Tampereen yliopisto

Sandra Raitaniemi

ANGULARIN JA VUE.JS:N VERTAILU YHDEN SIVUN SOVELLUKSEN TOTEUTUKSESSA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Huhtikuu 2019

TIIVISTELMÄ

Sandra Raitaniemi: Angularin ja Vue.js:n vertailu yhden sivun sovelluksen toteutuksessa
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Huhtikuu 2019

Tässä kandidaatintyössä selvitetään kahden yhden sivun sovelluksen kehitykseen tarkoitetun sovelluskehyksen, Angularin ja Vue.js:n välillä, kumpi sovelluskehys on kehittäjäystävällisempi. Vertailu suoritetaan sovelluskehyksillä kehitettyjen sovellusten pohjalta. Kehitettävä sovellus on päiväkirjasovellus, jolla on kolme vaatimusta: päiväkirjamerkintöjen tallentaminen, palvelimelta niiden hakeminen ja niiden tarkastelu käyttöliittymässä. Kumpikin sovellus käyttää samaa Node.js:n päällä pyörivää REST-rajapintaa. Vertailu tehdään kehittäjän näkökulmasta. Arviointikriteereinä työssä ovat sovelluksen kehitykseen kulutettu aika, kehityksen aikana kohdatut ongelmat ja kirjoitettujen koodirivien määrä.

Kehitettyjen sovellusten perusteella saatiin seuraavat tulokset: Angularilla sovelluksen kehitykseen kului 15 tuntia, ongelmia kohdattiin 5 kappaletta ja koodirivejä kirjoitettiin 108 kun taas Vue.js:llä sovelluksen kehitykseen kului 11,5 tuntia, ongelmia kohdattiin 8 kappaletta ja koodirivejä kirjoitettiin 91 kappaletta. Tulosten perusteella Vue.js on kehittäjäystävällisempi sovelluskehys.

Avainsanat: Yhden sivun sovellus, sovelluskehys, Angular, Vue.js, MVVM

SISÄLLYSLUETTELO

1 JOHDANTO

Yhden sivun sovelluksien kehitys alkoi 2000-luvun alussa. Silloin web-kehittäjät tunnistivat tarpeen nettisivuille, jotka kuluttaisivat entistä vähemmän laajakaistayhteyttä. Kehittäjät huomasivat tuolloin, että laajakaistaresurssit eivät riittäisi vastaamaan kiihtyvästi kasvavan internetin käyttäjäjoukon kulutukseen. Perinteinen selaimen ja palvelimen välinen kommunikaatio olisi liian verkkoa kuormittavaa, jos jokainen käyttäjä lataisi aina sivua päivittäessään uuden HTML-tiedoston (*Hypertext Markup Language*) palvelimelta. [1]

Yhden sivun sovellukset ovat yksi ratkaisu resurssien säästämiseen. Perinteisen HTML-sivuja palvelimen puolella renderöivän web-sovellusarkkitehtuurin sijaan, yhden sivun sovelluksen ja palvelimen välillä kulkee vain JSON-muotoista dataa. Nimensä yhden sivun sovellukset ovat saaneet siitä, että niiden sisältö on vain yhden HTML-sivun sisällä. AJAXin eli asynkroninsen kommunikoinnin yhdistelmäteknologian käyttäminen palvelimen ja asiakasohjelman välisessä kommunikoinnissa mahdollistaa datan lataamisen ohjelman käytön taustalla, jolloin käyttökokemus lähenee natiivin työpöytäsovelluksen käyttökokemusta.

Yhden sivun sovelluksien kehitykseen on tarjolla runsaasti erilaisia sovelluskehyksiä. Jokainen sovelluskehys tarjoaa oman tapansa yhden sivun sovelluksen kehitykseen ja mielenkiintoista on, millaisia eroja näiden sovelluskehysten välillä on ja miten nämä erot vaikuttavat työn kehitykseen. Tässä työssä selvitetään kumpi sovelluskehys, Angular vai Vue.js, on kehittäjäystävällisempi. Selvitys tehdään kehittämällä kummallakin sovelluksella saman määrittelyn mukaiset sovellukset ja sovelluskehyksiä vertaillaan kehitettyjen sovellusten pohjalta.

Luvussa 2 tutustutaan yhden sivun sovelluksiin ja yhden sivun sovelluksien kehitykseen tarkotettuihin sovelluskehyksiin. Yhden sivun sovelluksiin tutustutaan avaamalla niihin vahvasti linkittyneitä teknologioita Ajaxia ja JavaScriptiä. Luvussa 3 määritellään kehitettävä sovellus sekä sovelluksien arviointikriteerit. Lisäksi luvussa 3 tutustutaan Angulariin ja Vue.js:ään. Lopuksi käydään läpi sovellusten kehitys koodiesimerkkien avulla. Luvussa 4 käydään läpi työn tulokset ja niiden arviointi. Viimeinen luku on työn yhteenveto.

2 YHDEN SIVUN SOVELLUS JA SOVELLUSKEHYKSET

2.1 Yhden sivun sovellus

Yhden sivun sovellukset (engl. Single Page Application) ovat nimensä mukaan yhdelle sivulle mahtuvia web-sovelluksia. Yhden sivun sovelluksissa tavoitteena on tuottaa saumaton käyttökokemus ja runsas käyttöliittymä, josta käyttäjä saa haluamansa toiminnon tai tiedon mahdollisimman sujuvasti. [2]

Käyttökokemuksen parantamisen lisäksi yhden sivun sovelluksien toimintaperiaate on tiukasti kytketty pyrkimykseen säästää laajakaistaresursseja. Yhden sivun sovelluksen sisältö ladataan palvelimelle vain kerran ja tämän jälkeen selaimen ja palvelimen välillä kulkee HTML-tiedostojen sijaan vain dataa. [3]

2.1.1 Historiaa

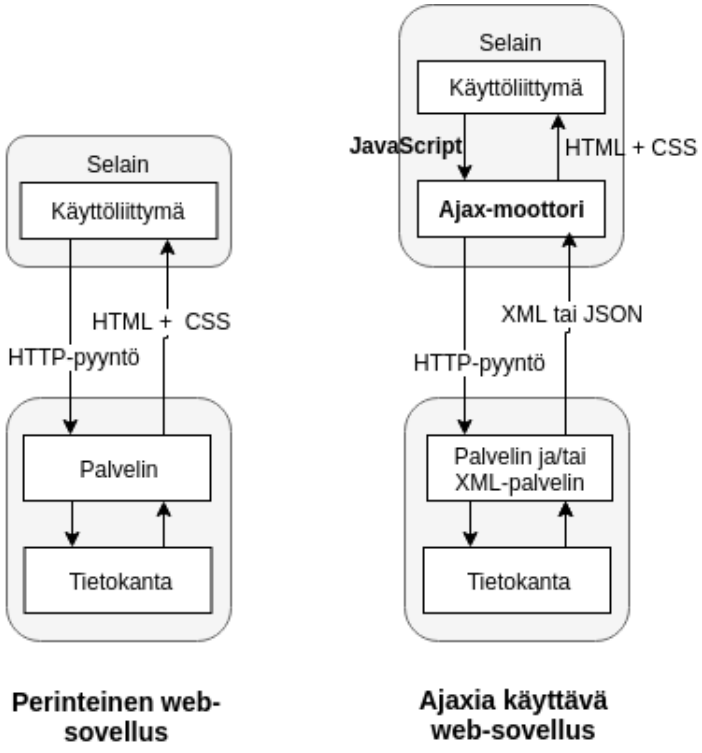
Yhden sivun sovelluksista on puhuttu jo 2000-luvun alkupuolelta, kun vuonna 2003 Marcio Galli, Roger Soares ja Ian Oeschnger julkaisivat artikkelin yhden sivun sovelluksista. Motivaationa tuolloin yhden sivun sovellusten kehitykselle oli internetin kasvanut käyttö. Vuonna 2008 julkaistun tilaston mukaan internetin käyttäjien määrä oli kasvanut 305 % vuodesta 2000 [4]. Digitaalisen sisällön jakaminen ja kuluttaminen oli nousussa ja tarvittiin ratkaisuja, jotka mahdollistaisivat kasvun jatkumisen. [1] Vaatimuksena oli siis tuottaa nettisivuja, jotka kuluttaisivat vähemmän resursseja kuten laajakaistayhteyttä. Resurssien säästön lisäksi haluttiin tuottaa web-sovelluksia, joilla olisi natiivien työpöytäsovellusten ulkonäkö ja käyttötuntuma. Tämän saavuttamiseksi kokeiltiin useita eri ratkasuita, kuten IFrameja, Java appletteja, Adobe Flashia ja Microsoft Silverlightia vaihtelevin tuloksin. [5]

Motivaatio yhden sivun sovelluksien kehitykselle tuli myös käyttäjien puolelta. Perinteisen monen sivun sovelluksen päivitys saattaa kestää ja näin ollen huonontaa käyttökokemusta. Yhden sivun sovelluksilla pyritään saumattomuuteen sivun käytössä ja miellyttävään käyttökokemukseen. [3]

2.1.2 Ajax

Ajaxia käytetään asynkroniseen kommunikointiin palvelimen kanssa ja sivun osittaiseen päivitykseen ilman koko sivun uudelleenlatausta. [5] Ajax koostuu sanoista Asyncronous JavaScript and XML, mutta nykyään sillä viitataan kaikkiin niihin teknologioihin, joita käytetään palvelimelta tiedon lataukseen ilman selaimen uudelleenlatausta [6]. Ensimmäinen maininta Ajaxista on Jesse James Garretin vuonna 2005 julkaisemassa artikkelissa. Garretin yli 13 vuotta vanhan kuvauksen mukaan Ajax hyödyntää XHTML:ää ja CSS:ää sisällön esitykseen, dokumentaatio-oliomallia (*Document Object Model*) eli DOMia sisällön dynaamiseen muokkaukseen ja esitykseen, XML:ää datan siirtoon ja XMLHttpRequestia asynkronisuuden toteuttamiseen. JavaScript sitoo yhteen nämä edellä mainitut teknologiat. [7] Nykyaikainen Ajax on kuitenkin hieman muuttunut Garretin kuvauksen mukaisesta Ajaxista. Esimerkiksi JSON on ohittanut suosiossa XML:n datan siirron muotona [8].

Kuvassa 2.1 ovat perinteisen web-sovelluksen ja Ajaxia käyttävän web-sovelluksen toimintaketjut palvelimen kanssa kommunikoidessa. Ajaxia käyttävä web-sovellus eliminoi palvelimen ja selaimen välisen synkronisen kommunikoinnin lisäämällä näiden väliin Ajax-moottorin. Sen sijaan, että uuden istunnon alussa selain lataisi web-sovelluksen, se lataakin asynkronisuuden mahdollistavan Ajax-moottorin. Ajax-moottori huolehtii käyttöliittymän renderöimisestä ja palvelimen kanssa kommunikoinnista. Jokainen tyypillisesti HTTP-kutsun synnyttävä käyttäjän toiminto ohjataankin JavaScriptinä Ajax-moottorille, joka kutsuu palvelinta asynkronisesti.



Kuva 2.1. *Perinteisen web-sovelluksen ja Ajaxia käyttävän web-sovelluksen kommunikointi palvelimen kanssa [7]*

Ajax kuuluu olennaisesti yhden sivun sovelluksiin. Pyrkimys tuoda yhden sivun sovellusten käyttökokemus lähelle natiivin työpöytäsovelluksen käyttökokemusta on hyvin riippuvainen ohjelman asynkronisuudesta ja sivun käyttöliittymän osittaisesta dynaamisesta päivityksestä. Sovelluksen asynkronisuus vähentää sivun latausaikaa sekä sivun ensimmäisen latauskerran aikana että sivun käytön aikana [7]. Koska Ajax on asynkroninen, käyttäjä voi jatkaa sovelluksen käyttöä samalla, kun asiakasohjelma (engl. client) lataa tietoa palvelimelta. [9] Esimerkiksi kaavakkeessa olevan suuntanumeron oikeellisuus voidaan tarkistaa palvelimelta samalla, kun käyttäjä jatkaa kaavakkeen täyttöä. [6]

2.1.3 JavaScript ja TypeScript

JavaScript on tulkattava komentokieli (engl. scripting language), jolla on olio-ohjelmoinnin ominaisuuksia. JavaScriptin ydin on syntaksiltaan C++:n ja Javan kaltainen. Edellä mainituista ohjelmointikielistä poiketen JavaScript on kuitenkin löyhästi tyypitetty eli JavaScript muuttujia ei tyypitetä. [10]

JavaScriptiä käytetään usein selaimen puolelle ohjelmoidessa merkintäkieli (engl. markup language) HTML:n rinnalla. JavaScriptin avulla voidaan kontrolloida selainta, muokata web-sivun sisältöä ja luoda dynaamista sisältöä, joka reagoi käyttäjän laukaisemiin tapahtumiin, kuten napin painallukseen. JavaScriptiä käytetään myös selaimen ulkopuolella toimivissa ympäristöissä, suosituimpana esimerkkinä palvelinten luontiin käytetty Node.js. Ainoa vaatimus JavaScriptin toimintaan on, että alustalla on JavaScript-moottori (engl. JavaScript engine), joka tulkkaa JavaScriptin konekielelle. [11]

Yhden sivun sovellukset nojaavat JavaScriptiin. Koska yhden sivun sovelluksien sisältö päivitetään dynaamisesti selaimen puolella, on JavaScriptin merkitys yhden sivun sovelluksille suuri. JavaScriptillä ohjelmoidaan mitkä osat sovelluksen käyttöliittymässä näkyvät käyttäjälle ja Yhden sivun sovelluksen logiikan sijaitessa selaimen puolella, on JavaScript vastuussa sen perinteisten käyttökohteiden lisäksi myös logiikasta. [5]

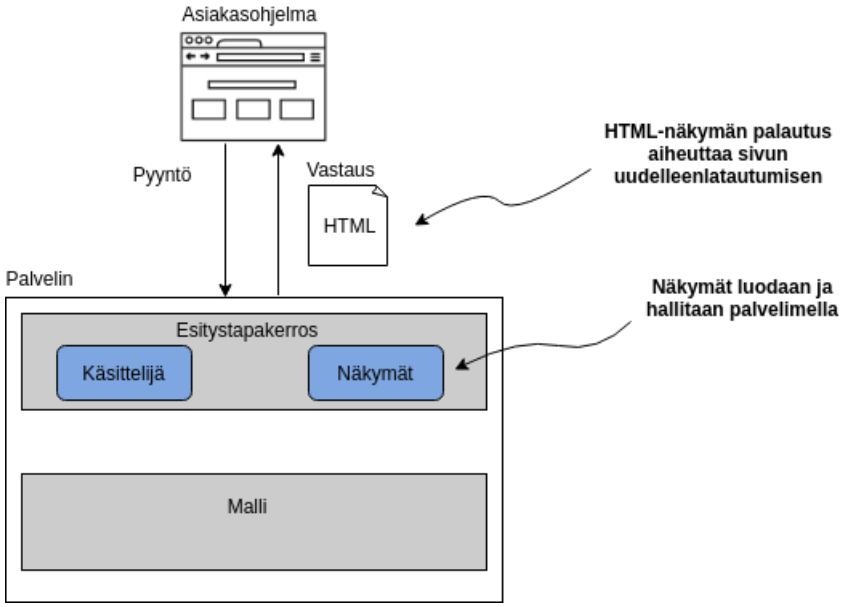
TypeScript on Microsoftin kehittämä ECMAScript 6:n standardien mukainen komentokieli. TypeScript on luokkiin pohjautuva staattisesti tyypitetty ylijoukko JavaScriptistä ja se on paradigmaltaan oliopohjainen. TypeScript kääntyy JavaScriptiksi ja se hyödyntää samaa semantiikkaa ja syntaksia kuin JavaScript. TypeScriptin avulla ohjelmassa käytetyt muuttujat, funktioiden parametrit ja paluuarvot voidaan tyypittää, mikä helpottaa ohjelman luettavuutta, ymmärrettävyyttä ja ohjelmassa olevien virheiden huomaamista. TypeScriptiin sisäänrakennettu tyypin tarkastaja käy läpi ohjelman muuttujien tyypit jo kääntämisen aikana. Tällöin mahdolliset väärän tyyppisten muuttujien käytöstä johtuneet virheet tulevat esiin jo ohjelman kehityksen aikana sen sijaan, että ne huomattaisiin vasta ohjelman testauksessa. [12] Tyypityksessä voi myös käyttää määrettä `any` tai jättää tyypityksen kokonaan pois, jolloin muuttujat ovat oletusarvoisesti tyyppiä `any`. Tällöin muuttujalle voidaan asettaa erityyppisiä arvoja. [13]

TypeSciptin sanotaan olevan syntaksiltaan ja toiminnaltaan miellyttävä C++-taustaiselle

ohjelmistokehittäjälle. Muuttujien staattisen tyypityksen lisäksi TypeScriptin sisältämät luokat ja niiden käyttämät privaattimuuttujat, TypeScriptin käyttämät monikot (engl. tuple) ja rajapinnat ovat monille C++-ohjelmoijille tuttuja konsepteja. [12]

2.1.4 Arkkitehtuuri

Yhden sivun sovelluksen perusideana on tuottaa mahdollisimman tehokas web-sovellus poistamalla turhat sivun uudelleenlataukset. Perinteisessä monen sivun sovelluksessa käyttäjän klikatessa linkkiä sivu tekee HTTP-pyynnön palvelimelle, palvelin vastaa pyyntöön palauttamalla HTML-tiedoston ja sivu uudelleenlatautuu. Yhden sivun sovelluksessa palvelimelta ladataan sovelluksen käytön alkaessa vain yksi HTML-sivu ja tätä sivua päivitetään dynaamisesti. Päivitys tapahtuu, kun asiakasohjelma lähettää AJAX-pyynnön palvelimelle, pyyntöön vastataan datalla esimerkiksi JSON-muodossa ja yhden HTML-sivun sisältöä päivitetään vain uuden datan osalta. [3]

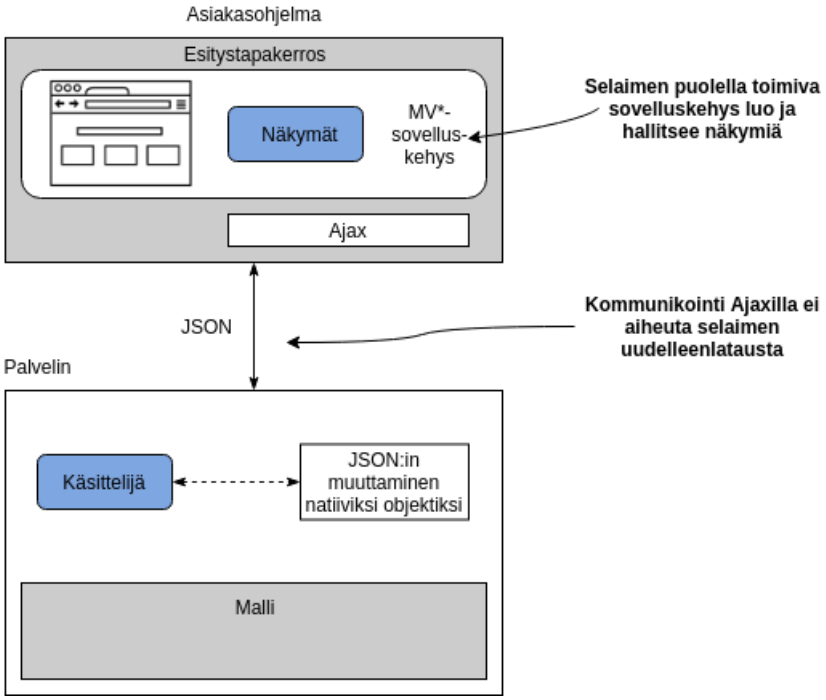


Kuva 2.2. *Monen sivun sovelluksen arkkitehtuuri [5]*

Monen sivun sovellukset (engl. Multi Page Application) eli MPA:t noudattavat klassista malli–näkymä–käsittelijä-mallia (engl. Model–View–Controller) eli MVC-mallia. Kuten kuvasta 2.2 voidaan nähdä, mallin mukaan toteutetussa sovelluksessa käyttäjä suorittaa toimintoja sovelluksen käyttöliittymällä ja sovellus kommunikoi palvelimella esitystapakerroksen (engl. presentation layer) sisällä sijaitsevan käsittelijän kanssa. Tämän jälkeen käsittelijä saa tarvitsemansa datan mallilta ja käsittelijä syöttää datan näkymälle. Kun näkymä ja data on yhdistetty, näkymä palautetaan selaimelle, joka päivittää asiakasohjelman näyttämään uutta sivua.

Yhden sivun sovelluksissa MVC-mallin näkymä sijaitsee asiakasohjelman puolella, kuten kuvasta 2.3 voidaan nähdä. Käsittelijä ja malli sijaitsevat edelleen palvelimen puolella. Yhden sivun sovellus poikkeaa monen sivun sovelluksesta käsittelijän ja näkymän väli-

sen suhteen osalta. Yhden sivun sovelluksissa käsittelijä ei syötä dataa näkymään, vaan näkymää hallitsee sovelluskehys. Sovelluskehys luo ja päivittää näkymiä selaimen puolella.



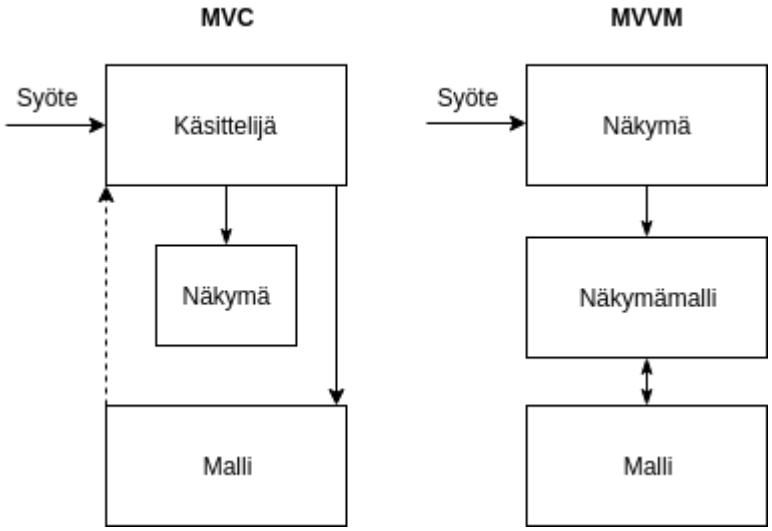
Kuva 2.3. *Yhden sivun sovelluksen arkkitehtuuri [5]*

Kuten kuvasta 2.3 voidaan nähdä, selaimen puolella toimivassa kerroksessa on mainittu MV*-sovelluskehys. Tämä tarkoittaa sitä, että yhden sivun sovellukset voivat noudattaa erilaisia variaatioita tutusta MVC-arkkitehtuurista, kun MVC:n C eli käsittelijä (engl. controller) korvataan jollain muulla ratkaisulla.

Yhden sivun sovelluksien tekoon tarkoitettujen sovelluskehysten yleinen korvike käsittelijälle on VM (*ViewModel*) eli näkymämalli. MV* yhdessä VM:n kanssa muodostavaa Model–View–ViewModel-arkkitehtuurin eli MVVM-arkkitehtuurin. MVVM-arkkitehtuurissa malli on edelleen palvelimen puolella, kuten kuvasta 2.3 voidaan todeta. Näkymä MVVM-arkkitehtuurissa on edelleen sovelluksen käyttöliittymä ja näkymä on linkitetty näkymämalliin.

MVVM-arkkitehtuurissa näkymämalli toimii näkymän ja mallin välissä liimana. Näkymämalli on yhteydessä näkymään datansidonnalla (engl. data binding). Datansidonnan avulla näkymä kommunikoi näkymämallille näkymässä suoritettuja tapahtumia. Riippuen ohjelman toiminnasta näkymämalli voi tämän jälkeen lähettää pyynnön mallille. [14]

Kuvasta 2.4 nähdään, että MVC- ja MVVM-arkkitehtuurit ovat toiminnaltaan hyvin erilaisia. Yhteistä näillä kahdella on kuitenkin se, että kummassakaan arkkitehtuurissa näkymä ja malli eivät ole suoraan yhteydessä toisiinsa. Jos malli ja näkymä olisivat suoraa yhteydessä toisiinsa, se saattaisi heikentää sovelluksen turvallisuutta ja tehokkuutta. MVVM-arkkitehtuurilla pyritään välttämään nämä ongelmat. [2]



Kuva 2.4. *MVC- ja MVVM-arkkitehtuurit [2]*

2.2 Sovelluskehykset

Sovelluskehys on työkalu web-sovellusten kehitykseen ja pyörittämiseen. Sovelluskehykset ovat kuin tukiranka, jonka päälle web-sovellus tulee. Useissa tapauksissa sovelluskehys määrittelee sovelluksen arkkitehtuurin, jonka mukaan kehittäjä ohjelmoi sovelluksensa. [15] Sovelluskehyksen kontrolli kehitettävän sovelluksen rakenteeseen kuitenkin vaihtelee. Sovelluskehys voi olla vahvasti ohjaileva (engl. opinionated). Vahvasti ohjaileva sovelluskehys ohjaa kehitystä selkeään suuntaan kehittäjän tavoitteiden saavuttamiseksi. [16] Vahvasti ohjailevien sovelluskehysten vastakohtana ovat kehykset, jotka antavat kehittäjälle hyvin vapaat kädet sovelluksen toteutuksessa. Esimerkkinä tällaisesta sovelluskehyksestä on reititykseen tarkoitettu Express.js [17].

Sovelluskehyksellä on kaksi pääasiallista toimintoa: palvelimen (engl.back-end) ja selaimen (engl. front-end) puolella toimiminen. Sovelluskehys voi olla tyypiltään vain palvelimella tai selaimella toimiva tai näiden kahden yhdistelmä. Palvelinpuolen kehyksellä toteutetaan sovelluksen käyttämän datan käsittely ja logiikka. Selainpuolen kehyksellä toteutetaan käyttöliittymä. Selainpuolen sovelluskehyksiä käytetään myös yhden sivun sovellusten kehitykseen. Sovelluskehykset helpottavat sovellusten kehitystä tarjoamalla olennaisia toimintoja. Sovelluskehyksien valmiit toiminnot kattavat usen välimuistin hallinnan, projektin rungon valmiiksi luonnin, sivun turvallisuuden hallinnan ja URL-osoitteiden ohjauksen. Sovellukehykset tarjoavat lisäksi valmiita kirjastoja tyypillisten web-sovellusten, kuten blogien ja geneeristen web-sivujen, luontiin. [15] Sovelluskehyksen valmiin arkkitehtuurin ja valmiiden toimintojen ansiosta kehittäjä voi keskittyä kehittämään sovelluksen logiikkaa [5].

Olennaista sovelluskehykselle on, minkä arkkitehtuurin sen määrää sovellukselle. MV*-arkkitehtuuria noudattavien sovelluskehysten käytössä on etuja. MV*-sovelluskehyksen käyttö lisää koodin selkeyttä ja siisteyttä monin tavoin, kun ohjelman rakennusosat (data,

logiikka ja näkymä) ovat selkeästi toisistaan eristetty. Taipumus kirjoittaa spagettikoodia vähenee, kun sovelluskehys asettaa tarkat raamit sovellukselle ja ohjaa sovelluksen koodin komponetteja löyhiin kytköksiin (engl. loose coupling). Lisäksi sovelluksen HTML siistiytyy, kun HTML:ään upottamisen sijaan JavaScript ja CSS ovat omissa tiedostoissaan. [5]

Sovelluskehyksien tarjoamien logiikkaa koskevien toiminnallisuuksien lisäksi sovelluskehykset helpottavat HTML-elementtien käyttöä. Esimerkiksi tilanne, jossa näkymän pitää näyttää listan jokainen data-alkio omana lista-alkionaan. Tämä on toteutettavissa pelkällä JavaScriptillä, jolloin HTML-listaelementti ja sen lista-alkiot luodaan JavaScriptillä, minkä jälkeen jokainen data-alkio käydään yksitellen läpi silmukassa lisäten ne HTML-listaan. Sovelluskehyksissä tämä on ratkaistu HTML-lista-alkioelementtiin upotettavalla funktiolla, joka käy siihen sidotun datalistan läpi ja luo lista-alkion jokaiselle uudelle data-alkiolle. Tämä menettely säästää kehittäjältä vaivaa ja aikaa, kun monen koodirivin sijaan, toiminnan toteuttamiseen tarvitaan vain yksi rivi.

3 SOVELLUKSEN KEHITYS

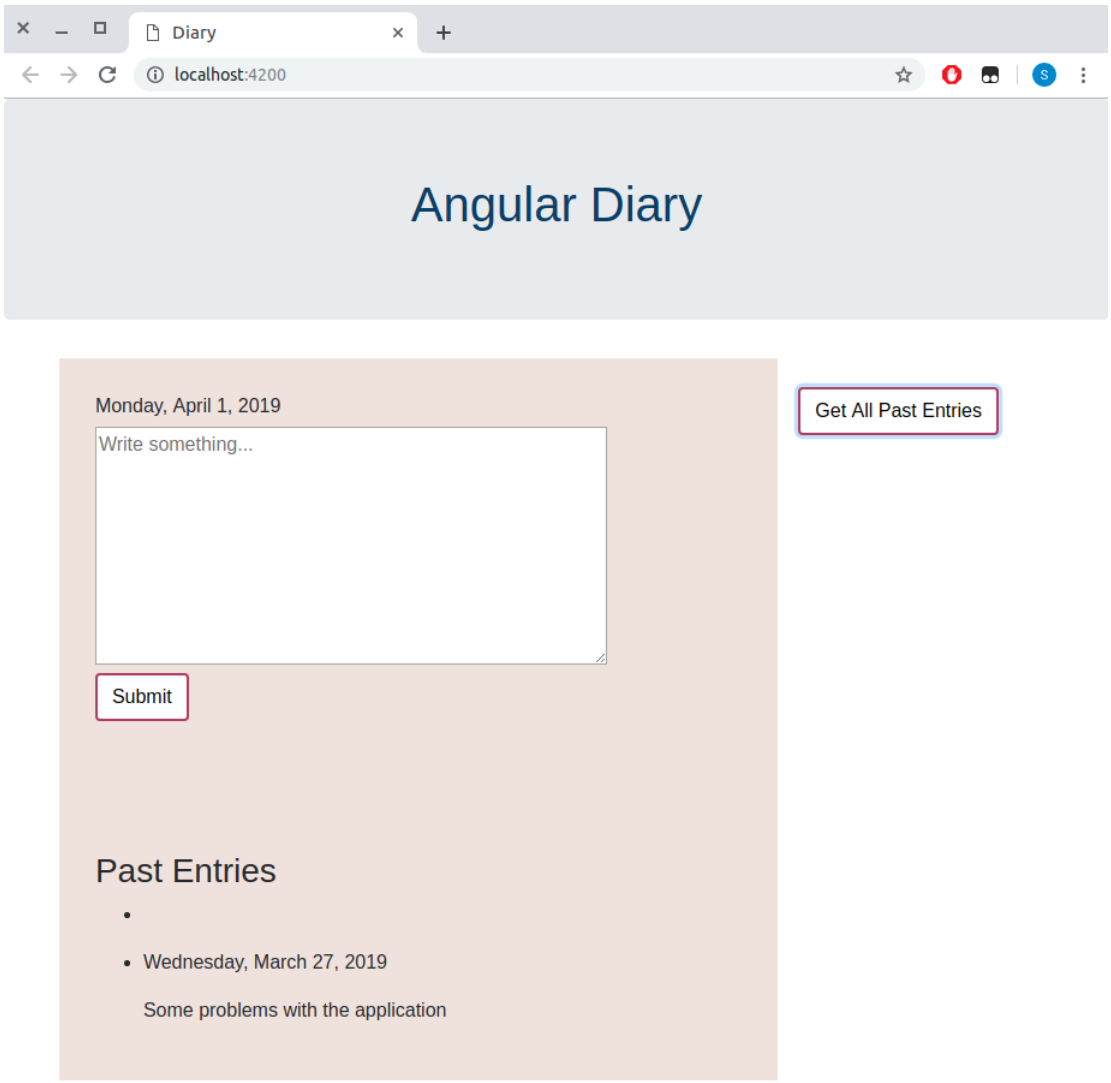
3.1 Kehitettävä sovellus

Tässä työssä toteutetaan kaksi yksinkertaista, saman toiminnallisuusmäärittelyn mukaista web-sovellusta. Koska työn tavoitteena on tutkia sovelluskehyksiä vain kehittäjän kannalta merkkittävien asioiden osalta, eikä käyttäjän näkökulmasta sovelluksen käytössä, sovelluksen käytettävyyttä ja ulkonäköä ei painoteta kehityksessä.

Työssä kehitettävä sovellus on yksikertainen päiväkirjasovellus, jossa käyttäjä voi kirjoittaa ja tallentaa päiväkirjamerkintöjä. Käyttäjä voi lisäksi selata vanhoja päiväkirjamerkintöjä vanhimmasta uusimpaan. Idea toteutettavaan sovellukseen tuli Flavio Copesin blogista [18].

Sovellukset kehitetään Atom-koodieditorissa ja ajetaan Googlen Chrome-selaimessa. Sovellusten käyttämä palvelin tehdään käyttäen Node.js:ää. Tietokantana käytetään MongoDB:tä. Electron.js:llä tehdään palvelimelle REST-rajapinta, jota kumpikin sovellus käyttää. Kutsumalla palvelimella toteutettua rajapintaa, asiakasohjelma voi lisätä päiväkirjamerkintöjä ja hakea kaikki aiemmat päiväkirjamerkinnät katseltavaksi.

Kuvassa 3.1 on Angularilla toteutettu päiväkirjasovellus. Kuvassa vasemmalla olevaan tekstikenttään käyttäjä syöttää päiväkirjamerkintänsä ja painaa tämän jälkeen Submit-nappia. Napin painalluksesta päiväkirjamerkintä lähetetään palvelimelle. Kuvassa oikealla puolella on nappi kaikkien entisten päiväkirjamerkintöjen hakemiseen. Tämän napin painalluksesta tekstikentän alapuolelle ilmestyvät vanhat päiväkirjamerkinnät.



Kuva 3.1. *Angularilla toteutettu sovellus Chrome-selaimessa.*

3.2 Arviointikriteerit

Tässä työssä tavoitteena on vertailla kahta sovelluskehystä. Arviointikriteereinä käytetään kehittäjäystävällisyyttä ja kirjoitustehokkuutta. Kehittäjäystävällisyyttä arvioidaan mittaamalla toteutukseen kulunutta aikaa ja eteen tulleiden ongelmien määrää. Kirjoitustehokkuutta arvioidaan mittaamalla ohjelmaan kirjoitettujen koodirivien määrä.

Eteen tulleisiin ongelmiin lasketaan vain itse sovelluskehyksestä johtuvat ongelmat. Tällöin ongelmien määrää eivät vääristä ongelmat, jotka johtuvat yleisiksi laskettavista web-ohjelmointiin tai JavaScript-ohjelmointiin liittyvistä ongelmista. Yleiseksi ongelmaksi voidaan laskea esimerkiksi JavaScriptissä usein käytetyn `this`-avainsanan käytöstä johtuva

ongelma. Kehitykseen kulutettu aika käsittää koko kehitykseen kulutetun ajan projektin alustuksesta sen loppumiseen. Tällöin sovelluksen kehitykseen kulutettuun aikaan sisältyy myös yleisiksi laskettavien ongelmien ratkaisuun käytetty aika. Vaikka tuloksien kannalta olisi oleellisempaa tietää vain itse sovelluskehyksestä johtuvien ongelmien kulutukseen käytetty aika, ajan mittaamisen haasteiden takia kello käy koko kehityksen ajan. Ajan mittaamisen haaste olisi kohdistaa kulutettu aika tietylle ongelmalle ja tämän jälkeen tunnistaa, että onko ongelma sovelluskehyksestä johtuva, onko se yleiseksi laskettava ongelma vai yleisen ja sovelluskehyksestä johtuvan ongelman yhdistelmä.

Kirjoitettujen koodirivien määrään lasketaan vain HTML-koodi ja itse kirjoitettu JavaScript- tai TypeSript-koodi. CSS-tyylittelyä ei oteta laskuissa huomioon, koska sovelluskehykset eivät vaikuta sen koodirivien määrään.

3.3 Sovelluskehyksien valinta

Sopivaa sovelluskehystä valittaessa on kiinnitettävä huomiota erinäisiin asioihin, jotka voivat vaikuttaa kehyksen käytettävyyteen. Tällaisia asioita ovat esimerkiksi sovelluskehystä käyttävä yhteisö, kehyksen päivitysten taajuus, tuki muille kirjastoille, riippuvuudet eri tiedostoihin ja modulaarisuus. Lisäksi sovelluskehyksen oppimisen nopeus ja helppous on yksi valintakriteeri. [2]

Työhön valitaan kaksi selaimen puolella toimivaa sovelluskehystä: Angular ja Vue.js. Kumpikin kehitetty sovellus käyttää samaa Node.js:llä tehtyä palvelinta. Tällöin voidaan keskittyä täysin sovelluskehyksien eroavaisuuksiin. Sovellukehykset ovat myös paremmin vertailtavissa, kun kumpikin valituista sovelluskehyksistä on tarkoitettu selaimen puolella toimivien asiakasohjelmien kehitykseen.

Jotta tulokset, etenkin kehityksessä kulutettu aika, eivät vääristyisi, pyritään tietoisesti valitsemaan kaksi erilaista sovelluskehystä. Tällöin vähennetään tilanteita, joissa ensimmäisen sovelluksen kehityksen aikana ilmenneeseen ongelmaan keksitty ratkaisu voitaisiin kopioida ratkaisemaan toisen sovelluksen kehityksen aikana ilmennyt ongelma.

Angular valikoitui ensimmäiseksi verrattavaksi sovelluskehykseksi sen suosion ja modulaarisuuden ansiosta. Angular on suosittu sovelluskehys, ja sitä on käytetty web-sivujen, kuten Microsoft Office Homen kehitykseen [19]. Angularin TypeScript-pohjaisuus ei vaikuttanut valintaan negatiivisesti oppimisen helppouden osalta, sillä JavaScript on ennestään tuttua ja JavaScript on pätevää koodia TypeScript-tiedostossa. Ohjailevuudeltaan Angular on vahva [20].

Vue.js valikoitui toiseksi verrattavaksi sovelluskehykseksi. Vue.js on JavaScript-pohjainen sovelluskehys ja se valikoitui sen suosion ja pieniin projekteihin soveltuvuuden takia. Vue.js:llä on tehty lukuisia web-sivuja, kuten Googlen työnhakualusta Google Careers. [21] Vue.js:n dokumentaatiossa Vue.js:ää kuvaillaan progressiiviseksi sovelluskehykseksi [22]. Progressiivisuus tarkoittaa sitä, että Vue.js:ää voidaan käyttää muiden projektis-

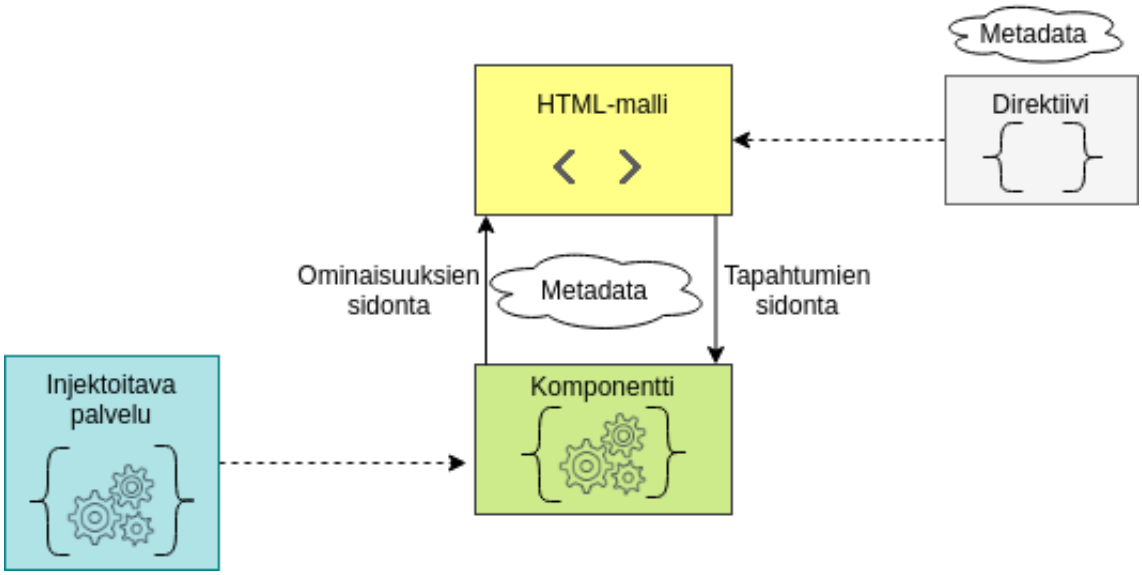
sa sovelluskehysten rinnalla pelkkänä kirjastona tai projekti voidaan kehittää käyttäen vain Vue.js:ää. Myös Vue.js:n erilaisuus Angulariin verrattuna vaikutti valintaan. Verrattuna vahvasti ohjailevaan Angulariin Vue.js on erittäin joustava.

3.4 Angular

Angular on Googlen kehittämä ja ylläpitämä TypeScript-pohjainen selaimen puolella toimiva sovelluskehys. Angular on todella monipuolinen sovelluskehys. Sen TypeScript-kirjastoilla voi toteuttaa web-sovellusten ydintoimintoja, kuten reitityksen, HTTP-liikenteen ja asynkronisten toimintojen luomiseen. Google julkaisi Angularin vuonna 2016 nimellä Angular 2. TypeScript-pohjainen Angular 2 kehitettiin JavaScript-pohjaisen, paremmin AngularJS:nä, tunnetun Angular 1:n pohjalta. Angular 2 syntyi, kun sen kehittäjät kirjoittivat Angular 1:n kokonaan uusiksi. Pyrkimys oli luoda sovelluskehys, joka olisi edeltäjäänsä skaalautuvampi ja modernimpi. [23]

Angular-sovelluksen käyttöliittymä muodostetaan yhdestä tai useammasta pienemmästä komponentista. Komponenteilla käyttöliittymä voidaan organisoida toisistaan riippumattomiin, uudelleenkäytettäviin palasiin, jotka sisältävät käyttämänsä datan ja logiikan. Yksittäinen komponentti muodostetaan siis komponentin käyttämästä logiikasta ja näkymästä eli HTML-mallista (engl. template). Angular-komponentin data ja logiikka sijaitsevat luokan sisällä. Jokainen sovelluksessa käytetyn komponentin luokka kytkeytyy HTML-malliin metadatan avulla. Kuvassa 3.2 näkyvät komponentin luokan ja HTML-mallin välinen yhteys ja niiden välisenä liittimenä toimiva metadata. Luokan tarkoitus on kontrolloida HTML-mallia. [24] Sovelluksen suunnitteluvaiheessa näkymä jaetaan komponentteihin niiden vastuualueiden perusteella. Esimerkiksi yhteystietoja tallentavassa ja näyttävässä sovelluksessa komponentit saatettaisiin jakaa yhteystietojen syötöstä vastaavaan komponenttiin ja yhteystietoja listaavaan komponenttiin. Jokainen näkymä kytketään sovellusprojektin juuressa olevaan index.html-tiedostoon, jolloin näkymät näkyvät yhdessä HTML-sivussa. Kytkemällä näkymät juuren HTML-tiedostoon, ne yhdistyvät projektin DOMiin.

Data ja logiikka, jotka ei liity tiettyyn komponenttiin ja ovat käytössä useammassa komponentissa, sijaitsevat palveluluokassa (engl. service). Komponentit hyödyntävät palveluluokkia riippuvuusinjektiota (engl. dependency injection) käyttäen. Näin voidaan hajauttaa sovelluksen käyttämää logiikkaa esimerkiksi asettamalla kaikki palvelimen kanssa kommunikoivat funktiot palveluluokkaan ja näin jokainen sovelluksen komponentti voi hyödyntää kyseisiä funktioita.



Kuva 3.2. *Angular-komponentin arkkitehtuuri [24]*

Kuvassa 3.2 näkyvät komponentti ja HTML-malli on linkitetty toisiinsa käyttäen kaksisuuntaista datansidontaa (engl. two-way data binding). Kaksisuuntaista datansidontaa on luonnehdittu kuvassa 3.2 HTML-mallin ja komponentin välisinä nuolina. Datansidonnalla HTML-malli on sidottu heijastamaan komponentin ominaisuuksien, kuten muuttujien tilaa ja komponttti puolestaan on sidottu HTML-mallin tapahtumiin.

3.5 Vue.js

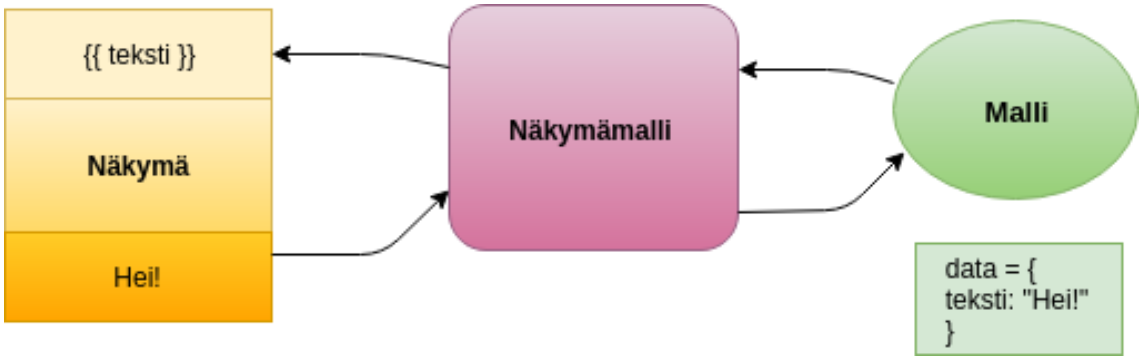
Vue.js on JavaScript-pohjainen selainpuolen sovelluskehys. Se syntyi alunperin tarpeesta kevyelle ja joustavalle kirjastolle, jolla prototyyppien rakentaminen kävisi nopeasti. Vue.js:n vahvuuksina ovat sen yksinkertaisuus ja kyky luoda uudelleenkäytettäviä komponentteja. [25]

Sovelluksen tekoon käytetty Vue.js:n versio 2.0 julkaistiin vuonna 2016. Siitä lähtien se on noussut yhdeksi GitHubin suosituimmista selaimen puolella toimivista sovelluskehyksistä melkein 140 000 kertaa tähtimerkinnän saadulla repositoriolla ja sen suosio kehittäjien keskuudessa on nousussa. [26]

Vue.js noudattaa MVVM-arkkitehtuuria. Kuvassa 3.3 on esitetty miten näkymä, näkymämalli ja malli ovat vuorovaikutuksessa toistensa kanssa. Verrattuna Angularin moniosaisempaan arkkitehtuuriin Vue.js:n määräämä sovellusrakenne on paljon minimalistisempi. Vue.js on siis ohjailevuudeltaan heikko. Vue.js:n alkuperä kevyenä ja joustavana prototyyppikirjastona voidaan tulkita selittävän Vue.js:n yksinkertaisen sovellusrakenteen.

Vue.js:llä on hyvin erilainen lähestymistapa käyttöliittymän osien luontiin verrattuna Angulariin. Sen sijaan, että yksittäinen näkymäkomponentti jaettaisiin kolmeen eri tiedostoon (TypeScript, HTML ja CSS), Vue.js yhdistää kaikki nämä kolme osaa yhteen tiedostoon. Yksi Vue.js-käyttöliittymänäkymä koostuu siis HTML-tukirankaan sijoitetuista `<script>` ja

`<style>` -elementeistä. Komponentin data ja logiikka sijoitetaan `<script>`-elementtiin ja komponentin CSS-tyylittely sijotetaan `<style>`-elementtiin.



Kuva 3.3. *MVVM-malli Vue.js:ssä. [25]*

3.6 Sovelluksen kehitys Angularilla

Tässä alaluvussa kerrotaan Angular-sovelluksen kehityksestä. Alaluvussa käsitellään lyhyiden koodiesimerkkien avulla muutamia kehityksen vaiheita, joissa Angularin määrittelemä ratkaisu ongelmaan tulee esille.

Sovellus on kehitetty Angular 7 -versiota ja Angular CLI -työkalua käyttäen. Työn alustus on vaivatonta, koska Angular CLI -työkalu luo valmiin pohjaprojektin, jonka päälle työ kehitetään. Angularin dokumentaatio on kattava ja yhden tutoriaaliprojektin avulla yhteinäistetty. Koodiesimerkkipätkien lisäksi tutoriaalista tarjolla on myös live-esimerkki, eli selaimessa pyörivä koodieditori, joka sisältää koko dokumentaatiossa esitellyn projektin [27].

Sovelluksen kehitys alkaa määrittelemällä komponenttien vastuut. Päiväkirjasovelluksesta on helposti eroteltavissa kolme eri komponenttia: yksi päiväkirjamerkintöjen tekemiseen, yksi niiden hakemiseen ja yksi haettujen päiväkirjamerkintöjen esittämiseen. Sovellukseen määritellään lisäksi palveluluokka, jonka sisään tehdään palvelimen kanssa kommunikointi. Komponenttien vastuiden määrittelyn jälkeen niille luodaan luokat. Näiden luokkien sisään määritellään muuttujat ja luokan metodit. Tyypillisen metodeja sisältävän luokan lisäksi luokka voidaan tehdä vain tarkoituksena säilöä sinne dataa. Tämänlaista luokkaa ei välttämättä sidota tiettyyn HTML-malliin, vaan luokkaa hyödynnetään pikemminkin objektin tavoin muissa luokissa.

Jotta tekstikenttä, johon käyttäjä syöttää päiväkirjamerkinnän, tallentaisi muuttujaan kentän sisällön, tekstikenttä on linkitettävä muuttujaan kaksisuuntaisella datansidonnalla. Angularissa datansidonta on toteutettu NgModelilla. NgModel käyttää hakasulkeita ja kaarisulkeita datansidonnan suunnan asettamiseen. Hakasulkeilla asetetaan data kulkemaan komponentilta näkymän elementtiin. Vastavuoroisesti kaarisulkeilla datan asetetaan kulkemaan näkymältä komponentille. Jos kaarisulkeet on asetettu hakasulkeiden sisään, data liikkuu kumpaankin suuntaan eli datansidonta on kaksisuuntaista. Alapuolella olevas-

sa esimerkissä käyttäjän syöttämä teksti päivitetään reaaliaikaisesti `content`-nimiseen muuttujaan. Kaksisuuntaiseen datansidontaan käytetään siksi, että tekstikenttä saadaan helposti tyhjennettyä komponentin logiikan puolelta asettamalla `content`-muuttujan arvo tyhjäksi.

```
<textarea rows="8" cols="45"
    [(ngModel)]="content"
    placeholder="Write something..."></textarea>
```

Hyvin tärkeä ominaisuus päiväkirjasovellukselle on päivämäärän näyttö ja tallentaminen. Jotta päivämäärän käsittely olisi helppoa, käytetään sovelluksessa JavaScriptin `Date`-luokkaa. `Date`-luokan rakentajan avulla muuttujaan voidaan alustaa `string`-muodossa selaimen sen hetkinen päivämäärä ja aika. `Date`-luokalla alustetun muuttujan tarkkuus on kuitenkin turhan tarkka päiväkirjasovellusta varten, joten tarvitaan ratkaisu päivämäärän esitystavan muuttamiseen. Angularilla tämä ratkaistaan käyttämällä putkea (engl. pipe).

Alla on esimerkki sovelluksessa käytetystä päivämäärän esitystavan muokkauksesta putken avulla. Angularilla putki määritellään siis yksinkertaisesti HTML:n sisään asettamalla pystyviiva muuttujan ja putken väliin ja kertomalla putkelle käsiteltävä muuttuja ja muuttajan muotoilutapa. [28] Tässä tapauksessa muuttuja muokataan esittämään vain päivämäärä, kuukausi ja vuosi.

```
<h6>{{date | date:"fullDate"}}</h6>
```

Koska yhden sivun sovellukset kommunikoivat palvelimen kanssa asynkronisesti, tarvitaan sovelluksen puolelle mekanismi, joka toteuttaa asynkronisuuden. Angular tarjotaa ratkaisuksi `Obervables`-luokkaa. `Observables`-luokka välittää viestejä julkaisijan (engl. publisher) ja tilaajan (engl. subscriber) välillä. Päiväkirjasovelluksessa julkaisijaksi asetetaan muuttuja, johon tallennetaan palvelimelta saadut päiväkirjamerkinnät. [29] Toteutuksessa käytetty `BehaviorSubject` on `Observables`-luokan tavoin toimiva viestinvälittäjä, ainoana erona on, että `BehaviorSubject` vaatii alustuksen. Palvelimen kanssa kommunikoivat funktiot sijaitsevat injektoitavassa palveluluokassa, jotta muut luokat voivat halutessaan käyttää näitä funktioita.

Alla on työssä käytetyn viestinvälittäjän toteutus. Viestinvälittäjä huolehtii, että HTTP-pyynnön toimittama päiväkirjamerkinnät sisältävän data siirtyy palveluluokasta päiväkirjamerkinnät listaavaan luokkaan. Viestinvälittäjässä julkaisijoita ovat `dataSource`-muuttuja, `currentData`-muuttuja ja myöhemmin esille tuleva `observable`-muuttuja. Tilaajina ovat päiväkirjamerkintöjä listaava luokka ja myöhemmin esitelty nuolifunktio. Toteutuksessa `dataSource`-muuttuja asetetaan `asObservable`-metodin avulla tarkkailtavaksi ja tallennetaan `currentData`-muuttujaan, jotta siinä tapahtuvat muutokset voidaan tilata muualla koodissa.

```
private dataSource = new BehaviorSubject({})
currentData = this.dataSource.asObservable();
```

Tarkkailun kohde, `dataSource`, palauttaa aina siihen tallennetun arvon, kun sitä käsitellään `next`-metodilla. HTTP-pyynnön vastaanottava muuttuja `observable` on kuuntelun kohteena, kun sen `subscribe()`-metodi suoritetaan. Kun palvelin vastaa HTTP-pyyntöön, jokainen `observable`-muuttujan tilannut taho saa tiedon, että HTTP-pyynnön vastaus on saapunut selaimelle. Tässä tapauksessa alla oleva nuolifunktio on tilannut `observable`-muuttujassa tapahtuneet muutokset ja funktio suoritetaan heti HTTP-pyynnön vastauksen saapuessa.

```
var observable = this.http.get('http://localhost:3000/entry')
observable.subscribe((response) => {
    this.dataSource.next(response)
})
```

HTTP-pyynnön vastauksena saatu päiväkirjamerkinnät sisältävä data kulkee palveluluokasta päiväkirjamerkinnät listaavaan luokkaan, kun merkinnät listaavassa luokassa tilataan `subscribe`-metodilla `currentData`-muuttujassa tapahtuvat muutokset.

Käyttäjän aikaisemmat päiväkirjamerkkinnät esitetään listana selkeästi luettavassa muodossa. Päiväkirjamerkintöjä listaava luokka saa päiväkirjamerkinnät objekteja sisältävänä taulukkona (engl. array). Taulukon objektien esittämiseen Angular tarjoaa alla näkyvää lista-alkioon upotettavaa `*ngFor`-funktiota, joka käy taulukon läpi ja luo lista-alkion jokaiselle taulukon alkiolle. [30]

```
<ul class="entries">
    <li *ngFor="let entry of entries">
        <p>{{entry.date | date:"fullDate"}}</p>
        <p>{{entry.content}}</p>
    </li>
</ul>
```

3.7 Sovelluksen kehitys Vue.js:llä

Tässä alaluvussa käydään läpi sovelluksen kehityksen vaiheita, joissa Vue.js:n määräämät ratkaisut ongelmiin tulevat esiin. Ratkaisut esitellään lyhyiden koodiesimerkkien avulla. Ne konseptit, jotka on selitetty jo edellisessä luvussa, käydään läpi lyhyemmin tässä luvussa.

Sovellus on kehitetty Vue.js 2.6.6.-versiolla Vue CLI -työkalua käyttäen. Työn alustus on vaativaa, sillä Atom-koodieditori ei tunnista Vue.js:n .vue-päätteisiä tiedostoja. Ongelmaan löytyy kuitenkin ratkaisu ja asentamalla Vuen sisältävän paketin, sovelluksen kehitys on mahdollista tehdä Atom-koodieditorilla. Vue.js-sovelluksen käyttöliittymien komponenttien määrittely on samanlainen kuin Angular-sovelluksen komponenttien määrittely. Näkymä koostuu toistamiseen kolmesta komponentista: päiväkirjamerkintäkoponentista, hakukomponentista ja päiväkirjamerkintöjen listauskomponentista.

Ensimmäisenä sovellukseen kehitetään päiväkirjamerkintäkomponentti, joka ottaa vastaan käyttäjän syöttämän tekstin. Taas kerran toteutukseen tarvitaan kaksisuuntaista datansidontaa. Vue.js:llä tämä toteutetaan käyttäen `v-model`-direktiiviä. Se tallentaa `content`-muuttujaan tekstikentän arvon uudestaan jokaisen muutoksen kohdalla. [31] Tämä näkyy alla olevassa koodissa.

```
<textarea rows = "8" cols = "45"
v-model = "content"
placeholder = "Write something..."></textarea>
```

Päivämäärän esitystyyliin Vue.js ei tarjoa sovelluskehykseen sisällettyä ratkaisua, vaan esitystyyli muokataan käyttäen filtteriä. Filtteri koostuu HTML-direktiivistä ja funktiosta. [32] Funktion sisässä tehdään päivämäärän näyttötavan muutos Moment.js-kirjaston avulla. Alla olevassa koodissa näkyy filtterin käyttämä funktio.

```
filters: {
    formatDate: function (date) {
        if (date) {
            return moment(String(date)).format('LL')
        }
    }
}
```

Filtterin käyttö HTML:n sisällä on toteutettu hyvin samanlaisella syntaksilla kuin Angularissa. Tässä tapauksessa pystyviivan jälkeen tuleekin filtterin sisäinen metodi `formatDate`. Tämä näkyy alla olevassa koodissa.

```
<h6> {{date | formatDate}} </h6>
```

Päiväkirjamerkintöjen listaukseen käytetään hyvin samanlaista tapaa kuin Angularissa. Vue.js:n `v-for`-funktio käy jokaisen taulukon alkion yksitellen läpi, ja alkion arvo esitetään direktiivin avulla. Alla olevassa koodissa näkyy päiväkirjamerkintöjä läpi käyvä funktio. [33]

```
<ul class = "entries">
    <li v-for = "entry in entries" :key = "entry._id">
        {{entry.date | formatDate}}
        {{entry.content}}
    </li>
</ul>
```

Jotta päiväkirjamerkintöjä listaava komponentti saisi tiedon siitä, että käyttäjä on painanut päiväkirjamerkintöjä hakevassa komponentissa nappia, on tieto välitettävä komponentilta toiselle. Vue.js tarjoaa paria eri ratkaisua erillisten komponenttien väliseen kommunikaatioon. Tässä sovelluksessa käytetään `EventBus`-komponenttia. `EventBus` luodaan alustamalla se globaalisti käytettävällä `Vue`-luokan instanssilla. Tämän jälkeen `EventBus`

voidaan ottaa käyttöön jokaisessa sovelluksen komponentissa. Alla olevassa koodissa näkyy `EventBus`-muuttujan alustus `Vue`-luokan instanssilla. [34]

```
const EventBus = new Vue();
```

`EventBus` emittoi ja vastaanottaa tapahtumia, jolloin sitä voidaan käyttää komponenttien väliseen tapahtumaperusteiseen viestintään. Alla olevassa esimerkissä `EventBus` emittoi napin painalluksen.

```
EventBus.$emit('click')
```

Alla olevassa esimerkissä `EventBus` kuuntelee napin painalluksen aiheuttamaa tapahtumaa. Kun kuuntelija havaitsee tapahtuman, se suorittaa `getEntries()`-funktion.

```
EventBus.$on('click', () => {
    this.getEntries()
})
```

4 TULOKSET JA ARVIOINTI

Tässä luvussa käydään läpi tulokset ja arvioidaan tuloksia ja syitä, jotka mahdollisesti vaikuttavat tuloksiin ja niiden tulkintaan. Ensin käsitellään kehitykseen kulutettua aikaa, sitten kehityksen aikana kohdattuja ongelmia ja viimeisenä käsittelyssä on koodirivien määrä. Sovelluksien kehityksestä saadut tulokset on koottu taulukkoon 4.1.

Verrattuna Vue.js-sovelluksen kehitykseen kulutettuun aikaan Angular-sovelluksen kehitykseen kulutettu aika on kolme ja puoli tuntia pidempi. Tulos ei yllätä, sillä Angular-sovelluksen kehityksen aikana moni yhden sivun sovelluksen kehitykseen liittyvä tekniikka, kuten datansidonta ja asynkronisen kommunikoinnin totetus, olivat tällöin uusia tekniikoita Angular-sovelluksen ollessa ensimmäinen kehittämäni yhden sivun sovellus. Vue.js-sovelluksen toteutus taas oli suoraviivaisempaa, sillä kehityksen aikana oli selkeää, millaisia tekniikoita työssä tultaisiin todennäköisesti tarvitsemaan ja kehityksen alussa kaikki mahdollisesti ongelmia aiheuttavat asiat olivat jo tiedossa. Sovellusta tehdessä aikaa veivät myös JavaScriptistä johtuvien ongelmien ratkaisu. Tällaisin ongelmia olivat this-avainsanan käyttö ja nuolifunktiot, joiden käyttö Angularin dokumentaatiossa oli erittäin yleistä.

Vue.js-sovelluksen kehityksen aikana ongelmia ilmeni kahdeksan kappaletta. Ongelmien määrien välinen ero voidaan osaksi selittää sovelluskehysten dokumentaatiolla ja sovelluskehyksen takana olevan organisaation tarjoamalla tuella. Angularin dokumentaatio ja sen tarjoamat esimerkit olivat oleellinen osa ongelmien välttämistä, sillä varsinkin luvussa 3.6. mainittu live-esimerkki ohjasi kehitystä oikeaan suuntaan. Vastakohtana Angularin kattavalle dokumentaatiolle oli Vue.js:n dokumentaatio, joka tarjosi suppeita esimerkkejä sovelluskehyksen käyttöön ja tämä teki ohjelman kokonaiskuvan hahmotuksesta vaikeaa. Lisäksi dokumentaation ja sovelluskehystä käyttävän yhteisön esimerkit ja tavat käyttää sovelluskehystä poikkesivat toisistaan esimerkiksi komponenttien luonnin osalta, jolloin yhteisön esimerkkien seuraaminen tarkoitti sitä, että dokumentaation esimerkeistä oli yhä vähemmän apua.

Taulukko 4.1. *Sovellusten kehityksestä saadut tulokset*

sovelluskehys	kulutettu aika (h)	ongelmat	koodirivit
Angular	15	5	108
Vue · js	11,5	8	91

Angular-sovelluksen kehityksen aikana ongelmia ilmeni yhteensä viisi kappaletta. Ensimmäinen ongelma koski luokkien käyttöä. Sovelluksen kehityksen alussa luotiin päiväkirjamerkintäolion kaltainen luokka pelkästään päiväkirjamerkintöjen datan säilömistä varten. Tämä mutkisti sovelluksen datan sitomista näkymiin. Toinen ongelma oli komponenttien asynkronisen kommunikoinnin luominen. Angularin dokumentaaio ei kerro, miten komponentit kommunikoivat asynkronisesti, joten ratkaisu piti rakentaa monien eri lähteiden ratkaisujen perusteella. Kolmas ongelma oli asynkronisen kommunikaatio toteutuksessa `BehaviourSubject`-luokkaa käyttäen. Etenkin tilaajien ja julkaisijoiden toisiinsa linkittämisen monivaiheisuus aiheutti ongelmia. Monivaiheisuus on nähtävissä luvun 3.6 koodiesimerkeissä. Neljäs ongelma oli *BehaviourSubject*-luokan alustuksen tyypitys. Jostain syystä objektin vastaanottava `BehaviourSubject`-muuttuja ei hyväksynyt objektia tyypikseen. Viides ja viimeinen ongelma oli POST-pyynnön lähetys palvelimelle.

Vue.js-sovelluksen kehityksen aikana ongelmia ilmeni yhteensä kahdeksan kappaletta. Ensimmäinen ongelma oli projektin alustus. Kehityksen aloittaminen vaati lisäyksiä koodieditoriin, sillä Atom ei tue Vue.js:n .vue-päätteisiä tiedostoja. Toinen ongelma oli komponenttien luominen. Vue.js:n dokumentaatiossa on esitelty kattavasti vain toinen kahdesta tavasta luoda komponentteja. Dokumentaatiossa esitelty ja käytetty tapa on globaalin komponentin tekoon tarkoitettu ja päiväkirjasovelluksessa käytettiin vain lokaaleja komponentteja, minkä takia kehityksessä oli luotettava dokumentaation sijaan enemmän yhteisön tekemiin ratkaisuihin. Kolmas ongelma oli päättää, miten komponenttien sisäinen data säilötään. Dokumentaatio esittelee kaksi tapaa säilöä dataa, `props`- ja `data`-objektit. Pian selvisi kuitenkin, että `props`-objektia käytetään vanhempikomponentin ja sen lapsikomponentin väliseen kommunikaatioon, mitä ei tarvittu kehitettyy sovellukseen. Sen jälkeen, kun `props`-objekti oli karsittu mahdollisista datansäilömispaikoista ilmeni kehityksen neljäs ongelma, joka oli päättää säilötäänkö data objektiin vai käytetäänkö datan säilömiseen funktiota. Viides ongelma oli päivämäärän esitystyylin muokkaus. Vue.js ei tarjonnut sisäänrakennettua mekanismia, joten toteutuksessa piti turvautua ulkoiseen Moment.js-kirjastoon. Kuudes ongelma tuli Moment.js-kirjaston käytöstä. Kirjaston käyttö JavaScriptin `Date`-luokan kanssa aihetti varoituksen kirjaston ja *Date*-luokan yhteensopivuudesta. Seitsemäs ongelma oli komponenttien välisen kommunikaation toteuttaminen. Ongelma tässä tapauksessa oli lähinnä eri toteutustapojen lukumäärä. Kahdeksas ja viimeinen ongelma oli dokumentaation ja yhteisön tarjoamien ratkaisujen yhteensovittaminen.

Koodirivien perusteella Vue.js on kirjoitustehokkuudeltaan parempi. Yksittäinen toteutettu toiminnallisuus Angular-sovelluksessa ei ole yksin vastuussa korkeammasta rivimäärästä. Ei siis voida tarkasti osoittaa, että jonkin tietyn toiminnallisuuden toteutus olisi rivimäärältään merkittävästi suurempi, kuin Vue.js-sovelluksen vastaavan toiminnallisuuden toteutus.

5 YHTEENVETO

Työssä vertailtiin kahta yhden sivun sovelluksen kehitykseen tarkoitettua sovelluskehystä. Sovelluskehyksiksi valikoituivat Angular ja Vue.js. Vertailu tehtiin sovelluskehyksillä kehitettyjen sovelluskehysten pohjalta. Arviointikriteereinä olivat sovelluksen kehitykseen kulutettu aika, kehityksen aikana kohdatut ongelmat ja kirjoitettujen koodirivien määrä. Kehitettävä sovellus oli yksinkertainen päiväkirjasovellus, jolla luodaan päiväkirjamerkintöjä, tallennetaan ne palvelimelle ja haetaan vanhat päiväkirjamerkinnät tarkasteltavaksi.

Tulosten vertailun pohjalta Vue.js on kehittäjäystävällisempi sovelluskehys. Vue.js:n puolesta puhuvat sen kanssa kehitykseen käytetty aika ja kirjoitustehokkuus eli koodirivien määrä. Tulosten arvioinnin perusteella Angular-sovelluksen kehitykseen kulutettu aika on kuitenkin hieman vääristynyt, sillä aika sisältää myös yhden sivun sovelluksen kehitykseen kuuluvien yleisten konseptien opettelua. Jotta tulos kertoisi enemmän itse sovelluskehyksestä, ennen Angular-sovelluksen kehitystä olisi pitänyt kehittää "lämmittelysovellus".

Valitut sovelluskehykset ovat ohjailevuudeltaan erivahvuisia. Ohjailevuus vaikuttaa erityisesti sovelluskehyksen käytön oppimiseen, mikä kuluttaa kehittäjän aikaa. Toisaalta ohjaileva kehys säästää kehittäjää sekaannuksilta ongelmien ratkaisussa, kun oikeita ratkaisuja on rajattu määrä. Saatujen tulosten perusteella on myös mahdollista pohtia, soveltuuko vahvasti ohjaileva sovelluskehys pienen ja yksinkertaisen sovelluksen, kuten työssä kehitetyn päiväkirjasovelluksen kehitykseen. Laajentamalla kehitettävän sovelluksen vaatimuksia tulokset kertoisivat paremmin itse sovelluskehyksistä sen sijaan, että ne kertoisivat sovelluskehyksen soveltuvuudesta pienen sovelluksen kehitykseen.

Arviointikriteerejä lisäämällä esimerkiksi dokumentaation kattavuuden arviointiin kertoisi enemmän kehittäjälle tärkeästä sovelluskehyksen kehittäjien tarjoamasta tuesta. Käytännössä dokumentaation kattavuutta voitaisiin mitata laskemalla se ongelmien osuus, jossa dokumetaatio ratkaisi ongelman.

LÄHDELUETTELO

[1] M. Galli, R. Soares ja I. Oeschger. *Inner-browsing extending the browser navigation paradigm*. 2003. URL: https://developer.mozilla.org/en-US/docs/Archive/Inner-browsing_extending_the_browser_navigation_paradigm (viitattu 08. 02. 2019).

[2] F. Monteiro. *Learnin Single-page Web Application Development*. Packt Publishing Ltd., 2014, pp. 9–1.

[3] W. Ezell. *What is a Single Page Application? (And Should You Use One?)* 2018. URL: https://dotcms.com/blog/post/what-is-a-single-page-application-and-should-you-use-one- (viitattu 08. 02. 2019).

[4] A. Mesbah. Analysis and Testing of Ajax-based Single-page Web Applications. Delft: Delft University of Technology, 19. kesäkuuta 2009, p.189.

[5] E. Scott. *SPA Design and Architecture: Understanding Single Page Web Applications*. Manning Publications, 2015, pp. 3–22.

[6] N. T. S. Ryan Asleson. *Ajax. Tehokas hallinta*. readme.fi, 2006, s. 14–15.

[7] J. J. Garrett. Ajax: A New Approach to Web Applications (2005). URL: https://adaptivepath.org/ideas/ajax-new-approach-web-applications/ (viitattu 08. 02. 2019).

[8] *Ajax*. Mozilla. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX (viitattu 20. 04. 2019).

[9] *What is AJAX?* Tutorialspoint. 2019. URL: https://www.tutorialspoint.com/ajax/what_is_ajax.htm (viitattu 08. 02. 2019).

[10] D. Flanagan. *JavaScript: The Definitive Guide*. 5. painos. O'Reilly, 2006, pp. 1–2.

[11] *What is JavaScript?* Mozilla. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript (viitattu 24. 03. 2019).

[12] P. Deeleman. *Learning Angular 2*. Packt Publishing, 2016, 39–40.

[13] *Basic Types*. Microsoft Corporation. 2019. URL: https://www.typescriptlang.org/docs/handbook/basic-types.html (viitattu 19. 04. 2019).

[14] V. Gaudioso. *Foundation Expression Blend 4 with Silverlight*. Apress, 2010, 341–367.

[15] A. Ryabtsev. *Web Frameworks: How To Get Started*. URL: https://djangostars.com/blog/what-is-a-web-framework/ (viitattu 28. 02. 2019).

[16] K. Bedell. Opinions on Opinionated Software. *Linux Journal* (2016). URL: https://www.linuxjournal.com/article/8686 (viitattu 06. 04. 2019).

[17] *Express - Node.js web application framework*. Node.js Foundation. 2017. URL: https://expressjs.com/ (viitattu 06. 04. 2019).

[18] F. Copes. *A list of sample Web App Ideas*. 18. helmikuuta 2018. URL: https://flaviocopes.com/sample-app-ideas/#a-personal-diary-app (viitattu 24. 03. 2019).

[19] L. Polepeddi. *Made With Angualar*. 2019. URL: https://www.madewithangular.com/categories/angular/ (viitattu 23. 03. 2019).

[20] F. Lardinois. Google launches final release version of Angular 2.0. *TechCrunch* (2016). URL: https://techcrunch.com/2016/09/14/google-launches-final-release-version-of-angular-2-0/?guccounter=1 (viitattu 17. 04. 2019).

[21] M. M. Armin Ulrich. *Websites Made With Vue.js*. 2018. URL: https://madewithvuejs.com/websites?page=2 (viitattu 02. 04. 2019).

[22] *Introduction*. Vue Team. 2019. URL: https://vuejs.org/v2/guide/ (viitattu 02. 04. 2019).

[23] D. Jabif. Learning Angular: What is Angular? (2019). URL: https://angular-templates.io/tutorials/about/learn-angular-from-scratch-step-by-step (viitattu 23. 04. 2019).

[24] *Architecture*. Google. 2019. URL: https://angular.io/guide/architecture (viitattu 12. 03. 2019).

[25] O. Filipova. *Learning Vue.js 2*. Packt Publishing, 2016, p. 334.

[26] *Vue.js GitHub repository*. Vue Team. 2019. URL: https://github.com/vuejs/vue (viitattu 21. 04. 2019).

[27] *Live example of an Angular project*. Google. 2019. URL: https://stackblitz.com/angular/jevqamnqkxr (viitattu 31. 03. 2019).

[28] *Pipes*. Google. 2018. URL: https://angular.io/guide/pipes (viitattu 05. 04. 2019).

[29] *Observables*. Google. 2018. URL: https://angular.io/guide/observables (viitattu 05. 04. 2019).

[30] *Displaying Data*. Google. 2018. URL: https://angular.io/guide/displaying-data (viitattu 05. 04. 2019).

[31] *Form Input Bindings*. Vue Team. 2019. URL: https://vuejs.org/v2/guide/forms.html (viitattu 27. 04. 2019).

[32] *Filters*. Vue Team. 2019. URL: https://vuejs.org/v2/guide/filters.html (viitattu 27. 04. 2019).

[33] *List Rendering*. Vue Team. 2019. URL: https://vuejs.org/v2/guide/list.html (viitattu 27. 04. 2019).

[34] A. Abrickis. List Rendering (2017). URL: https://medium.com/@andrejsabrickis/https-medium-com-andrejsabrickis-create-simple-eventbus-to-communicate-between-vue-js-components-cdc11cd59860 (viitattu 27. 04. 2019).