**Miika Kujala**

# Yksisivuisten web-sovellusten kehittäminen Angular 2 -sovelluskehyksellä

Tietotekniikan kandidaatintutkielma

17. toukokuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

**Tekijä:** Miika Kujala

**Yhteystiedot:** miika.m.kujala@student.jyu.fi

**Ohjaaja:** Tytti Saksa

**Työn nimi:** Yksisivuisten web-sovellusten kehittäminen Angular 2 -sovelluskehyksellä

**Title in English:** Developing single-page applications with Angular 2 framework

**Työ:** Kandidaatintutkielma

**Sivumäärä:** 22+0

**Tiivistelmä:** Yksisivuiset web-sovellukset ovat yleistyneet viime vuosina. Niiden kehityksessä hyödynnetään usein JavaScript-sovelluskehystä. Angular 2 on Google:n kehittämä JavaScript-sovelluskehys. Tämän tutkielman tavoitteena on tarkastella Angular 2 -sovelluskehystä ja sen soveltuvuutta yksisivuisten web-sovellusten kehityksessä. Tutkielmassa käydään läpi Angular 2 -sovelluskehyksen ominaisuuksia sekä Angular 2 -sovelluskehyksen käytössä ilmeneviä etuja ja haittoja.

**Avainsanat:** Angular 2 -sovelluskehys, yksisivuinen web-sovellus, web-komponentit

**Abstract:** Single-page applications have become more popular in recent years. JavaScript framework is often used to help on developing work. Angular 2 is JavaScript framework developed by Google. The aim of this thesis is to research Angular 2 framework and it's aptitude for developing a single-page applications. This thesis also presents features of Angular 2 framework and pros and cons of using Angular 2 framework.

**Keywords:** Angular 2 framework, single-page application, web components

# Termiluettelo

| | |
|---|---|
| Ajax | Ajax on lyhenne sanoista Asynchronous JavaScript and XML. Sen avulla voidaan siirtää pieniä määriä dataa selaimen ja palvelimen välillä. Se mahdollistaa verkkosivujen päivityksen ilman koko verkkosivun uudelleen lataamista. |
| DOM | Dokumenttioliomalli (engl. Document Object Model, DOM) on ohjelmointirajapinta HTML-, XML- ja SVG -dokumenteille. Se kuvaa dokumentin elementtihierarkian puurakenteena. DOM:n avulla voidaan muokata dokumentin rakennetta, sisältöä ja tyylejä. |
| JSON | JSON on lyhenne sanoista JavaScript Object Notation. Se on tiedostomuoto, jota käytetään tiedonsiirrossa. |

# Kuviot

# Sisältö

# 1 Johdanto

Yksisivuiset web-sovellukset (engl. single-page application, SPA) ovat yleistyneet viime vuosien aikana. Suuri syy yksisivuisten web-sovellusten kasvuun on niiden huomattavasti parempi käyttökokemus verrattuna perinteisiin web-sovelluksiin, jotka rakentuvat useasta eri HTML-sivusta. Yksisivuisia web-sovelluksia on nopeampi ja sujuvampi käyttää, koska niiden yhteydessä ei tule enää turhia sivulatauksia (Fink & Flatow 2014, s. 13). Yksisivuisten web-sovellusten kehityksen helpottamiseksi on ilmestynyt paljon työkaluja. Tärkeä osa yksisivuisten web-sovellusten kehitystä on selainpäässä käytettävät JavaScript-sovelluskehykset (engl. framework), kuten esimerkiksi Angular 2.

Tämän tutkielman tavoitteena on tarkastella Angular 2 -sovelluskehystä ja sen soveltuvuutta yksisivuisten web-sovellusten kehityksessä. Angular 2 on Googlen kehittämä ja ylläpitämä JavaScript-sovelluskehys joka helpottaa yksisivuisten web-sovellusten kehittämistä. Angular 2 on uusi versio suositusta AngularJS:sta, joka on yksi suosituimmista sovelluskehyksistä (Oscariza Jr, Pattabiraman ja Mesbah 2015).

Tutkielman luvussa 2 käsitellään yleisesti web-sovelluksia ja niiden toimintaa. Luku 3 esittelee web-komponentteja ja niiden tarjoamia mahdollisuuksia. Luvussa 4 käsitellään tarkemmin Angular 2 -sovelluskehystä.

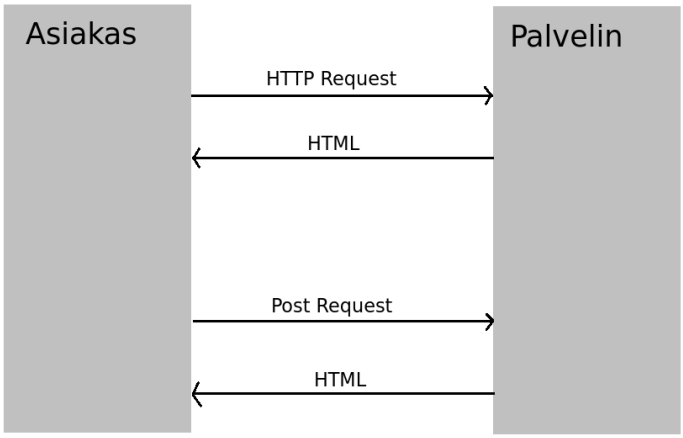
Tutkielmasta rajattiin pois Angular 2 -sovelluskehystä edeltävän version AngularJS:n käsittely, sekä versioiden välisten erojen esittäminen. Kirjoitushetkellä (17.3.2016) Angular 2 -sovelluskehyksestä on käytössä versio 2.0.0-beta.9.

# 2 Teoriaa web-sovelluksista

Tässä luvussa tarkastellaan perinteisen web-sovelluksen toimintaa sekä yksisivuisen web-sovelluksen toimintaa, ja näiden käytössä ilmeneviä eroja. Lisäksi käsitellään web-sovelluksien arkkitehtuureja ja sovelluskehyksen tarpeellisuutta.

## 2.1 Perinteinen web-sovellus

Perinteisessä web-sovelluksessa sivu ladataan selaimessa näkyviin siten, että selain tekee pyynnön (engl. HTTP Request) palvelimelle ja palvelin vastaa pyyntöön (engl. HTTP Response) lähettämällä HTML-dokumentin selaimelle (Fink ym. 2014, s. 7). Käyttäjän vuorovaikutuksesta, esimerkiksi käyttäjän lähettäessä sähköistä lomaketta, selain tekee uuden pyynnön (engl. Post Request) palvelimelle, ja palvelin vastaa lähettämällä uuden HTML-dokumentin selaimelle. Sivun päivityksessä vanha sivu korvataan kokonaan uudella sivulla. Kuviossa 1 esitetään miten asiakkaan ja palvelimen välinen tiedonsiirto toimii perinteisessä web-sovelluksessa.




Kuvio 1. Perinteisen web-sovelluksen tiedonsiirto

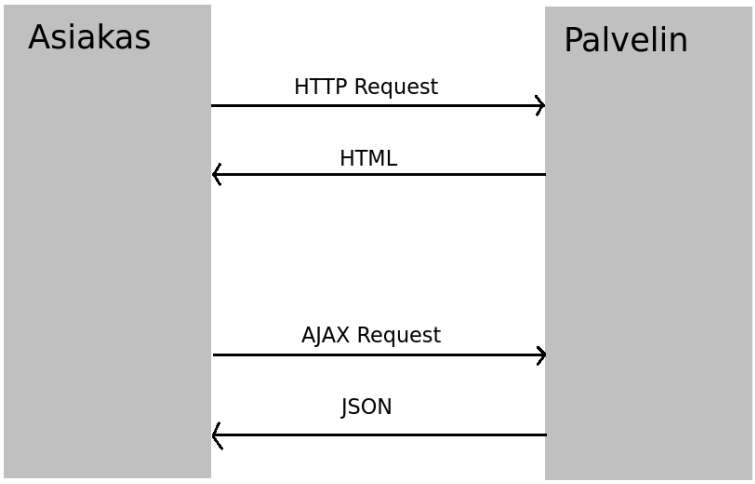
Freeman (2014, s. 45) kuvaa HTML-dokumentin uudelleen lataamisen olevan hidasta ja siten heikentävän käyttökokemusta. Mikowski ja Powell (2014, s. 20) esittävät uudelleen lautauksen aiheuttavan välähdyksen selaimessa. Kiireinen palvelin tai hidas internet-yhteys voivat pahimmillaan pitkittää välähdyksen usean sekun-

nin mittaiseksi.

Perinteisessä web-sovelluksessa selaimen rooli on esittää HTML-dokumentin sisältö (Freeman 2014, s. 45). Web-sovelluksen toimintalogiikka, data ja tila sijaitsevat palvelimella (Fink ym. 2014, s. 8).

## 2.2 Yksisivuinen web-sovellus

Yksisivuinen web-sovellus on nimensä mukaisesti sovellus, jossa on vain yksi sisällöltään muuttuva sivu. Yksisivuisessa web-sovelluksessa HTML-dokumentti ladataan selaimeen vain kerran. Toisin kuin perinteisessä web-sovelluksessa, HTML-dokumenttia ei enää käytön aikana korvata tai ladata kokonaan uudelleen (Freeman 2014, s. 46). Kuviossa 2 esitellään miten asiakkaan ja palvelimen välinen tiedonsiirto toimii yksisivuisessa web-sovelluksessa. Kuten Fink ym. (2014, s. 12) esittävät yksisivuisen ja perinteisen web-sovelluksen eroa, alkuperäisen HTML-dokumentin latauksen jälkeen yksisivuinen web-sovellus pyytää dataa palvelimelta Ajax:n avulla ja vastaus lähetetään JSON -muodossa.



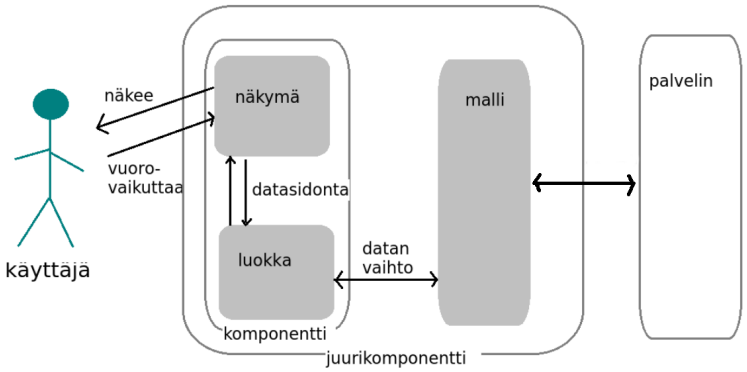

Kuvio 2. Yksisivuisen web-sovelluksen tiedonsiirto

Yksisivuinen web-sovellus tarjoaa merkittävästi paremman käyttökokemuksen kuin perinteinen web-sovellus (Mikowski ym. 2014, s. 20). Koska suurin osa toimintalogiikasta sijaitsee selaimessa, yksisivuisen web-sovelluksen käyttäminen on todella nopeaa ja sen jouheva toiminta muistuttaakin suuresti työpöytäsovelluksen käyt-

töä. Yksisivuinen web-sovellus toimii myös offline -tilassa, koska sovelluksen tila on tallennettu selaimen muistiin (Fink ym. 2014, s. 12). Mikowski ym. (2014, s. 20) toteavat vain datan validoinnin, autentikoinnin ja tietokantojen käyttämisen jäävän palvelinpään tehtäviksi.

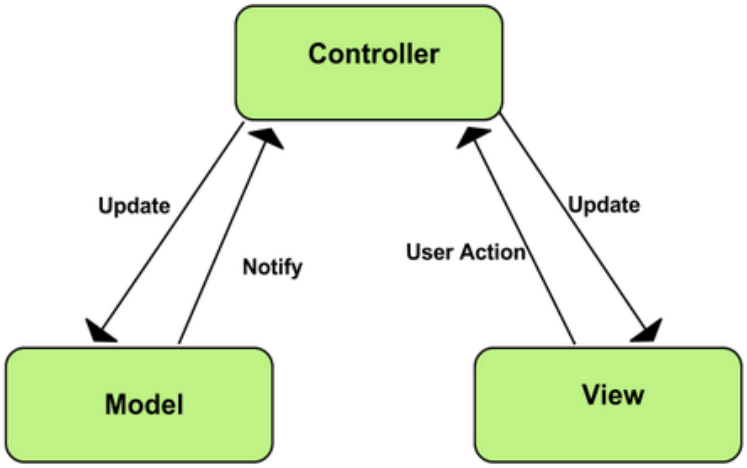
## 2.3 Web-sovelluksen arkkitehtuuri

Koska toteutukset eri web-sovellusten välillä ovat usein hyvin erilaisia, ei yhden ja saman suunnittelumallin käyttö ole aina paras vaihtoehto (Lerner ,Coury, Murray & Taborda 2015, s. 153). Suunnittelumalleja on olemassa useita erilaisia kuten esimerkiksi MVC-arkkitehtuuri, MVVM-arkkitehtuuri ja Flux. Angular 2 ei vaadi käyttämään mitään tiettyä suunnittelumallia, vaan mahdollistaa usean eri suunnittelumallin käytön. Joustavuuden ansiosta Angular 2 sopiikin käytettäväksi lähes jokaiseen web-projektiin. Usean suunnittelumallin mahdollisuus voi tosin johtaa myös huonoihin valintoihin. Kuviossa 3 esitetään esimerkki Angular 2 -sovelluksen arkkitehtuurista.




Kuvio 3. Esimerkki Angular 2 -sovelluksen arkkitehtuurista

Eräs ohjelmistokehityksessä yleisimmin käytettävistä suunnittelumalleista on Malli-Näkymä-Ohjain-arkkitehtuuri (engl. Model-View-Controller, MVC). MVC-arkkitehtuurin idea on jakaa erillisiin osiin sovelluksen data, toimintalogiikka ja näkymä. Ohjain kontrolloi ja välittää dataa mallin ja näkymän välillä. Freeman (2014, s. 47) kertoo MVC-arkkitehtuurin helpottavan sovelluksen kehittämistä, ylläpitämistä ja testaamista. Kuviossa 4 esitellään MVC-arkkitehtuurin rakenne.




Kuvio 4. Esimerkki MVC-arkkitehtuurista (Chrome 2016)

MVC-arkkitehtuurissa malli pitää sisällään sovelluksen käyttämän datan. Mallissa voidaan luoda, hallita ja muokata dataa. Näkymä ja ohjain ovat riippuvaisia mallista, mutta malli ei ole riippuvainen näkymästä tai ohjaimesta (Zhang ja Zhu 2013). Ohjain toimii linkkinä mallin ja näkymän välillä. Ohjain välittää näkymältä tulevat käyttäjän syötteet mallille. Mallissa tapahtuvien muutosten jälkeen ohjain päivittää muutokset näkymään. Näkymä sisältää käyttöliittymän, jonka käyttäjä näkee. Näkymä pyytää ohjaimen kautta mallilta dataa, jonka se esittää käyttöliittymässä.

Malli-Näkymä-Näkymämalli-arkkitehtuuri (engl. Model-View-ViewModel, MVVM) perustuu MVC-arkkitehtuuriin. MVVM-arkkitehtuuri on suunniteltu erottamaan näkymä ja malli kokonaan toisistaan (Cortez ja Vazhenin 2015). MVVM-arkkitehtuuri hyödyntää datasidontaa näkymän ja näkymämallin välillä (Li 2015).

## 2.4 Sovelluskehyksen hyöty

JavaScript-sovelluskehykset on tehty helpottamaan web-sovellusten kehittämistä. Sovelluskehykset auttavat esimerkiksi manipuloimaan DOM:a, tarjoavat valmiiksi kirjoitettuja kirjastoja sekä toiminnallisuuksia ja auttavat jäsentämään rakennetta.

Kuten Fain ja Moiseev (2015, s. 2) esittävät, sovelluskehyksen käyttö ei ole pakollista. Web-sovellusten kehittäminen onnistuu ilman sovelluskehystä JavaScript:n avul-

la. Jos sovelluskehystä ei käytetä, mahdollistaa se kehittäjälle enemmän vapautta toteutuksen suhteen. Ongelmia kuitenkin alkaa ilmetä, mikäli web-sovelluksen koko kasvaa (Aden, Aden & Wilken 2016, s. 2). Tällöin esimerkiksi datan hallinnasta tulee monimutkaista ja käyttäjän vuorovaikutuksiin reagoiminen menee sekavaksi. Myös kehityksessä käytettävä työmäärä kasvaa, jos web-sovellus optimoidaan toimimaan useilla eri selaimilla (Fain & Moiseev 2015, s. 2). Aden ym. (2016, s. 2) kuvaavat jokaisen kehittäjän törmäävän edellä mainittuihin vaikeuksiin.

Angular 2 -sovelluskehyksen avulla voidaan ratkaista aiemmin mainittuja ongelmia. Angular 2 -sovelluskehyksen avulla voidaan muun muassa yksinkertaistaa omien komponenttien luominen ja niiden lisääminen osaksi HTML-dokumenttia. Lisäksi Angular 2 -sovelluskehys mahdollistaa monipuolisen datasidonnan, riippuvuusinjektion käytön ja tukee modularisointia sekä tarjoaa reititysmekanismin (Fain & Moiseev 2015, s. 3). Angular 2:n ominaisuuksia käsitellään tarkemmin luvussa 4.

# 3 Web-komponentit -standardi

Tässä luvussa käsitellään web-komponentin ominaisuuksia sekä Web-komponentit -standardiin kuuluvien teknologioiden ominaisuuksia. Luvussa esitellään myös miten Angular 2 -sovelluskehyksessä hyödynnetään näitä edellä mainittuja ominaisuuksia.

## 3.1 Web-komponentti

Web-komponentti on web-sovelluksessa itsenäisesti toimiva osa. Web-komponentteja kehitetään HTML-, CSS- ja JavaScript -teknologioiden avulla. Olemassa oleva web-komponentti voidaan ottaa helposti käyttöön lisäämällä se HTML-dokumenttiin. Yksi web-komponenttien tärkeimmistä ominaisuuksista on olla uudelleen käytettäviä (Mozilla Developer Network 2016). Toinen tärkeä ominaisuus on estää luotua komponenttia vaikuttamasta ei-toivotulla tavalla sovelluksen muihin osiin, eli toisin sanoen komponentin toimintalogiikka kapseloidaan sen sisään (Aden ym. 2016, s. 7).

Angular 2 on suunniteltu siten, että web-komponenttien käyttö yhdessä Angular 2 -komponenttien kanssa olisi helppoa (Exbrayat 2016, s. 38). Angular 2 -komponentit ovat hyvin samankaltaisia kuin web-komponentit. Angular 2 käyttää selaimissa hyväksi web-komponenttien toimintaan suunniteltuja ominaisuuksia, kuten esimerkiksi Shadow DOM:n tukea, joka esitellään jäljempänä (Aden ym. 2016, s. 10).

## 3.2 Web-komponentit -standardiin kuuluvat teknologiat

Web-komponentit -standardi mahdollistaa itsenäisesti toimivien HTML-elementtien luomisen. Web-komponentit -standardi muodostuu seuraavista neljästä olemassa olevasta teknologiasta (Exbrayat 2016, s. 38):

- Custom elements -standardi
- Shadow DOM -standardi

- HTML Templates -standardi
- HTML imports -standardi.

Näitä kaikkia neljää Web-komponentit -standardin teknologiaa voidaan käyttää myös itsenäisesti. Kaikki selaimet eivät vielä täysin tue Web-komponentit -standardin teknologioita. Kirjoitushetkellä (10.3.2016) selaimista vain Chrome ja Opera tukevat kaikkia Web-komponentit -standardin teknologioita (webcomponents.org 2016).

## 3.3 Custom elements -standardi

Custom Elements -teknologian avulla voidaan luoda omia HTML-elementtejä. Luoduilla HTML-elementeillä voi olla oma ohjelmoitu toiminnallisuus ja tyylimuotoilu. Esimerkiksi `<oma-komponentti> </oma-komponentti>` on täysin validi HTML-elementti. Luotu elementti voidaan rekisteröidä DOM:iin helposti JavaScript:n avulla (Exbrayat 2016, s. 39). Elementille on mahdollista määritellä oma käyttöliittymä (engl. user interface) (Savage 2015).

Angular 2 hyödyntää web-sovelluksissa Custom Elements -teknologiaa. Sen avulla voidaan Angular 2 -sovelluksessa luoda omia komponentteja, jotka voidaan lisätä osaksi HTML-dokumenttia eli sovellusta. Esimerkiksi valmis komponentti voidaan liittää helposti sovellukseen lisäämällä komponentin HTML-tagi `<oma-komponentti> </oma-komponentti>` osaksi HTML-dokumenttia.

## 3.4 HTML Templates -standardi

HTML Templates -teknologian avulla voidaan määritellä omalle luodulle elementille sivupohja. Savage (2015) kuvaa `<template>`-elementin tarjoavan tarvittavat ominaisuudet hyvän sivupohjan luomiseksi. `<template>`-elementin sisältöä ei esitetä sivun latautuessa, mutta se voidaan ottaa myöhemmin käyttöön JavaScript:n avulla (Mozilla Developer Network 2016). Muu dokumentti ei vaikuta sivupohjan olemassaoloon. Sivupohja voidaan ottaa käyttöön vasta kun se on tarkoitus ottaa, eli se voidaan kapseloida erilleen muusta dokumentista.

Angular 2 hyödyntää HTML Templates -teknologiaa mahdollistamalla sivupohjan lisäämisen osaksi komponenttia. Näin jokaiselle komponentille voidaan määritellä haluttu ulkoasu.

## 3.5 Shadow DOM -standardi

Shadow DOM -teknologian avulla voidaan kapseloida elementin DOM erilleen muusta dokumentin DOM:sta. Elementin DOM kuuluu osaksi dokumentin DOM:a, mutta se on suojattu dokumentin CSS-määrittelyiltä ja DOM-manipulaatioilta (Savage 2015). Näin elementtiä ei muokata vahingossa ei-toivotulla tavalla.

Shadow DOM -teknologia mahdollistaa Angular 2 -sovelluksissa komponenttien itsenäisen toiminnan. Angular 2 -sovellukset rakentuvat tyypillisesti useasta komponentista, joten on tärkeää estää niitä vaikuttamasta ei-toivotulla tavalla toisiinsa.

## 3.6 HTML imports -standardi

HTML Imports -teknologia mahdollistaa ulkoisten HTML-dokumenttien tuonnin ja uudelleen käyttämisen muissa HTML-dokumenteissa (Exbrayat 2016, s. 41). Luotu komponentti voidaan tuoda toisen HTML-dokumentin käyttöön lisäämällä se osaksi HTML-dokumenttia.

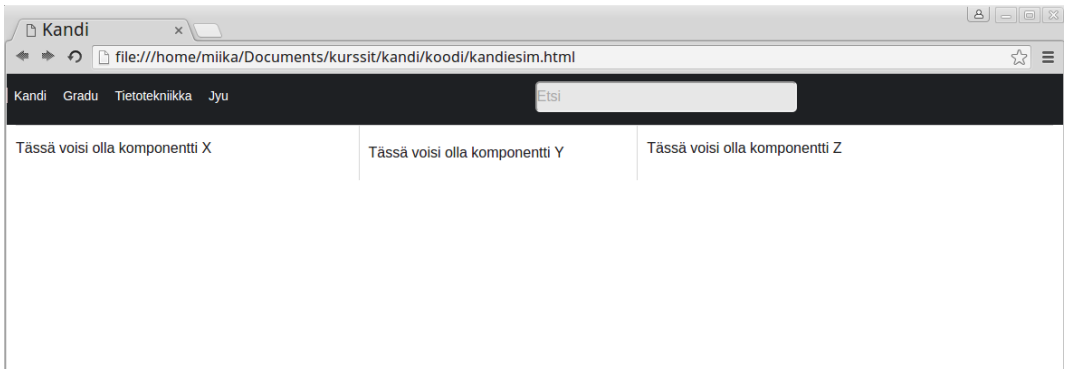
HTML Imports -teknologian avulla voidaan Angular 2:ssa tuoda valmis komponentti sovelluksen käyttöön. Tuotu komponentti voidaan lisätä osaksi sovellusta lisäämällä sen HTML-tagi sivupohjaan.

# 4 Angular 2

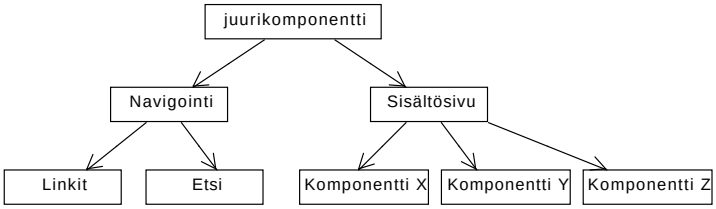
Tässä luvussa esitellään Angular 2 -sovelluksen rakenne ja rakennusosat. Luvussa käsitellään myös TypeScript:n ominaisuuksia, sekä esitellään Angular 2:n reititysmekanismia.

## 4.1 Angular 2 -sovellus

Angular 2 -sovellukset rakentuvat komponenteista ja yksittäinen Angular 2 -sovellus voidaan esittää komponenttipuuna. Ylimpänä komponenttipuussa on juurikomponentti, joka on varsinainen sovellus eli HTML-dokumentti. Komponentit ovat koottavia, eli suurempia komponentteja voidaan rakentaa pienemmistä komponenteista (Lerner ym. 2015, s. 71). Angular 2 -sovellus on siis komponentti, jonka selain hahmontaa (engl. render) sovelluksen latautuessa. Kuviossa 5 esitellään Angular 2 -sovellus ja kuviossa 6 se esitetään komponenttipuuna.



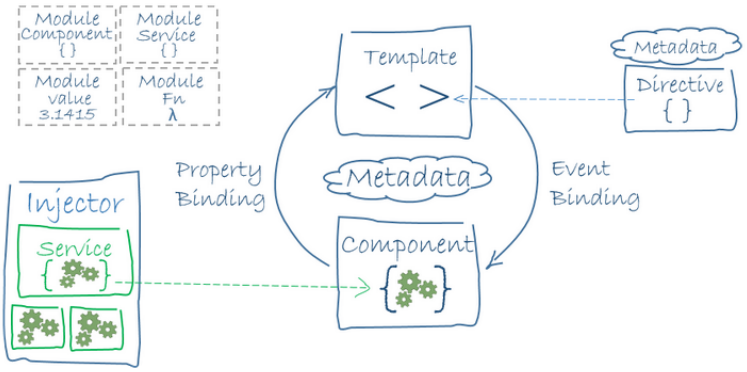

Kuvio 5. Esimerkki Angular 2 -sovelluksesta




Kuvio 6. Kuvion 5 sovellus komponenttipuuna

## 4.2 Rakennusosat

Angular 2 -sovelluksissa voidaan käyttää useita eri rakennusosia (Angular 2a 2016). Kuviossa 7 esitellään sovelluksen tärkeimmät rakennusosat, joita ovat moduuli, komponentti, sivupohja, metadata, datasidonta, direktiivi, palvelu sekä riippuvuusinjektio.




Kuvio 7. Sovelluksen tärkeimmät rakennusosat (Angular 2a 2016)

Moduuli (engl. module) on sovelluksen osa, joka on luotu johonkin tiettyyn tarkoitukseen. Moduuli vie tietoa sovellukselle, tyypillisesti yhden asian, kuten esimerkiksi luokan. Angular 2 -sovellukset ovat modulaarisia ja yleensä sovellukset koostuvat useista moduuleista (Angular 2a 2016). Moduuleja voidaan viedä (engl. Export) ja tuoda (engl. Import) muiden moduulien käytettäväksi (Angular 2c 2016). Moduuli voi pitää sisällään eri asioita, kuten esimerkiksi komponentin, palvelun tai direktiivin.

Komponentti (engl. component) on yksi sovelluksen tärkeimmistä rakennusosista (Angular 2c 2016). Se kontrolloi näkymää eli komponentin sivupohjaa. Komponentti on vuorovaikutuksessa näkymään API:n metodien ja ominaisuuksien kautta (Angular 2a 2016). Jokaisessa sovelluksessa on vähintään yksi komponentti. Yleensä sovellukset rakentuvat useista komponenteista. Komponentteja voidaan myös sijoittaa muiden komponenttien sisään ja ne voivat kommunikoida keskenään.

Sivupohja (engl. template) määrittelee komponentin käyttöliittymän. Sivupohja koostuu HTML-elementeistä sekä Angular 2 -merkinnöistä, jotka määräävät komponen-

tin ulkonäön ja esitystavan (Angular 2a 2016).

Metadata kertoo Angular 2:lle miten luokka käsitellään. Esimerkiksi luokka on vain normaali luokka, kunnes Angular 2:lle kerrotaan metadatan avulla luokan olevan komponentti (Angular 2a 2016).

Datasidonta (engl. data binding) on mekanismi, jonka avulla voidaan luoda datan välitys käyttöliittymän ja komponentin dataosan välille. Sen avulla reagoidaan DOM:ssa tapahtuneisiin muutoksiin tai muutetaan DOM:a. Datasidonta voidaan luoda neljällä eri tavalla:

- Pelkkä datan välitys (engl. interpolation) on helpoin tapa välittää data komponentilta käyttöliittymälle (Angular 2b 2016). Data liikkuu komponentista DOM:iin.
- Ominaisuuksien sidonta (engl. property binding) on tapa välittää data komponentilta käyttöliittymälle ja reagoida komponentin dataosassa tapahtuviin muutoksiin. Ominaisuuksien sidonnan avulla dataosassa tapahtuva muutos päivittyy näkyviin käyttöliittymässä (Angular 2e 2016). Ominaisuuksien sidonnassa data liikkuu komponentista DOM:iin.
- Tapahtumien sidonta (engl. event binding) reagoi käyttäjän vuorovaikutukseen käyttöliittymässä eli DOM:ssa tapahtuvaan muutokseen. Tapahtumien sidonta välittää käyttöliittymässä tapahtuvan muutoksen komponentin dataosalle (Angular 2e 2016). Tapahtumien sidonnassa data liikkuu DOM:sta komponenttiin.
- Kaksisuuntainen datasidonta (engl. two-way data binding) yhdistää ominaisuuksien sidonnan ja tapahtumien sidonnan. Kaksisuuntainen datasidonta välittää datan komponentilta käyttöliittymälle sekä päivittää käyttöliittymässä tapahtuvan muutoksen komponentin dataosaan (Angular 2e 2016). Kaksisuuntaisessa datasidonnassa data liikkuu komponentin ja DOM:n välillä molempiin suuntiin.

Direktiivi (engl. directive) on HTML-elementissä oleva ohje, joka ohjaa Angular 2:ta muokkaamaan DOM:a (Angular 2c 2016). Direktiivi määrittelee muokattavan koh-

teen ja ilmaisee kuinka sitä muokataan. Direktiivejä on kolmea eri tyyppiä (Angular 2a 2016):

- Komponentit ovat direktiivejä, joihin on lisätty sivupohja.
- Rakennedirektiivit muuttavat dokumentin rakennetta lisäämällä, poistamalla ja korvaamalla elementtejä DOM:ssa.
- Ominaisuusdirektiivit muuttavat olemassa olevan elementin ulkomuotoa tai käytöstä.

Palvelu (engl. service) on yleensä normaali yksittäinen luokka, joka tarjoaa jonkin tietyn toiminnallisuuden sovelluksen tarpeeseen (Angular 2a 2016). Esimerkiksi komponentit käyttävät palveluita suorittamaan suurimman osan tehtävistään. Ne delegoivat kaikki epätriviaalit tehtävät palveluille, kuten esimerkiksi sisäänkirjautumisen tai käyttäjän syötteiden validoinnin. Komponentit käyttävät palveluita riippuvuusinjektion avulla.

Riippuvuusinjektio (engl. dependency injection) on mekanismi, jonka avulla sovelluksen osia luodaan ja välitetään sovelluksen muihin niitä tarvitseviin osiin (Angular 2a 2016). Esimerkiksi, kun sovelluksen osa “A” tarvitsee sovelluksen osaa “B”, niin silloin sanotaan osan “A” riippuvan osasta “B“ eli osa “B” on osan “A” riippuvuus. Angular 2:ssa on oma riippuvuusinjektiojärjestelmä (engl. dependency injection system), joka hoitaa sovelluksen osalle kaikki riippuvuudet joita se tarvitsee (Angular 2c 2016). Suurin osa riippuvuuksista on palveluita.

## 4.3 TypeScript

Angular 2 -sovellusten kehityskieleksi suositellaan TypeScript:a sen hyvien ominaisuuksien vuoksi, mutta sovellusten kirjoittaminen on mahdollista myös JavaScript:lla tai Dart:lla (Aden ym. 2016, s. 10). TypeScript on ECMAScript 6:n (lyhen. ES6) päälle kehitetty ohjelmointikieli. ECMAScript 6 on uusi versio ECMAScript 5:stä (lyhen. ES5). ECMAScript 5 tunnetaan paremmin nimellä JavaScript, joka toimii kaikissa yleisimmissä selaimissa. TypeScript-koodi käännetään tavalliseksi JavaScript-koodiksi, koska selaimien tuki TypeScript:lle on vielä heikko (Aden ym. 2016, s. 20).

TypeScript:n avulla voidaan käyttää tiettyjä toiminnallisuuksia, joita JavaScript:ssa ei ole mahdollista käyttää. TypeScript sallii esimerkiksi muuttujien (engl. variable) tyypityksen. Tyypitys vähentää virheiden syntymistä koodia kirjoittaessa ja helpottaa koodia lukiessa sen ymmärtämistä (Lerner ym. 2015, s. 57). TypeScript tukee myös luokkien, rajapintojen ja perinnän käyttöä. Lisäksi TypeScript:n syntaksiset ominaisuudet vähentävät koodin kirjoittamista.

## 4.4 Reititys

Reititys (engl. routing) mahdollistaa navigoinnin Angular 2 -sovelluksen sisällä (Angular 2d 2016). Angular 2:n reititysmekanismin avulla voidaan yksisivuinen web-sovellus jaotella useaan eri sivuun ja määritellä jokaiselle sivulle oma URL-osoite. Sovelluksen jakaminen useaan eri sivuun helpottaa sovelluksen tilan ylläpitoa eli tietoa siitä, mitä osaa sovelluksesta käytetään (Lerner ym. 2015, s. 248).

Useista eri HTML-dokumenteista koostuvilla sivuilla navigointi tapahtuu hakemalla palvelimelta uusi sivu aina sivun vaihtuessa. Jokaisella sivulla on oma URL-osoite, jonka mukaan palvelin osaa lähettää oikean sivun selaimelle. Selainpäässä toimivassa yksisivuisessa web-sovelluksessa ei teknisesti vaadita uutta URL-osoitetta sivua vaihdettaessa. Lerner ym. (2015, s. 248) kertovat, että yhden URL-osoitteen käyttö kaikilla sivuilla aiheuttaa muun muassa seuraavanlaisia heikkouksia:

- Sivun uudelleenlatauksessa palataan aina etusivulle.
- Sivua ei voida kirjainmerkitä ja palata sille myöhemmin.
- Sivun URL-osoitetta ei voida jakaa muille käyttäjille.

Reititys on ratkaisu aiemmin mainittuihin ongelmiin. Reititys mahdollistaa URL-osoitteiden luonnin Angular-sovelluksessa, ja sen avulla voidaan määritellä missä kohtaa sovellusta käyttäjä on (Lerner ym. 2015, s. 248).

# 5 Yhteenveto

Web-sovellusten sujuva käyttäminen on tärkeä osa käyttökokemusta. Siitä onkin nykyään tullut vaatimus, jota ei voi sivuuttaa. Käyttäjät hylkäävät nopeasti sivut, jotka toimivat hitaasti ja/tai huonosti. Hyvä käyttökokemus on yksi suuri tekijä, joka on johtanut yksisivuisten web-sovellusten suosion kasvuun. Toinen tärkeä tekijä on Angular 2:n kaltaisten JavaScript-sovelluskehysten ilmestyminen web-kehitykseen. Koska nykyään web-sovellukset ovat yhä suurempia ja monimutkaisempia, on niiden kehittäminen ilman sovelluskehyksen apua todella hankalaa. Nykyään sovelluskehyksen valinta ja käyttäminen onkin usein tärkeässä roolissa web-sovellusten kehittämisessä.

Angular 2 on hyvä työkalu yksisivuisten web-sovellusten kehittämisessä. Se soveltuu erityisesti suurien ja monimutkaisten web-sovellusten kehittämiseen. Pienten ja yksinkertaisten web-sovellusten rakentaminen voi olla helpompi ja nopeampi tehdä itse alusta loppuun, tai käyttää jotain muuta sovelluskehystä tai kirjastoa. Lisäksi Angular 2:n käyttämistä harkittaessa kannattaa ottaa huomioon oppimiskynnys. Ei ole järkevää opetella hankalahkon sovelluskehyksen käyttämistä, mikäli aikomuksena on kehittää vain yksinkertainen web-sovellus.

Jatkotutkimusta voisi tehdä vertailemalla Angular 2:n sen aiemman version AngularJS:n käytön eroavaisuuksia. Tulevaisuudessa voisi myös tutkia Angular 2:n eroa muihin JavaScript-sovelluskehyksiin.

# Kirjallisuutta


Aden, D., Aden, J. & Wilken, J. 2016. *Angular 2 in Action*. Manning Publications.

Angular 2a. 2016. *Architecture Overview*. Saatavilla WWW-muodossa <URL: https://angular.io/docs/ts/latest/guide/architecture.html>. Viitattu 16.2.2016.

Angular 2b. 2016. *Displaying Data*. Saatavilla WWW-muodossa <URL: https://angular.io/docs/ts/latest/guide/displaying-data.html>. Viitattu 28.4.2016.

Angular 2c. 2016. *Glossary*. Saatavilla WWW-muodossa <URL: https://angular.io/docs/ts/latest/guide/glossary.html>. Viitattu 27.4.2016.

Angular 2d. 2016. *Routing*. Saatavilla WWW-muodossa <URL: https://angular.io/docs/ts/latest/guide/router.html>. Viitattu 27.4.2016.

Angular 2e. 2016. *templateSyntax*. Saatavilla WWW-muodossa <URL: https://angular.io/docs/ts/latest/guide/template-syntax.html>. Viitattu 28.4.2016.

Chrome. 2016. *Chrome*. Saatavilla WWW-muodossa <URL: https://developer.chrome.com/apps/app_frameworks>. Viitattu 27.4.2016.

Cortez, R. & Vazhenin, A. 2015. *Virtual Model-View-Controller Design Pattern: Extended MVC for Service-Oriented Architecture*. Teoksessa IEEJ Transactions on Electrical and Electronic Engineering. John Wiley & Sons, Inc., s. 411–422.

Exbrayat, C. 2016. *Become a ninja with Angular 2*.

Fain, Y. & Moiseev, A. 2015. *Angular 2 Development with TypeScript*. MEAP Edition. Manning Publications.

Freeman, A. 2014. *Pro AngularJS*. Apress.

Fink, G. & Flatow, I. 2014. *Pro Single Page Application Development*. Using Backbone.js and ASP.NET. Apress.

Lerner, A., Coury, F., Murray, N. & Taborda, C. 2015. *ng-book 2*. Prerelease version.

Li, X. 2015. *Application of MVVM Design Pattern in MES*. Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on, s. 1374–1378.

Mikowski, M. & Powell, J. 2014. *Single Page Web Applications*. Manning Publications.

Mozilla Developer Network. 2016a. *Template*. Saatavilla WWW-muodossa <URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template>. Viitattu 8.4.2016.

Mozilla Developer Network. 2016b. *Web Components*. Saatavilla WWW-muodossa <URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components>. Viitattu 8.4.2016.

Oscariza Jr, F., Pattabiraman, K. & Mesbah, A. 2015. *Detecting Inconsistencies in JavaScript MVC Applications*. Teoksessa Proceedings of the 37th International Conference on Software Engineering - Volume 1. IEEE Press, s. 325–335.

Savage, T. 2015. *Componentizing the web*. Communications of the ACM, 11, s. 55–61.

Zhang, H. & Zhu, S. 2013. *B/S implementation of internet-based electrical engineering lab with MVC architecture*. Teoksessa Control and Automation (ICCA), 2013 10th IEEE International Conference on. IEEE, s. 551–555.

webcomponents.org. 2016. *WebComponents.org*. Saatavilla WWW-muodossa <URL: http://webcomponents.org/>. Viitattu 11.3.2016.