Saana Lukka

PRODUCT CATALOG MANAGER -SOVELLUS

Tietojenkäsittelyn koulutusohjelma

2019

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

Product catalog manager -sovellus

Lukka, Saana
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Huhtikuu 2019
Ohjaaja: Nieminen, Hans
Sivumäärä: 26

Liitteitä:

Asiasanat: verkko-ohjelmointi, käyttöliittymät, teollisuusyritykset

---

Tässä opinnäytetyössä luotiin paranneltu versio Roima Intelligence-yrityksen tuotekonfiguraattorista. Käytetyt ohjelmointikielet sekä tekniikat olivat jQuery, CSS, Java sekä Kendo UI. Käyttöliittymää muokattiin rakenteeltaan selkeämmäksi jQuery-JavaScript-kirjastoa käyttämällä, ja sovellukseen lisättiin uuden datan tallennusmahdollisuus Java-ohjelmointikieltä käyttämällä.

Kirjallinen dokumentaatio sisältää lyhyen esittelyn tuotekonfiguraattoreista sekä käytettävyyden huomioon ottamisesta sovelluskehityksessä. Kappaleessa 4 käsitellään suunniteltujen muutosten toteutus askel askeleelta käytännön näkökulmasta, ja lopuksi tarkastellaan lopullista tuotosta kuvankaappausten avulla havainnollistettuna.

Product Catalog Manager -application

Lukka, Saana
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Information Technology
April 2019
Supervisor: Nieminen, Hans
Number of pages: 26
Appendices:

Keywords: web development, user interfaces, industrial companies

---

The purpose of this thesis was to create a new, improved version of a product configurator of Roima Intelligence. The technologies and programming languages used in the process were jQuery, CSS, Java and Kendo UI. The user interface was redesigned to achieve clearer and more structured feel using jQuery JavaScript library, and support for saving new data was added using Java.

The written documentation contains a brief introduction to product configurators and considering usability in software development. Chapter four describes the actual steps taken to create the planned changes, and finally the end result is presented using visual aid from screenshots of the application.
.

SISÄLLYS

1 JOHDANTO

Maailman ensimmäisenä tehtaana voidaan ainakin tietyillä kriteereillä pitää **Matthew Boultonin** vuonna 1766 rakennuttamaa ” The Soho Manufactory”-nimeä kantanutta massatuotantolaitosta. (McBeath 2017)

Tuohon aikaan nykymuotoisista tuotannon tehostamiseen suunnatuista digitaalisista ratkaisuista nykymuodossaan tuskin osattiin vielä edes haaveilla. Tarve, joka ne lopulta synnytti, oli sitä vastoin varsin todellinen, sillä yksi yrityksen kilpailukyvyn määrittävistä asioista on prosessien tehokkuus (Reijo Rautauoman säätiö 2019). Prosessien tehostamiseen ja automatisointiin suunnattujen työkalujen tarve ja laatuvaatimukset skaalautuvat suoraan yrityksen koon mukaan. Isompi yritys tarkoittaa suurempaa määrää muuttujia, ja suuri määrä muuttujia nostaa virheiden todennäköisyyttä, oli kyse sitten teknisistä vioista tai inhimillisistä virheistä.

Inhimillisten virheiden estämisessä nousee käytettävyys erittäin keskeiseen osaan. Mikäli asiantuntijatarpeisiin suunnitellut sovellukset ovat sekavia ja vaikeasti ymmärrettäviä, nousee inhimillisten virheiden todennäköisyys, ja kriittisessä tilanteessa tapahtunut virhepainallus tai ajatuskatkos saattaa pahimmillaan eskaloitua merkittäviksi liikevaihtotappioiksi. (Koski 2015)

Opinnäytetyön toimeksiantaja, **Roima Intelligence Oy,** on vuonna 2014 perustettu, valmistavan teollisuuden kilpailukyvyn parantamiseen erikoistunut yritys. Opinnäytetyön tarkoituksena oli suunnitella muutamia vuosia sitten talon sisällä toteutettuun myyntikonfiguraattoriin uusi käyttöliittymä. Ensimmäiseen versioon ei oltu tyytyväisiä, sillä se koettiin sekavaksi ja osin käyttötarkoitustaan vastaamattomaksi. Toinen keskeinen uudistustoive oli visuaalinen ilme, joka haluttiin saada vastaamaan Roiman omaa, kaikille yrityksen sovellustuotteille yhteistä tyylistandardia. Konfiguraattori oli toteutettu selainsovelluksena, jonka pohjana toimii Play Framework -arkkitehtuuri. Käyttöliittymä oli rakennettu JavaScriptin jQuery-laajennuskirjastolla ja Progress Software Corporationin tarjoamalla Kendo-komponenttikirjastolla. Käyttöliittymämuutoksia varten ei lähdetty tekemään

radikaaleja muutoksia olemassa oleviin pohjaratkaisuihin, vaan kaikki muutokset toteutettiin puhtaasti olemassa olevaa koodia muokkaamalla.

2 SUUNNITELMA

2.1 Konfiguraattori käsitteenä

Tuotteiden konfiguroitavuus on konsepti, jonka päämääränä on yhdistää massatuotanto ja yksilöidyt tuotteet. Konfiguroitavuudella halutaan saavuttaa yksilöllisten tuotteiden kyky vastata laajoihin sekä monimutkaisiin asiakastarpeisiin, sekä massatuotannon mahdollistamat lyhyet tuotantosyklit. Kyseinen tavoite kuitenkin asettaa melkoisia vaatimuksia sekä itse tuotteille että niiden ylläpitämiseen vaadittaville työkaluille. Hyvää konfiguroitavuus koostuu sekä tuotekokoonpanojen huolellisesta suunnittelusta että hyvästä tuotekonfiguraattorista. Hyvin ja huolella suunniteltu konfiguroitava tuote ei tarjoa mainittavia hyötyjä, mikäli konfiguroitavuuden käsittelyyn sekä ylläpitämiseen tarkoitettu työkalu ei kykene vastaamaan sille asetettuihin vaatimuksiin. Luonnollisesti tämä pätee myös toisin päin. (Tiihonen & Soininen 1997)

Erityisen suuriksi haasteet muodostuvat, mikäli konfiguraattorin pitää pystyä palvelemaan useiden erilaisten yritysten erilaisiin tarkoituksiin suunnattuja tuotekokoonpanoja. Juuri tästä syystä opinnäytetyön aiheena olevan konfiguraattorin ensimmäisestä prototyypistä haluttiin mahdollisimman yksinkertainen, jotta sen avulla pystyttäisiin kartoittamaan asiakkaiden tarpeita ja vastaamaan niihin mahdollisimman laajasti myöhemmissä vaiheissa.

Yksi tuotekonfiguraattorin keskeisiä toiminta-ajatuksia on yksinkertaistaa suunnitteluvaihetta. Pääsääntöisesti yrityksen myyntihenkilöstö ei ole perillä kokoonpanojen teknisistä ulottuvuuksista läheskään yhtä hyvin kuin tuotteen suunnittelusta ja/tai valmistuksesta vastaava henkilöstö, jolloin myyjä joutuu hankalaan välikäteen yrittäessään yhdistää asiakkaan vaatimukset tuotteen tarjoamiin mahdollisuuksiin, joutuen ottamaan samaan aikaan huomioon tuotekokoonpanojen eri

komponenttien yhteensopivuusongelmat sekä tekniset rajoitukset. Konfiguroitavat tuotekokoonpanot saattavat olla erittäin monimutkaisia, ja niiden oikeaoppinen yhdistely muuttuu haastavaksi. Tuotekonfiguraattori vastaa tähän tarpeeseen mahdollistamalla suunnittelijan tekemän esikonfiguraation, johon on määritelty kaikki toimivan kokoonpanon suunnitteluun vaikuttavat huomionarvoiset aspektit. Tätä pohjaa käyttämällä myyjä kykenee rakentamaan asiakkaalle kokoonpanon, jonka toimivuuden hän kykenee varmistamaan jo tekovaiheessa. (Tiihonen & Soininen 1997)

2.2 Muutama sana käytettävyydestä

Virallinen ISO 9241-11 määritelmä käytettävyydelle on seuraava: ”Laajuus, jossa tuotetta pystytään käyttämään määriteltyjen käyttäjien toimesta määriteltyjen päämäärien saavuttamiseksi tehokkaasti ja tyydyttävästi määritellyssä kontekstissa.” (The Interaction Design Foundation 2019). Teoksessaan “One-dimensional usability - influence of usability on consumers' product preference” professori **Turkka Keinonen** listaa käytettävyyden periaatteiksi mm. seuraavia:

- Johdonmukaisuus, joka tarkoittaa käyttöliittymän tarjoamien tilanteiden yhdenmukaisuutta,
- Hallittavuus, jonka mukaan käyttäjä ohjaa laitetta tai sovellusta eikä vain anna sille komentoja joiden mukaan sen pitää toimia,
- Sopiva esitystapa, joka tarkoittaa sitä että käyttäjä saa joka tilanteessa selkeää tietoa siitä mitä on tapahtumassa,
- Virheiden sieto, joka sisältää virheistä ilmoittamisen sekä niiden käsittelyn sekä varoitukset ja käskyn peruutusmahdollisuuden,
- Muistettavien asioiden määrä, joiden minimointi luodaan rajattuja vaihtoehtoja tarjoamalla jotta käyttäjän ei tarvitsisi joka tilanteessa muistaa kaikkea,
- Tehtävään sopivuus, joka tarkoittaa että näkymä esittää vain tarvittavat asiat johdonmukaisessa järjestyksessä,
- Opastus, joka on saatavilla ohjeina ja/tai käsikirjana. (Keinonen 1998)

Tuotekonfiguraattorin tapauksessa päätettiin keskiöön nostaa erityisesti johdonmukaisuus, muistettavien asioiden määrä sekä tehtävään sopivuus, sillä kaikki

nivoutuvat läheisesti yleiseen selkeyteen, joka oli konfiguraattorin käyttöliittymäuudistuksen tärkein yksittäinen aspekti.

3 TYÖKALUT

3.1 Kendo

Kendo UI on alun alkaen Telerik-nimisen sovelluskehitystyökaluja tarjoavan yhtiön tuote. Vuonna 2014 Telerik siirtyi Progress Softwaren omistukseen, jolloin aiemmin tietokanta -ja pilvipalveluihin keskittynyt Progress kykeni laajentamaan toiminta-alaansa myös käyttöliittymäratkaisujen tarjoajana. (Progress 2014). Kendo UI-komponentit tukevat suosituimpia JavaScript-arkkitehtuureita. Tuettuina ovat jQuery, Angular, React sekä Vue (Progress 2019).

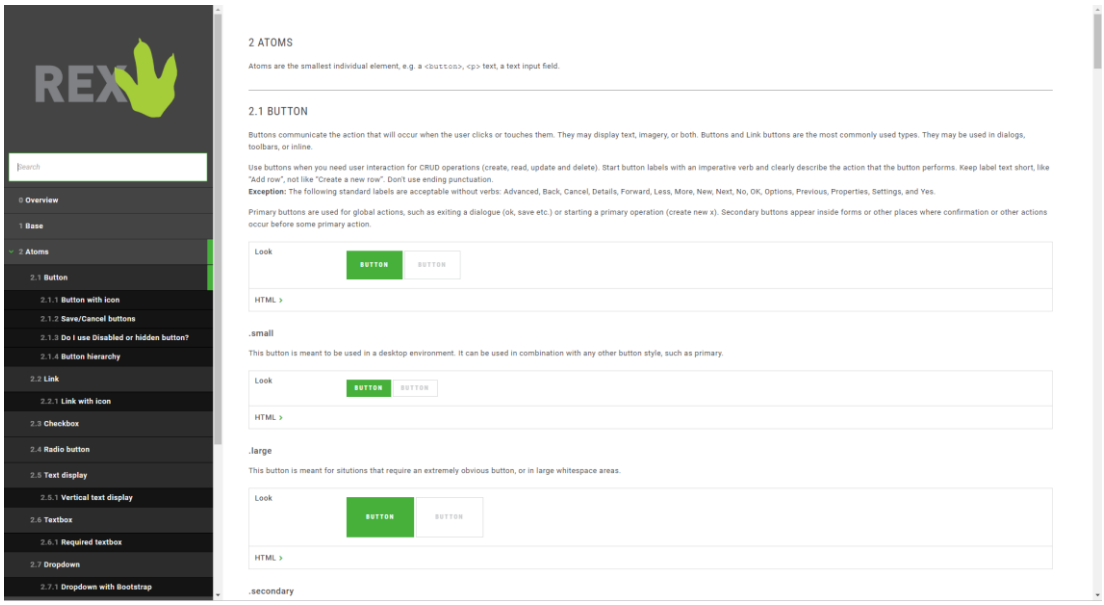
Kendo UI tarjoaa suoraviivaisen ja ylläpidollisesti helpon vaihtoehdon selainsovelluksen käyttöliittymän rakentamiseen (Kuva 1). Valmiit komponentit sisäänrakennettuine toiminnallisuuksineen nopeuttavat sovelluksen käyttöönottoa, sillä koko toiminnallisuuden suunnittelun ja ohjelmoimisen sijaan käyttöönotosta selviää valmiita komponentteja konfiguroimalla. Suoraviivaisuudella sekä vaivattomuudella on toki myös kääntöpuolensa. Monimutkaisempi toiminnallisuus on piilotettu ”konepellin” alle asiakkaan tavoittamattomiin, joten vaikka monimutkaisenkin toiminnallisuuden saattaa saada käyttöön muutamalla komennolla, yhtäkkiseltään helpon oloinen muokkaus voi aiheuttaa yllättävää päänvaivaa. Tiivistettynä Kendo UI soveltuu käyttöön sitä paremmin, mitä vähemmän käyttöliittymän rakentaminen vaatii sen komponenttien perusrakenteesta poikkeamista. Tämä tuli toiminnallisuutta muokattaessa todistettua useampaan otteeseen.

```
<div id="example">
    <div id="grid"></div>
    <script>
        $(document).ready(function () {
            $("#grid").kendoGrid({
                dataSource: {
                    type: "odata",
                    transport: {
                        read: "https://demos.telerik.com/kendo-ui/service/Northwind.svc/Customers"
                    },
                    pageSize: 20
                },
                height: 550,
                groupable: true,
                sortable: true,
                pageable: {
                    refresh: true,
                    pageSizes: true,
                    buttonCount: 5
                },
                columns: [{
                    template: "<div class='customer-photo'" +
                    "style='background-image: url(../content/web/Customers/#:data.CustomerID#.jpg);'></div>" +
                    "<div class='customer-name'>#: ContactName #</div>",
                    field: "ContactName",
                    title: "Contact Name",
                    width: 240
                }, {
                    field: "ContactTitle",
                    title: "Contact Title"
                }, {
                    field: "CompanyName",
                    title: "Company Name"
                }, {
                    field: "Country",
                    width: 150
                }]
            });
        });
    </script>
</div>
```

Kuva 1. Kendo UI grid-komponentin jQuery-versio (Progress 2019)

3.2 Roima REX

Roima Intelligencen Rex-tyylikirjaston määrittelyt vastasivat pääpiirteissään konfiguraattorin visuaalisesta ilmeestä (Kuva 2). Rex sisältää määrittelyt aina väripaletista ja fonteista elementtien asetteluun sekä valintanappien yhdenmukaiseen värikoodaukseen. Koska käyttöliittymäsuunnittelussa vastaan tulevat tilanteet eivät ole aina yksiselitteisesti määriteltävissä eivätkä samat määrittelyt välttämättä näytä visuaalisesti miellyttävältä eri tilanteissa, oli Rex asettelun puolesta käytössä enemmänkin suuntaa-antavana kuin kiveen hakattuna lopullisena totuutena. Tyylimäärittelyjen perimmäinen tarkoitus on kuitenkin antaa vaikutelma yhdenmukaisesta ilmeestä, joka aiheuttaa ehdollistumisen tiettyjen visuaalisten elementtien ja tietyn brändin yhteen liittämiselle. Mikäli tuotteen visuaalinen ilme aiheuttaa tämän ehdollistumisreaktion, on päämäärä saavutettu, eikä tässä tapauksessa ole suurtakaan merkitystä sillä, onko esimerkiksi marginaalien paksuus kenties viisi vai kuusi pikseliä.



Kuva 2. Roiman Rex-tyylikirjasto (Roima Intelligence 2019)

4 KÄYTÄNNÖN TOTEUTUS

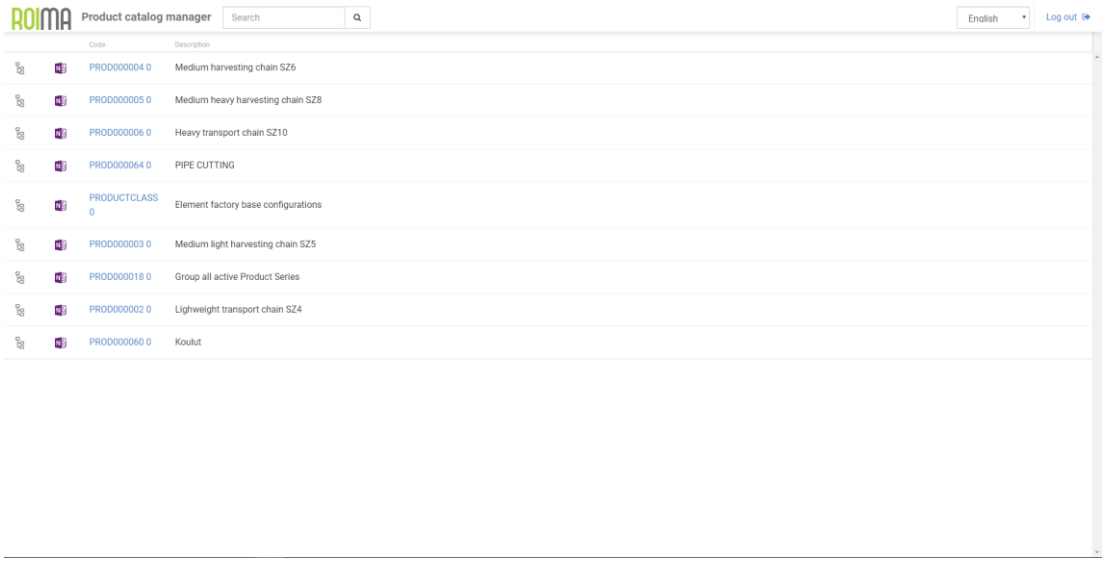
4.1 Lähtötilanne

Työn alkaessa olemassa olevan konfiguraattoriversion toiminnallisuus oli suunniteltu ja toteutettu tiedon haun ja järjestämisen osalta alusta loppuun asti, mutta sen esittäminen loppukäyttäjälle oli toteutettu täysin Kendon UI-komponenttien oletusasetuksia käyttäen. Koska konfiguraattorin haluttiin olevan pääasiassa myyjän työkalu, piti tiedon esittämisestä tehdä mahdollisimman selkeää ja loogista. Toinen tärkeä aspekti näkymien suunnittelussa koski visuaalista ilmettä, sillä lähtöversiossa käytetyt Telerikin oletusvärit ja -fontit haluttiin vaihtaa Rexin määrittelemään visuaaliseen ilmeeseen. Tällä keinoin konfiguraattori saataisiin näyttämään yhdenmukaiselta yrityksen muiden tuotteiden kanssa. Myös itse konfiguraationäkymä vaati uudistamista, ja se oli tarkoitus toteuttaa noudattamalla samoja periaatteita kuin myyntinäkymä, kuitenkin muutamilla keskeisillä eroavaisuuksilla. Lähtötilanteessa uusien tuoteperheiden, tuotteiden, ominaisuuksien eikä konfiguraatioiden lisääminen ollut mahdollista, joten myös tämä osa-alue kaipasi täydennystä. Muita kaivattuja

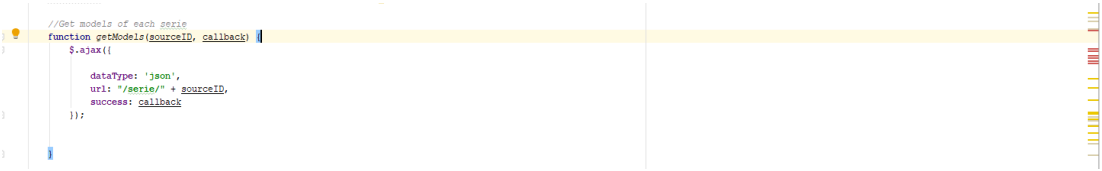
lisäyksiä olivat hintatietojen esitys ja yhteenveto kokonaislaskelmineen, sekä mahdollisuus muuntaa tarjous lähetettävään sekä tulostettavaan muotoon.

4.2 Päänäkymä

Päänäkymässä liikkeelle lähdettiin tilanteesta, jossa näkymä näytti listauksen kaikista saatavilla olevista tuoteperheistä (Kuva 3). Näkymä haluttiin pilkkoa konfiguraatio- sekä tarjousosioon, sekä mahdollistaa myös yksittäisten tuotteiden valitseminen koko tuoteperheen sijaan. Toiminnallisuutta lähdettiin toteuttamaan rajaamalla alkuperäinen taulukkonäkymä sivun vasempaan laitaan, ja sijoittamalla oikeaan laitaan toinen taulukkonäkymä, joka vasemman puolen konfiguraatiotaulukkoon lisätystä tuotevalinta-pudotusvalikosta valitsemalla näyttää kyseiselle tuottelle tallennetut tarjoukset. Tämä saavutettiin luomalla funktiot, joista toinen sivun latautuessa haki jokaisen saatavilla olevan tuoteperheen kaikki tuotteet (Kuva 4), ja toinen sijoitti ne pudotusvalikkoon (Kuva 5). Kun käyttäjä tekee valinnan pudotusvalikossa, hakee toinen funktio valitun tuotteen kaikki tarjoukset ja näyttää ne oikean laidan tarjouslistauksessa. Oikean reunan tarjousnäkymään haluttiin myös tarjoushaku, joka toteutettiin Kendon sisäänrakennetulla filtteröintitoiminnolla.



Kuva 3. Päänäkymä, lähtötilanne.



```
//Get models of each serie
function getModels(sourceID, callback) {
    $.ajax({

        dataType: 'json',
        url: "/serie/" + sourceID,
        success: callback
    });

}
```

Kuva 4. Tuoteperheen tuotteiden hakeminen palvelimelta. (jQuery)

```
//Create dropdown list containing models of each serie
function sourceCallback(result) { //Create select list from serie products
    var s = $('#' + result.ID);

    s.append($('<option value="" selected="selected"></option>'));
    for (var i = 0; i < result.Models.length; i++) {
        s.append($('<option value=' + result.Models[i].ID + '>' + result.Models[i].Desc1 + '</option>'));
    }

}
```

Kuva 5. Tuotteiden sijoittaminen pudotusvalikkoon. (jQuery)

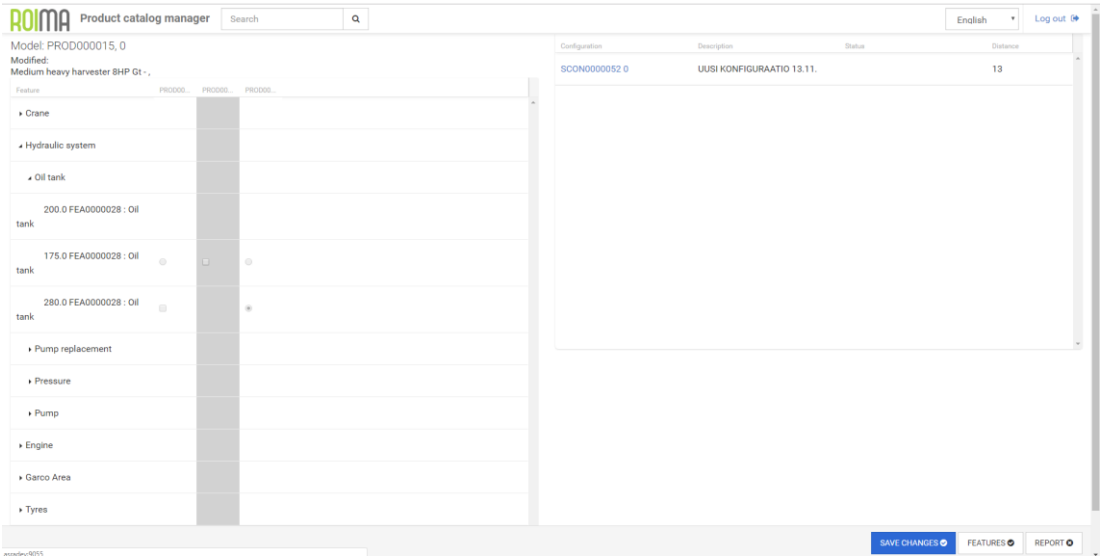
4.3 Myyjän näkymä

4.3.1 Yleisilme

Lähtötilanteessa myyjän näkymä oli kyllä sinänsä olemassa, mutta se ei vastannut suunnittelultaan myyjän tarpeita (Kuva 6). Keskeisiä ongelmia olivat sekavan oloinen rakennepuumalli, käyttötarkoitukseen nähden tarpeettoman suuri tuotekuvaikkuna, hintatietojen puuttuminen sekä yleisilmeeltään sekava sijoittelu. Sekaannusta aiheutti myös kaikkien tuoteperheen tuotteiden näkyminen samalla sivulla, sillä myyjä tulisi käyttämään näkymää vain yhdestä tuotteesta luodun konfiguraation tekemiseen.

Ensimmäinen tehtävä koskikin näkymän selkeyttämistä; rakennepuu ja itse valintojen tekeminen piti eriyttää omiksi kokonaisuuksikseen, tuotekuvanäkymä päätettiin korvata listatyyppisellä ominaisuusnäkymällä ja tuotekuvat sijoitettaisiin tuotevalintojen yhteyteen. Tietokannasta löytyi jo olemassa oleva hintatietosarake, joka voitaisiin lisätä tuotetietokyselyyn sekä Play Frameworkin rakenteisiin, jotta näkymä osaisi näyttää sen osana jokaista tuotetta. Perinteisten lomakevalintojen sijaan päätettiin käyttää kookkaita valintanappeja, jotka sisältäisivät tuotenimen lisäksi tuotekuvan sekä hinnan. Sijoittelu haluttiin jättää tarkoituksella väljäksi, jotta asiakkaan niin halutessa nappiin pystyy lisäämään myös muuta tuotetietoa. Näkymään haluttiin myös erillinen lisätietokenttä, jonka avulla tarjoukseen pystyttäisiin

lisäämään käsin ylimääräisiä huomautuksia tai tuotteita, sekä yhteenveto, joka ilmoittaisi valitun konfiguraation kokonaishinnan, sekä mahdollistaisi alennuksien lisäämisen.



Kuva 6. Myyjän näkymä, lähtötilanne

4.3.2 Rakennepuun pilkkominen

Kendon ideologian mukaisesti näkymässä esitetty data haetaan REST (Representational State Transfer) -kutsulla ns. datalähdekomponenttiin, josta se edelleen siirretään halutun käyttöliittymäkomponentin näytettäväksi. Näkymän latautuessa datalähdekomponenttiin sidottu käyttöliittymäkomponentti käynnistää datahaun, ja haun päätyttyä datalähdekomponentti lähettää tuloksen halutussa muodossa käyttöliittymäkomponentille, joka edelleen tulostaa datan selaimeen. Datan muuttuessa käyttöliittymäkomponentti ilmoittaa siitä datalähdekomponentille joka kutsuu vastaavaa toimenpidettä Play-sovelluksen kontrolleriluokasta. (Progress iii, 2019)

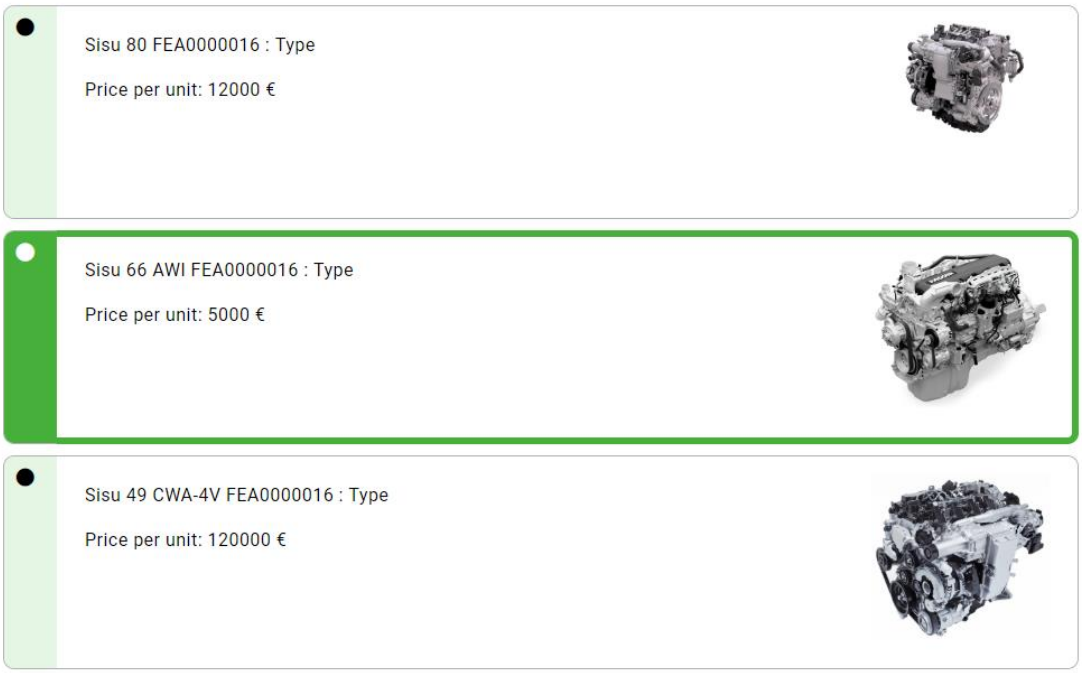
Koska käyttöliittymäkomponentti näyttää aina täsmälleen datasourcen hakeman datan, rakennepuun lehtisolmujen siirtäminen toiseen komponenttiin päätettiin hoitaa seuraavanlaisella tavalla: Määritellään ensin datasource, joka suorittaa REST-kutsun ja hakee koko rakennepuun sisällön. Datahaun käynnistyessä käynnistyy myös funktio, joka käy kokoelman jokaisen objektin läpi, siirtää lehtisolmut toiseen,

paikalliseen datalähdekomponenttiin, ja poistaa ne alkuperäisestä kokoelmasta. Alkuperäinen kokoelma sidotaan hierarkiseen rakennepuukomponenttiin, ja irrotetut lehtisolmut sidotaan näkymään lisättyyn taulukkonäkymäkomponenttiin. Koska jokainen lehtisolmu sisältää kentän, jonka arvona on isäntäobjektin id, pystytään taulukkonäkymälle kertomaan, minkä objektin lapset sen milloinkin halutaan näyttävän. Puurakenteen objektin valinta käynnistää taulukkonäkymän päivityksen. Tätä samaa periaatetta on tarkoitus käyttää sekä myyjän näkymässä että konfiguraationäkymässä, sillä erotuksella että myyjän näkymässä lehtisolmut sisältävää taulukkonäkymää pitää muokata varsin eri näköiseksi alkuperäisasetuksista.

4.3.3 Valintanapit

Valintanappien toteutus edellytti Kendon grid-komponentin toiminnallisuuden venyttämistä äärimmilleen. Visuaalinen suunnittelu kätkee alleen perinteiset html-lomakkeista tutut valintaruutu -ja valintanappielementit, mutta ne on upotettu css-tyylejä apuna käyttäen osaksi napin rakennetta. Niiden toiminnallisuus on kuitenkin säilytetty ennallaan. Konfiguraationäkymässä additional-arvolla varustettuja ominaisuuksia (valintaruutu) pystyy valitsemaan samanaikaisesti useamman mutta yhdenkään valitseminen ei ole pakollista (Kuva 7), mutta option-arvolla varustettuja ominaisuuksia (valintanappi) pystyy valitsemaan vain yhden kerrallaan (Kuva 8). Toinen option-arvon ominaisuus on pakollisuus, eli valintaa ei voi jättää tekemättä. Esimerkkinä voisi käyttää vaikkapa autoa, johon pystyy valitsemaan samanaikaisesti sekä navigaattorin että ilmastoinnin, joista kumpikaan ei ole kuitenkaan pakollinen auton käyttötarkoitusta vastaavan toiminnan takaamiseksi (additional), mutta korimallista tai moottorista voi valita vain yhden vaihtoehdon, ja näiden valintojen tekemättä jättäminen estää toimivan auton rakentamisen.

Kuva 7. Plus-merkki viestii ns. additional- valinnasta, joita saa olla samassa kategoriassa valittuna useampia samanaikaisesti, mutta yhtäkään ei tarvitse valita.



Kuva 8. Option-valinta, joita voi olla valittuna vain yksi kerrallaan. Esimerkkikuvassa voi valita kolmesta eri moottorivaihtoehdosta. Toisin kuin additional-valinnassa, tässä tapauksessa valinta on pakollinen.

4.3.4 Liikennevalot

Kenties merkittävin muutos myyjän näkymään on ns. liikennevalojärjestelmä, joka ilmoittaa yhdellä silmäyksellä tekeillä olevan tarjouksen tilan. Liikennevalojen logiikka on seuraava: ne rakennepuun yläkategoriat, joita ei ole vielä ollenkaan avattu, sisältävät keltaisen kysymysmerkkikuvakkeen, joka poistuu, kun valikko laajennetaan. Alakategoriat voivat sisältää joko vihreän oikein-merkin (ko.kategoriassa ei huomautettavaa), keltaisen kysymysmerkin (kategoriassa ei ole tehty yhtään valintaa, mutta se ei myöskään sisällä ehdottomasti pakollisia valintoja) sekä punaisen huutomerkin (kategoriasta puuttuu valintoja jotka on tehtävä ko. tuotteen toiminnan takaamiseksi). Mikäli liikennevaloja haluttaisiin havainnollistaa esim. tuotteen ollessa auto, keltainen kysymysmerkki saattaisi ilmoittaa tilanteesta, jossa lisävarusteena myytävää navigaattoria ei ole valittu. Auto on kuitenkin mahdollista toimittaa ilman navigaattoria, kun taas moottorivalinnan puuttuminen aiheuttaisi punaisella huutomerkillä havainnollistettavan virhetilanteen. Lisäksi liikennevalot ilmoittavat myös, mikäli kategoriassa on jo vierailtu, sillä kategorian laajentaminen aiheuttaa liikennevalokuvakkeen taustan värin vaihtumisen.

Liikennevalojen tekninen toteutus vaati jonkin verran luovaa ongelmanratkaisukykyä, sillä Telerikin Treelist-widget on suunniteltu konfiguroitavaksi datamallin pohjalta luotujen JavaScript-objektien ehdoilla, eikä itse näkymän elementteihin käsiksi pääseminen onnistu yhtä yksinkertaisesti. Ratkaisussa jouduttiin jakamaan liikennevalojen toiminnallisuuden ohjaaminen useaan eri kategoriaan: esim. lähtötilannetta (Kuva 9) kontrolloi eri funktio kuin tietojen päivittämisestä vastaava toiminnallisuus. Koska rakennepuun alakategorioiden lapsielementit irrotetaan sivun lataamisvaiheessa omaksi kokoelmakseen ja sivun keskelle aukeavaa taulukkonäkymää ei tässä vaiheessa ole vielä olemassa, on lähtötilanteen määrittelevän funktion haettava lapsielementtien statustiedot datamallista käymällä läpi sivun latausvaiheessa olemassa oleva konfiguraatio. Tämän jälkeen liikennevaloikonin tilan päivittävä funktio tarkistaa joka klikkauksella, onko taulukossa tapahtunut muutoksia, ja päivittää liikennevaloikonin tarpeen mukaan klikatun elementin yläkategoriaan. Reaaliaikainen päivitys vaatii itse html-elementtien tyylimääritysten muokkaamisen, johon onneksi löytyi työkaluksi Treelist-näkymässä jokaiselle riville rakennenäkymän luomisvaiheessa määräytyvä rivi-id. Funktio muuttaa kunkin rivi-id:n tiedoilla löytyvän css-määrityksen taustavärin, poistaa rivin sisältä löytyvän liikennevalosolun sisältämän kuvaketagin, ja korvaa sen vastaavalla uudella.

```
for (var i = 0; i < confDataSource._data.length; i++) {
    if (dataItem.ItemID === confDataSource._data[i].ItemID) { //Find the object that corresponds to the parameter dataitem
        children = confDataSource.childNodes(confDataSource._data[i]); //Create an array from the object's children editablerows
            for (var j = 0; j < children.length; j++) { //Iterate through the children

                if (children[j].columns[@modelid].type === 'string' && children[j].columns[@modelid].value === 'false') {
                    //Exclude the hidden children
                }
                else if (children[j].columns[@modelid].type === 'string' && children[j].columns[@modelid].value === 'true') {
                    //If standard option, then green
                    isGreen = true;
                    isAdditional = false;
                }
                else if (children[j].columns[@modelid].type === 'radio' && children[j].columns[@modelid].value === 'true') {
                    //If default option checked, then green
                    isGreen = true;
                    isAdditional = false;
                }
                else if (children[j].columns[@modelid].type === 'radio' && children[j].columns[@modelid].value === 'false') {
                    //If radio option present, even if not checked, group loses the "additional" status
                    isAdditional = false;
                }
            }
            if (isGreen === false && isYellow === false && isAdditional === false) {
                //If no default is checked and no standard option present, red
                isRed = true;
            }
            else {
                if (isAdditional === true) { //Checks if group contains only additional options
                    for (var k = 0; k < children.length; k++) {
                        if (children[k].columns[@modelid].type === 'checkbox' && children[k].columns[@modelid].value === 'true') {
                            isGreen = true;
                        }
                        else if (children[k].columns[@modelid].type === 'checkbox' && children[k].columns[@modelid].value === 'false') {
                            isYellow = true; //in case yellow was set to false by the string/false-children
                        }
                    }
```

Kuva 9. Liikennevalojen lähtötilanteen määrittävä funktio (jQuery)

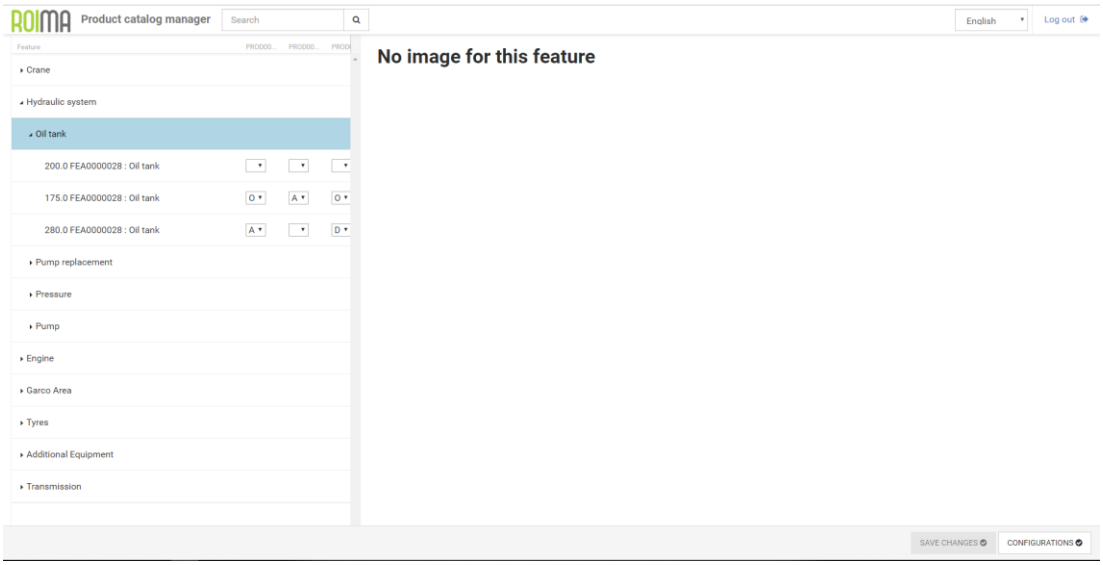
4.3.5 Tarjouksen yhteenveto

Tarjouksen yhteenvetoa varten myyjän näkymään lisättiin oma, erillinen taulukkonäkymäkomponentti, jonka näkyvyys rajoitettiin tilanteeseen, jossa käyttäjä valitsee hintayhteenvedon näyttämisen vasemman reunan puurakenteesta. Taulukkonäkymä listaa kaikki valitut ominaisuudet, niiden hinnat sekä yhteenlasketun hinnan, sekä tarjoaa käyttäjälle mahdollisuuden syöttää alennusprosentti taulukkohintaan. Mikäli alennusprosentti-kentässä on syötettyä dataa, laskee taulukko kyseisen ominaisuuden alennetun hinnan. Kokonaishinnan lisäksi yhteenvetoriviltä löytyy tarjoussarake, joka näyttää kokoonpanon yhteenlasketun hinnan siten, että mahdollinen alennettu hinta on valittu kokonaishintaan normaalihinnan sijasta.

4.4 Konfiguraationäkymä

4.4.1 Yleisilme

Konfiguraationäkymän perimmäinen tarkoitus on oletuskokoonpanojen alustava määrittely myyntikonfiguraatioita varten. Konfiguraationäkymä mahdollistaa tuotteeseen saatavilla olevien ominaisuusvaihtoehtojen määrittelyn, sekä sen, onko ominaisuus standardi, vaihtoehtoinen vai valinnainen. Toisin kuin uudistetussa myyntinäkymässä, konfiguraationäkymässä on tarkoituksenmukaista tarkastella kaikkia tuoteperheen tuotteita samanaikaisesti, jotta niiden väliset eroavaisuudet ovat selkeästi hahmotettavissa (Kuva 10). Suuri, lähes puolet näkymästä vievä 3D-tuella varustettu tuotekuvaikkuna ei ollut tarkoituksenmukainen konfiguraattorin käyttötarkoitusta ajatellen. Konfiguraationäkymä jaettiin myyntinäkymää mukaillen vasempaan reunaan sijoitettuun rakennepuuhun, itse valinnat sisältävää taulukkoon sekä oikeaan reunaan sijoitettuun inforuutuun (Kuva 11), joka näyttää taulukon rivin infokuvaketta klikkaamalla tuotekuvan sekä asiakkaan haluamia tuotetietoja (demodatan kyseessä ollessa tuotteen nimen sekä hinnan).



Kuva 10. Konfiguraationäkymä, lähtötilanne.

```
function showInfo(e) {

    $(".info-content").empty();

    $("#infocard").css("display","block");

    var tr = $(e.target).closest("tr");     // get the current table row (tr)
    var dataItem = this.dataItem(tr);          // get the date of this row


    var featureName = '<div class="info-desc">' + dataItem.description + '</div>';
    var featureImage = getImage(dataItem);
    var featurePrice;

    $(".info-content").append(featureName);

    if (dataItem.Price !== null) {

        featurePrice = '<div class="info-desc">Price: ' + dataItem.Price + ' €</div>';
        $(".info-content").append(featurePrice);
    }

    $(".info-image").html(featureImage);

} //Show data in infocard
/* ************************************************************
```

Kuva 11. inforuudun koodi (jQuery)

4.4.2 Tuotteiden lisääminen

Tuotteiden lisääminen vaati käyttöliittymämuutosten lisäksi tuen rakentamista myös palvelinpuolen koodiin (Kuva 10). Konfiguraattori on kytketty Roiman Aton-nimiseen tuotetiedon hallintajärjestelmään, joka käyttää datan hallinnoimiseen Oraclen SQL-tietokantaa. Tehtävää helpotti huomattavasti vastaavanlaisia tilanteita varten Roiman Aton-kehittäjien toimesta toteutettu PDMAPI-niminen sovellusrajapinta, jolla konfiguraattori saatiin kohtalaisen vaivattomasti keskustelemaan Atonin Oracle-kannan kanssa. Itse uuden tuotteen luomisen lisäksi metodi liittää tuoteperheen mahdolliset jo olemassa olevat ominaisuudet tuotteeseen, jolloin kytkös on tehty sekä tuotteen ja tuoteperheen välillä, että tuotteen ja ominaisuuskategorioiden välillä.

```
public void AddProductItem(String accessToken, String Desc, String serieid)
        throws Exception {


    try (Connection conn = app.getConnection(accessToken)) {

        conn.setAutoCommit(false);
        boolean commit = false;

        try {

            AuthenticationHelper.getAtonTicket(accessToken);
            ItemDataService ids = new ItemDataService(AuthenticationHelper.getAtonTicket(accessToken));

            ArrayList<ItemStructureRow> rows = new ArrayList<>();

            ItemDTO productItem = CatalogDAO.newProductItem(Desc);

            ids.saveItem(productItem);

            ItemDTO trparent = ids.getItem(serieid);
            ItemStructureRow treerow = CatalogDAO.addFeatureModelData(productItem,trparent);

            rows.add(treerow);
            ItemDao.reserveItems(conn, rows);
            ItemStrDAO.createRows(conn, rows);

            ItemDao.releaseItems(conn, rows);
            commit = true;

        } finally {
            if (commit) conn.commit();
            else conn.rollback();
        }
    }


}
```

Kuva 12. Uuden tuotteen lisääminen kantaan (Java)

4.4.3 Ominaisuuksien lisääminen

Myös ominaisuuksien lisääminen vaati käyttöliittymämuutosten lisäksi tuen rakentamista myös palvelinpuolen koodiin. Ominaisuuksien lisääminen ei kuitenkaan tapahtunut aivan yhtä suoraviivaisesti, sillä konfiguraatiodatan rakenne Atonissa aiheutti tehtävään muutamia rajoitteita. Huomattavin näistä oli tuoteperheiden ja ominaisuuskategorioiden kytkös toisiinsa itse ominaisuuksien avulla. Selkokielellä tämä tarkoittaa sitä, että tuoteperherakenne ja ominaisuuskategoriarakenne ovat toisistaan riippumattomia kokonaisuuksia, joita yhdistää vain tiettyyn ominaisuuteen, eli kummassakin kokonaisuudessa hierarkian alimmalla tasolla sijaitsevaan objektiin tallennettu tieto. Uuden ominaisuuden lisäys vaati siis ominaisuuskategorian lisäämistä rakenteeseen, muuten se ei ilmestyisi näkyviin vasemman reunan rakennepuussa valitulla tuoteperheellä (Kuva 13). Lisäksi tuotekuvan lisääminen vaati itse kuvatiedoston lataamisen lisäksi ns. dokumentin luontia Atoniin. Kuvatiedosto

sijoitettiin Aton-dokumenttiin, joka taas liitettiin luotuun ominaisuusobjektiin. Käytännössä tämä tarkoitti sitä, että tuotekuva pitäisi ensin lähettää palvelimelle erillisellä Ajax-kutsulla, jonka jälkeen luotaisiin uusi dokumentti, johon ladattu tiedosto sijoitetaan (Kuva 14). Termi ”Ajax” on lyhenne sanoista ”asynkroninen JavaScript ja XML”. Tekniikkaa käytetään datan asynkroniseen lähettämiseen ja vastaanottamiseen käyttöliittymän ja palvelimen välillä, ja asynkronisuus mahdollistaa toiminnan huomaamattomasti sovelluksen normaalin käytön taustalla. (Tutorialspoint, 2019)

Mikäli kutsun suorittaminen onnistuu, lähettää palvelin takaisin uuden dokumentin id:n, joka sijoitetaan seuraavaksi palvelimelle lähetettävään JavaScript-objektiin. Samalla kun javascript-objektin sisältämät tiedot sijoitetaan uuden Aton-nimikeobjektin kenttiin, tehdään dokumenttiliitos, jolloin konfiguraattori osaa näyttää oikean kuvan osana valittua kokonaisuutta.

```
function SendFeatureData(data) {

    var Group = $("#FeaGroup").val();
    var Feature = $("#Fea").val();


    if (($("#featurename").val() === '' || ($("#Fea").val() === "new" && $("#newfeature").val() === '')) || ($("#FeaGroup").val() === "new" && $("#newfeagroup").val() === '')) {

        alert("Please fill in all required fields");
    }

    else if ($("#Fea").val() === 'init' || $("#FeaGroup").val() === 'init') {

        alert("Please fill in all required fields");
    }

    else {

        console.log($("#FeaGroup").val());
        if ($("#FeaGroup").val() === "new") {


            var newFeatureGroup = CreateFeature($("#newfeagroup").val(), null, null, false);

            confDataSource.add(newFeatureGroup);
            $("#Fea").val("new");
        }

        if ($("#Fea").val() === "new" || $("#Fea").is(':disabled')) {

            console.log('newfeature');
            if ($("#FeaGroup").val() === "new") {

                Group = "newgroup";
            }


            var newFeature = CreateFeature($("#newfeature").val(), Group, null, false);
            confDataSource.add(newFeature);
            Feature = "newfeature";

        }

        var newFeatureValue = CreateFeature($("#featurename").val(), Feature, $("#featureprice").val(), true);

        // *** /CREATE FEATURE JSON *** //

        confDataSource.add(newFeatureValue);
        confDataSource.sync();

    }
```

Kuva 13. Uuden ominaisuuden lisääminen (jQuery)

```
public void AddFeatureImage(String token,File tmpFile, String docId)
        throws PdmException {
    try {
        ArrayList<MPair<String, String>> files = new ArrayList<>();
        files.add(new MPair<>("JPG", "spp_" + docId));

        DocumentDataService ds = new DocumentDataService(app.getTicket(token));

        try {
            HashMap<String, String> accessUrls = ds.importFiles(docId, files);

            MultipartBody request = Unirest.post(accessUrls.get("JPG"))
                    .field("name", "spp" + docId)
                    .field("file", tmpFile);

            HttpResponse<com.mashape.unirest.http.JsonNode> responseString = request.asJson();

        } catch (UnirestException e) {
            e.printStackTrace();

        }
    }finally {
        tmpFile.delete();
    }

}
```

Kuva 14. Uuden ominaisuuden kuvan lisääminen Aton-dokumenttiin (Java)

Käyttäjällä on mahdollisuus valita joko olemassa olevista kategorioista, tai luoda uusi alakategoria tai sekä ylä-että alakategoria. Visuaalisena piristeenä ikkunaan lisättiin kuvan esikatselumahdollisuus, joka tukee jpg-, png- sekä gif-tiedostomuotoja. Vaadittavia tietoja ovat nimi sekä kategoriat. Käyttäjän jättäessä jonkin näistä tyhjäksi sovellus näyttää ilmoituksen, joka kehottaa täyttämään puuttuvat kentät.

4.5 Visuaalinen ilme

Visuaalisen ilmeen uudelleen luomisessa työläimmäksi aspektiksi osoittautui Kendon oletustyylikirjastojen ylikirjoitus manuaalisesti REX-tyylikirjaston vastineilla. CSS on lyhenne sanoista ”Cascading Style Sheets”, ja vaikka sen ideologia perustuu melko keskeisesti juuri nimessäkin mainittuun tyylimääritysten kerrostumiseen, näiden kerrostettujen määritysten ylikirjoittamisessa useihin eri komponentteihin vierähti hetki jos toinenkin. Kaikeksi onneksi REX on johdonmukainen ja tarjoaa esimerkkiratkaisuja moniin eri tilanteisiin, joten komponenttien visuaaliseen suunnitteluun ei tarvinnut käyttää sen enempää aikaa. Lisäksi olemassa olevat REX-tyylikirjaston mukaan rakennetut tuotteet tarjosivat esimerkkiratkaisuja, mikäli jokin nimenomainen tilanne ei suoraan itse tyylikirjastossa esiintynytkään.

5 LOPPUTULOS

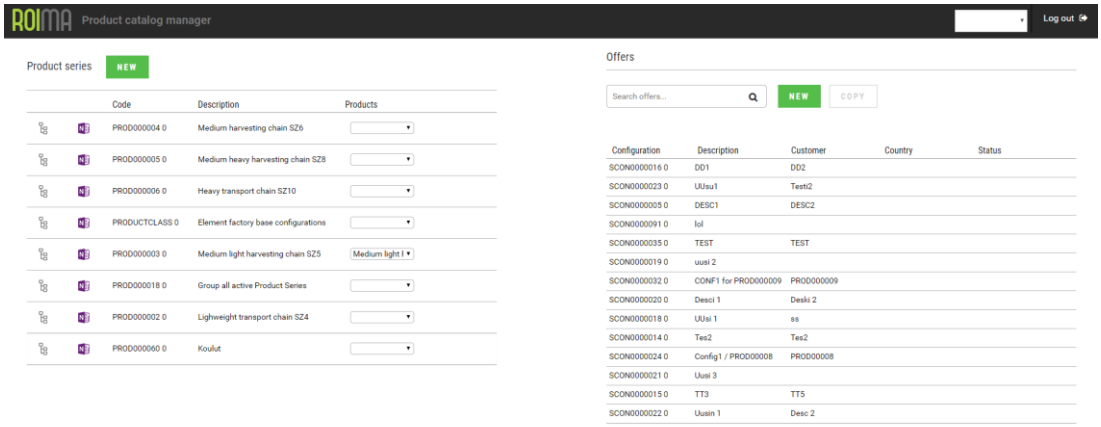
5.1 Yleisvaikutelma

Valmiissa versiossa sekä myyjän näkymä että konfiguraationäkymä on toteutettu samalla pääperiaatteella, mutta kuitenkin muutamalla olennaisella eroavaisuudella (Kuvat 15,16,17). Myyjän näkymässä keskeisessä asemassa olevat liikennevalot puuttuvat konfiguraationäkymästä, ja taulukkonäkymän asettelu on erilainen, mahdollistaen valintojen monipuolisuuden konfiguraationäkymässä sekä selkeyden myyjän näkymässä. Tulevaisuudessa liikennevaloja voi tarvittaessa jatkokehittää ilmoittamaan konflikteista valittujen ominaisuuksien välillä, mutta ensimmäisessä versiossa tätä ei nähty vielä ajankohtaisena.

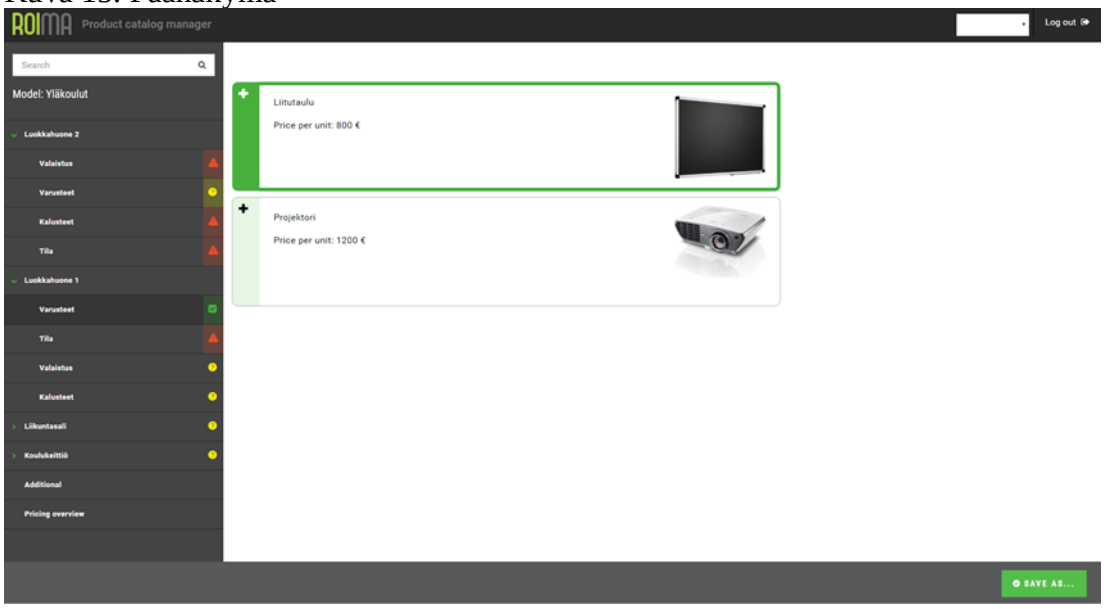
Päänäkymässä on myös mahdollisuus kopioida olemassa oleva tarjous ensin valitsemalla haluttu tarjous riviä napsauttamalla sekä sen jälkeen painamalla copy-painiketta. Painike on oletuksena pois käytöstä, mutta aktivoituu kun taulukkonäkymästä valitaan tarjous. Esikonfiguraatio suoritetaan konfiguraationäkymässä, ja tarjousnäkymä tarjoaa valittavaksi vain vaihtoehtoja jotka on määritelty kokoonpanoon kuuluvaksi konfiguraationäkymässä. Tyypillinen käyttötapaus voisi olla esim. seuraavanlainen:

1. Tuotesuunnittelija lisää uuden tuoteperheen.
2. Hän avaa tuoteperheen konfiguraationäkymään ja lisää sinne kaksi uutta tuotetta.
3. Hän rakentaa tuoteperheen ominaisuusvalikoiman lisäämällä ominaisuuksia valikon kautta (Kuva 18).
4. Hän määrittelee, mikä ominaisuus on saatavilla missäkin muodossa milläkin tuotteella.
5. Hän tallentaa valmiin kokoonpanon, ja voi halutessaan vielä käydä sen läpi myyjän näkymässä varmistuakseen että kaikki ominaisuudet on asianmukaisesti määritelty.
6. Myyjä valitsee haluamansa tuotteen tuoteperheeltä ja luo siitä uuden tarjouksen.

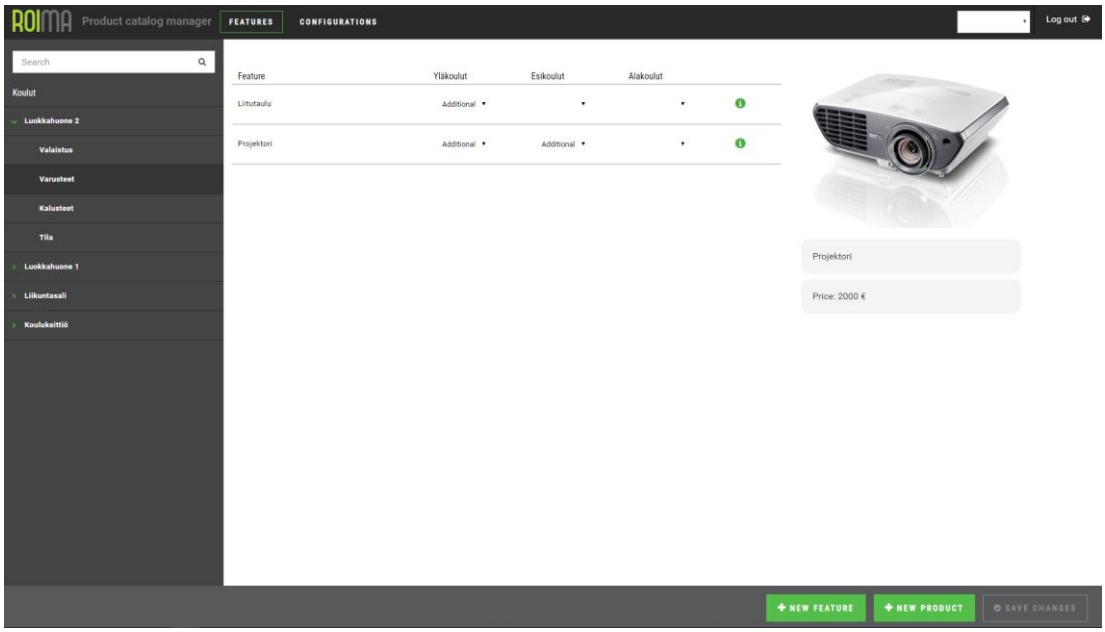
7. Myyjä määrittelee haluamansa kokoonpanon tuotesuunnittelijan tekemästä esikonfiguraatiosta.
8. Myyjä lisää haluamansa alennusprosentit ja varmistaa että tarjouksen loppusumma on asianmukainen.
9. Myyjä tallentaa tarjouksen pdf-versiona seuraavia toimenpiteitä varten.



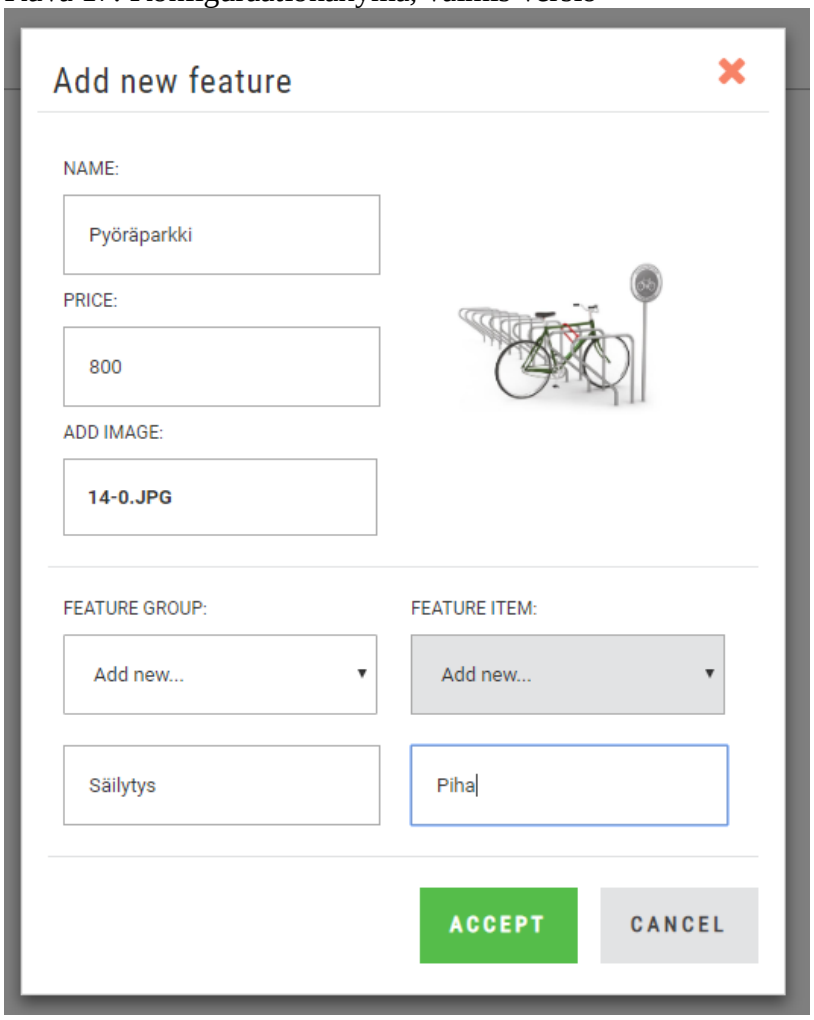
Kuva 15. Päänäkymä



Kuva 16. Myyjän näkymä, valmis versio



Kuva 17. Konfiguraationäkymä, valmis versio



Kuva 18. Dialogi uuden tuoteominaisuuden lisäämiseen.

5.2 Käytettävyyden toteutuminen

Kuten aikaisemmassa kappaleessa käytettävyydestä todettiin, päätettiin käytettävyyden osalta ohjenuoriksi ottaa erityisesti johdonmukaisuus, muistettavien asioiden määrä sekä tehtävään sopivuus. Johdonmukaisuutta saavutettiin elementtien yhdenmukaisella sijoittelulla molempien näkymien välillä, sekä tekemällä valintanapeista värin sekä järjestyksen suhteen yhdenmukaisia.
Muistettavien asioiden määrää rajoitettiin jakamalla kokonaisuus selkeiksi aihealueiksi: tuoteperheiden konfigurointi sekä siihen liittyvät toiminnot sivun vasemmalla laidalla, tarjousten luonti ja hallinnointi sivun oikealla laidalla. Uuden ominaisuuden luonnin yhteydessä tarvittavien tietojen syöttö on koottu yhteen dialogiin, joka antaa varoituksen mikäli tietoja puuttuu, jolloin käyttäjä voi varmistua siitä että tietoja ei jää puuttumaan. Tehtävään sopivuus saavutettiin nappien johdonmukaisella sijoittelulla. Uusien tuoteperheiden, tuotteiden sekä ominaisuuksien luomiseen tarkoitetut napit on sijoitettu paikkoihin, joissa käyttäjä niitä luontaisesti hakisi: uusien tuoteperheiden sekä tarjousten luominen päänäkymään, sekä uusien tuotteiden sekä ominaisuuksien luominen konfiguraationäkymään.

6 YHTEENVETO

Konfiguraattorin päivittäminen oli mielenkiitoinen ja laaja-alainen tehtävä, joka mahdollisti tutustumisen erittäin monipuolisesti web-sovelluksen eri kerroksiin aina tietokannasta sovellusrajapinnan kautta käyttöliittymäkomponentteihin ja tyylimäärittelyihin. Ohjelmointityöhön alkoi työn loppuvaiheessa muodostua rutiinia, josta on ollut korvaamatonta hyötyä muissa työtehtävissä. Suunnittelutyö tutustutti myös toimialaan ja pakotti asettumaan asiakkaan asemaan.

Erityisesti tutuiksi tulivat jQuery, CSS sekä Kendo-käyttöliittymäkomponentit, mutta työ tutustutti kohtuullisesti myös Java-ohjelmointikieleen, MVC-arkkitehtuuriin sekä SQL-kyselyihin, joita tarvittiin kerran jos toisenkin tiedon lisäämisen testaamisen yhteydessä.

LÄHTEET

Keinonen, T. 1998. Viitattu 1.4.2019.

Taideteollisen Korkeakoulun julkaisu A21. Helsinki.

Koski, R. 2015. Hukkaatko euroja huonolla käytettävyydellä? Viitattu 1.4.2019.

https://www.tulos.fi/artikkelit/hukkaatko-euroja-huonolla-kaytettavyydella/

McBeath, VL. 2017. Victorian England. Viitattu 1.4.2019.

https://valmcbeath.com/19th-century-handsworth-soho-manufac-

tory/#.XGq3s82xWUk

Progress 2014. Press Release. Viitattu 1.4.2019.

http://investors.progress.com/news-releases/news-release-details/progress-software-

announces-intent-acquire-telerik?ReleaseID=877501

Progress 2019. Kendo UI. Viitattu 1.4.2019.

https://www.telerik.com/kendo-ui

Reijo Rautauoman säätiö, 2019. Logistiikan Maailma. Viitattu 1.4.2019.

http://www.logistiikanmaailma.fi/logistiikka/tuotanto/prosessien-kehittaminen/

Tiihonen, J. & Soininen, T. 1997. Product Configurators – Information System Sup-

port for Configurable Products. Viitattu 1.4.2019.

Hewson Consulting Group.

The Interaction Design Foundation 2019. Viitattu 1.4.2019.

https://www.interaction-design.org/literature/topics/usability

TutorialsPoint 2019. Viitattu 7.4.2019.

https://www.tutorialspoint.com/ajax/what_is_ajax.htm