LAB University of Applied Sciences

Hitsauskoneen lisävarustevalitsimen toteutus

LAB-ammattikorkeakoulu
Tekniikka (AMK), Tietotekniikan koulutusohjelma
Syksy 2020
Hannu Holkeri

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Holkeri, Hannu	Opinnäytetyö, AMK	Syksy 2020
	Sivumäärä 37	

Työn nimi

Hitsauskoneen lisävarustevalitsimen toteutus

Tutkinto

Tieto- ja viestintätekniikka (AMK)

Ohjaavan opettajan nimi, titteli ja organisaatio

Matti Wélin, yliopettaja, LAB-ammattikorkeakoulu

Toimeksiantajan nimi, titteli ja organisaatio

Juhana Enqvist, CDO, Kemppi Oy

Tiivistelmä

Opinnäytetyön aiheena oli suunnitella ja toteuttaa hitsauskoneen lisävarustevalitsin upotettuna erilliselle web-sivulle. Työn toimeksiantajana oli Kemppi Oy.

Opinnäytetyössä suunniteltiin ja toteutettiin responsiivinen ja interaktiivinen sovellus, joka pystyy näyttämään useita eri hitsauskone-lisävaruste -yhdistelmiä ja tukee useita eri kieliä. Interaktiivisuuden haluttiin näkyvän sovelluksessa pyöritettävänä hitsauskone- /lisävaruste -mallina. Valitsimesta haluttiin myös saada käyttäjän tekemät valinnat käyttäjän sähköpostiin tai Kempin markkinoinnille käyttäjän niin halutessa.

Lisävarustevalitsin koostuu pääsääntöisesti käyttöliittymästä, mutta on sillä myös taustapalvelu, jonka tietoturva on yksi tämän työn tavoitteista. Sähköpostin lähettäminen julkisen rajapinnan takaa ison yrityksen nimissä pitää olla hyvin suojattu väärinkäytöltä.

Opinnäytetyön lopputuloksena syntyi Angular-pohjainen toteutus, joka myös julkaistiin syksyllä 2020 upotettuna web-sivustoon.

Asiasanat

Angular, Angular Elements, Amazon Web Services

Abstract

Author(s)	Type of Publication	Published
Holkeri, Hannu	Thesis, UAS	2020
	Number of Pages 37	

Title of Publication

Accessory selector for welding machine

Name of Degree

Bachelor of Information technology

Name, title and organization of the supervising teacher

Matti Wélin, senior teacher, LAB University of Applied Sciences

Name, title and organization of the client

Juhana Enqvist, CDO, Kemppi Oy

Abstract

The goal of this work was to design and implement accessory selector for a welding machine. The work includes embedding the project into existing website using Angular Elements. The selector has to be responsive and interactive.

The selector consists of Angular frontend and serverless backend hosted on Amazons cloud. The backend consists of single email sending Lambda-function of which security was important point and it is one of this works goals because email sending security from public interface in name of any company should be taken seriously.

The final product was published in autumn 2020. It was built with Angular Elements and then embedded into a website.

Keywords

Angular, Angular Elements, Amazon Web Services

Sisällys

1 Johdanto....1
2 Nykyaikainen hitsauskone....2
3 Responsiivinen web-suunnittelu....5
3.1 CSS Fluid Grid....5
3.2 Mediakyselyt....6
3.3 Responsiivinen media....6
3.4 Viewport metatieto....6
4 Käytetyt tekniikat....8
4.1 Angular....8
4.1.1 Angularin arkkitehtuuri....8
4.1.2 Angular Elementit....10
4.2 Amazon Web Services....14
4.2.1 Lambda-funktio....15
4.2.2 API Gateway....15
5 Käytetyt kirjastot....16
5.1 NGX-Translate....16
5.2 Lodash....19
5.3 Three Sixty....21
5.4 Angular Flex-Layout....22
5.5 Handlebars....23
6 Käyttöliittymä....25
6.1 Suunnittelu....25
6.1.1 Päätöspuu....25
6.1.2 Mahdolliset valinnat....26
6.1.3 Optimointi – kuvien lataus....26
6.2 Toteutus....26
6.2.1 Päätöspuun luominen....26
6.2.2 Optimoinnin toteutus....27
6.2.3 Infopisteet....28
6.2.4 Upotus web-sivustolle....30
7 Taustapalvelu....31
7.1 Suunnittelu....31
7.2 Toteutus....31

8 Yhteenveto.....34
Lähteet.....35

1 Johdanto

Nykyään moneen laitteeseen voi hankkia lisävarusteita ja tämä koskee myös hitsauskoneita. Lisävarusteita voi olla tarjolla useita erilaisia ja asiakkaan voi olla hankala hahmottaa jokaisen käyttötarkoitusta paperiselta esitteeltä tai staattiselta web-sivulta. Eikö olisi hienoa nähdä miten jokainen lisävaruste ”istuu” omaan hitsauskoneeseen? Tämä työ esittää yhden ratkaisun tähän ongelmaan.

Kemppi Oy on vuonna 1949 perustettu suomalainen hitsausalan edelläkävijä. Se kehittää älykkäitä laitteita, hitsaustuotannon hallintaohjelmistoja ja näitä tukevia asiantuntijapalveluja niin vaativiin teollisiin sovelluksiin kuin kuluttajan tarpeisiin. Kempin liikevaihto oli vuonna 2019 noin 150 miljoonaa euroa ja sen pääkonttori sijaitsee Lahdessa. Kemppi työllistää noin 800 henkilöä 17 maassa. (Kemppi Oy 2020a.)

Tämän työn tavoitteena on suunnitella ja toteuttaa Kemppi Oy:n uudelle hitsauskoneelle responsiivinen lisävarustevalitsin, joka upotetaan Kempin julkisille web-sivuille. Koska Kemppi käyttää paljon Angular-ohjelmistokehystä, toteutetaan myös tämä työ sillä. Lisävarustevalitsimen päätarkoitus on näyttää asiakkaalle, mitä lisävarusteita hän tarvitsee hitsauskoneeseensa, jotta ympäristön ja hitsaustyön asettamat vaatimukset täyttyvät. Työn teoriaosuudessa selvitetään Angularin periaatteita, käytettyjä kirjastoja ja olemassa olevalle sivustolle upottamista.

Valitsimessa tehdyt valinnat pitää pystyä lähettämään käyttäjän sähköpostiin. Mikäli käyttäjä haluaa ottaa yhteyttä suoraan markkinointiin, pitää valinnat tallentaa selaimeen evästeenä ja ohjata hänet Kempin sivuilla olevalle yhteydenottolomakkeelle. Itse valitsimen sekä sähköpostin pitää tukea kääntämistä useille eri kielille.

Kappaleessa kaksi käydään läpi, millainen on nykyaikainen hitsauskone, ja hieman sitä, että millaisia lisävarusteita sellaiseen on saatavilla. Kappaleessa kolme käydään läpi responsiivisen web-suunnittelun perusteita ja käsitteitä. Neljännessä kappaleessa on avattu työssä käytettyä Angular-ohjelmistokehystä ja Amazonin pilvipalveluita. Tämän jälkeen viidennestä kappaleesta löytyy työssä käytetyt kirjastot ja niiden toiminta. Kuudennessa kappaleessa päästään suunnittelemaan ja toteuttamaan itse käyttöliittymää Angularilla. Viimeisenä kappaleena, kappale seitsemän on työn taustapalvelun suunnittelu ja toteutus.

2 Nykyaikainen hitsauskone

Nykyaikainen MIG/MAG (kaasukaari)-hitsauskone muodostuu virtalähteestä, langansyöttölaitteesta, hitsauspolttimesta ja monitoimikaapelista, sekä maattokaapelista että -puristimista ja mahdollisesta suojakaasulaitteistosta. Usein nämä osat ovat koottu saman kuoren alle sisään kompaktiksi hitsauskoneeksi, mutta vaativaan hitsaukseen laitteet kootaan yleensä erillisistä moduuleista. Nykyaikainen hitsauskone on esitettynä seuraavassa kuvassa (Kuva 1). (Wikipedia 2020a.)

Kuva 1. Kemppi X5 FastMig hitsauskone (Kemppi Oy 2020b)

Edellä mainittujen osien lisäksi on hitsauskoneesta riippuen, tarjolla useita lisävarusteita helpottamaan erilaisia hitsaustilanteita. Esimerkiksi Kempin X5 FastMig -hitsauskoneeseen saa hankittua lisävarusteena muun muassa kevenninvarren, jonka avulla hitsaaja voi helposti väistää lattialla olevia esteitä, ja samalla hitsaajan ranteeseen

kohdistuva rasitus vähenee (Kemppi Oy 2020b). Seuraavassa kuvassa (Kuva 2) on X5 FastMig hitsauskone kevenninvarrella ja 4-pyöräisellä kärryllä.

Kuva 2. Kemppi X5 FastMig hitsauskone (Kemppi Oy 2020b)

On myös lisävarusteita, jotka mahdollistavat langansyöttölaitteen ripustamisen, oli kyse sitten yhdestä tai kahdesta laitteesta. Käyttämällä nosturia puomin ja ripustimen välissä saa lisää joustavuutta langansyöttölaitteen toimintaan. Joihinkin hitsauskoneisiin saa lisävarusteena toisen langansyöttölaitteen, joka helpottaa työskentelyä varsinkin silloin,

kun jatkuvasti hitsataan kahdella eri täytelangalla. Seuraavassa kuvassa (Kuva 3) on hitsauskone usealla lisävarusteella kuten puomiripustimella ja toisella langansyöttölaitteella. (Kemppi Oy 2020b.)

Kuva 3. Kemppi X5 FastMig hitsauskone (Kemppi Oy 2020b)

Nykyään hitsauskoneenkin saa yhdistettyä pilvipalveluun. Esimerkiksi Kempin WeldEye-hitsaushallintaohjelmistolla voidaan hallita hitsaukseen liittyvää dokumentointia, raportointia ja hitsausprosessin seurantaa (WeldEye 2020).

3 Responsiivinen web-suunnittelu

Web-sivut on jo alun perin ajateltu mediana, joka toimii erilaisissa ympäristöissä. Web-sivuja luodaan HTML:än (Hypertext Markup Language) avulla, joka on avoimesti standardoitu kuvauskieli. Sen alkuperäinen tarkoitus oli pikemminkin kuvata web-sivun rakennetta kuin sen ulkoasua, mutta sivujen tekijät halusivat myös paremmat mahdollisuudet vaikuttaa sivun ulkoasuun. Aluksi selainvalmistajat vastasivat tarpeeseen luomalla omia HTML-standardiin kuulumattomia elementtejä. Osa näistä elementeistä otettiin HTML-standardiin mukaan, mutta standardi on vuosien saatossa muuttunut eikä se tue enää kaikkia aiempia elementtejä. HTML:n ulkoasun luonnissa käytetään nykyään erillisiä tyyliohjeita, eli CSS-kieltä (Cascading Style Sheet). CSS:ään siirtyminen yksinkertaisti HTML:n rakennetta ja myös helpotti sivujen päivittämistä ja luomista. (Wikipedia 2020b.)

CSS:n tultua kehittäjien oli helpompaa määritellä sivun ulkoasua. Responsiivisuudesta ei kuitenkaan ollut tietoa, sillä paperimaailmasta tulleet suunnittelijat eivät vielä osanneet ajatella responsiivisuutta, vaan aluksi luotiin kiinteitä web-sivuja, jotka vaativat tiettyjä selainikkunan kokoja toimiakseen. Tällaisia sivuja tulee vielä vastaan aika ajoin. CSS kuitenkin tarjosi työkalut responsiiviseen ulkoasuun jo tullessaan. (Jyväskylän yliopiston informaatioteknologian tiedekunta 2016.)

Responsiivinen web-suunnittelu on ollut merkittävässä osassa jo pitkä aikaa. Mobiililaitteiden, niin puhelinten kuin tablettienkin, suosion lisääntymisen myötä web-sivuja käytetään paljon erikokoisilla näytöillä. Responsiivisuus on käyttäjän kannalta tärkeää, sillä nykyään web-sivujen käyttäjät odottavat sivuston skaalautuvan näytölle sopivaksi oli käytettävä laite sitten mikä tahansa. (Barron B. 2018.)

Responsiivisuus web-sivuilla on myös yrityksen kannalta tärkeää, sillä hyvin suunniteltu responsiivinen sivusto on käyttäjäystävällisempi ja se voi luoda yrityksestä paremman kuvan käyttäjälle. (Barron B 2018.)

3.1 CSS Fluid Grid

CSS:n mukautuva ruudukko (fluid grid) toimii kuten muutkin ruudukot. Sen avulla voidaan järjestellä elementtejä sivulla visuaalisesti huomiota kiinnittävällä tavalla. Toisin kuin tavalliset ruudukot, mukautuvan ruudukon koko muuttuu näytön koon mukaan ja se mukautuu mihin tahansa leveyteen, koska se käyttää suhteellisia mittayksiköitä kuten prosentteja tai em-yksiköitä. (Barron B. 2018.)

3.2 Mediakyselyt

Mediakyselyt (mediaqueries) auttavat responsiivisuuden luonnissa eri näyttöko'oille. Maallikon termeissä mediakyselyt auttavat päättelemään näytön koon ja sen mukaan käyttää tiettyjä CSS-tyylejä. (Barron B. 2018.)

Mediakyselyjen avulla on siis mahdollista määrittää eri näyttöko'oille omia tyylejä. Esimerkiksi voidaan body elementin taustaväri vaihtaa, jos sivustoa katsotaan alle 600px levyisellä näytöllä. Tästä on esimerkki kuvassa (Kuva 4), jossa alle 600px näytöllä body-elementin taustaväriksi asetetaan oliivinvihreä. (Barron B. 2018.)

```
@media screen and (max-width: 600px) {
  body {
    background-color: olive;
  }
}
```

Kuva 4. Mediakysely

3.3 Responsiivinen media

Kolmas responsiivisen web-suunnittelun perusasia on responsiivinen tai joustava media. Nykyajan web-sivut käyttävät paljon kuvia, videoita ja muita media tiedostoja. On siis ehdotonta, että nämä mediat skaalautuvat eri kokoisille näytöille. (Barron B. 2018.)

Normaalisti kuville määrättäisiin koko tyyliohjeen avulla, mutta se ei toimi responsiivisella sivulla, koska kiinteät mitat eivät ole responsiivisia. Kiinteiden mittojen sijaan voidaan kuvalle määrittää ”max-width” ominaisuus ja asettaa se sataan prosenttiin. Tämä mahdollistaa sen, että kuva ei ylitä ympäröivää elementtiä ja pysyy nätisti sen sisällä. (Barron B. 2018.)

3.4 Viewport metatieto

Selaimen ikkuna (viewport) on alue, jolla web-sisältöä voidaan näyttää. Usein ikkuna ei ole samankokoinen kuin renderöity sisältö. Tämän seurauksena selain lisää vierityspalkit, jotta käyttäjä voi liikutella web-sisältöä ja näin nähdä kaiken sisällön. (Mozilla 2020.)

Laitteet kapeammilla näytöillä renderöivät sivut virtuaalisessa ikkunassa, joka on usein leveämpi kuin laitteen näyttö. Tämän jälkeen selain kutistaa näkymän pienemmäksi, jotta sivu näkyy paremmin. Esimerkiksi laiteessa, jossa on 640 pikseliä leveä näyttö, voi laitteen selain renderöidä näkymän virtuaaliseen ikkunaan, jonka leveys on 980 pikseliä. Selain kutistaa tämän virtuaalisen näkymän siten, että se mahtuu laitteen 640 pikseliä leveälle näytölle. Tämän ansiosta useat sivut, jotka eivät ole suunnattu pienille näytöille, näyttävät paremmilta, sillä muuten sivut saattavat hajota tai näyttää huonoilta. (Mozilla 2020.)

Tämä toimintamalli ei kuitenkaan ole eduksi responsiivisille sivuille, jotka käyttävät mediakyselyitä pienemmän kokoisille näytöille kuin virtuaalisen ikkunan leveys. Apple kehitti tähän ratkaisun. Se on nimeltään kuvaportti metatieto (viewport metadata). Nykyään kyseistä metatietoa tukevat useimmat selaimet, vaikkei se ole osa web-standardia. Tämän metatiedon avulla sivujen kehittäjät voivat hallita virtuaalisen ikkunan kokoa ja skaalaa. Metatieto määritellään head-elementin sisällä, kuten kuvassa (Kuva 5) on määritelty. (Mozilla 2020.)

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <h1>Example</h1>
</body>
</html>
```

Kuva 5. Metatiedon määrittely

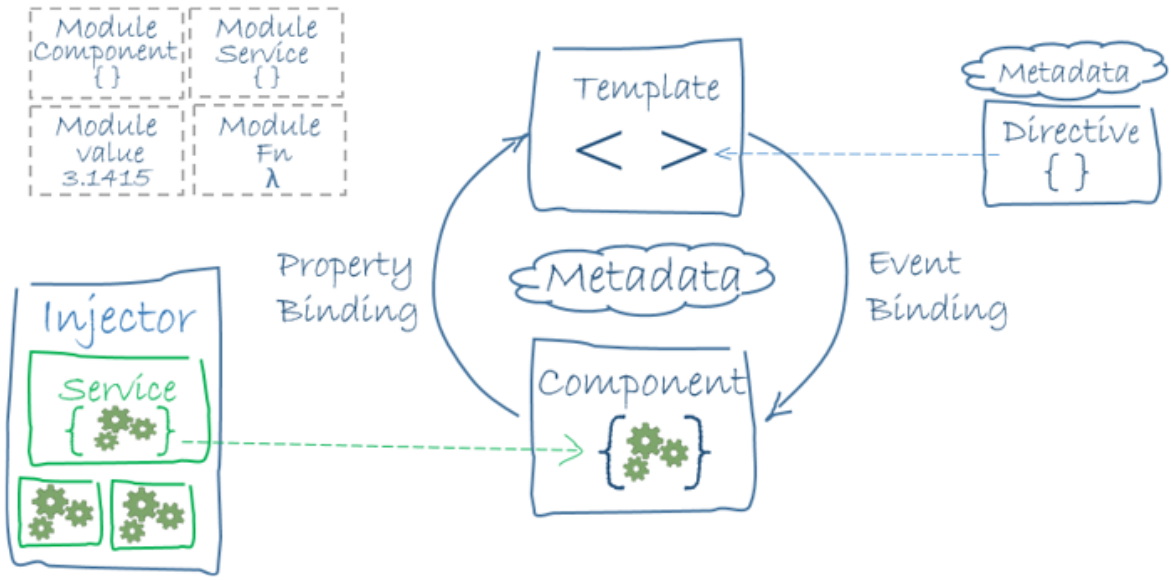
4 Käytetyt tekniikat

4.1 Angular

Angular on Googlen vuonna 2012 kehittämä TypeScript-pohjainen ohjelmistokehys ja kehitysalusta, jonka avulla voidaan rakentaa yksisivuisia sovelluksia käyttäen HTML:ää ja TypeScriptiä. Angular tukee myös useita eri tyyliohjeita perinteisen CSS:n lisäksi, kuten SASS, LESS ja SCSS. (Angular 2020a.)

4.1.1 Angularin arkkitehtuuri

Angularissa käytetään komponentteja (component), jotka kuuluvat aina moduuliin (NgModule). Komponentteja voi olla useampi yhdessä moduulissa ja moduuleita voi olla useampia. Yksi komponentti ei kuitenkaan voi kuulua kuin yhdelle moduulille. Moodulit voivat olla riippuvaisia toisistaan. Angularissa on myös palveluita (service), joiden avulla voidaan esimerkiksi jakaa dataa komponenttien välillä. Palvelut välitetään komponenteille riippuvuus injektoinnin (dependency injection) avulla. Kuvassa (Kuva 6) on esitettynä Angularin arkkitehtuuri. (Angular 2020a.)



Kuva 6. Angularin arkkitehtuuri (Angular 2020a)

Komponentit määrittelevät näkymiä, jotka koostuvat HTML elementeistä. Näkymissä voidaan myös käyttää muita komponentteja, mikäli ne ovat osa moduulia, tai ne on

toimitettu moduulin käyttöön. Komponenteille voidaan määritellä logiikkaa, jolla voidaan päättää esimerkiksi, että mitä näytetään ja miten näytetään. Niille voidaan myös syöttää muuttujia (property binding), sekä komponentti voi välittää dataa takaisinpäin tapahtumilla (event). Nämä yhdessä mahdollistavat kaksisuuntaisen datan sidonnan (two-way data binding). (Angular 2020a.)

Angularin palvelut eivät määrittele näkymiä tai niiden logikkaa suoraan. Palvelu voidaan injektoida komponenttiin riippuvuutena (dependecy), jolloin koodi pysyy modulaarisena, testattavana, uudelleenkäytettävänä ja tehokkaana. (Angular 2020a.)

Moduulit, komponentit ja palvelut ovat luokkia (class), jotka käyttävät dekoraattoreita (decorator). Nämä dekoraattorit kertovat niiden tyypin ja toimittavat metadataa, joka kertoo Angularille, miten käyttää kyseistä luokkaa. Komponentin metadata sisältää yleensä mm. polun, joka viittaa luokan näkymän malliin (template). Malli on komponentissa se osa, joka määrittelee näkymän. Myös palveluilla on dekoraattori, joka määrittää metadatan. Palvelun metadata sisältää tiedot, joita Angular tarvitsee, jotta palvelun voi injektoida komponenttien saataville riippuvuus injektoinnin avulla. Seuraavassa kuvassa (Kuva 7) riveillä 4–8 on määritelty komponentin dekoraattori. Dekoraattorin sisällä on määritetty näkymä- ja tyylitiedoston polut, sekä elementin valitsin (selector). Valitsimen avulla komponenttia voidaan käyttää toisen komponentin näkymässä. (Angular 2020a.)

```
1 import { Component, OnInit } from '@angular/core';
2 import { ExampleService } from './services/example.service';
3
4 @Component({
5   selector: 'app-example',
6   templateUrl: './example.component.html',
7   styleUrls: ['./example.component.scss'],
8 })
9 export class ExampleComponent implements OnInit {
10   constructor(private exampleService: ExampleService) {}
11
12   ngOnInit(): void {
13     this.exampleService.logStuff();
14   }
15 }
```

Kuva 7. Angular komponentti

Näkymä on yhdistelmä HTML:ää, sitovaa merkkausta (binding markup) ja direktiivejä (directive). Sitovan merkkauksen avulla Angular pystyy tuomaan arvoja komponentista HTML-näkymään automaattisesti. Esimerkiksi Angularin ngIf-direktiivillä voidaan poistaa näkymästä osia, jos haluttu ehto ei täyty. (Surehkumar, A. 2018.)

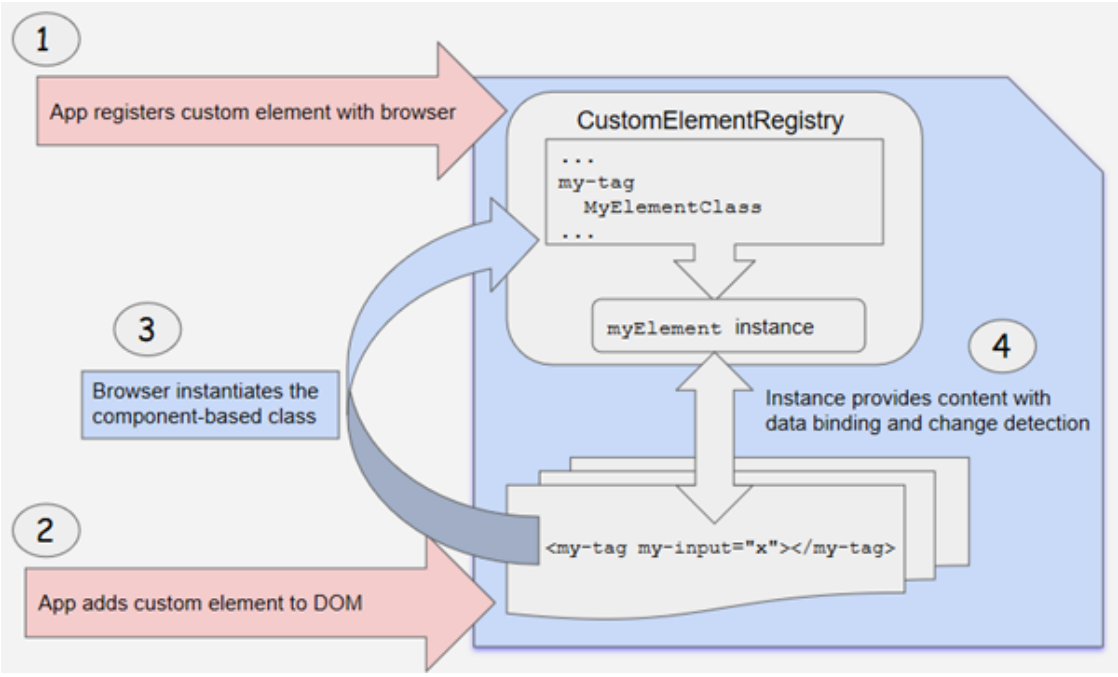
Angularin moduulit ovat mekanismi, jolla saadaan ryhmitettyä komponentteja, direktiivejä, putkia (pipe) ja palveluita. Angularin moduulin voi ajatella luokkana. Moduulissa voidaan määrittää, että mitä komponentteja, direktiivejä, putkia ja palveluita jaetaan muille moduuleille. Ne, joita ei jaeta, ovat käytössä vain moduulin sisäpuolella, ja niihin ei ole suoraa pääsyä moduulin ulkopuolelta. Seuraavassa kuvassa (Kuva 8) on moduuli, joka rivillä 6 määrittelee ExampleComponent nimisen komponentin ja asettaa tämän komponentin muiden moduulien saataville rivillä 8. Kuvan rivillä 7 tuodaan moduulille Angularin CommonModule niminen moduuli, joka sisältää mm. Angularin yleisimmät direktiivit ngIf ja ngFor. (Surehkumar, A. 2018.)

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ExampleComponent } from './example/example.component';
4
5 @NgModule({
6   declarations: [ ExampleComponent ],
7   imports: [CommonModule],
8   exports: [ ExampleComponent ],
9 })
10 export class ExampleModule {}
```

Kuva 8. Angular moduuli

4.1.2 Angular Elementit

Angular Elementit (Angular Elements) ovat Angular komponentteja pakattuina web-komponenteiksi, joita voidaan käyttää HTML:ssä samalla tavalla kuin HTML:n omia elementtejä. Elementit laajentavat HTML:ää sallimalla mukautettujen tagien määrittämisen, joiden sisältö on elementin JavaScript koodin luomaa. Angular Elements -kirjaston avulla saa siis upotettua Angularilla tehdyn sovelluksen/ohjelman muulla ohjelmistokehyksellä tehtyyn sivuun. Angular Elementtien toiminta on esitettynä seuraavassa kuvassa (Kuva 9). (Angular b 2020.)



Kuva 9. Angular Elementtien toiminta (Angular b 2020)

Oman web-komponentin luominen aloitetaan asentamalla itse Angular Elements. Tähän käytetään Angularin Angular CLI työkalua. Asentamisen jälkeen luodaan Angular-komponentti. Sille ei tarvitse antaa valitsinta dekoraattorissa, kuten yleensä komponentille annettaisiin. Sille voidaan määrittää itsenäiset näkymä- ja tyyliohje-tiedostot tarvittaessa. Komponentin sisällä voi myös käyttää Angularin muita ominaisuuksia, kuten palveluita, direktiiviä ja putkia. Komponentti voi myös ottaa syötteenä (input) dataa ulkopuolelta, jos niin halutaan. Se voi myös tarvittaessa lähettää tapahtumia ulospäin. Seuraavassa kuvassa (Kuva 10) on komponentti, jolle voidaan antaa syötteenä otsikko (title).

```
1 import { Component, Input } from '@angular/core';
2
3 @Component({
4   template: `Hei! Annoit Example elementille otsikoksi: {{title}}`
5 })
6 export class ExampleComponent {
7   @Input() title: string;
8   constructor() {}
9 }
```

Kuva 10. Esimerkki komponentti

Komponentti pitää tuoda jollekin moduulille, joka vastaa elementin luomisesta. Esimerkiksi seuraavassa kuvassa (Kuva 11) komponentti on tuotu päämoduulille (AppModule), jonka rakentajassa (constructor) luodaan oma elementti käyttäen Angular Elements -kirjaston ”createCustomElement” funktiota. Tämän jälkeen määritellään valitsin (kuvassa rivillä 17), jolla saadaan luotua komponentista instanssi sitten HTML:ssä.

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule, Injector } from '@angular/core';
3 import { createCustomElement } from '@angular/elements';
4
5 import { ExampleComponent } from './example/example.component';
6 import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
7
8 @NgModule({
9   declarations: [ExampleComponent],
10   imports: [BrowserModule, BrowserAnimationsModule],
11   entryComponents: [ExampleComponent],
12 })
13 export class AppModule {
14   constructor(injector: Injector) {
15     const customElement = createCustomElement(ExampleComponent, { injector });
16     // Määritetään "customElement"-elementin tagiksi example-component.
17     customElements.define('example-component', customElement);
18   }
19   ngDoBootstrap() {}
20 }
```

Kuva 11. Esimerkki moduuli

Koska build-vaiheessa projekti menee Angularin perinteisen build-prosessin läpi, syntyy useita eri JavaScript-tiedostoja, jotka tulee yhdistää yhdeksi tiedostoksi. Kuvassa (Kuva 12) on esimerkki shell-skriptistä, joka yhdistää tiedostot. Riippuen asetuksista, jotka on määritelty Angularille, tiedostojen nimissä ja tiedostopoluissa saattaa olla eroavaisuuksia kuvaan nähden.

```
#!/bin/sh
ng build example-component --prod --output-hashing=none && \
cat dist/example-component/runtime.js \
dist/example-component/polyfills.js \
dist/example-component/scripts.js \
dist/example-component/main.js > preview/example-component.js
```

Kuva 12. Shell-skripti

Build-prosessin jälkeen voidaan yksittäinen JavaScript-tiedosto ja prosessin luoma CSS-tyylitiedosto sijoittaa halutulle web-sivulle. Mikäli tyyliohjeet halutaan tuoda sivulle mukana, noudetaan tiedosto sivuston ”head”-elementin sisällä käyttäen HTML:n ”link”-elementtiä. JavaScript-tiedosto puolestaan tuodaan sivulle perinteisesti käyttäen ”script”-elementtiä. Kuvassa (Kuva 13) näkyy miten buildattua komponenttia voidaan käyttää.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Testi</title>
</head>
<body>
    <example-component title="Testi otsikko!"></example-component>
    <!-- Haetaan esimerkki-komponentin JavaScript tiedosto.
    Ilman tätä tiedostoa komponentti ei renderöidy. -->
    <script src="./example-component.js"></script>
</body>
</html>
```

Kuva 13. Oman web-komponentin käyttö HTML-sivulla

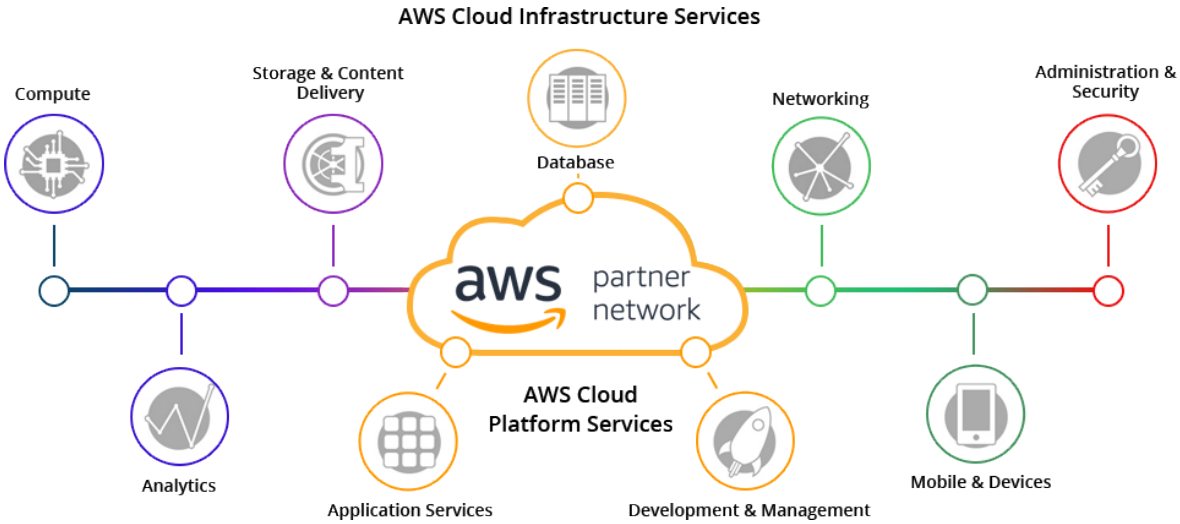
Näin onnistuttiin luomaan komponentti, jota voidaan käyttää helposti esimerkiksi useilla eri sivuilla tarvittaessa. Angular Elements-kirjaston avulla luodut komponentit käyttäytyvät HMTL:ssä, niin kuin ne käyttäytyisivät kokonaan Angularilla rakennetulla sivulla. Seuraavassa kuvassa (Kuva 14) näkyy selaimen renderöimän sivun lähdekoodi.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Testi</title>
</head>
<body>
    <example-component title="Testi otsikko!" ng-version="9.1.2">
        Hei! Annoit Example elementille otsikoksi: Testi otsikko!
    </example-component>
    <script src="./example-component.js"></script>
</body>
</html>
```

Kuva 14. Sivun lähdekoodi

4.2 Amazon Web Services

Vuonna 2006 Amazon Web Services alkoi tarjoamaan tietotekniikan infrastruktuuri-palveluita yrityksille pilvipalveluiden muodossa. Palveluita on monenlaisia, kuten laskentatehoa tai tallennustilaa tarjoavia palveluita (Kuva 15). Pilvipalveluiden avulla, yrityksen ei enää tarvitse suunnitella viikkojen tai kuukausien ajan palvelimia ja muita infrastruktuurin osia. (Amazon 2020c.)



Kuva 15. Amazonin infrastruktuurin palveluita (SoftFlame 2020)

Nykyään, Amazon Web Services tarjoaa erittäin luotettavan, skaalautuavan ja edullisen alustan pilvessä, jota jo sadat tuhannet yritykset käyttävät sadassa yhdeksässäkymmenessä maassa. Amazonin datakeskuksia sijaitsee Yhdysvalloissa, Euroopassa, Singaporessa, Japanissa ja Australiassa. (Amazon 2020c.)

Amazon Web Servicesistä maksetaan vain käytön mukaan, ilman etukäteen tehtäviä maksuja tai pitkäaikaisia sopimuksia. Hyvän hyötysuhteen ja asiantuntemuksen ansiosta Amazon onkin saanut laskettua hintojaan 15 kertaa viimeisen neljän vuoden aikana. (Amazon 2020c.)

Amazon Web Services tarjoaa massiivisen maailmanlaajuisen pilvialustan, joka mahdollistaa muun muassa nopeat kokeilut ja iteroinnit. Sen sijaan, että odottaisi viikkoja tai kuukausia omia palvelimia, voi heti ottaa käyttöön uusia sovelluksia, jotka skaalautuvat käytön mukaan. (Amazon 2020c.)

4.2.1 Lambda-funktio

Lambda-funktion koodi suoritetaan vain tarvittaessa. Se tarvitsee aina jonkun laukaisimen (trigger). Laukaisimia voi olla tarvittaessa useampi yksittäistä Lambda-funktiota kohden. Laukaisin voi olla esimerkiksi jokin toinen Amazonin palvelu, toinen Lambda-funktio tai se voi olla ajastettu. Palvelun hintaan vaikuttaa Lambda-funktion ajokertojen lukumäärä ja funktion suoritusaika. (Amazon 2020a.)

Lambda-funktioita voidaan kirjoittaa useilla eri kielillä kuten Java, Go, Python ja Node.js. Kuten myös muut Amazonin tarjoamat palvelut, on myös Lambda-funktio käytön mukaan skaalautuva. (Amazon 2020d.)

4.2.2 API Gateway

API Gateway on Amazonin tarjoama palvelu, jolla voi tehdä, julkaista, ylläpitää, monitoroida ja suojata REST, HTTP ja WebSocket rajapintoja. Tämän avulla kehittäjät voivat luoda rajapintoja Amazonin muille palveluille. (Amazon 2020b.)

API Gateway toimii “etuovena” taustapalvelussa oleville sovelluksille. Sen voi liittää esimerkiksi Lambda-funktioon laukaisimeksi, jotta saadaan kutsuttua funktiota HTTP-pyynnön yhteydessä. Siihen voi myös liittää eri käyttöoikeuksia, jos sovellus sellaisen vaatii. (Amazon 2020b.)

5 Käytetyt kirjastot

5.1 NGX-Translate

NGX-Translate –kirjasto on tehty kansainvälistämään Angular-projekteja. Se siis mahdollistaa sisällön näyttämisen eri kielillä. Myös kielen vaihtaminen on sujuvaa. NGX-Translate –kirjaston käyttöönottaminen on mutkatonta. (Github 2020a.)

NGX-Translate asennetaan perinteisesti Node Package Managerin (NPM) avulla. Tämän jälkeen tuodaan NGX-Translaten toimittama käännös-moduuli (TranslateModule) päämoduuliin (AppModule), kuten seuraavassa kuvassa (Kuva 16). Tämän jälkeen päämoduulin sisällä on käytettävissä käännösmoduulin tarjoamat palvelut, putket ja direktiivit. (Gwartney S. 2019.)

```
@NgModule({
  imports: [
    BrowserModule,
    TranslateModule.forRoot()
  ],
  bootstrap: [AppComponent]
})
```

Kuva 16. Käännös-moduulin tuominen toiseen moduuliin

Ehkä yleisin tapa ladata käännöstiedostot on lisätä käännöstiedostot ”assets”-kansioon ja ladata ne sieltä NGX-Translaten lataaja-moduulin (TranslateHttpLoader) avulla. Lataaja-moduuli pitää asentaa erikseen NPM:llä ja se pitää välittää käännös-moduulille (Kuva 17). (Gwartney S. 2019.)

```
export function createTranslateLoader(http: HttpClient) {
    return new TranslateHttpLoader(http);
}

@NgModule({
  imports: [
    BrowserModule,
    TranslateModule.forRoot({
      loader: {
        provide: TranslateLoader,
          useFactory: createHttpLoader,
          deps: [HttpClient]
      }
    })
  ],
  bootstrap: [AppComponent]
})
```

Kuva 17. Lataaja-moduulin välittäminen

Oletusarvoisesti lataaja-moduuli olettaa käännöstiedostojen sijaitsevan kansiossa ”/assets/il8n/”, jossa sijaitsevat käännöstiedostot nimettyinä esimerkiksi fi.json suomenkielen tiedostolle. Käännöstiedosto on rakenteeltaan perinteinen JSON objekti, joka koostuu avain-arvo pareista (key-value pair), kuten kuvassa (Kuva 18) on tehty. (Gwartney S. 2019.)

```
{
  "header": "Testi",
  "content": "Tässä suomenkielinen sisältö.",
  "buttons": {
    "back": "Takaisin",
    "next": "Seuraava"
  }
}
```

Kuva 18. Esimerkki suomenkielen käännöstiedostosta (fi.json)

Käyttääkseen käännöstiedostojen sisältämiä käännöksiä tulee kääntäjä-palvelulle (TranslateService) alustaa muutamia ominaisuuksia. Sille kerrotaan, mitä kieliä on saatavilla, mikä on oletusarvoinen kieli ja mitä kieltä halutaan käyttää juuri nyt. Tämä tarvitsee tehdä vain kerran eli alustukselle hyvä sijoituskohta on esimerkiksi AppComponent. Näytettävän kielen vaihto on helppoa, kutsutaan kääntäjä-palvelun ”use”-metodia ja välitetään sille haluttu kieli kutsuparametrina (Kuva 19). (Gwartney S. 2019.)

```
import {Component} from '@angular/core';
import {TranslateService} from '@ngx-translate/core';

@Component({
  selector:  'my-app',
  templateUrl:  './app.component.html',
  styleUrls:  [  './app.component.css'  ]
})
export class AppComponent {
  constructor(translate: TranslateService) {
    translate.addLangs(['en', 'fi'])
    translate.setDefaultLang('en');
    translate.use('fi');
  }
}
```

Kuva 19. Kääntäjä-palvelun alustaminen

Kääntäjä-palvelulla on kaksi metodia, joilla voi hakea käännöksiä. Metodi ”get” on asynkroninen ja se takaa sen, että kääntäjä-palvelu lataa käännöstiedoston ennen käännöksen palauttamista. Metodi ”instant” puolestaan on synkroninen. Jos palvelulla ei ole käännöstiedostoa ladattuna, kun metodi yrittää hakea käännöstä, se ei palauta käännöstä vaan määrittelemättömän arvon. (Gwartney S. 2019.)

Kääntäjä-palvelulla on myös käännös-putki (TranslatePipe), jota käytetään samalla tavalla kuin Angularin omia putkia. Translate putkelle annetaan ”polku”, jonka päässä haluttu käännös käännöstiedostossa sijaitsee (Kuva 20). (Gwartney S. 2019.)

```
<p>{{ 'welcomeMessage' | translate }}</p>
<input type="password" placeholder="{{ 'login.password' | translate }}">
```

Kuva 20. Esimerkki NGX-Translaten käännös-putken käytöstä

Myös direktiivi kääntämiselle löytyy ja sitä pystyy käyttämään kahdella eri tavalla. Ensimmäinen tapa on välittää suoraan direktiiville käännöksen polku ja toinen tapa on laittaa käännöksen polku elementin sisällöksi, molemmat tavat on esitetty seuraavassa kuvassa (Kuva 21). (Gwartney S. 2019.)

```
<!-- Polun välittäminen suoraan direktiiville -->
<label translate='login.username'></label>
<!-- Polku sisältönä -->
<p translate [translateParams]="{ firstName: user.firstName }">welcomeMessage</p>
```

Kuva 21. Käännös-direktiivin käyttäminen

5.2 Lodash

Lodash, eli yleisemmin "_", eli alaviiva on kirjasto, joka on luotu helpottamaan JavaScript-kehitystä ja koska TypeScript on JavaScriptiin pohjautuva, voi lodashia käyttää myös TypeScriptillä. Se sisältää paljon yleishyödyllisiä funktioita esimerkiksi taulukoiden iterointiin ja muuttujien syväkopiointiin. (Lodash 2020.)

Lodashin set-funktiolle annetaan kutsuparametreina objekti, polku ja arvo. Mikäli objektilla ei ole polun mukaisia ominaisuuksia, set-funktio luo ne ja asettaa "polun päähän" arvon, joka on välitetty funktiolle. Polun voi antaa funktiolle merkkijonona tai merkkijono taulukkona (Kuva 22). (Lodash 2020.)

```
1 let object = { 'a': [{ 'b': { 'c': 3 } }] };
2
3 _.set(object, 'a[0].b.c', 4);
4 console.log(object.a[0].b.c);
5 // => 4
6
7 _.set(object, ['x', '0', 'y', 'z'], 5);
8 console.log(object.x[0].y.z);
9 // => 5
```

Kuva 22. Lodashin set-funktio

Lodash-kirjastossa on myös vastakohta set-funktiolle, se on get-funktio. Sille annetaan kutsuparametreina objekti, polku ja vapaavalintainen oletusarvo. Get-funktio yrittää siis hakea objektista “polun päästä” arvoa. Mikäli arvoa ei löydy, palauttaa funktio oletusarvon, mikäli se on määritetty funktiokutsussa, jos sitä ei ole määritetty niin palauttaa funktio määrittelemättömän arvon. Seuraavassa kuvassa (Kuva 23) riveillä 3 ja 6 objektista yritetään hakea arvoa ilman oletusarvoa, kun taas rivillä 9 oletusarvoksi on asetettu merkkijono ”default”. (Lodash 2020.)

```
 1 let object = { 'a': [{ 'b': { 'c': 3 } }] };
 2
 3 _.get(object, 'a[0].b.c');
 4 // => 3
 5
 6 _.get(object, ['a', '0', 'b', 'c']);
 7 // => 3
 8
 9 _.get(object, 'a.b.c', 'default');
10 // => 'default'
```

Kuva 23. Lodashin get-funktio

Lodash-kirjaston isEqual-funktio puolestaan nimensä mukaisesti vertailee sille annettua kahta arvoa, että ovatko ne samanarvoisia. Funktio pystyy vertaamaan muun muassa: taulukoita, objekteja, numeroita ja funktioita. Funktio palauttaa totuusarvon tosi, mikäli annetut arvot ovat samanarvoisia. (Lodash 2020.)

Lodash-kirjastolla on myös tarjolla useita funktioita taulukoille. Lodashin funktio ”first” palauttaa elementin, joka on taulukon ensimmäisessä indeksissä, kun taas funktio ”last” palauttaa elementin, joka on taulukon viimeisessä indeksissä. ”nth”-funktio palauttaa halutusta indeksistä elementin. Jos haluttu indeksi on negatiivinen, palautus tehdään loppupäästä. Hyödyllisiä funktioita ovat myös ”chunk”, joka katkoo taulukon pienempiin taulukoiden määritetyn luvun mukaan, sekä ”slice”, joka puolestaan palauttaa toimitetusta taulukosta elementit halutulta indeksi väliltä. (Lodash 2020.)

5.3 Three Sixty

Three Sixty eli 360°-kirjasto mahdollistaa 360° näkymien helpon luonnin. Kirjasto tukee niin tavallisia tietokoneita kuin kosketusnäytöllisiälaitteitakin. Three Sixty -kirjasto luo 360° näkymän käyttäen useita kuvia hyödykseen. Näkymälle voi määrittää montako kuva halutaan käyttää ja kuinka suuren aste-alueen yksi kuva on näkyvissä. (Github 2020b.)

Three Sixty-kirjaston käyttö on tehty helpoksi kuten seuraavassa kuvassa on esitetty (Kuva 24). Ensin luodaan uusi ThreeSixty objekti, jolle annetaan kutsuparametreissä HTML:n canvas elementti ja asetus-objekti (kuvan rivillä 1). Asetus-objektilla voidaan määrittää muun muassa asteiden määrä (rivillä 2), yksittäisen kuvan aste-alue (rivillä 3) ja mahdolliset tekstit, joita halutaan näyttää tietyillä aste-alueilla. Tämän jälkeen tulee objekti alustaa initialize-metodin avulla. Metodille voidaan antaa kutsuparametreissä kuvat, joista 360° näkymä halutaan muodostaa. Alustamisen jälkeen voidaan kuvat esi-ladata preload-metodilla, jotta näkymän pyöritys toimii sulavammin. (Github 2020b.)

```
1 const threeSixty = new ThreeSixty(document.getElementsByTagName('canvas')[0], {
2     angles: 36,
3     anglesPerImage: 6
4 });
5
6 threeSixty.initialize({
7     0: [
8         'http://example.com/images/360/image-0.jpg',
9         'http://example.com/images/360/image-1.jpg',
10        'http://example.com/images/360/image-2.jpg',
11        'http://example.com/images/360/image-3.jpg',
12        'http://example.com/images/360/image-4.jpg',
13        'http://example.com/images/360/image-5.jpg'
14    ]
15 });
16
17 threeSixty.preload().then(() => console.log('ready to go'));
```

Kuva 24. Three Sixty -kirjaston käyttäminen

5.4 Angular Flex-Layout

CSS:n FlexBox ja CSS:n Grid ovat tehokkaita asettelu ominaisuuksia. Angular Flex-layout toimii yksinkertaisena rajapintana kyseisiin ominaisuuksiin ja se helpottaa responsiivisten Angular-sovellusten luontia. Niiden avulla luodaan käyttöliittymän perusta yhdessä HTML:n rakenteen kanssa. (Inatomi S. 2018.)

Angularissa on yleistä, että näkymä jaetaan HTML tiedostoon ja tyylitiedostoon. Koska FlexBox ja Grid liittyvät vahvasti HTML:n rakenteeseen, ei ole kätevää, että asettelu määritettäisiin tyylitiedostossa, eikä HTML tiedostossa. Angular Flex-Layout tarjoaa ratkaisun tähän ongelmaan. (Inatomi S. 2018.)

Angular Flex-Layout on itse Angular-tiimin tekemä ja se tarjoaa rajapinnan FlexBoxin sekä mediakyselyiden käyttöön. Flex-Layoutin asentaminen tapahtuu NPM:n avulla, jonka jälkeen tuodaan perinteiseen tapaan ”FlexLayoutModule” niminen moduuli siihen moduuliin, jossa kyseisiä FlexLayout/mediaQuery -ominaisuuksia halutaan käyttää. (Inatomi S. 2018.)

FlexLayoutModule tarjoaa useita direktiivejä avuksi luomaan responsiivisen käyttöliittymän. Esimerkiksi ”fxLayout”-direktiivillä saadaan asetettua elementille ”display: flex” ja ”flex-direction” tyylitykset. Se siis luo uuden FlexBox säiliön ja asettaa sisällön kulkusuunnan. ”flexLayoutGap”-direktiivin avulla puolestaan voidaan asettaa elementtien välinen etäisyys toisistaan. Se asettaa siis elementeille margin-arvoja. Ero ei kuulosta

suurelta, mutta eron huomaa, kun näkee puhtaan CSS-toteutuksen ja Angular Flex-Layout -toteutuksen vierekkäin, kuten seuraavassa kuvassa (Kuva 25) on. Kuvan molempien toteutuksien lopputulos on sama, mutta Angular Flex-layout:in direktiivien avulla on saatu tehtyä sama tyylittely luettavammin ja selkeämmin. (Inatomi S. 2018.)

```
<!-- Puhdas HTML/CSS ratkaisu -->
<style>
 .cardList {
   display: flex;
   flex-direction: row;
   flex-wrap: wrap;
   justify-content: flex-start;
 }
 .cardList > * {
   box-sizing: border-box;
 }
 .cardList > *:not(:last-child) {
   margin-right: 32px;
 }
 .cardListItem {
   flex: 0 1 calc(33.3% - 32px);
 }
</style>
<div class="cardList">
  <ng-container *ngFor="let card of [1,2,3,4,5,6]">
    <app-card class="cardListItem"></app-card>
  </ng-container>
</div>

<!-- Angular Flex-Layout toteutus -->
<div
  fxLayout="row wrap" fxLayoutGap="32px" fxLayoutAlign="flex-start">
  <ng-container *ngFor="let _ of [1,2,3,4,5,6]">
    <app-card fxFlex="0 1 calc(33.3% - 32px)"></app-card>
  </ng-container>
</div>
```

Kuva 25. Angular Flex-Layout:in käyttö

5.5 Handlebars

Handlebars on minimaalinen sapluuna kieli (templating language). Sapluunat näyttävät perinteisiltä tekstitiedostoilta, mutta niihin voidaan upottaa JavaScript-muuttujia.

Sapluunaan voidaan määrittää muuttujalle paikka käyttäen muuttujan nimen ympärillä “{{“- ja “}}”-merkkejä. Handlebars asennetaan NPM:n avulla. (Handlebars 2020.)

Handlebars kykenee myös muun muassa ehdolliseen määrittelyyn, taulukoiden iterointiin sekä se tukee myös objekteja. Kaikki edellä mainituista on tärkeitä ominaisuuksia sapluunoiden teossa. (Handlebars 2020.)

Ehdollisessa määrittelyssä käytetään sisäänrakennettua if-avainsanaa. Mikäli lauseke palauttaa jonkin seuraavista arvoista: false, undefined, null, 0, tyhjä taulukko tai tyhjä merkkijono, niin silloin kyseisen kappaleen sisältöä ei renderöidä. Muussa tapauksessa sisältö renderöidään. (Handlebars 2020.)

Taulukoiden iteroinnissa käytetään sisäänrakennettua each-avainsanaa. Iteroinnin aikana taulukon tämänhetkiseen soluun viitataan this-avainsanalla (Kuva 26). (HandleBars 2020.)

```
import * as handlebars from "handlebars";

const HandleBarsExample = () => {
    const htmlTemplate = handlebars.compile(
        `<ul>
        {{#each people}}
            <li>{{this.name}}</li>
        {{/each}}
        <ul>`
    );
    const people = [
        { name: "Hannu Hanhi" },
        { name: "Aku Ankka" }
    ];
    return htmlTemplate({
        people,
    });
};

console.log(HandleBarsExample());
/* Tulostaa:
<ul>
    <li>Hannu Hanhi</li>
    <li>Aku Ankka</li>
<ul>
 */
```

Kuva 26. Handlebars:in käyttö

6 Käyttöliittymä

6.1 Suunnittelu

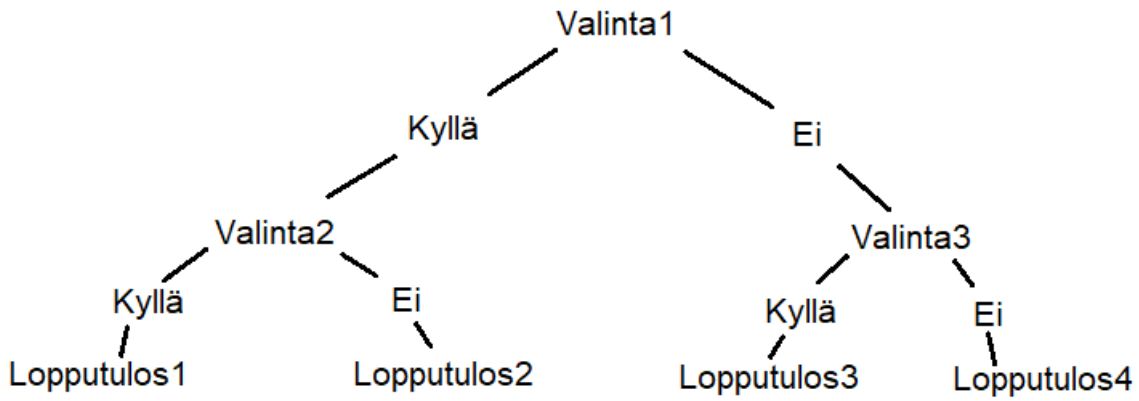
Käyttöliittymän tärkein tehtävä on tuoda käyttäjän nähtäväksi, millaisia erilaisia lisävarusteita hitsauskoneeseen on saatavilla. Käyttöliittymän pääsääntöinen kohderyhmä on hitsausalan ammattilaiset. Lisävarusteet pitää siis tuoda visuaalisesti hyvin esille. Tässä työssä haluttu lopputulos on interaktiivinen, ja avuksi interaktiivisuuteen käytetään kustomoitua Three Sixty –kirjastoa.

Käyttöliittymän tulee olla responsiivinen ja se toteutetaan käyttäen Angularin Flex-Layout-kirjastoa. Saman näkymän pitää toimia niin mobiililaitteella kuin tietokoneella. Selaimista sen pitää tukea uusimmat selaimet, Opera, Safari, Edge, Chrome ja Mozilla Firefox.

6.1.1 Päätöspuu

Lopullisen sovelluksen pitää pystyä näyttämään 52 eri yhdistelmää hitsauskoneesta ja sen lisävarusteista. Jokaiseen viidestäkymmenestäkahdesta tapauksesta päästään vain tietyllä valintojen yhdistelmällä. Jotta tämä olisi mahdollista ja järjestelmän ylläpidettävyys säilyisi, toteutetaan tämä päätöspuun avulla.

Päätöspuun voi ajatella karttana, jossa tehdyt valinnat muodostava polun, jonka toisessa päässä on lopputulos (Kuva 27). Tässä työssä tullaan käyttämään kahta päätöspuuta. Toinen päättelemään, mikä langansyöttölaite-kokonaisuus on valittuna ja toinen päättelemään koko kokonaisuutta.



Kuva 27. Päätöspuu

6.1.2 Mahdolliset valinnat

Käyttäjälle tulee useita eri valintakenttiä, joista voi muokata valintoja. Sovelluksesta pitää siis kytkeä pois päältä valinnat, jotka eivät ole mahdollisia aiemmin tehtyjen valintojen perusteella. Jos tätä ei tehdä, käyttäjän tekemien valintojen muodostama “polku” johtaa päätöspuussa harhaan.

Tähän tulee käyttää omaa palvelua, jonka tehtävä on huolehtia, että käyttäjän tekemät valinnat muodostavat oikeanlaisen polun. Palvelu voi muokata tarvittaessa käyttäjän tekemiä valintoja.

6.1.3 Optimointi – kuvien lataus

Kun päätöspuun lopputulema vaihtuu, lataa selain uuden kuvasarjan, joka koostuu kuudestakymmenestä kuvasta. Ilman optimointia olisi kuudenkymmenen kuvan lataaminen, varsinkin hitailla yhteyksillä, hidasta. Tämän vuoksi lataaminen pitää optimoida suosimaan myös hitaampia yhteyksiä.

6.2 Toteutus

Käyttöliittymän toteutus oli opinnäytetyön toteutuksen kannalta tärkein tehtävä, sillä kaikki toiminnallisuus sijaitsee käyttöliittymässä pois lukien Lambda-funktiota, joka vastaa sähköpostin lähetyksestä. Käyttöliittymän suunnittelun mukainen toteutus takaa responsiivisen ja käyttäjäystävällisen lopputuloksen.

Käyttöliittymän toteutuksen aikana voidaan vielä tehdä muutoksia, mikäli muutoksen tarvetta huomataan. Toteutuksen lopuksi käyttöliittymä upotetaan erilliselle sivustolle käyttäen Angular Elements -kirjastoa apuna.

6.2.1 Päätöspuun luominen

Päätöspuu on hyvin olennainen osa käyttöliittymää, vaikka se ei käyttäjälle suoranaisesti näy. Se kuitenkin päättää, mitä kuvasarjaa käyttäjälle näytetään milloinkin. Päätöspuun pohjana toimii työn toimeksiantajan toimittama Excel-taulukko, joka muutettiin kahdeksi TypeScript-objektiksi. Toinen objekteista sisältää tiedot eri langansyöttölaitteiden vaihtoehdoista ja toinen objekteista sisältää tiedot kokonaisuudesta.

Päätöspuun logiikkaa varten luotiin oma palvelu, joka luo TypeScript-objekteista päätöspuun Lodash-kirjaston set-funktion avulla (Kuva 28). Palvelulla on myös metodi, jonka avulla voidaan tarkastaa, mihin arvoon nykyisten valintojen muodostama polku osoittaa. Metodi muuttaa siis käyttäjän valinnat poluksi ja tarkastaa päätöspuusta, että

mitä kuvasarjaa pitäisi näyttää. Metodi kertoo kuvasarjan muutoksesta muille komponenteille, jotka ovat tilanneet muutosilmoitukset palvelusta.

```
1  makeDecisionTree(){
2      this.cases.forEach((_case) => {
3          _case.data.feederCableLength.forEach((_fcl, _fclI) => {
4              const path = [
5                  _case.data.feederCableLength[_fclI],
6                  _case.data.feederPlacement,
7                  _case.data.trolley,
8                  _case.data.feedeOption,
9                  _case.data.gunHolder,
10                 _case.data.wireDrumKit
11             ]
12
13             _.set(
14                 this.decisionTree,
15                 path,
16                 _case.name
17             );
18         });
19     });
20 }
```

Kuva 28. Päätöspuun luonti

6.2.2 Optimoinnin toteutus

Jos suuri osa käyttäjän sivulla viettämästä ajasta menee kuvien lataamiseen, niin kokemus sivustosta voi helposti jäädä negatiiviseksi. Kuvan vaihto täytyy siis optimoida niin, että ladataan ensimmäisenä sen kuvakulman kuva, joka tällä hetkellä käyttäjällä on aktiivisena. Tämän jälkeen ladataan joka kuudes kuva, jotta saadaan kuvan pyöritys toimimaan edes välttävästi. Joka kuudennen kuvan jälkeen ladataan joka toinen kuva, jonka jälkeen pyöritys toimii jo hyvinkin sulavasti. Viimeisenä ladataan puuttuvat kuvat ja pyöritys on tosi sulava. Seuraavassa kuvassa (Kuva 29) näkyy, missä järjestyksessä selain lataa kuvia, kun valintoja muutetaan. Kuvien nimissä viimeinen luku kertoo kuvakulman numeron, jolla kuva näkyy. Ensimmäisenä selain on ladannut kuvan numero 59, jonka jälkeen selain on ladannut joka kuudennen kuvan.

Name	Size	Time
8-1-59.png	169 kB	128 ms
data:image/png;base...	(memory cache)	1 ms
8-1-59.png	258 B	10 ms
8-1-53.png	155 kB	166 ms
8-1-47.png	173 kB	205 ms
8-1-41.png	176 kB	211 ms
8-1-35.png	174 kB	239 ms
8-1-29.png	168 kB	266 ms
8-1-23.png	153 kB	311 ms
8-1-17.png	171 kB	370 ms
8-1-11.png	174 kB	401 ms
8-1-05.png	177 kB	430 ms

Kuva 29. Kuvien lataus

6.2.3 Infopisteet

Infopisteiden tarkoitus on tuoda esille tietoa laitteen/lisävarusteiden eri ominaisuuksista (Kuva 30). Ne näkyvät käyttäjälle oranssivalkoisina palloina ja niitä klikkaamalla kyseisen kohdan tiedot tulevat esille. Infopisteiden sijainnit ja tekstit on tallennettu JSON-tiedostoon.

Infopisteet näytetään vain silloin, kun kuvasarjasta näytetään ensimmäinen kuva, eli ne ovat piilossa, kun käyttäjä pyörittää kuvasarjaa. Tähän syy on se, että käyttäjällä on mahdollisuus nähdä laite ilman infopisteitä.

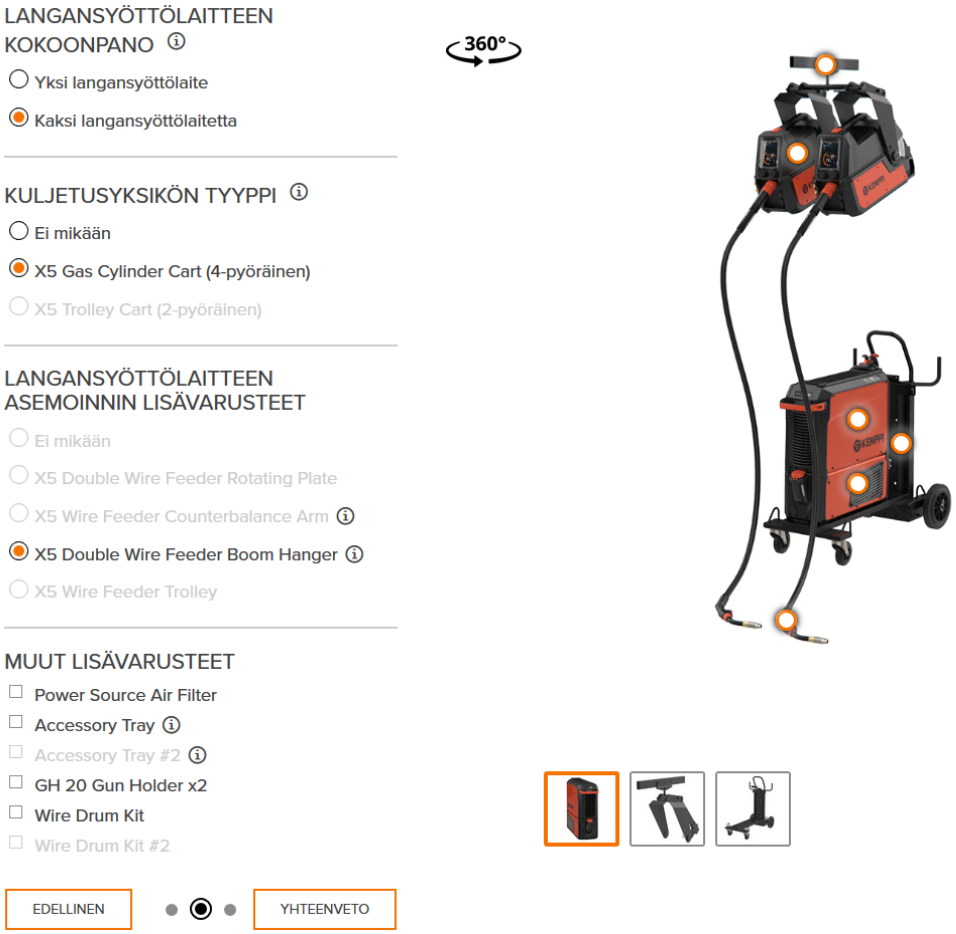
Kuva 30. Infopisteet

6.2.4 Upotus web-sivustolle

Viimeisenä vaiheena oli tehdyn sovelluksen upottaminen yrityksen web-sivustolle. Tämä vaihe tehtiin yhteistyössä sivuston ylläpitäjän kanssa.

Tässä vaiheessa Angular-projekti ajettiin Angularin build-prosessin läpi ja siitä muodostui Angular Elements -kirjaston avulla oma web-komponentti. Eli siirtämällä JavaScript- ja CSS-tiedostot sekä ”assets”-kansio sisältöineen erilliselle sivustolle saa käyttöön tämän kyseisen web-komponentin.

Sivuston upotuksen yhteydessä huomattiin, että sivusto, johon upotus tehtiin, yliajoi joitain HTML-elementtien tyyliohjeita lisävarustevalitsimen puolella. Ongelma poistettiin testaamalla upotusta muutamia kertoja, jonka jälkeen tyylivirheet saatiin korjattua ja upotus onnistui (Kuva 31). Upotus-testauksien yhteydessä testattiin myös markkinoinnin lomakkeelle siirtymistä ja evästeen välittämistä markkinoinnin käyttämälle Marketo-ohjelmalle.



Kuva 31. Valmis upotus

7 Taustapalvelu

7.1 Suunnittelu

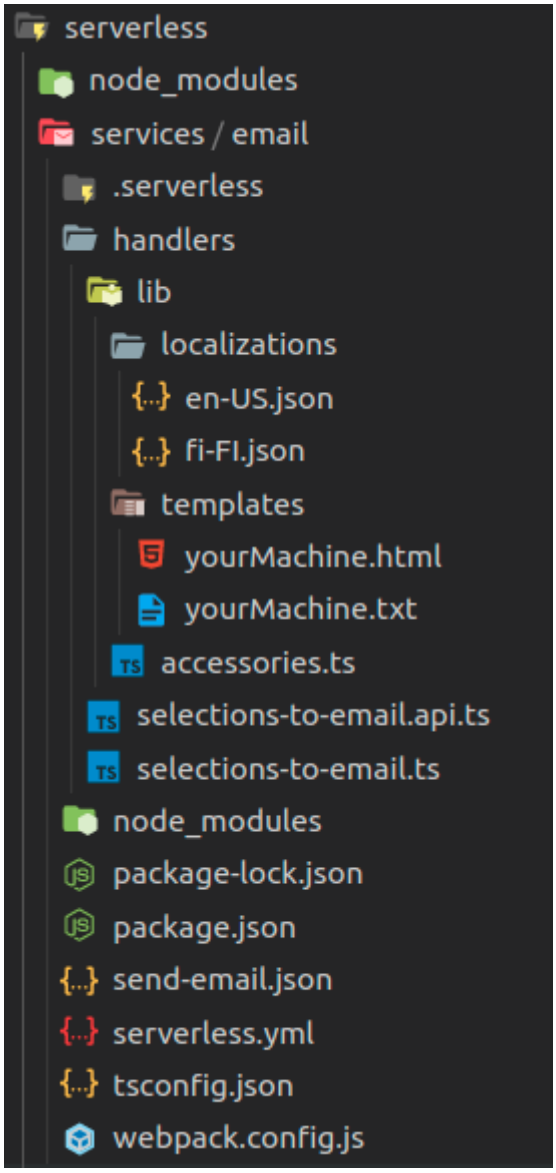
Taustapalvelun pitää pystyä lähettämään sähköposti haluttuun osoitteeseen. Sähköpostin sisältönä on käyttäjän valitsemat lisävarusteet ja muut valinnat. Amazonin pilvipalvelu tarjoaa tähän oivan työkalun Simple Email Servicen (SES), jonka avulla pystyy muun muassa lähettämään sähköposteja. SES liitettynä Amazonin Lambda-funktioon tarjoaa juuri oikeanlaisen ratkaisun. Jotta Lambda-funktiota voidaan kutsua käyttöliittymästä, tarvitsee siihen liittää API GateWay laukaisimeksi.

Taustapalvelun suurin tietoturva-aukko on sähköpostinlähetykseen tarkoitettu Lambda-funktio. Mikäli Lambda-funktiolle ei tehtäisi mitään suojausta, olisi väärinkäyttö helppoa, koska käyttöliittymä on julkinen ja se kertoo Lambda-funktiolle sähköpostin halutun sisällön.

Tähän ongelmaan ratkaisuna toimii HandleBars niminen sapluunakieli. Sen avulla saadaan näytettyä juuri se data, mitä sille annetaan. Ennen datan välittämistä sapluunakielen käyttöön täytyy käyttöliittymältä tuleva data validoida. Käyttöliittymä toimittaa Lambda-funktiolle valittujen lisävarusteiden tuotekoodit ja funktio sisällyttää niiden perusteella lisävarusteiden oikeat nimet sähköpostiin. Sähköpostin sisältöön ei laiteta kyselyparametreista suoraan mitään merkkijonoja.

7.2 Toteutus

Taustapalvelu toteutettiin käyttäen serverless-arkkitehtuuria Amazonin pilvessä. Taustapalvelu on kokonaisuudessaan pieni, sillä se koostuu ainoastaan yhdestä Lambda-funktiosta, jonka tarkoitus on lähettää käyttäjän tekemät valinnat käyttäjän sähköpostiin. Taustapalvelun hierarkia näkyy seuraavassa kuvassa (Kuva 32).



Kuva 32. Taustapalvelun hierarkia

Koska käyttöliittymä tukee eri kieliä, pitää myös sähköpostin tukea samoja kieliä. Sähköpostin kielituen olisi voinut toteuttaa kolmannen osapuolen kirjastolla, mutta koska sähköposti ei ole sisällöllisesti erityisen haastava, toteutettiin se käyttämällä yksinkertaista JSON-objektia, TypeScript-rajapintaa (Kuva 33) ja Handlebars-kirjastoa. JSON-objekti (Kuva 34) on rakennettu TypeScript-rajapinnan pohjalta ja objekti syötetään HandleBarssin sapluunalle.

```
1 export interface Translations {
2     header: string;
3     thankYou: string;
4     yourSelections: string;
5     wireFeederPlacement: string;
6     wireFeederSetup: string;
7     interConCableLength: string;
8     weldingGunCable: {
9         title: string;
10        sixMeter: string;
11        eightMeter: string;
12    };
13 }
```

Kuva 33. TypeScript-rajapinta

```
1 {
2     "header": "X5 FastMig -kokoonpanosi",
3     "thankYou": "Kiitos, kun käytit X5 FastMig Selector -työkaluamme. Ohessa lista
  valinnoistasi ja lisävarusteista, joita suosittelemme X5 FastMig
  -hitsauslaitteellesi.",
4     "yourSelections": "Valintasi X5 FastMig -hitsauslaitteelle",
5     "wireFeederPlacement": "Langansyöttölaitteen sijoittaminen",
6     "wireFeederSetup": "Langansyöttölaitteen kokoonpano",
7     "interConCableLength": "Välikaapelin pituus",
8     "weldingGunCable": {
9       "title": "Hitsauspolttimen kaapelin pituus",
10      "sixMeter": "* Saatavana GX 305 G, GX 305 W ja GX 405 W -mallit",
11      "eightMeter": "* Saatavana GX 305 G ja GX 405 W -mallit"
12    }
13 }
```

Kuva 34. JSON-objekti

8 Yhteenveto

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa hitsauskoneen lisävarustevalitsin, jonka päätarkoituksena oli näyttää asiakkaalle, mitä lisävarusteita hän tarvitsisi hitsauskoneeseensa, jotta ympäristön ja hitsaustyön asettamat vaatimukset täyttyvät. Tarkoituksena oli upottaa valmis sovellus erillisille web-sivulle käyttäen Angular-ohjelmistokehystä ja Angular Elements -kirjastoa.

Työn tuloksena syntyi responsiivinen ja interaktiivinen lisävarustevalitsin, joka julkaistiin syksyllä 2020. Lisävarustevalitsimen avulla käyttäjä näkee hyvin, miltä mikäkin lisävaruste näyttää yhdessä hitsauskoneen kanssa sekä erikseen. Käyttäjä pystyy lähettämään valintansa omaan sähköpostiin suoraan käyttöliittymästä vaivattomasti, sekä pystyy tarvittaessa ottamaan suoraan yhteyttä Kempin markkinointiin, jolloin markkinointi saa käyttäjän tekemät valinnat tallennetun evästeen avulla.

Lisävarustevalitsimen pyörityksen olisi voinut toteuttaa esimerkiksi ThreeJS-kirjastolla Three Sixty -kirjaston sijaan. Niiden suurin ero on se, että ThreeJS-kirjaston avulla olisi saanut käytettyä kuvasarjojen tilalla 3D-objektia. Myös itse upotusta olisi voinut koittaa hieman aiemmin, jotta kohdesivustolla olleet yliajavat tyyliohjeet olisi löydetty aiemmin ja näinollen ne olisi voitu korjata aiemmin.

Lisävarustevalitsimen seuraava vaihe on lisätä puuttuvat käännöstiedostot. Tällä hetkellä valitsin tukee suomea ja englantia, mutta tuettuja kieliä tulee olemaan hieman yli kymmenen. Lisävarustevalitsimeen voidaan tulevaisuudessa lisätä uusia lisävarusteita. Lisävarusteesta riippuen sen lisääminen valitsimeen pitäisi olla melko suoraviivaista, sillä lisävarusteiden lisäämistä jälkikäteen pyrittiin pitämään mielessä työtä tehdessä.

Lähteet

Amazon a. Viitattu 18.11.2020. Saatavissa: https://aws.amazon.com/lambda/

Amazon b. Viitattu 18.11.2020 Saatavissa: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

Amazon c. Viitattu 24.11.2020. Saatavissa: https://aws.amazon.com/about-aws/

Amazon d. What is AWS Lambda? Viitattu 25.11.2020. Saatavissa: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

Angular a. Viitattu 9.11.2020. Saatavissa: https://angular.io/guide/architecture

Angular b. Viitatu 18.11.2020 Saatavissa: https://angular.io/guide/elements

Barron B. 2018. What Is Responsive Web Design? (Definition + Examples). Saatavissa: https://business.tutsplus.com/tutorials/what-is-responsive-web-design-definition-examples--cms-30843

Github a. Viitattu 18.11.2020. Saatavissa: https://github.com/ngx-translate/core;

Github b. Viitattu 29.11.2020. Saatavissa: https://github.com/mediamanDE/three-sixty

Gwartney S. 2019. Internationalize Your Angular App with ngx-translate. Saatavissa: https://www.digitalocean.com/community/tutorials/angular-ngx-translate

Handlebars, Viitattu 18.11.2020. Saatavissa: https://handlebarsjs.com/guide/

Inatomi S. 2018. Angular Flex-Layout: Flexbox and Grid Layout for Angular Component. Saatavissa: https://medium.com/angular-in-depth/angular-flex-layout-flexbox-and-grid-layout-for-angular-component-6e7c24457b63

Lodash, Viitattu 18.11.2020. Saatavissa: https://lodash.com/

Jyväskylän yliopiston informaatioteknologian tiedekunta. 2016. Responsiivinen web-suunnittelu Saatavissa: http://appro.mit.jyu.fi/web-sovellukset/luennot/responsiivisuus/

Kemppi Oy a. Viitattu 24.11.2020. Saatavissa: https://www.kemppi.com/fi-FI/

Kemppi Oy b. Viitattu 25.11.2020. Saatavissa: https://www.kemppi.com/fi-FI/x5-fastmig-selector/

Mozilla. 2020. Using the viewport meta tag to control layout on mobile browsers. Saatavissa: https://developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag

SoftFlame. 2020. AWS Cloud Migration. Saatavissa: https://softflame.in/aws-cloud-migration-services-in-pune.php

Surehkumar, A. 2018. Rangle.io. Angular 2 Training. Saatavissa: https://angular-2-training-book.rangle.io/modules/introduction

WeldEye. Viitattu 25.11.2020 Saatavissa: https://www.weldeye.com/en-US/what-is-weldeye/introduction/weldeye/

Wikipedia a. Viitattu 25.11.2020. Saatavissa: https://fi.wikipedia.org/wiki/MIG/MAG-hitsaus

Wikipedia b. Viitattu 25.11. 2020. Saatavissa: https://fi.wikipedia.org/wiki/HTML