TAMPEREEN
AMMATTIKORKEAKOULU

WEB-KOMPONENTIN RAKENTAMINEN JA SEN SISÄLLÖNHALLINTA HTML5-SOVELLUKSESSA

Mikko Tossavainen

Opinnäytetyö
Toukokuu 2018
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TOSSAVAINEN, MIKKO:
Web-komponentin rakentaminen ja sen sisällönhallinta HTML5-sovelluksessa

Opinnäytetyö 33 sivua, joista liitteitä 0 sivua
Toukokuu 2018

Tämän opinnäytetyön toimeksiantajana oli Basware Oyj. Basware on maailman johtava verkottuneiden hankinnasta maksuun -ratkaisujen tarjoaja ja tarjoaa myös innovatiivisia verkkolasku- ja lisäarvopalveluita.

Opinnäytetyön tavoitteena oli kehittää Baswaren tuotteen päänäkymää. Osatavoitteena oli tutkia, kuinka sisällönhallintaa voitaisiin soveltaa komponenttitason sisällönhallintaan. Opinnäytetyön tarkoituksena oli rakentaa web-komponentti ja sen rinnalla pienimuotoinen sisällönhallintatyökalu, jonka avulla voitaisiin määritellä web-komponentille sisältöä.

Opinnäytetyössä käytetty AngularJS-sovelluskehys soveltui hyvin web-komponentin rakentamiseen. Sisällönhallintatyökaluun haluttiin toteuttaa tilanhallinta uudella tavalla. Tilanhallinnan rakentamiseen otettiin käyttöön MobX-kirjasto, joka yksinkertaisti tilanhallinnan rakentamista.

Opinnäytetyöstä saadut tulokset olivat hyviä. Komponenttitason sisällönhallinta pystyttiin toteuttamaan onnistuneesti, vaikka muutama ominaisuus jouduttiinkin jättämään pois. Web-komponentin toteutus onnistui hyvin, ja se saatiin tehtyä suunnitelmien mukaiseksi.

Asiasanat: sisällönhallinta, web-kehitys, angularjs, typescript, web-komponentti, tilanhallinta

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

TOSSAVAINEN, MIKKO:
Building Web-component and Managing its Content in HTML5-application.

Bachelor's thesis 33 pages, appendices 0 pages
May 2018

This thesis was commissioned by Basware Corporation. Basware is the global leader in providing networked purchase-to-pay solutions, e-invoicing and innovative financing services.

The goal of this thesis was to further develop the main view of Basware Corporation's product. Part of this goal was to study how content management could be applied to component level. The purpose of this thesis was to build a web component and content management tool which can be used to create and manage content for the web component.

AngularJS framework worked well for the building the web component. While developing content management tool, there was a need to implement a way to manage the state of the tool. Managing the state was done by using the MobX library. MobX allowed to build a simple yet effective state management solution.

The results of thesis were generally good. Implementing content management on a component level was successful. All the planned core features were implemented for the content management tool, but there were some quality of life features that ended up getting removed.

Keywords; web-development, angularjs, typescript, web-component, state management, content management

SISÄLLYS

ERITYISSANASTO

CMS	Content Management System, sisällönhallintajärjestelmä
SaaS	Software as a Service tarkoittaa ohjelmiston tarjoamista palveluna pelkän lisenssin sijaan.
DOM	Document Object Model tarkoittaa HTML-tiedoston tuottamaa rakennetta.
CSS	Cascading Style Sheet on merkkauskieli www-dokumentin tyylien hallitsemiseen.
HTML	Hypertext Markup Language on merkkauskieli, jota käytetään web-sivujen yhteydessä.

1 JOHDANTO

1.1 Tausta

Opinnäytetyön toimeksiantajana toimi Basware Oyj. Basware on maailman johtava verkottuneiden hankinnasta maksuun -ratkaisujen tarjoaja ja tarjoaa myös innovatiivisia verkkolasku- ja lisäarvopalveluita.

Opinnäytetyön aihe syntyi tarpeesta kehittää Baswaren tuotteen päänäkymää. Tavoitteeksi asetettiin uuden komponentin rakentaminen, jolla on tarkoitus parantaa käyttökokemusta siten, että käyttäjä voi navigoida sovelluksen eri ominaisuuksiin helpommin ja nopeammin.

1.2 Työn tavoitteet ja tarkoitus

Opinnäytetyön tavoitteena oli tutkia sisällönhallinnan perusteita ja soveltaa niitä komponenttitason sisällönhallintaan.

Opinnäytetyön tarkoituksena oli suunnitella ja rakentaa web-komponentti, jonka näkyvyyttä ja käyttäytymistä säädellään käyttöoikeuksien mukaisesti. Sen lisäksi komponentin rinnalle rakennettiin sisällönhallintatyökalu, jolla eri käyttäjille suunnattua sisältöä pystyy hallitsemaan. Sisällönhallintatyökalulla muokattu ja luotu data piti pystyä tallentamaan palvelimelle siten, että se on yhteensopiva rakennetun komponentin rajapinnan määrittelemän tietomallin kanssa.

1.3 Opinnäytetyön vaiheet

Opinnäytetyössä selvitetään lukijalle, mitä sisällönhallinnalla tarkoitetaan ja missä sitä pääasiallisesti käytetään. Tarkoitus on tutkia sisällönhallinnan mahdollisuuksia ja tarkastella tämän hetken käytetyimpiä sisällönhallintatyyppejä. Tämän jälkeen on tarkoituksena käydä läpi kehityksessä käytetty kehitysympäristö ja sitä ympäröivät kehitystyökalut.

Opinnäytetyössä käsitellään työn eteneminen ja sen eri toteutusvaiheet. Samalla kerrataan, mitä ongelmia kohdattiin kehitettäessä komponenttia ja sisällönhallintatyökalua, ja millaisilla ratkaisulla niistä ongelmista selvittiin.

Lopuksi eri vaiheiden läpikäymisen jälkeen pohditaan opinnäytetyöstä saatuja tuloksia ja sitä, kuinka tehtyä työtä voisi vielä jatkokehittää tilanteen niin vaatiessa.

2 SISÄLLÖNHALLINTA

Tässä luvussa käydään läpi, mitä sisällönhallinnalla tarkoitetaan. Sen lisäksi tutustutaan sisällönhallinnan ja tietomallin väliseen yhteyteen. Lopuksi tarkastellaan sisällönhallinnan eri kategorioita.

2.1 Sisällön määritelmä

Sisältö on tietoa, joka on luotu tarkoituksellisesti muiden käyttöä varten. Määritelmää voidaan suoraan soveltaa sisällönhallinnan kaksiosaiseen pääkäsitteeseen, jossa sisältöä luodaan ja hallitaan, minkä jälkeen se julkaistaan ja jaetaan.

2.2 Sisällönhallintajärjestelmä (CMS)

CMS:llä pyritään automatisoimaan prosesseja, joilla luotu sisältö tallennetaan ja julkaistaan muiden hyödynnettäväksi tai uudelleenkäytettäväksi. CMS:n tarkoituksena ei ole automatisoida itse sisällön luomista, vaan ainoastaan luodun sisällön hallitsemista, julkaisemista ja jakelua. (Barker 2016.)

CMS on yleensä palvelinpohjainen sovellus, joka on vuorovaikutuksessa tallennetun sisällön kanssa. CMS koostuu eri osista, joita ovat esimerkiksi sisällönmuokkauskäyttöliittymä, säilö sisällölle sekä sisällön julkaisutyökalut. Nämä osiot eivät ole välttämättä kuitenkaan aina saman sovelluksen alaisena, vaan ne ovat usein myös hajautettu useisiin eri järjestelemiin ja palveluihin. (Barker 2016.)

CMS:llä on yleensä useita eri käyttäjiä, jotka voivat muokata, lisätä ja julkaista sisältöä muiden hyödynnettäväksi (Barker 2016).

2.3 Sisällönhallinta ja tietomalli

Jokaisessa sisällönhallintajärjestelmässä käytetään eri tietomalleja. Tietomalli määrittelee, minkälaista sisällönhallintajärjestelmän tuottaman sisällön pitää olla.

Tietomalli koostuu määritellyistä attribuuteista, niiden tyypeistä sekä yhteyksistä toistensa välillä. Tietomallia tehtäessä pitää pyrkiä rakentamaan se niin tasapainoisesti, että se pysyy yksinkertaisena ja joustavana sekä on mahdollisimman valmis käytettäväksi kaikissa tilanteissa. Tämä ei kuitenkaan aina onnistu. Tällöin joudutaan yleensä rajoittamaan joustavuutta tai monimutkaistamaan tietomallia (Barker 2016).

2.4 Sisällönhallinnan eri kategoriat

Sisällönhallinta on laaja käsite, jota voidaan tulkita eri tavoin. Sisällönhallinta voidaan nykyään jakaa neljään yleisimpään pääkategoriaan, jotka ovat WCM, ECM, DAM sekä RM.

WCM

WCM (Web content management) eli web-sisällönhallinta. WCMS (web content management system) tarjoaa erilaisia työkaluja web-sivun sisällön luomiseen tai hallitsemiseen. WCM-järjestelmät toimivat usein siten, että niiden käyttäjät pystyvät luomaan ja hallitsemaan sisältöä ilman teknistä ymmärrystä web-ohjelmointi- tai merkkauskielistä. (Wikipedia 2018.)

ECM

ECM (Enterprise content management) eli yrityspohjainen sisällönhallinta. ECM on yleensä tarkoitettu yrityksen eri dokumenttien ja muun yrityksen sisäisen tiedon hallitsemiseen. ECM järjestelmät ovat hyviä työkaluja yhteistoiminnan hallitsemiseen, pääsyn rajoittamiseen ja yrityksen sisäisten tiedostojen hallintaan. (Barker 2016.)

DAM

DAM (Digital asset management) eli digitaalisen omaisuuden hallinta. DAM-järjestelmän avulla pystytään tarjoamaan keskitetty palvelu digitaalisen omaisuuden hallitsemiseen. DAM-palvelu on yleensä SaaS-pohjainen, jossa palveluntarjoaja hoitaa tarvittavan

kiintolevytilan ja muut vaadittavat tarpeet. Joskus DAM-palvelua pidetään myös omassa verkossa, esimerkiksi yrityksen sisäisen digitaalisen omaisuuden hallintaan. (Widen, n.d)

DAM-järjestelmän ei ole tarkoitus pelkästään säilöä sisältöä, vaan sillä pyritään tehokkaasti tuomaan sisältö muiden hyödynnettäväksi ja uudelleen käytettäväksi. DAM koostuukin eri tavoista tallentaa, järjestää, hakea, jakaa ja hankkia erityyppistä digitaalista omaisuutta, kuten ääniä, kuvia tai videoita. (Koppatz 2017.)

RM

RM (Records management) eli arkistonhallinta. Arkistonhallintaa käytetään liiketoiminnallisen tiedon ja muiden liiketoiminnan operaatioiden arkistojen hallitsemiseen. Arkistonhallinta on eniten käytössä arkistojen säilömisessä ja pääsynvalvonnassa. (Barker 2016.)

3 WEB-KEHITYKSEN TYÖKALUT JA OPINNÄYTETYÖSSÄ KÄYTETYT TEKNOLOGIAT

Tässä luvussa kuvataan web-kehityksessä käytettyjä perustyökaluja sekä eri teknologioita, joita käytettiin web-komponentin ja sisällönhallintatyökalun rakentamiseen tässä opinnäytetyössä.

3.1 Web-kehitys

Web-kehityksessä kehittäjällä on yleensä vapaus valita eri työkalujen ja teknologioiden välillä. Projekteissa voidaan käyttää myös asiakkaan valitsemia työkaluja ja teknologioita, sekä jo työstettävään projektiin valittuja teknologioita.

3.2 Lähdekoodieditori

Yleisesti kehittäjä saa valita käytettävän lähdekoodieditorin. Lähdekoodieditorissa pystyy yleensä tarkastelemaan projektin hierarkiaa, josta voidaan navigoida eri projektin tiedostojen välillä. Tämän lisäksi eri lisäosia pystytään lisäämään editoriin kehittäjän tarpeiden mukaisesti. Lisäosat tuovat usein helpotusta esimerkiksi koodin siisteyden, virheiden ja eri ohjelmointikielien syntaksin hallintaan. Moneen koodieditoriin on lisätty työkalut, joiden avulla versionhallinnan perustoiminnallisuuksia voidaan käyttää suoraan editorista käsin käyttämättä konsolia.

Tärkeitä lisäosia lähdekoodieditorissa ovat virheenjäljittäjä ja sisäänrakennettu konsoli, jolla voidaan havaita virheitä koodin suorituksen aikana. Virheidenjäljittäjä pystyy paikantamaan kohdan, jossa virhe tapahtui ja näyttämään sen tarkasti koodissa.

Tämän lisäksi lähdekoodieditorin ulkonäköä pystyy joko vapaasti muokkaamaan itse tai valitsemaan valmiista teemoista, jotka muokkaavat koko editorin ulkonäköä kehittäjän mielen mukaan.

Opinnäytetyössä selainpuolen kehitykseen käytettiin Visual Studio Code -lähdekoodieditoria.

Visual Studio Code on Microsoftin rakentama avoimeen lähdekoodiin perustuva lähdekoodieditori, jossa on valmiiksi sisäänrakennettuna virheidenkorjaus- ja git-versionhallintatyökalut sekä muita ominaisuuksia, kuten syntaksin korostaminen ja koodin automaattinen täydentäminen. Kustomoitavuutta editoriin tuo helppous lisätä lisäosia editoriin sisäänrakennetulla lisäosaselaimella ja -haulla. Tämän lisäksi Visual Studio Code antaa mahdollisuudet räätälöidä editorin väriteemoja ja muita tyylejä kehittäjän mieltymyksien mukaisiksi.

Opinnäytetyön projektissa käytettävässä AngularJS Javascript-sovelluskehyksessä käytetään Typescript-ohjelmointikieltä. Visual Studio Code valittiin käyttöön, sillä siinä on valmiina tuki Typescript-ohjelmointikielelle.

3.3 Projektinrakennus työkalut

Web-kehitystä tehtäessä on yleisesti työkaluja tai kirjastoja, joilla voidaan helpottaa esimerkiksi riippuvaisuuksien tai pakettienhallintaa. Sen lisäksi projekteissa on usein mukana jokin työkalu, jolla voidaan automatisoida tiedostojen kääntämistä.

3.3.1 Gulp

Gulp on työkalu, jolla voidaan automatisoida tehtäviä, joita tehdään web-kehityksen yhteydessä. Gulpin avulla kehitystä voidaan nopeuttaa, jos tehtävillä pyritään automatisoimaan normaalisti aikaa vieviä vaiheita. Yleisiä Gulpilla automatisoituja tehtäviä ovat esimerkiksi tiedostojen kääntäminen sekä kooditiedostojen optimointi.

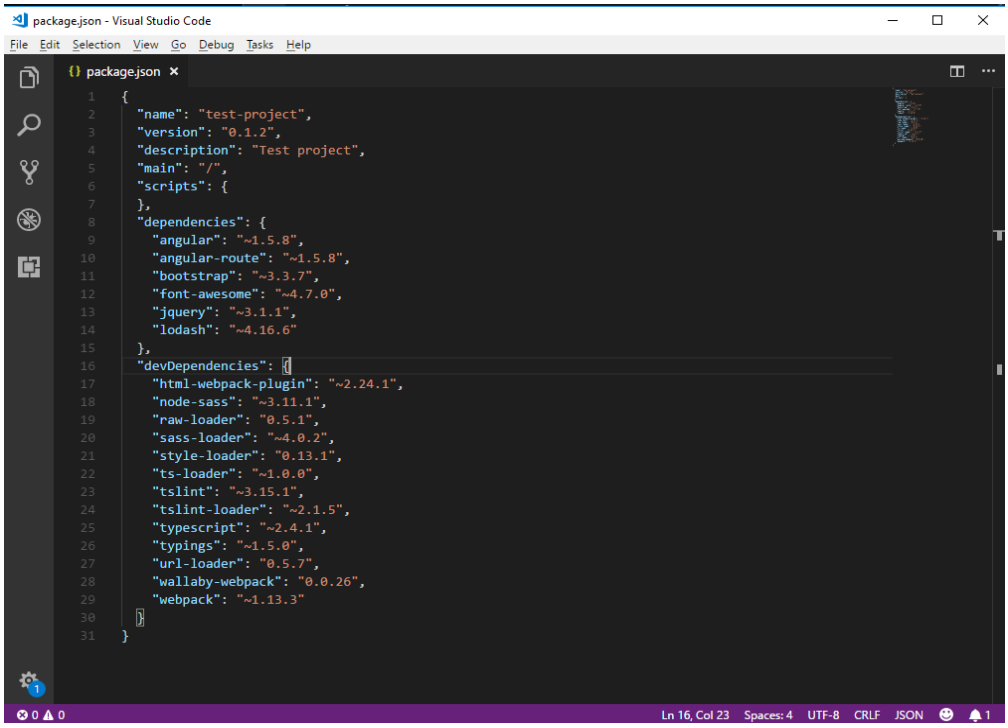
Opinnäytetyöprojektissa Gulpia käytetään esimerkiksi web-palvelimen pystytykseen, tiedostojen kääntämiseen sekä muutoksien päivittämiseen web-palvelimelle automaattisesti muutoksia tallennettaessa. Sen lisäksi Gulpilla on rakennettu useita muitakin tehtäviä, joiden tarkoituksena on nopeuttaa kehitysprosessia.

3.3.2 Yarn

Yarn on Facebookin kehittämä paketti- ja moduulienhallintatyökalu. Yarnin välityksellä kehittäjä pystyy lisäämään projektiin tarvittavia paketteja ja moduuleita. Yarnin kautta ladattavat paketit ja moduulit ovat hyvinkin erilaisia, ja niitä löytyy moneen eri tarpeeseen.

Paketit ovat yleensä valmiita node-pohjaisia ohjelmia, joita voidaan käyttää esimerkkipohjana rakennettavalle sovellukselle. Moduulit taas ovat yleensä työkaluja tai muita kehitystä ja projektia auttavia kirjastoja. Moduuleille, paketeille ja Node.js sovelluksille yhteistä on package.json tiedosto. Package.json-tiedostossa määritellään, mitä riippuvuuksia eri moduuleihin ja paketteihin projektissa on.

Node.js-paketeissa ja -moduuleissa on myös itsessään sisällä package.json-tiedosto, joka voi sisältää niiden tarvitsemia riippuvuuksia. Package.json-tiedostossa voidaan jakaa riippuvuudet kehityksessä tarvittaviin ja sovelluksen ajamisessa tarvittaviin riippuvuuksiin. Tämän lisäksi tiedosto sisältää tietoa esimerkiksi projektin nimestä ja versiosta. Kuvassa 1 näytetään, mitä package.json-tiedosto voi sisältää.



KUVA 1. Esimerkki package.json-tiedostosta Visual Studio Code -lähdekoodieditorissa.

Yarn-komentoa käytettäessä Yarn asentaa ja lataa tarvittavat riippuvuudet npm-rekisteristä turvallisesti. Yarn tarkistaa riippuvuuksien eheyden ennen kuin niiden koodia suoritetaan, jotta voidaan välttää tietoturvariskejä.

Opinnäytetyön projektissa Yarnin avulla asennettuja moduuleita ovat esimerkiksi Typescript, AngularJS, MobX sekä LESS.

3.4 Typescript

Typescript on Microsoftin kehittämä avoimeen lähdekoodiin perustuva ohjelmointikieli, joka on rakennettu Javascriptin päälle. Typescript tuo Javascriptiin lisää vahvan tyypityksen, joka auttaa välttämään tiedon tyyppivirheitä. Tämän lisäksi Typescript tuo Javascriptiin rajapintaluokat, joita käytetään yleisesti oliopohjaisessa ohjelmoinnissa. Typescriptissä on myös tuki ECMAScript 2015 -standardille, joka sisältää erilaisia ominaisuuksia, kuten luokat ja moduulit.

Typescriptin vahva tyypitys helpottaa ohjelmoijan tehtäviä siten, että ohjelmoija näkee mahdolliset tyypitysvirheet jo koodia kirjoitettaessa. Typescriptiä voi kuitenkin kirjoittaa niin kuin normaalia Javascriptiä, sillä Typescript ei pakota tyypitystä, jollei sitä niin ole määritetty. Typescript-koodi muuttuu käännettäessä suoraan yhteensopivaksi Javascriptiksi, eikä täten muuta sen alkuperäistä syntaksia.

3.5 AngularJS

AngularJS on Javascript-sovelluskehys, jonka avulla voidaan rakentaa erilaisia selainpohjaisia (client-side) web-sovelluksia. AngularJS pyrkii tarjoamaan ratkaisuja kehittäjien yleisesti kohtaamiin ongelmiin sekä vähentämään kirjoitetun koodin määrää. (Tarasiewicz & Böhm 2014.)

3.5.1 Historia ja tulevaisuus

Historia

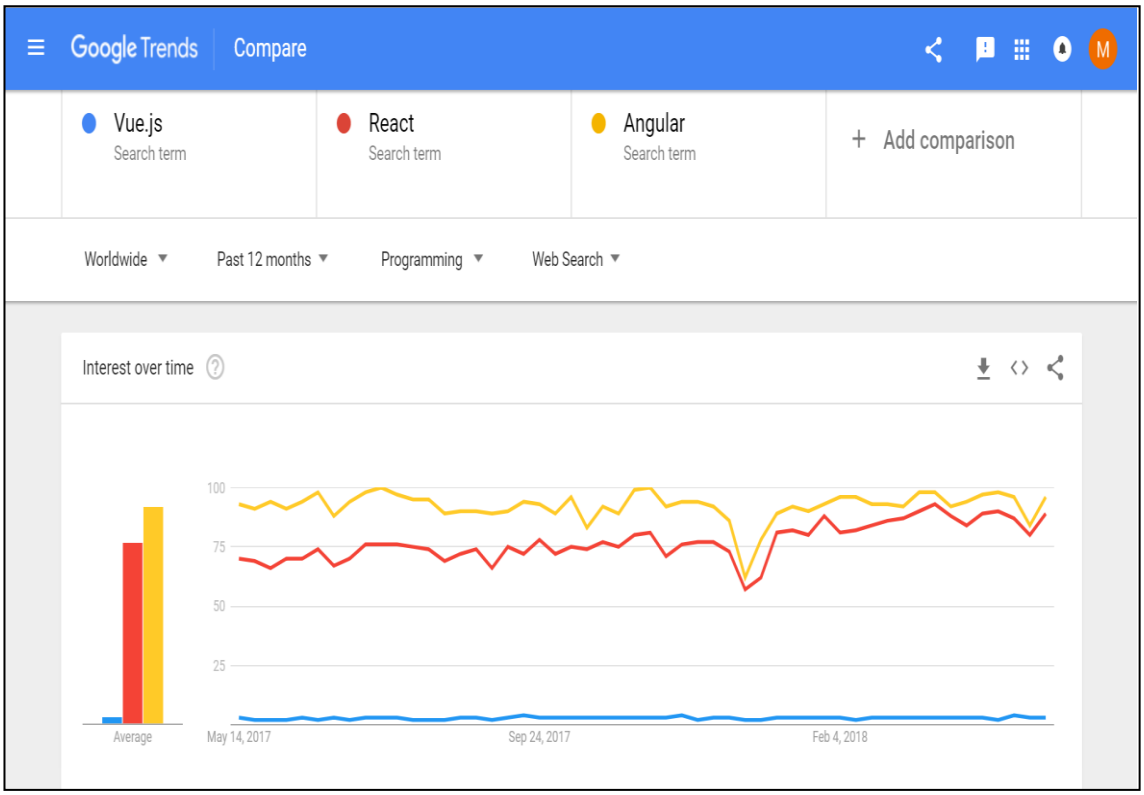
AngularJS:ää alettiin kehittää vuosina 2008-2009 Googlen työntekijän Miško Heveryn toimesta. Hevery työskenteli AngularJS:n parissa sivuprojektissa, jonka tavoitteena oli yksinkertaistaa web-kehitystä. Vuonna 2010 Google päätti sisäisten kokeilujen jälkeen, että se alkaa jatkokehittää AngularJS:ää. Sen tarkoituksena oli julkaista versio ennen 2010 syyskuuta. Erinäisten myöhästelyjen jälkeen AngularJS:n ensimmäinen versio tuli kuitenkin ulos vasta 2011 maaliskuussa. (Gudelli 2017.)

Nykyhetki ja tulevaisuus

AngularJS:n viimeisin versio on tällä hetkellä 1.6.X, mutta sen viimeistä versiota 1.7 ollaan kehittämässä. Se on tarkoitus julkaista heinäkuussa 2018. Tämän jälkeen AngularJS-tukea jatketaan vain kriittisten bugien tietoturvariskien korjaamiseksi, eikä uusia ominaisuuksia enää rakenneta. AngularJS-tuki lopetetaan kokonaan kesäkuussa 2021.

Vaikkakin AngularJS-kehitys loppuu, on Google rakentanut rinnalle Angular-nimisen Javascript-sovelluskehyksen, joka on täysin alusta alkaen uudelleen rakennettu. Angular jatkaa siitä mihin AngularJS jäi ja pyrkii olemaan yksi johtavista moderneista web-sovelluskehyksistä. Angular tuo paljon lisää ominaisuuksia AngularJS:ään nähden ja poistaa kokonaan esimerkiksi kaksisuuntaiset datakytkennät. Angularin uusin versio on Angular 6 ja se julkaistiin toukokuussa 2018.

Angularin tulevaisuus näyttää valoisalta, sillä se kamppailee kiinnostavimman Javascript-sovelluskehyksen paikasta kilpailijoidensa kanssa aktiivisesti. Angularin kiinnostavuus verrattuna kilpaileviin sovelluskehyksiin on hyvin tasaista. Kuvassa 2 näytetään Angularin esiintyvyys muihin kilpailijoihin nähden Google-hauissa.



KUVA 2. Angularin esiintyvyys Google-hauissa viimeisen vuoden aikana.

3.5.2 AngularJS syntaksi

Angularin syntaksi sisältää erilaisia lisäyksiä HTML-merkkauskieleen. HTML-elementtien toiminallisuutta laajennetaan direktiiveillä sekä datakytkentälausekkeilla, jotka lisätään suoraan HTML-tiedostoon. AngularJS-direktiivien avulla voidaan opettaa selainta tulkitsemaan AngularJS:llä tehtyjä HTML-elementtejä.

Kuvassa 3 näytetään, kuinka AngularJS voidaan ottaa käyttöön HTML-dokumentissa. Sen lisäksi siinä näytetään, miten ng-app-direktiivin avulla voidaan antaa haluttu osa DOMia AngularJS:n ohjattavaksi. HTML-rakenteesta löytyy myös ng-model-direktiivi, jota käytetään ”<input>”-tyyppisten elementtien sisältämän arvon tallentamiseen. Tätä arvoa voidaan käyttää muualla ng-app-direktiivin sisällä. Kuvassa ng-bind-direktiivi sitoo ng-modelin tallentaman arvon haluttuun <p>-elementtiin ja näyttää sen sisällön <p> -elementin tekstiosiossa dynaamisesti.

KUVA 3. AngularJS:n syntaksi HTML-tiedostossa.

3.5.3 AngularJS-moduulit

AngularJS-sovellus jaetaan yleensä loogisiin moduuleihin. Päätason moduulia käytetään ng-app-direktiivissä, joka rakentaa AngularJS-sovelluksen. Päätason moduuliin voidaan liittää riippuvuuksiksi muita moduuleita, joita sovelluksessa käytetään. Moduulit sisältävät eri sovelluksen osia, kuten direktiivejä, rajapintoja sekä eri tyyppisiä suodattimia ja ohjaimia.

Sovelluksen eri osat suositellaan jakamaan eri moduuleihin siten, että jokainen ominaisuus, komponentti ja sovelluksen päätason moduuli ovat kaikki omia moduuleitansa. Moduulit sisältävät yleensä ohjaimen, HTML-templaatin sekä HTML-templaattiin linkitetyn tyylitiedoston.

Moduuleiden sisällä oleva ohjain pystyy normaalisti käsittelemään tietoa ainoastaan sen oman HTML-templaatin sisällä, ellei sille ole määritelty laajempia oikeuksia. Ohjain toimii moduulin alustajana, jossa voidaan määritellä haluttuja arvoja.

3.5.4 AngularJS-direktiivit

AngularJS-syntaksissa käytetään AngularJS-direktiivejä, joilla voidaan lisätä ominaisuuksia tai opettaa selain tulkitsemaan AngularJS:llä luotuja uusia HTML-elementtejä. AngularJS-sovelluksessa yleisimpiä direktiivejä ovat ng-app, ng-model, ng-bind sekä ng-repeat.

Ng-app-direktiivi

Mihin tahansa HTML-elementtiin voidaan lisätä ng-app-direktiivi, joka määrittelee missä osassa DOM-puuta AngularJS:ää käytetään. Ng-app-direktiivi esiladataan automaattisesti sivua ladattaessa. Normaalisti ng-app-direktiivejä on vain yksi HTML-dokumenttia kohden. Haluttaessa voidaan käyttää useampaa kuin yhtä ng-app direktiiviä yhdessä HTML-dokumentissa, mutta silloin ne pitää esiladata manuaalisesti koodista angular-funktion avulla. Ng-appin käyttö kuitenkin rajoittuu niin, että ng-appin merkkaaman elementin sisällä ei voi olla toista ng-app-direktiiviä. (Angular documentation n.d)

Ng-model-direktiivi

Ng-model on direktiivi, joka voidaan lisätä HTML-lomakkeiden sisällä käytettäviin HTML-elementteihin. Lomake-elementin sisällä käytetään yleisesti HTML-elementtejä, joihin voidaan syöttää tekstiä tai numeroita, sekä valintaelementtejä, joista käyttäjä saa valittua yhden tai useamman vaihtoehdon. Ng-model sitoo kenttiin syötetyn arvon haluttuun muuttujaan, jonka se tarjoaa käytettäväksi muualla AngularJS-näkymän (scope) sisällä.

Ng-model lisää myös validointimahdollisuudet kenttiin, jotka voivat esimerkiksi sisältää sähköpostiosoitteen, postiosoitteen tai puhelinnumeron. Ng-model pitää yllä myös kentän tilaa ja lisää siihen eri tyyppisiä tyyliluokkia. Ng-model lisää tyyliluokkia riippuen siitä, onko kentän sisältö virheellistä vai ei ja onko kentän sisältöä muokattu. Haluttaessa kehittäjä voi reagoida AngularJS:n tyyliluokkiin lisäämiin tyylimäärityksiin, miten kentän näkymän pitää muuttua halutuissa tilanteissa. (AngularJS documentation n.d.)

Ng-bind-direktiivi

HTML-elementteihin voidaan myös lisätä ng-bind-direktiivi, joka korvaa kyseisen HTML-elementin tekstisisällön. Yleisesti ng-bindin sijasta käytetään aaltosulkunotaatiota, jossa aaltosulkujen sisään tuodaan haluttu sisältö ng-bindin sijasta.

Ng-repeat-direktiivi

Yleisesti käytetyn ng-repeat-direktiivin avulla voidaan dynaamisesti luoda uusia elementtejä siihen sidotusta HTML-elementistä. Ng-repeatia ohjastetaan siten, että sille annetaan

lista, jonka solujen määrän mukaan elementtejä luodaan. Ng-repeatille voidaan antaa erikoisattribuutteja sekä suodattimia, jotka määrittävät miten elementit luodaan ja kuinka monta elementtiä luodaan.

3.5.5 AngularJS-komponenttidirektiivi

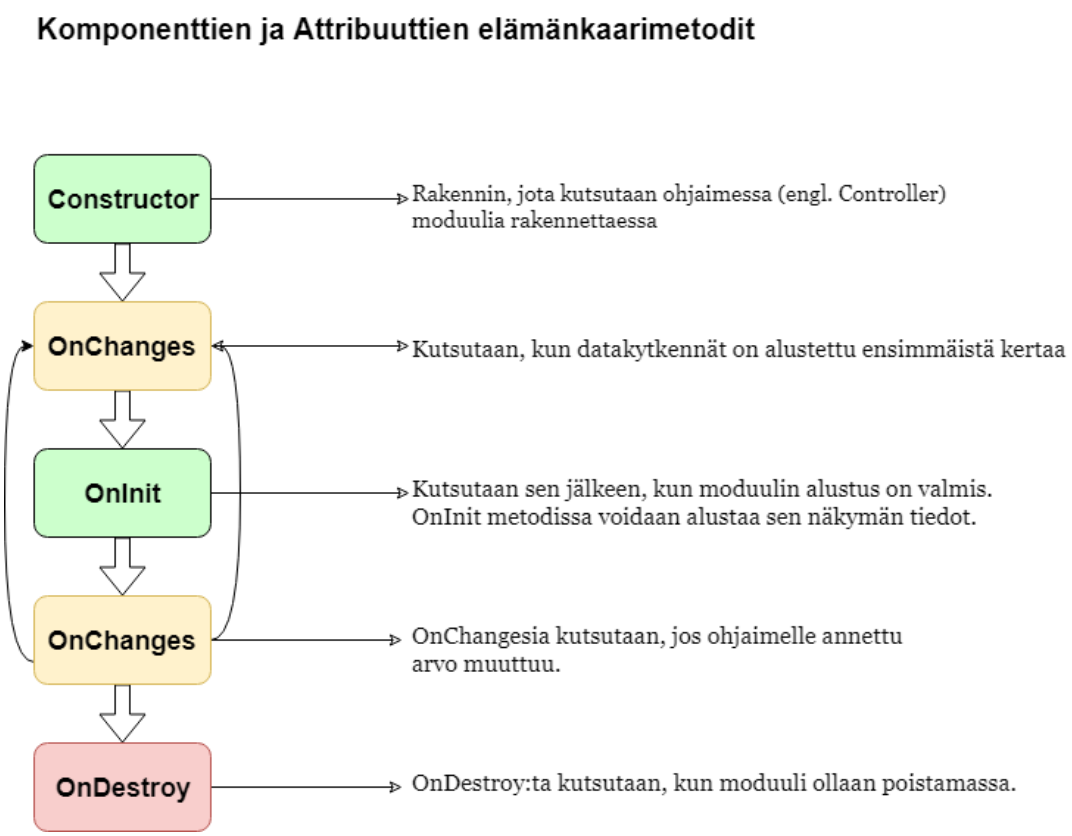
AngularJS-komponenttidirektiivi on erikoisdirektiivi, jonka ominaisuudet ovat yksinkertaisemmat kuin tavallisessa direktiivissä. Komponenttidirektiivillä on tiettyjä etuja normaaleihin direktiiveihin verrattuna. Ne ovat optimoituja komponenttipohjaiseen arkkitehtuuriin ja ne edistävät parhaita toimintatapoja. Komponenttidirektiivin käyttäminen mahdollistaa migraation uuteen Angular-versioon helposti, koska sen toimintamalli on hyvin samankaltainen.

3.5.6 Angular-komponenttien ja direktiivien elämänkaari

Angular-komponenteilla ja direktiiveillä on saatavilla elämänkaarifunktioita, joita kutsutaan ohjelmallisesti eri vaiheissa. Näiden avulla voidaan suorittaa eri toimintoja, joita halutaan esimerkiksi tehdä, kun direktiivi poistetaan DOM-puusta.

Kuviossa 3 näytetään direktiivien ja komponenttien jakamia elämänkaarimetodeja. Komponentilla on kuviossa näytettyjen elämänkaarimetodien lisäksi vielä muutama metodi lisää, mutta ne eivät ole kovin yleisesti käytössä.

Komponenteissa käytetään yleisesti rakentajafunktiota (constructor) rakentamaan näkymä. Tämän jälkeen ajetaan OnChanges-metodi, jolle kerrotaan parametrin avulla, mitä datakytkentöjä sille tulee ja mitä niiden arvot sisältävät. OnInit-metodissa on suositeltavaa alustaa komponentille tärkeät muuttujat. OnChangesia suoritetaan myös niissä tilanteissa, kun datakytkentöihin tuleva data muuttuu edellisestä tilasta, jolloin siihen voidaan reagoida ja päivittää komponenttia sen mukaisesti.



KUVIO 3. Angular-komponenttien ja direktiivien elämänkaari.

3.6 MobX

Angularin avulla voidaan hallita sovelluksen tilaa luomalla palveluita sekä tilanhallinta-ohjaimia, jotka käsittelevät ja sisältävät tiedon sovelluksen tai näkymän tilasta. Sovelluksen koon kasvaessa tästä voi tulla hankalaa, jolloin on hyvä käyttää jotain tilanhallintaan suunniteltua kirjastoa.

MobX on hyvä kirjasto tilanhallintaan, sillä sen avulla tilanhallintaa voidaan yksinkertaistaa ja keskittää. MobX:n perusperiaatteita on se, että haluttu tila pidetään yllä varastossa (store), eikä sen attribuutteja voida muuttaa, kuin sille määritetyillä toiminnoilla (actions). Attribuutteja voidaan hakea suoraan varastosta. Sen lisäksi erillisiä arvoja voidaan hakea arvon palauttavien toimintojen (computed actions) avulla. Arvon palauttavat toiminnot palauttavat arvon, joka voi määrittyä suoraan yhden tarkkailtavan attribuutin perusteella. On kuitenkin yleistä, että niiden palauttama arvo muodostuu useista eri asioista.

Tarkkailtavien attribuuttien muutokseen voidaan reagoida joko arvon palauttavien toimintojen avulla tai autorun-funktiolla, joka ajetaan aina, kun sen sisällä käsitelty varaston attribuutti muuttuu. Näiden kahden erona on kuitenkin se, että arvon palauttavia toimintoja kannattaa käyttää silloin, kun halutaan luoda arvo, jota muut tarkkailtavat attribuutit voivat käyttää. Autorun-funktiota suositellaan käytettäväksi silloin, kun halutaan vain tehdä jotain halutun attribuutin muuttuessa. (Mobx reference guide n.d.)

Opinnäytetyön projektissa MobX otettiin käyttöön rakennettaessa sisällönhallintatyökalua. Sen toimivuutta haluttiin testata sovelluksen tilanhallintaan. Sen toiminta todettiin hyväksi, ja sitä on tarkoitus käyttää jatkossa uusia ominaisuuksia rakennettaessa.

3.7 LESS

LESS eli Leaner Style Sheets on lisäosa CSS eli Cascasding Style Sheetsille, jota käytetään WWW-dokumentin tyylien määrittelemiseen. LESS tuo CSS:ään lisää ominaisuuksia, jotka helpottavat sen käyttöä. Isoimpia muutoksia ovat muun muassa funktiot sekä hierarkkinen rakenne (nested), jolla voidaan helposti määritellä tyylit vastaamaan WWW-dokumentin HTML-rakennetta.

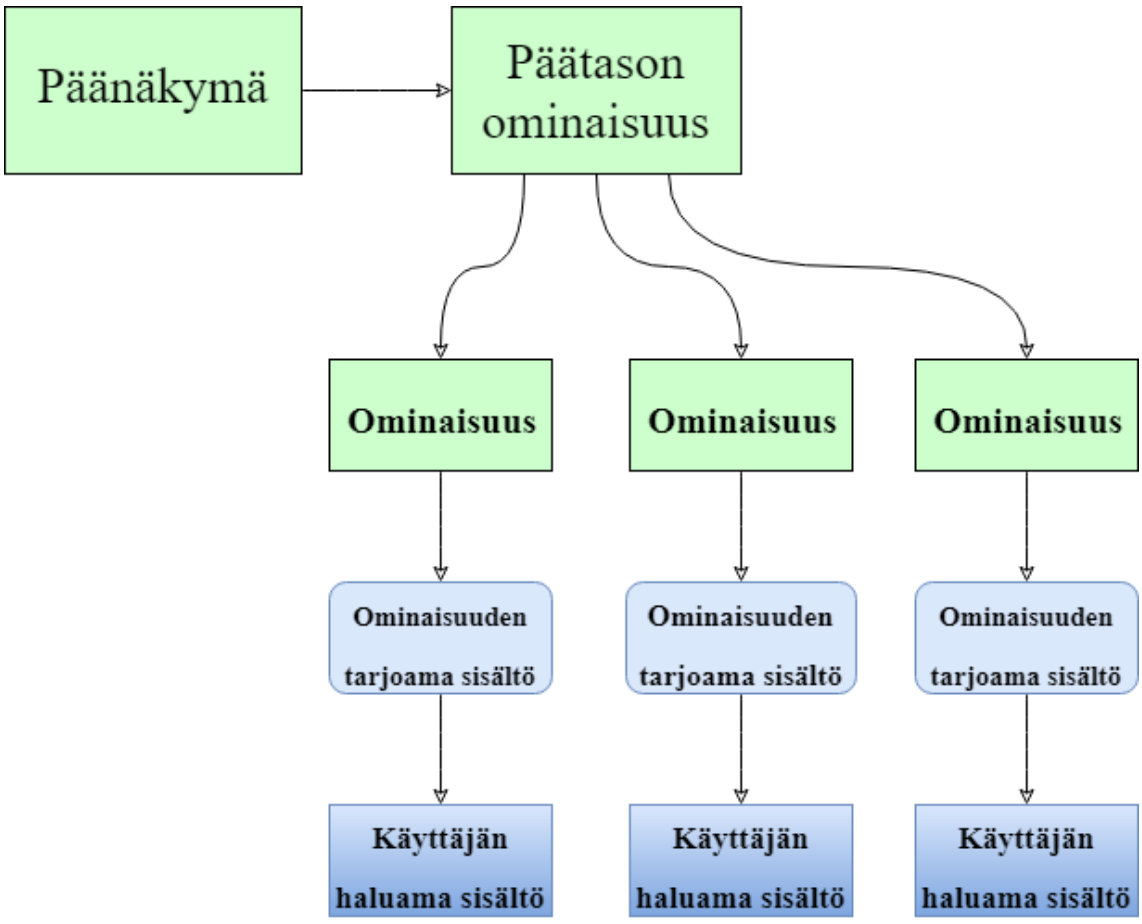
Opinnäytetyöprojektissa LESSiä on käytetty kaikkien sovelluksen näkymien tyylittelyyn. Sen rooli komponenttidesignissa on suuri, sillä LESS mahdollistaa monimutkaisten tyylien määrittelemisen helposti verrattuna normaaliin CSS:ään.

4 WEB-KOMPONENTTI JA SISÄLLÖNHALLINTATYÖKALU

4.1 Opinnäytetyön taustat

Opinnäytetyön taustoihin liittyi tarve, jossa käyttäjien navigointia Baswaren web-sovelluksen sisällä haluttaisiin nopeuttaa. Haluttiin luoda ratkaisu, jossa päänäkymään voitaisiin lisätä uusi osio. Uuden osion pitää sisältää keskitetty kokoelma, josta päästäisiin tuotteen yleisimmin käytettyihin ominaisuuksiin nopeasti. Sovelluksen ominaisuuksien määrä ja sisältö vaihtelevat käyttäjän mukaisesti. Siksi komponentin sisältö piti pystyä määrittelemään erikseen jokaiselle käyttäjäryhmälle.

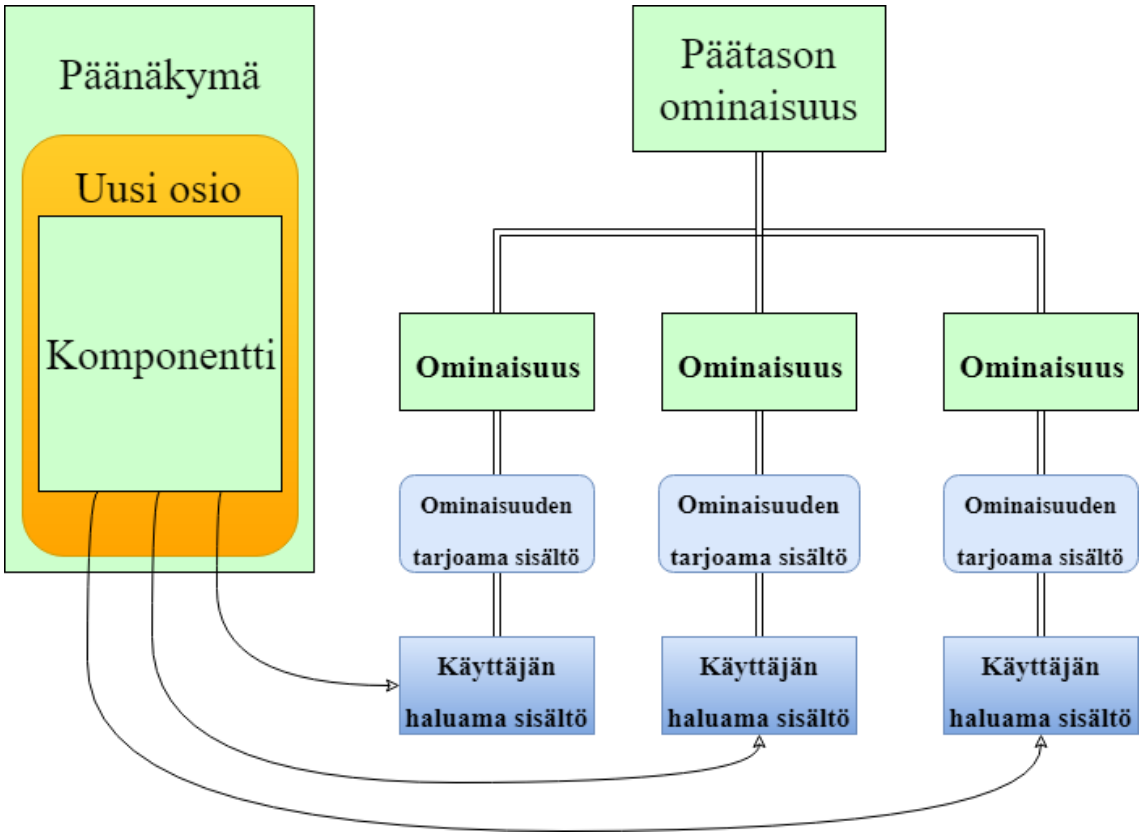
Kuviossa 4 näytetään, kuinka navigointi eri ominaisuuksiin ja niiden tarjoamiin sisältöihin tapahtuu. Kuviossa nuoli vastaa aina yhtä hiiren klikkausta tai toimintoa sovelluksessa.



KUVIO 4. Haluttuun sisältöön navigointi normaalisti.

Kuviossa 4 esiteltyjen toimintojen ja klikkauksien määrää haluttiin vähentää, jolloin käyttökokemuksesta tulisi parempi, kun käyttäjät löytäisivät haluamansa sisällön nopeammin.

Kuviossa 5 näytetään, kuinka uudesta osiosta voidaan navigoida suoraan käyttäjien tarvitsemiin ominaisuuksiin ja niiden sisältöihin. Uuteen osioon piti rakentaa web-komponentti, jossa näytettään määritellyt linkit käyttäjien haluamiin sisältöihin.



KUVIO 5. Haluttuun sisältöön navigointi uuden toteutuksen avulla.

4.2 Suunnittelu

Suunnitteluvaiheessa piti huomioida kehitettävää ominaisuutta vastaavat designit. Ominaisuuteen piti rakentaa sekä web-komponentti että sisällönhallintatyökalu. Sisällönhallintatyökalun avulla piti pystyä määrittelemään käyttöoikeuksien mukaista sisältöä, jota web-komponentin on tarkoitus näyttää.

4.3 Web-komponentti

Opinnäytetyön alkuvaiheissa rakennettiin designien mukainen prototyyppi web-komponentista. Web-komponentin oli tarkoitus olla karusellityyppinen. Karusellissa sisältöä näytetään eri tavoin riippuen sisällön määrästä. Sisällön määrän kasvaessa tarpeeksi karusellin piti mukautua siten, että se jakaisi sisällön useaan näkymään. Prototyypin tarkoituksena oli tutkia, miten karuselli kannattaisi rakentaa, kun huomioon piti ottaa vaaditut ominaisuudet. Karusellin piti olla helppokäyttöinen mobiililaitteita käytettäessä. Lisäksi sen piti pystyä näyttämään sisältöä määriteltyjen arvojen mukaisesti.

Projektin aikataulun huomioiden oli selvää, että aikaa alusta alkaen rakennettavaan karusellikomponenttiin ei ole. Tämän vuoksi tutkittiin valmiita moduuleita ja kirjastoja, jotka ovat jo toteuttaneet karusellikomponentteja. Prototyyppivaiheessa testattiin eri kirjastoja. Lopuksi päädyttiin Slick-nimiseen karusellikirjastoon, josta löytyi kaikki vaadittavat ominaisuudet. Slickin valinnan jälkeen se piti ottaa käyttöön ja testata sen toimivuus käytännössä.

Kun Slick-kirjasto oli lisätty projektiin, aluksi oli vaikeuksia saada Slick-kirjasto toimimaan sovelluksen muun ympäristön kanssa. Tämä johtui siitä, että tarvittavat Typescript-tyypitykset puuttuivat. Lisäksi kirjasto oli otettu käyttöön ilman mitään integraatiota AngularJS-sovellukseen. Ongelma ratkaistiin lisäämällä projektiin toinen paketti, joka tarjoaa Slickin ominaisuudet valmiiksi AngularJS:n yhteensopivana. Tämä nopeutti Slickin käyttöönottoa huomattavasti siten, että Slickiä pystyttiin käyttämään suoraan HTML-templaatissa AngularJS-direktiivinä.

4.3.1 Komponentin datakytkennät ja datarajapinta

Web-komponentti rakennettiin käyttämällä AngularJS Javascript -kehystä, jolla on määritelty tapa tehdä web-komponentteja. Komponentille määritellään eri datakytkentöjä, joiden avulla voidaan kertoa, minkä tyyppistä dataa kyseiselle komponentille voidaan syöttää. Komponentin designin tarpeiden mukaisesti pystyttiin luomaan tarvittavat datakytkennät, jotka tulisivat vastaamaan sille luodun sisällön datarajapintaa.

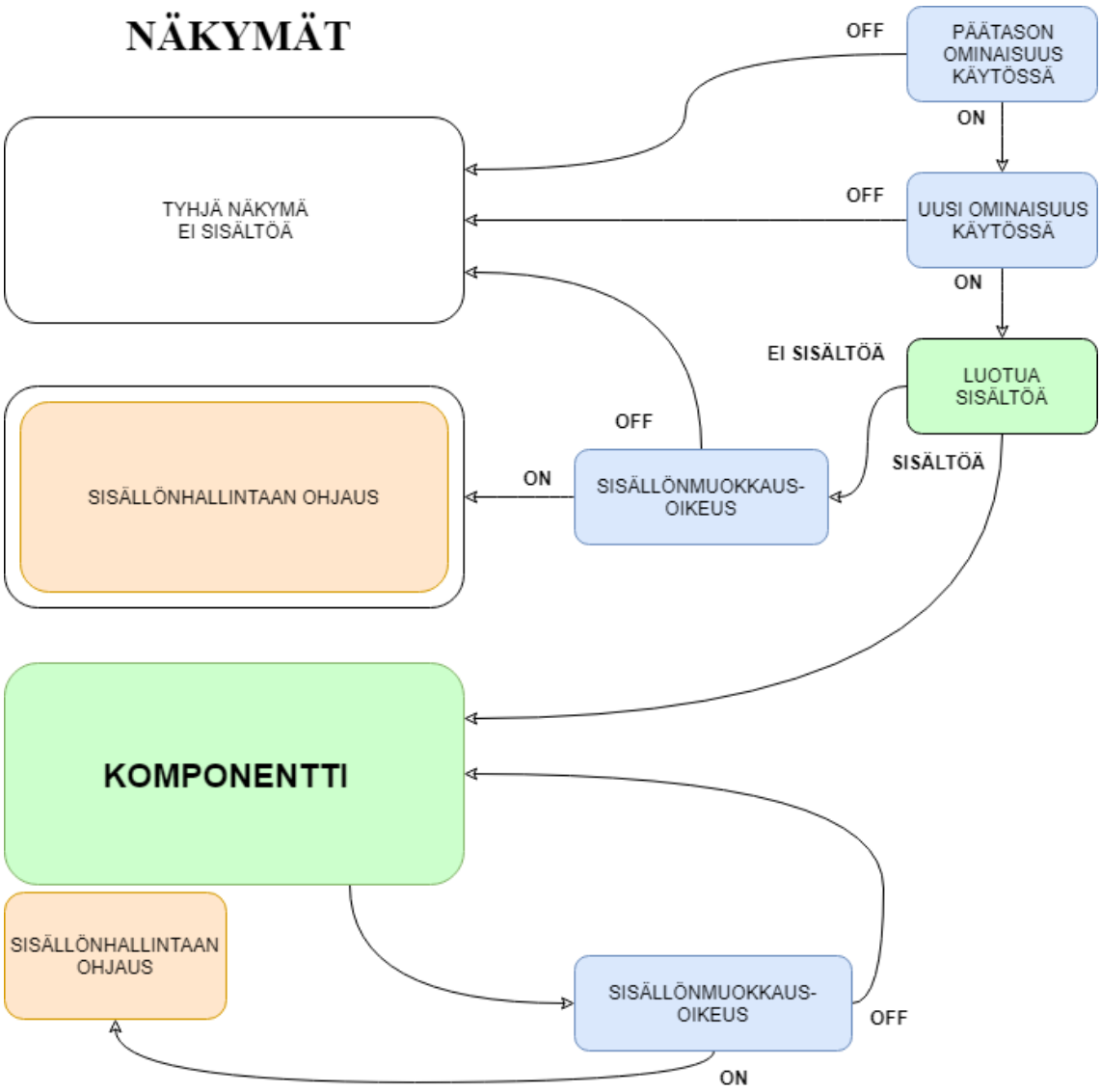
4.3.2 Web-komponentin tyylien rakentaminen

Web-komponentin suunnittelun ja toimivan prototyypin jälkeen tyyliteltiin komponentti designien mukaiseksi. Komponentin tyylittely tehtiin LESS CSS -esiprosessorin avulla. Se mahdollistaa eri lisätoimintojen ja syntaksin käytön perus CSS-perusominaisuuksien lisäksi.

4.3.3 Web-komponentin lisääminen päänäkymään

Komponentin lisääminen päänäkymään ei itsessään tuottanut ongelmia, mutta sen näyttäminen on riippuvainen käyttöoikeuksista sekä muiden ominaisuuksien tilasta. Komponenttia näytettäessä piti huomioida eri tasoisten ehtojen tärkeysjärjestys.

Kuviossa 6 näytetään, miten komponentille rakennettu osio muuttuu sen perusteella, mitkä ominaisuudet on kytketty päälle. Komponentin osiolle piti rakentaa kolme eri näkymää, joista kolmanteen tuotiin karusellikomponentin lisäksi pääsy sisällönhallintaan, jos käyttäjällä oli vaaditut sisällönhallintaoikeudet.



KUVIO 6. Komponentin näkymiin vaikuttavat tekijät

4.4 Sisällönhallintatyökalu

Komponentin rinnalle suunniteltiin sisällönhallintatyökalu, jossa käyttäjä pystyi määrittelemään komponenttiin sisältöä. Sitä rakennettaessa oli tiedossa, että sillä tultaisiin tässä versiossa luomaan sisältöä vain yhdelle komponentille. Tarkoituksena oli kuitenkin rakentaa työkalu siten, että se olisi helposti laajennettavissa jatkossa.

Rakennettaessa sisällönhallintatyökalua piti huomioida se, että työkalun käyttäjä voi määritellä oikeuksiensa mukaisesti sisältöä isommalle joukolle. Tämän myötä oli mahdollista,

että käyttäjä voi määritellä muille käyttäjille sellaista sisältöä, johon hänellä itsellään ei ole normaalisti käyttöoikeuksia.

4.4.1 Sisällönhallintatyökalun rakenne

Sisällönhallintatyökaluun piti rakentaa kolme eri paneelia, joiden näkyvyys ja aktivointi ovat riippuvaisia toisten paneeleiden tilasta. Paneelit rakennettiin siten, että jokaisella paneelilla oli vain tietyt toiminnot. Paneeleille tehtiin toteutus, jossa toiminnallisuus jaettiin kolmeen pääosaan.

Ensimmäisen paneelin tehtävänä oli valita kohde, jolle sisältöä tultaisiin määrittelemään. Toiseen paneeliin piti rakentaa näkymä, josta nähdään kohteelle luodut sisällöt. Toisessa paneelissa oli tämän lisäksi toiminnot, joiden avulla aktivoidaan kolmas paneeli. Kolmannessa paneelissa pystytään luomaan, muokkaamaan tai poistamaan jo luotua sisältöä.

Kolmen osion ympärille rakentui päätaso, joka sisälsi päätoiminnot. Näiden toimintojen tarkoituksena oli tallentaa sisältöön tehdyt muutokset sekä julkaista sisältö sen ollessa valmista. Päätason toimintojen tilan piti muuttua dynaamisesti sisällönmuokkausprosessin eri vaiheissa. Tästä johtuen tilaa hallitsevasta logiikasta tuli hyvin monimutkainen, minkä seurauksena päädyttiin testaamaan MobX-tilanhallintakirjastoa sisällönhallintatyökalun tilan hallitsemiseen.

4.4.2 MobX-kirjaston testaaminen ja käyttöönotto projektissa

Projektissa ei aikaisemmin ollut käytetty MobX-kirjastoa, joten sen toimivuutta haluttiin testata sisällönhallintatyökalua rakennettaessa. Projektissa aikaisemmin luodut tilanhallintaratkaisut oli toteutettu AngularJS:n tarjoamilla ominaisuuksilla. AngularJS:llä tehdyt tilanhallinnan ratkaisut ovat suorituskykyä vaativia, joten ne voivat hidastaa sovelluksen toimintaa.

MobX:n avulla voidaan poistaa suurin osa AngularJS:llä rakennetusta tilanhallinnasta aiheutuvista suorituskykyongelmista. MobX:ää käytetään erillisenä AngularJS:stä, joten

sen sisällä olevat muutokset eivät vaadi AngularJS:ää suorittamaan mitään toimintoja tilan päivittämiseen. Tämä vähentää huomattavasti tilanhallinnan vaatimaa suorituskykyä, sillä MobX päivittää näkymää vain silloin, kun sen säilössä olevat arvot muuttuvat.

Käyttöönotto

MobX:n käyttöönotto projektissa tuotti jonkin verran ongelmia, koska projektissa käytettiin AngularJS:sää ja Typescriptiä. MobX:n sen hetkisessä versiossa ei ollut tukea suoraan AngularJS-projektille eikä tyypityksiä Typescriptille. Tämän ongelman ratkaisemiseksi piti rakentaa väliaikaiset tyypitykset, joiden avulla voitaisiin käyttää MobX:n tarjoamia ominaisuuksia missä tahansa osassa projektia.

Lopputulos

MobX:n käyttöönotto onnistui ongelmien jälkeen lopuksi hyvin. Sen avulla rakennettiin selkeä tilanhallintaratkaisu, jota voidaan laajentaa jatkossa tarpeen vaatiessa. MobX todettiin toimivaksi ratkaisuksi tilanhallintaan, ja sitä on tarkoitus käyttää projektissa jatkossa uusia ominaisuuksia rakennettaessa.

5 POHDINTA

5.1 Opinnäytetyön tulokset ja tuotosten tarkastelu

Opinnäytetyössä tutkittiin sisällönhallinnan perusteita. Näiden perusteiden pohjalta haluttiin rakentaa sovelluksen yhteyteen sisällönhallintatyökalu, jolla voitaisiin luoda sisältöä opinnäytetyössä rakennettuun web-komponenttiin.

Sisällönhallintaa pystyttiin soveltamaan pienimuotoisena pienen kokonaisuuden rakentamiseksi. Sisällönhallinnasta opitut asiat auttoivat lopulta työkalun rakentamisessa. Työkalusta jouduttiin kuitenkin karsimaan joitakin ominaisuuksia pois, koska projektin aikataulua jouduttiin kiristämään. Lopputuloksena oli ratkaisu, jossa oli aluksi määritellyt ydinominaisuudet, mutta joitakin sisällönluomista nopeuttavia ominaisuuksia jäi puuttumaan. Sisällönhallintaa rakennettaessa lisätty MobX-tilanhallintakirjasto toimi erittäin hyvin, ja sen käyttö laajenee jatkossa projektin uusien ominaisuuksien yhteydessä.

Web-komponentin rakentaminen onnistui hyvin, ja sille saatiin rakennettua toimiva pohja hyvin varhaisessa vaiheessa projektia. Web-komponentissa käytetty Slick-kirjasto nopeutti komponentin rakentamista huomattavasti. Slick otettiin osaksi projektia, ja sitä voidaan jatkossa käyttää muita ominaisuuksia rakennettaessa.

5.2 Teknologioiden hallitseminen

Suurin osa opinnäytetyössä käytetyistä teknologioista oli minulle tuttuja jo ennestään. Angularia käyttäessäni koin oppineeni paljon uutta sen toiminnasta ja pystyn soveltamaan opittua uusien ominaisuuksien rakentamiseen. Projektin ohessa tuli myös selväksi se, miksi Angularista rakennettiin kokonaan uusi kirjasto, ja mitkä ovat AngularJS:n suurimmat ongelmat. Suurimpana ongelmana on AngularJS:n suorituskyky verrattuna Angulariin

MobX:n käyttöönotto ja sen opetteleminen oli yksi isoimmista haasteista. MobX:llä kuitenkin on laaja dokumentaatio, jonka avulla pystyin sekä toteuttamaan tilanhallinnan että

testaamaan sen toimivuutta. MobX:n käyttäminen oli järkevää, ja sen avulla pystyy jatkossa rakentamaan selkeitä sekä toimivia tilanhallintaratkaisuja.

5.3 Ajatuksia web-kehityksen käsitteistä ja sen tulevaisuudesta

Sisällönhallinta ja tilanhallinta

Sisällönhallinta ja tilanhallinta olivat molemmat uusia aihealueita, joita pääsin tutkimaan tätä opinnäytetyötä tehdessä. Sisällönhallintaa tutkiessa löysin paljon tietoa sen toiminnallisuudesta ja tarkoituksesta. Sisällönhallinta käsitteenä on hankala, sillä mielipiteet sen toteutustavasta ja tarpeellisuudesta vaihtelevat suuresti. Web-kehityksen kiihtyessä sisällönhallinnan käsite laajenee uusien toteutustapojen ja datatyyppien määrän kasvaessa.

Tilanhallinta oli uusi käsite, vaikkakin sen periaatteita on tullut sivuttua useasti web-kehityksen yhteydessä. Erityisesti web-komponentteja kehitettäessä on erittäin tärkeää huomioida, kuinka komponentin täytyy muuttua sen tilan muuttuessa. Opinnäytetyössä rakennetun sisällönhallintatyökalun kokonaisuudelle piti luoda tapa, jolla voidaan hallita tilaa keskitetysti. Sisällönhallintatyökalua rakennettaessa tilanhallinnan perusteet ja tärkeys tulivat hyvin selväksi.

Suosittelen käyttämään tilanhallintaa erityisesti, kun ollaan rakentamassa käyttöliittymiä. Jos käyttöliittymässä on useampi asia, jotka voivat muokata sen ulkonäköä, niin tällöin kannattaa rakentaa avuksi keskitetty tilanhallintaratkaisu. Tilanhallinnan avulla voidaan virhetilanteissa helposti osoittaa, mikä virheen aiheutti. Tulevaisuudessa uskon, että tilanhallintaan panostetaan web-sovelluksien monimutkaistuessa ja kasvaessa.

Teknologioiden tulevaisuus

Opinnäytetyössä käytettyjen teknologioiden tulevaisuus näyttää pääasiassa hyvältä. Suurin osa teknologioista on laajassa käytössä eri kokoisissa ja tyyppisissä projekteissa. Opinnäytetyössä käytettyjä teknologioita päivitetään lähes päivittäin, jotta ne pysyisivät kiihtyvän web-kehityksen vauhdissa mukana. Uusia teknologioita julkaistaan kaiken aikaa, joten todennäköisesti kahden vuoden päästä suurin osa teknologioista on jo korvattu uusilla. Viime vuosien web-kehitystrendejä analysoimalla voidaan todeta, että käytettävät teknologiat vaihtuvat kovaa tahtia, mutta tarkasti ei voida ennustaa, kuinka kauan teknologia pysyy pinnalla.

Opinnäytetyössä käytetty AngularJS on jäämässä vahvasti pois käytöstä, sillä sen kehitys loppuu kokonaan 2018 heinäkuussa, vaikkakin korjauksia tehdään vielä vuoteen 2021 heinäkuuhun asti. AngularJS:n käyttöä uusissa projekteissa kannattaa välttää ja pyrkiä korvaamaan se vanhemmissakin projekteissa, jos niitä kehitetään aktiivisesti edelleen. Uusi Angular-kirjasto tarjoaa paljon paremmat mahdollisuudet modernien web-sovellusten rakentamiseen ja se vähentää AngularJS:n aiheuttamia suorituskykyongelmia huomattavasti. Uuden Angularin tapa kehittää on paljon modernimpi verrattuna AngularJS:n, ja se pyrkii lisäämään uusia ominaisuuksia tämän hetkisten web-trendien mukaisesti.

5.4 Opinnäytetyön yhteenveto

Opinnäytetyölle asetetut tavoitteet saavutettiin pääosin, ja niihin liittyviin kysymyksiin saatiin vastaukset. Suurimpana opinnäytetyön onnistumisena pidin sitä, että toteutetut ominaisuudet saatiin rakennettua vaatimuksien mukaisiksi ja ne saatiin valmiiksi aikataulussa. Positiivisena pidän sitä, että opinnäytetyötä tehtäessä pystyttiin laajentamaan projektin mahdollisuuksia lisäämällä siihen uusia teknologioita. Uusia teknologioita voidaan käyttää jatkossa uusien ominaisuuksien rakentamiseen.

LÄHTEET

Angularjs documentation. N.D.a, Luettu 26.3.2018
https://docs.angularjs.org/api/ng/directive/ngApp

Angularjs Documentation. N.D.b Luettu 21.3.2018
https://docs.angularjs.org/api/ng/directive/ngModel

Barker, D. 2016. Web Content Management. Safari books online.. Luettu 2.16.2018.

DAM Basics. Widen. N.D
https://digitalassetmanagement.com/dam-basics/

Gudelli, A. 2017. AngularJs Wiki, History of AngularJS. Luettu 11.3.2018.
http://www.angularjswiki.com/angularjs/history-of-angularjs/

Koppatz, R. 2017. Mikä on DAM ja miksi sitä tarvitaan? Julkaistu 15.9.2017. Luettu 15.4.2018.
http://blogi.communicationpro.com/artikkelit/mika-on-dam-ja-miksi-sita-tarvitaan

Mobx reference guide. N.D. Luettu 14.3.2018
https://mobx.js.org/refguide/computed-decorator.html

Tarasiewicz, P. & Böhm, R. 2014. AngularJS. Ebook central. Luettu 10.3.2018.

Wikipedia 2018, Web content management system. Luettu 15.4.2018.
https://en.wikipedia.org/wiki/Web_content_management_system