LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

MVVM-mallin toteutus KnockoutJS-kirjastoa käyttäen

Heikkilä, Jukka

2015 Kerava

Laurea-ammattikorkeakoulu
Laurea Kerava

MVVM-mallin toteutus KnockoutJS-kirjastoa käyttäen

Jukka Heikkilä
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2015

Laurea-ammattikorkeakoulu **Tiivistelmä**
Laurea Kerava
Tietojenkäsittelyn koulutusohjelma

Jukka Heikkilä

MVVM-mallin toteutus KnockoutJS-kirjastoa käyttäen

Vuosi 2015 Sivumäärä 34

Opinnäytetyön aiheena oli web-sovelluksen käyttöliittymäohjelmoinnin toteutus käyttäen Model-View-ViewModel (MVVM)-mallia. Työn tarkoituksena oli luoda toimiva käyttöliittymä ja kerätä tietoa käyttöliittymän toteutustavasta.

Opinnäytetyön tutkimusongelmana oli selvittää, kuinka saadaan toteutettua käyttöliittymään yksinkertaisia elementtejä käyttäen MVVM-mallia ja mitä hyötyä sen käytöstä on sovelluskehityksessä. Tutkimustulokset ovat peräisin omista toteutustavoista käyttäen laadullista tutkimusmenetelmää. Aineiston keruu tapahtui oman työskentelyni kautta samalla, kun kehitin uutta toiminnallisuutta osaksi käyttöliittymää.

Tuloksena opinnäytetyössä oli valmis arkkitehtuuritoteutus sekä HTML-elementit, jotka toimivat MVVM-mallin avulla käyttäen KnockoutJS-JavaScript-kirjastoa. Tarkoituksena oli, että opinnäytetyössä näytettävät toteutustavat ovat toimivia kokonaisuuksia.

Asiasanat Web-sovelluskehitys, käyttöliittymä, JavaScript, MVVM

Laurea University of Applied Sciences **Abstract**
Laurea Kerava
Degree Programme in Business Information Technology

Jukka Heikkilä

Implementing MVVM pattern using KnockoutJS library

Year 2015 Pages 34

The subject of this thesis was making user interface for web application by using Model-View-ViewModel (MVVM)-pattern. The purpose of the work was to create a workable user interface and to gather information about implementation of user interface.

The research problem of the thesis was to find out how to implement elements of user interface by using MVVM-pattern and what are the benefits of using MVVM in application development. Research results are derived from my own methods of implementations using qualitative research method. The data collection happened while working on a project and developing new functionalities into the user interface.

As a result from the thesis was a complete architecture of user interface and HTML elements that work using MVVM-pattern and KnockoutJS JavaScript library. It was intended that methods of implementation are functional entities.

Keywords Web application development, user interface, JavaScript, MVVM

Sisällys

1 Johdanto

Tässä opinnäytetyössä tutkin, kuinka saadaan toteutettua web-sovelluksen selainrajapinta eli Frontend käyttäen JavaScriptiin sijoitettavaa Model-View-ViewModel -mallia ja KnockoutJS-kirjastoa. Kerron muun muassa sen, kuinka selaimelle saapuva data saadaan liitettyä selaimen tekstikenttiin ja taulukoihin, sekä kuinka sitä voidaan muokata hyödyntäen KnockoutJS-kirjaston ominaisuuksia. Käyn myös läpi KnockoutJS-kirjaston ja MVVM-mallin hyötyjä ja haittoja verrattuna perinteisen JavaScript-syntaksin käyttöön.

Aluksi käyn läpi työn lähtökohtia. Kerron mitä projektissa ollaan tekemässä ja mitä on tarkoitus saada aikaan. Kerron opinnäytetyöprosessista, opinnäytetyön aihealueesta, viitekehyksestä sekä tutkimuskysymyksistä. Sitten käyn läpi opinnäytetyön tutkimusmenetelmiä sekä kerron tutkimuksen validiteetista ja reliabiliteetista. Tämän jälkeen kerron itse projektista, jonka pohjalta opinnäytetyö on toteutettu. Projektiin liittyy ensin käyttöliittymän perustietojen läpikäynti, jossa esittelen projektissa käytettyjä tekniikoita ja sen jälkeen kerron, mitä itse kohdeprojektissa on tehty ja millaisia tuloksia on saatu.

2 Keskeiset käsitteet

HTML	Hypertext Markup Language. Hypertekstin merkintäkieli
JavaScript	Komentosarjakieli
Ajax	Asynchronous JavaScript And XML. Asynkroninen tiedonvälitystekniikka
Ajax-kutsu	Yksittäinen Ajax-tekniikalla tehty HTTP-pyyntö
HTTP	Hypertext Transfer Protocol. Tiedonsiirtoon käytettävä protokolla
Frontend	Web-sovelluksen esityskerros
Backend	Web-sovelluksen sovelluskerros
JSON	JavaScript Object Notation. Merkkijonomuotoinen tietojoukko
ASP.NET	Microsoftin kehittämä ohjelmointirunko
Silmukka/looppi	Toistorakenne, jossa ohjelma suorittaa tietyn koodin n-kertaa

3 Työn lähtökohdat

Tutkimuksen lähtökohtana on tarve rakentaa web-sovellus. Jotta web-sovellus saadaan toteutettua halutunlaiseksi, on sen toimittava tietyllä tavalla. Sovellusta varten määritetään tekniikat, joilla työ toteutetaan sekä käytännöllinen toiminta eli se, mitä toimintoja

sovelluksessa tulee olla ja kuinka niiden tulee toimia. Sovelluksen suunnittelijat ja arkkitehdit ovat määrittäneet käytettävät tekniikat sovelluskehitykseen. En itse ole ollut vaikuttamassa siihen, mitä tekniikoita projektissa käytetään, enkä näin ollen ota kantaa siihen, olisiko projekti parempi kehittää jollain muulla tekniikalla. Mahdollisia tapoja kehittää vastaavanlainen sovellus on useita, mutta luotan siihen, että sovelluksen arkkitehtuurillinen suunnittelu on pätevien arkkitehtien tekemää. Myös sovelluksen toiminnallisuudet on pääpiirteiltään määrittänyt suunnittelijat, vaikkakin kehittäjillä on mahdollisuus vaikuttaa lopullisiin toiminnallisuuksiin. Tutkimuksen kohteena olevassa projektissa on mukana vaihtelevasti 3-8 työntekijää riippuen työntekijöiden muista projekteista. Opinnäytetyön tutkimuksen virallisen kohteen eli sovelluksen selainosion toteutuksessa työskentelen pääosin kuitenkin yksin.

3.1 Aihealueen rajaus

Tämä opinnäytetyö on luonteeltaan toiminnallinen tutkimus. Työn tarkoituksena on rakentaa uutta ja samalla kerätä tietoa kohteesta. Työn tutkimusstrategiana on projektin toteutus alusta loppuun ja samalla projektissa tuotettavan tiedon keräys. Tiedon keräys tapahtuu projektin edetessä samalla, kun työ etenee ja uusia projektiin liittyviä aiheita tulee esiin. Projektin tuotetaan esimerkkejä MVVM-mallin toteutuksesta, jota käytetään apuna tiedon näyttämiseen opinnäytetyössä.

Tässä opinnäytetyössä ei ole tarkoitus tutkia koko sovelluksen toteutusta, eikä näin ollen keskitytä ollenkaan sovellukseen kokonaisuutena eikä tutkia esimerkiksi tietokantaa tai muita sovelluksen käyttöliittymärajapinnan ulkopuolelle jääviä osuuksia. Lisäksi tässä opinnäytetyössä ei näytetä, miten käyttöliittymäohjelmointi on todellisuudessa tehty projektissa vaan tutkitaan, miten saadaan toteutettua yksinkertaisia käyttöliittymäkomponentteja ja elementtejä käyttäen samoja tekniikoita, joita projektissa itsessään on käytetty.

Tavoitteena on toteuttaa web-sovelluksen selainnäkymä. Selainnäkymän eli sovelluksen Frontendin tulee pystyä toimimaan vaadituilla tavoilla. Selainnäkymän tulee esimerkiksi pystyä vastaanottamaan ja lähettämään tietoa asynkronisesti, tarkoittaen sitä, että sen on pystyttävä toimimaan yhdessä sovelluksen palvelinkerroksen kanssa ilman, että sivustoa tulee erikseen päivittää missään välissä. Käytännössä tämä tarkoittaa sitä, että kun käyttöliittymässä halutaan näyttää jotain tietokannasta haettavaa tietoa, tulee sovelluksen osata liittää tieto sovelluksen tiettyihin elementteihin ilman, että muut elementit reagoivat siihen mitenkään. Sovelluksen on myös pystyttävä varastoimaan dataa ja näyttämään sitä käyttäjälle useissa määritetyissä näkymissä. Lisäksi sovellus on kokonaisuudessaan toteutettava Microsoftin ASP.NET ympäristössä ja sen käyttöliittymä on toteutettava käyttäen

MVVM-mallia ja KnockoutJS-kirjastoa. KnockoutJS-kirjaston lisäksi projektissa on sallittua käyttää muita avoimen lähdekoodin JavaScript-kirjastoja, joita on sallittua käyttää kaupalliseen tarkoitukseen. Vaikka useita eri kirjastoja onkin sallittua käyttää, on koodi pääosin tarkoitus tuottaa itse ja käyttää kirjastoja vain niissä tapauksissa, joissa niille nähdään suuria hyötyjä.

3.2 Viitekehys ja tutkimuskysymykset

Työn viitekehyksenä käytetään niin minun, kuin muunkin projektitiimin näkemyksiä, sekä siihen liittyvää kirjallisuutta ja dokumentaatiota. Keskeisin tutkimuksessa käytettävä näkökulma on projektissa työskentelevien työntekijöiden tietous ja näkemys tutkimuksen kohteesta. Vaikka työssä hyödynnetäänkin useiden työntekijöiden näkemyksiä, työ toteutetaan pääosin oman näkemyksen ja osaamisen pohjalta.

Opinnäytetyön tutkimusongelmana on selvittää, kuinka saadaan toteutettua käyttöliittymään yksinkertaisia elementtejä käyttäen MVVM-mallia ja mitä hyötyä sen käytöstä on sovelluskehityksessä. Näin ollen tarkoituksena on rakentaa käyttöliittymä, joka käyttää MVVM-mallia sekä Knockoutin syntaksia ja kerätä tietoa näistä toteutustavoista.

Tutkimuksen tutkimuskysymykset ovat seuraavia:

- Kuinka liitetään MVVM-malli osaksi web-sovelluksen selainrajapintaa?
- Kuinka toteutetaan yleiset HTML-elementit käyttäen KnockoutJS-kirjaston syntaksia?
- Mitä hyötyä MVVM:stä ja Knockoutista on?

4 Käyttöliittymäohjelmoinnin tietoperusta

Tässä luvussa kerron käyttöliittymäohjelmoinnin perustiedoista, joita käytetään opinnäytetyön tutkimuskohteen toteutuksessa. Kaikki tämän luvun aiheet liittyvät suoraan opinäytetyön tutkimuskohteeseen.

4.1 Web-sovellus

Web-sovellus on ohjelma, joka toimii verkkoyhteyden kautta joko käyttäjän selaimella tai erillisessä paikallisella client-sovelluksella. Web-sovellus kattaa kokonaisen projektin aina tietokannasta lähtien käyttöliittymään asti (Casteleyn, Daniel, Dolog & Matera 2009, 3). Web-sovelluksella on mahdollista käsitellä tietokannassa sijaitsevaa tietoa käyttöliittymän kautta. Web-sovelluksesta hyvä esimerkki voisi olla vaikka Facebook, jossa jokaisella käyttäjällä on oma profiili, jonka tiedot, kuten esimerkiksi käyttäjätunnus ja salasana sekä

kaikki muutkin käyttäjäkohtaiset tiedot sijaitsevat tietokannassa. Lisäksi Facebookin käyttäjän jokainen Facebook-ystävä on sidoksissa käyttäjään tietokannan avulla.

Web-sovellus koostuu useasti tietokannasta, sovelluksesta eli backendistä ja käyttöliittymästä eli frontendistä. Backendistä voidaan käyttää myös nimeä sovelluskerros, palvelinkerros tai taustajärjestelmä sekä frontendistä nimeä web-kerros, web-rajapinta, selainkerros, käyttöliittymä tai esityskerros. Jokaisella web-sovelluksen osalla on oma tehtävänsä. Tietokanta varastoi kaikki web-sovelluksessa käytettävät datat tietokantatauluihin joista Backend kyselee niitä. Backend on yleensä sovelluksen monimutkaisin osio. Backendin tehtävä on kysellä dataa tietokannasta ja lähettää sitä eteenpäin selainkerrokselle sekä tehdä mahdollisesti datalla jotain siinä välissä. Sovelluskerroksessa dataa voi myös muokata tai datalla voi tehdä esimerkiksi laskuja tai algoritmeja. Lisäksi web-sovelluksen sovelluskerrosta voi käyttää erilaisten datasta riippumattomien toiminnallisuuksien toteutukseen, mikäli siihen on tarvetta. Backend on yleensä toteutettu jollain olio-ohjelmointikielellä, kuten esimerkiksi Javalla tai C#:lla, mikä mahdollistaa sen monipuolisen käytön. (Casteleyn, Daniel, Dolog & Matera 2009, 2-4.)

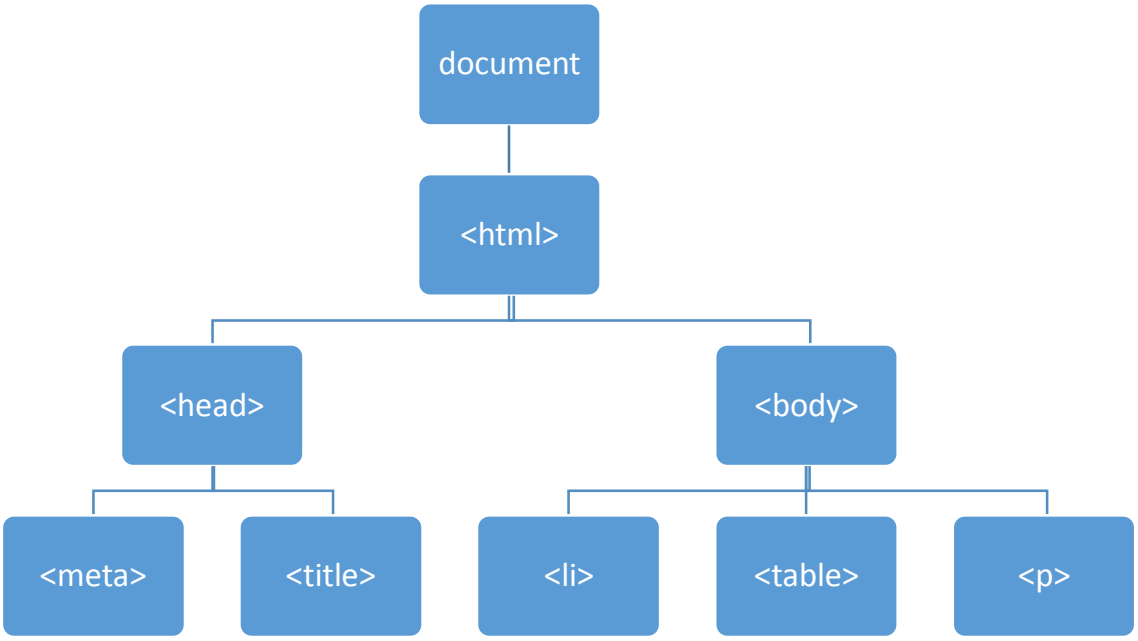
Frontend on web-sovelluksen näkyvä osa. Siihen kuuluu kaikki, mitkä liittyvät suoraan selaimella näkyviin sivuihin. Frontendiin kuuluu HTML-osuus, joka toimii keskeisimpänä osana käyttöliittymää. Sen viereen on mahdollista rakentaa toiminnallisuuksia ja käytettävyyttä hyödyntäviä ominaisuuksia sekä tyylejä, joiden avulla käyttöliittymän ulkoasusta voidaan rakentaa omiin tarpeisiin sopiva ja loppukäyttäjille miellyttävä. (Casteleyn ym. 2009, 2-4.)

4.2 Hypertext Markup Language ja Document Object Model

Hypertext Markup Language eli HTML on puhtaasti web-sivun näyttämiseen tarkoitettu ohjelmointikieli, joka tulee lyhenteestä Hypertext Markup Language. HTML-koodia on siis kaikki selaimella näkyvät elementit, kuten esimerkiksi painikkeet ja tekstikentät. HTML perustuu puumaiseen Document Object Model (DOM)-rakenteeseen ja sen sisältämiin nodeihin eli solmuihin. Alkujaan DOM kehitettiin jo 90-luvun puolivälissä, mutta se tuli suosituksi vasta vuosituhannen vaihteen IT-buumin aikaan. Ensimmäinen versio DOM-mallista oli nimeltään DOM taso 0, joka erosi myöhemmistä DOM-malleista siten, että se ei noudattanut yleistä W3C-standardia. DOM taso 0 mahdollisti muun muassa fonttien ja sivulle sijoitettavien elementtien sijainnin muokkaamisen. Vuonna 1998 W3C kehitti ensimmäinen standardoidun DOM-mallin nimeltään DOM 1. DOM 1 mahdollisti muistinvaraisen puumaisen rakenteen HTML-dokumentteihin, jota voidaan muokata erilaisilla ohjelmointikielillä. DOM 1:n mukana tuli myös DOM-mallin tuki muun muassa JavaScript-kielelle, joka vakiinnutti JavaScriptin osana web-ohjelmointia. Tärkeimmät selainvalmistajat ovat sitoutuneet tukemaan DOM-

standardeja, mikä on mahdollistanut esimerkiksi JavaScriptin nousun suureen suosioon (Peltomäki 2007, 10).
DOM (Document Object Model) tarkoittaa HTML-kokonaisuutta, jonka selain suorittaa sovellusta tai sivua käynnistettäessä. DOM kuvaa dokumentin hierarkiaa puuttumatta itse sisältöön. DOM-standardoitu HTML-kokonaisuus muodostaa puumaisen hierarkian, joka haaroittuu sovelluksen tai sivun solmuiksi eli nodeiksi (Kaavio 1). Tästä syystä DOMia kutsutaan DOM-puuksi. DOM on siis suositeltavaa ajatella puurakenteena, jonka juurielementti on HTML, ja se haaroittuu solmuiksi eli nodeiksi. (Peltomäki 2007, 252.)

Jokainen DOM-puun node on itsenäinen olio, joten niillä on ominaisuuksia ja metodeita. Ominaisuudet ovat esimerkiksi nimiä ja metodit jotain tekemistä, ja niitä voidaan käsitellä esimerkiksi JavaScript-koodilla. DOMin suorittaminen selaimen avautuessa tapahtuu ylhäältä alas, mikä tulee ottaa huomioon esimerkiksi sijoittaessa JavaScript-koodia osaksi sovellusta, ettei JavaScriptillä vahingossa yritä käsitellä elementtejä, joita ei ole vielä olemassa.



Kaavio 1: DOM-puu

4.3 JavaScript

JavaScript on Netscapen vuonna 1995 kehittämä ohjelmointikieli www-dokumenttien manipulointiin. Ensimmäinen versio JavaScriptistä, 1.0, kehitettiin alunperin nimellä LiveScript, mutta sen nimi vaihdettiin JavaScriptiksi markkinointisyistä. Ensimmäiset JavaSciprtiä tukevat selaimet olivat Netscape Navigator ja Internet Explorer. JavaScript versio 1.5 oli ensimmäinen versio, joka sisälsi tuen DOM 1-oliomallille. Se on nykypäivänä

yleisesti web-kehityksessä mukana oleva ohjelmointikieli. Sitä voidaan käyttää esimerkiksi lomakkeiden validointiin tai tehosteisiin. JavaScript on tulkattava, dynaamisesti tyypitetty skriptikieli, joka mahdollistaa myös oliomaisen käyttötavan. Sitä kutsutaan usein hybridikieleksi sekä skripti- että oliomaisesta käyttötavasta johtuen. Selainten selainmoottorit voivat erota hieman toisistaan, mistä johtuen JavaScriptin tulkinta voi vaihdella hieman selainten välillä. (Peltomäki 2007, 15.)

JavaScriptiä on mahdollisuus käyttää ulkoisena tai sisäisenä määrittelynä. Ulkoinen skriptimäärittely tarkoittaa, että koodi sijaitsee JavaScript-tiedostossa, jota kutsutaan HTML-koodissa. Tällöin koko JavaScript-tiedosto sisältää pelkästään JavaScript-koodia. Kun JavaScript-tiedosto on liitetty script-tagien avulla osaksi HTML-koodia, tulkitsee selain sitä, kuin se olisi kirjoitettu suoraan HTML-koodin sekaan. Sisäisellä skriptimäärittelyllä tarkoitetaan, että HTML-koodi sisältää script-tagin, joka sisältää JavaScript-koodin. Tällöin JavaScript-koodi kirjoitetaan siis suoraan HTML-koodin sekaan. Molemmat määrittelyt toimivat, mutta yleisesti on suositeltavaa kirjoittaa JavaScript-koodit erilliseen JavaScript-tiedostoon ymmärrettävyyden ja ylläpidon helpottamiseksi sekä siksi, että erillinen JavaScript-tiedosto mahdollistaa saman skriptin käyttämisen useilla eri sivuilla. (Peltomäki 2007, 15.)

4.4 Asynchronous JavaScript And XML

Asynchronous JavaScript And XML (Ajax) tarkoittaa tekniikkaa, jolla on mahdollista liikuttaa dataa selaimen ja palvelimen välillä asynkronisesti eli ilman, että tarvitsee ladata koko sivua uudelleen. Tällöin web-sovelluksesta saadaan huomattavasti käyttäjäystävällisempi ja monipuolisempi. Ajax-tekniikka perustuu asynkronisesti lähetettäviin HTTP-pyyntöihin, joita voidaan lähettää missä tilanteessa tahansa ilman, että muut sivut reagoi siihen mitenkään. HTTP-pyyntö siis lähetetään niin sanotusti selaimen taustalla, jonka jälkeen palautuva data voidaan siirtää haluttuun muuttujaan tai esimerkiksi HTML-elementtiin. (Ferguson & Heilmann 2013, 248.)

Ajax-kutsu eli HTTP-pyyntö on tapahtuma, jossa data liikkuu selaimen ja palvelimen välillä asynkronisesti. Sen toiminta perustuu XMLHTTPRequest-objektin käyttöön (Ferguson & Heilmann 2013, 249). XMLHTTPRequest-objektilla on ominaisuuksia kuten esimerkiksi status tai response, joiden avulla voidaan tarkastella onnistuiko kutsu tai että mitä dataa se palauttaa. XMLHTTPRequest-objekti on alun perin Microsoftin kehittämä, mutta muut selainvalmistajat ovat tehneet siitä puhtaasti JavaScript-olion. Siksi Microsoftin Internet Explorerille tulee toteuttaa oma logiikkansa XMLHTTPSRequest-objektin käyttöön. Osa kirjastoista, kuten esimerkiksi JQuery, osaa tulkita selainten väliset erot XMLHTTPRequest-objektia käsitellessä ja näitä kirjastoja on hyvä käyttää, mikäli Ajax on suuressa roolissa

projektissa. Näin selainten välisiä eroja ei tarvitse ottaa niin vahvasti huomioon Ajax-kutsuja toteutettaessa. (Ferguson & Heilmann 2013, 254.)

Ajax-kutsu on kaksivaiheinen tapahtuma; ensin se lähettää kutsun palvelimelle, ja sitten, jos palvelin vastaa takaisin, Ajax ottaa vastauksen kiinni. Kutsussa on mahdollista siirtää dataa sekä selaimelta että palvelimelle päin, tai vaikka kumpaankin suuntaan samalla HTTP-pyynnöllä. Sillä on kaksi tapaa siirtää dataa selaimen ja palvelimen välillä: Get ja Post. Molemmat kutsut toimivat pääpiirteiltään samalla tavalla, eli lähettävät ensin kutsun palvelimelle ja palauttavat sitten jotain. Eroina näissä kutsutyypeissä on, että Get siirtää dataa otsikon mukana, kun taas Post siirtää sitä kutsun body-osiossa. Yleensä periaatteena on, että selaimelle haettavat tiedot tuodaan Get-tyyppisenä ja palvelimelle vietävät tiedot Post-tyyppisenä Ajax-kutsuna. Tämä ei kuitenkaan ole välttämätöntä, sillä molemmat toimivat molemmissa tapauksissa. (Holdener 2008, 525.)

HTTP-pyynnön vastaus annetaan selaimelle numerokoodilla. Esimerkiksi 200 tarkoittaa onnistunutta vastausta ja 500 tarkoittaa, että HTTP-pyyntö ei löytänyt etsimäänsä määränpäätä palvelinpuolelta. Ajax-kutsulle voidaan myös määrittää erilaisia toimintaohjeita riippuen, onnistuuko Ajax-kutsu vai ei. Sille voidaan esimerkiksi määrittää, että jos se onnistuu, ladataan palautuva data muuttujaan ja jos se epäonnistuu, näytetään käyttäjälle virheilmoitus. Ajax-kutsulle on myös mahdollista määrittää toimintoja sille ajalle, kun se odottaa HTTP-pyynnön vastausta. Voidaan esimerkiksi näyttää hiiren kursorin tilalla tiimalasi tai vaikka estää joidenkin kenttien muokkaus sillä aikaa, kun dataa haetaan. Monesti Ajax-kutsuilla liikutettavat tiedot ovat JSON-muotoista dataa. (Holdener 2008, 68-91.)

JSON (JavaScript Object Notation) on datan merkkijonona oleva muoto. Se on merkkijono, jossa jäsenet ja tiedot on eroteltu pilkuin ja erilaisin sulkein. JSON-muotoisesta datasta tekee nykypäivänä hyvinkin suositun se, että se on kevyt tapa varastoida isoja määriä dataa. Lisäksi JSON-muotoinen data voi sisältää niin taulukoita, kuin listojakin sekä näitä sekaisin ja sitä on helppo käsitellä useilla eri ohjelmointikielillä, kuten esimerkiksi JavaScriptillä ja C#:lla (Crockford 2009).

4.5 Model-View-ViewModel

Web-sovelluskehityksen saralla on luotu useita kirjastoja ja arkkitehtuurimalleja, joilla on pyritty tekemään käyttöliittymäohjelmoinnista helpompaa ja ylläpidettävämpää. Varsinkin JavaScript-maailmassa kehittäjillä on monesti apuna erilaisia kirjastoja, kuten esimerkiksi JQuery, joiden avulla omasta koodin syntaksista saadaan yksinkertaisempaa ja helpommin luettavaa. KnockoutJS on tapa toteuttaa Model-View-ViewModel -malli osaksi käyttöliittymää yksinkertaisin ja helppoluettavin syntaksein. MVVM (Model-View-ViewModel) on Martin

Fowlerin (Microsoft) kehittämä web-sovelluskehitykseen tarkoitettu malli. MVVM-malli on saanut vaikutteita MVC-mallista, jossa sovellus jaetaan Model, View ja Controller-osiin. MVVM-patternin tarkoituksena on jakaa sovelluksen Model, View ja ViewModel-osiin (KnockoutJS.)

Model, View ja ViewModel toimivat ikään kuin erillisinä osina sovelluksen web-rajapinnassa, jotka ovat sidoksissa toisiinsa putkimaisesti (Kaavio 2). View-osiolla tarkoitetaan sovelluksen web-rajapinnan näkyvää osaa eli HTML-osiota ja sen elementtejä, kuten tekstikenttiä ja taulukoita. Model on MVVM-mallissa se osa, joka käsittelee dataa ja vastaa selaimen bineslogiikasta. Tämän opinnäytetyössä Model voisi sisältää AJAX-kutsun, joka kutsuu dataa sovellukselta. ViewModel sen sijaan on rajapinta Viewin ja Modelin välissä. Se toimii eräänlaisena kanava, joka on sidottu kiinni Viewiin ja jolle voidaan tuoda dataa Modelista. ViewModel on se, johon esityslogiikka on sijoitettu. Käytännössä tämä tarkoittaa sitä, että kun Viewin ja ViewModelin välille on rakennettu yhteys ja kun Modelista tuodaan dataa ViewModeliin, osaa MVVM-pattern automaattisesti näyttää sen Viewissä. ViewModel voi sisältää muuttujia, listoja ja funktioita ja niihin palataan myöhemmin tässä opinnäytetyössä. (Farrar 2015, 7.)

Kaavio 2: MVVM-malli

4.6 JavaScript-kirjastot

JavaScript-kirjastoilla tarkoitetaan ulkopuolisen ohjelmoijan tekemää koodia, joka voidaan ottaa käyttöön osaksi omaa projektia. JavaScript-kirjastot auttavat kehittää nopeuttamaan kehitysprosessia sekä tarjoavat kasan yksinkertaisia toiminnallisuuksia, joita voi hyödyntää kehitystyössä. Lisäksi tunnetuilla kirjastoilla on taustalla iso yhteisö kokeneita kehittäjiä kehittämässä toimivaa koodia (Raasch 2013, 29.). JavaScript-kirjastoja ovat esimerkiksi JQuery, jolla saadaan yleistä ohjelmointia helpotettua, Raphael, jolla saadaan helpotettua grafiikan piirtämistä sekä KnockoutJS, jolla saadaan MVVM-malli mukaan ohjelmointiin. JQuery on suosituin JavaScript-kirjasto.

JavaScript-kirjastot saadaan sovellukseen mukaan liittämällä ne osaksi HTML-koodia. Kirjastot voivat sijaita kehittäjän omalla paikallisella levyllä, jolloin HTML-koodiin määritetään vain kirjaston polku. Tämän lisäksi kirjastoja voi käyttää myös netin kautta siten, että kirjasto itsessään sijaitsee jossain muulla palvelimella, kuin kehittäjän omalla koneella. Kirjasto voi sijaita esimerkiksi Googlen palvelimella, kuten esimerkiksi JQuery. Tällöin kirjaston voi ottaa

käyttöön osaksi omaa koodia liittämällä HTML-koodiin www-osoite, josta kirjasto löytyy, eikä kirjastoa tarvitse ladata omalle palvelinkoneelle. Vaikka jotkut käyttävätkin kirjastoja esimerkiksi Googlen palvelimelta, yleensä sivustot käyttävät kirjastoja omalta palvelinkoneelta käsin. (Raasch 2013, 31.)

Tässä opinnäytetyössä käytetään kahta kirjastoa perinteisen JavaScript-koodin apuna. KnockoutJS-kirjasto tuo uuden mallin sovelluksen käyttöliittymäkehitykseen ja JQuery helpottaa muita toiminnallisuuksia, kuten esimerkiksi Ajax-kutsujen toteutusta. KnockoutJS on Steve Sandersonin tekemä JavaScript-kirjasto, joka julkaistiin kesäkuussa vuonna 2010. Se on MIT-lisenssin alle kuuluva avoimen lähdekoodin kirjasto, joka mahdollistaa MVVM-mallin liittämisen osaksi sovelluksen web-rajapintaa. Viimeisin päivitys Knockoutista on versio 3.2.0, julkaistiin elokuussa 2014. KnockoutJS on puhtaasti perinteisellä JavaScriptillä toteutettu kirjasto, joten sen käyttöönotto ei vaadi muita kirjastoja toimiakseen (KnockoutJS).

JQueryn on alunperin kehittänyt John Resigin ja ensimmäinen virallinen julkaisu JQuerysta tehtiin elokuussa 2005. Se on myös MIT-lisenssin alle kuuluva avoimen lähdekoodin kirjasto. JQueryn tehtävänä on helpottaa kehittäjän työtä JavaScriptin parissa. Se tuo käyttäjälle yksinkertaisemman JavaScript-syntaksin sekä paljon muuta, kuten esimerkiksi sen, että se yhdistää kaikki selaimet toimimaan samalla koodilla, eikä kehittäjän tarvitse näin ollen kirjoittaa erillisiä koodeja eri selaimille (Marakana jQuery Training 2011, 1-3)

Kirjastoille on määritetty lisenssit, jotka rajaavat kirjastojen käyttöä. Yleensä kaikki yleisesti käytössä olevat suositut JavaScript-kirjastot on lisensoitu jonkin lisenssin alle. Lisenssit voivat esimerkiksi määrittää, että kirjastoa saa käyttää omaan käyttöön mutta ei kaupalliseen tarkoitukseen. Kuitenkin suosituimmat kirjastot kuuluvat yleensä sellaisten lisenssien alle, jotka sallivat myös kaupallisen käytön. Esimerkiksi JQuery ja KnockoutJS kuuluvat MIT-lisenssit alle, joka sallii niin kaupallisen käytön kuin kirjaston muokkaamisenkin, kunhan kirjaston koodissa sijaitsevat tiedot esimerkiksi tekijöistä pidetään koskemattomina. Kirjastojen lisenssit on merkitty sekä kirjaston koodin sekaan, että sen dokumentaatioon, kuten esimerkiksi kirjaston nettisivuille ja mikäli kirjasto löytyy esimerkiksi GitHub-palvelusta, löytyy merkintä lisenssistä usein myös sieltä (jQuery; KnockoutJS).

5 Tutkimusmenetelmät

Tutkimusmenetelmillä tarkoitetaan tapaa, jolla jotain kohdetta tutkitaan. Tutkimus voidaan usein määritellä joko kvalitatiiviseksi eli laadulliseksi tai kvantitatiiviseksi eli määrälliseksi tutkimukseksi. Jotta tutkimus määritetään jompaan kumpaan kategoriaan, on tiedettävä mitä ja miten tutkitaan ja millaisia tutkimustuloksia halutaan saada. Monesti kvalitatiivisen ja

kvantitatiivisen tutkimuksen raja on mielletty hyvinkin jyrkäksi, ikään kuin kyseessä olisi kaksi toisensa täysin poissulkevaa tutkimustapaa. Kuitenkin näitä tutkimustapoja on mahdollista toteuttaa tarvittaessa myös yhdessä, mikäli halutut tutkimustulokset sen vaativat. (Tuomivaara, 2005, 29.)

Laadullisessa tutkimuksessa käytettävät henkilöt ovat usein valittu jonkin ominaisuuden tai muun tietoisuuden perusteella, eikä sattumanvaraisesti. Laadullinen tutkimus, verrattuna määrälliseen tutkimukseen, tutkii enemmän ilmiotä tiedostavien ihmisten näkökulmasta. Laadullinen tutkimus tutkii tutkimuksen kohteen laatua, ominaisuuksia ja merkitystä ottaen huomioon esimerkiksi kohteen ympäristön ja taustan. Sen sijaan määrällinen tutkimus perustuu matemaattisiin ilmiöihin ja numeerisiin ja laskennallisiin tutkimustuloksiin. (Stephen, MaryLynn & Frances 2012, 3.)

5.1 Aineiston keruu

Aineiston keruussa on aluksi huomioitava aineiston valikointi eli tutkimuksesta karsitaan pois tutkimuksen kannalta epäoleellinen aineisto tutkimusongelmien avulla. Tutkimuksen aineisto esitetään järjestetymmässä muodossa, jotta lukijan on helpompi löytää tutkimuksen tarkoitus. Lisäksi tiedon merkitys ja johtopäätökset selitetään esitetyn aineiston perusteella. (Taanila 2007.)

Tässä työssä tutkitaan sitä, miten saadaan toteutettua web-käyttöliittymässä yleisesti olevia elementtejä käyttäen MVVM-mallia ja KnockoutJS-JavaScript-kirjastoa. Tämä opinnäytetyö ei suoranaisesti ole kvalitatiivinen eikä kvantitatiivinen tutkimus, mutta kallistuu enemmän kvalitatiivisen tutkimuksen puolelle, sillä tutkimus perustuu omiin havaintoihini, eikä tutkimuksessa ole käytetty muita tutkijoita. Tutkimus on toiminnallinen tutkimus ja päällisin kysymys on, miten saadaan toteutettua käyttöliittymä käyttäen vaadittua arkkitehtuuria ja tekniikkaa, ja mitä hyviä ja huonoja puolia tämä tekniikka tuo mukanaan.

Aineiston keruu tapahtuu projektin edetessä, jolloin päällisimpinä aineistoina on omat huomiot. Kerättävä aineisto on itse toteutettua ohjelmointia tai muuta ohjelmointiin liittyvää. Mukaan kerätään sellaista aineistoa, joka on yleisesti validia, ja jota muiden kehittäjien on mahdollista hyödyntää. Tällä tarkoitetaan sitä, että projektissa olevia spesifisiä osia karsitaan ja opinnäytetyöhön tuodaan yksinkertaistettua koodia ja toteutuksia. Kerätty aineisto järjestetään siten, että alussa kerron yksinkertaisia asioita tutkittavasta kohteesta, jonka jälkeen siirrytään askel kerrallaan haastavimpiin osiin. Aineiston edetessä selitän ilmiötä kuvien avulla, jotta sitä olisi helpompi ymmärtää. Lopussa kerään yhteenvedon tutkitusta kohteesta ja sen tuloksista sekä kerron, millaisia hyötyjä ja haasteita tutkimuksen edetessä on huomattu.

5.2 Validiteetti ja reliabiliteetti

Validiteettia tutkiessa tulee ensimmäisenä miettiä, millainen tutkimus on validi. Kuinka hyvin tutkimuksessa käytetyt menetelmät vastaavat sitä, mitä halutaan tutkia. Onko tutkimus tehty hyvin ja perusteellisesti ja ovat tulokset päteviä? Miten tutkimus on tehty, millä menetelmillä ja mikä siinä on tavoitteena? Mitä tutkimuksessa haetaan ja millaisia tuloksia halutaan saada? Validiteetti keskittyy pääpiirteiltään siihen, millainen tutkimus on pätevä tutkimisen kannalta. (Hiltunen 2009.)

Validiteetti on hyvä sillon, kun tutkimuksessa on kysytty oikeat kysymykset oikealta kohderyhmältä. Hyvässä validiteetissa tutkijan on tiedettävä, millaisia tuloksia tutkimuksessa halutaan saada ja näin osata suunnitella tutkimus sen mukaan. Tutkijan on ymmärrettävä tutkimuksen tavoitteet ja millaisilla menetelmillä tavoitteet on mahdollista saavuttaa. (Hiltunen 2009).

Reliabiliteetista puhuttaessa puhutaan siitä, kuinka toistettava tutkimus on ja onko tutkimuksessa mahdollista saada useilla tutkimuskerroilla samanlaisia tuloksia, vai eroaako tutkimustulokset tutkimuskertojen välillä eli onko tutkimus reliaabeli eli toistettava vai ei. Jos esimerkiksi tutkimus toteutetaan useita kertoja peräkkäin, reliabiliteetilla mitataan, kuinka samanlaisia tuloksia tutkijat saavat tulokseksi. Reliabiliteetin tutkimiseen liittyy myös stabiliteetti ja konsistenssi. Näillä molemmilla mitataan pääpiirteiltään sitä, että jos useat tutkijat toteuttavat tutkimuksen useita kertoja, kuinka erilaisia tuloksia tutkijat saavat (Hiltunen 2009).

6 MVVM-mallin toteutus KnockoutJS-kirjastoa käyttäen

Tässä luvussa käydään läpi virallisen projektin toteutus. Tarkoituksena on näyttää koodiesimerkkejä sekä selittää MVVM-mallin toimintaa ja Knockoutin syntaksia. Käyn läpi muun muassa Ajax-kutsun sekä datan kulun Modelista ViewModelin kautaa Viewiin. Lopussa kerron myös hieman KnockoutJS-kirjaston mahdollistamista handlereista eli käsittelijöistä sekä käytettävistä selaimista.

Knockoutia käyttöönottaessa on hyvä ymmärtää sen perusajatus, eli miten se toimii ja miten se kannattaa sijoittaa JavaScript-koodin joukkoon. Vaikka käytössä olisi tämä kirjasto, ei se kuitenkaan estä kehittäjää käyttämästä tavallista JavaScriptiä tai muita kirjastoja. Tavallisen JavaScriptin tai muiden kirjastojen käyttäminen Knockoutin kanssa on oikeastaan lähes

pakollista, sillä Knockout ei itsessään tue esimerkiksi Ajaxia. Esimerkiksi Ajaxia käytettäessä Knockout ja MVVM tulee kuvioon vasta datan latauksen jälkeen.

Näytän esimerkkejä Knockoutista ja MVVM-mallista yhdessä jQuery-kirjaston kanssa, jossa Ajax-kutsu toteutetaan jQuerya käyttäen, mutta data tuodaan näkymään MVVM:n kautta. Syynä jQueryn käyttöön on se, että se selkeyttää ja lyhentää JavaScriptin syntaksia, jolloin ei tarvitse keskittyä niin paljoa muuhun, kuin Knockoutin selittämiseen. jQuerya käytettäessä kehittäjän tulee myös ottaa huomioon, että sekin on kirjasto ja se tulee liittää osaksi projektia samalla tavalla, kuin KnockoutJS-kirjastokin on liitetty. Näissä esimerkeissä siis käytetään sekä KnockoutJS- että jQuery-kirjastoa, mutta pääosin tutkitaan Knockoutin käyttöä ja jQuerya vain sivutaan hieman.

MVVM-malli on lopulta todella helppo ymmärtää, mutta sen hyödyt voi olla aluksi vaikea nähdä. MVVM koostuu siis kolmesta osasta: Modelista, Viewistä ja ViewModelista. Kerron, kuinka saadaan rakennettua yksinkertainen MVVM-malli JavaScriptiin ja kuinka sitä hyödynnetään tehokkaasti kun liitetään dataa kenttiin.

Jotta MVVM-malli saadaan lisättyä käyttöliittymäohjelmoinnin avuksi, on KnockoutJS-kirjasto liitettävä osaksi projektia. Knockoutin liittäminen projektiin tapahtuu samalla tavalla kuin muidenkin JavaScipt-kirjastojen eli liittämällä script-tagi HTML-koodiin, johon määritetään kirjaston polku. Kirjasto suositellaan ladattavan omalle levylle muun projektimateriaalin joukkoon, vaikka sitä on myös mahdollista käyttää internetin kautta liittämällä script-tagiin kirjaston www-osoite. Omalla levyllä käytettäessä sitä on mahdollista muokata haluamillaan tavoilla, mikäli se nähdään tarpeelliseksi. Lisäksi omalla levyllä olevaa kirjastoa on myös mahdollista käyttää ilman verkkoyhteyttä, mikäli se jostain syystä ei toimi tai on tukkoinen.

6.1 Ajax eli HTTP-pyyntö

Ajax on sovelluksen web-rajapinnassa toimiva HTTP-pyyntöjä lähettävä tekniikka, joka joko pyytää tai lähettää dataa frontendiltä backendille. MVVM-mallin näkökulmasta ajateltuna Ajax-kutsu sijaitsee MVVM:n Model- eli malliosiossa. Jotta Ajax-kutsu eli HTTP-pyyntö saadaan lähetettyä, on sille ensin määritettävä asetuksia, joiden perusteella se tietää esimerkiksi mihin osaan sovelluksen backendiä se suorittaa haun ja halutaanko lähettää Ajax-kutsun mukana jotain parametreja. Lisäksi Ajax-kutsuun tulee määrittää, mitä halutaan tehdä kun kutsun vastaus saapuu takaisin selaimelle. Ajax-kutsu voidaan rakentaa usealla eri tavalla. Kuvassa 1 on lyhyt esimerkki yksinkertaisesta Ajax-kutsusta, joka on toteutettu käyttäen jQuery-kirjaston syntaksia.

Kuvassa 1 näkyy hyvin yksinkertainen tapa rakentaa Ajax-kutsu. Se koostuu kahdesta osasta: HTTP-pyynnön lähettämisestä ja sen vastaanottamisesta. Kuvan esimerkissä ”var AjaxKutsu = $.ajax” tarkoittaa Ajax-kutsua, joka lähetetään ja AjaxKutsu.done on Ajax-kutsun vastauksen kiinni ottaminen, kun se palaa takaisin selaimelle. Kuten esimerkistä huomaa, sille on määritetty url-osoite. Se osoittaa sovelluskerroksen johonkin kohtaan tai tiedostoon, josta dataa lähdetään kyselemään. Vaikka backendissä olisi jotain toiminnallisuutta palautettavaan dataan liittyen, käyttöliittymästä katsottuna on mahdotonta nähdä, mitä url-osoitteen takana todellisuudessa tapahtuu, eikä siitä syystä perehdytä siihen tämän enempää.

Url-osoitteen lisäksi Ajax-kutsussa on määritetty datatyypiksi JSON, jotta selain osaa käsitellä palautuvaa dataa, mikäli se on JSON-muotoista. JSON-muotoinen data on varsin yleistä varsinkin sellaisissa tapauksissa, joissa dataa liikkuu paljon. JSON on kevyttä merkkijono-muotoista dataa, ja sitä on helppo ja nopea siirtää frontendin ja backendin välillä. Lisäksi JSON-muotoista dataa on helppoa ”parsia” eli pilkkoa helpommin käsiteltävään muotoon latausten jälkeen. Urlin ja datatyypin lisäksi Ajaxissa voi antaa määrityksen ”data”, joka ei kuitenkaan ole pakollinen kutsun toiminnan kannalta. Data-osioon määritetään data, joka lähetetään frontendiltä backendille päin. Jos esimerkiksi halutaan antaa jotain parametritietoja frontendiltä backendille, tulee lähetettävä data liittää tähän osioon. Näiden lisäksi Ajax-kutsussa on vielä ”type”-kohta, jossa määritetään, minkä tyyppinen kutsu lähetetään. Kuten aiemmin mainitsin, kutsuja on GET- ja POST-tyyppisiä ja molempia kannattaa hyödyntää riippuen tarkoituksesta.

Ajax-kutsun done-osio on MVVM:n kannalta se keskeisempi osa. Ylempänä olevassa esimerkissä done-osioon on liitetty console.log()-funktio, joka kirjoittaa selaimen konsoliin. Tässä esimerkissä Ajax-kutsu odottaa, että pyyntö vastaa ja kirjoittaa vastauksena tulevan response-muuttujan JSON-datan selaimen konsoliin.

```
var AjaxKutsu = $.ajax({
  dataType: JSON,
  type: GET
  url: "/tuote/tuoteTuotteet",
  data: $("tuoteFormi").searialize()
});
AjaxKutsu.done(function (status, response) {
    console.log(response);
});
```

Kuva 1: Ajax-kutsu

6.2 ViewModel ja muuttujat

Ajax-kutsu sijaitsee siis MVVM:n Model-osassa, jossa se tuo dataa sovelluksen web-rajapinnalle. Jotta se saadaan näytettyä selaimessa, on sille rakennettava niin sanottu reitti ViewModelin kautta. ViewModel toimii ikään kuin väylänä Modelin eli tässä tapauksessa Ajax-kutsun ja Viewin eli HTML-koodin välillä. ViewModel on MVVM-mallin keskeisin, johon liittyy muuttujia, listoja ja funktioita, jotka sidotaan kiinni HTML-koodin elementteihin.

Hyvin yksinkertainen esimerkki ViewModelista voisi sisältää muuttujan eli observablen, joka sidotaan kiinni HTML-koodissa olevaan tekstikenttään. Tällöin, jos muuttujalle annetaan arvo, osaa HTML-koodissa oleva tekstikenttä automaattisesti näyttää sen. Samalla periaatteella toimii esimerkiksi taulukkomuotoinen data. Taulukko määritetään ViewModeliin listamuuttujaksi, joka sidotaan kiinni HTML-koodissa olevaan taulu-elementtiin, jolloin taulu-elementti osaa automaattisesti näyttää kyseisen listamuuttujan arvot.

Knockout-kirjastossa käytetään muuttujille määritteitä observable eli merkkijono ja observableArray eli lista tai taulukko. Esimerkissä (Kuva 2) on määritetty ViewModel ja sen sisälle kaksi muuttujaa, observable ja observableArray, joista ”teksti” on tavallinen tekstiä sisältävä muuttuja ja ”taulukko” listamuuttuja. Kuten nimistäkin voi päätellä, tekstimuuttujaa tulee käyttää yksittäisten muuttujien näyttämiseen ja listamuuttujaa listojen ja taulukoiden näyttämiseen. Näitä voi käyttää myös ristiin, mutta esimerkiksi tekstimuuttujaan liitettyä taulukkodataa ei ole mahdollista esimerkiksi järjestää, sillä ViewModel tulkitsee sen tällöin vain pitkänä merkkijonona eikä niinkään listana.

```
var ViewModel = function () {
    teksti: new ko.obserable(“”),
    taulukko: new ko.obserableArray(“”)
}
ko.applybindings(ViewModel)
```

Kuva 2: ViewModel

Tapoja liittää dataa ViewModelissa sijaitsevaan muuttujaan on useita. Voidaan esimerkiksi liittää koko saapuva data ViewModelin muuttujaan. Palautuva data tulee tässä tapauksessa JSON-muotoisena ja se tulee ensin muuttaa luettavaan muotoon. JSON on pitkä merkkijono, jota on hankala sellaisenaan käsitellä. JavaScript sisältää kuitenkin JSON.Parse-metodin, jolla on helppo muuttaa JSON-muotoinen data listaksi tai taulukoksi. Datan liittäminen muuttujaan tapahtuu esimerkiksi määrittämällä Ajax-kutsun done-osioon ”var parsittuData = JSON.Parse(response)” (Kuva 3). Tällöin muuttujalle *parsittuData* annetaan arvoksi koko

palautuva data, joka samalla muutetaan taulukko- tai listamuotoiseksi dataksi. Tämän jälkeen koko parsittuData-muuttujan sisältö liitetään ViewModelissa olevaan *taulukko*-muuttujaan määrityksellä *ViewModel.taulukko(parsittuData)* (Kuva 3). Näin saadaan liitettyä Ajax-kutsun palauttama data.

Data voidaan liittää ViewModelin muuttujaan myös määrittämällä *ViewModel.teksti(parsittuData[0].teksti)*, jossa sen sijaan, että liitettäisiin koko parsittuData-muuttujan sisältö, liitetään vain muuttujan ensimmäisen jäsenen *teksti*-kentän arvo (Kuva 3). Tällä tavalla saadaan poimittua taulukosta tai listasta jokin haluttu arvo.

```
var AjaxKutsu = $.ajax({
  dataType: JSON,
  type: GET
  url: "/tuote/tuoteTuotteet",
  data: $("tuoteFormi").searialize()
});
AjaxKutsu.done(function (status, response) {
    var parsittuData = JSON.parse(response);
    // JOS HALUTAAN LIITTÄÄ KOKO DATA taulukko-MUUTTUJAAN
    ViewModel.taulukko(parsittuData)
    // JOS HALUTAAN LIITTÄÄ VAIN OSA DATASTA teskti-MUUTTUJAAN
    ViewModel.teksti(parsittuData[0].teksti)
});
```

Kuva 3: Ajax-kutsu ja tiedon liittäminen ViewModeliin

6.3 Muuttujan liittäminen HTML-elementtiin

Kun Ajax-kutsussa saapuva data on liitetty ViewModelissa sijaitsevaan muuttujaan, on vielä määritettävä muuttujan ja View-osiossa olevan HTML-elementin yhteys, jotta ViewModel osaa liittää halutun muuttujan arvon oikeaan HTML-elementtiin näkymässä. Muuttujan ja HTML-elementin yhteys määritetään kutsumalla HTML-koodissa ViewModelin muuttujaa. ViewModelissa voidaan halutessa rajoittaa ViewModelin rajaus eli voidaan määrittää, minkä HTML-divin sisällä ViewModelia voidaan käyttää. Mikäli ViewModelin rajaus on määritetty, on huomioitava, että HTML-koodissa Knockout-muuttujaa on kutsuttava rajoituksen sisäpuolella.

Knockout-kirjasto käyttää määritettä data-bind, jolla määritetään, mikä muuttuja halutaan sitoa tähän elementtiin. Knockoutin data-bind-määritteen sisältö riippuu siitä, minkälaiseen elementtiin dataa ollaan liittämässä. Esimerkiksi, jos dataa liitetään tekstinsyöttökenttään, määritetään ”data-bind=”value: teksti” *(Kuva 4)*. Tekstinsyöttökenttään on tällöin sidottu muuttuja ”teksti”, jonka arvo annetaan tämän kentän arvoksi eli valueksi. Jos taas dataa

halutaan liittää esimerkiksi span- eli tekstielementtiin, sidotaan muuttuja sen tekstiksi määrittämällä ”data-bind=”text: teksti” (Kuva 5). Tällöin tämäkin elementti osaa näyttää teksti-muuttujan arvon. Knockout-käsitteet, kuten value ja text, perustuvat perinteisen JavaScriptin syntaksiin. Knockoutissa käsitellään elementtejä samoin termein, kuin JavaScriptissäkin.

```
<input type=”text” data-bind=”value: teksti” />
```

Kuva 4: Tiedon näyttäminen tekstikenttäelementissä

```
<span data-bind=”text: teksti” ></span>
```

Kuva 5: Tiedon näyttäminen tekstielementissä

6.4 Foreach-silmukka

Monet web-sovellukset ovat täynnä erilaisia listauksia, taulukoita, alasvetovalikoita ja muita vastaavia. Myös nämä kaikki on mahdollista toteuttaa Knockout-kirjaston syntaksia käyttäen. Yleensä perinteisellä JavaScriptillä tai jQuerylla voi olla varsin yksinkertaista toteuttaa yksinkertainen lista tietoa näkymään. Jos taas kyseessä iso taulukko, jossa sarakkeita halutaan mahdollisesti muokata, kuten esimerkiksi pyöristää tietyn sarakkeen lukuja, on Knockout-syntaksi huomattavasti yksinkertaisempi ja varsinkin ylläpidettävämpi, kuin perinteinen JavaScript ja jQuery. Erilaiset taulukot ja listat toteutetaan Knockout-syntaksilla foreach-silmukkaa käyttäen. Knockoutin foreach-silmukka toimii samalla tavalla kuin yleensä ohjelmointikielen foreach-silmukka eli se käy kaikki jäsenet läpi ja tekee mahdollisesti niille jotain. Knockoutissa foreach-silmukka määritetään HTML-koodin sekaan, jossa silmukan sisään määritetään halutut elementit, joita loopataan läpi ja elementteihin määritetään loopattavat arvot.

Foreach-silmukka sijoitetaan siis osaksi HTML-koodia. Foreach-silmukka sijoitetaan siihen elementtiin, jonka sisältöä halutaan loopata. Kuvan 6 tapauksessa rakennetaan alasvetovalikko käyttäen Knockoutin foreach-silmukkaa. Foreach-silmukka pyörähtää niin monta kertaa kuin taulukkoMuuttuja-nimisessä taulukkomuuttujassa on jäseniä ja se suorittaa jokaisella pyörähdyskerralla foreach-silmukan sisällä olevat toiminnot eli tässä tapauksessa generoi aina jokaisella pyörähdyskerralla uuden valinnan alasvetovalikkoon. Lisäksi taulukkoMuuttuja-muuttujan jokaisessa sarakkeessa on tekstinArvo ja tekstinNimi –nimiset sarakkeet, joiden arvot annetaan alasvetovalikon tietoihin. Option-tagissa määritetään

valintojen arvo eli value ja teksti eli text. Valuella tarkoitetaan valinnan arvoa ja tekstillä valinnassa näkyvää tekstiä.

```
<select data-bind="foreach: taulukkoMuuttuja">
    <option data-bind="value: tekstinArvo, text: tekstinNimi"></option>
</select>
```

Kuva 6: Tiedon näyttäminen valinta-elementissä

Alasvetovalikko on hyvin yksinkertainen esimerkki Knockoutin foreach-silmukasta. Foreach-silmukkaa voi käyttää myös moniin muihin näkymän osiin. Yksi yleinen kohde foreach-silmukalle on näkymässä näkyvä taulukko. Taulukon generointi tapahtuu samalla tavalla kuin alasvetovalikonkin sillä erolla, että taulukossa määritetään useampia kenttiä kuin pelkkä arvo ja teksti.

Kuvan 7 esimerkissä on taulukko, jossa kayttajat on taulukkomuuttuja. Kohdassa data-bind="foreach: kayttajat" taulukkomuuttuja loopataan läpi. Silmukka on sijoitettu tbody-kohtaan. Tämä tarkoittaa sitä, että tbodyn sisällä olevat elementit loopataan niin monta kertaan, kuin *kayttajat*-taulukkomuuttujassa on jäseniä ja jokaisella kierroksella generoidaan uusi rivi. Jokaiselle riville annetaan soluihin tiedot id, etuNimi, sukuNimi ja ika. Tällä tavalla saadaan generoitua taulukon body-osio Knockout-syntaksilla.

```
<table>
    <th>ID</th>
    <th>Etunimi</th>
    <th>Sukunimi</th>
    <th>Ikä</th>
    <tbody data-bind="foreach: kayttajat">
        <tr>
            <td><span data-bind="text: id"></span></td>
            <td><span data-bind="text: etuNimi"></span></td>
            <td><span data-bind="text: sukuNimi"></span></td>
            <td><span data-bind="text: ika"></span></td>
            </tr>
    </tbody>
</table>
```

Kuva 7: Tiedon näyttäminen taulukko-elementissä

Taulukoiden kanssa työskennellessä kehittäjän tulee ottaa huomioon muutamia asioita, jotka poikkeavat tavallisesta JavaScriptistä. Ensinnäkin taulukossa näkyvä data on todellisuudessa

ViewModelissa sijaitsevan taulukkomuuttujan dataa, joka vain näytetään näkymässä. Toisin sanoen näkymässä näkyvä taulukko on vain aktiivinen lista, joka näyttää ViewModelissa sijaitsevan muuttujan sisällön ja se muuttuu sen mukaan, kun muuttujan sisältöä muutetaan. Lisäksi taulukoiden kanssa tulee huomioida se, että taulukkoa ei kannata tyhjentää perinteistä JavaScriptiä käyttäen. Vaikka perinteisellä JavaScriptillä voikin tyhjentää näkymiä ja poistaa elementtejä, sillä ei voi tyhjentää ViewModelissa sijaitsevaa muuttujaa. Knockoutilla toteutettujen kenttien tyhjentäminen perinteistä JavaScriptiä tai muuta JavaScript-kirjaston syntaksia, kuten esimerkiksi JQuerya, käytettäessä voi aiheuttaa bugeja, koska vaikka näkymä tyhjennetään, ViewModelissa sijaitseva muuttuja sisältää vielä dataa. Yleisenä tapana on, että ViewModelin sisällä olevia muuttujia ja muuttujien sisältöä tulee käsitellä aina käyttäen Knockoutin omia tekniikoita.

Knockoutin tapa tuottaa taulukko web-rajapinnalle saapuvasta datasta eroaa perinteisestä JavaScriptistä siten, että Knockoutissa for/foreach-silmukat toteutetaan HTML-koodin seassa, kun taas perinteisellä JavaScriptin syntaksilla silmukat toteutetaan JavaScript-tiedostoissa. Lisäksi, jos datan taulukon generointi toteutetaan JavaScriptilla, tulee JavaScript-koodiin määrittää for-silmukat, joiden sisällä määritetään tuotettavat HTML-elementit. Lisäksi, jos tietoja halutaan muokata eli halutaan esimerkiksi määrittää jollekin sarakkeelle tietty määrä desimaaleja ja toiselle sarakkeelle jokin toinen määrä tai halutaan esimerkiksi päivämäärä näytettävän toisenlaisessa formaatissa kuin se on saapuvassa datassa, voi for-loopista tulla helposti hyvinkin pitkä, vaikeasti luettava ja hankalasti ylläpidettävä. Kuitenkin, jos pitää löytää asia, joka perinteisessä JavaScriptin silmukoissa on parempaa kuin Knockoutin silmukoissa, niin se on suorituskyky. Perinteinen JavaScript suorittaa isojen datajen looppauksen huomattavasti nopeammin, kuin Knockout. Lisäksi Knockoutin silmukoiden sijoitus sisäkkäin voi olla tietyissä tapauksissa hankalaa. Sisäkkäisiin silmukoihin ei kuitenkaan keskitytä tässä opinnäytetyössä.

6.5 Funktiot ja tapahtumat

Knockoutissa on mahdollista rakentaa erilaisia funktioita. Funktioiden sisälle on mahdollista rakentaa erilaisia toiminnallisuuksia, jotka liittyvät muuttujiin ja muuttujien dataan. Funktiot tulee aina sijoittaa ViewModeliin, jonka sisällä ne voivat kutsua muuttujia tai muita funktioita. Yleensä tapana on, että funktioita kutsutaan jonkin tapahtuman, kuten esimerkiksi klikkauksen, seurauksena. Yksinkertainen esimerkki knockoutin funktiosta voisi olla esimerkiksi näkymässä näkyvän taulukon tyhjentäminen. Kuten aiemmin mainitsin, jos Knockoutilla on liitetty näkymään dataa, on sen tyhjentäminenkin kannattavaa tehdä Knockoutia käyttäen.

Kuten kuvassa 8 voidaan nähdä, *removeTaulukko*-funktio on sijoitettu ViewModeliin. Funktiossa määrityksellä ”this” viitataan ViewModeliin ja *removeTaulukko* on funktion virallinen nimi. Funktion sisältö määritetään aaltosulkeiden sisään ja tämän sisällön funktio suorittaa, kun sitä kutsutaan. Funktion sisällöksi on tässä tapauksessa määritetty ainoastaan ”this.taulukko([])”. Tässäkin *this*-määritteellä viitataan ViewModeliin ja taulukko-määritteellä taulukkomuuttujaan. Tässä esimerkissä ViewModelin sisällä olevalle taulukko-muuttujan arvo ylikirjoitetaan tyhjällä merkkijonolla. Näin saadaan tyhjennettyä ViewModelissa sijaitseva muuttuja ja samalla näkymässä oleva taulukko. Funktio voi tehdä myös paljon muutakin kuin tyhjentää muuttujia. Se voi esimerkiksi sisältää erilaisia ehtolausekkeita tai silmukoita, joilla muokataan dataa halutunlaiseksi tai sillä voidaan esimerkiksi generoida uusia rivejä taulukoihin tai listoihin.

```
var ViewModel = function () {
    taulukko: new ko.obserableArray("")
}
this.removeTaulukko = function () {
    this.taulukko([])
}
ko.applybindings(ViewModel)
```

Kuva 8: Funktion sijoitus ViewModeliin

6.6 Ehtolauseet

Knockout on monipuolinen siitäkin syystä, että erilaiset silmukat ja ehtolauseet on mahdollista sijoittaa myös HTML-koodin sekaan. Jos selainkerrokselle tulee vaikkapa suuri määrä taulukkomuotoista dataa, joka halutaan jakaa kahteen osaan jonkin sarakkeen arvon perusteella, voi sen tehdä projektin HTML-koodin seassa. Jos taulukkomuotoinen data on esimerkiksi jonkin yrityksen asiakastietoja, voidaan Knockoutin avulla data jakaa vaikka iän mukaan.

Kuvan 9 esimerkissä käytetään kahden erillisen taulukon generointiin foreach-silmukkaa. Tässä esimerkissä sama data eli *asiakkaat*-taulukkomuuttuja loopataan molemmissa taulukoissa läpi, mutta ehto määrittää, mitä dataa näytetään missäkin taulukossa. Ehtolausekkeet on sijoitettu kommenttimerkkien sisään ikään kuin kommentiksi HTML-koodissa. Knockout osaa kuitenkin käsitellä kommenttimerkkien sisällä olevaa koodia, mikäli kommenttiin on kirjattu ”ko”-määritys. Lisäksi, kuten kuvan 9 esimerkissä, ”ko”-määritys tulee sekä avata että sulkea.

Ensimmäinen taulukko sisältää ehdon *<!-- ko if: ika >= 35 -->* , jossa määritetään, että rivit, joissa ikä on suurempi tai yhtä suuri kuin 35, näytetään näkymässä. Eli ViewModelissa oleva *asiakkaat*-muuttuja siis sisältää koko datan, mutta näkymässä näytetään vain osa muuttujan sisällöstä. Toinen taulukko sisältää ehdon *<!-- ko if: ika < 35 -->*, jossa määritetään, että näytetään vain ne rivit, joissa ikä on pienempi kun 35. Näin saadaan jaettua yksi suuri data kahteen osaan, jotka eivät sisällä duplikaattirivejä. Tällä samalla tavalla on myös tarvittaessa mahdollista jakaa dataa useampaan osaan.

```
<table>
    <tbody data-bind="foreach: asiakkaat">
        <tr>
        <!-- ko if: ika >= 35 -->
            <td><span data-bind="text: id"></span></td>
            <td><span data-bind="text: etuNimi"></span></td>
            <td><span data-bind="text: sukuNimi"></span></td>
            <td><span data-bind="text: ika"></span></td>
            </tr>
        <!-- /ko -->
    </tbody>
</table>
<table>
    <tbody data-bind="foreach: asiakkaat">
        <tr>
        <!-- ko if: ika < 35 -->
            <td><span data-bind="text: id"></span></td>
            <td><span data-bind="text: etuNimi"></span></td>
            <td><span data-bind="text: sukuNimi"></span></td>
            <td><span data-bind="text: ika"></span></td>
            </tr>
        <!-- /ko -->
    </tbody>
</table>
```

Kuva 9: Ehtolauseen sijoitus taulukko-elementtiin

6.7 Handlerit

Handlerit eli erilaiset käsittelijät ovat yksi Knockout-kirjaston hienouksista. Niiden rakentaminen on hieman monimutkaisempaa kuin tavallinen datan näyttäminen, mutta niiden rakentaminen helpottaa koodin ylläpitämistä huomattavasti. Handlerit ovat käytännössä funktioita, joiden ideana on suorittaa näytettävälle datalle automaattisesti muutoksia datan näyttämisen yhteydessä. Handleri voisi suorittaa esimerkiksi jonkin numeroarvon pyöristämisen tai datan järjestämisen. Oikeastaan myös HTML-koodissa data-bind-osioon

sijoitetut value ja text -määritykset ovat valmiita handlereita, joiden tehtävä on vain näyttää dataa.

Kuvan 10 esimerkissä luodaan handleri, jonka nimi on *pyöristettyLuku*. Ensin siinä määritetään kohta init, mikä tarkoittaa sitä, että sen sisältämät muutokset otetaan käyttöön heti ensimmäisestä datan näyttämisestä lähtien. Sitten handlerissa annetaan ensin arvo-muuttujalle tuleva arvo, joka tämän jälkeen pyöristetään kahdella desimaalilla. ValueAccessor siis tarkoittaa saapuvaa dataa. Pyöristyksen jälkeen arvo palautetaan näkymälle *ko.bindingHandlers.text.update* -kohdassa. Jotta näkymä osaa näyttää tämän handlerin muodostamat muutokset, on data näytettävä handlerin kautta.

```
ko.bindingHandlers.pyoristettyLuku = {
    init: function (element, valueAccessor, allBindingsAccessor) {
        var arvo = valueAccessor().replace(",",".");
        uusiArvo = Math.round(arvo * 100) / 100;
        uusiArvo = uusiArvo.replace(".", ",");
        ko.bindingHandlers.text.update(element, function () {
            return uusiArvo;
        });
    }
};
```

Kuva 10: pyoristettyLuku-handleri

Kuvassa 11 generoidaan taulukko käyttäen handleria. Taulukossa on n-määrä rivejä ja kaksi saraketta. Sarakkeista ensimmäinen, ekaLuku, näytetetään sellaisenaan kuin se tulee datan mukana. Toisen sarakkeen arvot näytetään *pyoristettyTeksti*-handlerin kautta, joka pyöristää luvut siten kuin handlerissa on määritetty eli tässä tapauksessa kahdella desimaalilla. Samaa handleria on mahdollista käyttää myös useissa eri kohdissa, mikä tekee siitä kätevän.

```
<table>
    <tbody data-bind="foreach: lukuja">
        <tr>
            <td><span data-bind="text: ekaLuku"></span></td>
            <td><span data-bind="pyoristettyLuku: tokaLuku"></span></td>
        </tr>
    </tbody>
</table>
```

Kuva 11: Handlerin käyttäminen teksti-elementissä

Taulukoiden järjestäminenkin tapahtuu myös Knockoutin handeleria käyttäen. Toisin kuin edellisessä luvussa, taulukoita järjestettäessä handleria ei kutsuta taulukon body-osioissa, vaan sen otsikoissa eli head-osiossa. Tarkoituksena on siis se, että otsikossa on klikkaus-tapahtuma, joka kutsuu funktiota, joka suorittaa taulukon järjestämisen tietyn sarakkeen mukaan. Handleri on geneerinen funktio, jota voidaan kutsua useissa eri paikoissa. Yhtenä merkittävänä erona handlerin ja aiemmin esitellyn Knockout-funktion välillä on se, että Knockout-funktio on sidottu ViewModeliin, kun taas Handleri on geneerisempi ja sitä voidaan käyttää useiden ViewModelien muuttujien järjestämiseen.

Kuten kuvan 12 esimerkistä voidaan nähdä, se alkaa osiolla ”ko.bindingsHandlers.jarjesta”, joka määrittää, että kyseessä on handleri. Tämän esimerkissä näkyy funktio *sorttaa*, joka suorittaa taulukon järjestämisen. Funktion *sorttaa* sisällä oleva muuttuja *ValueAccessor* tarkoittaa saapuvaa dataa josta poimitaan ominaisuudet ”prop” ja ”arr”. Nämä annetaan parametreiksi HTML-koodissa (Kuva 13). Tämän jälkeen funktiossa määritetään ehtolauseke, joka järjestää datan jompaankumpaan suuntaan riippuen siitä, onko se järjestetty jo. Lopussa oleva ”element.onclick”-osio tarkoittaa elementin klikkausta, joka suorittaa funktion *sorttaa*.

Kuvan 12 esimerkissä tulee huomioida se, että tämä on hyvin yksinkertainen esimerkki järjestyshandlerista ja tässä järjestystapahtuma toimii ja se järjestää yksinkertaista merkkijonodataa oikein. Järjestäminen tapahtuu siten, että se vertailee tietoja merkki kerrallaan ja järjestää sen mukaan. Myös numerot ja päivämäärät tämä tulkitsee merkkijonoina, joten tätä esimerkkiä tulee jatkojalostaa, mikäli sitä käytetään numeroiden tai päivämäärien järjestämiseen.

```
ko.bindingHandlers.jarjesta = {
    init: function (element, valueAccessor) {
        var jarjestys;

        var sorttaa = function () {
            var property = valueAccessor().prop;
            var array = valueAccessor().arr;

            jarjestys = !jarjestys;
            if (jarjestys) {
                array.sort(function (a, b) {
                    return a[property] == b[property] ? 0
                    : a[property] < b[property] ? -1 : 1;
                });
            } else {
                array.sort(function (a, b) {
                    return a[property] == b[property] ? 0
                    : a[property] > b[property] ? -1 : 1;
                });
            }
        }

        element.onclick = function () {
            sorttaa();
        }
    }
};
```

Kuva 12: jarjesta-handleri

Kuvassa 13 on taulukko, jossa näytetään *teksteja*-taulukkomuuttujan sarakkeet *ekaTeksti* ja *tokaTeksti*. Tämän lisäksi sarakkeille on määritetty otsikot: *Teksti1* ja *Teksti2*. Taulukon otsikoihin on liitetty Knockoutille ominainen data-bind-käsittely, joka viittaa handleriin nimeltä *jarjesta*. Määrittämällä data-bind="jarjesta: { arr: teksteja, prop: ‘ekaTeksti’ }"> kutsutaan otsikkoa klikattaessa handleria nimeltä jarjesta ja jolle annetaan parametreiksi ”arr: teksteja” ja ”prop: ‘ekaTeksti’”. Näin handleri tietää että järjestetään *teksteja*-taulukkomuuttuja sarakkeen *ekateksti* mukaan.

```
<table>
    <thead>
        <th data-bind="jarjesta: { arr: teksteja, prop: 'ekaTeksti' }">
        Teksti1</th>
        <th data-bind="jarjesta: { arr: teksteja, prop: 'tokaTeksti' }">
        Teksti2</th>
    </thead>
    <tbody data-bind="foreach: teksteja">
        <tr>
            <td><span data-bind="text: ekaTeksti"></span></td>
            <td><span data-bind="text: tokaTeksti"></span></td>
        </tr>
    </tbody>
</table>
```

Kuva 13: Handlerin käyttäminen taulukon otsikossa

Myös valintapainikkeille on järkevää tehdä oma handleri varsinkin, jos saapuvan datan arvo on esimerkiksi 1 ja 0 arvojen True ja False sijaan. Handleri kannattaa myös tehdä, mikäli ei ole varmaa, millaisena valinnan arvot tulevat tai tiedetään, että ne voivat vaihtua datasta riippuen. Knockoutin data-bind osaa käsitellä arvoja True ja False, mutta esimerkiksi arvoja 1 ja 0 tai Y ja N se ei osaa käsitellä valintapainikkeiden yhteydessä.

Kuvan 14 esimerkki on yksinkertaisuudessaan handleri, joka kysyy, että onko saapuvan datan arvo True, 1 tai Y ja palauttaa arvon True, muuten False. Kuvassa 15 on checkbox-kenttä, johon on määritetty, että liitetään tähän muuttujan *valinta* arvo käyttäen *checked*-hanleria. Näin tämä valintapainike eli checkbox osaa näyttää valittua tilaa, mikäli siihen liitettävän datan arvo on True, 1 tai Y ja osaa olla näyttämättä sitä, mikäli se on False, 0 tai N.

```
ko.bindingHandlers.checked = {
    init: function (element, valueAccessor, allBindingsAccessor) {
        var value = valueAccessor();
        var checked;
        if (value == "True" || value == "1" || valu == "Y") {
            checked = true;
        } else {
            checked = false;
        }
        //ko.bindingHandlers.checked.update(element, function () {
        //    return checked;
        //});
        ko.bindingHandlers.checked.update(element, valueAccessor);
    }
};
```

Kuva 14: checked-handleri

```
<input type="checkbox" data-bind="checked: valinta" />
```

Kuva 15: Handlerin käyttäminen valintapainikkeessa

6.8 Tuetut selaimet

Projektin sovelluskehitys toteutettiin pääosin käyttäen uusinta versiota Google Chromesta, sekä rinnalla kulkivat koko kehityksen ajan Internet Explorer sekä Mozilla Firefox. Testauksessa käytettiin muutamia muitakin versioita näistä selaimista, mutta pääosin keskityttiin uusimpien selainversioiden testaukseen. KnockoutJS-kirjaston osalta jokainen toiminnallisuus toimi samalla tavalla riippumatta selaimesta tai selainversiosta.

KnockoutJS sanoo tukevansa kaikkia käytetyimpiä selaimia, kuten Google Chrome, Internet Explorer, Mozilla Firefox, Apple Safari for Mac OS, Apple Safari for iOS ja Opera, ainakin uusiempien versioiden osalta. KnockoutJS on testannut myös Firefoxin, IE:n ja Safari for iOS:in osalla vanhemmista versiosta. Näiden lisäksi KnockoutJS-kirjaston pitäisi toimia myös näillä selaimilla: Google Android OS Browser, Opera Mini, Google Chrome 5+, iOS Safari 5 ja Mac OS X Safari 3.1.2+, mutta KnockoutJS ei suorita jatkuvia testejä näille selaimille uusien KnockoutJS-päivitysten yhteydessä.

7 Yhteenveto tuloksista

Projektin käyttöliittymä saatiin toteutettua sekä projektista on saatu kerättyä hyvä määrä tietoa esimerkiksi tekniikasta ja siitä, kuinka MVVM on mahdollista toteuttaa. Tieto on kerätty tähän opinnäytetyöhön yksinkertaistamalla koodiesimerkkejä, jotta lukijan olisi helpompi sisäistää työn sisältö. Lisäksi tästä opinnäytetyöstä on jätetty pois asiat, jotka eivät suoraan liity MVVM-malliin. Mielestäni kaikki keskeisimmät asiat on kuitenkin tuotu osaksi työtä. Tämän opinnäytetyön ideana oli kerätä tietoa siitä, kuinka Knockoutin syntaksia käytetään hyödyksi osana käyttöliittymäohjelmointia ja tässä onnistuttiin hyvin.

Kuten tässä opinnäytetyössä on aiemminkin tuotu ilmi, kirjastojen tehtävänä on helpottaa kehittäjän työtä sekä tehdä syntaksista luettavampi ja ylläpidettävämpi. Kaikki tässä opinnäytetyössä näytetyt asiat on mahdollista toteuttaa perinteisellä JavaScriptillä, mutta kehittäminen olisi todennäköisesti huomattavasti vaikeampaa ja syntaksi olisi silloin huomattavasti vaikeampilukuisempaa. Käyttämällä näitä opinnäytetyössä näytettyjä koodiesimerkkejä, on yksinkertainen käyttöliittymäohjelmointi mahdollista suorittaa suhteellisen lyhyin koodimäärin. Esimerkiksi yksinkertaiset silmukat saadaan toteutettua hyvin yksinkertaisella syntaksilla.

Vaikka Knockout-kirjasto onkin tuonut projektiin paljon hyviä ominaisuuksia, on sen mukana tullut myös paljon haasteita. Ensinnäkin Knockoutin mallin ja syntaksin omaksuminen vei projektin aikana yllättävänkin paljon aikaa. Knockout oli alussa melko hankala kirjasto omaksua, mikä voi johtua myös siitä, että Knockout tuo täysin uudenlaisen toteutustavan osaksi sovellusta. Lisäksi esimerkiksi sisäkkäisten foreach-looppien toteutus voi Knockoutilla olla hyvinkin hankalaa, vaikkakin valmiiden Knockoutilla toteutettujen osioiden ylläpitäminen onkin hyvin helppoa ja nopeaa. Tämä voi johtua siitä, että Knockout-kirjaston syntaksi on tarvittaessa hyvinkin lyhyttä. Syntaksin lisäksi suorituskyvyn kanssa on projektin edetessä huomattu haasteita. Knockout-kirjasto suorittaa taulukoiden looppauksen huomattavasti hitaammin kuin perinteinen JavaScript. Kuitenkin tästäkin ongelmasta on päästy eroon esimerkiksi erilaisilla datan sivuutuksilla.

MVVM ja Knockout-kirjasto on kuitenkin ollut hyvä tapa kehittää web-sovelluksen käyttöliittymärajapintaa. Se on tuonut paljon helpotusta kehittämiseen, kuten esimerkiksi handlereiden avulla. Lisäksi yksinkertaiset datan liittämiset kenttiin on ollut yksinkertaista tehdä ja lähes kaiken Knockoutilla tehdyn toteutuksen ylläpidettävyys on ollu hyvin helppoa. Vaikka kaikki, mitä projektissa on toteutettu, olisi voitu toteuttaa vaikka ilman mitään kirjastoja, on mielestäni hyvä, että Knockout on otettu mukaan projektiin. Projektin käyttöliittymän syntaksi on loppujen lopuksi melko lyhyttä ja hyvin ymmärrettävää, näin ainakin kehittäjän näkökulmasta.

8 Pohdinta

Opinnäytetyön tavoitteena oli rakentaa käyttöliitymä käyttäen MVVM-mallia ja KnockoutJS-kirjastoa. Tavoite rakentaa käyttöliittymä onnistui hyvin. Käyttöliittymä saatiin tehtyä valmiiksi ja siitä saatiin kerättyä hyvin tietoa. Tuloksena saatiin toimiva käyttöliittymä osaksi web-sovellusta. Käyttöliittymä ottaa onnistuneesti dataa vastaan ja näyttää sitä näkymässä aivan kuten oltiin haluttukin. Pääpiirteiltään projekti onnistui hyvin, vaikkakin projektissa törmättiin usein erilaisiin haasteisiin liittyen niin käyttöliittymään, kuin muihinkin sovelluksen osiin. Kuitenkin projektin jälkeen olen sitä mieltä, että MVVM on hyvä valinta käyttöliittymän toteutustavaksi.

Tämän opinnäytetyön validiteetti on siinä määrin hyvä, että tutkimus on tehty omasta kokemuksesta tutkimusaiheen parissa työskentelystä. Tutkimuksen validiteettia laskee kuitenkin hieman se, että tein tutkimuksen yksin ilman, että siihen saadaan muiden kommentteja ja näin ollen tutkimuksessa tehdyt toteutustavat eivät välttämättä ole kaikkien mielestä järkevimpiä tai yksinkertaisimpia. Tämä, kuten yleensä monet muutkin ohjelmointiprojektit, voi jakaa mielipiteitä toteutustavasta, sillä tämäkin projekti on mahdollista toteuttaa useilla eri tavoilla, vaikka projektin otsikko ja tavoite olisikin sama.

Myös tämän opinnäytetyön reliabiliteetti on hyvä, sillä tämän opinnäytetyön tutkimusaineistolla eli toteutetulla koodilla on mahdollista toteuttaa samat asiat jokaisella toteutuskerralla ilman, että se jollain kerralla toimisi erilailla. Erilaisissa tutkimusympäristöissä voi olla eroja esimerkiksi koodin suorituskysyn kanssa, mutta samat tutkimustulokset saadaan jokaisella tutkimuskerralla, kunhan tutkimusvälineet eli esimerkiksi koodin suorittavat selaimet ovat samoja.

Tutkimuksen aineisto on saatu kerättyä järkevästi. Aineisto on järjestetty siten, että yksinkertaiset asiat on tuotu aineiston alkuun, josta askel kerrallaan siirrytään haastavampiin osioihin. Tämän tarkoituksena on tuoda lukijalle perustietoa tutkittavasta kohteesta ennen siirtymistä monimutkaisempiin osiin. Tutkittavasta kohteesta kerättyä aineistoa on havainnoitu kuvien avulla, jotta tutkittavan kohteen ymmärtäminen olisi helpompaa.

Lähteet

Kirjallisuus

Casteleyn, S., Daniel, F., Dolog, P. & Matera, M. 2009. Engineering Web Applications. Berlin: Springer.

Farrar, J. 2015. KnockoutJS Web Development. Birmingham: Packt Publishing.

Ferguson, R. & Heilmann, C. 2013. Beginning JavaScript with DOM Scripting and Ajax. 2. painos. Apress Media.

Holdener, A. 2008. Ajax: The Definitive Guide. United States of America: O'Reilly Media.

Peltomäki, J. 2007. JavaScript. Jyväskylä: WSOY.

Raasch, J. 2013. JavaScript Programming Pushing the Limits. United Kingdom: John Wiley & Sons.

Stephen D., MaryLynn T. & Frances J. 2012. Qualitative Research. San Francisco: John Wiley & Sons.

Sähköiset lähteet

Crockford, D. 2009. Introducing JSON. Viitattu 8.2.2015.
http://json.org/

Document Object Model (DOM): Objects and Collections. Viitattu 25.1.2015.
http://ptgmedia.pearsoncmg.com/images/9780137001316/samplechapter/JavaScriptFP_10_DHTMLObjColl.pdf

Franklin, S. 2011. Marakana jQuery Training. Viitattu 8.2.2015.
http://simeonfranklin.com/jquery/book_jquery.pdf

Hiltunen L. 2009. Validiteetti ja reliabiliteetti. Viitattu 25.1.2015.
http://www.mit.jyu.fi/ope/kurssit/Graduryhma/PDFt/validius_ja_reliabiliteetti.pdf

jQuery. Viitattu 8.2.2015.
https://jquery.com/

KnockoutJS. Viitattu 8.2.2015.
http://knockoutjs.com/index.html

Liu, W. 2007. Lecture 17: Ajax (Asynchronous JavaScript And XML). Viitattu 8.2.2015.
http://www.cs.toronto.edu/~wl/csc309/handouts/ajax.pdf

Open Source Initiative. Viitattu 8.2.2015.
http://opensource.org/licenses/

Taanila, A. 2007. Laadullisen aineiston analyysi. Viitattu 11.5.2015.
http://kelo.oulu.fi/jatkokoulutus/AT_Laadullisen_aineiston_analyysi_170407.pdf

Tuomivaara, T. 2005. Tieteellisen tutkimuksen perusteet. Viitattu 25.1.2015.
http://www.mv.helsinki.fi/home/ttuomiva/Y125luku6.pdf

Kaaviot

Kuvat