LAB University of Applied Sciences

Verkkosovelluksen kehittäminen musiikki-duolle

Case: DNG

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
2024
Joona Lehtoniemi

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Joona Lehtoniemi	Opinnäytetyö, AMK	2024
	Sivumäärä	
	36	

Työn nimi

Verkkosovelluksen kehittäminen musiikkiduolle

Case: DNG

Tutkinto ja koulutusala

Insinööri (AMK), Tieto- ja viestintätekniikka

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

DNG

Tiivistelmä

Opinnäytetyön tavoitteena oli kehittää DNG-nimiselle musiikkiduolle verkkosovellus, joka tarjoaa responsiivisen ja käyttäjäystävällisen käyttöliittymän. Sovelluksen tarkoituksena oli parantaa duon näkyvyyttä, sekä helpottaa ja lisätä esiintymisvarauksien tekemistä.

Verkkosovellus toteutettiin käyttäen Angular-kehystä, sekä lisäksi Angular materials -kirjastoa. Sovelluksen päätoiminnallisuuksiin kuuluivat varauslomake sekä mediakirjasto kuvien ja videoiden tallentamista ja esittämistä varten. Lisäksi työssä pyrittiin toteuttamaan ylläpitäjän sisäänkirjautumis- ja hallintanäkymä mediakirjaston sisällön ylläpitämiseksi sekä contact- ja about -sivut.

Saavutetut tulokset osoittivat, että Angular tarjoaa tehokkaat valmiudet responsiivisen ja helposti skaalattavan frontend-sovelluksen kehittämiseen. Toteutettu sovellus täytti alussa määritellyt vaatimukset.

Asiasanat

Angular, Webkehittäminen, Frontend-kehittäminen

Abstract

Author(s)	Type of Publication	Published
Joona Lehtoniemi	Thesis, UAS	2024
	Number of Pages	
	36	

Title of Publication

Development of a web application for a music duo

Case: DNG

Degree, Field of Study

Bachelor of Engineering, Information and Communications Technology

Organisation of the client (if the thesis work is commissioned by another party)

DNG

Abstract

The objective of the thesis was to develop a web application for a music duo named DNG, offering a responsive and user-friendly interface. The purpose of the application was to enhance the duo's visibility, as well as streamline their performance bookings.

The web application was implemented using the Angular framework, along with the Angular Materials library. The main functionalities of the application included a booking form and a media library for storing and presenting pictures and videos. Additionally, the thesis aimed to implement an administrator login and management view to maintain the content of the media library, as well as the contact and about pages.

The achieved results demonstrated that Angular provides effective capabilities for developing a responsive and easily scalable frontend application. The implemented application met the initially defined requirements.

Keywords

Angular, Web development, Frontend development

Sisällys

1 Johdanto

Sosiaalisen median ja sähköisten lähteiden lisääntyminen musiikkialalla on muuttanut muusikkojen ja yleisön välistä vuorovaikutusta, ja samalla koventanut kilpailua markkinoilla. Musiikkiduo DNG pyrkii vahvistamaan digitaalista läsnäoloaan ja parantamaan markkinointiaan sekä näkyvyyttään, mikä toimi motivaationa ryhtyä kehittämään frontend-verkkosovellusta.

Työn toimeksiantajana on DNG, joka on kahden muusikon muodostama musiikkiduo. Joel Pöllänen, duon solistikitaristi, on itseoppinut muusikko, joka on osallistunut aktiivisesti bänditoimintaan ja soittanut monia soittimia lapsuudesta asti. Toinen duon jäsen, Taneli Kleemola, on ammattimuusikko, joka vastaa duon haitaristin roolista. Duo on esiintyneet yhdessä vuodesta 2013. Toimeksiantona oli kehittää verkkosovellus, joka mahdollistaa esimerkiksi varauksien tekemisen ja yhteydenoton ja jonka on tarkoitus palvella mahdollisimman suurta yleisöä riippumatta käytetystä laitteesta.

Opinnäytetyön tavoitteena on kehittää toimeksiannon mukaan frontend-verkkosovellus käyttämällä Angular-kehystä, sekä esimerkiksi Angular Material -kirjastoa. Verkkosovelluksen tarkoituksena on parantaa näkyvyyttä ja helpottaa esimerkiksi esiintymisvarausten tekemistä. Verkkosovelluksen tulee myös olla käyttäjäystävällinen ja responsiivinen, jotta se palvelee mahdollisimman suurta yleisöä. Kehittäminen kattaa suunnittelun ja toteutuksen prosessin.

Työ kattaa keskeisten toiminnallisuuksien kehittämisen, kuten varauslomakkeen, mediakirjaston, ylläpitäjän käyttöliittymän sekä muut sovelluksen näkymät. Sovelluksen tulee myös olla responsiivinen, jotta se palvelee mobiililaitteiden käyttäjiä. Toteutus ei kata backend-kehittämistä eikä tuotantovaihetta, vaan keskittyy nimenomaan frontend-kehittämiseen. Tavoitteena on toteuttaa vankka ja käyttäjäystävällinen ratkaisu, joka vastaa DNG:n tarpeita ja toiveita niin visuaalisesti kun toiminnallisestikin.

2 Tekninen ja visuaalinen perusta

2.1 TypeScript

TypeScript on vahvasti tyypitetty ohjelmointikieli, joka on rakennettu JavaScriptin pohjalta. Tarkemmin sanottuna TypeScript on JavaScriptin yläjoukko eli superset. TypeScript tuo mukanaan staattisen tyypityksen, joka mahdollistaa ajonaikaisten ongelmien, sekä muuttujien tyyppeihin liittyvien virheiden löytymisen kehityksen aikaisessa vaiheessa. TypeScriptiä ei ajeta sellaisenaan, vaan se transpiloidaan JavaScript koodiksi joko ajon aikana tai ennen sen käyttämistä JavaScript ohjelmissa tai selain ympäristöissä. (Jansen ym. 2016, 2–4.)

Etuna JavaScriptiin nähden vahvan staattisen tyypityksen lisäksi ovat TypeScriptin tarjoamat rakenteelliset luokat. TypeScript tarjoaa kehittäjälle mahdollisuuden hyödyntää class-, interface- ja module-rakenteita, jotka mahdollistavat vahvemman tyypityksen lisäksi myös parantuneen koodin organisoinnin perinteiseen JavaScriptiin verrattuna. (Jansen ym. 2016, 7–9.)

Jansen ym. (2016, 4–6) mukaan TypeScript edistää olio-ohjelmoinnin mukaista koodin organisointia, joka parantaa tiimien yhteistyötä ja ylläpitää ohjelman ja koodin skaalautuvuutta ja ylläpidettävyyttä, sekä mahdollistaa esimerkiksi luokkien periyttämisen. TypeScript mahdollistaa tuomiensa etujen hyödyntämisen vain kehityksen aikana. Kun koodi transpiloidaan JavaScriptiksi, TypeScriptin tarjoamat rakenteet, tyypitykset ja tyyppitarkistukset poistuvat koodista.

2.2 CSS

CSS eli Cascading Style Sheets on kieli, joka määrittelee dokumentin visuaalisen ilmeen ja tyylisisällön. CSS:ää käytetään yleensä yhdessä HTML-merkintäkielen kanssa, mutta sitä voidaan hyödyntää myös SVG- ja XML-dokumenttien yhteydessä. (Mozilla Developer Network 2024b.)

CSS:n avulla voidaan vaikuttaa muun muassa tekstin ja taustan väriin, fontin kokoon sekä elementtien muotoon. Lisäksi CSS mahdollistaa myös rakenteiden, kuvien ja säiliöiden koon ja muodon muokkaamisen. CSS tarjoaa myös mahdollisuuksia animaatioiden toteuttamiseen käyttäen esimerkiksi kielen transition- ja transform-ominaisuuksia, sekä keyframe-arvoja, jotka määrittelevät animaation vaiheet eri ajanhetkillä. (Mozilla Developer Network 2024b.)

Edellä mainittuja ominaisuuksia hyödyntämällä voidaan esimerkiksi liu'uttaa elementti näkyviin tai muuttaa dokumentin sisällön väriä ja muotoa tietyn ajan kuluttua näkymän latautumisesta. (Mozilla Developer Network 2024b.)

Flexbox

Flexbox on CSS:n tarjoama yksiulotteinen layout eli asettelumenetelmä, jonka avulla voidaan järjestää elementtejä esimerkiksi riveihin ja sarakkeisiin sitä käyttävässä säiliössä. Flexboxia käyttävät elementit ovat joustavia, ne sekä pyrkivät täyttämään ympäröivän säiliönsä että kutistuvat tarpeen mukaan, mikä parantaa esimerkiksi flexboxia käyttävän verkkosivun responsiivisuutta. Flexboxia käyttämällä on myös mahdollista helposti keskittää sisältöä sekä pysty että leveys suunnassa suhteessa sitä ympäröivään säiliöön. (Mozilla Developer Network 2024a.)

Coyierin (2022) mukaan yleisimpiä flexboxin hyödyntämiä ominaisuuksia ovat display:flex, flex-direction, justify-content, sekä align-items. Edellä mainituista ensimmäisellä säiliö asetetaan noudattamaan flexbox asettelumenetelmää. Flex-direction määrittää, missä suunnassa sisältö joustaa säiliössä, vaihtoehtoiset arvot tähän ovat joko rivi (engl. row) tai sarake (engl. column). Justify-content ja align-items arvot määrittävät sisällön keskitykseen liittyviä ominaisuuksia, justify-content ja align-items vaikuttavat sisältöön suhteessa pääakseliin, joka määräytyy direction-arvon perusteella. Flexboxilla on mahdollista vaikuttaa myös muihin sisällön ominaisuuksiin, kuten elementtien välisiin väleihin ja elementtien kokojen suhteisiin toisiinsa.

Media queryt ja responsiivisuus

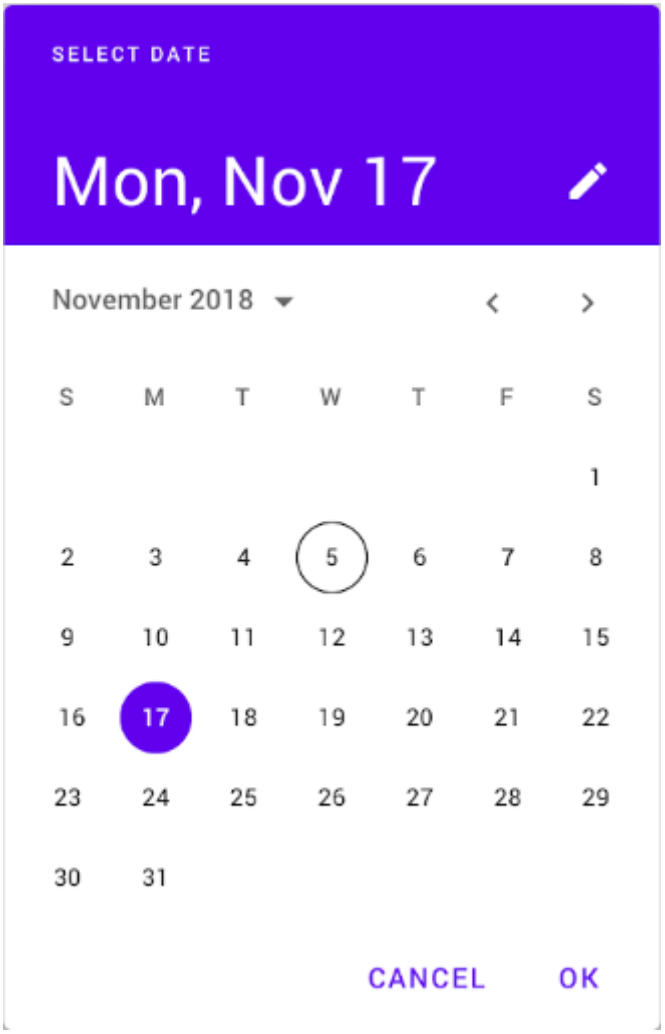
Media queryillä eli mediakyselyillä on mahdollista vaikuttaa CSS ominaisuuksiin riippuen käytettävän laitteen ominaisuuksista. Esimerkiksi käytettävän laitteen näytön leveyden pikseleinä, kuvasuhteen tai näytön orientaation perusteella voidaan rajoittaa näytettävän sisällön kokoa suhteessa näyttöpäätteeseen, joka mahdollistaa esimerkiksi mobiililaitteella sisällön sujuvan selaamisen. Esimerkki mediakyselyn perussyntaksista, jossa CSS-koodin toteutumisehdoksi on määritelty näytön orientaatio, on nähtävissä kuvassa 1. (Frain 2020, 57–62.)

```
@media screen and (orientation: portrait) {
    /* styles here */
}
```

Kuva 1. Esimerkki mediakysely syntaksista (Frain 2020)

2.3 Google Material Design 2

Google Material Design 2, joka yleisemmin tunnetaan nimellä Material Design, on Googlen suunnittelema tyylisäännöstö. Se sisältää suunnitteluperiaatteita ja vakiintuneita komponentteja sekä tyyliratkaisuja. Material Design kattaa myös suuren määrän yhteensopivia komponentteja, teemoja, väriskaaloja sekä esimerkiksi ikoneita ja tekstityylejä. Se kattaa sekä web- että mobiilikehityksessä käytettäviä komponentteja. Yksi tunnettu esimerkki komponenteista on lomakkeissa käytettävät kalenterinäkymät päivämäärän valintaa varten. Esimerkki kalenterinäkymä komponentista on nähtävissä kuvassa 2. (Material Design 2 a.)



Kuva 2. Date picker -komponentti (Material Design 2 b)

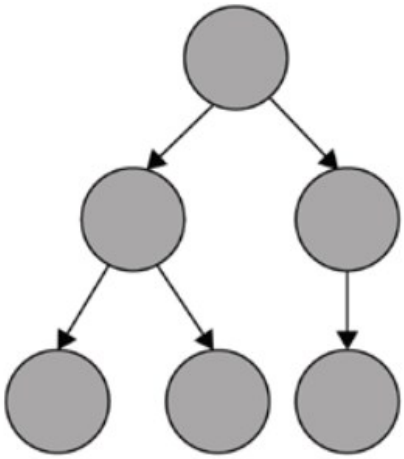
Material Designin tarkoituksena on tarjota kehittäjille tarkasti määriteltyjä ohjeita ja rajoituksia. Noudattamalla näitä ohjeita kehitystiimit kykenevät luomaan sekä visuaalisesti että toiminnallisesti eheitä ja käyttäjäystävällisiä käyttöliittymiä niin selain- kuin mobiilialustoille. (Material Design 2 a.)

3 Angular

3.1 Yleiskatsaus

Angular on avoimen lähdekoodin kehitysalusta, jonka ohjelmointikielenä toimii TypeScript. Se on komponenttilähtöinen kehys, joka tarjoaa web-kehitykseen laajan valikoiman sisäänrakennettuja toimintoja ja ominaisuuksia. Esimerkkinä ominaisuuksista on Angularin moduulit, komponentit ja palvelut, joita käsitellään myöhemmin tässä opinnäytetyössä. (Bampakos & Deeleman 2020, 7–10.)

Angularin komponenttilähtöisyyden ansiosta se on modulaarinen kehitysalusta, joka skaalautuu hyvin sekä suuriin että pieniin projekteihin. Sen pääasiallinen käyttötarkoitus on yksisivuisten sovellusten (SPA, Single Page Application) kehityksessä. Angular-sovellusten arkkitehtuuri koostuu komponenteista moduulien sisällä, jotka on järjestetty puurakenteeseen pää- ja lapsikomponenttien perusteella. Esimerkki puurakenteesta on nähtävissä kuviossa 1. (Bampakos & Deeleman 2020, 7–10.)



Kuvio 1. Komponenttiarkkitehtuuri (Bampakos & Deeleman 2020)

3.2 Angular CLI

Angular CLI on Angularin komentorivityökalu, joka vaatii toimiakseen yhteensopivan version Node.js:stä sekä NPM:stä (Node Package Manager). Angular CLI:tä hyödyntämällä voidaan suorittaa erilaisia Angular-toimintoja, joiden syntaksi on ng komento. Yleisimmin käytetyt komennot ovat serve, build, test, generate, add sekä new. (Bampakos 2021, 5–6.)

Uuden Angular-projektin alustaminen komentorivin kautta tapahtuu kuvan 3 osoittamalla tavalla.

Kuva 3. ng new komento

Ng new -komento luo Angular-ympäristön, joka sisältää minimalistisen Angular-projektin, mikä toimii sovelluskehityksen lähtökohtana. (Bampakos 2021, 5–6)

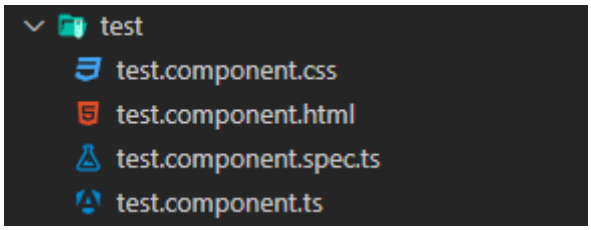
Angular CLI:tä käytetään tiiviisti läpi koko ohjelman kehityskaaren, sillä komentoriviä hyödyntämällä projektiin voidaan lisätä muun muassa moduuleja, komponentteja ja palveluita. Komentorivin kautta voidaan myös käynnistää lokaali toimintaympäristö Angular-ohjelmalle ng serve -komennolla, joka käynnistää sisäänrakennetun web-palvelimen ja näyttää ohjelman sisällön alkuperäisillä asetuksilla osoitteessa localhost:4200. Kun käyttäjä on ajanut ng serve -komennon, komentorivikäyttöliittymä seuraa aktiivisesti koodiin tehtyjä muutoksia ja päivittää reaaliaikaisesti selaimen näkymän, kun ohjelman muutokset tallennetaan. (Bampakos 2021, 11–13.)

Bampakos (2021, 19–21) mukaan test -komento suorittaa Angular-ohjelman sisäiset testit, jotka on määritelty komponenttien testitiedostossa. Generate-komento, jolle annetaan parametreiksi luotavan elementin tai komponentin tyyppi, kuten esimerkiksi service, luo Angular-ohjelmaan uuden palvelun. Add-komento lisää Angular-ohjelmaan kolmannen osapuolen kirjaston, joka on yhteensopiva Angular-ohjelman kanssa.

3.3 Komponentit

Angular-komponentti on keskeisin elementti Angular-sovelluskehityksessä. Komponentteja hyödyntämällä on mahdollista luoda uudelleenkäytettäviä ja hyvin organisoituja osia ja kokonaisuuksia. Angular-komponentti koostuu neljästä tiedostosta, jotka yhdessä muodostavat komponentin. Nämä ovat komponenttiluokka (.ts), komponentin mallitiedosto (.html), komponentin tyylitiedosto (.css) ja komponentin yksikkötestitiedosto (.spec.ts). (Angular.Io 2023b.)

Ainoastaan komponenttiluokka, eli TypeScript-tiedosto, on pakollinen, mutta tehokkaan ja toiminnallisen komponentin rakentamiseksi suositellaan kaikkien neljän osan käyttöä. Ng generate -komento Angular CLI:ssä luo komponenttikansion, joka sisältää kaikki komponentin osat. Esimerkki generoidusta komponentista ja sen osien nimeämiskäytännöistä on nähtävissä kuvassa 4.



Kuva 4. Angular komponentin kansiorakenne

Komponentin on oltava osana Angular Moduulia, moduulit esitellään tämän työn myöhäisemmässä vaiheessa. (Bampakos & Deeleman 2020, 66–69.)

3.3.1 Komponenttiluokka

Komponenttiluokka on TypeScript tiedosto, jonka tiedostopääte on .ts. Se on perimmiltään TypeScriptin luokka, joka on merkitty Angularin @Component dekoraattorilla, jolloin Angular-kääntäjä käsittelee sitä Angular komponenttina. Dekoraattori sisältää myös valitsimen (engl. selector), sekä templateUrl ja styleUrl määritykset. Esimerkki dekoraattorista ja sen sisältämistä kentistä on nähtävissä kuvassa 5.

```
5  @Component({
6      selector: 'app-test',
7      templateUrl: './test.component.html',
8      styleUrls: ['./test.component.css'],
9  })
10 export class TestComponent {
11     @Input() testInput: string = ''
12     @Output() testOutput = new EventEmitter<string>()
```

Kuva 5. Komponentin dekoraattori ja alustetut muuttujat

Kuvan 5 rivillä 4 on nähtävissä komponentin valitsin, jonka avulla Angular-sovellus käyttää kyseistä komponenttia. Vastaavasti rivit 5 ja 6 sitovat komponentin ja siihen kuuluvat HTML ja CSS tiedostot toisiinsa. (Bampakos & Deeleman 2020, 69.)

Komponenttiluokka sisältää komponentin logiikan lisäksi myös sen muuttujat, metodit, konstruktorin sekä elinkaari metodit. Komponenttiluokan metodit ja muuttujat noudattavat TypeScriptin määrityksiä ja syntaksia. (Angular 2023b.)

Angular-sovelluksen erityispiirteinä on, että muuttujille voidaan antaa Input-dekoraattori, joka määrittelee, että muuttujan arvo tulee sen ylemmältä (engl. parent) komponentilta. Input-dekoraattorista yhdistettynä muuttujaan on nähtävissä esimerkki kuvan 5 rivillä 11. Vastaavasti output-dekoraattori yhdistettynä EventEmitteriin mahdollistaa lapsikomponenttien tapahtumien välittämisen ylemmälle komponentille, ja tästä esimerkki on nähtävissä kuvan

5 rivillä 12. Ylempi komponentti voi kuunnella EventEmitterin lähettämää tapahtumaa, ja esimerkki tapahtuman kuunteluun käytetystä syntaksista on nähtävissä kuvassa 6.

```
<app-test-inner> (testOutput)=handleEvent($event) </app-test-inner>
```

Kuva 6. Tapahtuman kuuntelu esimerkki

Edellä mainitut mekanismit mahdollistavat suoraviivaisen kommunikaation pää- ja alikomponenttien välillä. Vaihtoehtoinen tapa välittää dataa komponenttien välillä on myöhemmin tässä työssä käsiteltävät palvelut(service). (Bampakos & Deeleman 2020, 69.)

Konstruktori

Srivastava (2023) kertoo artikkelissaan konstruktorin olevan komponenttiluokan metodi, joka suoritetaan komponentin initialisoinnin yhteydessä, mutta ennen komponentin elinkaarimetodeja. Angular-sovelluksissa konstruktoria käytetään yleisimmin riippuvuuksien, kuten palveluiden, injektoimiseen. Palvelun injektointi konstruktorissa tapahtuu seuraavan kuvan (kuva 7) esittämällä tavalla. Konstruktorissa voidaan lisäksi määrittää komponentin muuttujille alkuarvoja.

```
constructor(private testService:TestService){}
```

Kuva 7. Komponentin konstruktori

Elinkaarimetodit

Elinkaarimetodit ovat Angularin (2022a) tarjoamia metodeja, jotka suoritetaan tiettyinä ajankohtina komponentin elinkaaren aikana. Yleisimpiä esimerkkejä elinkaarimetodeista ovat ngOnInit, ngOnDestroy ja ngOnChanges. (Angular 2022a.)

NgOnInit suoritetaan juuri ennen komponentin näkymän ja sen lapsikomponenttien initialisointia, jonka ansiosta se sopii hyvin datan latausoperaatioille ja API-kutsuille. NgOnChanges puolestaan reagoi, kun komponentille annettu yksi tai useampi syöte eli input-dekoraattorilla merkitty arvo muuttuu, ja se mahdollistaa määritellyn toiminnallisuuden suorittamisen vastauksena muutoksiin. (Angular 2022a.)

NgOnDestroy suoritetaan, kun komponentti poistetaan ja tuhotaan Angularin toimesta. Se tarjoaa mahdollisuuden suorittaa siivousoperaatioita, kuten tilausten peruminen tai komponentin tilan nollaaminen lähtötilanteeseen. Lisäksi Angular sisältää myös muita elinkaarimetodeja, kuten ngDoCheck, ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, sekä ngAfterViewChecked, jotka tarjoavat kehittäjille enemmän mahdollisuuksia vaikuttaa komponentin toimintaan suhteessa sen elinkaareen. (Angular 2022a.)

3.3.2 Komponentin mallitiedosto

Komponentin mallitiedosto, eli template, on HTML-dokumentti, jonka perusteella Angular renderöi komponentin osaksi käyttöliittymää. Mallitiedosto noudattaa normaalia HTML-syntaksia, mutta se pystyy hyödyntämään Angularin rakenteellisia direktiivejä, kuten *ngFor ja *ngIf, tietojen dynaamiseen esittämiseen. Direktiivien käyttöä käsitellään laajemmin myöhemmin tässä työssä. (Angular 2023c.)

Mallitiedosto tukee myös yksi- ja kaksisuuntaista datansidontaa, mikä mahdollistaa komponenttiluokassa määriteltyjen muuttujien arvojen dynaamisen esittämisen käyttöliittymässä. Esimerkkejä datasidonnassa käytetyistä syntakseista on nähtävissä kuvassa 8.

```
1  {{ testVariable }}
2  <input [(ngModel)]="testVariable" />
```

Kuva 8. Datasidonnan syntaksit

Yksisuuntainen datansidonta käyttää kuvan 8 rivin 1 esittämää syntaksia muuttujan arvon renderöimiseen. Kaksisuuntainen datansidonta, toteutetaan yleisimmin kuvan 8 rivin 2 esittämällä ngModel-syntaksilla, joka on osa Angularin Forms-moduulia ja yleisesti käytetty käyttäjän syötettä vaativissa elementeissä, kuten tekstikentissä. (Angular 2023c.)

Komponenttiluokassa on mahdollista määritellä myös inline malli, joka korvaa erillisen .html tiedoston. Tällöin HTML-koodi sijoitetaan suoraan komponenttiluokan @Component dekoraattorin sisään, mikä voi tehdä komponentin rakenteesta tiiviimmän ja helpommin hallittavan. (Angular 2023c.)

3.3.3 Komponentin tyylitiedosto

Komponentin tyylitiedosto, eli tyypillisesti CSS-tiedosto, noudattaa CSS:n perussyntaksia. Tyylitiedostossa voidaan määritellä tyylejä eri CSS-valitsimille ja mediakyselyille, joka puolestaan mahdollistaa persoonallisen ja responsiivisen ulkoasun komponentille. Angularissa tyylitiedostot ovat komponenttikohtaisia, mikä tarkoittaa, että määritetyt tyylit vaikuttavat vain kyseisen komponentin ulkoasuun edistäen kapselointia ja vähentäen tyylimäärittelyn ristiriitoja ja päällekkäisyyksiä. (Angular 2022b.)

Komponenttiluokassa on mahdollista määritellä myös inline tyylitiedosto, joka korvaa erillisen CSS-tiedoston, ja tästä esimerkki on nähtävissä kuvassa 9.

```
@Component({
    selector: 'app-test',
    templateUrl: './test.component.html',
    styles: [
        `
            h1 {
                color: blue;
            }
            .test-input {
                border: 2px solid #888;
                margin-bottom: 10px;
                padding: 5px;
            }
        `,
    ],
})
```

Kuva 9. Inline tyylimäärittely

Tässä tapauksessa komponentin tyylimäärittelyt sijoitetaan kuvan 9 mukaisesti suoraan komponenttiluokan @Component-dekoraattorin styles-ominaisuuden yhteyteen. (Angular 2022b.)

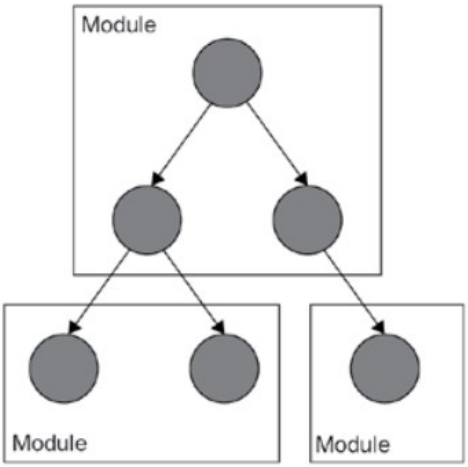
3.3.4 Komponentin yksikkötestitiedosto

Komponentin yksikkötestitiedosto on tiedosto, joka sisältää määritellyt testit kyseiselle komponentille. Tämän tiedoston tarkoituksena on testata, että kyseisen komponentin osat toimivat odotetusti ja ovat yhteensopivia keskenään. Angular-komponentin yksikkötestien tulisi keskittyä nimenomaan kyseisen komponentin toiminnallisuuden varmistamiseen. Tehokkaat yksikkötestit kattavat komponentin tärkeimmät toiminnallisuudet ja vuorovaikutukset esimerkiksi elinkaarimetodien kanssa. Angular kehityksessä yksikkötestit suoritetaan ’ng test’-komennolla. (Angular 2022c.)

3.4 Moduulit

Angular-moduuli, eli ngModule, on TypeScript-luokka, jonka dekoraattorina toimii @NgModule. Kyseisen dekoraattorin sisältämä metadata määrittelee moduulin sisällön ja moduuli toimii modulaarisena säiliönä sen sisältämille osille. Kuviossa 2 on nähtävillä graafinen esitys moduulin rakenteesta. Declarations-taulukossa määritellään moduulin sisältämät komponentit, palvelut, direktiivit ja putket, jotka voidaan asettaa koko Angular-ohjelman saataville hyödyntämällä dekoraattorin exports-taulukkoa. Imports-taulukon avulla moduuliin voidaan tuoda muita Angular-moduuleja, käyttäjän luomia moduuleja ja kolmannen osapuolen

moduuleja mahdollistaen modulaarisen ja järjestäytyneen ohjelmistorakenteen. (Angular 2022d.)



Kuvio 2. Komponenttien muodostaminen moduuleiksi (Bampakos & Deeleman 2020)

Kun Angular CLI:llä luodaan uusi sovellus ng new-komennolla, luo CLI samalla juurimoduulin, joka on nimeltään AppModule. Angular tarjoaa myös sisäänrakennettuja moduuleja, kuten FormsModule,HttpClientModule ja RouterModule, jotka tuovat sovelluksen käyttöön lisätoiminnallisuuksia, kuten lomakkeiden käsittelyn, HTTP-pyyntöjen suorittamisen ja Angular-ohjelman sisäisen reitityksen. Kehittäjät voivat myös luoda omia feature-moduuleita, mikä edistää sovelluksen koodin organisointia, kapselointia ja uudelleenkäytettävyyttä. (Angular 2022d.)

3.4.1 Juurimoduuli

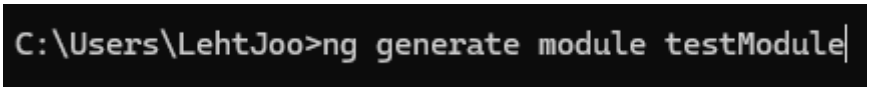
Juurimoduuli, joka tunnetaan myös nimellä AppModule, toimii Angular-sovelluksen lähtöpisteenä ja keskeisimpänä moduulina. Kyseinen moduuli on ainoa pakollinen moduuli Angular-sovelluksessa ja vastaa sovelluksen alkuperäisen rakenteen määrittelemisestä, sekä keskeisten komponenttien ja palveluiden tuomiseen ja alustamisesta. Juurimoduulissa importoidut palvelut, komponentit, putket ja direktiivit sekä moduulit ovat koko Angular-ohjelman saatavilla, jos ne löytyvät juurimoduulin dekoraattorin exports-taulukosta. Tämä mahdollistaa yleisten moduuleiden, komponenttien ja palveluiden uudelleenkäytön, mikä puolestaan tehostaa modulaarisuutta ja ylläpidettävyyttä. (Angular 2022d.)

Juurimoduulin providers-taulukossa määritellyt palvelut ovat ns. singleton-palveluita, eli palveluita, joita sovelluksessa saa esiintyä vain kerran. Providers-taulukon palvelut ovat koko sovelluksen laajuisesti saatavilla. Esimerkkinä singleton-palveluista on jo aiemmin mainittu HttpClient, joka vastaa HTTP-pyyntöjen välittämisestä ja käsittelystä. Juurimoduulin

@NgModule-dekoraattorissa sijaitseva bootstrap-taulukko sisältää Angular-sovelluksen juurikomponentin, eli komponentin, joka injektoidaan index.html-tiedostoon, kun Angular-sovellus suoritetaan. (Angular 2023d.)

3.4.2 Käyttäjän luomat moduulit

Angular-sovelluksessa on mahdollista luoda uusia moduuleja, jotka auttavat rajaamaan ja organisoimaan komponentteja ja koodia, esimerkiksi käyttötarkoitusten perusteella. Moduulien luominen tapahtuu Angular CLI:n generate-komennolla, josta esimerkki on nähtävissä kuvassa 10.



Kuva 10. Esimerkki moduulin generointikomennosta

Käyttäjän luomat moduulit tulee importoida juurimoduuliin, jotta Angular-ohjelmalla on pääsy niihin. Kun moduuli on importoitu juurimoduuliin, sen sisältämien komponenttien, palveluiden tai muiden elementtien käyttäminen on mahdollista juurimoduulin declarations-taulukon sisältämien komponenttien sisällä. (Angular 2022e.)

Käytetyimpiä moduulityyppejä käyttäjän luomille moduuleilla ovat feature-, core- ja shared-moduulit. Feature moduulit kattavat kehittäjän toteuttaman toiminnallisuuden ja siihen liittyvät tiedostot. Core-moduuli sisältää singleton-komponentit ja palvelut, joita ovat esimerkiksi verkkosivuilta yleensä esiintyviä header ja footer-komponentteja. Shared-moduuli sisältää yleisiä komponentteja ja palveluita, joita hyödynnetään useassa eri ohjelman osassa, kuten toistuvat rakenteet ja elementit. (Angular 2022f.)

3.5 Palvelut

Palvelut (engl. service) ovat oleellinen osa Angularin-ohjelmiston logiikkaa. Ne ovat yleensä tarkasti rajattuja luokkia, jotka sisältävät vain tiettyyn komponenttiin tai toiminnallisuuteen liittyviä ominaisuuksia. Ne eivät sisällä erillistä käyttöliittymää. Palveluiden avulla on mahdollista keskittää komponentin tai toiminnallisuuden logiikkaan liittyvä koodi yhteen palveluluokkaan, joka toteuttaa kapseloinnin ja parantaa koodin modulaarisuutta. (Angular 2022g.)

Palvelu tulee Angular-ohjelman käytettäväksi, kun kyseinen palvelu merkitään @Injectable-dekoraattorilla. Dekoraattorin metadatassa määritetään provideIn-kenttä, jonka oletusarvo on root. Silloin kyseinen palvelu on kaikkien juurimoduulin alaisten komponenttien tavoitettavissa. Komponentit voivat injektoida palvelun komponenttiluokan konstruktorissa, joka mahdollistaa palvelun metodien ja muuttujien hyödyntämisen. (Angular 2022g.)

Palveluita on mahdollista injektoida myös toisiin palveluihin, jolloin injektointi tapahtuu vastaavasti palvelun konstruktorissa. Palvelut toimivat vaihtoehtoisena tapana välittää dataa komponenttien välillä, aiemmin käsiteltyjen @input()- ja @output()-dekoraattoreiden lisäksi. (Angular 2022g.)

Palveluita luodaan Angular CLI:n generate-komennolla, josta esimerkki on nähtävissä kuvassa 11.

```
C:\Users\LehtJoo>ng generate service test
```

Kuva 11. Esimerkki palvelun generointikomennosta

Palveluita hyödynnetään usein kolmannen osapuolen API-kutsujen säilömisessä ja toteuttamisessa, autentikoinnissa sekä jo aiemmin mainitussa datan hallinnassa. (Angular 2022g.)

3.6 Direktiivit ja putket

Direktiivit ja putket (engl. directive and pipe) ovat Angular-ohjelmaan sisäänrakennettuja tai käyttäjän luomia ominaisuuksia, joilla on mahdollista vaikuttaa ohjelman ulkonäköön tai rakenteeseen. Direktiiveillä on mahdollista vaikuttaa HTML elementtien käyttäytymiseen ja toimintaan ja putkilla pystytään muotoilemaan ja muokkaamaan dataa ja sen esitysasua. (Flames in Tech 2023.)

Direktiivejä on kaksi eri luokkaa: rakenne- ja attribuuttidirektiivit. Attribuuttidirektiivit vaikuttavat HTML-elementtien ulkonäköön, ja rakennedirektiivit vaikuttavat rakenteeseen. Direktiivejä käytetään komponentin malli-, eli HTML-tiedostossa ja ne on sijoitettava suoraan HTML-elementtien sisään käyttöä varten. Sisäänrakennetut direktiivit ovat osa Angularin valmiita moduuleja, kuten CommonModule ja FormsModule. (Angular 2022h.)

Yleisimpiä attribuuttidirektiivejä ovat ngClass ja ngStyle. NgClass-direktiivillä voidaan dynaamisesti lisätä tai poistaa CSS-luokkia elementistä määriteltyjen ehtojen perusteella. NgStyle-direktiivillä voidaan asettaa elementtiin yksittäisiä CSS-ominaisuuksia. Edellä mainittujen lisäksi myös aikaisemmin tässä työssä kaksisuuntaisen datansitomisen yhteydessä käsitelty ngModel on attribuuttidirektiivi. (Angular 2022h.)

Käytetyimpiä rakennedirektiivejä ovat ngIf, ngFor ja ngSwitch, jotka vastaavat monista ohjelmointikielistä tuttuja if-, for- ja switch-rakenteita. Rakennedirektiivien avulla voidaan ohjelmallisesti määritellä näytettävä sisältö. Yhdessä HTML-elementissä, kuten div, voi olla kerrallaan käytössä vain yksi rakennedirektiivi. (Angular 2022h.)

Putkia käytetään datan formatointiin, suodattamiseen ja järjestämiseen. Putket otetaan käyttöön kuvan 12 osoittamalla syntaksilla mallitiedostossa, ja ne sijoitetaan esimerkiksi elementin sisään, joka noudattaa yksisuuntaista datansidontaa, jolloin putki vaikuttaa esitettyyn muuttujan arvoon.

```
{{testInput | uppercase}}
```

Kuva 12 Esimerkki putkien käytöstä

Esimerkkejä Angularin sisäänrakennetuista putkista ovat Currency, Date, LowerCase ja UpperCase. Currency-putki toteuttaa valuuttakonversion annetun parametrin perusteella, Date-putkella pystytään vaikuttamaan päivämäärän muotoon ja järjestykseen, ja LowerCase sekä UpperCase-putket muuttavat nimensä mukaisesti tekstin joko pieniksi tai isoiksi kirjaimiksi. (Abbas 2023.)

Sisäänrakennettujen putkien lisäksi käyttäjän on mahdollista luoda kustomoituja putkia. Putki on TypeScript-luokka, jonka täytyy sisältää @Pipe-dekoraattori. Dekoraattorin metadataan vaaditaan käyttäjän luoman putken nimi ja luokan sisään sen logiikka. (Abbas 2023.)

3.7 HttpClient ja HttpInterceptorit

HttpClient on yksi Angularin sisäänrakennetuista palveluista, jonka tarkoitus on käsitellä HTTP-kutsuja. HttpClient on osa HttpClientModule nimistä moduulia ja se vaatii toimiakseen kyseisen moduulin importoimisen, sekä itse palvelun injektoimisen sitä käyttävän komponentin konstruktoriin. Yleisin paikka importoida HttpClientModule on juurimoduulissa, jolloin se on koko Angular-ohjelmiston saatavilla. (Mavani 2022.)

HttpClient-plavelun vahvuutena on sen tuoma mahdollisuus luoda tyypitettyjä request ja response objekteja, joka lisää ohjelman tyyppiturvallisuutta. Esimerkkeinä HttpClient:illä käsiteltävistä http-kutsuista ovat niin sanotut CRUD-operaatiot, eli create-, read-, update- ja delete-operaatiot, eli palvelu tukee normaaleja HTTP-kutsuja kuten get, post, put ja delete. (Mavani 2022.)

HttpClientillä on lisäksi mahdollista lisätä HTTP-kutsuihin header-tietoja, kuten autentikoinnissa usein käytettyjä token-arvoja. Myös kutsun body on mahdollista parametrisoida, eli määritellä pyynnön mukana lähetettävä data. Esimerkki HTTP-kutsun toteuttamisesta HttpClientillä on nähtävillä kuvassa 13. (Mavani 2022.)

```
getTestData(){
  const url = 'https://api.url.example';
  return this.http.get(url);
}
```

Kuva 13. Esimerkki GET-metodista

HttpInterceptor, joka tunnetaan myös nimellä interceptori, on Angularin tarjoama rajapinta, joka sallii HTTP-pyyntöjen keskeyttämisen ja niiden sisällön tai header-tietojen muokkaamisen. HttpInterceptor-rajapinta on osa sisäänrakennettua HttpClientModule-moduulia. Interceptorit täytyy määritellä juurimoduulin providers-listassa käyttäen HTTP_INTERCEPTORS-tokenia, joka mahdollistaa niiden käytön yhdessä kaikkien lähtevien ja saapuvien HTTP-pyyntöjen kanssa. Interceptorien järjestys providers-listassa määrittää niiden suoritusjärjestyksen ajonaikana. (Patil 2023.)

Interceptorit toimivat Angular-ohjelman ja serverin välissä, jolloin sillä on mahdollista vaikuttaa liikenteen sisältöön ja muotoon. Yleisiä käyttötarkoituksia erilliselle interceptorille on esimerkiksi token-arvojen tai muiden autentikoinnissa käytettyjen tietojen sisällyttäminen header-tietoihin, sekä globaali virheenkäsittely HTTP-kutsujen yhteydessä. (Patil 2023.)

3.8 Navigointi Angular-sovelluksessa

Angularin reititin (engl. router) on Angularin sisäänrakennettu palvelu, joka mahdollistaa SPA-sovellusten navigoinnin eri komponenttien ja näkymien välillä ilman tarvetta sivun uudelleenlataamiselle. Angularin reititin on osa RouterModule-moduulia ja se tarjoaa metodeja reittien määrittelyyn ja navigointiin, sekä ominaisuuksia, kuten sisällön laiska lataaminen (engl. lazy loading), eli viivästetty lataaminen. (Angular 2023e.)

Angularin reititin vaatii toimiakseen routes-listan, joka sisältää sovelluksen reitit ja niitä vastaavat komponentit. Routes-lista määritellään yleisimmin erillisessä reititys moduulissa,

kuten app.routing.module.ts. Esimerkki reititys moduulista ja sen sisällöstä on nähtävissä kuvassa 14.

```
8  const routes: Routes = [
9      { path: 'home', component: HomeComponent },
10     { path: 'media', component: MediaLibraryComponent },
11     { path: 'about', component: AboutComponent },
12     { path: 'contact', component: ContactComponent },
13     {
14         path: 'admin',
15         loadChildren: () => import('./admin/admin.module').then((m) => m.AdminModule),
16     },
17     { path: '', redirectTo: '/home', pathMatch: 'full' },
18     { path: '**', redirectTo: '/home' },
19 ]
20
21 @NgModule({
22     imports: [RouterModule.forRoot(routes)],
23     exports: [RouterModule],
24 })
```

Kuva 14. Esimerkki Angular reitityksen määrittelystä

Routes-lista, joka alkaa kuvan 14 riviltä 8, lisätään reititys moduulin dekoraattorin imports-listaan. RouterModulen imports-listassa käyttämä syntaksi on nähtävissä kuvassa 14 rivillä 22. Syntaksin forRoot-osio lisää reitit juurimoduulin alaisten komponenttien käytettäväksi. Tarvittaessa on myös mahdollista luoda erillisiä reititysmoduuleita, jotka sisältävät omat routes-listansa. Nämä voivat olla tarpeen, kun luodaan esimerkiksi sisäkkäisiä näkymiä. Reittejä voidaan alustamisen jälkeen hyödyntää käyttämällä <router-outlet>-elementtiä. Router-outlet renderöi reitin näkymän routerLink-direktiivin arvon perusteella, joka yleensä määritellään navigointinäppäimillä. (Angular 2023e.)

Viivästetty lataaminen on ominaisuus, joka vähentää Angular-ohjelman alustavaa latausaikaa ja se mahdollistaa komponenttien ja elementtien lataamisen vasta, kun niitä aletaan käyttämään sovelluksessa. Viivästetty lataaminen toteutetaan yleensä käyttämällä loadChildren-metodia, joka sijoitetaan moduulin käyttämään routes-listaan. Metodi lataa sisällön vasta kun käyttäjä navigoi kyseiseen reittiin. Viivästetty lataaminen siis parantaa sovelluksen latausaikoja ja suorituskykyä. (Angular 2023f.)

Router guardit ovat luokkia, jotka tarjoavat metodeja navigoinnin hallintaan tietyissä näkymissä. Guard-toiminnallisuudet mahdollistavat esimerkiksi pääsynestämisen tiettyyn näkymään, jos määritelty ehto tai ehdot eivät täyty. Guard-toiminnallisuudet, kuten canActivate ja canMatch, toteuttavat erityyppisen validoinnin ennen navigoinnin mahdollistamista, joten niitä hyödynnetään eri käyttötarkoituksissa. (Bouillon 2023.)

CanActivate-toiminnallisuus tarkistaa sille annetun ehdon ja palauttaa boolean arvon, joka määrittää navigoinnin tilan. CanMatch-toiminnallisuutta hyödyntämällä voidaan estää

käyttäjää lataamasta viivästetysti ladattavaa sisältöä reitistä, johon hänellä ei ole pääsyä. Guard-toiminnallisuuksia hyödyntämällä voidaan siis varmistaa, että käyttäjä siirtyy pelkästään näkymiin, joihin hänellä on tarve ja oikeus päästä. (Bouillon 2023.)

3.9 Reaktiiviset lomakkeet

Angular reaktiiviset lomakkeet (engl. reactive forms) ovat työkalu, joka helpottaa lomakkeiden hallintaa ja käsittelyä synkronisesti. Reaktiiviset lomakkeet ja niiden käyttämät metodit ja instanssit ovat osa ReactiveFormsModule-moduulia, ja se täytyy olla lisättynä sitä käyttävän moduulin imports-listaan. (Angular 2023g.)

Reaktiiviset lomakkeet hyödyntävät myöhemmin käsiteltävän RxJS kirjaston tarkkailijarakenteita, jotka mahdollistavat, että lomakkeen kentän tila ja sen data muuttuu yhtäaikaisesti. Aina kun lomakkeen kontrollerissa tapahtuu muutoksia, luodaan uusi observable-objekti, jota käytetään ylläpitämään dataa synkronisesti. Reaktiiviset lomakkeet skaalautuvat rakenteensa ja ominaisuuksiensa ansiosta paremmin kuin perinteiset kiinteät lomakkeet. (Angular 2023g.)

Reaktiivisten lomakkeiden osat

Reaktiiviset lomakkeet vaativat toimiakseen ainakin yhden FormGroup-luokan instanssin, jonka täytyy sisältää vähintään yksi FormControl-luokan instanssi. Näiden lisäksi käytössä voi olla FormArray-instanssi monimutkaisempien rakenteiden hallintaan, sekä FormBuilder-palvelu määrittelyn ja initialisoinnin helpottamiseen. (Angular 2023g.)

FormControl-instanssi sitoo lomakekentän arvon sille annettuun dataan mahdollistaen lomakkeen tilan ja sen käyttämän datan synkronisen päivittämisen hyödyntämällä observables-objekteja. FormGroup-instanssi puolestaan sitoo sen sisältämät FormControl-instanssit yhteen, ja luo niistä rakenteellisia lomakkeita. FormGroup-instanssin tilalla on mahdollista käyttää FormArray-instanssia, joka tukee useita FormControl-instansseja. FormArray-instanssia voidaan siis käyttää esimerkiksi dynaamisesti muuttuvien listojen initialisoimiseen reaktiivisiksi lomakkeiksi. (Angular 2023g.)

FormBuilder-palvelu automatisoi FormControl-instanssien initialisoinnin, joka lyhentää ja selventää reaktiivisten lomakkeiden luonnin vaatimaa syntaksia. (Angular 2023g).

Validaattorit

Validaattorit ovat joukko metodeja, jotka käsittelevät FormControl-instanssin sisältöä. Validaattoreita käytetään pääasiallisesti sisällön validointiin ja sanitointiin. Esimerkkejä saatavilla olevista validaattoreista ovat esimerkiksi required, email ja pattern. Required-

validaattori tekee kyseisestä kentästä pakollisen, eikä lomake ole validi, jos tieto puuttuu. Email-validaattori varmistaa, että kontrollin arvo vastaa hyväksyttyä sähköpostirakennetta, ja pattern-validaattori hyväksyy parametriksi esimerkiksi regEx-lausekkeen. (Angular.)

3.10 RxJS

RxJS, eli Reactive Extensions for JavaScript, on JavaScript kirjasto, jota käytetään reaktiiviseen ohjelmointiin ja sen keskeisin ominaisuus on tarkkailijarakenteet. Tämän lisäksi RxJS sisältää metodeja ja keinoja käsitellä observables-objekteja. RxJS tarjoaa kehittäjälle käyttöön myös muita yleisesti listojen kanssa käytettyjä metodeja, kuten map ja filter. (RxJS.)

Observables-objektien perimmäinen tarkoitus on tarjota käyttäjälle mahdollisuus seurata asynkronisesti datan nykyistä tai tulevaa arvoa. Yleinen Angular-kehityksessä käytetty tapa hyödyntää observable-objekteja on käyttämällä subjektia. Subjektia käytettäessä kaikki sen tilanneet (engl. subscribe) komponentit saavat identtisen arvon samanaikaisesti, joka mahdollistaa ja parantaa sovelluksen komponenttien ja elementtien välistä kommunikointia ja hallintaa. (Angular 2023h.)

Subscription-objekti on objekti, jota käytetään Observables-objektin suorittamiseen. Subscription-objekti luodaan Angular-ohjelmassa kutsumalla subscribe()-metodia, joka palauttaa datan arvot, sekä mahdolliset virheet. Sen jälkeen kun subscription-objektia ei enää tarvita, täytyy käyttäjän kutsua unsubscribe()-metodia vapauttaakseen objektin viemät resurssit ja estääkseen mahdolliset muistivuodot. (Angular 2023h.)

3.11 Angular material

Angular material on Angular-tiimin kehittämä kirjasto visuaalisia komponentteja, elementtejä ja tyylimäärittelyjä, jotka seuraavat Google Material Design 2:n määrityksiä. Angular material -kirjasto sisältää suuren määrän valmiita UI komponentteja, joista esimerkkinä ovat kortti-, käyttäjänsyöte- ja työkalupalkki-komponentit. Valmiit komponentit mahdollistavat modernien ja yleisiä design määrityksiä noudattavien web-sovellusten UI näkymien kehittämistä. (Paredes 2022.)

Angular material-kirjasto lisätään Angular-ohjelmaan käyttämällä Angular CLI:n add-komentoa, esimerkki lisäämiskomennon syntaksista on nähtävissä kuvassa 15.

```
C:\Users\LehtJoo>ng add @angular/material
```

Kuva 15. Esimerkki ng add -komennosta

Asennuksen lisäksi erilliset komponentit pitää importoida ja lisätä niitä käyttävän moduulin imports-listaan. Angular material komponenttien valitsimet ovat muotoa <mat-komponentin-nimi> ja niitä käytetään osana komponentin mallitiedostoa. Kuvassa 16 on nähtävissä esimerkki mat-icon komponentin valitsimesta osana sitä käyttävän komponentin mallitiedostoa. (Angular Material.)

```
<h1>
    Example text
    <mat-icon>close</mat-icon>
</h1>
```

Kuva 16. Esimerkki angular material -kirjaston valitsimesta

4 Toteutusvaihe

4.1 Sovelluksen suunnittelu

Toteutusvaiheen alussa pidettiin yhdessä asiakkaan kanssa palaveri, jossa kartoitettiin sovelluksen vaadittuja toiminnallisuuksia, sekä visuaalista ilmettä. Palaverissa tutustuttiin esimerkkiverkkosivuihin sekä värimaailmoihin ja fontteihin, joiden pohjalta kokonaiskuvaa lähdettiin rakentamaan.

Alkuvaiheessa myös ominaisuuksien ja toiminnallisuuksien määrittelyt tarkentuivat, ja päätoiminnallisuuksiksi valittiin varauslomake, jolla käyttäjän on mahdollista varata päivämäärän perusteella itselleen esiintyminen. Lomakkeen lisäksi toiseksi päätoiminnallisuudeksi määräytyi mediakirjasto esiintymiskuville ja videoille. Videot päätettiin toteuttaa upotetulla YouTube-soittimella ja kuvia varten päätettiin kehittää toiminnallisuus kuvien lataamiselle suoraan ylläpitäjän laitteelta.

Päätoiminnallisuuksien lisäksi nousi esille tarve ylläpitäjän dashboard-näkymästä, jota kautta mediakirjaston sisältöä on mahdollista muokata ja lisätä. Dashboard näkymään pääsyä varten piti luoda myös ylläpitäjän sisäänkirjautumisnäkymä toiminnallisuuksineen.

Pääkomponenttien ja ylläpitäjän näkymien lisäksi sovelluksen sovittiin sisältävän contact ja about-näkymät. Contact-näkymään määriteltiin reaktiivinen lomake yhteydenottopyyntöjä varten. About-näkymässä esitellään duon jäsenet ja vastataan yleisimpiin kysymyksiin, hyödyntäen Angular Material -kirjaston elementtejä.

Asiakkaan kanssa tehtiin tiivistä yhteistyötä koko projektin aikana ja järjestettyjen palaverien pohjalta täsmennettiin ja tarkennettiin sovittuja sovelluksen toiminnallisuuksia ja visuaalista ilmettä. Asiakas toimitti projektin alussa myös valitsemansa fontit, väriteeman, sekä esimerkkejä hyödyntämällä alustavan rakenteen sivulle.

4.2 Sovelluksen toteuttaminen

Juurimoduulin alustuksen jälkeen projektiin lisättiin Angular material-kirjasto. Tarvittavien toiminnallisuuksien pohjalta luotiin lisäksi neljä erillistä moduulia: shared, core, feature sekä admin-moduulit. Lisäksi projektiin perustettiin juuritasolle models-kansio sovelluksen vaatimia TypeScript interface-tiedostoja varten.

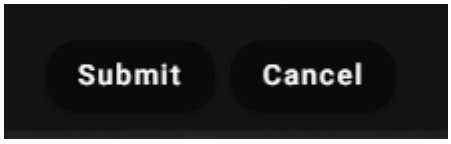
4.2.1 Shared-moduuli

Shared-moduulin kehitys aloitettiin luomalla moduulin sisäinen kansiorakenne. Moduuliin luotiin buttons, form-components ja components -kansiot. Luodut komponentit lajiteltiin

kansioihin niiden toiminnallisuuden ja luonteen perusteella. Shared-moduuliin luotujen komponenttien tarkoituksena oli pyrkiä lisäämään uudelleenkäytettävyyttä sekä koodin ja toiminnallisuuden kapselointia.

Shared-moduuli toimi myös yleisenä import-moduulina, johon lisättiin esimerkiksi sovelluksen käyttämät Angular material -kirjaston komponentit, sekä muut Angular moduulit. Tämä lähestymistapa mahdollisti Angularin ja kolmannen osapuolen komponenttien ja palvelujen keskitetyn hallinnan ja samalla piti esimerkiksi juurimoduulin module.ts-tiedoston siistinä ja organisoituna.

Buttons-kansioon luotiin sovelluksessa laajasti käytettyjä näppäimiä, kuten submit- ja cancel-näppäimet. Erillisten näppäinkomponenttien luominen mahdollisti näppäinten muotoilun ja toiminnallisuuden keskittämisen. Tällä tavoin saavutettiin laadukkaampi käyttäjäkokemus ja yhtenäisempi ulkoasu. Kuvassa 17 on esimerkki sovelluksessa käytetyistä submit- ja cancel-näppimistä.



Kuva 17. Sovelluksen submit- ja cancel-näppäimet

Form-components-kansioon puolestaan luotiin Angularin reaktiivisten lomakkeiden kanssa käytettäviä komponentteja, kuten combined-input, jonka mallitiedosto on nähtävissä kuvassa 18.

```
<div class="combined-input">
    <mat-form-field appearance="fill">
        <mat-label>{{ firstNameLabel }}</mat-label>
        <input
            matInput
            [formControl]="firstNameControl"
            required />
    </mat-form-field>

    <mat-form-field appearance="fill">
        <mat-label>{{ lastNameLabel }}</mat-label>
        <input
            matInput
            [formControl]="lastNameControl"
            required />
    </mat-form-field>
</div>
```

Kuva 18. Combined-input komponentti

Combined-input komponentti suunniteltiin yhdistämään kaksi mat-form-field-kenttää yhdeksi. Se käyttää @Input()-dekoraattoreita, joilla vastaanotetaan komponentin käyttämän lomakkeen kontrollerit datan sidontaa varten. Yhdistetyn komponentin luonti selkeyttää datan hallintaa ja helpottaa komponentin muotoilua.

Lisäksi kansioon luotiin custom-input ja custom-textarea -komponentit. Custom-input-komponenttiin sijoitettiin Angular-lomakkeiden mat-form-field:in lisäksi yksittäinen input-kenttä, ja custom-textarea-komponenttiin sijoitettiin skaalautuva ja venytettävä tekstikenttä. Form-components-kansion komponentteja hyödynnettiin varaus- ja yhteydenottolomakkeissa, jotka esitellään seuraavassa kappaleessa.

Components-kansioon toteutettiin sovelluksen päätoiminnallisuuteen liittyvät komponentit, joihin kuuluvat photo-album, video-player, contact-form sekä reservation-form -komponentit.

Photo-album-komponentti

Photo-album-komponentti suunniteltiin ja toteutettiin tarjoamaan käyttäjille mahdollisuus selata kuvia vaakataso näkymässä. Desktop-versiossa kuvien selaaminen tapahtuu scrollbutton-näppäimillä. Mobiiliversion kuvien selaaminen toteutettiin älypuhelimista tutulla swipe-ominaisuudella. Valittu kuva näytettiin suurennettuna modal-ikkunassa, joka oli mahdollista sulkea klikkaamalla sen ulkopuolista aluetta.

Photo-album.service-palvelu toteutettiin hallinnoimaan kuvia ja se hyödynsi RxJS-kirjaston BehaviorSubject-objekteja. Tämän ansiosta kuvien päivittäminen ja tila pystyttiin käsittelemään reaaliajassa, mikä puolestaan paransi sovelluksen käyttökokemusta ja yleistä suorituskykyä. Lisäksi palveluun luotiin useAPI boolean-arvo, jonka perusteella käsiteltiin tietoja joko lokaalissa sovelluksen muistissa tai käytettiin HttpClient:in metodeita.

AddPhoto-metodi kehitettiin mahdollistamaan kuvien lisäämisen palveluun. Kuvat lisätään joko lähettämällä ne backendiin, jos useAPI-flag on aktiivinen, tai säilyttämällä ne lokaalisti flagin ollessa pois päältä. Metodi käsittelee kuvan URL-osoitetta ja päivittää sen photosSource BehaviorSubjectiin. Lisääminen tapahtuu levittämällä nykyinen photosSource:n sisältö ja lisäämällä siihen uuden kuvan tiedot. Metodi on nähtävissä kuvassa 19.

```
addPhoto(imageUrl: string) {
    if (this.useAPI) {
        this.http
            .post('/placeholderAPI/photos', { imageUrl })
            .pipe(map((response) => response as Photo))
            .subscribe((newPhoto: Photo) => {
                const photos = this.photosSource.getValue()
                this.photosSource.next([...photos, newPhoto])
            })
    } else {
        const photos = this.photosSource.getValue()
        const newPhoto: Photo = { id: photos.length + 1, imageUrl }
        this.photosSource.next([...photos, newPhoto])
    }
}
```

Kuva 19. AddPhoto-metodi

RemovePhoto-metodi puolestaan lisää toiminnallisuuden kuvien poistoon. Riippuen useAPI-flagin tilasta, kuvat poistetaan joko lokaalista sovelluksenmuistista tai delete-kutsu välitetään backendille. Lokaalisti poistettaessa metodi hyödyntää listojen kanssa käytettyä filter-metodia. Metodi suodattaa kuvalistasta pois kuvan, jonka ID vastaa metodille parametriksi syötettyä ID-numeroarvoa. Metodi on nähtävissä kuvassa 20.

```
removePhoto(id: number) {
    if (this.useAPI) {
        this.http.delete(`/placeholderAPI/photos/${id}`).subscribe(() => {
            this.http.get<Photo[]>('/placeholderAPI/photos').subscribe((updatedPhotos: Photo[]) => {
                this.photosSource.next(updatedPhotos)
            })
        })
    } else {
        const photos = this.photosSource.getValue().filter((photo) => photo.id !== id)
        this.photosSource.next(photos)
    }
}
```

Kuva 20. RemovePhoto-metodi

Lisäksi palveluun luotiin myös uploadPhoto-metodi, joka mahdollistaa kuvien lataamisen suoraan käyttäjän koneelta. Metodi hyödyntää tiedostonlukijaa, eli FileReader:ia, joka lukee ladatun tiedoston ja muuttaa sen base64-koodatuksi URL-osoitteeksi. Kyseinen osoite välitetään addPhoto-metodille, joka lisää kuvan kokoelmaan. uploadPhoto-metodin ansiosta sovellukseen on mahdollista lisätä kuvia reaaliajassa käyttäjän omilta laitteilta. Metodi on nähtävissä kuvassa 21.

```
uploadPhoto(file: File) {
    const reader = new FileReader()
    reader.onload = (e) => {
        const photoUrl = e.target?.result
        if (typeof photoUrl === 'string') {
            this.addPhoto(photoUrl)
        }
    }
    reader.readAsDataURL(file)
}
```

Kuva 21. UploadPhoto-metodi

Edellä mainittujen metodien avulla photo-album.service-palvelu tarjoaa vahvan perustan kuvien ylläpitämiselle ja reaaliaikaiselle lataamiselle käyttäjän laitteelta. Kuvien käsittely tapahtuu admin-dashboard-komponentin kautta, jota käsitellään myöhemmin tässä työssä.

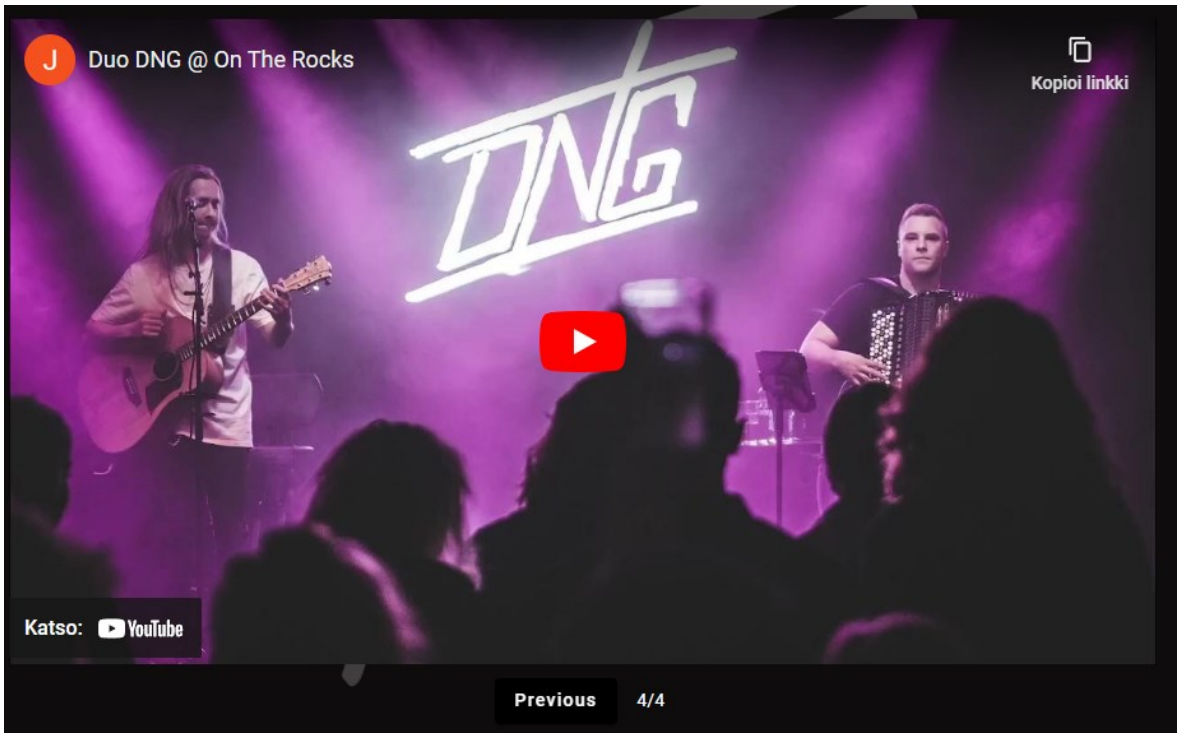
Video-player-komponentti

Video-player-komponentti suunniteltiin ja toteutettiin tarjoamaan käyttäjille mahdollisuus selata ja katsella ennalta määrättyä listaa videoita. Käyttöliittymään rakennettiin YouTube:n syvennetty soitin hyödyntämällä iframe-elementtiä, joka on nähtävissä kuvassa 22.

```
<div class="video-player">
    <iframe
        id="youtube-player"
        width="560"
        height="315"
        [src]="currentVideoUrl"
        frameborder="0"
        allowfullscreen></iframe>
</div>
```

Kuva 22. iframe-elementti

Iframe-elementille määriteltiin leveys ja korkeus, sekä videosource, joka noudettiin komponentin omasta palvelusta. Upotetun soittimen lisäksi komponenttiin kehitettiin näppäimet, joilla käyttäjä pystyy siirtymään videoiden välillä, sekä indikaatio videoiden määrästä. Video-player-komponentti käyttöliittymän näkökulmasta on nähtävissä kuvassa 23.



Kuva 23. Video-player-komponentti

Video-player.service-palvelun kehitettiin hallinnoimaan dynaamisesti listaa YouTube-linkeistä hyödyntäen RxJS-kirjaston BehaviorSubject-objektia, joka mahdollisti sisällön joustavan muokkaamisen reaaliajassa. Lisäksi palveluun luotiin useAPI boolean-arvo, jonka perusteella käsiteltiin tietoja joko lokaalissa sovelluksen muistissa tai käytettiin HttpClient:in metodeja.

AddVideo-metodi kehitettiin lisäämään uusia videoita palveluun. Lisäämisen yhteydessä normaali YouTube-linkki piti muotoilla uudestaan toteuttamaan embedded-linkin muotoilua. Tätä varten kehitettiin extractVideoId-metodi, joka erottaa normaalista YouTube-linkistä ID-arvot käyttämällä regEx-lausekkeita. Eroteltu ID palautetaan AddVideo-metodille, joka muodosti lopullisen linkin. Kun linkki on muotoiltu, lisää metodi sen suoraan videosSource BehaviorSubjectiin, tai useAPI-flagin perusteella lähettää sen backendille. AddVideo-metodi on nähtävissä kuvassa 24.

```
addVideo(url: string) {
    const videoId = this.extractVideoId(url)
    if (videoId) {
        const embeddedUrl = `https://www.youtube.com/embed/${videoId}`
        if (this.useAPI) {
            this.http.post<Video>('/placeholderAPI/videos', { url: embeddedUrl }).subscribe((newVideo: Video) => {
                const videos = this.videosSource.getValue()
                this.videosSource.next([...videos, newVideo])
            })
        } else {
            const videos = this.videosSource.getValue()
            const newVideo: Video = { id: this.lastIndex + 1, url: embeddedUrl }
            this.videosSource.next([...videos, newVideo])
        }
    }
}
```

Kuva 24. addVideo-metodi

RemoveVideo-metodi puolestaan kehitettiin mahdollistamaan videoiden poistamisen joko suoraan lokaalista sovelluksen muistista, tai mahdollisesta backendista, riippuen jälleen useAPI-flagin tilasta. Lokaalisti video poistettiin hyödyntämällä sen ID:tä, sekä listojen kanssa käytettävää filter-metodia. Videolistasta siis suodatettiin pois video, jonka ID-arvo vastasi RemoveVideo-metodin parametriksi annettua ID-arvoa. Metodi on nähtävissä kuvassa 25.

```
removeVideo(id: number) {
    if (this.useAPI) {
        this.http.delete(`/placeholderAPI/videos/${id}`).subscribe(() => {
            this.http.get<Video[]>('/placeholderAPI/videos').subscribe((updatedVideos: Video[]) => {
                this.videosSource.next(updatedVideos)
            })
        })
    } else {
        const videos = this.videosSource.getValue().filter((v) => v.id !== id)
        this.videosSource.next(videos)
    }
}
```

Kuva 25. removeVideo-metodi

Edellä mainittujen metodien avulla video-player.service-palvelu tarjoaa vahvan perustan YouTube-videoiden hallinnoille sovelluksessa ja mahdollistaa saatavilla olevien videoiden poistamisen ja lisäämisen dynaamisesti käyttöliittymästä.

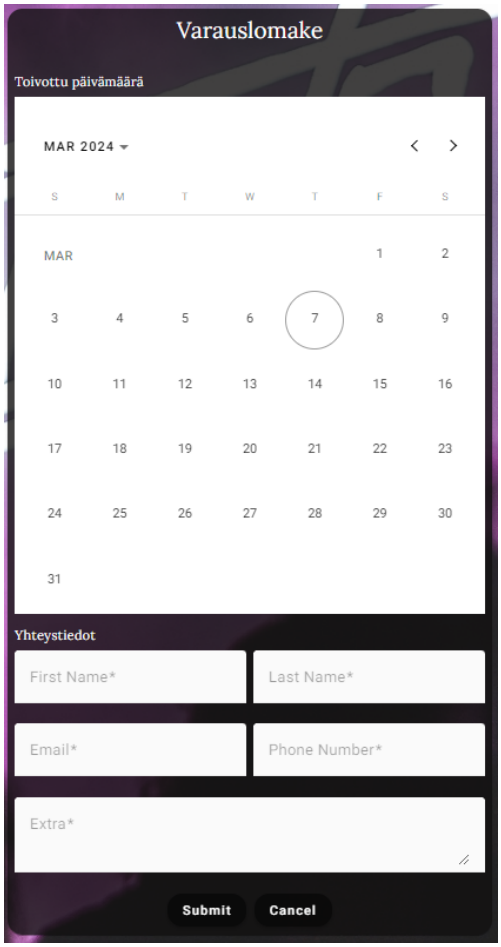
Video-player-komponentti asetettiin kuuntelemaan palvelun videolistaa heti komponentin initialisoinnin yhteydessä elinkaarimetodi ngOnInit:in välityksellä (kuva 26) ja samalla päivitettiin nykyisen videon indeksi.

```
ngOnInit() {
    this.videoService.videos$.subscribe((videos) => {
        this.videos = videos
        this.setCurrentVideo(this.currentIndex)
    })
}
```

Kuva 26. Video-player-komponentin ngOnInit

Reservation-form-komponentti

Reservation-form-komponentti, eli varauslomake-komponentti, toteutettiin käsittelemään esiintymisvarauksia. Komponenttiin sisällytettiin mat-calendar-komponentin kalenterinäkymä toivotun päivämäärän valintaa varten, varaajan yhteystietojen inputkentät, sekä vapaamuotoinen tekstikenttä. Lomakkeen loppuun lisättiin submit- ja cancel-näppäimet, jotka tarjoavat käyttäjälle mahdollisuuden lähettää tai perua lomake. Varauslomake-komponentin käyttöliittymänäkymä on nähtävissä kuvassa 27.



Kuva 27. Varauslomake-komponentti

Komponentti validoi lomakkeen ennen submit-tapahtuman hyväksymistä. Jos lomakkeen kaikki kentät on täytetty oikein, tulostetaan arvot selaimen konsolinäkymään. Komponentti

kehitettiin keräämään ja validoimaan potentiaalisen asiakkaan yhteystiedot, jotka voidaan onnistuneen validoinnin ja submit-tapahtuman jälkeen välittää sähköpostitse ylläpitäjille.

4.2.2 Core-moduuli

Core-moduuli kehitettiin säilömään singleton-komponentit ja -palvelut, eli komponentit ja palvelut, jotka esiintyvät Angular-sovelluksessa vain kerran. Moduulin kehittäminen aloitettiin luomalla header- ja sidebar-komponentit, interceptor token -arvojen käsittelyä varten sekä auth service -palvelu autentikoinnin käsittelyyn.

Core-moduulin elementit luotiin käyttämällä Angular CLI komentoriviä. Elementit sijoitettiin niiden nimien mukaisiin kansioihin, noudattaen Angularin parhaita käytäntöjä. Luodut komponentit lisättiin osaksi core.module-moduulitiedoston declarations- ja exports-listoja, jotta niitä voitiin käyttää muualla sovelluksessa.

Header-komponentti

Header-komponentin tarkoituksena oli toteuttaa yleisesti web-kehityksessä käytetty header eli verkkosivun yläreunassa sijaitseva työkalupalkki. Header-komponentti tyypillisesti sisältää navigoinnin ja muuta toiminnallisuutta. Header-komponenttiin sisällytettiin sosiaalisenmedian näppäimet, sekä navigointiin käytetyt näppäimet.

Navigointi toteutettiin käyttämällä Angular-reititintä, sekä sen sisältämiä metodeja. Reitin kutsut toteutettiin routerLink-toiminnallisuutta käyttäen, kuvassa 28 on nähtävissä header-komponentin mallitiedosto.

```
<mat-toolbar color="primary">
    <app-social-media-group *ngIf="!isSmallScreen"></app-social-media-group>

    <span class="spacer"></span>

    <button *ngIf="isSmallScreen" class="hamburger-button" mat-icon-button (click)="toggleSidebar()">
        <mat-icon>menu</mat-icon>
    </button>

    <div *ngIf="!isSmallScreen" class="menu">
        <button mat-button routerLink="/home">Home</button>
        <button mat-button routerLink="/media">Media</button>
        <button mat-button routerLink="/about">About</button>
        <button mat-button routerLink="/contact">Contact</button>
    </div>
</mat-toolbar>
```

Kuva 28. Header-komponentin mallitiedosto

Mobiilinäkymässä header-komponentti muokattiin hyödyntämään hampurilaisvalikkoa navigoinnin toteuttamiseen, sekä sosiaalisenmedian näppäinten sisällyttämiseen. Hampurilaisnäppäimen toggleSidebar-metodilla renderöitiin sidebar-komponentti.

Sidebar-komponentti

Sidebar-komponentti kehitettiin toteuttamaan mobiililaitteiden navigointi ja sisältämään sosiaalisenmedian näppäimet. Sidebar-komponentti asetettiin saataville vain, kun näyttöpäätteen maksimi leveys oli alle 768 pikseliä. Sidebar-komponentti määriteltiin sulkeutumaan joko kun Angular-sovellus suoritti navigointi tapahtuman, tai kun käyttäjä kosketti sidebar-komponentin taustaa.

Autentikointi-palvelu

Autentikointi-palvelu, eli auth.service.ts, toteutettiin autentikointiin liittyvien toiminnallisuuksien käsittelyyn sekä token-arvojen käsittelyn demonstrointiin sovelluksessa. Autentikointi-palvelun TypeScript-tiedosto on nähtävissä kuvassa 29.

```typescript
import { Injectable } from '@angular/core'

@Injectable({
    providedIn: 'root',
})
export class AuthService {
    private username: string = 'admin'
    private password: string = 'password'

    login(username: string, password: string): boolean {
        if (username === this.username && password === this.password) {
            const token = 'placeholder-token'
            sessionStorage.setItem('token', token)
            return true
        }
        return false
    }

    getToken(): string | null {
        return sessionStorage.getItem('token')
    }
}
```

Kuva 29. Authentication.service.ts

Palveluun luotiin privaatit muuttujat käyttäjätunnukselle ja salasanalle, sekä login- ja getToken-metodit autentikoinnin sisäänkirjautumisen demonstrointia varten. Login-metodilla lisättiin onnistuneen autentikoinnin jälkeen token-arvo sessiomuistiin, ja getToken-metodia käytettiin noutamaan tallennettu token-arvo. Token-arvoa hyödynnettiin demonstrointitarkoituksessa token-interceptorissa.

Token interceptori

Token-interceptor, eli token.interceptor.ts, toteutettiin demonstroimaan interceptorin toimintaa HTTP-kutsujen yhteydessä. Token-interceptoria käyttämällä HTTP-kutsujen header-

osioon lisättiin sessiomuistissa säilytetty token-arvo, joka välittyy jokaisen HTTP-kutsun yhteydessä.

4.2.3 Features-moduuli

Features-moduuli kehitettiin kattamaan sovelluksen päänäkymät, eli home-, about-, contact- ja media-library-komponentit. Tämän moduulin komponentit toimivat pääasiallisesti säiliöinä eri toiminnallisuuksille tarjoten jäsennellyn ja modulaarisen kokonaisuuden.

Näkymien erottaminen komponenteiksi mahdollisti myös reititinkomponentin hyödyntämisen näkymien välillä. Features-moduulin näkymät toimivat sovelluksen päänäkyminä ja eivät toteuta viivästettyä lataamista, koska features-moduulin täytyy olla sovelluksen saatavissa heti sen käynnistämisen yhteydessä. Moduulin komponentit lisättiin juuritason app.routing.module-moduuliin hyödyntämällä Angular-router:in metodeja. App.routing.module-moduulin konfiguraatio on nähtävissä kuvassa 30.

```
import { NgModule } from '@angular/core'
import { RouterModule, Routes } from '@angular/router'
import { HomeComponent } from './features/home/home.component'
import { MediaLibraryComponent } from './features/media-library/media-library.component'
import { AboutComponent } from './features/about/about.component'
import { ContactComponent } from './features/contact/contact.component'

const routes: Routes = [
    { path: 'home', component: HomeComponent },
    { path: 'media', component: MediaLibraryComponent },
    { path: 'about', component: AboutComponent },
    { path: 'contact', component: ContactComponent },
    {
        path: 'admin',
        loadChildren: () => import('./admin/admin.module').then((m) => m.AdminModule),
    },
    { path: '', redirectTo: '/home', pathMatch: 'full' },
    { path: '**', redirectTo: '/home' },
]

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule],
})
export class AppRoutingModule {}
```

Kuva 30. App.routing.module-moduuli

Reittien määrittelyssä otettiin käyttöön myös wildcard-reitti, joka on määritelty käyttämällä **-syntaksia. Wildcard-reitti ohjaa kaikki määrittelyn ulkopuoliset reitit sen määrittämään polkuun.

4.2.4 Admin-moduuli

Admin-moduuli kehitettiin sisältämään sovelluksen tarvitsemat ylläpitäjän näkymät ja palvelut. Moduuliin kehitettiin admin-login ja admin-dashboard komponentit, admin-routing.module-moduuli ja auth.guard.ts guardi.

Admin-moduulin reitit määriteltiin admin-routing.module-moduulissa, joka mahdollisti sisällön viivästetyn lataamisen. Juurireititinmoduuli määriteltiin lataamaan admin-moduulin sisältö käyttämällä loadChildren-metodia, joka samalla aktivoi admin-routing-moduulin käytön. Admin-routing-moduulissa otettiin käyttöön myös canActivate toiminnallisuus, jolla estettiin admin-dashboard komponenttiin navigointi ilman onnistunutta autentikointia.

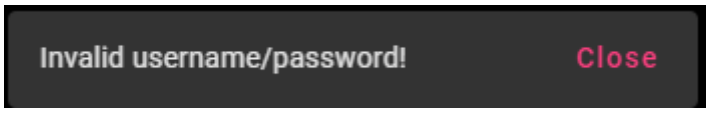
Auth.guard.ts-komponentti kehitettiin hyödyntämään canActivate-metodia, joka palautti boolean arvon minkä perusteella navigointi sallittiin tai estettiin. Ehdoksi kyseiseen guard-toiminnallisuuteen määriteltiin token-arvon löytyminen, joka lisätään autentikointi palvelun kautta sessionStorage-muistiin onnistuneen autentikoinnin yhteydessä.

Admin-login-komponentti

Admin-login-komponentin kehittämisen tavoitteena oli toteuttaa ylläpitäjän sisäänkirjautumistoiminnallisuus. Komponenttiin luotiin loginForm-lomake, joka sisältää kaksi kenttää: username ja password. Komponentti kehitettiin kutsumaan onLogin-metodia joko submit-painikkeella tai enter näppäimellä, hyödyntäen keydown-tapahtumaa.

Kun lomake on validi, komponentti kutsuu core-moduulin auth.service-palvelun login-metodia. Tämä metodi huolehtii autentikoinnista ja tallentaa onnistuneen kirjautumisen esimerkki token-arvon selaimen sessionStorage-muistiin. Onnistuneen sisäänkirjautumisen jälkeen komponentti ohjaa käyttäjän admin-dashboard näkymään käyttäen Angularin reititintä.

Mikäli autentikointi epäonnistuu väärän salasanan tai käyttäjätunnuksen takia, komponenttiin lisätty Angular Material -kirjaston snackbar-komponentti ilmoittaa näkymän yläreunassa epäonnistuneesta sisäänkirjautumisyrityksestä. Samalla lomakkeen kentät myös tyhjennetään. Snackbar-komponentti on nähtävissä kuvassa 31.

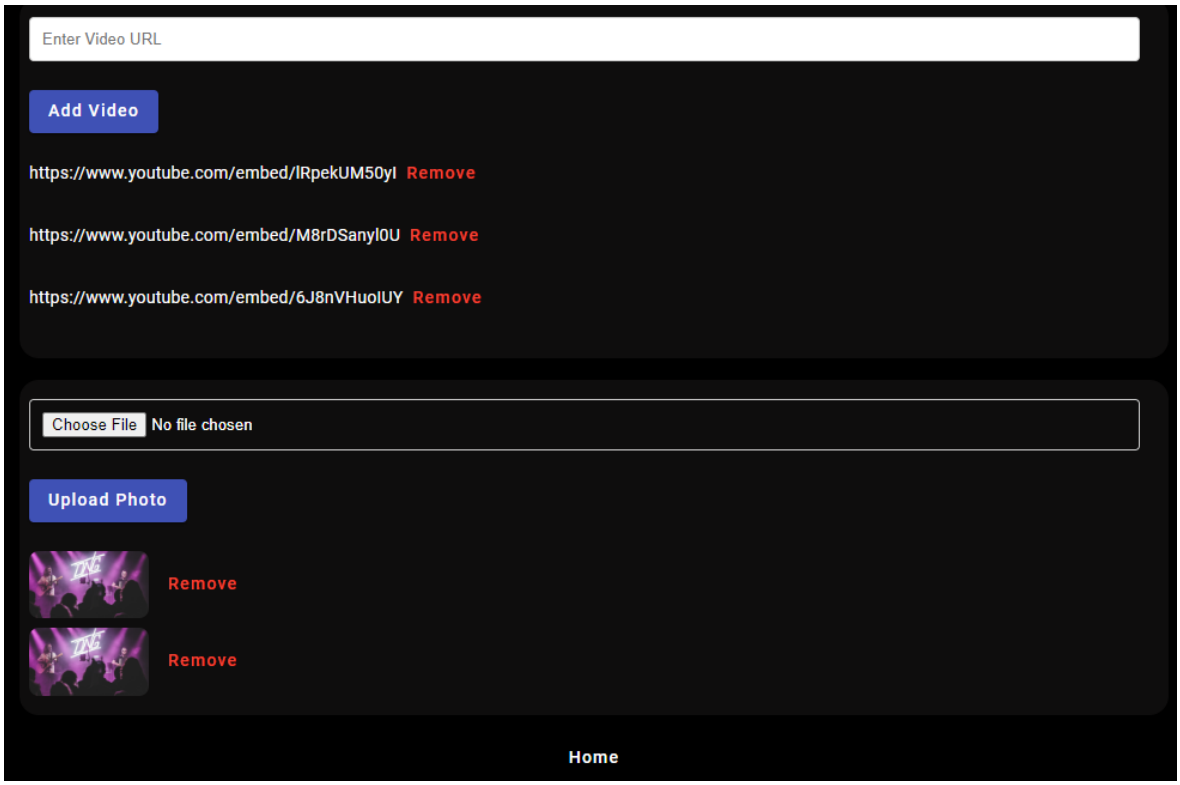


Kuva 31. Snackbar-komponentti

Admin-dashboard-komponentti

Admin-dashboard-komponentti kehitettiin tarjoamaan ylläpitäjille käyttöliittymä kuvien ja videoiden lisäämiseen ja poistamiseen. Komponentti käyttää PhotoAlbumService- ja

VideoPlayerService-palveluita sisällön hallinnassa. Käyttöliittymään luotiin input-kentät YouTube linkkejä ja kuvien lataamista varten, sekä mahdollisuus poistaa kuvia ja videoita yksitellen. Lisäksi näkymään lisättiin home-näppäin, jolla ylläpitäjä pysty siirtymään takaisin sovelluksen aloitussivulle. Admin-dashboard-komponentin käyttöliittymä on nähtävissä kuvassa 32.



Kuva 32. Admin-dashboard komponentin käyttöliittymä

5 Yhteenveto

Työn tavoitteena oli toteuttaa responsiivinen ja käyttäjäystävällinen Angular frontend-sovellus. Päätoiminnallisuuksiksi määriteltiin varauslomake ja mediakirjasto. Lisäksi tarkoituksena oli toteuttaa ylläpitäjän sisäänkirjautumis- ja ylläpitonäkymä mediakirjaston ylläpitämiseen, sekä verkkosivuille tavanomaiset contact ja about -näkymät.

Tavoitteissa onnistuttiin hyvin ja kaikki sovitut toiminnallisuudet saatiin toteutettua asiakkaan toiveiden mukaisesti. Työn aikana tehtiin tiivistä yhteistyötä asiakkaan kanssa, jonka avulla pystyttiin takaamaan myös toivottu ulkoasu asiakkaan ideoiden ja toiveiden pohjalta. Myös toivottu responsiivisuus onnistuttiin toteuttamaan.

Toteutuksen haasteena oli työn rajaus, eli pelkkä frontend kehittäminen. Koska aiheena oli pelkkä frontend-sovelluksen kehittäminen, jouduttiin joissain teknisissä asioissa joko joustamaan, tai tyytymään demonstroimaan esimerkein toiminnallisuutta. Esimerkiksi backendin puuttuminen aiheutti sen, että autentikointi ja mediakirjaston tiedostojen hallinta piti toteuttaa lokaalisti frontend-sovelluksessa, jolla jäljiteltiin niiden toiminnallisuutta. Hyvänä puolena voidaan mainita kuitenkin Angularin monipuoliset komponentit ja palvelut, joilla kyettiin valmistelemaan, sekä osakseen testaamaan frontend-sovellus myös kokonaisvaltaista sovelluskehittämistä varten.

Jatkokehittämisehdotuksina on ehdottomasti backendin kehittäminen ja integroiminen sovellukseen, jolloin olisi mahdollista toteuttaa kaikki sovelluksen vaatimat toiminnallisuudet. Lisäksi olisi hyödyllistä ottaa käyttöön Azure Devops tai vastaava palvelu. Tämän avulla projekti voisi hyödyntää esimerkiksi ketteriä kehitysmenetelmiä, kuten sprint rakennetta. Vastaavasti myös tehtävien jakaminen ja aikatauluttaminen, sekä resurssienhallinta olisi helpompaa pitkällä aikavälillä. Vastaavaa palvelua hyödyntämällä myös sovelluksen kääntäminen ja testaaminen olisi mahdollista automatisoida.

Yhteenvetona voidaan todeta, että työn alussa asetettuihin tavoitteisiin päästiin ja palautteen perusteella asiakas oli myös tuotokseen tyytyväinen.

Lähteet

Abbas, A. 2023. Pipes in Angular. Medium. Viitattu 28.2.2024. Saatavissa https://medium.com/@aqeelabbas3972/pipes-in-angular-6a871589299d

Angular. Validators. Viitattu 5.3.2024. Saatavissa https://angular.io/api/forms/Validators

Angular. 2022a. Component Lifecycle. Viitattu 24.2.2024. Saatavissa https://angular.io/guide/lifecycle-hooks

Angular. 2022b. Component styles. Viitattu 24.2.2024. Saatavissa https://angular.io/guide/component-styles

Angular. 2022c. Basics of testing components. Viitattu 24.2.2024. Saatavissa https://angular.io/guide/testing-components-basics

Angular. 2022d. NgModules. Viitattu 26.2.2024. Saatavissa https://angular.io/guide/ngmodules

Angular. 2022e. Feature modules. Viitattu 26.2.2024. Saatavissa https://angular.io/guide/feature-modules

Angular. 2022f. Guidelines for creating ngModules. Viitattu 26.2.2024. Saatavissa https://angular.io/guide/module-types

Angular. 2022g. Creating an injectable service. Viitattu 28.2.2024. Saatavissa https://angular.io/guide/creating-injectable-service

Angular. 2022h. Built in directives. Viitattu 28.2.2024. Saatavissa https://angular.io/guide/built-in-directives

Angular. 2023a. What is Angular? Viitattu: 23.2.2024. Saatavissa https://angular.io/guide/what-is-angular

Angular. 2023b. Component. Viitattu: 23.2.2024. Saatavissa https://angular.io/api/core/Component

Angular. 2023c. Introduction to components and templates. Viitattu: 23.2.2024. Saatavissa https://angular.io/guide/architecture-components

Angular. 2023d. Launching your app with a root module. Viitattu 26.2.2024. Saatavissa https://angular.io/guide/bootstrapping

Angular. 2023e. Common Routing Tasks. Viitattu 1.3.2024. Saatavissa https://angular.io/guide/router

Angular. 2023f. Lazy-loading feature modules. Viitattu 1.3.2024. Saatavissa https://angular.io/guide/lazy-loading-ngmodules

Angular. 2023g. Reactive forms. Viitattu 4.3.2024. Saatavissa https://angular.io/guide/reactive-forms

Angular. 2023h. The RxJS library. Viitattu 4.3.2024. Saatavissa https://angular.io/guide/rx-library

Angular Material. Getting Started with Angular Material. Viitattu 4.3.2024. Saatavissa https://material.angular.io/guide/getting-started

Bampakos, A. 2021. Angular Projects: Build Modern Web Apps by Exploring Angular 12 with 10 Different Projects and Cutting-edge Technologies. Birmingham: Packt Publishing Ltd.

Bampakos, A. & Deeleman, P. 2020. Learning Angular: A No-nonsense Beginner's guide to Building Web Applications with Angular 10 and Typescript. Birmingham: Packt Publishing Ltd.

Bouillon, P. 2023. Demystifying Angular Route Guards: A Beginner's Guide to Secure Navigation. Viitattu 1.3.2024. Saatavissa https://dev.to/this-is-angular/demystifying-angular-route-guards-a-beginners-guide-to-secure-navigation-597b

Coyier, C. 2022. A Complete Guide to Flexbox. Viitattu 22.2.2024. Saatavissa https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Frain, B. 2020. Responsive Web Design with HTML5 and CSS: Develop Future-proof Responsive Websites Using the Latest HTML5 and CSS Techniques, 3rd Edition. Birmingham: Packt Publishing.

Flames In Tech. 2023. Angular Directives and Pipes: A Guide to Enhancing Your Applications. Viitattu 28.2.2024. Saatavissa https://medium.com/startit-up/angular-directives-and-pipes-a-guide-to-enhancing-your-applications-3ee202b3bb86

Jansen, R. H., Vane, V., de Wolf, I. G., 2016, TypeScript: Modern JavaScript Development, Birmingham: Packt Publishing.

Mavani, K. Angular HttpClient. Medium. Viitattu 1.3.2024. Saatavissa https://medium.com/@kevinmavani/angular-httpclient-1f4eb6ff3785

Material Design 2 a. Google. Viitattu 22.2.2024. Saatavissa https://m2.material.io/

Material Design 2 b. Google. Viitattu 22.2.2024. Saatavissa https://m2.material.io/components/date-pickers

Mozilla Developer Network. 2024b. What is CSS? Viitattu 22.2.2024. Saatavissa https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS

Mozilla Developer Network. 2024a. Flexbox. Viitattu 22.2.2024. Saatavissa https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

Patil, J. 2023. HTTP Interceptors in Angular. Medium. Viitattu 1.3.2024. Saatavissa https://medium.com/@jaydeepvpatil225/http-interceptors-in-angular-6e9891ae0538

Paredes, D. 2022. Angular Basics: What Is Angular Material, When to Use It, When To Look Elsewhere. Telerik. Viitattu 4.3.2024. Saatavissa https://www.telerik.com/blogs/angular-basics-what-is-angular-material-when-use-when-look-elsewhere

RxJS. Introduction. Viitattu 4.3.2024. Saatavissa https://rxjs.dev/guide/overview

Srivastava, V. 2023. Angular Basics: Angular Constructor Overview. Telerik. Viitattu 24.2.2024. Saatavissa https://www.telerik.com/blogs/angular-basics-angular-constructor-overview