jamk.fi

Mobiilikäyttöön optimoidun web-sovelluksen kehittäminen Angular-sovelluskehyksellä

Aleksi Karhunen

Opinnäytetyö
Marraskuu 2018
Luonnontieteiden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Karhunen, Aleksi	Opinnäytetyö, AMK	Marraskuu 2018
	Sivumäärä 40	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Mobiilikäyttöön optimoidun web-sovelluksen kehittäminen Angular-sovelluskehyksellä

Tutkinto-ohjelma

Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)

Tommi Tuikka

Toimeksiantaja(t)

Protacon Solutions Oy

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutkia, kuinka kehitetään mobiilikäyttöön optimoitu web-sovellus käyttäen Angular-sovelluskehystä, ja kehittää sellainen toimeksiantajan asiakasyritykselle heidän vaatimuksien mukaisesti. Sovelluksen käyttöliittymä toteutettiin käyttäen Angular Material -ulkoasukomponenttikirjastoa.

Kehitystutkimus aloitettiin etsimällä web-sovelluksen kehittämiseen tarvittavaa teoriatietoa eri verkkolähteistä. Angular-sovelluskehys on jatkuvan kehityksen alla, joten tietoa kerättiin vain tuoreista lähteistä. Löydettyjä tietoja ja Angularin omaa dokumentaatiota käytettiin hyödyksi sovelluksen kehittämisessä.

Tutkimuksen aikana löydettiin web-sovelluksen kehittämiseen tarvittavat tiedot. Sovelluksen toteuttaminen onnistui ja vaatimusten mukainen mobiilikäyttöön optimoitu web-sovellus julkaistiin asiakasyrityksen käyttöön marraskuussa 2018.

Tutkimuksessa todettiin, että mobiilikäyttöön optimoidun käyttöliittymän tulee olla yhdenmukainen, minimaalinen ja yksittäisen näkymän tulee palvella vain yhtä päätarkoitusta. Lisäksi käyttöliittymän kannattaa muistuttaa natiivia mobiilisovellusta, jotta se on käyttäjälle tuttu jo ennestään. Kun käyttöliittymä toteutetaan noudattaen näitä ohjeita, se on käyttäjäystävällinen ja helposti opittava.

Angular-sovelluskehyksen parhaat kehityskäytänteet ovat litteä hakemisto- ja tiedostorakenne, yhdenmukaiset nimeämiskäytänteet sekä yhden sääntö. Yhden säännön mukaan yksittäinen toiminto palvelee vain yhtä tarkoitusta ja yksittäinen tiedosto sisältää vain yhden toiminnon. Angularin parhaat käytänteet eivät välitä sovellusalustasta, mutta niitä kannattaa aina noudattaa parhaan mahdollisen lopputuloksen takaamiseksi.

Avainsanat (asiasanat)

Angular, mobiilioptimointi, web-sovelluskehitys

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Karhunen, Aleksi	Bachelor's thesis	November 2018
	Number of pages 40	Language of publication: Finnish
		Permission for web publication: x

Title of publication

Developing web application optimized for mobile use with Angular framework

Degree programme

Business Information Technology

Supervisor(s)

Tuikka, Tommi

Assigned by

Protacon Solutions Oy

Description

The purpose of the thesis was to research how to develop a web application optimized for mobile use with Angular framework and develop it to the assigner's client company according to their requirements. The application's user interface was implemented using the Angular Material library.

The development research started by searching from different online sources for the theoretical information needed to develop the web application. As the Angular framework is under constant development, information was only collected from recent sources. The found information and Angular's own documentation were used in the development of the application.

The necessary information to develop the web application was found during the research. The application was developed successfully, and a web application optimized for mobile use was published for the client company's use in November 2018.

The research discovered that a user interface optimized for mobile use must be uniform, minimal and a single view must serve only one main purpose. Additionally, the user interface should resemble a native mobile application, so it is already familiar to the user. When the user interface is implemented following these guidelines, it is user friendly and easy to learn.

The best development practices of the Angular framework are flat directory and file structure, uniform naming practices and the rule of one. According to the rule of one, a single feature serves only one purpose and a single file contains only one feature. The best practices of Angular do not pay attention to the software platform; however, they should be always followed to guarantee the best possible result.

Keywords (subjects)

Angular, mobile optimization, web application development

Miscellanous

Sisältö

Kuviot

Käsitteet

Angular: Googlen kehittämä suosittu JavaScript-sovelluskehys, joka on kirjoitettu TypeScript-ohjelmointikielellä.

Angular Material: Googlen materiaalisuunnitteluun perustuva ulkoasukomponenttikirjasto Angular-sovelluskehystä varten.

Back-end: Sovelluksen osa, joka vastaa tietokannan ja ulkoisten rajapintojen kanssa vuorovaikuttamisesta.

CSS (Cascading Style Sheets): Kieli, jonka avulla verkkosivulle asetetaan tyyliohjeita.

Front-end: Sovelluksen osa, joka määrittää käyttöliittymän ja sen toiminnallisuudet.

HTML (Hypertext Markup Language): Kieli, jonka avulla määritetään verkkosivun rakenne.

JavaScript: Ohjelmointikieli, joka mahdollistaa toiminnallisuuden lisäämisen web-sovellukseen.

JSON (JavaScript Object Notation): Yksinkertainen ja helposti luettava tietomuoto tiedonsiirtoa varten.

JWT (JSON Web Token): JSON-pohjainen menetelmä tiedonsiirtoon. Käytetään usein käyttäjän autentikoimiseen.

Käyttäjäkokemus (engl. user experience): Kokemus, jonka käyttäjä saa sovelluksen kanssa vuorovaikuttamisesta.

Käyttöliittymä (engl. user interface): Sovelluksen käyttäjälle näkyvä osa, jonka kanssa käyttäjä voi vuorovaikuttaa.

Liittymä (engl. interface): Määritelmä, jossa asetettuja ominaisuuksia ja niiden tyyppejä liittymään kuuluvan olion täytyy noudattaa.

Luokka (engl. class): Määritelmä, joka tarjoaa olio-ohjelmoinnista tutut toiminallisuudet, kuten olioiden luomisen ja perinnöllisyyden.

Metodi (engl. method): Luokkaan kuuluva funktio

Rajapinta (engl. application programming interface): Määritelmä, jonka avulla eri sovellukset voivat kommunikoida toistensa kanssa.

Sovelluskehys (engl. framework): Ohjelmointikielen päälle rakennettu kirjasto, joka tarjoaa työkaluja helpompaa ja tehokkaampaa kehittämistä varten.

TypeScript: JavaScript-ohjelmointikielen ylijoukko, joka tarjoaa lisätoiminallisuuksia. TypeScript käännetään JavaScriptiksi ennen verkkoselaimessa suorittamista.

Web-sovellus (engl. web application): Verkkoselaimessa toimiva sovellus.

1 Johdanto

Digitalisaatio tarkoittaa vanhojen toimintamallien korvaamista tai parantamista digitaalisilla ratkaisuilla. Digitalisaatio säästää oikein toteutettuna yrityksille merkittävästi aikaa ja rahaa. (Bouza 2018.) Digitalisaatio muuttaa arkea automatisoimalla ja nopeuttamalla vanhoja prosesseja. Digitalisaatiota on esimerkiksi tiedon helppo saatavuus ja palveluiden siirtyminen verkkoon sekä mobiiliin. (Digitalisaatio n.d.)

Tässä opinnäytetyössä digitalisaatio näkyy mobiilikäyttöön optimoidun web-sovelluksen kehittämisenä toimeksiantajan asiakasyritykselle. Sovelluksen on tarkoitus korvata asiakasyrityksen aiempi toimintamalli, ja se toteutetaan käyttäen suosittuja Angular- sekä Angular Material -teknologioita. Sovelluksen kehittämisen tueksi opinnäytetyössä tutkitaan, millainen mobiilikäyttöön optimoidun käyttöliittymän tulee olla, ja mitkä ovat Angular-sovelluskehyksen parhaat kehityskäytänteet.

Google julkaisi vuonna 2016 Angular-sovelluskehyksen, joka korvaa aiemman AngularJS-sovelluskehyksen. Angular on kehitetty TypeScript-ohjelmointikielellä, joka sisältää olio-ohjelmoinnista tutut ominaisuudet, kuten luokat ja perinnöllisyyden. Angularin reitittimen avulla Angular-sovelluksesta voi tehdä yksisivuisen sovelluksen, jolloin sovelluksen näkymien välillä voi liikkua päivittämättä verkkosivua. Angular tarjoaa myös Angular CLI -komentorivityökalun, jonka avulla osa kehitystyöstä voidaan automatisoida, joten sen käyttö säästää merkittävästi aikaa. Muun muassa nämä ominaisuudet tekevät Angularista yhden tämän hetken suosituimmista ja moderneimmista JavaScript-sovelluskehyksistä. Angularia kehitetään aktiivisesti ja sen ympärillä on laaja kehittäjäyhteisö. (Bodrov-Krukowski 2018.)

Angular Material -ulkoasukomponenttikirjasto pohjautuu Googlen materiaalisuunnitteluun ja se tarjoaa saumattoman yhteensopivuuden Angular-sovelluskehyksen kanssa. Material tekee modernin käyttöliittymän toteuttamisesta helppoa ja nopeaa. Material sisältää myös teemaominaisuuden, jonka avulla Angular-sovellukseen on helppo integroida haluttu värimaailma. (Angular Material n.d.)

2 Tutkimusasetelma

Tässä luvussa käydään läpi opinnäytetyön tavoite, tutkimusongelma, tutkimuskysymykset ja valittu tutkimusmenetelmä. Luvussa kerrotaan lisäksi kehitettävän web-sovelluksen vaatimuksista, kehitystyön lähtökohdista ja opinnäytetyön rajauksesta.

Toimeksiantaja

Opinnäytetyön toimeksiantaja on jyväskyläläinen ohjelmistoyritys Protacon Solutions Oy. Toimeksiantajayritys toimii nykyään useilla eri paikkakunnilla ja se työllistää yli 100 henkilöä. Protacon Solutions Oy on opinnäytetyön tekijän työnantaja ja opinnäytetyöhön liittyvä kehitystyö kuuluu osaksi opinnäytetyön tekijän työtehtäviä.

Toimeksiantajan asiakasyrityksen liiketoiminnan turvaamiseksi tässä opinnäytetyössä ei mainita asiakasyritystä nimeltä. Kehitystyönä toteutettavan web-sovelluksen käyttötarkoitusta ei paljasteta siinä määrin, että sen pystyy yhdistämään asiakasyritykseen.

2.1 Tutkimusongelma ja -kysymykset

Tutkimuksen perusta on tarkasti määritellyssä asiaongelmassa ja siitä johdetussa tutkimusongelmassa. Tutkimusongelmasta johdetaan tutkimuskysymykset. Tutkimuksen kannalta on tärkeää, että tutkija sekä toimeksiantaja käsittävät tutkimuksen tavoitteen ja tutkimusongelman samalla tavalla. (Vilkka 2015, 59−60.)

Tämän opinnäytetyön tavoitteena on kehittää mobiilikäyttöön optimoitu web-sovellus toimeksiantajan asiakasyritykselle. Sovelluksen on tarkoitus korvata asiakasyrityksen aikaisempi toimintamalli ja kehittää heidän liiketoimintaa. Tavoitteen kautta tutkimusongelmaksi muodostuu asiakasyrityksen vaatimuksien mukaisen mobiilikäyttöön optimoidun web-sovelluksen kehittäminen. Tutkimusongelman pohjalta seuraavat kysymykset muodostuvat tutkimuskysymyksiksi:

- Kuinka toteutetaan mobiilikäyttöön optimoitu Angular-web-sovellus?
- Miten web-sovelluksen käyttöliittymä optimoidaan mobiilikäyttöön?
- Mitkä ovat Angular-sovelluskehyksen parhaat kehityskäytänteet?

Ensimmäinen tutkimuskysymys toimii tämän tutkimuksen pääkysymyksenä ja kaksi seuraavaa tutkimuskysymystä toimivat alakysymyksinä. Alakysymykset tukevat pääkysymystä ja niihin vastamaalla voidaan johtaa vastaus tutkimuksen pääkysymykseen. Vilkan (2015, 60) mukaan alakysymyksiä kutsutaan myös teoreettisiksi tutkimuskysymyksiksi ja ne ovat kysymykset, joihin tutkimuksessa halutaan vastata.

2.2 Tutkimusmenetelmät

Tämän opinnäytetyön tutkimusmenetelmänä on kehittämistutkimus. Kehittämistutkimus valikoitui tutkimusmenetelmäksi, sillä kirjallisen osuuden lisäksi opinnäytetyön lopputuloksena on mobiilikäyttöön optimoitu web-sovellus. Kananen (2012, 42) tiivistää kehittämistutkimuksen seuraavasti: ”Kehittämistutkimus ei tuota pelkästään tekstejä vaan käytännössä toimivia ratkaisuja.”

Kehittämistutkimuksen lopputuloksena on aina muutos entiseen. Muutos tarkoittaa olemassa olevan ongelman poistamista tai jonkin asian kehittämistä paremmaksi. Kehittämistutkimuksessa ei pyritä yleistämään, vaan muutos on aina tapauskohtainen. Kaikki kehittämisen kautta aiheutuneet muutokset eivät kuitenkaan ole kehittämistutkimusta, vaan kehitetyn muutoksen lisäksi vaaditaan tutkimuksellinen ote ja tutkimusosio. (Kananen 2012, 43−44; Kananen 2015, 40.) Tässä opinnäytetyössä kehittämistutkimuksesta seuraava muutos on asiakasyrityksen ongelman poistaminen digitalisaation keinoin, web-sovelluksen muodossa.

2.3 Web-sovelluksen vaatimukset

Tässä kappaleessa käydään läpi kehittämistutkimuksen kehitystyönä toteutettavan web-sovelluksen tärkeimmät vaatimukset. Asiakasyrityksen asettamat päävaatimukset sovellukselle ovat:

- Mobiilikäyttöön optimoitu
- Helppokäyttöinen
- Hyvä suorituskyky.

Mobiilikäyttöön optimointi on edellä mainituista vaatimuksista tärkein, sillä asiakasyrityksen mukaan sovellusta tullaan käyttämään vain mobiililaitteilla eli puhelimilla ja

tableteilla. Asiakasyritys haluaa sovellukseen yksinkertaisen brändiinsä sopivan käyttöliittymän, jossa kaikki olennainen on selkeästi esillä. Sovelluksen täytyy toimia sulavasti hyvän käyttäjäkokemuksen takaamiseksi. Lisäksi sovelluksen hakutoiminnon tulee toimia myös heikon verkkoyhteyden varassa.

Toimeksiantaja ja asiakasyritys ovat yhdessä päätyneet web-sovellustoteutukseen. Sovelluksesta ei tehdä mobiilisovellusta, vaikka sen käyttö tapahtuu ainoastaan mobiililaitteilla. Web-sovellustoteutus pystyy täyttämään kaikki sovellukselle asetetut vaatimukset. Asiakasyrityksen mukaan sovellusta tullaan käyttämään ainoastaan Android-pohjaisilla laitteilla ja Google Chrome -verkkoselaimella, joten sovelluksen ei tarvitse tukea muita käyttöjärjestelmiä tai verkkoselaimia.

2.4 Kehitystyön lähtökohdat

Kehitettävä web-sovellus on pieni toteutus, joten se sopii hyvin opinnäytetyön kehitystyöosuudeksi. Sovelluksen pienen koon myötä se voidaan kehittää yhden kehittäjän voimin järkevässä ajassa. Sovelluksen koko toteutusta voidaan arvioida vain tiettyjen osuuksien sijasta, sillä opinnäytetyön tekijä on yksin vastannut sovelluksen teknisestä toteutuksesta. Toimeksiantaja on kuitenkin muodostanut projektitiimin opinnäytetyön tekijän tueksi. Projektitiimi koostuu opinnäytetyön tekijän lisäksi projektipäälliköstä, seniorikehittäjästä ja graafisesta koordinaattorista. Projektitiimi vastaa yhdessä sovelluksen suunnittelusta ja opinnäytetyön tekijälle on kehitystyöhön liittyviin ongelmiin monipuolista apua tarjolla. Graafinen koordinaattori on luonut sovellusta varten käyttöliittymädemon, mutta opinnäytetyön tekijän tehtäväksi jää lopullisen käyttäjäkokemuksen luominen.

Toimeksiantaja on valinnut web-sovelluksen kehittämiseen käytettävät teknologiat, jotka ovat:

- Angular 6
- Angular Material 6
- Symfony 4.

Sovelluksen front-end kehitetään käyttäen Angular-sovelluskehystä ja Angular Material -ulkoasukomponenttikirjastoa. Back-end kehitetään PHP:hen pohjautuvalla Symfony-sovelluskehyksellä ja tietokantana on käytössä SQL-tietokanta MariaDB.

Web-sovellus tulee käyttöön asiakasyrityksen työntekijöille ja sen käyttäminen vaatii ulkoisen rajapinnan kautta toimivan kirjautumisen. Käyttäjän autentikointi saadaan lähes valmiina toisesta toimeksiantajan kehittämästä Angular-sovelluksesta. Opinnäytetyön tekijän tehtäväksi jää kirjautumisnäkymän toteuttaminen ja autentikoinnin muokkaaminen kehitettävään sovellukseen sopivaksi.

Web-sovelluksen käyttötarkoitus on tietojen hakeminen kahdesta ulkoisesta rajapinnasta ja haettujen tietojen esittäminen käyttäjäystävällisessä muodossa. Tietoja haetaan vain yhdestä rajapinnasta kerrallaan ja haussa käytettävä rajapinta riippuu käyttäjän antamista rajausehdoista. Sovellus sisältää pikahaun sekä tarkennetun haun, joka mahdollistaa useiden hakuehtojen antamisen kerralla. Käyttäjien tekemät haut tallennetaan sovelluksen MariaDB-tietokantaan. Toimeksiantajan kehittämästä Symfony 4 -pohjasta löytyy jo valmiiksi hakujen kirjaaminen tietokantaan, joten opinnäytetyön tekijän ei tarvitse tehdä sitä. Sovelluksen ei tarvitse esittää tietokantaan tallennettuja hakuja, vaan asiakasyritys saa tarvittaessa tietokantarekisterin toimeksiantajalta.

Tämä opinnäytetyö koskee vain kehitettävän web-sovelluksen ensimmäistä julkaistavaa versiota, mutta asiakkaan mukaan on todennäköistä, että sovellusta jatkokehitetään tulevaisuudessa. Sovelluksen ensimmäisen version kehityksessä otetaan huomioon mahdolliset jatkokehitystarpeet, jotta jatkokehittäminen sujuu tulevaisuudessa vaivattomasti. Sovelluksen rakenteen tulee mahdollistaa helppo jatkokehitys ja ohjelmakoodin täytyy olla hyvin dokumentoitua, koska jatkokehittäjä voi olla joku muu kuin opinnäytetyön tekijä.

Kehitystyö alkoi heinäkuussa 2018 ja ensimmäinen versio web-sovelluksesta on tarkoitus julkaista asiakkaan käyttöön vuoden 2018 aikana. Opinnäytetyön tekijälle on varattu reilu 400 tuntia aikaa sovelluksen kehittämiseen.

2.5 Opinnäytetyön rajaus

Tämän opinnäytetyön teoria- ja kehitystyöosioista on rajattu pois web-sovelluksen back-end- ja tietokantaosuudet. Rajauksen tarkoitus on pitää opinnäytetyö järkevän kokoisena. Sovelluksen back-end ja tietokanta ovat lisäksi huomattavasti front-endiä pienemmässä osassa koko toteutuksen näkökulmasta.

Opinnäytetyön teoria- ja kehitystyöosioissa keskitytään web-sovelluksen front-endiin eli käyttöliittymään sekä Angular- ja Angular Material -teknologioihin. Teoriaosiossa käydään läpi front-endin kehittämisen kannalta olennaiset asiat, kuten mobiilikäyttöön optimoidun käyttöliittymän luominen ja Angular-sovelluskehyksen parhaat kehityskäytänteet. Kehitystyöosiossa käydään läpi, kuinka sovellus toteutettiin, ja miten teoriaosion tietoja hyödynnettiin toteutuksessa.

3 Mobiilikäyttöön optimoitu web-sovellus

Tässä luvussa käydään läpi, millainen mobiilioptimoidun käyttöliittymän tulee olla, jotta se on käyttäjäystävällinen. Luvussa käsitellään mobiilikäyttöliittymän eri osa-alueita ja esitellään mobile first -kehitysmalli.

3.1 Käyttöliittymä

Son (2017) mukaan puhelimet ovat näytön kooltaan alle seitsemän tuumaa ja tabletit ovat näytön kooltaan yli seitsemän tuumaa. Puhelinkäyttöliittymän tulee sisältää vain käytön kannalta olennaiset toiminnot. Tablettikäyttöliittymä voi näyttää työpöytäversiolta, mutta sen täytyy teknisesti toimia kuten mobiiliversion. (So 2017.)

Babichin (2018a) mukaan hyvä muistisääntö suunnitellessa käyttöliittymää mobiililaitteelle on sisällön minimaalisuus. Näkymien tärkeimmät tiedot ja painikkeet tulee näyttää käyttäjälle heti. Vähemmän olennaiset tiedot ja painikkeet piilotetaan käyttäjän vuorovaikutuksen taakse esimerkiksi valikkoon, jotta näytöllä ei ole liikaa sisältöä kerralla. Minimaalisuutta noudattamalla käyttöliittymästä tulee selkeä ja käyttäjä ei

hämmenny liiasta samanaikaisesta tiedosta. Laajoissa, paljon vuorovaikutusta vaativissa toiminnoissa, kuten verkkokaupasta tilaamisessa prosessi tulee jakaa useisiin eri vaiheisiin. Käyttäjän on helpompi seurata prosessin kulkua, kun tietoa on näytöllä vain vähän kerrallaan. (Babich 2018a.)

Käyttöliittymän tärkeimpien elementtien tulee erottua edukseen. Hyviä keinoja elementtien korostamiseen ovat esimerkiksi fontin paino, koko ja väri. Hyvän käyttäjäkokemuksen takaamiseksi käyttöliittymän tulee olla yhdenmukainen. Elementtien tulee näyttää samalta ja samannäköisten painikkeiden tulee tehdä samankaltainen asia kaikkialla sovelluksessa. (Babich 2018a.)

Toiminnot

Minimaalisen käyttöliittymän periaate pätee myös toimintoihin. Käyttäjän tulee pystyä suorittamaan halutut toiminnot mahdollisimman vähällä vuorovaikutuksella. Esimerkiksi lomakkeiden tulee olla mahdollisimman lyhyitä ja vain käytön kannalta olennaiset asiat tulee kysyä, jotta käyttäjä välttyy turhalta kirjoittamiselta. (Babich 2018a.)

Babichin (2018a) mukaan jokaisen näkymän tulisi sisältää vain yksi sille asetettu päätoiminto, jotta sovelluksen käyttö on selkeää. So (2017) kertoo, että tärkeimmät toiminnot tulee esittää painikkeina tekstilinkkien sijaan. Painikkeen ulkoasu osoittaa käyttäjälle heti, että sen on tarkoitus tehdä jotain. Pelkkään tekstiin sidottu toiminto on huomattavasti vaikeampi havaita. Painikkeisiin on myös helppo lisätä taustaväri, jolla voi korostaa esimerkiksi sovelluksen teemaa tai yrityksen brändiä. (So 2017.) Painikkeiden tulee helposti erotettavan ulkoasun lisäksi ilmaista selkeästi niiden sisältämä toiminto, koska mobiililaitteilla toimintoihin liittyvien vihjeiden antaminen on hankalampaa, sillä työpöytäkäytöstä tuttu kursorin leijuntaefekti ei ole käytettävissä. (Babich 2018a.)

Toimintojen tulee antaa käyttäjälle välittömästi palautetta. Jos toimintoa napauttaessa ei tapahdu mitään käyttäjä voi luulla, että sovellus ei rekisteröinyt napautusta tai, että se pysähtyi. Esimerkiksi sovelluksen sisältämän hakutoiminnon tulee sisältää jokin tietojen hakemista kuvaava animaatio, jotta käyttäjä tietää haun olevan käynnissä. (Babich 2018b.)

Tyhjien näkymien tulee ilmaista selkeästi, miksi sisältö puuttuu. Tyhjään näkymään on suositeltavaa lisätä painike, josta pääsee sisältöä tuottavaan toimintoon. (So 2017.) Myös sovelluksen käyttöön liittyvissä virhetilanteissa tulee ilmaista selkeästi mikä meni vikaan ja miksi. Geneeriset ”tapahtui virhe”-tyyppiset virheilmoitukset eivät auta käyttäjää, vaan pikemminkin hämmentävät. (Babich 2018a.)

Navigointi ja siirtymät

Sovelluksessa navigoimisen tulee olla yksinkertainen ja käyttäjäystävällinen kokemus. Käyttäjälle täytyy aina olla selvää, missä osassa sovellusta hän on, ja miten hän pääsee haluamaansa paikkaan. Navigointivalikon tulee olla samassa paikassa jokaisessa näkymässä, jotta se on käyttäjäystävällinen ja helppokäyttöinen. (Babich 2018b.)

Babich (2018a) kertoo, että näkymien välisiä siirtymiä tulee kuvata animaatioilla, jotta käyttäjä tietää mitä tapahtuu. Son (2017) mukaan mobiililaitteilla käytetään yleensä kahdenlaisia siirtymiä − oikealta vasemmalle tai alhaalta ylös. Oikealta vasemmalle -siirtymää kutsutaan työnnöksi (engl. push) ja alhaalta ylös -siirtymää kutsutaan modaaliksi (engl. modal). Työntöä käytetään, kun siirrytään kokonaan uuteen näkymään. Modaalia puolestaan käytetään nykyiseen näkymään liittyvien tietojen tarkempaan esittämiseen, muokkaamiseen tai uuden tiedon syöttämiseen. Modaalinäkymä voi täyttää koko näytön tai vain osan siitä, sen koko riippuu sisällön määrästä. Esimerkiksi useita syötteitä sisältävän modaalinäkymän kannattaa olla koko näytön kokoinen, jotta sitä on helpompi käyttää. Modaalinäkymissä on yleensä peruuta- ja hyväksy-painikkeet. (So 2017.)

3.2 Mobile first -kehitysmalli

Perinteisessä kehitysmallissa verkkosivun tai web-sovelluksen toteutus aloitetaan työpöytäversiosta ja pienempiin näyttökokoihin siirrytään asteittain. Työpöytäversiota varten luodaan täysi kokemus sivusta tai sovelluksesta ja pienempiin näyttökokoihin siirryttäessä vähemmän olennaista sisältöä karsitaan tai käyttäjän vuorovaikutuksen taakse, jotta kaikki olennainen saadaan mahtumaan pienemmälle näytölle. Mobile first -kehitysmallissa tehdään täysin päinvastoin. Kehitystyö aloitetaan puhelimen pienestä näytöstä ja isompikokoisiin näyttöihin siirrytään asteittain. Suunnitteluvaiheessa valitaan tärkein sisältö, joka mahtuu puhelimen näytölle ja vähemmän

olennaista sisältöä lisätään näyttökoon kasvaessa. Perinteisessä kehitysmallissa käyttäjäkokemusta heikennetään asteittain ja mobile first -kehitysmallissa sitä parannetaan asteittain. (Hood 2015.)

Verkkosivujen ja web-sovellusten kehittämisen kannalta olennainen reagoiva muotoilu (engl. responsive design) sopii hyvin yhteen mobile first -kehitysmallin kanssa. Mediakyselyt (engl. media query) tehdään mobile first -kehitysmallissa siten, että kyselyjen sisältämät tyylit otetaan käyttöön näyttökoon kasvaessa eikä näyttökoon pienentyessä, kuten perinteisessä kehitysmallissa. Mobiililaitteet ovat suorituskyvyltään työpöytäversiota käyttäviä laitteita heikompia, joten mediakyselyjen sisältämien tyylien lataaminen tehokkaammilla laitteilla on järkevämpää. (Hood 2015.)

4 Angular

Angular on Googlen vuonna 2016 julkaisema JavaScript-sovelluskehys, joka on kehitetty TypeScript-ohjelmointikielellä. Angular on seuraava versio vuonna 2012 julkaistusta AngularJS-sovelluskehyksestä. Angular on edeltäjänsä tavoin saavuttanut nopeasti suuren suosion kehittäjien keskuudessa. Toiminnaltaan Angular on moderni komponenttipohjainen sovelluskehys, joka tukee suoraan web-, mobiili- ja työpöytäalustoja sekä yksikkö- ja end-to-end-testausta. (Bodrov-Krukowski 2018.) Fluinin (2018) mukaan Angularin nykyinen pääversio on toukokuussa 2018 julkaistu Angular 6.

Angular tarjoaa sovelluskehyksen lisäksi Angular CLI -komentorivityökalun, jonka tarkoitus on automatisoida osa tarvittavasta kehitystyöstä. Angular CLI:n avulla on helppo esimerkiksi aloittaa uusi Angular-sovellus, lisätä toimintoja kehitettävään sovellukseen, testata kehitysversiota verkkoselaimessa tai luoda tuotantoversio kehitettävästä sovelluksesta. Myös yksikkö- ja end-to-end-testien ajo toimii Angular CLI:n kautta. Angular-sovellus on mahdollista luoda, kehittää ja julkaista käyttämättä Angular CLI:tä, mutta se ei ole järkevä toimintapa, sillä Angular CLI:n käyttäminen säästää merkittävästi aikaa. (Van de Moere 2018.) Fluinin (2017) mukaan Angular CLI on yksi Angular-kehittäjän tärkeimmistä työkaluista.

Lukuisten työkalujen lisäksi Angular tarjoaa kehittäjille ohjeita muun muassa sovelluksen arkkitehtuuriin ja projektin ylläpitoon. Angularin suosion myötä sen ympärille on muodostunut laaja yhteisö, joten aktiivisesti kehitettäviä kolmannen osapuolen kirjastoja on paljon tarjolla ja kehitysongelmiin löytyy runsaasti apua verkosta. (Bodrov-Krukowski 2018.)

4.1 Arkkitehtuuri

Angular-sovellus koostuu moduuleista (engl. module), komponenteista (engl. component) ja palveluista (engl. service). Sovelluksen näkymien välillä navigoidaan Angularin reitittimen (engl. router) avulla. (Architecture overview n.d.)

Moduuli

Moduuli on TypeScript-luokka, joka toteuttaa @NgModule-funktion. Funktio ottaa vastaan yhden metatieto-olion, joka määrittää moduulin. (Introduction to modules n.d.) Moduuli on myös Angular-sovelluksen perusrakenneosa, joka kokoaa toisiinsa liittyvät komponentit ja palvelut yhteen. Angular-sovelluksessa täytyy olla vähintään päämoduuli, joka tarjoaa sovelluksen käynnistämiseen vaadittavan esilatausmekanismin. Angular-sovelluksessa on yleensä useita eri toimintoihin liittyviä moduuleita, jotka kokoavat näihin toimintoihin liittyvät komponentit ja palvelut yhteen toimiviksi sovelluksen osiksi. Moduuli voi sisältää myös muita moduuleita. Angular-sovelluksen jakaminen loogisiin moduuleihin helpottaa kehitystyötä ja tekee moduuleista helposti uudelleenkäytettäviä. (Architecture overview n.d.)

Komponentti

Komponentti on TypeScript-luokka, joka toteuttaa @Component-funktion. Komponentin tehtävä on vastata Angular-sovelluksen yksittäisen näkymän tiedoista ja logiikasta. Angular-sovelluksessa täytyy olla vähintään pääkomponentti, joka yhdistää komponenttihierarkian verkkosivun dokumenttioliomalliin. Moduulien tapaan Angular-sovelluksessa on yleensä useita komponentteja, joista jokaisen tehtävä on vastata yksittäisestä näkymästä. (Architecture overview n.d.)

Komponentin näkymä on HTML-tiedosto, johon voidaan direktiivien ja tietojen sitomisen (engl. data binding) avulla yhdistää komponentin sisältämä logiikka ja tiedot. Direktiivit ja sidottavat tiedot lisätään HTML-elementteihin ja dokumenttioliomalliin

ennen näkymän esittämistä. Komponentti voi sisältää kahdenlaista tietojen sitomista: käyttäjän vuorovaikutukseen vastaamista päivittämällä tietoa ja ohjelmallisesti lasketun tiedon lisäämistä näkymään. Angularin tukema kaksisuuntainen tietojen sitominen tarkoittaa näkymän ja komponentin reaaliaikaista vuorovaikutusta. (Architecture overview n.d.)

Palvelu

Palvelu on TypeScript-luokka, joka toteuttaa @Injectable-funktion ja sille määritetään yleensä yksi tarkka toiminnollinen tarkoitus. Palvelu suorittaa komponentin kutsusta sille määritetyn toiminnon, joka ei liity suoranaisesti komponentin näkymään. Palvelun tarkoitus on tarjota eri komponenteille niiden tarvitsema sama toiminto, kuten tiedon hakeminen palvelimelta. Palvelut voivat myös kutsua toisiaan. Komponenttien ja palveluiden eriyttäminen edistää Angular-sovelluksen modulaarisuutta ja parantaa toimintojen uudelleenkäytettävyyttä. (Introduction to services and dependecy injection n.d.)

Komponentille annetaan oikeus palvelun kutsumiseen riippuvuusinjektion (engl. dependency injection) avulla. Palvelun toteuttama @Injectable-funktio mahdollistaa palvelun injektoimisen komponenttiin riippuvuutena. Palvelun käyttäminen vaatii lisäksi sen rekisteröimisen tarjoajaksi vähintään yhdessä moduulissa tai komponentissa. Rekisteröimällä palvelun tarjoajaksi moduulissa voidaan sitä käyttää kaikissa kyseisen moduulin sisältämissä komponenteissa ja palveluissa. Rekisteröimällä palvelun tarjoajaksi yksittäisessä komponentissa voidaan sitä käyttää vain kyseisessä komponentissa. Riippuvuusinjektion avulla voidaan palvelun sijaan injektoida myös yksittäinen funktio tai arvo. (Introduction to services and dependecy injection n.d.)

Reititin

Angularin reititin mahdollistaa Angular-sovelluksen näkymien välillä siirtymisen. Reititin on toiminnaltaan JavaScript-reititin eli se päivittää verkkoselaimen osoitteen, kun sovelluksen tila muuttuu ja se päivittää sovelluksen tilan, kun selaimen osoite muuttuu. Reititin mahdollistaa yksisivuisen sovelluksen (engl. single-page application) toteuttamisen, sillä selaimen osoitteen muuttuessa uutta HTML-sivua ei ladata palvelimelta, vaan näkymä päivitetään dynaamisesti taustalla. (Van de Moere 2018.)

Reititin tallentaa siirtymät näkymien välillä verkkoselaimen historiaan, joten selaimen eteenpäin- ja taaksepäin-painikkeiden käyttäminen on mahdollista, vaikka selain pysyy samalla sivulla koko ajan. Reititys on myös mahdollista yhdistää mihin tahansa käyttäjän tekemään vuorovaikutukseen, kuten linkin tai painikkeen painamiseen. Reititin ja sen tilat täytyy konfiguroida ennen kuin sen käyttäminen on mahdollista. (Architecture overview n.d.)

4.2 TypeScript

TypeScript on Microsoftin vuonna 2012 julkaisema avoimen lähdekoodin ohjelmointikieli ja JavaScript-ohjelmointikielen ylijoukko. TypeScript sisältää kaiken saman toiminnallisuuden kuin JavaScript sekä lisäksi vielä omat toiminnallisuutensa. Näitä toiminnallisuuksia ovat esimerkiksi liittymät, luokat ja tyypit. Ennen verkkoselaimessa esittämistä TypeScript käännetään tavalliseksi JavaScriptiksi, koska TypeScript ei ole tuettu verkkoselaimissa. (Lease 2018.)

TypeScript on vahvasti tyypitetty ohjelmointikieli ja yksi sen tärkeimmistä ominaisuuksista on staattinen tyypitys, joka mahdollistaa tyypin antamisen muuttujalle määrittämättä sen arvoa. Modernit ohjelmointiympäristöt tukevat TypeScriptin tyypityksen tarkistusta, joka tarkistaa tyypityksen jo koodia kirjoittaessa ja antaa virheilmoituksen, jos muuttujaan yritetään asettaa erityyppinen arvo. Tämä auttaa välttymään useilta tyypitykseen liittyviltä ohjelmointivirheiltä, jotka ovat etenkin laajojen JavaScript-sovellusten ongelma, sillä se ei tue tyypityksen tarkistusta. (Lease 2018.)

Liittymän avulla olion ominaisuuksille voidaan määrittää tyypit. TypeScript tarkistaa, että liittymän ominaisuuksien tyypit vastaavat asetettuja, ja että kaikille liittymässä määritetyille ominaisuuksille on asetettu arvo. TypeScript antaa virheilmoituksen, jos annetut ominaisuudet tai niiden tyypit eivät täsmää. Liittymä käännettynä JavaScriptiksi on ei mitään. Liittymä on vain asetettujen määritelmien noudattamisen seuraamista varten ja sen ainoa tarkoitus on helpottaa kehitystyötä. (Parsy 2018.)

Luokka tarjoaa olio-ohjelmoinnista tutun olioiden luomisen luokan ominaisuuksien pohjalta ja perinnöllisyyden, eli perivä luokka voi käyttää perittävän luokan ominaisuuksia ja metodeja. JavaScript aloitti luokkien tarjoamisen ECMAScript 6 -standardin myötä, mutta TypeScript on tarjonnut luokat jo ennen tätä. (Classes n.d.)

4.3 Angular Material

Angular Material on Googlen kehittämä ulkoasukomponenttikirjasto, jota voidaan käyttää Angular-sovellusten kanssa. Material perustuu Googlen materiaalisuunnitteluun ja sen tarkoitus on helpottaa kauniiden ja reagoivien Angular-sovellusten kehittämistä. Material toimii saumattomasti Angularin kanssa, joten myös se tukee web-, mobiili- ja työpöytäalustoja. (Angular Material n.d.)

Ulkoasukomponentit

Angular Material sisältää paljon valmiita ulkoasukomponentteja, joiden avulla kauniin käyttöliittymän luominen onnistuu nopeasti. Materialin ulkoasukomponentit ovat valmiiksi tyyliteltyjä ja toiminallisuuksiltaan paranneltuja verkkosivun rakenneosia. Materialin ulkoasukomponentteja ovat esimerkiksi kortti, painike, syöte ja valikko. Material sisältää myös laajan ikonikirjaston, jonka saa käyttöön lisäämällä ikonifontin sovellukseen. (Bouchefra 2018.)

Ulkoasukomponentit sisältävät paljon konfigurointimahdollisuuksia, joten komponenttien toimintaa ja ulkoasua voi helposti muokata. Yksittäisen ulkoasukomponentin konfiguraation muokkaaminen onnistuu lisäämällä komponenttiin HTML-attribuutti ja arvo. Kaikki komponenttikohtaiset konfiguraatiomahdollisuudet ja paljon komponenttien käyttöön liittyviä esimerkkejä on listattu Materialin dokumentaatiossa. (Components n.d.) Komponenttien tyylejä voi myös muokata CSS:n avulla, jos konfiguraatiot eivät tarjoa riittäviä kustomointimahdollisuuksia. Angular tarjoaa kaksi tapaa sovellusten tyylittelyyn − globaalin ja komponenttikohtaisen. (Customizing Angular Material component styles n.d.)

Teema

Ulkoasukomponenttien lisäksi Material sisältää teeman, jonka avulla Angular-sovellukseen on helppo integroida haluttu värimaailma. Teema koostuu useista väripaleteista, joista jokainen määrittää sovelluksen eri osissa käytettävät värit. Teema koostuu seuraavista paleteista:

- Pääpaletti – sovelluksen päävärit
- Korostuspaletti – interaktiivisten ulkoasukomponenttien värit
- Varoituspaletti – virhetilanteita ilmaisevat värit

- Etualapaletti – tekstissä ja ikoneissa käytettävät värit
- Taustapaletti – ulkoasukomponenttien ja elementtien taustavärit.

Teeman jokaisen paletin voi muokata mieluisekseen ja nopeampaa kehitystä varten Material tarjoaa neljä valmista teemaa, jotka voi ottaa suoraan käyttöön. Eri ulkoasukomponenteille voi myös luoda omat teemansa ja Angular-sovelluksella voi olla useita teemoja. (Theming your Angular Material app n.d.)

Component Dev Kit

Materialin tarjoaman komponenttien kehityspakkauksen avulla on mahdollista luoda omia uudelleenkäytettäviä ulkoasukomponentteja. Kehityspakkausta hyödyntämällä omien ulkoasukomponenttien luominen on huomattavasti nopeampaa kuin tyhjältä pohjalta aloittaessa. Komponenttien kehityspakkaus sai alkunsa Materialin ulkoasukomponenttien jakamien rakenteellisten yhtenäisyyksien pohjalta. Kehityspakkaus on jaettu viiteen eri osaan, joista jokainen kattaa ulkoasukomponentin kokonaisuuden yhden osa-alueen. Nämä osat ovat helppokäyttöisyys, kaksisuuntainen teksti, päällys, pohja ja taulu. (Elbourn 2018.)

4.4 Parhaat kehityskäytänteet

Angularin parhaat käytänteet korostavat erityisesti selkeää rakennetta, helposti luettavaa koodia ja toimintojen uudelleenkäytettävyyttä. (Style Guide n.d.)

Rakenne

Kaikki Angular-sovelluksen ohjelmakoodi sijoitetaan src-hakemistoon. Kolmannen osapuolen tekemä koodi, kuten käytettävät apukirjastot sijoitetaan src-hakemiston ulkopuolelle. Päämoduuli, pääreititysmoduuli ja pääkomponentti sijoitetaan src-hakemistossa olevaan app-hakemistoon. Sovelluksen eri osat sijoitetaan app-hakemiston alle omiin hakemistoihinsa ja niillä jokaisella on oma moduulinsa. Hakemisto- ja tiedostorakenteen kanssa kannattaa pyrkiä litteään rakenteeseen. Jokaista tiedostoa ei kannata laittaa omaan alihakemistoonsa, jotta tiedostopolut pysyvät lyhyinä. (Style Guide n.d.)

Komponenttien, palveluiden ja muiden Angular-sovelluksen toimintojen tulisi aina palvella vain yhtä tarkoitusta. Kun jokainen sovelluksen osa suorittaa vain yhden sille

määritetyn tehtävän, niin sovellusta on helpompi kehittää ja testata. Tätä yhden tarkoituksen periaatetta kutsutaan yhden säännöksi (engl. rule of one) ja se koskee myös tiedostoja. Esimerkiksi tiedostossa, joka sisältää komponentin tulee määrittää vain kyseisen komponentin toiminnasta vastaava luokka. Mikäli komponentti tarvitsee esimerkiksi liittymän, se tulee sijoittaa omaan tiedostoonsa ja importoida komponenttiin. Noudattamalla yhden sääntöä tulee ohjelmakoodista helppolukuista ja uudelleenkäytettävää. (Style Guide n.d.)

Nimeäminen

Nimeämiskäytänteiden tulee olla yhdenmukaiset koko Angular-sovelluksen laajuisesti. Yhdenmukaiset nimet tekevät kokonaisuudesta selkeän ja ne auttavat tiedostojen sekä koodin löytämisessä. Käytettävien nimien tulee myös selkeästi kuvata toimintoa. Selkeästi kuvaavat nimet ovat parempia kuin lyhyenteitä käyttävät nimet (Style Guide n.d.)

Angular-sovelluksen toimintojen tiedostonimet koostuvat kolmesta osasta − kuvauksesta, tyypistä ja tiedostopäätteestä. Kuvausosan sanat erotetaan toisistaan väliviivalla. Kuvaus-, tyyppi- ja tiedostopääteosat erotetaan toisistaan pisteellä. Tiedoston nimen tulee ilmaista sen tyyppi, jotta sen nimestä voi helposti päätellä mitä se sisältää. (Style Guide n.d.)

Luokkien ja liittymien nimet kirjoitetaan ylemmällä kamelikirjoituksella eli ensimmäinen kirjain on isolla. Muuttujien, konstanttien ja metodien nimet kirjoitetaan alemmalla kamelikirjoituksella eli ensimmäinen kirjain on pienellä. Moduulien, komponenttien ja muiden toimintojen luokkien nimiin on suositeltavaa lisätä päätteeksi kyseinen toiminto. (Style Guide n.d.)

Komponentti

Komponentin tulee sisältää vain näkymään liittyvä logiikka. Muut metodit tulee sijoittaa palveluihin. Uudelleenkäytettävän näkymään liittyvän logiikan voi kuitenkin sijoittaa palveluun, jos sille on käyttöä useassa eri paikassa. (Style Guide n.d.)

Jos komponentin näkymän mallinne on yli kolme riviä, niin se tulee laittaa omaan HTML-tiedostoonsa. Monimutkainen logiikkaa kannattaa sijoittaa mallinteen sijasta

komponenttiin, jotta kaikki näkymään liittyvä monimutkaisempi logiikka on yhdessä paikassa. (Style Guide n.d.)

Palvelu

Palvelua käytetään tiedon ja toiminallisuuksien jakamiseen. Palvelulla tulisi olla vain yksi tietty käyttötarkoitus. Monitarkoituksellinen palvelu on epäselkeitä ja sitä on hankalampi testata. Laajat monitarkoituksellinen palvelu saattaa myös aiheuttaa ylimääräistä kuormaa komponenteille tai muille palveluille, jos palvelu sisältää paljon metodeja, joita sitä käyttävä toiminto ei tarvitse. (Style Guide n.d.)

Palvelu importoidaan käytettäväksi komponentin tai toisen palvelun constructor-funktiossa. Tarjoamalla palvelun sovelluksen päämoduulissa kaikki luokat pystyvät käyttämään sitä. Tarjoamalla palvelun useassa eri moduulissa syntyy siitä useita instansseja, mikä saattaa aiheuttaa ei haluttuja tilanteita. Palvelun @Injectable-funktioon kannattaa antaa metatietona tieto moduulista, jossa palvelu tarjotaan. Tällöin Angular CLI osaa sovelluksen tuotantoversiota kääntäessä karsia pois käyttämättömät palvelut. (Style Guide n.d.)

5 Web-sovelluksen kehittäminen

Tässä luvussa käydään läpi opinnäytetyön kehitystyönä kehitetyn web-sovelluksen arkkitehtuuri, käyttöliittymä ja tekninen toteutus. Kehityksen aikana tehdyt valinnat perustellaan ja koodiesimerkkejä käytetään apuna.

Web-sovelluksen kehitystyöhön käytettiin toimeksiantajan tarjoamia työkaluja. Kehitysympäristönä oli käytössä PhpStorm, joka tukee Angular-kehityksessä käytettävää TypeScript-ohjelmointikieltä. PhpStorm osaa ilmoittaa esimerkiksi tyypitysvirheistä reaaliajassa, jolloin ohjelmointivirheiltä on helpompi välttyä. Sovelluksen ohjelmakoodin hallintaan käytettiin GitHub-versionhallintaa.

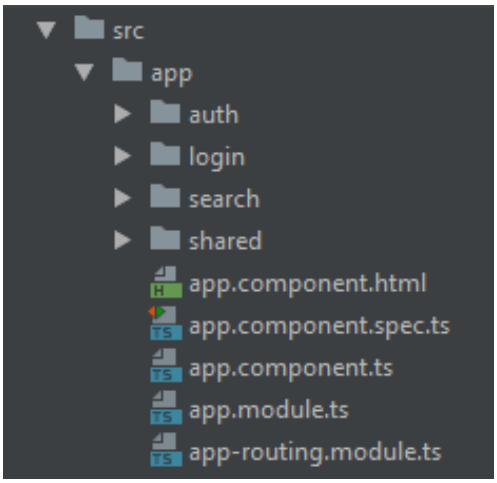
Kehityksen aikana web-sovellusta ajettiin paikallisesti verkkoselaimessa kehitystyöhön käytettävällä tietokoneella. Angular CLI tarjoaa ”ng serve”-komennon, joka kääntää TypeScript-koodin JavaScript-koodiksi ja tarjoilee sovelluksen, jonka jälkeen sitä

voi käyttää verkkoselaimella osoitteessa localhost:4200. PhpStorm tunnistaa tehdyt koodimuutokset ja tallennettaessa Angular CLI päivittää selaimessa pyörivän sovelluksen automaattisesti, joten tehdyt muutokset on helppo nähdä ja testata välittömästi.

5.1 Arkkitehtuuri

Kehitettävä web-sovellus on pieni kokonaisuus, joka on jaettu loogisesti eri moduuleihin, jotta kehitystyö sujuu helpommin ja sovellus on helposti laajennettavissa. Helppo laajennettavuus on tärkeää, sillä sovellusta jatkokehitetään tulevaisuudessa hyvin todennäköisesti.

Web-sovellus koostuu päämoduulin lisäksi autentikointi-, kirjautumis-, haku- ja jaettumoduuleista. Moduulit on sijoitettu omiin hakemistoihinsa ja ne on jaettu niiden sisältämän toiminnallisuuden mukaan selkeästi erilleen (ks. kuvio 1). Jokaisen moduulin omasta hakemistosta löytyy moduuliin liittyvä komponentti ja muut sen toimintaan liittyvät osat, kuten palvelut ja liittymät.



Kuvio 1. Web-sovelluksen app-hakemiston sisältö

Web-sovelluksen yleishyödylliset osat kuten konstanttit, Material-moduuli ja pikapalkkipalvelu on sijoitettu shared-hakemistoon. Material-moduulissa tarvittavat Materialin ulkoasukomponentit importoidaan sovellukseen ja pikapalkkipalvelun kautta käyttäjälle näytetään sovelluksen käyttöön liittyviä virheilmoituksia. Shared-hakemiston sisältämien toimintojen yhteen kokoamisesta vastaa hakemiston juuressa oleva moduuli.

App-hakemiston lisäksi src-hakemistosta löytyy assets-, environments- ja styles-hakemistot. Web-sovelluksessa käytettävät kuvatiedostot ja JSON-tiedosto, joka sisältää suomenkieliset tekstit on sijoitettu assets-hakemistoon. Sovelluksen vaatimuksiin ei kuulunut monikielisyys, mutta ngx-translate-kirjaston käyttäminen toteutuksessa aiheutti vain vähän vaivaa.

Environments-hakemistossa ovat ympäristötiedostot, jotka määrittävät web-sovelluksen konfiguraation kehitys- ja tuotantoversioita varten. Angular-sovelluksesta voi tehdä tuotantoversion ajamalla ”ng build --prod”-komennon Angular CLI:ssä.

Web-sovelluksen tyylitiedostot löytyvät styles-hakemistosta. Sovelluksen käyttämät tyylit on jaettu loogisiin osiin hajauttamalla ne useisiin eri tyylitiedostoihin. Hakemistosta löytyy myös components-, fonts- ja palettes-hakemistot. Components-hakemisto sisältää komponenttikohtaiset tyylitiedostot. Halutut tyylit on helppo löytää, kun yhden ison tyylitiedoston sijasta on useita selkeästi jaoteltuja tyylitiedostoja. Sovelluksen käyttämät fontit on määritetty fonts-hakemistossa ja sovelluksen teeman muodostavat pää- ja varoituspaletit on määritetty palettes-hakemistossa.

5.2 Käyttöliittymä

Web-sovelluksen käyttöliittymä kehitettiin ennakkoon tehdyn käyttöliittymädemon pohjalta. Demo sisälsi vain näkymien ulkoasut, joten lopullisen käyttäjäkokemuksen luominen jäi opinnäytetyön tekijän vastuulle. Käyttöliittymä muuttui kehitystyön aikana hieman. Suurin osa käyttöliittymämuutoksista tehtiin opinnäytetyön tekijän aloitteesta ja ne pohjautuivat testaamisen aikana havaittuihin asioihin. Salassapitosopimuksesta johtuen sovelluksen käyttöliittymästä ei esitetä kuvankaappauksia.

Kirjautuminen

Web-sovelluksen kirjautumisen käyttöliittymä on hyvin simppeli. Kirjautumisnäkymä koostuu kirjautumislomakkeesta, joka sisältää käyttäjätunnus- ja salasanasyötteet sekä ”Kirjaudu sisään”-painikkeen. Salasanasyötteen oikeassa laidassa on ikonipainike, josta salasanan saa näkyviin ja takaisin piiloon. Kirjautumisen käyttöliittymä ei sisällä palveluun rekisteröitymistä tai unohtuneen salasanan palauttamista, sillä kirjautuminen tapahtuu ulkoisen rajapinnan kautta ja nämä toimenpiteet tehdään kyseisen rajapinnan omassa palvelussa.

Kirjautumislomake sisältää myös virheenkäsittelyä. Käyttäjätunnus ja salasana ovat pakollisia tietoja ja niiden syötteet antavat virheilmoituksen, jos tietoa ei ole annettu. Virheellisestä käyttäjätunnuksesta tai salasanasta sovellus antaa virheilmoituksen pikapalkkipalvelun kautta, joka luo näytön ylälaitaan virheellisistä kirjautumistiedoista ilmoittavan viestin.

Rajausdialogi

Käyttäjän kirjauduttua hänet siirretään päänäkymään ja avataan rajausdialogi, jossa käyttäjän täytyy asettaa kahteen valintasyötteeseen rajausehdot, joita käytetään rajapintahakujen hakuparametreissä. Jälkimmäisen syötteen näyttämät rajausvaihtoehdot pohjautuvat ensimmäisen syötteen rajausehtoon, joten jälkimmäinen syöte on pois päältä, kunnes ensimmäiseen syötteeseen on annettu rajausehto. Syötteet ilmoittavat puuttuvasta tiedosta, jos rajausehtoa ei ole annettu. Rajausehdot asetetaan ”Aseta rajaukset”-painikkeesta ja niiden asettamisen jälkeen käyttäjä pääsee päänäkymään, jossa hän voi tehdä rajapintahakuja.

Rajausdialogissa on myös ”Peruuta”-painike, joka palauttaa käyttäjän takaisin päänäkymään. Sitä ei näytetä käyttäjälle heti kirjautumisen jälkeen, vaan vasta kun käyttäjä on jo kertaalleen asettanut rajausehdot ja tulee myöhemmin muuttamaan niitä.

Päänäkymä

Päänäkymä koostuu ylätunnisteesta, toissijaisesta ylätunnisteesta, hakutuloslistasta ja alatunnisteesta. Ylätunnisteen vasemmassa laidassa on web-sovelluksen nimi ja oikeassa laidassa ikonipainike, josta aukeaa valikko. Toissijaisen ylätunnisteen vasemmassa laidassa näkyy hakutulosten määrä ja oikeassa laidassa on ikonipainikkeet hakutulosten järjestämisvalikolle ja rajausdialogin avaamiselle. Hakutuloslista sisältää haetut tiedot esitettynä Materialin korttiulkoasukomponentteina, joiden avulla hakutulokset on helppo erottaa toisistaan. Alatunniste sisältää pikahakusyötteen ja tarkennettu haku -dialogin avaamispainikkeen.

Alatunnisteen pikahakusyötettä napauttamalla aukeaa mobiililaitteen näppäimistö ja alatunniste asettuu mobiilinäppäimistön päälle, jolloin käyttäjän on helppo nähdä syötettävä hakusana. Haku tehdään mobiilinäppäimistön enter-näppäimestä. Haun

alkaessa pikahakusyöte palautetaan takaisin alkutilaan, mobiilinäppäimistö piilotetaan, hakutuloslista tyhjennetään ja käynnissä olevaa hakua aletaan ilmaisemaan Materialin edistymishyrrällä. Haun valmistuessa edistymishyrrä piilotetaan ja löydetyt hakutulokset esitetään korttilistana. Jos haku ei löydä tuloksia tai haussa tapahtuu virhe, niin näytetään ”Ei hakutuloksia”- tai ”Haussa tapahtui virhe, ole hyvä ja yritä uudestaan”-teksti.

Hakutuloslistan korteilla näytetään olennaisimmat tiedot ja yksittäistä korttia napauttamalla aukeaa tarkemmat tiedot -dialogi, jossa näkyy kyseisen hakutuloksen perustietojen lisäksi sen tarkemmat tiedot. Dialogin avaaminen ei ole aina välttämätöntä käytön kannalta. Käyttäjä voi sulkea dialogin ”Sulje”-painikkeesta, mobiililaitteen omasta taaksepäin-painikkeesta tai napauttamalla dialogin ulkopuolelta

Tarkennettu haku -dialogi

Tarkennettu haku -dialogissa on hakulomake, joka sisältää kuusi eri syötettä, joihin käyttäjä voi syöttää tarkentavia hakuehtoja. Dialogissa on lisäksi ”Hae”- ja ”Peruuta”-painikkeet. Käyttäjän tulee antaa tieto vähintään yhteen syötteeseen, jotta haun voi tehdä. Haun alkaessa käyttäjä siirretään takaisin päänäkymään, hakutuloslista tyhjennetään ja edistymishyrrä alkaa ilmaisemaan käynnissä olevaa hakua.

Hakulomakkeen tarkistus ei tapahdu perinteisesti syötteittäin, vaan tarkistetaan, että ainakin yhteen syötteeseen on annettu hakuehto. Jos haun yrittää suorittaa antamatta yhtäkään hakuehtoa, niin käyttäjälle näytetään virheilmoitus pikapalkkipalvelun kautta näytön ylälaidassa.

Ulkoasukomponentit

Käyttöliittymä kehitettiin käyttäen Materialin ulkoasukomponentteja ja niiden hyödyntäminen säästi merkittävästi kehitysaikaa, sillä ne tarjosivat valmiin pohjan käyttöliittymälle. Eri ulkoasukomponentteja käytettiin monipuolisesti ja niiden konfiguroiminen tarkoitukseen sopivaksi onnistui helposti. Web-sovelluksen käytetyimmät ulkoasukomponentit olivat:

- Painike – sovelluksen painikkeet ovat button-elementtejä, joilla on mat-button- tai mat-raised-button-attribuutti

- Syöte – sovelluksen lomakkeissa on käytössä Materialin päivämäärävalitsin-, teksti- ja valintasyötteitä
- Dialogi – sovelluksessa käytettäviä dialogeja ovat hakutuloksen tarkemmat tiedot -näkymä, rajausnäkymä ja tarkennettu haku -näkymä.

Rajausnäkymä sekä tarkennettu haku -näkymä ovat koko näytön dialogeja, koska molemmat sisältävät lomakkeen ja vaativat käyttäjän vuorovaikutusta. Hakutuloksen tarkemmat tiedot -näkymä ei ole koko näytön dialogi, koska sen tarkoitus on ainoastaan esittää tietoa. Muita web-sovelluksessa käytettäviä Materialin ulkoasukomponentteja ovat edistymishyrrä, ikoni, jakaja, kortti, lista, merkki, pikapalkki, työkalupalkki ja valikko.

Materialin ulkoasukomponentit nopeuttivat ja helpottivat käyttöliittymän tekemistä, mutta ne eivät täyttäneet web-sovelluksen kaikkia tarpeita. Rajausdialogin lomakkeessa täytyi käyttää ngx-mat-select-search-kirjastoa, jotta saatiin valintasyöte, jonka vaihtoehtoja voi suodattaa tekstisyötteen avulla. Toinen Materialista ilmennyt puute oli, että päivämäärävalitsimeen voi antaa päivämäärän vain yhdessä muodossa. Päivämäärä annetaan nyt muodossa pp.kk.vvvv, mutta muoto kk.vvvv olisi ollut myös hyödyllinen. Tarve usealle eri päivämäärämuodolle ei ollut kriittinen, joten siihen ei etsitty kolmannen osapuolen ratkaisua.

Käyttöliittymän optimoiminen mobiilikäyttöön

Web-sovelluksen käyttöliittymä on tehty mobiilikäyttö mielessä ja se muistuttaa hyvin pitkälti natiivia mobiilisovellusta. Käyttöliittymän mobiilioptimointia korostaa erityisesti elementtien mobiilisovellusmainen asettelu ja ikonien monipuolinen hyödyntäminen. Käyttöliittymän isoin visuaalinen ero natiivisovellukseen on verkkoselaimen osoitepalkki, joka rajoittaa käytettävissä olevaa näyttötilaa.

So (2017) mainitsi, että mobiilisovelluksen tablettikäyttöliittymän tulisi näyttää työpöytäversiolta, mutta toimia teknisesti kuten mobiiliversion. Kehitettävästä web-sovelluksesta ei kuitenkaan tehdä työpöytäversiota, joten tablettikäyttöliittymä on vain tilavampi versio puhelinkäyttöliittymästä. Tabletin näytön koosta on otettu hyöty irti, joten painikkeet ja syötteet ovat selkeästi erillä toisistaan. Sovelluksen vähäisten toimintojen myötä kaiken tarvittavan sovittaminen viiteen eri näkymään onnistui hyvin

myös puhelimella, jolla käyttöliittymä on huomattavasti tiiviimpi. Painikkeiden, syötteiden ja muiden elementtien välissä on kuitenkin riittävät välit, jotta virhenapautuksia ei synny. Sovellusta käyttäessä ei tule tunnetta, että näytöllä on liikaa asioita kerralla.

Käyttöliittymän mobiilioptimoinnin isoin ongelmakohta oli mobiililaitteen näppäimistö. Kirjautumisnäkymän alatunniste täytyi piilottaa ohjelmallisesti mobiilinäppäimistön noustessa esiin, koska se nousi näppäimistön päälle. Kyseisestä näppäimistöongelmasta oli myös hyötyä. Päänäkymän alatunnisteessa oleva pikahakusyöte nousee samalla tavoin mobiilinäppäimistön päälle, mutta tässä tilanteessa sen sijainti helpottaa annettavan hakusanan lukemista.

Teema

Web-sovelluksen teemaa varten määritettiin pää- ja varoituspaletit. Pääpaletin värejä käytetään edistymishyrrissä, ikonipainikkeissa, jakajissa, otsikoissa, painikkeissa, syötteissä ja työkalupalkissa. Päävärin monipuolinen käyttö luo sovellukselle yhdenmukaisen käyttöliittymän, joka sopii asiakasyrityksen brändiin. Varoituspaletin värejä käytetään sovelluksen virheilmoituksissa sekä merkissä, joka ilmaisee käytössä olevien rajausehtojen määrän.

5.3 Tekninen toteutus

Tässä kappaleessa käydään web-sovelluksen päätoiminnot läpi moduuleittain. Salassapitosopimuksesta johtuen sovelluksen tarkkaan käyttötarkoitukseen viittaavien metodien nimet on muutettu ja kuvankaappauksista sellaiset termit on sensuroitu. Muutettujen metodien nimiin on asetettu jokin kirjain salassa pidettävän termin tilalle.

Kirjautuminen ja autentikointi

Kirjautumismoduuliin kuuluu reititysmoduuli, kirjautumiskomponentti ja -palvelu. Reititysmoduuli vastaa kirjautumisnäkymän reitittämisestä, kirjautumiskomponentti vastaa kirjautumisnäkymän logiikasta ja kirjautumispalvelu vastaa autentikointimoduulin kanssa vuorovaikuttamisesta sekä käyttäjän ohjaamisesta oikeaan reittiin sisään- ja uloskirjautuessa.

Reititysmoduulissa kirjautumisreitti on määritetty noudattamaan autentikointimoduulista löytyvää anonyymivahtia (engl. guard), joka päästää vain kirjautumattoman käyttäjän kirjautumisnäkymään. Jos kirjautunut käyttäjä yrittää siirtyä verkkoselaimen osoitepalkin kautta kirjautumisnäkymään hänet ohjataan takaisin päänäkymään, koska hän on jo kirjautunut sisään. Vastaavasti hakumoduulin reititysmoduuli on määritetty noudattamaan autentikointivahtia, joka päästää vain kirjautuneen käyttäjän muuhun kuin kirjautumisnäkymään.

Kirjautumisnäkymän kirjautumislomakkeen ”Kirjaudu sisään”-painike kutsuu kirjautumiskomponentin logIn-metodia, joka tarkistaa, että käyttäjätunnus ja salasana on annettu. Lomakkeen ollessa validi kutsutaan kirjautumispalvelun logIn-metodia (ks. kuvio 2) ja välitetään parametrinä käyttäjän syöttämät kirjautumistiedot.

```
/**
 * Log user in with given credentials
 *
 * @param {Object} credentials
 */
public logIn(credentials) {
  this.authService.authenticate(credentials)
    .subscribe(() => {
        this.router
          .navigate(['/'])
          .then(() => { });
      },
      err => {
        if (err.status === 403) {
          this.snackBarService.showErrorMessage('login.error.invalid_username_or_password', 5000);
        }
      });
}
```

Kuvio 2. Kirjautumispalvelun logIn-metodi

Kirjautumispalvelun logIn-metodi kutsuu autentikointipalvelun authenticate-metodia välittäen kirjautumistiedot parametrinä. Autentikointipalvelu kutsuu ulkoista rajapintaa kirjautumistiedoilla ja niiden kelvatessa rajapinta palauttaa JSON-olion, joka sisältää kaksi JSON Web Tokenia – pääsypoletin (engl. access token) ja päivityspoletin (engl. refresh token). Poletit asetetaan paikalliseen varastoon (engl. local storage) talteen ja pääsypoletti puretaan auki, jonka jälkeen sen sisältämien käyttäjätietojen pohjalta asetetaan kirjautunut käyttäjä. Pääsypolettia käytetään myös web-sovelluksen back-endiin autentikoimiseen. Päivityspolettia käytetään pääsypoletin päivittämi-

seen, jos se on vanhentunut. Kirjautuneen käyttäjän asettamisen jälkeen tieto onnistuneesta autentikoinnista palautuu kirjautumispalveluun ja käyttäjä ohjataan sovelluksen päänäkymään. Jos käyttäjän antamat kirjautumistiedot ovat virheelliset kirjautumispalvelu kutsuu pikapalkkipalvelua, joka näyttää käyttäjälle virheilmoituksen.

Web-sovelluksen päänäkymän ylävalikossa on ”Kirjaudu ulos”-painike, joka kutsuu kirjautumispalvelun logOut-metodia (ks. kuvio 3), joka kutsuu autentikointipalvelun logout-metodia. Metodi poistaa sisäänkirjautuessa asetetut JWT:t paikallisesta varastosta ja asettaa kirjautunut käyttäjä -arvon tyhjäksi. Uloskirjautumistiedon palautuessa kirjautumispalveluun käyttäjä ohjataan kirjautumisnäkymään.

```
/**
 * Log current user out
 */
public logOut() {
  this.authService.logout()
    .subscribe(() => {
      // It's possible that filter dialog is open when this is called, so close all dialogs
      this.dialog.closeAll();
      this.router
        .navigate(['/login/'])
        .then(() => { });
    },
    () => {
      this.dialog.closeAll();
      this.zone.run(() => this.router.navigate(['/login/']));
    });
}
```

Kuvio 3. Kirjautumispalvelun logOut-metodi

Kirjautumismoduuli on kokonaan opinnäytetyöntekijän toteuttama, mutta suurin osa käytettävästä autentikointimoduulista on otettu toisesta toimeksiantajan toteuttamasta Angular-sovelluksesta. Autentikointimoduulin sovittaminen kehitettävään web-sovellukseen vaati kuitenkin muutoksia, sillä autentikointimoduuli on toteutettu Angular 5:llä käyttäen RxJS 5 -kirjastoa. Web-sovellus kehitetään Angular 6:lla käyttäen RxJS 6 -kirjastoa. RxJS 6:n myötä monet aiemman version tarjoamat metodit ovat vanhentuneita, joten vanhentuneet metodikutsut täytyi muuttaa. Lisäksi useiden metodien sijainnit ovat muuttuneet uuden version myötä, joten niiden importointikutsut täytyi muuttaa. Autentikointimoduuli sisälsi myös toimintoja, joita ei tarvittu, joten ne poistettiin.

Hakumoduuli

Hakumoduuliin kuuluu reititysmoduuli, hakukomponentti, kolme dialogikomponenttia ja hakupalvelu. Dialogikomponentit ovat rajauskomponentti, tarkemmat tiedot -komponentti ja tarkennettu haku -komponentti. Hakukomponentti vastaa web-sovelluksen päänäkymästä, jossa käyttäjä voi tehdä rajapintahakuja ja selata hakutuloksia. Rajauskomponentti vastaa hauissa käytettävien rajausehtojen asettamisesta, tarkemmat tiedot -komponentti vastaa hakutulosten tarkempien tietojen esittämisestä ja tarkennettu haku -komponentti vastaa tarkennettujen hakujen tekemisestä. Hakupalvelu (ks. kuvio 4) vastaa hakukomponentin ja back-endin välisestä vuorovaikuttamisesta.

```
/**
 * Get [redacted]
 *
 * @param request - request params
 *
 * @returns {Observable<[redacted][]>}
 */
public search[redacted](request): Observable<[redacted][]> {
  return this.http.get<[redacted][]>(this.API_URL + 'search/[redacted]', request)
    .pipe(
      map((res: any) => {
        return res;
      }),
      catchError((err: any) => throwError(err))
    );
}

/**
 * Get [redacted]
 *
 * @param request - request params
 *
 * @returns {Observable<[redacted][]>}
 */
public search[redacted](request): Observable<[redacted][]> {
  return this.http.get<[redacted][]>(this.API_URL + 'search/[redacted]', request)
    .pipe(
      map((res: any) => {
        return res;
      }),
      catchError((err: any) => throwError(err))
    );
}
```

Kuvio 4. Hakupalvelun metodit

Rajausdialogi

Sisäänkirjautumisen jälkeen käyttäjä siirretään päänäkymään ja rajausdialogi avataan. Käyttäjän täytyy antaa kaksi rajausehtoa ennen kuin hän voi aloittaa hakujen tekemisen. Käyttäjän valitessa rajausehdon ensimmäisestä valintasyötteestä (selectionChange)-attribuutti kutsuu rajauskomponentin getX-metodia, joka kutsuu hakupalvelun getX-metodia välittäen parametrinä valintasyötteeseen annetun rajausehdon. Hakupalvelun getX-metodi lähettää hakuparametrin mukaisen pyynnön back-endiin. Back-end lähettää pyynnön ulkoiseen rajapintaan, joka palauttaa rajauslomakkeen toisessa syötteessä näytettävät rajausvaihtoehdot. Ennen rajausvaihtoehtojen palautumista rajauskomponenttiin jälkimmäinen valintasyöte on poissa käytöstä. Rajausvaihtoehtojen haun aikana jälkimmäisen valintasyötteen ikonin tilalla pyörii Materialin edistymishyrrä, joka ilmaisee käynnissä olevaa hakua. Jälkimmäisen valintasyötteen näyttämät rajausvaihtoehdot haetaan aina käyttäjän muuttaessa ensimmäisen valintasyötteen arvoa. Rajausehtojen hakeminen kestää vain muutaman sekunnin hitaammallakin verkkoyhteydellä.

Toinen mahdollinen toteutustapa rajausvaihtoehtojen hakemiselle olisi voinut olla kaikkien tarvittavien rajausvaihtoehtojen hakeminen heti sisäänkirjautumisen jälkeen ja niiden tallentaminen paikalliseen varastoon. Tällöin rajausvaihtoehdot olisivat saatavilla heti, kun käyttäjä on asettanut rajausehdon ensimmäiseen valintasyötteeseen. Asiakasyrityksen mukaan käyttäjien tarvitsee vaihtaa rajausehtoja korkeintaan muutaman kerran päivän aikana, joten yksinkertaisempi toteutus sopii käyttötarkoitukseen hyvin.

Kun käyttäjä on asettanut rajausehdot ja napauttaa ”Aseta rajaukset”-painiketta kutsutaan rajauskomponentin setFilters-metodia (ks. kuvio 5). Metodi tarkistaa, että rajauslomake on validi ja sulkee rajausdialogin välittäen asetetut rajausehdot hakukomponenttiin. Jos rajausvaihtoehtojen hakeminen epäonnistuu ja käyttäjä yrittää hyväksyä rajausehdot ilman jälkimmäisen rajausvaihtoehdon asettamista, kutsuu setFilters-metodi pikapalkkipalvelua, joka näyttää käyttäjälle virheilmoituksen. Jälkimmäisen valintasyötteen ikoni korvataan ikonipainikkeella, jota napauttamalla käyttäjä voi yrittää rajausvaihtoehtojen hakemista uudestaan. Rajausvaihtoehtohaun aikana rajauslomake on hetkellisesti validi, joten rajausehtojen hyväksyminen haun ollessa käynnissä on estetty.

```
/**
 * Sets filter form values as new filter values
 */
public setFilters() {
  if (this.error && this.filters.value.type !== 7) {
    this.snackBarService.showErrorMessage('form.error.check_connection_and_try_again', 2500);
  } else if (this.filters.valid && !this.loading) {
    this.dialogRef.close(this.filters.value);
  }
}
```

Kuvio 5. Rajausdialogikomponentin setFilters-metodi

Rajausdialogin jälkimmäinen valintasyöte käyttää ngx-mat-select-search-kirjastoa, joka mahdollistaa valintasyötteen vaihtoehtojen suodattamisen tekstisyötteen avulla. Jälkimmäinen valintasyöte voi sisältää satoja rajausvaihtoehtoja, joten tekstisuodatus on erittäin tarpeellinen hyvän käyttäjäkokemuksen kannalta.

Hakukomponentti

Päänäkymästä tehdään hakuja alatunnisteen pikahakulomakkeen kautta, joka kutsuu hakukomponentin search-metodia. Metodissa annettu hakuehto asetetaan hakuparametreihin ja kutsutaan hakukomponentin searchY-metodia, jossa hakuparametreihin lisätään rajausdialogissa asetetut rajausehdot. Tämän jälkeen searchY-metodi kutsuu hakupalvelun searchY-metodia välittäen hakuparametrit. Hakupalvelun searchY-metodi lähettää hakuparametrien mukaisen pyynnön back-endiin, joka lähettää pyynnön ulkoiseen rajapintaan. Hakutulokset palautuvat hakupalvelun kautta hakukomponenttiin, jonka jälkeen ne näytetään listana päänäkymässä. Hakuparametrien asettaminen on jaettu kahteen eri metodiin, koska myös tarkennettu haku -dialogissa tehtävät haut käyttävät searchY-metodia rajausehtojen lisäämiseen ja hakutulosten näyttämiseen.

Tarkemmat tiedot -dialogi

Hakutuloksen napauttaminen kutsuu hakukomponentin openZDialog-metodia, joka avaa tarkemmat tiedot -dialogin (ks. kuvio 6). Dialogiin välitetään parametrinä hakutulosolio ja dialogissa näytetään kaikki sen sisältämät tiedot. Tarkemmat tiedot -dialogi tarjoaa paljon hyödyllistä lisätietoa hakutulokseen liittyen, mutta kaikki olennaisimmat tiedot näkyvät kuitenkin hakutuloslistassa, joten käyttäjän ei ole välttämätöntä aina napauttaa dialogia auki.

```
import { Component, OnInit, Inject } from '@angular/core';
import { MAT_DIALOG_DATA } from '@angular/material';

import { [redacted] } from '../interfaces';

@Component({
  selector: 'app-dialog-[redacted]',
  templateUrl: './dialog-[redacted].component.html',
})
export class Dialog[redacted]Component implements OnInit {
  /**
   * Constructor of the class
   *
   * @param [redacted]
   */
  constructor(@Inject(MAT_DIALOG_DATA) public [redacted]: [redacted]) { }

  public ngOnInit() { }
}
```

Kuvio 6. Tarkemmat tiedot -dialogikomponentti

Tarkennettu haku -dialogi

Tarkennettu haku -dialogissa on hakulomake, joka sisältää kuusi hakuehtosyötettä. Haun tehdäkseen käyttäjän täytyy antaa hakuehto vähintään yhteen syötteeseen. Hakulomaketta alustettaessa sen validoijaksi asetetaan atLeastOne-metodi, joka tarkistaa, että ainakin yhteen syötteeseen on annettu hakuehto. Haku tehdään napauttamalla ”Hae”-painiketta, joka kutsuu tarkennettu haku -komponentin search-metodia. Metodi tarkistaa onko lomake validi, asettaa annetut hakuehdot hakuparametreihin ja sulkee dialogin välittäen hakuparametrit hakukomponenttiin. Jos lomake ei ole validi kutsuu search-metodi pikapalkkipalvelua, joka näyttää puutteellisista hakuehdoista kertovan virheilmoituksen.

Dialogin sulkemisen jälkeen hakukomponentin openFocusedSearchDialog-metodi (ks. kuvio 7) tarkistaa sisältääkö dialogilta tullut vastaus hakuparametrejä. Jos vastaus sisältää hakuparametrejä kutsutaan hakukomponentin searchY-metodia, joka lisää rajausehdot hakuparametreihin ja kutsuu hakupalvelun searchY-metodia välittäen hakuparametrit. Hakupalvelu lähettää hakuparametrien mukaisen pyynnön back-endiin, joka lähettää pyynnön edelleen ulkoiseen rajapintaan. Lopuksi ulkoisesta rajapinnasta palautuneet hakutulokset näytetään päänäkymässä listana.

Web-sovelluksen jokainen dialogin avaus -metodi sisältää kutsun renderöijään, jonka avulla päänäkymän body-elementtiin lisätään tyyliluokka, joka estää dialogin alle jäävän sisällön liikuttamisen liu'uttamalla.

```
/**
 * Open focused search dialog
 * After close execute search if response contains request
 */
public openFocusedSearchDialog() {
  this.renderer.addClass(document.body, 'no-scroll');
  this.dialog.open(DialogFocusedSearchComponent, {
    height: '100%',
    minWidth: '100%',
    autoFocus: false,
    backdropClass: 'dialog-backdrop',
    closeOnNavigation: true
  })
    .afterClosed()
    .subscribe(response => {
      if (response && response.request) {
        this.search[redacted](response.request);
      }
    });
}
```

Kuvio 7. Hakukomponentin openFocusedSearchDialog-metodi

6 Johtopäätökset

Tässä luvussa vastataan tutkimuskysymyksiin ja käydään läpi, kuinka hyvin kehitetty web-sovellus vastaa tutkimustuloksia ja asiakasyrityksen vaatimuksia.

Miten web-sovelluksen käyttöliittymä optimoidaan mobiilikäyttöön?

Ohjeita web-sovellusten mobiilioptimointiin oli hyvin vähän tarjolla. Lähes kaikki löydetyt ohjeet käsittelivät vain mobiilisovelluksia. Tämän perusteella mobiilisovelluksia varten tehtyjä käyttöliittymäohjeita kannattaa käyttää mobiilikäyttöön optimoidun web-sovelluksen kehittämiseen. Ohjeita hyödyntäessä täytyy kuitenkin huomioida mobiilisovelluksen ja web-sovelluksen erot. Web-sovelluksella on rajallisemmat mahdollisuudet hyödyntää mobiililaitteen antureita, laitepainikkeita ja muita toimintoja.

Lisäksi verkkoselaimen osoitepalkki rajoittaa näyttötilaa, joten käyttöliittymää varten sitä jää vähemmän. Web-sovelluksesta voi kuitenkin tehdä progressiivisen web-sovelluksen (engl. progressive web app), jolloin saadaan muun muassa ilmoitukset käyttöön, osoitepalkki piiloon ja verkkoyhteydetön käyttö mahdolliseksi.

Tutkimuksessa parhaaksi todetut ohjenuorat mobiilioptimoidun käyttöliittymän kehittämistä varten ovat:

- Yhdenmukaisuus
- Minimaalisuus
- Yhden tarkoituksen näkymät.

On erittäin tärkeää, että sovelluksen käyttöliittymä on yhdenmukainen. Sovelluksen käyttö on selkeämpää ja käyttäjä oppii nopeammin, kun käyttöliittymä on yhdenmukainen. Yhdenmukaisessa käyttöliittymässä näkymien ulkoasut vastaavat toisiaan, pääelementit sijaitsevat samassa paikassa ja samanlainen painike suorittaa samankaltaisen toiminnon sovelluksen jokaisessa näkymässä.

Minimaalinen käyttöliittymä auttaa käyttäjää hahmottamaan paremmin ja oppimaan sovelluksen käytön nopeammin. Mobiililaitteen näyttö ei saa olla täynnä sisältöä, vaan ainoastaan näkymän toiminnan kannalta olennainen sisältö tulee olla esillä. Vähemmän olennainen sisältö piilotetaan käyttäjän vuorovaikutuksen taakse, jolloin se on kuitenkin helposti saatavilla.

Yhden tarkoituksen näkymät helpottavat hahmottamista. Käyttäjä hämmentyy helposti, jos sama näkymä sisältää useita toisiinsa liittymättöimä toimintoja. Loogisinta on jakaa eri toiminnot omiksi näkymikseen, jolloin käyttäjän on helppo keskittyä yhteen toimintoon kerrallaan. Yhden tarkoituksen periaate on erittäin tärkeä varsinkin puhelimella, jolla näyttötila on hyvin vähissä, joten kaiken esillä olevan sisällön täytyy olla käytön kannalta olennaista. Tabletilla näkymät voivat näyttää työpöytämäisemmiltä, mutta käyttäjäkokemuksen täytyy edelleen tuntua mobiililta.

Kaikki tärkeimmät ohjenuorat korostavat helppoa hahmottamista ja käytön nopeaa oppimista, joten mobiilioptimoidun käyttöliittymän tulee olla helppo oppia ja käyttää. Tärkeimmät ohjenuorat ovat kuitenkin hyvin yleispäteviä eikä niiden pohjalta

pysty luomaan tarkkaa mielikuvaa hyvästä käyttöliittymästä. Käyttöliittymän hahmottaminen ja oppiminen nopeutuu, jos käyttäjällä on jo ennestään kokemusta vastaavanlaisesta. Mobiililaitteella web-sovellusta käyttävällä käyttäjällä on varmasti kokemusta mobiilisovelluksista, joten mobiilioptimoidun käyttöliittymän kannattaa muistuttaa mobiilisovellukselle tyypillistä käyttöliittymää. Käyttöliittymä, jossa yhdistyy yhdenmukaisuus, minimaalisuus, yhden tarkoituksen näkymät ja ennestään tuttu kokemus on hyvin käyttäjäystävällinen.

Mitkä ovat Angular-sovelluskehyksen parhaat kehityskäytänteet?

Angular korostaa tyyliohjeessaan erityisesti selkeää ja litteää rakennetta, helposti luettavaa koodia sekä toimintojen uudelleenkäytettävyyttä. Angular-sovelluksen selkeän rakenteen perustana toimii:

- Yhden säännön noudattaminen
- Yhdenmukaiset nimeämiskäytänteet
- Looginen hakemisto- ja tiedostorakenne.

Yhden säännön mukaan jokaisen moduulin, komponentin ynnä muun toiminnon tulee vastata vain yhdestä sille asetetusta tarkoituksesta. Esimerkiksi yksittäisen komponentin tulee vastata vain yhden näkymän ja sen logiikan toteuttamisesta. Samalla periaatteella yksittäisen tiedoston tulee sisältää vain yksi luokka, liittymä tai muu vastaava. Yhden säännön noudattaminen edistää toimintojen uudelleenkäytettävyyttä ja se antaa selkeän pohjan hakemisto- ja tiedostorakenteille.

Yhdenmukaiset nimeämiskäytänteet helpottavat Angular-sovelluksen kehittämistä ja sen rakenteen hahmottamista. Tiedostojen nimien tulee koostua kuvaus-, tyyppi- ja tiedostopääteosista. Tyyppiosan avulla voidaan nimeä vilkaisemalla päätellä tiedoston sisältämä toiminto. Kooditasolla nimeämiskäytänteiden täytyy erottaa eri tason toimijat. Esimerkiksi luokkien ja liittymien nimet tulee kirjoittaa ylemmällä kamelikirjoituksella ja muuttujien sekä metodien nimet tulee kirjoittaa alemmalla kamelikirjoituksella. Tällöin ne pystytään erottamaan toisistaan pelkän nimen kirjoitusasun perusteella.

Myös loogiset hakemisto- ja tiedostorakenteet helpottavat Angular-sovelluksen kehittämistä. Sovelluksen jokaiselle moduulille tulee tehdä oma hakemisto app-hakemistoon pois lukien päämoduuli, joka sijoitetaan app-hakemiston juureen. Moduulin sisältämien toimintojen tiedostot sijoitetaan moduulin omaan hakemistoon. Angular-sovelluksen hakemistorakenteen tulisi olla mahdollisimman matala, jotta tiedostopolut pysyvät lyhyinä ja tiedostot ovat helposti löydettävissä.

Kuinka toteutetaan mobiilikäyttöön optimoitu Angular-web-sovellus?

Hyvin suunniteltu ja toteutettu käyttöliittymä on ylivoimaisesti tärkein asia hyvän mobiilikäyttäjäkokemuksen kannalta. Käyttöliittymän tulee olla yhdenmukainen, minimaalinen ja helposti opittava. Käyttäjä oppii käyttämään sovellusta nopeammin, jos käyttöliittymä on tutun oloinen. Paras keino tähän on tehdä mobiilisovellukselta näyttävä käyttöliittymä.

Angular Materialin ulkoasukomponentit tukevat web-, mobiili- ja työpöytäalustoja, joten ne sopivat hyvin mobiilikäyttöön optimoidun web-sovelluksen käyttöliittymän perustaksi. Ulkoasukomponentit sopivat suoraan mobiililaitteen näytölle, joten mobiilikäyttöliittymän toteuttaminen sujuu luontevasti. Myös ulkoasukomponenttien konfiguraatioissa on huomioitu mobiililaitteet. Esimerkiksi päivämäärävalitsimen konfiguraatiossa on kosketuskäyttöliittymä vaihtoehto. Materialin käyttö säästää myös merkittävästi aikaa, kun kaikkea ei tarvitse tehdä itse.

Tutkimuksessa todetut Angularin parhaat käytänteet eivät välitä käytettävästä sovellusalustasta. Parhaita käytänteitä kannattaa kuitenkin noudattaa, jotta kehitettävän sovelluksen lopputulos on paras mahdollinen, olipa sovellusalusta mikä hyvänsä.

Web-sovellus

Mobiilikäyttöön optimoitu web-sovellus julkaistiin asiakasyrityksen testikäyttöön marraskuussa 2018. Rajallinen ryhmä käyttäjiä testaa aluksi sovellusta ja se viimeistellään käyttäjiltä saadun palautteen perusteella.

Web-sovelluksen kehityksessä käytettiin vahvasti hyödyksi tutkimuksen aikana löydettyä tietoa. Sovelluksen hakemisto- ja tiedostorakenteet ovat tutkimustulosten mukaisesti litteät ja tiedostopolut ovat vain sen pituisia, että tiedostorakenteesta saatiin selkeä. Sovelluksen päämoduuli, -reititysmoduuli ja -komponentti on sijoitettu

app-hakemistoon. Autentikointi-, kirjautumis-, haku- ja jaettumoduulit on sijoitettu app-hakemiston alle omiin hakemistoihinsa. Tiedostot on nimetty kolmeosaisesti käyttäen kuvaus-, tyyppi- ja tiedostopääteosia. Sovellus noudattaa yhden sääntöä, joten se on jaettu toimintojen mukaan loogisesti eri moduuleihin ja yksittäiset tiedostot sisältävät aina vain yhden luokan, liittymän tai muun vastaavan.

Myös käyttöliittymän kehittämisessä käytettiin hyödyksi tutkimuksessa löydettyjä tietoja. Web-sovelluksen käyttöliittymä muistuttaa vahvasti mobiilisovellusta, joten käyttäjien on helppo oppia käyttämään sitä ennestään tutun kokemuksen ansiosta. Käyttöliittymästä tehtiin yhdenmukainen Material-teeman ja Materialin ulkoasukomponenttien avulla. Teemaa on käytetty myös hyödyksi asiakasyrityksen brändin korostamiseen. Asiakasyrityksen tunnusväri on asetettu teeman pääpalettiin ja sitä käytetään esimerkiksi painikkeissa, syötteissä ja tekstissä. Käyttöliittymä on hyvin minimaalinen, sillä se ei sisällä mitään käyttötarkoituksen kannalta epäolennaista. Sovelluksen jokaisella näkymällä on vain yksi määritetty käyttötarkoitus, joten ne ovat selkeästi hahmotettavia. Puhelimen pienellä näytöllä sovelluksen käyttöliittymä on tiivis, mutta elementtien välillä on kuitenkin tarpeeksi tilaa, jotta virhepainalluksia ei synny. Tabletilla sovelluksen käyttöliittymä on täysin sama, mutta tilavampi.

7 Pohdinta

Opinnäytetyön tarkoitus oli tutkia, kuinka kehitetään mobiilikäyttöön optimoitu web-sovellus Angular-sovelluskehyksellä ja toteuttaa sellainen toimeksiantajan asiakasyrityksen vaatimuksien mukaisesti. Tutkimuksen aikana löydettyä teoriatietoa hyödynnettiin vahvasti web-sovelluksen toteuttamisessa.

Tutkimuksen teoriaosio ei ole erityisen syvällinen eikä se tuottanut uutta tietoa, mutta kehitystyönä toteutettu web-sovellus tarjoaa digitaalisen ratkaisun asiakasyrityksen ongelmaan. Teoriaosio kuitenkin kasaa mobiilikäyttöön optimoidun web-sovelluksen kehittämiseen tarvittavat parhaat käytänteet yhteen. Löydetty teoriatieto oli riittävän syvällistä, sillä sovelluksen toteuttaminen onnistui sen pohjalta. Lisäksi

teoriaosiolla oli iso merkitys opinnäytetyön tekijän ammatillisen kehittymisen kannalta, sillä aikaisempi osaaminen tutkittavista asioista oli vähäistä. Tutkimusta voidaan pitää onnistuneena, sillä asiakasyrityksen vaatimuksien mukainen web-sovellus saatiin toteutettua.

Tutkimuksen luotettavuus

Tutkimuksen teoriaosio pohjautuu blogikirjoituksiin, verkkoartikkeleihin ja Angularin virallisiin ohjeisiin. Kirjoja ei käytetty teoriaosiossa lähteenä, koska tuoreimpiin teoksiin oli hankala päästä käsiksi ja vanhemmat kirjat sisältävät runsaasti vanhentunutta tietoa. Angularista julkaistaan noin puolen vuoden välein uusi pääversio, joten hyödynnettävän tiedon täytyy olla tuoretta, jotta sitä voidaan pitää luotettavana. Myös mobiilikäyttöliittymäohjeet elävät hetkessä ja esille nousee jatkuvasti uusia trendejä. Teoriaosiossa käytetyt lähteet ovat erittäin tuoreita, sillä vain muutamat niistä ovat yli vuoden vanhoja. Blogikirjoituksien ja verkkoartikkeleiden kirjoittajien taustat tarkistettiin ja heiltä todettiin löytyvän vahvaa osaamista kertomistaan asioista. Teorian Angular-osassa on käytetty paljon Angularin virallisia ohjeita. Esimerkiksi ”Parhaat kehityskäytänteet”-kappale pohjautuu täysin Angularin viralliseen tyyliohjeeseen. Teorian ”Mobiilikäyttöön optimoitu web-sovellus”-luvussa olisi voitu käyttää useampia lähteitä luotettavuuden lisäämiseksi. Käytetyt lähteet ovat kuitenkin pitkälti samaa mieltä asioista eikä niiden välillä havaittu ristiriitoja.

Tutkimuksen teoriaosio on yleistettävissä, sillä se esittelee Angularin ja web-sovelluksen käyttöliittymän mobiilioptimoinnin parhaat käytänteet. Kehittämisosio keskittyy täysin kehitettyyn web-sovellukseen, joten suurin osa sen sisältämästä tiedosta ei ole yleistettävissä, vaan se on juuri tälle sovellukselle tapauskohtaista. Jotkin kehitysosion asiat, kuten autentikointi ja kirjautuminen voivat kuitenkin toimia samanlailla eri sovelluksissa, joten nämä kohdat ovat yleistettävissä. Tiedon hyödyntäjällä täytyy kuitenkin olla selkeä käsitys, kuinka hän haluaa sovelluksensa autentikoinnin ja kirjautumisen toimivan.

Mahdolliset jatkotutkimusaiheet

Tekemällä web-sovelluksesta progressiivisen web-sovelluksen, päästään lähemmäksi natiivin mobiilisovelluksen käyttäjäkokemusta. Angular 6 tarjoaa valmiit työkalut

web-sovelluksen muuntamiseen progressiiviseksi web-sovellukseksi. Jatkotutkimuksena voisi tutkia, mitkä ovat progressiivisen web-sovelluksen hyödyt ja milloin web-sovelluksesta kannattaa tehdä progressiivinen. Jatkotutkimuksessa voisi myös selvittää, kuinka web-sovelluksen muuntaminen progressiiviseksi web-sovellukseksi tapahtuu, miten se konfiguroidaan käyttötarkoitukseen sopivaksi, ja kuinka välimuistia voidaan hyödyntää sovelluksen verkkoyhteydettömään käyttöön.

Lähteet

Angular Material. N.d. Angular Material -ulkoasukomponenttikirjaston viralliset verkkosivut. Viitattu 21.10.2018. https://material.angular.io/.

Architecture overview. N.d. Angular-sovelluksen arkkitehtuurin dokumentaatio. Viitattu 22.8.2018. https://angular.io/guide/architecture.

Babich, N. 2018b. 10 Do's and Don'ts of Mobile UX Design. Verkkoartikkeli. Viitattu 31.8.2018. https://theblog.adobe.com/10-dos-donts-mobile-ux-design/.

Babich, N. 2018a. A Comprehensive Guide To Mobile App Design. Verkkoartikkeli. Viitattu 31.8.2018. https://www.smashingmagazine.com/2018/02/comprehensive-guide-to-mobile-app-design/.

Bodrov-Krukowski, I. 2018. Angular Introduction: What It Is, and Why You Should Use It. Verkkoartikkeli. Viitattu 10.8.2018. https://www.sitepoint.com/angular-introduction/.

Bouchefra, A. 2018. Creating UIs with Angular Material Design Components. Viitattu 17.8.2018. https://www.sitepoint.com/angular-material-design-components/.

Bouza, A. 2018. What is Digital Transformation, Digitalization, and Digitization. Verkkoartikkeli. Viitattu 21.10.2018. https://medium.com/api-product-management/what-is-digital-transformation-digitalization-and-digitization-c76277ffbdd6.

Classes. N.d. TypeScriptin virallinen dokumentaatio luokista. Viitattu 16.8.2018. https://www.typescriptlang.org/docs/handbook/classes.html.

Components. N.d. Angular Materialin dokumentaatio ulkoasukomponenteista. Viitattu 13.8.2018. https://material.angular.io/components/categories.

Customizing Angular Material component styles. N.d. Angular Materialin komponenttien tyylittelyohje. Viitattu 17.8.2018. https://material.angular.io/guide/customizing-component-styles.

Digitalisaatio. N.d. Kuvaus digitalisaatiosta. Viitattu 21.10.2018. https://www.protacon.com/digitalisaatio/.

Elbourn, J. 2018. A Component Dev Kit for Angular. Blogikirjoitus. Viitattu 13.8.2018. https://blog.angular.io/a-component-dev-kit-for-angular-9f06e3b4b3b4.

Fluin, S. 2018. Version 6 of Angular Now Available. Blogikirjoitus. Viitattu 13.8.2018. https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4.

Fluin, S. 2017. The Past, Present, and Future of the Angular CLI. Blogikirjoitus. Viitattu 26.8.2018. https://blog.angular.io/the-past-present-and-future-of-the-angular-cli-13cf55e455f8.

Hood, C. 2015. Mobile First Design: Why It is Great and Why It Sucks. Blogikirjoitus. Viitattu 18.8.2018. https://mayvendev.com/blog/mobilefirst.

Introduction to modules. N.d. Angularin virallinen perehdytys moduuleihin. Viitattu 26.8.2018. https://angular.io/guide/architecture-modules.

Introduction to services and dependency injection. N.d. Angularin virallinen perehdytys palveluihin ja riippuvuusinjektioon. Viitattu 26.8.2018. https://angular.io/guide/architecture-services.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas: Miten kirjoitan kehittämistutkimuksen vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä: Kehittämistutkimuksen kirjoittamisen käytännön opas. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Lease, D. 2018. TypeScript: What is it & when is it useful? Verkkoartikkeli. Viitattu 16.8.2018. https://medium.com/front-end-hacking/typescript-what-is-it-when-is-it-useful-c4c41b5c4ae7.

Parsy, V. 2018. Typescript : class vs interface. Verkkoartikkeli. Viitattu 16.8.2018. https://medium.com/front-end-hacking/typescript-class-vs-interface-99c0ae1c2136.

So, Y. 2017.Designing for Mobile Apps: Overall Principles, Common Patterns, and Interface Guidelines. Verkkoartikkeli. Viitattu 28.8.2018. https://medium.com/blueprint-by-intuit/native-mobile-app-design-overall-principles-and-common-patterns-26edee8ced10.

Style Guide. N.d. Angularin virallinen tyyliohje. Viitattu 7.10.2018. https://angular.io/guide/styleguide.

Theming your Angular Material app. N.d. Angular Materialin virallinen teemoitteluohje. Viitattu 13.8.2018. https://material.angular.io/guide/theming.

Van de Moere, J. 2018. An Introduction to Component Routing with Angular Router. Verkkoartikkeli. Viitattu 27.8.2018. https://www.sitepoint.com/component-routing-angular-router/.

Van de Moere, J. 2018. The Ultimate Angular CLI Reference Guide. Verkkoartikkeli. Viitattu 10.8.2018. https://www.sitepoint.com/ultimate-angular-cli-reference/.

Vilkka, H. 2015. Tutki ja kehitä. Juva: Bookwell.