

Juha Suomijoki

AngularJS

Yksisivuisen web-sovelluksen käyttöliittymän toteutus
AngularJS:llä

Metropolia Ammattikorkeakoulu

Medianomi

Viestinnän koulutusohjelma

Opinnäytetyö

20.4.2015

Tekijä(t) Otsikko	Juha Suomijoki AngularJS
Sivumäärä Aika	51 sivua 20.4.2015
Tutkinto	Medianomi
Koulutusohjelma	Viestinnän koulutusohjelma
Suuntautumisvaihtoehto	Digitaalinen viestintä
Ohjaaja(t)	Digitaalisen viestinnän lehtori Markus Norrena
<p>Opinnäytetyössä tutkittiin mikä on AngularJS-JavaScript-ohjelmistokehys ja miten se soveltuu yksisivuisten web-sovelluksen käyttöliittymän toteutukseen.</p> <p>AngularJS on vuonna 2012 julkaistu Googlen ylläpitämä JavaScript-ohjelmistokehys, joka on tarkoitettu ensisijaisesti yksisivuisten web-sovellusten kehittämiseen.</p> <p>Opinnäytetyön teoriaosuudessa tutkittiin mikä AngularJS on ja mitkä ovat AngularJS:n keskeiset konseptit ja sovelluskomponentit. Tarkastelu pohjautui AngularJS:stä kirjoitettuun kirjallisuuteen, verkkoartikkeleihin ja AngularJS:n virallisen dokumentaatioon.</p> <p>Opinnäytetyön toiminnallisessa osassa toteutettiin AngularJS:llä yksisivuisten web-sovelluksen käyttöliittymä. Web-sovelluksen idea oli koota yhteen tietoa Helsingin seudulla järjestettävistä tapahtumista.</p> <p>Toiminnallisen osan perusteella tehtiin havaintoja ja päätelmiä siitä miten AngularJS yksisivuisten web-sovelluksen käyttöliittymän toteutukseen soveltuu. Kaiken kaikkiaan AngularJS soveltuu tarkoitettuun tehtävään hyvin. Muun muassa AngularJS:n MVC-arkkitehtuuri, reititys ja aktiivisen kehittäjäyhteisön tuki arvioitiin asioiksi, jotka tekivät AngularJS:stä hyvin soveltuvan ohjelmistokehysten yksisivuisten web-sovelluksen käyttöliittymän toteutukseen.</p>	
Avainsanat	AngularJS, JavaScript-ohjelmistokehys, yksisivuinen web-sovellus

Author(s) Title	Juha Suomijoki AngularJS
Number of Pages Date	51 pages 20 April 2015
Degree	Bachelor of Arts and Culture
Degree Programme	Media
Specialisation option	Digital Media
Instructor(s)	Markus Norrena, Senior Lecturer in Digital Media
<p>This thesis examined the parameters of the AngularJS JavaScript framework and how it applies to developing a user interface of a single-page web application.</p> <p>AngularJS is a JavaScript framework that is primarily intended for developing single-page web applications. AngularJS was published in 2012 and is maintained by Google.</p> <p>The theoretical part of the thesis examined what AngularJS is and what are AngularJS' key concepts and application components. The examination was based on literature, online articles and the official documentation of AngularJS.</p> <p>In the operational part of the thesis, the author created a user interface of a single-page web application. The idea of the web application was to bring together information about events in the Helsinki metropolitan region.</p> <p>Based on the operational part of the thesis, observations and conclusions were made about how AngularJS is suited for developing a user interface of a single-page web application. All in all, AngularJS suited well for this purpose. For instance, AngularJS' MVC architecture, routing system and the support of an active developer community were considered as matters that made AngularJS well-suited for developing a user interface of a single-page web application.</p>	
Keywords	AngularJS, JavaScript framework, single-page web application

Sisällys

1	Johdanto	1
2	AngularJS	2
2.1	AngularJS:n taustaa	3
2.2	AngularJS:n käyttöönotto	4
2.3	Konseptit	5
2.3.1	MVC-arkkitehtuuri	5
2.3.2	Kaksisuuntainen datakytkentä	6
2.3.3	Riippuvuusinjektio	8
2.3.4	Direktiivi	9
2.4	Sovelluskomponentit	10
2.4.1	Moduuli	10
2.4.2	Scope	11
2.4.3	Lauseke	13
2.4.4	Kontrolleri	14
2.4.5	Suodatin	15
2.4.6	Palvelut	17
2.4.7	Reititys	19
3	Yksisivuisen web-sovelluksen käyttöliittymän toteutus AngularJS:llä	21
3.1	Hakemistorakenne	23
3.2	Esityöt	24
3.3	Sovelluksen säilöminen moduuliin	26
3.4	Reititys	26
3.5	Palvelinkommunikointi	28
3.6	Sovelluslogiikka	30
3.6.1	Tapahtumien listaus	33
3.6.2	Yksittäisen tapahtuman esittäminen	36
3.6.3	Päivämäärän valinta	38
3.6.4	Tapahtumakategorian valinta	40
3.6.5	Sivutus	41
4	Yhteenveto	43
	Lähteet	49

1 Johdanto

Opinnäytetyön aiheena on AngularJS-JavaScript-ohjelmistokehys. Opinnäytetyössä tutkitaan mikä AngularJS on ja miten se soveltuu yksisivuisen web-sovelluksen käyttöliittymän kehitykseen.

AngularJS on Googlen ylläpitämä avoimen lähdekoodin JavaScript-ohjelmistokehys, joka on suunniteltu pääasiassa yksisivuisten web-sovellusten kehittämiseen (Lerner 2013, 8).

Yksisivuinen web-sovellus on sovellusmalli, jossa tarvittava sovelluskoodi ladataan sivulle yhdellä sivulatauksella, minkä jälkeen sisältöä muutetaan dynaamisesti JavaScriptillä (Markov 2015).

Opinnäytetyö jakautuu teoreettiseen ja toiminnalliseen osaan.

Opinnäytetyön teoreettisessa osassa tutkitaan mikä AngularJS on ja mitkä ovat AngularJS:n keskeiset konseptit ja sovelluskomponentit. Teoreettinen osa pohjautuu AngularJS:stä kirjoitettuun kirjallisuuteen, verkkoartikkeleihin sekä AngularJS:n viralliseen dokumentaation ja kehittäjille suunnattuihin oppaisiin.

Opinnäytetyön toiminnallisessa osassa toteutetaan AngularJS:llä yksisivuisen web-sovelluksen käyttöliittymä. Toiminnallisen osan tarkoituksena on tarkastella miten yksisivuisen web-sovelluksen käyttöliittymän toteutus AngularJS:llä käytännössä tapahtuu.

Opinnäytetyön yhteenvedossa esitetään päätelmiä ja havaintoja siitä miten AngularJS yksisivuisen web-sovelluksen käyttöliittymän kehitykseen soveltui. Lisäksi yhteenvedossa kerrataan teoreettisessa osassa saavutettuja tuloksia.

Opinnäytetyö on luonteeltaan kvalitatiivinen eli laadullinen. Laadullisen tutkimusmenetelmän valinnan tarkoituksena on mahdollistaa AngularJS:n kokonaisvaltainen ymmärtäminen. Opinnäytetyössä AngularJS:ää pyritään lähestymään avoimesti laadukkaan tutkimusaineiston pohjalta. Tutkimusaineistoa on rajattu harkinnanvaraisesti mahdollisimman laadukkaan aineiston varmistamiseksi. Tästä huolimatta on pyritty varmistamaan riittävän kattava tutkimusaineisto, jotta opinnäytetyössä esitetyt havainnot olisivat

perusteltuja. Tutkimusaineistosta on pyritty erittelemään piirteitä, jotka ovat AngularJS:lle ominaisia ja ilmiötä kuvaavia.

Opinnäytetyön tavoite on toimia johdatuksena AngularJS:ään ja tarjota perusvalmiudet AngularJS-sovelluksen kehittämiseen. Opinnäytetyön kohdeyleisö on web-kehitystä työkseen tekevät tai opiskelevat henkilöt ja muut AngularJS:stä kiinnostuneet. Opinnäytetyön ymmärtämiseksi lukijan tulee omata hyvät perustiedot HTML:stä, CSS:stä ja JavaScriptista.

Luvussa 2 tutkitaan mikä AngularJS on. Luvussa eritellään tutkimusaineiston pohjalta AngularJS:n keskeisiä konsepteja ja sovelluskomponentteja. Konseptien ja sovelluskomponenttien käyttöä havainnollistetaan koodiesimerkkien avulla.

Luvussa 3 kuvaillaan yksisivuisen Tapahtumia Helsingissä -web-sovelluksen käyttöliittymän toteutusprosessi AngularJS:llä. Luvun tarkoitus on havainnollistaa miten yksisivuisen web-sovelluksen käyttöliittymä AngularJS:llä käytännössä toteutetaan. Tämän tiedon pohjalta AngularJS:n sovelutuvuutta voidaan arvioida.

Luvussa 4 esitetään päätelmiä ja havaintoja siitä miten AngularJS yksisivuisen web-sovelluksen käyttöliittymän toteutusprosessiin soveltui. Lisäksi yhteenvedossa käydään läpi opinnäytetyön teoreettisen osan tuloksia.

2 AngularJS

Tässä luvussa tutkitaan mikä AngularJS on. Aluksi tarkastellaan AngularJS:n taustaa ja käydään läpi kuinka AngularJS otetaan käyttöön. Tämän jälkeen tutkitaan mitkä ovat AngularJS:n keskeiset konseptit ja sovelluskomponentit. Keskeisten konseptien ja sovelluskomponenttien rajausta perustuu tutkimusaineistosta esiin nousseisiin teemoihin AngularJS:ää määrittävistä ominaisuuksista.

Käsiteltävien aiheiden tarkoitus on johdattaa lukija AngularJS:n perusteisiin, kartoittaa AngularJS:n taustalla vaikuttavia konsepteja sekä tarkastella niitä sovelluskomponentteja, joiden toiminta on oleellista AngularJS:n ymmärtämiseksi. AngularJS:n konseptien ja sovelluskomponenttien toimintaa havainnollistetaan lukuisten koodiesimerkkien avulla. Esimerkeissä käytetään AngularJS:n kehittäjille suunnatun virallisen oppaan suosituksia.

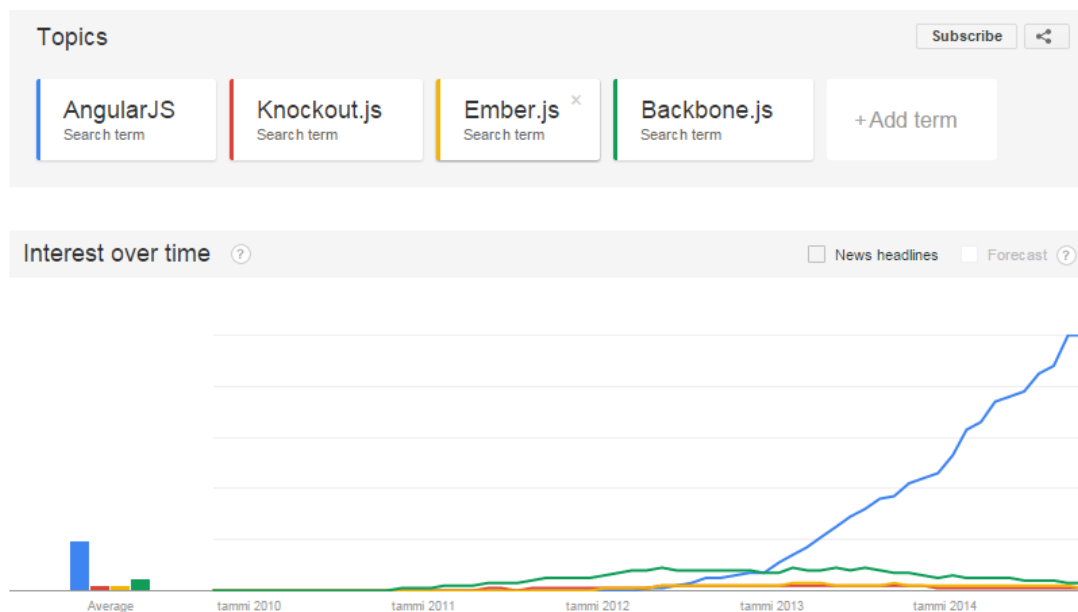
telemaa merkintätapaa, aina kun se on esimerkin kannalta luontevaa (Vrt. Developer Guide: Dependency Injection).

2.1 AngularJS:n taustaa

AngularJS on Googlen ylläpitämä avoimen lähdekoodin JavaScript-ohjelmistokehys, joka on suunniteltu pääasiassa yksisivuisten web-sovellusten (engl. single-page application) kehittämiseen. AngularJS on kirjoitettu kokonaan JavaScriptillä. (Lerner 2013, 8.) AngularJS:n lähdekoodi on avoimesti saatavilla GitHubissa MIT-lisenssillä (GitHub 2015a).

AngularJS-ohjelmistokehityksen kehityksen aloitti Googlen työntekijä Miško Hevery yhdessä ystävänsä Adam Abronsin kanssa vuonna 2009. Hevery esitteli AngularJS:n myöhemmin työnantajalleen Googlelle, joka kiinnostui projektista ja otti ohjelmistokehityksen suojiinsa. (Krill 2013.) Ensimmäinen vakaa versio AngularJS 1.0 julkaistiin GitHubissa kesäkuussa 2012 (GitHub 2015b). AngularJS:n seuraava merkittävä versio-uudistus 2.0 on kehitteillä, mutta uuden version julkaisuajankohta ei toistaiseksi ole tiedossa (Esteva 2013). AngularJS on nimetty HTML:ssä käytettävien kulmasulkeiden (engl. angle brackets) mukaan (Krill 2013).

AngularJS:n suosio on ollut ohjelmistokehityksen julkaisusta lähtien voimakkaassa kasvussa (Google Trends 2015). Kuvio 1 havainnollistaa AngularJS:n ja sen kolmen kilpailijan – Knockout.js, Ember.js, Backbone.js – suosion kehitystä Google-hauilla mitattuna vuosina 2010–2014.



Kuvio 1. AngularJS:n, Knockout.js:n, Ember.js:n ja Backbone.js:n suosion kehitys Google-hauilla mitattuna vuosina 2010–2014.

AngularJS sisältää monia nykyaikaiseen web-kehitykseen vakiintuneita ulottuvuuksia kuten sovelluksen jakaminen osiin, AJAX-palvelut, selainhistorian hallinta, riippuvuusinjektio ja testaus (Lerner 2013, 8). AngularJS ei kuitenkaan sisällä varsinaisesti uusia ideoita, vaan ohjelmistokehykseen on sisällytetty toimivaksi havaittuja tapoja muista kehitysympäristöistä (Green & Seshadiri 2013, 1).

AngularJS:ssä omaleimaista on sen tapa sallia HTML:n syntaksin laajentaminen uusilla merkitsijöillä, joita AngularJS:ssä kutsutaan direktiiveiksi. Direktiivien ohella AngularJS:n kaksisuuntainen datakytkentä ja riippuvuusinjektio ovat ohjelmistokehyksen keskeisiä konsepteja, jotka helpottavat kehittäjän työtä vähentämällä kirjoitettavan koodin määrää. AngularJS on toteutettu MVC-arkkitehtuurin periaatteiden mukaisesti. (Bogucki 2014)

2.2 AngularJS:n käyttöönotto

AngularJS:n käyttöönotto eli toimivan AngularJS-sovelluksen luominen edellyttää AngularJS:n lähdekoodin lataamista sivulle sekä ng-app -direktiivin kiinnittämistä DOMiin. AngularJS:n lähdekoodi ladataan sivulle <script>-elementillä. (Diez 2013.) Ng-app-

direktiivi kertoo AngularJS:lle mikä osa DOMista on AngularJS:n käytössä. (Green & Seshadiri 2013, 11.)

Kuvio 2 havainnollistaa AngularJS:n käyttöönottoa. Esimerkissä AngularJS:n lähdekoodi lisätään aluksi `<script>`-elementillä sivulle. Tämän jälkeen `<html>`-elementtiin merkityllä `ng-app`-direktiivillä kerrotaan AngularJS:lle mikä osa DOMista on AngularJS:n hallinnassa. Tässä esimerkissä koko HTML-koodi on AngularJS:n hallinnassa.

```

1  <!DOCTYPE html>
2  <html ng-app>
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body>
8
9  </body>
10 </html>

```

Kuvio 2. AngularJS:n käyttöönotto.

2.3 Konseptit

Tässä luvussa tutkitaan mitkä ovat AngularJS-ohjelmistokehystä määrittävät keskeiset konseptit. Tarkastelun kohteeksi rajataan MVC-arkkitehtuuri, kaksisuuntainen datakytkentä, riippuvuusinjektio sekä HTML:n syntaksia laajentavat direktiivit.

Rajaus perustuu lähdekirjallisuuden, verkkoartikkeleiden ja AngularJS:n virallisen dokumentaation pohjalta tehtyyn arvioon AngularJS:n keskeisistä konsepteista. Tutkimusaineistossa esimerkiksi Brad Green ja Shyam Seshadri, Dmitri Lau sekä Sandeep Panda nostavat mainitut konseptit esille (Ks. Green & Seshadri 2013, 3–6; Seshadri & Green 2014, 4–9; Lau 2013; Panda 2014, 2–4).

2.3.1 MVC-arkkitehtuuri

MVC-arkkitehtuuri (Model-View-Controller eli malli-näkymä-kontrolleri) on ohjelmistosuunnittelussa vaikuttava ohjelmistoarkkitehtuuriperiaate, jonka perusajatus on jakaa sovellus toisistaan selkeästi erottuviin osiin. MVC-arkkitehtuurissa toisistaan erotetaan sovelluksen tietorakenne, käyttöliittymä ja sovelluslogiikka – eli malli, näkymä ja kontrolleri. (vrt. Janssen 2015b.)

AngularJS on toteutettu MVC-arkkitehtuurin periaatteiden mukaisesti. AngularJS:n MVC-arkkitehtuurissa näkymää edustaa DOM, kontrollerit ovat JavaScript-funktioita ja malli on data, joka säilötään tavallisen JavaScript-objektin ominaisuuksiin. (Green & Seshadiri 2013, 3.)

MVC-arkkitehtuuri AngularJS:ssä antaa kehittäjälle selkeän käsityksen sovelluksen rakenteesta ja helpottaa koodin organisoimista, ylläpitoa ja testausta (Green & Seshadiri 2013, 3).

AngularJS:n ja MVC-arkkitehtuurin suhde ei kuitenkaan ole aivan yksioikoinen, sillä esimerkiksi Sandeep Panda mainitsee, että AngularJS:llä voi MVC-arkkitehtuurin lisäksi rakentaa myös niin sanotun MVVM-arkkitehtuurin (Model-View-ViewModel) mukaisia sovelluksia (Panda 2014, 13). MVVM-arkkitehtuurissa tavoitteena on erottaa sovelluksen käyttöliittymä ja sovelluslogiikka entistä selkeämmin toisistaan (Osmani 2012). Virallisesti AngularJS onkin nimetty MVW-ohjelmistokehykseksi, jossa sana MVW muodostuu kirjaimista Model-View-Whatever (Panda 2014, 13).

2.3.2 Kaksisuuntainen datakytkentä

Datakytkennällä (engl. data binding) tarkoitetaan web-kehityksessä prosessia, jossa muodostetaan yhteys web-sovelluksen käyttöliittymän ja tietorakenteen välille (Janssen 2015a).

Perinteisissä web-sovelluksissa käytetään tyypillisesti yksisuuntaista datakytkentää (Branas 2014, 103). Tällaisissa perinteisissä web-sovelluksissa käyttöliittymä valmistellaan palvelimella yhdistämällä sovelluksen HTML-sisältö sovellusdataan, minkä jälkeen valmis käyttöliittymä lähetetään asiakasohjelmalle. Aina kun osaa käyttöliittymästä halutaan muuttaa, palvelin lähettää kokonaan uuden HTML:n ja datan asiakasohjelmalle. (Seshadiri & Green 2014, 4.)

AngularJS toimii tässä suhteessa toisin, koska AngularJS muokkaa sovelluksen näkymää reaaliaikaisesti (Lerner 2013, 11). AngularJS:ssä datakytkentä on automaattista ja kaksisuuntaista (engl. two-way data binding). Kaksisuuntaisessa datakytkennässä sovelluksen näkymässä tapahtuvat muutokset heijastuvat automaattisesti malliin – ja vastaavasti muutokset mallissa levittyvät automaattisesti näkymään. (Developer Guide: Data Binding.)

AngularJS toteuttaa kaksisuuntaisen datakytkennän niin sanotulla dirty checking -tekniikalla, jossa potentiaalisen muutoksen ilmetessä AngularJS käy läpi web-sovelluksen koko mallin, ja katsoo onko muutoksia tapahtunut. (Lerner 2013, 12) Dirty checking -tekniikan etu on, että AngularJS-sovelluksen kehittäjä ei joudu kirjoittamaan ylimääräistä koodia datakytkennän toteuttamiseksi. Ongelmia saattaa kuitenkin ilmetä tehokkuudessa, jos läpikäytävä malli sisältää runsaasti dataa (Horsmalahti 2014).

Kuviot 3 ja 4 havainnollistavat AngularJS:n datakytkentää käytännössä. Kuviossa 3 AngularJS-sovellus käynnistetään lataamalla AngularJS:n lähdekoodi sivulle ja kertomalla ng-app-direktiivillä, mikä osa DOMista on AngularJS:n hallinnassa. <body>-elementin sisällä oleva <input>-tekstikenttä kytketään sovelluksen malliin ng-model-direktiivillä. Ng-model-direktiiville annetaan arvo text, jonka perusteella AngularJS muodostaa vastaavan nimisen ominaisuuden sovelluksen malliin. Mallin sisältämä ominaisuus voidaan tämän jälkeen tulostaa näytölle lausekkeella {{text}}.

Kuvio 4 esittää kuvion 3 ohjelman selaimessa. Kun käyttäjä kirjoittaa selaimessa tekstikenttään, sovelluksen mallissa oleva data päivittyy automaattisesti ja muutokset heijastuvat käyttäjälle näkymään. Huomionarvoista on, että sovelluksen kehittäjän ei tarvitse kirjoittaa riviäkään koodia lopputuloksen saavuttamiseksi, koska AngularJS:n kaksisuuntainen datakytkentä huolehtii automaattisesti datan synkronoinnista mallin ja näkymän välillä.

```

1  <!DOCTYPE html>
2  <html ng-app>
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body>
8
9    <input ng-model="text" />
10   <h1>{{text}}</h1>
11
12 </body>
13 </html>

```

Kuvio 3. Kaksisuuntainen datakytkentä AngularJS:ssä.

Hello world

Hello world

Kuvio 4. Kaksisuuntainen datakytkentä AngularJS:ssä. Kuvion 3 ohjelma selaimessa.

2.3.3 Riippuvuusinjektio

Riippuvuusinjektioilla (engl. dependency injection) tarkoitetaan web-kehityksessä prosessia, jossa määritellään kuinka sovelluksen komponentit pääsevät käsiksi riippuvuuksiinsa (Developer Guide: Dependency Injection). Riippuvuudella tarkoitetaan sovelluskomponenttia, jonka toiminnallisuutta jokin toinen sovelluskomponentti tarvitsee toimiakseen.

AngularJS:ssä riippuvuuksia ei tarvitse erikseen määrittää kirjoittamalla ylimääräistä koodia, sillä sovelluskomponentti voi yksinkertaisesti ”pyytää” tarvitsemaansa riippuvuutta. AngularJS huolehtii automaattisesti siitä, että sovelluskomponenttiin injektoidaan sen tarvitsema riippuvuus, jonka AngularJS on etukäteen etsinyt ja kertonut riippuvuudelle tämän kohteen. (Lerner 2013, 149)

Kuvio 5 havainnollistaa riippuvuusinjektion toimintaa AngularJS:ssä. Esimerkissä AngularJS-sovellus säilötään aluksi moduuliin (Ks. luku 2.4.1), johon lisätään angular.module API:n controller()-metodilla myController-niminen kontrolleri (Ks. luku 2.4.4). Esimerkissä oleellista on havaita kuinka kontrollerille syötetään kaksi riippuvuutta: \$scope ja \$location. Riippuvuuksien toiminta ei esimerkissä ole oleellista. Tärkeintä on huomioida kuinka sovelluskomponentti voi AngularJS:ssä yksinkertaisesti ”pyytää” tarvitsemansa riippuvuudet, minkä jälkeen AngularJS huolehtii automaattisesti siitä, että riippuvuudet ovat sovelluskomponentin käytettävissä.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9  <script>
10   angular.module('myApp', [])
11     .controller('myController', ['$scope', '$location',
12       function($scope, $location) {
13         // ...
14       }
15     ]);
16 </script>
17
18 </body>
19 </html>

```

Kuvio 5. Riippuvuusinjektio AngularJS:ssä.

2.3.4 Direktiivi

AngularJS:ssä direktiivit (engl. directive) ovat HTML:n syntaksia laajentavia merkitsijöitä, joiden avulla DOMiin lisätään haluttua toiminnallisuutta (Developer Guide: Directives).

AngularJS sisältää lukuisia sisäänrakennettuja direktiivejä, jotka erotetaan muista HTML-attribuuteista etuliitteellä ng-. HTML-validaattoreita varten voidaan käyttää myös muotoa data-ng- (Green & Seshadiri 2013, 119–120). AngularJS:n sisäänrakennettuja direktiivejä ovat esimerkiksi:

- ng-app alustaa AngularJS-sovelluksen
- ng-model kytkee HTML:n lomakekentän (input, select, textarea) arvon sovellusdataan
- ng-init asettaa lähtöarvoja sovelluksen malliin
- ng-repeat iteroi annetun kokoelman läpi
- ng-click määrittää toiminnallisuuden elementin klikkaukselle

Kattava dokumentaatio AngularJS:n sisäänrakennetuista direktiiveistä löytyy AngularJS:n virallisesta dokumentaatiosta (Ks. API: ng).

Kuvio 6 havainnollistaa kolmen direktiivin – ng-app, ng-model ja ng-init – toimintaa AngularJS:ssä. Ng-app-direktiivi alustaa AngularJS-sovelluksen ja kertoo mikä osa DOMista on AngularJS:n hallinnassa. Ng-model-direktiivi puolestaan kytkee <input>-elementin arvon name-muuttujaan, jolle ng-init-direktiivi asettaa lähtöarvon 'John Doe'.

```

1  <!DOCTYPE html>
2  <html ng-app>
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-init="name='John Doe'">
8
9    <input type="text" ng-model="name" placeholder="Kirjoita nimi..." />
10   <p>{{name}}</p>
11
12 </body>
13 </html>

```

Kuvio 6. Direktiivit AngularJS:ssä.

Sisäänrakennettujen direktiivien lisäksi AngularJS:ssä on usein hyödyllistä luoda omia direktiivejä. Omien direktiivien avulla web-sovelluksen käyttöliittymään voidaan lisätä

räätälöityä toiminnallisuutta ja kapseloida koodia helposti uudelleen käytettävään muotoon. Tyypillisesti räätälöityjä direktiivejä käytetään esimerkiksi DOM-manipulaatioihin tai tapahtumien kuunteluun (esim. hiiren klikkaus). (Developer Guide: Directives) HTML:ään merkitty räätälöity direktiivi voi olla attribuutti, elementin nimi, kommentti tai CSS-luokka. Lisätietoa räätälöidyn direktiivin luomisesta löytyy esimerkiksi AngularJS:n virallisesta dokumentaatiosta (Ks. Developer Guide: Directives).

2.4 Sovelluskomponentit

Tässä luvussa tutkitaan mitkä ovat AngularJS:n keskeiset sovelluskomponentit. Sovelluskomponenteista tarkastelun kohteeksi rajataan moduuli, scope, lauseke, kontrolleri, suodatin, palvelu ja reititys.

Rajaus perustuu lähdekirjallisuuden, verkkoartikkeleiden ja AngularJS:n virallisen dokumentaation pohjalta tehtyyn arvioon AngularJS:n keskeisistä sovelluskomponenteista. Tutkimusaineistossa esimerkiksi Ari Lerner, Rodrigo Branas, Sandeep Panda, Adam Freeman sekä Shyam Seshadri ja Brad Green nostavat mainitut sovelluskomponentit esille (Ks. Lerner 2013; Branas 2014; Panda 2014; Seshadri & Green 2014; Green & Seshadri 2013).

2.4.1 Moduuli

AngularJS:ssä moduuli (engl. module) on paikka, johon AngularJS-sovellus ja sen osat voidaan tallettaa (Lerner 2013, 18). Moduulin tarkoitus on koota sovelluksen eri komponentteja yhteen nimettyyn paikkaan (Panda 2014, 23). Moduuli voi sisältää esimerkiksi kontrollereja, palveluja ja direktiivejä. Moduuli voi myös olla riippuvainen toisista moduuleista – tällöin moduulin tarvitsemat riippuvuudet määritellään moduulia luotaessa. (Seshadri & Green 2014, 15.)

Moduulien avulla AngularJS-sovelluksen osia voidaan kapseloida tehokkaasti, pitää sovelluksen globaali nimiavaruus puhtaana ja välttää ristiriitoja muiden kirjastojen kanssa (Lerner 2013, 18). Moduulit helpottavat koodin uudelleen käyttämistä, mahdollistavat sovelluksen eri osien suorittamisen halutussa järjestyksessä, helpottavat testausta ja tekevät koodista ymmärrettävämpää (Developer Guide: Modules).

AngularJS:ssä uusi moduuli luodaan `angular.module()`-metodilla, jolle annetaan kaksi parametria. Ensimmäinen parametri on moduulin nimi. Toinen parametri on taulukko, jossa määritellään moduulin riippuvuudet. Jos moduulilla ei ole riippuvuuksia, toinen parametri on tyhjä taulukko. (Panda 2014, 24.) Esimerkki moduulin luomisesta:

```
angular.module('myModule', []);
```

Olemassa olevaan moduuliin voidaan viitata antamalla `angular.module()`-metodille vain yksi parametri: moduulin nimi (Lerner 2013, 19). Esimerkki moduuliin viittaamisesta:

```
angular.module('myModule');
```

Kuvio 7 havainnollistaa moduulin luomista AngularJS:ssä. Esimerkissä AngularJS-sovellus säilötään moduuliin. Uusi moduuli luodaan `angular.module()`-metodilla, jolle annetaan kaksi parametria: moduulin nimi ja lista moduulin tarvitsemista riippuvuuksista. Esimerkissä moduulilla ei ole riippuvuuksia, jolloin toinen parametri on tyhjä taulukko. Huomioitavaa esimerkissä on se, että moduulille annetun nimen tulee vastata `ng-app`-direktiivissä määriteltyä nimeä. Moduulin luomisen jälkeen moduuliin voidaan liittää haluttuja sovelluskomponentteja. Esimerkissä moduuliin lisätään `angular.module` API:n `controller()`-metodilla yksi kontrolleri.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9  <script>
10   angular.module('myApp', [])
11     .controller('myController', ['$scope',
12       function($scope) {
13         // ...
14       }
15     ]);
16 </script>
17
18 </body>
19 </html>

```

Kuvio 7. AngularJS-sovelluksen säilöminen moduuliin.

2.4.2 Scope

AngularJS:ssä scope on tavallinen JavaScript-objekti, johon voidaan kiinnittää ominaisuuksia ja funktiota (Panda 2014, 70). Scopen tehtävä on toimia liimana AngularJS-

sovelluksen näkymän ja kontrollerin välissä (Branas 2014, 103). Kontrollerissa voidaan esimerkiksi lisätä scopeen dataa ja toiminnallisuutta, jotka vaikuttavat tiettyyn näkymään (Kozłowski & Darwin 2013,14). AngularJS:ssä sekä kontrollerilla että näkymällä on pääsy scopeen, mutta ei toisiinsa (Developer Guide: Scopes).

AngularJS:n MVC-arkkitehtuurissa scope muodostaa viittauksen sovelluksen malliin. Scope määrittää myös näkyvyysalueen, jossa AngularJS:n lausekkeet evaluoidaan. (Lerner 2013, 20.) Esimerkiksi AngularJS:n lauseke `{{name}}` ei tarkoita mitään, ellei sitä evaluoida vasten sitä scope-objektia, jossa ominaisuus `name` on määritetty (Developer Guide: Scopes).

AngularJS:ssä scope-objektia ei yleensä tarvitse erikseen luoda, koska AngularJS luo scope-objektin automaattisesti (Panda 2014, 43). Jokaisella AngularJS-sovelluksella on vähintään yksi scope-objekti, `$rootScope`, jonka AngularJS luo automaattisesti, kun sivuun kiinnitetään `ng-app`-direktiivi (Panda 2014, 46). `$rootScope`-objekti on AngularJS:ssä kaikkein lähimpänä sovelluksen globaalia kontekstia (Lerner 2013, 21). `$rootScope`-objektilla voi olla useita perillisiä, jotka järjestyvät hierarkkisesti. Esimerkiksi `ng-controller`-direktiivi luo sovellukseen automaattisesti uuden `$scope`-objektin, joka on `$rootScope`-objektin jälkeläinen. (Panda 2014, 46.)

Kuvio 8 havainnollistaa scopen toimintaa AngularJS:ssä. Esimerkissä AngularJS-sovellus säilötään moduuliin, johon lisätään `angular.module` API:n `controller()`-metodilla kontrolleri. Uusi kontrolleri muodostaa sovellukseen automaattisesti uuden `$scope`-objektin, joka nimetään `controller()`-metodin riippuvuudeksi. Kontrolleri kiinnitetään sovelluksen näkymään `ng-controller`-direktiivillä, joka määrittää DOMiin `$scope`-objektin näkyvyysalueen. Tämän jälkeen kontrollerissa voidaan asettaa `$scope`-objektille ominaisuuksia ja funktioita. Esimerkissä `$scope`-objektille luodaan ominaisuus `name`. Sovelluksen näkymässä `$scope.name`-ominaisuuden sisältämä arvo tulostetaan näytölle lausekkeella `{{name}}`.


```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9    <h1>Hello {{name}}</h1>
10
11  <script>
12    angular.module('myApp', [])
13      .controller('myController', ['$scope',
14        function($scope) {
15          $scope.name = 'John Doe';
16        }
17      ]);
18  </script>
19
20 </body>
21 </html>

```

Kuvio 8. Scopen toiminta AngularJS:ssä.

2.4.3 Lauseke

AngularJS:ssä lauseke (engl. expression) on kaksilla aaltosulkeilla merkitty kappale koodia, joka palauttaa jonkin arvon (Seshadiri & Green 2014, 53). Lauseke voi sisältää esimerkiksi muuttujan, merkkijonon tai laskutoimituksen (W3 Schools 2015). Lausekkeen tarkoitus on esittää mallin sisältämä data sovelluksen näkymässä (Panda 2014, 13). AngularJS:ssä lausekkeita ovat esimerkiksi:

```

{{ expression }}
{{ 'Hello world' }}
{{ 3 + 2 }}

```

AngularJS:ssä lausekkeet toimivat suunnilleen samoin kuin JavaScriptissa. (Lerner 2013, 31) Erona on, että JavaScriptissa lausekkeet evaluoidaan globaalissa kontekstissa, mutta AngularJS:ssä evaluointi tapahtuu aina scopea vasten. Lisäksi AngularJS:ssä lausekkeita voidaan muotoilla suodattimilla toisin kuin JavaScriptissa. (Developer Guide: Expressions)

Kuvio 9 havainnollistaa lausekkeiden käyttöä AngularJS:ssä. Sovelluksen näkymässä on kaksi lauseketta: `{{name}}` ja `{{25+25}}`. Sovelluksen kontrollerissa ensin mainitulle lausekkeelle asetetaan lähtöarvo. Lähtöarvo asetetaan syöttämällä kontrollerin riippuvuudeksi sovelluksen mallin sisältävä `$scope`-objekti, jolle määritetään kontrollerissa lauseketta vastaava ominaisuus `$scope.name`. Lauseke `{{25+25}}` evaluoidaan automaattisesti AngularJS:n toimesta.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9    <h1>{{name}} is {{25 + 25}} years old.</h1>
10
11  <script>
12    angular.module('myApp', [])
13      .controller('myController', ['$scope',
14        function($scope) {
15          $scope.name = 'John Doe';
16        }
17      ]);
18  </script>
19
20 </body>
21 </html>

```

Kuvio 9. Lausekkeiden toiminta AngularJS:ssä.

2.4.4 Kontrolleri

AngularJS:ssä kontrolleri (engl. controller) on funktio, jossa määritellään sovelluksen liiketoimintalogiikka (Lerner 2013, 25; Freeman 2014, 22). Kontrollerilla on AngularJS:ssä kaksi tehtävää: asettaa lähtöarvoja ja lisätä toiminnallisuutta \$scope-objektiin. (Lerner 2013, 25)

AngularJS:ssä uusi kontrolleri luodaan angular.module API:n controller()-metodilla. Kontrolleri kiinnitetään DOMiin ng-controller-direktiivillä. Uusi kontrolleri muodostaa sovellukseen automaattisesti uuden \$scope-objektin, joka voidaan syöttää kontrollerin riippuvuudeksi. Kontrollerissa \$scope-objektiin voidaan asettaa lähtöarvoja ja luoda toiminnallisuutta. (Developer Guide: Controllers)

Kuvio 10 havainnollistaa kontrollerin toimintaa AngularJS:ssä. Esimerkissä AngularJS-sovellus säilötään moduuliin, johon lisätään angular.module API:n controller()-metodilla uusi kontrolleri. Sovelluksen näkymässä kontrolleri kiinnitetään <body>-elementtiin ng-controller-direktiivillä. Uusi kontrolleri muodostaa sovellukseen automaattisesti uuden \$scope-objektin, joka syötetään riippuvuusinjektion kautta kontrollerille. Kontrollerissa \$scope-objektille määritetään ominaisuus \$scope.name, joka kytketään {{name}}-lausekkeella sovelluksen näkymään. Sovellukseen luodaan toiminnallisuutta lisäämällä kontrolleriin \$scope.changeName()-funktio, jolla vaikutetaan \$scope.name-ominaisuuden arvoon. \$scope.changeName()-funktiota kutsutaan <button>-elementtiin kiinnitetyllä ng-click-direktiivillä sovelluksen näkymässä.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9    <h1>{{name}}</h1>
10   <button ng-click="changeName()">Change name</button>
11
12  <script>
13    angular.module('myApp', [])
14      .controller('myController', ['$scope',
15        function($scope) {
16          $scope.name = 'John Doe';
17
18          $scope.changeName = function(){
19            $scope.name = 'Peter Jones';
20          };
21        }
22      ]);
23  </script>
24
25 </body>
26 </html>

```

Kuvio 10. Kontrollerin toiminta AngularJS:ssä.

2.4.5 Suodatin

AngularJS:ssä käyttäjälle esitettävää dataa voidaan muotoilla suodattimien (engl. filter) avulla. Suodattimia käytetään tyypillisesti AngularJS:n lausekkeissa, mutta suodattimia voidaan käyttää myös esimerkiksi kontrollerissa tai palvelussa (Diez 2013, 47; Branas 2014, 55). Suodattimet ovat hyödyllisiä esimerkiksi päivämäärien muotoiluun (Branas 2014, 55).

AngularJS:n syntaksissa suodatin erotetaan lausekkeesta pystyviivalla (W3 Schools 2015b). Suodattimia voidaan myös ketjuttaa eli yhdessä lausekkeessa voi olla useita suodattimia. Suodattimella voi olla myös argumentteja. (Developer Guide: Filters) AngularJS:ssä suodattimia voidaan käyttää alla osoitetulla tavalla:

```

{{ expression | filter }}
{{ expression | filter1 | filter2 }}
{{ expression | filter:argument1:argument2 }}

```

AngularJS sisältää useita sisäänrakennettuja suodattimia (Lerner 2013, 37). Sisäänrakennettuja suodattimia ovat esimerkiksi päivämäärän muotoiluun käytetty date, taulukon tai merkkijonon elementtien lukumäärää rajoittava limitTo tai halutun lausekkeen perusteella taulukon järjestystä muuttava orderBy (Ks. Lerner 2013, 38–42). Täydelli-

nen lista AngularJS:n sisäänrakennetuista suodattimista löytyy AngularJS:n virallisesta dokumentaatiosta (Ks. API: filter components in ng).

Kuviot 11 ja 12 havainnollistavat suodattimien toimintaa AngularJS:ssä. Kuviossa 11 suodattimia käytetään sekä AngularJS:n lausekkeessa että ng-repeat-direktiivissä. Esimerkissä AngularJS-sovellus säilötään aluksi moduuliin, johon lisätään angular.module API:n controller()-metodilla myController-niminen kontrolleri. Kontrollerin riippuvuudeksi syötetään \$scope-objekti, jolle määritetään ominaisuus \$scope.persons. \$scope.persons sisältää yksinkertaisen taulukon, joka iteroidaan ng-repeat-direktiivillä sovelluksen näkymässä. Ng-repeat-direktiiviin lisätään kaksi suodatinta. \$scope.persons-taulukko järjestetään ensin orderBy-suodattimella aakkosjärjestykseen sukunimen mukaan, minkä jälkeen taulukon lukumäärä rajoitetaan kahteen limitTo-suodattimella. Ng-repeat-direktiivin sisällä \$scope.persons-taulukon sisältämät nimet esitetään sovelluksen näkymässä kahdella lausekkeella. {{person.firstName}}-lausekkeeseen lisätään lowercase-suodatin, joka muuttaa tekstin pienillä kirjaimilla kirjoitetuksi. {{person.lastName}}-lausekkeeseen lisätään uppercase-suodatin, joka muuttaa tekstin isoilla kirjaimilla kirjoitetuksi. Kuvio 12 esittää kuvion 11 ohjelman selaimessa.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9    <ul>
10   <li ng-repeat="person in persons | orderBy:'lastName' | limitTo:2">
11     {{person.firstName | lowercase}} {{person.lastName | uppercase}}
12   </li>
13 </ul>
14
15 <script>
16 angular.module('myApp', [])
17   .controller('myController', ['$scope',
18     function($scope) {
19       $scope.persons = [
20         {firstName: 'John', lastName: 'Doe'},
21         {firstName: 'Anna', lastName: 'Smith'},
22         {firstName: 'Peter', lastName: 'Jones'}
23       ];
24     }
25   ]);
26 </script>
27
28 </body>
29 </html>

```

Kuvio 11. Suodattimien toiminta AngularJS:ssä.

- john DOE
- peter JONES

Kuvio 12. Suodattimen toiminta AngularJS:ssä. Kuvion 11 ohjelma selaimessa.

Sisäänrakennettujen suodattimien lisäksi AngularJS:ssä on usein hyödyllistä luoda omia suodattimia (Branas 2014, 61). Oma suodatin luodaan angular.module API:n filter()-metodilla. Filter()-metodi hyväksyy kaksi parametria: suodattimen nimen ja funktion, joka luo suodattimesta uuden instanssin (Panda 2014, 235).

Kuvio 13 havainnollistaa oman suodattimen luomista AngularJS:ssä. Esimerkissä toteutettu suodatin muuttaa merkkijonon isoilla kirjaimilla kirjoitetuksi. Toiminnaltaan se siis vastaa AngularJS:n sisäänrakennettua uppercase-suodatinta. Esimerkissä AngularJS-sovellus säilötään aluksi moduuliin, johon luodaan angular.module API:n filter()-metodilla uusi suodatin. Filter()-metodin ensimmäinen parametri on suodattimen nimi: myFilter. Toinen parametri on funktio, joka luo suodattimesta uuden instanssin. Funktio palauttaa uuden funktion, jonka parametri, input, on suodattimessa manipuloitava yksikkö. Funktio muuttaa merkkijonon isoilla kirjaimilla kirjoitetuksi JavaScriptin toUpperCase()-metodilla. Suodatinta käytetään näkymässä erottamalla suodatin pystyviivalla lausekkeesta: {{'John Doe' | myFilter}}.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body>
8
9    <h1>{{'John Doe' | myFilter}}</h1>
10
11  <script>
12    angular.module('myApp', [])
13      .filter('myFilter', function() {
14        return function(input) {
15          return input.toUpperCase();
16        };
17      });
18  </script>
19
20 </body>
21 </html>

```

Kuvio 13. Räätelöity suodatin AngularJS:ssä.

2.4.6 Palvelut

AngularJS:ssä palvelu (engl. service) on sovelluskomponentti, jota käytetään sovelluskoodin organisoimiseen ja jakamiseen sovelluksen eri osien kesken (Developer Guide:

Services). Palveluiden avulla sovelluksessa voidaan pitää haluttua dataa saatavilla koko sovelluksen elinkaaren ajan (Lerner 2013, 157). Palveluiden tarkoitus on säilyttää sellaista sovelluslogiikkaa, jota sovelluksessa tarvitaan toistuvasti (Panda 2014, 99). Palveluita käytetään AngularJS:n riippuvuusinjektion kautta sovelluksen kontrollerissa, direktiivissä, suodattimessa tai toisessa palvelussa (Developer Guide: Services).

AngularJS sisältää lukuisia sisäänrakennettuja palveluita (Lerner 2013, 157). Sisäänrakennettuja palveluita ovat esimerkiksi etäpalvelimen kanssa kommunikointiin käytetty `$http` tai selaimen URL-osoitteen tarkasteluun ja muokkaamiseen käytetty `$location` (Ks. API: `$http`; API: `$location`). Täydellinen lista AngularJS:n sisäänrakennetuista palveluista löytyy AngularJS:n virallisesta dokumentaatiosta (Ks. API: service components in ng)

Sisäänrakennettujen palvelujen lisäksi AngularJS:ssä on usein tarpeellista luoda omia palveluja (Panda 2014, 99). Oman palvelun luomiseen voidaan AngularJS:ssä käyttää viittä eri `angular.module()` API:n metodia:

- `value()`
- `constant()`
- `service()`
- `provider()`
- `factory()`

(Developer Guide: Providers).

Yleisin ja joustavin tapa palvelun luomiseen on käyttää `factory()`-metodia (Lerner 2013, 158). `Factory()`-metodi hyväksyy kaksi parametria. Ensimmäinen parametri on palvelun nimi. Toinen parametri on funktio, joka palauttaa objektin tai funktion. (Panda 2014, 104.)

Kuvio 14 havainnollistaa oman palvelun luomista AngularJS:ssä. Esimerkissä AngularJS-sovellus säilötään aluksi moduuliin, johon lisätään `angular.module` API:n `factory()`-metodilla palvelu. `Factory()`-metodin ensimmäinen parametri on palvelun nimi: `myFactory`. `Factory()`-metodin toinen parametri on funktio, joka palauttaa objektin, jonka `users`-ominaisuus sisältää yksinkertaisen taulukon. Esimerkissä `myFactory`-palvelu otetaan käyttöön sovelluksen kontrollerissa nimeämällä palvelu kontrollerin riippuvuu-

deksi. Kontrollerissa palvelun sisältämä taulukko tallennetaan `$scope.users`-muuttuun ja esitetään sovelluksen näkymässä `ng-repeat`-direktiivillä.

```

1  <!DOCTYPE html>
2  <html ng-app="myApp">
3  <head>
4    <title>AngularJS</title>
5    <script src="js/angular.min.js"></script>
6  </head>
7  <body ng-controller="myController">
8
9    <ul>
10   <li ng-repeat="user in users">
11     {{user}}
12   </li>
13 </ul>
14
15 <script>
16   angular.module('myApp', [])
17     .controller('myController', ['$scope', 'myFactory',
18     function($scope, myFactory) {
19       $scope.users = myFactory.users;
20     }
21   ])
22     .factory('myFactory', function() {
23       return {
24         users: ['John Doe', 'Anna Smith', 'Peter Jones']
25       }
26     });
27 </script>
28 </body>
29 </html>
30

```

Kuvio 14. Palvelun luominen AngularJS:ssä.

2.4.7 Reititys

AngularJS on kehitetty ensisijaisesti yksisivuisten web-sovellusten kehitykseen (Lerner 2013, 8). Vaikka yksisivuisissa web-sovelluksissa on teknisesti katsoen vain yksi sivu, käytännössä on usein tarpeellista luoda sovelluksen eri sisällöille useita URL-osoitteita, joiden välillä käyttäjä voi navigoida, lisätä kirjanmerkkeihin tai jakaa sosiaalisessa mediassa. AngularJS:ssä tällaista yksittäistä URL-osoitetta kutsutaan reitiksi.

Reititystä varten AngularJS:ssä on oma moduulinsa `ngRoute`, joka on erotettu AngularJS:n ytimeä erilliseksi tiedostoksi (Seshadri & Green 2014, 139). `NgRoute`-moduuli sisältää neljä sovelluskomponenttia: `$route`, `$routeProvider`, `$routeParams` ja `ng-view` (Ks. API: `ngRoute`).

- `$route`-palvelu hallinnoi reittejä assosioimalla halutun URL-osoitteen haluttuun kontrolleriin ja näkymään
- `$routeProvider`-palvelu konfiguroi reitit
- `$routeParams`-palvelu palauttaa reittien parametrit
- `ng-view`-direktiivi sisällyttää reitissä määritellyn näkymän ulkoasuun

(API: `ngRoute`.)

Reittien konfiguroinnissa käytetyllä `$routeProvider`-palvelulla on kaksi metodia, joita käyttämällä reitit luodaan:

- `when()`-metodi luo uuden reittimäärittelyn `$route`-palveluun
- `otherwise()`-metodi määrittää reitin, jota käytetään silloin kun mikään erikseen määritetyistä reiteistä ei täsmää

(API: `$routeProvider`.)

Kuviot 15, 16 ja 17 havainnollistavat reitityksen toteuttamista AngularJS:ssä. Reitityksen luomisen edellytyksenä on, että AngularJS:n ytimestä erotettu `ngRoute`-moduuli lisätään `<script>`-elementillä sivulle ja nimetään `myApp`-moduulin riippuvuudeksi.

Tämän jälkeen luodaan `angular.module` API:n `config()`-metodilla konfiguraatiokomponentti, jossa reitit määritellään. Komponentin riippuvuudeksi nimetään `$routeProvider`-palvelu, jolla reitit konfiguroidaan.

Konfiguraatiokomponentissa `$routeProvider`-palvelun `when()`-metodilla luodaan kaksi reittiä, joihin assosioidaan näkymä ja kontrolleri. Molempien reittien URL-templatessa käytetään kaksoispisteellä erotettua loppuliitettä, jolla merkitään reittiparametri. Kun reittiin on esimerkiksi määritetty `route1/:param` ja käyttäjä navigoi selaimessa osoitteeseen `route1/1234` reittiparametri on `1234`. Reittikonfiguraatio viimeistellään `otherwise()`-metodilla, jossa tehdään uudelleenohjaus sovelluksen juureen, silloin kun mikään erikseen määritetyistä reiteistä ei täsmää. Tämän jälkeen kahdelle määritetylle reitille luodaan reittikonfiguraatiota vastaavat näkymät: `template-1.html` ja `template-2.html`. Lopuksi sovelluksen `index.html`-sivulle lisätään `ng-view`-direktiivi, joka sisällyttää aktiivisen reitin näkymän pääsivulle. Lopuksi sovellukselle luodaan kontrolleri, jonka riippuvuudeksi nimetään `$routeParams`. `$routeParams`-palvelun avulla kontrollerissa päästään käsiksi aktiivisen reitin parametreihin. Kontrollerin sisällä kyseiset parametrit tallennetaan `$scope.param`-muuttujaan, joka esitetään näkymässä lausekkeella `{{param}}`.


```

1 <!DOCTYPE html>
2 <html ng-app="myApp">
3 <head>
4   <title>AngularJS</title>
5   <script src="js/angular.min.js"></script>
6   <script src="js/angular-route.min.js"></script>
7 </head>
8 <body>
9
10  <a ng-href="#/route1/abcd">Route 1 + param</a><br/>
11  <a ng-href="#/route2/1234">Route 2 + param</a><br/>
12
13  <div ng-view></div>
14
15  <script>
16    angular.module('myApp', ['ngRoute'])
17      .config(['$routeProvider', function($routeProvider) {
18        $routeProvider
19          .when('/route1/:param', {
20            templateUrl: 'template-1.html',
21            controller: 'myController'
22          })
23          .when('/route2/:param', {
24            templateUrl: 'template-2.html',
25            controller: 'myController'
26          })
27          .otherwise({
28            redirectTo: '/'
29          });
30      })
31      .controller('myController', ['$scope', '$routeParams', function($scope, $routeParams) {
32        $scope.param = $routeParams.param;
33      }]);
34  </script>
35
36 </body>
37 </html>

```

Kuvio 15. Reititys AngularJS:ssä.

```

1 <h1>Route 1 + {{param}}</h1>

```

Kuvio 16. template-1.html.

```

1 <h1>Route 2 + {{param}}</h1>

```

Kuvio 17. template-2.html.

3 Yksisivuisen web-sovelluksen käyttöliittymän toteutus AngularJS:llä

Tässä luvussa kuvaillaan miten AngularJS:llä toteutetaan yksisivuisen web-sovelluksen käyttöliittymä. Toteutettavan web-sovelluksen nimi on Tapahtumia Helsingissä. Web-sovelluksen idea on koota yhteen tietoa Helsingin seudulla järjestettävistä tapahtumista. Web-sovelluksen datalähteenä on avoin ohjelmointirajapinta HKI Linked Events API, joka sisältää tapahtumadataa pääkaupunkiseudulla järjestettävistä tapahtumista (Ks. Linked Events).

Tapahtumia Helsingissä -web-sovellus käsittää AngularJS:llä toteutetun käyttöliittymän lisäksi MongoDB-tietokannan, Node.js-web-palvelimen sekä ExpressJS:llä toteutetun ohjelmointirajapinnan. Web-sovelluksessa käytetään myös muita web-kehitystä helpottavia työkaluja kuten Twitter Bootstrap -CSS-ohjelmointikehystä. Tässä opinnäytetyössä tarkastelun kohteena on kuitenkin vain AngularJS:llä toteutettu käyttöliittymä. Web-sovelluksen palvelinohjelmistoa, tietokantaa tai tyylimäärityitä ei käsitellä.

Tapahtumia Helsingissä -web-sovelluksen käyttöliittymä käsittää kolme näkymää: etusivun, tapahtumien listaussivun sekä yksittäisen tapahtuman näkymän. Web-sovelluksen etusivu on yksinkertainen ”landing page”, joka ei sisällä erityisiä toiminnallisuksia. Tapahtumien listaussivulla käyttäjä voi valita tapahtumakategorian ja päivämäärän, joiden perusteella tapahtumia listataan. Tapahtumien listaussivu sisältää myös sivutuksen, jossa käyttäjä voi näyttää lisää tapahtumia painamalla ”Näytä lisää” -painiketta. Yksittäisen tapahtuman sivulla esitetään yksittäisen tapahtuman tiedot.



Kuvio 18. Tapahtumia Helsingissä -web-sovelluksen etusivu.

Tapahtumia Helsingissä web-sovelluksen käyttöliittymän toteutus aloitetaan luomalla tarvittavat tiedostot ja hakemistot. Hakemistorakenteen luomisen jälkeen AngularJS otetaan web-sovelluksen käyttöön ja index.html-sivulle sisällytetään tarvittavat JavaScript- ja CSS-tiedostot. Varsinainen sovelluskehitys aloitetaan luomalla moduuli, johon web-sovellus ja sen osat säilötään. Tämän jälkeen web-sovellukseen luodaan reititys etusivua, tapahtumalistausta ja yksittäistä tapahtumaa varten. Reitityksen jäl-

keen luodaan web-sovelluksen palvelinkommunikoinnista vastaava palvelu. Lopuksi sovellukseen luodaan kontrolleri, joka sisältää käyttöliittymän sovelluslogiikan. Lisäksi käyttöliittymän näkymiin luodaan tarvittava HTML-sisältö sekä AngularJS:n lausekkeet ja direktiivit.

3.1 Hakemistorakenne

Tapahtumia Helsingissä -web-sovelluksen toteuttaminen aloitetaan luomalla tarvittavat tiedostot ja hakemistot. Kuvio 19 kuvastaa Tapahtumia Helsingissä -web-sovelluksen hakemistorakennetta.

```
tapahtumia-helsingissa/
----- app/
----- bower_components/
----- angular/
----- angular.min.js
----- angular-i18n/
----- angular-locale_fi-fi.js
----- angular-resource/
----- angular-resource.min.js
----- angular-ui-router/
----- angular-ui-router.min.js
----- bootstrap/
----- dist
----- css/
----- bootstrap.min.css
----- fontawesome/
----- css/
----- font-awesome.min.css
----- config/
----- node_modules/
----- public/
----- css/
----- app.css
----- img/
----- front.jpg
----- loading.gif
----- js/
----- app.js
----- controllers.js
----- routes.js
----- services.js
----- views/
----- home.html
----- list.html
----- view.html
----- index.html
----- bower.json
----- package.json
----- server.js
```

Kuvio 19. Tapahtumia Helsingissä web-sovelluksen hakemistorakenne.

Web-sovelluksen backend-koodia sijaitsee kansioissa app, config ja node_modules sekä tiedostossa server.js. Hakemistot bower_components ja public puolestaan sisäl-

tävät tarkastelun kohteena olevan käyttöliittymän tiedostot. Bower_components-kansio sisältää käyttöliittymän tarvitsemat kirjastot – kuten AngularJS:n lähdekoodin. Public-kansio sisältää käyttöliittymän varsinaisen sovelluskoodin: AngularJS-sovelluskoodin, käyttöliittymässä tarvittavat HTML-templatet, CSS-tyylit sekä index.html-sivun.

3.2 Esityöt

Web-sovelluksen toteuttaminen aloitetaan ottamalla AngularJS web-sovelluksen käyttöön ja lisäämällä index.html-sivulle tarvittavat JavaScript- ja CSS-tiedostot. Lisäksi index.html-sivulle lisätään alustava HTML-sisältö.

Kuvio 20 esittää Tapahtumia Helsingissä -sovelluksen index.html-sivun kokonaisuudessaan. Kuviossa AngularJS otetaan web-sovelluksen käyttöön lisäämällä AngularJS:n lähdekoodi (angular.min.js) <script>-elementillä index.html-sivulle ja kertomalla <html>-elementtiin merkityllä ng-app-direktiivillä mikä osa DOMista on AngularJS:n hallinnassa. Ng-app-direktiivissä sovelluksen nimeksi annetaan app.

AngularJS:n lähdekoodin lisäksi sivulle lisätään kaksi moduulia, joita tarvitaan myöhemmin sovelluksessa. Ensimmäinen moduuli on kolmannen osapuolen kehittämä Angular UI Router -moduuli (angular-ui-router.min.js), jolla toteutetaan sovelluksen reititys. Toinen moduuli on AngularJS:n ngResource-moduuli (angular-resource.min.js), jonka avulla kommunikoidaan sovelluksen palvelimen kanssa. Moduulien lisäksi sivulle lisätään lokalisoitietiedosto (angular-locale_fi-fi.js), jonka avulla esimerkiksi päivämäärät ovat oletusarvoisesti suomenkielisessä muodossa (Developer Guide: i18n and l10n).

Tämän jälkeen sivulle lisätään varsinaisen AngularJS-sovelluskoodin sisältävät tiedostot (app.js, routes.js, services.js ja controllers.js). Tiedostoon app.js luodaan myöhemmin sovelluksen moduuli, joka kokoaa kaikki sovelluksessa käytettävät komponentit yhteen paikkaan. Tiedostoon routes.js luodaan myöhemmin sovelluksen reititys ja tiedostoon services.js sovelluksen palvelinkommunikoinnista vastaava sovelluskomponentti. Tiedostoon controllers.js luodaan myöhemmin sovelluksen tarvitsema liiketoimintalogiikka.

Tarvittavien JavaScript-tiedostojen lisäämisen jälkeen sivun <head>-osioon lisätään Twitter Bootstrap -CSS-ohjelmistokehys (bootstrap.min.css) sekä sovelluksen oma

tyylitiedosto (app.css). Lisäksi sivun <head>-osioon lisätään kaksi Google-fonttia ja Fontawesome-ikonikirjasto (font-awesome.min.css).

Tarvittavien tiedostojen lisäämisen jälkeen index.html-sivulle luodaan HTML-sisältöä. Sivun yläosaa varten luodaan <header>-elementti, joka sisältää linkit sovelluksen etusivuun ja tapahtumalistaukseen. Yläosan alle luodaan <section>-elementti, johon myöhemmin lisätään dynaamisesti HTML-sisältöä. Sivulla näkyvät ui-sref- ja ui-view-direktiivit liittyvät sovelluksen reititykseen ja Angular UI Router-moduulin toimintaan, jota käsitellään luvussa 3.4. Ui-sref-direktiivi mahdollistaa navigoinnin web-sovellukseen luotavien reittien välillä ja ui-view-direktiivi merkitsee elementin, jonka sisään lisätään dynaamisesti HTML-sisältöä.

```

1  <!DOCTYPE html>
2  <html ng-app="app">
3  <head>
4    <meta charset="utf-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <title>Tapahtumia Helsingissä</title>
7    <meta name="description" content="Satoja tapahtumia pääkaupunkiseudulla.">
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9
10   <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Roboto:400,700">
11   <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Alfa+Slab+One">
12   <link rel="stylesheet" href="/bower_components/fontawesome/css/font-awesome.min.css">
13   <link rel="stylesheet" href="/bower_components/bootstrap/dist/css/bootstrap.min.css">
14   <link rel="stylesheet" href="css/app.css">
15 </head>
16 <body>
17
18 <header class="header">
19   <div class="container">
20     <div class="row">
21       <div class="col-md-12">
22         <div class="logo">
23           <a ui-sref="home">Tapahtumia Helsingissä</a>
24         </div>
25         <nav class="nav">
26           <a ui-sref="list">Selaa tapahtumia</a>
27         </nav>
28       </div>
29     </div>
30   </div>
31 </header>
32
33 <section class="content" ui-view></section>
34
35 <script type="text/javascript" src="/bower_components/angular/angular.min.js"></script>
36 <script type="text/javascript" src="/bower_components/angular-ui-router/release/angular-ui-router.min.js"></script>
37 <script type="text/javascript" src="/bower_components/angular-resource/angular-resource.min.js"></script>
38 <script type="text/javascript" src="/bower_components/angular-i18n/angular-locale_fi-fi.js"></script>
39
40 <script type="text/javascript" src="js/app.js"></script>
41 <script type="text/javascript" src="js/routes.js"></script>
42 <script type="text/javascript" src="js/services.js"></script>
43 <script type="text/javascript" src="js/controllers.js"></script>
44
45 </body>
46 </html>

```

Kuvio 20. Tapahtumia Helsingissä -web-sovelluksen index.html-sivu.

3.3 Sovelluksen säilöminen moduuliin

Esitöiden jälkeen Tapahtumia Helsingissä -web-sovellukseen luodaan moduuli, jonka tehtävä on koota sovelluksen komponentit yhteen nimettyyn paikkaan. Moduuli luodaan tiedostoon app.js (Kuvio 21).

Moduuli luodaan `angular.module()`-metodilla, joka hyväksyy kaksi parametria. Ensimmäinen parametri on moduulin nimi `app`, joka vastaa `index.html`-tiedoston `<html>`-elementtiin merkityn `ng-app`-direktiivin nimeä (Ks. kuvio 20). Toinen parametri on taulukko, joka sisältää moduulin riippuvuudet. Moduulin riippuvuudeksi nimetään moduulit Angular UI Router ja `ngResource` (`ui.router` ja `ngResource`), jotka edellisessä luvussa lisättiin `index.html`-sivulle (Ks. kuvio 20). Lopuksi moduuli tallennetaan muuttuunaan `app`, jonka avulla sovellusmoduuliin jatkossa viitataan. Jatkossa moduuliin lisätään sovelluskomponentteja viittaamalla moduuliin ja käyttämällä `angular.module` API:n metodeja. Esimerkiksi kontrolleri lisätään moduuliin kirjoittamalla `app.controller()`.

```

1
2  /**
3   * app
4   */
5
6  var app = angular.module('app', ['ui.router', 'ngResource']);
7

```

Kuvio 21. Tapahtumia Helsingissä -web-sovelluksen säilöminen moduuliin (app.js).

3.4 Reititys

Tapahtumia Helsingissä -web-sovellukseen luodaan seuraavaksi reititys, joka mahdollistaa navigoimisen web-sovelluksen näkymien – etusivun, tapahtumalistauksen ja yksittäisen tapahtumasivun – välillä. Reititys luodaan tiedostoon `routes.js` (Kuvio 22).

Tapahtumia Helsingissä -web-sovelluksen reitit luodaan käyttämällä kolmannen osapuolen kehittämää Angular UI Router -moduulia (Ks. Angular UI Router 2015). Angular UI Router -moduulia käytetään samankaltaisesti kuin AngularJS:n omaa reititystä, jota tarkasteltiin luvussa 2.4.7. Angular UI Router tarjoaa kuitenkin AngularJS:n reititykseen verrattuna enemmän ominaisuuksia – esimerkiksi sisäkkäisten näkymien esittäminen – reitityksen toteuttamiseen (Panda 2014, 93–94). Angular UI Router -moduuli eroaa AngularJS:n omasta reitityksestä siinä, että käyttäjälle näytettävä näkymä määritetään web-sovellukselle asetettavan ”tilan” (engl. state) perusteella toisin kuin AngularJS:n

omassa reitityksessä, jossa näkymä määritetään URL-osoitteen perusteella (Sevilleja 2014).

Angular UI Router -moduulin käyttöä edellyttävät toimenpiteet toteutettiin aiemmin luvuissa 3.2 ja 3.3. Luvussa 3.2 Angular UI Router -moduuli sisällytettiin `<script>`-elementillä `index.html`-sivulle ja luvussa 3.3 Angular UI Router -moduuli määritettiin sovellusmoduulin riippuvuudeksi. Seuraavaksi web-sovellukseen luodaan reitit käyttämällä Angular UI Router -moduulin `$stateProvider`- ja `$urlRouterProvider`-palveluita.

Reitityksen luominen aloitetaan lisäämällä *Tapahtumia Helsingissä* -web-sovelluksen moduuliin konfiguraatiokomponentti `angular.module` API:n `config()`-metodilla. Konfiguraatiokomponentin riippuvuudeksi nimetään Angular UI Router -moduulin `$stateProvider`- ja `$urlRouterProvider`-palvelut.

Konfiguraatiokomponentissa sovellukselle määritetään `$stateProvider`-palvelun `state()`-metodilla kolme reittiä sovelluksen etusivua, tapahtumalistausta ja yksittäisen tapahtuman esittämistä varten. Kullekin reitille määritetään oma nimi, URL-osoite ja HTML-template. Reittien `templateURL`-määrittäjiä vastaavat HTML-tiedostot – `home.html`, `list.html` ja `view.html` – luotiin `views`-kansioon sovelluksen hakemistorakennetta luotaessa luvussa 3.1. Reittikonfiguraatio viimeistellään luomalla `$urlRouterProvider`-palvelun `otherwise()`-metodilla sovelluksen juuriosoitteeseen ohjaava reitti, joka ohjaa käyttäjän sovelluksen etusivulle, jos käyttäjä navigoi muualle kuin erikseen määritettyyn reittiin.

Aikaisemmin sovelluksen esitöissä luvussa 3.2 `index.html`-sivun `<section>`-elementtiin lisättiin Angular UI Router -moduulin `ui-view`-direktiivi, jolla aktiivisen reitin HTML-template saadaan nyt dynaamisesti sisällytettyä `index.html`-sivuun (Ks. kuvio 20). Toisin sanoen kun käyttäjä navigoi esimerkiksi osoitteeseen `/events`, `index.html`-sivulla `ui-view`-direktiiviin sisään lisätään HTML-template `list.html`. `Index.html`-sivulla käytettiin myös `ui-sref`-direktiiviä, jonka avulla voidaan navigoida nyt luotujen ”tilojen” välillä (Ks. kuvio 20).

```

1  /**
2   * Routes
3   */
4
5
6  app.config(['$stateProvider', '$urlRouterProvider',
7    function($stateProvider, $urlRouterProvider) {
8
9      $stateProvider
10       .state('home', {
11         url: '/',
12         templateUrl: 'views/home.html'
13       })
14       .state('list', {
15         url: '/events',
16         templateUrl: 'views/list.html'
17       })
18       .state('view', {
19         url: '/events/:eventId',
20         templateUrl: 'views/view.html'
21       });
22
23       $urlRouterProvider.otherwise('/');
24     }
25   ]);

```

Kuvio 22. Tapahtumia Helsingissä -web-sovelluksen reititys (routes.js).

Reitityksen luomisen jälkeen voidaan web-sovelluksen etusivua varten luoda tarvittava HTML-sisältö tiedostoon home.html (Kuvio 23). Etusivua varten luodaan yksinkertainen näkymä, joka sisältää web-sovelluksen nimen, sloganin ja linkin tapahtumien listaussivulle. Etusivun näkymä selaimessa esitettiin kuviossa 18. Tapahtumien listaussivun ja yksittäisen tapahtuman HTML-sisältö luodaan sovellukseen myöhemmin luvussa 3.6.

```

1  <section class="banner">
2    <div class="home">
3      <h1>Tapahtumia Helsingissä.</h1>
4      <p>Satoja tapahtumia pääkaupunkiseudulla.</p>
5      <a ui-sref="list" class="btn btn-primary">
6        Selaa tapahtumia <i class="fa fa-chevron-circle-right"></i>
7      </a>
8    </div>
9  </section>
10

```

Kuvio 23. Etusivun HTML-template (home.html)

3.5 Palvelinkommunikointi

Tapahtumia Helsingissä -web-sovelluksen luodaan seuraavaksi sovelluksen palvelinkommunikoinnista vastaava sovelluskomponentti. Sovelluskomponentin tarkoitus on muodostaa web-sovelluksen palvelimelle HTTP-kyselyitä, joiden perusteella palvelin palauttaa haluttua tapahtumadataa, joka esitetään käyttäjälle sovelluksen näkymässä.

Palvelinkommunikointi toteutetaan AngularJS:n ngResource-moduulin avulla. Moduulin käyttöä edellyttävät toimenpiteet – moduulin lisääminen sivulle ja moduulin nimeäminen sovellusmoduulin riippuvuudeksi – toteutettiin luvuissa 3.2 ja 3.3 (Ks. kuviot 20 ja

21). Palvelinkommunikoinnista vastaava sovelluskomponentti organisoidaan AngularJS:n palvelun avulla itsenäiseksi kokonaisuudeksi, jota voidaan käyttää toistuvasti sovelluksen kontrollerissa. Palvelinkommunikointi luodaan tiedostoon services.js (Kuvio 24).

Palvelinkommunikoinnin toteuttaminen aloitetaan luomalla palvelu, jonka sisällä varsinainen palvelinkommunikointi tapahtuu. Palvelu luodaan angular.module API:n factory()-metodilla, joka hyväksyy kaksi parametriä. Ensimmäinen parametri on palvelun nimi Events. Toinen parametri on lista palvelun riippuvuuksista. Palvelun riippuvuudeksi määritetään ngResource-moduulin \$resource-palvelu.

\$resource-palvelu AngularJS:ssä on suunniteltu erityisesti palvelinpuolen REST-ohjelmointirajapintojen kanssa kommunikointiin. \$resource-palvelu kommunikoi palvelimen kanssa HTTP-kyselyillä selaimen XMLHttpRequest-objektin kautta. AngularJS:ssä \$resource-palvelu on korkeamman tason abstraktio \$http-palvelusta, jota voidaan myös käyttää palvelinkyselyiden muodostamiseen. Tapahtumia Helsingissä -web-sovelluksessa käytetään kuitenkin \$resource-palvelua, koska \$resource-palvelu sisältää joukon valmiita metodeja, joita hyödyntämällä kirjoitettavan koodin määrä voidaan minimoida (Lerner 2013, 185). Oletuksena \$resource-palvelu sisältää viisi oletustoimintoa, joista Tapahtumia Helsingissä -web-sovelluksessa käytetään kuitenkin vain get()-metodia. Kaikkiaan \$resource-palvelu sisältää seuraavat toiminnot:

```
{ 'get': {method: 'GET'},
  'save': {method: 'POST'},
  'query': {method: 'GET', isArray: true},
  'remove': {method: 'DELETE'},
  'delete': {method: 'DELETE'} };
```

Events-palvelun sisällä \$resource-palvelulle määritetään kaksi parametriä. Ensimmäinen parametri on URL-osoite, johon kyselyt osoitetaan. Toinen parametri on objekti, joka sisältää URL-parametrien oletusarvot. URL-osoite sisältää kaksoispisteellä erotetun loppuliitteen eventId. Loppuliitteellä kerrotaan, että kyseinen parametri halutaan osaksi URL-polkua. Muut mahdolliset URL-parametrit lisätään URL-osoitteeseen kysymysmerkillä erotettuna. Esimerkiksi URL-template /api/events/:eventId ja parametri {eventId: 1234} johtaa kyselyyn /api/events/1234. Sen sijaan URL-template /api/events/:eventId ja parametri {limit: 20, offset: 0 } johtavat URL-osoitteeseen

/api/events?limit=20&offset=0. Tämän lisäksi URL-parametrin eventId oletusarvo sisältää @-etuliitteen, jolla kerrotaan että kyseisen parametrin arvo halutaan erottaa objektista kyselyä tehtäessä.

```

1
2  /**
3   * Services
4   */
5
6  app.factory('Events', ['$resource',
7    function($resource) {
8      return $resource(
9        'http://localhost\\:8080/api/events/:eventId',
10       {eventId: '@_id'}
11     );
12   }
13   ]);
14

```

Kuvio 24. Palvelimen kanssa kommunikointi (services.js).

Events-palvelu otetaan seuraavassa luvussa käyttöön sovelluksen kontrollerissa nimellä palvelu kontrollerin riippuvuudeksi (Ks. luku 3.6). Palvelua käytetään Tapahtumia Helsingissä -web-sovelluksen kontrollerissa viittaamalla palvelun nimeen ja käyttämällä get()-metodia. Alla on kolme esimerkkikyselyä, jotka palvelun avulla sovelluksen kontrollerissavoidaan muodostaa:

- Events.get() suorittaa HTTP GET -kyselyn osoitteeseen /api/events
- Events.get({limit: 20, offset: 0}) suorittaa HTTP GET -kyselyn osoitteeseen /api/events?limit=20&offset=0
- Events.get({eventId: 1234}) suorittaa HTTP GET -kyselyn osoitteeseen /api/events/1234

3.6 Sovelluslogiikka

Tapahtumia Helsingissä -web-sovellus käsittää toistaiseksi moduulin, reitityksen sekä palvelinkommunikoinnista vastaavan Events-palvelun. Seuraavaksi sovellukseen lisätään kontrolleri, jossa luodaan web-sovelluksen tarvitsema sovelluslogiikka. Sovelluslogiikkaa tarvitaan tapahtumien listausta, yksittäisen tapahtuman esittämistä, päivämäärän valintaa, tapahtumakategorian valintaa sekä sivutusta varten. Sovelluslogiikan luomisen yhteydessä tapahtumien listaussivulle ja yksittäisen tapahtuman sivulle luodaan tarvittavat HTML-sisällöt, joihin lisätään toiminnallisuutta ja yhdistetään dataa AngularJS:n direktiivien, lausekkeiden ja suodattimien avulla. Sovelluksen kontrolleri luodaan tiedostoon controllers.js (Kuvio 25). Kuvio 25 esittää sovelluksen kontrollerin

kokonaisuudessaan. Seuraavissa luvuissa kontrollerin toimintaa tarkastellaan yksityiskohtaisemmin.

Sovelluksen kontrolleri luodaan `angular.module` API:n `controller()`-metodilla, joka hyväksyy kaksi parametriä. Ensimmäinen parametri on kontrollerin nimi `EventsController`. Toinen parametri on lista kontrollerin riippuvuuksista. Kontrollerin riippuvuudeksi nimitään `$scope`-objekti, `$filters`-palvelu, edellisessä luvussa luotu `Events`-palvelu sekä luvussa 3.4 luotuun reititykseen liittyvä `Angular UI Router` -moduulin `$stateParams`-palvelu. `$scope`-objekti on sovelluksen datamalli, johon kontrollerissa lisätään ominaisuuksia, joihin sovelluksen näkymällä on pääsy. `$filters`-palvelun avulla voidaan sovelluksen kontrollerissa muokata esimerkiksi päivämääriä haluttuun muotoon. `Events`-palvelun avulla muodostetaan kyselyitä sovelluksen web-palvelimelle, joka palauttaa kyselyiden perusteella haluttua tapahtumadataa. `Angular UI Router` -moduulin `$stateParams`-palvelua käytetään erottamaan URL-osoitteesta haluttuja reittiparametreja.

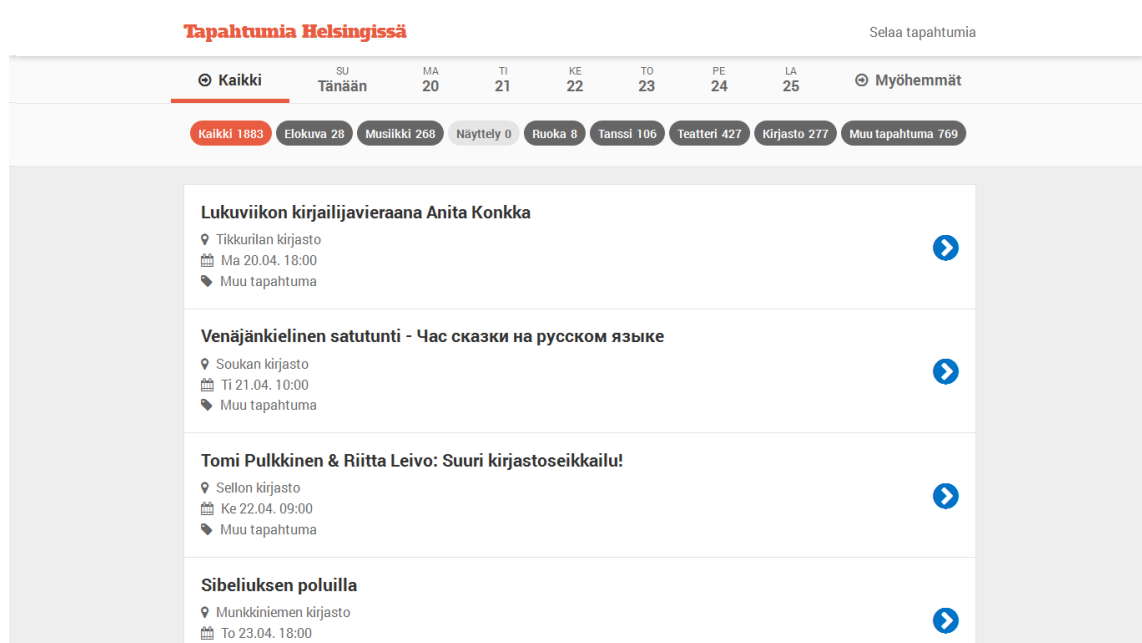
```

1
2 /**
3  * Controllers
4  */
5
6 app.controller('EventsController', ['$scope', '$stateParams', '$filter', 'Events',
7
8     function($scope, $stateParams, $filter, Events) {
9
10         /**
11          * Initial settings
12          */
13         $scope.events = [];
14         $scope.categories = [];
15         $scope.total = 0;
16         $scope.loading = false;
17
18         /**
19          * Query filters
20          */
21         var filters = {
22             category: 'Kaikki',
23             start_time: $filter('date')(new Date(), 'yyyy-MM-dd'), // Today
24             end_time: $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'), // Next Year
25             limit: 20,
26             offset: 0
27         };
28
29         /**
30          * Pagination
31          */
32         $scope.showMore = function() {
33             filters.offset += 20;
34             $scope.find();
35         };
36
37         $scope.hasMore = function() {
38             if($scope.loading === false)
39                 return $scope.events.length < $scope.total;
40         };
41
42         /**
43          * Dates for date select
44          */
45         $scope.dates = [];
46
47         for(var i=0; i<7; i=i+1){
48             var date = new Date(Math.abs(new Date()) + (86400000 * i));
49             date = $filter('date')(date, 'yyyy-MM-dd');
50             $scope.dates.push(date);
51         }
52
53         /**
54          * Select date
55          */
56         $scope.selectDate = function(date) {
57             filters.start_time = date; // Selected date
58             filters.end_time = $filter('date')(new Date(Math.abs(new Date(date)) + 86400000), 'yyyy-MM-dd'); // The day after
59             filters.offset = 0;
60             $scope.events = [];
61             $scope.find();
62         };
63
64         $scope.selectDateAll = function() {
65             filters.start_time = $filter('date')(new Date(), 'yyyy-MM-dd'); // Today
66             filters.end_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'); // Next year
67             filters.offset = 0;
68             $scope.events = [];
69             $scope.find();
70         };
71
72         $scope.selectDateFuture = function() {
73             filters.start_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 7)), 'yyyy-MM-dd'); // Next week
74             filters.end_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'); // Next year
75             filters.offset = 0;
76             $scope.events = [];
77             $scope.find();
78         };
79
80         /**
81          * Select category
82          */
83         $scope.selectCategory = function(category) {
84             filters.category = category;
85             filters.offset = 0;
86             $scope.events = [];
87             $scope.find();
88         };
89
90         /**
91          * Find events
92          */
93         $scope.find = function() {
94             $scope.loading = true;
95
96             Events.get(filters, function(data) {
97                 $scope.events = $scope.events.concat(data.events);
98                 $scope.categories = data.categories;
99                 $scope.total = data.total;
100                 $scope.loading = false;
101             });
102         };
103
104         /**
105          * Find one event
106          */
107         $scope.findOne = function() {
108             Events.get({eventId: $stateParams.eventId},
109                 function(event) {
110                     $scope.event = event;
111                 });
112         };
113     }
114 });
115

```

Kuvio 25. Tapahtumia Helsingissä web-sovelluksen kontrolleri (controllers.js).

3.6.1 Tapahtumien listaus



Kuvio 26. Tapahtumien listaussivu selaimessa.

Tapahtumien listausta varten kontrolleriin luodaan `$scope.find()`-funktio, jonka pääasiallisena tehtävänä on muodostaa HTTP-kysely web-palvelimelle ja tallentaa palvelimelta saatu data `$scope`-objektiin. Tämän lisäksi `$scope.find()`-funktio vaikuttaa muuttujaan `$scope.loading` arvoon, jonka perusteella sovelluksen näkymässä näytetään tai piilotetaan hakuanimaatio.

`$scope.find()`-funktiossa palvelinkysely tehdään käyttämällä luvussa 3.5 luotua Events-palvelua. Kysely käynnistetään kutsumalla `Events.get()`-metodia. `Events.get()`-metodin ensimmäisenä parametrina on `filters`-objekti, joka sisältää joukon kyselyssä käytettäviä hakuparametreja (Kuvio 27). `Filters`-objekti sisältää ominaisuudet `category`, `start_time`, `end_time`, `limit` ja `offset`. `Filters`-objektissa `start_time` on oletusarvoisesti nykyinen päivä ja `end_time` vuoden päässä oleva päivä. `Filters`-objektin ominaisuudet `limit` ja `offset` määrittävät kuinka monta tapahtumaa ja mistä kohtaa listaa tapahtumia halutaan.

Oletusarvoisesti `Events.get()` muodostaa HTTP GET -kyselyn osoitteeseen `/api/events?category=Kaikki&start_time=2015-01-01&end_time=2016-01-`

01&limit=20&offset=0. Toisin sanoen palvelimelle muodostetaan kysely, jossa listataan kaksikymmentä ensimmäistä tapahtumaa mistä tahansa kategoriasta seuraavan vuoden ajalta.

Web-palvelin palauttaa kyselyn perustella vastauksen JSON-formaatissa (Kuvio 29). Palvelimelta saatu data tallennetaan Events.get()-metodin toisena parametrina olevassa funktiossa \$scope-objektin ominaisuuksiin: \$scope.events-taulukkoon tallennetaan itse tapahtumadata, \$scope.categories-taulukkoon tapahtumakategoriadata ja \$scope.total-muuttujaan kaikkien tapahtumien lukumäärän (Kuvio 28).

```

17
18
19
20
21
22
23
24
25
26
27
28
  /**
  * Query filters
  */
  var filters = {
    category: 'Kaikki',
    start_time: $filter('date')(new Date(), 'yyyy-MM-dd'), // Today
    end_time: $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'), // Next Year
    limit: 20,
    offset: 0
  };

```

Kuvio 27. Kyselyparametrit sisältävä filters-objekti (controllers.js).

```

89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
  /**
  * Find events
  */
  $scope.find = function() {
    $scope.loading = true;

    Events.get(filters, function(data) {
      $scope.events = $scope.events.concat(data.events);
      $scope.categories = data.categories;
      $scope.total = data.total;
      $scope.loading = false;
    });
  };

```

Kuvio 28. Sovelluslogiikkaa tapahtumien listausta varten sovelluksen kontrollerissa (controllers.js).

```

1 {
2   "events": [
3     {
4       "_id": "54d39c4d1b6657262063829a",
5       "title": "Suomalais-venäläinen satutuokio",
6       "category": "Kirjasto",
7       "street": "Suursuonlaita 6",
8       "zip": "00630",
9       "city": "Helsinki",
10      "description": "<p> Satutuokiossa luetaan... Maunulan kirjasto.</p>",
11      "venue": "Maunulan kirjasto",
12      "_v": 0,
13      "end_time": "2015-02-18T17:00:00.000Z",
14      "start_time": "2015-02-18T16:15:00.000Z",
15      "created": "2015-02-05T16:37:33.402Z"
16    },
17  ],
18  {}
19 ],
20 "categories": [
21   {"name": "Kaikki", "count": 3428},
22   {"name": "Elokuva", "count": 87},
23   {"name": "Musiiikki", "count": 325},
24   {"name": "Näyttely", "count": 0},
25   {"name": "Ruoka", "count": 10},
26   {"name": "Tanssi", "count": 88},
27   {"name": "Teatteri", "count": 720},
28   {"name": "Kirjasto", "count": 508},
29   {"name": "Muu tapahtuma", "count": 1690}
30 ],
31 "total": 3428
32 }
33

```

Kuvio 29. Esimerkki web-palvelimen palauttamasta tapahtumadatasta JSON-formaatissa.

Sovelluslogiikan luomisen jälkeen lisätään tapahtumalistauksen HTML-templateen (list.html) tarvittava HTML-sisältö, direktiivit ja lausekkeet (Kuvio 30), joiden avulla tapahtumalista esitetään käyttäjälle. Sivulle lisätään aluksi ng-controller-direktiivi, jolla kerrotaan että EventsController-kontrolleri hallinnoi kyseistä sivua. Tämän jälkeen \$scope.find()-funktia kutsutaan sovelluksen näkymässä ng-init-direktiivillä. Ng-init-direktiivin avulla \$scope.find()-funktio käynnistää tapahtumakyselyn välittömästi, kun käyttäjä siirtyy tapahtumien listaussivulle. Lopuksi \$scope.events-taulukon sisältämä data esitetään käyttäjälle iteroimalla taulukko ng-repeat-direktiivillä. Ng-repeat-direktiivin sisällä määritellään yksittäisen tapahtumaelementin HTML-sisältö. \$scope-objektiin tallennettu tapahtumadata esitetään näkymässä AngularJS:n lausekkeilla. Tapahtuman päivämäärän muotoilussa käytetään AngularJS:n sisäänrakennettua date-suodatinta. Lisäksi HTML-templateen merkitään kommentteilla paikat, johon luvuissa 3.6.3, 3.6.4 ja 3.6.5 lisätään päivämäärävalikko, tapahtumakategoriavalikko sekä sivutuksessa käytettävät elementit.

```

1 <div ng-controller="EventsController" ng-init="find()">
2
3 <!-- Select date -->
4 <!-- END Select date -->
5
6 <!-- Select category -->
7 <!-- END Select category -->
8
9 <section class="main">
10 <div class="container">
11 <div class="row">
12 <div class="col-md-12">
13
14 <!-- List events -->
15 <div class="component events">
16 <div class="item" ng-repeat="event in events">
17 <a ng-href="#/events/{{event._id}}">
18 <div class="body">
19 <h1 class="title">{{event.title}}</h1>
20 <div class="location">
21 <i class="fa fa-map-marker"></i> {{event.venue}}
22 </div>
23 <div class="time">
24 <i class="fa fa-calendar"></i> {{event.start_time | date:'EEE dd.MM. HH:mm'}}
25 </div>
26 <div class="tag">
27 <i class="fa fa-tag"></i> {{event.category}}
28 </div>
29 </div>
30 <div class="chevron">
31 <i class="fa fa-chevron-circle-right"></i>
32 </div>
33 </a>
34 </div>
35 </div>
36 <!-- END List events -->
37
38 <!-- Pagination -->
39 <!-- END Pagination -->
40
41 </div>
42 </div>
43 </div>
44 </section>
45 </div>
46

```

Kuvio 30. Tapahtumien listaussivun HTML-template (list.html).

3.6.2 Yksittäisen tapahtuman esittäminen

Tapahtumia Helsingissä
Selaa tapahtumia

[Takaisin listaukseen](#)

Helsinki Beer Festival 2015

Kaapelitehdas täyttyy jälleen oluenystävistä. Helsinki Beer Festival esittelee oluita ja siidereitä tehtaan täydeltä sekä huipputunnelmaa kahden päivän ajan. Tapahtumassa julistetaan vuoden olut-, siideri- ja viskikilpailun voittajat. Tämän vuoden teemamaa on Itävalta.

Kaapelitehdas

Alkaa: Pe 10.04.2015 03:00

Päättyy: Su 12.04.2015 00:00

Ruoka

Tallberginkatu 1 C 15, C-porras, 2. krs, 00180 Helsinki

Kuvio 31. Yksittäisen tapahtuman sivu selaimessa.

Yksittäisen tapahtuman esittämistä varten kontrolleriin luodaan `$scope.findOne()`-funktio, jonka tehtävä on tapahtumalistauksen tapaan muodostaa HTTP-kysely web-palvelimelle ja tallentaa palvelimelta saatu data `$scope`-objektiin.

`$scope.findOne()`-funktio muodostaa kyselyn palvelimelle samalla tapaa kuin `$scope.find()`-funktio tapahtumalistauksessa (Kuvio 32). Erona on, että `$scope.findOne`-funktiossa `Events.get()`-metodin ensimmäisenä parametrina on `filters`-objektin sijasta tapahtuman yksilöllinen tunnus `eventId`, jonka arvo erotetaan aktiivisen reitin polusta Angular UI Router -moodulin `$stateParams`-palvelun avulla. Esimerkiksi jos käyttäjä siirtyy tapahtumalistaussivulta osoitteeseen `/events/1234`, `$stateParams` palvelu erottaa URL-osoitteesta arvon `1234`, jonka perusteella tehdään palvelimelle kysely osoitteeseen `/api/events/1234`. Palvelin palauttaa jälleen datan JSON-formaatissa (Kuvio 33). `Events.get()`-metodin toisena parametrina olevassa funktiossa palvelimen palauttama data tallennetaan `$scope.event`-muuttujaan.

Lopuksi yksittäisen tapahtuman HTML-templeen (`view.html`) lisätään tarvittava HTML-sisältö, direktiivit ja lausekkeet (Ks. kuvio 34). Aluksi sivulle lisätään `ng-controller`-direktiivi, jolla kerrotaan että `EventsController`-kontrolleri hallinnoi kyseistä sivua. Tämän jälkeen `$scope.findOne()`-funktia kutsutaan `ng-init`-direktiivillä. `Ng-init`-direktiivin avulla `$scope.findOne()`-funktio käynnistää tapahtumakyselyn välittömästi, kun käyttäjä siirtyy yksittäisen tapahtuman sivulle. Lopuksi `$scope.event`-objektin data esitetään käyttäjälle AngularJS:n lausekkeiden avulla. Tapahtuman päivämäärän muo-
toilussa käytetään AngularJS:n sisäänrakennettua `date`-suodatinta.

```

103
104
105
106
107
108
109
110
111
112
113
114
115

```

```

/**
 * Find one event
 */
$scope.findOne = function() {
  Events.get({eventId: $stateParams.eventId},
    function(event) {
      $scope.event = event;
    });
};

```

Kuvio 32. Sovelluslogiikkaa yksittäisen tapahtuman esittämistä varten sovelluksen kontrollerissa. (`controllers.js`)

```

1 {
2   "_id": "54d39c4d1b6657262063829a",
3   "title": "Suomalais-venäläinen satutuokio",
4   "category": "Kirjasto",
5   "street": "Suursuonlaita 6",
6   "zip": "00630",
7   "city": "Helsinki",
8   "description": "<p> Satutuokiossa luetaan... Maunulan kirjasto.</p>",
9   "venue": "Maunulan kirjasto",
10  "v": 0,
11  "end_time": "2015-02-18T17:00:00.000Z",
12  "start_time": "2015-02-18T16:15:00.000Z",
13  "created": "2015-02-05T16:37:33.402Z"
14 }
15

```

Kuvio 33. Esimerkki web-palvelimen palauttamasta yksittäisen tapahtuman datasta JSON-formaatissa.

```

1 <div ng-controller="EventsController" ng-init="findOne()">
2   <section class="main">
3     <div class="container">
4       <div class="row">
5         <div class="col-md-12">
6
7           <div class="component back">
8             <a ui-sref="list"><i class="fa fa-chevron-circle-left"></i> Takaisin listaukseen</a>
9           </div>
10
11          <div class="component event">
12            <h1 class="component title">{{event.title}}</h1>
13            <div class="component description">
14              {{event.description}}
15            </div>
16            <div class="component info">
17              <ul>
18                <li><i class="fa fa-map-marker"></i> {{event.venue}}</li>
19                <li><i class="fa fa-calendar"></i> Alkaa: {{event.start_time | date:'EEE dd.MM.yyyy HH:mm'}}</li>
20                <li><i class="fa fa-calendar"></i> Päättyy: {{event.end_time | date:'EEE dd.MM.yyyy HH:mm'}}</li>
21                <li><i class="fa fa-tag"></i> {{event.category}}</li>
22                <li><i class="fa fa-flag"></i> {{event.street}}, {{event.zip}} {{event.city}}</li>
23              </ul>
24            </div>
25          </div>
26        </div>
27      </div>
28    </div>
29  </div>
30 </section>
31 </div>
32

```

Kuvio 34. Yksittäisen tapahtuman HTML-template (view.html).

3.6.3 Päivämäärän valinta

Tapahtumalistauksen päivämäärän valintaa varten kontrolleriin luodaan lisää sovelluslogiikkaa (Kuvio 35).

Aluksi kontrolleriin luodaan päivämäärävalikko varten `$scope.dates`-taulukko, johon lisätään for-loopin avulla päivämäärät seuraavan viikon ajalle. Päivämäärät muotoillaan haluttuun formaattiin AngularJS:n `$filters`-palvelun avulla.

Tämän jälkeen sovelluksen kontrolleriin luodaan kolme funktiota: `$scope.selectDate()`, `$scope.selectDateAll()` ja `$scope.selectDateFuture()`. `$scope.selectDate()`-funktio mää-

rittää toiminnallisuuden yksittäisen päivämäärän valintaa varten. Funktion ainoa parametri on käyttäjän valitsema päivämäärä. Funktiossa muokataan filters-objektin ominaisuuksia start_time ja end_time niin, että hakukriteerit vastaavat käyttäjän valitsemaa päivää. Tämän jälkeen funktiossa tyhjennetään \$scope.events-taulukko aiemmista tapahtumista ja asetetaan filters.offset-ominaisuudelle sen oletusarvo. Lopuksi funktiossa kutsutaan \$scope.find()-funktioita, jotka hakee palvelimelta tuoretta dataa, joka vastaa käyttäjän tekemiä valintoja. \$scope.selectDateAll() ja \$scope.selectDateFuture() toimivat samankaltaisesti kuin \$scope.selectDate(). Erona on, että funktioilla ei ole parametreja, vaan päivämäärät määritellään funktion sisällä.

Seuraavaksi tapahtumien listaussivulle lisätään HTML-sisältö päivämäärävalikolle varattuun paikkaan (Kuvio 36). \$scope.dates-taulukon sisältämät päivämäärät esitetään sovelluksen näkymässä ng-repeat-direktiivin avulla. Päivämäärien lisäksi valikkoon lisätään kohdat "Kaikki" ja "Myöhemmät", joiden avulla käyttäjä voi valita tapahtumia yksittäisen päivämäärän sijaan pidemmältä ajalta. Päivämäärävalikkoon luodaan myös ng-init, ng-class ja ng-click-direktiivien avulla mekaniikka, joka merkitsee 'active'-CSS-luokan käyttäjän tekemään aktiiviseen valintaan. Ng-click-direktiiveillä kutsutaan myös äsken luotuja funktioita, jotka hakevat palvelimelta tuoretta dataa.

```

41
42
43  /**
44   * Dates for date select
45   */
46   $scope.dates = [];
47
48   for(var i=0; i<7; i=i+1){
49     var date = new Date(Math.abs(new Date()) + (86400000 * i));
50     date = $filter('date')(date, 'yyyy-MM-dd');
51     $scope.dates.push(date);
52   }
53
54   /**
55   * Select date
56   */
57   $scope.selectDate = function(date) {
58     filters.start_time = date; // Selected date
59     filters.end_time = $filter('date')(new Date(Math.abs(new Date(date)) + 86400000), 'yyyy-MM-dd'); // The day after
60     filters.offset = 0;
61     $scope.events = [];
62     $scope.find();
63   };
64
65   $scope.selectDateAll = function() {
66     filters.start_time = $filter('date')(new Date(), 'yyyy-MM-dd'); // Today
67     filters.end_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'); // Next year
68     filters.offset = 0;
69     $scope.events = [];
70     $scope.find();
71   };
72
73   $scope.selectDateFuture = function() {
74     filters.start_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 7)), 'yyyy-MM-dd'); // Next week
75     filters.end_time = $filter('date')(new Date(Math.abs(new Date()) + (86400000 * 365)), 'yyyy-MM-dd'); // Next year
76     filters.offset = 0;
77     $scope.events = [];
78     $scope.find();
79   };

```

Kuvio 35. Sovelluslogiikkaa päivämäärän valintaa varten (controllers.js).

```

3  <!-- Select date -->
4  <section class="date-select">
5    <div class="container">
6      <div class="row">
7        <ul ng-init="activeDate='all';">
8          <li class="first" ng-class="{ 'active': activeDate=='all' }">
9            <div ng-click="activeDate='all'; selectDateAll();">
10              <abbr>&nbsp;</abbr>
11              <span><i class="fa fa-arrow-circle-q-right"></i> Kaikki</span>
12            </div>
13          </li>
14          <li ng-repeat="date in dates" ng-class="{ 'active': activeDate==date }">
15            <div ng-click="activeDate=date; selectDate(date);">
16              <abbr>{{date | date: 'EEE'}}</abbr>
17              <span ng-show="date==activeDate">Tänään</span>
18              <span ng-show="date!=activeDate">{{date | date: 'dd'}}</span>
19            </div>
20          </li>
21          <li class="last" ng-class="{ 'active': activeDate=='future' }">
22            <div ng-click="activeDate='future'; selectDateFuture();">
23              <abbr>&nbsp;</abbr>
24              <span><i class="fa fa-arrow-circle-q-right"></i> Myöhemmät</span>
25            </div>
26          </li>
27        </ul>
28      </div>
29    </div>
30  </section>
31  <!-- END Select date -->

```

Kuvio 36. Päivämäärän valinta (list.html).

3.6.4 Tapahtumakategorian valinta

Tapahtumien listausta käsittelevässä luvussa 3.6.1 palvelimelle muodostettiin kysely, jonka perusteella palvelin palautti tapahtumadataa (Ks. kuvio 28). Tapahtumadata sisälsi myös tietoa tapahtumakategorioista (Ks. kuvio 29.), jotka tallennettiin muuttujaan `$scope.categories`.

Seuraavaksi tapahtumien listaussivulla lisätään HTML-sisältö tapahtumakategoriavalikolle varattuun paikkaan (Kuvio 37). Kategoriavalikko luodaan iteroimalla `$scope.categories`-taulukon sisältämä kategoriadata `ng-repeat`-direktiivillä. `Ng-repeat`-direktiivin sisään määritetään yksittäisen tapahtumakategoriavalikon nimi ja kategoriata vastaavien tapahtumien lukumäärä. Kategoriavalikkoon luodaan myös `ng-init`, `ng-class` ja `ng-click` direktiivien avulla mekaniikka, joka lisää 'active'-CSS-luokka aktiivisena olevaan valintaan. Lisäksi valikossa käytetään `ng-disabled`-direktiiviä, joka lisää tapahtumakategorian `<button>`-elementtiin HTML:n `disabled`-attribuutin silloin kun kyseinen kategoria ei sisällä yhtään tapahtumaa. Lopuksi `ng-click`-direktiiviin lisätään `selectCategory()`-funktio, jonka avulla kategoriavalintaa vaihdetaan. Seuraavaksi `selectCategory()`-funktioille määritetään toiminnallisuus sovelluksen kontrollerissa (Kuvio 38). Toisin sanoen kontrolleriin luodaan `$scope.selectCategory()`-funktio, jonka ainoa parametri on käyttäjän valitsema kategoria. Funktiossa vaikutetaan `filters`-objektin ominaisuuteen `category` niin, että hakukriteerit vastaavat käyttäjän valitsemaa kategoriata. Tämän jäl-

keen funktiossa tyhjennetään `$scope.events`-taulukko aiemmista tapahtumista ja asetetaan `filters.offset`-ominaisuudelle sen oletusarvo. Lopuksi funktiossa kutsutaan `$scope.find()`-funktioita, joka hakee hakukriteereitä vastaavaa dataa palvelimelta.

```

33 <!-- Select category -->
34 <section class="category-select">
35   <div class="container">
36     <div class="row">
37       <div class="col-md-12">
38         <ul ng-init="activeCategory=0">
39           <li ng-repeat="category in categories" ng-class="{ 'active': activeCategory==index}">
40             <button ng-click="$parent.activeCategory=index; selectCategory(category.name)"
41               ng-disabled="category.count == 0">
42               {{category.name}} <span class="count">{{category.count}}</span>
43             </button>
44           </li>
45         </ul>
46       </div>
47     </div>
48   </div>
49 </section>
50 <!-- END Select category -->
51

```

Kuvio 37. Tapahtumakategorian valinta (list.html).

```

79
80 /**
81  * Select category
82  */
83 $scope.selectCategory = function(category) {
84   filters.category = category;
85   filters.offset = 0;
86   $scope.events = [];
87   $scope.find();
88 };
89

```

Kuvio 38. Sovelluslogiikkaa kategorian valintaa varten sovelluksen kontrollerissa. (controllers.js)

3.6.5 Sivutus

Tapahtumalistauksen sivutusta varten `EventsController`-kontrolleriin luodaan kaksi funktiota: `$scope.showMore()` ja `$scope.hasMore()` (Kuvio 39).

`$scope.showMore()`-funktiossa vaikutetaan `filters`-objektin `offset`-ominaisuuteen, joka määrittää minkä “sivun” tapahtumadatasta web-palvelin palauttaa. Funktiossa näytettävien tapahtumien määrää lisätään aina kahdellakymmenellä. Tämän jälkeen funktiossa kutsutaan `$scope.find()`-funktioita, joka suorittaa uuden kyselyn palvelimelle päivitettyillä hakukriteereillä. `$scope.showMore()`-funktioita kutsutaan tapahtumien listaussivun näkymässä `ng-click`-direktiivillä (Kuvio 40).

`$scope.hasMore`-funktion tehtävä on määrittää onko web-palvelimella lisää tapahtumia kullakin hakuehdolla. `$scope.hasMore()`-funktio saa arvon `true` tai `false` riippuen siitä

ylittääkö kaikkien tapahtumien lukumäärä kullakin hetkellä näytettävien tapahtumien lukumäärän. `$scope.hasMore()`-funktioita kutsutaan tapahtumien listaussivun näkymässä `ng-if`-direktiivillä (Kuvio 40). `Ng-if` piilottaa "Näytä lisää" -painikkeen sovelluksen näkymästä silloin kun valituilla hakukriteereillä ei enää ole näytettäviä tapahtumia.

Kuviot 41 ja 42 esittävät sivutukseen liittyvät komponentit selaimessa.

```

28
29
30  /**
31   * Pagination
32   */
33  $scope.showMore = function() {
34      filters.offset += 20;
35      $scope.find();
36  };
37
38  $scope.hasMore = function() {
39      if($scope.loading == false)
40          return $scope.events.length < $scope.total;
41  };

```

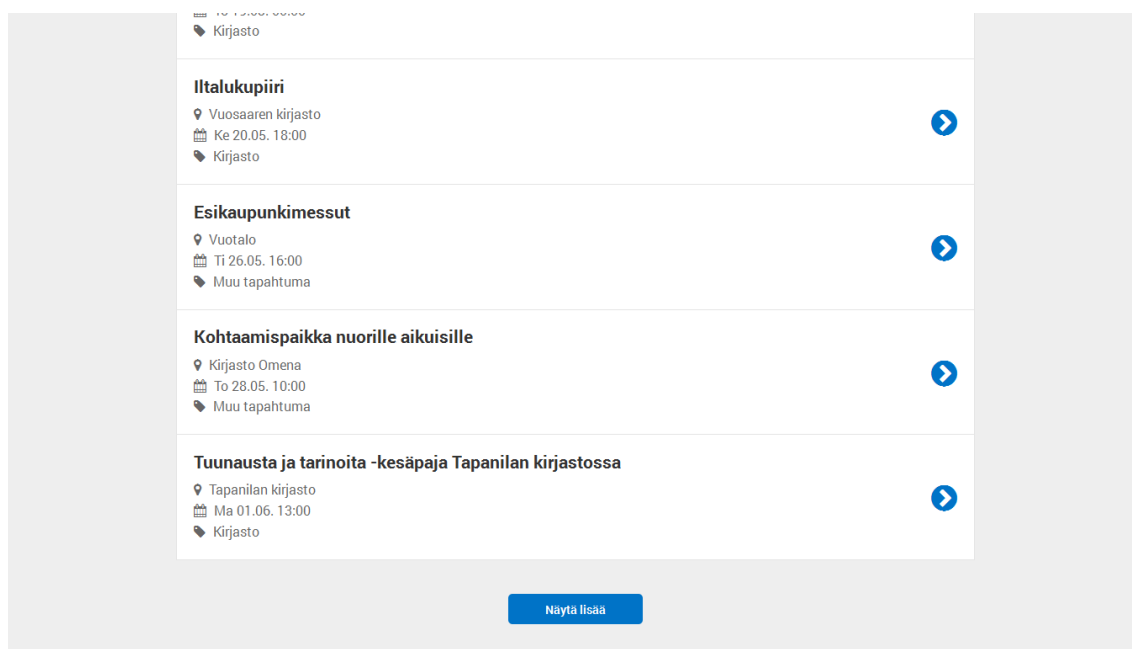
Kuvio 39. Sovelluslogiikkaa sivutusta varten sovelluksen kontrollerissa.

```

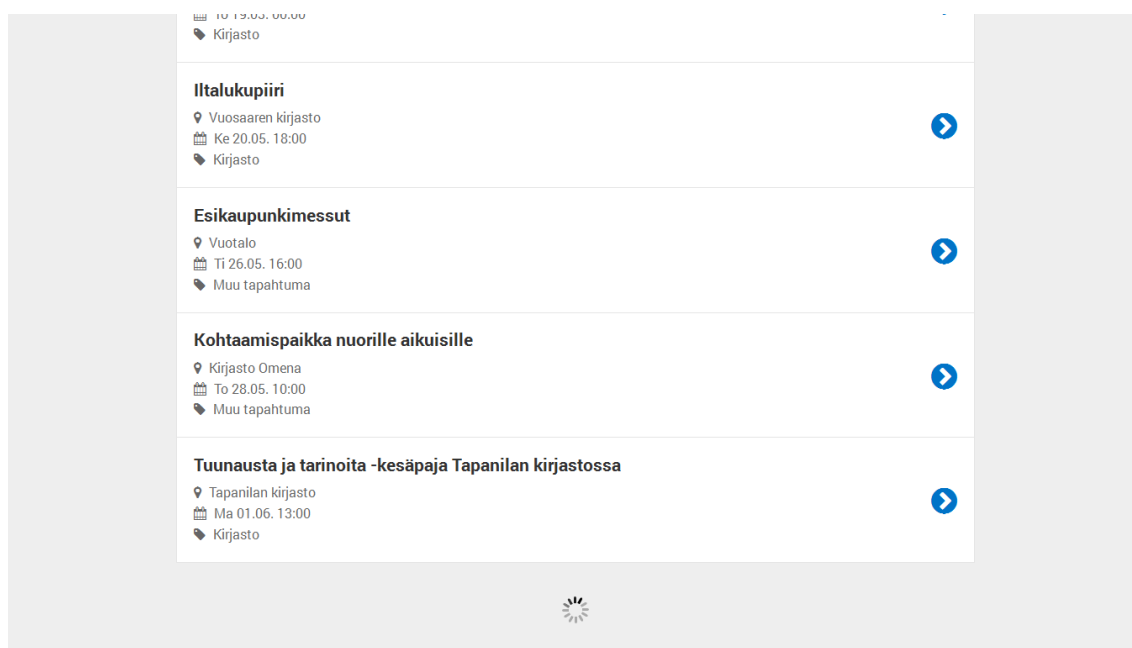
81
82  <!-- Pagination -->
83  <div class="loading" ng-if="loading">
84      
85  </div>
86
87  <div class="show-more" ng-if="hasMore()">
88      <button class="btn btn-secondary" ng-click="showMore()">Näytä lisää</button>
89  </div>
90  <!-- END Pagination -->

```

Kuvio 40. Sivutus (list.html).



Kuvio 41. ”Näytä lisää” -painike tapahtumien listauksessa.



Kuvio 42. Hakuanimaatio tapahtumien listauksessa.

4 Yhteenveto

Opinnäytetyössä tutkittiin mikä on AngularJS-JavaScript-ohjelmistokehys ja miten se soveltuu yksisivuisen web-sovelluksen käyttöliittymän toteutukseen.

Opinnäytetyön teoreettisessa osassa vastattiin tutkimuskysymykseen AngularJS:n olemuksesta tarkastelemalla AngularJS:n taustaa, käyttöönottoa sekä keskeisiä konsepteja ja sovelluskomponentteja.

AngularJS:n taustasta kerrottiin, että AngularJS on Googlen kehittämä avoimen lähdekoodin JavaScript-ohjelmistokehys, joka on suunniteltu pääasiassa yksisivuisten web-sovellusten kehittämiseen. AngularJS: käyttöönotosta kerrottiin, että AngularJS otetaan käyttöön lisäämällä AngularJS:n lähdekoodi sivulle ja kertomalla AngularJS:n ng-app direktiivillä, mikä osa sivusta on AngularJS:n hallinnassa.

AngularJS:ää määrittävistä konsepteista tarkastelun kohteeksi rajattiin MVC-arkkitehtuuri, kaksisuuntainen datakytkentä, riippuvuusinjektio ja direktiivi. Tarkasteltaviksi sovelluskomponenteiksi rajattiin moduuli, scope, lauseke, kontrolleri, suodatin, palvelu ja reititys. Konseptien ja sovelluskomponenttien toimintaa tarkasteltiin AngularJS:stä kirjoitetun kirjallisuuden, verkkoartikkeleiden sekä AngularJS:n virallisen dokumentaation pohjalta. Lisäksi konseptien ja sovelluskomponenttien toimintaa havainnollistettiin käytännössä lukuisten koodiesimerkkien avulla.

AngularJS:n taustan, käyttöönoton sekä keskeisten konseptien ja sovelluskomponenttien tarkastelun pohjalta hahmottui kokonaiskuva siitä mikä AngularJS on ja mitkä ominaisuudet AngularJS:ää määrittävät.

Opinnäytetyön toiminnallisessa osassa toteutettiin AngularJS:llä yksisivuisen web-sovelluksen käyttöliittymä. Toiminnallisen osa tavoitteena oli kuvailla miten yksisivuisen web-sovelluksen käyttöliittymän AngularJS:llä käytännössä tapahtuu. Toiminnallisen osan tarkoituksena oli luoda perusta, jonka pohjalta tässä luvussa vastataan tutkimuskysymykseen siitä miten AngularJS yksisivuisen web-sovelluksen käyttöliittymän kehitykseen soveltui.

Toiminnallisessa osassa toteutetun web-sovelluksen nimeksi annettiin Tapahtumia Helsingissä. Tapahtumia Helsingissä -web-sovelluksen idea oli listata tietoa Helsingin alueella järjestettävistä tapahtumista. Web-sovelluksen datalähteenä käytettiin avointa ohjelmointirajapintaa. Web-sovellus käsitti AngularJS:llä toteutetun käyttöliittymän lisäksi MongoDB-tietokannan, Node.js-web-palvelimen sekä ExpressJS:llä toteutetun ohjelmointirajapinnan.

Web-sovelluksen käyttöliittymän toteutus aloitettiin luomalla tarvittavat hakemistot ja tiedostot. Tämän jälkeen AngularJS otettiin web-sovelluksen käyttöön ja index.html-sivulle sisällytettiin tarvittavat JavaScript- ja CSS-tiedostot. Varsinainen sovelluskehitys aloitettiin luomalla moduuli, johon web-sovellus ja sen osat säilöttiin. Moduulin luomisen jälkeen toteutettiin web-sovelluksen reititys hyödyntämällä kolmannen osapuolen Angular UI Router -moduulia. Reitityksessä web-sovellukseen luotiin kolme reittiä web-sovelluksen etusivua, tapahtumien listausta ja yksittäisen tapahtuman esittämistä varten. Kullekin reitille luotiin oma HTML-template. Reitityksen jälkeen web-sovellukseen luotiin palvelu, jonka avulla sovelluksen palvelinkommunikointi organisoitiin omaksi kokonaisuudeksi. Palvelinkommunikointi toteutettiin käyttämällä AngularJS:n ngResource-moduulia. Lopuksi sovellukseen lisättiin kontrolleri, jossa luotiin sovelluksen tarvitsema sovelluslogiikka. Samalla sovelluksen näkymiin lisättiin tarvittava HTML-koodi sekä AngularJS:n direktiivit, lausekkeet ja suodattimet.

Toiminnallisen osan pohjalta seuraavaksi esitetään havaintoja ja päätelmiä siitä miten AngularJS yksisivuisen Tapahtumia Helsingissä -web-sovelluksen käyttöliittymän toteutukseen soveltui. Päätelmissä nostetaan esille AngularJS:n MVC-arkkitehtuuri, moduuli, palvelu, riippuvuusinjektio, aktiivisen kehittäjäyhteisön tuki, kolmannen osapuolen sovelluskomponentit, reititys, palvelinkommunikointi, HTML-template, lausekkeet, direktiivit ja suodattimet.

Ohjelmistoarkkitehtuurin ja koodin organisoinnin näkökulmasta AngularJS soveltui hyvin Tapahtumia Helsingissä -web-sovelluksen kehitykseen. Erityisesti AngularJS:n MVC-arkkitehtuuri, moduuli ja palvelu auttoivat sovelluksen rakenteen jäsentelyssä ja tekivät ohjelmistoarkkitehtuurista helposti ymmärrettävän.

AngularJS:n MVC-arkkitehtuuri toimi lähtökohtana Tapahtumia Helsingissä -web-sovelluksen ohjelmistoarkkitehtuurille. Tapahtumia Helsingissä -web-sovelluksessa toisistaan erotettiin käyttäjälle näkyvä osa, kontrollerin sisältämä sovelluslogiikka sekä \$scope-objektiin säilötty sovellusdata. AngularJS:n MVC-arkkitehtuuri tarjosi ennen kaikkea selkeän ajatusmallin, joka perusteella sovelluksen ohjelmistoarkkitehtuuri muotoutui jäsennellysti.

MVC-arkkitehtuurin lisäksi AngularJS:n moduuli osoittautui hyödylliseksi työkaluksi sovelluskoodin organisoimiseen. Tapahtumia Helsingissä -web-sovelluksessa moduu-

lin avulla kaikki sovelluskomponentit koottiin yhteen nimettyyn paikkaan, joka muodosti web-sovelluksesta yhden selkeän kokonaisuuden.

Ohjelmistorakenteen näkökulmasta myös AngularJS:n palvelut tarjosivat käytännöllisen tavan organisoida sovelluskoodia. Tapahtumia Helsingissä -web-sovelluksessa luotiin yksi oma palvelu, jonka avulla sovelluksen palvelinkommunikointi organisoitiin yhdeksi kokonaisuudeksi. Tämän jälkeen palvelua voitiin toistuvasti hyödyntää sovelluksen kontrollerissa.

Tapahtumia Helsingissä -web-sovelluksessa myös erilaisten sovelluskomponenttien riippuvuuksien hallinnointi onnistui vaivattomasti AngularJS:n riippuvuusinjektion kautta. AngularJS:n riippuvuusinjektio helpotti myös sovelluksen testausta. Kun jokin asia ei sovelluksessa toiminut, oli ongelman jäljittäminen yksinkertaista aloittaa eristämällä yksittäinen sovelluskomponentti pois sovelluksesta. Riippuvuusinjektio vähensi myös kirjoitettavan koodin määrää, koska sovelluskomponenttien riippuvuuksia ei AngularJS:ssä tarvitse erikseen määritellä.

Tapahtumia Helsingissä -web-sovelluksen kehitystyötä helpotti olennaisesti AngularJS:n ympärille muodostunut vilkas keskustelu web-kehitykseen keskittyneillä foorumeilla ja verkkosivustoilla. Aktiivisen kehittäjäyhteisön tuki helpotti ongelmanratkaisua ja tiedon hankintaa useissa Tapahtumia Helsingissä -web-sovelluksen kehitykseen liittyvissä ongelmatilanteissa. Myös AngularJS:n virallinen dokumentaatio tarjosi paljon tarpeellista informaatiota teknisiin kysymyksiin.

Aktiivisen kehittäjäyhteisön ansiosta saatavilla oli myös lukuisia AngularJS:lle kehitettyjä kolmannen osapuolen sovelluskomponentteja, jotka tarjosivat valmiita ratkaisuja kehitysongelmiin ja paikkasivat joitakin AngularJS:n puutteita. Tapahtumia Helsingissä -web-sovelluksessa käytettiin yhtä kolmannen osapuolen kehittämää sovelluskomponenttia – Angular UI Router -moduulia – joka osoittautui toimivaksi ratkaisuksi web-sovelluksen reitityksen toteuttamiseen. Vaikka Angular UI Router -moduulin kaikkia ominaisuuksia ei hyödynnetty Tapahtumia Helsingissä -web-sovelluksessa täysimittaisesti, mahdollisti moduuli monipuolisen reitityskokonaisuuden. Kaiken kaikkiaan kolmannen osapuolen sovelluskomponentit olivat merkittävä lisä AngularJS:ään.

Ylipäättään reitityksen huomioiminen oli yksi merkittävä tekijä, joka teki AngularJS:stä soveltuvan ohjelmistokehyksen Tapahtumia Helsingissä -web-sovelluksen toteuttami-

seen. Tapahtumia Helsingissä -web-sovelluksen reitityksen toteuttaminen esimerkiksi tavallisella JavaScriptilla olisi ollut huomattavasti vaivalloisempaa kuin AngularJS:llä toteutettu reititys. Tapahtumia Helsingissä -web-sovelluksessa reititys luotiin nopeasti assosioimalla haluttu näkymä haluttuun URL-osoitteeseen.

AngularJS soveltui ohjelmistokehyksenä hyvin myös palvelinpuolen ohjelmointirajapintojen kanssa kommunikointiin. Vaikka Tapahtumia Helsingissä -web-sovelluksessa kommunikointiin palvelimen ohjelmointirajapinnan kanssa vain HTTP GET -kutsuilla, osoittautui web-sovelluksessa käytetty ngResource-moduuli hyödylliseksi sovelluskomponentiksi, joka tarjosi kätevän abstraktion ohjelmistorajapinnan kanssa kommunikointiin. Voidaan sanoa, että ngResource-moduulin hyödyt olisivat osoittautuneet entistä hyödyllisemmiksi sovelluksessa, joissa palvelimen kanssa olisi kommunikoitu monipuolisemmin. Tapahtumia Helsingissä -web-sovelluksen perusteella voidaan kuitenkin sanoa, että AngularJS:ssä palvelinkommunikointi on huomioitu tavalla, joka soveltuu hyvin yksisivuisen web-sovelluksen kehittämiseen.

Tapahtumia Helsingissä -web-sovelluksessa myös käyttöliittymän kuvailu onnistui vaittomasti AngularJS:n avulla. Käyttöliittymän kuvailu oli Tapahtumia Helsingissä -web-sovelluksessa erityisen helppoa, koska AngularJS:ssä näkymät kuvaillaan käyttämällä tavallista HTML:ää.

AngularJS:n direktiivit olivat erinomainen tapa lisätä toiminnallisuutta suoraan HTML:ään. Tapahtumia Helsingissä -web-sovelluksessa esimerkiksi ng-click- tai ng-show-direktiivien avulla käyttöliittymään saatiin lisättyä haluttu toiminnallisuus kirjoittamatta riviäkään koodia. Direktiivien ansiosta käyttöliittymästä tuli myös varsin luettava.

Myös HTML:ään upotettavat lausekkeet olivat selkeä tapa kertoa mitä dataa käyttöliittymään halutaan yhdistää. Lausekkeet olivat helposti ymmärrettävä keino esittää esimerkiksi yksittäisen tapahtuman data Tapahtumia Helsingissä -web-sovelluksen käyttöliittymässä. Myös lausekkeet tekivät käyttöliittymästä luettavan.

Lausekkeiden muotoilu suodattimien avulla oli myös selvä etu Tapahtumia Helsingissä -web-sovelluksen kehityksessä. Esimerkiksi tapahtumien päivämäärien muotoilu JavaScriptistä ymmärrettävään muotoon onnistui AngularJS:n suodattimilla yksinkertaisesti.

Tiivistetysti voidaan sanoa, että AngularJS soveltui yksisivuisen web-sovelluksen käyttöliittymän toteutukseen erittäin hyvin. Käytetyt sovelluskomponentit olivat tarkoituksenmukaisia ja sopivat hyvin yksisivuisen web-sovelluksen kehitystarpeisiin. Ohjelmistorakenne muotoutui AngularJS:n avulla jäsennellysti ja käyttöliittymän näkymien kuvailu oli helppoa ja ymmärrettävää. Aktiivinen kehittäjäyhteisö ja AngularJS:n dokumentaatio tarjosivat riittävästi tietoa kehityskysymysten selvittämiseen. AngularJS ei aiheuttanut mainittavia vaikeuksia missään Tapahtumia Helsingissä -web-sovelluksen kehitysvaiheessa. AngularJS:n soveltuvuuden kannalta olennaisia asioita olivat AngularJS:n MVC-arkkitehtuuri, moduuli, palvelu, riippuvuusinjektio, aktiivisen kehittäjäyhteisön tuki, kolmannen osapuolen sovelluskomponentit, reititys, palvelinkommunikointi, HTML-templatet, direktiivit, lausekkeet ja suodattimet.

Lähteet

AngularUI Router 2015. GitHub. [verkkosivu] <<https://github.com/angular-ui/ui-router>> (luettu 6.2.2015)

API: \$compile. AngularJS. [verkkosivu]
<[https://docs.angularjs.org/api/ng/service/\\$compile](https://docs.angularjs.org/api/ng/service/$compile)> (luettu 12.2.2015)

API: \$http. AngularJS. [verkkosivu] <[https://docs.angularjs.org/api/ng/service/\\$http](https://docs.angularjs.org/api/ng/service/$http)> (luettu 11.2.2015)

API: \$location. AngularJS. [verkkosivu]
<[https://docs.angularjs.org/api/ng/service/\\$location](https://docs.angularjs.org/api/ng/service/$location)> (luettu 27.3.2015)

API: \$resource. AngularJS. [verkkosivu]
<[https://docs.angularjs.org/api/ngResource/service/\\$resource](https://docs.angularjs.org/api/ngResource/service/$resource)> (luettu 11.2.2015)

API: \$routeProvider. AngularJS. [verkkosivu]
<[https://docs.angularjs.org/api/ngRoute/provider/\\$routeProvider](https://docs.angularjs.org/api/ngRoute/provider/$routeProvider)> (luettu 1.4.2015)

API: filter components in ng. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/api/ng/filter>> (luettu 12.2.2015)

API: ng. AngularJS. [verkkosivu] <<https://docs.angularjs.org/api/ng#directive>> (luettu 12.2.2015)

API: ngRoute. AngularJS. [verkkosivu] <<https://docs.angularjs.org/api/ngRoute>> (luettu 27.1.2015)

API: service components in ng. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/api/ng/service>> (luettu 12.2.2015)

Bogucki, Katelyn 2014. AngularJS: One Framework to Rule Them All?. Huffington Post. [verkkosivu] <http://www.huffingtonpost.com/2014/03/24/angularjs-one-framework-t_n_5022395.html> (luettu 27.1.2015)

Branas, Rodrigo 2014. AngularJS Essentials: Design and construct reusable, maintainable, and modular web applications with AngularJS. Packt Publishing.

Data Binding Overview. Microsoft Developer Network. [verkkosivu]
<[https://msdn.microsoft.com/en-us/library/ms752347\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx)> (luettu 9.3.2015)

Developer Guide: Controllers. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/controller>> (luettu 27.1.2015)

Developer Guide: Data Binding. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/databinding>> (luettu 27.1.2015)

Developer Guide: Dependency Injection. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/di>> (luettu 12.2.2015)

Developer Guide: Directives. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/directive>> (luettu 27.1.2015)

Developer Guide: Expressions. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/expression>> (luettu 22.3.2015)

Developer Guide: Filters. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/filter>> (luettu 27.1.2015)

Developer Guide: i18n and l10n. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/i18n>> (luettu 13.2.2015)

Developer Guide: Modules. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/module>> (luettu 26.3.2015)

Developer Guide: Providers. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/providers>> (luettu 12.2.2015)

Developer Guide: Scopes. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/scope>> (luettu 27.1.2015)

Developer Guide: Services. AngularJS. [verkkosivu]
<<https://docs.angularjs.org/guide/services>> (luettu 12.2.2015)

Diez, Frederik 2013. AngularJS Succintly. Syncfusion.

Esteva, Santiago 2013. AngularJS 2.0 Status and Preview. ng-learn. [verkkosivu]
<<http://ng-learn.org/2014/03/AngularJS-2-Status-Preview/>> (luettu 27.1.2015)

Freeman, Adam 2014. Pro AngularJS. Apress.

GitHub 2015a. angular/angular.js. GitHub. [verkkosivu]
<<https://github.com/angular/angular.js>> (luettu 13.2.2015)

GitHub 2015b. Releases angular/angular.js. GitHub. [verkkosivu]
<<https://github.com/angular/angular.js/releases>> (luettu 13.2.2015)

Green, Brad & Seshadri, Shyam 2013. AngularJS. Sebastopol: O'Reilly Media.

Horsmalahiti, Panu 2014. AngularJS ja datakytkentä. Cybercom. [verkkosivu]
<<http://www.cybercom.com/fi/Suomi/Yritys/Blogit/Blogit/AngularJS-ja-datakytkenta/>> (luettu 9.3.2015)

Janssen, Cory 2015a. Data Binding. [verkkosivu]
<<http://www.techopedia.com/definition/15652/data-binding>> (luettu 22.3.2015)

Janssen, Cory 2015b. Model View Controller (MVC). Technopedia. [verkkosivu]
<<http://www.techopedia.com/definition/3842/model-view-controller-mvc>> (luettu 21.3.2015)

Krill, Paul 2013. What's so special about Google's AngularJS. InfoWorld. [verkkosivu]
<<http://www.infoworld.com/article/2612801/javascript/what-s-so-special-about-google-s-angularjs.html>> (luettu 27.1.2015)

Kozlowski, Pavel & Darwin, Peter Bacon 2013. Mastering Web Application Development with AngularJS: Build single-page web applications using the power of AngularJS. Packt Publishing.

Lau, Dmitri 2013. 10 Reasons Why You Should Use AngularJS. SitePoint. [verkkosivu] <<http://www.sitepoint.com/10-reasons-use-angularjs/>> (luettu 26.3.2015)

Lerner, Ari 2013. ng-book: The Complete Book on AngularJS. Fullstack.io.

Linked Events. HKI Linked Events API. [verkkosivu] <<http://api.hel.fi/linkedevents/v0.1/>> (luettu 15.4.2015)

Markov, Danny 2015. Making a Single Page Application Without a Framework. Tutorialzine. [verkkosivu] <<http://tutorialzine.com/2015/02/single-page-app-without-a-framework/>> (luettu 27.3.2015)

Osmani, Addy 2012. Understanding MVVM - A Guide For JavaScript Developers. Addy's eng articles. [verkkosivu] <<http://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/>> (luettu 21.3.2015)

Panda, Sandeep 2014. AngularJS: Novice to Ninja. SitePoint.

Seshadri, Shyam & Green, Brad 2014. AngularJS: Up and Running. Sebastopol: O'Reilly Media.

Sevilleja, Chris 2014. AngularJS Routing Using UI-Router. scotch.io. [verkkosivu] <<http://scotch.io/tutorials/javascript/angular-routing-using-ui-router>> (luettu 6.2.2015)

W3 Schools 2015a. AngularJS Expressions. [verkkosivu] <http://www.w3schools.com/angular/angular_expressions.asp> (luettu 22.3.2015)

W3 Schools 2015b. AngularJS Filters. [verkkosivu] <http://www.w3schools.com/angular/angular_filters.asp> (luettu 27.3.2015)

Kuviolähteet

Kuvio 1. AngularJS:n, Knockout.js:n, Ember.js:n ja Backbone.js:n suosion kehitys Google-hauilla mitattuna vuosina 2010–2014.

Google Trends 2015. <<http://www.google.com/trends/explore?hl=en-US#q=AngularJS%2C%20Knockout.js%2C%20Ember.js%2C%20Backbone.js&date=1%2F2011%2049m&cmpt=q&tz=>>> (luettu 11.2.2015)

