisesta näkökulmasta tarkasteltuna myös projekteilla, jotka syystä tai toisesta epäonnistuvat käytännön tasolla, voi olla huomattavaa teoreettista merkitystä. March ja Smith (1995) sanovat, että tulosta arvioitaessa peruskysymys on, kuinka hyvin se toimii. Arviointi vaatii mitta-asteikon kehittelyn ja sitten systeemin mittaamisen tuota mitta-asteikkoa käyttäen. Mitta-asteikon tarkoitus on määritellä, mitä yritetään saavuttaa. Marchin ja Smithin (1995) mielestä innovaation rakentaminen ja sen käyttö, siis innovaation elinkaari, tulee arvioida ideasta ensimmäiseen toteutukseen, sitten käyttöön ja lopulta hävitykseen. Hävitys tarkoittaa joko siirtymistä vanhan innovaation käytöstä uuden käyttöön tai vanhan innovaation käytön lopettamista. (Lukka, 2001.; Järvinen & Järvinen, 2011.).

4.2 Tutkimusprosessi

Konstruktiivista tutkimusotetta voi parhaiten kuvata tarkastelemalla sille tyypillistä tutkimusprosessia. Tutkimus alkaa *lähtötilasta*, jossa on jokin relevantti ongelma ja mahdollisuus teoreettiseen kontribuutioon. Joko innovaation puuttuminen tai vanhan innovaation huonot tulokset motivoivat rakentamaan uutta innovaatiota (Järvinen & Järvinen, 2011). Tutkijan tulisi pohtia tutkittavaa asiaa sekä käytännön että teorian kannalta. Hyvä tutkimusaihe on sellainen, jolla on käytännöllistä merkitystä tai jota on alianalysoitu aikaisemmassa kirjallisuudessa. On melkein aina mahdollista tunnistaa konstruoinnin lähtökohta ja myös senhetkinen ajatus toivotusta lopputilasta, eli toimivasta innovaatiosta (Järvinen & Järvinen, 2011). Tutkimusaiheen määrittelyn jälkeen tutkimusaiheesta

tulee hankkia syvällinen tuntemus sekä käytännöllisesti että teoreettisesti. Tämä vaihe muistuttaa hyvin paljon tavanomaista kenttätutkimusta. Etnografisia menetelmiä hyödyntämällä tutkija alkaa perehtyä tutkittavaan aiheeseen pyrkien saavuttamaan syvällisen yleisnäkemyksen lähtötilanteesta. Analyysin tulee paljastaa tutkimusaiheen suorat ja epäsuorat ongelmat ja tavoitteet. Seuraavaksi tutkijan tulee innovoida ratkaisumalli ja kehittää ongelman ratkaiseva konstruktio, jolla voi olla myös teoreettista kontribuutiota. Tätä vaihetta pidetään kriittisenä, koska jos innovatiivista konstruktiota ei pystytä kehittämään, tutkimusta ei ole syytä jatkaa. Vaihe on luonteeltaan luova ja heuristinen, ja siksi siihen ei ole tarjolla juurikaan yleispäteviä menetelmiä. Kun tutkija on innovoinut ratkaisumallin ja kehittänyt ongelman ratkaisevan konstruktion, on vuorossa *toteuttaminen* ja *testaus*. Toteuttaminen käsittää metodin, jonka avulla uskomme saavamme aikaan halutun muutoksen tavoitetilaan (Järvinen & Järvinen, 2011). Innovoitua konstruktiota ei testausvaiheessa testata pelkästään teknisesti, vaan samalla tutkimusprosessin toimivuus testataan kokonaisuudessaan. Toteuta ja testaa -tutkimusvaiheen saavuttaminen on hyvin vaativa tehtävä, ja sitä voidaan pitää positiivisena signaalina kyseisen konstruktiivisen tutkimusprosessin onnistumisesta. Tämän vaiheen jälkeen tutkijan tulee alkaa pohtia läpikäymäänsä oppimisprosessia. Prosessissa tärkeintä on tavoitetilan ja sen ennakkoehtojen analysoiminen. Tavoitetila kuvaa mallia siitä tilanteesta, jossa toivoimme asioiden olevan, kun olemme toteuttaneet ideamme (Järvinen & Järvinen, 2011). Tutkimusprosessin viimeisessä vaiheessa tutkijan tulisi tunnistaa ja analysoida teoreettinen kontribuutio. Tutkijan tulee pystyä selittämään projektin teoreettinen kontribuutio esimerkiksi reflektoimalla havaintonsa mahdollisesti jo olemassa olevaan teoriaan. Teoreettinen kontribuutio on mahdollista saavuttaa kahdessa mielessä: a) konstruktio on itsessään uusi, b) kehitetyn uuden teoreettisen kontribuution takana on riippuvuussuhde aikaisempiin tutkimuksiin. (Lukka, 2001.). Kuviossa 19 on esitelty tutkimusprosessin tärkeimmät pääkohdat, jotka ovat lähtötila, toteuttaminen ja tavoitetila.



KUVIO 19 Tutkimusprosessin tärkeimmät pääkohdat (Järvinen & Järvinen, 2011, 108)

4.3 Innovaation arviointi

March ja Smith (1995) määrittelevät arviointitutkimuksen yhdeksi ensimmäiseksi tehtäväksi mittarin määrittelyn. Määritellyllä mittarilla voidaan mitata käsitteistön, mallin, metodin sekä realisaation suoritusta. Mittarit ja niiden menetelmät on kuvattu taulukossa 3. Marchin ja Smithin (1995) mittareita ja menetelmiä käytettiin apuna tulosten ja tutkimuksen arvioinnissa. Taulukon 3 kriteereistä osa painottaa täsmällisyyttä ja osa relevanssia. Taulukko 3 on hyvä tapa osoittaa, että on olemassa useita arviointikriteereitä eri innovaatiotyypeille (kä-

sitteistö, mallit, metodit ja realisaatiot). Edellä mainittujen kriteerien lisäksi tulee muistaa, että tärkeää on aina arvioida, missä määrin asetetut tavoitteet saavutettiin. (Järvinen & Järvinen, 2011.).

TAULUKKO 3 Konstruktiivisen tutkimuksen tulosten arviointikriteerejä (Järvinen & Järvinen, 2011, 123)

Tutkimustulos	March ja Smith (1995) sekä heidän yleiset mittarinsa
Käsitteistö	täydellisyys, yksinkertaisuus, eleganssi, ymmärrettävyys ja helppokäyttöisyys; Järvinen ja Järvinen (2011) lisäävät vielä uutena kriteerinä hyödynnettävyyden
Malli, tavoitetila	mallin ja todellisuuden vastaavuus, täydellisyys, yksityiskohtaisuus, lujuus ja sisäinen johdonmukaisuus
Metodi	operationaalisuus (metodin kyky suorittaa tehtävä tai ihmisten kyky tehokkaasti käyttää algoritmista metodia), tehokkuus, yleisyys ja helppokäyttöisyys
Realisaatio	innovaation tehokkuus ja vaikuttavuus sekä sen vaikutukset ympäristöön ja käyttäjiin; Lientz (1983) lisää vielä uusina kriteereinä korjaavan, sopeuttavan, parantavan ja ehkäisevän ylläpidon

Lientz (1983) laati yhteenvedon vuosien 1977–1979 katsaustutkimuksista, jotka koskivat ohjelmiston ylläpitoa. Lientz (1983) havaitsi, että noin puolet ohjelmointi- ja suunnitteluhenkilöstön ajasta kului systeemin ylläpitoon. Toinen havainto oli, että noin 60 % ylläpidosta oli niin sanottua parantavaa ylläpitoa eikä korjaavaa ylläpitoa, kuten oli odotettu. Lano ja Haughton (1992) ovat kertoneet ylläpitokustannusten olevan arvioitu nousevan 80 %:iin ohjelman kaikista kustannuksista. Edellisen perusteella Järvinen ja Järvinen (2011) ehdottivat, että ylläpitokustannukset ja kokonaiskustannukset otetaan myös osaksi ohjelmiston arviointikriteereihin. (Järvinen & Järvinen, 2011.).

5 SOVELLUSTEN MÄÄRITTELY

Luvussa esitetään suunnittelutieteellisen, konstruktiivisen tutkimusotteen kohteena olleet web-sovellukset, aikaisempi tutkimus, sovellusten vertailussa käytetyt mittausmenetelmät ja ohjelmointivaihe. Lopulliset ja toimivat web-sovellukset on kuvattu tutkimuksen liitteissä 2 ja 3 ohjelmakooditasolla. Ratkaisukonstruktion toimivuutta on testattu ja pohdittu luvuissa kuusi ja seitsemän.

5.1 Sovellusten esittely

Tutkimuksen tavoite oli toteuttaa kaksi toiminnallisuuksiltaan sekä ulkoasultaan samanlaista web-sovellusta erilaisilla menetelmillä. Sovellusten toteutuksessa käytettyjen menetelmien pääpaino oli JavaScriptissä. Tutkimukseen valittujen web-sovellusten tarkoitus on nauhoittaa ja tallentaa käyttäjän kirjaamia työtehtäviä muistiin. Sovellusten yksi valintakriteeri oli, että ne ovat kohtuullisessa ajassa JavaScriptin avulla toteutettavissa olevia web-sovelluksia. Tavoitteena oli myös, että sovellusten avulla pystytään testaamaan aikaisemmissa luvuissa esiteltyjä tiedonsiirtotapoja ja nykyaikaisessa web-sovelluksessa käytettäviä menetelmiä. Sovellusten toteutuksessa käytettiin apuna tutkimuksen aikana hankittua teoreettista tietämystä JavaScriptiin liitettävistä käsitteistä ja sen toimintaympäristöstä. Lisäksi tutkija oli hankkinut käytännön kokemusta JavaScriptistä Tieturi Oy:n 29.10.2014 järjestämässä Angular.js-koulutuksessa. Koulutuksessa käsiteltiin JavaScriptiä ja Angular.js-sovelluskehyksen perusteita. Web-sovellukset toteutettiin käyttäen progressiivista kehitysmallia. Sovellusten suunnittelussa ja niiden vertailussa ei otettu huomioon ulkoisia laatutekijöitä. Sovelluksia on vertailtu toisiinsa määritettyjen mittausmenetelmien perusteella luvussa 6.

Kuviossa 20 on kuvattu toteutettavien sovellusten prototyyppimalli, joka auttaa lukijaa hahmottamaan sovellusten tärkeimpiä toiminnallisuuksia. Prototyyppiä voidaan kutsua sivun ulkoasuksi, mutta tässä tapauksessa se tarkoittaa luonnostelmaa, jonka avulla hahmotellaan näkymät, painikkeet, tekstikentät ja

muut sovelluksen kannalta tärkeät elementit (Fling, 2009). Sovellukset on suunniteltu *puhelimet ensin* (*mobile first*) periaatteella. Puhelimet ensin viittaa tapaan suunnitella ensin web-sivun ydintoiminnallisuudet sisältävä mobiilinäkymä ja laajentaa sitä lisätoiminnallisuuksilla tarpeen mukaan isompiin selainikkunoihin (Fielding, 2014).

KUVIO 20 Toteutettavien web-sovellusten prototyyppimalli

Kuviosta 20 voidaan nähdä sovelluksen kolme päänäkymää: kirjautuminen, nauhoitus ja nauhoitukset. Käyttäjä pääsee kirjautumaan sovellukseen sisään, kun hän on ensin rekisteröitynyt sen käyttäjäksi. Käyttäjä voi palauttaa tämän lisäksi unohtuneen salasanan, joka lähetetään hänen sähköpostiosoitteeseensa. Halutessaan käyttäjä voi valita ”muista minut”, jolloin käyttäjän tiedot tallennetaan selaimen *istuntoon* (*session*). Rekisteröinnin jälkeen käyttäjä voi kirjautua sisään sovellukseen ja aloittaa työtehtävien nauhoituksen. Työtehtävän nauhoittamiseksi käyttäjän täytyy valita työtehtävän tyyppi ja syöttää työtehtävän kuvaus. Tämän jälkeen käyttäjä voi aloittaa tehtävän nauhoituksen. Nauhoituksen pysäyttäminen tallentaa työtehtävän nauhoituksiin näyttäen työtehtävän tyypin, kuvauksen ja siihen kuluneen ajan. Käyttäjä voi halutessaan jatkaa työtehtävän nauhoitusta valitsemalla nauhoituksen viimeaikaiset työtehtävät -listalta.

5.2 Aikaisempi tutkimus

JavaScriptin uusien kirjastojen ja sovelluskehysten käytöstä on kirjoitettu paljon erilaisia oppaita. Oppaita löytyy teknologian kehittäjän omilta kotisivuilta ja esimerkiksi it-ebooks.info-sivulta. Aikaisemmat tutkimukset keskittyvät pääasiassa JavaScript-kirjastojen ja sovelluskehysten vertailuun, jonka lisäksi on tutkittu kielen dynaamisia ominaisuuksia, käyttömahdollisuuksia, valmiiden olioiden ja funktioiden suorituskykyä, muistin käyttäytymistä, tyypillisiä ongelmatilanteita sekä tietoturvaongelmia. Tästä huolimatta aikaisemmissa tutkimuksissa ei ole keskitytty riittävästi vertaamaan kahdella erilaisella menetelmällä toteutettua web-sovellusta, joiden pääpaino on JavaScriptissä. Aikaisemmat tutkimukset antavat hyvän pohjan tutkimuksen toteuttamiseen. Niiden pohjalta voidaan innovoida ratkaisumalli ja kehittää tarvittavat mittausmenetelmät sovellusten vertailuun. Ilman mittausmenetelmien luontia ja määrittelyä tutkimusta ei olisi syytä jatkaa. Mittausmenetelmien luonti on luonteeltaan luova ja heuristinen, ja siksi siihen ei ole tarjolla juurikaan yleispäteviä menetelmiä, vaan menetelmät pitää innovoida aikaisempien menetelmien ja tutkimusten pohjalta.

5.3 Mittausmenetelmät

Web-sovellusten *tehokkuutta*, *ylläpidettävyyttä* ja muita ominaisuuksia voidaan mitata standardointijärjestöjen määrittelemien *laatustandardien* avulla. Standardeja ovat kehittäneet mm. ISO/IEC-, ANSI- ja IEEE-standardointijärjestöt. Jørgensen (1999) totesi, että yleisimmät laadun määrittelyyn käytetyt mittausmenetelmät ovat seuraavat kolme menetelmää:

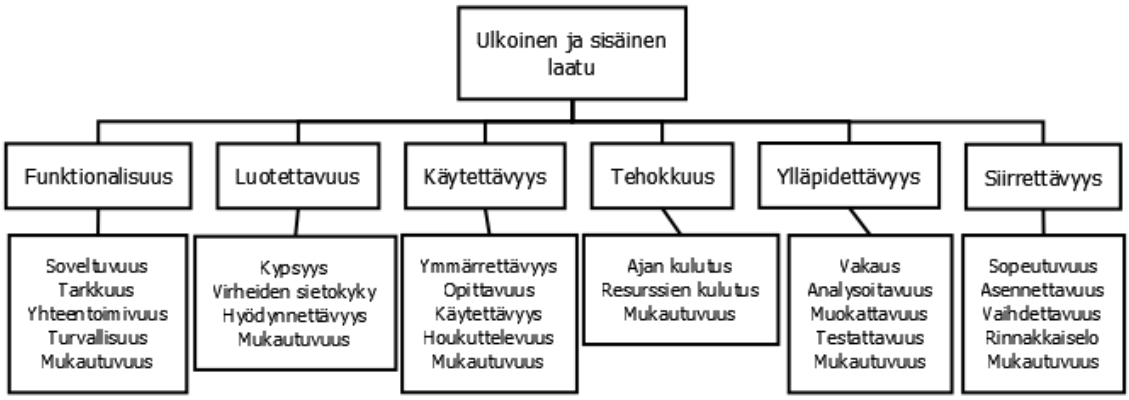
- Sovelluksen laatu määritellään jonkin standardin, kuten *ISO/IEC 8402–1986*:n tai *IEEE 610.12–1990*:n perusteella.
- Sovelluksen laatu päätellään käyttäjätyytyväisyyden perusteella.
- Sovelluksen laatu päätellään sen perusteella, miten paljon sovelluksessa on virheitä tai odottamattomia toimintoja.

Edellä mainitut mittausmenetelmät perustuvat samanlaiseen intuitioon laadun olemassaolosta, jonka jokainen käyttäjä arvioi omien tarpeidensa ja kokemustensa perusteella. Esimerkiksi eri henkilöt voivat jakaa saman mielipiteen siitä, että sovelluksen on täytettävä käyttäjän tarpeet. Henkilöt voivat kuitenkin tarkoittaa käyttäjän tarpeiden tyydyttämisellä eri asioita. Henkilö A voi pitää riittävänä, että käyttäjä saa ladattua pdf-tiedoston web-sivulta. Henkilö B voi sen sijaan pitää laatuna pdf-tiedoston löytämiseen kulunutta aikaa. Mitä tai miten mitataan, tarkoittaakin pitkälti henkilön omaa näkemystä kyseisestä asiasta. (Jørgensen, 1999.). Tässä tutkimuksessa käytetty laatustandardi on *ISO/IEC 9126-1*, jonka tavoitteena on määritellä yhdenmukaiset mittaustavat sovellusten

laatuvaatimuksille. *ISO/IEC 9126* on korvannut nykyään vanhentuneen *ISO/IEC 8402* -laatustandardin.

5.3.1 ISO/IEC 9126-laatustandardi

ISO/IEC 9126 on tällä hetkellä yksi yleisimmistä sovellusten laadun mittaamiseen käytetyistä standardeista. *ISO* ja *IEC* tulevat sanoista *International Organization for Standardization* ja *International Electrotechnical Commission*. ISO/IEC 9126 on jaoteltu neljään osaan: *laatumalli*, *sisäiset mittaukset*, *ulkoiset mittauks*et ja *käytön laatu*. Mittausmenetelmät rajataan *ISO/IEC 9126-1* -laatumalliin. ISO/IEC 9126-1 on standardin ensimmäinen osa ja se sisältää nykyisessä muodossaan laatuun liittyviä malleja ja laadun mittaamiseen käytettäviä mittareita. Näitä malleja ja mittareita voidaan käyttää ohjelmiston sisäisen ja ulkoisen laadun mittaamiseen. Kaikkia ominaisuuksia, joita voidaan testata kehitysprosessin aikana, kutsutaan sisäisiksi laatutekijöiksi. Ulkoinen laatu tarkoittaa käyttäjän määrittelemää laatua esimerkiksi sovelluksen käytettävyydestä. Ulkoista laatua voidaan ISO/IEC 9126-1 -standardin mukaan mitata, kun ohjelma on valmis ja käytössä. ISO/IEC 9126-1:llä on ohjelmistotuotannon kannalta keskeinen merkitys, sillä sen avulla voidaan määritellä, täyttääkö sovellus määritellyt laatuvaatimukset. (Botella ym., 2004.; ISO/IEC, 2000.). Kuviossa 21 on kuvattu ISO/IEC FDIS 9126-1:n (2000) määrittelemä laatumalli, joka jaottelee ohjelmiston sisäistä ja ulkoista laatua mittaavat attribuutit.



KUVIO 21 Sisäinen ja ulkoinen laatumalli

Funktionaalisuus (*functionality*) tarkoittaa sovelluksen kykyä täyttää asiakkaan ja käyttäjän asettamat tarpeet ja siihen kohdistuvat odotukset, kun sovellusta käytetään määritellyissä olosuhteissa. *Luotettavuus* (*reliability*) on sovelluksen kyky säilyttää tietty laatutaso, kun sitä käytetään määritellyissä olosuhteissa. *Käytettävyys* (*usability*) sisältää aspekteja, jotka kertovat, miten helppoa sovellusta on käyttää tietyin edellytyksin. *Tehokkuus* (*efficiency*) kuvaa sitä, miten hyvin sovellus vastaa käyttäjän pyyntöihin ja miten tehokas se on toiminnassaan määritellyissä olosuhteissa. *Ylläpidettävyys* (*maintainability*) on ohjelmistokehittäjiä var-

ten suunniteltu laadun osa-alue. Ylläpidettävyys kuvaa sitä, kuinka helposti sovellukseen voidaan tehdä muutoksia tai parannuksia ja miten hyvin se mukautuu ympäristössä tapahtuviin muutoksiin ja testaukseen. *Siirrettävyys* (*portability*) kuvaa, miten helposti sovellus on siirrettävissä eri alustoille, esimerkiksi siirrettäessä sovellus toiseen käyttöjärjestelmään. (ISO/IEC, 2000.).

5.3.2 Web-sovellusten vertailussa käytetyt laatuattribuutit

Web-sovellusten vertailuun ISO 9126-1 -standardista valitut laatuattribuutit ovat *tehokkuus* ja *ylläpidettävyys*. ISO 9126-1 -standardista jätettiin pois seuraavat laatuattribuutit: *funktionaalisuus*, *luotettavuus, käytettävyys* ja *siirrettävyys*. Mukaan otettujen kahden laatuattribuutin lisäksi web-sovellusten arvioinnissa otettiin huomioon vastuualueiden jakaminen eli *kytkentä* ja *koko*. Funktionaalisuus, luotettavuus ja käytettävyys jätettiin pois, koska niiden avulla mitataan enemmän ulkoista kuin sisäistä laatua. Kuten alaluvussa 5.1 mainittiin, sovellusten suunnittelussa ja niiden vertailussa ei otettu huomioon ulkoisia laatutekijöitä. Ulkoinen laatuvertailu sopii paremmin sovelluksiin, joissa vertaillaan kahta toiminnallisuuksiltaan ja ulkoasultaan erilaista web-sovellusta. Lisäksi tällaiset laatuattribuutit, kuten luotettavuus, vaatisivat laajaa testausta, joka ei olisi ollut järkevää tämän tutkimuksen puitteissa. Sovellukset toteutettiin käyttämällä MV*-mallia. Sovellukset kommunikoivat ohjelmointirajapinnan avulla palvelinpään kanssa (alaluku 3.2.3). Siirrettävyyden testaus olisi vaatinut kohdesovelluksen, johon toteutettujen sovellusten ohjelmakoodit olisi siirretty. Tällaisen sovelluksen ohjelmointi ja siirrettävyyden testaus ei olisi ollut järkevää tämän tutkimuksen puitteissa.

Alaluvussa 4.3 tarkasteltiin Marchin ja Smithin (1995) yleisiä mittareita konstruktiivisen tutkimuksen tulosten arviointiin. Sovellusten vertailussa ei kuitenkaan käytetty näistä yleisistä mittareista *käsitteistöä*, *mallia* ja *metodia*. Nämä kolme mittaria viittaavat ulkoisiin vaatimuksiin, joita tässä tutkimuksessa ei käsitelty. Järvinen ja Järvinen (2011) ovat todenneet, että käsitteistöön ja malliin liittyvät mittarit täydellisyys ja yksityiskohtaisuus ovat myös ongelmallisia vaatimuksia. Malli ei voi olla koskaan täydellinen, koska mallin kuvaus perustuu joihinkin muuttujiin, samalla kun toiset muuttujat on jätetty mallin ulkopuolelle. Jos malli olisi yksityiskohtainen, sen kuvaus olisi tavattoman laaja. Metodin arvioinnissa kyse taas on normatiivisen metodin arvioinnista. Normatiivinen metodi määrittää, mitä toimenpiteitä käyttäjän tulee tehdä. (Järvinen & Järvinen, 2011.).

Tehokkuus

Tehokkuus (*efficiency*) kuvaa sitä, miten hyvin sovellus vastaa käyttäjän pyyntöihin ja kuinka tehokas se on toiminnassaan määritellyissä olosuhteissa (ISO/IEC, 2000). Tehokkuudella voidaan mitata esimerkiksi sovelluksen kykyä suoriutua sille annetuista laskutehtävistä. Sovellukset ajettiin Mozilla Firefox 34.0.5 -selaimessa, joka käyttää Gecko-selainmoottoria. Sovellukset ajettiin sa-

massa selaimessa, koska selainten kyky suorittaa JavaScriptiä vaihtelee. Selainten tehokkuutta suorittaa JavaScriptiä voidaan testata esimerkiksi SunSpiderin avulla. SunSpider keskittyy ainoastaan JavaScriptiin eikä mihinkään muuhun selaimen ominaisuuteen (SunSpider, 2013).

Sovellusten tehokkuuden mittaamisessa käytettiin valmiita työkaluja, koska tutkimukseen varatut resurssit olivat rajalliset. Käytetyt työkalut ovat alaluvussa 3.1 mainittu Mozilla Firefoxiin asennettava Firebug-kehittäjätyökalu ja iMacros. iMacros on web-selaimiin asennettava kolmannen osapuolen lisäosa, jonka avulla voidaan nauhoittaa haluttuja toimintoja (iMacros, 2014). iMacrosiin nauhoitettiin sovelluksen testaava testitapaus, joka ajettiin molemmissa sovelluksissa viisi kertaa läpi. Jokaisen ajokerran päätteeksi testitapaukseen kulunut aika otettiin talteen Firebugin net-välilehdeltä. Lopuksi näistä viidestä ajokerrasta laskettiin keskiarvo, jotta mittaustulokset olisivat luotettavampia.

Ylläpidettävyys

Ylläpidettävyys (*maintainability*) kuvaa sitä, miten helposti sovellukseen voidaan tehdä muutoksia tai parannuksia ja kuinka hyvin se mukautuu ympäristössä tapahtuviin muutoksiin ja testaukseen (ISO/IEC, 2000). Hyvin toteutetun ohjelmakoodin ja rakenteen avulla voidaan selkeyttää sovelluksen ylläpitoa ja parantaa luettavuutta. Helposti luettava ohjelmakoodi on tärkeä osatekijä ohjelmistokehityksessä, koska usein ohjelmakoodin kanssa on tekemisissä usea ohjelmistokehittäjä. Luettavuus vähentää lisäksi ylläpitokustannuksia, jotka ovat merkittävä osa ohjelmiston elinjaksoa. Lano ja Haughton (1992) ovat todenneet, että ylläpitokustannusten on arvioitu nousevan 80 %:iin ohjelman elinjakson kustannuksista.

Ylläpidettävyys sisältää Marchin ja Smithin (1995) määrittelemän realisaation, joka sisältää Lientz (1983) määrittelemät ylläpidon lajit: korjaava, sopeuttava, parantava ja ehkäisevä huolta sekä Järvisen ja Järvisen (2011) ehdottamat kokonaiskustannukset. Kokonaiskustannuksissa arvioitiin sovellusten kehittämiseen kulunutta aikaa. Lisäksi arvioitiin ohjelmakoodin luettavuutta ja laajennettavuutta intuition ja aikaisemman kokemuksen perusteella.

Kytkentä

Kytkennässä (*coupling*) on kyse sovelluksen eri vastuualueiden välisestä riippuvuudesta. Sovelluksessa voi esiintyä *tiukkaa kytkentää* (*tight coupling*) tai *löyhää kytkentää* (*loose coupling*). (Yoon, 2014.). Käytännössä kahden olion välinen kytkentä on sitä tiukempi, mitä enemmän ne tarvitsevat ja käyttävät tietoa toistensa toteutuksesta. Sovelluksessa esiintyy tiukkaa kytkentää, kun sovelluksen eri vastuualueet ovat riippuvaisia toisistaan. Tiukkaa kytkentää esiintyy esimerkiksi silloin, kun olio muuttaa suoraan toisen olion sisältöä. Tiukassa kytkennässä ohjelmoijan on otettava huomioon vastuualueiden välinen riippuvuus. Jos sovelluksesta muutetaan yhtä osaa, joka on tiukasti kytketty toiseen, on suuri todennäköisyys, että muutos vaikuttaa molempiin osiin. Tämä taas johtaa mm. sovelluksen suurempiin ylläpitokustannuksiin. (Berard, 1993.).

Löyhässä kytkennässä sovelluksen eri vastuualueet koostuvat yksiköistä, jotka ovat itsenäisiä kokonaisuuksia (Berard, 1993). Löyhä kytkentä on usein haluttu tila ohjelmistokehityksessä. Löyhä kytkentä voidaan saavuttaa esimerkiksi asettamalla sovelluksen eri vastuualueet viittaamaan toisiinsa ainoastaan rajapinnan avulla. Löyhän kytkennän etuna on sovelluksen parempi laajennettavuus ja päivitettävyys kuin tiukassa kytkennässä. (Kähkönen, 2014.; Kaye, 2003.).

Sovellusten toteutuksessa käytettiin MV*-mallia, jossa malli ja näkymä ovat aina mukana, mutta ohjain korvataan tilanteeseen sopivalla tavalla (Osmani, 2012a). Kytkentää arvioitiin intuition ja aikaisemman kokemuksen perusteella.

Koko

Koko (*size*) kertoo sovelluksen ohjelmakoodin ja siihen liitettyjen tiedostojen viemän tilan. Sovelluksen koko voidaan selvittää Windowsista löytyvien työkalujen avulla. Sovelluksen koko voi tietyissä tilanteissa vaikuttaa sovelluksen suorituskykyyn ja latautumisnopeuteen. Web-sovelluksen latautumisnopeudella voi olla iso merkitys etenkin mobiilikäyttäjien kannalta, koska mobiililaitteet käyttävät yleensä hitaampaa Internet-yhteyttä kuin perinteiset pöytäkoneet. Koko oli yksi vertailtava tekijä tässä tutkimuksessa toteutettujen sovellusten mittaamisessa.

5.4 Ohjelmointivaihe

Ohjelmointi on keskeinen osa ohjelmistokehitystä, mutta vain osa siitä (Dooley, 2011). Varsinaista ohjelmointivaihetta web-sovellusten toteutuksesta ei käydä tässä tutkimuksessa läpi. Sovellusten asiakaspään ohjelmakoodit on sijoitettu liitteisiin 2 ja 3. Sovelluksen palvelinpään logiikassa käytettiin PHP 5.6.3 -ohjelmointikieltä ja MySQL-tietokantaa. Palvelinpään tehtävä on pitää yllä tietokantaa ja bisneslogiikkaa eli palvelun älykkyyttä ja luoda liittymä muihin järjestelmiin (Kovanen, 2013). Käytetyt palvelinpään teknologiat valittiin usean vuoden työkokemuksen perusteella edellä mainituista teknologioista. Aikaisempi työkokemus auttoi ehkäisemään palvelinpään ohjelmointivaiheen aikana ilmenneitä ongelmia. Palvelinpään ohjelmakoodia ei esitellä tutkimuksessa, vaan tutkimus keskittyy ainoastaan asiakaspään toteutukseen. Sovelluksen toteutuksen kannalta palvelinpään toteutus on kuitenkin välttämätöntä.

Toteutetut asiakaspään sovellukset kommunikoivat ohjelmointirajapinnan avulla palvelinpään kanssa käyttämällä seuraavia teknologioita: REST, HTTP ja JSON. Pyynnöt suoritetaan AJAXin avulla käyttäen asynkronisia kutsuja, jotka tapahtuvat käyttäjältä piilossa. Vastaanotettu tieto muokataan käyttäjälle näkyväksi JavaScriptin avulla. Molemmat sovellukset ovat niin sanottuja yhden sivun sovelluksia ja noudattavat MV*-mallia, jossa ohjain korvataan tilanteeseen sopivalla tavalla. Asiakaspään ohjelmointivaiheen aikana tulleita ongelmia py-

rittiin estämään aikaisemmalla Angular.js-koulutuksella, lukemalla alan kirjallisuutta ja käyttämällä apuna tutkimuksen ulkopuolisia henkilöitä. Sovellus toteutettiin ja testattiin paikallisesti asennetun kehitysympäristön avulla. Testit suoritettiin iMacrosin ja Firebugin avulla. Niiden avulla testitapauksiin kulunut aika saatiin otettua talteen alaluvussa 5.3.2 mainitulla menetelmällä. Sovellukset testattiin ja ajettiin Mozilla Firefox 34.0.5 -selaimessa, jossa niiden piti toimia.

5.5 Yhteenveto

Luvussa innovoitiin web-sovellusten vertailuun käytettävä ratkaisumalli ja ongelman ratkaiseva konstruktio. Luvun alussa esiteltiin suunnittelutieteellisen, konstruktiivisen tutkimusotteen kohteena olevat web-sovellukset. Luvussa määriteltiin, että tutkimukseen valittujen web-sovellusten tarkoitus on nauhoittaa ja tallentaa käyttäjän kirjaamia työtehtäviä muistiin. Samanlaisesta aiheesta on tehty vähän aikaisempaa tutkimusta. Aikaisempi tutkimus keskittyy enemmän JavaScript-kirjastojen ja sovelluskehysten vertailuun ja kielen dynaamisiin ominaisuuksiin, käyttömahdollisuuksiin, muistin käyttäytymiseen, tyypillisiin ongelmatilanteisiin sekä tietoturvaongelmiin. Lisäksi JavaScriptin valmiiden olioiden ja funktioiden suorituskykyä on tutkittu tieteellisissä julkaisuissa ja kirjoissa. Aikaisempi tutkimus antoi hyvän pohjan tutkimuksen toteuttamiselle. Tutkimuksessa web-sovellusten ominaisuuksia mitattiin standardointijärjestöjen määrittelemien *laatustandardien* avulla. Käytettäväksi laatustandardiksi valittiin ISO/IEC 9126-1, joka määrittelee laatuun liittyviä malleja ja laadun mittaamiseen käytettäviä mittareita. Web-sovellusten vertailuun standardista valitut laatuattribuutit ovat *tehokkuus* ja *ylläpidettävyys*. Lisäksi mukaan otettiin *kytkentä* ja sovelluksen *koko*. Sovellusten ohjelmointivaihetta ei käyty läpi, vaan sovellusten asiakaspään ohjelmakoodit on sijoitettu liitteisiin 2 ja 3. Palvelinpään ohjelmakoodia ei esitellä tutkimuksessa, vaan tutkimus keskittyi ainoastaan asiakaspään toteutukseen.

6 SOVELLUSTEN ANALYSOINTI

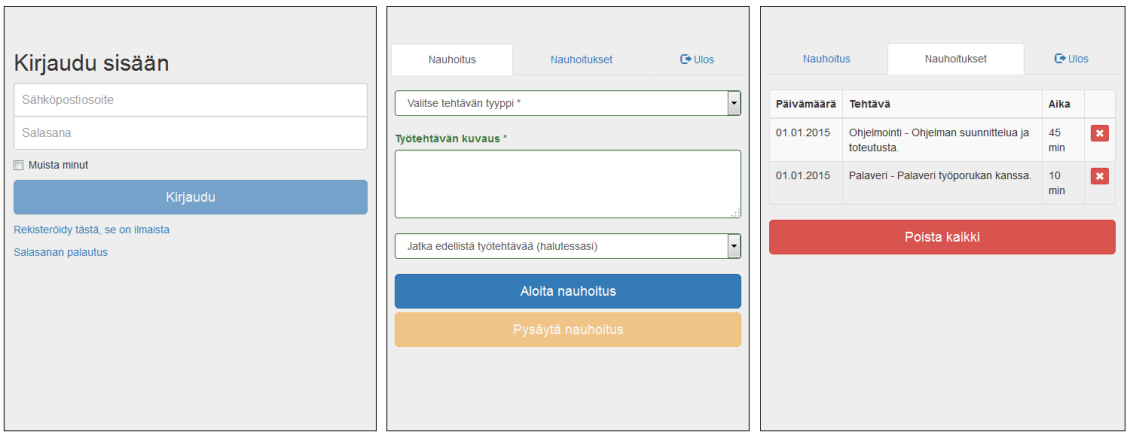
Luvussa on analysoitu ja vertailtu valmiita sovelluksia. Alaluvussa 6.1 esitellään web-sovellusten kehitysvaiheessa esille tulleita yleisiä asioita sekä valmiin sovelluksen lopullinen ulkoasu. Tämän jälkeen alaluvuissa 6.2, 6.3, 6.4 ja 6.5 on vertailtu valmiiden sovellusten tehokkuutta, ylläpidettävyyttä, kytkentää ja kokoa. Edellä mainitut mittausmenetelmät on määritelty alaluvussa 5.3. Esityksessä web-sovelluksista käytetään nimiä *Trick* ja *Trick2*. Trick kuvaa sovellusta, joka käyttää apuna alaluvussa 3.3 valittuja kirjastoja, automaatiotyökalua ja sovelluskehystä (jQuery, Modernizr, Grunt ja Angular.js). Trick2 taas kuvaa sovellusta, joka ei käytä apuna kolmannen osapuolen tekemiä kirjastoja eikä sovelluskehyksiä.

6.1 Yleiset havainnot

Tutkimuksen tavoite oli toteuttaa kaksi toiminnallisuuksiltaan sekä ulkoasultaan samanlaista web-pohjaista tuntikirjanpitosovellusta erilaisilla menetelmillä. Järvinen ja Järvinen (2011) ovat sanoneet, että tavoitetilaa ei aina kuitenkaan saavuteta, vaan toteutunut lopputila voi poiketa siitä. Voidaan puhua, että ”jäätiin hiukan” tavoitetilasta tai että ”mentiin hiukan yli” eli saavutettiin parempi tulos kuin osattiin alussa toivoa. Tämän tutkimuksen kohdalla voidaan sanoa, että halutusta tavoitetilasta jäätiin hiukan, koska Trick2-sovelluksessa selaimen takaisin-painike vie väärälle sivulle (ks. 3.2.2). Ongelma ei kuitenkaan vaikuta esimerkiksi sovelluksen tehokkuuteen tai testattavuuteen, koska automaatiotestit käyttävät sivulta toiselle siirtymisessä sovelluksen sisäisiä linkkejä. Selaimen takaisin-painikkeen korjaus olisi vienyt liikaa aikaa siitä saatuun hyötyyn nähden. Takaisin-painikkeen toiminta vaikuttaa käyttäjän näkökulmasta sovelluksen käytettävyyteen, joka taas mittaa enemmän ulkoista kuin sisäistä laatua. Tästä huolimatta luokittelen selaimen takaisin-painikkeen käytettävyyden myös yhdeksi osaksi sovelluksen toiminnallisuutta. Trick-sovelluksessa on täl-

lainen toiminnallisuus, minkä vuoksi sovellukset eivät ole toiminnallisuuksiltaan täysin identtiset eli tavoitetilasta jäätiin hiukan.

Pienistä semanttisista eroista huolimatta sovellukset täyttävät alaluvussa 5.1 asetetun vaatimuksen. Web-sovellusten tarkoitus on nauhoittaa ja tallentaa käyttäjän kirjaamia työtehtäviä muistiin. Sovellukset pystyttiin toteuttamaan myös kohtuullisessa ajassa. Toteutukseen kulunut aika on kuvattu alaluvuissa 6.1.1 ja 6.1.2. Lisäksi molemmat sovellukset täyttävät alaluvussa 3.2 määritellyt nykyaikaisen web-sovelluksen edellytykset ja käyttävät alaluvussa 2.6 kuvattuja tiedonsiirtotapoja. Sovellukset on rakennettu progressiivisen kehitysmallin mukaisesti. Aikaisempi työkokemus, tutkimuksen aikana hankittu teoreettinen tietämys, alan kirjallisuus ja Tieturi Oy:n järjestämä Angular.js-koulutus auttoivat ratkaisemaan sovellusten toteutuksen aikana tulleita ongelmia. Ongelmia olivat mm. tiettyjen ominaisuuksien toteuttaminen. Kaiken kaikkiaan sovellukset pystyivät täyttämään niille asetetut vaatimukset hyvin. Kuviossa 22 on kuvattu sovellusten lopullinen ulkoasu, joka on identtinen kummassakin toteutuksessa. Sovellusten ulkoasu vastaa alaluvussa 5.1 määritettyä prototyyppimallia.



KUVIO 22 Web-sovellusten lopullinen ulkoasu

Koska tutkimuksen tarkoitus oli analysoida kahta web-sovellusta, voitiin sovellusten analysointi aloittaa tarkastelemalla yleisellä tasolla ohjelmakoodia. Liitteistä 2 ja 3 nähdään, että sovellukset toteuttavat oliomaailman luokkien käsittelyn eri tavalla. Esimerkiksi Trick-sovelluksessa oliot tuodaan ohjaimen (ctrl) ja palvelun (service) käyttöön app.factoryn kautta parametreinä. Trick2-sovelluksessa oliot luodaan ohjaimessa ja palvelussa new-operaattorin avulla. Kähkönen (2014) on sanonut, että esimerkiksi valmiin sovelluskehyksen ohjelmakoodin hyödyntäminen toisessa ympäristössä on hankalaa, koska sovelluksen osien ohjelmakoodi on riippuvainen kehyksen rakenteesta. Tällaisen ohjelmakoodin sovittaminen toiseen projektiin vaatii ohjelmakoodissa käytettyjen luokkien ja sovelluskehyskohtaisten ominaisuuksien muokkaamista kohteena olevan projektin mukaiseksi. Pitää kuitenkin huomioida, että ongelma ei koske ainoastaan sovelluskehyksen ohjelmakoodia. Sama ongelma koskee myös ilman

sovelluskehystä toteutettua sovellusta. Jos tällainen sovellus siirretään osaksi projektia, joka käyttää valmista sovelluskehystä, on sovelluksen ohjelmakoodi hyvä muuttaa sovelluskehyksen rakenteen mukaiseksi. Tämä selkeyttää ja helpottaa ohjelmakoodin luettavuutta. Ohjelmakoodin tarkastelun lisäksi liitteestä 1 voidaan huomata, että molemmat sovellukset käyttävät lähes samanlaista hakemistorakennetta. Hakemistorakenteet on tehty samankaltaisiksi sovellusten vertailun helpottamiseksi.

Alaluvuissa 6.1.1 ja 6.1.2 on analysoitu tarkemmin sovellusten rakennetta, käytettyä kokonaisaikaa, keskeisiä ohjelmistoteknisiä piirteitä sekä niiden vaikutusta sovelluksen laatuominaisuuksiin. Sovellusten tehokkuutta, ylläpidettävyyttä, kytkentää ja kokoa on arvioitu tarkemmin luvun 6 muissa alaluvuissa.

6.1.1 Trick

Trick-sovelluksen toteutuksessa käytettiin apuna Angular.js-sovelluskehystä, jQuery- ja Modernizr-kirjastoa sekä Grunt-automaatiotyökalua. Sovelluskehyksen ja kirjastojen käytön pitäisi vähentää sovelluksen toteuttamiseen tarvittavan ohjelmakoodin määrää. Trick-sovelluksen kohdalla itse kirjoitetun ohjelmakoodin määrä oli 857 riviä, joka on 40 riviä vähemmän kuin Trick2:ssa. Sovellusten ohjelmakoodit on muotoiltu Eclipsen CTRL + SHIFT + F -komentosarjalla, joka poistaa tyhjät välimerkit ja muotoilee automaattisesti käyttäjän valitseman koodin tiettyyn muotoon. Tämän ansiosta sovellusten ohjelmakoodin määrä on helposti vertailtavissa. Liitteissä 2 ja 3 olevia ohjelmakoodeja ei ole muotoiltu Eclipsen avulla tilan säästämiseksi.

Sovelluskehyksen ja kirjastojen käytön tehokkuus tulee esiin myös muina ominaisuuksina. Angular.js tarjosi valmiin ominaisuuden esimerkiksi sivujen lataamiseen ngRoute-kirjaston avulla. Lisäksi sovelluskehyksen avulla pystyttiin pienentämään palvelinpäälle tehtävien pyyntöjen määrää ja pitämään ohjelmalogiikka erossa HTML-koodista. Pyyntöjen määrän pitäminen pienenä pitäisi säästää resursseja ja vaikuttaa sovelluksen tehokkuuteen. Ohjelmalogiikan erottaminen HTML-koodista auttaa parantamaan ylläpidettävyyttä.

Angular.js on saanut kritiikkiä mallin muutoksien tarkkailusta. Angular.js käyttää mallin muutoksien tarkkailussa niin sanottua *likaista tarkistusta* (*dirty check*). Tietyn laskennallisen, käyttäjälle näytettävän arvon muuttuessa jonkin pitkän prosessin osana järjestelmä joutuu reagoimaan jokaiseen muutokseen erikseen, vaikka järjestelmän riittäisi reagoida vasta prosessin lopuksi (AngularJS, 2014.).

Kirjastojen osalta jQuery osoittautui sovelluksen teossa turhaksi sen vähäisen käytön vuoksi. jQueryn avulla tehtävät ominaisuudet olisi pystytty toteuttaa ilman kyseisen kirjaston apua. Angular.js tarjosi myös valmiin ja tehokkaan tavan DOMin käsittelyyn, mikä omalta osaltaan vähensi jQueryn käyttöä. Modernizr-kirjaston käyttö helpotti huomattavasti selaimen tuen tarkistusta eri ominaisuuksille, mikä säästi sovelluksen kehitykseen käytettyä aikaa.

Trick-sovelluksen kehitykseen kulunut kokonaisaika oli 25,5 tuntia, josta Gruntin asennukseen käytetty aika oli 1,5 tuntia. Sovellukseen valittujen kirjas-

tojen käyttö oli helppoa, mutta suurin osa ajasta kului sovelluskehyksen opetteluun. Angular.js-sovelluskehyksen opettelussa on korkea oppimiskynnys, joka kuitenkin tasoittuu taitojen kasvaessa.

6.1.2 Trick2

Trick2-sovelluksen toteutuksessa ei käytetty apuna kolmannen osapuolen tekemiä kirjastoja eikä sovelluskehyksiä. Tällaisen sovelluksen tekeminen ei ole aina helppoa, koska jokainen ominaisuus täytyy suunnitella ja toteuttaa itse. Sovellus toteutettiin Trick-sovelluksen jälkeen, mikä auttoi sovelluksen suunnittelussa ja toteutuksessa. Trick2-sovelluksessa itse kirjoitetun ohjelmakoodin määrä oli 897 riviä, mikä tarkoittaa 4,6 % kasvua verrattuna Trick-sovelluksen ohjelmakoodin määrään.

Trick2-sovelluksen kehittäminen oli alussa nopeaa, koska tietyn sovelluskehyksen tai kirjaston edellyttämää syntaksia ei tarvinnut opetella. Vaikka sovelluksen aloittaminen osoittautui lopulta yllättävän helpoksi, suurimmat ongelmat alkoivat sovelluksen laajentuessa ja vaikeimpien ominaisuuksien toteuttamisvaiheessa. Tämä tarkoittaa, että ohjelmakoodiin pitää tehdä paljon ”puukotuksia”, mikä tekee luettavuudesta, luotettavuudesta ja ylläpidosta vaikeaa. Puukotukset lisääntyvät sitä enemmän mitä vaikeammasta toteutuksesta on kyse. Palvelinpäälle tehtävien pyyntöjen määrä on Trick2-sovellusta käytettäessä selvästi suurempi kuin Trick-sovelluksessa. Lisäksi Trick2-sovelluksessa DOMin käsittelystä pitää huolehtia itse, mikä voi huonolla toteutuksella vaikeuttaa HTML-koodin luettavuutta. Ilman kolmannen osapuolen tekemiä kirjastoja, automaatiotyökalua tai sovelluskehystä pystyy kuitenkin tekemään yhtä luettavaa, luotettavaa ja ylläpidettävää ohjelmakoodia kun edellä mainittuja menetelmiä käyttämällä. Tällaisen sovelluksen tekeminen vaatii vain paljon aikaa ja taitoa.

Trick2-sovelluksen kehitykseen kulunut kokonaisaika oli 20,5 tuntia eli 5 tuntia vähemmän kuin Trick-sovellukseen käytetty kokonaisaika. On kuitenkin vaikea sanoa, onko aika toistettavissa, koska sovellus toteutettiin Trick-sovelluksen jälkeen. Trick-sovelluksesta saatiin Trick2-sovellukseen valmis sovelluksen hakemistorakenne ja toimintalogiikka. Saatu hyöty oli arviolta 4 tuntia.

6.2 Tehokkuus

Tehokkuuteen liittyvät mittausmenetelmät ja niiden tulokset määriteltiin alaluvussa 5.3.2. Ennen jokaista mittausta selaimen historiatiedot nollattiin, jotta sovellukset eivät voineet ladata tiettyjä osia *välimuistin* (*cache*) avulla. Välimuistin tarkoitus on pitää yllä sellaisten osien sisältöä, joihin web-sivulla tullaan todennäköisesti viittaamaan. Välimuistin avulla voidaan välttyä ylimääräiseltä tiedonsiirrolta käyttäjän ja palvelimen välillä. Välimuistin käyttö edellyttää, että

selain on saanut ladattua käytettävät osat muistiin ennen sivulle saapumista (Juels, Jakobsson & Jagatic, 2006.). Tehokkuutta testattaessa haluttiin jäljittää tilannetta, jossa käyttäjä saapuu sivulle ensimmäistä kertaa.

Trick-sovelluksesta mitattiin kaksi erillistä tulosta. Toisessa versiossa JavaScript-tiedostot on pakattu Grunt-automaatiotyökalun avulla, toisessa tiedostoja ei ole pakattu lainkaan. Menetelmän avulla pystytään paremmin havainnollistamaan Gruntin käytöstä saatava hyöty toteutetuissa web-sovelluksissa.

iMacros-automaatiotestit on kuvattu liitteessä 4. Automaatiotestit ovat testitapausten kannalta identtisiä. Testit on kuvattu erikseen, koska iMacros käyttää sovelluksissa liikkuessaan erilaisia tunnisteita apunaan. Ensimmäisessä sovelluksessa iMacros liikkuu TXT-tunnisteen avulla ja toisessa sovelluksessa käytetään apuna ID-tunnistetta. iMacros-testien tulokset on kuvattu taulukossa 4. Testitulokset ovat keskiarvoja ajetuista testeistä (ks. alaluku 5.3.2).

Sivun alustusvaiheeseen ja automaatiotestin suorittamiseen kuluneet ajat on kuvattu erikseen. Sivun alustusvaihe tarkoittaa tilannetta, jossa selain pyytää sovelluksen aloitussivua sekä mahdollisesti muita siihen liittyviä tiedostoja ensimmäisen kerran. Automaatiotesti tarkoittaa tilannetta, jossa iMacros-testillä testataan sovelluksen tehokkuutta. Näiden vaiheiden jälkeen saadaan selvitettyä pyyntöjen määrä ja koko, onload-aika ja kokonaisaika. Pyyntöjen määrä kertoo sovelluksen tekemien HTTP-pyyntöjen määrän. Yksi pyyntö voi olla esimerkiksi index.css-tiedoston lataus. Koko muodostuu tiedonsiirtojen määrän koosta. Onload tarkoittaa aikaa, joka kuluu sivun eri osien lataukseen ja sovelluksen toimintakuntoon saamiseen. Näitä osia ovat mm. HTML-, CSS- ja JavaScript-tiedostot (Zerr, 2013). Kokonaisaika tarkoittaa automaatiotestien suorittamiseen kulunutta aikaa.

TAULUKKO 4 Firebug ja iMacros-testien tulokset

	Trick	**Trick + Grunt**	**Trick2**
Sivun alustusvaihe			
Pyynnöt	19	7	15
Koko (kb)	115,6	109,7	15,1
Onload (ms)	890,6	893,8	298,6
Automaatiotesti			
Pyynnöt	49	36	64
Koko (kb)	173	164,4	76,4
Kokonaisaika (s)	2,8	2,6	3,0
Yhteensä (s) onload + kokonaisaika	3,69	3,49	3,30

Taulukosta 4 nähdään, että Trick2 latautuu keskimäärin kolme kertaa niin nopeasti sivun alustusvaiheessa kuin Trick- tai Trick + Grunt -sovellus. Tämä johtuu sovellukseen ladattavien JavaScript-tiedostojen koosta ja määrästä. Koska kyseessä on hyvin yksinkertainen sivu, ei alustusvaiheessa ole suurta eroa. Käytettävyyden osalta käyttäjä kokee kummankin sivun latautuvan yhtä nopeasti.

Vaikka Trick ja Trick2 -sovellusten pyyntöjen määrä olisi sama, Trick latautuisi hitaammin, koska se lataa useita kirjastoja CDN:ää (ks. alaluku 2.3.1) käyttäen. Sovelluksen nopeutta ei voi päätellä pelkkien pyyntöjen määrästä, vaan yhtä tärkeää on selvittää tiedonsiirtojen koko. Ylimääräisten kirjastojen käyttöä tulee välttää, koska niiden lataaminen hidastaa sovelluksen toimintaa. Sovelluksen viemän muistitilan koon voidaan olettaa olevan sama kuin alustusvaiheen koon.

Sovelluskehyksen sekä kirjastojen tehokkuus tulee esille automaatiotestiä ajettaessa eli sovelluksen normaalikäytössä. Vaikka Trick joutuu lataamaan suurempia tiedostoja kuin Trick2, Trick on normaalikäytössä tehokkaampi vaihtoehto kuin Trick2. Trick lataa erilaiset toimenpiteet muistiin jo sivun alustusvaiheessa; sen sijaan Trick2 lataa tarvittavat toimenpiteet sovelluksen käytön aikana. Käyttäjän kokema viive muodostuu useammasta eri tekijästä ja palvelinpuolen merkitys vähenee, kun osa prosesseista siirretään käyttäjän selaimeen. Yleensä yli 80 % web-sivulla vietetystä ajasta käytetään selaimessa HTML-tiedoston latauksen jälkeen, joten verkkoliikenteen ja palvelinpuolen alle 20 %:n osuus ei ole oleellisin kokonaisuutta katsottaessa. Käyttöliittymässä käytetystä ajasta suuri osa menee sivuun liittyvien staattisten tiedostojen, kuten JavaScript-tiedostojen lataukseen (Souders, 2008). JavaScript-tiedostojen tiivistämisen hyöty tulee hyvin esille taulukosta 4. Taulukosta nähdään, että Grunt parantaa sovelluksen suorittamiseen kulunutta kokonaisaikaa entisestään. Useita eri menetelmiä käyttämällä voidaan saavuttaa selviä parannuksia sovelluksen tietyissä osa-alueissa jopa hyvin yksinkertaisella sivulla. Hyöty voi olla suurempi, jos samat optimoinnit toteutetaan sivulla, jolla on enemmän esitysgrafiikkaa ja esityslogiikkaa.

6.3 Ylläpidettävyys

Sovelluksen ylläpidettävyyteen vaikuttaa omalta osaltaan siinä käytettävä esitysmalli. Tässä tapauksessa molemmissa sovelluksissa päätettiin käyttää samanlaista esitysmallia eli passiivista MVC-mallia (ks. 3.2.1). Esitysmalli valittiin sovellusten toteutuksen aikana. Esitysmallin valintaan vaikutti käyttöönoton helppous, selkeys ja nopeus rakentaa sovellukset valmiiksi ja toimiviksi kokonaisuuksiksi. Kaikilla esitysmalleilla voidaan kuitenkin saavuttaa lähes samat hyödyt niin kauan kuin ohjelmakoodi pidetään siistinä ja eroteltuna (Osmani, 2012b). MVC-mallin käyttö helpotti sovellusten toteutusvaiheissa virheiden etsimistä ja tietyn vastuualueen jakamista omaksi kokonaisuudeksi sekä vähensi ohjelmakoodin toistojen määrää. Virheiden etsiminen JavaScriptistä on yleensä vaikeaa, koska web-selaimet antavat usein viallisesta ohjelmakoodista vaikeaselkoisen ilmoituksen selaimen konsolityökaluun (Ocariza ym., 2011; Mikkonen & Taivalsaari, 2007). Vastuualueiden jakaminen mahdollisti virheiden tehokkaan etsimisen ja toimivien osien poisrajauksen. Lisäksi jakaminen mahdollisti näkymän ja ohjaimen toiminnan muuttamisen mallia muuttamatta. Selkeän rakenteen ja ohjelmakoodin jaottelun ansiosta toistojen määrää pystyttiin vähentämään mm. käyttämällä samoja funktioita useissa eri paikoissa. Kerr (2013)

on todennut, että ilman kunnollista MVC-pohjaista sovelluskehystä ohjelmakoodista tulee helposti sekavaa ja rakenteeltaan epäjärjestelmällistä. Tällöin koodin ylläpidettävyys ja testattavuus kärsivät. Pelkän esitysmallin tai rakenteen perusteella ei voida kuitenkaan sanoa, onko jokin sovellus ylläpidettävämpi kuin toinen. Esitysmallin ja rakenteen lisäksi on tärkeää keskittyä sovelluksen ohjelmakoodin ja saatavissa olevan dokumentaation tutkimiseen. Näistä kahdesta ohjelmakoodilla on suurempi osa ylläpidettävyyden tarkastelussa.

Sovellukset eroavat merkittävästi toisistaan ohjelmakooditasolla (ks. liite 2 ja 3). Trick-sovelluksessa käytettävä sovelluskehys ja kirjastot tarjoavat monia valmiita ominaisuuksia, joiden ansiosta mm. ohjelmakoodin puukottamiselta voidaan välttyä tehokkaammin, kuten alaluvussa 6.1.2 todettiin. Valmiiden ominaisuuksien ansiosta kaikkea ei myöskään tarvitse kirjoittaa alusta alkaen itse. Ohjelmakoodin kirjoittaminen itse johtaa yleensä tietyn syntaksin ja sääntöjen puuttumiseen. Tämä ei välttämättä haittaa tietyn koodipätkän kirjoittajaa, mutta se voi häiritä muita projektissa työskenteleviä henkilöitä. Tällainen toimintamalli vaikeuttaa yleensä ohjelmakoodin luettavuutta ja ohjelmointia. Sovelluskehystä tai kirjastoja käyttäessä projektissa työskentelevät henkilöt joutuvat noudattamaan sovelluskehyksen ja kirjastojen tuomaa syntaksia ja sääntöjä. Angular.js hoitaa esimerkiksi tietyllä tavalla DOMin käsittelyn eli kaikki projektissa työskentelevät henkilöt tietävät oletuksena sovelluskehyksen määrittelemän toimintamallin DOMin käsittelylle.

Dokumentaatio on yksi osa sovelluksen ylläpidettävyyttä. Dokumentaation ansiosta sovelluskehittäjä voi tarkistaa esimerkiksi tuetut ominaisuudet tai tavan toteuttaa tietty ominaisuus. Trick-sovelluksessa käytettävät kirjastot tarjoavat kattavan ja helpon dokumentaation käyttäjän tueksi. Angular.js tarjoaa kattavan, mutta vaikeasti lähestyttävän dokumentaation. Vaikeasti lähestyttävä dokumentaatio hidastaa sovelluksen ohjelmointia. Trick2-sovellukseen ei ole saatavilla omia dokumentaatioita, joten tietyn ominaisuuden lisääminen pitää selvittää muuta kautta. Tämä voi viivästyttää ratkaisun saamista ongelmaan ja maksaa paljon rahaa.

6.4 Kytkentä

Kytkentä perustuu vahvasti MVC-mallin ja sen komponenttien välisen viestinnän toteutustapaan. Molemmissa sovelluksissa on käytetty samanlaista esitysmallia, eli näkymä, ohjain ja malli on selkeästi erotettu omiin tiedostoihinsa. Molemmat sovellukset pyrkivät vähentämään osien välistä kytkentää käyttämällä näkymän ja mallin välisessä tiedonsiirrossa tarkkailijamallia eli tässä tapauksessa ohjainta. Tämän ansiosta mallin ei tarvitse tietää, mitä näkymä tekee, kun se päivittää tilansa mallin uutta tilaa vastaavaksi. Sovelluskomponenttien välisten riippuvuuksien vähentäminen auttaa parantamaan sovelluksen ylläpidettävyyttä. Löyhästi kytketty rakenne helpottaa riippuvuuksien poistoa.

Kumpikaan sovellus ei onnistunut saavuttamaan löyhää kytkentää. Löyhän kytkennän saavuttaminen JavaScriptiä käyttävässä sovelluksessa on vaike-

aa, koska sovelluksen eri osat ovat aina vuorovaikutuksessa toistensa kanssa. Jos JavaScriptin kanssa aikoo tehdä minkäänlaista DOMin muokkaamista, on JavaScriptin tiedettävä mitä muokataan ja minne muokataan. Trick2-sovelluksessa esiintyy enemmän tiukkaa kytkentää kun Trick-sovelluksessa. Trick2 joutuu hakemaan mm. näkymästä lähetetyt arvot ohjaimessa ja muuttamaan ne mallille oikeaan muotoon. Trick pystyy palauttamaan ohjaimelle oikeat arvot oikeassa muodossa suoraan näkymästä, jolloin ohjaimen ei tarvitse huolehtia näkymältä tulevista arvoista. Lisäksi Trick käyttää apunaan Angular.js:n määrittelemiä rajapintaluokkia tiedon sisäiseen välitykseen. Trick2 käyttää perinteisiä olioviitteitä eri luokkien välillä. Rajapintaluokkien käyttö vähentää sovelluksen sidoksellisuutta ja nopeuttaa sen toimintaa.

Molempien sovellusten etuna on asiakaspään ohjelmakoodin erottaminen palvelinpään ohjelmakoodista ohjelmointirajapinnan avulla. Ohjelmointirajapinnan ansiosta sovellukset voidaan muuttaa ottamaan tietoa vastaan eri järjestelmästä rajapinnan osoitetta muuttamalla. Koska asiakaspään ja palvelinpään toimintalogiikka on eroteltu toisistaan, olisi asiakaspään ohjelmakoodi sovellettavissa esimerkiksi mobiilisovellusta tehtäessä. Käyttöliittymän ja palvelinpuolen erottaminen helpottaa myös tehtävien jakamista palvelin- ja käyttöliittymäasiantuntijoiden välillä. Tämä parantaa yleensä tuottavuutta, koska jokainen projektiin osallistuva henkilö voi keskittyä siihen osa-alueeseen, jonka hän tuntee parhaiten.

Vaikka sovelluksissa ei pystytty saavuttamaan löyhää kytkentää, Trick on kuitenkin lähellä sitä. Trick pystyy ilman ohjaimen muutoksia mukautumaan helposti esimerkiksi näkymältä tulevaan uuteen tietoon. Uutta sovellusta toteuttaessa ylläpidon helpottamiseksi on kannattavaa miettiä, miten sovelluksessa pystytään saavuttamaan mahdollisimman löyhä sidos.

6.5 Koko

Sovelluksen kokoon vaikuttavat sovelluksen laajuus ja käytettävien työkalujen määrä. Käytettävät työkalut tarkoittavat tässä tapauksessa erilaisia kirjastoja ja automaatiotyökaluja. Sovellukseen kokoon vaikuttaa myös olennaisesti se, tuodaanko käytettävät työkalut ulkopuolisesta lähteestä vai ladataanko ne samaan kansioon sovelluksen kanssa. Taulukossa 5 on esitetty erikseen seuraavat koot: Trick, Grunt, ladattavat JavaScript-tiedostot ja Trick2. Koon mittauksessa Trick-sovelluksessa ei ollut Gruntin käyttöön vaadittavia tiedostoja. Ladattavat JavaScript-tiedostot tarkoittavat ainoastaan tiedostoja, jotka Trick lataa käyttäen CDN:ää. Trick-sovelluksen yhteenlaskettu koko muodostuu itse kirjoitetusta ohjelmakoodista, Gruntista ja sovelluksessa ladattavista JavaScript-tiedostoista.

Alaluvussa 6.2 tehdyssä tehokkuusmittauksessa saatiin tulokseksi, että Trick latautuu sivun alustusvaiheessa hitaammin kuin Trick2. Tämä voidaan selittää ainakin osittain sovelluksen suuremmalla koolla. Sivun latausnopeuteen vaikuttaa myös käytettävä selain, verkkoyhteyden nopeus ja käyttöjärjestelmä. Vaikka Trick-sovelluksen koko on huomattavasti suurempi kuin Trick2:n,

selain ei joudu lataamaan Trick-sovelluksen kaikkia sisäisiä tiedostoja. Selain ei lataa Gruntin käyttöön vaadittavia tiedostoja, koska Grunt ei ole käytössä sovelluksen suorituksen aikana. Näin ollen vertailtavilla sovelluksilla on oikeastaan hyvin pieniä kokoeroja. Pienikin kokoero näkyy kuitenkin sovelluksen alustusvaiheessa hitaampina latausaikoina (ks. taulukko 4).

TAULUKKO 5 Sovellusten koot

	Trick	**Grunt**	**Ladattavat JS-tiedostot**	**Trick2**
Koko (kb)	26,4	9031,7	295	22,8
Yhteensä (kb)	9353,1			22,8

6.6 Yhteenveto

Tehtyjen havaintojen perusteella voidaan sanoa, että pienten semanttisten erojen vuoksi tavoitetilasta jäätiin hiukan. Sovellukset pystyvät tästä huolimatta täyttämään niille asetetut vaatimukset hyvin. Nykyaikaisten web-sovellusten toteutus pystyttiin tekemään kohtuullisessa ajassa ja sovellusten avulla voidaan nauhoittaa ja tallentaa käyttäjän kirjaamia erilaisia työtehtäviä muistiin. Sovellukset käyttävät apunaan alaluvussa 2.6 kuvattuja tiedonsiirtotapoja. Trick2 latautuu sivun alustusvaiheessa kolme kertaa niin nopeasti kuin Trick- tai Trick + Grunt-sovellus. Tämä johtuu mm. sovellukseen ladattavien JavaScript-tiedostojen koosta ja määrästä. Trick-sovellus suoriutui tästä huolimatta automaatiotestistä nopeammin kuin Trick2. MVC-mallin käyttö helpotti sovellusten toteutusvaiheissa virheiden etsimistä ja tietyn vastuualueen jakamista omaksi kokonaisuudeksi sekä vähensi ohjelmakoodin toistojen määrää. Kytkennän osalta kumpikaan sovellus ei onnistunut saavuttamaan löyhää kytkentää, koska JavaScriptiä käytettäessä sovelluksen eri osat ovat aina vuorovaikutuksessa toistensa kanssa. Sovellusten kehitykseen kulunut kokonaisaika ja ohjelmakoodin rivimäärä on esitetty taulukossa 6.

TAULUKKO 6 Sovellusten kehitykseen kulunut kokonaisaika ja ohjelmakoodin rivimäärä

	Trick	**Trick2**	**Ero %**
Kokonaisaika (tuntia)	25,5	20,5	19,6
Ohjelmakoodin määrä (riviä)	857	897	4,6

7 POHDINTA

Tämän suunnittelutieteellisen, konstruktiivisen tutkimuksen tarkoituksena oli selvittää, miten nykyaikaiset JavaScript-kirjastot, sovelluskehys ja automaatiotyökalu vaikuttavat asiakaspään web-sovelluksen tehokkuuteen, ylläpidettävyyteen, kytkentään ja kokoon. Alaluvussa 7.1 esitetään ensin tutkimuksen aikana käyty oppimisprosessi ja siinä esille tulleet asiat. Alaluvussa 7.2 on tutkimuksen tulokset ja niistä tehdyt johtopäätökset. Alaluvussa 7.3 kerrotaan tutkimuksen tulosten hyödyntämis- ja jatkokehitysmahdollisuuksista. Alaluvussa 7.4 on tunnistettu ja analysoitu tutkimuksen teoreettinen kontribuutio.

7.1 Oppimisprosessi

Tutkimus alkoi mielenkiintoa herättävästä aiheesta ja aikaisemmin tehdystä puutteellisista tutkimuksista. JavaScriptiin liittyvät käsitteet olivat alussa pääosin vieraita ja niiden tarkka käyttötarkoitus ei ollut tuttua. Tämä johti tutkimuksen alussa hajanaiseen rakenteeseen. Tutkimuksen edetessä JavaScriptin erilaiset käsitteet ja toimintaympäristö alkoivat hahmottua paremmin, mikä auttoi näkemään ja valikoimaan oleellisen tiedon sekä kytkemään yksityiskohdat kokonaisuuksiksi. Aikaisemmat tutkimukset ja teoriat antoivat hyvän pohjan tutkimuksen toteuttamiseen, vaikka ne eivät suoraan vastanneet tutkimuksen tutkimusongelmaan. Aikaisempien tutkimusten vanhat tutkimustulokset ja teoriat pystyttiin tulkitsemalla ja soveltamalla sulauttamaan tutkimuksen uudeksi teoriaksi ja malliksi. Näiden avulla pystyttiin innovoimaan ratkaisumalli ja kehittämään tarvittavat mittausmenetelmät sovellusten vertailuun. Ilman mittausmenetelmien luontia ja määrittelyä tutkimusta ei olisi ollut syytä jatkaa. Kun tarvittavat menetelmät tutkimuksen toteuttamiseen oli kehitetty, pystyttiin opittua tietoa käyttämään sovellusten toteutukseen, testaukseen ja tulosten arviointiin. Edellä mainitut asiat eivät olisi onnistuneet ilman tutkimuksen aikana hankittua kokemusta erilaisista käsitteistä ja menetelmistä nykyaikaisen web-sovelluksen toteutukseen. Sovellusten toteutus oli suoraviivainen vaihe, jolle oli

määritetty selkeä tavoite. Tavoitteen avulla pystyttiin määrittelemään, milloin sovellukset olivat valmiit ja täyttivätkö ne vaaditut laatuattribuutit sovellusten arvioinnille. Vaikka sovellusten toteutus sujui pääsääntöisesti hyvin, ei aikaisemmista lisäkoulutuksista olisi ollut haittaa. Aikaa kului paljon uuden asian opetteluun ja sen omaksumiseen. Toisaalta itselle uutta asiaa tutkiessa on harvoin täydellistä tietoa aiheesta – eihän se silloin olisi uusi asia. Arviointivaiheessa sovellusten tarkoitus ei ollut kilpailla paremmuudesta, vaan arvioinnissa keskityttiin toimintamallin pätevyyden ja todenmukaisuuden kriittiseen tarkasteluun. Tutkimus eteni koko prosessin ajan loogisessa järjestyksessä ja tieto on jäsennetty ja tulkittu uuden ajattelu- ja toimintamallin mukaisesti. Tutkimuksessa halutusta tavoitetilasta jäätiin hiukan (ks. alaluku 6.1), mutta pienistä semanttisista eroista huolimatta sovellukset pystyivät kaiken kaikkiaan täyttämään niille asetetut vaatimukset hyvin.

7.2 Tulokset ja johtopäätökset

Tutkimuksen menetelmiä, ratkaisun toimivuutta ja tuloksia käsitellään ensimmäisessä alaluvussa. Toiseen alalukuun on otettu mukaan konstruktiivisen tutkimuksen tulosten arviointikriteereistä (ks. taulukko 3) realisaatio. Realisaatiossa käsitellään innovaation tehokkuutta, vaikuttavuutta ja sen vaikutuksia ympäristöön ja käyttäjiin. Taulukosta 3 on jätetty alaluvussa 5.3.2 selitetyt kohdat pois.

7.2.1 Menetelmien ja ratkaisun toimivuus

Tutkimusongelman selvittämisessä on käytetty apuna aihepiirin kirjallisuuteen perustuvaa teoreettis-käsitteellistä tutkimusta ja suunnittelutieteellistä, konstruktiivista tutkimusta. Ilman aihepiiriin perustuvaa teoreettis-käsitteellistä tutkimusta kohteesta ei olisi pystytty hahmottamaan käsitteellisiä malleja, selityksiä ja rakenteita. Käsitteellisellä mallintamisella, selityksillä ja rakenteilla tarkoitetaan kohdealueen ja tutkittavana olevan systeemin sekä systeemin käsitteiden tunnistamista, suunnittelua ja määrittelyä (Tampereen yliopisto, 2013). Edellä mainitut asiat antavat tutkimuksen toteuttamiselle toimivan lähtökohdan, jota pystytään jalostamaan tutkimuksen edetessä toimivaksi kokonaisuudeksi. Mallien, selitysten ja rakenteiden avulla tutkimukseen pystyttiin innovoimaan ja kehittämään tarvittavat menetelmät tutkimuksen toteuttamiseksi. Toimivan kokonaisuuden suunnittelu oli paikoittain hankalaa, koska tietoa on saatavissa valtavasti. Olennaisen tiedon selvittäminen on tieteellisessä tutkimuksessa yksi tärkeä osa-alue. Olennaisen tiedon avulla tutkimus voidaan rajata tiettyyn osa-alueeseen ilman, että tutkimus laajenee liian suureksi. Ongelmana tutkimuksen teoriaosuudessa oli miettiä, kuinka sovellusten vertailutuloksista saa mahdollisimman täsmällisiä ja objektiivisia. Hankalaa oli myös innovoida sopivan laaja sovellus, joka kattaa tutkimuksessa määritellyt asiat ja vaatimuk-

set. Liian laajan sovelluksen kehittäminen olisi johtanut tutkimuksen kannalta liian suureen työpanokseen ja sen tutkiminen ei olisi ollut mieleistä. Lisäksi sovellusten koodiesimerkit tuli miettiä niin, että ne tuovat parhaiten esille sovellusten välisiä haittoja ja etuja.

Tutkimuksessa käytettävän suunnittelutieteellisen, konstruktiivisen tutkimuksen avulla voitiin toteuttaa ja testata käytännössä tutkimuksen teoreettis-käsitteellisessä osassa innovoituja menetelmiä. Tällainen lähestymistapa soveltuu hyvin tutkimukseen, jonka tarkoituksena on analysoida kahta web-sovellusta. Toimivilla sovelluksilla pystyttiin ajamaan aikaisemmissa luvuissa määriteltyjä testitapauksia ja vastaamaan tutkimusongelmaan. Ajetut testitapaukset antoivat konkreettisia tuloksia, joita ei olisi pystytty saamaan ilman toteutettuja sovelluksia. Tulosten mukaan kirjastojen, automaatiotyökalun ja sovelluskehyksen käyttö heikensi sovelluksen tehokkuutta sivun alustusvaiheessa ja kasvatti sen kokoa. Alustuksen jälkeen näitä menetelmiä käyttävä sovellus toimi kuitenkin tehokkaammin kuin verrokkisovellus. Menetelmien käyttö parantaa myös sovelluksen ylläpidettävyyttä ja kytkentää. Nämä asiat vaikuttavat sovelluksen kehitys- ja ylläpitokustannuksiin. Sovellusten toteutuksessa ei voida käyttää aina samoja menetelmiä, koska projektit ovat erityyppisiä ja erilaiset ihmiset arvostavat eri asioita. Projektit voivat olla pitkä- tai lyhytkestoisia; niissä voi työskennellä yksi tai sata henkilöä. Yksi heistä voi haluta sovellukselta äärimmäistä suorituskykyä, kun taas toiselle sovelluksen ylläpidettävyys on ratkaiseva tekijä. Jokaisen tulee itse miettiä, mitä menetelmiä sovelluksen toteutukseen tarvitaan.

Tutkimukseen valitut menetelmät ja sovellukset toimivat hyvin ja ne vastasivat tutkijan omia odotuksia. Tutkimuksesta olisi voinut jättää pois JavaScript-kirjastot ja keskittyä vain sovelluskehykseen ja automaatiotyökaluun. Kirjastot eivät olleet olennainen osa toteutetuissa sovelluksissa vaan enemmänkin mukava lisä tutkimukseen. Ilman kirjastojen käyttöä tutkimuksessa olisi jäänyt enemmän aikaa keskittyä kuvaamaan laajemmin käytettyä sovelluskehystä ja automaatiotyökalua. Tästä huolimatta tutkimuksessa saavutettiin toteutus- ja testausvaihe. Lukka (2001) on sanonut, että tällaisen vaiheen saavuttaminen on hyvin vaativa tehtävä, ja sitä voidaan pitää positiivisena signaalina kyseisen konstruktiivisen tutkimusprosessin onnistumisesta. Tutkimuksessa on lisäksi tehty kaikki voitava luotettavuuden ja tulosten varmistamiseksi.

7.2.2 Realisaatio

Realisaatio eli toteuma on Marchin ja Smithin (1995) esittämä artefakti realisaation arviointiin. Realisaation arvioinnissa otetaan huomioon tehokkuus, vaikuttavuus ja sen vaikutus ympäristöön ja käyttäjiin. Kriteerit painottavat innovaation hyödyllisyyttä ja sivuvaikutuksia (Järvinen & Järvinen, 2011). Tutkimuksessa toteutettua innovaatiota voidaan verrata aikaisempiin tutkimuksiin eli vanhaan käsitteistöön, malliin, metodiin tai realisaatioon. Tässä tapauksessa innovaatiota verrataan aikaisempien tutkimusten realisaatioon. Aikaisemmissa samankaltaisissa tutkimuksissa on verrattu kattavasti JavaScript-kirjastojen ja

sovelluskehysten tehokkuutta toisiinsa. Tutkimuksissa innovaation tehokkuutta on perusteltu mm. pakatuilla tiedostoilla, valmiiden työkalujen antamilla tuloksilla, heuristisilla ja luovilla arviointimenetelmillä ja tiettyjen funktioiden tehokkuudella. Tutkimuksessa innovaation tehokkuutta on mitattu automaatiotestien avulla eli mittaustuloksiin eivät vaikuta käyttäjän tekemät toimet. Lisäksi arvioinneissa on käytetty apuna heuristista ja luovaa arviointimenetelmää. Heuristinen ja luova arviointimenetelmä antaa käyttäjän aikaisemmasta kokemuksesta riippuen erilaisia tuloksia. Tällä tarkoitetaan sitä, että tutkimuksen ulkopuolinen henkilö voi aikaisemmasta kokemuksestaan riippuen olla eri mieltä tutkijan tekemien heuristisen ja luovien arvioiden kanssa. Kehitetty innovaatio ei ole kovin vaikuttava, koska siinä käytetyt menetelmät ja työkalut ovat jo aikaisemmin kehittyjä. Innovaatio antaa tästä huolimatta selkeät ja yhtenäiset menetelmät toteuttaa kahden systeemin vertailuun perustuva tutkimus ja vaikuttaa näin omalta osaltaan ympäristöön sekä käyttäjiin.

7.3 Tulosten hyödyntäminen ja jatkokehitysmahdollisuudet

Seuraavaksi kerrotaan, kuinka tämän tutkimuksen tuloksia voidaan hyödyntää esimerkiksi yrityksissä tai muissa yhteyksissä. Tämän jälkeen kerrotaan tutkimuksen jatkokehitysmahdollisuuksista.

7.3.1 Tulosten hyödyntäminen

Tutkimuksen avulla yritykset tai muut alan henkilöt saavat suuntaa antavaa tietoa siitä, onko sovelluskehysten, kirjastojen tai automaatiotyökalujen käyttö järkevää erilaisissa web-sovelluksissa. Tutkimus on toteutettu hyvin käytännönläheisesti, joten siitä hyötyvät lisäksi web-kehityksestä tai JavaScriptistä vähemmän tietävät henkilöt. Menetelmien käytön tehokkuus ja niiden tähän tutkimukseen tuoma hyöty voidaan osoittaa konkreettisesti tutkimuksen tulosten avulla. Tutkimuksessa saatuja tuloksia ei voida kuitenkaan yleistää kahden, varsin pienen esimerkkisovelluksen perusteella.

Liitteissä kuvatut hakemistorakenteet ja ohjelmakoodit auttavat lukijoita toteuttamaan asiakaspään web-sovelluksen tutkimuksessa kuvatuilla tekniikoilla ja menetelmillä. Web-sovellusten ollessa itsenäisiä kokonaisuuksia voi jokainen toteuttaa palvelinpään toteutuksen haluamallaan tavalla. Tutkimuksessa rakennettuja innovaatioita voidaan käyttää hyödyksi myös muissa tutkimuksissa uuden innovaation ja teorian luonnissa. Tutkimuksen tärkein tehtävä on herättää ennen kaikkia ajatuksia JavaScriptin nopeasta kehityksestä, tulevaisuudesta ja mahdollisista tekniikoista, joita voidaan käyttää nykyaikaisissa websovelluksissa.

7.3.2 Jatkokehitysmahdollisuudet

Tämä tutkimus osoitti muun muassa, että JavaScriptiä voidaan käyttää apuna asiakaspään web-sovelluksissa. JavaScriptiä voidaan hyödyntää nykypäivänä myös palvelinpäässä Node.js:n ansiosta. Ehdotan tälle tutkimukselle jatkokehitysmahdollisuudeksi kahden palvelinpään toteutuksen vertaamista toisiinsa. Palvelinpään sovellusten toteutuksissa voidaan käyttää PHP:tä ja Node.js:ää. Sovelluksia voidaan verrata toisiinsa tutkimuksen menetelmiä ja innovaatioita apuna käyttäen. PHP:n ja Node.js:n vertailu osoittaisi, ovatko vanhat ja perinteiset menetelmät tehokkaampia palvelinpään toteutuksessa kuin Node.js. Menetelmiä vertailtaessa asiakaspään toteutuksessa on mahdollista käyttää tutkimuksessa (ks. liite 2 ja 3) toteutettuja web-sovelluksia. Sovelluksista on mahdollista valita itselleen parhaiten toimivia toteutus.

7.4 Teoreettinen kontribuutio

Tutkimusprosessin viimeisessä vaiheessa tutkijan tulee tunnistaa ja analysoida teoreettinen kontribuutio (ks. alaluku 4.2). Tutkimuksessa teoreettinen kontribuutio saavutettiin kahdessa mielessä: a) konstruktio on itsessään uusi, b) kehitetyn uuden teoreettisen kontribuution takana on riippuvuussuhde aikaisempiin tutkimuksiin. Konstruktiota voidaan pitää itsessään uutena, jos kehitetyn uuden konstruktion on todettu toimivan sille kehitetyssä case-ympäristössä. Tässä tapauksessa konstruktio tuottaa luonnollisen tietämyksen lisän aiempaan kirjallisuuteen (Lukka, 2001.). Koska tällaisessa tilanteessa konstruktiivinen tutkimusprosessi on luonteeltaan luova ja heuristinen, voi kysymys kääntyä helposti muotoon: kun olemme nyt saaneet kehitetyksi uuden konstruktion, joka näyttää toimivan ainakin tutkitussa case-ympäristössä, mitkä ovat ne keino-lopputulos-suhteet sekä rakenteelliset tai prosessuaaliset riippuvuussuhteet, jotka paljastuvat tässä uudessa todellisuudessa (Lukka, 2001). Tutkimuksessa kehitettyyn konstruktioon tulee suhtautua uutena, koska sen avulla pystyttiin saavuttamaan ennalta määritelty tavoite ja testaamaan se sille kehitetyssä case-ympäristössä tutkimuksessa määriteltyjen menetelmien (*keinojen*) avulla. Edellä mainittu tavoite on määritelty tutkimuksen tutkimusongelmassa ja sen selvittämiseksi määritetyt keinot on kuvattu luvussa 5. Tutkimuksesta saadut lopputulokset on kerrottu luvussa 6.

Uuden konstruktion rakentamisyrityksen ja sen toimivuuden testaamisen lisäksi konstruktiivinen tutkimusprojekti voi olla osa olemassa olevan tutkimuksen rakenteita ja prosesseja koskevan teoreettisen tietämyksen jalostamista, soveltamista, testaamista ja kehittämistä tai vaatimattomimpana vaihtoehtona – sen havainnollistamista (Keating & Harris, 1995; Lukka, 1991). Tässä tapauksessa tutkimuksessa kehitetyn uuden teoreettisen kontribuution takana on riippuvuussuhde aikaisempiin tutkimuksiin. Tutkimuksessa on kyse aikaisemman *teorian jalostuksesta* (*refinement*), joka on luultavasti konstruktiivisen tutkimus-

projektin tyypillisin tulos (Lukka, 2001). Alaluvussa 7.1 sanottiin, että aikaisemmat tutkimukset ja teoriat antoivat hyvän pohjan tutkimuksen toteuttamiseen, mutta eivät suoraan vastanneet tutkimuksen tutkimusongelmaan. Aikaisempien tutkimusten tutkimustulokset ja teoriat on pystytty tulkitsemalla ja soveltamalla jalostamaan tämän tutkimuksen uudeksi teoriaksi ja malliksi. Tutkimuksessa kehitettyjen keinojen ja saavutettujen lopputulosten lisäksi tutkimuksessa on kehitetty kaksi sovellusta, joiden avulla määriteltyjä keinoja voidaan testata. Kuten alaluvussa 7.3.1 todettiin, ei tutkimuksessa saatuja tuloksia voida kuitenkaan yleistää kahden, varsin pienen esimerkkisovellusten perusteella.

8 YHTEENVETO

JavaScript on saanut kritiikkiä hitaudesta ja lisäksi se on kärsinyt maineesta lelukielenä, joka on hyödyllinen vain suhteellisen yksinkertaisiin tehtäviin (Mikkonen & Taivalsaari, 2007). Nykyään ohjelmistokehittäjät ovat ottaneet JavaScriptin käyttöön sen yksinkertaisuuden, joustavuuden, monipuolisten lisäosien ja tehokkaan suorituskyvyn ansiosta. Uusimpien tekniikoiden avulla on mahdollista siirtää sovelluksen esityslogiikka palvelinpäältä käyttäjän selaimeen. Tämä menetelmä auttaa hajauttamaan sovelluksen prosessointia käyttäjien laitteistoille. JavaScriptin avulla saavutetaan myös monia muita hyötyjä web-kehityksessä, koska se on todella dynaaminen ohjelmointikieli. Yksi tapa saavuttaa näitä hyötyjä on käyttää AJAX-kutsuja. AJAXin avulla sivulta voidaan päivittää haluttu määrä tietoa kerrallaan, mikä nopeuttaa sovelluksen toimintaa ja säästää käyttäjän aikaa ja hermoja. Crawford (2014) on sanonut, että JavaScriptillä tehtäessä voidaan säästää myös ohjelmistokehittäjien henkisiä voimavaroja, koska kehittäjien ei tarvitse opetella montaa teknologiaa. JavaScriptin avulla esimerkiksi web-sivun toiminnallisuudet voidaan kirjoittaa kokonaan yhdellä kielellä, JavaScriptillä.

Ohjelmistokehittäjät voivat käyttää JavaScriptin tukena erilaisia kirjastoja, sovelluskehyksiä ja automaatiotyökaluja. Menetelmien käyttötarkoitus on pohjimmiltaan sama eli parantaa sovelluksen ylläpitoa ja suorituskykyä. Erilaisia menetelmiä valittaessa on muistettava, että sovellusten toteutuksessa ei voi käyttää aina samoja menetelmiä. Projektit ovat yleensä erityyppisiä ja eri ihmiset arvostavat erilaisia asioita. Projektit voivat olla pitkä- tai lyhytkestoisia, niissä voi työskennellä yksi tai sata henkilöä ja yksi heistä voi haluta sovellukselta äärimmäistä suorituskykyä, kun taas toiselle sovelluksen ylläpidettävyys on ratkaiseva tekijä. Jokaisen tulee miettiä itse, mitä menetelmiä sovelluksen toteutukseen tarvitaan.

JavaScriptiä käyttäessä ohjelmakoodin hahmottaminen ja ylläpidettävyys voivat muuttua haasteellisiksi, kun sovelluslogiikka kasvaa monimutkaiseksi kokonaisuudeksi. Kehityksen ja ylläpidon helpottamiseksi web-sovelluksen eri vastuualueet on syytä erotella toisistaan. Vastuualueiden jakaminen auttaa ylläpidon lisäksi testauksessa, virheiden etsimisessä, laajennettavuudessa sekä

vähentää ohjelmistokoodin toistojen määrää ja mahdollistaa samanaikaisen kehitystyön. Vastuualueiden jakamisen voi toteuttaa web-sovellukseen itse tai käyttää valmiita sovelluskehyksiä, jotka noudattavat sovelluskehyksestä riippuen erilaisia esitysmalleja.

Tutkimuksen tavoitteena oli täydentää aikaisempia tutkimustuloksia toteuttamalla erilaisilla menetelmillä kaksi toiminnallisuudeltaan ja ulkoasultaan samanlaista web-sovellusta (ks. luku 5). Sovellusten toteutuksessa käytettävien menetelmien pääpaino oli JavaScriptissä. Ensimmäinen versio (Trick) käytti apuna alaluvussa 3.3 valittuja kirjastoja, automaatiotyökalua ja sovelluskehystä. Toisessa versiossa (Trick2) ei käytetty tällaisia apuvälineitä. Näiden web-sovellusten tehokkuutta, ylläpidettävyyttä, kytkentää ja koko mitattiin standardointijärjestöjen määrittelemien laatustandardien avulla.

Tutkimuksen tavoitetilasta jäätiin hiukan pienien semanttisten erojen vuoksi (ks. alaluku 6.1). Sovellukset pystyvät tästä huolimatta täyttämään niille asetetut vaatimukset hyvin. Nykyaikaisten web-sovellusten toteutus pystyttiin tekemään kohtuullisessa ajassa ja sovellusten avulla voidaan nauhoittaa ja tallentaa käyttäjän kirjaamia erilaisia työtehtäviä muistiin. Trick2-sovelluksen kehitykseen kulunut kokonaisaika oli 20,5 tuntia eli 5 tuntia vähemmän kuin Trick-sovellukseen käytetty kokonaisaika. Itse kirjoitetun ohjelmakoodin osalta Trick2:ssa oli 897 riviä koodia, mikä on 4,6 % enemmän kuin Trick-sovelluksessa. Trick2 latautui sivun alustusvaiheessa kolme kertaa niin nopeammin kuin molemmat Trick-sovellukset. Tämä johtui mm. sovellukseen ladattavien JavaScript-tiedostojen koosta ja määrästä. Trick-sovellus suoriutui tästä huolimatta automaatiotestistä nopeammin kuin Trick2. Esitysmallin (MVC-malli) käyttö helpotti sovellusten toteutusvaiheissa virheiden etsimistä ja tietyn vastuualueen jakamista omaksi kokonaisuudeksi sekä vähensi ohjelmakoodin toistojen määrää. Kytkennän osalta kumpikaan sovellus ei onnistunut saavuttamaan löyhää kytkentää. Lopuksi todettiin, että sovelluksen koko vaikuttaa yhtenä osana sovellusten latausnopeuteen käyttäjän selaimessa.

Tuloksena saatiin tieto, että valittujen kirjastojen, automaatiotyökalun ja sovelluskehyksen käyttö heikensi sovelluksen tehokkuutta sivun alustusvaiheessa ja kasvatti sen kokoa. Alustuksen jälkeen näitä menetelmiä käyttävä sovellus toimi kuitenkin tehokkaammin kuin verrokkisovellus. Menetelmien käyttö parantaa myös sovelluksen ylläpidettävyyttä ja kytkentää. Nämä asiat vaikuttavat sovelluksen kehitys- ja ylläpitokustannuksiin. Mikkonen ja Taivalsaari (2007) ovat todenneet, että kehittyneiden ja monipuolisten kirjastojen sekä ominaisuuksien ansiosta negatiivinen käsitys JavaScriptistä paranee jatkuvasti.

LÄHTEET

Aken, J. E. v. (2004). Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules. *Journal of Management Studies* 41 (2), 219-246.

Allamaraju, S. (2010). *RESTful Web Services Cookbook: solutions for improving Scalability and Simplicity.* Sebastopol: O'Reilly Media, Inc.

AngularJS (2014). What are Scopes? Haettu 4.1.2015 osoitteesta https://docs.angularjs.org/guide/scope.

Berard, E. (1993). *Essays on Object-Oriented Software Engineering.* (Volume 1. painos). Michigan: Prentice Hall.

Bosworth, A. & McKusick, M. (2003). A conversation with Adam Bosworth. *ACM Queue Magazine*.

Botella, P., Burgués, X., Carvallo, J., Franch, X., Grau, G., Marco, J. & Quer, C. (2004). ISO/IEC 9126 in practice: what do we need to know? Teoksessa *Proceedings of the First Software Measurement European Forum (SMEF)*.

Bouvier, D. J. (1995). The state of HTML. *ACM SIGICE Bulletin* 21 (2), 8-13.

Burbeck, S. (1992). Applications programming in smalltalk-80 (tm): How to use model-view-controller (MVC). *Smalltalk-80 v2*5.

Cascaval, C., Fowler, S., Montesinos-Ortego, P., Piekarski, W., Reshadi, M., Robatmili, B., Weber, M. & Bhavsar, V. (2013). Zoomm: a parallel web browser engine for multicore mobile devices. Teoksessa *ACM SIGPLAN Notices* (s. 271-280). ACM.

Crawford, S. (2014). Four Hot Web Development Trends Learned at Conferences. Haettu 22.9.2014 osoitteesta http://www.oreillyschool.com/2014/04/web-development-trends/.

Dooley, J. (2011). *Software Development and Professional Practice.* New York: Apress.

ECMA (2011). European Computer Manufacturers Association. *ECMAScript Language Specification*.

Fielding, J. (2014). *Beginning Responsive Web Design with HTML5 and CSS3.* New York: Apress.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Haettu 26.11.2014 osoitteesta https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). *Hypertext Transfer Protocol–HTTP/1.1*.

Flanagan, D. (2011). *JavaScript: the Definitive Guide.* (6. painos). Sebastopol: O'Reilly Media, Inc.

Fling, B. (2009). *Mobile Design and Development: Practical concepts and techniques for creating mobile sites and web apps.* Sebastopol: O'Reilly Media, Inc.

Gao, L., Zhang, C. & Sun, L. (2011). RESTful web of things API in sharing sensor data. Teoksessa *Internet Technology and Applications (iTAP), 2011 International Conference on* (s. 1-4). IEEE.

Gofore (2013). Yhden sivun web-sovellukset tulevat, oletko valmis? Haettu 9.11.2014 osoitteesta http://gofore.com/ohjelmistokehitys/yhden-sivun-web-sovellukset-tulevat-oletko-valmis/.

Google (2014). Google Trends - Web Search interest - Worldwide, 2004 - present. Haettu 11.11.2014 osoitteesta http://www.google.com/trends/explore?hl=fi#cat=0-5-31&q=ember%20js%2C%20angular%20js%2C%20backbone%20js&date=1%2F2010%2060m&cmpt=q.

Grunt (2014). Grunt: The JavaScript Task Runner. Haettu 10.11.2014 osoitteesta http://gruntjs.com/.

Hales, W. (2012). *HTML5 and JavaScript Web Apps*. Sebastopol: O'Reilly Media, Inc.

iMacros (2014). Browser Automation, Data Extraction and Web Testing Software. Haettu 25.11.2014 osoitteesta http://imacros.net/.

ISO/IEC (2000). Information technology - Software product quality - Part 1: Quality model FDIS 9126-1. *IEEE Software* , 1-25.

Jaggavarapu, M. (2012). Presentation Patterns : MVC, MVP, PM, MVVM. Haettu 6.11.2014 osoitteesta http://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/.

jQuery (2014). jQuery: The Write Less, Do More, JavaScript Library. Haettu 10.11.2014 osoitteesta http://jquery.com/.

Juels, A., Jakobsson, M. & Jagatic, T. (2006). Cache Cookies for Browser Authentication (extended abstract). Haettu 27.11.2014 osoitteesta http://markus-jakobsson.com/papers/jakobsson-oakland06.pdf.

Jyväskylän yliopisto (2014). Teoreettinen tutkimus. Haettu 27.11.2014 osoitteesta https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/tutkimusstrategiat/teoreettinen-tutkimus.

Järvinen, P. & Järvinen, A. (2011). *Tutkimustyön metodeista.* Tampere: Opinpajan kirja.

Jørgensen, M. (1999). Software quality measurement. *Advances in Engineering Software* 30 (12), 907-912.

Kasten, E. (1995). HTML. *Linux Journal* 1995 (15).

Kaye, D. (2003). *Loosely Coupled: The Missing Pieces of Web Services*. United States of America: RDS Press.

Keating, P. J. & Harris, E. (1995). *A framework for classifying and evaluating the theoretical contributions of case research in management accounting.*

Kerr, D. (2013). An introduction to MVC frameworks. Haettu 6.1.2015 osoitteesta http://www.scottlogic.com/blog/2013/12/06/JavaScript-MVC-frameworks.html.

Kienle, H. M. (2010). It's about time to take JavaScript (more) seriously. *Software, IEEE* 27 (3), 60-62.

King, R. S. (2011). The top 10 programming languages. *Spectrum, IEEE* 48 (10), 84-84.

Korpela, J. (2011). *HTML5–Uudet ominaisuudet.* Jyväskylä: Docendo.

Kosonen, P., Peltomäki, J. & Silander, S. (2005). *Java 2: ohjelmoinnin peruskirja.* Jyväskylä: Docendo.

Kovanen, P. (2013). Ohjelmointikielet vertailussa. Haettu 25.9.2014 osoitteesta http://67.prosenttia.fi/2013/09/12/ohjelmointikielet-vertailussa/.

Krithinakis, A., Athanasopoulos, E. & Markatos, E. P. (2010). Isolating JavaScript in dynamic code environments. Teoksessa *Proceedings of the 2010 Workshop on Analysis and Programming Languages for Web Applications and Cloud Applications* (s. 45-49). ACM.

Kähkönen, J. (2014). MVC-malliin perustuvien Javascript-sovelluskehysten vertailua. Tietojenkäsittelyopin pro gradu -tutkielma. Tampereen yliopisto.

Lano, K. & Haughton, H. (1992). Software maintenance research and applications, In Leponiemi (Ed.), NordData'92 Precedings. Teoksessa (s. 123-143).

Lebresne, S., Richards, G., Östlund, J., Wrigstad, T. & Vitek, J. (2009). Understanding the dynamics of JavaScript. Teoksessa *Proceedings for the 1st workshop on Script to Program Evolution* (s. 30-33). ACM.

Lerner, A. (2013). *ng-book: The Complete Book on AngularJS.* Fullstack.io.

Li, X. & Xue, Y. (2014). A survey on server-side approaches to securing web applications. *ACM Computing Surveys (CSUR)* 46 (4), 1-29.

Lientz, B. P. (1983). Issues in software maintenance. *ACM Computing Surveys (CSUR)* 15 (3), 271-278.

Lin, B., Chen, Y., Chen, X. & Yu, Y. (2012). Comparison between JSON and XML in Applications Based on AJAX. Teoksessa *Computer Science & Service System (CSSS), 2012 International Conference on* (s. 1174-1177). IEEE.

Lin, Z., Wu, J., Zhang, Q. & Zhou, H. (2008). Research on web applications using ajax new technologies. Teoksessa *MultiMedia and Information Technology, 2008. MMIT'08. International Conference on* (s. 139-142). IEEE.

Loui, R. P. (2008). In praise of scripting: Real programming pragmatism. *Computer* 41 (7), 22-26.

Lukka, K. (2001). Konstruktiivinen tutkimusote. Haettu 17.11.2014 osoitteesta http://metodix.wordpress.com/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Lukka, K. (1991). Laskentatoimen tutkimuksen epistemologiset perusteet. *Liiketaloudellinen aikakauskirja* 40 (2), 161-186.

MacCaw, A. (2011). *JavaScript Web Applications.* Sebastopol: O'Reilly Media, Inc.

Maras, J., Carlson, J. & Crnkovic, I. (2011). Client-side web application slicing. Teoksessa *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering* (s. 504-507). IEEE Computer Society.

March, S. T. & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems* 15 (4), 251-266.

McFarland, D. S. (2011). *Javascript & jQuery: The Missing Manual.* (2. painos). Sebastopol: O'Reilly Media, Inc.

Meijer, E. & Drayton, P. (2004). Static typing where possible, dynamic typing when needed: The end of the cold war between programming languages.

Teoksessa *Intended for submission to the Revival of Dynamic Languages. Microsoft Corporation* (s. 1-6). Citeseer.

Mikkonen, T. & Taivalsaari, A. (2007). Using JavaScript as a real programming language (s. 1-17). Sun Microsystems, Inc.

Miller, M.S. (2010). Future of JavaScript. Haettu 3.11.2014 osoitteesta https://gist.github.com/paulmillr/1208618.

Modernizr (2014). Modernizr: front-end development done right. Haettu 10.11.2014 osoitteesta http://modernizr.com/.

Morin, R. & Brow, V. (1999). Scripting languages. A cross-os perspective. Haettu 2.11.2014 osoitteesta http://www.mactech.com/articles/mactech/Vol.15/15.09/ScriptingLanguages/index.html.

Mozilla Foundation (2014). Firebug, web development evolved. Haettu 18.12.2014 osoitteesta http://getfirebug.com/.

Mäkelä, M. (2010). *JavaScript-kehityskirjastojen vertailu.* Ammattikorkeakoulututkinnon opinnäytetyö. Hämeen ammattikorkeakoulu.

Nicol, G., Wood, L., Champion, M. & Byrne, S. (2001). Document Object Model (DOM) level 3 core specification. W3C Working Draft 13 September 2001. 1-146.

Ocariza, F., Pattabiraman, K. & Zorn, B. (2011). JavaScript errors in the wild: An empirical study. Teoksessa *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on* (s. 100-109). IEEE.

Open Source Initiative The MIT License (MIT). Haettu 11.11.2014 osoitteesta http://opensource.org/licenses/MIT.

Ortiz, A. (2010). Building server-side web language processors. Teoksessa *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (s. 2-6). ACM.

Osmani, A. (2012a). Journey through the JavaScript MVC jungle. Haettu 6.11.2014 osoitteesta http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/.

Osmani, A. (2012b). *Learning JavaScript Design Patterns.* Sebastopol: O'Reilly Media, Inc.

Peltomäki, J. & Nykänen, O. (2006). *Web-selainohjelmointi.* Jyväskylä: Docendo Oy.

Pihlajaniemi, J. (2012). *REST-pohjaisen web-rajapinnan kehittäminen*. Ammattikorkeakoulututkinnon opinnäytetyö. Metropolia ammattikorkeakoulu.

Pillora, J. (2014). *Getting Started with Grunt: The JavaScript Task Runner.* United States of America: Packt Publishing Ltd.

Reenskaug, T. (1979). Models-views-controllers. *Technical note, Xerox PARC* 32, 55.

Reis, C. & Gribble, S. D. (2009). Isolating web programs in modern browser architectures. Teoksessa *Proceedings of the 4th ACM European Conference on Computer Systems* (s. 219-232). ACM.

Severance, C. (2012a). Java script: Designing a language in 10 days. *Computer* 45 (2), 0007-8.

Severance, C. (2012b). Discovering JavaScript Object Notation. *Computer* 45 (4), 6-8.

Souders, S. (2008). High-performance web sites. *Communications of the ACM* 51 (12), 36-41.

Stanley, E. P. (2010). *Project Management for Dummies.* (3. painos). Wiley Publishing, Inc., Indianapolis, Indiana: Wiley Publishing, Inc.

SunSpider (2013). SunSpider 1.0.2 JavaScript Benchmark. Haettu 25.11.2014 osoitteesta https://www.webkit.org/perf/sunspider/sunspider.html.

Swiech, M. & Dinda, P. (2013). Making javascript better by making it even slower. Teoksessa *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on* (s. 70-79). IEEE.

Tampereen yliopisto (2013). Käsitteellinen mallintaminen. Haettu 15.1.2015 osoitteesta http://www.uta.fi/sis/tie/kmal/index.html.

TechTerms.com (2014a). Server. Haettu 11.10.2014 osoitteesta http://www.techterms.com/definition/server.

TechTerms.com (2014b). Web Browser. Haettu 22.9.2014 osoitteesta http://www.techterms.com/definition/web_browser.

TechTerms.com (2009a). Dynamic Website. Haettu 22.9.2014 osoitteesta http://www.techterms.com/definition/dynamicwebsite.

TechTerms.com (2009b). User Interface. Haettu 2.10.2014 osoitteesta http://www.techterms.com/definition/user_interface.

Tesarik, J., Dolezal, L. & Kollmann, C. (2008). User interface design practices in simple single page web applications. Teoksessa *Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the* (s. 223-228). IEEE.

TIOBE Software (2014). TIOBE Index for October 2014. Haettu 3.11.2014 osoitteesta http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

Tiwari, D. & Solihin, D. (2012). Architectural characterization and similarity analysis of sunspider and Google's V8 Javascript benchmarks. Teoksessa *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on* (s. 221-232). IEEE.

Valkama, S. (2014). *JavaScript nykyaikaisessa web-kehityksessä.* Ammattikorkeakoulututkinnon opinnäytetyö. Metropolia ammattikorkeakoulu.

W3C (2004). Web Services Architecture. Haettu 15.10.2014 osoitteesta http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

W3Techs (2014). Usage statistics and market share of JQuery for websites. Haettu 10.11.2014 osoitteesta http://w3techs.com/technologies/details/js-jquery/all/all.

Watson, A. (2012). *Learning Modernizr.* United Kingdom: Packt Publishing Ltd.

Wilton, P. & McPeak, J. (2010). *Beginning JavaScript.* (4. painos). Wiley Publishing, Inc., Indianapolis, Indiana: John Wiley & Sons.

Wright, T. (2012). *Learning JavaScript: A Hands-On Guide to the Fundamentals of Modern JavaScript.* United States: Addison-Wesley.

Yoon, J. (2014). Software Metrics. Haettu 26.11.2014 osoitteesta https://wiki.cites.illinois.edu/wiki/display/cs242sp14/Software+Metrics.

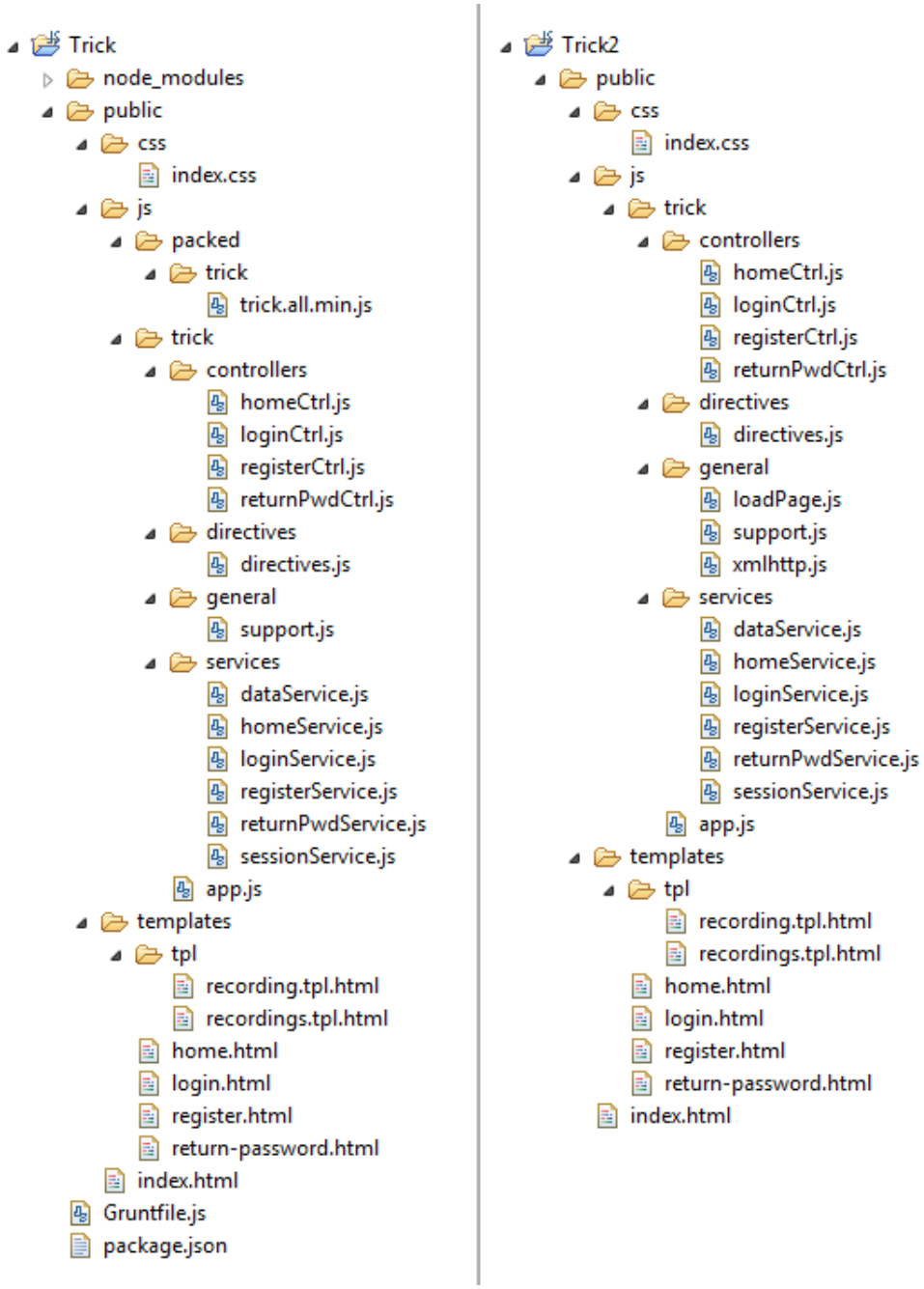
Yu, D., Chander, A., Inamura, H. & Serikov, I. (2008). Better abstractions for secure server-side scripting. Teoksessa *Proceedings of the 17th international conference on World Wide Web* (s. 507-516). ACM.

Yue, C. & Wang, H. (2013). A measurement study of insecure JavaScript practices on the Web. *ACM Transactions on the Web (TWEB)* 7 (2), 7.

Zakas, N. C. (2012). *Professional JavaScript for Web Developers.* (3. painos). Indianapolis, Indiana: John Wiley & Sons.

Zerr, J. (2013). Using Firebug to figure out why your website is running slow. Haettu 6.1.2015 osoitteesta http://www.jeremyzerr.com/using-firebug-figure-out-why-your-website-running-slow.

LIITE 1 SOVELLUSTEN HAKEMISTORAKENNE



LIITE 2 TRICK-LÄHDEKOODIT

index.css

```
body {
    padding-top: 40px; padding-bottom: 40px; background-color: #eee;
}

.container,.container .js-disabled,.container .alerts-style {
    max-width: 500px; padding: 15px; margin: 0 auto;
}

.container .js-disabled,.container .alerts-style {
    border: solid 1px gray; color: green;
}

.container .alerts-nice-style,.container .alerts-danger-style {
    max-width: 500px; padding: 15px; margin: 0 auto; margin-bottom: 10px;   border: solid 1px gray;
}

.container .alerts-nice-style {
    color: green;
}

.container .alerts-danger-style {
    color: red;
}

.inside-container {
    margin-left: 5px; margin-right: 5px; margin-bottom: 5px; margin-top: 20px;
}

.form h2,.form button[type="submit"],.form a[type="button"] {
    margin-bottom: 10px; margin-top: 10px;
}

.form .checkbox {
    margin-top: 10px; font-weight: normal;
}

.form .form-control {
    position: relative; height: auto; -webkit-box-sizing: border-box; -moz- box-sizing: border-box; box-sizing: border-
    box; padding: 10px; font-size: 16px;
}

.form .form-control:focus {
    z-index: 2;
}

.form input[type="email"] {
    margin-bottom: -1px; border-bottom-right-radius: 0; border-bottom-left-radius: 0;
}

.form input[type="password"] {
    margin-top: -1px; border-top-left-radius: 0; border-top-right-radius: 0;
}
```

homeCtrl.js

```
'use strict';

app.controller('homeCtrl', ['$scope', '$cookieStore', 'loginService', 'homeService', 'dataService', function($scope,
$cookieStore, loginService, homeService, dataService) {

    /* Määritellään tehtävän tyypit. */
    $scope.tasks = [{ name: "Ohjelmointi" }, { name: "Palaveri" }, { name: "Sekalaista" }];
    dataService.getData($scope); // Haetaan käyttäjän kaikki nauhoitukset.
    $scope.timerRunning = false; // Lukitaan oletuksena "Pysäytä nauhoitus" -painike.
    $scope.msgError = "";
    $scope.logout = function(){
        loginService.logout($cookieStore); // Kutsutaan loginServicen logout-funktiota.
    };
    $scope.start = function(myRecording) {
        homeService.start(myRecording, $scope);
        $scope.timerRunning = true;
    };
    $scope.stop = function() {
        homeService.stop($scope);
        $scope.timerRunning = false;
    };
    $scope.deleteOne = function(id, index) {
        homeService.deleteOne(id, $scope, index);
    };
    $scope.deleteAll = function(allRecordings) {
        homeService.deleteAll(allRecordings, $scope);
    };
}]);
```

loginCtrl.js

```
app.controller('LoginCtrl', ['$scope', '$cookieStore', 'loginService', function($scope, $cookieStore, loginService) {
    $scope.msgtxt = "";
    $scope.login = function(data) {
        loginService.login(data, $scope, $cookieStore);
    };
}]);
```

registerCtrl.js

```
'use strict';

app.controller('RegisterCtrl', ['$scope', 'registerService', function($scope, registerService) {
    $scope.register = function(data) {
        registerService.register(data, $scope);
    };
}]);
```

returnPwdCtrl.js

```
'use strict';

app.controller('ReturnPwdCtrl', ['$scope', 'returnPwdService', function($scope, returnPwdService) {
    $scope.returnPwd = function(data) {
        returnPwdService.returnPwd(data, $scope);
    };
}]);
```

directives.js

```
'use strict';

app.directive('recordingDirective', function() {
    return { templateUrl: 'templates/tpl/recording.tpl.html' };
});

app.directive('recordingsDirective', function() {
    return { templateUrl: 'templates/tpl/recordings.tpl.html' };
});
```

support.js

```
'use strict';

$(document).ready(function() {

    /* Testataan selaimen tukea erilaisille ominaisuuksille Modernizr-kirjaston avulla. */
    var newSupport = new support(["window.JSON", "Modernizr.input.placeholder", "Modernizr.sessionstorage"]);
    newSupport.addJsCheck("html", ".js", ".js-disabled"); // Tarkistetaan JavaScript-tuki.
    newSupport.addText("Huomautus! Seuraavat ominaisuudet eivät ole käytössä:", "Sovellus ei toimi oikein.");
    newSupport.createAlert();
});

function support(features) {
    this.addJsCheck = function(class1, class2, removeElement) {
        var boolean = $(class1).hasClass(class2);
            if(!boolean) {
                $(removeElement).remove();
            }
    };

    this.addText = function(alertText1, alertText2) {
    this.alertText1 = alertText1;
    this.alertText2 = alertText2;
};

this.createAlert = function() {
    if(features instanceof Array) {
        var newFeatures = "";
        for (var i = 0; i < features.length; i++) {
            if(!eval(features[i])) {
                newFeatures += splitText(features[i]) + ", ";
            }
        } if(newFeatures !== "") {
        $( ".alerts" ).text(this.alertText1 + " " + newFeatures.replace(/,\s*$/, ".") + " " +
        this.alertText2).addClass("alerts- style");
    }
} else {
       if(!eval(features)) {
            $( ".alerts" ).text(this.alertText1 + " " + splitText(features) + " " + this.alertText2).addClass("alerts-style");
       }
    }
};

var splitText = function(text) {
    var strs = text.split(".");
    return strs.length === 3 ? strs[2] : strs[1];
};
}
```

dataService.js

```
'use strict';

app.factory('dataService', ['$http', 'sessionService', function($http, sessionService) {

    var getData = function(scope) {
        var usrEmail = sessionService.get('email'); // Haetaan sessiosta käyttäjän tiedot.

        /* Haetaan tietokannasta käyttäjän kaikki nauhoitukset sähköpostiosoitteen perusteella. */
        var $promise = $http.post('http://backend.localhost/getData.php', {email: usrEmail});

        $promise.then(function(msg) {
            scope.oldTasks = msg.data;
            scope.allRecordings = msg.data;
        });
    };

    return {
        getData: getData
    };
}]);
```

homeService.js

```
'use strict';

app.factory('homeService', ['$http', 'sessionService', function($http, sessionService) {
    this.uniquerNumber = "";
    var usrEmail = sessionService.get('email'); // Haetaan sessiosta käyttäjän tiedot.
    var start = function(myRecording, scope) {
        this.uniquerNumber = Math.random().toString(36).substring(7);

        /* Kun käyttäjä aloittaa tai jatkaa nauhoitusta, niin palvelinpäässä tallennetaan seuraavat tiedot tietokantaan:
         *
         * nauhoituksen aloitusaika, käyttäjän sähköpostiosoite, tehtävän tyyppi,
         * nauhoituksen yksilöivä numerosarja, tehtävän kuvaus ja nauhoituksen tila. */
        var $promise = $http.post('http://backend.localhost/recording.php', {myRecording: myRecording, email:
        usrEmail, unique: this.uniquerNumber});

        $promise.then(function(msg) {
            if(msg.data != 'success') {
                scope.msgError = "Tarvittavat tiedot puuttuvat! Yritä uudelleen.";
                scope.timerRunning = false;
                $(".alerts-danger").addClass("alerts-danger-style");
            }
        });
};

var stop = function(scope) {

    /* Tallennetaan nauhoituksen kestoon kulunut kokonaisaika, ja palautetaan käyttäjän kaikki nauhoitukset. */
    var $promise = $http.post('http://backend.localhost/stopRecording.php', {email: usrEmail, unique:
    this.uniquerNumber});

    $promise.then(function(msg) {
        var returnValues = msg.data;
        if(typeof(returnValues) == "undefined") {
            scope.msgError = "Nauhoitusta ei voida lopettaa. Yritä myöhemmin uudelleen.";
            $(".alerts-danger").addClass("alerts-danger-style");
        } else {
```

```
            scope.oldTasks = returnValues;
            scope.allRecordings = msg.data;
        }
    });
};

var deleteOne = function(id, scope, index) {

    /* Poistetaan käyttäjän nauhoitus tietokannasta id:n perusteella. */
    var $promise = $http.post('http://backend.localhost/deleteOne.php', {id: id});

    $promise.then(function(msg) {
        if(msg.data == 'success') {
            scope.allRecordings.splice(index, 1); // Poistetaan nauhoitus käyttöliittymästä.
        }
    });
};

var deleteAll = function(allRecordings, scope) {

    /* Poistetaan käyttäjän kaikki nauhoitukset tietokannasta käyttäjän sähköpostiosoitteen perusteella. */
    var $promise = $http.post('http://backend.localhost/deleteAll.php', {email: usrEmail});

    $promise.then(function(msg) {
        if(msg.data == 'success') {
            scope.allRecordings.splice(0, allRecordings.length); // Poistetaan nauhoitukset käyttöliittymästä.
        }
    });
};
    return { deleteAll: deleteAll, deleteOne: deleteOne, start: start, stop: stop };
}]);
```

loginService.js

```
'use strict';

app.factory('loginService', ['$http', '$location', 'sessionService', function($http, $location, sessionService) {
    return {
        login: function(data, scope, cookieStore) {

        /* Haetaan tietokannasta käyttäjätiedot ja katsotaan löytyykö kyseistä käyttäjää. */
        var $promise = $http.post('http://backend.localhost/user.php', data);

        $promise.then(function(msg) {
            var uid = msg.data;
            if(uid) {
                if(data.rememberMe == true) {
                    cookieStore.put('trickCookie', data.mail); // Tallennetaan käyttäjän sähköpostiosoite sessioon.
                }
                sessionService.set('uid', uid);
                sessionService.set('email', data.mail);
                $location.path('/home');
            } else {
                scope.msgtxt = "Käyttäjätunnus tai salasana on virheellinen!";
                $(".alerts-danger").addClass("alerts-danger-style");
            }
        });
},

logout:function(cookieStore){
```

```
    /* Poistetaan käyttäjä tietokannasta automaattisten testien helpottamiseksi. */
    $http.post('http://backend.localhost/deleteUser.php', {email: sessionService.get('email')});

    cookieStore.remove('trickCookie');
    sessionService.destroy('uid');
    $location.path('/');
},

islogged:function(){

    /* Katsotaan onko käyttäjä kirjautunut sisään (tarkistetaan session tila). */
    var $checkSessionServer = $http.post('http://backend.localhost/check_session.php');
    return $checkSessionServer;
}
};
}]);
```

registerService.js

```
'use strict';

app.factory('registerService', ['$http', '$location', function($http, $location) {
    return {
        register: function(data, scope) {

        /* Luodaan käyttäjätunnus annetuilla tiedoilla. */
        var $promise = $http.post('http://backend.localhost/register.php', data);

        $promise.then(function(msg) {
            if(msg.data == 'success') {
                scope.msgRegister = "Rekisteröinti onnistui!";
                $(".alerts-danger").removeClass("alerts-danger-style");
                $(".alerts-danger").addClass("alerts-nice-style");
            } else {
                scope.msgRegister = "Sähköpostiosoite on jo olemassa! Anna toinen sähköpostiosoite.";
                $(".alerts-danger").removeClass("alerts-nice-style");
                $(".alerts-danger").addClass("alerts-danger-style");
            }
        });
    },
};
}]);
```

returnPwdService.js

```
'use strict';

app.factory('returnPwdService', ['$http', '$location', function($http, $location) {
    return {
        returnPwd: function(data, scope) {

        /* Palautetaan salasana käyttäjän sähköpostiosoitteen avulla. */
        var $promise = $http.post('http://backend.localhost/returnPwd.php', data);

        $promise.then(function(msg) {
            if(msg.data == 'success') {
                scope.msgReturnPwd = "Salasanan palautus onnistui! Voit nyt kirjautua sisään.";
                $(".alerts-danger").removeClass("alerts-danger-style");
                $(".alerts-danger").addClass("alerts-nice-style");
            } else {
                scope.msgReturnPwd = "Sähköpostiosoitetta ei löydy. Anna toinen sähköpostiosoite.";
```

```
                $(".alerts-danger").removeClass("alerts-nice-style");
                $(".alerts-danger").addClass("alerts-danger-style");
            }
        });
    },
};
}]);
```

sessionService.js

```
'use strict';

app.factory('sessionService', ['$http', function($http){
    return{
        set:function(key, value) {
            return sessionStorage.setItem(key, value);
        },
        get:function(key) {
            return sessionStorage.getItem(key);
        },
        destroy:function(key) {
        /* Tuhotaan sessio, kun käyttäjä kirjautuu ulos. */
        $http.post('http://backend.localhost/destroy_session.php');
            return sessionStorage.removeItem(key);
        }
    };
}]);
```

app.js

```
'use strict';

/* Ladataan Angular.js:n käyttämät lisäosat. */
var app = angular.module('trickApp', ['ngRoute', 'ui.bootstrap', 'ngCookies']);

/* Ladataan .html-tiedosto sivun mukaan. */
app.config(['$routeProvider', function($routeProvider) {
    $routeProvider.when('/', {
        title: 'Kirjaudu sisään',
        templateUrl: 'templates/login.html',
        controller: 'LoginCtrl'
    });

    $routeProvider.when('/register', {
        title: 'Rekisteröidy',
        templateUrl: 'templates/register.html',
        controller: 'RegisterCtrl'
    });
    $routeProvider.when('/return-password', {
        title: 'Palauta salasana',
        templateUrl: 'templates/return-password.html',
        controller: 'ReturnPwdCtrl'
    });
    $routeProvider.when('/home', {
        title: 'Tervetuloa',
        templateUrl: 'templates/home.html',
        controller: 'homeCtrl'
    });
    $routeProvider.otherwise({ redirectTo: '/' });
}]);
```

```
app.run(['$location', '$rootScope', '$cookieStore', 'loginService', function($location, $rootScope, $cookieStore,
loginService) {

    var routespermission = ['/home']; // Estetyt sivut ilman sisäänkirjausta.

    /* Tarkistetaan käyttäjän oikeudet niille sivuille, joihin vaaditaan sisäänkirjaus. */
    $rootScope.$on('$routeChangeStart', function(){
        var userCookie = $cookieStore.get('trickCookie');
        if(routespermission.indexOf($location.path()) != -1 || typeof userCookie !== 'undefined') {
            var connected = loginService.islogged();
                connected.then(function(msg){
                    if(!msg.data && typeof userCookie == 'undefined') {
                        $location.path('/');
                    }
                 });
        }
});

/* Ladataan ja asetetaan sivukohtainen title .index.html-tiedostoon. */
$rootScope.$on('$routeChangeSuccess', function (event, current, previous) {
    if (current.hasOwnProperty('$$route')) {
        $rootScope.title = current.$$route.title;
    }
});
}]);
```

recording.tpl.html

```
<div class="inside-container">
    <div class="form-group has-success">
        <form role="form" name="recordingForm" novalidate>

            <select class="form-control" data-ng-model="myRecording.tasks" required data-ng-options="task.name
            for task in tasks" ><option value="">Valitse tehtävän tyyppi *</option></select><br />

            <label class="control-label" for="description1">Työtehtävän kuvaus *</label>
             <textarea id="description1" class="form-control" rows="3" required data-ng mod-
             el="myRecording.description"></textarea><br />

             <select class="form-control" data-ng-model="myRecording.oldTask" data-ng-options="oldTask.name for
             oldTask in oldTasks"><option value="">Jatka edellistä työtehtävää (halutessasi)</option></select><br />

             <button class="btn btn-lg btn-primary btn-block" type="submit" data-ng-disabled="timerRunning" data-
             ng-click="start(myRecording)">Aloita nauhoitus</button>

             <button class="btn btn-lg btn-warning btn-block" data-ng-model="stopButton" type="submit" data-ng-
             disabled="!timerRunning" data-ng-click="stop()">Pysäytä nauhoitus</button><br />

             <div class="alerts-danger">{{ msgError }}</div>
        </form>
    </div>
</div>
```

recordings.tpl.html

```
<div class="inside-container">
    <div class="form-group has-success">
        <table class="table table-striped table-bordered">
            <tr>
                <td><b>Päivämäärä</b></td>
                <td><b>Tehtävä</b></td>
```

```
            <td><b>Aika</b></td>
            <td></td>
          </tr>

          <tbody>
            <tr ng-repeat="recording in allRecordings | orderBy:start_time:reverse">
              <td>{{recording.start_time}}</td>
              <td>{{recording.name}}</td>
              <td>{{recording.full_time}} min</td>
              <td><button class="btn btn-xs btn-danger" type="submit" data-ng-click="deleteOne(recording.id,  $in-
              dex)"><span class="glyphicon glyphicon-remove" aria-hidden="true"></button></span></td>
            </tr>
          </tbody>
        </table>

        <button class="btn btn-lg btn-danger btn-block" type="submit" data-ng-click="deleteAll(allRecordings)">Poista
        kaikki</button>
    </div>
</div>
```

home.html

```
<tabset justified="true">
    <tab heading="Nauhoitus"><div data-recording-directive>Kutsutaan recording.tpl.html</div></tab>
    <tab heading="Nauhoitukset"><div data-recordings-directive>Kutsutaan recordings.tpl.html</div></tab>
    <tab select="logout()">
    <tab-heading><i class="glyphicon glyphicon-log-out"></i> Ulos</tab-heading></tab>
</tabset>
```

login.html

```
<form class="form" role="form" name="form1">
      <h2>Kirjaudu sisään</h2>
      <label for="inputEmail" class="sr-only">Sähköpostiosoite</label>
      <input type="email" id="inputEmail" class="form-control" placeholder="Sähköpostiosoite" required autofocus
      data-ng-model="user.mail" />
      <label for="inputPassword" class="sr-only">Salasana</label>
      <input type="password" id="inputPassword" class="form-control" placeholder="Salasana" required data-ng-
      model="user.password" />

      <div class="checkbox"><label><input type="checkbox" data-ng-model="user.rememberMe"> Muista
      minut</label></div>

      <button class="btn btn-lg btn-primary btn-block" type="submit" data-ng-disabled="form1.$invalid" data-ng-
      click="login(user)">Kirjaudu</button>

</form>

<p><a href="#/register" title="">Rekisteröidy tästä, se on ilmaista</a></p>
<p><a href="#/return-password" title="">Salasanan palautus</a></p>

<div class="alerts-danger">{{ msgtxt }}</div>
```

register.html

```
<form class="form" role="form" name="form3">
      <h2>Rekisteröidy</h2>
      <label for="inputEmail" class="sr-only">Anna sähköpostiosoite</label>
      <input type="email" id="inputEmail" class="form-control" placeholder="Anna sähköpostiosoite" required auto-
      focus data-ng-model="newUser.mail" />
```

```
    <label for="inputPassword" class="sr-only">Anna salasana</label>
    <input type="password" id="inputPassword" class="form-control" placeholder="Anna salasana" required data-
    ng-model="newUser.password" />

    <button class="btn btn-lg btn-primary btn-block" type="submit" data-ng-disabled="form3.$invalid" data-ng-
    click="register(newUser)">Rekisteröidy</button>

    <a class="btn btn-lg btn-warning btn-block" type="button" href="#/">Peruuta</a>

</form>

<div class="alerts-danger">{{ msgRegister }}</div>
```

return-password.html

```
<form class="form" role="form" name="form4">
    <h2>Palauta salasana</h2>
    <label for="inputEmail" class="sr-only">Anna sähköpostiosoite</label>
    <input type="email" id="inputEmail" class="form-control" placeholder="Anna sähköpostiosoite" required auto-
    focus data-ng-model="user.mail" />

    <button class="btn btn-lg btn-primary btn-block" type="submit" data-ng-disabled="form4.$invalid" data-ng-
    click="returnPwd(user)">Palauta</button>

    <a class="btn btn-lg btn-warning btn-block" type="button" href="#/">Peruuta</a>
</form>

<div class="alerts-danger">{{ msgReturnPwd }}</div>
```

index.html

```
<!DOCTYPE html>
<html class="no-js" lang="fi" data-ng-app="trickApp">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="Tuntikirjanpitosovellus - Trick">
    <meta name="author" content="Petteri Aho">

    <title data-ng-bind="'Trick - ' + title">Trick</title>

    <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css">
    <link rel="stylesheet" href="css/index.css">

    <script src="http://cdn.jsdelivr.net/modernizr/2.8.3/modernizr.js"></script>
    <script src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.27/angular.min.js"></script>
    <script src="https://code.angularjs.org/1.2.27/angular-route.min.js"></script>
    <script src="https://code.angularjs.org/1.2.27/angular-cookies.min.js"></script>
     <script src="http://cdnjs.cloudflare.com/ajax/libs/angular-ui-bootstrap/0.12.0/ui-bootstrap-
     tpls.min.js"></script>

    <!--<script src="js/packed/trick/trick.all.min.js"></script>-->

    <script src="js/trick/general/support.js"></script>
    <script src="js/trick/app.js"></script>
    <script src="js/trick/directives/directives.js"></script>

    <script src="js/trick/services/dataService.js"></script>
    <script src="js/trick/services/loginService.js"></script>
```

```
        <script src="js/trick/services/sessionService.js"></script>
        <script src="js/trick/services/homeService.js"></script>
        <script src="js/trick/services/registerService.js"></script>
        <script src="js/trick/services/returnPwdService.js"></script>

        <script src="js/trick/controllers/loginCtrl.js"></script>
        <script src="js/trick/controllers/homeCtrl.js"></script>
        <script src="js/trick/controllers/registerCtrl.js"></script>
        <script src="js/trick/controllers/returnPwdCtrl.js"></script>
    </head>
<body>
    <div class="container">
        <div data-ng-view=""></div> <!-- Ladataan app.js-tiedostossa .html-tiedosto sivun mukaan. -->
        <div class="js-disabled">Huomautus! JavaScript ei toimi oikein selaimessasi. Kaikki ominaisuudet eivät ole käy-
        tössä.</div>
        <div class="alerts"></div> <!-- Palautetaan support.js-tiedoston luomat varoitukset. -->
    </div>
</body>
</html>
```

Gruntfile.js

```
module.exports = function(grunt) {
    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        uglify: {
            options: {
                banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n',
            },
            build: {
                src: 'public/js/trick/**/*.js',
                dest: 'public/js/packed/trick/trick.all.min.js'
            }
        }
  });
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.registerTask('default', ['uglify']);
};
```

package.json

```
{
    "name": "Trick",
    "version": "0.1.0",
    "devDependencies": {
        "grunt": "~0.4.5",
        "grunt-contrib-uglify": "~0.5.0"
    }
}
```

LIITE 3 TRICK2-LÄHDEKOODIT

index.css-tiedosto on sama kuin Trick-sovelluksessa.

homeCtrl.js

```
'use strict';

var home = new homeService();

var homeCtrl = {
    logout: function() {
        service = new loginService();
        service.logout(); // Kutsutaan loginServicen logout-funktiota.
    },

getTasks: function(element) {
    var tasks = [{ name: "Ohjelmointi" }, { name: "Palaveri" }, { name: "Sekalaista" }];
    home.createOption(element, tasks);
},

getOldTasks: function(element) {
    dataService.getData(element);
},

start: function() {
    var getRecordingData = function() {
        var oldTask = document.getElementById("oldTasks");

        return { // Otetaan ja asetetaan arvot palvelinpään edellyttämään muotoon.
            "tasks" : {"name" : document.getElementById("tasks").value},
            "description" : document.getElementById("description1").value,
            "oldTask" : {
            "name" : oldTask.value,
            "id" : oldTask[oldTask.selectedIndex].id
        },
    };
};

home.start(getRecordingData());
},

stop: function() {
    home.stop(home);
},

deleteOne: function(id, index) {
    home.deleteOne(id, index);
},

deleteAll: function() {
    home.deleteAll();
}
};
```

loginCtrl.js

```
'use strict';

var loginCtrl = function() {
    var service = new loginService();
    var getLoginData = function() {
        return {
            "mail" : document.getElementById("inputEmail").value,
            "password" : document.getElementById("inputPassword").value,
            "checkbox" : document.getElementById("checkbox").checked
        };
    };
    service.login(getLoginData());
};
```

registerCtrl.js

```
'use strict';

var registerCtrl = function() {
    var service = new registerService();
    var getRegisterData = function() {
        return {
            "mail" : document.getElementById("inputEmail").value,
            "password" : document.getElementById("inputPassword").value,
        };
    };
    service.register(getRegisterData());
};
```

returnPwdCtrl.js

```
'use strict';

var returnPwdCtrl = function() {
    var service = new returnPwdService();
    var getPwdData = function() {
        return {
            "mail" : document.getElementById("inputEmail").value,
        };
    };
    service.returnPwd(getPwdData());
};
```

directives.js

```
'use strict';

function directives(template, id, element) {

    if(typeof element !== 'undefined') {
        document.getElementsByClassName("active")[0].className = "ng-isolate-scope";
        element.parentNode.className = "ng-isolate-scope active";
    }

    loadPage(template, id);

    if(template.substring(4, 14) == 'recordings') {
```

```
        dataService.getData(); // Haetaan käyttäjän kaikki nauhoitukset.
    } else {
        setTimeout(function(){
        homeCtrl.getTasks(document.getElementById("tasks"));
        homeCtrl.getOldTasks(document.getElementById("oldTasks"));
    }, 100);
    }
    window.location.href = '#/home';
}
```

loadPage.js

```
'use strict';

function loadPage(template, id) {
    var changeTitle = function(page) {
        var name = "";
        if(page === '/register') name = "- Rekisteröidy";
        else if(page === '/return-password') name = "- Palauta salasana";
        else if(page === '/home' || page.substring(0, 4) === '/tpl') name = "- Tervetuloa";
        else if(page === '/') name = "- Kirjaudu sisään";
       document.title = "Trick " + name;
    };

    var getTemplate = function(template) {

        /* Tarkistetaan käyttäjän oikeudet niille sivuille, joihin vaaditaan sisäänkirjaus. */
        var allowed = routesPermission();

        if(template === 'login' || allowed === false) {
            window.location.href = '#/';
        } else {
            window.location.href = '#/' + template;
        }

        changeTitle(document.URL.split("#")[1]);

        return "templates/" + template + ".html";
    };

httpRequest(getTemplate(template), "", function(result){
    document.getElementById(id).innerHTML = result;
});
}
```

support.js

```
'use strict';

function javaScriptEnabled(class1, class2) {
    var changeClass = function() {
        var elements = document.getElementsByClassName("no-js");
        elements[0].className = 'js-disabled js';
    };
    changeClass();
}

function support() {
    this.addJsCheck = function(class1) {
        var elements = document.getElementsByClassName(class1);
```

```
        if(typeof elements[0] != 'undefined') {
            elements[0].parentNode.removeChild(elements[0]);
        }
};

this.addText = function(alertText1, alertText2) {
    this.alertText1 = alertText1;
    this.alertText2 = alertText2;
};

this.createAlert = function(divElement) {

    var newFeatures = "";
    var fieldNameElement = document.getElementById(divElement);
    var input = document.getElementById('placeholderTest');

    if (!JSON && typeof JSON.parse !== 'function') {
        newFeatures += "json, ";
    } if(input.placeholder === undefined) {
        newFeatures += "placeholder, ";
    } if (!window.sessionStorage) {
        newFeatures += "sessionStorage, ";
    }

    if(newFeatures !== "") {
        fieldNameElement.innerHTML = this.alertText1 + " " + newFeatures.replace(/,\s*$/, ".") + " " + this.alertText2;
        fieldNameElement.className = "alerts-style";
    }
};
}
```

xmlhttp.js

```
'use strict';

function httpRequest(url, data, callback) {
    var xmlhttp;
    if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
    } else { // IE6, IE5
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }

    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            callback(xmlhttp.responseText);
        }
    };

    xmlhttp.open("POST", url, true);
    xmlhttp.send(data);
}
```

dataService.js

```
'use strict';

var dataService = {
    getData: function(element) {
        var sessionHandler = new sessionService();
        var usrEmail = sessionHandler.get('email'); // Haetaan sessiosta käyttäjän tiedot.
```

```
        var json = JSON.stringify({email : usrEmail});

        /* Haetaan tietokannasta käyttäjän kaikki nauhoitukset sähköpostiosoitteen perusteella. */
        httpRequest('http://backend.localhost/getData.php', json, function(msg){
            var newMsg = eval(msg);
                if(typeof element !== 'undefined') {
                    var home = new homeService();
                    home.createOption(element, newMsg);
                } else {
                    var tbody = document.getElementById("allRecordings");

                for (var i = 0; i < newMsg.length; i++) {
                   var row = tbody.insertRow(0);
                   row.insertCell(0).innerHTML = newMsg[i].start_time;
                   row.insertCell(1).innerHTML = newMsg[i].name;
                   row.insertCell(2).innerHTML = newMsg[i].full_time + " min";
                    row.insertCell(3).innerHTML = "<button class='btn btn-xs btn-danger' " + "on-
                    click='homeCtrl.deleteOne(" + newMsg[i].id + ", this)' type='submit'>" + "<span class='glyphicon glyph-
                    icon-remove' aria-hidden='true'></button></span>";
                }
            }
        });
    }
};
```

homeService.js

```
'use strict';

function homeService() {
    var sessionHandler = new sessionService();
    this.uniquerNumber = "";
    this.createOption = function(element, loopValues) {

        for (var i = 0; i < loopValues.length; i++) {
            var opt = document.createElement("option");
            opt.value = loopValues[i].name;
            if(loopValues[i].id) {
                opt.id = loopValues[i].id;
            }

        opt.innerHTML = loopValues[i].name;
        element.appendChild(opt);
    }
};

this.start = function(myRecording) {
    this.uniquerNumber = Math.random().toString(36).substring(7);
    var usrEmail = sessionHandler.get('email'); // Haetaan sessiosta käyttäjän tiedot.
    var json = JSON.stringify({myRecording : myRecording, email : usrEmail, unique: this.uniquerNumber});

    /* Kun käyttäjä aloittaa tai jatkaa nauhoitusta, niin palvelinpäässä tallennetaan seuraavat tiedot tietokantaan:
     *
     * nauhoituksen aloitusaika, käyttäjän sähköpostiosoite, tehtävän tyyppi,
     * nauhoituksen yksilöivä numerosarja, tehtävän kuvaus ja nauhoituksen tila. */
    httpRequest('http://backend.localhost/recording.php', json, function(msg){
        if(msg != 'success') {
            var msgtxt = document.getElementById("alerts-danger");
            msgtxt.innerHTML = "Tarvittavat tiedot puuttuvat! Yritä uudelleen.";
            msgtxt.className = "alerts-danger-style";
        } else {
```

```
            document.getElementById("stop").className = "btn btn-lg btn-warning btn-block";
            document.getElementById('start').className = "btn btn-lg btn-primary btn-block disabled";
        }
    });
};

this.stop = function(homeService) {

    var json = JSON.stringify({email: sessionHandler.get('email'), unique: this.uniquerNumber});

    /* Tallennetaan nauhoituksen kestoon kulunut kokonaisaika, jonka lisäksi haetaan ja palautetaan käyttäjän kaikki
    nauhoitukset. */
    httpRequest('http://backend.localhost/stopRecording.php', json, function(returnValues){
        if(typeof returnValues == "undefined") {

            var msgtxt = document.getElementById("alerts-danger");
            msgtxt.innerHTML = "Nauhoitusta ei voida lopettaa. Yritä myöhemmin uudelleen.";
            msgtxt.className = "alerts-danger-style";
        } else {

            var oldTasks = document.getElementById("oldTasks");
            for(var i = oldTasks.options.length - 1; i >= 0; i--){
                oldTasks.remove(i + 1);
        }

        homeService.createOption(oldTasks, eval(returnValues));
        document.getElementById("stop").className = "btn btn-lg btn-warning btn-block disabled";
        document.getElementById('start').className = "btn btn-lg btn-primary btn-block";
    }
});
};

this.deleteOne = function(id, index) {

    var json = JSON.stringify({id: id});

    /* Poistetaan käyttäjän nauhoitus tietokannasta id:n perusteella. */
    httpRequest('http://backend.localhost/deleteOne.php', json, function(msg){
        if(msg == "success") {
            var row = index.parentNode.parentNode;

            // Poistetaan nauhoitus käyttöliittymästä.
            document.getElementById("allRecordings").deleteRow(row.rowIndex - 1);
        }
    });
};

this.deleteAll = function() {
    var json = JSON.stringify({email: sessionHandler.get('email')});

    /* Poistetaan käyttäjän kaikki nauhoitukset tietokannasta käyttäjän sähköpostiosoitteen perusteella. */
    httpRequest('http://backend.localhost/deleteAll.php', json, function(msg){
        if(msg == 'success') {
            var tbody = document.getElementById("allRecordings");
            tbody.innerHTML = "";
        }
    });
};
}
```

loginService.js

```
'use strict';

function loginService() {
    var sessionHandler = new sessionService();
    this.login = function(data) {
        var json = JSON.stringify(data);

        /* Haetaan tietokannasta käyttäjätiedot ja katsotaan löytyykö kyseistä käyttäjää. */
        httpRequest('http://backend.localhost/user.php', json, function(uid){
            if(uid.length !== 0) {
                if(data.checkbox == true) {
                    sessionHandler.put('trickCookie', data.mail); // Tallennetaan käyttäjän sähköpostiosoite sessioon.
                }

                sessionHandler.set('uid', uid);
                sessionHandler.set('email', data.mail);
                loadPage('home', 'mainContent');
               directives('tpl/recording.tpl', 'subContent');
            } else {
                var msgtxt = document.getElementById("alerts-danger");
                msgtxt.innerHTML = "Käyttäjätunnus tai salasana on virheellinen!";
                msgtxt.className = "alerts-danger-style";
            }
    });
};

this.logout = function() {
    var json = JSON.stringify({email: sessionHandler.get('email')});

    /* Poistetaan käyttäjä tietokannasta automaattisten testien helpottamiseksi. */
    httpRequest('http://backend.localhost/deleteUser.php', json, function(msg){});
        sessionHandler.remove('trickCookie');
        sessionHandler.destroy('uid');
        loadPage('login', 'mainContent');
    };

this.islogged = function(callback) {

    /* Katsotaan onko käyttäjä kirjautunut sisään (tarkistetaan session tila). */
    httpRequest('http://backend.localhost/check_session.php', "", function(msg){
        callback(msg);
    });
};
};
```

registerService.js

```
'use strict';

function registerService() {
    this.register = function(data) {
        var json = JSON.stringify(data);

        /* Luodaan käyttäjätunnus annetuilla tiedoilla. */
        httpRequest('http://backend.localhost/register.php', json, function(msg){
            var msgRegister = document.getElementById("alerts-danger");
            if(msg == 'success') {
                msgRegister.innerHTML = "Rekisteröinti onnistui!";
                msgRegister.className = "alerts-nice-style";
```

```
            } else {
                msgRegister.innerHTML = "Sähköpostiosoite on jo olemassa! Anna toinen sähköpostiosoite.";
                msgRegister.className = "alerts-danger-style";
            }
        });
};
};
```

returnPwdService.js

```
'use strict';

function returnPwdService() {
    this.returnPwd = function(data) {
        var json = JSON.stringify(data);

        /* Palautetaan salasana käyttäjän sähköpostiosoitteen avulla. */
        httpRequest('http://backend.localhost/returnPwd.php', json, function(msg){
            var msgRegister = document.getElementById("alerts-danger");
                if(msg == 'success') {
                    msgRegister.innerHTML = "Salasanan palautus onnistui! Voit nyt kirjautua sisään.";
                    msgRegister.className = "alerts-nice-style";
                } else {
                    msgRegister.innerHTML = "Sähköpostiosoitetta ei löydy. Anna toinen sähköpostiosoite.";
                    msgRegister.className = "alerts-danger-style";
                }
            });
        };
};
```

sessionService.js

```
'use strict';

function sessionService() {
    this.remove = function(name) {
    var cookie = name + '=;';
    cookie += 'expires=' + (new Date()).toString() + ';';
    document.cookie = cookie;
};

this.put = function(name, value) {
    var date = new Date();
    var cookie = name + '=';
    cookie += value + ';';
    date.setDate(date.getDate() + 365);
    cookie += 'expires=' + date.toString() + ';';
    document.cookie = cookie;
};

this.set = function(key, value) {
    return sessionStorage.setItem(key, value);
};

this.get = function(key) {
    return sessionStorage.getItem(key);
};

this.destroy = function(key) {

    /* Tuhotaan sessio, kun käyttäjä kirjautuu ulos. */
```

```
    httpRequest('http://backend.localhost/destroy_session.php', "", function(msg){});
        return sessionStorage.removeItem(key);
    };
};
```

app.js

```
'use strict';

document.addEventListener('DOMContentLoaded', function(){
    javaScriptEnabled(); // Tarkistetaan JavaScript-tuki.
    var newSupport = new support(); // Testataan selaimen tuki eri ominaisuuksille.
    newSupport.addJsCheck("js");
    newSupport.addText("Huomautus! Seuraavat ominaisuudet eivät ole käytössä:", "Sovellus ei toimi oikein.");
    newSupport.createAlert("alerts");

    loadPage("login", "mainContent"); // Ladataan sivun sisältö, kun tullaan ensimmäistä kertaa sivulle.
});

function checkValidation(one) {
    var password;
    var email = document.getElementById("inputEmail").value;
    var checkList = email.length !== 0;
    if(typeof one === 'undefined') {
        password = document.getElementById("inputPassword").value;
        checkList = email.length !== 0 && password.length !== 0;
    }

var button = document.getElementById('button');
    if(checkList) {
        button.disabled = false;
        button.className = "btn btn-lg btn-primary btn-block";
    } else {
        button.disabled = 'disabled';
        button.className = "btn btn-lg btn-primary btn-block disabled";
    }
}

function routesPermission() {
    var routespermission = ['/home']; // Estetyt sivut ilman sisäänkirjausta
    var sessionHandler = new sessionService();
    var userCookie = sessionHandler.get('trickCookie');
    var url = document.URL.split("#");

    /* Tarkistetaan käyttäjän oikeudet niille sivuille, joihin vaaditaan sisäänkirjaus. */
    if(routespermission.indexOf(url[1]) != -1 || userCookie !== null) {
        var returnValue = new loginService();
        returnValue = returnValue.islogged(function(msg) {
            if(typeof returnValue == 'undefined') {
                return false;
            }
        });
    }
}
```

recording.tpl.html

```
<div class="inside-container">
    <div class="form-group has-success">
        <form role="form" name="recordingForm" novalidate>
```

```
        <select id="tasks" class="form-control" required><option value="">Valitse tehtävän tyyppi *</option>
        </select><br />

        <label class="control-label" for="description1">Työtehtävän kuvaus *</label><textarea id="description1"
        class="form-control" rows="3" required></textarea><br />

        <select id="oldTasks" class="form-control"><option value="">Jatka edellistä työtehtävää (ha-
        lutessasi)</option></select><br />

        <a class="btn btn-lg btn-primary btn-block" id="start" type="button" onclick="homeCtrl.start();"
        href="#/home">Aloita nauhoitus</a>

        <a class="btn btn-lg btn-warning btn-block disabled" id="stop" type="button" onclick="homeCtrl.stop();"
        href="#/home">Pysäytä nauhoitus</a><br />

        <div id="alerts-danger"></div>
      </form>
    </div>
</div>
```

recordings.tpl.html

```
<div class="inside-container">
    <div class="form-group has-success">
        <table class="table table-striped table-bordered">
            <tr>
                <td><b>Päivämäärä</b></td>
                <td><b>Tehtävä</b></td>
                <td><b>Aika</b></td>
                <td></td>
            </tr>

            <tbody id="allRecordings"></tbody>
        </table>

        <a class="btn btn-lg btn-danger btn-block" type="button" onclick="homeCtrl.deleteAll();"
        href="#/home">Poista kaikki</a>
    </div>
</div>
```

home.html

```
<ul class="nav nav-tabs nav-justified">
    <li class="ng-isolate-scope active"><a class="ng-binding" onclick="directives('tpl/recording.tpl', 'subContent',
    this);" href="#/home">Nauhoitus</a></li>

    <li class="ng-isolate-scope"><a class="ng-binding" onclick="directives('tpl/recordings.tpl', 'subContent', this);"
    href="#/home">Nauhoitukset</a></li>

    <li class="ng-isolate-scope"><a class="ng-binding" onclick="homeCtrl.logout();" href="#/"><i class="glyphicon
    glyphicon-log-out"></i> Ulos</a></li>
</ul>

<div id="subContent"></div> <!-- Ladataan .html-tiedosto sivun mukaan. -->
```

login.html

```
<form class="form" role="form">
    <h2>Kirjaudu sisään</h2>
    <label for="inputEmail" class="sr-only">Sähköpostiosoite</label>
```

```
    <input type="email" id="inputEmail" class="form-control" onkeyup="checkValidation();" placehold-
    er="Sähköpostiosoite" required autofocus />
    <label for="inputPassword" class="sr-only">Salasana</label><input type="password" id="inputPassword"
    class="form-control" onkeyup="checkValidation();" placeholder="Salasana" required />

    <div class="checkbox"><label><input id="checkbox" type="checkbox"> Muista minut</label></div>
    <a class="btn btn-lg btn-primary btn-block disabled" type="button" id="button" onclick="loginCtrl();"
    href="#/home">Kirjaudu</a>
</form>

<p><a href="#/register" onclick="loadPage('register', 'mainContent');" title="">Rekisteröidy tästä, se on il-
maista</a></p>
<p><a href="#/return-password" onclick="loadPage('return-password', 'mainContent');" title="">Salasanan palau-
tus</a></p>

<div id="alerts-danger"></div>
```

register.html

```
<form class="form" role="form" name="form3">
    <h2>Rekisteröidy</h2>
    <label for="inputEmail" class="sr-only">Anna sähköpostiosoite</label>
    <input type="email" id="inputEmail" class="form-control" onkeyup="checkValidation();" placeholder="Anna
    sähköpostiosoite" required autofocus />
    <label for="inputPassword" class="sr-only">Anna salasana</label>
    <input type="password" id="inputPassword" class="form-control" onkeyup="checkValidation();" placehold-
    er="Anna salasana" required />

    <a class="btn btn-lg btn-primary btn-block disabled" type="button" id="button" onclick="registerCtrl();"
    href="#/register">Rekisteröidy</a>

    <a class="btn btn-lg btn-warning btn-block" type="button" onclick="loadPage('login', 'mainContent');"
    href="#/">Peruuta</a>
</form>

<div id="alerts-danger"></div>
```

return-password.html

```
<form class="form" role="form" name="form4">
    <h2>Palauta salasana</h2>
    <label for="inputEmail" class="sr-only">Anna sähköpostiosoite</label>
    <input type="email" id="inputEmail" class="form-control" onkeyup="checkValidation('one');" placehold-
    er="Anna sähköpostiosoite" required autofocus />

    <a class="btn btn-lg btn-primary btn-block disabled" type="button" id="button" onclick="returnPwdCtrl();"
    href="#/return-password">Palauta</a>

    <a class="btn btn-lg btn-warning btn-block" type="button" onclick="loadPage('login', 'mainContent');"
    href="#/">Peruuta</a>
</form>

<div id="alerts-danger"></div>
```

index.html

```
<!DOCTYPE html>
<html lang="fi">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="Tuntikirjanpitosovellus - Trick">
        <meta name="author" content="Petteri Aho">

        <title>Trick</title>

        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css">
        <link rel="stylesheet" href="css/index.css">

        <script src="js/trick/general/support.js"></script>
        <script src="js/trick/general/xmlhttp.js"></script>
        <script src="js/trick/general/loadPage.js"></script>

        <script src="js/trick/services/dataService.js"></script>
        <script src="js/trick/services/loginService.js"></script>
        <script src="js/trick/services/sessionService.js"></script>
        <script src="js/trick/services/homeService.js"></script>
        <script src="js/trick/services/registerService.js"></script>
        <script src="js/trick/services/returnPwdService.js"></script>

        <script src="js/trick/controllers/loginCtrl.js"></script>
        <script src="js/trick/controllers/homeCtrl.js"></script>
        <script src="js/trick/controllers/registerCtrl.js"></script>
        <script src="js/trick/controllers/returnPwdCtrl.js"></script>

        <script src="js/trick/directives/directives.js"></script>
        <script src="js/trick/app.js"></script>
    </head>
<body>
    <div class="container">
            <input type="hidden" id="placeholderTest"> <!-- Testataan selaimen placeholder-tuki. -->
            <div id="mainContent"></div> <!-- Ladataan .html-tiedosto sivun mukaan. -->
            <div class="js-disabled no-js">Huomautus! JavaScript ei toimi oikein selaimessasi. Kaikki ominaisuudet eivät
            ole käytössä.</div>
            <div id="alerts"></div> <!-- Palautetaan support.js-tiedoston luomat varoitukset. -->
    </div>
</body>
</html>
```

LIITE 4 AUTOMAATIOTESTIT

Trick

```
VERSION BUILD=8881205 RECORDER=FX
TAB T=1
URL GOTO=http://trick.localhost/#/
TAG POS=1 TYPE=A ATTR=TXT:Rekisteröidy<SP>tästä,<SP>se<SP>on<SP>ilmaista
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form3 ATTR=ID:inputEmail CONTENT=petteri@gmail.com SET !EN-
CRYPTION NO
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form3 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=BUTTON FORM=NAME:form3 ATTR=TXT:Rekisteröidy
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form3 ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form3 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=BUTTON FORM=NAME:form3 ATTR=TXT:Rekisteröidy
TAG POS=1 TYPE=A ATTR=TXT:Peruuta
TAG POS=1 TYPE=A ATTR=TXT:Salasanan<SP>palautus
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form4 ATTR=ID:inputEmail CONTENT=tuntematon@gmail.com
TAG POS=1 TYPE=BUTTON FORM=NAME:form4 ATTR=TXT:Palauta
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form4 ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=BUTTON FORM=NAME:form4 ATTR=TXT:Palauta
TAG POS=1 TYPE=A ATTR=TXT:Peruuta
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form1 ATTR=ID:inputEmail CONTENT=tuntematon@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form1 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=BUTTON FORM=NAME:form1 ATTR=TXT:Kirjaudu
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form1 ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form1 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=INPUT:CHECKBOX FORM=NAME:form1 ATTR=* CONTENT=YES
TAG POS=1 TYPE=BUTTON FORM=NAME:form1 ATTR=TXT:Kirjaudu
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm
ATTR=TXT:Valitse<SP>tehtävän<SP>tyyppi<SP>*OhjelmointiPalaveriSekalai* CONTENT=%0
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Ohjelmointia
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Aloita<SP>nauhoitus
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Pysäytä<SP>nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm
ATTR=TXT:Valitse<SP>tehtävän<SP>tyyppi<SP>*OhjelmointiPalaveriSekalai* CONTENT=%1
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Viikkopalaveri
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Aloita<SP>nauhoitus
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Pysäytä<SP>nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm
ATTR=TXT:Valitse<SP>tehtävän<SP>tyyppi<SP>*OhjelmointiPalaveriSekalai* CONTENT=%2
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Kauppareissu
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Aloita<SP>nauhoitus
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Pysäytä<SP>nauhoitus
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm
ATTR=TXT:Jatka<SP>edellistä<SP>työtehtävää<SP>(halutessasi)Ohjelmoint* CONTENT=%0
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Aloita<SP>nauhoitus
TAG POS=1 TYPE=BUTTON FORM=NAME:recordingForm ATTR=TXT:Pysäytä<SP>nauhoitus
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=BUTTON ATTR=TXT:
TAG POS=1 TYPE=BUTTON ATTR=TXT:Poista<SP>kaikki
TAG POS=1 TYPE=TAB-HEADING ATTR=TXT:Ulos
```

Trick2

```
VERSION BUILD=8881205 RECORDER=FX
TAB T=1
URL GOTO=http://trick2.localhost/#/
TAG POS=1 TYPE=A ATTR=TXT:Rekisteröidy<SP>tästä,<SP>se<SP>on<SP>ilmaista
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form3 ATTR=ID:inputEmail CONTENT=petteri@gmail.com SET !EN-
CRYPTION NO
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form3 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form3 ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:form3 ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=A ATTR=TXT:Peruuta
TAG POS=1 TYPE=A ATTR=TXT:Salasanan<SP>palautus
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form4 ATTR=ID:inputEmail CONTENT=tuntematon@gmail.com
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:form4 ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=A ATTR=TXT:Peruuta
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:NoFormName ATTR=ID:inputEmail CON-
TENT=tuntematon@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:NoFormName ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=INPUT:EMAIL FORM=NAME:NoFormName ATTR=ID:inputEmail CONTENT=petteri@gmail.com
TAG POS=1 TYPE=INPUT:PASSWORD FORM=NAME:NoFormName ATTR=ID:inputPassword CONTENT=1234
TAG POS=1 TYPE=INPUT:CHECKBOX FORM=NAME:NoFormName ATTR=ID:checkbox CONTENT=YES
TAG POS=1 TYPE=A ATTR=ID:button
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm ATTR=ID:tasks CONTENT=%Ohjelmointi
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Ohjelmointia
TAG POS=1 TYPE=A ATTR=ID:start
TAG POS=1 TYPE=A ATTR=ID:stop
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm ATTR=ID:tasks CONTENT=%Palaveri
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Viikkopalaveri
TAG POS=1 TYPE=A ATTR=ID:start
TAG POS=1 TYPE=A ATTR=ID:stop
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm ATTR=ID:tasks CONTENT=%Sekalaista
TAG POS=1 TYPE=TEXTAREA FORM=NAME:recordingForm ATTR=ID:description1 CONTENT=Kauppareissu
TAG POS=1 TYPE=A ATTR=ID:start
TAG POS=1 TYPE=A ATTR=ID:stop
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitus
TAG POS=1 TYPE=SELECT FORM=NAME:recordingForm ATTR=ID:oldTasks CONTENT=%Ohjelmointi<SP>-
<SP>Ohjelmointia
TAG POS=1 TYPE=A ATTR=ID:start
TAG POS=1 TYPE=A ATTR=ID:stop
TAG POS=1 TYPE=A ATTR=TXT:Nauhoitukset
TAG POS=1 TYPE=BUTTON ATTR=TXT:
TAG POS=1 TYPE=A ATTR=TXT:Poista<SP>kaikki
TAG POS=1 TYPE=A ATTR=TXT:Ulos
```