Ernesto Koskinen

Dataperusteinen lomake AngularJS-direktiivillä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

20.9.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Ernesto Koskinen Dataperusteinen lomake AngularJS-direktiivillä 24 sivua + 2 liitettä 20.9.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Yliopettaja Markku Karhu

Insinöörityön tavoitteena oli toteuttaa suomalaiselle ohjelmistoyritykselle mahdollisuus tuottaa dataperusteisia lomakkeita AngularJS-kehykseen.

Projekti toteutettiin olemassa olevan logiikkapalvelimen mukaisesti kehitteillä olevan käyttöliittymän direktiivikomponentiksi. Kyseessä oli hajautettu järjestelmä, jossa sama logiikkapalvelin tarjosi dataa useille eri käyttöliittymille. Käyttöliittymät sisälsivät myös oman pienemmän palvelintoteutuksensa.

Projekti osoittautui tarpeelliseksi työkaluksi tavallisten lomakkeiden luonnin tueksi. Sen avulla voitiin ylläpitää ja tyylitellä lomakkeita suoraan direktiivissä, jolloin tyyli- ja logiikkamuutokset heijastuivat koko verkkosovellukseen.

Projekti valmistui määritellyn aikarajan puitteissa, ja se on käytössä yrityksen useimmissa peruslomakkeissa, joiden käyttöliittymä on rakennettu AngularJS-kehyksen päälle.

Avainsanat	AngularJS, lomake, direktiivi

Author Title Number of Pages Date	Ernesto Koskinen Data driven forms using AngularJS directive 24 pages + 2 appendices 20 September 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Markku Karhu, Principal Lecturer

The goal of this study was to implement an AngularJS directive that provides data-driven forms for a user interface.

Programming in the project was done using a decentralized system containing one logic server that offered data to a number of different user-interfaces. Each user-interface also contained its own smaller server logic.

The project helped to maintain and stylize the forms directly in the directive, so that the style and logic changes were implemented over the entire web application.

The project was completed within the time frame specified and is in use in most of the company's basic forms, which interface is built on top of the frame of AngularJS.

Keywords	AngularJS, Form, Directive

Sisällys

Lyhenteet

AJAX	Asynchronous JavaScript And XML. Tiedon ja tiedostojen siirtoon vakiintunut tekniikka, jonka avulla voidaan muuttaa vain osaa verkkosivun sisällöstä ilman, että koko sivua tarvitsee ladata uudestaan.
CSS	Cascading Style Sheets. Tiedostomuoto, joka sisältää joukon määrittelytietoja liittyen sivuston elementtien ulkoasuun.
DOM	Document Object Model. Näkyvällä sivulla kulloinkin sijaitseva HTML-elementtipuu.
HTML	HyperText Markup Language. Sivuston merkintäkieli. Sivustojen elementtien ja sisällön paikat määrittävä kuvauskieli.
JSON	JavaScript Object Notation. Joukko avain-arvo pareja. Tiedonsiirrossa yleisesti käytettävä tiedostomuoto.
Validaattori	Ohjelman looginen osa, joka toteutuksesta riippuen määrittelee syötteen tapauskohtaisen oikeellisuuden.

1 Johdanto

Insinöörityössä tehtävän lomakkeen on tarkoitus rakentaa AngularJS-direktiivi, joka osaa rakentaa verkkosivustoon lomakkeita saadun datan perusteella. Lomake on kokoelma kenttiä, joihin odotetaan usein käyttäjän vaikutusta, esimerkiksi tekstikenttä, johon käyttäjä kirjoittaa oman nimensä.

Tarkoitus on määritellä hyvin yleiset tavat esittää erityyppisiä kenttiä, kuten tekstikentät, numerokentät, alasvetolistat ja valintaruudut, jolloin voidaan aina yksinkertaisissa lomakkeissa niiden määrittelyn sijasta käyttää tätä direktiiviä käyttöliittymän muodostukseen. Tarkoitus on myös saavuttaa sivuston yhteneväinen ulkoasu, kun kaikki normaalilomakkeet käyttävät tässä direktiivissä kerran määriteltyjä tyylisääntöjä ja toiminnallisuutta.

Direktiivin ei ole tarkoitus olla ratkaisu kaikkiin verkkosivuston lomakkeisiin, vaan ainoastaan lomakkeisiin, jotka käyttäytyvät keskenään samalla tavalla, ehkä voisi sanoa sivustolle ominaisella tavalla. Tämä tapa määritellään siten, että dataperusteinen lomake säästää kehittäjältä aikaa ja vaivaa tilanteissa, joiden toistaminen olisi puuduttavaa.

Lomakerunko on myös paljon helpommin ylläpidettävä, kun yksi muutos yhteen kenttään vaikuttaa koko verkkosivuston samaa runkoa käyttäviin lomakkeisiin.

Erikoisia toiminnallisuuksia direktiivin ei ole tarkoitus tukea, vaan ne jätetään edelleen erikseen ohjelmoitaville muille direktiiveille. Muuten tästä direktiivistä tulisi aivan liian monimutkainen, eikä se enää palvelisi yksinkertaisten lomakkeiden luontia.

2 Käytetyt tekniikat

Insinöörityön keskeisiä tekniikoita ovat AngularJS-ohjelmointikehys ja HTML5. Näiden lisäksi tyylien pohjana käytetään Bootstrap-tyylikirjastoja.

Koska insinöörityö käsittelee nimenomaan käyttöliittymäpuolen toteutusta, jätän palvelinpuolen tekniikkojen esityksen vähälle. Dataperusteisien lomakkeiden tarkoitus on nimenomaan rakentaa lomake datasta, riippumatta taustalla olevan logiikkapalvelimen toteutuksesta. Insinöörityö tehdään .NET Core -palvelinta varten, jossa tietokantana toimii T-SQL.

2.1 AngularJS-ohjelmointikehys

AngularJS on Googlen kehittämä vapaan lähdekoodin ohjelmointikehys, joka on helppo sekoittaa ohjelmointikirjastoon, koska se on kevyempi kuin ohjelmointikehykset yleensä. Se mahdollistaa verkkosivuston ohjelmoinnin enemmän sovelluksena kuin staattisena sivustona, jolloin lopputuloskin tuntuu enemmän työpöytäsovellukselta kuin perinteiseltä verkkosivulta. [1.]

AngularJS jakaa verkkosovelluksen komponentteihin, joita oikein käytettäessä saavutetaan parempi skaalautuvuus ja ylläpidettävyys sekä mahdollistetaan sivun eri osien itsenäinen päivittäminen ilman koko DOM-puun latausta tai virkistystä navigointien välissä. Tällöin myös selaimen välimuisti on paremmin ohjelmoijan käytettävissä, kun sitä voidaan käyttää koko istunnon ajan sen sijaan, että jokaisen sivun sisäisen linkin painalluksen jälkeen virkistettäisiin välimuisti tyhjäksi, kuten perinteisillä verkkosivuilla on tapana. Tämä ei sinänsä ole aivan uusi asia verkkokehityksessä, onhan ennen sama mahdollistunut käyttämällä AJAX-tekniikoita. AngularJS tarjoaa AJAXin ominaisuuksien lisäksi uuden tehokkaan tavan kirjoittaa verkkosovelluksia, se nitoo kaikki tärkeimmät menetelmät yhdeksi paketiksi, jota käytetään yhdellä tavalla. [1.]

AngularJS tarjoaa myös datan molemminpuoleisen sitomisen (two way databinding), jolloin JavaScriptissä määritellyn muuttujan arvon muuttuessa päivittyy uusi arvo myös näkymään. Tai jos näkymässä oleva syöttökentän arvo on sidottu JavaScript-muuttujaan, käyttäjän manipuloidessa syötettä myös JavaScript-puolen muuttujan arvo muuttuu.

AngularJS on täysin JavaScriptillä kirjoitettu, joten se tukee sekä PC- että mobiiliselaimia. Sivujen itsensä skaalautuvuuteen eri resoluutioille AngularJS ei sinänsä ota kantaa, vaan skaalautuvuus tapahtuu käyttäen vakiintuneita HTML5-tekniikoita, kuten tyylitiedostoja, joiden pohjana tässä työssä käytetään Bootstrapia. [1.]

Direktiivi

Direktiivi tarkoittaa sellaista osaa AngularJS-verkkosovelluksessa, joka sisältää yhden määritellyn mahdollisimman autonomisen toiminnallisuuden sivustossa ja jota voidaan käyttää useamman kerran sivulla mutta se toimii hiukan eri tavalla riippuen käyttöpaikakassa annetuista parametreista. Direktiivin tarkoitus on luoda DOM-elementtiin toiminnallisuus tai muokata HTML-elementtiä ajonaikaisesti. [2.]

Esimerkiksi yhteystietokortti voisi olla direktiivi, jolle annetaan parametrina henkilön tiedot, jolloin DOM-puu rakentuu visuaalisesti samannäköisenä direktiiviä toistettaessa, mutta sisältää eri henkilöiden nimen ja osoitteen. Tällöin samaa HTML-runkoa ei tarvitse kirjoittaa useampaan kertaan, kun AngularJS rakentaa tarvittavat HTML-elementit, jotka kerran on direktiivissä määritelty.

2.2 Bootstrap-tyylit

Bootstrap on opinnäytetyön kirjoitushetken suosituin HTML-, CSS- ja JavaScript-kehys, ja sillä saavutetaan verkkosivuston responsiivisuus mobiililaitteissa ja PC:n työpöydällä. Tässä insinöörityössä Bootstrapilla tarkoitetaan CSS-kirjastoa eikä käytetä Bootstrapin tarjoamaa JavaScript-kehystä, joka tarjoaisi erilaisia valmiita käyttöliittymäkomponentteja esimerkiksi ponnahdusikkunoita ja listoja varten. [3.]

Bootstrap-tyylit itsessään ovat mielestäni hyvä lähtökohta ohjelmistokehittäjälle perinteisten HTML-tyylien sijaan, koska ne ovat moderniin ulkoasuun pyrkiviä jo lähtökohtaisesti. Tällöin kehittäjän lähtötilanne on usein parempi, kun sivuston tyylit ovat valmiiksi oikein skaalautuvia mobiililaitteissa ja lähempänä totuttua ulkoasua. Bootstrap-tyyleistä on myös helppo laajentaa omia tyylejä. [4.]

3 Datamalli

Jotta lomake pystyisi rakentamaan itsensä täysin datan perusteella, palvelimelta tulevaan dataan on ujutettu kenttäkohtaisia määrittelytietoja, joista vapaavalintaisia tietueita ovat ”CodeTableReference”, jota käytetään myöhemmin pudotusvalikon listan muodostukseen, ”DisplayName”, jota käytetään ensisijaisena otsikkotietueena, ja ”DisplayOrder”, joka määrittelee kenttien tulostusjärjestyksen (kuva 1).

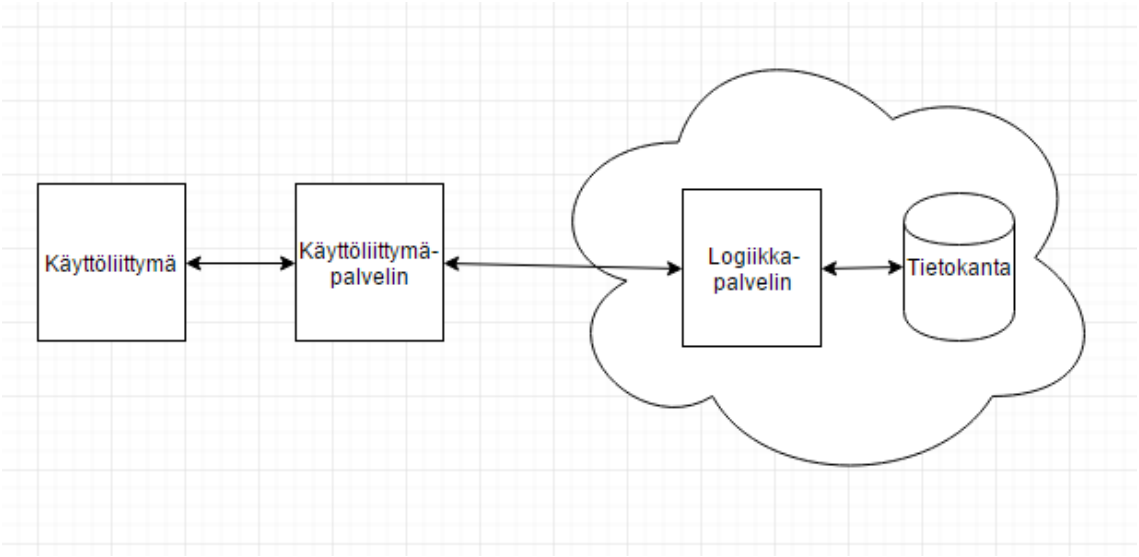
```
{
    "IsEditable": true,
    "IsDeletable": false,
    "Fields": [
      {
          "FieldName": "",
          "Value": null,
          "Type": "",
          "CodeTableReference": "00000000-0000-0000-0000-000000000000",
          "DisplayName": "",
          "DisplayOrder": 0,
          "IsRequired": false,
          "IsNullable": true,
          "IsReadOnly": false,
      },
    ]
}
```


Kuva 1. JSON-muotoiset kenttäkohtaiset määrittelytiedot.

Omassa palvelinarkkitehtuurissani oli logiikkapalvelin, joka ei vielä muodosta määrittelytietokehystä vaan sisältää pelkästään arvoja. Se keskustelee käyttöliittymäpalvelimen kanssa, joka on sama palvelin, joka antaa myös tarvittavat HTML- ja JavaScript-tiedostot käyttöliittymälle. Käyttöliittymäpalvelin vasta rakentaa määrittelykehyksen kutsun perusteella.

Koska käyttöliittymässä on käytössä AngularJS:n tarjoama datan sidonta näkymän ja JavaScript-koodin välillä, voidaan näitä määrittelyitä myös ajonaikaisesti muuttaa. Jos esimerkiksi käyttäjälle tarjottaisiin lukittuja kenttiä, jotka käyttäjä itse voisi avata lukituksen avaus -napista, voidaan datan saamisen jälkeen asettaa JavaScriptissä kenttien ”IsReadOnly” muuttujan arvoksi ”true”, minkä jälkeen nappia painamalla käydään kentät uudestaan läpi ja asettaan muuttuja takaisin oletusarvoonsa. Tällöin käyttöliittymä seuraa automaattisesti muutosten perässä lukiten ja avaten kentät.

Kuvailtu palvelinten hajautus on mielestäni hyvä tehdä skaalautuvissa isommissa verkkosovelluksissa. Tällöin saavutetaan logiikkapalvelimen eriytys käyttöliittymäkohtaisista komponenteista, jolloin saman logiikan perusteella voidaan jatkossa tehdä useampia käyttöliittymiä (kuva 2).



Kuva 2. Opinnäytetyössä käytetty palvelinmalli.

Jos rakennettava verkkosovellus on tarkoitus pitää pienempänä kokonaisuutena, määrittelytiedon sitomisen voi toteuttaa myös logiikkapalvelimena. Tällöin on hyvä kysyä, tarvitaanko dataperusteisia lomakkeita sivustolla, jos lomakkeita ei sivustolle määritellä muutamaa enempää.

4 Dataperusteinen lomake

4.1 Perinteinen ja dataperusteinen lomake

Kaksi perinteistä hyvin yksinkertaistettua yhteystietolomaketta voisi näyttää HTML-kuvauskielessä kuvan 3 mukaiselta.

```
<form>
    <label>Nimi</label>
    <input type="text" value="Matti Malli"><br>
    <label>Osoite</label>
    <input type="text" value="Mallikuja 5"><br>
    <label>Opiskelunumero</label>
    <input type="number" value="1231123"><br>
</form>
<form>
    <label>Nimi</label>
    <input type="text" value="Mikko Malli"><br>
    <label>Osoite</label>
    <input type="text" value="Mallikuja 3"><br>
    <label>Opiskelunumero</label>
    <input type="number" value="321321"><br>
</form>
```

Kuva 3. Kaksi lomaketta, jotka molemmat sisältävät kolme syöttökenttää. Perinteinen HTML-toteutus.

Kuvan 3 tapauksessa on esitelty kaksi henkilöä ja molemmilla henkilöillä on nimen lisäksi osoite ja opiskelijanumero. Ohjelmoijan täytyy tässä toteutuksessa tietää, että näytettävät lomakkeet sisältävät opiskelijoiden tietoja, mitä tiedot itsessään ovat ja kaiken lisäksi kirjoittaa samat HTML-määritykset (label, input ...) yhä uudestaan opiskelijamäärän lisääntyessä.

Dataperusteisilla lomakkeilla pyritään ratkaisemaan kaikki nämä ongelmat. Ensimmäinen vaihe olisi käyttää AngularJS:n tarjoamaa muuttujan sitomista, jolloin ohjelmoijan ei enää tarvitse tietää itse opiskelijoiden nimiä, osoitteita ja puhelinnumeroita, kun ne tulevat palvelimelta. Seuraava vaihe olisi käyttää AngularJS:n tarjoamaa taulukon toistoa, jolloin ei tarvitsisi tietää, montako opiskelijaa näytetään.

Viimeinen olisi kirjoittaa eristetty lomakedirektiivi, jolloin ei tarvitse enää tietää, että kyseessä on opiskelijan tietoja ollenkaan, vaan sokeasti datan perusteella voi antaa direktiivin hoitaa koko lomakkeen muodostus saadun datan perusteella.

Kuvailtu tavoitetilanne voisi esimerkiksi HTML-kuvauskielessä näyttää kuvan 4 mukaiselta.

```
<div ng-repeat="opiskelija in opiskelijat">
    <formpanel title="{{opiskelija.name}}" fields="opiskelija"></formpanel>
</div>
```

Kuva 4. Määrittelemätön määrä lomakkeita, jotka sisältävät määrittelemättömän määrän kenttiä. Dataperusteinen toteutus.

Kuvan 4 esimerkissä HTML:n sekaan on lisätty AngularJS:n omaa HTML-syntaksia. AngularJS:ssä on olemassa valmiita yleiskäyttöisiä direktiivejä, jotka alkavat sanalla ”ng-”, ja muuttujien sitominen näkymästä JavaScript-puolelle kirjoitetaan antamalla muuttujan nimi ”{{}}”-merkkien sisälle. Tämän esimerkin perinteisestä HTML-kielestä poikkeavat termit ovat seuraavat:

ng-repeat

Toistaa Javascriptissä määritellyn taulukon soluja. Tätä voi hyvin verrata ohjelmointisilmukkaan, jossa taulukon solut käydään läpi ja jokaiselle solulle määritellään tietty toteutus. Tässä tapauksessa käydään kaikki opiskelijat läpi ja annetaan koko opiskelijasolu dataperusteiselle lomakkeelle. Taulukon nimi on ”opiskelijat”, ja muuttuva solu on nimeltään ”opiskelija”.

formpanel title=..

Tämä on dataperusteiselle lomakkeelle antamani nimi, koska tässä sovelluksessa kaikki lomakkeet muistuttavat tyylillisesti paneeleita. Nimi on vapaasti valittavissa, ja sen määrittelyä avaan enemmän JavaScript-osiossa.

fields=”opiskelija”

Dataperusteisen lomakkeen parametriosa, jälleen määritelty JavaScript-puolella. HTML-osuudessa annetaan muuttujaan ”fields” arvoksi aina yksi opiskelijasolu, joka sisältää itsessään omat kenttänsä. Tämä datarakenne on määritelty luvussa 3.

Tässä toteutuksessa opiskelijoita voi olla datassa kuinka monta tahansa eri tiedoilla ja direktiivi generoisi lopulta aivan saman HTML-koodin kuin edellinen staattinen esimerkki, jos opiskelijadatassa olisi kaksi tietuetta, Mikko ja Matti.

Jos palvelimelta saadun datan sisältö tai määrittely muuttuu, käyttöliittymässä olevan lomakkeen kentät muuttuvat sisällöltään, määrältään tai tyypeiltään datan muutosten mukaisiksi ilman toimenpiteitä käyttöliittymäkoodissa.

4.2 AngularJS-direktiivi

4.2.1 Validaattoridirektiivi

Ennen varsinaista dataperustaista lomakedirektiiviä tehdään pienempi direktiivi tekstisyötekenttää varten. Tämän direktiivin tarkoitus on toimia ”input”-elementillä siten, että käyttäjän syöttäessä kenttään merkkejä direktiivi estää numeroiden ja erikoismerkkien syötteen kokonaan ja estää jopa niiden muodostumisen kenttään.

Seuraavaksi esitellään tässä kontekstissa AngularJS-direktiivin perusominaisuudet. Tämä toteutus on suppeampi kuin itse dataperusteisen lomakkeen direktiivissä.

Ensimmäiseksi kirjoitetaan AngularJS-määrittelyiden mukainen direktiivirunko (kuva 5). Tässä tapauksessa AngularJS-ohjelmamuuttujan nimeksi on annettu ”app”, joten direktiivi sidotaan pisteellä siihen kiinni. [5]

```
app.directive('dynamicTextInput', function () {
    return {
        require: 'ngModel',
        restrict: 'A',
        link: function (scope, element, ngModelCtrl) {
```

Kuva 5. AngularJS-direktiivin määrittelyosa.

Direktiivin nimi on tässä ”dynamicTextInput”, joka käyttäytyy eri lailla täällä kuin HTML-puolella. HTML-standardissa elementtien tai attribuuttien nimissä ei käytetä camel casingia, kun taas JavaScript-standardissa käytetään. Tämän vuoksi AngularJS kääntää automaattisesti tässä annetun nimen HTML-puolella käytettäessä muotoon ”dynamic-text-input”.

Seuraavaksi määritellään direktiivin sisältö. Direktiiviluokka määritellään suoraan ”return”-lohkon sisään.

require: ’ngModel’

Kun annetaan direktiivimäärittelyssä ”ngModel”-muuttuja suoraan, sitä ei tarvitse enää parametrina HTML-puolelta antaa direktiiville. Näin direktiivi saa automaattisesti käyttöönsä muuttujan ”element”, joka sisältää HTML-elementin, johon direktiivi itse on sidottu.

restrict: ’A’,

Restrict-osa direktiivimäärittelyssä antaa mahdollisuuden määritellä, miten direktiiviä HTML-puolella käytetään. Vaihtoehdot ovat ”A” (attribuutti), ”E” (elementti), ”C” (CSS luokka) ja ”M” (Kommentti). Näitä voi myös yhdistellä antamalla arvoksi vaikka ’AE’, jolloin direktiiviä voi käyttää HTML:n seassa joko omana elementtinään tai toisen elementin attribuuttina.

Validaattorille valitaan arvo ’A’, koska halutaan sitoa direktiivi olemassa olevaan elementtiin, jota direktiivi sitten validoi. Tämä tarkoittaa käytännössä, että direktiiviä käytetään HTML:ssä kuvan 6 mukaisesti.

```
<input
       type="text"
       dynamic-text-input />
```

Kuva 6. Validaattori kiinnitettynä HTML-syöttökenttään.

link: function(scope, element, ngModelCtrl) {

link-funktio on direktiivin lähtöpiste, joka määrittelee omat parametrinsa AngularJS-sääntöjen mukaisesti.

Ensimmäinen parametri ”scope” on eristetty AngularJS scope-olio, johon sidotaan direktiivikohtaiset muuttujat sekä metodit, joita saatetaan kutsua HTML-näkymän puolelta. Scopea käytetään kuten mitä tahansa JavaScript-oliota, ja siihen esimerkiksi muuttujan sitominen tehdään kuvan 7 mukaisesti. [6]

```
scope.muuttuja = "arvo";
```

Kuva 7. Esimerkki AngularJS:n ”scope” -muuttujan käytöstä.

Seuraavaksi parametrina määritellään ”element”, joka on itse HTML-elementti, johon validaattori on sidottu. Tätä kautta päästään myös elementin kulloinkin kirjoitettuun syötteeseen käsiksi.

Viimeisenä parametrina injektoidaan ngModelCtrl, joka on saatavilla ”require”-määrittelyn ansiosta. Se sisältää suoran viittauksen elementin HTML-näkymään. Sillä on myös joukko AngularJS-apumetodeita, jotka alkavat merkillä $.

Link-funtion sisälle määritellään itse direktiivin toteutus. Koska kyseessä on validaattori, toteuttaa direktiivi validointifunktion ja antaa sen suoraan AngularJS-parserille, joka käynnistyy aina elementin mallin muuttuessa (kuva 8).

```
function inputChange(text) {
    var transformedInput = text.replace(/[^A-Za-z ]/g, '');
    if (transformedInput !== text) {
        ngModelCtrl.$setViewValue(transformedInput);
        ngModelCtrl.$render();
    }
    return transformedInput;
}
ngModelCtrl.$parsers.push(inputChange);
```

Kuva 8. Direktiivin looginen toteutus.

Tässä parametri ”text” sisältää jo seuraavan halutun merkkijonon, jota voidaan sitten funtiossa muokata ja lopulta palauttaa syöttökentän näkymään tuleva tekstijono. Tässä toteutuksessa käytetään apuna JavaScript ”string”-prototyypin funktiota ”replace”, joka tässä tapauksessa korvaa kaikki muut kuin annetun tyyppiset arvot tyhjällä merkillä, ’’, eli suodattaa sen pois merkkijonosta. Jos muokattu merkkijono poikkeaa alkuperäisestä, asetetaan muokattu merkkijono näkymään apumetodilla $setViewValue, minkä jälkeen elementti renderöidään uudestaan AngularJS:n apumetodilla $render.

”$parsers.push”-osuus lisää funktion AngularJS:n omien parserien joukkoon, jolloin se tulee suoritetuksi käyttäjän kirjoittaessa.

4.2.2 Dataperusteinen lomake

Lomakkeen direktiivi alustetaan kuten validaattorin, seuraavaksi esiteltäviä muutamia poikkeuksia lukuun ottamatta (kuva 9).

```
app.directive('dynamicform', function ($http) {
    return {
        restrict: 'E',
        templateUrl: 'dynamicform.html',
        scope: {
            fields: '=',
        },
        link: function (scope, element) {
```

Kuva 9. Dataperusteisen lomakkeen direktiivin määrittelyosa.

restrict: ’E’,

Tässä osassa annetaan tällä kertaa arvo ”E” eli element. Tällöin direktiiviä käytetään aina omana elementtinään, nimellä ”dynamicform”. Direktiiviä käytetään HTML-puolella kuvan 10 mukaisesti.

```
<dynamicform fields="formFields"></dynamicform>
```

Kuva 10. Direktiivin käyttö ”restrict”-muuttujan arvon ollessa ”E”.

scope: {

Tällä kertaa direktiiville annetaan myös scopeen valmiiksi määriteltyjä muuttujia. Tällä tavalla voidaan määritellä parametreja direktiiville. Tässä tapauksessa annetaan parametrina itse dataolion kentät, joiden perusteella pystytään HTML-osuudessa rakentamaan itse lomake. ”fields” on siis muuttujan nimi, johon parametri asettuu direktiivin omassa ”scope”ssa, ja ’=’ merkki on AngularJS:n omaa syntaksia, joka kertoo, että parametrin odotetaan olevan olio. Esimerkiksi merkkijonoa odottaessa tämä arvo olisikin ’&’. [2.]

Direktiivin sisäinen toteutus tässä tapauksessa koskee JavaScript-puolella lähinnä pudotusvalikkoja, koska ne ovat esimerkissä ainoita elementtejä, joiden toteutus ei tule luonnostaan pelkän AngularJS:llä rikastetun HTML:n avulla (kuva 11).

```
scope.codetableContainer = [];

scope.getCodeTable = function (field) {
    if (!scope.codetableContainer[field.CodeTableReference]) {
        scope.codetableContainer[field.CodeTableReference] = [];
        $http.get(
            createRelativeUrl("api/Common/getCodeTableById?tableId=" + field.CodeTableReference)).
        success(function (data, status, headers, config) {
            populateCodeTable(field.CodeTableReference, data.CodeTableRows, field);
        });
    }
};

var populateCodeTable = function (tableId, data, field) {
    scope.codetableContainer[tableId] = data;
    updateUINames(field, data);
}

scope.selectDropdownValue = function (field, value) {
        field.Value = value;
}
```

Kuva 11. Dataperusteisen lomakkeen direktiivin looginen toteutus. Pudotusvalikon taulujen haku palvelimelta.

scope.codetableContainer = []

Pudotusvalikkoja varten on hyvä pitää ajonaikaisessa muuttujassa tieto jo ladatuista listoista, jottei hakuja tehtäisi aina uudestaan käyttäjän avatessa jo aiemmin avattua pudotusvalikkoa. Tässä käytän listan alkioerottimena tauluun liittyvää GUID-arvoa, jonka puuttuessa haetaan lista palvelimelta ja asetetaan se ”Value”-muuttujaan.

4.3 Direktiivin HTML-toteutus

4.3.1 Kehys

Direktiivin HTML-osuus tehdään omaan tiedostoonsa. Sitä kutsutaan direktiivin HTML-malliksi (directive template). Tässä tapauksessa malli käy läpi parametrina annetun datan kentät, joista jokaiselle valitaan sopiva syöte-elementti ja se sidotaan lopuksi JavaScript-muuttujaan sekä näytetään DOM-puussa käyttäjälle. Tiedoston alku näyttää kuvan 12 mukaiselta.

```
<form name="dynamicForm">
    <div ng-if="fields"
         ng-repeat="field in fields.Fields | orderBy:'+DisplayOrder'"
         ng-init="fields.title = title">
```

Kuva 12. Dataperusteisen lomakkeen HTML-kehys.

Ensimmäisellä rivillä määritellään lomake alkaneeksi tavalliseen HTML-tapaan, annetaan staattinen nimi ”dynamicForm”, joka on jokaisella lomakkeella sama, jotta voi halutessaan kysyä kaikkia sivulla tällä hetkellä näkyviä lomakkeita välimuuttujaan. Nimen voi myös jättää pois, jos tietää, ettei tällaiselle ominaisuudelle ole tarvetta.

Seuraavalla rivillä määritellään jokaista kenttää ympäröivä kehys. Rivillä lukeva ”ng-if” on Angularin valmisdirektiivi, joka estää elementin näkymisen kokonaan, jos parametrina annetulla oliolla ei ole ”fields”-nimistä listaa.

Seuraavaksi edelleen samassa kehyselementissä määritellään silmukka ”ng-repeat”. Se käy kaikki annetun olion ”fields”-nimisen taulukon solut läpi ja järjestää ne apumuuttuujan ”DisplayOrder” mukaisesti. Jos ”DisplayOrder”-muuttujaa ei löydy datasta itsestään, se voidaan ujuttaa vaikka kesken ajon oliolle tai jättää kokonaan pois, jos kenttien järjestyksellä ei ole väliä.

Seuraavan rivin ”ng-init” on jälleen valmisdirektiivi, jolle annettu toiminto suoritetaan vain kerran, kun lomake luodaan. Tässä tapauksessa asetetaan vain lomakkeelle annettu otsikko haluttuun muuttujaan, ”fields”-olion juureen.

4.3.2 Kentät

Seuraavaksi direktiivin HTML-osuuden on pystyttävä päättelemään, millaista kenttää käytetään. Opinnäytetyön puitteissa määritellään direktiivi pystymään rakentamaan kolme erilaista kenttää: Tekstikenttä, Tekstisyötekenttä ja Pudotusvalikko, jota toteutetaan tekemällä tarkennuskutsu palvelimelle direktiivin sisällä.

Valitsin nämä kenttätyypit työhön esimerkeiksi, koska mielestäni ne edustavat hyvin mahdollisten monimutkaisuuksien eri tasoja, yksinkertaisemmasta monimutkaisempaan.

4.3.3 Tekstikenttä

Vaikka lomakkeen jokaisella syötekentällä voi olla otsikko, tehdään silti puhdas tekstikenttä ilman syötemahdollisuutta, jotta voidaan otsikoida osa-alueita myös lomakkeen sisällä. Tulos näyttää siis tavalliselta tekstiltä (kuva 13).

```
<!--Text-->
<span ng-if="field.Type == 'Text'"
      class="dynamic-text">
    {{field.Value}}
</span>
```

Kuva 13. Direktiivin tekstikentän HTML-toteutus.

span…

Yleinen HTML-elementti, joka soveltuu esimerkiksi tekstin juurielementiksi.

ng-if=”field.Type == ’Text”

Kenttätasolla ensimmäinen direktiivilooginen asia on päätellä kentän elementtityyppi. Oliolla field on siksi datassa määritelty avain ”Type”, joka on tässä tekstityyppinen kenttä, jolla on vaihtoehdot ”Text”, ”InputText” ja ”dropdown”. Tämä on vain yksi tapa ratkaista ongelma, jota tulisi aina lähestyä palvelimen antaman datan perusteella.

class=”dynamic-text”

CSS-tyyliluokka. HTML:n normaalisääntöjen mukaisesti.

{{field.Value}}

Tässä tapauksessa jokaisella ”fields”-taulukon oliolla on datassa määritelty arvo. Se on tässä tapauksessa itse teksti, joka tulee käyttöliittymään tulostumaan.

4.3.4 Tekstisyötekenttä

Tekstisyötekenttää voisi sanoa ensimmäiseksi oikeaksi lomakekentäksi, sillä se on tarkoitettu käytäjän syötettä varten. Tässä esimerkissä rajoitetaan käyttäjän syötettä kuitenkin siten, ettei hän saa syöttää muuta kuin kirjaimia. Numerot ja erikoismerkit siis kielletään, jotta päästään tarkastelemaan myös kentän validointia (kuva 14).

Kuva 14. Direktiivin tuottama käyttöliittymän tekstisyötekenttä. Esimerkki.

Kuvassa 15 on jo hieman enemmän rivejä, jotta voidaan päätellä kaikki kentän ominaisuudet, joita yleistilanteissa voitaisiin tarvita.

```
<!--String Input-->
<label ng-if="field.Label">
    {{ field.Label }}
</label>
<input ng-if="field.Type == 'TextInput'"
       type="text"
       class="dynamic-input"
       ng-required="field.IsRequired"
       ng-disabled="field.IsReadOnly"
       ng-model="field.Value"
       dynamic-text-input />
```

Kuva 15. Direktiivin tekstisyötekentän toteutus.

```
<label ng-if="field.Label">{{field.Label}}
```

Tässä on tehty mahdolliseksi datassa määritellä syötekentälle otsikko. Elementin ”ng-if” varmistaa tällä kertaa, että jos otsikkoa ei ole kentälle määritelty, ei myöskään elementtiä rakenneta sivulle. Jälleen kaarisuluin on määritelty datan sidonta itse otsikon sisältöön. Esimerkissä ”field.Label” on arvoltaan ”Nimi”.

```
ng-required="field.IsRequired"
```

AngularJS tarjoaa valmiiksi jo jonkinlaisia apudirektiivejä lomakkeen validointiin, tässä tapauksessa direktiivin ”ng-required”, jonka parametrina annetun arvon ollessa ”true” kenttä merkitään virheelliseksi kentän ollessa tyhjä. Jos field-oliolla ei ole avainta ”IsRequired”, JavaScript-sääntöjen mukaan kentän arvo on silloin ”undefined”, jonka JavaScript tulkitsee myös ”false”ksi.

Mallikuvan 14 reunat ovat punaiset, koska kentän ”field” muuttujan avain ”IsRequired” on arvoltaan ”true”. Toisella kentällä saman lomakkeen sisällä ”IsRequired” voisi olla vaikka ”false”, jolloin kun se on tyhjä, punaisia reunuksia ei synny ja kenttä merkitään käyttökelpoiseksi.

```
ng-disabled="field.IsReadOnly"
```

Jälleen AngularJS:n sisäinen apudirektiivi. Tätä käytetään pitkälti samoin kuin edellistä, paitsi tässä arvon ollessa ”true” kentän käyttö estetään, eikä käyttäjä pääse itse muuttamaan kentän arvoa. Sisäisesti tämä asettaa DOM:ssa input-elementille vain HTML-pe-rusattribuutin ”disabled”.

```
ng-model="field.Value"
```

Tämä AngularJS:n sisäinen direktiivi toteuttaa syötekentän sisällön sitomisen JavaScript-muuttujaan. Jos käyttäjä syöttää kenttään nimensä, nimi päivittyy myös tämän ansiosta ”field.Value”-muuttujaan reaaliajassa.

dynamic-text-input

Opinnäytetyön JavaScript-osiossa määritelty validaattori. Direktiiviä käytetään HTML-elementin attribuuttina direktiivin määrittelyn mukaisesti, mikä erottaa sen käyttötavallaan konkreettisesti esimerkiksi itse ”formpanel”-direktiivistä. Näin käytettynä direktiivi saa jo parametrinaan isäntäelementin, tässä tapauksessa ”input”-kentän, joten mitään ei tarvitse syöttää itse parametrina direktiiville.

4.3.5 Pudotusvalikko

Lomakkeen pudotusvalikko toteutettiin, ettei sisältöä ladata palvelimelta ollenkaan, jos käyttäjä ei koskaan klikkaa sitä auki. Tämä saavutettiin tässä toteutuksessa antamalla datassa kentälle arvo ”CodeTableReference”, joka on GUID-tyyppinen muuttuja. Palvelintoteutuksessa jokainen GUID viittaa tiettyyn tauluun, jolloin rajapintaan on rakennettu GET-metodi, joka palauttaa halutun taulun GUID-arvon mukaan. Esimerkkikuvassa (kuva 16) taulu sisältääkin listan eri kaupunkeja.

Kuva 16. Direktiivin tuottama käyttöliittymän tekstisyötekenttä. Esimerkki.

Kuvien 17 ja 18 elementeille määritellyt CSS-luokat (”class”) on määritelty Bootstrap-tyylisääntöjen mukaisiksi.

```
<!--Guid (list of items from codetable)-->
<div class="dropdown"
     ng-if="field.Type == 'Dropdown'"
     ng-init="getCodeTable(field)">
    <div ng-if="field.ready">
        <!--Button to open dropdown-->
        <button type="button" class="form-control dynamic-form-control btn btn-default dropdown-toggle"
                ng-disabled="field.IsReadOnly"
                data-toggle="dropdown">
            <span>{{field.UIName}}</span> <span class="caret"></span>
        </button>
```

Kuva 17. Direktiivin pudotusvalikkokentän toteutus. Osa 1.

ng-init=”getCodeTable(field)”

Jos kentän tyyppi on ensin varmistettu tyypiksi ”dropdown”, elementtiä rakentaessa ajetaan direktiivin ”getCodeTable(field)”-funktio, jonka toteutus määriteltiin opinnäytetyön AngularJS JavaScript -osiossa. Tämän rivin avulla varmistetaan, että direktiivi hakee taulun palvelimelta pudotusvalikon sisällöksi.

ng-if=”field.ready”

Tässä käytetään ajonaikaista muuttujaa ”ready”, joka on määritelty direktiivin JavaScript-puolella. Tällöin kentän sisältö kootaan vasta, kun taulu itsessään on ladattu.

{{field.UIName}}..

JavaScript-puolella ujutetaan taulun koonnin yhteydessä jokaiselle taulun arvolle näkyvä nimi, joka voidaan koota tapauskohtaisesti. Esimerkin kaupunkitaulua koottaessa voidaan syöttää ”UIName”n arvoksi suoraan kentän ”Value” arvo.

```
<!--Selection list and search-->
<ul class="dropdown-menu emce-formpanel-dropdown"
    role="menu">
    <li role="presentation">
        <div class="input-group input-group-sm">
            <input type="text" class="form-control dynamic-form-control"
                   placeholder="{{ lang.language().UI_Search }}"
                   ng-model="query">
            <div class="input-group-addon">
                <i class="glyphicon glyphicon-search"></i>
            </div>
        </div>
    </li>
    <li role="presentation"
        ng-repeat='value in codetableContainer[field.CodeTableReference]
        | toArray | filter:{UIName: query}'>
        <a ng-click="selectDropdownValue(field, value)"> {{value.UIName}} </a>
    </li>
</ul>
```

Kuva 18. Direktiivin pudotusvalikkokentän toteutus. Osa 2.

ng-model=”query”

Muuttuja, jota ei ole esitelty muilla tasoilla kuin suoraan tässä HTML-osuudessa. Se on sidottu hakukentän arvoksi ja myöhemmin ”ng-repeat”in suodattimena. Näin toteutettuna lista suodattuu automaattisesti hakusyötteen muuttujassa.

ng-repeat=’value in….

Tässä silmukassa käydään läpi kaikki pudotusvalikkoa varten haetussa taulussa olevat oliot. ”filter”-osa on AngularJS:n sisäinen direktiivi, joka auttaa suodattamaan ng-repeatilla tuotettuja listoja. Sille määritellään, mikä muuttuja suodattaa tuloksia. Tässä tapauksessa se on ”query”, joka suodattaa listan soluja muuttujan ”UIName” mukaan.

<a ng-click=”selectDropdownValue(field,value)”…

AngularJS tarjoaa valmisdirektiivin nimeltä ”ng-click”, jonka avulla jokaiseen HTML-elementtiin voidaan sitoa kutsu JavaScript-funktioon klikattaessa. Tässä on toteutettu oma logiikka taulukon solun klikkaamiseen, joka saa parametreikseen silmukan kierroksessa olevan solun ”field” sekä oman arvonsa.

5 Yhteenveto

AngularJS on tehokas työkalu käyttöliittymän hajauttamiseen komponentteihin, muuttujien sitomiseen näkymän ja koodin välillä sekä erilaisten validaattorien, filttereiden ja parsereiden toteuttamiseen.

Opinnäytetyössä toteutettu dataperusteinen lomake parantaa peruslomakkeiden ylläpitoa, niin että ylläpito tapahtuu yhdessä hyvin määritellyssä komponentissa, joka määrittelee sekä lomakkeiden ulkoasun että toteutuksen.

Dataperusteiset lomakkeet sopivat hyvin isompiin ohjelmistoprojekteihin, joissa käytetään lomakkeita käyttäjän syötteen vastaanottamiseksi.

Opinnäytetyössä määriteltyjen kolmen lomakekentän mahdollistamiseksi ei tarvittu rivimäärältään suuria määriä JavaScript- tai HTML-koodia, mutta niille mahtuukin melkoinen määrä loogisia lauseita, jolloin perehtyminen AngularJS-tekniikoihin on oma oppimiskokonaisuutensa. Tässä onkin mielestäni AngularJS:n vahvuus ja heikkous: päästäkseen tavoitteeseen ohjelmoija selviää suhteellisen vähällä määrällä koodia, mutta pystyäkseen tuottamaan sen ohjelmoijan täytyy omaksua melkoinen määrä pelkästään tähän ohjelmistokehykseen liittyvää tietoa.

Lähteet

1 AngularJS FAQ. Verkkodokumentti. Super-powered by Google. https://docs.angularjs.org/misc/faq. Luettu 10.9.2016.

2 Creating custom Directives. Verkkodokumentti. Super-powered by Google. https://docs.angularjs.org/guide/directive. Luettu 10.9.2016.

3 Bootstrap Get Started. Verkkodokumentti. W3Schools. http://www.w3schools.com/bootstrap/bootstrap_get_started.asp. Luettu 11.9.2016.

4 Gimmer, Christopher. Top 5 Reasons to use Bootstrap. Verkkodokumentti. https://bootstrapbay.com/blog/reasons-to-use-bootstrap/. Luettu 11.9.2016.

5 ngApp documentation. Verkkodokumentti. Super-powered by Google. https://docs.angularjs.org/api/ng/directive/ngApp. Luettu 13.9.2016.

6 Rajcok, Mark. Understanding Scopes. Verkkodokumentti. https://github.com/angular/angular.js/wiki/Understanding-Scopes. Luettu 13.9.2016.

dynamicform.html

```
<form name="dynamicForm">
   <div ng-if="fields"
        ng-repeat="field in fields.Fields | orderBy:'+DisplayOrder'"
        ng-init="fields.title = title">

       <div class="form-group emce-slide-up " ng-if="!field.IsHidden">
          <label ng-if="!(field.Type == 'Boolean') && !(field.Type == 'Text')">
             {{ field.DisplayName ? field.DisplayName : field.FieldName }}
          </label>
          <!--String Input-->
          <label ng-if="field.Label">
             {{ field.Label }}
          </label>
          <input ng-if="field.Type == 'TextInput'"
                 type="text"
                 class="dynamic-input"
                 ng-required="field.IsRequired"
                 ng-disabled="field.IsReadOnly"
                 ng-model="field.Value"
                 dynamic-text-input />
          <!--Text-->
          <span ng-if="field.Type == 'Text'"
                class="dynamic-text">
             {{field.Value}}
          </span>

          <!--Guid (list of items from codetable)-->
          <div class="dropdown"
               ng-if="field.Type == 'Dropdown'"
               ng-init="getCodeTable(field)">
             <div ng-if="field.ready">
                <!--Button to open dropdown-->
                <button type="button" class="form-control dynamic-form-control btn btn-default dropdown-toggle"
                        ng-disabled="field.IsReadOnly"
                        data-toggle="dropdown">
                   <span>{{field.UIName}}</span> <span class="caret"></span>
                </button>

                <!--Selection list and search-->
                <ul class="dropdown-menu emce-formpanel-dropdown"
                   role="menu">
                   <li role="presentation">
                      <div class="input-group input-group-sm">
                         <input type="text" class="form-control dynamic-form-control"
                                placeholder="{{ lang.language().UI_Search }}"
                                ng-model="query">
                         <div class="input-group-addon">
                            <i class="glyphicon glyphicon-search"></i>
                         </div>
                      </div>
                   </li>
                   <li role="presentation"
                      ng-repeat='value in codetableContainer[field.CodeTableReference]
                      | toArray | filter:{UIName: query}'>
                      <a ng-click="selectDropdownValue(field, value)"> {{value.UIName}} </a>
                   </li>
                </ul>
             </div>
          </div>

   <!--loading indicator-->
   <loading ng-if="!fields"></loading>

</form>
```

app.directives.js

```
app.directive('dynamicTextInput', function () {
    return {
        require: 'ngModel',
        restrict: 'A',
        link: function (scope, element, ngModelCtrl) {
            function inputChange(text) {
                var transformedInput = text.replace(/[^A-Za-z ]/g, '');
                if (transformedInput !== text) {
                    ngModelCtrl.$setViewValue(transformedInput);
                    ngModelCtrl.$render();
                }
                return transformedInput;
            }
            ngModelCtrl.$parsers.push(inputChange);
        }
    };
});

app.directive('dynamicform', function ($http) {
    return {
        restrict: 'E',
        templateUrl: 'dynamicform.html',
        scope: {
            fields: '=',
        },
        link: function (scope, element) {
            scope.codetableContainer = [];

            scope.getCodeTable = function (field) {
                if (!scope.codetableContainer[field.CodeTableReference]) {
                    scope.codetableContainer[field.CodeTableReference] = [];
                    $http.get(
                        createRelativeUrl("api/Common/getCodeTableById?tableId="
+ field.CodeTableReference)).
                    success(function (data, status, headers, config) {
                        populateCodeTable(field.CodeTableReference, data.CodeTa-
bleRows, field);
                    });
                }
            };

            var populateCodeTable = function (tableId, data, field) {
                scope.codetableContainer[tableId] = data;
                updateUINames(field, data);
            }

            scope.selectDropdownValue = function (field, value) {
                    field.Value = value;
            }
        }
    };
});
```