Tampereen yliopisto

Teemu Jalonen

YHDEN JA MONEN SIVUN SOVELLUKSET WEB-KEHITYKSESSÄ

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Huhtikuu 2019

TIIVISTELMÄ

Teemu Jalonen: Yhden ja monen sivun sovellukset web-kehityksessä
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Huhtikuu 2019

Web-sovelluksia on perinteisesti kehitetty monen sivun sovelluksina, mutta niiden rinnalle ovat viimeisen vuosikymmenen aikana nousseet yhden sivun sovellukset, jotka ovat saaneet suuren suosion. Tässä kandidaatintyössä selvitetään yhden ja monen sivun sovelluksien toimintaperiaatteita sekä mitä hyviä ja huonoja puolia kummallakin sovellustyypillä on. Hyviä ja huonoja puolia tarkastellaan kehittäjien sekä käyttäjien näkökulmista käyttäen hyväksi kirjallisuudesta löytyvää tietoa.

Tulokseksi saadaan, että sovellustyypit eroavat selvästi toisistaan ja kummallakin on ominaispiirteensä. Yhden sivun sovellukset tarjoavat loistavan käyttäjäkokemuksen ja ne sopivat hyvin pienempiin sovelluksiin. Monen sivun sovellukset puolestaan sopivat hyvin laajoihin sovelluksiin hyvän skaalautuvuutensa ansiosta, mutta ne tarjoavat huonomman käyttäjäkokemuksen. Molemmilla sovellustyypeillä on siis paikkansa web-kehityksessä.

Avainsanat: Yhden sivun sovellus, monen sivun sovellus, web-sovellus, web-kehitys

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

Ajax	engl. Asynchronous JavaScript and XML, monen teknologian joukko, jonka tarkoituksena on datan välittäminen sovelluksen taustalla
API	engl. Application programming interface, ohjelmointirajapinta
CSS	engl. Cascading Style Sheet, tyyliohjeiden laji
DOM	engl. Document Object Model, ohjelmointirajapinta, joka käsittelee verkkosivua puurakenteena
HTML	engl. HyperText Markup Language, avoimesti standardoitu kuvauskieli
HTTP	engl. HyperText Transfer Protocol, protokolla tiedonsiirtoon
JSON	engl. JavaScript Object Notation, avoimen standardin tiedostomuoto tiedonvälitykseen
MPA	engl. Multiple-page application, monen sivun sovellus
MVC	engl. Model-view-controller, malli-näkymä-ohjain
REST	engl. Representation State Transfer, arkkitehtuurimalli ohjelmointirajapinnoille
SEO	engl. Search Engine Optimization, hakukoneoptimointi
SPA	engl. Single-Page Application, yhden sivun sovellus
XHR	engl. XMLHttpRequest, ohjelmointirajapinta tiedonvälitykseen web-selaimen ja web-palvelimen välillä
XML	engl. Extensible Markup Language, laajennettava merkintäkieli
XSS	engl. Cross-site scripting, tietoturva-aukko web-sovelluksissa

1. JOHDANTO

Yhä useammat ihmiset käyttävät internetiä tänä päivänä. Viimeisen kymmenen vuoden aikana internetin käyttäjien määrä on kasvanut 2,3 miljardilla [18]. Ennen paikan päällä hoidettavia palveluita lopetetaan ja siirretään internetiin, jossa ne ovat useampien saatavilla nopeasti ja kätevästi. Huono käyttäjäkokemus web-sovelluksessa saattaa karkottaa käyttäjät pois jonkun kilpailevan palveluntarjoajan web-sovellukseen, sillä vaihto voi helpoimmassa tapauksessa olla pelkästään toiseen osoitteeseen siirtyminen. Täten kehittäjät etsivätkin yhä parempia tapoja kehittää web-sovelluksia käyttäjäkokemuksen maksimoimiseksi.

Perinteisesti web-sovelluksia on kehitetty monen sivun sovelluksina (engl. MPA, Multiple-page application). Niiden evoluutiona alettiin kehittämään yhden sivun sovelluksia (engl. SPA, Single-page application), jotka pyrkivät tarjoamaan erityisesti käyttäjille paremman kokemuksen.

Tässä kandidaatintyössä selvitetään, miten yhden sivun sovellukset ja monen sivun sovellukset toimivat web-kehityksen menetelminä. Lisäksi selvitetään molempien menetelmien hyvät ja huonot puolet sekä missä tilanteissa kannattaa kehittää yhden sivun sovellus monen sivun sovelluksen sijaan. Selvitys tehdään vertailemalla kirjallisuudesta löytyvää tietoa.

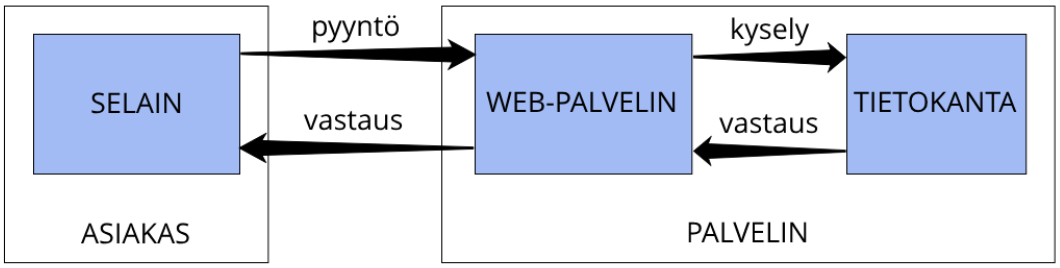
Työn luvussa 2 perehdytään yhden ja monen sivun sovelluksien kannalta keskeisiin webteknologioihin. Luvussa 3 esitellään yhden ja monen sivun sovellukset sekä niiden toimintaperiaatteet. Luvussa 4 selvitetään näiden web-kehityksen menetelmien hyvät ja huonot puolet kehittämisen ja käyttämisen näkökulmista. Luvussa 5 esitellään kootusti vertailun tulokset. Lopuksi tehdään yhteenveto työstä.

2. WEB-TEKNOLOGIAT

Web-sovellusten kehityksessä käytetään lukuisia eri teknologioita apuna. Seuraavissa alaluvuissa esitellään asiakas—palvelin-malli, Ajax sekä MVC-arkkitehtuuri, jotka ovat yhden ja monen sivun sovelluksien kannalta keskeisiä.

2.1 Asiakas—palvelin-malli

Eri web-sovellusten kehitysmenetelmien ymmärtämiseksi on tärkeää tietää, mikä asiakas—palvelin-malli (engl. client-server model) on. Web-sovellusten tapauksessa asiakkaana on selain ja palvelimena sekä web-palvelin että tietokanta. Asiakas—palvelin-malli kuvaa, miten nämä keskustelevat keskenään, ja mallin perusrakenne on esitetty kuvassa 1.



Kuva 1. *Asiakas—palvelin-malli web-sovelluksessa. [9]*

Keskustelu alkaa siitä, kun asiakas lähettää pyynnön web-palvelimelle. Web-palvelin käsittelee pyynnön, ja sen tyypin perusteella mahdollisesti poistaa, muuttaa tai hakee dataa palvelimessa sijaitsevasta tietokannasta kyselyllä. Lopulta web-palvelin lähettää vastauksen takaisin asiakkaalle. Vastaus voi olla esimerkiksi tietokannasta haettu data, tieto operaation onnistumisesta tai sen mahdollinen epäonnistuminen.

2.2 Ajax

Ajax (Asynchronous JavaScript and XML) on keskeinen käsite modernissa web-kehityksessä. Ajax ei ole itsessään teknologia, vaan se tarkoittaa monen eri teknologian käyttöä

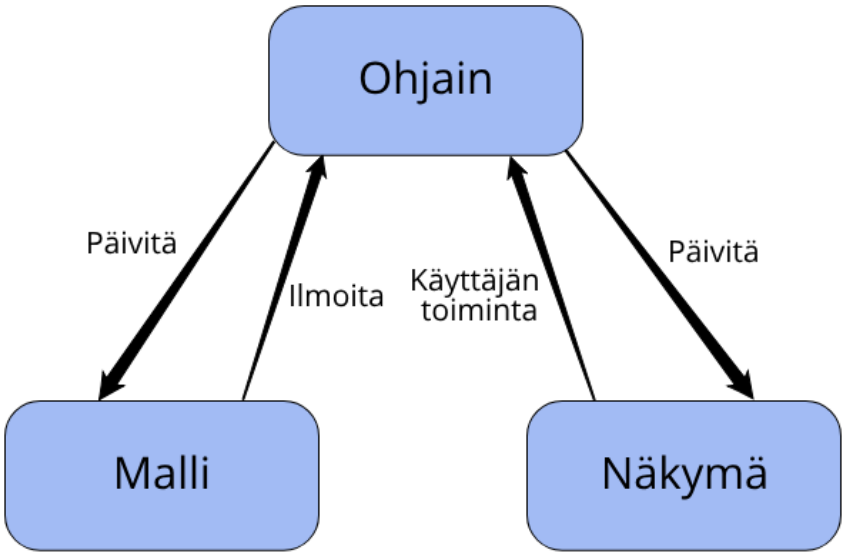
yhdessä. Näihin teknologioihin kuuluvat muun muassa HTML (HyperText Markup Language), CSS (Cascading Style Sheet), JavaScript, DOM (Document Object Model), XML (Extensible Markup Language) ja tärkeimpänä XHR-olio (XMLHttpRequest). [12]

Ajaxilla pystyy tekemään nopeita ja inkrementaalisia päivityksiä verkkosivun käyttöliittymään lataamatta koko sivua uudestaan. Tämä lisää verkkosivun käytettävyyttä nopeuttamalla verkkosivua ja tekemällä sen herkemmäksi käyttäjän toiminnoille. [12]

Ajaxissa käytetään tiedonvälityksessä usein XML:n sijasta JSON:ia (JavaScript Object Notation), vaikka Ajaxin nimestä voisi päätellä, että tiedonvälityksessä käytetään XML:ää. Tämä johtuu muun muassa siitä, että JSON on XML:ää kevyempi ja se on osa JavaScriptiä. [12]

2.3 MVC-arkkitehtuuri

MVC-arkkitehtuuri on malli, joka kuvaa tavan tehdä uudelleenkäytettävää, ylläpidettävää ja helposti laajennettavaa koodia. Lisäksi se erottaa käyttöliittymän bisneslogiikasta, mikä mahdollistaa eri komponenttien samanaikaisen kehittämisen. [14]



Kuva 2. *MVC-mallin rakenne. [14]*

Kuvasta 2 nähdään, että MVC-arkkitehtuurissa on kolme komponenttia: malli (Model), näkymä (View) ja ohjain (Controller). Sovelluksen data on tallennettu malliin, ja kun malli muuttuu se ilmoittaa ohjaimelle muuttumisestaan. Näkymä puolestaan määrittää sovelluksen käyttöliittymän ulkoasun, miten mallien data esitetään käyttöliittymässä sekä miten käyttäjä vuorovaikuttaa sen kanssa. Ohjain sijaitsee mallin ja näkymän välissä ja se

hoitaa niiden välisen kanssakäymisen. Jos malli muuttuu, ohjain päivittää näkymän vastaamaan muutosta. Lisäksi käyttäjän manipuloidessa näkymää ohjain päivittää mallin tietoja tapahtuman määrittelemällä tavalla. [14]

3. WEB-KEHITYKSEN MENETELMÄT

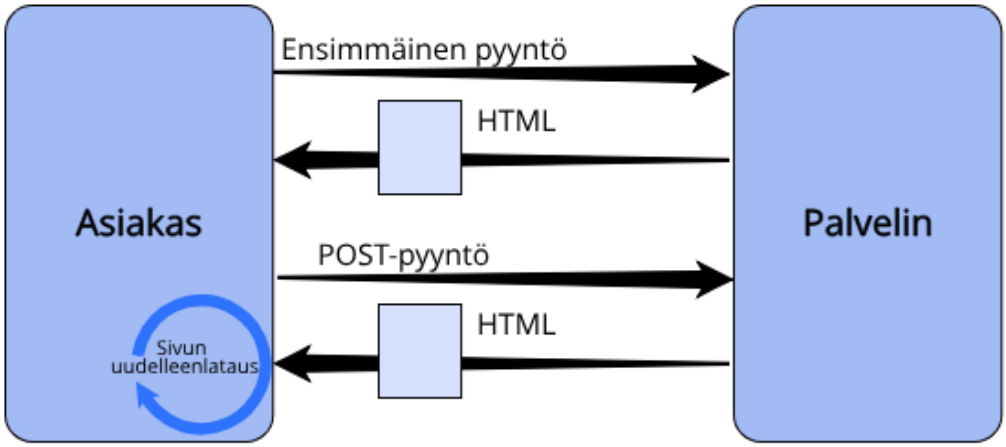
Web-sovelluksia voi kehittää usealla eri tavalla. Perinteinen tapa on kehittää niin sanottuja monen sivun sovelluksia. Ne koostuvat nimensä mukaisesti useasta staattisesta HTML-sivusta, joiden välillä siirtyminen vaatii sivun uudelleenlatauksen.

Toinen modernimpi tapa on yhden sivun sovellukset, jotka koostuvat nimensä mukaisesti vain yhdestä HTML-sivusta. Yhden sivun sovelluksissa kaikki tarvittavat resurssit haetaan kerralla palvelimelta, eikä sivua uudelleenladata sivua käytettäessä.

3.1 Monen sivun sovellukset

Monen sivun sovellus koostuu useasta staattisesta sivusta, joissa voi olla esimerkiksi tekstiä ja kuvia. Niiden lisäksi sivulla on linkkejä, joita painamalla käyttäjä voi navigoida sivuston muihin osiin. Toiseen osaan siirtyminen aiheuttaa sivun uudelleenlatauksen, jolloin kaikki sisältö ladataan uudestaan palvelimelta, vaikka sisältö olisikin sivujen välillä samaa. [9]

Monen sivun sovelluksien pääpaino on palvelimella, jossa suurin osa sovelluksen logiikasta sijaitsee. Asiakas ainoastaan renderöi palvelimelta saamansa sivut. JavaScriptiä voidaan kuitenkin käyttää pienten käyttöliittymämuutosten tai animaatioiden tekemiseen asiakaspuolella. [1]



Kuva 3. *Monen sivun sovelluksen elinkaari. [10]*

Kuten kuvasta 3 voi nähdä, asiakas pyytää ensin sivua palvelimelta (initial request), mihin palvelin vastaa palauttamalla HTML-dokumentin. Sivua käytettäessä käyttäjä voi esimerkiksi täyttää ja lähettää lomakkeen palvelimelle (POST-pyyntö), mihin palvelin vastaa jälleen palauttamalla HTML-dokumentin, ja koko sivu ladataan uudelleen. [1] Uudelleenlatauksien määrää voidaan kuitenkin vähentää käyttämällä Ajaxia, jonka avulla dataa voidaan lähettää ja vastaanottaa asynkronisesti uudelleenlataamatta sivua [9].

3.2 Yhden sivun sovellukset

Yhden sivun sovelluksien toimintaperiaate eroaa selvästi monen sivun sovelluksien toimintaperiaatteesta. Riippuvuus palvelimeen on pienempi ja palvelimelta pyritään lähettämään vain uutta tietoa asiakkaalle. Tässä alaluvussa käsitellään tarkemmin yhden sivun sovelluksien toimintaperiaatetta sekä hakukoneoptimointia (engl. SEO, Search Engine Optimization).

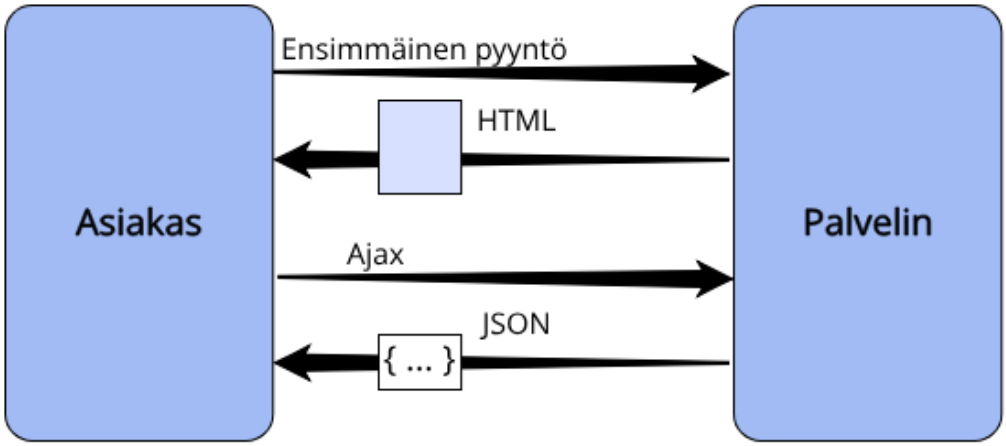
3.2.1 Toimintaperiaate

Yhden sivun sovellus koostuu vain yhdestä HTML-sivusta, joka toimii pohjana kaikille sovelluksen muille sivuille. Toisin kuin monen sivun sovellukset, jotka turvautuvat vahvasti palvelimeen, yhden sivun sovellukset painottuvat asiakaspuoleen, joka on toteutettu JavaScriptillä, CSS:llä ja HTML:llä. Yhden sivun sovellukset muistuttavatkin paljon natiiveja sovelluksia, jotka toimivat omissa prosesseissaan. Niistä poiketen yhden sivun sovellukset toimivat kuitenkin omien prosessien sijaan selainprosessissa. [1]

Yhden sivun sovelluksissa käytetään yhtä HTML-sivua, jonka sisältöä korvataan dynaamisesti siirryttäessä sivulta toiselle sovelluksen sisällä. Tämä toteutetaan asiakaspuolen reitittämisen ja mallimoottorin (engl. template engine) avulla. Asiakaspuolen reititin hallitsee navigointia sivulta toiselle sovelluksen sisällä ja mallimoottori on vastuussa näiden sivujen renderöimisestä tai uudelleenrenderöimisestä. Dynaamisen sisällönkorvauksen ansiosta yhden sivun sovelluksissa ei ole kokonaisia sivun uudelleenlatauksia. [1]

Palvelimella on yhden sivun sovelluksissa erilainen rooli kuin monen sivun sovelluksissa. Yhden sivun sovelluksissa palvelimen tehtävänä on lähettää sovellus asiakkaalle sekä tarjota asiakkaan tarvitsemia resursseja. Lisäksi palvelimella on ohjelmointirajapinta (engl. API, Application Programming Interface), jota voidaan käyttää sovelluksessa esimerkiksi todentamiseen (engl. authentication), valtuuttamiseen (engl. authorization) ja tietokannan muuttamiseen. Kuten suurin osa ohjelmointirajapinnoista, myös yhden sivun

sovelluksien palvelimien ohjelmointirajapinnat pyrkivät noudattamaan REST-arkkitehtuurimallia (Representational State Transfer). [1]



Kuva 4. *Yhden sivun sovelluksen elinkaari. [10]*

Kuvassa 4 on esitetty yhden sivun sovelluksen elinkaari. Jos sitä verrataan kuvan 3 vastaavaan monen sivun sovelluksen elinkaareen, voidaan huomata, että pääero on asiakkaan ja palvelimen välisessä kanssakäymisessä ensimmäisen pyynnön ja vastauksen jälkeen.

Yhden sivun sovelluksissa käytetään Ajaxia pyyntöjen tekemiseen. Palvelin vastaa pyyntöön käyttäen jotakin tiedostomuotoa, kuten JSON:ia tai XML:ää, tai esirenderöityjä (engl. prerendered) HTML-sivuja. Vastauksen saatuaan asiakas renderöi HTML-sivun osittain esittääkseen saamansa datan. Monen sivun sovelluksista poiketen, yhden sivun sovelluksissa sovelluksen sisäinen sivulta toiseen navigointi tapahtuu asiakkaan eikä palvelimen puolella. [1]

Yhden sivun sovelluksissa voidaan lisäksi hyödyntää tehokkaasti muistia tilanhallinnassa. Sovelluksen tila voidaan tallentaa selaimen paikalliseen muistiin, koska sovellus koostuu vain yhdestä sivusta, jota käyttäjä ei päivitä eikä poistu siltä. Tämän lisäksi sovelluksen tila voidaan tallentaa välimuistiin, jossa se voidaan säilyttää, vaikka käyttäjä menettäisi internet-yhteyden tai sulkisi selaimen. [1]

JavaScriptin suosion kasvaessa yhden sivun sovellusten asiakaspuolen koodista alkoi tulla yhä monimutkaisempaa. Tämän ratkaisemiseksi alettiin käyttämään MVC-mallia, jonka tarkoitus on auttaa kehittäjiä kirjoittamaan paremmin organisoitua koodia. [14]

Yhden sivun sovelluksia kehitettäessä käytetään usein jotakin JavaScript-sovelluskehystä asiakaspuolen toteutukseen. Suurin osa sovelluskehyksistä käyttää MVC-mallia

tai jotain sen varianttia pohjanaan. [3] Tällä hetkellä suosituimpia JavaScript-sovelluskehyksiä ovat muun muassa Angular, React ja Vue.js [2].

3.2.2 Hakukoneoptimointi yhden sivun sovelluksissa

Hakukoneoptimoinnin ymmärtämiseksi on tärkeää ymmärtää mitä hakukoneryömijät (engl. search engine crawlers) ovat. Hakukoneryömijöiden tehtävä on ryömiä websivujen läpi, jotta ne voidaan indeksoida hakukoneita varten. [1] Esimerkiksi Googlen käyttämä hakukoneryömijä Googlebot lataa ensin HTML-sivun, jonka jälkeen se poimii sivulta löytyvät linkit, käy jokaisen linkin sivulla samanaikaisesti ja jatkaa samaan tapaan eteenpäin. Kun ryömijä on valmis, sivuilta ladatut resurssit lähetetään indeksoijalle, joka lisää sivun indeksiin. [5] Indeksissä on kaikki hakukoneryömijöiden löytämä tieto [11]. Myöhemmin tätä hakukoneen indeksiin lisättyä tietoa käytetään hakukoneen algoritmeissa määrittämään sivujen järjestys hakutuloslistauksessa [1].

Portney kertoo blogikirjoituksessaan [5], että prosessista tulee kuitenkin monimutkaisempi, kun kyseessä on JavaScriptillä tehty yhden sivun sovellus. JavaScript-koodi täytyy ensin jäsentää (engl. parse), kääntää (engl. compile) ja suorittaa ennen kuin sivu voidaan antaa hakukoneryömijälle ryömittäväksi. Prosessiin tulee siis viivettä ja ylimääräisiä toimenpiteitä, jotka tekevät siitä hitaamman. [5]

Hakukoneoptimoinnissa sovellus optimoidaan hakukoneryömijöitä varten. Mitä paremmin web-sovellus on hakukoneoptimoitu, sitä paremman sijoituksen se saa hakukoneiden, esimerkiksi Googlen, hakutuloksissa. Parempi sijoitus ja ilmentyminen useammin hakutuloksissa vaikuttavat web-sovelluksen kävijämäärään, joten hakukoneoptimointi on tärkeä osa sovelluksen kehitystä. [1]

Vaikka Googlen hakukoneryömijät pystyvätkin ryömimään yhden sivun sovelluksia, niiden onnistuminen indeksoinnissa on Portneyn mukaan epäjohdonmukaista [5]. Yksikin pieni virhe sivun JavaScript-koodissa voi kokonaan estää Googlen indeksoimasta sivua, ja jos sivun skriptien suorittamisessa kestää liian kauan, Googlebot saattaa vain lopettaa sivun indeksoimisen. Tämän lisäksi ei ole näyttöä, että muut hakukoneet, kuten Bing, renderöisivät JavaScriptiä ollenkaan. [7] Tästä syystä yhden sivun sovelluksissa käytetäänkin usein muita keinoja hakukoneoptimoinnin toteuttamiseen [5]. Näistä keinoista esitellään seuraavaksi kaksi.

Portneyn mukaan yksi menetelmä hakukoneoptimoinnin toteuttamiseen yhden sivun sovelluksissa on tarjota palvelimelta valmiiksi renderöityjä sivuja hakukoneryömijöille [5]. Tämä valmiiksi renderöity sivu annetaan vain hakukoneryömijöille, ja käyttäjät saavat

normaalin version JavaScriptin kanssa [7]. Menetelmä voidaan toteuttaa käyttämällä jotakin väliohjelmistoa (engl. middleware), kuten *Prerender.io*:ta [16], tai käyttämällä palvelimella esimerkiksi *Headless Chromium* -työkalua [17].

Toinen menetelmä on niin sanottu isomorfinen JavaScript (engl. Isomorphic JavaScript), jota kutsutaan myös universaaliksi JavaScriptiksi (engl. Universal JavaScript) [5]. Siinä sekä käyttäjille että hakukoneryömijöille lähetetään valmiiksi renderöity sivu [7], eli ryömijöiden ei tarvitse huolehtia JavaScriptistä.

4. MENETELMIEN VERTAILU

Yhden ja monen sivun sovelluksilla on molemmilla hyviä ja huonoja puolia. Tässä luvussa vertaillaan menetelmiä kehittämisen sekä käyttämisen näkökulmista hyvien ja huonojen puolien selvittämiseksi. Vertailu tehdään eri lähteitä hyväksi käyttäen kiinnittämällä huomiota erityisesti siihen, mitä yhteistä ja mitä eroavaisuuksia lähteissä mainituilla kriteereillä on.

4.1 Vertailun kriteerit

Taulukossa 1 on esitetty kriteerit, joiden pohjalta menetelmiä vertaillaan ja lähteet, joissa kriteerit esiintyvät. Rasti tarkoittaa, että kriteeri on mainittu lähteessä. Kriteereistä kehittämisen helppous, hakukoneoptimointi, turvallisuus sekä skaalautuvuus liittyvät web-sovellusten kehittämiseen. Kriteereistä puolestaan nopeus ja käytettävyys liittyvät web-sovellusten käyttämiseen. Kriteereitä tarkastellaan tarkemmin alaluvuissa 4.2 ja 4.3. Lähteiksi on valittu alan ammattilaisten kirjoittamia tekstejä niiden luotettavuuden, alkuperän sekä tuoreuden perusteella.

Taulukko 1 *Vertailtavat kriteerit ja niiden esiintyminen eri lähteissä.*

Kriteeri	Fink ja Flatow [1]	Pukha [6]	Skólski [8]
Kehittämisen helppous	X	X	X
Hakukoneoptimointi	X	X	X
Turvallisuus		X	X
Skaalautuvuus		X	
Nopeus	X	X	X
Käytettävyys	X	X	

Ensimmäinen lähde [1] on Finkin ja Flatowin kirjoittama kirja ”Pro single page application development: Using backbone.js and ASP.NET”. Toinen lähde [6] on Pukhan kirjoittama blogikirjoitus ”Single-Page Apps vs Multiple-Page Web Apps: What to Choose for Web Development”. Kolmas lähde [8] on Skólskin kirjoittama blogikirjoitus ”Single-page application vs. multiple-page application”.

4.2 Web-sovellusten kehittäminen

Web-sovellusten kehittämiseen liittyy useita tekijöitä, jotka voivat vaikuttaa positiivisesti tai negatiivisesti kehittäjien työhön. Tässä alaluvussa vertaillaan näitä tekijöitä yhden ja monen sivun sovellusten välillä, ja pyritään selvittämään mitä kaikkea on syytä ottaa huomioon kutakin menetelmää käytettäessä.

4.2.1 Kehittämisen helppous

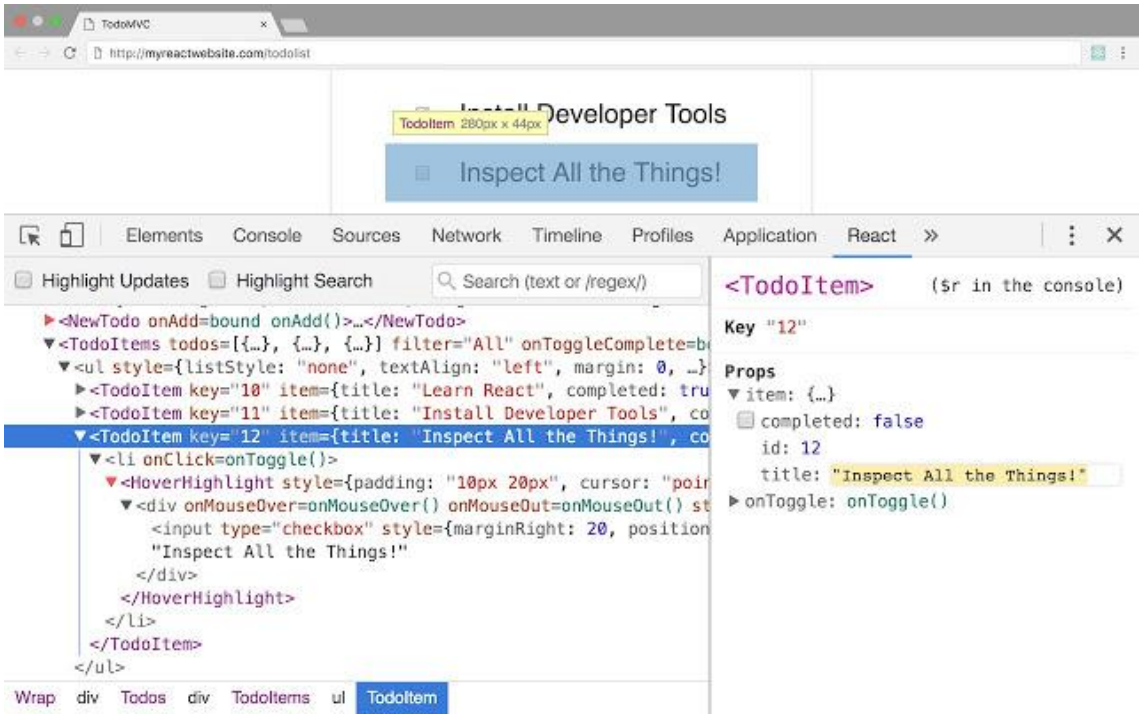
Skólski kertoo blogikirjoituksessaan [8], että yhden sivun sovelluksen kehittäminen on yksinkertaista ja suoraviivaista. Kaikki sivujen renderöimiseen liittyvä koodi sijaitsee asiakaspuolella, joten sitä ei tarvitse kirjoittaa palvelimelle, toisin kuin monen sivun sovelluksissa [8]. Tämä pitää palvelimen koodin yksinkertaisempana ja auttaa eriyttämään sovelluksen datan ja käyttöliittymän. Koska käyttöliittymä on eriytetty sovelluksen datasta, kehittäminen on lisäksi helpompi aloittaa, sillä palvelinta ei tarvita alussa [8].

Myös Pukha mainitsee blogikirjoituksessaan [6] datan ja käyttöliittymän eriyttämisen olevan hyvä asia yhden sivun sovelluksissa. Hän painottaa erityisesti sitä, että se auttaa huomattavasti sovelluksen testauksessa. Tämän lisäksi eriyttäminen mahdollistaa palvelimen muuttamisen ilman, että käyttöliittymää tarvitsee muuttaa välttämättä ollenkaan. [6] Tämä mahdollistaa kehittämisen helpon jakamisen niin sanottuihin backend- ja frontend-kehittäjiin, jotka tarkoittavat vastaavasti palvelinkehittäjiä ja käyttöliittymäkehittäjiä. Koska monen sivun sovelluksissa palvelimen ja käyttöliittymän kehitys ovat lähellä toisiaan [8], voi tällainen kehittämisen jako olla hankalampaa.

Toisaalta Skólski mainitsee, että jos käyttäjä kytkee JavaScriptin pois päältä selaimessa, voi kehittäjälle aiheutua ongelmia. Sivu on mahdollista luoda palvelimella ja lähettää käyttäjälle ilman JavaScriptiä, mutta sivun muu JavaScriptistä riippuvainen toiminnallisuus voi lakata toimimasta. [8] Tämä voi estää sovelluksen koko toiminnan. Palvelimesta tulee myös monimutkaisempi, jos sinne kehitetään sivujen renderöintiä, ja edellä mainittu etu datan ja käyttöliittymän eriyttämisestä pienenee.

Pukha kertoo lisäksi blogikirjoituksessaan [6], että asiakkaan selaimen taakse- ja eteenpäin siirtymiseen tarkoitetut napit eivät toimi suoraan yhden sivun sovelluksissa. Sovelluksen kehittäjät joutuvat erikseen kehittämään kyseisen toiminnallisuuden käyttämällä *History* API:a (Application Programming Interface) [6], joka mahdollistaa käyttäjän sivuhistorian manipuloinnin [13]. Monen sivun sovelluksissa puolestaan sivulta toiselle navigointi käyttäen selaimen painikkeita toimii suoraan ilman erillistä kehittämistä.

Pukhan sekä Skólskin mukaan yhden sivun sovelluksien testaus on myös helpompaa. Kaikki koodi sijaitsee yhdellä sivulla, joten sen tutkiminen selainten kehitystyökaluilla on helppoa. [6][8] Monen sivun sovelluksissa puolestaan joutuu navigoimaan sivulta toiselle, jotta kehitystyökaluilla pääsee tutkimaan jokaisen sivun koodia erikseen. Pukha mainitsee lisäksi, että esimerkiksi Google Chromelle on saatavilla useiden eri JavaScript sovelluskehyksien laajennuksia, joiden avulla yhden sivun sovelluksien testaus on entistä helpompaa [6]. Näistä esimerkiksi kuvassa 5 esitetty React Developer Tools -laajennus [15] tarjoaa hyödyllisiä ominaisuuksia Reactilla tehtyihin yhden sivun sovelluksiin.



Kuva 5. *Esimerkki React Developer Tools -laajennuksen toiminnasta. [15]*

Lisäksi on hyvä pitää mielessä, että yhden sivun sovelluksien tekeminen edellyttää vahvaa JavaScriptin osaamista. Jos kehitystiimillä on aiempaa kokemusta vain monen sivun sovelluksien tekemisestä ja muista ohjelmointikielistä, voi siirtyminen tai nykyisen sovelluksen päivitys yhden sivun sovellukseen olla liian haastavaa ja kallista.

Skólski sekä Pukha kertovat blogikirjoituksissaan myös, että yhden sivun sovelluksissa käytettävää palvelinta voidaan käyttää helpommin mobiilisovelluksen tekemiseen [6][8]. Koska data ja käyttöliittymä ovat eriytetty yhden sivun sovelluksissa, mobiilisovellukseen tarvitsee vain kehittää käyttöliittymä. Pukha lisää, että mobiilisovelluksen käyttöliittymässäkin voidaan hyödyntää yhden sivun sovelluksen käyttöliittymän suunnittelua, sillä se muistuttaa muutenkin enemmän natiivia sovellusta kuin web-sovellusta [6]. Vaikka myös

monen sivun sovelluksen palvelinta voidaan hyödyntää mobiilisovelluksessa, on se haastavampaa, koska data ja käyttöliittymä ovat lähempänä toisiaan.

4.2.2 Hakukoneoptimointi

Vertailuun valitut lähteet [1][6][8] ovat kaikki yksimielisiä siitä, että hakukoneoptimointi on yhden sivun sovelluksissa haastavampaa kuin monen sivun sovelluksissa. Tämä johtuu suurimmaksi osaksi siitä, että yhden sivun sovelluksissa sivun sisältö generoidaan käyttämällä JavaScriptiä sekä sitä muutetaan dynaamisesti Ajaxilla. Vaikka Google pystyykin jossain määrin käsittelemään JavaScriptiä sovellusta indeksoidessa, siihen liittyy useita ongelmia, joita on käsitelty luvussa 3.2.2.

Hakukoneoptimointi on kuitenkin mahdollista toteuttaa yhden sivun sovelluksiin esimerkiksi luvussa 3.2.2 esitetyillä tavoilla. Onkin oleellista selvittää, kuinka tärkeää hakukoneoptimointi on sovellukselle ja kuinka paljon aikaa ja rahaa siihen on kannattavaa käyttää. Jos se on sovellukselle kriittistä, voi olla järkevää palkata hakukoneoptimointiin erikoistunut asiantuntija varsinkin, jos kehitystiimillä ei ole paljon aikaisempaa kokemusta yhden sivun sovelluksien hakukoneoptimoinnista.

Toisin kuin yhden sivun sovelluksissa, hakukoneryömijät toimivat suoraan staattisilla sivuilla, joita käytetään monen sivun sovelluksissa. Täten monen sivun sovellukset eivät vaadi välttämättä ollenkaan hakukoneoptimointia kehittäjiltä. Monen sivun sovelluksissakin käytetään kuitenkin usein Ajaxia sivujen osittaiseen uudelleenrenderöintiin, joten hakukoneoptimointi voi olla jossain määrin niissäkin tarpeen, jos Ajax renderöi hakukoneiden kannalta kriittistä tietoa sivulle.

4.2.3 Turvallisuus

Sekä Skólski että Pukha väittävät, että yhden sivun sovellukset ovat vähemmän turvallisia kuin monen sivun sovellukset. He mainitsevat XSS:n (cross-site scripting) olevan ongelma erityisesti yhden sivun sovelluksissa. [6][8] XSS:n avulla hyökkääjä voi hyödyntää sovelluksessa mahdollisesti piilevää heikkoutta injektoimaan vahingollisia skriptejä sovellukseen. Tämän seurauksena uhri voi olla haavoittuva esimerkiksi evästeiden varastamiseen tai näppäilyjen tallentamiseen (engl. key logging). [4] Ongelman välttämiseksi kehittäjien tulisi suojata sovelluksen datan päätepisteet (engl. endpoints) erityisen huolellisesti [6].

Modernit yhden sivun sovelluksien kehitykseen käytettävät JavaScript-sovelluskehykset tarjoavat kuitenkin lukuisia turvallisuutta parantavia ominaisuuksia. Kehittäjien täytyy

vain olla tietoisia mahdollisista turvallisuusriskeistä ja käyttää sovelluskehyksien tarjoamia hyväksi todettuja toimintatapoja riskien ehkäisemiseksi.

Monen sivun sovellukset eivät tietenkään ole täysin turvallisia automaattisesti. Ne ovat kuitenkin vuosikymmeniä vanha menetelmä, joten niiden kehittämiseen käytetyt sovelluskehykset ovat ehtineet kypsyä runsaasti ja tarjoavat lukuisia vuosien saatossa testattuja menetelmiä turvallisuusriskien ehkäisemiseksi.

4.2.4 Skaalautuvuus

Pukha kertoo blogikirjoituksessaan [6], että monen sivun sovellukset ovat paremmin skaalautuvia kuin yhden sivun sovellukset. Tämä johtuu siitä, että monen sivun sovelluksissa sisältö ja ominaisuudet ovat jaettu usealle sivulle. Uusien ominaisuuksien lisääminen vaatii vain uusien sivujen luomista.

Yhden sivun sovelluksissa puolestaan ominaisuudet sijaitsevat sovelluksen yhdellä sivulla. Liian monen ominaisuuden lisääminen tähän yhteen sivuun johtaa latausaikojen pitenemiseen [6]. Tämän vuoksi on tärkeää selvittää etukäteen, kuinka paljon sisältöä sivulla tulee olemaan, ja valita sen perusteella oikea menetelmä sovelluksen kehittämiseen.

4.3 Web-sovellusten käyttäminen

Web-sovelluksille on erittäin tärkeää, että niiden käyttäminen on mahdollisimman helppoa ja nopeaa. Tässä alaluvussa esitetään yhden ja monen sivun sovellusten eroja nopeudessa ja käytettävyydessä.

4.3.1 Nopeus

Skólski kertoo blogikirjoituksessaan [8], että yhden sivun sovellus on kokonaisuudessaan nopeampi kuin monen sivun sovellus. Tämä johtuu siitä, että suurin osa resursseista, kuten HTML, CSS ja skriptit, ladataan vain kerran palvelimelta sovelluksen käytön aikana [8]. Kun sovellusta käytetään, pelkästään dataa lähetetään asiakkaan ja palvelimen välillä, ja lähetettävä data on usein vain sivun osan uudelleenrenderöintiä varten. Tämän lisäksi Pukha mainitsee blogikirjoituksessaan [6], että yhden sivun sovelluksissa vasteajat ovat lyhyempiä, koska datan prosessointi suoritetaan palvelimen sijasta asiakkaalla.

Fink ja Flatow kertovat kirjassaan [1], että monen sivun sovellukset ovat hitaampia, koska käyttäjän tekemät toiminnot käsitellään palvelimella. Tämän lisäksi sovelluksen tilaa ja dataa joudutaan kysymään ja päivittämään palvelimella jatkuvasti, mikä hidastaa sovellusta entisestään [1].

Vaikka yhden sivun sovellukset ovat kokonaisuudessaan nopeampia, on ensimmäinen lataus niissä hidas. Tämä johtuu siitä, että ne ovat erittäin asiakaspainotteisia, ja raskaat JavaScript-sovelluskehykset sekä HTML ja CSS joudutaan lataamaan käyttäjän selaimeen [6][8].

4.3.2 Käytettävyys

Yhden sivun sovelluksien nopeus johtaa suoraan myös parempaan käytettävyyteen monen sivun sovelluksiin verrattuna. Koska sivua ei tarvitse ladata uudelleen ja vasteajat ovat lyhyempiä, käyttäjän suorittamat toiminnot ovat sovelluksessa hyvin nopeita. Tämä johtaa sulavaan käyttäjäkokemukseen ilman, että käyttäjän toiminta katkeilee sisältöä ladatessa ja sovelluksen sisällä navigoidessa, kuten monen sivun sovelluksissa. Fink ja Flatow vertaavatkin yhden sivun sovelluksien käytettävyyttä natiiveihin sovelluksiin [1].

Monen sivun sovelluksissa käyttäjä joutuu odottamaan jatkuvasti, kun palvelimen kanssa keskustellaan, mikä johtaa huonoon käyttäjäkokemukseen. Monen sivun sovelluksien käyttäjäkokemusta voidaan kuitenkin parantaa käyttämällä Ajaxia ainoastaan sivun osien uudelleenrenderöintiin kokonaisen uudelleenlatauksen sijaan. [1] Kaikki vertailussa olevat lähteet [1][6][8] ovat kuitenkin kokonaisuudessaan yksimielisiä yhden sivun sovelluksien paremmasta käytettävyydestä.

Kuten edellisessä alaluvussa mainittiin, sovelluksen koosta riippuen ensimmäinen lataus yhden sivun sovellukseen mentäessä voi kuitenkin olla hidas. Tämä voi häiritä käyttäjiä, jotka odottavat pääsevänsä käyttämään sovellusta nopeasti, mutta joutuvatkin odottamaan. Tällaisista pitkistä palvelinoperaatioista voidaan kuitenkin ilmoittaa käyttäjälle erilaisilla ilmoituksilla tai animoiduilla kuvilla, mikä parantaa käytettävyyttä [1].

Kaikki vertailussa käytettävät lähteet [1][6][8] mainitsevat, että yhden sivun sovelluksien käytettävyyttä parantaa myös tehokas välimuistin käyttö. Sovellukseen mentäessä palvelimelta saadut resurssit voidaan tallentaa välimuistiin ja käyttöä voidaan jatkaa, vaikka käyttäjä menettäisi internetyhteyden. Kun internetyhteys palaa, voidaan sovelluksen asiakaspuoli synkronoida palvelimen kanssa. [6] Tämä ei ole mahdollista monen sivun sovelluksissa [1], koska ne ovat hyvin riippuvaisia palvelimen kanssa keskustelemisesta.

5. VERTAILUN TULOKSET

Työssä suoritetun vertailun perusteella selvitettiin yhden ja monen sivun sovellusten hyviä ja huonoja puolia. Tuloksena saatiin molemmille web-kehityksen menetelmille ominaisia piirteitä. Taulukossa 2 on esitetty yhteenveto yhden ja monen sivun sovellusten ominaisuuksista, jotka selvitettiin kandidaatintyössä.

Taulukko 2 *Yhden ja monen sivun sovellusten ominaisuuksia.*

Ominaisuus	Yhden sivun sovellus	Monen sivun sovellus
Data ja käyttöliittymä erotettu	Kyllä	Ei
Riippuvainen JavaScriptistä	Kyllä	Osittain, jos JavaScriptiä käytetty asiakaspuolella
Navigointi selaimen painikkeilla	Kyllä, mutta pitää kehittää erikseen	Kyllä
Testattavuus	Helppo	Melko helppo
Saman palvelimen käyttö mobiilisovelluksessa	Helppo	Melko vaativa
Hakukoneoptimointi	Melko vaativa	Helppo
Turvallisuus	Kohtalainen	Hyvä
Skaalautuvuus	Huono	Hyvä
Nopeus	Hyvä, mutta ensimmäinen lataus hidas	Kohtalainen
Käytettävyys	Hyvä	Huono, mutta parannettavissa Ajaxilla
Käyttö ilman internetyhteyttä	Kyllä	Ei

Yhden sivun sovellukset ovat erityisen vahvoja käytettävyyden suhteen. Ne ovat nopeita ensimmäistä latausta lukuun ottamatta ja tarjoavat käyttäjille sulavan käyttäjäkokemuksen ilman katkeilua sovellusta käytettäessä. Lisäksi niiden käyttöä voidaan jatkaa ilman internetyhteyttä, jos käyttäjän toiminnot eivät vaadi palvelinta.

Yhden sivun sovellukset tarjoavat lisäksi kehittäjille selviä etuja. Datan ja käyttöliittymän erottaminen helpottaa kehittämistä, ja sovelluskehyksille on tarjolla testausta helpottavia laajennuksia. Lisäksi yhden sivun sovelluksen palvelinta ja käyttöliittymän suunnittelua voidaan hyödyntää helposti mobiilisovelluksen kehityksessä.

Yhden sivun sovelluksissa on myös huonoja puolia, jotka täytyy ottaa huomioon. Ne ovat riippuvaisia JavaScriptistä, ja kehittäjiltä vaaditaan vahvaa JavaScriptin osaamista sovelluksen kehityksessä. Kehittäjien täytyy myös kehittää erikseen selaimen painikkeilla navigointi, ja hakukoneoptimointi vaatii vaivaa, jos se on tärkeää sovellukselle. Lisäksi yhden sivun sovelluksien turvallisuuteen pitää kiinnittää erityistä huomiota ja huonon skaalautuvuuden vuoksi ne eivät sovi sovelluksiin, joissa on paljon ominaisuuksia ja sisältöä.

Monen sivun sovellukset tarjoavat kaikin puolin huonomman käyttäjäkokemuksen kuin yhden sivun sovellukset. Suuri riippuvuus palvelimesta tekee sovelluksesta hitaan, ja sivujen uudelleenlataukset huonontavat käytettävyyttä. Käytettävyyttä voidaan kuitenkin parantaa Ajaxilla, mutta se vaatii ylimääräistä työtä ja lisää monimutkaisuutta. Monen sivun sovelluksia ei voida myöskään käyttää ilman internetyhteyttä.

Monen sivun sovelluksilla on kuitenkin etuja kehittämiseen liittyen. Hyvä skaalautuvuus mahdollistaa menetelmän käytön hyvin suurissa sovelluksissa, mikä on suuri etu yhden sivun sovelluksiin verrattaessa. Lisäksi hakukoneoptimointi toimii monen sivun sovelluksissa ilman suurta vaivaa ja selaimen painikkeilla navigointi toimii suoraan. Monen sivun sovellukset eivät myöskään vaadi välttämättä vahvaa JavaScript osaamista ja ne ovat turvallisia. Palvelimen ja käyttöliittymän läheisyys tekee kuitenkin kehittämisestä monimutkaisempaa ja vaikeuttaa palvelimen käyttöä mobiilisovelluksessa.

Valinta yhden ja monen sivun sovelluksen välillä riippuu täysin siitä, mitä sovelluksella on tarkoitus tehdä. Yhden sivun sovellus toimii hyvin, jos tarkoitus on tarjota vähäinen määrä tuotteita tai palveluita ja käyttäjäkokemus on tärkeä. Monen sivun sovellus puolestaan toimii hyvin, jos sovellus on monimutkainen ja tarjoaa paljon erilaisia tuotteita tai palveluita.

6. YHTEENVETO

Tässä työssä selvitettiin, miten yhden ja monen sivun sovellukset toimivat sekä vertailtiin niitä kirjallisuuden avulla niiden hyvien ja huonojen puolien selvittämiseksi. Tuloksina saatiin selville, että yhden sivun sovellukset ovat vahvoja käytettävyydessä ja nopeudessa, ja että ne tarjoavat kehittäjille monia etuja datan ja käyttöliittymän eriyttämisen ansiosta. Yhden sivun sovellukset sopivat hyvin pienempiin sovelluksiin, joiden tavoitteena on mahdollisimman hyvä käyttäjäkokemus. Monen sivun sovelluksista saatiin selville, että ne ovat hyviä hakukoneoptimoinnin kannalta. Lisäksi huonommasta käytettävyydestä ja nopeudesta huolimatta ne sopivat hyvin laajoihin sovelluksiin hyvän skaalautuvuuden ansiosta.

Työn vertailussa olisi voitu käyttää enemmän lähteitä useampien näkökulmien saamiseksi. Lisäksi työssä olisi voitu tehdä esimerkkisovellukset kirjallisuuden tueksi. Esimerkkisovelluksien avulla olisi voitu mitata esimerkiksi nopeutta konkreettisesti.

Jatkossa aihetta voisi laajentaa tarkastelemalla hybridisovelluksia, joissa pyritään yhdistämään yhden ja monen sivun sovelluksien parhaat puolet. Lisäksi voitaisiin tarkastella enemmän luvussa 3.2.2 lyhyesti mainittua isomorfista JavaScriptiä, jonka sanotaan olevan evoluutio yhden sivun sovelluksista.

LÄHTEET

[1] G. Fink, I. Flatow, Pro single page application development: Using backbone.js and ASP.NET, Apress, 2014.

[2] J. Hannah, The Ultimate Guide to JavaScript Frameworks, 2018. Saatavissa (viitattu 23.03.2019): https://jsreport.io/the-ultimate-guide-to-javascript-frameworks/

[3] D. Kerr, An introduction to MVC frameworks, 2013. Saatavissa (viitattu 23.03.2019): https://blog.scottlogic.com/2013/12/06/JavaScript-MVC-frameworks.html

[4] K. Nirmal, B. Janet, R. Kumar, Web Application Vulnerabilities – The Hacker's Treasure, International Conference on Inventive Research in Computing Applications (ICIRCA), 2018. Saatavissa (viitattu 02.04.2019): https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8597221

[5] D. Portney, An SEO's survival guide to Single Page Applications (SPAs), 2018. Saatavissa (viitattu 29.03.2019): https://searchenginewatch.com/2018/04/09/an-seos-survival-guide-to-single-page-applications-spas/

[6] D. Pukha, Single-Page Apps vs Multiple-Page Web Apps: What to Choose for Web Development. Saatavissa (viitattu 20.03.2019): https://yalantis.com/blog/single-page-apps-vs-multiple-page-apps/

[7] T. Rudzki, The Ultimate Guide to JavaScript SEO, 2018. Saatavissa (viitattu 30.03.2019): https://www.onely.com/blog/ultimate-guide-javascript-seo/

[8] P. Skólski, Single-page application vs. multiple-page application, 2017. Saatavissa (viitattu 18.03.2019): https://neoteric.eu/blog/single-page-application-vs-multiple-page-application/

[9] V. Solovei, O. Olshevska, Y. Bortsova, The difference between developing single page application and traditional web application based on mechatronics robot laboratory onaft application, Automation technological and business-processes, Vol 10, 2018. Saatavissa (viitattu 24.02.2019): https://doi.org/10.15673/atbp.v10i1.874

[10] M. Wasson, ASP.NET - single-page applications: Build modern, responsive web apps with ASP.NET, 2013. Saatavissa (viitattu 10.02.2019): https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[11] What Is Search Engine Index, Brick Marketing. Saatavissa (viitattu 11.03.2019): https://www.brickmarketing.com/define-search-engine-index.htm

[12] Ajax, MDN Web Docs. Saatavissa (viitattu 17.03.2019): https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

[13] Manipulating the browser history. Saatavissa (viitattu 20.03.2019): https://developer.mozilla.org/en-US/docs/Web/API/History_API

[14] MVC Architecture, Chrome Developer. Saatavissa (viitattu 23.03.2019): https://developer.chrome.com/apps/app_frameworks

[15] React Developer Tools, Chrome Web Store. Saatavissa (viitattu 29.03.2019): https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

[16] Prerender.io. Saatavissa (viitattu 30.03.2019): https://prerender.io/

[17] Headless Chromium, verkkosivu. Saatavissa (viitattu 30.03.2019): https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md

[18] Number of internet users worldwide from 2005 to 2018 (in millions), Statista, 2019. Saatavissa (viitattu 07.04.2019): https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/