TAMPEREEN
AMMATTIKORKEAKOULU

ANGULARJS WEB-SOVELLUSTEN KEHITYKSESSÄ

Tero Mustikkamaa

Opinnäytetyö
Elokuu 2016
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotanto

TERO MUSTIKKAMAA
AngularJS web-sovellusten kehityksessä

Opinnäytetyö 28 sivua, joista liitteitä 6 sivua
Elokuu 2016

Websivut ovat kehittyneet paljon alkuaikojen staattisista ja tylsän näköisistä viritelmistä, jotka liitettiin yhteen hyperlinkeillä. Modernit websivut operoivat pääasiassa yhdellä sivulla ja voivat olla täysin itsenäisiä sovelluksia. Pelkällä HTML-kielellä näitä ei pystyisi toteuttamaan. Kaikkein merkittävimmän lisäyksen websivujen kehitykseen on toistaiseksi tarjonnut Javascript-ohjelmointikieli. Sitäkin on ehditty jalostaa useilla lisäkirjastoilla ja sovelluskehyksillä.

Tämän työn aihe eli AngularJS on Misko Heveryn alun perin kehittämä ja nykyisin Googlen ylläpitämä sovelluskehys. Sekin on Javascriptillä kirjoitettu ja on mahdollisesti jQueryn jälkeen suosituin web-sovelluskehys. Opinnäytetyön tavoitteena on tutustua AngularJS:n toimintaan ja kartoittaa sen erityisvahvuudet ja -heikkoudet. Työ on tamperelaisen Dicode Oy:n toimeksiantama ja perustuu kokemuksiini heillä työharjoittelussa tekemääni työhön Kelpolounas-nimisen sopimusruokailun maksujärjestelmän parissa.

Tässä opinnäytteessä käsitellään Angularin nykyistä, vakaata 1.5-versiota. Angularista on kuitenkin hiljattain 2016 toukokuussa julkaistu 2.0-versio, jota en tarkemmin käsittele. Suosittelen kuitenkin tämän työn lukijalle ja Angulariin tutustuvalle jatkoluettavaksi Angular 2.0:aan perehtymistä, koska sovelluskehyksen tulevaisuus todennäköisesti on sen harteilla.

Asiasanat: Angularjs, Javascript, sovelluskehys, webkehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree of Business Information Systems
Software Development

TERO MUSTIKKAMAA
AngularJS in Web Application Development

Bachelor's thesis 28 pages, appendices 6 pages
August 2016

Web pages have come far from their early days of jumping around multiple dull looking pages via hyperlinks. Modern web pages mainly operate on a single page and can be completely independent applications. Such pages would not be possible to make with HTML alone. The Javascript programming language has been the most significant addition in the world of web development and nowadays there are dozens of extra libraries and frameworks to shore up its shortcomings.

AngularJS, which is the topic of this thesis, was originally developed by Misko Hevery and is now maintained by Google. It too is a Javascript based framework and is possibly the most popular web development framework out there after jQuery. The goal of this thesis was to introduce the basic workings of Angular and chart out its special strengths and weaknesses. The thesis was commissioned by Dicode Oy (Ltd.) and was based on the author's experiences gained during the time he was working on a lunch payment system called Kelpolounas for the company.

In this thesis, the focus was on version 1.5 of AngularJS which is the current stable version. However, the Angular development team released the first release candidate of Angular 2.0 in May 2016. While this version is not covered in depth here, the author does recommend any reader of this thesis who is interested in AngularJS to familiarize themselves with Angular 2.0 as the future of the framework probably rests on its shoulders.

Key words: Angularjs, Javascript, framework, web development

SISÄLLYS

LYHENTEET JA TERMIT

AngularJS	Javascriptillä toteutettu sovelluskehys yksisivuisten web-sovellusten tuottamiseen.
SPA	Single Page Application. Web-sovellus, joka toimii yhdellä sivulla hyperlinkeillä yhdistettyjen useiden sivujen sijaan.
Javascript	Ohjelmointikieli, joka on yksi WWW-sisällön tuottamisen peruspilari HTML:n ja CSS:n rinnalla.
MVC	Model View Controller. 1970-luvulla kehitetty ohjelmistoarkkitehtuurimalli.
MVVM	Model View ModelView. MVC:stä johdettu ohjelmistoarkkitehtuurimalli.
AJAX	Asynchronous Javascript and XML. Teknologia, jonka avulla web-sovellus voi asynkronisesti hakea dataa palvelimelta ja sijoittaa sen verkkosivulle sitä päivittämättä.
JSON	Javascript Object Notation. Avain- ja arvopareihin perustuva datasäilöntäformaatti.
DOM	Document Object Model. API jossa HTML, XHTML tai XML dokumentin jokaista elementtiä vastaa oma olionsa.
API	Application Programming Interface. Kokoelma määritelmiä, protokollia ja työkaluja, jotka toimivat ohjelmoijan rakennuspalikoina. Ohjelmointikielissä tämä tarkoittaa ohjelmistokirjastoa.
jQuery	Maailman suosituin avoimen lähdekoodin Javascript-kirjasto.
ECMAScript	Skriptejä tukevien ohjelmointikielien standardi. Perustuu alunperin Javascriptiin, mutta nykyään Javascriptin ajatellaan noudattavan ECMAScriptin standardeja.
TypeScript	Microsoftin kehittämä oliopohjainen ohjelmointikieli, joka voidaan ajaa kääntäjän läpi Javascriptiksi.

1 JOHDANTO

Web-sovellukset ovat tasaista tahtia nostaneet suosiotaan sitä mukaa, kun internet palveluntarjoajien datasiirtonopeudet ovat nousseet. Aikaisemmin tapana on ollut tuottaa pöytäkoneelle, kannettavalle tai mobiililaitteelle asennettava sovellus. Vaikka asennettavat sovellukset eivät olekaan mihinkään poistumassa, on sovelluskehittäjille houkuttelevaa tehdä sovelluksia, joihin pääsee käsiksi millä tahansa laitteella, jossa on internetselain. Tällä tavalla ohitetaan tarve tehdä alustariippuvaisia sovelluksia; esimerkiksi pöytäkoneiden kohdalla Windows, Mac tai Linux ja älypuhelimilla iPhone, Windows Phone tai Android kymmenine versioineen.

Ollessani työharjoittelussa Dicode Oy:llä 2015 toukokuusta syyskuuhun, sain työskennellä Kelpolounas nimisen web-sovelluksen kehityksessä. Kelpolounas on sähköinen sopimusruokailun maksujärjestelmä yrityksille ja se käyttää AngularJS sovelluskehystä MVC-mallin (Model View Controller) mukaisen käyttöliittymärakenteen luomiseen, jossa palvelinpuolen osuus on toteutettu Javalla.

Tämän opinnäytetyön tavoitteena on esitellä AngularJS:n tai lyhyemmin Angularin ominaisuudet sekä laatia arvio sen vahvuuksista ja heikkouksista web-sovellusten kehityksessä. AngularJS ei ole suinkaan ainoa työkalu yksisivuisten web-sovellusten toteuttamiseen (Single Page Application, SPA), joten Dicode Oy teki toimeksiannon tälle työlle helpottamaan heidän arvioitaan, mihin tuleviin projekteihin AngularJS voisi sopia.

2 TAUSTAA

HTML-kieli eli Hypertext Markup Language, jolla websivut perinteisesti kirjoitetaan, julkaistiin jo vuonna 1991 ja se suunniteltiin alun perin kuvaamaan staattisten websivujen rakennetta. Nykyisen mallisia yksisivuisia ja interaktiivisia websivuja ei voisi toteuttaa pelkällä HTML:llä.

Vuosien varrella HTML:n puutteita paikkaamaan on tehty useita ohjelmointikieliä ja kehyksiä, kuten CSS sivujen ulkoasun määrittelemiseen, Javascript ohjelmalogiikan suorittamiseen ja AJAX HTML:n elementtien muokkaamiseen sivua uudelleen lataamatta. AJAXillakin elementtien muokkaaminen on kuitenkin hyvin vaivalloista, joka näkyy monimutkaisempia websivuja tehdessä.

Toinen huolenaihe websivujen ja sovellusten toimintojen laajetessa on tietoturva. HTML ja Javascript ovat internetselaimen käsissä ja helposti esiin kaivettavissa. Esimerkiksi kirjautumistiedot ja niiden käsittely pitää saada pois asiakkaan (client) selaimelta. Yksi ratkaisu on erottaa käyttäjälle näytettävä käyttöliittymä sovelluslogiikasta MVC-mallin (Model View Controller) mukaisesti. Javascriptin puolella lähetetään vain ulkoiselle palvelimelle pyyntö tarvittavasta tiedosta, joka puolestaan käsittelee pyynnön ja palauttaa tiedon.

Tässä työssä käsiteltävä AngularJS on juurikin MVC-mallia noudattava, Javascriptin päälle rakennettu ohjelmistokehys.

3 ANGULARJS

3.1 Yleisesti

AngularJS:n kehitti alun perin Misko Hevery vuonna 2009 Brat Tech LLC:n käyttöön. Yritys kuitenkin hylkäsi JSON tallennusjärjestelmänsä, jota Angular pyöritti ja julkaisi Angularin avoimena lähdekoodina. AngularJS:n nykyinen vakaa versio on Googlen ylläpitämä 1.5 versio.

AngularJS on Javascript-pohjainen ohjelmistokehys. Käytännössä se on Javascriptillä toteutettu kirjasto, jonka ottaminen websivun käyttöön vaatii vain yhden <script>-tagin. Angularille kerrotaan, mitä websivujen osia sen tulee hallinnoida. Tämä tapahtuu käyttämällä erityisiä HTML-tageja, joita Angularissa kutsutaan direktiiveiksi. Seuraava on esimerkki käyttöönotosta Angularin omasta tutoriaalista (kuva 1):

```
<!doctype html>
<html lang="en" ng-app>
<head>
  <meta charset="utf-8">
  <title>My HTML File</title>
  <link rel="stylesheet" href="bower_components/bootstrap
/dist/css/bootstrap.css">
  <link rel="stylesheet" href="css/app.css">
  <script src="bower_components/angular/angular.js"></script>
</head>
<body>

  <p>Nothing here {{'yet' + '!'}}</p>

</body>
</html>
```

Kuva 1: AngularJS:n käyttöönotto (AngularJS.org 2016a)

Huomion arvoinen on <html>-tagin attribuuttina oleva *ng-app*, joka on Angularin ydindirektiivi. Direktiivin voi sijoittaa jonkin muunkin tagin sisään, jos Angularin ei haluta hallitsevan koko websivua vaan vain osaa siitä.

3.2 Ominaisuudet

3.2.1 Scope

Scope on yksi Angularin keskeisimmistä ominaisuuksista. Scope on konkreettisesti Javascript olio *$scope*, joka matkii websivun DOM-mallin (Document Object Model) rakennetta, jossa koko rakenne on *$rootScope* nimisen olion alla. DOM on mallinnus websivun rakenteesta, jossa jokaista elementtiä vastaa oma olionsa. Tällä tavalla saavutetaan yksi-yhteen kytkös jokaisen HTML-elementin ja sen *$scope* olion välille. Samalla vältetään Javascriptin nimiavaruuden sotkemista. Javascript on pahamaineinen globaalien muuttujiensa käytöstä. *$rootScope* on Angularissa lähimpänä Javascriptin globaalia kontekstia.

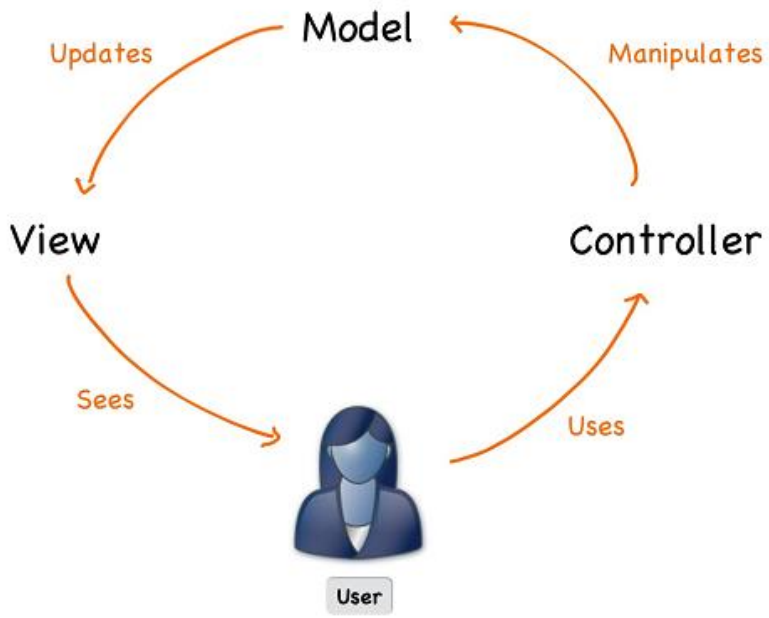
Scopet tarjoavat sovellukselle seuraavat ominaisuudet:

- Tarkkailijoita seuraamaan muutoksia DOM-mallissa.
- Mahdollisuuden tehdä muutoksia DOMin sisällä sovelluksesta käsin.
- Scopeja voi sijoittaa sisäkkäin erottelemaan toiminnallisuutta.
- Tarjoavat ympäristön totuusarvojen arvioimiseen.

(Lerner 2013, 23)

3.2.2 MVC ja MVVM

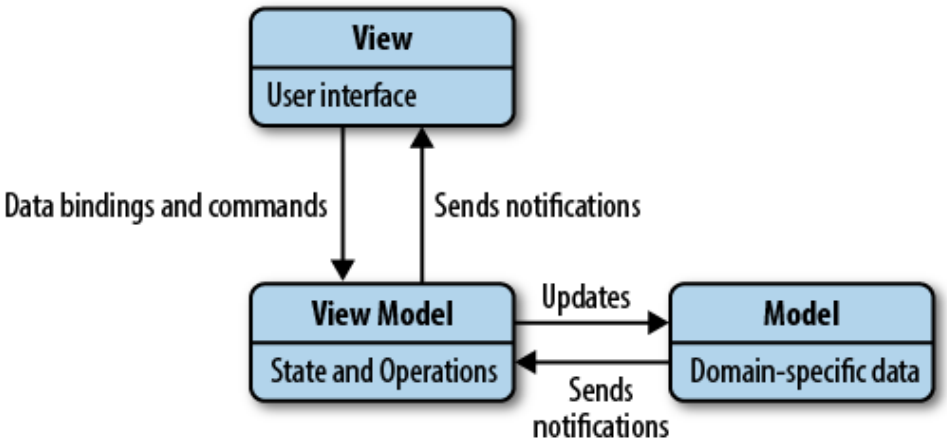
Lyhenne MVC tulee sanoista Model View Controller tai Malli Näkymä Käsittelijä. MVC on jo 1970-luvulla kehitetty arkkitehtuurimalli tietokonesovellusten käyttöliittymien suunnitteluun. Sen perusajataus on, että sovelluksen sisäinen logiikka täytyy pitää erillään käyttäjälle esitettävästä informaatiosta. Malli viittaa käyttäjältä piilotettavaan sovelluksen sisäiseen dataan ja logiikkaan. Näkymä viittaa käyttöliittymänäkymään. Kolmas komponentti eli kontrolleri tyypillisesti hallinnoi käyttäjän syötteitä ja niiden logiikkaa. MVC on nykyään suosittu web-sovellusten suunnittelussa. (Osmani 2015)



Kuva 2: MVC-malli

MVVM eli Model View ViewModel on puolestaan arkkitehtuurimalli, joka on johdettu MVC:stä ja sitä sovelletaan etenkin web-sovellusten kehityksessä. MVVM pyrkii erottelemaan käyttöliittymään sovelluslogiikasta MVC:tä enemmän hyödyntämällä datasidontoja. (Osmani 2015)

MVVM:n model eli malli on sama kuin MVC:ssä. Näkymä on vastaavasti käyttöliittymä ja vastaa sekin MVC:n näkymää. Kolmas komponentti eli viewmodel, vapaasti suomennettuna näkymämalli, on hieman erilainen kuin MVC:n käsittelijä. Niiden tehtävä on sama eli käsitellä käyttäjän syötteet ja päivittää näkymää tai mallia. Näkymämallin vastuulla on lisäksi ohjata datasidontoja ja siirtää dataa mallin ja näkymän välillä. (Osmani 2015)



Kuva 3: MVVM-malli

Kehityksensä alkuvuosina AngularJS ilmeisesti implementoi MVC-mallin periaatteita, mutta on myöhempien päivitystensä mukana siirtynyt lähemmäs MVVM-mallia. (StackOverflow.com ja Branas 2014)

3.2.3 Kaksisuuntainen datasidonta

Kaksisuuntainen datasidonta (two-way data binging) on yksi Angularin merkittävimmistä ominaisuuksista. Muissa sovelluskehyksissä kuten esimerkiksi Railsissa esitettävä data tyypillisesti vedetään palvelimelta, joka laitetaan jonkun DOM-elementin paikalle. AngularJS puolestaan luo koko mallin (template) dynaamisesti kaikkine DOM-elementteineen (Lerner 2013, 12). Seuraavassa esimerkissä (kuva 4) Angular etsisi kontrolleristaan items nimistä taulukkoa ja loisi niin monta div-elementtiä kuin kyseisessä taulukossa on alkioita.

```
1  <div ng-repeat=item in items">
2      <span>{{item.title}}</span>
3  </div>
```

Kuva 4: Datasidonta

Kaksoisaaltosulkeissa oleva osa on esimerkki Angularin datasidonnasta. *Ng-repeat* käy *items*-taulukon alkiot läpi ja ottaa ne yksi kerrallaan *item*-muuttujaan. *Item.title* olettaa, että Javascriptin puolella *items*-taulukon alkioilla on kullakin ominaisuus *title*, joka on oletettavasti merkkijono. Lopulliselle websivulle tulostuisi joukko div-elementtejä, joissa olisi kussakin yksi merkkijono.

Kaksisuuntaisuus kaksisuuntaisessa datasidonnassa tarkoittaa sitä, että jos missään vaiheessa *items*-taulukkoon vaikka lisätään elementti, voi Angular välittömästi päivittää muutoksen websivun näkymään. Vastaavasti näkymän puolella voi olla käyttöliittymä, jolla käyttäjä pystyy poistamaan yhden kirjauksen ja Angular poistaa vastaavan elementin *items*-taulukosta.

3.2.4 Riippuvuusinjektio

Riippuvuusinjektio on Angularin tapa hankkia sovelluskomponenttien riippuvuudet. Riippuvuudet annetaan niitä tarvitsevalle komponentille parametreina ja suositeltu tapa tehdä tämä on niin kutsutun taulukkoannotaation kautta. Alla on esimerkki uuden kontrollerin luomisesta, jolle ilmoitetaan riippuvuudet *$scope* ja *$filter* (kuva 5).

```
1  var myApp = angular.module('myApp', []);
2  myApp.controller('SomeController',
3                   ["$scope", "$filter", function($scope, $filter) {
4      $scope.appTitle = $filter("uppercase")("Something...");
5  }]);
```

Kuva 5: Riippuvuusinjektio

Puretaan tätä sekavan näköistä viritelmää hieman. Ensimmäisellä rivillä esitellään moduuli nimeltä ”myApp”, jolla ei ole omia erityisriippuvuuksia (tyhjä taulukko []). Rivit kaksi ja kolme ovat kontrollerin ”SomeController” määrittelyä. Rivillä kolme näkyy aikaisemmin mainittu taulukkoannotaatio. AngularJS etsii riippuvuuksia nimen perusteella merkkijonona, jonka takia *$scope* ja *$filter* ovat lainausmerkeissä. Taulukossa viimeisenä mainittu alkio on funktio, joka sisältää kontrollerille halutun logiikan ja sille Angular antaa riippuvuudet parametreina.

Periaatteessa Angular tukee yksinkertaisempaakin tapaa riippuvuusinjektiolle eli mallin, jossa omassa esimerkissämme annettaisiin vain kontrollerin nimi ”SomeController” ja funktio *$scope* ja *$filter* parametreineen, jolloin taulukkoa ei tarvittaisi. Tämä tapa ei kuitenkaan toimi, jos koodin sisältävä Javascript-tiedosto minifioidaan, koska minifiointiprosessi nimeää funktioiden parametrit geneerisiksi A:ksi ja B:ksi, jonka jälkeen Angular ei enää pysty nimen perusteella yhdistämään riippuvuuksia oikein. (Branas 2014) Javascript tiedostojen minifiointi on hyvin yleinen käytäntö, jossa lähdekoodin tiedostosta poistetaan kaikki suorituksen kannalta tarpeettomat merkit, kuten välit, rivinvaihdot ja kommentit. Tämän jälkeen tiedosto ei ole enää kovin luettava, mutta se pienentää tiedoston kokoa.

3.3 Rakenne

3.3.1 Moduulit

Javascriptilla on helppo kirjoittaa toimivaa koodia suoraan globaaliin nimiavaruuteen, mikä on kyllä nopeaa ja mistä tahansa sovelluksen osasta saatavilla, mutta se on myös kaikkien hyvien ohjelmointitapojen vastaista. Globaalin nimiavaruuden täyttäminen aiheuttaa helposti vaikeasti testattavia nimeämisristiriitoja ja antaa turhaan toimintoja sovelluksen osien käyttöön, jotka eivät niitä tarvitse. Yleinen käytäntö on kapseloida tavalla tai toisella ohjelman toiminnallisuus pienempiin paketteihin ja nimiavaruuksiin. Angularissa näitä paketteja kutsutaan moduuleiksi.

Moduuli on Angular-sovelluksen peruspalanen ja sovellus voi sisältää yhden tai vaikka kuinka monta moduulia. Itse asiassa Angularilla voi tehdä yksinkertaisia sovelluksia käyttämättä yhtään moduulia. Angular API:ssa on metodi ”angular.module()”, jolla voimme esitellä moduulin. Metodi ottaa vastaan kaksi muuttujaa: Nimen ja riippuvuudet. (Freeman 2014)

```
angular.module(’myApp’, []);
```

3.3.2 Kontrollerit

Kontrollerien tarkoitus on laajentaa AngularJS-sovelluksen näkymiä. Kontrolleri on Javascript-funktio, joka antaa jonkun näkymän scopelle lisää toimintoja. Sen kautta voi asettaa näkymän alkutilan ja antaa erityistoimintoja juuri sen näkymän scopelle. Kontrollerien vastuulla on muun muassa:

- Hakea palvelimelta dataa käyttöliittymässä näytettäväksi. (Tai ainakin aloittaa haku)
- Päättää mitä osia datasta käyttäjälle näytettään.
- Hallinnoida ja validoida käyttäjäsyötteitä.

(Green & Seshadri 2014)

Yksinkertainen kontrollerin luominen voisi näyttää vaikka tältä (kuva 6):

```
1  var app = angular.module('app', []);
2  app.controller('FirstController', function($scope) {
3      $scope.counter = 0;
4      $scope.add = function(amount) {
5          $scope.counter += amount;
6      };
7  });
```

Kuva 6: Kontrolleri

Rivillä 1 esittelemme uuden moduulin sisältämään kontrollerimme ja rivit 2-7 ovat kontrollerimme sisältö. Muuttuja ’counter’ ja metodi ’add()’ ovat sidottuja tämän kyseisen kontrollerin scopeen ja ainoastaan siihen.

3.3.3 Direktiivit

AngularJS:n direktiivit ovat karkeasti merkintöjä HTML-elementeissä kuten attribuutteja. (AngularJS.org. 2016b) Angular tarjoaa liudan oletusdirektiivejä ja lisäksi mahdollisuuden tehdä omia direktiivejä HTML:ssä kutsuttavaksi. Toinen tapa lähestyä direktiivejä on ajatella niitä funktioina. Niiden tarkoitus on laajentaa HTML-elementtien ominaisuuksia. Esimerkiksi *ng-click*-direktiivi antaa elementille kyvyn kuunnella käyttäjän hiiren klikkauksia ja kutsua Javascript-funktiota sellaisen havaittuaan. (Lerner 2013, 105)

Angularin direktiivit voidaan jakaa kahteen luokkaan: Data binding direktiiveihin ja template direktiiveihin. Data binging direktiivit ottavat arvoja modelista ja lisäävät ne HTML-dokumenttiin. Template direktiivit voivat puolestaan luoda dynaamisesti uusia HTML-elementtejä. (Freeman 2014)

3.3.4 Palvelut

Toisin kuin kontrollerit, AngularJS:n palvelut tarjoavat tavan säilöä dataa sovelluksen koko elinkaaren ajaksi. Kontrollerit ladataan ja puretaan sitä mukaa, kun niille tulee tarvetta tai se tarve poistuu. Palvelut alustetaan ainoastaan kerran käyttäen riippuvuusinjektiota. Ne ovat olioita, jotka niin sanotusti ladataan laiskasti eli vasta sitten, kun niitä ensimmäisen kerran tarvitaan.

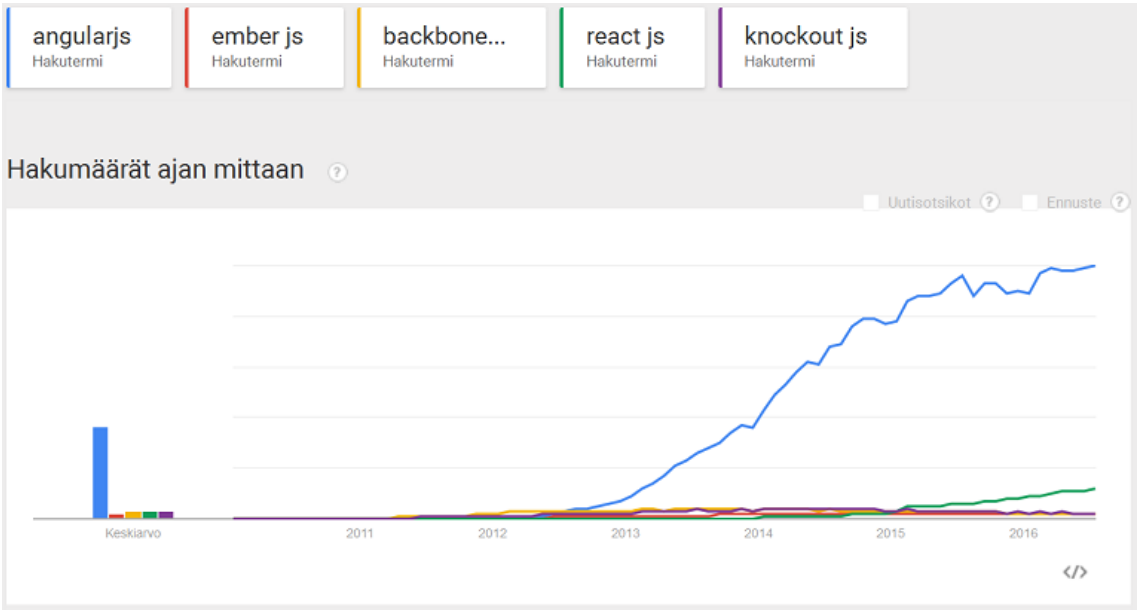
Käytetään esimerkkinä *$http*-palvelua. *$http* on yksi Angularin ydinpalveluista ja sillä on yhteys selaimen XMLHttpRequest-olioon. Sillä tyypillisesti siirretään dataa asiakaslaitteen ja palvelimen välillä. *$http*-palvelun ansiosta Angular-sovelluksen kirjoittajan ei tarvitse täyttää koodiaan toistuvilla kutsuilla XMLHttpRequestiin, vaan *$http* voi suorittaa nämä kulisseissa. (Lerner 2013, 161)

4 VAHVUUDET JA HEIKKOUDET

Javascript on sementoinut paikkansa web-sovellusten maailmassa, mutta kielen ilmeiset heikkoudet ovat johtaneet moniin laajennuksiin ja kehyksiin, joihin Angularkin lukeutuu. Kaikkein suosituin ja levinnein näistä on ehdottomasti jQuery ja sitä monesti käytetäänkin samoihin tarkoituksiin yksisivuisten web-sovellusten toteutuksessa kuin Angularia. Angularissa on kuitenkin useita ominaisuuksia, mitä jQueryllä ei ole ja sitä ei oikeastaan voi pitääkään Angularin kilpailijana.

4.1 Suosio ja kilpailijat

Muita Angularin kanssa samankaltaisia Javascript-sovelluskehyksiä ovat Ember.js, Backbone.js, React.js ja Knockout.js, mutta yksikään näistä ei ole saavuttanut lähellekään samaa suosiota kuin AngularJS. Google Trends (kuva 7) paljastaa, että Angular alkoi merkittävästi nostaa suosiotaan vuoden 2012 puolivälin tienoilla. Vuoteen 2015 mennessä maailmanlaajuisesti Angulariin liittyviä hakuja on tehty kuukausittain moninkertaisesti, vaikka kaikki muut kilpailijat laskettaisiin yhteen. Syitä ylivoimaiseen suosioon on vaikea arvioida, koska Angularistakin tuntuu löytyvän vähintään yhtä paljon moittivia kuin kehuviakin mielipiteitä.



Kuva 7: Google Trends

Erinäisiltä yksityishenkilöiltä löytyy paljon näitä kehyksiä vertailevia blogikirjoituksia, mutta kirjallisia tutkimuksia ei niinkään. Monikaan näistä ei tunnusta yhtään kehystä selkeästi muita vahvemmaksi, mutta Angular on kuitenkin saavuttanut suurimman suosion.

4.2 Tehokkuus

Angularin kritisoijat tuovat kaksisuuntaisesta datasidonnasta heikkoutena esiin tehottomuuden. Angularin *$watch*, joka vastaa DOM-mallin muutosten kuuntelusta, käyttää niin sanottuja implisiittisiä käsittelijöitä. Tämä tarkoittaa, että kun mallissa jokin muuttuu, kymmenet tai sadat kuuntelijametodit aktivoituvat selvittämään tarvitseeko niiden reagoida jotenkin. Vaihtoehtona olisi luoda elementeille eksplisiittiset kuuntelijat, jolloin esimerkiksi painonapin klikkaaminen kutsuisi vain ja ainoastaan kyseiselle napille asetetun kuuntelijametodin. (Medium.com)

Tehokkuuden ongelma on aina näkyvämpi mobiililaitteilla, joilla ei ole yhtä paljon prosessointivoimaa. Angularin kehitystiimi on itsekin ilmeisesti tiedostanut ongelman, mutta se on toistaiseksi vain kierretty rajaamalla yhden AngularJS-sovelluksen kuuntelijat 2000:een. Joissain tapauksissa näkymän latauksessa Angularilta saattaa jäädä käyttöliittymään selvittämättömiä ekpressioita (expression), kunnes ne hyppäävät paikoilleen kehyksen ladatessa niitä. Käyttäjä saattaa nähdä esimerkiksi merkkijonon keskellä jonkun kaksoisaaltosulkeissa olevan ilmaisun *{{customer.name}}*, johon olisi ollut tarkoitus sijoittaa scopesta muuttujan ”customer” ominaisuus ”name”.

4.3 Testaus

AngularJS:n arkkitehtuurissa on otettu huomioon automaattinen yksikkötestaus. Yksikkötestit ovat pieniä parametreja, joilla testausohjelma voi ajaa varsinaisen sovelluksen osia ja tarkistaa niiden perustoimivuuden. Javascriptillä itsellään ei ole edes perinteistä kääntäjää, joka nappaisi edes syntaksi- eli kirjoitusvirheet ohjelmakoodissa. Hyvin suunnitellut yksikkötestit voivat varmistaa sovelluskomponenttien toimivuuden, vaikka niiden sisäistä toteutusta myöhemmin muutettaisiinkin esimerkiksi tehokkuutta parantaessa.

Yksikkötestien kirjoittaminen on sitä helpompaa, mitä pienempiin itsenäisiin osiin sovellus on jaettu. Luvussa kaksi esitelty riippuvuusinjektio on yksi Angularin arkkitehtuurin osa, joka auttaa sovelluksen pilkkomisessa. Jokainen riippuvuus voi olla itsellään yksikkötestattava osa, jolloin kun ne tuodaan yhteen jonkin toisen komponentin käyttöön, niiden yleinen toimivuus on jo yksikkötesteillä varmistettu. Toinen Angularin testausta avustava ominaisuus on Javascriptin DOMin abstrahointi, jota voidaan testata varsinaisen DOMin sijaan.

AngularJS:n dokumentaatiossa esitellään kaksi lisätyökalua yksikkötestaamiseen; Karma ja Jasmine. Karma on Javascript-komentorivityökalu, joka voi luoda väliaikaisen webpalvelimen ja ajaa kirjoitetut yksikkötestit useammalla eri verkkoselaimella. Jasmine on Javascript-pohjainen sovelluskehys, joka tarjoaa funktioita erillisten yksikkötestien hallinnoimiseen ja ryhmittelyyn. Karmaa ja Jasminea voi käyttää yhdessä Jasminelle tarkoitetulla karma-jasmine nimisellä liitännäisellä.

5 YHTEENVETO JA POHDINTA

Kun tutustuin Angulariin työharjoittelussani Dicodella, minulla oli vähän kokemusta Javascriptistä, joten ensimmäisenä jouduin totuttautumaan sen syntaksiin. AngularJS:n syntaksi on lisäksi Javascriptin hankalammasta päästä. Esimerkiksi uutta kontrolleria tehdessä, sen koko toteutus annetaan funktiona, joka menee AngularJS-kehyksen oman *controller* funktion parametriksi. Liitteestä 2 näkee, kuinka koko Javascript-tiedosto on käytännössä yhden moduulin ja kahden kontrollerin sisälle kirjoitettu hirvitys.

Moni Angularin heikkouksista on kuitenkin johdettavissa takaisin sen kielen puutteisiin, jolla Angular on kirjoitettu eli Javascriptiin. Syntaksi, testattavuus ja virheenhallinta ovat kaikki Javascriptin heikkoja puolia ja ne heijastuvat jossain määrin Angulariinkin. Siitäkin huolimatta Angularin vahvuuksia ei käy kieltäminen. Kaksisuuntainen datasidonta säästää paljon toistuvaa koodinkirjoittamista, jota esimerkiksi jQueryllä pitäisi käyttää. Angularin asynkroninen koodin suorittaminen on hyvin toimivaa. Moduulit paketoivat sovelluksen osat ja ison kommuunin ansiosta omiin ongelmiin löytää suhteellisen helposti apua. (Airpair.com)

Nyt Google on kuitenkin julkaisemassa Angularista 2.0-versiota, joka ei ole suoraan yhteensopiva vanhemmilla versioilla kirjoitettujen sovellusten kanssa. Se on monilta osin täysin uudelleen kirjoitettu. Vaikka Angular onkin saavuttanut vahvan aseman web-sovelluskehyksenä, sen tulevaisuus riippuu todennäköisesti tämän 2.0-version onnistumisesta. Jatkossa kukaan tuskin aloittaa sovelluksensa toteuttamista AngularJS 1.5:llä ja jos vanhojen sovellusten siirtäminen 2.0:aan osoittautuu liian vaivalloiseksi, saattaa moni sovelluskehittäjä jäädä vaikeaan paikkaan sovelluksensa jatkokehityksen ja ylläpidon kanssa.

5.1 AngularJS 2.0

AngularJS:n 2.0-versiosta ilmoitettiin syyskuussa 2014. Se avattiin julkiseen betatestaukseen joulukuussa 2015 ja ensimmäinen ehdokas julkaisuversioksi julkaistiin toukokuussa 2016. Angular 2 ei ole tyypillinen versiokorotus vaan lähes täysi uudelleenkirjoitus, mikä on herättänyt huolta kehyksen yhteensopivuudesta vanhemmilla versiolla kirjoitettujen

sovellusten kanssa. Tämän työn pääpaino ei ole AngularJS 2.0-versiossa, koska se on niin uusi, ettei tutkittavaa kirjallisuutta yksinkertaisesti ole.

Angular 2:ssa tehtyjä muutoksia ovat muun muassa:

- Ydinpakettia on pilkottu edelleen moduuleihin Angularin rungon keventämiseksi.
- ECMAScript 5:stä on siirrytty ECMAScript 6 standardiin. (Nykyinen Javascript seuraa ECMAScriptin standardia, vaikka ECMAScript alun perin perustuukin Javascriptiin.)
- Sovellusten kirjoittamiseen suositellaan Microsoftin TypeScript kieltä, joka on syntaksiltaan huomattavasti enemmän perinteistä ohjelmointikieltä muistuttava kuin Javascript. TypeScript kuitenkin ajetaan kääntäjäsovelluksen läpi, joka tekee siitä Javascriptiä.
- Riippuvuusinjektiota on parannettu.
- Kontrollerit ja *$scope* on korvattu direktiiveillä ja uusilla komponenteiksi kutsutuilla erityisdirektiiveillä.
- Diary.js työkalut on lisätty, joilla voi seurata, missä sovelluksen suoritusaika kuluu.

(Angular.io)

LÄHTEET

Airpair.com. Luettu 20.4.2016.
https://www.airpair.com/js/Javascript-framework-comparison

Angular.io. 2016. Features. Luettavissa: https://angular.io/features.html

AngularJS.org. 2016a. Tutorial. Luettavissa: https://docs.angularjs.org/tutorial/step_00

AngularJS.org. 2016b. Directives. Luettavissa: https://docs.angularjs.org/guide/directive

Branas, R. 2014. AngularJS Essentials. Packt Publishing.

Cozlowski, P. & Darwin, P.B. 2013. Mastering Web Application Development with AngularJS. Packt Publishing.

Freeman, A. 2014. Pro AngularJS. Apress.

Google Trends. Luettu 5.7.2016. https://www.google.fi/trends/

Green, B. & Seshadri, S. 2013. AngularJS. O'Reilly Media, Inc.

Green, B. & Seshadri, S. 2014. AngularJS: Up and Running. O'Reilly Media, Inc.

Lerner, A. 2013. ng-book: The Complete Book on AngularJS. FULLSTACK.io.

Medium.com. Luettu 23.4.2016.
https://medium.com/@mnemon1ck/why-you-should-not-use-angularjs-1df5ddf6fc99

Osmani, A. 2015. JavaScript Design Patterns. O'Reilly Media, Inc.

StackOverflow.com. Luettu 31.5.2016.
http://stackoverflow.com/questions/20286917/angularjs-understanding-design-pattern

LIITTEET

Liitteet osioon olen kerännyt muutaman tiedoston Kelpolounas sovelluksesta. En ole kirjoittanut niiden sisältöä kokonaisuudessaan itse, mutta näitä olen työharjoitteluni aikana työstänyt.

Liite 1 on esimerkki, miten AngularJS näkyy HTML-tiedostossa direktiiveineen ja datasidontoineen. Liite 2 on esimerkki Angularin kontrollerista, joka vastaa käyttäjäsyötteiden logiikasta ja palvelinpyyntöjen eteenpäin lähettämisestä. Liitteessä 3 on AngularJS palvelu, jonka funktioita kontrolleri käyttää kommunikoidakseen palvelimen suuntaan. Palvelinpuolen toteutusta en liitteissä esittele, koska se ei liity tämän työn aiheeseen eli Angulariin. Lisäksi palvelinpuolen koodi sisältää Dicode Oy:n yrityssalaisuuksia, joita en halua vahingossa paljastaa.

Näiden kolmen tiedoston sisältö esitetään toimeksiantajan luvalla.

Liite 1. Kelpolounas HTML esimerkki.

```
<!--
  ~ Copyright (c) 2015. Dicode Oy
  -->

<h2 translate="company_offices_title"></h2>

<div class="borderContainer">
    <div class="innerContainer">
        <button class="btn help-icon pull-right" title="{{'generic_help' |
        translate}}" popover="{{'company_offices_tooltip_help' | translate}}"
        popover-placement="left">
            <span class="glyphicon glyphicon-question-sign pull-right blue "
                aria-hidden="true"></span>
        </button>

        <br/>
        <br/>

        <table ng-show="offices.length" tr-ng-grid='' items='offices'>
            <thead>
            <tr>
                <th style="width:200px;" field-name="unitCode" display-name=
                "company_offices_unit_code" ></th>
                <th field-name="name" display-name="company_offices_name" ></th>
                <th>
                    <a role-restricted="company_admin" class="pull-right btn  btn-xs"
                     data-nodrag ng-click="addNewOffice()">
                        <span class="glyphicon glyphicon-plus"></span>
                    </a>
                </th>
            </tr>
            </thead>
            <tbody>
            <tr>
                <td>
                    <a class="btn-xs pull-right" data-nodrag ng-href=
                    "#settings?office={{gridItem.id}}">
                        <span class="glyphicon glyphicon-pencil"></span>
                    </a>
                    <a role-restricted="company_admin" class="btn-xs pull-right"
                    data-nodrag ng-click="removeOffice(gridItem)">
                        <span class="glyphicon glyphicon-trash"></span>
                    </a>
                </td>
            </tr>
            </tbody>
        </table>
    </div>
</div>

<script type="text/ng-template" id="addNewOffice.html">
    <div class="modal-header">
        <h3 class="modal-title" translate="company_offices_add_office_title"></h3>
    </div>

    <div class="modal-body">
        <p translate="required_fields_notice"></p>
        <form class="standarForm" name="newOfficeForm">

            <label class="fiftyPercentLabel" translate="company_settings_office_name"
             for="officeName"></label>
            <input type="text" id="officeName" ng-model="newOffice.name" required>
            <span field-error="name:officeName"></span>

            <br>

            <label class="fiftyPercentLabel" translate="company_settings_unit_code"
            for="unitCode"></label>
            <input type="text" id="unitCode" ng-model="newOffice.unitCode">
```

```
62
63              <br>
64
65              <h5 translate="company_settings_office_contact_person"></h5>
66
67              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_first_name" for="contact_first_name"></label>
68              <input type="text" id="contact_first_name" ng-model=
                "newOffice.address.contactPersonFirstName" required>
69              <span field-error="contactPersonFirstName"></span>
70
71              <br>
72
73              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_last_name" for="contact_last_name"></label>
74              <input type="text" id="contact_last_name" ng-model=
                "newOffice.address.contactPersonLastName" required>
75              <span field-error="contactPersonLastName"></span>
76
77              <br>
78
79              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_phone" for="contact_phone" ></label>
80              <input type="text" id="contact_phone" ng-model=
                "newOffice.address.contactPhone"required>
81              <span field-error="contactPhone"></span>
82
83              <br>
84
85              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_email" for="contact_email"></label>
86              <input type="email" id="contact_email" ng-model=
                "newOffice.address.contactEmail">
87              <span field-error="contactEmail"></span>
88
89              <br>
90
91              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_street_address" for="contact_street"></label>
92              <input type="text" id="contact_street" ng-model=
                "newOffice.address.streetAddress" required>
93              <span field-error="streetAddress"></span>
94
95              <br>
96
97              <label class="fiftyPercentLabel" translate=
                "company_settings_contact_postal_code" for="postal_code"></label>
98              <input type="text" id="postal_code" ng-model=
                "newOffice.address.postalCode" required>
99              <span field-error="postalCode"></span>
100
101             <br>
102
103             <label class="fiftyPercentLabel" translate=
                "company_settings_contact_postal_city" for="postal_city"></label>
104             <input type="text" id="postal_city" ng-model=
                "newOffice.address.postalCity" required>
105             <span field-error="postalCity"></span>
106
107             <br>
108         </form>
109     </div>
110
111     <div class="modal-footer">
112         <button class="btn btn-primary" ng-click="ok()"
113                 prevent-double-click="">{{'generic_ok' | translate}}</button>
114         <button class="btn btn-warning" ng-click="cancel()">{{'generic_cancel' |
            translate}}</button>
115     </div>
116 </script>
```

Liite 2. Kelpolounas AngularJS controller esimerkki.

```
/*
 * Copyright (c) 2015. Dicode Oy
 */

angular.module('companyManagementControllers').controller('companyOfficesController',
['$scope', '$rootScope', '$log',
        '$modal', '$timeout', 'AlertsService','companyService',
        'loggedInLuncherService','luncherService', '$translate',
    function ($scope, $rootScope, $log, $modal, $timeout, AlertsService,
    companyService, loggedInLuncherService, luncherService, $translate) {
        $scope.offices = companyService.getOfficesWithData();
        $scope.officeCreationAllowed = false;

        $scope.loggedInLuncher = loggedInLuncherService.getLoggedInLuncher().$promise
        .then(function (data) {
            $scope.loggedInLuncher = data;
        });

        $scope.addNewOffice = function (size) {
            var modalInstance = $modal.open({
                templateUrl: 'addNewOffice.html',
                controller: 'newOfficeModalCtrl',
                size: size,
                resolve: {
                    office: function () {
                        return $scope.newOffice;
                    }
                }
            });

            modalInstance.result.then(function (newOffice) {
                $scope.offices = companyService.getOfficesWithData();
            }, function () {
            });
        };

        $scope.removeOffice= function (office) {
            if (confirm($translate.instant("office_management_delete_office_confirm")
             + " " + office.name + "?")) {
                companyService.removeOffice(office.id).$promise.then(function () {
                    AlertsService.addSuccessByKey(
                    'office_management_delete_office_success');
                    $scope.offices = companyService.getOfficesWithData();
                });
            }
        };
    }])
    .controller('newOfficeModalCtrl', ['$scope','$translate', '$modalInstance',
    'AlertsService', 'companyService','office',
        function ($scope,$translate, $modalInstance,AlertsService,companyService,
        office) {
            'use strict';
            $scope.newOffice = {};

            $scope.ok = function () {
                AlertsService.clearAndUpdate();
                companyService.saveNewOffice($scope.newOffice).$promise.then(function
                 (newValue) {
                    AlertsService.addSuccessByKey("new_office_create_success");
                    $modalInstance.close(newValue);
                });
            };

            $scope.cancel = function () {
                $modalInstance.dismiss('cancel');
            };
        }]);
```

Liite 3. Kelpolounas AngularJS service esimerkki.

```
/*
 * Copyright (c) 2015. Dicode Oy
 */

angular.module('companyManagementServices')
    .service('companyService', [ '$rootScope','$resource','$location','$log', '$http'
    , 'dateHelper','AlertsService',
        function ($rootScope, $resource, $location,$log, $http, dateHelper,
        AlertsService) {
            'use strict';
            var _baseUrl = "/api/company";
            var _companyResource = $resource(_baseUrl+"/:action/:sub",{ action:'',
            sub:''},
                {
                    storeImage: {
                        method: 'POST',
                        params: {detailType: 'logo'},
                        transformRequest: angular.identity,
                        headers: {'Content-Type': undefined}
                    }
                });

            this.getCompanyToModify = function(id) {
                return _companyResource.get({action:'edit_current', companyId:id});
            };

            this.saveModifications = function(company, success, error) {
                if(company.logo && typeof(company.logo) === "object") {
                    var logo = company.logo;
                    company.logo = company.logo.name;
                }
                _companyResource.save({action : "update"},company).$promise.then(
                function(data) {
                    if(logo) {
                        var fd = new FormData();
                        fd.append('file', logo);
                        _companyResource.storeImage({action: "logo", companyId:
                        company.id}, fd).$promise.then(function(data) {
                            if (success) {
                                success(data);
                            } else {
                                AlertsService.addSuccessByKey(
                                "company_general_setting_save_success");
                            }
                        },error || function(error){
                            AlertsService.addAlert(error.data.message);
                        });
                    } else {
                        if (success) {
                            success(data);
                        } else {
                            AlertsService.addSuccessByKey(
                            "company_general_setting_save_success");
                        }
                    }
                },error || function (error) {
                    AlertsService.addAlert(error.data.message);
                });
            };

            this.getOffices = function(success, wihtoutCompany) {
                return _companyResource.query({action : "get_offices", wihtoutCompany
                : wihtoutCompany}, success);
            };

            this.listOffices = function() {
                return _companyResource.query({action : "list_offices"});
            };
```

```
60          };
61
62          this.getOffice = function(id) {
63              return _companyResource.get({action: "office", id : id});
64          };
65
66          this.getOfficeView = function(id) {
67              return _companyResource.get({action: "office", sub: id});
68          };
69
70          this.listOfficeEmployeesLunchTicketStatus = function(params) {
71              params.action="office_lunch_ticket_list" ;
72              return _companyResource.query(params);
73          };
74
75          this.removeOffice = function(id) {
76              return _companyResource.remove({action: "office", id:id});
77          };
78
79          this.saveOfficeInfo = function(office) {
80              return _companyResource.save({action: "office"},office);
81          };
82
83          this.getOfficesWithData = function() {
84              return _companyResource.query({action : "get_offices_data"})
85          };
86
87          this.saveNewOffice = function(office) {
88              return _companyResource.save({action: "new_office"},office);
89          };
90
91          this.getOfficeBillingToModify = function(id) {
92              return _companyResource.get({action : "billing", id : id});
93          };
94
95          this.getOfficeExpenseAccountBillingToModify = function(id) {
96              return _companyResource.get({action : "expense_account_billing", id :
                 id});
97          };
98
99          this.modifyOfficeBilling = function(billing) {
100             return _companyResource.save({action : "billing"},billing);
101         };
102
103         this.modifyOfficeExpenseAccountBilling = function(billing) {
104             return _companyResource.save({action : "expense_account_billing"},
                billing);
105         };
106
107         this.modifyCompanyBilling = function(billing) {
108             return _companyResource.save({action : "company_billing"},billing);
109         };
110
111         this.modifyCompanyExpenseAccountBilling = function(billing) {
112             return _companyResource.save({action :
                "company_expense_account_billing"},billing);
113         };
114
115         this.getCompensationTypes = function() {
116             return _companyResource.query({action : "compensation_types"})
117         };
118
119         this.getCompanyTaxationToModify = function(id) {
120             return _companyResource.get({action : "taxation", companyId:id});
121         };
122
123         this.saveCompanyTaxation = function(taxation) {
124             return _companyResource.save({action : "taxation"},taxation);
125         };
126
```

```
127             this.getOfficeTaxationToModify = function(id) {
128                 return _companyResource.get({action : "office_taxation",id:id});
129             };
130
131             this.saveOfficeTaxation = function(id,taxation) {
132                 return _companyResource.save({action : "office_taxation",id:id},
                    taxation);
133             };
134
135             this.listEmployees = function(params) {
136                 return $http.get(_baseUrl+"/employees", {params: params});
137             };
138
139             this.listSummaries = function(params) {
140                 return $http.get(_baseUrl+"/summary", {params: dateHelper.toDates(
                    params, ["beginDate","endDate"])});
141             };
142
143             this.getCompanyBillingToModify = function(id) {
144                 return _companyResource.get({action:"company_billing", companyId:id});
145             };
146
147             this.getCompanyExpenseAccountBillingToModify = function(id) {
148                 return _companyResource.get({action:"company_expense_account_billing"
                    , companyId:id});
149             };
150             this.listRestaurants = function(params) {
151                 return _companyResource.query({action : "restaurants_for_report",
                    params: dateHelper.toDates(params || {},
152                         ["effectiveAt"])})
153             };
154         }]);
```