

Opinnäytetyö (AMK)

Tietojenkäsittely

2021

Aleksi Valojää

TUOTE-ESITTELYSIVUSTON TOTEUTTAMINEN ANGULAR- ALUSTALLA

Alexi Valojää

TUOTE-ESITTELYSIVUSTON TOTEUTTAMINEN ANGULAR-ALUSTALLA

Tämän opinnäytetyön päämääränä oli selvittää, miten selainpohjainen tuote-esittelysovellus kehitetään Angular-ohjelmistokirjastoa hyväksi käyttäen. Työn aikana selvitettiin, mitä tämän kaltainen sovellus tarvitsee sekä toiminnallisuutta, että käyttöliittymää silmällä pitäen, ja miten Angular-kirjastoa pystyi toteutuksessa hyödyntämään.

Opinnäytetyössä toteutettiin dynaaminen tuotelistaus joka linkittyy laajempiin tuote-esittelyihin, tuotelistauksen sivutus sekä tuotehaku. Listauksen sisältö asetettiin noudettavaksi ulkoisesta Firebase-tietokannasta. Sovelluksen kehityksessä otettiin huomioon responsiivinen ulkoasu, joka mahdollistaa sen käytön kaikilla laitteilla ja näyttökoilla, Bootstrap 4-tyylikirjastoa hyödyntäen. Tämän lisäksi huomiota kiinnitettiin ympäristön saavutettavuuteen, eli käytettävyyteen erilaisilla liikunta- ja näkörajoitteisten käyttäjien apuvälineillä.

Toteutuksen aikana saatiin selville, että Angular-kirjasto on ohjelmistollisesti raskas yksinkertaista sovellusta silmällä pitäen, ja että kevyemmät ohjelmakirjastot, kuten Vue tai React olisivat kyenneet toteuttamaan saman toiminnallisuuden kevyemmällä laitteistokuormituksella. Toisaalta Angular havaittiin helpommin laajennettavaksi alustaksi, mikäli sovellusta tulaisiin tulevaisuudessa kehittämään pidemmälle.

ASIASANAT:

Angular, Angular 2, Bootstrap, Javascript, saavutettavuus, verkkosovellus

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2021 | 21 pages

Aleksi Valojää

PRODUCING A PRODUCT INTRODUCTION WEBSITE WITH ANGULAR-LIBRARY

The goal of this thesis was to determine how a browser-based product introduction app could be developed using the Angular program library. The work involved determining what this type of application needs to be useful, concerning both its functionality and interface, and how the Angular-library could be applied in the production.

The thesis produced a dynamic product listing that is linked to wider product introductions, a pagination for the listing, as well as a product search function. The contents of the list were set to be imported from an external Firebase API. The production of the application took into consideration the responsive interface that can be used with all devices and display sizes by making use of the Bootstrap 4 style library. Apart from this, attention was given to accessibility of the interface, making it usable with tools intended for the sight- and mobility-impaired users.

During the production it was discovered that the Angular program library was heavy in terms of its resource use for a relatively simple application and that the same functionality could have been achieved with Vue or React libraries with lighter processing load. On the other hand, Angular was found to be easily expandable platform, if the application was to be developed further in the future.

KEYWORDS:

Angular, Angular 2, Bootstrap, Javascript, accessibility, web application

SISÄLTÖ

1 JOHDANTO	5
2 TEKNINEN TOTEUTUS	6
3 ANGULAR	7
3.1 Moduulit	7
3.2 Komponentit	8
3.3 Palvelut	8
3.4 Putket	8
3.5 RxJs	8
4 VAATIMUSMÄÄRITTELY	9
4.1 Tietokannan hyödyntäminen	9
4.2 Laajennettavuus	9
4.3 Sivutus	9
4.4 Saavutettavuus	10
4.5 Tuotehaku	10
5 TOTEUTUS	11
5.1 Sivuston komponentit	11
5.2 Sivuston palvelut	14
5.3 Firebase-tietokanta	15
5.4 Saavutettavuusmäärittely	16
5.5 Haku	18
5.6 Sivutus	18
6 LOPPUPÄÄTELMÄT	20
LÄHTEET	21

1 JOHDANTO

Tämä opinnäytetyö selvittää miten yrityskäyttöön tarkoitettun tuote-esittelysovelluksen suunnittelu ja toteuttaminen tapahtuu Angular 8-ohjelmakirjastoa hyödyntäen. Tarkoituksena on luoda mahdollisimman modulaarinen järjestelmä, joka on sovellettavissa pienin muutoksin useisiin eri yrityksiin ja ympäristöihin niin ohjelmistollisesti kuin visuaalisestikin.

Opinnäytetyö keskittyy kehityksen front-end-puoleen, sillä oletuksella, että käytettävä tuotedata on peräisin olemassa olevasta tietokannasta, jonka muutokset heijastuvat automaattisesti sivustoon. Tämä erottaa sen useimmista vastaavista palveluista, jotka yleensä muodostavat ja ylläpitävät omaa tietokantaansa, joka on synkronoitava mahdolliseen kolmanteen osapuoleen erillisellä back-end-ratkaisulla.

Front-end viittaa sovelluksen selaimen päässä tapahtuvaan toteutukseen, kuten ulkoasuun ja käyttäjän hallinnoimien toiminnallisuuksien, kuten sisältöhaun ja linkkien toimintaan. Back-end puolestaan merkitsee palvelimen puolella tapahtuvaa toiminnallisuutta, joka vastaa front-endistä tuleviin kutsuihin.

Työ jakautuu kolmeen vaiheeseen: vaatimusmäärittelyyn, suunnitteluun ja toteutukseen.

Vaatimusmäärittelyssä käydään läpi sovelluksessa tarvittavat ominaisuudet, suunnittelussa käydään läpi toteutuksessa tarvittavat teknologiat ja toteutus koostuu itse koodin työstämisestä.

2 TEKNINEN TOTEUTUS

Opinnäytetyössä kehitettiin verkkopalvelun front-end-ympäristö käyttäen seuraavia työvälineitä:

Työn rakenteen muodostuksessa käytettiin HTML5, eli "Hypertext Markup Language"-merkintäkieltä, joka määrittää kaikkien sivujen rakenteen ja sisällön. HTML keksittiin vuonna 1989, ja se on pysynyt kaikkien selainpohjaisten käyttöliittymien pohjana nykypäivään asti (Longmann, 1998).

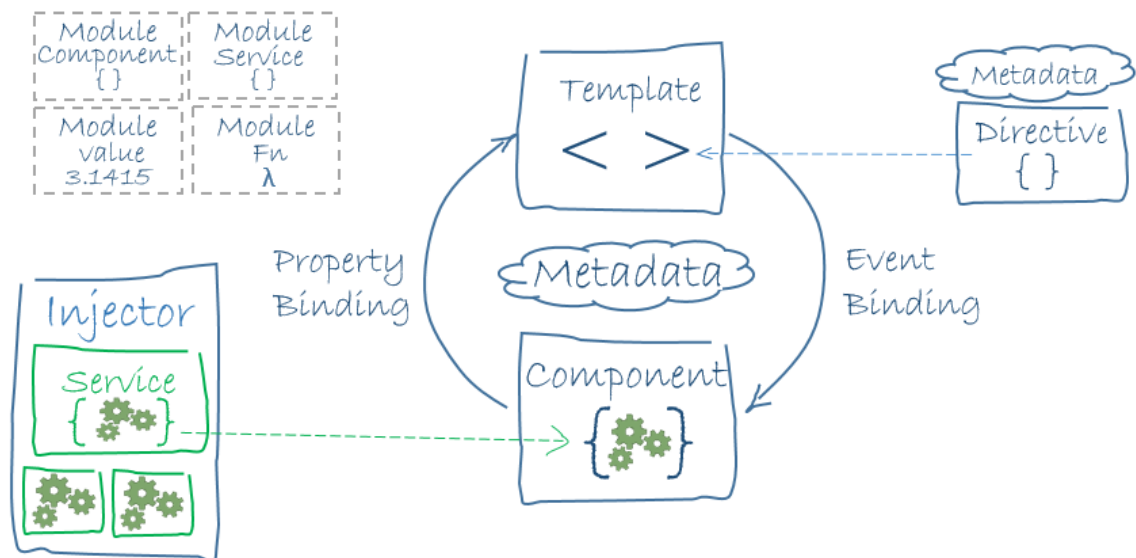
Sivuston ulkoasu, sekä sen responsiivinen rakenne toteutettiin CSS3:lla, eli "Cascading Style Sheet"-tyylinhallintakielellä. Front-endin ohjelmallinen puoli perustuu JavaScriptiin, verkkosovellusten yleisimpään käytössä olevaan ohjelmointikieleen, jota selain lukee ja suorittaa verkkosivun ajon aikana.

Toteutuksen pohjana oli Angular 2+ ohjelmointikirjasto, johon perehdytään tarkemmin seuraavassa luvussa. Kirjasto rakentuu TypeScriptin, Microsoftin JavaScriptille kehittämän avoimen lähdekoodin laajennuksen päälle, joka tulkitaan koodin ajon aikana tavalliseksi JavaScript-kieleksi. TypeScript tuo olio-ohjelmoinnin periaatteita JavaScript-ympäristöön mm. antamalla ohjelmoijalle mahdollisuuden rajoittaa käytettävien muuttujien tyyppejä, kuten kielen nimi antaa ymmärtää.

3 ANGULAR

Angular on Googlen kehittämä, Javascriptistä johdettuun Typescript-ohjelmointikieleen perustuva avoimen lähdekoodin ohjelmointikirjasto, joka on optimoitu sekä verkkopalveluiden, että mobiilisovellusten front-end-toteutusta silmällä pitäen. Yleisesti Angulariin viitataan Angular 2+:na, erotuksena sitä edeltäneestä AngularJS-kirjastosta. Uusin Angular-versio kirjoitushetkellä on Angular 9, mutta opinnäytetyön toteutuksessa hyödynnettiin Angular 8:aa (Gavigan, Dave 2018)

Angularin arkkitehtuuri koostuu ensisijaisesti moduuleista, komponenteista ja palveluista, joiden lisäksi on vielä mainittava putket. Kuva 1 esittää karkean mallinnuksen Angularin toiminnallisuuksista:



Kuva 1. Angular-sovelluksen arkkitehtuuri (Angular v2 Archive 2019).

3.1 Moduulit

Moduulit (module) ovat Angularin perusrakennuspalikka, jota määrittävät kaikkien muiden elementtien toiminnan. Angular-sovelluksessa on käytettävä ainakin yhtä moduulia, joka kertoo sovellukselle mitä komponentteja ja palveluita se käyttää. Monimutkaisemmissa sovelluksissa käytetään useita moduuleja määrittämään sovelluksen eri osa-alueiden käyttämät elementit toisistaan itsenäisesti ohjelmarakenteen selkeyttämiseksi. (Trivendi 2016)

3.2 Komponentit

Komponentit (engl. component) muodostavat Angular-sovelluksen näkyvän sisällön. Ne koostuvat elementin sivurakenteen ilmaisevasta pohjasta (template), joka on toteutettu ohjelmakomennoilla täydennetyllä HTML-kielellä, sekä TypeScript-kielellä toteutetusta metadatatista. Komponentit voivat sisältää toisia komponentteja, ja välittää dataa keskenään hierarkkisesti (Trivendi 2016).

3.3 Palvelut

Palvelut (engl. service) ovat komponenttien ulkopuolisia elementtejä, joita ei ole Angularin puitteissa sen tarkemmin määritetty. Niiden yleisin käyttötarkoitus on siirtää dataa vapaasti komponenttien välillä ilman, että sen tarvitsisi kulkea hierarkkisesti koko komponenttipuun lävitse (Trivendi 2016).

3.4 Putket

Putket (engl. pipe) ovat Angularin sisäänrakennettu toiminto syötetyn datan suodattamiseksi. Angulariin on valmiiksi rakennettu useita putkia, joilla manipuloida tapoja, joilla tietokantasisältöjä sovelluksessa näytetään, esimerkiksi date-putki voi muuntaa päivämäärän esittelytapaa käytettävän selaimen kielen perusteella. Kehittäjät voivat myös luoda omia putkia suodattaakseen dataa haluamillaan tavoilla (Angular 2021).

3.5 RxJs

RxJs ei ole osa Angularia, vaan erillinen Javascript-kirjasto, mutta useimmat Angular-sovellukset hyödyntävät sitä epäsynkronisten toimintojen (asynchronous action) toteutuksessa. Epäsynkroniset toiminnot ovat ohjelmanpätkiä, joiden tapahtuma-aika ja kesto ovat ohjelmoijalle tuntemattomia, joten ne on suoritettava muun koodin taustalla, etteivät ne jumittaisi sovellusta epämääräiseksi ajaksi. RxJs:ää hyödynnetään yleensä, kun käyttäjän suorittama toiminto, esimerkiksi napin painallus, aktivoi jonkin palvelun, tai jos sovelluksen käsittelemää dataa päivitetään reaaliajassa.

4 VAATIMUSMÄÄRITTELY

Kehitystä varten on määritettävä mitä vaatimuksia valmiille sovellukselle tulee olemaan. Tässä luvussa käsitellään, mitä tuote-esittelysivuston toteutus vaatii sekä toiminnallisuutensa, että käytettävyyden suhteen.

4.1 Tietokannan hyödyntäminen

Sovelluksessa käytettävä data on tuotava ulkoisesta tietokannasta Angularin tukemaa rajapintaa hyväksi käyttäen. Useimmat nykyaikaiset verkkosovellukset erottavat datasisältönsä erilliseen, sovelluksesta riippumattomaan tietokantaan, josta sisältöä voidaan noutaa millä tahansa kannan kanssa yhteensopivalla ohjelmistorajapinnalla, eikä tämä ole poikkeus. Tällä järjestelyllä on useita etuja, mm. se, että sisältöä voidaan hallita täysin eri ympäristössä, kuin loppukäyttäjille suunniteltu yksisuuntainen käyttöliittymä, parantaen tietoturvaa. Tietokanta sisältää tuotelistan, johon kuuluu tuotteen tunnus, tuotenimi, kuvaus ja linkki tuotekuvaan.

4.2 Laajennettavuus

Tuotelistan tuote-elementtiä klikkaamalla käyttäjän on päästävä käsiksi laajempiin tuotetietoihin. Tuotelistan tehtävä on antaa käyttäjälle kokonaiskuva tarjolla olevista tuotteista, mutta kaikkien tuotetietojen näyttäminen kerralla tekisi käyttöliittymästä sekavan ja listasta vaikean seurata. Perinteisillä verkkosivuilla tuotelinkkien klikkaaminen avaisi uusia tuotesivuja, mutta Angular-sovelluksen avulla klikattavan tuotteen tiedot on mahdollista ladata dynaamisesti samalle sivulle päivittämättä selainikkunaa.

4.3 Sivutus

Tuotelistaus täytyy olla hajautettavissa useammalle sivulle, kun se kasvaa liian pitkäksi. Liian suuri lista yhdellä sivulla on vaikeasti seurattava ja raskas ladata selaimelle. Listan hajauttaminen monelle sivulle helpottaa sen hahmottamista, ja nopeuttaa sivun näkyvän osuuden lataamista.

4.4 Saavutettavuus

Nykyaikaisten verkkoympäristöjen täytyy ottaa huomioon muutkin, kuin perinteisten hallintalaitteiden käyttäjät ollakseen kaikkien saavutettavissa. Sivuston täytyy olla täysin käytettävissä näkövammaisille tarkoitetuilla lukulaitteilla ja navigoitavissa puhtaasti näppäimistöä käyttäen. Sen tekstin täytyy myös olla selkeästi luettavaa, ja linkkien erotuttava selvästi muusta sisällöstä.

4.5 Tuotehaku

Kun sisältöä kertyy paljon, yksittäisten tuotteiden löytäminen käy nopeasti työlääksi. Tuotteiden täytyy olla haettavissa tuotelistauksesta nimen perusteella. Pitkää tuotelistausta täytyy olla mahdollista rajoittaa haluttujen tuotteiden löytämisen helpottamiseksi, ja tätä tarkoitusta varten tarvitaan tuotehaku, joka etsii niitä nimen perusteella.

5 TOTEUTUS

5.1 Sivuston komponentit

Angular-sovellusta rakennettaessa suunnittelu aloitetaan komponenteista, joista sivun ulkoasu koostuu. Yksinkertaisessa tuote-esittelyssä kolme komponenttia on riittävä määrä sivun toteutukseen: tuotelista, tuotelistan yksittäinen tuote, sekä laajennettu tuotenäkymä, joka ilmestyy käyttäjän näkyville tuotelinkkiä klikkaamalla.

Tuotelistan komponentti toistaa yksittäisen tuote-elementin komponenttia niin monta kertaa, kuin palvelussa on tuotteita. Kun komponentin metadatassa on lähdesijainti, josta tuote-elementit noudetaan, tämä listaelementti huolehtii niiden toistamisesta. Tässä vaiheessa on hyvä myös päättää sivuston ulkoasun asettelusta, että lopputuloksesta tulisi selkeä ja responsiivinen mobiililaitteita silmällä pitäen. Tähän tarkoitukseen sopiva väline on Bootstrap 4, CSS-tyylikirjasto joka huolehtii responsiivisesta asettelusta automaattisesti HTML:n tyyliluokkien avulla. Bootstrapia käyttäen tuote-elementit asettuvat siististi ruudukkoon, jossa rinnakkain olevien elementtien määrä riippuu käytettävän näyttölaitteen koosta (Bootstrap 4 2020).

Tuotelistassa on myös otettava huomioon vaatimusten mukainen sivutus. Ilman sivutusta tuotelista voi täytyä sadoista tuote-elementeistä samanaikaisesti. Sivutuksen avulla näkyville voi jättää halutun määrän elementtejä kerralla, ja listaa voi navigoida sivulinkkejä klikkaamalla. Sivutuksen voisi rakentaa alusta alkaen palveluna, mutta pyörän keksiminen uudestaan on turhaa, koska Angular-kirjastoon on saatavilla lukemattomia valmiiksi rakennettuja laajennuspaketteja eri tarkoituksiin, ja yksi näistä laajennuksista on nimeltään Ngx-Pagination-moduuli, jota voidaan kutsua sovelluksen päämoduulissa, ja sitä kautta hyödyntää sovelluksen kaikissa komponenteissa.

Ngx-Pagination voidaan ottaa käyttöön missä tahansa ngFor-komentoa käyttävässä listassa lisäämällä komentoon putkitus, jolla määritetään kerralla näytettävien elementtien määrä ja muut tekijät. Tämän lisäksi listan perään lisäksi kutsutaan moduuliin kuuluvaa sivulinkkikomponenttia, joka lisää sovellukseen sivunavigoinnin (Pagination for Angular 2020).

Yksittäistä tuote-elementtiä kuvaavan komponentin kohdalla täytyy ottaa huomioon, millaista dataa tuotteen kuvaukseen lopulta käytetään. Tämä riippuu tietysti esiteltävästä

tuotteesta ja sen ominaisuuksista, mutta tässä sovelluksessa tuotteen malli on yksinkertainen, ja sisältää ainoastaan id-tunnuksen, nimen, kuvalinkin ja tuotekuvauksen. Joissain tapauksissa tuotteen laajennetulla näkymällä voisi olla eri datalähde ja -sisältö, mutta tässä yksinkertaistetussa esimerkissä kaikki tiedot ovat peräisin samasta tietokannasta.

Ensin tuotteelle rakennetaan malli (model), jossa yllä mainitut tekijät määritetään Typescriptin konstruktorin (constructor) avulla. Kun mallia kutsutaan komponenteissa, se kertoo kyseiselle komponentille mitä tietoja tietokannasta kutsuttavan tietueen tulisi pitää sisällään (Angular v2 Archive 2019). Komponenttia rakennettaessa riittää kuitenkin, että sen HTML-sisältö on kohdallaan. Itse data tullaan myöhemmin kutsumaan palvelun (service) kautta. Tuote-elementin komponenttiin sisällytetään tuotteen nimi, pikkukuva ja lyhyt kuvaus, tuotteen laajemman kuvauksen komponenttiin taas nimi, suurempi kuva ja laajempi tekstikuvaus (Kuva 2, Kuva 3). Kuva 4 näyttää, miltä lopputulos näyttää käyttäjälle.

```
1  export class Product {
2      public id: number;
3      public name: string;
4      public shortDescription: string;
5      public longDescription: string;
6      public thumbnailUrl: string;
7      public pictureUrl: string;
8
9      constructor(
10         id: number,
11         name: string,
12         shortDesc: string,
13         longDesc: string,
14         thumbUrl: string,
15         picUrl: string) {
16         this.id = id;
17         this.name = name;
18         this.shortDescription = shortDesc;
19         this.longDescription = longDesc;
20         this.thumbnailUrl = thumbUrl;
21         this.pictureUrl = picUrl;
22     }
23 }
```

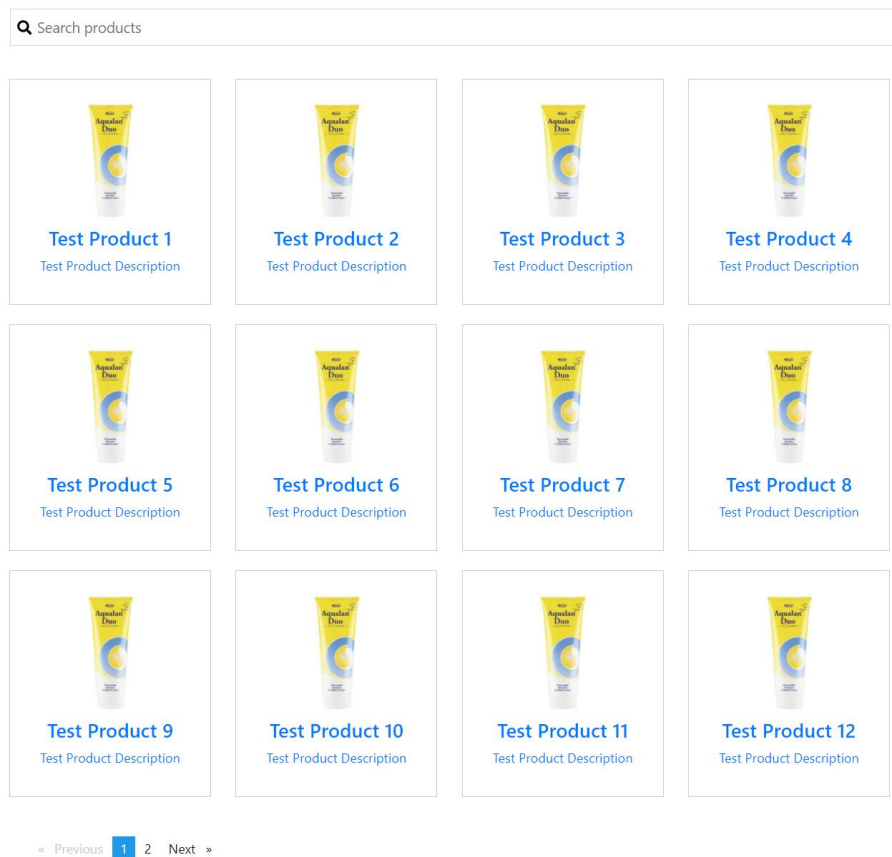
Kuva 2. Tuotemalli.

```

1 <div class="row">
2   <div class="col-12">
3     <div class="form-group">
4       <input type="text" class="form-control search" placeholder="Search products" [(ngModel)]="searchTerm" />
5     </div>
6   </div>
7 </div>
8
9 <div class="row product-list">
10  <app-product-small
11  *ngFor="let productElement of products | productFilter: searchTerm | paginate: { itemsPerPage: 2, currentPage: p }"
12  [product]="productElement"
13  class="col-lg-3 col-md-4 col-sm-6 col-xs-12"></app-product-small>
14
15 </div>
16 <div class="row pagination">
17   <div class="col-sm-12 pagination-controls">
18     <pagination-controls (pageChange)="p = $event"></pagination-controls>
19   </div>
20 </div>
21 </div>

```

Kuva 3. Tuotelistaus.



Kuva 4. Tuotelistaus selaimessa.

5.2 Sivuston palvelut

Angular-sovelluksessa palvelu on sovelluksen taustalla toimiva elementti, jota komponentit hyödyntävät toiminnoissaan. Palveluiden yleisin käyttötarkoitus on välittää informaatiota komponenttien välillä, ja näin välttää monimutkaiset järjestelyt, joita tarvittaisiin toimintojen tai tietueiden siirtelyyn pitkien komponenttipuiden halki (Angular v2 Archive 2019). Tässä sovelluksessa palvelua hyödynnetään tuotedatan noutamiseen tietokannasta, sekä sen välittämiseen tuotelistaukseen ja pidempään tuotekuvaukseen.

Koska kyseessä on yksinkertainen sovellus, sillä on ainoastaan kaksi palvelua, tuotepalvelu (ProductService) sekä datapalvelu (DataService). Tuotepalvelu puskee sivulle tuotelistauksen ja kuuntelee tuotelinkkien klikkaustapahtumia. Kun tuotelinkkiä klikataan, tuotepalvelu aktivoi tuotteiden hakumetodin, joka avaa laajennetun tuotekuvauksen sivulle. Datapalvelu puolestaan hakee sovellukseen ulkopuolisesta tietokannasta tulevan raakadatan ja käsittelee sen tuotepalvelun käyttöön. Kuva 5 esittää, miten tuotepalvelu noutaa tuotelistauksen. Toisin kuin valmiissa versiossa, kuvassa näkyy myös tietokantaan vietävä tuotelistaus, koska käytetty Firebase-tietokanta täytettiin testiä varten saman sovelluksen kautta.

```

1 import { Product } from './product.model';
2 import { EventEmitter, Injectable } from '@angular/core';
3
4 @Injectable({providedIn: 'root'})
5 export class ProductService {
6   productSelected = new EventEmitter<Product>();
7
8   private products: Product[] = [
9     new Product(
10      1,
11      'Test Product 1',
12      'Test Product Description',
13      'Test Product Long Description',
14      '../../../../assets/img/testthumb.jpg',
15      '../../../../assets/img/testthumb.jpg'),
16     new Product(
17      2,
18      'Test Product 2',
19      'Test Product Description',
20      'Test Product Long Description',
21      '../../../../assets/img/testthumb.jpg',
22      '../../../../assets/img/testthumb.jpg'),
23     new Product(
24      3,
25      'Test Product 3',
26      'Test Product Description',
27      'Test Product Long Description',
28      '../../../../assets/img/testthumb.jpg',
29      '../../../../assets/img/testthumb.jpg'),
30     new Product(
31      4,
32      'Test Product 4',
33      'Test Product Description',
34      'Test Product Long Description',
35      '../../../../assets/img/testthumb.jpg',
36      '../../../../assets/img/testthumb.jpg')
37   ];
38
39
40   getProducts() {
41     return this.products.slice(); //Returns a copy of the array
42   }
43
44   getProduct(i: number) {
45     return this.products[i];
46   }
47
48   setProducts(products: Product[]) {
49     this.products = products;
50   }
51 }

```

Kuva 5. Tuotepalvelu.

5.3 Firebase-tietokanta

Sovellus on suunniteltu niin, että sen kanssa voidaan hyödyntää lähes mitä tahansa tietokantaratkaisua. Tätä silmällä pitäen sen sisältödatan tuontia varten on rakennettu oma palvelu, DataService, jonka toiminnallisuus voidaan muokata vastaamaan datan lähdettä vaikuttamatta sovelluksen muuhun toiminnallisuuteen millään tavalla. Testausta varten lähde on kuitenkin pidetty yksinkertaisena, pelkkänä JSON-tietokantana ilman ylimääräistä back-end-toiminnallisuutta, kuten back-endissä tapahtuvaa sivutusta tai hakua. Tähän tarkoitukseen on käytetty Googlen FireBase-tietokantaa, jonka JSON-tiedostosta datapalvelu koostaa tuotelistauksen.

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 import { Product } from '../product/product.model';
5 import { ProductService } from '../product.service';
6
7 @Injectable({providedIn: 'root'})
8 export class DataService {
9     constructor(
10        private http: HttpClient,
11        private productService: ProductService) {}
12
13     storeProducts() {
14         const products = this.productService.getProducts();
15
16         this.http.put('https://opproject-f49a0-default-rtdb.firebaseio.com/products.json', products)
17             .subscribe(response => {
18                 console.log(response);
19             });
20     }
21
22     fetchProducts() {
23         this.http.get<Product[]>('https://opproject-f49a0-default-rtdb.firebaseio.com/products.json')
24             .subscribe(products => {
25                 this.productService.setProducts(products);
26             });
27     }
28 }

```

Kuva 6. Datapalvelu.

5.4 Saavutettavuusmäärittely

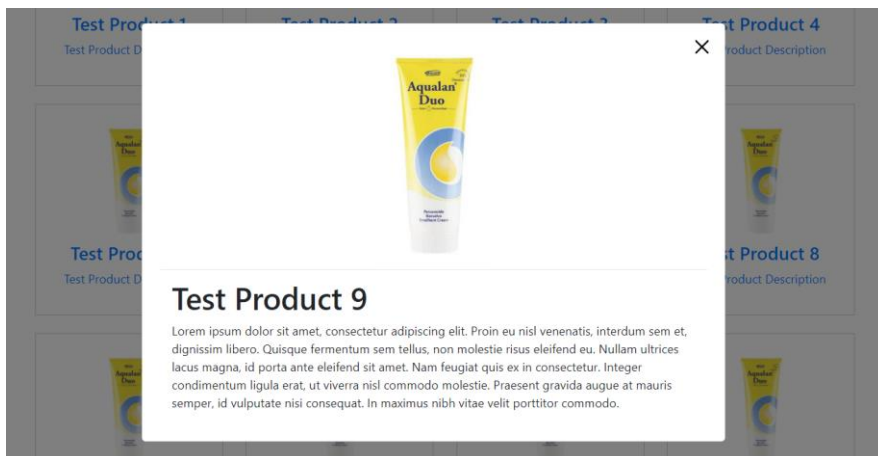
Verkkosivujen saavutettavuudessa on kyse niiden käytettävyydestä erilaisia apuvälineitä tarvitseville ihmisille. Kuva ilman tekstiselitystä ei kerro mitään sokeiden lukulaitteille, ja linkit, joita ei voi navigoida näppäimistöllä ovat liikuntarajoitteisen käyttäjän saavuttamattomissa. World Wide Web Consortium (W3C) on kehittänyt näitä erityistarpeita varten saavutettavuusmäärittelyn, jota kaikkien nykyaikaisten verkkosivustojen pitäisi noudattaa, mikäli niiden halutaan olevan todella kaikkien käytettävissä. Olemassa olevan palvelun saavutettavuuden toteuttamisesta voisi itsessään tehdä kokonaisen opinnäytetyön, mutta kun vaatimukset on otettu huomioon jo toteuttamisvaiheessa, ei tämän päämäärän saavuttaminen ole erityisen monimutkaista.

Ensiksi on otettava huomioon värikontrastit sivun ulkoasussa. Liian haaleat tai huonosti yhteensopivat värit voivat tehdä tekstin lukemisen vaikeaksi keskivertokäyttäjällekin, puhumattakaan heikkonäköisistä ja värisokeista. Myös linkkien ja painikkeiden erottuvuus normaalista tekstistä on otettava huomioon (W3 Accessibility 2020).

Saavutettavuutta toteutettaessa on myös otettava huomioon, miten hyvin sivu on seurattavissa näkövammaisille suunnatuilla lukulaitteilla. Lukulaitteet tulkitsevat sivurakenteen otsikkotasojen perusteella, joten on tärkeää, että sivuston otsikkohierarkia on kohdallaan. Kullakin sivulla tulee olla vain yksi pääotsikko, ja alaotsikoiden tulee esiintyä loogisessa järjestyksessä, eli kolmannen tason otsikon täytyy seurata toista tasoa, ja sen sisällön täytyy jollakin tapaa liittyä toisen tason otsikkoon, jonka alla se sijaitsee. Toisin kuin artikkelitekstejä sisältävällä sivustolla, tuotelistauksessa otsikkopuu on harvoin kahta tasoa syvempi, tuoteotsikot pääotsikon alla (ibid).

Ulkoasun puolesta, tämä sivusto avaa tuotelinkkejä klikattaessa modaali-ikkunan muun sisällön päälle kuvan osoittamalla tavalla, mikä tekee auki olevan tuotteen heti selväksi näkeväälle käyttäjälle. Lukulaitteen näkökulmasta tämä kuitenkin vain lisää uuden tekstikentän sivustolle, ja ilmoittaa tästä käyttäjälle. Tuotekuville tulee antaa tyhjä alttekstiarvo, että lukulaitteet ymmärtävät, että ne eivät lisää merkittävää sisältöä sokean käyttäjän näkökulmasta. Sen sijaan sulkunapin ruksi-ikonin tulee sisältää aria label-tekstiattribuutti, joka kertoo sen käyttötarkoituksen sokealle käyttäjälle (ibid).

Lopuksi on vielä otettava huomioon sivuston käytettävyys pelkkää näppäimistöä käyttäen, sekä sokeiden, että liikuntarajoitteisten käyttäjien huomioimiseksi. Jokaisen linkin ja napin tulee olla kohdistettavissa tabulaattorinäppäimellä, ja kohdistuksen tulee liikkua niiden välillä loogisessa järjestyksessä. Myös tuoteikkunoiden sulkeminen täytyy olla mahdollista esc-näppäimellä (ibid).



Kuva 7. Yksittäisen tuotteen kuvaus.

5.5 Haku

Ilman erillistä back-end-sovellusta, haku on suoritettava suodattamalla front-endin sisältöä tekstisyötteen perusteella. Angularissa on tätä tarkoitusta varten putkitustoiminto (piping), jonka avulla olemassa olevasta datataulukosta voidaan seuloa sisältöä.

Angularin putki toteutetaan yksinkertaisena funktiona, joka palauttaa kaikki lähteestä tulevat elementit, jotka sisältävät hakusanaa vastaavan merkkijonon. Kun putki on sisällytetty sovelluksen moduuliin, sitä voidaan kutsua missä tahansa komponentissa, ja se muuntaa tai seuloa kaikki elementit, joita sen funktio koskee. Haun tapauksessa tekstisyöte lisätään tuotelistauksen yhteyteen, ja sen sisältö sidotaan hakumuuttujaan Angularin ngModel-komennolla. Tällöin, aina kun tekstisyötteen sisältöä muutetaan, putkea kutsutaan uudelleen seulomaan haettavat sisällöt uudelleen. Putkien käyttäminen varomattomasti voi aiheuttaa raskasta prosessorikuormitusta sovellukselle, mutta tämän haun yksinkertainen merkkijonoseulonta ei aiheuta ongelmia tämän suhteen (Filter pipe in Angular 2021).

```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 import { Product } from './product.model';
4
5 @Pipe({
6   name: 'productFilter'
7 })
8
9 export class ProductFilterPipe implements PipeTransform {
10  transform(products: Product[], searchTerm: string) {
11    if (!products || !searchTerm) {
12      return products;
13    }
14
15    return products.filter(product =>
16      product.name.toLowerCase().indexOf(searchTerm.toLowerCase()) !== -1);
17  }
18 }
```

Kuva 8. Tuotehakuputki.

5.6 Sivutus

Sivutus on välttämätön ominaisuus millä tahansa pitkiä listoja sisältävällä sivustolla navigoinnin ja käytettävyyden helpottamiseksi. Lisäksi sivutus voi keventää sovelluksen aiheuttamaa kuormitusta selaimelle, mutta koska sovelluksen nykyinen versio sisältää

ainoastaan front-end-osuuden, ei sivutuksen hallinnointi back-endin puolella ole tämän opinnäytetyön tapauksessa mahdollista toteuttaa.

Tämä sovellus hyödyntää valmista ratkaisua nimeltä Ngx-pagination. Se voidaan asennuksen jälkeen lisätä sovelluksen moduuliin ja käytettyyn komponentin listaelementtiin putkena, samaan tapaan kuin yllä mainittu haku, ja mukaan sisällytetty sivunhallinta lisätään listan jälkeen. Ngx-pagination on monipuolinen Angular-lisäosa, joka ottaa huomioon mm. saavutettavuusvaatimukset, kuten eteen- ja taakse-nuolikonien aria-label attribuutit, sekä näppäimistöhallinnan automaattisesti (Pagination for Angular 2020).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { NgxPaginationModule } from 'ngx-pagination';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';
import { ProductListComponent } from './product/product-list/product-list.component';
import { ProductSmallComponent } from './product/product-small/product-small.component';
import { ProductLargeComponent } from './product/product-large/product-large.component';
import { ProductFilterPipe } from './product/product-filter.pipe';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductSmallComponent,
    ProductLargeComponent,
    ProductFilterPipe
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    NgxPaginationModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Kuva 9. Sovelluksen moduuli.

6 LOPPUPÄÄTELMÄT

Toteutuksen aikana selvisi, että monet Angular-sovelluksissa käytetyt ominaisuudet eivät olleet hyödynnettävissä tämän projektin rajoissa. Sovellus jättää hyödyntämättä mm. Angularin reititysominaisuutta, joka mahdollistaa sovelluksen osioiden aktivoimisen käytetyn URL-osoitteen kautta. Tämä ominaisuus on erittäin käyttökelpoinen, kun koko sivusto on toteutettu Angular-rajapinnan kautta. Mutta tämä tuotelistaussovellus on tehty sitä mahdollisuutta silmällä pitäen, että se voidaan upottaa jo olemassa oleviin verkkoympäristöihin, joissa Angularin reititys saattaisi aiheuttaa konflikteja alkuperäisen ympäristön reitityksen kanssa.

Toinen käyttämättä jätetty ominaisuus on NgRx-tilanhallinta. NgRx ei ole varsinaisesti osa Angular-ympäristöä, mutta se on laajalti Angular-sovelluksissa hyödynnetty tilanhallintakirjasto, jonka avulla sekä käyttäjä että palvelin voivat dynaamisesti päivittää sovelluksen hyödyntämää dataa. Se on erittäin hyödyllinen työkalu monimutkaisemmissa Angular-sovelluksissa, joissa näytetty sisältö saattaa päivittyä reaaliajassa, esimerkiksi verkkokauppaympäristössä, jossa käyttäjä voi nähdä saatavilla olevien tuotteiden tarkan määrän, joka päivittyy sitä mukaa kun niitä ostetaan. Tämä tuote-esittelysovellus on kuitenkin sisällöltään enimmäkseen staattinen, eivätkä aktiivisesti päivittyvät tuotekuvaukset hyödytä tavallista käyttäjää tarpeeksi oikeuttaakseen uuden ominaisuuden lisäystä sovellukseen (NgRx 2021).

Vaikka opinnäytetyön projekti oli suhteellisen yksinkertainen sovellus, sen luonti oli pitkälinen prosessi, jonka aikana testattiin useita teknologioita ja ominaisuuksia, jotka eivät päätyneet lopulliseen tuotantoon, koska ne eivät lopulta parantaneet sovelluksen toiminnallisuutta tai helpottaneet sen käyttöä. Angular-ohjelmakirjasto osoittautuikin suhteellisen raskaaksi ratkaisuksi yksinkertaiselle sovellukselle, ja monet sen tarjoamista eduista jäivät hyödyntämättä. Kilpailevat Vue- tai React-kirjastot olisivat kyenneet samaan toiminnallisuuteen huomattavasti pienemmällä laitteistokuormituksella, ja niiden upottaminen toisiin ympäristöihin ja sovelluksiin olisi huomattavasti helpommin toteutettavissa kuin Angularin. Toisaalta, Angular-sovellus on helpommin laajennettavissa, ja mikäli sovellus haluttaisiin muuttaa front-endin kautta ylläpidettäväksi, Angular olisi siihen tehtävään enemmän omiaan.

LÄHTEET

Boxhall, Alice - Dodson, Robin - Gash, Dave – Kearney, Meggin: Accessibility. Viitattu 12.9.2020.

<https://developers.google.com/web/fundamentals/accessibility>

Angular v2 Archive. Viitattu 15.12.2019.

<https://v2.angular.io/docs/ts/latest/guide/architecture.html>

Angular 2021. Viitattu 3.3.2021.

<https://angular.io/>

Bootstrap 4. Viitattu 12.9.2020.

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>

Filter pipe in Angular 2021. Viitattu 3.3.2021

<https://www.youtube.com/watch?v=1TFSibbnkj0>

Gavigan, Dave: The History of Angular 2018. Viitattu 25.5.2021

<https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7>

Longman, Addison Wesley: A History of HTML 1998. Viitattu 25.5.2021

<https://www.w3.org/People/Raggett/book4/ch02.html>

NgRx 2021. Viitattu 29.4.2021.

<https://ngrx.io/docs>

Pagination for Angular. Viitattu 20.10.2020.

<https://www.npmjs.com/package/ngx-pagination>

Trivedi, J. 2016. Basic Architecture of Angular 2 Applications. Viitattu 5.1.2020.

<http://www.c-sharpcorner.com/article/basic-architecture-of-angular-2-applications/>

W3 Accessibility. Viitattu 12.9.2020.

<https://www.w3.org/standards/webdesign/accessibility>

