Haaga-Helia
ammattikorkeakoulu Oy

AngularJS yksisivuisen web-applikaation kehitysalustana

Tuomo Karhu

18.11.2015

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

18.11.2015

Tekijä(t)
Tuomo Karhu

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
AngularJS yksisivuisen web-applikaation kehitysalustana

Sivu- ja liitesivumäärä
25 + 0

Opinnäytetyön nimi englanniksi
AngularJS as a development platform for Single-Page Applications

Yksisivuiset web-applikaatiot (SPA-sovellukset) ovat yleistyneet viimeisten kymmenen vuoden aikana, ja näiden avulla on ollut mahdollista tuoda verkkosivuston käyttökokemus lähemmäksi aitojen ohjelmasovellusten vastaavaa. Yksisivuisten web-applikaatioitten kehitystyöhön tarkoitetuista sovelluskehyksistä AngularJS on yksi käytetyimmistä ja suosituimmista.

Työn tavoitteena on selvittää miten AngularJS-sovelluskehys soveltuu SPA-sivustojen kehitykseen sovelluskehittäjän näkökulmasta, sekä millaisia ongelmatilanteita aloittelevat sovelluskehittäjät saattavat useimmiten kohdata AngularJS:ään tutustuessaan. Työ toteutetaan syksyllä 2015.

Työssä käsitellään AngularJS-sovelluskehyksen toiminnan kannalta keskeiset osa-alueet teorian tasolla, mutta varsinaista sovellusta ei tämän työn yhteydessä tuoteta. Lähdemateriaalina AngularJS:n kohdalla esiin nousevien ongelmatilanteiden määrittelyssä on käytetty internet-artikkeleita, StackOverflow-sivustolla esitettyjen AngularJS-aiheisten kysymysten jakaumaa aihepiireittäin sekä kirjoittajan omakohtaisia kokemuksia.

Tuloksissa esitellään AngularJS-sovelluskehystä luonnehtivan haasteellisuuden pääasiallisiksi lähteiksi niin ongelmat direktiivien ohjelmoinnissa, JavaScriptin prototyyppipohjaisen periytyvyyden käsitteellinen haastavuus, kuin AngularJS:n muista JavaScript-kirjastoista poikkeava tapa käsitellä HTML-sivuston dokumenttiobjektimallia.

Pohdinnassa todetaan, että AngularJS on haasteellinen ympäristö omalaatuisten toiminnallisten ratkaisujensa kannalta sekä aikaisempaa kokemusta muista sovelluskehyksistä omaaville että aloitteleville web-kehittäjille. Lisäksi johtopäätelmissä todetaan, että monimutkaisempien yksisivuisten web-applikaatioiden kohdalla AngularJS:n tuomat hyödyt ylittävät kaikesta huolimatta sovelluskehyksen opiskeluun käytetyn ajallisen panostuksen.

Asiasanat
AngularJS, Yksisivuiset web-applikaatiot, Web-kehitys, JavaScript

HAAGA-HELIA
University of Applied Sciences

Abstract

18.11.2015

Author(s)
Tuomo Karhu

Degree programme
Degree programme in Business Information Technology

Report/thesis title	**Number of pages and appendix pages**
AngularJS as a development platform for Single-Page Applications	25 + 0

Single-Page Applications (SPAs) have become very popular during the last ten years, mostly because SPAs offer fluent, traditional software like user experience for web-pages. AngularJS is one of the most widely used SPA development frameworks these days.

The aim of this thesis is to find out how suitable AngularJS is for this purpose in the developer's point of view. The main points of interest of this study are problem situations that are prevalent among developers new to AngularJS. This thesis is executed in autumn 2015.

The main aspects of AngularJS framework are introduced here in theory. However, fully functional application does not belong in the scope of this thesis. Source material for this thesis is based on online articles, AngularJS specific posts on StackOverflow community site and author's personal experience on the subject matter.

Results of this thesis show that the main factors behind AngularJS specific difficulties include problems with writing directives, conceptual ambiguity of JavaScript's prototypal inheritance and restrictions in handling Document Object Model in AngularJS.

The discussion concludes that AngularJS is a challenging framework for both novice web developers and developers who have previous experience in other frameworks, mainly because of functional aspects unique to AngularJS. On the other hand, benefits of AngularJS can be seen worth the effort of learning how to cope with this framework in situations where SPA-application becomes more complex.

Keywords
AngularJS, Single Page Application, Web development, JavaScript

Sisällys

1 Johdanto

Internet-sivustojen kehitystrendi perinteisistä useisiin alisivuihin jaetuista ratkaisuista kohti dynaamisia yksisivuisia web-applikaatioita on viimeisten kymmenen vuoden aikana ollut voimakkaassa nosteessa (Rufus, 2014, 9). Yksisivuiset web-applikaatiot (single page application, SPA) ovatkin vakiintuneet lähestulkoon normiksi osana sujuvaa internetin käyttökokemusta. Siirtymä staattisesta internetistä kohti interaktiivisempaa ”vuoropuhelua” käyttäjän ja websivuston välillä ei sen sijaan ole sujunut täysin ilman vastoinkäymisiä.

Perinteisiä websivustoja luonnehtivat hyperlinkkilistaukset, joiden takaa löytyy jokin dokumentti tai sivuston aihepiiriin liittyvä tietosegmentti. Tällainen sisällön tiedostorakennemainen esitystapa yhdistettynä siihen, että sivustolla vierailevan kävijän rooli on ainoastaan vastaanottaa tietoa, on sanellut periaatteet internet-selainteknologian kehitykselle. Tämän seurauksena siirtymä kohti dynaamisempia sekä interaktiivisempia websivustoja ei ole tapahtunut pelkästään näitä varten kehitettyjen teknologioiden ehdoilla, sillä selainten asettamat rajoitteet on jouduttu huomioimaan kaiken muun lisäksi.

Käsite Web 2.0 lanseerattiin kuvastamaan muutosta, jossa vuorovaikutuksellinen osallistuminen nousi keskeiseksi tekijäksi internetissä. Web 2.0 ei ole erityisen tarkasti rajattu kokonaisuus teknisessä mielessä, vaan se on pikemminkin paradigmamuutos sille, miten internetsivustojen kehitystyö muuttui palvelemaan paremmin kävijöiden tarpeita ja odotuksia niin vuorovaikutuksellisuuden kuin käytettävyyden osalta. Siinä missä internetin tarkoituksena oli aikaisemmin toimia tiedonlähteenä, nyt siitä oli tulossa alusta käyttäjien luoman sisällön jakamiselle. Käytännön tasolla siirtymä voidaan havaita esimerkiksi henkilökohtaisten websivujen suosion vähenemisessä ja painopisteen siirtymisestä kohti blogien pitämistä. (O'Reilly, 2005.)

Jotta sivusto palvelisi kävijöitään moitteettomasti Web 2.0:n asettamissa puitteissa, web-kehittäjien tuli kiinnittää huomiota siihen miten sivuston käytettävyydestä saatiin mahdollisimman sujuvaa. Uuden sivun lataaminen palvelimelta vastauksena jokaiseen kävijän suorittamaan toimenpiteeseen ei enää ollut ratkaisuna tällaisissa tarkoitusperissä kelvollinen. Sen sijaan teksti-, grafiikka- ja multimedia-elementtien päivittämisellä ainoastaan osassa sivunäkymää saatiin aikaan ohjelmasovelluksen kaltainen käyttökokemus.

Ei liene sattumaa, että interaktiivisten internetsivustojen kehitystrendi sekä internetin välityksellä toimivien palveluiden määrän kasvu osuvat ajallisesti yhteen. Kilpailutilanteessa asiakkaista palveluntarjoajat pyrkivät, tai ainakin tulisi pyrkiä tarjoamaan parasta käyttäjäkokemusta asiakkailleen. Tämän seurauksena dynaamiset internetsivustot ovat vaihtoehtona väistämättä houkuttelevammat verrattuna perinteisiin sivustoihin. Toinen suuri murros internetsivustojen kehityksessä on ollut mobiilin internetin aikakausi kosketusnäytöllisten älypuhelinten ilmaannuttua markkinoille.

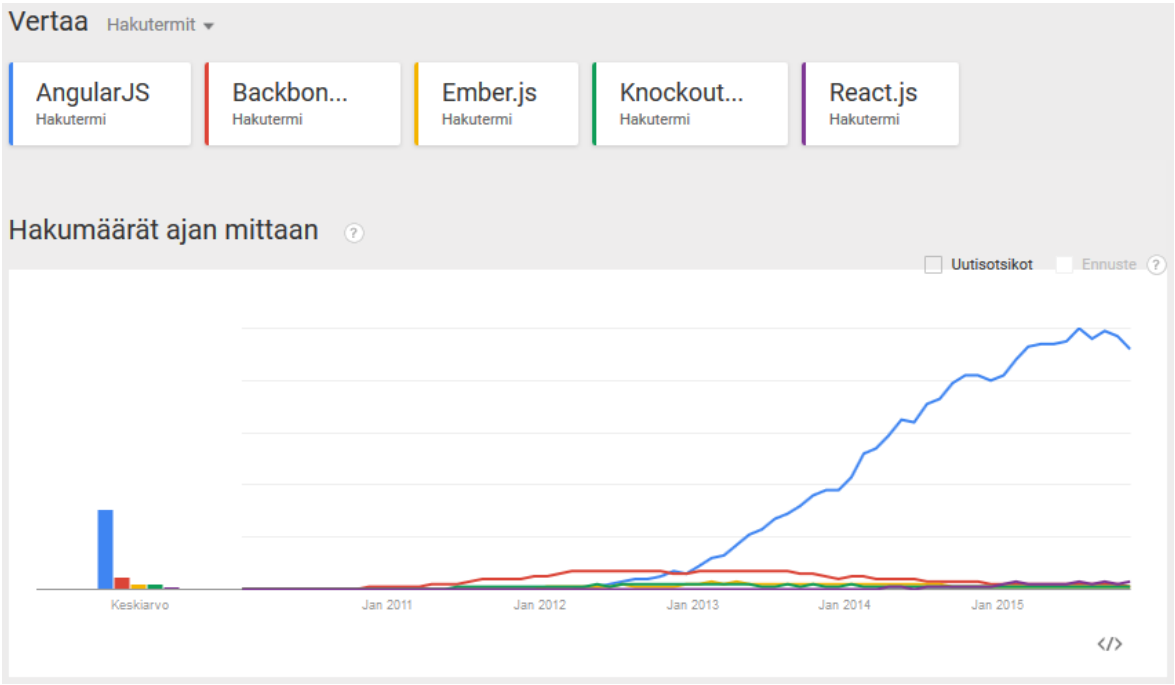
Mobiilin internetin mukanaan tuomiin web-kehityksen haasteisiin lukeutuvat muun muassa sivustojen skaalautuvuus erikokoisille päätelaitteille sekä käytettävissä olevan langattoman internet-yhteyden tiedonsiirtokapasiteetti. Perinteiset internetsivustot soveltuvat näihin rajoitteisiin huonosti. Perinteisissä internetsivustoissa sivun templaatti voi toistua jokaisella alisivulla hieman erilaisina variantteina, jolloin tiedonsiirron rasitteena on toistuva saman materiaalin lataaminen palvelimelta näkymää vaihdettaessa. Rakenteeltaan dynaamisemmat internetsivut soveltuvat teknologisessa mielessä mobiililaitteisiin paremmin, sillä varsinaisen sisällön osuus siirtyvästä tiedosta on tehokkaampaa ja siirtyminen sivuston näkymästä toiseen on sulavampaa. Sulavuuden hintana tällaisissa yksisivuisissa web-applikaatiossa on se, että sivuston ensimmäinen latautuminen on hidasta. Tällöin palvelimelta ladataan sekä sivusto että sen toiminnan kannalta olennaiset skriptikomponentit (Rufus, 2014, 11).

Yksisivuisia web-applikaatioita voidaan pitää sekä dynaamisempaan että mobiililaitteisiin paremmin soveltuvaan internetiin johtaneena kehityskaaren päänä.

1.1 Tavoitteet, tutkimusongelman asettelu sekä rajaukset

Tämän tutkimuksen tavoitteena on kartoittaa miten Googlen kehittämä, yksisivuisten web-applikaatioitten kehitystyökalu, AngularJS-sovelluskehys soveltuu tarkoitukseensa sovelluskehittäjän näkökulmasta. Tutkielmassa tarkastellaan millaisia vahvuuksia AngularJS-ympäristö tarjoaa, ja vastaavasti millaisiin ongelmakohtiin web-kehittäjä saattaa törmätä hyödyntäessään sovelluskehystä osana työtään. Vaikka JavaScript-pohjaisia sovelluskehyksiä on tällä hetkellä runsaasti tarjolla vastaavanlaiseen tarkoitukseen, keskitytään tutkimuksessa AngularJS:ään siitä syystä, että tällä hetkellä AngularJS on ylivoimaisesti suosituin JavaScript-pohjainen sovelluskehys. Kuvassa 1 on esitetty Google-hakujen määrällä mitattuna AngularJS-, Backbone.js-, Ember.js-, Knockout.js- ja React.js-ympäristöjen suosion kehitystrendit aikavälillä 2010 - 2015. AngularJS:llä onkin suosionsa perusteella vahva asema liiketoiminnassa web-sovellusten kehitysalustana, joten kirjoittajalla on AngularJS:stä tästä syystä omakohtaisesti eniten

kokemusta. Aineistona käytetään kirjallisuudessa ja aihepiiriin liittyvien keskustelupalstojen käyttäjien kokemuksia vertailukohtana omille havainnoille, sekä lähteenä sille, millaisissa ongelmatilanteissa on useimmiten jouduttu hakemaan ulkopuolista apua osana ongelman ratkaisua.



Kuva 1. AngularJS:n, Backbone.js:n, Ember.js:n, Knockout.js:n sekä React.js:n suosion kehitystrendit perustuen Google-hakujen lukumääriin aikavälillä 2010-2015 (Google Trends, 2015).

1.2 Käsitteet

AJAX (Asynchronous JavaScript and XML)

Teknologiakehys, joka mahdollistaa selaimen ja palvelimen välisen, epäsynkronisesti toimivan kommunikaation.

AngularJS

Googlen kehittämä ja ylläpitämä JavaScript-pohjainen sovelluskehys.

DOM (Document Object Model)

Dokumenttioliomalli, jonka avulla voidaan kuvata html-, xml- ja xhtml-dokumentit puumaisena rakenteena.

JavaScript

Scriptikieli, jolla voidaan tuottaa selainpäädyssä dynaamisia toiminnallisuuksia.

MVC-arkkitehtuuri

Ohjelmistoarkkitehtuurityyli, jossa ohjelma jaotellaan rakenteellisesti kolmeen erilliseen osaan:

1. Malliin (M=model), jolla kuvataan järjestelmän tietokantakuvaus tiedon tallentamisen ja käsittelyn muodossa
2. Näkymään (V=view), jossa määritellään ohjelman ulkoasu käyttöliittymän ja tietojen esitystavan muodossa
3. Ohjaimeen (C=controller), joka vastaanottaa käyttäjän komennot muuttaakseen sekä mallia että näkymää vastauksena näihin.

Perinteinen web-sivusto

HTTP-protokollan kautta palvelimelta verkon yli siirrettävä dokumenttimuotoinen sivusto, jossa navigointi sivujen välillä tapahtuu linkkien kautta.

Sovelluskehys (framework)

Sovelluskehys on uudelleenkäytettävä ohjelmistokehitykseen tarkoitettu ympäristö, joka sisältää eriasteisesti valmiiden toiminnallisuuksien rungon, jonka päälle kehitettävä sovellus voidaan rakentaa.

Yksisivuinen web-applikaatio (single page application, SPA)

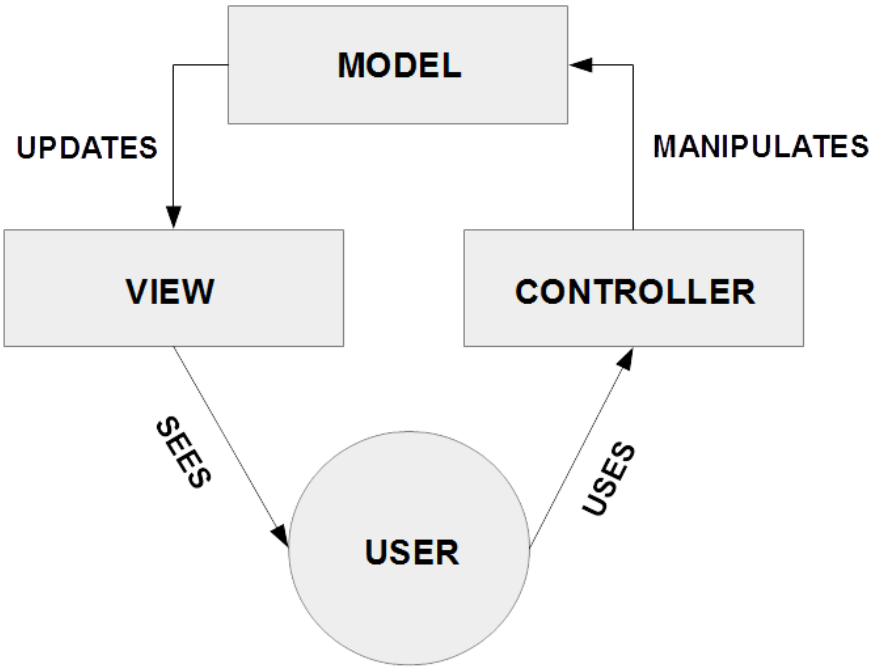
Internetsivuston toteutustapa, jossa näkymän päivitys toteutetaan yhdellä html-sivulla ilman tarvetta erillisille sivulatauksille. Toteutus mahdollistaa dynaamisemman ja varsinaista ohjelmasovellusta muistuttavan käytettävyyden.

2 Tietoperusta

2.1 MVC-arkkitehtuuri

MVC-arkkitehtuurilla tarkoitetaan sovelluskehityksessä periaatetta, jossa sovelluksen rakenne jaetaan kolmeen erilliseen osaan. Lyhenteen MVC mukaisesti nämä ovat sovelluksen tietorakenteesta vastaava malli (Model), käyttöliittymästä vastaava näkymä (View) sekä sovelluslogiikasta vastaava ohjain (Controller). (Osmani, 2012a.)

MVC-mallin mukainen sovellusarkkitehtoninen ratkaisu, jossa ohjelman vastuualueet hajautetaan toisistaan erillisiksi kokonaisuuksiksi sekä näiden väliset keskinäiset vuorovaikutukset on esitetty kuvassa 2.



Kuva 2. MVC-arkkitehtuurin mukaiset sovelluksen rakenteelliset osat: malli, näkymä sekä ohjain. Kuvassa on esitetty lisäksi sovelluksen käyttäjä, sekä vuorovaikutukset osien välillä. (RegisFrey, 2010.)

MVC-arkkitehtuurin pohjalta kehitetty, web-applikaatioihin tarpeisiin sovellettu versio on nimeltään MVVM-arkkitehtuuri (Model-View-ViewModel). MVVM-arkkitehtuuri eroaa MVC-arkkitehtuurista siinä, että ViewModel toimii ikään kuin ohjaimena joka muuntaa mallin sisältämää tietoa näkymässä hyödynnettäväksi tiedoksi ja vuorostaan välittää näkymässä suoritettuja komentoja malliin. Keskeinen piirre MVVM-arkkitehtuurille on käyttöliittymän ja sovelluslogiikan entistä perusteellisempi erottelu toisistaan. (Osmani, 2012b.)

2.2 Yksisivuinen web-applikaatio

Käsite yksisivuinen web-applikaatio (SPA) kertoo jo lähtökohtaisesti keskeisimmän: kyse on yksittäisestä HTML-sivusta, joka ladataan muiden tarpeellisten resurssien kuten css- ja skripti-tiedostojen kanssa palvelimelta kokonaisuudessaan applikaation käynnistyessä. Tämän jälkeen sivun koko esityslogiikka on selaimen käytettävissä, jotta käyttäjälle voidaan luoda interaktiivinen, perinteisen sovelluksen käytettävyyttä jäljittelevä käyttökokemus (Rufus, 2014, 9). Yksisivuisten web-applikaatioitten keskeinen eroavaisuus perinteisiin web-sivustoihin nähden muodostuu näin ollen palvelimen vähäisemmästä merkityksestä esityslogiikan osalta. Näin ollen SPA:t toimivat käyttäjän selaimessa paikallisesti ilman, että verkon yli siirrettäisiin toistuvasti kokonaisia HTML-dokumentteja (Rufus, 2014, 11).

Palvelimeen kohdistuvan kuorman väheneminen SPA-ratkaisuissa perustuu siihen, että MVC-arkkitehtuurin mukainen ohjelmiston rakenne on siirtynyt palvelimelta enenevissä määrin käyttäjän selaimen hoidettavaksi. Tällaista MVC-mallin jakoa palvelimen ja selaimen välillä eri sivustoarkkitehtonisissa ratkaisuissa voidaan kuvata kolmiportaisesti:

1. Perinteisissä sivustoissa käyttäjän suorittamat toimenpiteet, hyvänä esimerkkinä käyttäjän syötteiden validointi, käsitellään palvelimella. Palvelimen vastuulle jää tällöin MVC-mallin kaikki osa-alueet, sillä palvelin huolehtii myös selainnäkymän ohjaamisesta uudelle sivulle.
2. Ajaxia hyödyntävissä sivustoissa näkymän osia voidaan päivittää jo selaimessa. Esimerkkinä tästä mainittakoon edellä esitetty käyttäjän syötteiden validointi, jolloin yhteen syötekenttään annetun tiedon perusteella validoidaan jo valmiiksi taustalla toisen syötekentän arvoa (kuten postitoimipaikkakunnan ja postinumeron vastaavuus). Tällä tasolla toteutetuissa sivustoissa MVC-mallin mukaisen näkymän prosessoinnista vastaavat sekä selain että palvelin, mutta mallin ja ohjaimen käsittely kuuluu edelleen palvelimelle.
3. SPA-ratkaisuissa MVC-malli voi sen sijaan olla toteutettuna selainpäädyssä. Muun muassa SPA-sivustoiden toteutuksessa käytettävä AngularJS-sovelluskehys vastaa niin mallin, ohjaimen kuin näkymän käsittelystä. Ohjain vastaanottaa käyttäjän syötteet ja käsittelee mallia, joka puolestaan päivittää näkymää. Kuten aikaisemmin on jo mainittu, SPA-sivustoissa näkymät eivät ole varsinaisesti erillisiä HTML-sivuja vaan eräänlaisia virtuaalisia presentaatioita tiedosta yhdellä ja samalla sivulla, joka muodostuu MVC-mallin kautta.

2.3 AngularJS

Googlen kehittämä AngularJS on sovelluskehys dynaamisille web-applikaatioille, joka soveltuu varsinkin yksisivuisten web-applikaatioitten kehitystyöhön. AngularJS:n ominaispiirteisiin kuuluvat HTML-pohjaiset templaatit sekä laajennettu HTML-syntaksi, jonka kautta web-applikaation toiminnallisuus liitetään MVC-mallin mukaiselle näkymätasolle.

AngularJS-sovelluskehyksen liittäminen osaksi sivustoa tapahtuu lataamalla sovelluskehyksen toiminnan kannalta keskeinen JavaScript-tiedosto <script>-elementin kautta. Kuvan 3 mukaisella esimerkkisivustolla on AngularJS:stä käytössä versio 1.3.14. Kirjoitushetkellä AngularJS:n uusimman tuotantoversion versionumero on 1.4.7. (AngularJS, 2015a.)

```
1  <!DOCTYPE html>
2  <html>
3  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
4  <body>
5
6  <div ng-app="">
7    <p>Name: <input type="text" ng-model="name"></p>
8    <p ng-bind="name"></p>
9  </div>
10
11 </body>
12 </html>
```

Kuva 3. AngularJS-sovelluskehyksen liittäminen sivustolle <script>-elementissä rivillä 3. (W3Schools, 2015a.)

AngularJS ratkaisee HTML:n dokumenttimaisen rakenteen ja sivustojen vuorovaikutukselliselle käyttökokemukselle kohdistuvan teknologisen tarpeen välisen ristiriidan lisäämällä HTML:ään uusia komponentteja. Keskeisimmät näistä ovat direktiivit, joiden välityksellä AngularJS-sovelluksen on tarkoitus käsitellä dokumenttiobjektimallia (DOM). Toisin sanoen direktiivien välityksellä voidaan vaikuttaa HTML-dokumentin attribuutteihin ja elementteihin. Sovelluskehyksen tarjoamien valmiiden direktiivien lisäksi sovelluskehittäjä voi tehdä näitä itse tarpeen mukaan.

AngularJS:n tarjoamat valmiit direktiivit ovat ng-etuliitteellä merkittyjä, laajennettuja HTML-attribuutteja (W3Schools, 2015b). Direktiiveistä voidaan käyttää myös data-ng-etuliitteellistä muotoa, jonka HTML-validointityökalut hyväksyvät (AngularJS, 2015b).

Sovelluskehyksen tarjoamista valmiista direktiiveistä keskeisimmät ovat:

1. ng-app-direktiivi; AngularJS-sovelluksen alustukseen tarvittava pakollinen direktiivi, jolla määritetään AngularJS-applikaation ”omistava” elementti
2. ng-init-direktiivi, jolla voidaan asettaa tarvittavat lähtöarvot sovelluksen malliin
3. ng-model-direktiivi, jolla voidaan asettaa arvoja sovelluksen malliin HTML:n input-elementtien kautta

Kuvassa 4 olevassa erimerkkisovelluksessa ng-app-direktiivi on esitelty -elementissä, joka tällöin ”omistaa” sovelluksen ja määrittää lisäksi sovelluksen juurielementin (engl. root element). Jotta AngularJS-sovellus käsittäisi koko sivuston, tulee ng-app-direktiivi esitellä <html>-elementissä. Ng-app-direktiivi alustaa sovelluksen automaattisesti kun sivusto on latautunut. Esimerkkitapauksesta poiketen ng-app-direktiiville on tyypillisesti määritetty jokin arvo, jolloin sovellusmoduleita voidaan yhdistellä toisiinsa. Kuvassa 4 on lisäksi ng-init-direktiivin kautta asetettu muuttujan firstName alkuarvoksi 'John', sekä kytketty tämä muuttuja <input>-elementin arvoon ng-model-direktiivillä. (W3Schools, 2015b.) Hyviin kehityskäytäntöihin tuotantoympäristössä kuuluu, että muuttujat tulisi alustaa ng-init-direktiivin asemasta sovelluksen ohjain-moduleissa (Rufus, 2014, 12).

```
1  <div ng-app="" ng-init="firstName='John'">
2
3  <p>Name: <input type="text" ng-model="firstName"></p>
4  <p>You wrote: {{ firstName }}</p>
5
6  </div>
```

Kuva 4. AngularJS-sovelluksen liittäminen elementtiin, lähtöarvon asettaminen sekä mallin yhdistäminen muuttujaan.

Kokoelmien iterointiin tarkoitettu ng-repeat-direktiivi on yksi AngularJS:n keskeisimmistä sisäänrakennetuista direktiiveistä. Ng-repeat-direktiivi käy annetun kokoelman läpi ja tekee HTML-elementeistä klooneja yhden kutakin kokoelman solua kohti. (W3Schools, 2015b.) Kuvassa 5 on esitetty ng-repeat-direktiivin käyttö AngularJS-sovelluksen näkymässä osana (listan) -elementtiä. Annettu kokoelma tulostuu näkymässä muodossa:

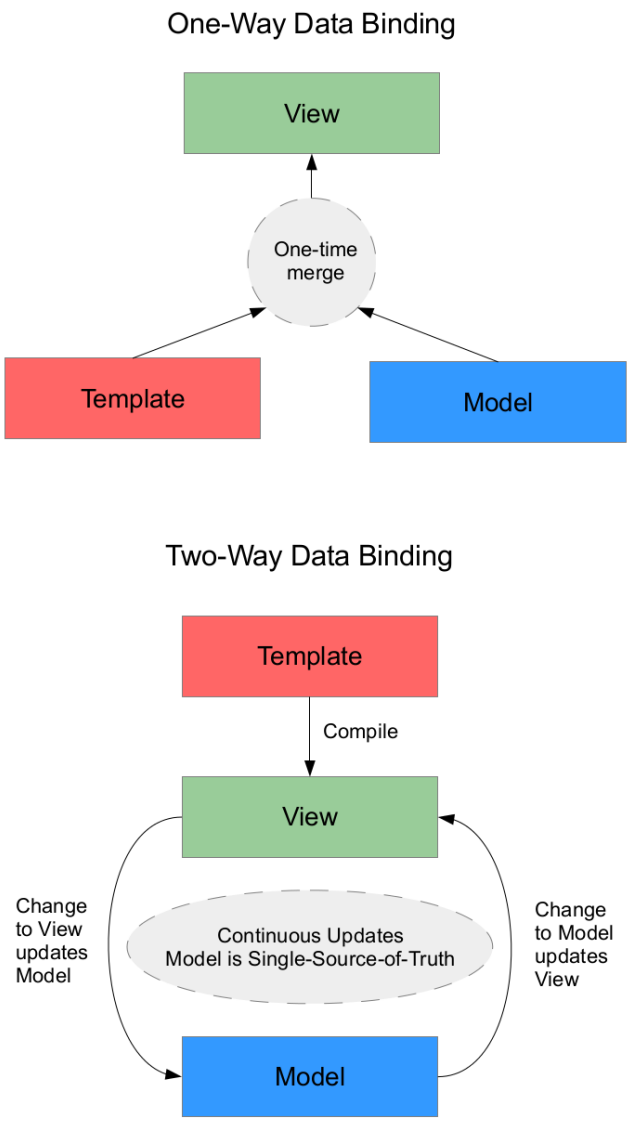
Jani, Norway
Hege, Sweden
Kai, Denmark

```
<div ng-app="" ng-init="names=[
{name:'Jani',country:'Norway'},
{name:'Hege',country:'Sweden'},
{name:'Kai',country:'Denmark'}]">

<ul>
  <li ng-repeat="x in names">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>
```

Kuva 5. Kokoelmien iterointiin tarkoitetun ng-repeat-direktiivin käyttö.

AngularJS:n kannalta yksi keskeisimmistä toiminnallisista periaatteista on kaksisuuntainen datakytkentä (two-way data binding). MVC-arkkitehtuurin tasolla datakytkennän kaksisuuntaisuudella tarkoitetaan sitä, että näkymässä tehdyt muutokset välittyvät malliin ja vastaavasti mallissa tapahtuvat muutokset päivittyvät näkymään (Kuva 6). (AngularJS, 2015c.)

Välittäjänä tässä datakytkennässä toimii ohjain. Mallin data esitetään AngularJS:ssä skoopin (scope) välityksellä, ja skoopin tilan asetus suoritetaan ohjaimen kautta. Skoopin ominaisuuden arvo esitetään näkymän tasolla AngularJS:n syntaksissa kaksoisaaltosulkeissa {{}}. AngularJS:ssä myös lausekkeet ilmaistaan näkymässä skoopin ominaisuuksien tavoin kaksoisaaltosulkeissa (Kuva 7). Lausekkeet voivat sisältää muun muassa merkkijonoja, laskutoimituksia ja muuttujia (W3Schools, 2015c). Esimerkkinä AngularJS:n datakytkennästä mallin ja näkymän välillä on kuvassa 7, rivillä 8 oleva lauseke {{ lastName }}.



Kuva 6. Yksi- ja kaksisuuntaisen datakytkennän eroavaisuudet. AngularJS:n näkymässä ja mallissa tapahtuvat muutokset välittyvät molempiin suuntiin.

```
1  // Merkkijono
2  {{ "abcdefg" }}
3
4  // Laskutoimitus
5  {{ 1 + 2 }}
6
7  // Muuttuja $scope.lastName
8  {{ lastName }}
```

Kuva 7. Lausekkeiden ja skoopin ominaisuuden arvon tulostus näkymässä (W3Schools, 2015c).

Toiminnallisuus AngularJS-sivustoihin toteutetaan lisäämällä html-koodiin elementtikohtaisia ohjaimia direktiivien välityksellä. Ohjaimessa asetetaan skoopin tila, lisäksi eri komponenttien kytkeminen toisiinsa on ohjaimen vastuulla. Sovelluksen business-logiikka, eli varsinainen käsittelylogiikka toteutetaan ohjaimissa. (Horsmalahti, 2014.)

Kuvan 8 esimerkkisovelluksessa ohjain 'myCtrl' on määritetty -elementtiin ng-controller-direktiivin välityksellä. Funktio 'myCtrl' on JavaScript-funktio, ja $scope (skooppi) on AngularJS:n yhteydessä applikaatio-objekti joka ”omistaa” tämän muuttujat ja funktiot. (W3Schools, 2015d.)

Kuvan 8 esimerkkisovelluksen ohjaimessa määritellyt skoopin ominaisuudet:

$scope.firstName=”John”, ja
$scope.lastName=”Doe”

liitetään näkymään lausekkeella:

{{firstName + ” ” + lastName}}

Näkymässä tulostuu tällöin rivi:

Full Name: John Doe

```
<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
});
</script>
```

Kuva 8. Ohjaimen käyttö AngularJS:ssä.

Näkymistä riippumaton käsittelylogiikka kuuluu sen sijaan Angularissa palveluiden (services) piiriin. Palveluiden kautta voidaan käsitellä dataa ja suorittaa tietokantaan liittyviä toimenpiteitä, kuten tiedon hakemista ja tallentamista. Palvelut ovat ohjain-riippumattomia, joten niitä voidaan kierrättää ja hyödyntää myös sovelluskoodin muissa osissa. (Horsmalahti, 2014.)

Jotta ohjaimella olisi pääsy palveluissa sijaitseviin funktioihin, tarvitaan riippuvuusinjektiota (dependency injection). AngularJS-kehittäjä voi määritellä ohjainmoduulin käytettävissä olevat palvelut injektoimalla ne tähän. Riippuvuusinjektio on konseptina keskeinen osa AngularJS:ää, sillä niin direktiivit, filtterit, ohjaimet, palvelut kuin muutkin komponentit kytketään ja luodaan tarvittaville paikoilleen sovelluksessa injektion kautta. (AngularJS, 2015d.)

Kuvan 9 esimerkkitapauksessa on esitetty riippuvuusinjektio AngularJS-sovelluksen 'MyController'-nimiseen ohjaimeen. 'dep1' ja 'dep2' ovat injektoituja palveluita, jotka ovat injektion jälkeen kyseisen ohjaimen käytettävissä. Ohjain-moduli liittyy aina tiettyyn elementtiin dokumenttiobjektimallissa (DOM), joten injektoimalla ohjaimeen $scope-objekti mahdollistetaan ohjaimen pääsy tähän. (AngularJS, 2015d.)

```
someModule.controller('MyController', ['$scope', 'dep1', 'dep2', function($scope, dep1, dep2) {
  ...
  $scope.aMethod = function() {
    ...
  }
  ...
}]);
```

Kuva 9. Riippuvuusinjektio AngularJS-ohjaimessa.

Yhteys palvelimeen muodostetaan AngularJS:n sisäänrakennetun $http-palvelun avulla. $http on niin kutsuttu kääreluokka (wrapper), joka liittyy palvelinyhteyksissä käytettäviin XMLHttpRequest ja JSONP -teknologioihin (AngularJS, 2015e).

3 Tutkimus ja tutkimusmenetelmä

Tässä tutkimuksessa käsitellään AngularJS-sovelluskehyksen soveltuvuutta yksisivuisten web-applikaatioitten kehitykseen sovelluskehittäjän näkökulmasta tilanteessa, jossa kehittäjällä ei ole aikaisempaa kokemusta sovelluskehyksestä. Tutkimus toteutetaan lähdeaineistoja vertailemalla. Tarkoituksena on tuottaa katsaus AngularJS:lle tyypillisistä ominaisuuksista, joiden hyödyllisyyttä ja etenkin helppokäyttöisyyttä pohditaan yksisivuisten web-applikaatioiden kehitystyön näkökulmasta yleisellä tasolla. Projektin keskeinen tutkimuskysymys on:

Millaisissa tilanteissa on perusteltua suositella AngularJS:n käyttöä yksisivuisen web-applikaation kehitysalustana?

Tutkimuskysymys on relevantti sillä JavaScript-kirjastoissa, kuten laajalti käytetyssä jQueryssä, näkymän ja sovelluksen toiminnallisuuden välillä on perusteellisempi jakolinja. AngularJS:n tarjoamat dynaamisemmat mahdollisuudet näkymätasolla parhaimmillaan helpottavat ja nopeuttavat kehitystyötä, mutta oletettavasti voivat aiheuttaa päänvaivaa ja riskin perustavanlaatuisille virheille pelkästään sovelluskehysarkkitehtuurin erilaisuudesta johtuen.

Tutkimuksen aineistona käytetään sovelluskehittäjien laatimia, AngularJS:n käytettävyyttä työvälineenä arvioivia internet-artikkeleita. Aineistona hyödynnetään myös Stackoverflow-kehittäjäsivustolla julkaistuja AngularJS:ää koskevia, kehitystyön parissa eteentulevia yleisimpiä ongelmakohtia käsittelevien kysymysten jakaumaa aihepiireittäin. Lisäksi aineistoa verrataan omakohtaisiin kokemuksiin AngularJS:n parissa eteen tulleista ongelmista.

4 Tutkimuksen toteuttaminen

Tutkimuksessa lähdettiin kartoittamaan AngularJS:n soveltuvuutta yksisivuisten web-applikaatioitten kehitykseen tarkastelemalla sivustoja, joille on koottu kehittäjäyhteisön kohtaamia ongelmatilanteita joita sovelluskehyksen käyttöönoton yhteydessä on havaittu. Tämä lähestymistapa valittiin, jotta tutkimuskysymykseen voitaisiin vastata web-kehittäjälähtöisistä näkökulmista.

Tutkimuksessa käsiteltiin kolmea AngularJS-aiheista internet-sivustoa, jotta voitiin selvittää millaisia sovelluskehykseen liittyviä ongelmakohtia on eri lähteissä havaittu. Lisäksi StackOverflow-sivustoa päädyttiin käyttämään osana aineistoa, jotta eri ongelmakohtien määrällisistä eroavaisuuksista saatiin muodostettua käsitys.

4.1 AngularJS-ongelmakohdat

AngularJS:n virallisella sivustolla (AngularJS, 2015f) on listattu kehittäjäyhteisön IRC-kanavan (#angularjs Freenode-verkossa) kautta ilmenneitä yleisimpiä ongelmakohtia, joihin uudet kehittäjät ovat törmänneet sovelluskehyksen parissa. Vailla varsinaista tietoa ongelmakohtien suhteellisesta yleisyydestä, nämä ovat:

1. Dokumenttioliomallin (DOM) virheellinen käsittely, kuten elementtien lisääminen, poistaminen tai näiden näkyvyyden tilan asettaminen jQueryn avulla ohjaimessa
2. Olemassa olevien AngularJS:n toiminnallisuuksien, kuten ng-repeat, ng-show ja ng-class-direktiivien suhteen päällekkäisen ohjelmakoodin tarpeeton tuottaminen
3. Ongelmat $watch ja $apply -metodien käytössä rekisteröitäessä omia direktiivejä osaksi applikaatiota, jolloin sovellus ei reagoi skoopin muuttuneeseen arvoon
4. Kokoelmien iterointiin tarkoitetun ng-repeat-direktiivin ja muiden direktiivien lisäämisestä samaan elementtiin aiheutuvat ongelmat
5. Globaalien muuttujien huolimattomasta käytöstä aiheutuvat ongelmat AngularJS:n $rootScope-objektissa

Chandermani (2014) on listannut Technovert-sivustolla yleisimmät AngularJS:n ongelmakohdat, joita hän on aktiivijäsenenä StackOverflow-kehittäjäyhteisössä kohdannut ja joita on hänen työyhteisössään useimmiten noussut esille. Chandermani nostaa seuraavat aiheet esiin ilman erityistä tärkeysjärjestystä:

1. Puutteellinen tai virheellinen tulkinta AngularJS:n kannalta keskeisestä käsitteestä skoopin prototyyppipohjainen periytyvyys, ja tästä johtuvat suunnitteluvirheet ohjelmakoodissa
2. Ongelmat JavaScriptin asynkronisen, eli ei-reaaliaikaisen luonteen omaksumisessa kun sovelluskehittäjällä on taustaa synkronisista ohjelmointikielistä
3. AngularJS:n templaattien luoma virheellinen vaikutelma toiminallisuuden synkronisesta luonteesta, joka ilmenee Promise Unwrapping -ominaisuuden kautta
4. Epäselvyys siitä miten ja missä tilanteessa AngularJS-applikaation kontekstista irrralliset ulkoiset kutsut suoritetaan ja $apply -metodin käyttö
5. Epäselvyys siitä miten AngularJS-applikaation kontekstista irrallinen skoopin arvo asetetaan
6. AngularJS:n keskeinen eroavaisuus jQuerysta MVC-rakennemallin suhteen siinä, että AngularJS:ssä malli ohjaa näkymää, ja tästä aiheutuvat ohjelmistoarkkitehtoniset ongelmat
7. Direktiivien käytön haasteellisuus
8. Tarpeettomat toistot AngularJS-moduleitten lisäämisessä ja näistä aiheutuvat ongelmat
9. Kokoelmien iterointiin tarkoitetun ng-repeat-direktiivin virheellinen käyttö
10. Internet-selaimen konsoli-ikkunan kautta saatavan hyödyllisen tiedon käyttämättömyys ohjelmakoodin ongelmia selvitettäessä

Herskovits (2014) nostaa Backand-sivustolla blogissaan esille samansuuntaisia ongelmia, joita on hänen kokemuksiensa mukaan noussut AngularJS -kehittäjäyhteisön parissa esille. Herskovitsin mukaan viisi yleisintä ongelmakohtaa eroavat toisistaan vakavuudeltaan, mutta ne voivat yhtäkaikki aiheuttaa turhaa haittaa sovelluksen kehitystyölle:

1. Virheellinen tapahtumankäsittelijän käyttötapa tilanteissa, joissa datan esittämiseen näkymätasolla tarkoitettuihin skooppeihin liitetään funktionaalista toimintaa
2. Virheellisesti ymmärretyt sisäkkäisten skooppien (nested scope) toimintaperiaatteet JavaScriptin primitiivi- ja ei-primitiiviobjektien kohdalla
3. Turhan monimutkaiset AngularJS-ohjainkomponentit ja näihin kuulumattomien toimintojen lisääminen
4. Internet-selaimen konsoli-ikkunan kautta saatavan hyödyllisen tiedon käyttämättömyys ohjelmakoodin ongelmia selvitettäessä
5. Epäidiomaattisen ohjelmakoodin tuottaminen, toisin sanoen AngularJS:lle ei-tyypillisiin ratkaisuihin turvautuminen ohjelmakoodissa, kuten dokumenttiobjektimallin (DOM) virheellinen käsittely

4.2 AngularJS-ongelmakohtien lukumäärät

Ongelmat etenkin direktiivien (n=11579), skooppien (n=5881) ja ng-repeat-direktiivin (n=4318) käytössä erottuvat StackOverflow-kehittäjäsivustolla (StackOverflow, 2015) esitettyjen AngularJS-aiheisten kysymysten (n=128602) lukumäärässä (Taulukko 1). Kysymyksen aihepiirin määritys perustuu käyttäjän kysymyksen yhteyteen lisäämään tagiin, eli merkkiin.

Taulukko 1. StackOverflow-sivustolla esitetyt AngularJS-aiheiset kysymykset aihepiireittäin. *) AngularJS-aiheiset kysymykset kokonaisuudessaan, **) Angular-UI-tiimin kehittämä sovelluskehyksen natiivin reititysmoduulin korvaava komponentti.

Tagi	**Lukumäärä (n)**
angularjs*	128602
angularjs-directive	11579
angularjs-scope	5881
angular-ui-router**	4438
angularjs-ng-repeat	4318
angularjs-service	1145
angularjs-routing	1003
angularjs-controller	779
angular-promise	764
angular-resource	506
angularjs-factory	281
angularjs-http	191
angularjs-module	130

5 Tulokset

AngularJS-sovelluskehykseen liittyvissä yleisimmissä ongelmakohdissa ilmenee kolmen lähdeartikkelin välillä jossain määrin eroavaisuuksia, mutta tutkimusaineiston perusteella osa ongelmakohdista nousee aineistossa toistuvasti esille. Huomioitaessa lisäksi StackOverflow-sivustolta kerätty aineisto AngularJS-aiheisten kysymysten lukumääristä aihepiireittäin, voidaan AngularJS:n toiminnallisuuden kannalta ongelmallisimmat osa-alueet jäsennellä tärkeysjärjestyksessä seuraavien aihekokonaisuuksien alle:

1. Direktiivien käyttö ja dokumenttiobjektimallin käsittelysäännöt AngularJS:ssä
2. AngularJS:n skoopit ja näiden prototyyppipohjainen periytyvyys
3. Kokoelmien iteroinnissa käytettävän ng-repeat-direktiivin käyttö
4. Ohjainmoduleitten suunnittelussa huomioitavat rajoitteet ja suositukset
5. Ohjelmakoodiin liittyvien ongelmien jäljitettävyys kehitystyössä

6 Pohdinta

MVC-arkkitehtuuriin pohjautuva AngularJS-sovelluskehys on yksisivuisten web-applikaatioitten kehitykseen tarkoitettu sovelluskehys. Yhtäältä AngularJS:n toiminnallisuuksien kehitystyön taustalla on vaikuttanut yksisivuisten web-applikaatioiden perinteisistä web-sivustoista poikkeavat teknologiset tarpeet. Toisaalta AngularJS:ää voi, osin seurauksena edellisestä, luonnehtia suhteellisen haastavaksi ympäristöksi tutustuttaessa tähän ensimmäistä kertaa.

6.1 AngularJS:n haasteellisuuteen vaikuttavat tekijät

Omakohtaiset kokemukset AngularJS:n kohdalla ilmenneistä ongelmista, joissa on joutunut turvautumaan internettiin tiedonlähteenä ratkaisua haettaessa, jakautuvat kahtalaisesti sen mukaan onko kyseessä tekninen ratkaisu johonkin selvään ongelmaan vai onko jokin konsepti alkujaankin liian abstrakti ollakseen intuitiivisesti käytettävissä ja vaatinut perusteellisempaa selvitystyötä ennen toteutusvaihetta.

Tuloksissa esitellyt, eri lähteistä peräisin olevat tyypillisimmät angularJS:n ongelmakohdat eivät tuo tätä ongelmien taustalla vaikuttavaa kaksijakoista näkökulmaa erityisen selvästi esille, mutta niin Herskovits (2014), Chandermani (2014) kuin AngularJS:n virallisella sivustolla esitelty aineisto (AngularJS, 2015f) nostavat hyvin yhdenmukaisia ongelmatilanteita esille. Näistä varsinkin AngularJS:n direktiiviaiheiset ongelmat voidaan luokitella vahvasti teknislähtöisiksi ongelmiksi. Myös StackOverflow-kehittäjäsivustolla (StackOverflow, 2015) esitettyjen direktiiviaiheisten kysymysten suuri lukumäärä (n=11579) herättää huomiota. Oletettavasti suuri osa näistä kysymyksistä onkin tarkasti kohdennettuja, ohjelmakoodin syntaksiin liittyviä, ja joiden taustalla vaikuttava kysymys on ”miten?”. Omakohtaisten kokemusten perusteella StackOverflow-sivuston direktiiviaiheisista kysymyksistä on usein löytynyt omaa syntaksilähtöistä ongelmatilannetta läheltä sivuava kysymys, ja sovellettavissa oleva tekninen ratkaisu tähän.

Direktiiveihin liittyy epäsuorasti myös astetta konseptuaalisempi ongelma dokumenttiobjektimallin (DOM) käsittelyn haasteista. Kuten AngularJS:n kehitystyön hyviin käytäntöihin kuuluu, tulee kaikki dokumenttiobjektimallin käsittely tapahtua direktiivien kautta (AngularJS, 2015e). DOMin virheellinen käsittelytapa nousee tuloksissa yhdeksi tyypillisimmistä sovelluskehittäjien kohtaamista ongelmatilanteista. Tämä on toisaalta ymmärrettävää sikäli kun kehittäjällä on taustaa muista JavaScript-sovelluskehyksistä tai -kirjastoista kuten jQuerystä. Oletettavasti vanhasta tottumuksesta

toteutettu nopea ja muissa JavaScript-ympäristöissä validi dokumenttiobjektimallin muutostoimenpide toimii, mutta on samalla ratkaisuna AngularJS:n periaatteiden vastainen. Sovelluksen laajentuessa näiden muutosten aikaansaamat vaikutukset voivat kumuloitua ja muodostua varsinaiseksi ongelmaksi vasta myöhemmin.

Toinen aineistossa toistuvasti esille nouseva ongelmakohta liittyy skooppeihin sekä näiden prototyyppipohjaiseen periytyvyyteen. Skooppien toimintaan liittyvät epäselvyydet voidaan luokitella käsitteellisten ongelmien kategoriaan, sillä taustalla vaikuttavat syyt ovat monitahoisia ja nämä vaativat perusteellista paneutumista paitsi AngularJS:n toimintaperiaatteisiin, mutta lisäksi myös JavaScriptin prototyyppipohjaisen periytyvyyden sääntöihin. Chandermani (2014) tiivistää skooppeihin liittyvien ongelmien pääsääntöiseksi aiheuttajaksi tilanteet, joissa sovelluskehittäjä:

1. Tuottaa AngularJS-sovellukseen sisäkkäisiä ohjainmoduleita
2. Käyttää skooppeja muodostavia direktiivejä kuten ng-repeat, ng-if, ng-include ja ng-controller
3. Tuottaa itse direktiivejä

Kaikissa näissä tilanteissa skooppeihin liittyvien ongelmien keskiössä on AngularJS:n kaksisuuntainen datakytkentä. Luvussa 2.3 kerrottiin kaksisuuntaisen datakytkennän mahdollistavan sen, että näkymässä tehdyt muutokset heijastuvat malliin ja toisinpäin ohjaimen välityksellä. Sen sijaan sisäkkäiset ohjainmodulit ja direktiivit kuten edellämainittu ng-repeat saavat aikaan tilanteen, jossa skoopista muodostuu niin kutsuttuja ”parent-child”-pareja. Parent-skooppi toimii tällöin prototyyppinä uudelle child-skoopille, jolla on vanhempansa ominaisuudet mukaan lukien skoopin nimi (GitHub, 2015). Tästä prototyyppipohjaisesta periytyvyydesta aiheutuvat ongelmat voidaan tiivistää siten, että sovelluskehittäjä yrittää manipuloida parent-skoopin ominaisuuksia väärällä elementtien sisäkkäisyyden tasolla, jolloin toimenpiteiden kohteena onkin child-skooppi. Skooppien prototyyppipohjaiseen periytyvyyteen liittyvä käsitteellinen epäselvyys on varsin ymmärrettävästi useiden kehitystyössä ilmenevien ongelmien lähde. Tämän olen huomannut myös omalla kohdallani AngularJS-kehitystyön yhteydessä.

On huomionarvoinen asia, että ng-repeat-direktiiveihin ja AngularJS:n ohjainmoduleihin liittyvät ongelmatilanteet nousevat aineistossa toistuvasti esille (AngularJS, 2015f; Chandermani, 2014; StackOverflow, 2015). Oletettavasti ongelmien taustalla vaikuttavat syyt liittyvät ainakin osittain edellä käsiteltyyn skooppien prototyyppipohjaiseen periytyvyyteen. Kokoelmien iterointiin tarkoitetun ng-repeat-direktiivin käytöltä on AngularJS-sovellusympäristössä vaikea välttyä, mutta kokematon kehittäjä päätyy hyvin

todennäköisesti yhdistämään samaan elementtiin muita AngularJS:n sisäänrakennettuja direktiivejä, kuten elementin näkyvyyttä säätteleviä ng-show- tai ng-hide-direktiivejä. Tästä seuraa väistämättä toiminnallisia ongelmia. Oikea tapa toimia olisi hajauttaa muut direktiiviit iteroitavan elementin wrapper- tai child-elementteihin. Kokematon AngularJS-kehittäjä saattaa yrittää kiertää tällaisia virhetilanteita siirtämällä AngularJS:n sisäänrakennettujen direktiiveihin nähden päällekkäistä ohjelmakoodia ohjainmoduleihin.

Oman kokemukseni pohjalta AngularJS-kehittäjän paras apu ongelmatilanteita selvitettäessä on ollut selaimen konsoli-ikkunan kautta saatava tieto. Konsoli-ikkuna tarjoaa paitsi yksityiskohtaista tietoa virhetilanteiden aiheuttajista, mutta lisäksi se antaa riittävästi tietoa jonka perusteella ongelmaan voi hakea ratkaisua ulkopuolisista lähteistä tarkennetusti.

6.2 AngularJS ja yksisivuiset web-applikaatiot

Vastattaessa tutkimuskysymykseen: ”Millaisissa tilanteissa on perusteltua suositella AngularJS:n käyttöä yksisivuisen web-applikaation kehitysalustana?”, nousevat kehitystyöstä vastaavan henkilön tai tiimin jo olemassa olevat taidot ja tietämys AngularJS:stä varsin merkittäväksi valintaperusteeksi käytettävälle teknologialle.

Edellisessä kappaleessa käsitellyt asiat huomioon ottaen, AngularJS on kieltämättä haasteellinen kehitysympäristö sekä kokemattomille että kokeneille web-kehittäjille. Aloittelevan web-kehittäjän ensimmäiseksi JavaScript-pohjaiseksi sovelluskehykseksi AngularJS vaatii perusteellista paneutumista paitsi AngularJS:n omintakeisiin toiminnallisiin ratkaisuihin, mutta lisäksi myös JavaScriptin prototyyppipohjaiseen periytyvyyteen. Kokeneemmille web-kehittäjille siirtyminen AngularJS:ään muista JavaScript-pohjaisista ympäristöistä ei sen sijaan ole välttämättä yhtään sen helpompaa. Muissa ympäristöissä vakiintuneet ja hyväksi koetut käytännöt saattavatkin vaikuttaa ylimääräisenä painolastina kehitystyön osalta siirryttäessä AngularJS:ään. Lisäksi AngularJS:n viralliselta sivustolta (https://docs.angularjs.org/guide) löytyvä kehittäjille suunnattu opetusmateriaali ei vakuuta helppotajuisuudellaan.

Tällä ei tietenkään tarkoiteta sitä, etteikö AngularJS:ää voisi hyödyntää web-applikaatioissa ilman aikaisempaa kokemusta sovelluskehyksestä. Kyse on lähinnä siitä, että AngularJS:ään tutustuminen kannattaa aloittaa pienemmistä projekteista. Esimerkiksi kehitystyötä tekevän yrityksen tai organisaation päivitystä kaipaavat omat kotisivut voisivat olla oivallinen valinta ensimmäiseksi AngularJS-projektiksi. Tällainen projekti todennäköisesti mahdollistaisi joustavan aikataulun uuden sovelluskehyksen opetteluun.

Lisäksi kotisivujen päivitysprojekti olisi luultavasti laajuudeltaan sopivaa kokoluokkaa, jotta AngularJS:n eri toiminnallisuuksia päästäisiin kokeilemaan käytännössä. Varsinainen hyöty saavutettaisiin siinä, että tulevien asiakasprojektien toteutuksessa AngularJS olisi varteenotettava valinta, tai ainakin tiedettäisiin millaisia tekijöitä laajemmassa projektissa tulisi huomioida jo suunnitteluvaiheessa.

Yksisivuisen web-applikaation laajuus onkin toinen keskeinen asia, joka tulee huomioida vastattaessa tutkimuskysymykseen. Asiaa voidaan lähestyä siten, että toteutettaessa astetta kompleksisempia yksisivuisia web-applikaatiota, auttaa AngularJS:lle ominainen näkymä-tason laajennettu HTML-syntaksi hahmottamaan sovelluksen rakennetta ja toimintaa paremmin. Yksisivuisen web-applikaation kompleksisuutta voidaankin pitää eräänlaisena ”vedenjakajana” sen suhteen milloin SPA-ratkaisu on vielä mielekästä toteuttaa esimerkiksi jQueryllä, ja vastaavasti milloin AngularJS:n tuomat hyödyt ja etenkin säästetyn työajan rahallinen arvo ylittävät kokonaan itse toteutetun ratkaisun vastaavat.

Tätä kompleksisuuden raja-arvoa on toisaalta hyvin vaikea määritellä tarkasti ja yleispätevästi, mutta dokumenttiobjektimallin kaksisuuntaisten muutostoimenpiteiden hallinta edellä mainitulla jQueryllä johtaa varsin nopeasti vaikeasti hallittavaan ja hahmotettavaan tilanteeseen. Lisäksi kun huomioidaan sovelluskehitysprojektiin sisältyvät muut osa-alueet varsinaisen ohjelmakoodin tuottamisen ohella, kuten testaus ja ohjelmistokoodin ylläpito, nousee AngularJS helposti varteenotettavimmaksi vaihtoehdoksi sisäänrakennettujen toiminnallisuuksiensa puolesta.

6.3 Opinnäytetyöprojekti ja oman oppimisen arviointi

Opinnäytetyöni aiheeksi valikoitunut aihe oli lähellä työnkuvaani kuuluvaa web-kehitystyötä. Tutustumiseni AngularJS:ään ja yksisivuisiin web-applikaatioihin tapahtui alkujaan jo aloitetun projektin kautta, jossa AngularJS-ummikkona tulin jatkamaan kokeneemman kehittäjän työtä, joka selvästi oli itsekin tutustunut AngularJS:ään vasta kyseisen projektin kautta. Eli käytännössä vaikeimman kautta. Opinnäytetyön tekeminen oppimisprosessina oli sen sijaan varsin hyödyllinen kokemus, sillä monet AngularJS:ään liittyneet epäselvät asiat, joita olen työssäni joutunut lähestymään kysymyksellä ”miten?”, jouduin kirjoitustyön yhteydessä käsittelemään näkökulmasta ”miksi?”. Oppimiskokemuksena tämä auttaa kirjoittaa sikäli, että nämä asiat täytyy kyetä avaamaan myös lukijalle, jolla ei ole aihepiiristä välttämättä entuudestaan tietämystä. Omien ajatusten pukeminen sanoiksi onkin oppimista parhaimmillaan.

Projektina tämä opinnäytetyö on kokenut vastatuulta pääsääntöisesti aikataulusyistä. Tekstin tuottaminen sujui yleensä hyvin aina kun tarjoutui tilaisuus kirjoittaa yhtäjaksoisesti useampana päivänä perätysten. Valitettavasti näitä jaksoja osui kohdalle harvakseltaan. Projektin rajauksen kannalta aihepiiri tarjosi kiusauksia lähteä käsittemään asioita laajemmaltikin, mutta jo pelkästään AngularJS:n teknologisista osa-alueista riittäisi käsiteltävää yksisivuisten web-applikaatioitten skoopin tarpeisiin nähden enemmän kuin tarpeeksi. Tutkimuksessa käytetty aineisto oli suppea, mutta ongelmalähtöinen lähestymistapa aiheeseen oli mielestäni varsin kiinnostava, etenkin kun AngularJS:ää pidetään usein yhtenä parhaimmista yksisivuisten web-applikaatioitten kehitysalustoista. Samalla tämän sovelluskehyksen kohdalla on aina se riski, että käyttöönoton esteenä on vaikeaksi koettu opiskeluvaihe. Opinnäytetyötä tehdessäni heräsi mielessäni aihe jatkotutkimukselle, jossa voitaisiin selvittää onko AngularJS-tietotaidon puute nähty ohjelmistoyrityksissä esteenä sovelluskehyksen valinnalle, vaikka se muutoin olisi ollut teknologiavalinnan osalta vahvoilla.

Lähteet

AngularJS.org. 2015a. Downloads. Luettavissa: https://code.angularjs.org/. Luettu: 24.10.2015.

AngularJS.org. 2015b. Creating Custom Directives. Luettavissa: https://docs.angularjs.org/guide/directive. Luettu: 28.9.2015.

AngularJS.org. 2015c. Data Binding. Luettavissa: https://docs.angularjs.org/guide/databinding. Luettu: 28.9.2015.

AngularJS.org. 2015d. Dependency Injection. Luettavissa: https://docs.angularjs.org/guide/di. Luettu: 1.10.2015.

AngularJS.org. 2015e. Conceptual Overview. Luettavissa: https://docs.angularjs.org/guide/concepts. Luettu 3.10.2015.

AngularJS.org. 2015f. Miscellaneous / FAQ. Luettavissa: https://docs.angularjs.org/misc/faq. Luettu: 24.10.2015.

Chandermani, A. 2014. Common Pitfalls using AngularJS. Luettavissa: http://blog.technovert.com/2014/02/common-pitfalls-angularjs/. Luettu 24.10.2015.

Google Trends, 2015. Luettavissa: https://www.google.fi/trends/explore#q=AngularJS%2C%20Backbone.js%2C%20Ember.js%2C%20Knockout.js%2C%20React.js&date=1%2F2010%2071m&cmpt=q&tz=Etc%2FGMT-2. Luettu: 30.11.2015.

Herskovits, I 2014. Five Common Angular Mistakes. Luettavissa: http://blog.backand.com/five-angular-mistakes/. Luettu 22.10.2015.

Horsmalahti, P. 26.5.2014. AngularJS ja datakytkentä. Cybercom Group. Blogit. Luettavissa: http://www.cybercom.com/fi/Suomi/Yritys/Blogit/Blogit/AngularJS-ja-datakytkenta/. Luettu: 28.9.2015.

Osmani, A. 27.7.2012a. Journey Through The JavaScript MVC Jungle. Luettavissa: http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle-2/. Luettu: 5.10.2015.

Osmani, A. 10.4.2012b. Understanding MVVM - A Guide For JavaScript Developers. Luettavissa: http://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/. Luettu 5.10.2015.

O'Reilly, T. 30.9.2005. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. Luettavissa: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html. Luettu: 18.9.2015.

GitHub 2015. Understanding Scopes. Luettavissa: https://github.com/angular/angular.js/wiki/Understanding-Scopes. Luettu: 4.11.2015.

RegisFrey 2010. MVC-Process. Wikimedia Commons. Luettavissa: https://commons.wikimedia.org/wiki/File:MVC-Process.svg. Luettu: 2.10.2015.

Rufus, V. 2014. AngularJS Web Application Development Blueprints. Packt Publishing Ltd.

StackOverflow.com. 2015. Tags. Luettavissa: http://stackoverflow.com/tags. Luettu: 24.10.2015.

W3Schools.com. 2015a. AngularJS Extends HTML. Luettavissa: http://www.w3schools.com/angular/angular_intro.asp. Luettu: 15.10.2015.

W3Schools.com. 2015b. AngularJS Directives. Luettavissa: http://www.w3schools.com/angular/angular_directives.asp. Luettu: 24.10.2015.

W3Schools.com. 2015c. AngularJS Expressions. Luettavissa: http://www.w3schools.com/angular/angular_expressions.asp. Luettu 30.9.2015.

W3Schools.com. 2015d. AngularJS Controllers. Luettavissa: http://www.w3schools.com/angular/angular_controllers.asp. Luettu: 1.10.2015.