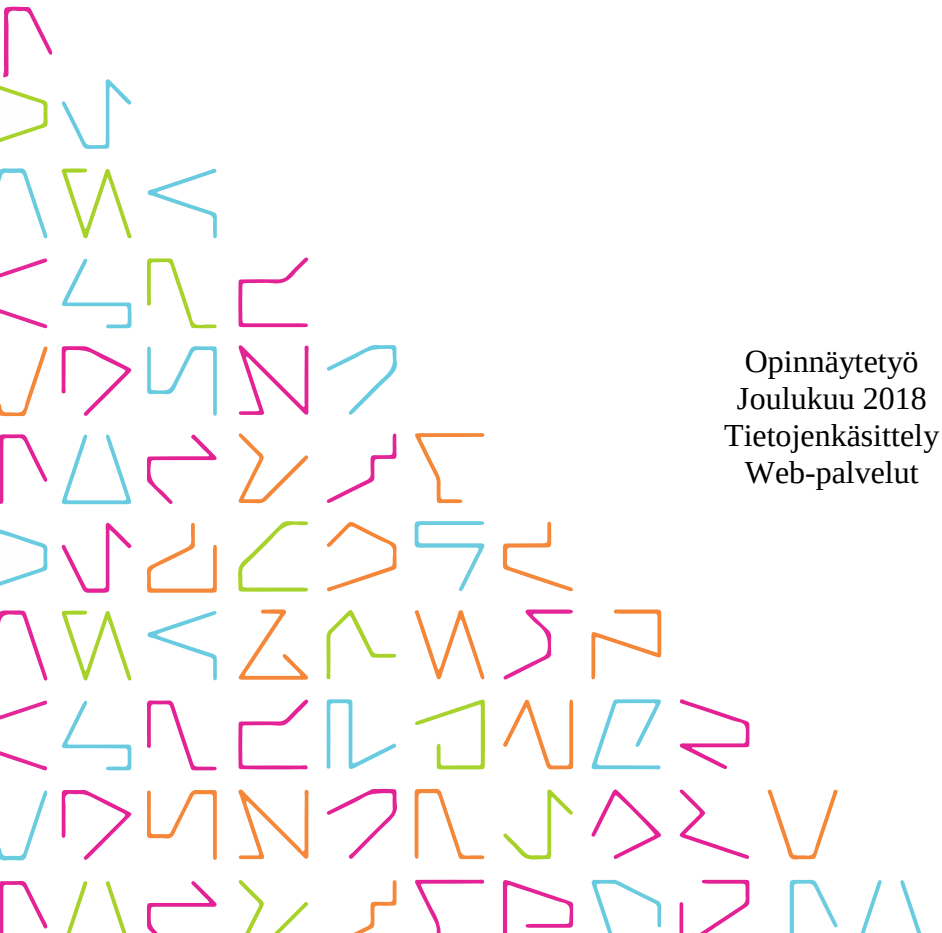
TAMPEREEN
AMMATTIKORKEAKOULU

WEB-KÄYTTÖLIITTYMÄTEKNIIKOIDEN SELVITYSTYÖ

Case Remion Oy

Tuomas Ukkola



Opinnäytetyö
Joulukuu 2018
Tietojenkäsittely
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

UKKOLA, TUOMAS:
Web-käyttöliittymätekniikoiden selvitystyö
Case Remion Oy

Opinnäytetyö 41 sivua
Joulukuu 2018

Web-sovellukset ovat suositumpia kuin koskaan ennen digitalisaation vuoksi. Web-sovelluksien kehittäminen ei kuitenkaan ole suoraviivaista ja sovellus voi jäädä jälkeen parhaiden käytäntöjen ja tekniikoitten kannalta. Tässä opinnäytetyössä vertailtiin muutamaa ajankohtaista web-sovelluskehystä. Tavoitteena oli löytää paras mahdollinen web-sovelluskehys Remionilla kehitettävien tuotteiden kannalta.

Opinnäytetyön tuloksiin kuuluu koodiesimerkkejä vertailussa esiteltyjen sovelluskehysten käytöstä nykyisen rinnalla, mutta koska ne sisältävät luottamuksellisia tietoja, kyseiset esimerkit ovat poistettu julkisesta raportista. Julkinen tulos on opinnäytetyön tekijän päätelmä siitä, mikä on sopivin sovelluskehys käytettäväksi Remionin tuotteissa.

Ohjelmistosala kehittyy niin nopeasti, että vuoden tai parin takaiset ratkaisut voivat hyvinkin olla jo vanhentuneita. Kuitenkin tulevaisuuden standardit, kehittäjäyhteisöt ja jo vakiintuneet sovelluskehykset tuovat vakautta web-alustalle.

Asiasanat: web-sovellukset, JavaScript, sovelluskehykset

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Web Services

UKKOLA, TUOMAS:
Research of Web User Interface Technologies
Case Remion Oy

Bachelor's thesis 41 pages
December 2018

Web applications power our day-to-day lives. Developing web applications that meet the current requirements, however, is very complicated and time-consuming. Thus, software can lag behind in terms of practices and technologies used. The objective of this thesis was to find what modern web frameworks could be used in products of Remion Oy, the commissioner of this thesis. In order to find an answer to that question, a few modern web frameworks were investigated and compared.

The results of this thesis include code examples, which could not be made public. The public report consists of the conclusion of what the future web frameworks should be.

The subject is difficult because the software industry changes so rapidly. Future standards, communities and mature frameworks should, however, bring stability to a web platform.

Key words: web applications, JavaScript, frameworks

SISÄLLYS

LYHENTEET JA TERMIT

AJAX	Asynchronous JavaScript and XML. Joukko teknologioita, jotka sallivat palvelimen ja asiakkaan kommunikoinnin asynkronisesti.
API	Application Programming Interface. Ohjelmointirajapinta.
CSS	Cascading Style Sheets. Web-dokumenttien tyyliohje.
DOM	Document Object Model. Ohjelmoitava kuvaus HTML tai XML dokumentista.
ECMAScript	Spesifikaatio yleiskäyttöisen ohjelmointikielen luomiseksi.
GraphQL	Palvelinpäässä toimiva kyselykieli.
HTML	Hypertext Markup Language. Avoimen standardin mukainen kuvauskieli web-dokumenteille.
HTTP	Hypertext Transfer Protocol. Protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon
JavaScript	Ohjelmointikieli, joka on ECMAScript-spesifikaation määritysten mukainen. Tunnetaan parhaiten verkkoselaimeen upotettuna ohjelmointikielenä.
JSX	JavaScript XML. XML-syntaksia muistuttava laajennus ECMAScript-syntaksiin.
LTS	Long Term Support. Ohjelmiston elinkaaren vaihe, jossa päivitykset sisältävät vain virheenkorjauksia.
REST	Representational State Transfer. Arkkitehtuurityyli, joka on rakennettu HTTP-protokollan päälle.
SaaS	Software as a Service. Sovelluksen jakelumalli, jossa sovellus tarjotaan käyttäjille tietoverkkojen yli palvelumaksua vastaan.
TypeScript	Tyypitetty ohjelmointikieli, joka on rakennettu JavaScriptin päälle.
VDOM	Virtual DOM. Virtuaalinen esitys DOMista, jota käytetään DOM-operaatioiden laskemiseen.
WebSocket	Teknologia, jolla asiakas ja palvelin voivat kommunikoida keskenään reaaliajassa.
XML	Extensible Markup Language. Metakieli, jolla määritellään rakenteellisia merkkauskieliä.

XMLHttpRequest	Web-API, jolla voidaan noutaa tietoja palvelimelta.

1 JOHDANTO

Vuonna 2018 Stack Overflow:n teettämän kyselyn mukaan JavaScript on toiseksi halutuin ja seitsemänneksi rakastetuin ohjelmointikieli (Stack Overflow 2018). Se ei ole mikään ihme, koska JavaScriptiä voidaan käyttää millä tahansa päätelaitteella. Siitä syystä JavaScriptiä voidaan kutsua web-alustan omaksi kieleksi (Tunguz 2013). Internet ei enää pelkästään koostu yksinkertaisista web-sivuista vaan myös täysimittaisista sovelluksista, mikä tuo aivan uudenlaisia haasteita web-alustalle ja -kehittäjille.

Viimeisten 10 vuoden aikana on kehitetty monia JavaScript-sovelluskehyksiä kehittäjien työn helpottamiseksi. Jotkut niistä ovat saavuttaneet suuren suosion, jotkut ovat nähneet takaiskuja ja muutama on merkityksellinen edelleen. Opinnäytetyön toimeksiantajan eli Remion Oy:n ja sen tuotteiden puolesta on kuitenkin aika arvioida käytettyjä web-sovelluskehyksiä ja selvittää mitä kehyksiä toimeksiantajan tuotteissa kannattaisi tulevaisuudessa käyttää. Selvitystyö tapahtuu kartoittamalla nykytilanne ja pohtimalla mitä voitaisiin tehdä paremmin. Lopuksi muutamaa ajankohtaista web-sovelluskehystä tutkitaan ja vertaillaan.

Opinnäytetyön raportin luvussa kaksi esitellään lyhyesti toimeksiantaja sekä web-sovelluksien historia ja nykytilanne. Luvussa kolme avataan tarkemmin taustaa toimeksiantajan tilanteesta ja yhdestä toimeksiantajan käyttämästä web-sovelluskehyksestä sekä käydään läpi koko opinnäytetyöprojektin motivaatio. Luvussa neljä esitellään vertailuun valitut sovelluskehykset. Lopuksi luvussa viisi esitellään tulokset ja luvussa kuusi johtopäätökset ja pohdinta.

Vertailussa vertaillaan, kuinka helppoa uutta kehystä on käyttää nykyisen rinnalla, millainen kehyksen ekosysteemi on sekä millainen kehittäjäyhteisö ja -kokemus on. Vertailuun valittiin React, Angular ja web-komponenttistandardi, joista viimeinen ei varsinaisesti ole sovelluskehys vaan standardi, jolla voidaan tehdä samoja asioita, joita sovelluskehyksetkin tekevät.

Työn käytännön osuudessa erääseen toimeksiantajan tuotteeseen valittiin ominaisuus, joka toteutettiin jokaisella vertailuun valitulla sovelluskehyksellä. Ominaisuus rajattiin

siten, että se ei ole liian aikaa vievä toteuttaa mutta ei myöskään liian pintapuolinen. Kehitettävän ominaisuuden piti siis hyödyntää mahdollisimman laajasti sekä sovelluskehykselle että tuotteelle tyypillisiä ominaisuuksia. Tämän ominaisuuden toteutuksia ei opinnäytetyön julkiseen raporttiin voitu sisällyttää, vaan ne ovat korvattu yleisillä esimerkeillä luvuissa 4.1.4, 4.2.4 ja 4.3.4.

2 TAUSTA

2.1 Remion Oy

Remion Oy on vuonna 2001 perustettu tamperelainen tuotetalo, jonka erikoisalaa on teollisen internetin ratkaisut ja palvelut. Remion aloitti vaatimattomista oloista alihankkijan roolissa ja samalla ryhtyi kehittämään tuoteperhettä, joka nykyään tunnetaan nimellä Regatta. Vuosia myöhemmin, tarkemmin vuonna 2015, fokus vaihtui SaaS-palveluihin, ja Remion lanseerasi Regatta Portaalin. SaaS eli Software as a Service tarkoittaa sovelluksen jakamista palveluna, kuten esimerkiksi Google Docs.

2.2 Web-sovellukset nykypäivänä

Jos estät käyttämästäsi selaimestasi JavaScriptin suorittamisen tänä päivänä, on hyvin todennäköistä, että törmäät ilmoitukseen, jossa selaimesta kehoitetaan sallimaan JavaScriptin suorittaminen tai haluamasi sivu ei lataudu ollenkaan. Näin ei vielä ollut 20 vuotta sitten, vaan jokaisen käyttäjän suorittaman toiminnon jälkeen sivu jouduttiin lataamaan uudestaan ja samalla edellinen pyyhittiin pois muistista. Tilanhallinta ja bisneslogiikka siis sijaitsi palvelinpäässä. (Nutter 2017.)

Vuonna 2000 Internet Explorer 5.0 -selain julkaistiin ja yksi sen uusista ominaisuuksista oli nimeltään XMLHttpRequest (Swartz 2005). Muutamia vuosia myöhemmin kyseinen ominaisuus ja muutama muu muodostivat termin AJAX eli Asynchronous Javascript and XML (Garret 2005). Nämä tekniikat yhdessä sallivat sivun päivittämisen ilman sivun uudelleen latausta.

Nykyään web-sivut, tai paremminkin web-sovellukset, ovat niin monimutkaisia ja kehittyneitä, että niiden kehittämiseen on luotu suuri määrä erilaisia työkalua. Luonnollisesti tekniikat ovat myös kehittyneet ja uusiakin on syntynyt ajan saatossa. Kuitenkin yleisin tapa tänä päivänä tuottaa web-sovellus on käyttäen jotain JavaScript-sovelluskehystä. Kehystä hyödyntäen rakennetaan käyttöliittymä, joka huomaamattomasti kommunikoi palvelimelle.

Kommunikointi tavallisesti tapahtuu REST APIa käyttämällä. REST eli Representational State Transfer on arkkitehtuurityyli, joka on rakennettu HTTP-protokollan päälle. API eli Application Programming Interface tarkoittaa ohjelman rajapintaa, joka on yhteydessä palvelimeen tai tietokantaan. (Kivisaari 2016.) Jos sovellus on hyvin moderni voi kommunikointi tapahtua esimerkiksi GraphQL:n tai WebSocketin välityksellä. GraphQL on kyselykieli, joka suoritetaan palvelinpäässä ohjelman suorituksen aikana (GraphQL n.d.). WebSocket tarkoittaa teknologiaa, jolla asiakas ja palvelin voivat kommunikoida keskenään reaaliajassa (MDN web docs n.d. a). Tämä palvelin, joka web-sovelluksen kaikille tarjoaa, yleisesti sijaitsee niin sanotusti pilvessä. On myös mahdollista, että sovellus ei ole yhteydessä mihinkään palvelimeen vaan toimii täysin paikallisesti mutta siinä tapauksessa sovelluksen mahdollisuudet ovat selkeästi rajoitetummat. Esimerkiksi tietoja ei voida jakaa eri laitteiden välillä.

3 LÄHTÖKOHTA REMIONILLA

3.1 Nykytilanne

Remionilla web-sovelluksia kehitetään AngularJS-sovelluskehyksellä. AngularJS:n on alun perin kehittänyt Googlen työntekijät Miško Hevery ja Adam Abrons sivuprojektina, josta kuitenkin loppujen lopuksi kehkeytyi täysi projekti. Tammikuussa 2010 AngularJS julkaistiin avoimena lähdekoodina. (Miško Hevery and Brad Green… YouTube 2014.) Heinäkuussa 2018 AngularJS siirtyi kolme vuotta kestävään LTS-vaiheeseen (Darwin 2018). LTS eli Long Term Support tarkoittaa sitä, että päivitykset sisältävät vain virheenkorjauksia.

Aikanaan AngularJS oli mullistava tuote ja myös hyvin suosittu. Viimein kehittäjien oli mahdollista rakentaa varsinaisia web-sovelluksia selaimiin ajettavaksi. AngularJS on myös yhteensopiva jQueryn kanssa, joka edelleen on käytössä 73% verkkosivuista, mikä oli hyvä uutinen sen ajan kehittäjille, koska monella oli kokemusta siitä (W3Techs n.d.). Itse asiassa AngularJS hyödyntää jQuerya itsekin (AngularJS n.d. a).

3.2 Ongelmat

AngularJS siis on edistyksellinen sovelluskehys. Kuitenkaan sekään ei ole täydellinen ja tästä AngularJS:n kehitystiimi oli myös tietoinen. Näin ollen he päättivät kirjoittaa koko sovelluskehyksen alusta loppuun saakka kokonaan uudestaan TypeScriptillä. Pääkohdat jotka uudelleenkirjoittamisella haluttiin korjata, olivat muun muassa modulaarisuus, suorituskyky ja komponenttimallin puuttuminen. (Miško Hevery and Brad Green… YouTube 2014).

Modulaarisuus tarkoittaa sitä, että sovelluskehys on jaettu eri moduuleihin, jonka vuoksi kehittäjä voi sisällyttää sovellukseensa vain ne moduulit mitä hän käyttää sovelluksessa, mikä näkyy nopeammissa latausajoissa. Toinen etu moduuleille on niiden korvattavuus. Esimerkiksi jos kehyksen käyttämä reitittäjämoduuli ei sovi kehitettävän sovelluksen käyttötarkoituksiin voi sen halutessaan korvata jollain toisella reitittäjämoduulilla.

AngularJS:n suorituskykyyn vaikuttaa pääosin sen tietojen sidonta. Saavuttaakseen tämän kehys rekisteröi jokaiselle sidotulle attribuutille funktion, joka tarkastelee sen arvoa ja lisää sen tarkkailulistalle. Jokainen kerta, kun käyttäjä tekee jonkun toiminnon tai ohjelmakoodi niin käskee, lista iteroidaan läpi ja nykyistä ja edellistä arvoa verrataan keskenään. Jos arvo ei ole muuttunut iterointi jatkuu seuraavaan funktioon. Jos yksikin arvo on muuttunut, koko lista iteroidaan uudelleen siltä varalta, että jokin muuttunut arvo vaikuttaa johonkin toiseen arvoon. Iterointi päättyy, kunnes muutoksia ei enää löydy. Sen päätyttyä kehys päivittää DOMin uusimmilla arvoilla. Iteraatio voidaan suorittaa korkeimmillaan kymmenen kertaa tai kehys heittää poikkeuksen. (The Digest Loop… n.d.) DOM on ohjelmoitava kuvaus HTML tai XML dokumentista (MDN web docs n.d. b). HTML eli Hypertext Markup Language tarkoittaa kuvauskieltä, jolla luodaan web-dokumentteja. XML on lyhenne sanoista Extensible Markup Language ja on metakieli, jolla määritellään rakenteellisia kuvauskieliä.

Kritiikin kohteeksi yleensä myös päätyy kehyksen käyttämä $scope-muuttuja. $scope on kontrollereille ominainen suorituskonteksti ja sen tarkoitus on yhdistää mallin (template) kontrolleriin (AngularJS n.d. b). Kontrolleri on objekti, joka määrittää $scope-muuttujan arvon ja metodit (AngularJS n.d. c). $scope aiheuttaa kehittäjille päänvaivaa lähinnä siksi, että oletusarvoisesti se on periytyvä ja joidenkin direktiivien tapauksessa taas ei. Itse kehitetyissä direktiiveissä taas kehittäjä itse pystyy määrittämään, periytyykö $scope tai onko sillä kokonaan omansa. (Eidnes 2014) Direktiivi on merkki, joka käskee AngularJS:n kääntäjää lisäämään erikoistoimintoja elementtiin, kuten esimerkiksi tapahtumankuuntelijan (AngularJS n.d. d). Muita kehityskohteita kehyksessä oli mobiiliystävällisyys, monimutkainen API, vianetsintä ja tuki tulevaisuuden standardeille, kuten ECMAScript 6 ja web-komponentit (Miško Hevery and Brad Green… YouTube 2014).

3.3 Motivaatio

AngularJS:ssä parannettavaa riitti ja epäilemättä tiimi teki parhaansa uudelleen kirjoituksen aikana. Kuitenkin kriittinen tekijä oli se, että suoraa migraatiopolkua ei ollut versioon 2.0, mikä tänä päivänä tunnetaan nimellä Angular. AngularJS-sovellusta ei siis voi päivittää vaiheittain Angular-sovellukseksi ilman ulkoisia kirjastoja tai työkaluja. Niinpä kynnys päivittää uudempaan versioon on osoittautunut monelle liian suureksi. Lisäksi,

kun täysin eri kehykseen vaihtaminen ei juurikaan maksa yhtään enempää kuin Angulariin vaihtaminen, miksi ei sitä tutkisi mitä kilpailijoilla on tarjottavana.

Mitä toimeksiantajan web-sovelluksilta nykyään ja tulevaisuudessa toivotaan, on tehokkaampi suorituskyky, nopeammat latausajat, modulaarisuus ja kattavampi kustomoitavuus asiakkaiden toiveiden mukaan.

Totta kai toimeksiantajan kehittäjillä on myös omat toiveensa tuotteen ja käytettävän kehyksen kannalta, joita ovat esimerkiksi aktiivinen kehittäjäyhteisö, komponenttimalliin perustuva kehitys, ohjelman tilanhallinnan vaihtoehdot, kolmannen osapuolen kirjastojen yhteensopivuus ja helpompi testaaminen.

Opinnäytetyön toimeksiantaja on kasvuvaiheessa oleva yritys, vaikka se on toiminut jo melkein kahdenkymmenen vuoden ajan. Sen vuoksi uudistusprojekti on myös aiheellinen, koska uudemmat teknologiat ja kehykset houkuttelevat alan osaajia enemmän sekä vaikuttavat positiivisesti kunkin kehittäjän urakehitykseen.

4 VAIHTOEHTOJEN VERTAILU

4.1 React

Toukokuussa 2013 Tom Occhino ja Jordan Walke pitivät esityksen JSConf US -tapahtumassa. Esityksen otsikkona oli JS Apps at Facebook. Saman päivän aikana, vain noin puoli tuntia ennen puhetta, Facebook julkaisi avoimena lähdekoodina JavaScript-kirjaston nimeltä React. (JSConfUS… YouTube 2013.) Tänä päivänä React on front-end-kehittäjien keskuudessa kestosuosikki. (The State of JavaScript 2018a)

4.1.1 Perustoiminnallisuus

React on JavaScript-kirjasto, jota käytetään käyttöliittymien rakentamiseen. Se tunnetaan JSX:n, virtuaalisen DOMin ja kuvaannollisen ohjelmointitavan hyödyntämisestä. React-sovellukset koostuvat komponenteista, jotka sisältävät oman sisäisen tilan tai ne ovat tilattomia. (React n.d. a)

Kuvaannollinen ohjelmointi Reactissa tarkoittaa komponenttien kuvaamista miltä niiden pitäisi näyttää tilanteessa kuin tilanteessa. Esimerkiksi mitä piirretään näytölle, kun käyttäjä on kirjautuneena palveluun tai päinvastoin.

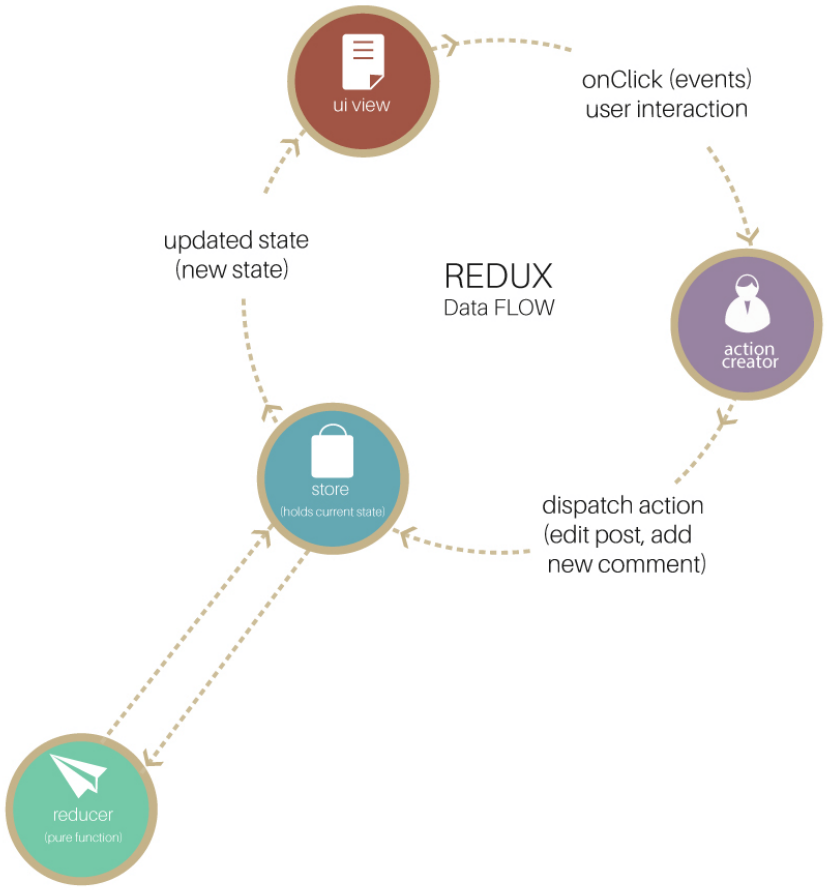
JSX eli JavaScript XML on syntaksi JavaScriptin kirjoittamiseksi, joka on suunniteltu Facebookin toimesta helpottamaan kuvaannollisen JavaScript-koodin luettavuutta ja kirjoittamista. Vaikka JSX muistuttaa XML/HTML-syntaksia on se puhdasta JavaScript-koodia kääntämisen jälkeen. JSX on myös riippumaton Reactista, mikä käytännössä tarkoittaa sitä, että mikä tahansa JavaScript-kirjasto tai -sovelluskehys voi sitä hyödyntää. (JSConfUS… YouTube 2013.)

KUVA 1. JSX ennen ja jälkeen kääntämisen (React n.d. a)

Virtuaalinen DOM eli lyhyesti VDOM on abstraktio varsinaisesta DOMista, joka itsessään on abstraktio strukturoidusta tekstistä, kuten esimerkiksi HTML:stä. Eli yksinkertaistettuna VDOM on kopio DOMista, jota voidaan muokata ilman, että muutokset heijastuvat DOMiin. Vertailemalla VDOMia ennen ja jälkeen päivityksen saadaan tulokseksi täsmällisiä operaatioita, jotka voidaan sitten suorittaa varsinaisessa DOMissa kustannustehokkaasti. (Dace 2017.)

4.1.2 Ohjelman tilanhallinta

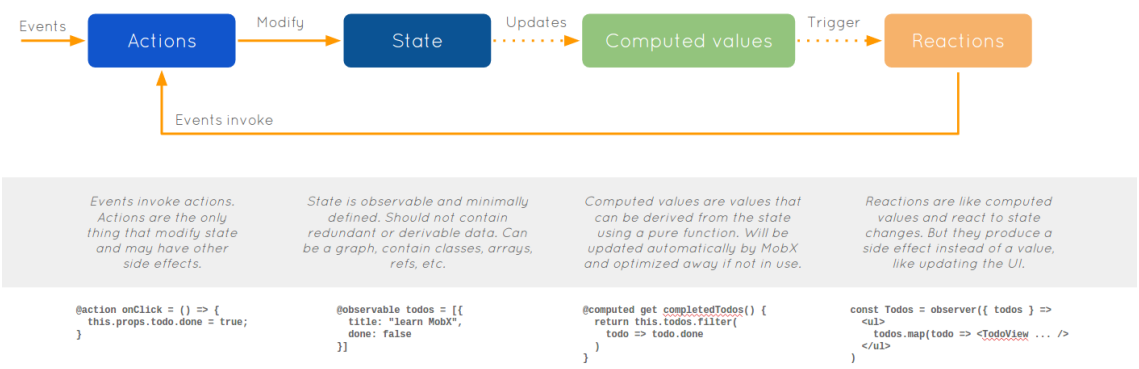
Ehdottomasti suosituin tilanhallintakirjasto React-sovelluksissa on Redux (The State of JavaScript 2018b). Redux on ennustettava tilasäilö JavaScript-sovelluksille (Redux n.d. a). Reduxissa on kolme periaatetta: tilalle on vain yksi lähde, tilaa ei saa suoraan muokata ja tilaa muokataan vain puhtailla funktioilla (Redux n.d. b).



KUVA 2. Tilan kulku Reduxissa (Bachuk 2016)

Reduxin vahvuuksia ovat sen vianetsinnän helppous, liikkuminen tilan eri vaiheiden välillä, testattavuus, synergia yksisuuntaisen tiedonkulun kanssa sekä yhteensopivuus palvelinpään piirtämisen kanssa. (Abramov 2016; Bachuk 2016). Huonoja puolia Reduxissa myös on; alkuun pääsemiseen vaadittava koodin määrä inhottaa monia, oppimiskäyrä on jyrkkä eikä kirjasto itsenään tue sivuvaikutuksellisia toimintoja, kuten tietojen noutamista palvelimelta.

MobX mainostaa olevansa yksinkertainen ja skaalautuva tilanhallintakirjasto. MobX hyödyntää funktionaalisen ja reaktiivisen ohjelmoinnin piirteitä. MobX:n periaatteita ovat yksisuuntainen tiedonkulku, synkroniset derivoitavat arvot ja reaktiot, laskennalliset arvot käsitellään laiskasti eli vain tarvittaessa ja laskennallisien arvojen tulee olla puhtaiden funktioiden tuottamia. (MobX n.d.)



KUVA 3. Tiedonkulku MobX-kirjastossa (MobX 2018)

MobX:n vahvuus on ehdottomasti sen yksinkertaisuus ja vapaamuotoisuus. MobX:ssä sivuvaikutteiset toiminnot toimivat ilman välikappaleita toisin kuin Reduxissa. Vapaamuotoisuus voi myös olla heikkous, koska silloin ei ole vain yhtä tapaa tehdä asiaa, jonka vuoksi kirjaston käyttäjät jakautuvat eri mielipiteisiin ja tekniikkoihin. Muita heikkouksia MobX:ssä on suorituskyky ja tietojen mutatointi, mikä voi Abramovin (2017) mukaan olla ongelmallista Reactin tulevissa versioissa. Mutatointi tarkoittaa lähdearvon muokkaamista. Mutatoinnilla voi olla odottamattomia sivuvaikutuksia ja niiden vianetsintä on yleensä vaikeaa. (Knüssel 2017.) React myös itsessään on liikkumassa suuntaan, jossa luokkapohjaisia komponentteja vältettäisiin, ja kaikki komponentit olisivat funktiopohjaisia (React n.d. b). Sen takia MobX, joka pohjautuu luokkiin, voi olla hämmentävä tulevaisuudessa.

4.1.3 Sovelluksen tyylittely

React-sovelluksissa voidaan käyttää neljää eri tekniikkaa, joilla CSS-tyylittelyitä voidaan määritellä: avoimet tyylit, sisäiset ja ulkoiset tyylit, CSS-moduulit ja CSS-in-JS -ratkaisut. (Krzywda 2017.) Kaksi jälkimmäistä ovat suosituimpia tekniikoita mutta myös ulkoisia tyylejä käytetään React-sovelluksissa.

CSS-moduuli tarkoittaa CSS-tiedostoa, jossa kaikki luokkien ja animaatioiden nimet ovat kapseloituja oletusarvoisesti. CSS-moduulit ei ole virallinen spesifikaatio eikä selaimen toteuttama ominaisuus vaan yksi sovelluspaketoijan toiminto, joka kapseloi luokkien nimet ja CSS-valitsimet. CSS-moduulien tarkoitus on tehdä CSS:stä modulaarisempaa, helpommin ylläpidettävää ja korjata globaalin näkyvyysalueen ongelmat (Rendle 2016.)

CSS-in-JS on abstraktio CSS:stä, joka käyttää JavaScriptiä saavuttaakseen kuvaannollisen ja ylläpidettävän tyylimäärittelyn. Se kääntää JavaScript-koodin CSS-tyylimäärittelyiksi hyvin tehokkaasti niin palvelinpuolella kuin asiakaspuolellakin. Sen ydin on matalatasoinen ja sovelluskehysriippumaton. (JSS n.d.)

CSS-in-JS jakaa mielipiteitä kehittäjien keskuudessa, mutta silti se on hyvin suosittu tekniikka. Tekniikan keskuudessa kuitenkin Styled Components -kirjasto on jo vakiintunut vaihtoehto. Kirjaston myyntivaltti on merkityt merkkijonot, joiden ansiosta JavaScript-koodissa voidaan kirjoittaa puhtaita CSS-tyylimäärittelyjä merkkijonon sisällä. (Styled Components n.d.) Huomioitavaa on, että koska kirjasto käsittelee tyylimäärittelyt merkkijonoina, on sillä joitain rajoitteita. Yksi näistä on yhteensopimattomuus CSS-esikäsittelijöiden kanssa, kuten SCSS (Lorefnon 2017). CSS-moduulit ovat parempi vaihtoehto, jos projektissa halutaan välttämättä käyttää CSS-esikäsittelijää.

Vaihtoehtoja React-komponenttikirjastoille on monia, esimerkiksi: reactstrap, Material-UI, Semantic UI React ja Ant Design of React. Edellä mainittujen välillä on kuitenkin merkittävä ero; Semantic UI React ja reactstrap ovat käyttöliittymäkehyksiä, jotka tarjoavat mahdollisuuden muokata teeman sen näköiseksi kuin haluaa. Material-UI toteuttaa Material Design -muotokielen mukaisesti React-komponentteja ja vastaavasti Ant Design of React toteuttaa Ant Design -muotokielen mukaan.

4.1.4 Soveltuvuusselvitys

Tämän luvun koodiesimerkeissä on käytetty pohjana tehtävälistasovellusta (NG6-starter… n.d.). Esimerkeissä käytettiin react2angular-nimistä kirjastoa, joka mahdollistaa React-komponenttien upottamisen AngularJS-sovellukseen (React2Angular n.d.). Kirjaston käyttäminen on hyvin yksinkertaista. Kuvassa 4 määritellään HelloReact-niminen React-komponentti.

```
1  import React, {Component} from 'react';
2  import PropTypes from 'prop-types';
3
4  export default class HelloReact extends Component {
5    render () {
6      const {name, bar} = this.props;
7      return (
8        <div>
9          <p>Hello {name}</p>
10         <p>Bar: {bar}</p>
11       </div>
12     );
13   }
14 }
15
16 HelloReact.defaultProps = {
17   bar: 42,
18 };
19
20 HelloReact.propTypes = {
21   name: PropTypes.string.isRequired,
22   bar: PropTypes.number,
23 };
24
```

KUVA 4. HelloReact-niminen React-komponentti

Kuvassa 5 kirjasto ja HelloReact-komponentti tuodaan AngularJS-sovelluksen määrittelytiedostoon ja se konfiguroidaan käytettäväksi ohjelmakoodissa. Kuvassa 4 määriteltyjen propTypes- ja defaultProps-ominaisuuksien vuoksi react2angular-funktiolle ei tarvitse antaa argumenteiksi muuta kuin rivillä kahdeksan tuotu luokka. Komponenttia voidaan nyt käyttää AngularJS-komponentin tavoin ohjelmakoodissa.

```
1   1    import angular from 'angular';
    2  + import {react2angular} from 'react2angular';
2   3    import Components from './components/components';
3   4    import services from './services/services';
4   5    import 'normalize.css';
5   6
6   7    import AppComponent from './app.component';
    8  + import HelloReact from './components/HelloReact';
7   9
8   10   angular
9   11     .module('app', [Components.name, services.name])
10      -  .component('app', AppComponent);
    12 +   .component('app', AppComponent)
    13 +   .component('helloReact', react2angular(HelloReact));
11  14
12  15   angular.bootstrap(document.body, ['app']);
13  16
```

KUVA 5. HelloReact-komponentin tuonti sovellukseen

4.2 Angular

Angular 2.0 julkaistiin syyskuussa 2016 Googlen toimesta. Sen on kirjoittanut AngularJS:n alkuperäiskehittäjät viisastuttuaan AngularJS:n kehittämisen jälkeen. Sen voidaan sanoa olevan hengellinen seuraaja AngularJS:lle mutta yhtäläisyydet loppuvat lyhyeen. Versio 7.0 julkaistiin lokakuussa 2018.

4.2.1 Perustoiminnallisuus

Angular-sovellukset koostuvat NgModuuleista. Moduulit koostuvat pääosin komponenteista mutta ne voivat sisältää myös muita moduuleita. Moduuli siis tuo yhteen sovelluksen eri osia ja koostaa niistä yhdessä toimivan pakkauksen. (Angular n.d. a.)

Komponentti on vastuussa siitä mitä näytölle piirretään. Se myös määrittelee, kuinka se käyttäytyy ja reagoi, mikä tapahtuu määrittelemällä sisään- ja ulostulo-ominaisuuksia. (Angular n.d. b.) Komponentilla siis on ikään kuin oma rajapintansa.

Jos komponentti kertoo mitä piirretään niin malli (template) kertoo, kuinka se piirretään. Angular mallit pohjautuvat HTML:ään, joten kaikki HTML-elementit ovat käytettävissä paitsi script, joka on kielletty turvallisuussyistä. HTML ei kuitenkaan usein yksin riitä niinpä mallit voivat myös sisältää JavaScript-koodia. Sitä hyödyntäen voidaan esimerkiksi tulkita muuttujia, joita komponentti sisältää, sitoa HTML-elementtien arvoja komponentin muuttujiin ja päinvastoin tai vaikka ehdollisesti piilottaa osa mallista. (Angular n.d. c.)

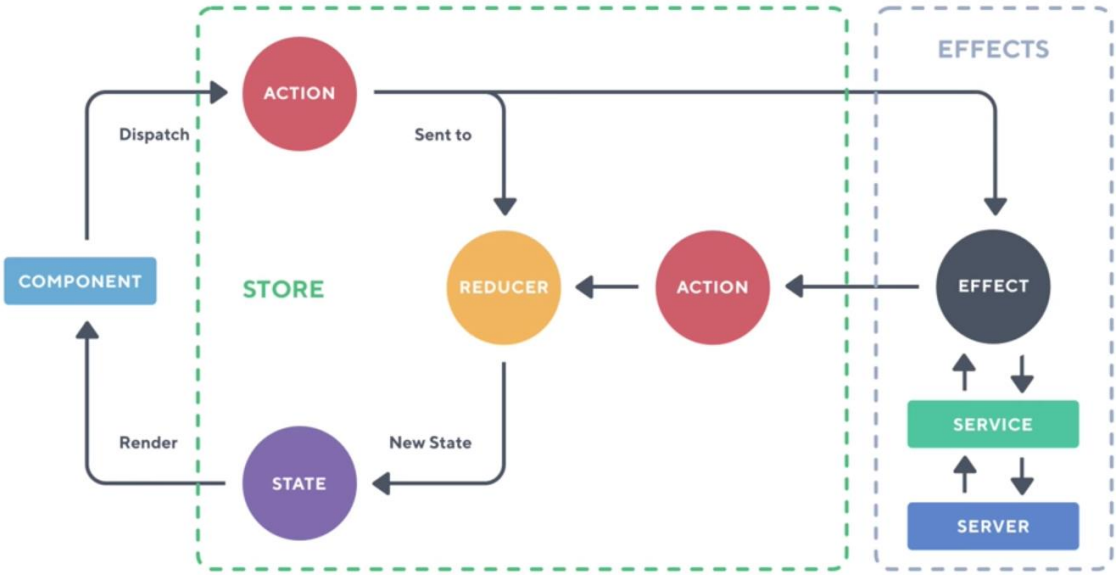
```
<div *ngIf="hero">
  <h2>{{hero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="hero.name" placeholder="name"/>
    </label>
  </div>
  <button (click)="goBack()">go back</button>
  <button (click)="save()">save</button>
</div>

```

KUVA 6. Angular-komponentin malli (Angular n.d. c)

Yksi Angularin tärkeimmistä ominaisuuksista on riippuvuuksien injektointi. Riippuvuusinjektointi on sovelluskehityksessä käytettävä suunnittelumalli, jonka määrä on edistää modulaarisuutta ja testattavuutta. (Angular n.d. d.) Riippuvuus on objekti mikä sallii sitä pyytävän objektin toimivuuden. Injektointi tarkoittaa riippuvuuden toimittamista riippuvaiselle objektille. Riippuvainen objekti ei siis luo riippuvuuksia itse vaan pyytää niitä ja näin ollen riippuu niistä.

4.2.2 Ohjelman tilanhallinta

Angular-sovelluksissa suosituin tilanhallintakirjasto on NgRx, joka on saanut inspiraationsa Reduxista. NgRx käyttää taustalla RxJS-kirjastoa tehdäkseen tilasta reaktiivisen. Se on kehitetty testattavuus ja suorituskyky mielessä. NgRx käyttää Reduxin tavoin puhtaita funktiota tilan muokkaamiseen, ja sen vuoksi Angular-komponenteissa voidaan käyttää onPush-muutoksentunnistusstrategiaa, mikä on hyvin tehokasta. (What is NgRx… YouTube 2018.) Oikeastaan NgRx eroaa Reduxista hyvin vähän. Tapa miten tila välittyy komponenteille, on ainut merkittävä eroavaisuus. NgRx-kirjastoon kuitenkin sisältyy erillinen paketti, toisin kuin Reduxissa, jonka avulla sivuvaikutukselliset toiminnot ovat mahdollisia.



KUVA 7. Kuvakaappaus tiedonkulusta NgRx-kirjastossa (Ultimate NGRX… YouTube 2017)

NgRx:ssä on samanlaisia huonoja puolia kuin Reduxissa; koodia tarvitsee suhteellisen paljon pelkästään alkuunpääsyyn ja oppimiskäyrä on jyrkkä. Ei ole siis ihme, että NgRx:n päälle on kehitetty monenlaisia ratkaisuja, joista tunnetuin on NGXS-kirjasto.

Vaikka Angularia varten on kehitetty erilaisia tilanhallintakirjastoja, on Angularin palvelut (service) tilanhallinnan ratkaisuna usein myös kelvollinen vastaus ongelmaan. Moni web-sovellus ei koskaan tarvitse erillistä tilanhallintaa ja koska Angular on niin täysimittainen sovelluskehys se ei ole ongelma Angular-sovelluksissa. Jokaisen sovelluksen kohdalla se on arvioitava erikseen, onko erillinen tilanhallinnan ratkaisu tarpeellinen.

Muita maininnan arvoisia tilanhallintakirjastoja, joita Angular-sovelluksissa esiintyy: @angular-redux/store, Akita, MobX.

4.2.3 Sovelluksen tyylittely

Angularissa CSS-tyylimäärittelyt ovat oletusarvoisesti kapseloituja, koska Angular käyttää emuloitua varjo-DOMia (shadow DOM) eli Angular jäljittelee aitoa varjo-DOMia itse kehittämällä toteutuksella. Aitoa varjo-DOMia voidaan myös käyttää, mutta emuloitu on käytössä oletusarvoisesti, koska selaintuki on vielä kehno varjo-DOMin kannalta. Varjo-DOM voidaan myös kytkeä kokonaan pois käytöstä, jolloin kapselointia ei CSS-tyylimäärittelyissä ole, vaan kaikki CSS-tyylimäärittelyt ovat näkyvyysalueeltansa globaaleja. (Angular n.d. e.) Angularissa CSS-esikäsittelijöiden käyttäminen on hyvin yksinkertaista; komponentin määrittelytiedostossa tuodaan ulkoinen esikäsittelijän mukainen tyylimäärittelytiedosto komponenttiin tavallisen CSS-tiedoston sijaan.

Suosittuja Angular-komponenttikirjastoja ovat muun muassa NG Bootstrap, joka toteuttaa Bootstrap 4 -komponentteja Angular-komponenttien muodossa. Angular Material toteuttaa Material Design -muotokielen mukaisesti Angular-komponentteja. PrimeNG on käyttöliittymäkehys, joka sisältää yli 80 Angular-komponenttia. Muita vähemmän suosittuja ovat esimerkiksi NG-ZORRO ja Clarity Angular.

4.2.4 Soveltuvuusselvitys

Tämän luvun koodiesimerkeissä on käytetty pohjana tehtävälistasovellusta (NG6-starter… n.d.). Esitellyistä vaihtoehdoista Angularin sommittelu AngularJS-sovellukseen on kaikkein monimutkaisin. Ensimmäiseksi projektissa täytyy ottaa käyttöön TypeScript. TypeScript on ohjelmointikieli, joka on rakennettu JavaScriptin päälle. TypeScript sisältää hyödyllisiä ominaisuuksia, joita JavaScriptissä ei ole, kuten muuttujien tyypitykset. TypeScript on käännettävä JavaScript-koodiksi, jotta se voidaan suorittaa. (TypeScript n.d.) Esimerkkiprojektissa TypeScriptin käyttöönotto onnistui vaivattomasti. Kuvassa 8 on projektin TypeScript-kokoonpanoasetukset.

```
1  {
2    "compilerOptions": {
3      "target": "es5",
4      "module": "commonjs",
5      "moduleResolution": "node",
6      "sourceMap": true,
7      "emitDecoratorMetadata": true,
8      "experimentalDecorators": true,
9      "lib": [ "es2015", "dom" ],
10     "noImplicitAny": false,
11     "suppressImplicitAnyIndexErrors": true
12   }
13 }
14
```

KUVA 8. Esimerkkiprojektin TypeScript-kokoonpanoasetukset

Seuraava askel on muuttaa sovelluksen alustuskoodi yhteensopivaksi TypeScriptin ja Angular-AngularJS-yhdistelmäsovelluksen kanssa (Julien 2018). Kuvassa 9 tiedoston alussa olevat tuontilauseet on vaihdettu TypeScript-yhteensopiviksi. Tuontilauseiden korvaamisen vuoksi myös rivi yhdeksän on muutettu. Rivillä kahdeksan esitellään uusi muuttuja nimeltä MODULE_NAME, joka määritellään oletusarvoiseksi ulostuloksi rivillä 13. Tiedoston nimi on myös muutettu app.js:stä app.module.ajs.ts:ksi.

```
1     - import angular from 'angular';
2     - import Components from './components/components';
3     - import services from './services/services';
      1 + import * as angular from 'angular';
      2 + import './components/components';
      3 + import './services/services';
4     4   import 'normalize.css';
5     5
6     6   import AppComponent from './app.component';
7     7
      8 + const MODULE_NAME = 'app';
8     9   angular
9       -   .module('app', [Components.name, services.name])
      10 +   .module('app', ['app.components', 'app.services'])
10    11    .component('app', AppComponent);
11      -
12      - angular.bootstrap(document.body, ['app']);
13    12
      13 + export default MODULE_NAME;
```

KUVA 9. Eroavaisuudet AngularJS-sovelluksen alustuksessa

Kun AngularJS-sovelluksen alustuskoodi on yhteensopiva, on luotava Angular-sovelluksen alustuskoodi. Alustuskoodi tehdään tiedostoon app.module.ts. (Julien 2018.) Tiedoston sisältö on kuvassa 10. Rivillä kuusi tuodaan Angular-komponentti, joka on kuvattu kuvassa 13. Rivillä 17 käytetään erityistä upgrade-metodia. Upgrade-metodin käyttö sallii yhdistelmäsovelluksen ajon (Angular n.d. f).

```
1  import {NgModule} from '@angular/core';
2  import {BrowserModule} from '@angular/platform-browser';
3  import {UpgradeModule} from '@angular/upgrade/static';
4  import moduleName from './app.module.ajs';
5
6  import {HelloAngular} from './components/HelloAngular.component';
7
8  @NgModule({
9    imports: [BrowserModule, UpgradeModule],
10   declarations: [HelloAngular],
11   entryComponents: [HelloAngular],
12 })
13 export class AppModule {
14   constructor (private upgrade: UpgradeModule) {}
15
16   ngDoBootstrap () {
17     this.upgrade.bootstrap(document.documentElement, [moduleName], {
18       strictDi: true,
19     });
20   }
21 }
22
```

KUVA 10. Angular-yhdistelmäsovelluksen alustus (Julien 2018, muokattu)

Alustamisen viimeinen osa on main.ts-tiedosto (kuva 11). Tiedostossa riveillä yksi ja kaksi tuodaan kaksi polyfill-skriptiä. (Julien 2018.) Polyfill-skripti tarkoittaa JavaScript-koodia, joka tuo puuttuvat ominaisuudet selaimiin, jotka eivät tue ominaisuutta. AngularJS-sovelluksen tarvitsemat riippuvuudet tuodaan riveillä 4–6 (Julien 2018). Rivillä kahdeksan kutsutaan setAngularLib-funktiota mikä kertoo ohjelmalle, että AngularJS-kehystä käytetään myöhemmin sovelluksessa. Rivillä yhdeksän viimein käynnistetään yhdistelmäsovellus (Julien 2018).

```
1  import 'zone.js';
2  import 'reflect-metadata';
3  import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
4  import {setAngularLib} from '@angular/upgrade/static';
5  import * as angular from 'angular';
6  import {AppModule} from './app/app.module';
7
8  setAngularLib(angular);
9  platformBrowserDynamic().bootstrapModule(AppModule);
10
```

KUVA 11. Sovelluksen käynnistystiedosto main.ts (Julien 2018, muokattu)

Jotta sovellus saadaan taas kääntymään pitää sovelluksen sisääntulo- ja index.html-tiedosto päivittää vastaamaan tehtyjä muutoksia (kuva 12).

```
1  1   // gulpfile.babel.js
2      - entry: path.join(__dirname, root, 'app/app.js'),
       2 + entry: path.join(__dirname, root, 'main.js'),
3  3
4  4   // index.html
5      - <body ng-strict-di ng-cloak>
       5 + <body>
6  6       <app>
7  7           Loading...
8  8       </app>
9  9   </body>
10 10
```

KUVA 12. Sisääntulotiedoston ja index.html:n muutokset

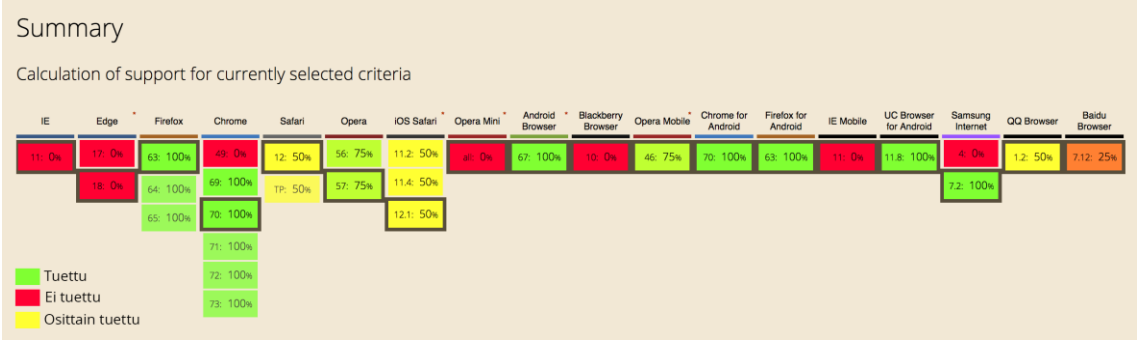
Kuvassa 13 on HelloAngular-komponentti. Komponentti muutetaan rivillä 21 AngularJS-yhteensopivaksi käyttäen downgradeComponent-funktiota. Komponentti on nyt valmis käytettäväksi yhdistelmäsovelluksessa AngularJS-komponenttien tavoin.

```
import {Component, Input} from '@angular/core';
import { downgradeComponent } from '@angular/upgrade/static';
declare var angular: angular.IAngularStatic;
import MODULE_NAME from '../app.module.ajs';

@Component({
  selector: 'hello-angular',
  template: `
    <div>
      <p>Hello {{name}}</p>
      <p>Bar: {{bar}}</p>
    </div>
  `,
})
export class HelloAngular {
  @Input() name: string;
  @Input() bar: number = 42;
}

angular.module(MODULE_NAME)
  .directive('helloAngular', downgradeComponent({component: HelloAngular}));

```

KUVA 13. AngularJS-yhteensopiva Angular-komponentti

4.3 Web-komponentit

Web-komponentit ovat tämän opinnäytetyön kirjoittamishetkellä vielä monelle web-kehittäjälle tuntematon termi, sillä se ei vain ole laaja käsite, joka sisältää useampia alikäsitteitä vaan myös uusi lisäys koko web-alustaan. Nykyisessä muodossa tunnettuja web-komponentteja on kehitetty jo vuodesta 2011 alkaen (Web Components and Model… Fronteers 2011). Aikaisempiakin samaan suuntaan viittaavia yrityksiä on nähty jo vuonna 1998 Microsoftin toimesta (Using HTML Components… 2013). Mozilla julkisti vuonna 2001 XBL-spesifikaatioehdotuksen (W3C 2001). Molemmat ovat nykyään poistettu käytöstä eikä yksikään moderni selain tue niitä.



KUVA 14. Web-komponenttien selaintuki (Can I Use 2018, muokattu)

Web-komponentti-käsitteeseen kuuluu neljä spesifikaatiota: Custom Elements, Shadow DOM, HTML Templates ja ES Modules. Nämä yhdessä muodostavat standardin, jota kutsutaan web-komponenteiksi (Web Components n.d.).

4.3.1 Perustoiminnallisuus

Kustomoitu elementti tarkoittaa juuri sitä mitä nimi ilmentää eli kustomoitua elementtiä. Kustomoituja elementtejä on olemassa kahdentyyppisiä: autonomisia ja kustomoituja luontaiselementtejä. Ne eroavat toisistaan syntaksiltaan ja käyttötarkoitukseltaan. (Milner 2018.)

Kustomoituja luontaiselementtejä käytetään jo olemassa olevien HTML-elementtien kustomointiin ja jatkokehittämiseen, ja ne periytyvät HTML-elementistä. Periyttämisen vuoksi alkuperäisen HTML-elementin ominaisuudet, kuten esimerkiksi DOM-ominaisuudet ja saavutettavuus, säilyvät ennallaan. Samasta syystä ne luodaan koodissa hieman eri tavalla kuin autonomiset. (Bidelman 2018a.)

```
1 class FancyButton extends HTMLButtonElement {...}
2
3 customElements.define('fancy-button', FancyButton, {extends: 'button'});
4
5 // Usage in HTML
6 <button is="fancy-button" disabled>Fancy button!</button>
7
```

KUVA 15. Kustomoidun luontaiselementin määrittely ja luonti (Bidelman 2018a, muokattu)

Autonomiset elementit periytyvät HTMLElement-rajapinnasta tai muista autonomisista elementeistä (Bidelman 2018a). Autonomisia elementtejä käytetään, kun halutaan tehdä jotain kokonaan uutta, kuten vaikka karusellikomponentti.

```
class FancyButton extends HTMLElement {...}

customElements.define('fancy-button', FancyButton);

// Usage in HTML
<fancy-button>Fancy button!</fancy-button>

```

KUVA 16. Autonomisen kustomoidun elementin määrittely ja luonti (Bidelman 2018a, muokattu)

Kustomoituja elementtejä käytetään tavallisten HTML-elementtien tavoin HTML-koodissa. Kehittäjien on mahdollista määrittää kuinka selain konstruoi elementin ja kuinka se reagoi muutoksiin. Kehittäjien valta ja vastuu erottaa kustomoidut elementit HTML-elementeistä.

Varjo-DOM piilottaa toteutuksen yksityiskohdat ja kapseloi CSS-tyylimäärittelyt itsensä sisälle. Kapselointi tarkoittaa käytännössä sitä, että kustomoidun elementin tyylit eivät vuoda ulkopuolelle eivätkä sen ulkopuolella määritellyt tyylit vaikuta siihen, ellei kehittäjä ole erikseen sallinut sitä. (Bidelman 2018b.) Tämä on erittäin tärkeä ominaisuus komponenttipohjaisessa kehityksessä.

```
customElements.define('fancy-tabs', class extends HTMLElement {
  constructor() {
    super();

    // Attach a shadow root to <fancy-tabs>.
    const shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.innerHTML = `
      <style>#tabs { ... }</style> <!-- styles are scoped to fancy-tabs! -->
      <div id="tabs">...</div>
      <div id="panels">...</div>
    `;
  }
  ...
});

```

KUVA 17. Varjo-DOMin käyttö kustomoiduissa elementissä (Bidelman 2018b, muokattu)

Komponenttien toinen tärkeä ominaisuus on niiden muodostaminen toisista komponenteista, jossa slot-elementti tulee apuun. Slot-elementin avulla komponentin varjo-DOMiin

voidaan tuoda HTML-koodia sen ulkopuolelta. Tämän ansioista web-komponentin rajapinta säilyy siistinä ja kuvaannollisena.

```
customElements.define('fancy-tabs', class extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.innerHTML = `
      <style>#tabs { ... }</style>
      <div id="tabs">
        <slot id="tabsSlot" name="title"></slot> <!-- named slot -->
      </div>
      <div id="panels">
        <slot id="panelsSlot"></slot>
      </div>
    `;
  }
  ...
});

// Usage in HTML
<fancy-tabs background>
  <button slot="title">Tab 1</button>
  <button slot="title" selected>Tab 2</button>
  <button slot="title">Tab 3</button>
  <section>content panel 1</section>
  <section>content panel 2</section>
  <section>content panel 3</section>
</fancy-tabs>

```

KUVA 18. Slot-elementtien käyttö (Bidelman 2018b, muokattu)

ES moduuli -spesifikaatio määrittelee, kuinka web-komponentteja ja JavaScript-koodia tuodaan käytettäväksi HTML- ja JavaScript-koodissa. Se korvaa aiemmin suunnitellun HTML Imports -spesifikaation, mikä tuntui monille web-kehittäjille oudolta ja epäluonnolliselta (Milner 2018).

HTML-malleista puhuttaessa viitataan template-elementtiin. Template-elementti on erikoinen siinä mielessä, että selain ei koskaan piirrä sitä itsestään vaan se täytyy luoda JavaScript-koodissa. (Bidelman 2018a.) Template-elementillä voidaan siis luoda uudelleen käytettäviä HTML-palasia.

```
/* It is important to note that you need to defer execution of this script until
 * the document has been parsed
 */
const tmpl = document.querySelector('#x-foo-from-template');

customElements.define('x-foo-from-template', class extends HTMLElement {
        constructor () {
            super();
            let shadowRoot = this.attachShadow({mode: 'open'});
            shadowRoot.appendChild(tmpl.content.cloneNode(true));
        }
    }
);

// Usage in HTML
<template id="x-foo-from-template">
  <style>
    p { color: green; }
  </style>
  <p>I'm in Shadow DOM. My markup was stamped from a &lt;template&gt;.</p>
</template>

<x-foo-from-template></x-foo-from-template>

```

KUVA 19. Template-elementin käyttö (Bidelman 2018a, muokattu)

4.3.2 Ohjelman tilanhallinta

Koska web-komponentit ovat enimmäkseen puhdasta HTML ja JavaScript-koodia, ei niiden ympärille ole vielä tähän päivään mennessä kehitetty omaa tilanhallintakirjastoa. Loppujen lopuksi sille ei edes ole tarvetta, koska mikä tahansa JavaScript-tilanhallintakirjasto on yhteen sopiva web-komponenttien kanssa. Kaikesta huolimatta eri web-komponenttikehyksien ja -kirjastojen keskuudessa Redux on yleensä otettu huomioon. Näitä ovat muun muassa Polymer, Stencil ja LitElement. Vaikka web-komponentit eivät ole rajoitettu mihinkään tilanhallintakirjastoon on Redux-trendi selkeä.

4.3.3 Sovelluksen tyylittely

Web-komponentteja voidaan tyylitellä useammalla tapaa: sen ulkopuolelta, sisäpuolelta ja ylikirjoittamalla oletusarvoja CSS-muuttujilla (Bidelman 2018b). Tärkeintä on muistaa, että oletusarvoisesti ulkopuoliset CSS-valitsimet eivät vaikuta varjo-DOMin sisältämiin elementteihin, eikä sen sisällä määritellyt CSS-tyylimäärittelyt vaikuta sen ulkopuolelle.

Web-komponentteja tyylitellään sisäpuolelta käyttäen style-elementtiä tai tuomalla ulkoinen CSS-tiedosto link-elementtiä käyttäen. CSS-tyylimäärittelyjä voi normaaliin tapaan sijoittaa style-elementin sisälle. CSS-tyylimäärittelyissä on kuitenkin käytettävä host-näennäisvalitsinta, jos tyyleihin halutaan vaikuttaa komponentin ulkopuolelta. Host-näennäisvalitsin viittaa varjoisäntään, joka pitää sisällään kaikki varjo-DOMin lapsielementit (MDN Web Docs n.d. c). Host-näennäisvalitsimen tyylit voidaan ylikirjoittaa komponentin ulkopuolelta elementtivalitsimella.

```
customElements.define(
    'x-component',
    class extends HTMLElement {
        constructor () {
            super();
            const shadowRoot = this.attachShadow({mode: 'open'});
            shadowRoot.innerHTML = `
                <style>:host { color: red; }</style>
                <p>Lorem ipsum dolor sit amet</p>
            `;
        }
    }
);

// In HTML
<style>
    x-component { color: orange; }
</style>
<x-component></x-component> <!-- Result is orange text -->

```

KUVA 20. Yksinkertaisen komponentin tyylien ylikirjoittaminen

Jos web-komponentti on monimutkaisempi ja sen CSS-tyylimäärittelyissä on käytetty muita CSS-valitsimia host-näennäisvalitsin ei yksin riitä. CSS-muuttujilla voidaan antaa komponentin käyttäjälle enemmän kustomointi mahdollisuuksia mutta niistä on vastuussa web-komponentin kehittäjä.

```
customElements.define(
    'x-component',
    class extends HTMLElement {
        constructor () {
            super();
            const shadowRoot = this.attachShadow({mode: 'open'});
            shadowRoot.innerHTML = `
                <style>
                    .footer {
                        color: var(--footer-color, red);
                    }
                </style>
                <p>Lorem ipsum dolor sit amet</p>
                <span class="footer">Consectetur adipiscing elit consequat mus</span>
            `;
        }
    }
);

// In HTML
<style>
    x-component {
        color: orange;
        --footer-color: blue;
    }
    x-component .footer { /* Does not work */
        color: violet;
    }
</style>
<x-component></x-component> <!-- Result is orange and blue text -->
```

KUVA 21. CSS-muuttujien käyttö web-komponenteissa

Web-komponenttien ekosysteemi on vielä tämän opinnäytetyön kirjoittamishetkellä heikko. Web-komponenttipohjaisia käyttöliittymäkirjastoja on olemassa mutta niiden kattavuusalue on pieni. Esimerkiksi Vaadin, Brightspace UI ja Material Web Components tarjoavat valmiita komponentteja parista muutamaan kymmeneen. Yksittäisiä web-komponentteja voi etsiä webcomponents.org -sivustolla. On syytä kuitenkin tarkistaa mitä kirjastoa web-komponentti käyttää ennen kuin komponentin tuo omaan sovellukseen. Jos web-komponentti käyttää eri kirjastoa kuin muu sovellus voi se aiheuttaa pidentyneitä latausaikoja. Tavallisesti web-komponenttikirjastot ovat kuitenkin hyvin minimaalisia eli vaikutus on pieni.

4.3.4 Soveltuvuusselvitys

Tämän luvun koodiesimerkeissä on käytetty pohjana tehtävälistasovellusta (NG6-starter… n.d.). Web-komponenttien käyttö AngularJS-sovelluksessa on helppoa. Kuvassa 22 määritellään yksinkertainen web-komponentti, jolla on name- ja bar-määritteet.

```
class HelloWeb extends HTMLElement {
  constructor () {
    super();
    const shadowRoot = this.attachShadow({mode: 'open'});
    const name = this.hasAttribute('name') ? this.getAttribute('name') : '';
    const bar = this.hasAttribute('bar') ? this.getAttribute('bar') : 42;
    shadowRoot.innerHTML = `
      <div>
        <p>Hello ${name}</p>
        <p>Bar: ${bar}</p>
        <slot></slot>
      </div>
    `;
  }
}

customElements.define('hello-web', HelloWeb);

```

KUVA 22. HelloWeb-nimisen web-komponentin määrittely

Kuvassa 23 riveillä 12 ja 13 tuodaan sovellukseen mukaan kaksi polyfill-skriptiä. Rivillä 14 tuodaan kuvassa 22 määritelty HelloWeb-komponentti. AngularJS-sovelluksessa on nyt mahdollista käyttää HelloWeb-komponenttia.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>TodoMVC based on NG6-starter by @AngularClass</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black">
    <meta name="description" content="TodoMVC by @Fesor">
    <link rel="icon" href="data:;base64,iVBORw0KGgo=">
+   <script src="https://unpkg.com/@webcomponents/webcomponentsjs@2.2.1/custom-elements-es5-adapter.js"></script>
+   <script src="https://unpkg.com/@webcomponents/webcomponentsjs@2.2.1/webcomponents-loader.js"></script>
+   <script type="module" src="./app/components/HelloWeb.js"></script>
</head>
<body ng-strict-di ng-cloak>
    <app>
        Loading...
    </app>
</body>
</html>
```

KUVA 23. HelloWeb-komponentin tuonti sovellukseen

5 TULOKSET

5.1 React

Jo kolmena vuotena peräkkäin React on ollut suosituin sovelluskehys JavaScript-kehittäjien mielestä (The State of JavaScript 2016, 2017, 2018a). Myös Stack Overflow:n (2016, 2017, 2018) teettämissä kyselyissä React on ollut erittäin pidetty sovelluskehys. Kysymys kuuluu, miksi React on niin suosittu.

Ensimmäinen asia mihin henkilöllä huomio kiinnittyy React-koodissa, on varmasti JSX. Facebook kehitti JSX:n parantamaan kuvaannollisen JavaScript-koodin luettavuutta. Syntaksiltaan se muistuttaa HTML ja XML syntaksia, mikä voi helposti johtaa sekaannuksiin, sillä todellisuudessa JSX on JavaScript-koodia. JSX siis jakaa mielipiteitä mutta myös muutamat muutkin sovelluskehykset käyttävät samaa syntaksia, kuten Hyperapp ja Vue, joka tukee syntaksia mutta suosittelee mallien käyttöä (Vue n.d.). React-koodissa on erittäin harvinaista olla käyttämättä JSX-syntaksia syystä, että se täydentää Reactin kuvaannollista ohjelmointitapaa.

Reactiin tutustuessa nopeasti huomaa, että React ei yksin välttämättä riitä suuremman sovelluksen kehittämiseen ja se on tarkoituksellista. React (n.d. a) markkinoi itseään kirjastona, joka ei tee olettamuksia. Sen vuoksi React-sovelluksia voidaan piirtää palvelinpäässä ja myös mobiilisovelluksina (React n.d. a). Reactin kehittäjäyhteisö on hyvin aktiivinen ja on tuottanut paljon ratkaisuja puuttuvien osien tilalle. Reactin ekosysteemiin React-kehittäjät ovatkin hyvin tyytyväisiä kuten myös kehittäjien työkaluihin, joita he ovat itse kehittäneet ja julkaisseet. (The State of JavaScript 2018b.)

Reactin ensimmäinen versio julkaistiin toukokuussa 2013, joten sen voi ajatella olevan jo vanha ratkaisu. Kehitys on kuitenkin ollut tiivistä Facebookin React-tiimin puolesta ja paljon uusia ominaisuuksia on vielä luvassa (Abramov 2018). Reactin-kehittäjätiimi onkin hyvin aktiivinen yhteisössä ja kommunikoi ajatuksiaan avoimesti niin blogeissa, konferensseissa kuin myös keskustelupalstoilla. Facebookin portfolio ei silti lopu Reactiin. Muita maailmalla tunnettuja Facebookin luomia kehittäjätyökaluja on muun muassa GraphQL, Flux-arkkitehtuuri, Flow, Reason, Jest ja Immutable.js.

Kuluneiden vuosien aikana React on levinnyt myös muiden suurten yrityksen käyttöön, kuten Airbnb, Paypal, Walmart, Dropbox ja Netflix. Facebook ei pelkästään itse riipu Reactista, vaan edellä mainitut yritykset ovat panostaneet vahvasti React-ekosysteemiin, jopa julkaisemalla omia ratkaisuja avoimen lähdekoodin alla.

Suorituskyky Reactissa on samaa luokkaa kuin kilpailevissa sovelluskehyksissä, kuten Angular ja Vue.js. Vue on edellä mainituista nopein mutta ero on hyvin pieni. JavaScript ja web-komponentit ovat luonnollisesti suorituskykylistan kärkijoukossa. (JS Web Frameworks Benchmark n.d.)

5.2 Angular

Angular 2.0 julkistettiin virallisesti tammikuussa 2014 ng-conf-tapahtumassa. Lokakuussa Angularin kehittäjät esittelivät muutoksia kehyksessä ng-europe-tapahtumassa ja siitä lähtien Angularin maineessa on ollut tahra, joka vaikuttaa vielä nykyään. Kun versio 2.0 viimein julkaistiin syyskuussa 2016, React oli ehtinyt jo saavuttaa ykkössijan kehittäjien suosiossa (The State of JavaScript 2016).

Angular on hyvä kokonaisuus vastaamaan web-sovelluskehityksen haasteisiin ja samaan aikaan se voi olla liian suuri kokonaisuus. Kehyksen tarjoamien ominaisuuksien määrän kasvaessa myös oppimiskäyrä kasvaa. Oppimiskäyrään vaikuttaa myös merkittävästi TypeScript ja reaktiivisen ohjelmoinnin ajattelutapa. Molemmat ovat välttämättömiä tekniikoita, jotka täytyy hallita ollakseen sujuva kehyksen käytössä. On selvää, että Angular on kehitetty TypeScript mielessä, koska ilman sitä kehystä on kankea käyttää.

Angularin mallisyntaksi on hyvin kiistanalainen aihe. On ymmärrettävää, että jotkut haluavat pitää HTML:n puhtaana tai vähintään mahdollisimman helppolukuisena mutta Angularin käyttämällä syntaksilla se ei yksinkertaisesti ole mahdollista. Sen lisäksi mallisyntaksi myös eroaa muista mallisyntakseista, mikä voi aiheuttaa lisää hämmennystä (Jiang 2016).

Huolestuttavaa Angularin kohdalla on, että Googlen lisäksi on vaikea löytää yrityksiä tai tuotteita, jotka käyttävät kehystä. Wappalyzer-palvelun mukaan Angular on noin 4 600

web-sivulla käytössä (Wappalyzer n.d.). Se ei tietenkään tarkoita, etteikö Angularia olisi tuotantoympäristössä käytössä, vaan että siitä ei ainakaan avoimesti kerrota.

5.3 Web-komponentit

Web-komponentti-standardi on askel oikeaan suuntaan ja se sisältää hyviä ideoita. Standardia on kehitetty jo ainakin seitsemän vuotta ja vasta nyt sen vaatimat ominaisuudet alkavat ilmestyä moderneihin selaimiin. Vielä kuluu vuosia ennen kuin kaikki selaimet tukevat web-komponentteja ja on myös täysin mahdollista, että kaikki selaimet eivät tule koskaan tukemaan web-komponentteja. Ongelmaan on silti hätäkeino eli polyfill-skriptit mutta sekään ei ole takuuvarma ratkaisu ongelmaan. Aikaa kuluu myös yhteisön ja ekosysteemin muodostumiseen ja parhaiden toimintatapojen löytämiseen. Web-komponentit ovat lupaavia web-alustalle mutta ne eivät ole vielä valmiita tuotantoympäristöön.

DOMin tiedetään olevan vaikea käyttää suoraan ohjelmakoodista. Siksi sen päälle on kehitetty monia abstraktioita, jotka tekevät DOMin käytöstä yksinkertaisempaa. Web-komponentit vaativat DOMin käyttöä, joten niiden käytöstä tulee nopeasti liian monimutkaista. Näin ollen monia web-komponenttikirjastoja on kehitetty helpottamaan web-komponenttien kehittämistä, joista tunnetuin on Polymer. Polymer-kirjaston on kehittänyt Google, mutta kirjasto on jo vanhentunut ja Google suosittelee LitElementin käyttöä (Roadmap update… 2018). Jos web-komponentteja siis suunnittelee käyttävänsä, on tarpeen myös selvittää, mikä web-komponenttikirjasto on sopivin sovelluksen toteutukseen.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Kaikki huomioon ottaen on mahdotonta määritellä, mikä on paras web-sovelluskehys. Jokaisessa kehyksessä on hyvät ja huonot puolensa. Toimeksiantajan tuotteiden tulevaisuuden suunnitelmissa ei myöskään ole rajoittavia tekijöitä, joten kehyksen valinta pohjautuu pääasiassa riskien minimoimiseen ja tulevaisuuden varautumiseen. Varmin valinta näiden näkökohtien puolesta on React. Kehittäjät rakastavat sitä, sen ekosysteemi kukoistaa, yhteisö on laaja ja aktiivinen ja se on suurten yritysten tukema.

Seuraava askel toimeksiantajan uudistamisprojektissa on selvittää, kuinka valittujen tuotteiden uudistaminen käytännössä toteutetaan. Tässä opinnäytetyössä uudistamisen ensiaskelta eli web-sovelluskehyksien rinnakkaisajoa kokeiltiin vain kevyesti. Web-sovelluksiin kuuluu paljon muutakin, kuin vain yksi kapseloitu ominaisuus, joka toimii itsenään. Uudistaminen on pitkäaikainen projekti ja siinä on omat haasteensa ja toimintatavat.

Opinnäytetyössä on käytetty paljon internetin kautta haettuja lähteitä, mikä tarkoittaa sitä, että lähteet ovat usein mielipidepohjaisia ja mahdollisesti vanhentuneita ja siten epämääräisiä. Onneksi aiheesta on olemassa määrällisiä tutkimuksia, kuten The State of JavaScript- ja Stack Overflow Developer Survey -kyselyt. Kyseiset aineistot antavat paremman kuvan kehittäjien näkökulmasta yleisesti.

LÄHTEET

Abramov, D. 2018. React 16.x Roadmap. Luettu 27.11.2018. https://reactjs.org/blog/2018/11/27/react-16-roadmap.html

Abramov, D. 2017. React Beginner Question Thread. Luettu 21.11.2018. https://dev.to/royriojas/comment/1n61

Abramov, D. 2016. You Might Not Need Redux. Luettu 20.11.2018. https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

Angular. N.d. a. Introduction to modules. Luettu 31.10.2018. https://angular.io/guide/architecture-modules

Angular. N.d. b. Introduction to components. Luettu 31.10.2018. https://angular.io/guide/architecture-components

Angular. N.d. c. Template Syntax. Luettu 31.10.2018. https://angular.io/guide/template-syntax

Angular. N.d. d. Dependency Injection in Angular. Luettu 31.10.2018. https://angular.io/guide/dependency-injection

Angular. N.d. e. View encapsulation. Luettu 27.11.2018. https://angular.io/guide/component-styles#view-encapsulation

Angular. N.d. f. UpgradeModule. Luettu 4.12.2018. https://angular.io/api/upgrade/static/UpgradeModule

AngularJS. N.d. a. angular.element. Luettu 23.10.2018. https://docs.angularjs.org/api/ng/function/angular.element

AngularJS. N.d. b. What are Scopes? Luettu 23.10.2018. https://docs.angularjs.org/guide/scope

AngularJS. N.d. c. Understanding Controllers. Luettu 23.10.2018. https://docs.angularjs.org/guide/controller

AngularJS. N.d. d. What are Directives? Luettu 23.10.2018. https://docs.angularjs.org/guide/directive

Bachuk. 2016. Redux · An Introduction. Luettu 20.11.2018. https://www.smashingmagazine.com/2016/06/an-introduction-to-redux/

Bidelman, E. 2018a. Custom Elements v1: Reusable Web Components. Luettu 7.11.2018. https://developers.google.com/web/fundamentals/web-components/customelements

Bidelman, E. 2018b. Shadow DOM v1: Self-Contained Web Components. Luettu 7.11.2018. https://developers.google.com/web/fundamentals/web-components/shadowdom

Can I Use. 2018. https://caniuse.com/#search=v1

Dace. 2017. Understanding React's Virtual DOM vs. the real DOM. Luettu 30.10.2018. https://web.archive.org/web/20181001050801/https://medium.com/@hidace/understanding-reacts-virtual-dom-vs-the-real-dom-68ae29039951

Darwin, P. 2018. Stable AngularJS and Long Term Support. Luettu 19.10.2018. https://blog.angular.io/stable-angularjs-and-long-term-support-7e077635ee9c

Eidnes, L. 2014. AngularJS: The Bad Parts. Luettu 23.10.2018. https://larseidnes.com/2014/11/05/angularjs-the-bad-parts/

Garret, J. 2005. Ajax: A New Approach to Web Applications. Luettu 17.10.2018. http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

GraphQL. N.d. Introduction to GraphQL. Luettu 8.12.2018. https://graphql.org/learn/

Jiang, Z. 2016. Angular 2 is terrible. Luettu 6.12.2018. https://meebleforp.com/blog/36/angular-2-is-terrible

JS Web Frameworks Benchmark. N.d. Luettu 5.12.2018. https://stefankrause.net/js-frameworks-benchmark8/table.html

JSConfUS 2013 Tom Occhino and Jordan Walke: JS Apps at Facebook. YouTube 2013. Katsottu 20.10.2018. https://www.youtube.com/watch?v=GW0rj4sNH2w

JSS. N.d. Luettu 25.11.2018. https://cssinjs.org

Julien, S. Get Started with ngUpgrade: Going from AngularJS to Angular Luettu 2.12.2018. https://scotch.io/tutorials/get-started-with-ngupgrade-going-from-angularjs-to-angular

Kivisaari, T. 2016. API:t ovat modernin integraatiostrategian ydin. Luettu 8.12.2018. http://blog.digia.com/rest-api

Knüssel, F. 2017. Arrays, Objects and Mutations. Luettu 9.12.2018. https://medium.com/@fknussel/arrays-objects-and-mutations-6b23348b54aa

Krzywda, A. 2017, 4. Four ways to style react components. Luettu 25.11.2018. https://codeburst.io/4-four-ways-to-style-react-components-ac6f323da822

Lorefnon. 2017. RE: CSS Modules vs Styled Components. Luettu 25.11.2018. https://hashnode.com/post/css-modules-vs-styled-components-ciz2g9dt7000h7c535j35rbfu/answer/ciz2vokku000cvv53puuy8yfv

MDN web docs. N.d. a The WebSocket API (WebSockets). Luettu 8.12.2018. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

MDN web docs. N.d. b Introduction to the DOM. Luettu 8.12.2018. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

MDN web docs. N.d. c. :host(). Luettu 28.11.2018. https://developer.mozilla.org/en-US/docs/Web/CSS/:host()

Milner, J. 2018. Web Components in 2018. Luettu 10.11.2018. https://www.sitepen.com/blog/2018/07/06/web-components-in-2018/

Miško Hevery and Brad Green - Keynote - NG-Conf 2014. YouTube 2014. Katsottu 18.10.2018. https://www.youtube.com/watch?v=r1A1VR0ibIQ

MobX. N.d. Luettu 20.11.2018. https://mobx.js.org/

NG6-starter TodoMVC Example. N.d. https://github.com/AngularClass/NG6-todomvc-starter

Nutter, M. 2017. Modern Web Development. Luettu 17.10.2018. https://hackernoon.com/modern-web-development-bf0b2ef0e22e

React. N.d. a. Luettu 22.9.2018. https://reactjs.org/

React. N.d. b. Introducing Hooks. Luettu 25.10.2018. https://reactjs.org/docs/hooks-intro.html

React2Angular. N.d. The easiest way to embed React components in Angular 1 apps. https://github.com/coatue-oss/react2angular

Redux. N.d. a. Read me. Luettu 20.11.2018. https://redux.js.org/

Redux. N.d. b. Three Principles. Luettu 20.11.2018. https://redux.js.org/introduction/threeprinciples

Rendle, R. 2016. What are CSS Modules and why do we need them? Luettu 25.11.2018. https://css-tricks.com/css-modules-part-1-need/

Roadmap update, part 1: 3.0 and beyond. 2018. Luettu 6.12.2018. https://www.polymer-project.org/blog/2018-05-02-roadmap-update

Stack Overflow. 2016. Developer Survey Results. Luettu 29.11.2018. https://insights.stackoverflow.com/survey/2016

Stack Overflow. 2017. Developer Survey Results. Luettu 29.11.2018. https://insights.stackoverflow.com/survey/2017

Stack Overflow. 2018. Developer Survey Results. Luettu 29.11.2018. https://insights.stackoverflow.com/survey/2018/

Styled Components. N.d. Luettu 25.11.2018. https://www.styled-components.com/

Swartz, A. 2005. A Brief History of Ajax (Aaron Swartz's Raw Thought). Luettu 17.10.2018. https://web.archive.org/web/20181031031149/http://www.aaronsw.com/weblog/ajaxhistory

The Digest Loop and $apply. N.d. Luettu 27.10.2018. https://www.ng-book.com/p/The-Digest-Loop-and-apply/

The State of JavaScript. 2016. Front-end Frameworks. Luettu 22.9.2018. http://2016.stateofjs.com/2016/frontend/

The State of JavaScript. 2017. Front-end Frameworks – Results. Luettu 22.9.2018. https://2017.stateofjs.com/2017/front-end/results

The State of JavaScript. 2018a. Front-end Frameworks – Overview. Luettu 19.11.2018. https://2018.stateofjs.com/front-end-frameworks/overview/

The State of JavaScript. 2018b. React. Luettu 19.11.2018. https://2018.stateofjs.com/front-end-frameworks/react/

Tunguz, T. 2013. The Language of the Web. Luettu 29.11.2018. https://tomtunguz.com/the-language-of-the-web/

TypeScript. N.d. https://www.typescriptlang.org/

Ultimate NGRX: @Effects and Concepts. YouTube 2017. Katsottu 21.11.2018. https://www.youtube.com/watch?v=owFiLgOqw_Y

Using HTML Components to Implement DHTML Behaviors in Script. N.d. Luettu 23.10.2018. https://docs.microsoft.com/en-us/previous-versions//ms532146(v=vs.85)

Vue. N.d. Render Functions & JSX. Luettu 4.12.2018. https://vuejs.org/v2/guide/render-function.html

W3C. 2001. XML Binding Language. Luettu 23.10.2018. https://www.w3.org/TR/2001/NOTE-xbl-20010223/

W3Techs. N.d. Usage statistics and market share of jQuery for websites. Luettu 23.10.2018. https://w3techs.com/technologies/details/js-jquery/all/all

Wappalyzer. N.d. Websites using Angular. Luettu 6.12.2018. https://www.wappalyzer.com/technologies/angular

Web Components. N.d. Introduction. Luettu 10.11.2018. https://www.webcomponents.org/introduction

Web Components and Model Driven Views by Alex Russell. Fronteers 2011. Katsottu 23.10.2018.

What is NgRx in Angular with Mike Ryan. YouTube 2018. Katsottu 21.11.2018. https://www.youtube.com/watch?v=bycWTlHMv2Y