jamk.fi

UI-sovelluskehysten vertailu ASP.NET-ympäristössä

Jani Kerttula

Opinnäytetyö
Syyskuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Kerttula, Jani	Opinnäytetyö, AMK	Syyskuu 2018
	Sivumäärä 57	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

UI-sovelluskehysten vertailu ASP.NET-ympäristössä

Tutkinto-ohjelma

Ohjelmistotekniikan tutkinto-ohjelma

Työn ohjaaja(t)

Ari Rantala

Toimeksiantaja(t)

Solteq Oyj

Tiivistelmä

Opinnäytetyössä tehtiin Proof of Concept (PoC), kuinka modernisoida tuotteen teknologioita ja niiden potentiaalia vanhojen menetelmien korvaajina. Samalla tutkittiin vaihtoehtoisia tapoja siirtää dataa sovelluskerrosten välillä. Tavoitteena oli myös kehittää tekijän omaa osaamista sovelluskehittämisen eri osa-aluilla. Taustalla oli olemassa oleva tuote energia-alan sovellusperheestä, jonka sisällä kokeilu toteutettiin. Työssä tehtiin aluksi alkumäärittelyt, suunnitelmat ja työmääräarviot.

Työssä toteutettiin REST-rajapinta käyttäen ASP.NET:n Web API-sovelluskehystä ja erillinen näyttömalli, joka lähetetään käyttöliittymille. Näyttömalli tehtiin noudattaen DTO-mallia ja niistä generoitiin TypeScript-luokat TypeWriter-lisäosaa käyttäen. Käyttöliittymät toteutettiin käyttäen Vue.js- ja React-sovelluskehyksiä. Molemmat käyttöliittymät luotiin noudattaen näiden ominaisia kehitystapoja sekä käyttäen TypeScript-ohjelmointikieltä. Näiden sovelluskehysten välillä käytiin vertailu, kumpi sovelluskehys sopii paremmin kyseiselle tuotteelle ja kehittäjille.

Tuloksena tuli uusi rajapinta, joka hallinnoi tiedonsiirtoa sovelluskerrosten välillä. Olemassa olevasta tietokantamallista tehtiin typistetyt DTO-mallit. Käyttöliittymät tehtiin samalla sivulle rinnakkain. Kaikkien näiden tueksi tehtiin tarvittavia konfiguraatiomuutoksia olemassa oleviin konfiguraatiotiedostoihin.

Vue.js-sovelluskehys sopii paremmin tuotteelle tekijän ja kehitystiimin mielestä. REST-API todettiin toimivaksi, mutta se vaatii vielä jatkokehitystä. DTO-malli ei sovellu tuotteen käyttötarkoitukseen suoraan, vaan vaatii jatkojalostamista. TypeWriter todettiin toimivaksi tietomallin eheyden ylläpitämiseksi.

Avainsanat (asiasanat)

ASP.NET, TypeScript, TypeWriter, C#, Web API 2, REST, React, Vue, DTO

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Kerttula, Jani	Bachelor's thesis	September 2018
		Language of publication: Finnish
	Number of pages: 57	Permission for web publication: x

Title of publication

UI-framework comparison in ASP.NET-environment

Degree programme

Software Engineering

Supervisor(s)

Rantala, Ari

Assigned by

Solteq Oyj

Abstract

The goal of the thesis was to do a Proof of Concept (PoC) on modernization of an existing product in the energy industry, mainly focusing on its technologies and finding alternative ways to transfer data across application layers. One of the goals was also to improve the skillset of the writer regarding the needs of modern web development. The thesis starts with original requirements, plans and workload estimates.

The work consists of a REST-API made with Web API 2-framework, and a separate view model, which was made with DTO-mindset. From these DTOs were generated separate interfaces and classes with the TypeWriter-addon. The user interfaces were made with React- and Vue.js-frameworks, and both used TypeScript-language. Between these two frameworks, a comparison was made regarding the needs of the product and the opinion of the development team.

As a result, an API was made to control the transference of data. The heavily truncated DTO-models were made from the existing database model, only consisting of the relevant data the thesis needed. The user interfaces were made to the same page side to side. To support the changes, some configurations were changed/added to the product.

As a result, Vue was considered superior framework to support the future development of the product. REST-API was considered functional, but that it still needs some more development. The DTO-model was not suitable to the needs of product as is and needs further examination. The TypeWriter-addon was deemed useful at securing the integrity of the data.

Keywords/tags (subjects)

ASP.NET, TypeScript, TypeWriter, C#, Web API 2, REST, React, Vue, DTO

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Sanasto

API

Ohjelmointirajapinta (Application Programming Interface) on ohjelmisto, joka sallii eri ohjelmien keskustelun keskenään.

Bootstrap

Bootstrap on moderni avoimen lähdekoodin tyylikirjasto responsiivisten käyttöliittymien kehittämiseen.

CSS

CSS (Cascading Style Sheets) on yksinkertainen mekanismi tyylimääritysten laadintaan WWW-dokumenteille.

ECMAScript

ECMAScript on European Computer Manufacturers Associationin (ECMA) kehittämä ja standardisoitu JavaScript-syntaksi.

Entity Framework

Entity Framework on “object-relational mapper” (O/RM) .NET-ympäristöille, jota käytetään objektien kääntämiseen tietokantaan.

Entrypoint

Entrypoint on käsite, joka määrittää sovelluksen lähtöpisteen, josta se aloitetaan.

HTML

Hypertext Markup Language on standardisoitu avoin merkintäkieli web-sivujen luontiin.

Import/Export

Nimeämistapa, jolla tuodaan/viedään moduuleja toisiinsa.

Intellisense

Intellisense on yleinen nimitys Visual Studion eri ominaisuuksille, jotka auttavat ohjelmistokehityksessä antamalla lisätietoja ja virheilmoituksia, listaamalla käytettäviä parametrejä ja automaattisesti viimeistelemällä kirjoitettavaa koodia.

Interface

Interface on yleinen nimitys jaetulle rajapinnalle, jolla eri komponentit jakavat tietoja.

JavaScript

HTML:n yhteydessä pääasiassa selainskriptien tekemiseen käytetty kieli, joka suoritetaan HTML-dokumentin näyttämisen yhteydessä selaimen JavaScript-tulkilla eli moottorilla.

LINQ

LINQ (Lanquage Integrated Query) on yhtenäinen kyselysyntaksi, jota voidaan C#- ja VB-kielissä käyttää suorittamaan kyselyjä eri lähteisiin.

OOP

OOP (Object Oriented Programming) on ohjelmointitapa, jossa ratkaisu yritetään tehdä eri objektien yhteistyöllä.

PoC

PoC (Proof of Concept) on kevyempi projektimalli, jossa tutkitaan sovelluksen tai sivuston toimivuutta ja käytettävyyttä.

Razor

Razor on merkintäkieli (markup language), jolla voi upottaa palvelintason koodia web-sivuille.

REST

REST (Representational State Transfer) on arkkitehtuurityyli, jolla standardisoidaan, kuinka eri järjestelmät keskustelevat keskenään.

SFC

SFC (Single File Component) on määrite, jolla määritetään komponentin tulevan vain yhdestä tiedostosta.

SignalR

SignalR on ASP.NET-kirjasto, joka mahdollistaa reaaliaikaisen kommunikaation eri sovelluskerrosten välillä.

Sovelluskehys

Sovelluskehys (framework) on yleinen termi kirjastolle, joka tuo ominaisuuksia ja ohjelinjoja ohjelmiston tuottamiseen. Monet tuovat mukanaan oman syntaksin, jolla koodi kirjoitetaan.

Visual Studio

Visual Studio on Microsoftin kehittämä IDE (Integrated Development Environment), joka tukee monia eri käyttötarkoituksia web-sovelluksista mobiilisovellusten kehittämiseen.

1 Työn lähtökohdat

1.1 Tehtävän kuvaus

Opinnäytetyön tehtävänä oli tehdä kokeilu, miten saataisiin modernisoitua tuotteen arkkitehtuuria ja samalla helpotettua uusien käyttöliittymien luontia asiakkaille. Samalla tarkoituksena oli etsiä vaihtoehtoisia tapoja siirtää tietoa käyttöliittymätason ja palvelintason välillä. Kokeilun tarkoituksena oli selvittää, miten modernit teknologiat saataisiin toimimaan olemassa olevalla tuotteella, ja vertailla eri teknologioiden yhteensopivuutta järjestelmän kanssa.

1.2 Toimeksiantaja

Toimeksiannon tilaajana toimi Solteq Oyj:n Utilities-yksikkö. Solteq Oyj on pohjoismainen toimialariippumaton digitaalisen liiketoiminnan ratkaisuihin erikoistunut toimija. Yritys toteuttaa ratkaisuja pääosin Pohjoismaihin monella eri sektorilla, kuten ajoneuvokauppaan, energiatoimialalle, tukkukauppaan ja valmistavaan teollisuuteen. Yritys toimii viidessä eri maassa, Suomessa, Ruotsissa, Norjassa, Tanskassa ja Puolassa. Utilities-yksikkö on erikoistunut energiatoimialan ratkaisuihin, jonka tuoteperheitä ovat i4U (inPulse4Utilities) sekä inWorks. (Yritys - Solteq n.d).

1.3 Työn kohde

Kokeilun kohteena oli yksi osa-alue i4U-tuoteperheen pääkäyttäjän (Admin) käyttöliittymästä, tarkemmin liittymätilauksen hallinnan ”Tilauksen tiedot” -välilehti.

Tuotteen taustalla toimivat ASP.NET-teknologiat, pääosassa Entity Framework. Tuotteen käyttöliittymä oli tuotettu käyttäen Razor- ja SignalR-kirjastoja ja tyylimäärityksiin oli käytetty Bootstrap-tyylikirjastoa. Koska tuote on todella laaja, sen parissa työskentelee useita tiimejä sen eri osa-alueilla.

1.4 Tavoitteet

Tavoitteena oli vertailla valittuja käyttöliittymätason kehityskehyksiä ja löytää, sopiiko niistä jokin kyseessä olevaan tuotteeseen. Samalla yksi tavoitteista oli kartuttaa tekijän omaa ammattitaitoa ja osaamista tuotekehityksen eri osa-alueilla.

Toimeksiantajan tavoitteena oli löytää tapa, jolla nopeuttaa ja modernisoida käyttöliittymäkehitystä. Tavoitteena oli myös löytää ratkaisu, miten suuria tietomalleja saataisiin pilkottua hallittavampiin osiin. Samalla päästäisiin tutustuttamaan kehitystiimiä erilaisiin tapoihin, miten kehitystä voi tehdä ja löytää mahdollisesti järkeviä suuntia, mihin kehitystä voidaan viedä.

2 Ongelman määrittely

2.1 Alkumäärittely

Aloitusmäärittelyissä käytiin läpi, mitä toimeksiantaja haluaa saada aikaiseksi. Koska kyseessä oli PoC (Proof of Concept), työ rajattiin aluksi kapeaksi. Tästä voidaan halutessa laajentaa, jos työ todetaan kannattavaksi.

Alkuperäisenä ongelmana oli, että järjestelmän tietomallit olivat paisuneet ja niiden hallinta oli vaikeutunut. Käyttöliittymien teko oli tämän vuoksi työlästä. Tietomallien ympärillä oli monenlaista ratkaisua, ja tarve oli yhtenäistää kehitystapoja, jotta saataisiin ketteröitettyä ja helpotettua kehittämistä.

Toimeksiantaja oli jo aikaisemmin tehnyt käyttökokeiluja Vue.js-käyttöliittymäkehyksen parissa ja halusi tarkentaa sen käyttöönottoa ja laajentaa sen käyttökohteita. Toimeksiantaja myös toivoi, että tehtäisiin kokeiluja Web API-kontrollerin kanssa, koska liiketoiminnan tarpeet siirtyä REST-rajapintoihin olivat kasvaneet. Toimeksiantaja myös ilmaisi mielenkiintoaan TypeWriter-lisäosaan Visual Studio -ympäristössä ja pyysi kokeilua myös tähän.

Työlle annettiin toimeksiantajan toimesta yksi kuukausi työaikaa käytettäväksi.

2.2 Työalueen rajaus

Koska tuote itsessään oli todella kattava, päätettiin rajata kokeilu yhteen välilehteen. Työn monien eri osa-alueiden laajuuden takia supistettiin työalue yhteen palaseen tätä näkymää, koska muutoin työmäärät olisivat kasvaneet liian suuriksi työn aikaikkunaan nähden.

Taustalle päätettiin tehdä kokeilu, miten REST-rajapinta toteutettaisiin. Tätä rajapintaa sitten kutsuttaisiin tarvittaessa käyttöliittymältä. Koska yksi ongelma oli paisuneet tietomallit, joiden hakeminen vie aikaa, todettiin, että tarvitsisi pilkkoa tätä tietomallia hallittavimpiin osiin.

3 Suunnittelu

3.1 Työn yleiset vaatimukset

Työssä noudatettiin yrityksen yleisiä vaatimuksia ohjelmistokehityksestä.

3.1.1 Palvelin

Koska tuote käsittelee arkaluontoisia henkilötietoja, palvelintasolla vaatimuksina oli, etteivät ulkopuoliset pääse yhteyteen mihinkään osaan sovellusta, joka tarjoaa näitä tietoja. Palvelut tulee olla saatavilla kirjautuneelle ja tunnistautuneelle käyttäjälle. Tasolla myös vaaditaan, että tiedon eheys tulee tarkistaa. Tietokannasta ei myöskään saa poistaa näitä prosesseja, jotka käsittelevät liittymätilauksia.

Osana työtä toteutettiin REST-rajapinta sovelluksen yhteyteen, jonka kautta saataisiin kutsuttua haluttu tieto. Tämän rajapinnan kautta tuli myös käsitellä pyynnöt muokata näitä tietoja.

3.1.2 Käyttöliittymä

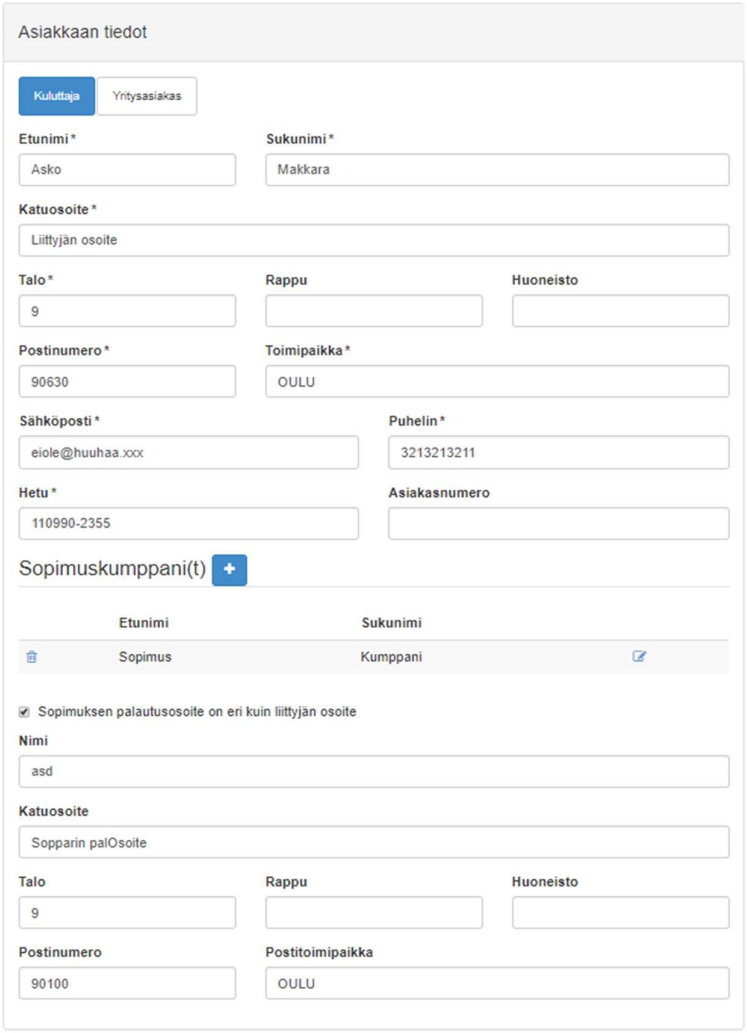
Käyttöliittymän tuli olla taaksepäin yhteensopiva vähintään Internet Explorer 11:een asti. Tämä käytännössä tarkoittaa sitä, että kaiken selaimen päällä olevan JavaScript-koodin tuli olla ECMAScript 5-standardin mukaista. Toteutettavan kokeilun tuli myös olla ainoastaan saatavilla kirjautuneelle ja tunnistautuneelle käyttäjälle.

3.2 Suunnitelma

Työn pääpohjana toimi REST-rajapinta, jonka kautta saatiin siirrettyä tietoa eri kerrosten välillä. Tämän lisäksi tehtiin pari eri versiota käyttöliittymästä eri sovelluskehysten avulla, jotta pystyttiin myös vertailemaan, miten kehitys eri kehysten ympärillä toimii. Täten pystyttiin vertaamaan sekä olemassa olevaan toteutukseen että toiseen mahdolliseen toteutustapaan.

3.2.1 Tietomalli

Selvittäessä mitä tietoa tulee käsitellä, aloitettiin tarkastelu nykyisestä toteutuksesta (ks. kuvio 1).

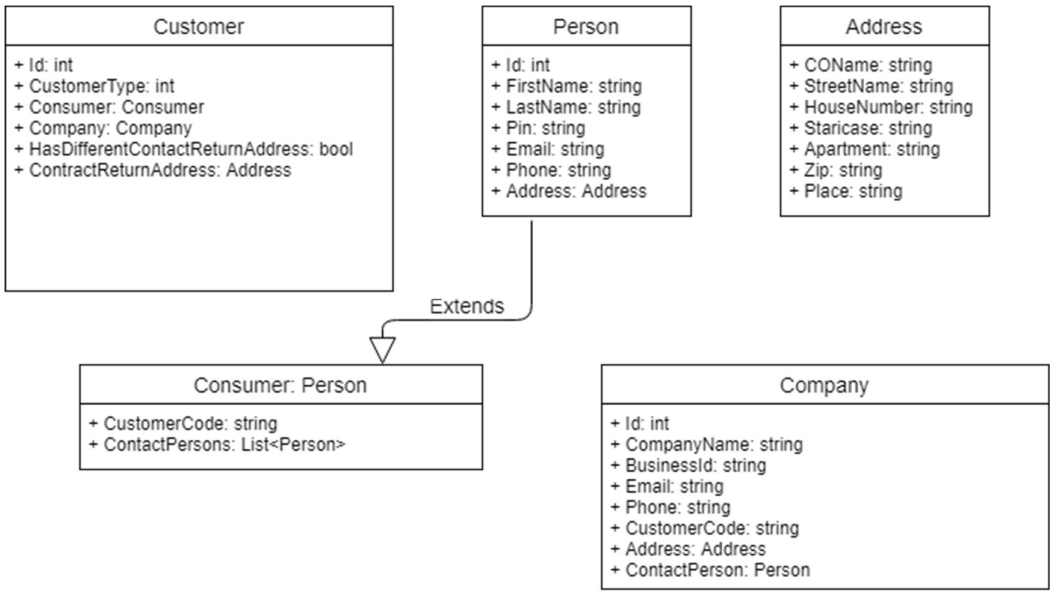


Kuvio 1. Asiakkaan tiedot -paneeli

Kuviosta 1 näkyy, mitä tietoja työn yhteydessä tulee käsitellä. Paneelissa on myös erikseen kohta, jolla voidaan vaihtaa asiakkaan tyyppi (Kuluttaja/Yritysasiakas), joka myös vaihtaa kuviossa 1 näkyvää näkymää. Tietomallia tehdessä tarvitaan seuraavat tietokentät:

- Asiakastyyppi
- Kuluttaja
    - Etunimi
    - Sukunimi
    - Osoitetiedot
        - Katuosoite, talo, rappu, huoneisto, postinumero, toimipaikka
    - Sähköposti
    - Puhelin
    - Henkilötunnus
    - Asiakasnumero
    - Mahdolliset sopimuskumppanit
        - Etunimi, sukunimi, sähköposti, puhelin, henkilötunnus, osoitetiedot
- Yritysasiakas
    - Yrityksen nimi
    - Y-tunnus
    - Yhteyshenkilön tiedot
        - Etunimi, sukunimi, sähköposti, puhelin
    - Osoitetiedot
        - Katuosoite, talo, rappu, huoneisto, postinumero, toimipaikka
    - Sähköposti
    - Puhelin
    - Asiakasnumero
- Sopimuksen palautusosoite
    - Onko eri kuin liittyjän
    - Nimi
    - Osoitetiedot
        - Katuosoite, talo, rappu, huoneisto, postinumero, toimipaikka

Nämä tiedot tuli vähintään saada näyttömalliin. Näistä tiedoista luotaisiin myös Type-Writeria käyttäen käyttöliittymälle interfacet, joilla voitiin kehityksen aikana validoida tietomallin eheyttä. Alustava suunnitelma tietomallin toteutuksesta on esitetty kuviossa 2.



Kuvio 2. UML-kaavio tietomallista

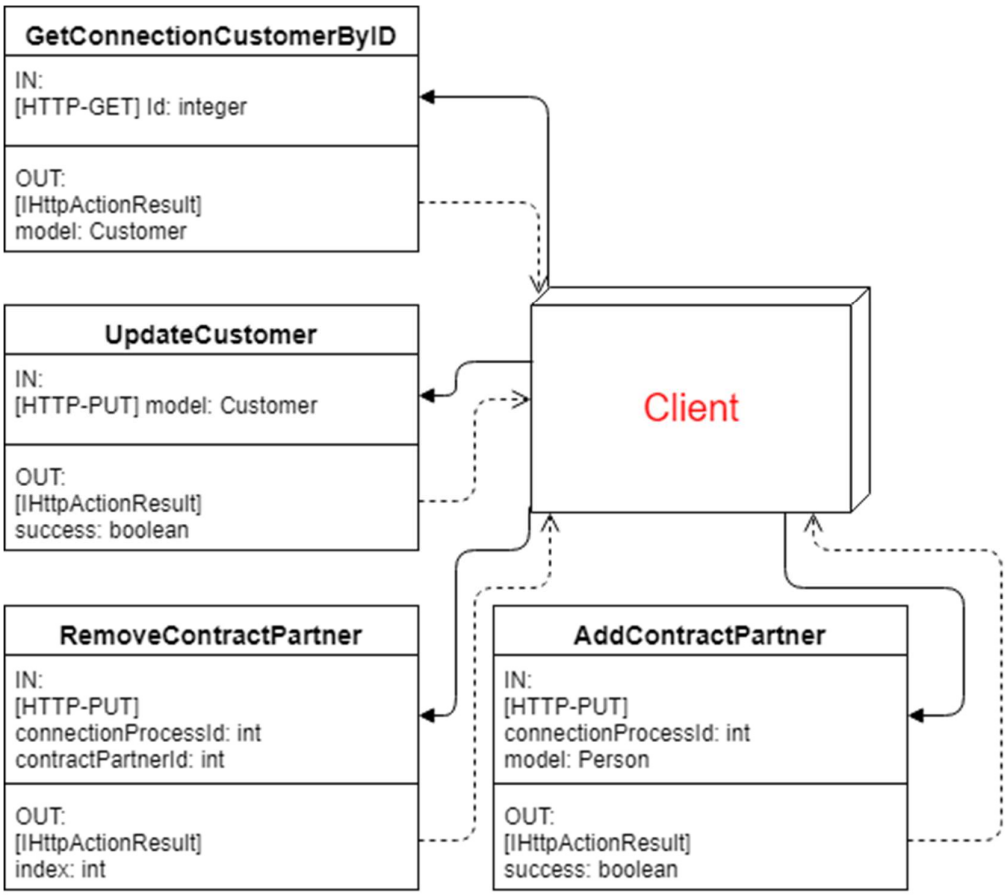
3.2.2 REST-rajapinta

Rajapinnalla tarvitaan eri kutsuja, joilla erotellaan, mitä kutsuja käyttöliittymältä tulee. Ensimmäinen on GET-metodi, jonka avulla saadaan käsiteltävä tieto käyttöliittymälle. Jotta pystyttäisiin muokkaamaan olemassa olevaa tietoa, tarvitaan PUT-metodi, joka käsittelee tiedon päivittämisen takaisin tietokantaan. Tarve muille lisämetodeille selviää tarkastelemalla näyttömallia.

Näyttömallista pääteltiin, että tarvitaan erikseen metodit, joilla hallitaan kuluttaja-asiakkaan sopimuskumppaneita. Koska niiden lisääminen, muokkaaminen tai poistaminen on osa prosessin muokkausta, näitä varten luotaisiin PUT-metodit.

Rajapinnan saatavuus voitaisiin rajoittaa tuotteella jo olemassa olevilla metodeilla.

Kuviossa 3 on esitetty UML-kaaviona suunnitellun rajapinnan toiminnallisuudet.

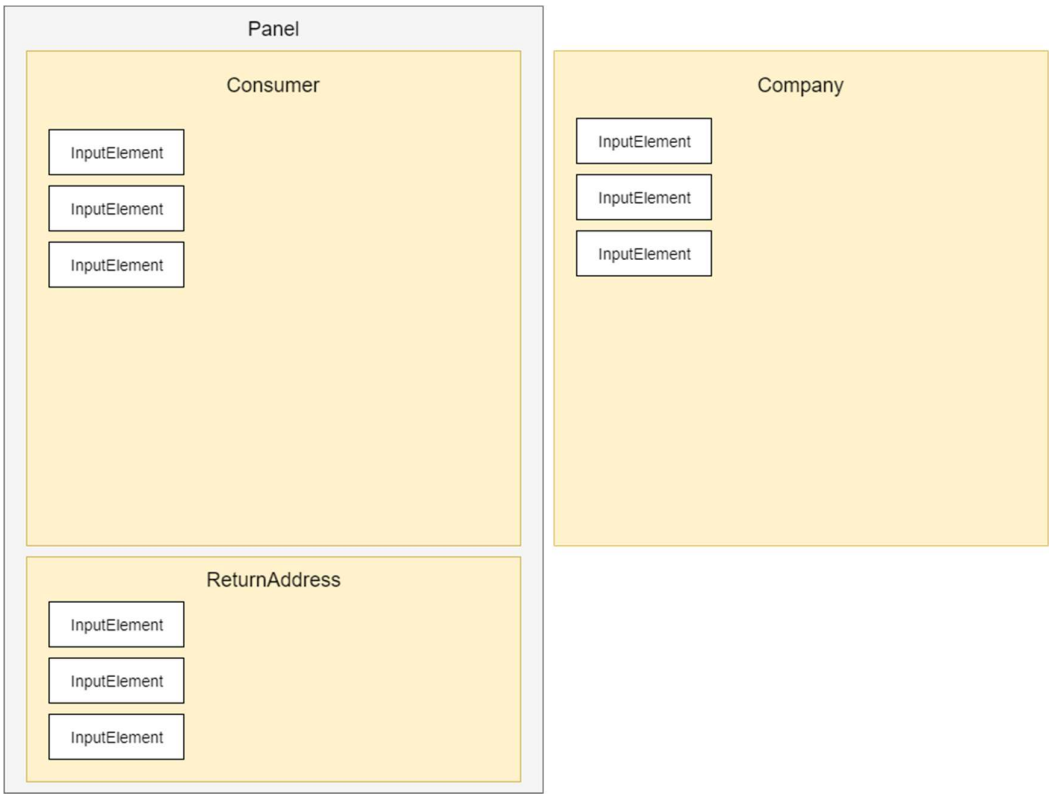


Kuvio 3. REST-toiminnallisuudet

3.2.3 Käyttöliittymä

Käyttöliittymä tulisi olla vastaava kuin olemassa oleva, tyylillisiä muutoksia ei tarvitse tehdä. Tämän toteuttamiseen tulisi käyttää kahta eri sovelluskehystä, jotta voitaisiin toteuttaa vertailu ja arviointi niiden välillä. Näkymästä tulisi saada responsiivinen, ja pyrkiä tehokkuuteen.

Aloittaessa kannattaa aloittaa kokonaiskuvasta ja lisätä toiminnallisuuksia sen jälkeen. Suunnitellun käyttöliittymän jaottelu on esitetty kuviossa 4.



Kuvio 4. Käyttöliittymän jaottelu osiin

3.3 Työmääräarviot

Taulukossa 1 on arvioita, kuinka kauan minkäkin työn osan luontiin menisi. Arviot on tehty henkilötyöpäivinä (htp).

Taulukko 1. Työmääräarviot

ID	*Kuvaus*	*TMA*
T001	Tietomallin yleismallinnus	1 htp
T002	Tietomallin hajauttaminen hallittaviin osiin	1 htp
T003	Tietomallin tiedonhaku kannasta	2-3 htp
T004	Kannan päivitys tietomallista	1-2 htp
T005	REST-API:n konfigurointi	0,5 htp
T006	API-GET -kutsu	1 htp
T007	API-PUT -kutsu	1-2 htp
T008	Sopimuskumppanin muutoshallinta (kokonaisuutena)	2 htp
T009	TypeWriter-lisäosan käyttöönotto	3 htp
T010	1. sovelluskehyksen käyttöönotto	1 htp
T011	1. sovelluskehyksen yleismallinnus	2 htp
T012	1. sovelluskehyksen toiminnallisuudet	2 htp
T013	2. sovelluskehyksen käyttöönotto	1 htp
T014	2. sovelluskehyksen yleismallinnus	2 htp
T015	2. sovelluskehyksen toiminnallisuudet	2 htp

4 Menetelmät

Tässä luvussa esitetään testattujen teknologioiden valintaperusteet. Monet valinnat perustuvat suoraan olemassa olevaan tuotteen taustateknologioihin ja sen rajoituksiin. Tuotteen kehityksessä oli käytössä Visual Studio 2017 sekä muita työkaluja, jotka eivät suoraan liity tähän työhön.

4.1 Palvelin

4.1.1 ASP.NET ja Web API 2 -kontrollerit

ASP.NET Web API -kehys on .NET-kehyksessä toimiva tapa rakentaa API-rajapintoja (Wasson 2017). Tämä tekniikka valittiin, koska tuote itsessään toimii ASP.NET-ympäristössä ja koska tavoitteena oli toteuttaa REST-pohjainen rajapinta, jota käyttää.

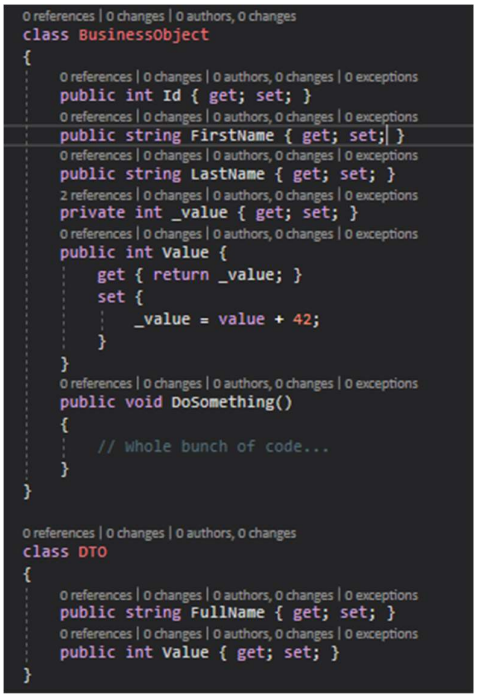
ASP.NET itsessään on ilmainen sovelluskehys, jolla voi luoda web-sovelluksia käyttäen perinteisiä HTML, CSS ja JavaScript-ohjelmointikieliä (Get building n.d). Koska tuotteella käytössä on ASP.NET MVC, pystytään hyödyntämään osaa olemassa olevasta toiminnallisuudesta ja siirtämään vastuuta Web API:lle pala kerrallaan.

4.1.2 DTO-tietomalli

DTO (Data Transfer Object) -tietomalli on kevyt tapa esittää ja siirtää tietoa eri sovelluskerrosten välillä. Wassonin (2014) mukaan DTO:n suurimmat hyödyt verrattuna tietokantamalliin ovat seuraavat:

- Estää ristikkäiset viittaukset
- Piilottaa ominaisuuksia, joita ei haluta näyttää
- Jättää ominaisuuksia pois, joita ei tarvita pienentääkseen lähetyspakettien kokoa
- Litistää tietomallin rakennetta
- Eristää näyttömallin ja tietokantamallin toisistaan.

DTO-mallit sisältävät vain ominaisuuksia, ja niihin ei ole rakennettu sisään mitään funktionalisuuksia, kuten metodeja tai vastaavaa (ks. kuvio 5). DTO:n tarkoitus on siirtää tietoa eri kerrosten välillä. (Koirala 2015.)



```
0 references | 0 changes | 0 authors, 0 changes
class BusinessObject
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int Id { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string FirstName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string LastName { get; set; }
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    private int _value { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int Value {
        get { return _value; }
        set {
            _value = value + 42;
        }
    }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public void DoSomething()
    {
        // Whole bunch of code...
    }
}

0 references | 0 changes | 0 authors, 0 changes
class DTO
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string FullName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int Value { get; set; }
}
```

Kuvio 5. Perinteinen tietomalli vs DTO

DTO-mallit valittiin työhön niiden keveyden ja yksiselitteisyyden vuoksi, ja ne sopivat yleisesti hyvin REST-rajapintaan.

4.2 Käyttöliittymä

4.2.1 Yleistä

Vaihtoehtoja, miten lähteä lähestymään käyttöliittymän puolella teknologioita, on runsaasti. Sovelluskehyksiä käyttöliittymäkehitykseen on useita, kuten AngularJS, React, Vue.js ja Knockout.js. Tuotteessa itsessään löytyy valmiiksi työtä Vue.js- ja AngularJS-sovelluskehysten parista, ja toimeksiantajan toiveesta yksi kokeiluun valittavista kehyksistä oli Vue.js.

Opinnäytetyön tekijän omasta taustasta löytyi osaamista React-kirjaston parissa, joten tämä valittiin vertailtavaksi sovelluskehykseksi Vue.js-kirjaston rinnalle.

API-kutsuihin käytettiin olemassa olevaa jQuery-kirjastoa, jossa on hyvä tuki http-kutsujen luontiin.

4.2.2 TypeScript ja Webpack

TypeScript on Microsoftin kehittämä laajennus JavaScriptiin. TypeScript itsessään ei ole selainluettava kieli, eli se ei suoraan toimi selaimissa ilman kääntämistä. Lauerin (2017c) mukaan TypeScript tuo mukanaan monia etuuksia verrattuna tavalliseen JavaScriptiin, kuten

- Luokat (Classes)
- Moduulit (Modules)
- Tyyppimääritykset (Types)
- Kaikki ECMAScript toiminnallisuudet uusimmista versioista.

TypeScriptin avulla voidaan siis kirjoittaa tuttavallista OOP-tyylistä (Object Oriented Programming) koodia. Kuvion 6 mukainen luokka käännetään halutulla kääntäjällä myöhemmin JavaScript-koodiksi (ks. kuvio 7). TypeScript otettiin käyttöön yrityksen linjausten puolesta, joissa toivotaan kaiken uuden käyttöliittymätason koodin olevan TypeScriptiä.

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

Kuvio 6. TypeScript-luokka

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
```

Kuvio 7. Kuvion 6 luokka käännetty JavaScriptiksi

Miten sitten voidaan TypeScript kääntää JavaScriptiksi? Tässä päädyttiin käyttämään Webpack-paketoijaa (bundler).

Webpack on moduulien paketointityökalu, jolla pystytään paketoimaan koodi ja kääntämään se haluttuun muotoon. Webpackin yleisin käyttötarkoitus on kääntää ja paketoida JavaScript ja CSS-koodia, niiden eri muodoissa. (What Is Webpack n.d.)

Vaikka monia muitakin paketoijia löytyy, tuotteelle oli jo aikaisemmin tehty Webpack-konfiguraatioita, joita pystyttiin työssä hyödyntämään.

4.2.3 Vue.js

Vue.js on progressiivinen sovelluskehys käyttöliittymien luontiin. Toisin kuin muut samankaltaiset sovelluskehykset Vue.js on suunniteltu inkrementaalisesti soviteltavaksi projekteihin. (Introduction n.d.)

Vue.js:stä löytyy myös TypeScript-tuki. Kuviossa 8 esitellään, kuinka Vue.js:llä voidaan kirjoittaa komponentteja TypeScript-laajennusta hyödyntäen.

```
import Vue from 'vue'
import Component from 'vue-class-component'

// The @Component decorator indicates the class is a Vue component
@Component({
  // All component options are allowed in here
  template: '<button @click="onClick">Click!</button>'
})
export default class MyComponent extends Vue {
  // Initial data can be declared as instance properties
  message: string = 'Hello!'

  // Component methods can be declared as instance methods
  onClick (): void {
    window.alert(this.message)
  }
}
```

Kuvio 8. Vue-komponentti TypeScriptillä

Vue.js tukee myös SFC-komponentteja (Single File Component) sekä vastuunjakoa tämänkin sisällä (ks. kuvio 9). Tapa, jolla tässä työssä kehittiin Vue.js-komponentteja, on SFC, josta TypeScript-koodi irrotetaan erilliseen tiedostoon. Tällä saadaan saavutettua Visual Studion täysi tuki TypeScriptiin.

Kuvio 9. SFC-komponentti Vue.js:llä

4.2.4 React

React on Facebookin kehittämä JavaScript-kirjasto käyttöliittymien kehittämiseen. Reactia käytetään yleisesti JSX-muotoisten tiedostojen kautta SFC-komponenttien luontiin. JSX on syntaksilaajennus perinteisiin JS-tiedostoihin lisäten tuen html-elementtien syöttöön suoraan JavaScriptin sekaan. Tämä sallii html-elementtien hallinnan suoraan JavaScriptin kautta (ks. kuvio 10).

React tarjoaa monia sisäänrakennettuja toiminnallisuuksia ”out-of-the-box”, ja on ollut yksi suosituimmista sovelluskehyksistä moderneissa arkkitehtuureissa.

```
import React from 'react'
import ReactDOM from 'react-dom'

class Esimerkki extends React.Component {
    state = {
        value1: 3,
        value2: "Moikka"
    }

    render() {
        const { value1, value2 } = this.state;
        return (
            <div>{value2}, arvosi on {value1}</div>
        );
    }
}
```

Kuvio 10. React-luokka

4.2.5 jQuery

jQuery on nopea, kevyt ja toiminnallinen JavaScript-kirjasto, joka tarjoaa monipuolisia tapoja hallita HTML-dokumentteja ja lähettää http-kutsuja (What is jQuery n.d). Työssä käytettiin jQueryn .ajax()-toiminnallisuutta pääasiassa, mutta tuotteessa itsessään löytyy paljon erilaisia tapoja, miten jQueryä on käytetty.

```
1 $.ajax({
2   method: "POST",
3   url: "some.php",
4   data: { name: "John", location: "Boston" }
5 })
6   .done(function( msg ) {
7     alert( "Data Saved: " + msg );
8   });
```

Kuvio 11. Yksinkertainen Ajax-kutsu

4.2.6 TypeWriter

TypeWriter on Visual Studion lisäosa, jonka avulla voi generoida TypeScript-tiedostoja C#-koodista. Tämä tapahtuu TypeScript Template-tiedoston (.tst) kautta, ja generoitu koodi päivittyy automaattisesti, kun lähteenä käytettävä koodi muuttuu. (TypeWriter n.d.)

TypeWriterilla voidaan siis generoida mm. C#-luokista TypeScript-interfaceja, luokkia yms. Tämä mahdollistaa näyttömallien tyypittämisen suoraan TypeScript-koodissa, jota käytetään käyttöliittymien luontiin.

TypeWriter otettiin kokeiluun työssä kehitystiimin toiveesta ja mielenkiinnosta koodia generoiviin mahdollisuuksiin.

Models.tst | CustomerModel.cs | CustomerModel.ts

```
module App { $Classes(*Model)[
    export class $Name { $Properties[
        public $name: $Type;]
    }]
}
```

Kuvio 12. TypeWriter-template

Models.tst | CustomerModel.cs | CustomerModel.ts

```
using System;
using System.Collections.Generic;

namespace Demo
{
    public class CustomerModel
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public ICollection<OrderModel> Orders { get; set; }
    }
}
```

Kuvio 13. TypeWriter-esimerkissä käytetty luokka

Models.tst CustomerModel.cs **CustomerModel.ts**

```
module App {
    export class CustomerModel {
        public id: number;
        public name: string;
        public orders: OrderModel[];
    }
}
```

Kuvio 14. TypeWriterilla generoitu koodi

5 Toteutus

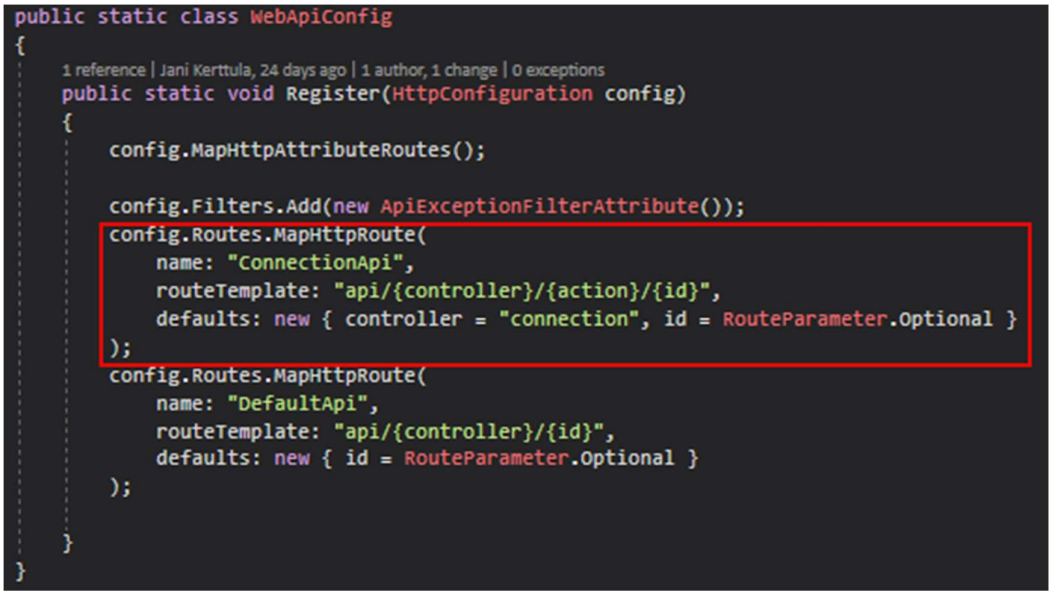
5.1 Palvelin

5.1.1 Yleistä

Ennen kuin päästiin toiminnalliseen osuuteen, tarvittiin ensin muutamia konfiguraatiomuutoksia. Ensin tarvittiin muutoksia WebApiConfig.cs nimiseen tiedostoon. Tiedostoa kutsutaan järjestelmää käynnistäessä, ja tiedoston avulla pystyttiin määrittämään, mistä REST-rajapintaa pystytään kutsumaan.

Kuviossa 15 punaisella ympyröity kohta on lisätty olemassa olevaan konfiguraatioon. Tämän konfiguraation avulla voidaan ohjata ConnectionController-kontrollerin kutsut kuuntelemaan ”api/connection/httpmethod/”-aluetta. {action} -nimitys viittaa kontrollerilla käytettäviin metodeihin.

Tarvittiin myös muokkauksia tsconfig.json -tiedostoon. Projektissa ennestään oli toteutettu konfiguraatiot Vue.js -sovelluskehystä varten, joten tarve oli lisätä tuki React-sovelluskehykselle.



```
public static class WebApiConfig
{
    1 reference | Jani Kerttula, 24 days ago | 1 author, 1 change | 0 exceptions
    public static void Register(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();

        config.Filters.Add(new ApiExceptionFilterAttribute());
        config.Routes.MapHttpRoute(
            name: "ConnectionApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { controller = "connection", id = RouteParameter.Optional }
        );
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

    }
}
```

Kuvio 15. WebApiConfig.cs

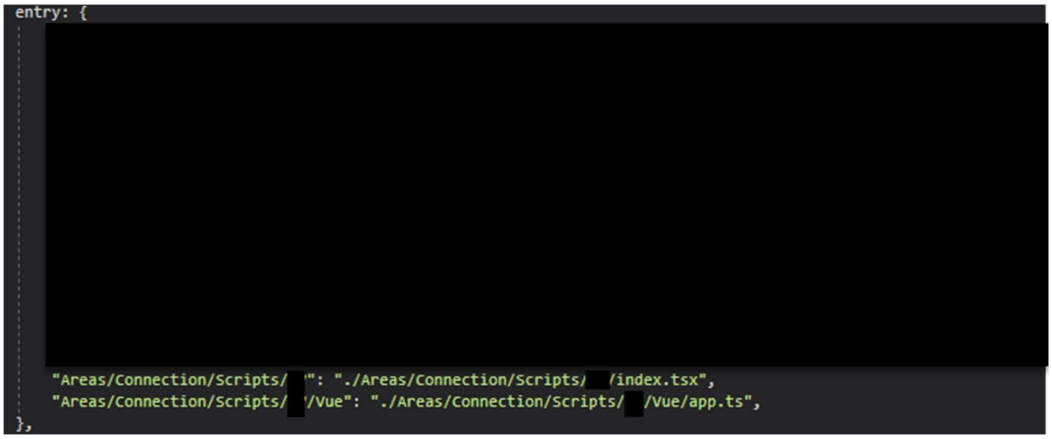
Kuviossa 16 punaisella ympyröidyt kohdat lisättiin tiedostoon, jotta saataisiin TypeScript-tuki toimimaan myös React:n kanssa. ”allowSyntheticDefaultImports” tuo tuen synteettisille importeille, jolla määritetään tietyt importit ulkoisiksi. Tämä estää TypeScriptiä hakemasta virheenkorjausta näille importeille. ”baseUrl” ja ”paths” tuo TypeScriptille tuen ymmärtää, mistä haetaan myöhemmin tässä työssä tämän tiedoston sisällön.

Myös olemassa olevaan webpack.config.json-tiedostoon tarvittiin muutoksia uusien tiedostojen myötä. Tärkeimpinä muutoksina oli tuoda entry-objektin sisälle uudet sovellusten aloituspisteet (entrypoint) (ks. kuvio 17) ja lisätä .tsx-tiedostoille säännöt.

Lisäksi määritettiin, että tietyt importit jätetään pakkauksen ulkopuolelle. Näihin kuuluivat React- ja ReactDOM-kirjastot, jotka haetaan erikseen html-tiedostossa (ks. kuvio 18).

```
{
    "compileOnSave": true,
    "compilerOptions": {
      "target": "es5",
      "module": "es6",
      "moduleResolution": "node",
      "sourceMap": true,
      "emitDecoratorMetadata": true,
      "experimentalDecorators": true,
      "allowSyntheticDefaultImports": true,
      "removeComments": true,
      "noImplicitAny": false,
      "noEmitOnError": false,
      "typeRoots": [
        "Scripts/typings",
        "node_modules/@types"
      ],
      "lib": [
        "es5",
        "es6",
        "dom"
      ],
      "jsx": "react",
      "baseUrl": "",
      "paths": {
        "components": [ "Scripts/react/common-components/test.tsx" ]
      }
    },
    "include": [
      "./Areas/**/*",
      "./Scripts/**/*"
    ],
    "exclude": [
      "obj",
      "bin",
      "SqlServerTypes",
      "App_Data"
    ]
}
```

Kuvio 16. Tsconfig.json



```
entry: {
    "Areas/Connection/Scripts/  ": "./Areas/Connection/Scripts/  /index.tsx",
    "Areas/Connection/Scripts/  '/Vue": "./Areas/Connection/Scripts/  /Vue/app.ts",
},
```

Kuvio 17. Lisäykset entry-objektiin webpack.config.json -tiedostossa

```
externals: {
    "react": "React",
    "react-dom": "ReactDOM"
},
```

Kuvio 18. Kirjastojen jättö ulkopuolelle pakkauksesta

Lisäksi lisättiin sääntö Webpackille, mihin viitataan tietyillä viittauksilla, että Webpack osaa hakea moduulit sieltä. Esimerkkinä toimi text.tsx, joka olisi vaatinut ilman näitä muutoksia vastaavan import-kutsun:

- *import { Module } from '../../../../../Scripts/react/common-components/test'*

Muutosten avulla moduulia voidaan kutsua vastaavasti:

- *import { Module } from 'components'*

Kuviossa 19 näytetään, miten tämä toiminnallisuus saavutetaan.

```
resolve: {
    extensions: ['.ts', '.tsx', '.js', '.vue', '.json'],
    alias: {
        'vue$': 'vue/dist/vue.esm.js',
        components: path.join(__dirname, "Scripts/react/common-components/test.tsx")
    }
},
```

Kuvio 19. Alias-määritys webpack.config.json -tiedostossa

5.1.2 Tietomallit

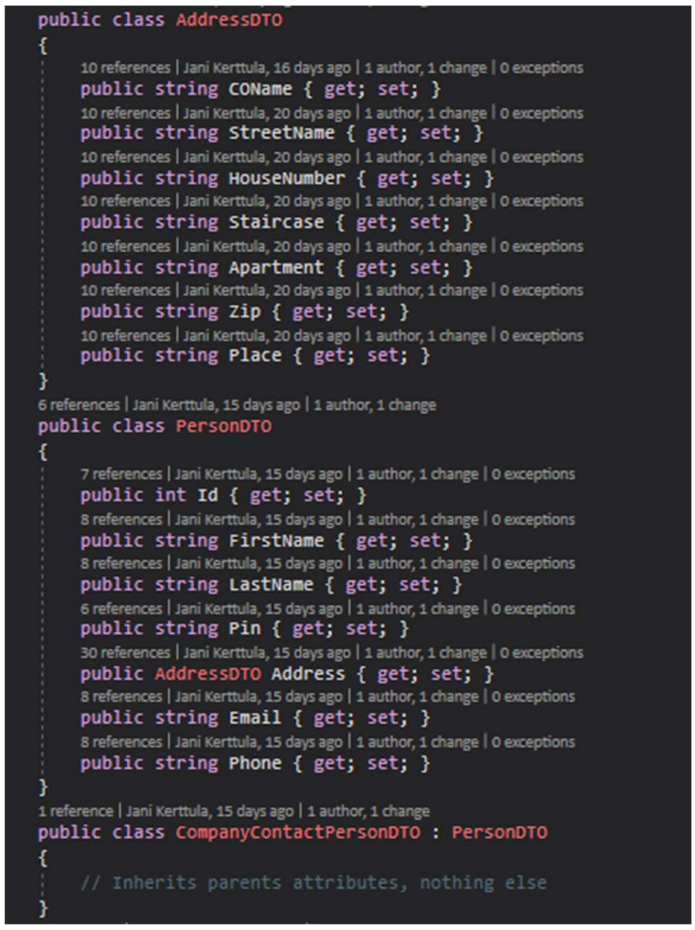
Tietomallien toteuttaminen oli suhteellisen suoraviivaista, käytännössä oli tarve vain kirjoittaa luvussa 3.2.1 määritelty tietomalli C#-luokiksi. Kuviossa 20 on esitetty toteutunut ConnectionCustomerDTO-luokka.

Kuvio 20. ConnectionCustomerDTO

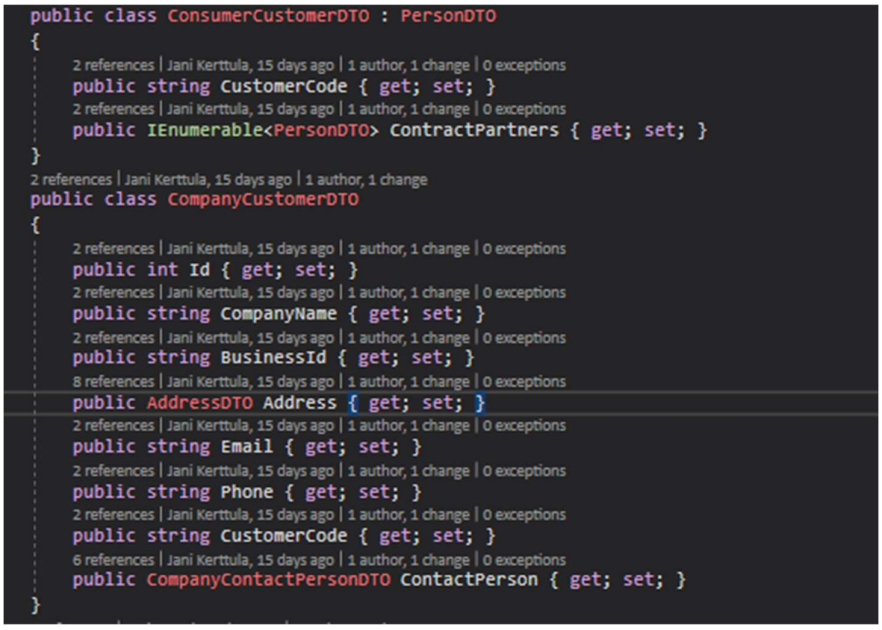
Kuviossa 20 huomattavaa on DataAnnotations-lisäykset ominaisuuksille. Näiden avulla, kun API:lla saadaan tietoa takaisin päin, voidaan verrata, että tieto täyttää ehdot, joita on asetettu kyseiselle luokalle ja se ominaisuuksille. Työssä on käytetty ainoastaan Required-annotaatiota testaustarkoituksessa.

Kuviossa 21 erikoishuomatuksena on CompanyContactPersonDTO, joka ainoastaan perii PersonDTO:n ominaisuudet. Tämä on toteutettu sen takia, vaikka CompanyContactPersonDTO tarvitsee vähemmän tietoja kuin PersonDTO, voimme silti käyttää olemassa olevaa luokkaa koska tiedot ovat kuitenkin samat.

ConsumerCustomerDTO puolestaan laajentaa PersonDTO-luokkaa, mutta käyttää muuten samoja ominaisuuksia kuin PersonDTO (ks. kuvio 22).



Kuvio 21. AddressDTO ja PersonDTO



Kuvio 22. ConsumerCustomerDTO ja CompanyCustomerDTO

5.1.3 REST-API

REST-rajapinnalle luotiin neljä eri metodia: GetCustomerById, UpdateCustomer, RemoveContractPartner ja AddContractPartner. Kaikkiin metodeihin on tehty ”try-catch”-kehys, ja virhetilanteessa kutsujalle palautetaan virheilmoitus.

Pääsy rajapinnalle rajattiin käyttämällä valmiiksi konfiguroitua Authorize-attribuuttia, jolle voidaan määrittää, mitkä käyttäjäryhmät pääsevät normaalisti kiinni rajapintaan. Kirjautumattomalle käyttäjälle rajapinta palauttaa 401-virhekoodin, joka ilmoittaa, että käyttäjä ei ole oikeutettu pääsemään rajapintaan.

GetCustomerById-metodi vaatii, että sille tuodaan Id-parametri, jolla haetaan haluttu prosessi. Tämän jälkeen metodi muuttaa haetun Entity-objektin ConnectionCustomerDTO:ksi, ja palauttaa sen kutsun lähettäjälle. Tämä on toteutettu käyttämällä C#:n LINQ -toimintoa, joka hakee halutusta kontekstista halutuilla parametreillä tiedon, joka voidaan samassa yhteydessä muuttaa DTO:ksi.

UpdateCustomer toimi hieman eri tavalla. Metodille lähetetään PUT-pyyntönä kokonainen DTO, joka validoidaan ja yritetään kääntää oikealle Entity-objektille. Tämän jälkeen muutokset tallennetaan Entity-objektiin, joka huolehtii muutosten päivittämisen tietokantaan. Lähetettävä data tulee tulla JSON-muotoisena stringinä ja se käännetään samassa yhteydessä DTO:ksi. Tässä on huomattavaa se, että kääntäjä yrittää kääntää kaikenmuotoisen tiedon, jota sille lähetetään, DTO:ksi, jolloin metodille voidaan lähettää käytännössä mitä tahansa. Model.IsValid-metodille voidaan asettaa ehtoja DataAnnotations-moduulin kautta, jolloin tietomallin eheys voidaan todentaa.

RemoveContractPartner toimi myös hieman eri tavalla. Metodille tulee lähettää kaksi integer-muotoista arvoa, connectionProcessId ja contractPartnerId. Nämä tulee pakata ennen lähetystä JSON-objektiksi, jotta kutsu pystytään parsimaan. Tätä varten tuli myös tarve luoda RemoveContractPartnerRequest-luokka, joka sitoo lähetettävän datan luettavaan muotoon. Jos kaikki ehdot täyttyvät, niin metodi poistaa prosessilta sopimuskumppanin contractPartnerId:n perusteella.

AddContractPartner pyytää myös kahta parametria, connectionProcessId:n ja PersonDTO:n. ConnectionProcessId:n avulla valitaan oikea prosessi, ja sille lisätään PersonDTO:n sisällä oleva data Entity-objektiin. Tätäkin varten tarvittiin erillinen AddContractPartnerRequest-luokka, ja sitoo parametrit keskenään toisiinsa.

Liite 1 sisältää toteutetun kontrollerin.

5.1.4 TypeWriter

Työtä varten tehtiin TypeScript Template, joka muuntaa DTO-luokat TypeScript interfaciksi sekä tekee niistä rakennusfunktiot (constructor function).

```
$Classes(c => !c.Name.StartsWith("DTO") && c.Name.Contains("DTO"))[
function $Name() {
    return {
        //$BaseClass[$Name]
        $BaseClass[$Properties[$Name: $DetermineValue,
        ]]
        //$Name
        $Properties[$Name: $DetermineValue,
        ]
    }
}]

$Classes(c => !c.Name.StartsWith("DTO") && c.Name.Contains("DTO"))[
interface I$Name { $BaseClass[$Properties[ $Name: $FilterType, ]] $Properties[ $Name: $FilterType; ] }
]

export {
    $Classes(c => !c.Name.StartsWith("DTO") && c.Name.Contains("DTO"))[$Name,
    ]
    $Classes(c => !c.Name.StartsWith("DTO") && c.Name.Contains("DTO"))[I$Name,
    ]
}
```

Kuvio 23. TypeWriterin toteutunut koodi

Kuviossa 23 ylimpänä näkyy, miten muodostetaan rakennusfunktiot, keskellä interfacet, ja lopussa nämä exportataan, jotta niitä voidaan käyttää moduulimaisesti käyttöliittymätasolla. TypeWriterilla on omia attribuutteja, jotka tunnistaa $-alkumerkistä. Hakasulkeilla määritetään, kun halutaan hakea attribuutin sisällä olevia attribuutteja.

$Classes-attribuutti määrittää, mitä C#-luokkia sisällytetään generointiin. TypeWriterilla on myös muitakin samanlaisia attribuutteja, joilla voidaan käsitellä esim. Enum-tyyppejä. Työssä määritettiin, että tarvitaan vain työssä tehdyt DTO-luokat, ja LINQ-operaattorin kautta suoritetaan suodatus niihin.

$Name-attribuutti palauttaa suoraan kyseisen instanssin nimen. $BaseClass-attribuutti puolestaan palauttaa perittävän luokan nimen. $Properties-attribuutti palauttaa luokan ominaisuudet, ja tätä käytetään hakasulkeiden kanssa, jotta käydään jokainen ominaisuus läpi.

TypeWriterilla on myös mahdollista luoda erillisiä funktioita, joita työssä on toteutettu kaksi: $FilterType ja $DetermineValue (ks. kuvio 24).

```
${
    // Enable extension methods by adding using Typewriter.Extensions.*
    using Typewriter.Extensions.Types;

    string FilterType(Property property)
    {
        if (property.Type.Name.Contains("DTO")) return "I" + property.Type.Name;
        else return property.Type.Name;
    }
    string DetermineValue(Property property) {
        if (property.Type.Name.Contains("DTO")) {
            if(property.Type.Name.Contains("[]"))
                return "new Array<I" + property.Type.Name.Substring(0,property.Type.Name.IndexOf("[")) + ">()";
            return "" + property.Type.Name + "()";
        }
        else if (property.Type.ToString().Equals("string"))
            return "\"\"";
        else if (property.Type.ToString().Equals("number"))
            return "0";
        else return "null";
    }
}
```

Kuvio 24. TypeWriterin kustomoidut funktiot

$FilterType-funktio muokkaa annettavaa tyyppiä, ja jos ominaisuuden tyyppi normaalisti olisi DTO-muotoinen, muutetaan se osoittamaan generoitavaan interfaceen.

$DetermineValue-funktio puolestaan on käytössä rakennusfunktioilla, ja sillä annetaan oikea arvo ominaisuuden tyypistä riippuen rakennusfunktioille.

Liittestä 2 löytyy lopputulos, mitä TypeWriter generoi.

5.2 Käyttöliittymä

5.2.1 Yleistä

Ennen kuin voidaan edetä itse sovelluskehyksiin, tarvittiin hieman valmisteluja. Ensinnäkin tarvittiin lisätä juurielementit html-dokumenttiin, joihin sovelluskehykset sitten luovat sisällön (ks. kuvio 25). Toisekseen lisättiin riippuvuudet, ja pakatut sovellukset, jotka Webpack loi (ks. kuvio 26).

```
<div class="row">
    <div id="root" data-id="@ViewBag.Id"></div>
    <div id="vue"></div>
</div>
```

Kuvio 25. Juurielementit html-tiedostoon

Kuviossa 25 myös huomattavaa on, että työssä tuodaan liittymäprosessin Id-arvon Razorin avulla suoraan juurielementtiin arvoksi, jota voidaan sovelluskehyksissä hakea käyttäen jQueryn .data()-metodia.

```
<script src="@Url.GetReact()"></script>
<script src="@Url.GetReactDOM()"></script>
<script src="@Url.GetFile("~/Areas/Connection/Scripts/[redacted]app.js")" type="text/javascript"></script>
<script src="@Url.GetFile("~/Areas/Connection/Scripts/[redacted]/Vueapp.js")"></script>
```

Kuvio 26. Riippuvuudet html-tiedostoon

Kuviossa 26 huomattavaa on, että Reactin kirjastot haetaan käyttäen System.Web.Helpers.UrlExtensions-metodin kautta, jonne tehtiin tarvittavat lisäykset (ks. kuvio 27).

```
204
205         /// <summary>
206         /// Get url to React dependency, checks for TestMode
207         /// </summary>
208         /// <returns>React-dependency URL</returns>
            1 reference | Jani Kerttula, 25 days ago | 1 author, 1 change | 0 exceptions
209         public static string GetReact(this UrlHelper helper)
210         {
211             bool IsDev = ConfigurationManager.AppSettings["TEST_MODE"] == "true";
212             if (IsDev)
213             {
214                 return "/Scripts/react/react.development.js";
215
216             } else
217             {
218                 return "/Scripts/react/react.production.min.js";
219             }
220         }
221         /// <summary>
222         /// Get url to ReactDOM dependency, checks for TestMode
223         /// </summary>
224         /// <returns>ReactDOM-dependency URL</returns>
            1 reference | Jani Kerttula, 25 days ago | 1 author, 1 change | 0 exceptions
225         public static string GetReactDOM(this UrlHelper helper)
226         {
227             bool IsDev = ConfigurationManager.AppSettings["TEST_MODE"] == "true";
228             if (IsDev)
229             {
230                 return "/Scripts/react/react-dom.development.js";
231
232             }
233             else
234             {
235                 return "/Scripts/react/react-dom.production.min.js";
236             }
237         }
```

Kuvio 27. System.Web.Helpers.UrlExtensions lisäykset

Kuvat toteutuneesta käyttöliittymästä löytyvät liitteestä 3.

5.2.2 React

React-sovellus on toteutettu ajatuksella, joka esitettiin luvussa 3.2.3. Pääosassa on TestApp-niminen komponentti (ks. kuvio 28), joka hallinnoi tiedon muutoksia ja lähettämistä. Tämän sisällä oli useampi eri osia, jotka toteutettiin ”Stateless Component”-periaatteella. Näitä olivat Consumer-, Company-, ContractPartnerModal- ja HasDifferentContactAddress-komponentit.

```
210  class TestApp extends Component<
211      {
212          /* PropTypes go here */
213      },
214      {
215          /* StateTypes come here */
216          data: IConnectionCustomerDTO,
217          readOnly: boolean,
218          contractPartner: IPersonDTO,
219          cpReadOnly: boolean,
220          backup: IConnectionCustomerDTO
221      }> {
222      state = {
223          data: ConnectionCustomerDTO(),
224          readOnly: true,
225          contractPartner: PersonDTO(),
226          cpReadOnly: true,
227          backup: null
228      }
229
230      componentDidMount() {
231          this.getDataFromApi();
232      }
233
```

Kuvio 28. TestApp-komponentin määritys

TestApp-komponentille määritettiin aluksi, minkä tyyppisiä kukin komponentin state-tilan ominaisuus on. Tähän käytettiin TypeWriterilla generoituja interfaceja, jotka importattiin tiedostoon. Komponentille annettiin myös yksi lifecycle-metodi, componentDidMount, joka on Reactin oma metodi, jota kutsutaan, kun komponentti ensimmäisen kerran piirretään näytölle. Tämä metodi kutsuu getDataFromApi-metodia, joka lähettää GET-pyynnön aikaisemmin tehdylle rajapinnalle (ks. kuvio 29).

```
getDataFromApi = () => {
    let self = this;
    $.get('/api/connection/getcustomerbyid/' + connectionProcessId, function (data) {
        console.log("react-GET: ", data);
        self.setState({ data: data });
    });
}
```

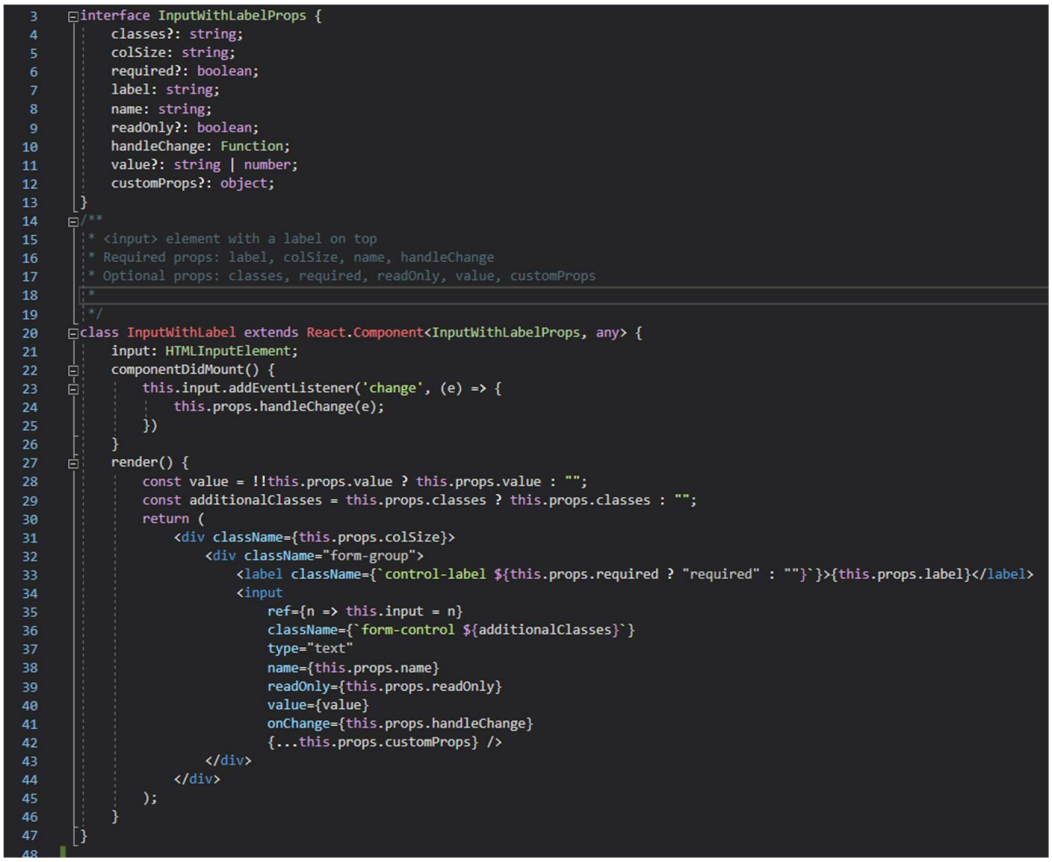
Kuvio 29. GetDataFromApi-metodi

Staten muokkaamiseen tehtiin erikseen oma getNewState-metodi, joka otti vastaan pohjan (base), polun (path) ja arvon (value). Pohja on objekti, jota muokataan, polku on tietty ominaisuus, jota halutaan muuttaa ja otetaan string-taulukkona vastaan, ja arvo on muutettava arvo. Metodi palauttaa uuden base-objektin, johon haluttu arvo on muutettu (ks. kuvio 30).

```
242    getNewState = (base: any, path: Array<string>, value: string | number) => {
243        let depth = path.length;
244        base = {
245            ...base,
246            [path[0]]: depth === 1 ? value : {
247                ...base[path[0]],
248                [path[1]]: depth === 2 ? value : {
249                    ...base[path[0]][path[1]],
250                    [path[2]]: depth === 3 ? value : {
251                        ...base[path[0]][path[1]][path[2]],
252                        [path[3]]: value
253                    }
254                }
255            }
256        };
257        return base;
258    }
```

Kuvio 30. GetNewState-metodi

Työssä luotiin myös common-components-kansio, jonne tehtiin test.tsx-tiedosto. Tänne luotiin yleisesti käytettyjä komponentteja, joita voidaan käyttää uudestaan eri paikoissa. Tärkein oli InputWithLabel-komponentti (ks. kuvio 31), jota käyttämällä saatiin luotua tarvittavat input-elementit lomakkeelle.



```
interface InputWithLabelProps {
    classes?: string;
    colSize: string;
    required?: boolean;
    label: string;
    name: string;
    readOnly?: boolean;
    handleChange: Function;
    value?: string | number;
    customProps?: object;
}
/**
 * <input> element with a label on top
 * Required props: label, colSize, name, handleChange
 * Optional props: classes, required, readOnly, value, customProps
 *
 */
class InputWithLabel extends React.Component<InputWithLabelProps, any> {
    input: HTMLInputElement;
    componentDidMount() {
        this.input.addEventListener('change', (e) => {
            this.props.handleChange(e);
        })
    }
    render() {
        const value = !!this.props.value ? this.props.value : "";
        const additionalClasses = this.props.classes ? this.props.classes : "";
        return (
            <div className={this.props.colSize}>
                <div className="form-group">
                    <label className={`control-label ${this.props.required ? "required" : ""}`}>{this.props.label}</label>
                    <input
                        ref={n => this.input = n}
                        className={`form-control ${additionalClasses}`}
                        type="text"
                        name={this.props.name}
                        readOnly={this.props.readOnly}
                        value={value}
                        onChange={this.props.handleChange}
                        {...this.props.customProps} />
                </div>
            </div>
        );
    }
}
```

Kuvio 31. InputWithLabel-komponentti

Komponentille annettiin componentDidMount-elinkaarimetodi, joka lisää sille kuuntelijan. Tämä kuuntelija tarvittiin, kun Reactin ulkopuolelta tuli muutoksia kenttiin koodissa. Tässä tapauksessa ainut oli jQueryUI:n Autocomplete-lisäosa, jota käytettiin osoitetietojen hakuun ja syöttöön halutuilla paikoilla.

Olemassa olevaan Autocomplete-määrityksiin piti tehdä muutoksia, koska alkuperin ne kutsuivat jQueryn omaa .change()-metodia, eikä tämä ratkaisu toiminut Reactin kanssa (ks. kuvio 32).

```
$(".streetAutocomplete-react").autocomplete({
    source: function (req, resp) {
        commonHub.searchStreetAddress(req.term).done(function (data) {
            resp($.map(data, function (item) {
                return { label: item.Item1 + ", " + item.Item2 + " " + item.Item3, value: item.Item1, zip: item.Item2, place: item.Item3 };
            }));
        });
    },
    select: function (event, ui) {
        var streetEl = this;
        var street = ui.item.value;
        var target = $(streetEl).attr('data-zipContainer');
        if (target == null)
            target = $(streetEl).attr('data-zipcontainer');
        var zip = <HTMLInputElement>document.getElementById(target);

        zip.value = ui.item.zip;
        zip.dispatchEvent(new Event('change', { bubbles: true }));
        streetEl.value = street;
        streetEl.dispatchEvent(new Event('change', { bubbles: true }));

    },
    minLength: 4
});
```

Kuvio 32. StreetAutocomplete-react

Tilattomat komponentit (Stateless Component) eivät sisällä mitään metodeja, ainoastaan CustomerPanel-komponentti suodattaa ContractPartners-taulukosta tyhjät pois (ks. kuvio 33). Renderöinnissä välillä käytettiin React.Fragment-elementtiä, joka käytännössä ei renderöidy ollenkaan, vaan mahdollistaa useampien rinnakkaisten elementtien renderöinnin.

```
const CustomerPanel = (props) => {
    const contractPartners = props.customer.ContractPartners.filter(partner => {
        if (!partner.FirstName)
            return false;
        else
            return true;
    });
    registerAutoCompleteEventHandlers();
    return (
        <React.Fragment>

            <Row>
                <InputWithLabel
                    colSize="col-xs-4"
                    label="Etunimi"
                    required
                    readOnly={props.readOnly}
                    handleChange={props.handleChange}
                    value={props.customer.FirstName}
                    name="Consumer.FirstName" />
                <InputWithLabel
                    colSize="col-xs-8"
                    label="Sukunimi"
                    required
                    readOnly={props.readOnly}
                    handleChange={props.handleChange}
                    value={props.customer.LastName}
                    name="Consumer.LastName" />
            </Row>
```

Kuvio 33. CustomerPanel-komponentin ContractPartners suodatus

Lopulta Reactilla rekisteröidään pääkomponentti juurielementtiin (ks. kuvio 34).

```
811     ReactDOM.render(
812         <TestApp />,
813         document.getElementById("root")
814     );
815
```

Kuvio 34. Komponentin rekisteröiminen juurielementtiin

5.2.3 Vue.js

Vue-sovellus tehtiin hyvin paljolti samalla tavalla, pääsovelluksen alla on eri komponentteja näkymille. Aluksi luotiin sovelluksen rekisteröinti juurielementtiin (ks. kuvio 35).

```
1   import Vue from 'vue'
2   import CustomerPanel from './CustomerPanel.vue'
3
4   // @ts-ignore
5   window.VueRoot = new Vue({
6       el: "#vue",
7       components: {
8           CustomerPanel
9       },
10      template: `<CustomerPanel />`
11  })
```

Kuvio 35. Juurielementin rekisteröinti Vuella

Työssä Vue-instanssi lisättiin globaaliin muuttujaan VueRoot, jota tarvitsimme autocomple-toiminnallisuudessa.

Vue-komponentit toteutettiin kaksi osaisesti: Vue-template, joissa on näyttömallit, ja TypeScript-tiedosto, joka pitää toiminnallisuudet sisässään. CustomerPanel- pääkomponentille annettiin elinkaarimetodi mounted(), joka kutsuu samaa GetCustomerById-metodia API:lta (ks. kuvio 36).

```
    // Lifecycle methods
    mounted(): void {

        let self = this;
        //console.log(this.test);
        $.get('/api/connection/getcustomerbyid/' + connectionProcessId, function (data) {
            console.log("vue-GET: ", data);
            self.state = data;
        });
    }

}
```

Kuvio 36. Vue-komponentin mounted-elinkaarimetodi

CustomerPanel-komponentti piti sisällään kaiken logiikan, kuinka muutoksia käsitellään, ja alikomponentit pystyivät kutsumaan näitä @Emit-toiminnallisuuden kautta. Consumer-komponentilla käytettiin myös Vuen computed-ominaisuutta suodattamaan ContractPartners-taulukosta tyhjät kentät (ks. kuvio 37).

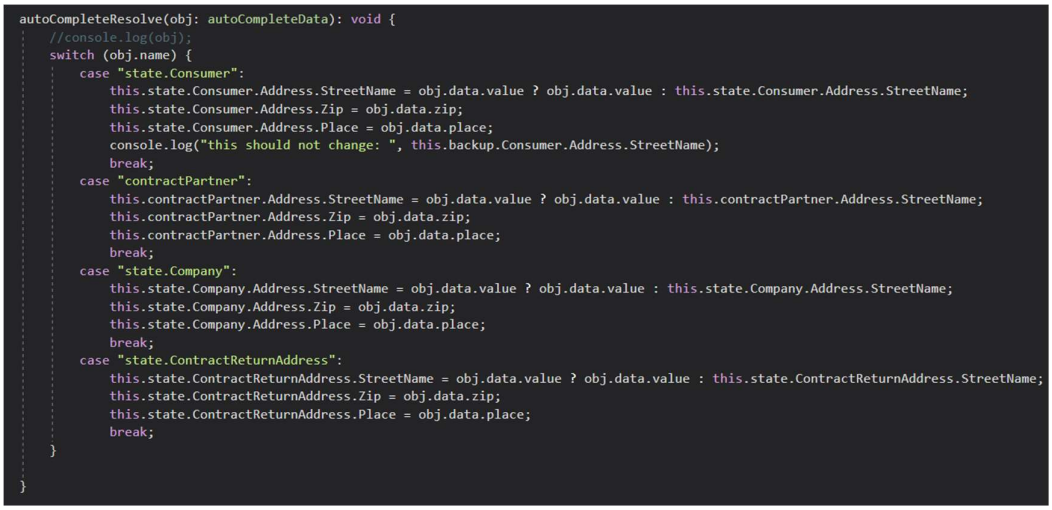
```
1   import Vue from 'vue'
2   import { Component, Prop, Emit } from 'vue-property-decorator'
3   import { IPersonDTO } from '../../../../Models/Default'
4   import registerAutoCompleteEventHandlers from '../../autocomplete'
5
6   @Component
7
8   export default class Consumer extends Vue {
9       @Prop() state: any;
10      @Prop() readOnly: boolean;
11      get filterContractPartners(): Array<IPersonDTO> {
12          return this.state.Consumer.ContractPartners.filter((cp) => {
13              if (cp.FirstName != null)
14                  return true;
15              return false
16          });
17      }
18      @Emit('modify-contract-partner') modifyContractPartner(id: number): void { }
19      @Emit('add-contract-partner') addContractPartner(): void { }
20      @Emit('remove-contract-partner') removeContractPartner(contractPartnerId: number): void { }
21      mounted(): void {
22          //console.log("working", this.state);
23          registerAutoCompleteEventHandlers();
24      }
25  }
```

Kuvio 37. Consumer.component.ts

Autocomplete-toiminnallisuuteen piti myös tehdä muutoksia, ja tässä käytettiin aikaisemmin globaaliksi asettamaa VueRoot-muuttujaa (ks. kuvio 38). Muuttujan kautta kutsuttiin suoraan pääkomponentissa luotua autoCompleteResolve-metodia joka esitetään kuviossa 39.

```
// autocomplete streetnames
$(".streetAutocomplete-vue").autocomplete({
    source: function (req, resp) {
        commonHub.searchStreetAddress(req.term).done(function (data) {
            resp($.map(data, function (item) {
                return { label: item.Item1 + ", " + item.Item2 + " " + item.Item3, value: item.Item1, zip: item.Item2, place: item.Item3 };
            }));
        });
    },
    select: function (event, ui) {
        VueRoot.$children[0].autoCompleteResolve({data: ui.item, name: event.target.name});
    },
    minLength: 4
});
```

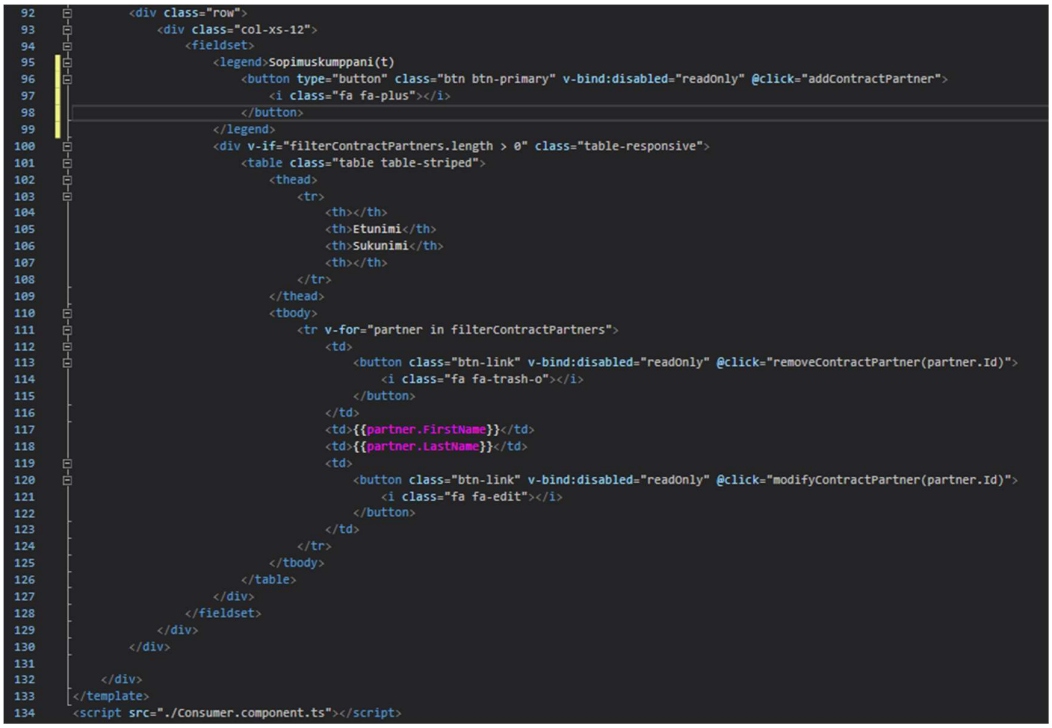
Kuvio 38. streetAutocomplete-vue



```
autoCompleteResolve(obj: autoCompleteData): void {
    //console.log(obj);
    switch (obj.name) {
        case "state.Consumer":
            this.state.Consumer.Address.StreetName = obj.data.value ? obj.data.value : this.state.Consumer.Address.StreetName;
            this.state.Consumer.Address.Zip = obj.data.zip;
            this.state.Consumer.Address.Place = obj.data.place;
            console.log("this should not change: ", this.backup.Consumer.Address.StreetName);
            break;
        case "contractPartner":
            this.contractPartner.Address.StreetName = obj.data.value ? obj.data.value : this.contractPartner.Address.StreetName;
            this.contractPartner.Address.Zip = obj.data.zip;
            this.contractPartner.Address.Place = obj.data.place;
            break;
        case "state.Company":
            this.state.Company.Address.StreetName = obj.data.value ? obj.data.value : this.state.Company.Address.StreetName;
            this.state.Company.Address.Zip = obj.data.zip;
            this.state.Company.Address.Place = obj.data.place;
            break;
        case "state.ContractReturnAddress":
            this.state.ContractReturnAddress.StreetName = obj.data.value ? obj.data.value : this.state.ContractReturnAddress.StreetName;
            this.state.ContractReturnAddress.Zip = obj.data.zip;
            this.state.ContractReturnAddress.Place = obj.data.place;
            break;
    }

}
```

Kuvio 39. autoCompleteResolve-metodi

Itse .vue-päätteiset tiedostot noudattavat yksinkertaista rakennetta. Ensiksi määritetään template-tagin sisään rakenne, joka halutaan tulostaa, ja loppuun lisätään script-tagi, jolla viitataan komponentin TypeScript-tiedostoon (ks. kuvio 40). Elementeillä voidaan käyttää Vuen omia attribuutteja, kuten v-for ja v-bind. Arvoja voidaan myös tulostaa käyttöliittymälle suoraan käyttämällä kaksoisaaltosulkeita. Kuviossa 40 näkyy myös ns. ”shorthand”-viittauksia toimintoihin, kuten @click-attribuutti. Tämä viittaa attribuuttiin v-on:click.



```
92          <div class="row">
93              <div class="col-xs-12">
94                  <fieldset>
95                      <legend>Sopimuskumppani(t)
96                          <button type="button" class="btn btn-primary" v-bind:disabled="readOnly" @click="addContractPartner">
97                              <i class="fa fa-plus"></i>
98                          </button>
99                      </legend>
100                     <div v-if="filterContractPartners.length > 0" class="table-responsive">
101                         <table class="table table-striped">
102                             <thead>
103                                 <tr>
104                                     <th></th>
105                                     <th>Etunimi</th>
106                                     <th>Sukunimi</th>
107                                     <th></th>
108                                 </tr>
109                             </thead>
110                             <tbody>
111                                 <tr v-for="partner in filterContractPartners">
112                                     <td>
113                                         <button class="btn-link" v-bind:disabled="readOnly" @click="removeContractPartner(partner.Id)">
114                                             <i class="fa fa-trash-o"></i>
115                                         </button>
116                                     </td>
117                                     <td>{{partner.FirstName}}</td>
118                                     <td>{{partner.LastName}}</td>
119                                     <td>
120                                         <button class="btn-link" v-bind:disabled="readOnly" @click="modifyContractPartner(partner.Id)">
121                                             <i class="fa fa-edit"></i>
122                                         </button>
123                                     </td>
124                                 </tr>
125                             </tbody>
126                         </table>
127                     </div>
128                 </fieldset>
129             </div>
130         </div>
131
132     </div>
133 </template>
134 <script src="./Consumer.component.ts"></script>
```

Kuvio 40. Osa Consumer.vue-tiedostosta

6 Tulokset

6.1 Rajapinta

Web API 2-tekniikka oli sangen yksinkertainen tapa toteuttaa REST-rajapinta ASP.NET-ympäristöön. Visual Studion Intellisense ja valmiit moduulit tekivät työn toteuttamisesta helppoa, ja ongelmatilanteissa Visual Studion debug-ominaisuudet tarjosivat kattavaa tietoa ongelmista. Rajapintaa testattiin Fiddler-työkalulla, jolla testattiin minkä muotoisena kutsut tulee lähettää rajapinnalle ja miten rajapinta palauttaa dataa. Samalla työkalulla testattiin myös Authorize-toiminnon toimivuus, ja todettiin se toimivaksi. Kun Authorize oli määritetty kontrollerille, ainoastaan järjestelmän kautta saatiin yhteys rajapintaan. DataAnnotations-moduulin avulla tiedon eheys saatiin varmistettua, ja virheelliset kutsut palautettua oikein.

6.2 DTO-mallit

DTO-malli oli kevyt ja nopea tapa abstraktoida näyttökerrokselta tietokantamalli, ja otettua vain oleellinen data käyttöön käyttöliittymällä. Koska DTO:t eivät sisällä mitään toiminnallisuuksia sisällään, käännökset tarvittiin tehdä kontrollerilla. Tämä tapa ei aja parhaiten tuotteen toimintatapaa, jossa eri asiakkuuksissa tarvitaan mahdollisesti eri tietoja. Työssä itsessään tapa on riittävä, mutta jatkoa ajatellen olisi kannattavampaa tehdä käännökset tietokantamalliin joko mallissa itsessään, tai jossain muussa yhteydessä.

6.3 TypeWriter

TypeWriterin käyttöönotto kesti yllättävän kauan, suurilta osin sen syntaksin vuoksi. TypeWriter-generoinnin teko oli ”yritys-erehdys”-toimintatavan mukaista, eli tehtiin muutokset, ja tarkasteltiin toimiiko. Kun lopulta saatiin TypeWriter generoimaan koodi oikein, oli sen käyttö varsin helppoa. Todettiin, että tämä voisi olla hyvä tapa jatkossakin ylläpitää näyttömallien eheyttä käyttöliittymillä.

6.4 React vs Vue.js

6.4.1 React

Koska React käyttää hyväkseen JSX-tiedostoja, saadaan komponentit toteutettua helposti noudattamaan SFC-periaatetta. React tuo mukanaan monia toiminnallisuuksia, ja varsinkin metodien vienti alikomponenteille oli varsin selkeää ja ongelmatonta.

Reactilla tiedonkäsittely osoittautui ongelmalliseksi, koska kaikkiin tietokenttiin tulee viedä metodi, joka käsittelee muutokset. Koska malli oli sisäkkäistetty, vaati tämän geneerinen toteuttaminen turhan monimutkaisen ratkaisun.

Autocomplete-moduuli saatiin lopulta yksinkertaisesti lisättyä sovelluskehykseen, tarvitsi vain lähettää lisäosalta natiivi change-kutsu, ja lisätä kuuntelija input-elementeille, jotta ne osaisivat lähettää tiedon eteenpäin.

Reactilta sai hyvin ilmi ongelmatilanteissa, jos jokin oli vialla. Reactin virheilmoitukset eivät aina olleet yksiselitteisiä, mutta pienellä tutkimisella sai ongelmat selville.

6.4.2 Vue.js

Tuotteen kääntäminen Vuelle oli yllättävän suoraviivaista. Käytännössä kopioitiin olemassa oleva html-muotoinen koodi, ja lisättiin template-tagin sisään. Tämän jälkeen lisättiin toiminnallisuudet, ja pilkottiin alikomponentteihin.

Tapa, jolla Vue-tiedostot tehtiin, piti selkeästi erillään näytettävän osuuden ja toiminnallisen osuuden. Varsinkin input-elementeille muutosten käsittely oli hyvin helppoa: tarvitsi vain lisätä ”v-model”-tagi elementille. Toisaalta tapa, jolla Vuella lähetetään kutsu ylemmälle komponentille, osoittautui turhan hankalaksi käyttää. Alikomponentilla tuli käyttää metodia, joka lähettää Emit-kutsun ylemmäs, ja ylemmällä komponentilla tarvitsi ottaa tämä vastaan ”v-on:kutsu”-muotoisena vielä, joka sitten kutsuu haluttua metodia.

Vuen virheenkäsittely osoittautui myös heikoksi. Joissain tapauksissa Vue ei renderöinyt ollenkaan mallia näytölle, eikä Vue myöskään ilmoittanut mistään virheistä näissä tapauksissa.

Autocomplete-lisäosan kanssa tuli puolestaan ongelma vastaan. Koska Vue ei käytä ollenkaan natiivia change-tapausta, piti keksiä jokin kiertävä tapa. Tässä tapauksessa käytettiin Vue-instanssia globaalin muuttujan kautta, mutta tämä tapa ei ole hyvä kestävän kehityksen näkökulmasta, ja vaatii lisätutkimuksia, miten toiminnallisuus kannattaisi toteuttaa.

6.4.3 Johtopäätökset

Molemmat sovelluskehykset tarjoavat paljon hyviä toiminnallisuuksia, mutta kehittäjien ja tuotteen puolesta Vue on parempi ratkaisu. Vuen lisääminen olemassa olevaan järjestelmään oli yllättävän helppoa, kun taas React vaati paljon erilaisia riippuvuuksia ja lisäilyjä. React myös ajoittain tuntui tahmealta kehittää, ja kaikkeen toiminnallisuuteen tarvitsi tehdä metodeita ja kutsuja. Vue puolestaan aiemmin mainituista ongelmista huolimatta tarjosi nopean tavan kehittää ja uudistaa käyttöliittymää, ja monet Vuen sisäänrakennetuista toiminnoista tuntuivat selkeiltä käyttää.

7 Pohdinta

7.1 Johtopäätökset

Työ toteutti hyvin opinnäytetyön tekijän henkilökohtaiset tavoitteet. Tekijä pääsi tutustumaan moniin eri teknologioihin, jotka alalla kirjoittamisen hetkellä kovassa nousussa. Työ sisälsi monia eri osa-alueita nykyaikaisessa ohjelmistokehittämisessä, ja oli kokonaisuutena yllättävän haastava.

Osittain työ oli pienoinen pettymys myös, koska tekijä halusi paljon suurempaa kokonaisuutta, kuin mitä työssä lopulta toteutettiin. Mutta työn aikarajoitteet supistivat työn skaalaa pienemmäksi, kuin mitä alkuperin oli ajateltu. Työhön kului yhteensä n. 22 htp, eli työ suoritettiin suunnitellussa aikataulussa.

Itse työn osa-alueet todettiin toimiviksi, lukuun ottamatta DTO-malleja. DTO-mallit itsessään olivat toimiva ratkaisu opinnäytetyön raameissa, mutta tuotteen puolesta nämä eivät täytä kaikkia vaatimuksia. Tämä johtui siitä, että tuotteen eri asiakkaat käyttävät eri tietoja järjestelmissään, ja mallien kääntäminen rajapinnassa olisi vaivalloista tehdä. ViewModelit tarjoavat ylikirjoitettavia toiminnallisuuksia ja metodeita, joita pystyttäisiin käyttämään mallin kääntämiseen tietokantamalliin. Tämä tarjoaisi eri käännökset eri asiakkaiden ympäristöihin paljon selkeämmin, ja poistaisi käännösvastuun rajapinnalta mallille.

REST-rajapintaa voidaan helposti laajentaa, ja sen tuonti eri tuotteen osa-alueille olisi suhteellisen helppoa. REST-rajapinta tarjoaa myös monipuolisia mahdollisuuksia tiedon käsittelyyn. Rajapinnan rajoituksena on kuitenkin sen riippuvuus kutsuista. Reaaliaikaisen datan hallinta tosin vaatisi muita ratkaisuja, kuten WebSocket, joka mahdollistaa viestinnän käyttöliittymän ja palvelimen välillä, jolloin tiedon muutoksista palvelimella pystyttäisiin viestittämään käyttöliittymää.

Vue osoittautui tuotteen tapauksessa ketterämmäksi tavaksi tehdä käyttöliittymiä, sekä uusia, että olemassa olevien päivittämisen puolesta. Vue oli myös helpommin omaksuttava sovelluskehys verrattuna Reactiin, eli Vuen opettelu olisi nopeampaa.

Olemassa oleviin osiin kuitenkin tämä kehitys ei tuntunut kannattavalta liiketoiminnan puolesta: saatavat hyödyt muunnoksesta olisivat liian marginaalisia verrattuna käytettävään työpanokseen.

Vuen käyttöönotto kehityksessä tarvitsisi kuitenkin keskustelua arkkitehtuurisesta suunnasta, jossa voitaisiin määrittää geneerisiä komponentteja, joita voitaisiin käyttää myös muillakin osa-alueilla. Tämä myös vaatisi yhteisen linjanvedon tuotekehityksen tekemisestä myös eri asiakaskonteksteissa.

7.2 Muita mahdollisia jatkotutkimuskohteita

7.2.1 Koodin automaattinen generointi

Yksi mahdollisuus kehittää työskentelyä olisi tutkia tapoja, joilla voidaan generoida koodia ja näyttömalleja. Koska tietomallit ovat kasvaneet suuriksi, suurien näyttömallien ylläpito on työlästä.

7.2.2 AutoMapper

AutoMapper on C#:n moduuli, jolla voidaan määrittää eri objektien riippuvuudet keskenään. Tämä vähentäisi tarvetta käsin kirjoittaa käännöksiä, ja saisi koodin luettavuudesta parempaa. Aikarajoitteiden puitteissa tämä kuitenkin jäi tästä työstä ulkopuolelle.

7.2.3 Datan validoinnin menetelmät

DataAnnotations tarjosi paljon hyviä tapoja validoida dataa, mutta tässä työssä moduulin käyttö oli vähäistä. Moduulin kautta kuitenkin pystyy saamaan virheilmoituksia kaikista määritellyistä osista, ja niitä pystyy muokkaamaan halutun muotoisiksi. Lisäosa tarvitsisi jatkotutkimusta ja kokeiluja, jotta sen hyödyt ja rajoitteet selviäisivät tuotteen tapauksessa.

7.2.4 Store-ratkaisut

Sekä React että Vue tarjoavat store-ratkaisuja, Redux ja Vuex. Nämä ratkaisut toisivat datan hallinnoimiseen paljon hyviä puolia, esimerkiksi muutoshistorian. Nämä veisivät myös tarpeen hallinnoida dataa itse komponenteilla, jolloin komponentit pysyvät siisteinä ja yksinkertaisina. Aikarajoitteiden puitteissa tähän työhön niitä ei tehty.

Lähteet

ASP.NET MVC – Razor. N.d. Tutorialspointin Razorin esittely. Viitattu 4.9.2018. https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_razor.htm

Eng, R. 2017. Chapter 3: What is Object-Oriented Programming?. Viitattu 4.9.2018. https://medium.com/learn-how-to-program/chapter-3-what-is-object-oriented-programming-d0a6ec0a7615

Get building. N.d. ASP.NET:n kotisivut. Viitattu 4.9.2018. https://www.asp.net/

Introduction. N.d. Vue:n kotisivut. Viitattu 30.8.2018. https://vuejs.org/v2/guide/index.html

Koirala, S. 2015. Data Transfer Object Design Pattern in C#. Viitattu 30.8.2018. https://www.codeproject.com/Articles/1050468/Data-Transfer-Object-Design-Pattern-in-Csharp

Lauer, R. 2017a. Kuvion 6 luokka käännetty JavaScriptiksi. Kuvakaappaus nettisivuilta. Viitattu 30.8.2018. https://developer.telerik.com/topics/web-development/what-is-typescript/

Lauer, R. 2017b. TypeScript-luokka. Kuvakaappaus nettisivuilta. Viitattu 30.8.2018. https://developer.telerik.com/topics/web-development/what-is-typescript/

Lauer, R. 2017c. What is TypeScript. Viitattu 30.8.2018. https://developer.telerik.com/topics/web-development/what-is-typescript/

Miller, R., South, S., Vega, D. & Wenzel, M. 2018. Entity Framework 6. Viitattu 4.9.2018. https://docs.microsoft.com/en-us/ef/ef6/

PoC eli Proof of Concept. N.d. Omni Partners Oy:n sanakirja. Viitattu 4.9.2018. https://omnipartners.fi/sanakirja/poc-eli-proof-of-concept/

React – A JavaScript library for building user interfaces. N.d. Reactin kotisivut. Viitattu 30.8.2018. https://reactjs.org/

Rouse, M. 2005. ECMAScript (European Computer Manufacturers Association Script). Viitattu 4.9.2018. https://searchsqlserver.techtarget.com/definition/ECMAScript

SFC-komponentti Vue.js:llä. N.d. Kuvakaappaus nettisivuilta. Viitattu 30.8.2018. https://vuejs.org/v2/guide/single-file-components.html

TypeWriter. N.d. TypeWriterin github-sivut. Viitattu 30.8.2018. https://frhagn.github.io/Typewriter/

TypeWriterilla generoitu koodi. N.d. Kuvakaappaus TypeWriterin github-sivuilta. Viitattu 30.8.2018. https://frhagn.github.io/Typewriter/

TypeWriter-esimerkissä käytetty luokka. N.d. Kuvakaappaus TypeWriterin github-sivuilta. Viitattu 30.8.2018. https://frhagn.github.io/Typewriter/

TypeWriter-template. N.d. Kuvakaappaus TypeWriterin github-sivuilta. Viitattu 30.8.2018. https://frhagn.github.io/Typewriter/

Vue-komponentti TypeScriptillä. N.d. Kuvakaappaus Vue.js:n kotisivuilta. Viitattu 30.8.2018. https://vuejs.org/v2/guide/typescript.html

Wasson, M. 2014. Create Data Transfer Objects (DTOs). Viitattu 30.8.2018. https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

Wasson, M. 2017. Get Started with ASP.NET Web API 2 (C#). Viitattu 29.8.2018. https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

What is an API (Application Programming Interface). N.d. MuleSoft:n määritelmä API:sta. Viitattu 4.9.2018. https://www.mulesoft.com/resources/api/what-is-an-api

What is CSS?. N.d. W3C:n esittely CSS:stä. Viitattu 4.9.2018. https://www.w3.org/Style/CSS/

What is jQuery?. N.d. jQueryn kotisivut. Viitattu 30.8.2018. https://jquery.com/

What is LINQ? N.d. TutorialsTeacher.comin määritelmä LINQ:sta. Viitattu 6.9.2018. http://www.tutorialsteacher.com/linq/what-is-linq

What is REST?. N.d. Codecademyn esittely REST-arkkitehtuurista. Viitattu 4.9.2018. https://www.codecademy.com/articles/what-is-rest

What Is Webpack. N.d. Webpackin kotisivut. Viitattu 30.8.2018. https://survivejs.com/webpack/what-is-webpack/

Yksinkertainen Ajax-kutsu. N.d. Kuvakaappaus nettisivuilta. Viitattu 30.8.2018. https://api.jquery.com/jQuery.ajax/

Yritys - Solteq, N.d. Solteq Oyj:n kotisivut. Viitattu 29.8.2018. https://www.solteq.com/fi/

Liitteet

Liite 1. ConnectionController-lähdekoodi

```
using Common.Logging;
using [redacted]
using [redacted]
using [redacted]
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace [redacted].ApiControllers
{
    [Authorize([redacted])]
    [AdminAccessLevelAttribute([redacted])]
    1 reference | JaniKerttula, 9 days ago | 2 authors, 4 changes
    public class ConnectionController : ApiController
    {
        protected static readonly ILog Log = LogManager.GetLogger(typeof(ConnectionController));
        private readonly Inpulse4UtilitiesContext ctx = new Inpulse4UtilitiesContext();

        [HttpGet]
        0 references | JaniKerttula, 9 days ago | 1 author, 1 change | 0 requests | 0 exceptions
        public IHttpActionResult GetCustomerById(int id)
        {
            try
            {
                var cons = from con in ctx.ConnectionProcesses
                           where con.ConnectionProcessId == id
                           select new ConnectionCustomerDTO()
                           {
                               Id = con.ConnectionProcessId,
                               CustomerType = con.CustomerType,
                               // Consumer
                               Consumer = new ConsumerCustomerDTO()
                               {
                                   Id = con.ConsumerCustomer.ConsumerCustomerId,
                                   FirstName = con.ConsumerCustomer.FirstName,
                                   LastName = con.ConsumerCustomer.LastName,
                                   Pin = con.ConsumerCustomer.Pin,
                                   Email = con.ConsumerCustomer.Email,
                                   Phone = con.ConsumerCustomer.Phone,
                                   CustomerCode = con.ConsumerCustomer.CustomerCode,
                                   Address = new AddressDTO()
                                   {
                                       COName = con.ConsumerCustomer.CustomerAddress.COName,
                                       StreetName = con.ConsumerCustomer.CustomerAddress.StreetAddress,
                                       HouseNumber = con.ConsumerCustomer.CustomerAddress.HouseNumber,
                                       Staircase = con.ConsumerCustomer.CustomerAddress.Staircase,
                                       Apartment = con.ConsumerCustomer.CustomerAddress.Apartment,
                                       Zip = con.ConsumerCustomer.CustomerAddress.ZipCode,
                                       Place = con.ConsumerCustomer.CustomerAddress.PostOffice,
                                   },
                                   ContractPartners = con.ConsumerCustomer.ContractPartners
                                    .Select(x => new PersonDTO()
                                    {
                                        Id = x.ContractPartnerId,
                                        FirstName = x.FirstName,
                                        LastName = x.LastName,
                                        Pin = x.Pin,
                                        Email = x.Email,
                                        Phone = x.Phone,
                                        Address = new AddressDTO()
                                        {
                                            COName = x.Address.COName,
                                            StreetName = x.Address.StreetAddress,
                                            HouseNumber = x.Address.HouseNumber,
                                            Staircase = x.Address.Staircase,
                                            Apartment = x.Address.Apartment,
                                            Zip = x.Address.ZipCode,
                                            Place = x.Address.PostOffice,
                                        }
                                    }).ToList(),
                               },
                               Company = new CompanyCustomerDTO()
                               {
                                   Id = con.CompanyCustomer.CompanyCustomerId,
                                   CompanyName = con.CompanyCustomer.Name,
                                   BusinessId = con.CompanyCustomer.BusinessId,
                                   Email = con.CompanyCustomer.Email,
                                   Phone = con.CompanyCustomer.Phone,
                                   CustomerCode = con.CompanyCustomer.CustomerCode,
                                   ContactPerson = new CompanyContactPersonDTO()
                                   {
                                       Id = con.CompanyCustomer.ContactPersons.FirstOrDefault().ContactPersonId,
                                       FirstName = con.CompanyCustomer.ContactPersons.FirstOrDefault().FirstName,
                                       LastName = con.CompanyCustomer.ContactPersons.FirstOrDefault().LastName,
                                       Email = con.CompanyCustomer.ContactPersons.FirstOrDefault().Email,
                                       Phone = con.CompanyCustomer.ContactPersons.FirstOrDefault().Phone,
                                   },
                                   Address = new AddressDTO()
                                   {
```

```
                                Address = new AddressDTO()
                                {
                                    COName = con.CompanyCustomer.CompanyAddress.COName,
                                    StreetName = con.CompanyCustomer.CompanyAddress.StreetAddress,
                                    HouseNumber = con.CompanyCustomer.CompanyAddress.HouseNumber,
                                    Staircase = con.CompanyCustomer.CompanyAddress.Staircase,
                                    Apartment = con.CompanyCustomer.CompanyAddress.Apartment,
                                    Zip = con.CompanyCustomer.CompanyAddress.ZipCode,
                                    Place = con.CompanyCustomer.CompanyAddress.PostOffice,
                                },
                            },

                            HasDifferentContactReturnAddress = con.HasDifferentContactReturnAddress,
                            ContractReturnAddress = new AddressDTO()
                            {
                                COName = con.ContractReturnAddress.COName,
                                StreetName = con.ContractReturnAddress.StreetAddress,
                                HouseNumber = con.ContractReturnAddress.HouseNumber,
                                Staircase = con.ContractReturnAddress.Staircase,
                                Apartment = con.ContractReturnAddress.Apartment,
                                Zip = con.ContractReturnAddress.ZipCode,
                                Place = con.ContractReturnAddress.PostOffice,
                            },
                        };
            // since Linq-query always returns a IQueryable-listitem, return the result as a single entry
            var retval = cons.Single();
            Log.Debug("Testing API-GET");
            if (retval == null)
                return NotFound();
            return Ok(retval);
        }
        catch (Exception ex)
        {
            return BadRequest("Exception occured: " + ex.Message);
        }
    }

    [HttpPut]
    0 references | JaniKerttula, 10 days ago | 1 author, 1 change | 0 requests | 0 exceptions
    public IHttpActionResult UpdateCustomer(ConnectionCustomerDTO model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest("Model is not valid");
        }
        try
        {
            var existing = ctx.ConnectionProcesses.Where(x => x.ConnectionProcessId == model.Id).FirstOrDefault();
            if (existing == null)
                return NotFound();

            existing.CustomerType = model.CustomerType;

            // Consumer
            existing.ConsumerCustomer.ConsumerCustomerId = model.Consumer.Id;
            existing.ConsumerCustomer.FirstName = model.Consumer.FirstName;
            existing.ConsumerCustomer.LastName = model.Consumer.LastName;
            existing.ConsumerCustomer.Pin = model.Consumer.Pin;
            existing.ConsumerCustomer.Email = model.Consumer.Email;
```

```
            existing.ConsumerCustomer.Email = model.Consumer.Email;
            existing.ConsumerCustomer.Phone = model.Consumer.Phone;
            existing.ConsumerCustomer.CustomerCode = model.Consumer.CustomerCode;

            existing.ConsumerCustomer.CustomerAddress.COName = model.Consumer.Address.COName;
            existing.ConsumerCustomer.CustomerAddress.StreetAddress = model.Consumer.Address.StreetName;
            existing.ConsumerCustomer.CustomerAddress.HouseNumber = model.Consumer.Address.HouseNumber;
            existing.ConsumerCustomer.CustomerAddress.Staircase = model.Consumer.Address.Staircase;
            existing.ConsumerCustomer.CustomerAddress.Apartment = model.Consumer.Address.Apartment;
            existing.ConsumerCustomer.CustomerAddress.ZipCode = model.Consumer.Address.Zip;
            existing.ConsumerCustomer.CustomerAddress.PostOffice = model.Consumer.Address.Place;
            foreach (var cp in model.Consumer.ContractPartners)
            {

                var existingCp = existing.ConsumerCustomer.ContractPartners.Where(y => y.ContractPartnerId == cp.Id);
                if (existingCp.Count() > 0)
                {
                    var x = existingCp.First();
                    x.ContractPartnerId = cp.Id;
                    x.FirstName = cp.FirstName;
                    x.LastName = cp.LastName;
                    x.Pin = cp.Pin;
                    x.Email = cp.Email;
                    x.Phone = cp.Phone;

                    x.Address.COName = cp.Address.COName;
                    x.Address.StreetAddress = cp.Address.StreetName;
                    x.Address.HouseNumber = cp.Address.HouseNumber;
                    x.Address.Staircase = cp.Address.Staircase;
                    x.Address.Apartment = cp.Address.Apartment;
                    x.Address.ZipCode = cp.Address.Zip;
                    x.Address.PostOffice = cp.Address.Place;
                }
                else
                {
```

```
                else
                {
                    var x = new ContractPartner
                    {
                        FirstName = cp.FirstName,
                        LastName = cp.LastName,
                        Pin = cp.Pin,
                        Email = cp.Email,
                        Phone = cp.Phone,
                        Address = new Address
                        {
                            COName = cp.Address.COName,
                            StreetAddress = cp.Address.StreetName,
                            HouseNumber = cp.Address.HouseNumber,
                            Staircase = cp.Address.Staircase,
                            Apartment = cp.Address.Apartment,
                            ZipCode = cp.Address.Zip,
                            PostOffice = cp.Address.Place,
                        }
                    };
                    existing.ConsumerCustomer.ContractPartners.Add(x);
                }
            }
            // Company
            existing.CompanyCustomer.CompanyCustomerId = model.Company.Id;
            existing.CompanyCustomer.Name = model.Company.CompanyName;
            existing.CompanyCustomer.BusinessId = model.Company.BusinessId;
            existing.CompanyCustomer.Email = model.Company.Email;
            existing.CompanyCustomer.Phone = model.Company.Phone;
            existing.CompanyCustomer.CustomerCode = model.Company.CustomerCode;

            existing.CompanyCustomer.ContactPersons.FirstOrDefault().ContactPersonId = model.Company.ContactPerson.Id;
            existing.CompanyCustomer.ContactPersons.FirstOrDefault().FirstName = model.Company.ContactPerson.FirstName;
            existing.CompanyCustomer.ContactPersons.FirstOrDefault().LastName = model.Company.ContactPerson.LastName;
            existing.CompanyCustomer.ContactPersons.FirstOrDefault().Email = model.Company.ContactPerson.Email;
            existing.CompanyCustomer.ContactPersons.FirstOrDefault().Phone = model.Company.ContactPerson.Phone;

            existing.CompanyCustomer.CompanyAddress.COName = model.Company.Address.COName;
            existing.CompanyCustomer.CompanyAddress.StreetAddress = model.Company.Address.StreetName;
            existing.CompanyCustomer.CompanyAddress.HouseNumber = model.Company.Address.HouseNumber;
            existing.CompanyCustomer.CompanyAddress.Staircase = model.Company.Address.Staircase;
            existing.CompanyCustomer.CompanyAddress.Apartment = model.Company.Address.Apartment;
            existing.CompanyCustomer.CompanyAddress.ZipCode = model.Company.Address.Zip;
            existing.CompanyCustomer.CompanyAddress.PostOffice = model.Company.Address.Place;

            // ContractReturnAddress
            existing.HasDifferentContactReturnAddress = model.HasDifferentContactReturnAddress;

            existing.ContractReturnAddress.COName = model.ContractReturnAddress.COName;
            existing.ContractReturnAddress.StreetAddress = model.ContractReturnAddress.StreetName;
            existing.ContractReturnAddress.HouseNumber = model.ContractReturnAddress.HouseNumber;
            existing.ContractReturnAddress.Staircase = model.ContractReturnAddress.Staircase;
            existing.ContractReturnAddress.Apartment = model.ContractReturnAddress.Apartment;
            existing.ContractReturnAddress.ZipCode = model.ContractReturnAddress.Zip;
            existing.ContractReturnAddress.PostOffice = model.ContractReturnAddress.Place;

            existing.Updated = DateTime.Now;

            ctx.SaveChanges();
            return Ok("Request is ok");

        }
        catch (Exception ex)
        {
            return BadRequest("Exception occured: " + ex.Message);
        }
    }
    1 reference | 0 changes | 0 authors, 0 changes
    public class RemoveContractPartnerRequest
    {
        [Required]
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public int ConnectionProcessId { get; set; }
        [Required]
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public int ContractPartnerId { get; set; }
    }
    [HttpPost]
    0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
    public IHttpActionResult RemoveContractPartner(RemoveContractPartnerRequest req)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest("Model is not valid");
        }
        try
        {
            var con = ctx.ConnectionProcesses.Where(x => x.ConnectionProcessId == req.ConnectionProcessId).First();
            if (con != null)
            {
```

```
264                 if (con != null)
265                 {
266                     var removethis = con.ConsumerCustomer.ContractPartners.Find(x => x.ContractPartnerId == req.ContractPartnerId);
267                     con.ConsumerCustomer.ContractPartners.Remove(removethis);
268                     con.Updated = DateTime.Now;
269                     ctx.SaveChanges();
270                     return Ok("Request is ok");
271                 }
272                 return NotFound();
273             }
274             catch (Exception ex)
275             {
276                 return BadRequest("Exception occured: " + ex.Message);
277             }
278         }
            1 reference | 0 changes | 0 authors, 0 changes
279         public class AddContractPartnerRequest
280         {
281             [Required]
                1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
282             public int ConnectionProcessId { get; set; }
283             [Required]
                12 references | 0 changes | 0 authors, 0 changes | 0 exceptions
284             public PersonDTO Data{ get; set; }
285         }
286         [HttpPost]
            0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
287         public IHttpActionResult AddContractPartner(AddContractPartnerRequest req)
288         {
289             if (!ModelState.IsValid)
290             {
291                 return BadRequest("Model is not valid");
292             }
293             try
294             {
295                 var con = ctx.ConnectionProcesses.Where(x => x.ConnectionProcessId == req.ConnectionProcessId).First();
296                 if (con != null)
297                 {
298                     var addthis = new ContractPartner
299                     {
300                         FirstName = req.Data.FirstName,
301                         LastName = req.Data.LastName,
302                         Pin = req.Data.Pin,
303                         Email = req.Data.Email,
304                         Phone = req.Data.Phone,
305                         Address = new Address
306                         {
307                             COName = req.Data.Address.COName,
308                             StreetAddress = req.Data.Address.StreetName,
309                             HouseNumber = req.Data.Address.HouseNumber,
310                             Staircase = req.Data.Address.Staircase,
311                             Apartment = req.Data.Address.Apartment,
312                             ZipCode = req.Data.Address.Zip,
313                             PostOffice = req.Data.Address.Place,
314                         }
315                     };
316                     con.ConsumerCustomer.ContractPartners.Add(addthis);
317                     con.Updated = DateTime.Now;
318                     ctx.SaveChanges();
319
```

```
318                     ctx.SaveChanges();
319
320                     return Ok(con.ConsumerCustomer.ContractPartners.Last().ContractPartnerId);
321                 }
322                 return NotFound();
323             }
324             catch (Exception ex)
325             {
326                 return BadRequest("Exception occured: " + ex.Message);
327             }
328         }
329     }
330 }
331
332
```

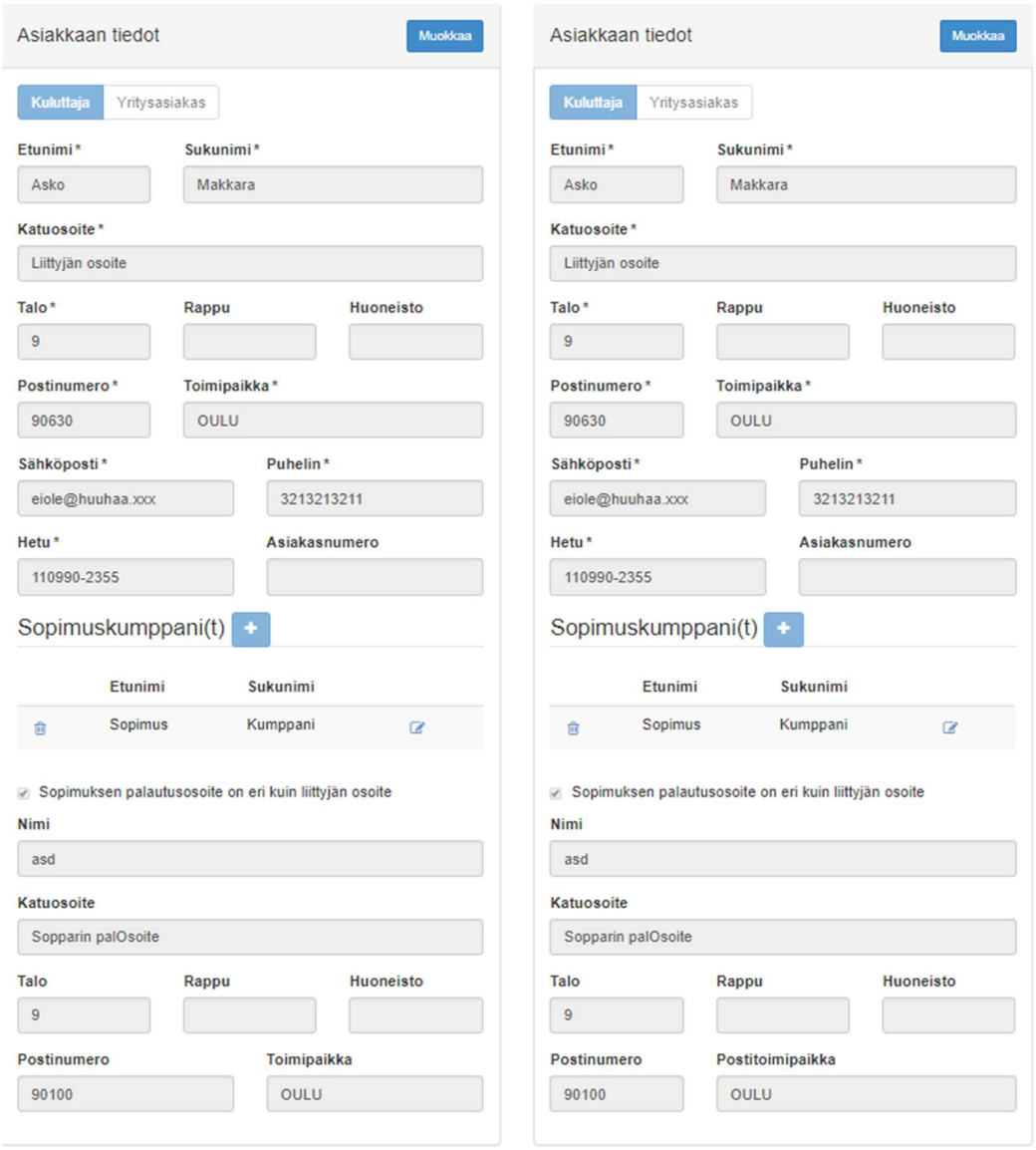
Liite 2. TypeWriterin generoima koodi

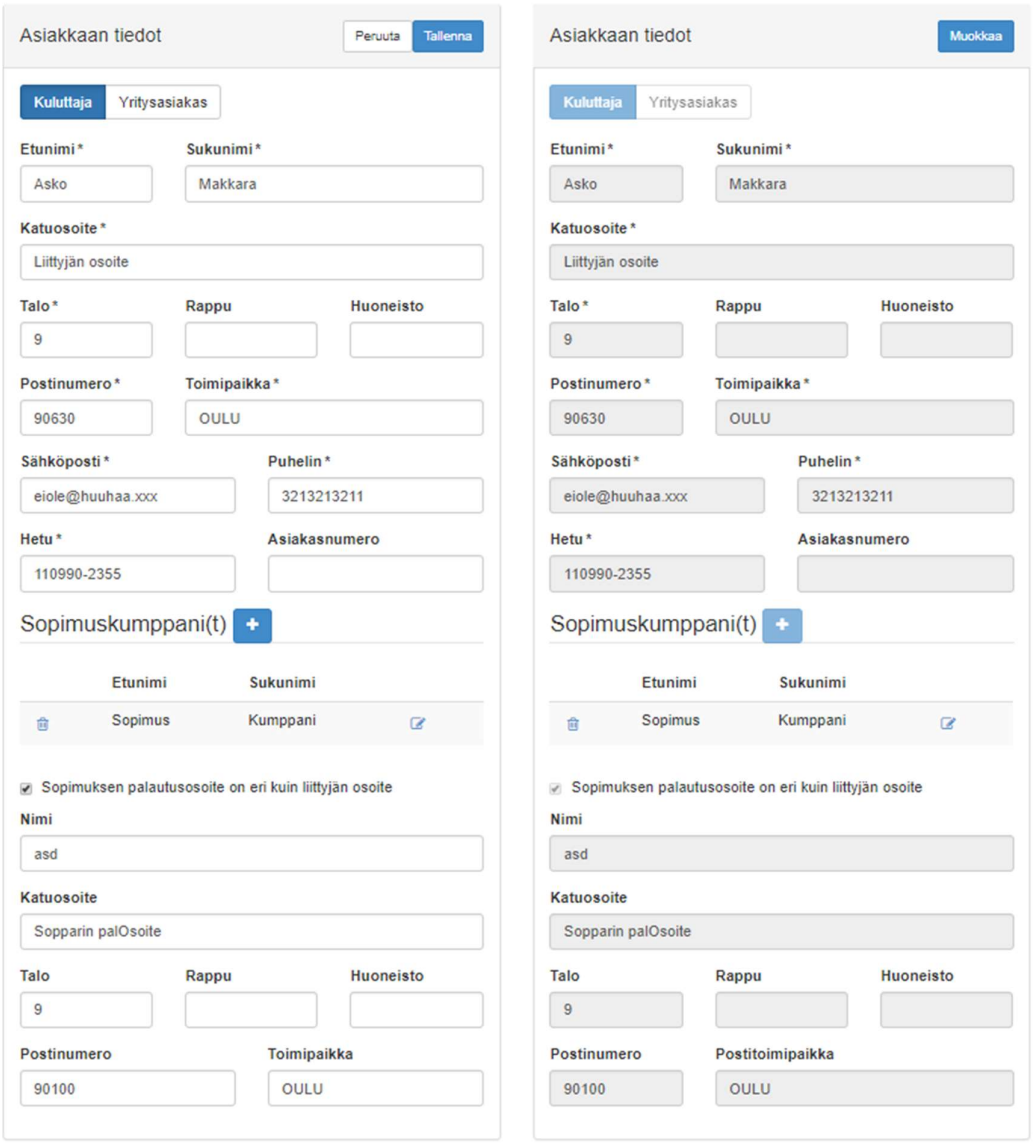
```
function AddressDTO() {
    return {
        //
        //AddressDTO
        COName: "",
        StreetName: "",
        HouseNumber: "",
        Staircase: "",
        Apartment: "",
        Zip: "",
        Place: "",

    }
}
function PersonDTO() {
    return {
        //
        //PersonDTO
        Id: 0,
        FirstName: "",
        LastName: "",
        Pin: "",
        Address: AddressDTO(),
        Email: "",
        Phone: "",

    }
}
function CompanyContactPersonDTO() {
    return {
        //PersonDTO
        Id: 0,
        FirstName: "",
        LastName: "",
        Pin: "",
        Address: AddressDTO(),
        Email: "",
        Phone: "",

        //CompanyContactPersonDTO

    }
}
function ConsumerCustomerDTO() {
    return {
        //PersonDTO
        Id: 0,
        FirstName: "",
        LastName: "",
        Pin: "",
        Address: AddressDTO(),
        Email: "",
        Phone: "",
```

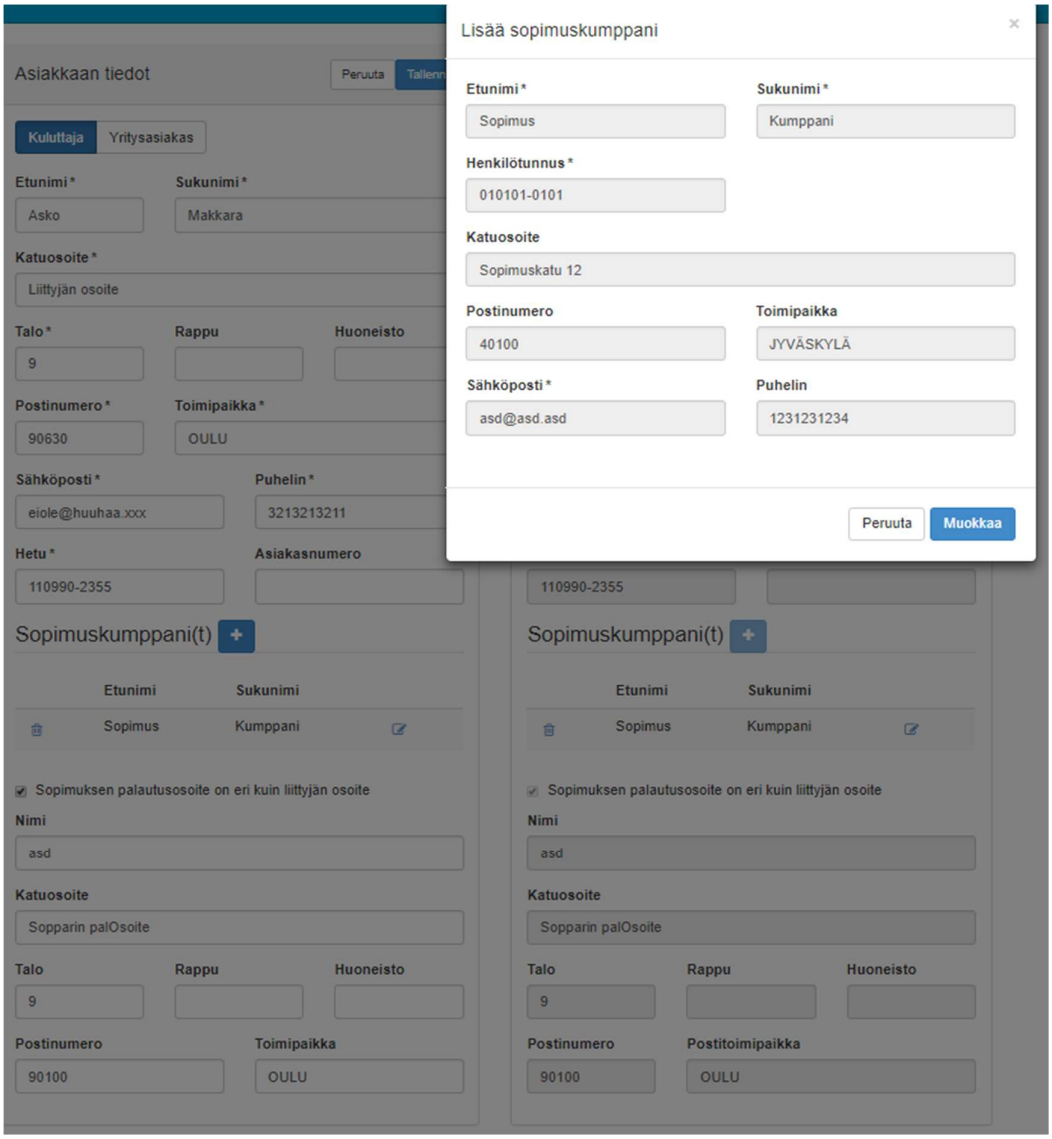
```
        //ConsumerCustomerDTO
        CustomerCode: "",
        ContractPartners: new Array<IPersonDTO>(),

    }
}
function CompanyCustomerDTO() {
    return {
        //

        //CompanyCustomerDTO
        Id: 0,
        CompanyName: "",
        BusinessId: "",
        Address: AddressDTO(),
        Email: "",
        Phone: "",
        CustomerCode: "",
        ContactPerson: CompanyContactPersonDTO(),

    }
}
function ConnectionCustomerDTO() {
    return {
        //

        //ConnectionCustomerDTO
        Id: 0,
        CustomerType: 0,
        Consumer: ConsumerCustomerDTO(),
        Company: CompanyCustomerDTO(),
        HasDifferentContactReturnAddress: null,
        ContractReturnAddress: AddressDTO(),

    }
}

interface IAddressDTO {  COName: string;  StreetName: string;  HouseNumber: string;  Staircase: string;  Apartment: string;  Zip: string;  Place: string;  }

interface IPersonDTO {  Id: number;  FirstName: string;  LastName: string;  Pin: string;  Address: IAddressDTO;  Email: string;  Phone: string;  }

interface ICompanyContactPersonDTO {  Id: number,  FirstName: string,  LastName: string,  Pin: string,  Address: IAddressDTO,  Email: string,  Phone: string,  }

interface IConsumerCustomerDTO {  Id: number,  FirstName: string,  LastName: string,  Pin: string,  Address: IAddressDTO,  Email: string,  Phone: string,  CustomerCode: string;  ContractPartners: IPersonDTO[];  }

interface ICompanyCustomerDTO {  Id: number;  CompanyName: string;  BusinessId: string;  Address: IAddressDTO;  Email: string;  Phone: string;  CustomerCode: string;  ContactPerson: ICompanyContactPersonDTO;  }

interface IConnectionCustomerDTO {  Id: number;  CustomerType: number;  Consumer: IConsumerCustomerDTO;  Company: ICompanyCustomerDTO;  HasDifferentContactReturnAddress: boolean;  ContractReturnAddress: IAddressDTO;  }
```

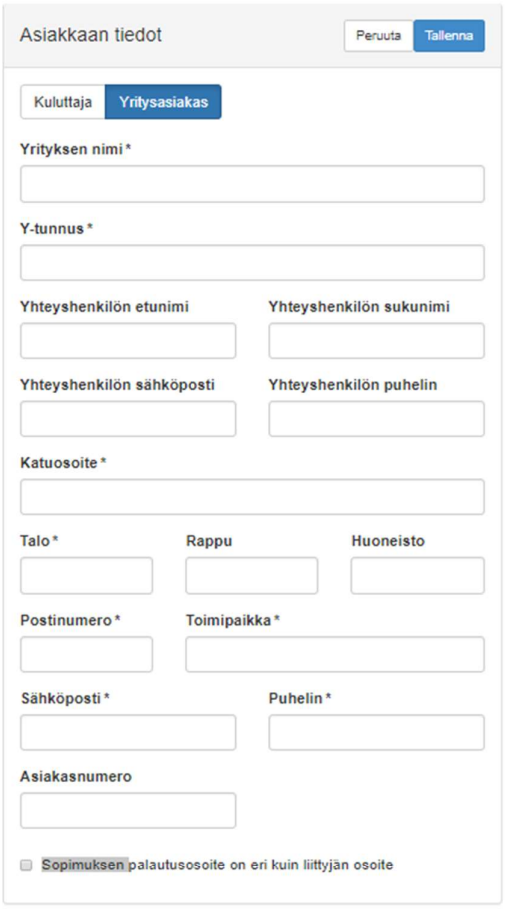
```

export {
    AddressDTO,
    PersonDTO,
    CompanyContactPersonDTO,
    ConsumerCustomerDTO,
    CompanyCustomerDTO,
    ConnectionCustomerDTO,

    IAddressDTO,
    IPersonDTO,
    ICompanyContactPersonDTO,
    IConsumerCustomerDTO,
    ICompanyCustomerDTO,
    IConnectionCustomerDTO,

}
```

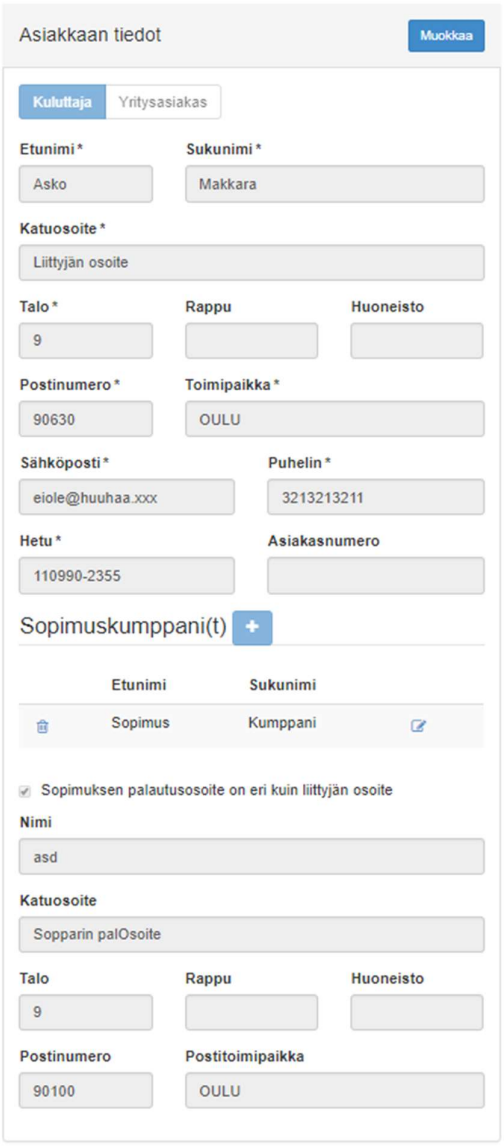
Liite 3. Kuvakaappaukset käyttöliittymästä

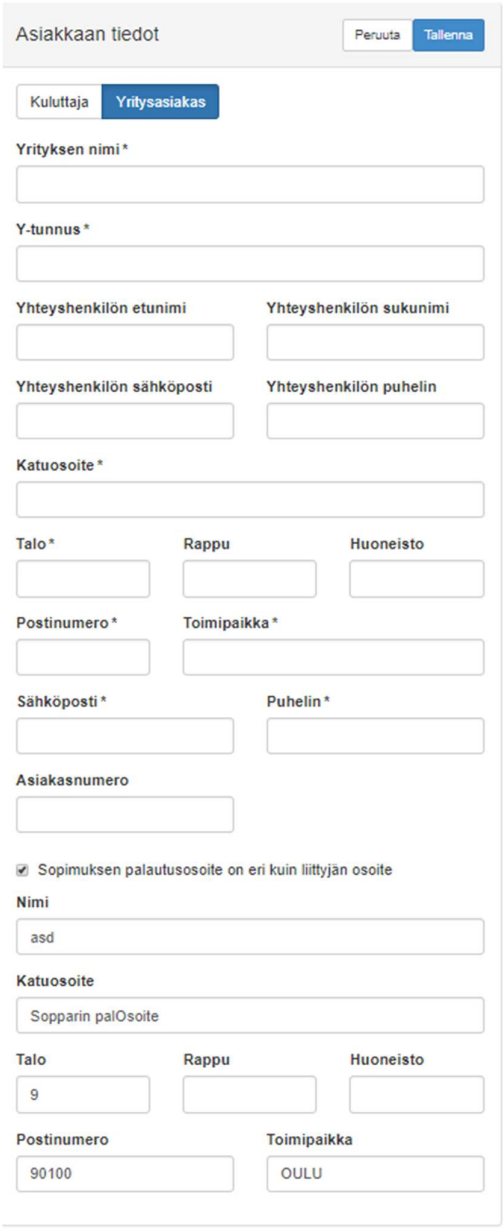


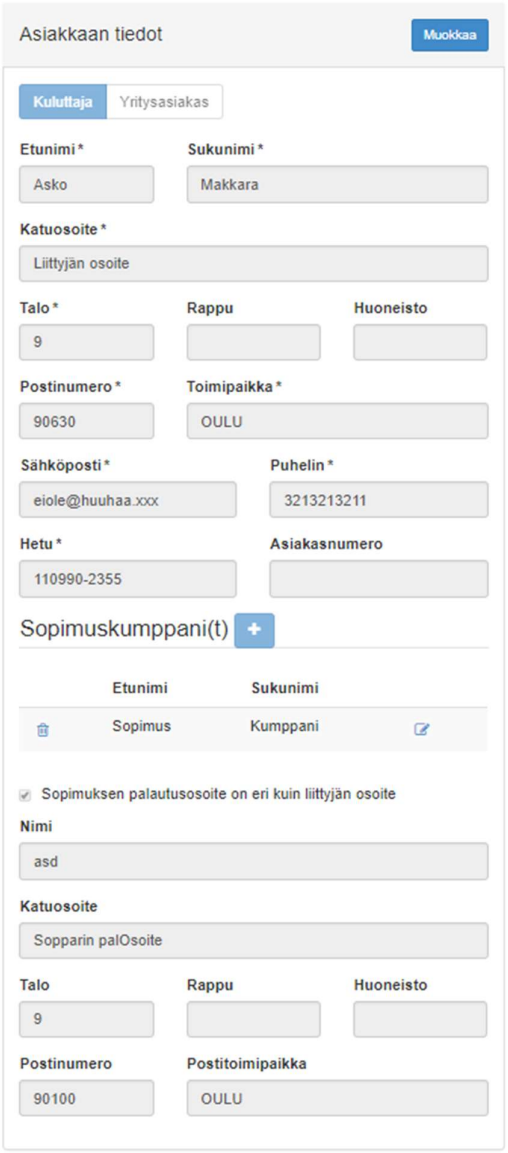

Asiakkaan tiedot
Peruuta
Tallenna
Kuluttaja
Yritysasiakas
Etunimi *
Asko
Sukunimi *
Makkara
Katuosoite *
Liittyjän osoite
Talo *
9
Rappu
Huoneisto
Postinumero *
90630
Toimipaikka *
OULU
Sähköposti *
eiole@huuhaa.xxx
Puhelin *
3213213211
Hetu *
110990-2355
Asiakasnumero
Sopimuskumppani(t)
Etunimi
Sukunimi
Sopimus
Kumppani
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Toimipaikka
OULU
Asiakkaan tiedot
Muokkaa
Kuluttaja
Yritysasiakas
Etunimi *
Asko
Sukunimi *
Makkara
Katuosoite *
Liittyjän osoite
Talo *
9
Rappu
Huoneisto
Postinumero *
90630
Toimipaikka *
OULU
Sähköposti *
eiole@huuhaa.xxx
Puhelin *
3213213211
Hetu *
110990-2355
Asiakasnumero
Sopimuskumppani(t)
Etunimi
Sukunimi
Sopimus
Kumppani
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Postitoimipaikka
OULU


Asiakkaan tiedot
Peruuta
Kuluttaja
Yritysasiakas
Etunimi *
Asko
Sukunimi *
Makkara
Katuosoite *
Liittyjän osoite
Talo *
9
Rappu
Huoneisto
Postinumero *
90630
Toimipaikka *
OULU
Sähköposti *
eiole@huuhaa.xxx
Puhelin *
3213213211
Hetu *
110990-2355
Asiakasnumero
Sopimuskumppani(t)
Etunimi
Sukunimi
Sopimus
Kumppani
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Toimipaikka
OULU
Lisää sopimuskumppani
Etunimi *
Sopimus
Sukunimi *
Kumppani
Henkilötunnus *
010101-0101
Katuosoite
Sopimuskatu 12
Postinumero
40100
Toimipaikka
JYVÄSKYLÄ
Sähköposti *
asd@asd.asd
Puhelin
1231231234
Peruuta
Muokkaa
Postitoimipaikka


Asiakkaan tiedot
Peruuta
Tallenna
Kuluttaja
Yritysasiakas
Yrityksen nimi *
Y-tunnus *
Yhteyshenkilön etunimi
Yhteyshenkilön sukunimi
Yhteyshenkilön sähköposti
Yhteyshenkilön puhelin
Katuosoite *
Talo *
Rappu
Huoneisto
Postinumero *
Toimipaikka *
Sähköposti *
Puhelin *
Asiakasnumero
Sopimuksen palautusosoite on eri kuin liittyjän osoite


Asiakkaan tiedot
Muokkaa
Kuluttaja
Yritysasiakas
Etunimi *
Asko
Sukunimi *
Makkara
Katuosoite *
Liittyjän osoite
Talo *
9
Rappu
Huoneisto
Postinumero *
90630
Toimipaikka *
OULU
Sähköposti *
eiole@huuhaa.xxx
Puhelin *
3213213211
Hetu *
110990-2355
Asiakasnumero
Sopimuskumppani(t)
Etunimi
Sukunimi
Sopimus
Kumppani
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Postitoimipaikka
OULU


Asiakkaan tiedot
Peruuta
Tallenna
Kuluttaja
Yritysasiakas
Yrityksen nimi *
Y-tunnus *
Yhteyshenkilön etunimi
Yhteyshenkilön sukunimi
Yhteyshenkilön sähköposti
Yhteyshenkilön puhelin
Katuosoite *
Talo *
Rappu
Huoneisto
Postinumero *
Toimipaikka *
Sähköposti *
Puhelin *
Asiakasnumero
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Toimipaikka
OULU


Asiakkaan tiedot
Muokkaa
Kuluttaja
Yritysasiakas
Etunimi *
Asko
Sukunimi *
Makkara
Katuosoite *
Liittyjän osoite
Talo *
9
Rappu
Huoneisto
Postinumero *
90630
Toimipaikka *
OULU
Sähköposti *
eiole@huuhaa.xxx
Puhelin *
3213213211
Hetu *
110990-2355
Asiakasnumero
Sopimuskumppani(t)
Etunimi
Sukunimi
Sopimus
Kumppani
Sopimuksen palautusosoite on eri kuin liittyjän osoite
Nimi
asd
Katuosoite
Sopparin palOsoite
Talo
9
Rappu
Huoneisto
Postinumero
90100
Postitoimipaikka
OULU