HAAGA-HELIA
ammattikorkeakoulu

Single-Page Application -arkkitehtuurin käyttö verrattuna perinteiseen web-sovellukseen.

Aija Kaakinen

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2014

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

7.5.2014

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Aija Kaakinen	**Ryhmätunnus tai aloitusvuosi** 2010
Raportin nimi Single-Page Application -arkkitehtuurin käyttö verrattuna perinteiseen web-sovellukseen	**Sivu- ja liitesivumäärä** 40+1
Opettajat tai ohjaajat Mirja Jaakkola	

SPA-sovellukset ovat viime vuosien aikana nostaneet suosiotaan kehittäjien sekä asiakkaiden keskuudessa. Tähän ratkaisevana tekijänä ovat toimineet kehittyneet tekniikat, joiden avulla sovelluksen rakentaminen on helpompaa. Sovellukset toimivat Ajaxin avulla ja noudattavat RESTful-periaatteita.

Työ toteutetaan keväällä 2014 ja sen tavoitteena on selvittää selkeimmät eroavaisuudet SPA-sovelluksen ja perinteisen sovelluksen välillä ja perehtyä SPA-sovelluksen keskeisimpiin arkkitehtuurin muodostaviin tekniikkoihin. Työssä kuvataan sovellusten toimintaperiaate ja web-tekniikoiden, Ajaxin ja sovelluskehysten merkitys. Tutkimuksessa ei rakenneta esimerkkisovellusta.

Lähdemateriaalina on käytetty pääsääntöisesti kirjallisuutta, sekä web-artikkeleita. Tutkimuksessa on haastateltu kahta web-kehityksen asiantuntijaa, jotta saataisiin käytännönläheistä tietoa sovelluksista ja vertailumateriaalia kirjallisuuslähteisiin.

Tuloksissa esitellään, kuinka yhden html-sivun varaan perustuva SPA-sovellus eroaa perinteisestä web-sovelluksesta toimintalogiikoiden perustuessa selaimeen tai palvelimeen. Lisäksi esitellään kaavioita vasteaikojen vaikutuksesta käyttäjiin, sekä JavaScriptin toiminnasta eri selaimissa.

Johtopäätöksissä todetaan, että SPA-arkkitehtuuri on suhteellisen uusi toteutustapa, joka kuitenkin on yleistynyt viime vuosina. SPA-sovellus toimii vähemmin vuorovaikutuksessa palvelimen kanssa, kuin perinteinen web-sovellus ja asynkronisen kommunikoinnin avulla sen toiminnan pitäisi olla sujuvampaa kuin perinteisen. Myös vasteajat ovat lyhyemmät.

Asiasanat
Ajax, Asynkroninen, JavaScript, Single-Page Application, Web

HAAGA-HELIA
University of Applied Sciences

Abstract

7.5.2014

Degree programme in Business Information Technology

Authors Aija Kaakinen	**Group or year of entry** 2010
The title of thesis Single-Page Application architecture and comparing Single-Page Applications to traditional web applications	**Number of report pages and attachment pages** 40+1
Advisor(s) Mirja Jaakkola	

Single-Page Applications (SPAs) have become more popular among developers and consumers during the last years. The key factor has been the developed techniques which make it easier to build an application. SPAs use Ajax software and follow the RESTful principles.

The purpose of this thesis was to clarify the explicit differences between the SPAs and traditional applications and to get acquainted with the central techniques related to forming SPA-architecture. The principles of the applications, the importance of web-techniques, Ajax and application frameworks were included within the scope of this study. Building a sample application was excluded from the scope of the study.

The background material for this thesis was gathered from literal and online sources. In addition, two web-development experts were interviewed in order to get practical information about applications and comparison material for the theoretical background.

The thesis showed how an SPA, based on a single HTML page, differs from a traditional web-application when the operating models are based either on the front-end or back-end. Furthermore, the thesis presented charts about how response times affect users and how JavaScrip functions in different browsers.

The thesis concludes that SPA-architecture is a relatively new way of building applications, but it has become more common over the last years. According to this thesis, SPAs interact less with the server side than traditional web applications and with asynchronous communication, they should function more fluently than traditional applications.

Key words
Ajax, Asynchronous, JavaScript, Single-Page Application, Web

Sisällys

1 Johdanto

Internet, sen sivut ja sovellukset mahdollistavat ihmisten ja yritysten välisen kommunikoinnin, markkinoinnin ja kaupankäynnin ympäri maapallon. Erilaiset sovellukset ovat perusta monelle yrityskulttuurille ja tarjoavat käyttäjilleen erilaisia palveluja kuten musiikkia, hakukoneita, kauppa- ja tv-palveluja. Jotta käyttäjälle saadaan toimiva ja eheä käyttökokemus, on sovellus rakennettava sen käyttötarpeen mukaan. Web-sovellukset olivat aluksi staattisia, yksinkertaisia html-sivuja, joiden rakenne pohjautui tekstirivien, linkkien ja kuvien varaan. Käyttäjät vaativat kuitenkin pian enemmän käytettävyyttä joten sivulle ryhdyttiin rakentamaan dynaamisia ominaisuuksia. Nykyään sovelluksilta vaaditaan vielä enemmän; niiden kautta on pystyttävä näkemään kaupungin kartta, etsimään reittejä, pelaamaan online-pelejä tai esimerkiksi näyttämään videoita. Käyttökokemuksen nopeuden ja toimivuuden maksimoimiseksi on kehitetty teknologioita, jotka mahdollistavat sen, ettei koko sivua tarvitse jatkuvasti ladata uudelleen. (Asleson & Schutta, 2006, 50.)

Viimeisten vuosien aikana teknologian kehityksessä on otettu harppauksia eteenpäin niin web-sovellusten kehityksessä kuin niitä tukevissa kehitysmenetelmissä. Web-sivustot ovat muuttuneet dokumenttikirjastoista monimutkaisiksi, runsaasti dataa sisältäviksi komplekseiksi, joiden käytettävyyden parantamiseksi kehitetään koko ajan uusia teknologioita. Niistä pyritään saamaan nopeampia ja niihin sisällytetään erilaisia toimintoja, kuten slide(liuku)-tyyppisiä siirtymisiä ja liikkuvia objekteja. Sovellusten on toimittava ja skaalauduttava erilaisiin laitteisiin ja niiden käyttäjämäärät ovat kasvaneet. Perinteisten, dokumenttikirjastotyyppisten web-sovellusten rinnalle ovat tulleet interaktiiviset, modernit sovellukset. (Asleson & Schutta, 2006, 1-4.)

Edellä mainitut syyt ovat ajaneet kehityksen kohti tehokkaampia ratkaisuja. Palvelinpäässä toimivista sovelluksista on siirrytty liiketoimintalogiikaltaan asiakaspäähän painottuviin, yhden sivun web-sovelluksiin. Ne eivät ole täysin uusi asia, mutta kehittyvän teknologian myötä niiden rakentaminen, konfigurointi, sekä ylläpito on helpottunut huomattavasti. Viime vuosien aikana toimintalogiikaltaan selainpäähän sijoittuvan Single-Page Application-arkkitehtuurin varaan on rakennettu yhä enemmän sovelluksia ja se tarjoaa käyttäjälle vähemmän sivulatauksia vaativan sovelluksen.

1.1 Tutkimuksen tavoitteet ja rajaus

Tutkimuksessa on tarkoituksena saada tiivistettyä yhteen tärkein yhden sivun web-sovelluksiin liittyvä teoriatieto siinä käytettävistä eri tekniikoista ja joistakin kehitystyökaluista. Tämä tarkoittaa yleisimmän termistön esittelyä, sekä sovellusten rakenteen ja toimintaperiaatteiden selvitystä. Kerätyn teoriatiedon sekä haastattelujen pohjalta selvitetään tilanteet, missä SPA-sovelluksen käyttö on kannattavampaa kuin monen erillisen HTML-sivun varaan rakennetun perinteisen web-sovelluksen.

Tutkimuksessa käsitellään tutkimuskysymyksiä, jotta saataisiin selvyys siihen, mitä nämä kaksi sovellusarkkitehtuuria ovat. Milloin niitä kannattaa käyttää ja mitkä ovat niiden parhaat puolet ja heikkoudet.

- Mikä on SPA-arkkitehtuuria käyttävän sovelluksen ja perinteisen web-sovelluksen ero?
- Mitä SPA-sovellukset sekä perinteiset web-sovellukset ovat ja miten ne toimivat?
- Minkälaisiin käytettävyysongelmiin on mahdollista törmätä kehittäjän tai käyttäjän näkökulmasta?
- Millaisia hyötyjä tai haittoja on SPA-arkkitehtuuria noudattavissa sovelluksissa, entä perinteisissä?
- Missä tilanteissa SPA-arkkitehtuurin valitseminen on perinteistä sovellusarkkitehtuuria kannattavampaa?

Tutkimuksessa ei rakenneta kummankaan arkkitehtuurityylin mukaista esimerkkisovellusta, vaan tutkimus pohjautuu täysin teoreettiseen tietoon, omiin havaintoihin sekä haastatteluihin. Tutkimuksen tavoitteena ei myöskään ole tarkoitus perehtyä tarkasti palvelinpuolen toimintaan, vaan tutkimuksen pääpainopiste on asiakaspuolella, eli selainpäässä. Palvelinpuolen toimintaan perehdytään siinä määrin, että ymmärretään selaimen ja palvelimen välisen tiedonvälityksen periaatteet.

1.2 Käsitteet

Ajax (Asynchronous JavaScript and XML)

Sovellusliittymä, joka mahdollistaa asynkronisen kommunikoinnin selaimen ja palvelimen välillä.

ACSII (American Standard Code for Information Interchange)

Tietokoneen 7-bittinen merkistö, joka on laajuudeltaan 128 merkkipaikkaa.

CSS (Cascading Style Sheets)

Ohjelmointikieli, jolla voidaan määrittää näytettävän sovelluksen tai sivuston ulkoasua ja rakennetta.

DOM (Document Object Model)

HTML- ja XML-dokumenttien mallintamiseen käytettävä dokumenttipuu.

ECMAScript

ECMA internationalin standardoima skriptikieli. ECMAScript-kieliä käytetään selainpäässä.

Framework

Sovelluskehys on ohjelmistokehityksen apuvälineenä käytetty kehys, jonka tavoitteena on mahdollistaa helpompi sovelluksen kehitettävyys tarjoamalla kehitettävälle sovellukselle valmis runko.

HTML (HyperText Markup Language)

on johtava ohjelmointikieli www-dokumenttien luomiseen. HTML:n avulla määritellään web-sovelluksen rakenteellinen ulkoasu käyttäen erilaisia HTML-elementtien sisään rakennettuja tageja ja attribuutteja. (Taft, 2014, 9.)

HTTP (Hypertext Transfer Protocol) & HTTPS (Hypertext Transfer Protocol Secure)

Salattuun (HTTPS) tai salaamattomaan (HTTP) tiedonsiirtoon käytetty protokolla.

JavaScript

Selainpäässä toimiva, web-sivujen dynaamisen toiminnallisuuden luomiseen käytettävä skriptikieli.

MVC (Model-View-Contoller)

Ohjelmistoarkkitehtuurin malli, jonka avulla käyttöliittymä erotellaan sovelluslogiikasta. Sovellus jaetaan kolmeen toiminnalliseen osaan; malliin, näkymään ja ohjaimeen ja niiden avulla kuvataan tiedon tallennusta, käsittelyä sekä ylläpitoa.

MV* (Model-View-whatever)

MV* on ohjelmistoarkkitehtuurimallien yleisnimike, joka sisältää toisistaan poikkeavia MV-malleja (kuten MVC-malli). MV*-mallien avulla pyritään erottelemaan käyttöliittymä sovelluslogiikasta.

SPA (Single-Page Architecture)

Sovellus, joka perustuu liiketoimintalogiikaltaan selainpäähän ja muodostuu yhdestä HTML-dokumentista.

RESTful (Representational state transfer)

Ohjelmistoarkkitehtuurin malli, sekä rakenneperiaate asiakasohjelman ja palvelimen väliselle yhteydelle.

URL (Uniform Resource Locator)

Www-sivujen osoittamiseen käytettävä yksilöllinen merkkijono, jolla voi luoda linkkejä www-sivuun, sekä eri sivujen välille.

W3C

The World Wide Web Consortium on www-standardeja kehittävä yhteisö.

XHR (XMLHttpRequest)

XHR on olio, jonka avulla Ajax sovellusliittymällä muodostetaan asynkroninen tai synkroninen kommunikointi selaimen ja palvelimen välille.

2 Web-teknologiat

Kaikki selainmoottorilla näytettävät sivustot ja sovellukset käyttävät erilaisia web-teknologioita. Niiden avulla luodaan käyttäjälle näkymä, jollaisena hän näkee sovelluksen sitä käytettäessä. Se, mitä tekniikkaa missäkin määrin käytettävä sovellus tarvitsee, riippuu täysin siitä, mihin tarkoitukseen sovellus on tehty. Staattinen sivu tarvitsee vain tietyt HTML- ja CSS-määrittelyt, kun taas paljon toiminnollisuutta sisältävään sivuun lisätään runsaasti JavaScriptiä. Seuraavissa kappaleissa esitellään SPA-arkkitehtuurin kannalta tärkeimmät web-tekniikat, joista myös osaa käytetään perinteisissä web-sovelluksissa. (Lehdonvirta & Korpela, 2013, 12-13.)

Käsitteistössä määritellyt HTML ja CSS ovat osana kumpaakin sovellusarkkitehtuuria. Niin perinteiset kuin SPA-arkkitehtuuria noudattavat sovellukset on rakennettu käyttäen HTML- ja CSS-määrittelyjä. Perinteisessä sovelluksessa selain näyttää erilaisia palvelimelta ladattuja HTML-dokumentteja käyttäjän toimien mukaisesti, kun taas SPA-arkkitehtuurin sovelluksissa itse ulkoasullinen osa HTML:sta voi olla vähäinen. HTML-osuus on voitu koostaa muun muassa pelkistä script- sekä style- elementeistä, joilla otetaan JavaScript- ja CSS -koodeja käyttöön. (Lehdonvirta & Korpela, 2013, 32.)

2.1 JavaScript

JavaScript on ECMAScript-standardin mukainen selainpäähän painottuva ohjelmointikieli, jota ylivoimainen enemmistö verkkosovelluksista, sivuista ja selaimista käyttää. Tästä syystä JavaScript on eniten läsnä kaikista ohjelmointikielistä. Kun HTML:n ja CSS:n avulla luodaan sovelluksen rakenne, JavaScript määrittää puolestaan sen, miten sovellus käyttäytyy. Sen avulla luodaan sivun dynaaminen toiminta. (Flanagan, 2011, 1.)

JavaScript on jaettavissa kolmeen eri osa-alueeseen:

- Selainpään (Client-side) JavaScript on web-selaimiin ja sovelluksiin kohdentuvien toimintojen parantamiseen ja käsittelyyn.
- Palvelinpään (Server-side) JavaScript on tarkoitettu palvelinpään toimintoihin.

– JavaScript kieli (Core JavaScript), jolla tarkoitetaan JavaScript-kieltä itsessään. Core JavaScript on olennainen osa niin selain- kun palvelinpuolen JavaScriptiä.

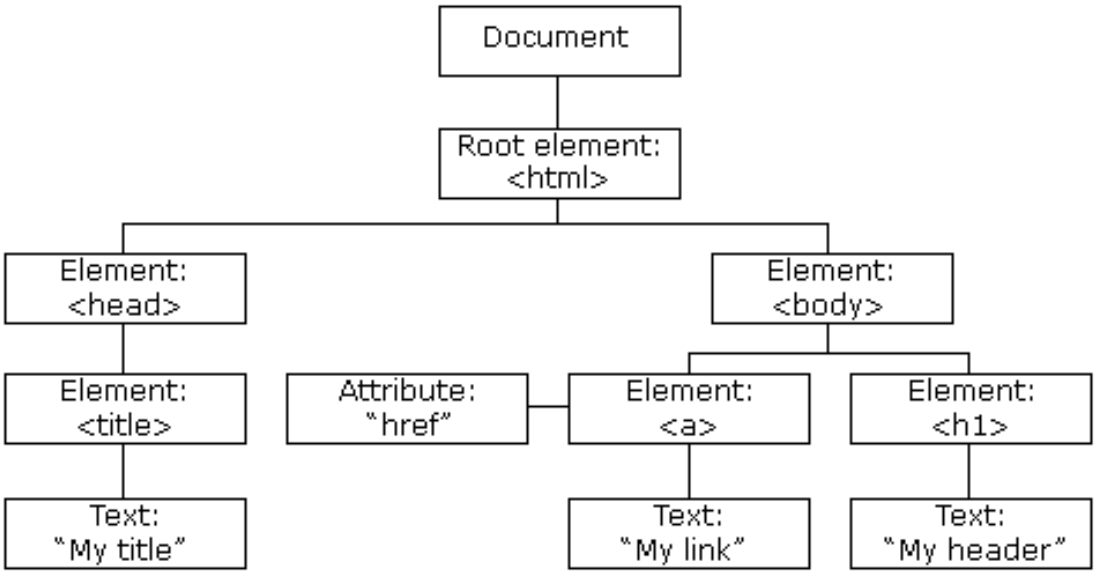
(Flanagan, 2011, 4, 289, 307.)

Lehdonvirta & Korpela toteavat (2013, 35) JavaScriptin käytön korostuvan varsinkin web-selaimissa, minkä vuoksi se on olennainen osa SPA-sovellusta. Sovelluksen toiminnalliset osat suoritetaan käyttäen JavaScriptiä, jolloin esimerkiksi HTML:n määrä voi vähetä, eikä JavaScript toimi enää sivuille annettavan lisähöysteen muodossa. Esimerkkinä JavaScriptin vaikutuksesta SPA-ja perinteisten sovellusten välillä voi pitää sitä, että hyvin koodattu perinteinen web-sovellus jatkaa toimimistaan JavaScriptin poistettaessa. JavaScriptin ollessa SPA-sovelluksen perusta, ei sovelluksen odoteta toimivan ilman sitä. (Flanagan, 2011, 311.)

2.2 DOM

DOM (Document Object Model) on W3C:n määrittämä yhteinen ohjelmointirajapintana toimiva dokumenttipuu, joka toimii kaikissa selaimissa sekä ympäristöissä ohjelmointikielestä riippumatta. DOM tarjoaa jäsennellyn esityksen käytettävästä dokumentista ja määrittää tavan, jolla tiedoston rakenteeseen pääsee käsiksi ohjelmilla niin että sitä voi muuttaa. Periaatteessa DOM mallintaa dokumentin näyttämisen ja muokkaamisen kannalta olennaisia olioita. Mallin avulla on saatu aikaiseksi standardoitu dokumenttien käsittelytapa, joka helpottaa dynaamisten sovellusten tiedon käsittelyä.(Asleson & Schutta, 2006, 39.)

Sisäkkäin olevat HTML-dokumenttien elementit näytetään DOMissa hierarkisena puunmallisena rakenteena, jonka osia kuvataan solmuina (node) (kuva 1). Puun solmut edustavat HTML-tageja -sekä elementtejä, kuten <body> ja <title>, sekä merkkijonoja ja kommentteja. Jos koodi on jostain syystä muodoltaan sopimatonta tai vajavaista, muodostaa DOM korjatun version puuhun. (Flanagan, 2011, 361-264.)



Kuva 1, Document Object Model- malli (http://www.w3schools.com/js/js_html-dom.asp)

DOM itsessään ei ole ohjelmointikieli, vaan rajapinta, jonka avulla HTML dokumenttien käsittely on tehty helpommaksi. Sen avulla määritellään, miten skripti käsittelee dokumentin rakennetta. Document Object Model -mallia käytetään varsinkin Ajaxin avulla usein yhteydessä JavaScriptin kanssa, jolloin on mahdollista sekoittaa se JavaScriptin osaksi. JavaScript ohjelmointikielenä käyttää hyväkseen Document Object Model -mallia. Dokumentin osat ovat myös osia Document Object Model -mallista, joten sen ominaisuudet ja metodit otetaan käyttöön käyttäen JavaScriptiä. (Asleson & Schutta, 2006, 45.)

2.3 Http-protokolla

Viestintään palvelimen ja selaimen välillä käytetään HTTP-yhteyttä (HyperText Transfer Protocol), joka on määritelmä standardeja, joiden avulla määritellään, miten informaatiota webin (World Wide Web) sisällä formatoidaan ja siirretään. Päästäkseen käsiksi web-sivulle, asetetaan sivun osoitteen eteen HTTP://, jonka avulla selaimelle annetaan informaatio kommunikoida HTTP-yhteyden kautta. Modernit selaimet eivät kuitenkaan enää tarvitse HTTP:// -liitettä erikseen, sillä se on asetettu kommunikaation oletusmetodiksi. HTTP:tä kutsutaan myös tilattomaksi protokollaksi, sillä sen jo-

kainen komento toteutetaan itsenäisesti, ilman tietoa aiemmin tapahtuneista komennoista. Tilattomuus on ollut yksi syy siihen, miksi on haasteellista tehdä web-sivuja, jotka vastaavat käyttäjän toimiin heti. (w3schools.com.)

Selaimen tarvitessa uutta tietoa palvelimelta, lähettää se palvelimelle HTTP-pyynnön (request) johon palvelin vastaa HTTP-vastauksen (response) avulla, lähettäen takaisin selaimen tarvitseman informaation. Kaksi yleisimmin käytettyä metodia pyyntö-vastaus-sykliin ovat GET ja POST -metodit. GET-metodilla pyydetään dataa ja POST-metodilla lähetetään. Metodit ovat samankaltaisia, mutta niihin kohdistuvista toiminnoista löytyy muutama selkeä eroavaisuus. (w3schools.com.)

Taulukko1, GET- ja POST- metodien eroavaisuuksia

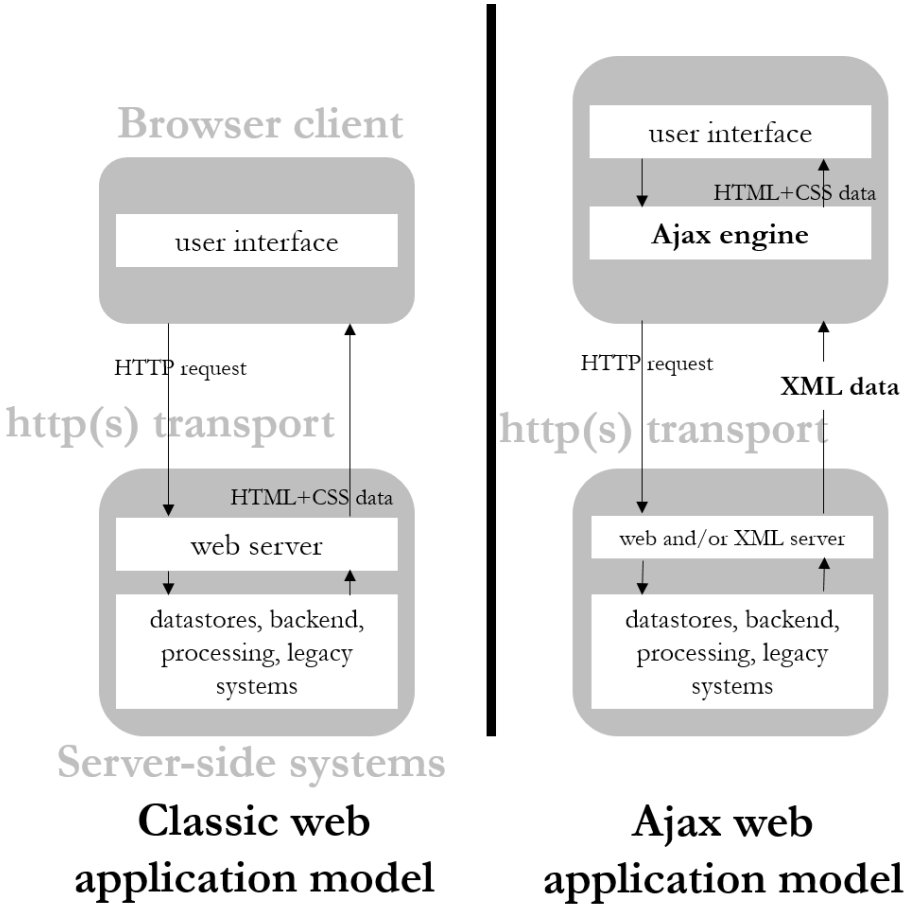
Toiminto	**GET**	**POST**
Lisäys kirjanmerkkeihin	kyllä	ei
Merkintä välimuistiin	kyllä	ei
Tallennus sivuhistoriaan	kyllä	ei
Rajoitteet datan pituudessa	kyllä (GET-metodi lisää lähetettävän datan URL-osoitteeseen.)	ei
Rajoitteet datan tyypissä	kyllä (vain ACSII-merkit sallittu)	ei
Näkyvyys	kyllä (data on näkyvissä URL-osoitteessa)	ei

Taulukon tietoja tarkastellessa voi huomata, että POST-metodi on hieman GET-metodia turvallisempi, sillä POST lähettää parametrit osana pyyntöä, kun taas GET sisällyttää ne URL-osoitteeseen. Tästä syystä esimerkiksi salasanoja ei pitäisi koskaan lähettää käyttäen GET-metodia, sillä salasana näkyisi URL-osoitteessa. GET-metodia käytetään useammin tiedon vastaanottamiseen, kun taas POST-metodia suositellaan käytettäväksi muutettaessa tietomallin tilaa (esimerkiksi tiedon tallennus). (Asleson & Schutta, 2006, 58.)

2.4 Ajax

Ajax on tunnettu sovellusliittymä, jonka avulla web-sivut ja sovellukset käyttävät interaktiivisesti JavaScriptiä, ja jota lähes kaikki selaimet tukevat tänä päivänä. Ajaxia käytettäessä JavaScript kommunikoi suoraan palvelimen kanssa. Tällöin HTTP-pyynnöllä sivu voi tehdä sekä pyynnön, että saada vastauksen palvelimelta lataamatta uudelleen sivua. Käyttäjä ei välttämättä huomaa skriptin tapahtumia, vaan palvelimen kanssa toimiminen tapahtuu sovelluksen taustalla. (Lehdonvirta & Korpela, 2013, 55.)

Ajaxin voidaan ymmärtää tarkoittavan skriptin käynnistämää HTTP- tai HTTPS- yhteyttä, joka ei kuitenkaan johda sivun uudelleenlataukseen. Se kommunikoi palvelimen kanssa asynkronisesti, jolloin käyttäjäkokemuksesta vähenee pyyntö/vastaustyylinen sykli. Asynkronisten pyyntöjen lisäksi Ajaxin avulla on mahdollista suorittaa synkronisia pyyntöjä. Ajaxin ollessa käytössä ja käyttäjän aloittaessa uuden toiminnon, JavaScriptin ja HTML:n on mahdollista päivittää hetkessä käyttäjän näkymä, samalla muodostaen yhteyden palvelimeen. Pyynnön palatessa HTML ja CSS päivittyvät vastaavasti aiheuttamatta sivulatausta (Kuva 2). Tarkoituksena on, ettei käyttäjä välttämättä huomaa koodin kommunikoivan samanaikaisesti palvelimen kanssa ja sovellus tuntuu vastaavan toimintoihin heti. (Teare, 2005.)



Kuva 2, Perinteisen sovelluksen ja SPA-sovelluksen kommunikointi palvelimen kanssa (Garret, 2005)

Itsessään Ajax ei ole yksi ainoa teknologia, vaan se on nimi sovellusliittymälle, jonka sujuvaan toimivuuteen kuuluu useampi erillinen teknologia. Ajaxiin sisältyy muun muassa seuraavia teknologioita; standardeihin perustuvan näkymän esittämiseen HTML ja CSS, dynaamiseen näyttämiseen ja vuorovaikutukseen Document Object Model (DOM). Asynkroniseen kommunikointiin palvelimen kanssa käytetään XHR-pyyntöä. Edellä mainittujen teknologioiden toiminnat taas nidotaan yhteen käyttäen JavaScriptiä. Näiden teknologioiden toimiessa samassa komponentissa, muodostuu Ajax. (Asleson & Shcutta, 2006, 31-34.)

Ajax ei toimi perinteisessä sovellusrakenteessa totutun pyyntö-vastaus vuorovaikutuksen mukaisesti, vaan sen toiminta kulkee alla kuvatulla tavalla. Liitteessä 1 on nähtävissä XHR eli XMLHttpRequest-yhteyden muodostus kokonaisuudessaan.

1. Käyttäjä suorittaa sovelluksessa tapahtuman, esimerkiksi yksinkertaisen onChange-tapahtuman, jossa lomakkeelle syötetty tieto tarkistetaan. Tämä käynnistää Ajaxin-toiminnan.
2. Seuraavaksi muodostetaan XHR-yhteys (XMLHttpRequest) palvelimeen käyttäen open()- ja send()- metodeja. Samalla kerrotaan, millä metodilla palvelinta lähestytään (GET/POST) ja lisätään tarkistettava informaatio URL-parametriin. Jotta suoritus on Ajaxin toimintaperiaatteen mukainen, tarkistetaan, että open()-metodi on varmasti asynkroninen.
3. Seuraavaksi pyyntö lähetetään palvelimelle käyttäen send()-metodia.
4. Vastaus palautetaan selaimelle.
5. Selain ryhtyy käsittelemään vastaanottamaansa dokumenttia käyttäen DOM-dokumenttipuuta mallina (Asleson & Schutta, 2006, 29-31.)

Ajaxia ei käytetä perinteisissä vastaus-pyyntö-sykliin perustuvissa sovelluksissa, vaan se on selkeämmin yhteen sivuun perustuvien sovellusten osa. Se luo mahdollisuuden asynkroniseen kommunikointiin selaimen ja palvelimen välillä. Tästä syystä SPA-sovelluksia kutsutaan joissain tapauksissa myös Ajax-sovelluksiksi. Ajax on osa, joka mahdollistaa, että käyttäjä voi samalla jatkaa sovelluksen käyttöä, vaikka samanaikaisesti suoritetaan pyyntöjä palvelimelle. (Flanagan, 2011, 311.)

2.5 RESTful- periaatteet

REST on Roy Fieldingin kehittämä arkkitehtuurimalli, jonka tarkoituksena on mahdollistaa toiminta hypermediajärjestelmissä, joiden osat kehittyvät eriaikaisesti, sekä toisistaan riippumattomasti. REST on niin sanottu rakenneperiaate asiakasohjelman, sekä palvelimen väliseen yhteyteen ja sitä käytetään HTTP-protokollan yhteydessä. RESTful-periaatteet eivät ota kantaa itse ohjelmiston osien toteutustapaan, mutta ne asettavat ohjelmistolle tiettyjä rajoitteita, joiden avulla muodostetaan käytettävä, yhteinen rajapinta. RESTful-periaatteita noudattavat sovellukset käyttävät HTTP-yhteyttä kaikkiin datan välitykseen liittyviin CRUD-toimintoihin (Create, Read, Update, Delete), eli datan näyttämiseen, siirtoon, poistoon ja päivittämiseen. (Fielding, 2000.)

Korpelan ja Lehdonvirran (2013, 54) mukaan REST-periaatteilla on tiettyjä tunnuspiirteitä.

- Ne noudattavat asiakas-palvelin-mallia, jolloin dataa liikutetaan asiakaspuolen pyynnön tapahtuessa.
- Käytettävissä on HTTP-yhteyskäytäntö ja sen metodit GET, POST, PUT (resurssin tallennus palvelimelle) ja DELETE (resurssin poisto palvelimelta).
- Tilattomuus, joka tarkoittaa sitä, että jokainen lähetetty pyyntö sisältää kaiken tarvittavan tiedon, minkä palvelin tarvitsee vastatakseen pyyntöön.
- Resursseihin viittaaminen. Kaikki nimettävät asiat (kuten tiedosto, kuva) ovat resursseja ja niihin viitataan URL-osoitteella.

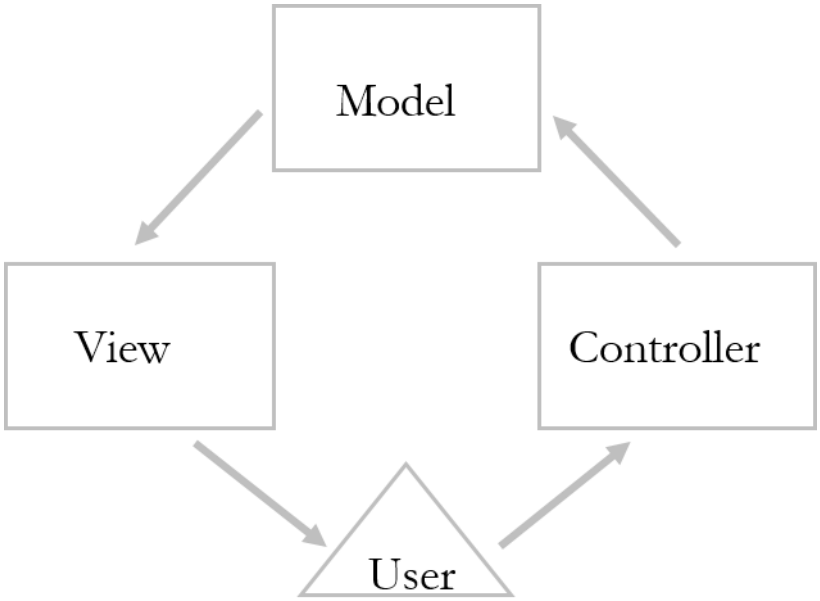
SPA-sovellusta kehitettäessä pyritään useimmiten noudattamaan RESTful-periaatteita, mutta niiden noudattaminen perinteisissä sovelluksissa ei ole myöskään tavatonta. (Heiskanen, 2013.)

2.6 MV*-arkkitehtuuri ja MVC-malli

Monet sovelluskehykset noudattelevat MV*- malliin kuuluvaa arkkitehtuurirakennetta, joista yleisimmin käytetty MVC-malli (Model-View-Controller) toimii monien JavaScript-sovelluskehysten pohjana. MVC-malli ei kuitenkaan ole ainoa laatuaan vaan samantapaisia, toisistaan hieman poikkeavia malleja löytyy useita. Tästä syystä niistä käytetään yhteistä termiä MV* (Model-View-Whatever). MVC-malli koostuu kolmesta osasta, joiden avulla pyritään erottelemaan käyttöliittymä itse sovelluslogiikasta. (Osmani, 2012.)

MVC-mallille olennaisia piirteitä ovat Model (malli), View (näkymä) ja Controller (ohjain). Mallin avulla hallitaan sovelluksen dataa. Näkymällä tarkoitetaan käyttöliittymän ulkoasua, eli visuaalista näkymää mallista ja siitä, mitä valittuja osia siitä vuoroin näytetään käyttäjälle. Sovelluksen toiminnallisuus, interaktiot ja sen tila tapahtuvat mallissa. Ohjaimen avulla käsitellään tapahtumaa mallin ja näkymän välillä, eli sillä aiheutetaan

käyttäjän toimintoja vastaavia tilamuutoksia. Kun käyttäjä suorittaa toimintoja näkymässä, on mallin ja näkymän tehtävänä päivittyä, näiden päivitysten ohjaamiseen käytetään ohjainta (kuva 3). (Osmani, 2012.)



Kuva 3, MVC-malli

2.7 Sovelluskehykset

Sovellusta rakennettaessa käytetään usein apukeinona erilaisia sovelluskehyksiä (framework). Kehyksiin sisällytetään jo valmiiksi rakennettuja ohjelmisto-osia, kuten funktioita ja moduuleja. Tällöin kehysten käyttö kehityksen apuna tekee ohjelmiston toteutuksesta nopeamman ja sujuvamman prosessin. Niin perinteiseen, kuin SPA-arkkitehtuurin on tarjolla useita eri sovelluskehysvaihtoehtoja ja omaan käyttötarpeeseen oikean kehyksen valitseminen voi olla haasteellista. Kehittäjän on myös mahdollista rakentaa oma sovelluskehyksensä, tai tehdä sovellus kokonaan ilman sitä. (Jäppinen, 2014).

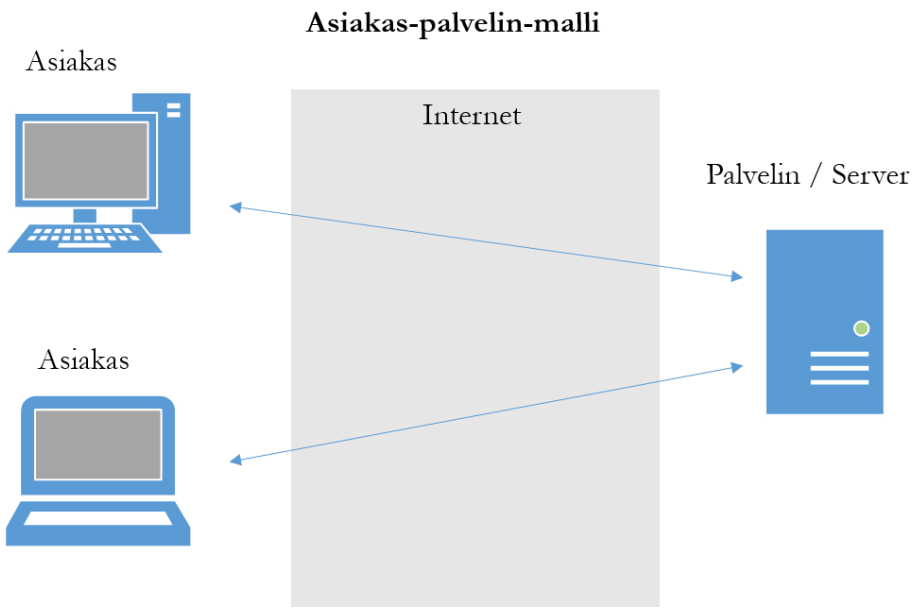
Sovelluskehykset perinteisessä- ja SPA-arkkitehtuurissa eivät perusperiaatteiltaan eroa suuremmin toisistaan. Ne tarjoavat käyttäjälle valmiita toiminnallisuuksia, valmiiksi testattua koodia ja ikkunanhallintaa. Ohjelmointikielestä riippumatta sovelluskehystarjonta

on laajaa. Varsinkin yhden sivun sovelluksille on viime vuosien aikana kehitetty runsaasti käytännölliseen MV*-malliin perustuvia uusia sovelluskehyksiä, jotka ovat itsessään nostaneet SPA-sovellusten kehitysmäärää. (Heiskanen, 2013.)

Heiskasen (2013) mukaan MV*-malleihin lukeutuvia JavaScript-sovelluskehyksiä ovat esimerkiksi AngularJS, Ember.js ja Backbone.js. Edellä mainituista sovelluskehyksistä Ember.js noudattelee eniten MVC-mallia. Sen vahvin ominaispiirre on automaattinen objektien alustaminen ja luominen käyttäen erilaisia nimeämiskäytäntöjä. Tästä syystä Ember.js määrittää pitkälti kehitettävän sovelluksen arkkitehtuurin. Backbonen ominaisuudet ovat kolmesta sovelluskehyksestä yksinkertaisimmat, eikä se omasta puolestaan aseta laajoja arkkitehtuurillisia rajoitteita. Backbonea suositellaan käytettäväksi tapauksissa, jossa valtaa ohjelmiston arkkitehtuurin toteutuksesta halutaan vierittää enemmän kehittäjän harteille. AngularJS on Googlen kehittämä sovelluskehys, joka tarjoaa hieman laajemman pohjan kehitettävää sovellusta varten. AngularJS:n erityisominaisuuksia ovat HTML-merkintöjen laajennusmahdollisuus käyttäen omia komponentteja, sekä mallimuutosten kuunteleminen. Nämä ovat ominaisuuksia, joita on odotettavissa tulevaisuuden sovelluksiin. (Heiskanen, 2013.)

3 SPA-arkkitehtuuri ja perinteinen sovellusrakenne

Webissä on yleisesti käytössä asiakas-palvelin-toimintamalli (kuva 4). Tämä malli koostuu asiakasohjelmasta ja palvelinohjelmasta, sekä niiden välisestä yhteyskäytännöstä. Käyttäjän käyttäessä sovellusta, asiakasohjelma lähettää pyyntöjä palvelinohjelmalle, joka vastaa niihin lähettämällä dataa. Yhteyskäytäntö määrittää sen, kuinka nämä kaksi kommunikoivat keskenään. Webiä käsiteltäessä, asiakkaalla tarkoitetaan yleensä selainta ja palvelimella sovellusta, joka ylläpitää www-sivua. (Medianurkka, 2014.)



kuva 4, Asiakas-palvelin-malli

Vastaanottaakseen tai hakeakseen informaatiota, sovelluksen on toimittava yhteydessä palvelinsovelluksen kanssa vähintään kerran, sivun ensilatauksella. Se, kuinka usein ja millä tavoin palvelimeen otetaan yhteyttä, riippuu sovelluksen tyypistä ja sisällöstä. Jos kyseessä on staattinen dokumentti, tarvitaan pyyntö ladata sivu ensimmäisen kerran. Jos sivustolla liikutaan uusiin näkymiin, tai ladataan uutta tietoa, tarvitaan useampia pyyntöjä.

Selaimessa näytettävien sovellusten ja sivujen määrä on laaja. Osa niistä on staattisia, pelkästään informaatiota sisältäviä, ja niihin on mahdollisesti lisätty dynaaminen toiminta käyttäen JavaScriptiä. Osa taas on kokonaan JavaScriptin varaan rakennettuja,

erittäin graafisia ja toiminnoiltaan monipuolisia. Osassa yhdistyvät kummankin ominaisuudet. (Flanagan, 2013, 307.)

3.1 Sovellukset ja verkkosivut

Verkkosivu on internetissä julkaistu sivu, jonka pääasiallisena tarkoituksena on jakaa informaatiota. Sivut ovat joko staattisia, eli muuttumattomia, ellei niiden koodia tarkoituksella muuteta, tai dynaamisia, jolloin selaimen kohdistama pyyntö palvelimelle käynnistää uuden www-sivun latauksen. Kooltaan ja ulkoasultaan verkkosivut voivat olla hyvinkin erinäköisiä, mutta niille yhteinen piirre on linkkien kautta tapahtuva sivunvaihto. Usean verkkosivun muodostamia, samaa aiheitta käsitteleviä sivujoukkoja kutsutaan verkkosivustoiksi. (Borodescu, 2014.)

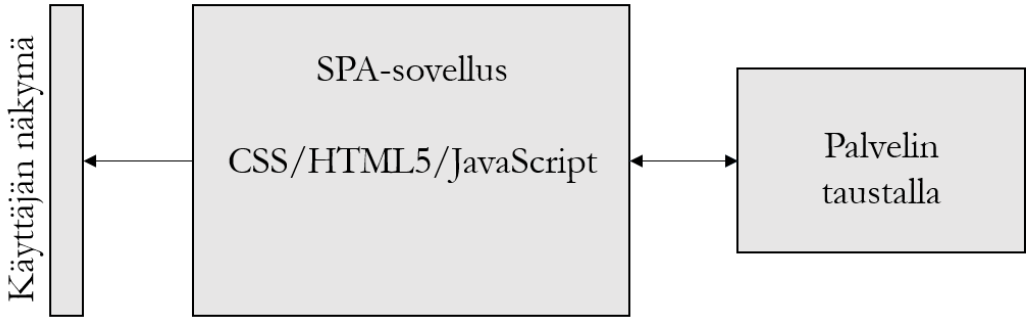
Sovelluksesta puhuttaessa skaala erilaisten sovellusten välillä on laaja. Sovelluksella (web-application) voidaan tarkoittaa perinteistä verkkosivun tyylistä sovellusta, johon on rakennettu joku tietty toiminto ja joka toimii perinteisen verkkosivun tavoin pyyntövastaus-tyylisessä vuorovaikutuksessa palvelimen kanssa. Sovelluksesta puhuessa voidaan myös tarkoittaa esimerkiksi mobiilisovellusta, jonka käyttö saattaa vaatia ainoastaan yhden latauskerran, eikä lainkaan internet-yhteyttä ja joka ei rakenteellisesti muistuta missään määrin verkkosivua. Sovelluksia voidaan rakentaa toimimaan palvelinlähtöisesti ja selainlähtöisesti. Tulevissa kappaleissa painotetaan juuri selainlähtöiseen toimintaan perustuvan, yhden HTML-sivun varaan rakennetun sovelluksen toimintaa verrattuna palvelinpainotteisesti toimiviin sovelluksiin ja sivuihin. (Borodescu, 2014.)

Toisinaan raja sovelluksen ja sivun, sekä eri arkkitehtuurimallien välillä voi olla hyvinkin häilyvä. Yhä useammin rakennetaan sivuja tai sivustoja, joihin lisätään sovellus, tai sovelluksen ominaisuuksia. Toisinaan valmiin sivun päälle rakennetaan jopa kokonaan uusi SPA-toiminnollisuus. Käyttäjän onkin joissain tapauksissa lähes mahdoton havaita, käyttääkö hän sivua vai sovellusta ja toimiiko sivusto pääsääntöisesti palvelimen vai selaimen kautta. Toisaalta käyttäjän ei ole edes välttämättä oleellista huomata eroa. (Jäppinen, 2014.)

3.2 Toimintaperiaatteet

SPA-sovellus

Yhden sivun sovellukset (Single-Page Application) perustuvat liiketoimintalogiikaltaan käyttäjän selainpäähän ja ne on toteutettu JavaScript-ohjelmointikielellä. Rakenteeltaan SPA-sovellus on yksisivuinen dokumentti, jonka sisältö ja toiminta muuttuvat käyttäjän toimiessa. Useimmiten sovellus on yhdistetty taustalla toimivaan palvelimeen, ja rajapintana toimiva Ajax suorittaa käyttäjän lähettämät pyynnöt niin, ettei koko sivua ladata uudelleen jokaisen pyynnön yhteydessä (kuva 5). (Heiskanen, 2013.)



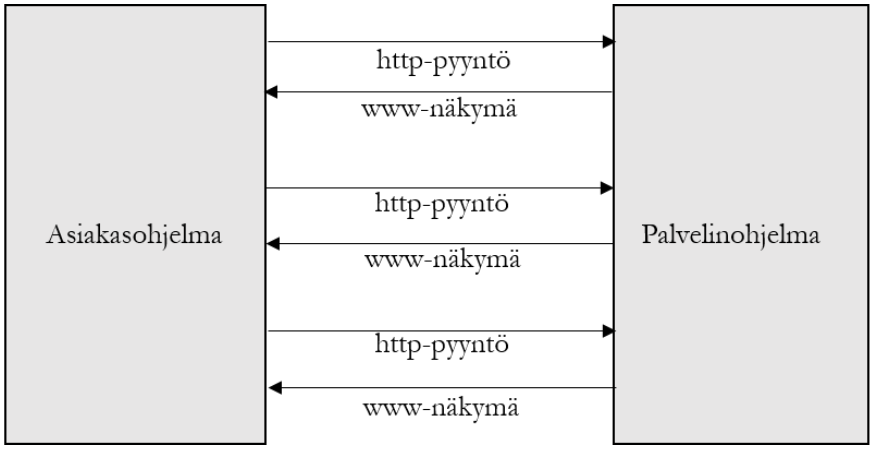
Kuva 5, SPA-sovelluksen rakenne yksinkertaistettuna

Käytännön kannalta huomattava seikka yksisivuisessa sovelluksessa on sen sivulatausten määrän vähentyminen. Tämä tarkoittaa sitä, että sovelluksessa nähtävä sisältö muuttuu, vaikkei käyttäjä siirry uudelle sivulle. Käytännössä sovellukseen rakennetut JavaScript-toiminnot ohjaavat sovellusta näyttämään käytön kannalta tarvittavia osia, kun taas uusi sisältö haetaan palvelimelta. (Lehdonvirta & Korpela, 2013, 35).

Perinteinen web-sovellus

Perinteisen web-sovelluksen toiminta perustuu puolestaan palvelinpäähän. Siinä palvelimen puolelle rakennettu sovellus näyttää HTML-dokumentin sisältöä käyttäjän antamien syötteiden mukaisesti. Liikkuminen sivuston sisäisesti suoritetaan siirtymällä sen hetkisestä tilasta seuraavaan, käyttäen sivunvaihtoa, www-palvelin vastaanottaa HTTP

-pyyntöjä ja lähettää ne takaisin staattisina sivuina (kuva 6). Tällöin joka siirrolla ladataan uusi sivu palvelimelta. (Heiskanen, 2013.)

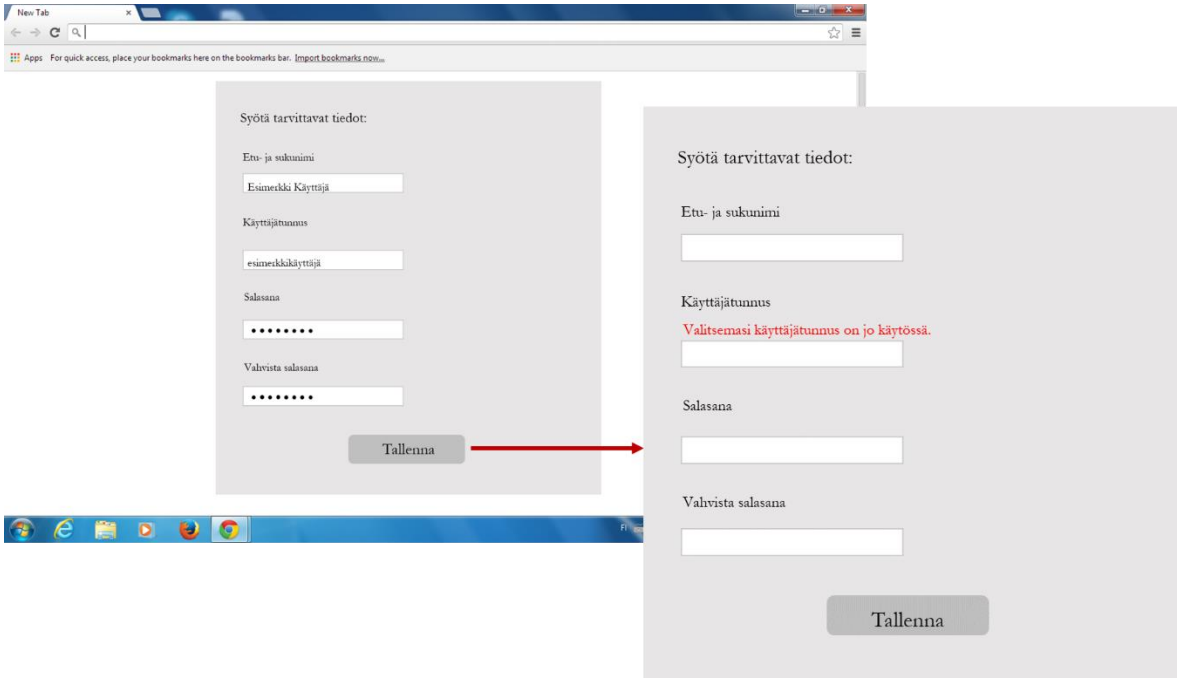


Kuva 6, perinteisen sovellusarkkitehtuurin toimintaperiaate

3.3 Esimerkit SPA-sovelluksen ja perinteisen toiminnasta

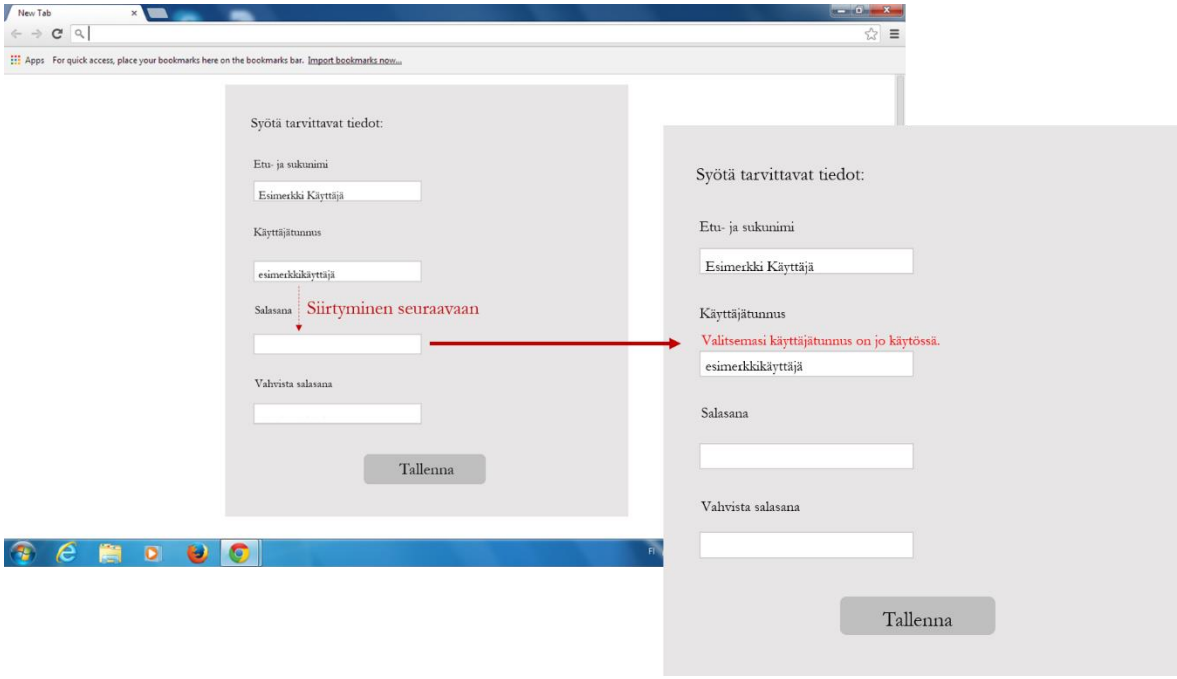
Kuvataan esimerkein eroavaisuuksia SPA-sovelluksen, sekä perinteisen web-sovelluksen käytössä. Käytetään esimerkkitapauksina käyttäjätunnuksen luomista, sekä hakutoimintoa.

Kuvassa 7 kuvataan, kuinka perinteisessä web-sovelluksessa varaustapahtumassa käyttäjän on navigoitava alusta loppuun varauslomake läpi, lisätäkseen tarvittavat tiedot, kuten käyttäjätunnus ja salasana. Käyttäjä joutuu syöttämään tiedot ja odottamaan, kun lomake kommunikoi palvelimen kanssa tarkastaakseen, ovatko syötetyt tiedot sääntöjen mukaisia. On mahdollista, että syötetty käyttäjätunnus on jo varattu ja huonoimmassa tapauksessa palvelimelta palautuu lomake, jonka tiedot ovat nollautuneet ja kaikki jo syötetyt tiedot on lisättävä uudelleen. (Clark, 2006.)



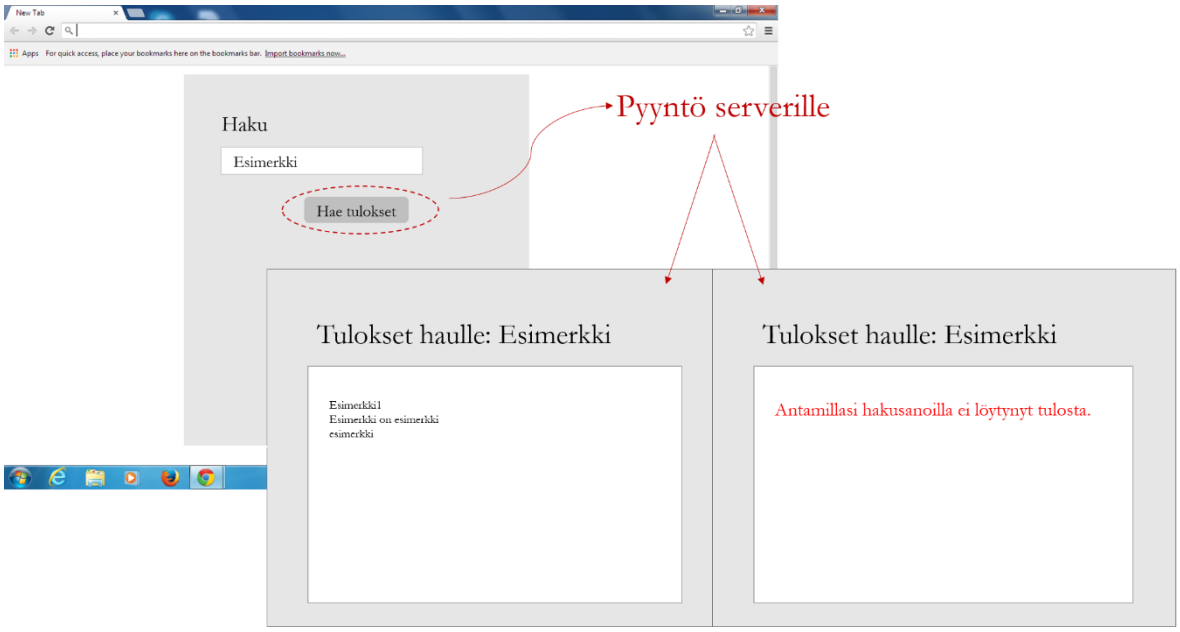
Kuva 7, tietojen tallentaminen perinteisessä web-sovelluksessa

Kuvassa 8 kuvataan, miten SPA-arkkitehtuurin avulla edellä mainittu tilanne voidaan välttää. Sen sijaan, että lähetettäisiin koko lomakkeen tiedot tarkistettavaksi palvelimelle, Ajax lähettää pyynnön palvelimelle tarkistaakseen vain valitun osan lomakkeesta (tässä tapauksessa käyttäjätunnuksen), toimien taustalla niin, ettei käyttäjä välttämättä huomaa pyyntöä. Jos käyttäjätunnus on viallinen tai varattu, pystytään käyttäjälle antamaan tästä lähes reaaliaikainen ilmoitus lataamatta koko lomaketta uudelleen. (Clark, 2006.)



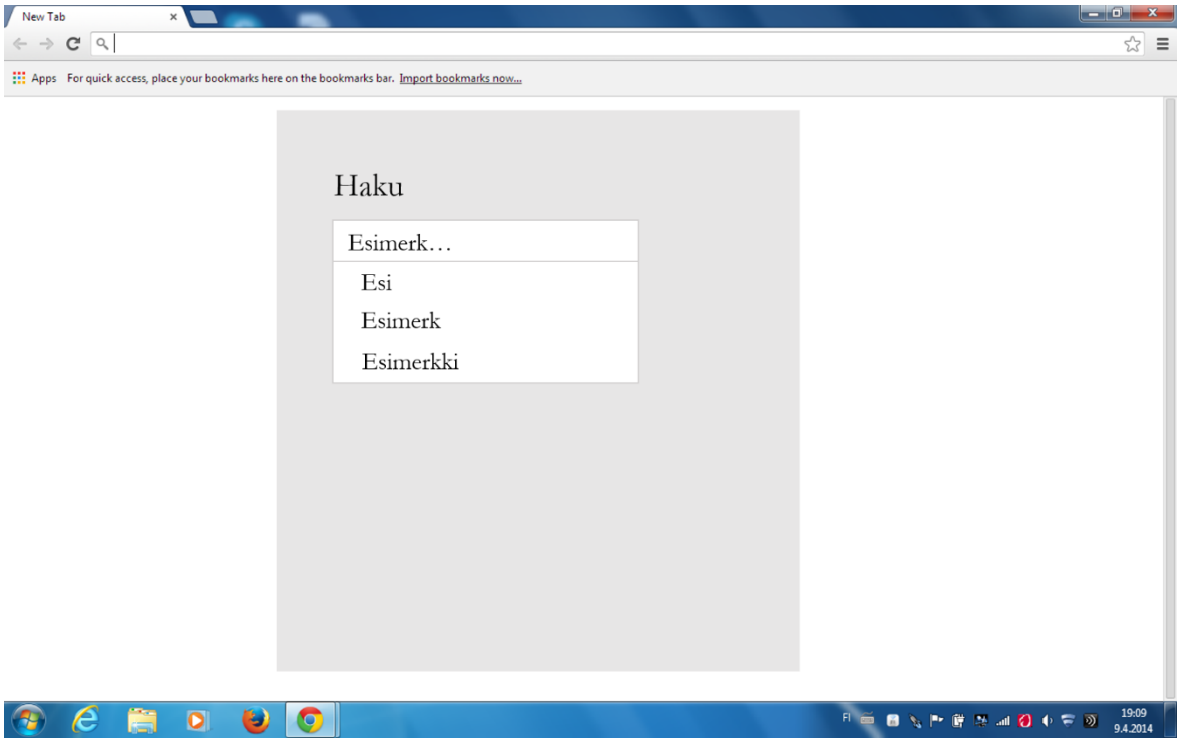
Kuva 8, tietojen tallentaminen SPA-sovelluksessa

Kuvassa 9 kuvataan informaation etsimistä hakutoimintoa käyttäen perinteisessä web-sovelluksessa. Perinteisessä web-sovelluksessa lomakkeen hakukenttään kirjoitetaan tarvittava tieto, jonka jälkeen painetaan hakupainiketta. Lomake lähettää pyynnön palvelimelle, joka vastaa ja lähettää selaimeen hakukriteeriä vastaavat tiedot ladattuna uudelle sivulle. Jos valitun hakusanan avulla ei löydy tietoa, palautuu tyhjä sivu tai ilmoitus siitä, ettei haettua tietoa löydy. (Clark, 2006.)



Kuva 9, esimerkki hakutapahtumasta perinteisessä web-sovelluksessa

Kuvassa 10 näytetään, kuinka SPA-sovelluksessa käyttäjän on mahdollista nähdä reaaliaikaisesti, löytyykö haettua informaatiota vastaan tietoa. Kun käyttäjä ryhtyy kirjoittamaan lomakkeeseen tietoa, kommunikoi selain Ajaxin kautta taustalla, lähettäen useita pyyntöjä palvelimeen ja päivittäen hakukentän alle tietoja samanaikaisesti käyttäjän kirjoittaessa. Käyttäjän ei täten välttämättä tarvitse painaa hakupainiketta nähdäkseen, löytyykö haettavaa tietoa. (Clark, 2006.)



Kuva 10, esimerkki hakutapahtumasta SPA-sovelluksessa

4 Tutkimus & tutkimusmenetelmä

Tutkimuksessa pyritään selvittämään vastausta tutkimusongelmaan; mikä on SPA-arkkitehtuuria käyttävän sovelluksen ja perinteisen web-sovelluksen ero? Jotta tutkimusongelma saataisiin ratkaistuksi, on se jaettu neljään tutkimuskysymykseen, joita tutkimalla vastaus ongelmaan löydetään.

Ensimmäinen tutkimuskysymys on; Mitä SPA-sovellukset sekä perinteiset web-sovellukset ovat, ja miten ne toimivat? Jotta pystytään vertailemaan kahden sovellusarkkitehtuurin välisiä eroja, on ensin tiedettävä miten ne toimivat ja mihin tarkoitukseen ne on alun perin kehitetty. Tästä syystä selvitetään SPA-sovelluksen, sekä perinteisen web-sovelluksen toimintaperiaate ja yleinen, kaikille sovelluksille yhtenäinen rakenne. Vastataan myös siihen, mihin käyttötarkoitukseen sovellukset soveltuvat.

Seuraavassa tutkimuskysymyksessä vastataan, minkälaisiin käytettävyysongelmiin on mahdollista törmätä kehittäjän, tai käyttäjän näkökulmasta. Ensimmäisessä tutkimuskysymyksessä selvitettyihin eroihin saadaan syvempää näkökulmaa käytettävyyden kautta. Minkälaisia ongelmia syntyy sovellusten eroavan rakenteen vuoksi niiden käytössä. Käytettävyysongelmien kautta on mahdollista havainnollistaa konkreettisia eroja vertailtavien sovellusarkkitehtuurien välillä.

Kolmas tutkimuskysymys kuuluu seuraavasti; millaisia hyötyjä on SPA-arkkitehtuuria noudattavassa sovelluksessa, entä perinteisessä? Aiemmissa tutkimuskysymyksissä on selvitetty toimintaperiaatteet, yleisimmät teknologiat ja ongelmat käytettävyydessä. Saatuja tuloksia tarkastelemalla saadaan poimittua kummankin sovellusarkkitehtuurin toimivat osat, jolloin saadaan selville, miten arkkitehtuurin käytöstä hyötyy.

Neljännessä tutkimuskysymyksessä käytetään hyväksi aiempien tutkimuskysymysten tuloksia. Vertailemalla niitä yhteen, pystytään vastaamaan kysymykseen; missä tilanteissa SPA-arkkitehtuurin valitseminen on perinteistä sovellusarkkitehtuuria kannattavampaa ja missä tilanteissa kannattaa pysyä perinteisessä sovellusarkkitehtuurissa?

Tutkimus toteutetaan kirjallisuustutkimuksena ja haastatteluina. Teoriataustan lähteinä käytetään erilaisia kirjallisuuslähteitä ja henkilöhaastatteluja. Erilaisia tutkimusmenetelmiä käytetään, jotta saadaan monipuolista ja myös näkökulmaltaan erilaista tietoa tutkimusongelman ratkomista varten. Tutkimuksessa ei vertailla ennalta valittuja sovelluksia tai sovelluskehyksiä, vaan pyritään saamaan kokonaiskuva arkkitehtuurieroavaisuuksiin.

Kirjallisissa lähteissä pääpainona ovat tutkimusaiheen tyypin ja ajankohtaisuuden takia internetartikkelit, tutkimukset ja myös kirjat. Kirjallisten lähteiden perusteella pyritään hahmottamaan kokonaisvaltainen kuva käsiteltävistä kahdesta arkkitehtuurista. Niiden avulla selvitetään mitä teknologioita arkkitehtuureihin kuuluu ja miten ne toimivat.

Haastatteluissa pyritään saamaan asiantuntijan näkökulma tutkimusaiheesta. Haastateltaville on valittu ennalta kysymykset, jotka ohjaavat vastaamaan tutkimuskysymyksiin. Haastattelun avulla saadaan käytännönläheistä tietoa, sekä aitoja kokemuksia ja näkökulmia. Haastatteluista saatuja tuloksia verrataan kirjallisiin lähteisiin, jolloin pystytään huomaamaan mahdolliset eroavaisuudet tulosten välillä.

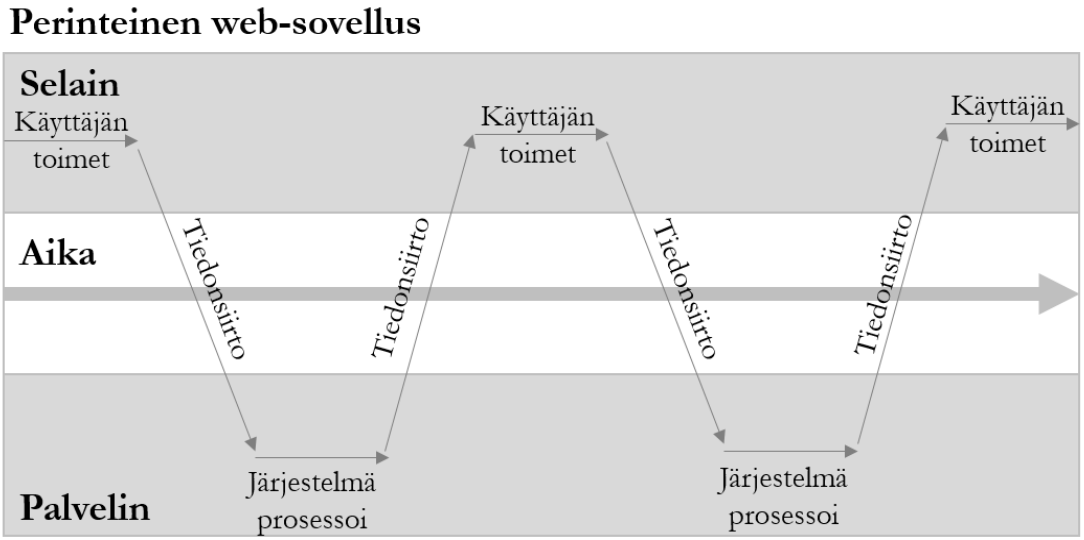
5 Tulokset

Webin käyttö on noussut huomattavasti viime vuosina. Samalla myös sivujen sisältämä datamäärä on kasvanut, mutta käyttäjät vaativat silti sovelluksilta sujuvaa interaktiota ilman pitkiä latausaikoja. Nopeasti kehittyneet selainpään teknologiaratkaisut ovat avanneet kehittäjille parempia mahdollisuuksia rakentaa jäsennellympiä yhden sivun arkkitehtuuriin perustuvia sovelluksia ilman, että sovelluksesta muodostuisi välttämättä vaikeasti hallittava spagettikoodirykelmä. (Heiskanen, 2013.)

SPA-sovellukset ovat yhden HTML-sivun varaan perustuvia, liiketoimintalogiikaltaan selainpäässä toimivia sovelluksia. Merkittävin seikka, joka erottaa ne perinteisistä web-sovelluksista on sivulatausten määrän vähentyminen ja mahdollisuus suorittaa sovelluksen toimintoja lataamatta sivua kokonaan uudelleen. SPA-sovelluksen tarvitsemaa asynkronista yhteyttä hallitaan Ajax-sovellusliittymällä, joka on yhdistelmä erilaisia tekniikoita. Ohjelmointikieleltään SPA-sovellukset toteutetaan pääsääntöisesti JavaScriptillä. (Heiskanen, 2013.)

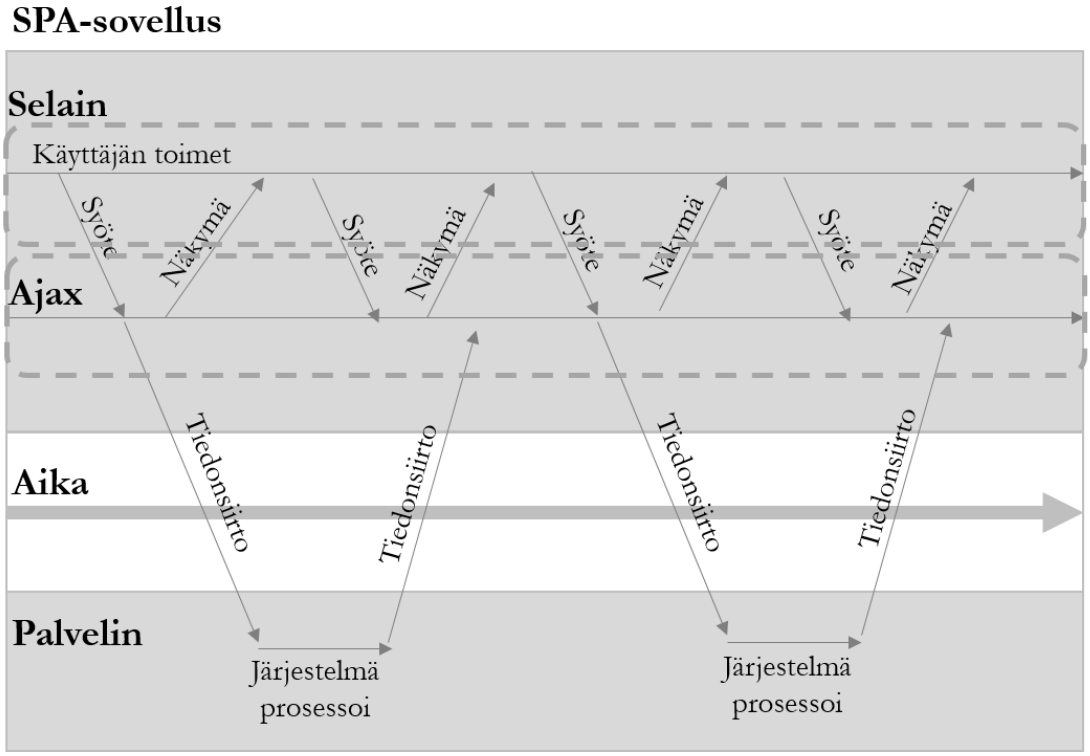
Sovellusarkkitehtuurista riippuen painopiste selaimen ja palvelimen tärkeydellä vaihtelee huomattavasti. Yhden sivun sovellukset noudattelevat periaatetta, jonka mukaan palvelinpäähän kohdistuva toiminta pyritään minimoimaan. Suurin osa SPA-sovelluksesta rakennetaan selainpään puolelle ja myös käyttöliittymän tila pidetään siellä. Tästä syystä esimerkiksi tallentuvien sessioiden määrä palvelinpään puolella vähenee. Palvelinta tarvitaan kuitenkin uuden tiedon hakemiseen, sekä tiedon tallentamiseen. SPA-sovelluksessa suora yhteys palvelimeen vähenee, mutta yhteydenpito suoritetaan sovelluksen taustalla. Palvelin ei ole yhtä suuressa merkityksessä, kuin perinteisessä sovelluksessa, jossa jokaiseen siirtymään tarvitaan suora kommunikointi palvelimen ja selaimen välillä. (Lehdonvirta & Korpela, 2013, 40.)

Perinteisessä sovellusarkkitehtuurissa käyttäjän toiminta katkeaa aina uuden interaktion alkaessa. Tällöin palvelimelta haetaan vanhan sivun tilalle uusi. Kuvassa 11 havainnollistetaan käyttökokemuksen katkeamista perinteisessä sovellusrakenteessa. (Asleson & Schutta, 2006, 37.)



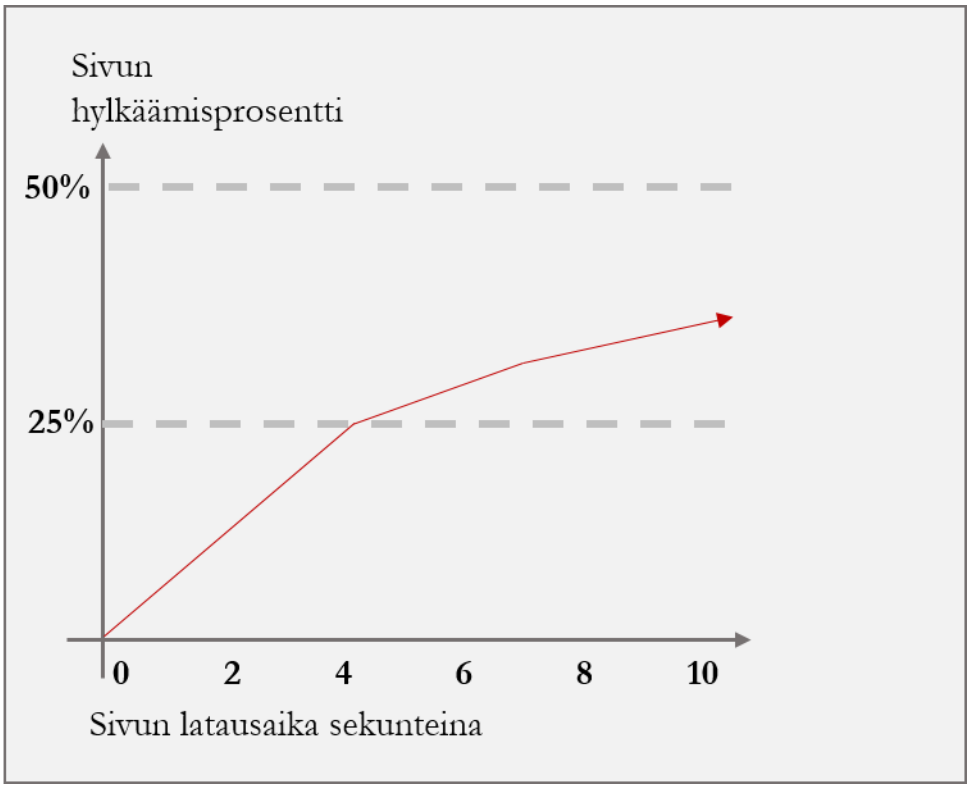
Kuva 11, perinteisen web-sovelluksen käytön kulku (Garret, 2005)

Palvelimelle asynkronisesti lähetetyt pyynnöt tekevät SPA-sovelluksen toiminnasta joustavamman. Selaimen kommunikoidessa joustavasti palvelimen kanssa, käyttäjäkokemus parantuu, ja käyttäjä voi toimia sovelluksessa ilman keskeytystä (kuva 12). Turhien sivulatausten välttäminen parantaa myös sovelluksen vasteaikoja, jolloin käyttäjälle jää tunne sovelluksen reaaliaikaisesta reagoinnista interaktioihin. (Asleson & Schutta, 2006, 37.)



Kuva 12, SPA-sovelluksen käytön kulku (Garret, 2005)

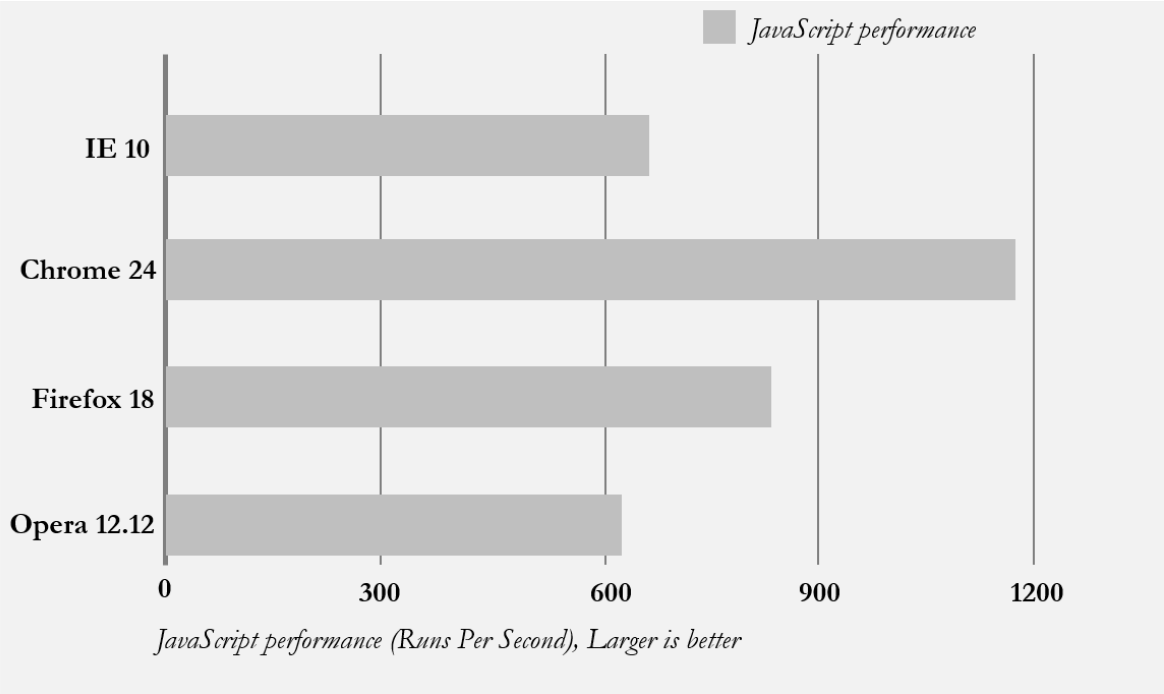
Eaton (2012) kuvaa artikkelissaan yhdysvaltalaisten tottumuksia sivun lataamisajoista. Amazonin kerrotaan laskeneen, että sivulatauksen hidastuminen vain sekunnilla tuottaisi 1,6 miljoonan dollarin myyntitappion. Myös Google on laskenut omat arvionsa sivulatauksen hidastumiseen. Jo neljän sekunnin kymmenyksen hidastuminen merkitsisi 8 miljoonan haun vähentymistä. Artikkelissa kerrotaan myös, kuinka monessa sekunnissa käyttäjän kiinnostus sivustoa kohtaan vähentyy, sekä näytetään kuinka monen sekunnin kohdalla käyttäjä luopuu sivun käyttämisestä (kuva 13). (Eaton, 2012.)



Kuva 13, sivulatauksen kesto sekunteina ja sen vaikutus käyttäjän toimintaan (Eaton, 2012)

Koska SPA-sovelluksen toiminta perustuu JavaScriptin varaan, on tärkeää että selain, jossa sovellusta suoritetaan, pystyy toistamaan sitä tarpeeksi nopeasti. Gordon (2013) esittelee artikkelissaan Peacekeeper-sivuston kautta tehtyä JavaScriptin suoritusnopeuteen perustuvaa testiä eri selainten välillä. Testissä on suoritettu suosittuja web-sovelluksia ja sivustoja, kuten Youtube, Facebook ja Gmail ja laskettu keskiarvoinen nopeus-

tulos benchmark-kaaviossa. Vaikka testissä ei ole testattu kaikkia selaimia, on nähtävissä että selainten välillä on havaittavissa selkeitä eroavaisuuksia. Kuvassa 14 esitetään artikkelissa mainitut tulokset. (Gordon, 2013.)



Kuva 14, JavaScript-nopeustesti (Gordon, 2013)

SPA-sovellusarkkitehtuurin ominaisuudet voivat osoittautua ongelmallisiksi joissakin tilanteissa. On huomioitava, että yhden sivun sovelluksessa kaikki perinteisistä web-sovelluksista sisäistetyt käyttötottumukset eivät toteudu samalla tavalla. XHR-tekniikkaa käytettäessä sivusto ei välttämättä tallennu selaimeen. Tästä saattaa muodostua ongelma käyttäjän pyrkiessä edelliseen näkymätilaan takaisin-painikkeen avulla. Myös POST-metodin aiheuttama URL-osoitteen muuttumattomuus saattaa luoda ongelmatilanteita, jos sivusta halutaan luoda kirjanmerkki tai sen osoite halutaan jakaa eteenpäin. Kehittäjän kannalta SPA-sovellusarkkitehtuuri avaa mahdollisuuden tehdä laajoja JavaScript-toiminnallisuuksia. Haasteena on kuitenkin pitää skriptimäärä hallittavassa mitassa niin, ettei siitä muodostu liian laaja, eikä se hidasta sovelluksen toimintaa. (Asleson & Schutta, 2006, 18.)

On myös huomioitava että erittäin pienellä osalla webin käyttäjistä on käytössään erittäin vanhoja selainversioita, jotka eivät tue SPA-sovelluksen teknologioita. Yhden sivun sovellukset ovat verrattain uusi tulokas web-sovellusten maailmassa ja tästä syytä niille

ei ole ehtinyt vielä vakiintua täysin yhteen toimivia arkkitehtuurimalleja ja sovelluskehyksiä. Mallien odotetaan kuitenkin vakiintuvan tulevaisuudessa, vaikka tällä hetkellä sovelluskehyksistä on lähes liikaa tarjontaa. (Lyytinen, 2013.)

5.1 Jäppinen J. Haastattelu 18.3.2014

Mitkä ovat mielestäsi suurimmat eroavaisuudet SPA-sovellusten, sekä perinteisten, useaa HTML-sivun varaan rakennettujen sovellusten välillä, miksi?

Perinteiset web-sovellukset tarkoittavat sitä, että käyttäjät lataavat jatkuvasti kokonaisia dokumentteja. On hassua vertailla suoraa eroa näiden kahden välillä. Vertauksena voisi käyttää sitä, että katsoo televisiosta still-kuvia ja videoita. Eroa on yritetty kuroa kiinni perinteisissä sovelluksissa lisäämällä niihin erilaisia JavaScript- toiminnallisuuksia, jotka tekevät sivulatauksesta suuremman asian. Se on periaatteessa liukuva gradientti perinteisestä sovelluksesta SPA-sovellukseen.

Ääripäissä käyttäjän kannalta tärkeimpänä erona on sivulatausten aiheuttama keskeytys käyttökokemukseen. Jos sivu sisältää animaatiota, se tavallaan jokisena sivulatauksen kohdalla keskeytyy. Kaiken välillä on niin sanottu seinä ja kaikki täytyy aloittaa alusta sivulatauksen tapahtuessa. On vaikea sanoa mitään tiettyä syytä, sillä perinteisiä sovelluksia ja SPA-sovelluksia on kaikenlaisia.

Mitkä perusteet vaikuttavat siihen, kumman arkkitehtuuriratkaisun valitset lähtiessäsi kehittämään sovellusta?

Käyttökokemuksesta täytyy miettiä sitä, kuinka tärkeässä roolissa sitä pidetään. Kun sovellusta lähdetään kehittämään, onko siitä tulossa sen tyyppinen, että se hyötyy SPA-arkkitehtuurista. Onko se esimerkiksi task-pohjainen sovellus, vai lukusovellus, tällöin et välttämättä käyttäjänä hyödy siitä, että mahdollistetaan niin sanottu jatkuva flow, eli sovelluksen toiminnan keskeytymättömyys. SPA-sovelluksen pitää myös aina tukea normaalin web-sovelluksen sujuvuutta.

Hyötyjen täytyy olla merkittäviä, jotta kannattaa lähteä kehittämään SPA-sovellusta. Käyttökokemuksen rooli, sisällön tyyppi ja se, mille alustalle se tehdään (mobiili, web) vaikuttavat valintaan. Pitää miettiä, onko tarkoitus kohdentaa sovellus sellaisille käyttä-

jille ja alustoille, jotka pystyvät pyörittämään lähtökohtaisesti raskaampaa SPA-sovellusta. Myös kehitystiimin taidot määräävät, kannattaako lähteä tekemään erikoisempaa ratkaisua.

Onko SPA-sovelluksen tekeminen haasteellisempaa kuin perinteisen?

SPA-sovelluksen tekeminen ei periaatteessa ole haasteellisempaa. Se riippuu paljon omasta kokemuksesta. Jos on tottunut tekemään perinteisiä sovelluksia, voi tekeminen olla vaikeampaa. Jos taas on Desktop- tai mobiilikehittäjän tausta, on SPA-sovelluskehitys paljon lähempänä.

Mitä asioita otat huomioon lähtiessäsi kehittämään sovellusta SPA-, tai perinteisen arkkitehtuurin mukaan?

Teen itse sovelluksia niin, että katson mikä on helpoin tapa lähteä liikkeelle sovelluksen kanssa. Valitsen arkkitehtuurin, minkä pystyn ennen pitkään järkevällä vaivalla kehittämään lopulliseen muotoon. Se, lähdenkö tekemään SPA-sovellusta, on eri asia, kuin se, että päädynkö heti tekemään sen. Saatan lähteä perinteisellä Frameworkilla, joka toimittaa sivut ja näkymät normaaleina web-näkyminä käyttäjälle. Jossain vaiheessa rakennan sen päälle JavaScript -toiminnollisuuden, joka tarjoaa mahdollisuuden sivulatauksien tekemiseen JavaScriptin kautta. Eli silloin tehdään normaali web-sovellus, jonka päälle tuodaan SPA-toiminnallisuus. Näin se useimmiten menee, sillä samat materiaalit toimitetaan samalla tavalla käyttäjälle. Jos URL-osoite vaihtuu ja se osoittaa tiettyyn näkymään, täytyy sen osoittaa siihen aina, vaikka navigoit JavaScriptillä tai avaat sivun uudelleen. Toimitettavan näkymän täytyy silloin pääsääntöisesti olla sama.

Ajattelen niin, että kasaan SPA-toiminnallisuuden perustan päälle, jos siihen on tarvetta. Se yleensä tapahtuu silloin, kun JavaScript-toiminnallisuutta on sovelluksessa muutenkin paljon. Silloin on helpoin ottaa sivulatauslogiikka ja muu mukaan, koska joutuu muutenkin työskentelemään paljon JavaScriptin kanssa. Jos on paljon esimerkiksi pop-up-ikkunoita, on helpompi ladata kaikki JavaScriptillä, kuin ladata isot näkymät web-latauksina ja pienet eri tavalla. On helpompi tehdä kaikki samalla tavalla.

Oletko kohdannut tilanteita, joissa arkkitehtuurin valinta on ollut vaikea, miksi? Onko ollut tilanteita, joissa on kesken projektin päädytty vaihtamaan toiseen ratkaisuun?

Vaihtaminen ei välttämättä ole hyvä termi, sillä perinteinen sovellus ja SPA-sovellus eivät sulje toisiaan pois. Hyvin tehty SPA-sovellus tukee myös täysin normaalia arkkitehtuuria.

Ennen, kun en ollut vielä toiminut niin paljon sovellusten parissa, tuli muutoksia ja uudelleenkirjoitusta sovellusten välillä enemmän. Nyt enemmän opiskelleena, ei enää ajattele näitä kahta sovellusta erillisinä. Olemme esimerkiksi vaihtaneet Rails- ohjelmiston kokonaan toisenlaiseen arkkitehtuuriin, jotta saimme SPA-toiminnallisuuden sen päälle. Ei niinkään sen takia, etteikö sitä olisi voinut tehdä Railsin päälle. Meillä oli jo kaikki Railsin päälle tehdyt kustomoinnit, joilla yritimme saada samaa käyttökokemusta, muttemme saaneet. Siitä tuli iso häkkyrä, jota oli mahdoton viedä loppuun. Aloitimme alusta ja otimme SPAn paremmin huomioon.

Millaisia hyötyjä näet SPA-sovellusarkkitehtuurissa, entä perinteisessä?

Perinteisellä sovellusarkkitehtuurilla on helppo lähteä liikkeelle. Siinä on vähemmän huolehdittavaa kehittäjälle. Siihen on myös enemmän valmiita, hyvin toimivia frameworkeja. Se on tavallaan teknisesti paljon turvallisempi vaihtoehto.

SPA-sovellus näkökulmastani on sitä, mitä kaikkien sovellusten pitäisi olla. Se on paljon lähempänä sitä, miten normaalisti ohjelmistot ovat toimineet. Idea on se, että ohjelma on kerran käynnistetty ja se pysyy käynnissä, eikä sitä tarvitse ladata koko ajan uudelleen näkymän vaihtuessa. Siinä ei myöskään ole mielivaltaista erottelua näkymien välillä. Ei niin, että yksi sivulataus olisi yksi näkymä, sillä näkymä voi olla esimerkiksi kokonainen sivu, pop-up-ikkuna, sivupalkki tai mikä tahansa osio mistä tahansa osasta dokumenttia. Se on paljon joustavampi ja siinä on paljon helpompi tehdä monipuolisia näkymäratkaisuja. Mukavana puolena on myös se, mitä enemmän saa siirrettyä rendauslogiikkaa JavaScriptiin, niin pystyy menemään vähitellen siihen, että kirjoittaa koko ohjelmansa JavaScriptillä, eli yhdellä kielellä. Kun taas palvelinpään frameworkia käytettäessä joudut silti kirjoittamaan sen päälle jonkun verran JavaScriptia saadaksesi siitä nykystandardien mukaisen. Et voi kuitenkaan tehdä kaikkea, mutta jos pystyt tekemään

kaiken yhdellä teknologialla, se helpottaa huomattavasti kehitystyötä. Reaaliaikaisuutta vaativat sovellukset (monitorointisoftat) lähtökohtaisesti vaativat SPA-arkkitehtuuria.

Millaisia haittoja näet SPA-sovellusarkkitehtuurissa, entä perinteisessä?

Jos vertaa siihen, tehdäänkö sivulataukset JavaScriptin puolella, vai palvelinpuolella, ei palvelinpuoli ole tavallaan reaaliaikainen. Sitä ei voi suorasanaisesti päivittää. Monesti joutuu duplikoimaan logiikkaa: jos olet tehnyt templaterendauksen palvelinpäässä (haluat päivittää esimerkiksi sovellusta ajon aikana), joudut kuitenkin tekemään siihen JavaScriptin, eli joudut monistamaan logiikan.

SPA-sovellukset vaativat enemmän tehoa ja ne pitää koodata hyvin, ettei tapahdu esimerkiksi muistivuotoja. Niissä tarvitsee enemmän oikeaa ohjelmointia, kuin web-templatointia. SPA-sovellukset vaativat myös modernimpia selaimia.

Mistä sovelluskehyksistä (SPA-arkkitehtuuri) sinulla on kokemusta ja millä perustein päädyit valitsemiisi ratkaisuihin? Onko tilanteita, joissa et ole käyttänyt mitään sovelluskehystä?

Olen toiminut melkein aina ilman sovelluskehystä. Ei ole oikeastaan ikinä ollut kokonaisratkaisua, joka olisi hoitanut kaiken. Olen käyttänyt erilaisia JavaScript-kirjastoja ja muita osapalasia, joista on voinut koostaa sellaisen SPA-sovelluksen, minkä haluaa. Olen tehnyt myös itse kirjastoja, jotka hoitavat näkymienhallintaa, rendausta ja latailua. En ole törmännyt ikinä yhteen hyvään sovelluskehykseen, mikä hoitaisi kaiken SPA-puolella. Toisaalta en ole törmännyt sellaiseen backend-puolellakaan. Kaikissa sovelluskehyksissä on hyvät ja huonot puolensa, joita pitää täydentää ja muokata, jotta niistä saa sellaisen minkä haluaa.

Mitä mieltä olet SPA-sovellusten käytettävyydestä (käyttäjän näkökulma) verrattuna perinteisiin web-sovelluksiin (esimerkiksi nopeus, sujuvuus)?

Käyttäjän ja asiakkaiden näkökulmasta SPA-sovelluksia ja perinteisiä sovelluksia vastaan aseteltaessa on yllättävän vähän eroa. Käyttäjät ovat niin tottuneita normaaleihin web-sovelluksiin, joissa lataillaan sivuja uudelleen. Se ei tunnu haittaavan ja toisaalta se, että niitä nopeutetaan JavaScript-toiminnoilla, ei tuo sinänsä niin suurta hyötyä käyttä-

jälle. JavaScript mahdollistaa sen, että voi käyttää paljon animaatioita ja tehdä nopeampia sivulatauksia. Uudet, paremmat teknologiat aiheuttavan yleensä sen, että kehittäjät innostuvat liikaa ja SPA-sovelluksista tulee monimutkaisia häkkyröitä, jolloin käytettävyys huononee. Perinteinen web-sovellus on käyttäjälle tuttu ja se toimii odotetulla tavalla.

5.2 Vesalainen, A. Haastattelu 23.4.2014

Mitkä ovat mielestäsi suurimmat eroavaisuudet SPA-sovellusten, sekä perinteisten, usean HTML-sivun varaan rakennettujen sovellusten välillä, miksi?
Käyttöliittymän layoutin rakentaminen ja sovelluksen tilan ylläpitäminen tapahtuu perinteisellä tavalla toteutettuna palvelimen päässä ja SPA:ssa selaimessa.

Mitkä perusteet vaikuttavat siihen, kumman arkkitehtuuriratkaisun valitset lähtiessäsi kehittämään sovellusta?
SPA on de facto -tapa toteuttaa moderneja selainpohjaisia sovelluksia. Perinteistä tapaa kannattaa käyttää enää joissakin rajatapauksissa.

Onko SPA-sovelluksen tekeminen haasteellisempaa kuin perinteisen?
Ei kategorisesti.

Mitä asioita otat huomioon lähtiessäsi kehittämään sovellusta SPA-, tai perinteisen arkkitehtuurin mukaan?
Suurin toteuttamiseen vaikuttava tekijä molemmissa tapauksissa on teknologiavalinnat. Yrittäisin pitää molemmissa tapauksissa toteutuksen uskollisena valitulle paradigmalle.

Oletko kohdannut tilanteita, joissa valinta on ollut vaikea, miksi? Onko ollut tilanteita, joissa on kesken projektin päädytty vaihtamaan toiseen ratkaisuun?
Olen ollut projektissa, jossa alun perin oli valittu teknologiaksi GWT, mutta jossa päädyttiin loppujen lopuksi käyttämään Ruby on Railsia. Tällöin ei ollut vielä varteenotettavaa sovelluskehystä SPA:n tekoon. Lisäksi sovelluksen piti tukea vanhoja selaimia.

Millaisia hyötyjä näet SPA-sovellusarkkitehtuurissa, entä perinteisessä?

SPA-sovelluksessa etuna on esimerkiksi se, että käyttöliittymälogiikka on yhdessä paikassa. Perinteisillä menetelmillä usein päädytään hybridimalliin, jossa näkymän html:ää luodaan ensin palvelimella sovelluskehyksen kielellä ja sitten uudestaan selaimessa jonkun komponentin tilan päivittämisen yhteydessä JavaScriptillä.

Millaisia haittoja näet SPA-sovellusarkkitehtuurissa, entä perinteisessä?
Uudet sovelluskehykset tarjoavat melko abstraktit raamit sovelluksille, joten tekijän harteille jää tärkeitäkin päätöksiä arkkitehtuurin suhteen. Perinteinen tapa ei ole kovin joustava, kun on kyseessä on ns. rikas internet-sovellus.

Mistä sovelluskehyksistä (SPA-arkkitehtuuri) sinulla on kokemusta ja millä perustein päädyit valitsemiisi ratkaisuihin? Onko tilanteita, joissa et ole käyttänyt mitään sovelluskehystä?
Tunnetuista sovelluskehyksistä kokemusta on lähinnä AngularJS:ää ja pienissä määrin Backbonesta. Angular tarjoaa Backbonea hieman valmiimman raamin. Sovelluskehyksen käyttäminen on mielestäni useimmiten välttämätön paha. Koodaaminen on parhaimmillaan hauskaa puuhaa; sovelluskehysten kanssa tappeleminen ei koskaan.

Mitä mieltä olet SPA-sovellusten käytettävyydestä (käyttäjän näkökulma) verrattuna perinteisiin web-sovelluksiin (esimerkiksi nopeus, sujuvuus)?
SPA-sovelluksella pitäisi kaiken järjen mukaan olla pienempi vasteaika. Jossain on vikaa, mikäli näin ei ole.

6 Johtopäätökset

Tutkimuksen tavoitteena oli kartoittaa SPA-arkkitehtuuriin perustuvan web-sovelluksen yleisimmät teknologiaratkaisut ja selvittää, missä tilanteissa yhden sivun sovelluksen käyttö on suositeltavampaa verrattuna perinteiseen sovellusarkkitehtuuriin. Teoriataustan perusteella saatiin selvyys SPA-sovelluksen perusrakenteesta sen eroista perinteiseen sovellukseen. Hyötyjen ja haasteiden tarkasteluun käytettiin niin teoriamateriaalia, kuin haastatteluja.

Tulosten perusteella huomataan, että kokonaisuudessaan SPA-sovellukset ovat varsin uusi, mutta kasvava trendi web-sovellusten keskuudessa. Niiden kasvua ovat siivittäneet kehittyneet teknologiat ja arkkitehtuurimallit, joiden avulla sovelluksen rakentamisesta saa jäsennellymmän. SPA-sovellusten määrä kasvanee tulevien vuosien aikana huomattavasti.

6.1 Hyödyt ja haasteet

Yhden sivun sovelluksen suurimmaksi hyödyksi voi laskea sen toimintamahdollisuuden ilman jatkuvia sivulatauksia. Se, että käyttäjän toimet eivät aiheuta kerta toisensa jälkeen tapahtuvia katkoksia sovelluksen käytössä, parantavat käyttökokemusta huomattavasti. Myös saman tiedon lataamiselta palvelimesta useaan kertaan uudelleen vältytään käyttämällä SPA-arkkitehtuuria. Tarvittava tieto on ladattu kertaalleen ja kommunikointi palvelimen kanssa tapahtuu vain tarvittaessa, tai sovelluksen taustalla. Tämä tarkoittaa lyhentyneitä vasteaikoja ja reaaliaikaiselta tuntuvaa toimivuutta. Asynkronisen kommunikoinnin avulla SPA-sovelluksesta saadaan parhaimmillaan joustavasti ja katkotta toimiva sovellus.

SPA-sovelluksen liiketoimintalogiikan sijoittuessa selainpuoleen saadaan samalla käyttöliittymälogiikka pidettyä yhdessä paikassa toisin kuin perinteisissä verkkosovelluksissa, joissa käytetään yleensä niin sanottua hybridiratkaisua, jotta saavutetaan nykystandardien mukainen sovellus. Liiketoimintalogiikan painottuessa yhteen paikkaan sen hallinta

ja myös itse kehitystyö helpottuu. SPA-arkkitehtuuri avaa myös mahdollisuuksia erilaisiin näkymäratkaisuihin, kun tietyt rakenteeltaan pakolliset näkymärajoitelmat eivät ole käytössä. Yksi sivulataus ei vastaa aina välttämättä yhtä näkymää, vaan näkymänä voidaan käyttää mitä tahansa sovelluksen osaa, kuten sivupalkkia tai pop-up-ikkunaa.

Niin perinteisille, kuin SPA-arkkitehtuurin sovelluksille on saatavilla laaja määrä erilaisia sovelluskehyksiä. Varsinkin JavaScript-sovelluskehyksistä tuntuu olevan miltei ylitarjontaa. Oikean sovelluskehyksen valinta saattaakin muodostua haasteeksi, jos kehystä valitessa ei olla täysin varmoja siitä, mikä vaihtoehdoista sopii parhaiten omalle kohdalle. Perinteisille web-sovelluksille on tarjolla enemmän testattuja ja hyväksi todettuja sovelluskehyksiä. Kuitenkin kaikissa sovelluskehyksissä on omat haasteensa, eikä yhtä oikeaa vaihtoehtoa ole. On myös huomioitava, ettei sovelluskehyksen käyttö ole kaikissa tapauksissa välttämätöntä.

SPA-sovelluksen kehittämistä tunnutaan pidettävän hieman perinteistä sovellusta haasteellisempana, sillä sen luomiseen tarvitaan usein suuria määriä JavaScriptiä. Haasteellisuuteen vaikuttaa paljon kehittäjien lähtötaso ja aiempi kokemus vastaavanlaisesta sovelluskehityksestä. Toisin sanoen SPA-sovelluksen toteuttaminen poikkeaa perinteisen sovelluksen toteutuksesta ja saattaa muodostua haasteellisemmaksi silloin, jos taitotaso toteutukseen ei ole tarpeeksi korkea. Joissakin tapauksissa haasteeksi tai haitaksi voivat muodostua käyttäjien liian vanhat selainversiot, jotka eivät tue tarpeeksi SPA-sovelluksessa käytettyjä tekniikoita. SPA-sovellukset eivät myöskään toimi, jos JavaScript ei ole päällä.

6.2 Arkkitehtuurin valinta

Sovellusta kehitettäessä ja varsinkin sitä suunniteltaessa on mietittävä, mihin käyttötarpeeseen sovellus tullaan sovittamaan. Sovelluksen käyttötarve, kohderyhmä ja se, missä laitteissa ja selaimissa sen halutaan toimivan, ovat vaikuttavia tekijöitä siihen, lähdetäänkö sovelluksesta kehittämään selainpohjaisesti toimivaa SPA-sovellusta, vai perinteistä verkkosovellusta. Vaikka SPA-sovellus joustavine toimintoineen kuulostaa paremmalta ratkaisulta, voi sen toteutus joissakin tietyissä tapauksissa olla täysin turhaa.

SPA-sovellusarkkitehtuuri soveltuu varsinkin paljon liikkuvaa ja paikkaansa vaihtavaa sisältöä tarjoaviin web-sovelluksiin. Modernit ja reaaliaikaisuutta vaativat toiminnallisuudet tarvitsevat samalla myös ajan tasalla olevia selaimia, joten kohdeasiakaskunnan käyttäessä vanhempia selainmalleja on huomioitava, pystyykö selain suorittamaan tarvittavat ominaisuudet. Tällä hetkellä kuitenkin suurin osa selaimista tukee SPA-sovellusarkkitehtuurin vaatimia tekniikoita.

Jos sovellukseen ei ole tarkoitus lisätä kovinkaan paljon sovellusmaisia piirteitä, kuten interaktiota käyttäjän ja sovelluksen välillä, tai dynaamisia toiminnallisuuksia, on tällöin perinteinen web-sivu mahdollisesti parempi ratkaisu. Useimmin tällä hetkellä käytetty toimintatapa tuntuu kuitenkin olevan se, että perinteisen mallin päälle rakennetaan joko SPA-toiminnallisuus tai osia siitä. Suurin osa web-sivuista ja sovelluksista on rakennettu näin, sillä useissa sovelluksissa vaaditaan tietty määrä dynaamista vuorovaikutusta yltääkseen käyttäjän vaatimiin standardeihin.

Sovelluksen arkkitehtuurin valinta ei aina välttämättä välity käyttäjälle asti ja joissakin tapauksissa käyttäjä voi kokea perinteisestä sovelluksesta toiminnallisuudeltaan poikkeavan SPA-sovelluksen hankalampana käyttää, sillä tottumukset ovat vielä perinteisessä sovellusrakenteessa. Yhtenä esimerkkinä takaisi- painike, joka ei toimi SPA-sovelluksissa aina totuttuun vanhaan tapaan, eli johda edelliseen näkymään..

6.3 Opinnäytetyöprojekti ja oma oppiminen

Opinnäytetyöprojekti käynnistyi ja eteni nopealla aikataululla. Aiemmin valitun opinnäytetyöprojektin aiheen toteutus ei onnistunut, joten nyt valittu aihe valittiin nopeasti, jotta alkuperäisessä valmistumissuunnitelmassa pysyttäisiin. Aiheen valintaan vaikutti eniten sen ajankohtaisuus, SPA-sovellukset ovat olleet viimeisen vuoden aikana esillä, ja niistä näkyi mainintoja muissa opinnäytetöissä. Aiheen valinnan jälkeen koottiin aikataulu projektin toteutukselle.

Tiukka aikataulu toimi jo itsessään haasteena, mutta toimi myös hyödyllisenä rajaajana, sillä aikaa ylimääräisten asioiden käsittelemiseen ei jäänyt. Tämän ansiosta työ on pysy-

nyt koko projektin ajan suhteellisen hyvin alussa määritellyissä tavoitteissa. Itse työn toteutukseen jäi kokonaisuudessaan aikaa noin kuukausi. Vaikka aikataulu oli kiireinen, työnjako onnistui suunnitellulla tavalla, eikä minkäänlaista paniikinomaista kiirettä päässyt syntymään.

Aikataulun asettamien rajoitteiden puitteissa työhön kirjattava teoriatieto rajattiin jo heti alussa pieneksi ja työn tavoitteeksi asetettiin tästä syystä tavoite saada yleiskuva perinteisen- ja SPA-sovellusarkkitehtuurin eroista, sekä löytää hyötyjen ja haittojen avulla ratkaisu siihen, kumpaa arkkitehtuuriratkaisua on järkevä hyödyntää. Teoriatiedon ja tulosten perusteella esitettyihin tutkimuskysymyksiin löytyivät ennalta odotetut ratkaisut. Jotta projektista olisi saanut selville parhaan mahdollisen lopputuloksen, olisi ajan salliessa ollut järkevää rakentaa pienehköt, samantyyliset sovellukset vertailtavilla arkkitehtuuriratkaisuilla. Sovellusten avulla olisi pystynyt saamaan omakohtaista testituloksia niiden vasteajoista ja toimivuudesta eri selaimilla. Sovelluksia olisi voinut myös voinut testata pienellä käyttäjäjoukolla. Koska tämä ei ollut mahdollista projektin tiukan aikataulun vuoksi, olisi se suositeltavaa toteuttaa projektille jatkotoimenpiteenä.

Projektin aihealue ei ollut entuudestaan tuttu, joten projekti toimi samalla oppimiskokemuksena niin aiheeseen, kuin myös projektinhallintaan kiireisessä aikataulussa liittyen. Muita kuin aikatauluun liittyviä haasteita ei tullut vastaan ja myös projektin aihe säilytti mielenkiintonsa loppuun asti.

7 Yhteenveto

Opinnäytetyö suoritettiin opinnäytetyöprojektina, johon sisältyi työn lisäksi projektinhallintaa. Opinnäytetyön tavoitteena oli perehtyä SPA-sovelluksen rakenteeseen sekä sen eroavaisuuksiin verratessa perinteiseen sovellusarkkitehtuuriin. Työssä pyrittiin myös selvittämään kahden valitun arkkitehtuurin käyttöön ja kehitykseen liittyvät hyödyt ja haitat, sekä seikat, joiden perusteella käytettävä arkkitehtuuriratkaisu kannattaa valita. Työssä käytettiin lähdemateriaalina kirjallisuuslähteitä, kuten kirjoja ja nettiartikkeleja, sekä näiden lisäksi haastatteluja.

Tutkimuksen aikana ilmeni, että SPA-arkkitehtuurin ja perinteisen arkkitehtuurin välillä on havaittavissa selkeitä eroavaisuuksia niin niiden käytettävyyden, kuin teknologioidenkin osalta. SPA-sovelluksen toiminta perustuu toiminnaltaan pääasiallisesti selaimen puolelle ja sen toiminnallisuudet mahdollistavat joustavan ja reaaliaikaiselta tuntuvan sovelluksen käyttökokemuksen. Perinteinen sovellus taas toimii kokoaikaisessa vuorovaikutuksessa palvelimen kanssa, jolloin uuden sivun, tai informaation lataus aiheuttaa jokaisella kerralla sivulatauksen. SPA-sovellukset ovat perinteisiin sovelluksiin verrattuna uusia tulokkaita www-kentällä, mutta niiden määrän voidaan odottaa nousevan tulevien vuosien aikana.

Johtopäätöksissä esitellään tuloksien kautta saadut päätelmät, joista on löydettävissä vastaukset projektille asetettuihin tutkimuskysymyksiin. Johtopäätöksissä mainitaan selkeimmät eroavaisuudet perinteisen ja SPA-arkkitehtuurin välillä, sekä analysoidaan erojen lisäksi sovellusten hyötyjä ja haittoja. Näiden lisäksi pohditaan perusteita, jotka vaikuttavat arkkitehtuurivalinnan ratkaisussa. Jotta tutkimuksen tuloksista saataisiin konkreettisempaa ja käytännönläheisempää tietoa, olisi jatkotoimenpiteenä suositeltavaa rakentaa esimerkkisovellukset niin perinteisesti, kuin SPA-arkkitehtuuria noudattavista sovelluksista.

Lähteet

Asleson, R., Schutta, N. 2006. Ajax - Tehokas hallinta. Readme.fi. Helsinki.

Borodescu, C. 2014. Web Sites vs. Web Apps: What the experts think. Luettavissa: http://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts-think/. Luettu: 11.4.2014.

Clark, Jason A. Online. Artickle. Nov/Dec2006, Vol. 30 Issue 6, p31-34. 4p. AJAX(Asynchronous JavaScript and XML): This Isn't the Web I'm Used To. Luettavissa: www.helmet.fi, Hakusana: Ajax. Luettu: 27.3.2014.

Degreave.com. 2010. Simple Ajax Example. Luettavissa: http://www.degraeve.com/reference/simple-ajax-example.php). Luettu: 28.3.2014

Eaton, K. 15.3.2012. How one second could cost amazon $1.6 billion in sales. Luettavissa: http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. Luettu:15.4.2014.

Fielding, R., 2000. Architectural Styles and the Design of Network-based Software Architectures. Luettavissa: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Luettu: 27.3.2014.

Flanagan, D. 2011. JavaScript: The Definitive Guide, Sixth Edition. O'Reilly Media. Yhdysvallat.

Garret, J. 18.2.2005. Ajax: A New Approach to Web Applications. Luettavissa: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/. Luettu: 27.3.2014

Gordon, W. Lifehacker. 15.1.2013. Browser Speed Tests: Chrome 24, Firefox 18, Internet Explorer 10, and Opera 12.12. Luettavissa: http://lifehacker.com/5976082/browser-speed-tests-chrome-24-firefox-18-internet-explorer-10-and-opera-1212. Luettu: 23.4.2014.

Heiskanen, H. 31.5.2013. Yhden sivun web-sovellukset tulevat, oletko valmis? Luettavissa: http://gofore.com/ohjelmistokehitys/yhden-sivun-web-sovellukset-tulevat-oletko-valmis/. Luettu: 14.3.2014.

Jäppinen, J. 18.3.2014. Käyttöliittymäsuunnittelija. Eficode Oy. Haastattelu. Helsinki.

Korpela, J., Lehdonvirta, P. 2013. HTML5 Sovellusalustana. RPS-yhtiöt. Helsinki.

Korpela, J. 21.6.2009. Web-julkaisemisen opas, luku 2 Reseptit: Miten luon Web-sivujen kokonaisuuden, ja muuta vaativampaa: Luettavissa: https://www.cs.tut.fi/~jkorpela/webjulk/2.4.html. Luettu: 27.3.2014.

Lyytinen, T. 5.6.2013. Html5 web-sovellukset. Luettavissa: http://67.prosenttia.fi/2013/06/05/html5-web-sovellukset/. Luettu 27.3.2014.

Medianurkka. 2014. Web-Tekniikka. Luettavissa: http://ohjelmointi.medianurkka.com/?p=5. Luettu: 27.3.2014

Osmani, A. 27.7.2012. Journey Through The JavaScript MVC Jungle. Luettavissa: http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle-2/. Luettu: 27.3.2014.

Pupius, D. 6.6.2013. Rise of the SPA. Luettavissa: https://medium.com/tech-talk/fb44da86dc1f. Luettu: 27.3.2014.

Taft, Darryl K. eWeek. 2/10/2014, p9-9. 1p. Luettavissa: http://rphelsp.helmet.fi/ebsco-w-b/ehost/pdfviewer/pdfviewer?sid=d57705c9-cab2-4246-b0df-a42452baf58a%40sessionmgr111&vid=2&hid=103. Luettu: 27.3.2014.

Teare, D. Oracle. 2005. An Introduction To Ajax. Luettavissa: http://www.oracle.com/technetwork/articles/entarch/ajax-introduction-096831.html. Luettu: 27.3.2014.

Vesalainen, A. 23.4.2014. Lead GUI Designer. Eficode Oy. Haastattelu. Helsinki.

W3schools.com. HTTP Methods: GET vs. POST. Luettavissa: http://www.w3schools.com/tags/ref_framemethods.asp. Luettu: 11.4.2014.

Liitteet

Liite 1. Ajax XHR-pyyntö

(Degreave.com, 2010.)

```
<html>
<head>
<title>Simple Ajax Example</title>
<script language="Javascript">
function xmlhttpPost(strURL) {
   var xmlHttpReq = false;
   var self = this;
   // Mozilla/Safari
   if (window.XMLHttpRequest) {
      self.xmlHttpReq = new XMLHttpRequest();
   }
   // IE
   else if (window.ActiveXObject) {
      self.xmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
   }
   self.xmlHttpReq.open('POST', strURL, true);
   self.xmlHttpReq.setRequestHeader('Content-Type', 'applica-tion/x-www-form-urlencoded');
   self.xmlHttpReq.onreadystatechange = function() {
      if (self.xmlHttpReq.readyState == 4) {
         updatepage(self.xmlHttpReq.responseText);
      }
   }
   self.xmlHttpReq.send(getquerystring());
}

function getquerystring() {
   var form     = document.forms['f1'];
   var word = form.word.value;
   qstr = 'w=' + escape(word);  // NOTE: no '?' before querystring
   return qstr;
}

function updatepage(str){
   document.getElementById("result").innerHTML = str;
}
</script>
</head>
<body>
<form name="f1">
  <p>word: <input name="word" type="text">
  <input value="Go" type="button" on-click='JavaScript:xmlhttpPost("/cgi-bin/simple-ajax-exam-
ple.cgi")'></p>
  <div id="result"></div>
</form>
</body>
</html>
```