jamk.fi

Verkkokaupan tuotevalitsinmoduulin suunnittelu ja toteutus

Jami Hankomäki

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), mediatekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Hankomäki, Jami	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 28	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Verkkokaupan visuaalisen tuotevalitsimen suunnittelu ja toteutus

Tutkinto-ohjelma
Mediatekniikka

Työn ohjaaja(t)
Kari Niemi

Toimeksiantaja(t)
Solteq Oyj

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Solteq Oyj. Solteq on monikanavaisen kaupankäynnin asiantuntijayritys. Solteqille luotiin verkkokauppoihin liitettävissä oleva visuaalinen varaosavalitsinmoduuli. Moduulin avulla käyttäjälle voidaan visualisoida varaosan sijainti itse päätuotteessa.

Verkkosovelluskehityksessä kehitysprosessia voidaan tehostaa hajauttamalla sovellus useaan eri osaan eli moduuliin. Moduuleja voidaan kehittää itsenäisesti pääprojektista erillään ja liittää siihen moduulin valmistuttua. Tämä helpottaa sovelluksen testausta, kun moduuleja voidaan testata omina kokonaisuuksina ja mahdollisesti löytyvien virheiden alkuperä on helpompi paikallistaa. Modulaarisuus voi myös nopeuttaa kehitysprosessia, jos moduulit ovat helposti kopioitavissa ja liitettävissä eri kohtiin pääprojektia.

Työssä tutustuttiin aluksi erilaisiin kehitystyökaluihin, joita käytettiin itse moduulin kehittämiseen. Yksi tärkeimmistä oli vektorigrafiikkaformaatti SVG. SVG:n avulla moduulista voitiin tehdä responsiivinen ja siihen voitiin liittää toiminnollisuuksia, joita perinteinen rastergrafiikka ei tue. Moduulin prototyyppi luotiin HTML-tekniikoilla.

Työn tuloksena oli varaosavalitsimen prototyyppi. Prototyypin pohjalta voidaan tulevaisuudessa kehittää visuaalinen tuotevalitsinmoduuli eri tuotteille eri verkkokauppoihin.

Työn lopputulosta ei otettu vielä käyttöön yhteenkään tiettyyn Solteqin asiakasprojektiin. Tästä syystä työ jäi prototyyppiasteelle.

Avainsanat (asiasanat)
Modulaarisuus, web-suunnittelu, web-kehitys

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Hankomäki, Jami	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages: 28	Permission for web publication: x

Title of publication

Design and development of visual product selector for online store

Degree programme

Media engineering

Supervisor(s)

Niemi, Kari

Assigned by

Solteq Oyj

Abstract

The thesis was assigned by Solteq Oyj, a company offering expertise in multi-channelled trading. A visual spare parts selector module was created to be implemented in webstores. With the module, the location of a spare part on the actual product can be visualized.

In web development, the development process can be made more efficient by separating the application into several modules. Modules can be developed independently from the main project and be added to it once they are ready, which makes testing easier since the modules can be tested on their own, and if there are any bugs, their origins can be found more easily. Modularity might also speed up the development process if the modules are easily copied and added to different parts of the main project.

At first, different development tools to be used in the development of the module were familiarised with. One of the most important ones was the vector graphics format SVG. With SVG, the module could be made responsive, and functionality could be added to it, which would not be possible with traditional rasterised graphics. The prototype of the module was created with HTML techniques.

Result of the work was a prototype of the selection module for the spare parts. In the future, the prototype can be further developed for different products and implemented in different webstores.

The result was not implemented into any Solteq's particular customer project, which is why the work remained in a prototype phase.

Keywords/tags (subjects)

Modularity, web design, web development

Miscellaneous

Sisältö

Kuviot

Koodit

1 Työn lähtökohdat

1.1 Taustaa ja toimeksiantaja

Yksi sovelluskehityksessä käytetyistä keinoista kehitysprosessin tehostamiseen on modulaarisuus. Tällöin kehittäjällä on käytössään pienempiä sovelluksen osia, moduuleja, jotka ovat vähällä vaivalla liitettävissä useisiin eri sovelluksiin. Erityisesti selaimessa toimivissa verkkosovelluksissa modulaarisuus on tehokas tapa nopeuttaa kehitysprosessia, koska verkkosovellukset toimivat samoilla alustoilla ja ovat usein rakenteeltaan samankaltaisia, mikä lisää joustavuutta moduulien käyttöön.

Työn toimeksiantajana toimi Solteq Oyj. Solteq on ohjelmistopalveluyhtiö, joka tarjoaa ohjelmistoratkaisuja ydinliiketoiminnan kehittämiseen niin fyysisessä kaupassa kuin verkossa. Yrityksen liiketoiminta on jaettu kahteen alueeseen: Solteqin digitaaliset ratkaisut tarjoaa palveluita digitaaliseen kaupankäyntiin ja -markkinointiin liittyen. Solteq asiakasratkaisut tarjoaa palveluita logistiikan, myymälä- ja ravintolatoiminnan, asiakaspalveluiden, maksamisen ja kanta-asiakkuuksien hallintaan sekä toiminnan- ja taloudenohjauksen järjestelmiä. (Liiketoiminta-alueet 2016.)

Solteq perustettiin Tampereella vuonna 1982 nimellä TH Tiedonhallinta. Yrityksen nimi muutettiin Solteqiksi vuonna 2000 (Korhonen 2015; Solteq Oyj (STQ) 2017). Historiansa aikana Solteq on tehnyt kaksi merkittävää yrityskauppaa. Vuonna 2012 Solteq osti Aldata Solution Finlandin, minkä myötä yrityksen henkilöstö kasvoi 74 työntekijällä (Laakso 2012). Vuonna 2015 Solteq osti Descom Group Oy:n, jolloin Descomin 240 työntekijää siirtyi Solteqille (Salminen 2015).

Solteq numeroina (Vuosikertomus 2015. 2016):

- Työllistää 500 työntekijää
- Toimipisteitä kolmessa eri maassa: Suomi, Ruotsi ja Puola
    - Suomessa Jyväskylä, Tampere ja Helsinki
- Liikevaihto 54,2 miljoonaa euroa (vuonna 2015)

1.2 Tehtävä ja tavoitteet

Työn tavoitteena oli luoda verkkokaupoissa toimiva moduuli, jossa visuaalisen käyttöliittymän kautta käyttäjät voivat valita varaosia ostoskoriin lisättäväksi. Käyttöliittymän pohjana on sen tuotteen kuva, johon varaosat käyvät, ja kuvan päältä voi klikkaamalla valita haluamansa osan. Moduulin tarkoituksena on olla havainnollistava apuväline tuotteen valinnassa.

Työssä moduulia lähdettiin kehittämään autojen varaosien valitsemista varten. Työtä tehdessä pidettiin kuitenkin mielessä mahdolliset muut käyttötapaukset. Työn lopputuloksena oli prototyyppi, jota toimeksiantaja voi mahdollisesti myydä asiakkailleen.

Työssä keskityttiin moduulin front-endin visuaalisen käyttöliittymän kehitykseen. Back-end rajattiin työn ulkopuolelle.

2 Modulaarinen web-sovelluskehitys

2.1 Modulaarisuus yleisesti

Modulaarisessa kehityksessä kehitettävä tuote jaetaan pienempiin kokonaisuuksiin, jotka voidaan tuottaa itsenäisesti ja, niiden valmistuttua, liittää osaksi lopullista tuotetta. Usein yksittäinen moduuli on jokin sellainen osa tuotteesta, joka toistuu hyvin tai täysin samanlaisena eri puolella koko tuotetta.

Jokaisella moduulilla on tietty rooli kokonaisprojektissa, joka sen tulee täyttää, mikä asettaa jonkin verran rajoituksia niiden kustomointiin. Moduulien valmistamisessa tulee myös pitää ne yhteensopivana itse pääprojektin kanssa, ilman että pääprojektia tai muita moduuleita tarvitsee muokata. Muuten moduulit voivat olla vapaasti kustomoitavissa. Tämän ansiota saattaa koko tuotteen kustomointi ja laajentaminen helpottua.

Modulaarista kehitystä käytetään useilla aloilla, kuten auto-, laiva- ja rakennusteollisuudessa. Tässä työssä keskitytään modulaarisuuteen web-sovellusten kehityksessä.

2.2 Modulaarisuus web-sovelluksissa

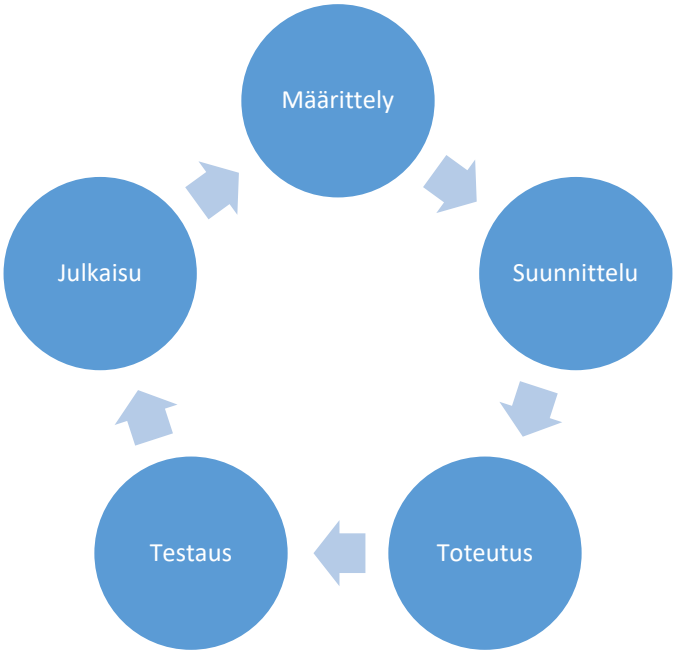
2.2.1 Yleistä web-modulaarisuudesta

Virtuaalisissa sovelluksissa modulaarisuus on erityisen tehokas tapa tehostaa tuotantoa. Siinä missä perinteisen teollisuuden aloilla joudutaan turvautumaan moduulien sarjatuotantoon, voidaan virtuaalisovelluksissa valmistaa moduuli kerran ja käyttää sitä rajattomasti eri puolilla sovellusta ja mahdollisesti jopa eri sovelluksissa. Koska verkkoselaimissa toimivat sovellukset ovat yleensä rakenteeltaan hyvin samanlaisia, on modulaarisuudesta erityisesti niiden kehityksessä hyötyä.

Käytännössä modulaarisuus verkkosivuilla tapahtuu jakamalla sivun eri osat, kuten esimerkiksi header ja footer, omiin komponentteihinsa. Komponenteilla on usein omat tiedostonsa, jotka ladataan yhden isäntäkomponentin sisälle. Linkitys komponenttien välillä voi tapahtua staattisesti tai dynaamisesti. Staattisessa linkityksessä komponenttien suhteet pysyvät samana; tietyn komponentin sisälle tulee aina samat lapsikomponentit. Dynaamisella linkityksellä komponenttien suhteita voidaan muuttaa ohjelmallisesti. Tämä mahdollistaa ns. single-page—sovellusten luomisen, kun esimerkiksi sivun header voidaan pitää aina näkyvillä ja vain sen alapuolella olevaa sisältöä päivitetään tarvittaessa.

2.2.2 Prosessi

Modulaarisuuden yksi merkittävimmistä eduista on, että koko projektin eteneminen ei välttämättä ole riippuvainen yhden moduulin valmistumisesta. Yksittäisen moduulin voi jopa valmistaa kokonaan pääprojektin ulkopuolella ja liittää siihen moduulin valmistuttua. Sekä moduulia että pääsovellusta kehitettäessä voidaan seurata perinteistä sovelluskehityssykliä (ks. kuvio 1). Moduulia kehittäessä tulee vain pitää huolta, että se täyttää modulaarisuuden kriteerit: sen tulee muun muassa olla liitettävissä pääprojektiin ilman, että pääprojektia tarvitsee muokata.



Kuvio 1. Perinteinen sovelluskehityssykli

2.2.3 Edut

Sovelluksen jakaminen moduuleihin on tehokas tapa pitää sovelluksen koodi ja rakenne mahdollisimman luettavana. Kun jokainen moduuli on omassa tiedostossaan, pysyvät tiedostot lyhyempinä, kuin jos kaikki sovelluksen sisältö olisi saman tiedoston sisällä. Tämä voi olla merkittävä etu varsinkin sovelluksissa, joiden kehitykseen osallistuu monta henkilöä, kun jokainen tekijä voi keskittyä yhden moduulin kehittämiseen. Näin myös työhön sisälle pääseminen helpottuu kaikille, jotka eivät ole olleet projektin alusta asti mukana tai jotka siirtyvät projektin sisällä toisen moduulin työstämiseen.

Modulaarisuudesta on hyötyä sovelluksen testaamisessa. Kun sovellus on jaettu useaan osaan, voidaan jokaista moduulia testata erikseen ennen liittämistä pääsovellukseen. Mahdollisesti pääsovelluksen sisälläkin voi testauksen rajata koskemaan tiettyä moduulia. Kun itse pääsovellusta testataan, löytyvien vikojen alkuperä voidaan myös paremmin rajata tiettyyn moduuliin.

2.2.4 Haitat ja puutteet

Pienissä sovelluksissa modulaarisuus ei välttämättä ole eduksi. Se voi lisätä monimutkaisuutta sen sijaan, että sovelluksen rakenne selkeytyisi. Usean moduulin tuottaminen voi lisätä projektin työmäärää, kun moduulien välisten suhteiden toteuttamiseen tarvitaan resursseja, jotka mahdollisesti olisivat enemmän hyödyksi muualla.

Joissakin tapauksissa modulaarisuus voi myös vaikeuttaa sovelluksen ylläpidettävyyttä ja jatkokehitystä. Kun sovellus on hajautettu moneen osaan, voi esimerkiksi tietyn vian kohdentaminen vaikeutua, jos kehittäjä ei tarkalleen tiedä, millaiset suhteet eri moduulien välillä on. Sovelluksen rakenne eli moduulit ja niiden suhteet eivät välttämättä käy millään lailla ilmi, kun sovellusta käytetään selaimessa. Tämä ei päde sovelluksiin, joita ei ole rakennettu modulaarisesti, vaan niissä selaimessa näkyvä rakenne on täysin samanlainen kuin kehitysympäristössä.

3 Työssä käytetyt teknologiat

3.1 Kehitystyökalut

Node.js

Node.js on avoimen lähdekoodin JavaScript-ajoympäristö. Perinteisesti JavaScriptia on käytetty lähinnä selainpohjaisten sovellusten front-endin ohjelmointiin upottamalla JavaScript ohjelmia HTML-sivuille, mutta Node.js:n avulla voidaan ajaa JavaScriptilla koodattuja ohjelmia esimerkiksi palvelinympäristössä. Alun perin Node.js olikin tarkoitettu palvelinpuolen ohjelmointiin, mutta sittemmin sitä on alettu käyttämään myös kehitysympäristöissä tehtävien automatisointiin (Dierx 2016). Node.js on rakennettu Googlen Chrome V8 JavaScript-moottorin päälle (Node.js 2017).

Kun Node.js on asennettu koneelle, sitä voidaan käyttää komentoriviltä. Node.js käynnistetään komennolla ’node’. Tämän jälkeen komentorivillä voidaan ajaa perus JavaScript-komentoja. JavaScript-koodi voidaan myös tallentaa omiin tiedostoihinsa. Näiden koodien ajaminen tapahtuu komennolla ’node *tiedostonimi*.js’.

Jokainen Node.js:n päälle rakennettu projekti tarvitsee projektin juurikansioon package.json-tiedoston (ks. Koodi 1). Tässä tiedostossa luetellaan mm. projektin

nimi, versio, kuvaus ja myös projektin käyttämät laajennukset. Package.json on tärkeä varsinkin, kun projektista halutaan tehdä mahdollisimman helposti kopioitava muille sitä työstäville. Tiedoston avulla on helppo asentaa kehitysympäristöön kaikki tarvittavat laajennukset.

```
{
  "name": "package",
  "version": "1.0.0",
  "description": "Esimerkki package.json",
  "main": "index.js",
  "dependencies": {
    "grunt-contrib-watch": "^1.0.0",
    "grunt": "^1.0.1",
    "grunt-sass": "^1.2.1"
  },
  "devDependencies": {
    "cheerio": "^0.22.0",
    "grunt-browser-sync": "^2.2.0",
    "grunt-contrib-watch": "^1.0.0",
    "grunt-image": "^2.2.2"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Koodi 1. Esimerkki package.json-tiedosto

Yksi Node.js:n isoimmista eduista on juuri laajennettavuus. Siihen on tarjolla useita eri JavaScript-kirjastoja, joiden avulla on mahdollista lisätä toimintoja ja ominaisuuksia Node.js:ään. Laajennusten asennus ja hallinta ovat helppoa Node.js:n mukana tulevan npm:n avulla.

Tässä työssä Node.js:ää tarvittiin Grunt-työkalun käyttämiseen.

NPM

NPM (Node Package Manager) on automaattisesti Node.js:n mukana tuleva työkalu, joka on tarkoitettu laajennusten asentamiseen ja hallintaan. Sillä voi asentaa laajennuksia paikallisesti tai globaalisti. Kun laajennuksia asennetaan paikallisesti, ne lada-

taan projektin sisällä olevaan node_modules -kansioon. Nämä laajennukset ovat käytettävissä vain projektin sisällä. Globaalisti asennetut laajennukset taas ovat käytettävissä kaikkialla koneella, johon ne ovat asennettu.

Node.js ja npm mahdollistavat pitämään projektin mahdollisimman kevyenä ja kehitysympäristön helposti käyttöönotettavana. Kun projektin vaatimat Node.js-laajennukset luetellaan package.json-tiedostossa dependencies-kohdassa (ks. Koodi 1), ei itse laajennuksia tarvitse tallentaa esimerkiksi versionhallintaan, vaan jokainen kehittäjä voi asentaa tarvittavat laajennukset npm:n avulla. Asennus tapahtuu helposti ajamalla komento ’npm install’ projektikansiossa. Tällä komennolla npm osaa automaattisesti katsoa package.jsonin sisältä mitä laajennuksia projekti tarvitsee ja asentaa ne sitten node_modules-kansioon.

Package.jsonissa kohdassa ’ devDependencies’ lueteltuja laajennuksia komento ’npm install’ ei asenna. Tämän avulla kehittäjät voivat ottaa käyttöön omaan kehitysympäristöönsä haluamansa laajennukset joita kokonaisprojekti ei kuitenkaan tarvitse.

NPM:ää tarvittiin tässä työssä yhdessä Node.js:n kanssa Grunt-työkalua varten.

Grunt

Grunt on Node.js:n päällä toimiva automatisointityökalu. Sen avulla voidaan nopeuttaa kehitysprosessia automatisoimalla tietyt, yleensä toistuvat tehtävät kuten tiedostojen minifiointi ja testaamiseen liittyviä tehtäviä. Gruntin käyttöön tarvitaan Node.js ja sen paketinhallintajärjestelmä npm, jonka avulla ladataan paketit, jotka sisältävät Gruntin liitännäiset. Liitännäiset ladataan ja määritellään Gruntfile.js -tiedostossa, minkä jälkeen Grunt-tehtävät voidaan ajaa komentoriviltä.

GruntFile.js-tiedosto alkaa funktiolla, jonka sisälle tulee kaikki Gruntin asetukset (ks. Koodi 2 ensimmäinen rivi). Funktion sisällä alustetaan asetukset niille tarkoitetun objektin sisällä (ks. Koodi 2 toinen rivi), jonka sisällä olevaan pkg-nimiseen avaimeen lisätään arvoksi Node.js:n package.json-tiedosto (ks. Koodi 2 kolmas rivi). Grunt luo package.jsonin pohjalta JavaScript-objektin, jonka avulla Grunt voi käyttää siinä määriteltyjä laajennuksia.

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json')
  });
};
```

Koodi 2. GruntFile.js:n vaatimat määrittelyt

Projektissa käytettävät tehtävät määritellään initConfig-objektin sisällä. Määrittely tapahtuu antamalla ensin tehtävän nimi, joka on yleensä laajennuksen nimi. Nimi toimii avain-arvo—parin avaimena, jonka avaimeksi tulee objekti. Objektin sisällä annetaan laajennuksen asetukset. Esimerkiksi laajennukselle grunt-sass, joka kääntää sass-tiedostoja css-muotoon, määritellään lähde- ja kohdetiedostot. Myös sourcemapin luonti voidaan kääntää päälle tai pois. (ks. Koodi 3.) Laajennuksen vaatimat ja mahdolliset vaihtoehtoiset asetukset vaihtelevat laajennuksittain.

```
sass: {
  options: {
    sourcemap: true,
  },
  files: {
    'app/css/main.css' : 'app/sass/main.scss'
  }
}
```

Koodi 3. Grunt-tehtävän määrittely GruntFile.js-tiedostossa

Laajennuksien lataaminen Gruntiin tapahtuu initConfig-objektin jälkeen. Lataaminen tapahtuu komennolla ”grunt.loadNpmTasks(’*laajennuksen-nimi*’);”. Laajennusten tulee olla asennettuina npm:n kautta, jotta Grunt voi niitä käyttää.

Lopuksi GruntFile.js-tiedostossa luodaan Grunt-tehtävät. Tehtävät luodaan komennolla ”grunt.registerTask(’*tehtävän-nimi*’, [’*suoritettava-laajennus*’])”. Tehtävän sisällä voidaan määritellä suoritettavaksi usea eri laajennus. Laajennukset suoritetaan annetussa järjestyksessä. Tehtävä ajetaan komentorivillä komennolla ”grunt *tehtävän-nimi*”. GruntFile.js-tiedostossa voidaan määritellä oletustehtävä antamalla sille nimi ’default’. Tämä tehtävä ajetaan, kun komentorivillä annetaan vain komento ’grunt’. Tähän tehtävään yleensä sidotaankin projektissa useimmiten käytetyt laajennukset.

```
module.exports = function(grunt) {
    grunt.initConfig({
        pkg: grunt.file.readJSON('package.json'),
        sass: {
          options: {
            sourcemap: true,
          },
          files: {
            'app/css/main.css':'app/sass/main.scss'
          }
        },
        watch: {
            css: {
                files: '**/*.scss',
                tasks: ['sass', 'browserSync']
            }
        },
        browserSync: {
        bsFiles: {
            src : [
                'app/css/*.css',
                'app/*.html',
                'app/js/*.js'
            ],
        },
        options: {
            watchTask: true,
            server: {
                baseDir: "./app"
            }
        }
    }
    });
    grunt.loadNpmTasks('grunt-sass');
    grunt.loadNpmTasks('grunt-contrib-watch');
    grunt.loadNpmTasks('grunt-browser-sync');

    grunt.registerTask('default',['browserSync',
'watch']);

};
```

Koodi 4. Esimerkki GruntFile.js-tiedosto

Tässä työssä Gruntia käytettiin testaamisen automatisointiin ja Sass-tiedostojen kääntämiseen css muotoon.

Sass

Sass (Syntactically awesome stylesheets) on tyylien merkintäkieli, joka on tarkoitettu laajennukseksi perinteiselle css-merkintäkielelle. Syntaksiltaan sass on hyvin samanlainen kuin css, ja sass-tiedostoon voikin kirjoittaa puhdasta css:ää ilman sassin omaa syntaksia. Selaimilla ei kuitenkaan ole sass-tukea, minkä takia sass-muotoiset tyylitiedostot joudutaan kääntämään css-muotoon, ennen kuin niitä voidaan käyttää itse sovelluksessa. Sass-tiedostoilla on tiedostopääte .scss.

Sass mahdollistaa muun muassa muuttujien käytön. Muuttujat auttavat paljon esimerkiksi sivuston värimaailman luonnissa, kun usein käytetyt värit varastoidaan muuttujiin. Näin väriä ei tarvitse erikseen määritellä kaikkialla, missä sitä käytetään, vaan voidaan tämän sijaan viitata värimuuttujaan. Sivuston ilmeen uudistaminen helpottuu huomattavasti, kun värimaailman muuttamiseen riittää muuttujien arvojen määrittäminen uudelleen, eikä kehittäjän tarvitse yksi kerrallaan löytää muutettavia värejä. Sassissa muuttujat määritellään $-merkin avulla (ks. Koodi 5).

Yksi sassin helpoimmin huomattavissa olevista ominaisuuksista on valitsimien nestaus. Tässä tapauksessa nestaaminen tarkoittaa, että valitsimia voidaan kirjoittaa toistensa sisälle. Esimerkiksi .header-valitsimen sisälle voidaan kirjoittaa .title-valitsin, joka tällöin käsittää kaikki title-luokkaiset elementit header-luokkaisten sisällä. Nestaus helpottaa myös pseudoluokkien tyylittelyä. Sassissa viitataan isäntävalitsimeen et-merkillä (&). Näin voidaan esimerkiksi minkä tahansa elementin hover-pseudoluokan tyylit kirjoittaa helposti kyseisen elementin valitsimen sisällä (ks. Koodi 6).

Sassin avulla voidaan myös jakaa kirjoitetut tyylit useaan eri tiedostoon, partiaaliin. Esimerkiksi värimuuttujat voidaan kirjoittaa omaan partiaaliin, ja jokaiselle sivulle voi olla omat partiaalinsa. Tämä helpottaa ylläpidettävyyttä, kun tyylit on jaettu selkeisiin kokonaisuuksiin ja eri tyylien löytäminen helpottuu. Esikääntäjä yhdistää toisiinsa liittyvät partiaalit yhteen css-tiedostoon kääntämisen yhteydessä, minkä ansiota sovelluksen suorittamien http-kutsujen määrä ei pääse kasvamaan, toisin kuin usean eri css-tiedoston kanssa. Sass-partiaalit määritellään alaviivalla (_) tiedostonimen edessä. Näin projektissa yleisesti käytetyille muuttujille tehty partiaali voi olla nimeltään esimerkiksi _variables.scss (ks. Koodi 5).

```
$primary-color: #aacc00;
$secondary-color: #00aacc;
```

Koodi 5. Esimerkki sass-muuttujien partiaalista

Partiaalit tuodaan sass-tiedoston sisälle lauseella "@import '*tiedostonimi*'". Lauseessa tiedostonimen kohdalle ei tule alaviivaa eikä .scss-päätettä, vaan esikääntäjä osaa automaattisesti etsiä tiedoston jossa nämä ovat (ks. Koodi 6). Projektit, joissa olen ollut mukana, on usein ollut käytäntönä tehdä omat partiaalit jokaiselle sivuston eri näkymälle ja komponentille. Näin on helppo tarvittaessa löytää esimerkiksi tyylejä, joita täytyy muokata.

```
@import 'variables';

.header {
  background-color: $secondary-color;

  .title {
    color: $primary-color;

    &:hover {
      color: white;
    }
  }
}
```

Koodi 6. Esimerkki headerin sass-partiaalista

Partiaalit voidaan lopulta tuoda yhteen isäntätiedostoon (ks. Koodi 7), josta esikääntäjä luo yhden css-tiedoston. Näin tyylitiedostoihin käytetyt http-kutsut pysyvät mahdollisimman vähäisinä varsinkin, kun selain tallentaa tiedoston paikallismuistiin. On myös mahdollista luoda sivuston eri osille omat tyylitiedostot. Tällöin tiedostokoot pysyvät pieninä, mutta sivustolla liikkuminen saattaa hidastua, kun käyttäjän tulee siirtyessä ladata uusi tyylitiedosto.

```
@import 'header';
@import 'content';
@import 'footer';
```

Koodi 7. Esimerkki isäntä sass-tiedostosta

Esikääntäjä hakee jokaisen kutsutun partiaalin sisällön ja tuottaa niiden perusteella css-tiedoston. Yllä mainittujen esimerkkien pohjalta luotu css-tiedosto näkyy alla (ks. Koodi 8). Koodiin ei ole merkitty content- ja footer-partiaalien sisältöä (ks. Koodi 7).

```
.header {
  background-color: #00aacc;
}
.header .title {
  color: #aacc00;
}
.header .title:hover {
  color: white;
}
.
.
.
```

Koodi 8. Sass-tiedostojen pohjalta luotu css-tiedosto

Yksi mahdollinen haittapuoli tyylitiedostojen luonnissa sassilla on se, että selainten kehitystyökalut eivät voi saamansa css-tiedoston perusteella tietää missä kohdin mitäkin sass-partiaalia tietty tyyli on annettu. Normaalisti kehitystyökaluissa tarkasteltaessa sovelluksen tyylejä, selain osaa nimetä sen css-tiedoston nimen, jossa jokin tyyli on annettu, ja rivinumeron. Tämä auttaa kehittäjää suuresti esimerkiksi vikojen korjaamisessa. Sassin uusimmat versiot mahdollistavat tämän myös sass-tiedostojen kanssa sourcemapien avulla.

Sourcemapit ovat tiedostoja, jotka kartoittavat css-tyylien alkuperän sass-tiedostoista selaimelle. Näin selaimen kehitystyökaluilla voidaan suoraan nähdä tyylien alkuperä, mikä nopeuttaa korjausten tekoa huomattavasti (Hettler 2014).

Adobe Illustrator CC 2017

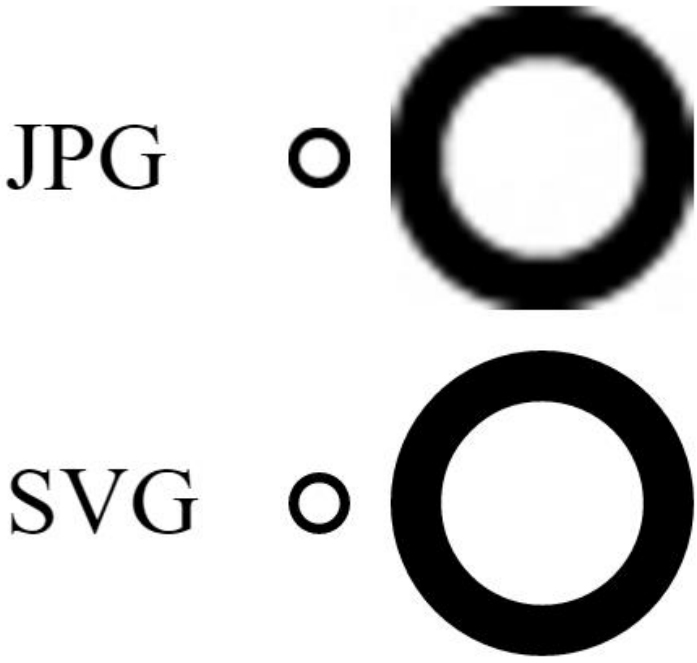
Illustrator on vektorigrafiikan luomiseen ja käsittelyyn tarkoitettu ohjelma. Tässä työssä sitä käytettiin autojen ja varaosien kuvien kehittämiseen ja muokkaamiseen.

3.2 Vektorigrafiikka

3.2.1 Perinteisen grafiikan puutteet

Yleinen tapa esittää grafiikkaa digitaalisessa muodossa on rasterigrafiikka. Rasterigrafiikka toimii tiedostomuodosta huolimatta aina samojen periaatteiden mukaisesti: kuva koostuu pikseleistä eli kuvapisteistä, jotka on asetettu riveihin ja jokaiselle pikselille on annettu oma väriarvonsa. Eri tiedostomuotojen väliset erot ovat lähinnä kuvien pakkaamiseen liittyviä, mutta myös esimerkiksi läpinäkyvyyden ja animoinnin tuki vaihtelee eri muodoissa.

Rasterigrafiikan toimintaperiaate tuo mukanaan tiettyjä puutteita. Yksi on kuvan pikselöityminen sitä suurennettaessa. Kun kuva koostuu tietystä määrästä kuvapisteitä, isommaksi skaalattaessa sen laatu alkaa nopeasti kärsiä, kun yksittäiset kuvapisteet alkavat erottua. Kuviossa 2 ylempänä on 30 pikseliä leveä jpg-kuva ja sen vieressä sama kuva venytetty 150 pikselin leveyteen. Kuvan venytys alkuperäistä isommaksi on aiheuttanut selvää pikselöitymistä. Samassa kuviossa on myös vastaava kuva svg-muodossa, joka on isommaksi skaalautuneena pysynyt selkeänä. Tämän lisäksi kuviossa käytetyn svg:n tiedostokoko on 1 KB, kun taas jpg:n on 40 KB.



Kuvio 2. Ero skaalautuvuudessa jpg:n ja svg:n välillä

Pikselöitymisen estämiseksi kuvista saatetaan tehdä isompia versioita, joita voidaan suurentaa enemmän, ennen kuin niiden laatu alkaa heikentyä. Kuvien suurentaminen kasvattaa kuitenkin myös tiedostokokoa, mikä taas hidastaa sovelluksen toimintaa. Yksi ratkaisu tähän on vektorigrafiikka.

3.2.2 Vektorigrafiikka yleisesti

Vektorigrafiikalla tarkoitetaan matemaattisilla funktioilla piirrettyä grafiikkaa. Vektorikuvat rakentuvat koordinaatistoon sijoitetuista, geometrisesti melko yksinkertaisista muodoista, kuten ympyröistä, monikulmioista ja suorista, jotka voidaan esittää matemaattisilla kaavoilla. Kun sovellus piirtää kuvan näiden kaavojen avulla, voi kuvaa skaalata rajattomasti sen laadun kärsimättä. Aina kuvan suurentuessa ja pienentyessä sovellus laskee uudelleen, miltä kuvan tulisi näyttää, ja pikselöitymistä ei pääse tapahtumaan (SVG 1.1 Concepts 2011).

Toinen vektorigrafiikan tuoma etu rasteriin verrattuna on pitkälti vektorikuvan rakenteen ja toimintaperiaatteen ansiota. Kun vektorikuva on luonnostaan jaettu pienempiin kokonaisuuksiin, erilaisiin geometrisiin muotoihin, voidaan eri osia käsitellä ohjelmallisesti ja niihin voidaan sitoa eri toiminnollisuuksia.

Rasterigrafiikassa vaihtoehto klikattavien alueiden lisäämiseksi kuviin on imagemap. Imagemapilla kuvan päälle määritetään alueita, joihin voidaan sitoa toiminnollisuuksia. Imagemapeissa on kuitenkin paljon puutteita, ja niiden lisääminen sovellukseen voi tuoda paljon erilaisia ongelmia. Esimerkiksi imagemapit skaalautuvat huonosti ja heikentävät helppokäyttöisyyttä. SVG:llä ei ole vastaavia ongelmia (Using SVG as an Alternative To Imagemaps 2013).

SVG:n selaintuki on hyvä; periaatteessa kaikki nykyisin käytössä olevat selaimet tukevat sitä, eikä tavanomaisessa verkkosovelluskehityksessä tarvitse tämän takia rajata vektorigrafiikkaa pois. Tällä hetkellä käytössä olevista selaimista 97,96 % tukee svg:tä (SVG (basic support) 2017).

3.2.3 SVG ja HTML5

SVG-tiedostot ovat XML-muotoisia (ks. Koodi 9). Tämän ansiota niitä voidaan upottaa suoraan HTML-dokumentin sisälle. SVG:n merkintäkieli mahdollistaakin HTML5-teknologioiden kanssa käytettynä useita eri toiminnollisuuksia. Yksi tärkeimmistä on mahdollisuus käsitellä SVG-kuvia CSS:n ja JavaScriptin avulla. Kuvaa ei tarvitse käsitellä yhtenä kokonaisuutena kuten rasterigrafiikkaa, vaan kuvan elementtejä voidaan käsitellä toisistaan erillisinä. Elementeille voidaan myös antaa luokkia, mikä tekee kuvan käsittelystä joustavampaa. Muita SVG:n antamia etuja selainpohjaisissa sovelluksissa ovat esimerkiksi

- Attribuuttien avulla kuvan eri elementteihin voidaan laittaa linkkejä ja niihin voidaan sitoa toiminnollisuuksia.
- Attribuutit mahdollistavat myös ’alt’-tekstin lisäämisen kuvan eri elementeille, mikä lisää käytettävyyttä varsinkin ruudunlukuohjelmia käyttäville; erityisesti jos kuvan elementteihin on sidottu toiminnollisuuksia

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0
50 50">
    <circle cx="25" cy="25" r="25"/>
</svg>
```

Koodi 9. Yksinkertaisen ympyrän SVG-tiedoston sisältö

Attribuuttien lisääminen SVG-elementeille tapahtuu samalla tavalla kuin HTML-elementtien kanssa, eli elementin aloitustunnisteen sisälle voidaan esimerkiksi kirjoittaa ” id=’logo’ ”. Tässä tapauksessa voidaan nyt tyylitiedostossa viitata elementtiin, jonka id on logo. SVG tukee myös joitakin samoja elementtejä kuin HTML. Esimerkiksi hyperlinkin voi määrittää SVG:n sisällä a-elementillä. (ks. Koodi 10.)

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0
50 50">
    <a href="/esim1.html">
        <circle id="logo" cx="25" cy="25" r="25"/>
    </a>
</svg>
```

Koodi 10. SVG, jossa hyperlinkkinä toimiva ympyrä, jonka id on logo

SVG:n aloitustunnisteen sisällä on xmlns-attribuuttina annettu XML nimiavaruus. Tämän avulla määritetään SVG:lle oma nimiavaruus, eli käytännössä selaimelle kerrotaan elementin ja sen sisällön olevan SVG-muotoista. Ilman nimiavaruutta selain tulkitsee SVG-elementit virheellisesti merkatuiksi HTML-elementeiksi. Käytäntönä on määrittää nimiavaruudet URI:n avulla, kuten SVG:n nimiavaruus määritellään URI:lla ”http://www.w3.org/2000/svg”. Selain ei kuitenkaan hae URI:n takaa mitään tietoa, vaan tämä on vain merkkijono, jonka avulla selain tunnistaa elementin SVG-elementiksi (XML Namespaces N.d.).

3.2.4 Vektorigrafiikan puutteet

Vektorigrafiikka soveltuu lähinnä geometrisesti melko yksinkertaisten kuvien luomiseen. Mitä monimutkaisempaa kuvaa luodaan, sitä vähemmän vektorigrafiikan käyttö kannattaa ja valokuvien tapaisten kuvien näyttäminen vektorigrafiikalla onkin jo käytännössä mahdotonta. Jos kuvassa näkyvät monimutkaiset muodot yritetään muodostaa matemaattisin funktioin, joko kuvan laatu kärsii, kun sitä joudutaan yksinkertaistamaan, tai sitten sen esittämiseen tarvitaan niin monta eri vektorigrafiikkaelementtiä, että tiedostokoko voi kasvaa kohtuuttoman suureksi. Tästä syystä monimutkaiset kuvat, jotka eivät seuraa selkeitä ja yksinkertaisia matemaattisia kaavoja, onkin parempi näyttää rasterimuotoisina. Tällöin tiedostokoko pysyy mahdollisimman pienenä, eikä kuvan laatu välttämättä kärsi.

4 Tuotevalitsinmoduulin kehitys

4.1 Työn vaatimukset

Sovelluksen käyttötarkoituksena oli olla verkkokaupassa toimiva, havainnollistava työkalu varaosien valinnassa. Ongelmana pelkässä tekstipohjaisessa valitsimessa on se, ettei osan sijaintia tuotteessa voi välttämättä tarkkaan kuvailla lyhyehköllä tekstillä. Esimerkiksi puhuttaessa tuotteen ’oikeasta puolesta’ voi joku käsittää sen niin, että tarkoitetaan oikeaa puolta edestäpäin katsottuna, kun taas toiset käsittävät oikeaksi puoleksi takaapäin katsottuna. Sovelluksessa tuotteen kuvan avulla käyttäjät voivat valita haluamansa varaosan haluamalleen puolelle.

Työn lopputuloksen toivottiin olevan mahdollisimman helposti sovellettavissa eri verkkokauppoihin, mikä tarkoitti, että sen tuli olla helposti kustomoitavissa. Sovelluksen toivottiin myös täyttävän nykyaikaisen verkkosovelluksen vaatimukset. Sen tuli olla siis responsiivinen, eli toimia eri päätelaitteilla sellaisenaan, ja sen tuli olla helppokäyttöinen. Helppokäyttöisyydellä tarkoitetaan, että käyttäjät, jotka selaavat verkkokauppaa ruudunlukuohjelmaa käyttäen, lähinnä yleensä näkövammaiset ihmiset, voivat myös käyttää varaosavalitsinta.

Sovellus oli alunperin tarkoitettu autojen varaosien valitsimiseen, mutta toimeksiantajan toivomuksesta sovellusta kehitettiin mahdolliset muut käyttökohteet mielessä pitäen.

4.2 Suunnittelu

Toimeksiantajalla oli jo ennen työn alkua selkeä kuva siitä, millainen sovelluksen tulisi olla. Toimeksiantaja oli kokenut verkkokauppojen kehittäjä, ja tämä näkyi myös tämän työn suunnitteluvaiheessa. Heillä oli jo ennen työn alkua selkeä visio siitä, miten sovellus toimii ja osittain myös siinä käytettävistä teknologioista. Tästä syystä oma roolini jäi melko vähäiseksi työn varsinaisessa suunnitteluvaiheessa.

Sovelluksen toiminta oli suunniteltu erityisesti niin, että siinä hyödynnettäisiin vektorigrafiikkaa. Sen avulla sovelluksessa voitaisiin dynaamisesti korostaa valittu varaosa ja itse valintakin voitaisiin sitoa svg-elementteihin, jolloin käyttäjät voivat valita haluamansa osan suoraan kuvan päältä klikkaamalla tai sormella painamalla. Vektorigrafiikkaa hyödyntämällä sovelluksesta voitaisiin tehdä myös helppokäyttöinen, kun ruudunlukuohjelmia varten voitaisiin svg-elementeille lisätä puhtaasti tekstimuotoista dataa.

Vaikka työ olikin periaatteeltaan pitkälle suunniteltu, ennen kuin liityin sen kehitykseen, oli siinä käytettävien teknologioiden suhteen loppujen lopuksi aika paljonkin jätetty avoimeksi. Esimerkiksi sovelluksen käsittelemää, tietokannasta saapuvaa dataa ja sen muotoa ei ollut määritelty. Kaikkia sovelluksen ominaisuuksia ei voinut tarkalleen määritellä siitä syystä, kun sillä ei ollut varsinaista ostajaa, vaan sen oli tarkoitus olla mahdollinen myytävä tuote usealle potentiaaliselle asiakkaalle ja useaan eri tarkoitukseen.

Sovelluksen käyttökohteeksi kehitysvaihetta varten valikoitui autojen varaosakauppa, koska toimeksiantajalla oli tällainen kehitysprojekti jo erään asiakkaan kanssa sovittuna. Asiakas ei kuitenkaan ollut tilannut visuaalista varaosavalitsinta, vaan valitsinta oltaisiin mahdollisesti myyty erikseen.

4.3 Toteutus

4.3.1 Kuvien luonti

Ensimmäinen vaihe työn toteutuksessa oli auton svg-kuvan luonti. Kuvan luonti tapahtui Illustratorilla. Kuvan piirtämisessä käytettiin apuna oikean auton kuvaa. Kuva asetettiin Illustratorin työtilan pohjalle ja tämän päälle oli helppo piirtää auton isoimmat osat, kuten runko, ikkunat ja peruutuspeilit. Piirtämisessä tuli kuitenkin vastaan ongelmia.

Ensimmäisenä ongelmia tuli siitä, että varaosakaupassa, johon työ mahdollisesti tultaisiin upottamaan, on osia useisiin erilaisiin autoihin. Jokaisen auton piirtäminen olisi ollut liian iso työ. Se myös tekisi sovelluksen ylläpidettävyyden vaikeaksi, kun aina uusien autojen osien myötä tulisi piirtää uusia kuvia. Tästä syystä sovellukseen päätettiin tehdä yksi yleisluontoinen auton kuva, jota käytettäisiin kaikkien automallien kanssa. Tämä kuitenkin tarkoitti väistämättä sitä, että sovelluksessa kuva ei vastaisi todellisuutta kaikkien osien sijainnin ja koon suhteen.

Ongelmia tuotti myös auton sisäosien piirtäminen. Erityisesti auton moottori oli vaikea piirtää vektorigrafiikkana. Siksi prototyyppiä varten päädyttiin moottorin sijaan piirtämään pelkkä konepelti. Jos käyttäjä siis valitsisi moottorin osan, korostettaisiin kuvasta konepelti.

Koska kuva oli tarkoitettu käyttöön web-sovelluksessa, piti kuvaa luodessa pyrkiä tekemään kuvasta mahdollisimman pieni tiedostokooltaan. Käytännössä tämä tarkoitti sitä, että kuvan eri elementit pyrittiin pitämään geometrisesti mahdollisimman yksinkertaisina. Sovelluksen toiminnollisuuksia silmällä pitäen kuvan rakennetta jäsenneltiin ryhmittämällä osia kokonaisuuksiin, joita sovelluksessa saatettaisiin tarvita. Esimerkiksi jarrupalat olivat ryhmitetty. Näin voitaisiin sovelluksessa helposti korostaa

kaikki jarrupalat korostamalla ryhmä sen sijaan, että jokainen pala jouduttaisiin erikseen korostamaan.

Illustratorin työkalut SVG-tiedoston luontiin olivat jokseenkin puutteelliset, vaikka ne ovatkin työssä käytetyssä Illustrator CC:ssä kehittyneemmät, kuin aiemmissa versioissa. CC:ssä voitiin suoraan minifioida SVG-tiedosto, eikä tiedostoon tullut Illustratorin käyttämää metadataa. Merkittävä puute työn kannalta oli erityisesti se, ettei kuvan elementeille voitu antaa omia luokkia. Sen sijaan Illustratorilla voitiin antaa vain id:t, jotka tulivat tasojen (layer) nimien perusteella. Luokkia tarvittiin sovelluksen toiminnollisuuksia varten, joten ne jouduttiin lisäämään manuaalisesti.

4.3.2 HTML-prototyypin kehitys

Kun autosta oli luotu svg-kuva, alettiin kehittää selaimessa toimivaa html-prototyyppiä. Ensimmäinen vaihe oli kehitysympäristön pystyttäminen. Työkoneella oli jo valmiiksi asennettuna Node.js, npm ja Grunt. Kehitystä varten asennettiin Gruntiin liitännäiset sass-tiedostojen kääntämistä varten ja sovelluksen testausta varten. Testaamiseen käytettiin Gruntin grunt-browser-sync ja grunt-contrib-watch liitännäisiä.

Browser-sync mahdollisti sovelluksen testaamisen usealla eri selaimella samanaikaisesti. Sen avulla voitiin sovellusta käyttää yhdessä selaimessa ja kaikki suoritetut toiminnot, kuten linkkien klikkaukset, tapahtuivat samaan aikaan muissakin selaimissa, joissa sovellus oli auki. Tämä vähensi runsaasti testaamiseen kuluvaa aikaa. Periaatteessa sovellusta tarvitsi testata vain kerran yhdellä alustalla sen sijaan, että sitä testattaisiin kerran jokaisella eri alustalla.

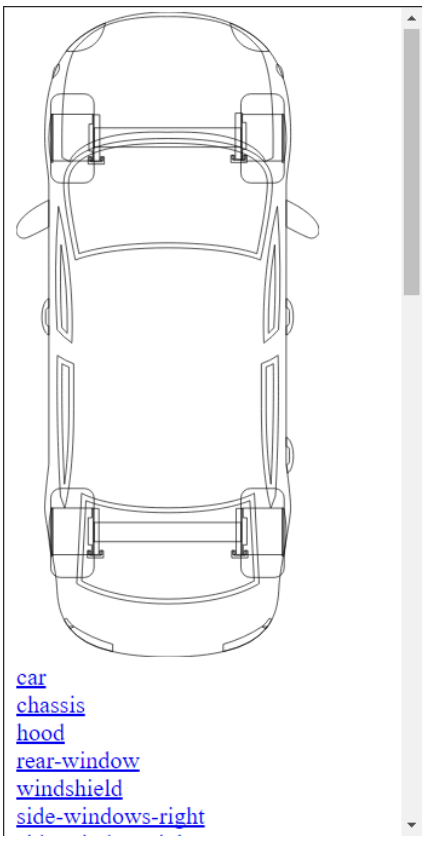
Grunt-contrib-watch on tarkoitettu tiedostojen tarkkailuun muutosten varalta ja sen voi määrittää suorittamaan minkä tahansa Grunt-tehtävän, kun se huomaa muutoksen tiedostossa. Tässä työssä se määriteltiin tarkkailemaan sovelluksen sass-, html-, css- ja JavaScript-tiedostoja. Jos sass-tiedostossa havaittiin muutos, ajettiin tehtävä, joka käänsi sass-tiedostoon tulleet muutokset sovelluksen css-tiedostoon. Kun muutos tapahtui jossain selaimen käyttämässä tiedostossa, kutsuttiin browser-sync—tehtävää, joka päivitti muutokset kaikkiin testauksessa oleviin selaimiin.

Itse prototyypin kehitys alkoi html-tiedoston luonnilla, johon upotettiin autosta luotu SVG-kuva ja linkitettiin css-tiedosto ja JavaScript-tiedosto. SVG-kuvalle määritettiin

tyyleissä kaikille osille mustat ääriviivat. Niiden sisäosat määriteltiin läpinäkyviksi, koska selaimet oletuksena piirtävät SVG-elementit kokonaan mustana. Sovellukseen kirjoitettiin JavaScriptilla toiminto, joka listasi kuvassa olevat osat ja teki niistä hyperlinkit (ks. Koodi 11. Auton SVG on mainImage-muuttujassa.). Hyperlinkkiä klikkaamalla voitaisiin korostaa kyseinen osa. Hyperlinkkien luontia varten valittiin kaikki SVG:ssä olevat elementit. Jokaiselta elementiltä otettiin sen id ja luotiin hyperlinkki, joka linkitettiin kyseiseen osaan. Linkin tekstiksi tuli osan id. Linkit lisättiin luonnin jälkeen sivulle (ks. Kuvio 3 alaosa).

```
mainImage.querySelectorAll('*').forEach(function
(item) {
        // create link element
        var link = document.createElement('a');
        link.href = '#' + item.id;
        link.innerHTML = item.id;
        link.classList.add('part-link');
        document.body.appendChild(link);
});
```

Koodi 11. Hyperlinkkien luonti SVG-elementtien pohjalta

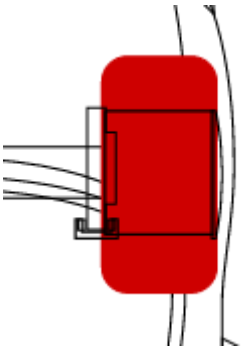


Kuvio 3. Valitsimen ensimmäinen prototyyppi

Linkeille lisättiin tapahtumankuuntelijat klikkausta varten (ks. Koodi 12). Linkeiltä myös poistettiin oletustoiminto, eli linkin osoitteen seuraaminen, joka olisi päivittänyt sivun. Kun osan linkkiä klikattiin, muokattiin SVG:tä muun muassa niin, että auton osalle lisättiin luokka, jonka avulla sitä voitiin tyylien avulla korostaa. Tässä tuotti ongelmia kuitenkin se, että SVG:n muokkausmahdollisuudet tyyleillä ovat rajalliset. SVG-elementtien järjestystä ei voi muuttaa niin kuin HTML-elementtien järjestystä z-indexillä. Tästä syystä korostettu osa saattoi jäädä toisten osien alle, kun niiden äärireunat piirtyivät korostetun osan päälle (ks. Kuvio 4).

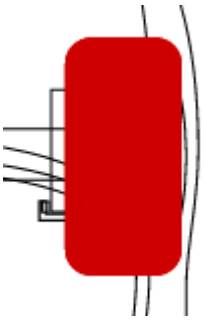
```
document.querySelectorAll('.part-link').fo-
rEach(function (item) {
    item.addEventListener('click', function (e) {
        e.preventDefault();

        if (activePartLink) {
            activePartLink.classList.remove('active-
part-link');
        }

        activePartLink = e.target;
        activePartLink.classList.add('active-part-
link');
        var id = e.target.href.split('#')[1];
        modifySVG(id);
    });
});
```

Koodi 12. Osien linkkien käsittely



Kuvio 4. Korostettu oikea eturengas, joka ei ole kuvassa päällimmäisenä

SVG-elementit piirtyvät siinä järjestyksessä, kuin ne on kirjattu. Tästä syystä viimeisenä kirjattu elementti tulee kuvassa päällimmäiseksi. Ongelman ratkaisemisessa tätä käytettiin hyväksi lisäämällä SVG:n loppuun use-elementti. SVG:n use-elementtiin voidaan linkittää mikä tahansa muu SVG-kuva tai -elementti. Kun tähän elementtiin sitten linkitettiin korostettu osa, piirtyi se kuvassa päällimmäiseksi (ks. Kuvio 5).



Kuvio 5. Korostettu oikea eturengas, joka piirtyy kuvassa päällimmäiseksi

Use-elementti piti kuitenkin poistaa ja lisätä uudestaan sen sijaan, että vain sen sisältöä olisi dynaamisesti muutettu. Muutoin kuvassa ei tapahtunut muutosta, kun uusi osa valittiin. Sama ongelma toistui myös, jos use-elementti lisättiin SVG:n sisälle JavaScriptin appendChild-metodilla. Tämä luultavasti johtui selaimen moottorin tavasta piirtää SVG-kuva. Ongelma ratkaistiin lisäämällä elementti kuvalle sen innerHTML-ominaisuuden avulla (ks. Koodi 13).

```
var useNode = document.getElementById('highlight-
ed');

mainImage.querySelectorAll('*').forEach(function
(item) {
        item.classList.remove('part--active')
    });

    mainImage.querySelector('#' +
id).classList.add('part--active');

    if(useNode) {
        mainImage.removeChild(useNode);
    }

    mainImage.innerHTML = mainImage.innerHTML +
"<use id='highlighted' href='#" + id + "'></use>";
```

Koodi 13. Use-elementin poistaminen ja uudelleen lisääminen

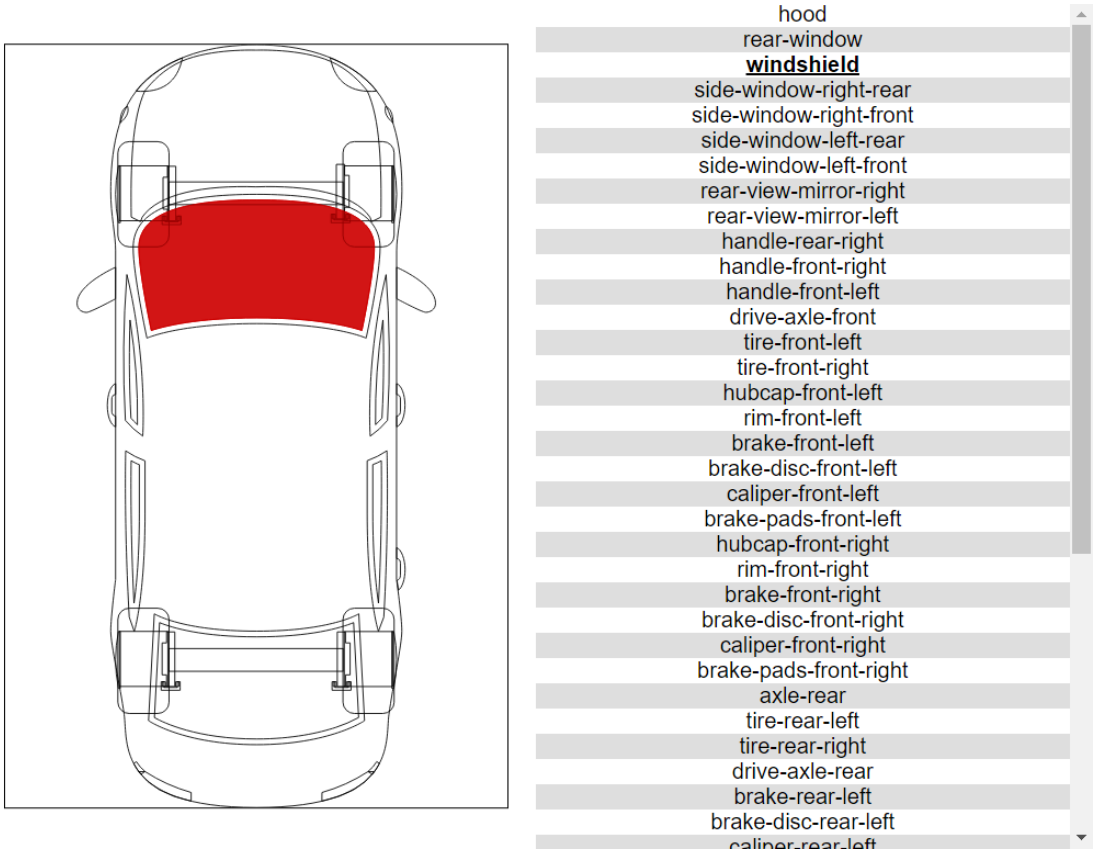
Koska kaikkia kuvassa näkyviä osia ei varaosakaupasta löytynyt, ei kaikkia haluttu myöskään listata, vaikka ne kuvassa olivatkin. Näitä oli esimerkiksi auton runko. Tämän ongelman ratkaisemiseksi tarvittiin kuvan elementeille lisätä luokkia, joiden perusteella osat joko lisätään listalle tai jätetään siltä pois. Koska Illustratorilla ei voinut luokkia lisätä, ne piti tehdä manuaalisesti tekstieditorilla. Työssä osille, joita ei haluttu listata annettiin luokka ’non-highlightable’.

Lisäksi haluttiin varmistaa, että listalle lisätään linkkejä vain tiettyihin elementteihin. SVG:llä voi olla sellaisia elementtityyppejä, joita ei haluta listalle listata, kuten tekstejä ja filttereitä. Sovellus rajattiin listaamaan vain elementtejä, jotka ovat geometrisia muotoja, kuten ympyröitä, suoria tai nelikulmioita. (ks. Koodi 14.)

```
mainImage.querySelectorAll('circle, ellipse, g,
line, path, polygon, polyline, rect').forEach(func-
tion (item) {
    if(!item.classList.contains('non-highlighta-
ble')){
      // linkkien luonti
    }
});
```

Koodi 14. Linkkien luonti graafisille elementeille, joilla ei ole luokkaa non-highlightable

Lopuksi sovellukseen lisättiin oma elementti osien linkeille ja sovellusta tyyliteltiin siistimmän näköiseksi. Joka toiselle linkille lisättiin eri värinen tausta ja tekstiä muokattiin niin, ettei niissä näy selainten oletustyylit. Kuva ja lista laitettiin myös rinnakkain isommilla näytöillä ja pienemmillä ne menevät allekkain.



Kuvio 6. Valitsimen lopullinen prototyyppi

5 Tulokset ja pohdinta

Työn lopputuloksena oli yksinkertainen autojen varaosien valitsin, joka täytti sille asetetut vähimmäisvaatimukset. Työ eteni pitkälti ilman suurempia teknisiä ongelmia ja ne ongelmat, joita vastaan tuli saatiin ratkaistua. Prototyyppiin jäi kuitenkin vielä paljon asioita mahdolliseen jatkokehitykseen. Tähän syynä oli melko yleisluontoinen määrittely, jossa moni asia sovelluksessa jätettiin määrittelemättä. Näitä oli esimerkiksi itse sovelluksen käyttökohde ja sen käyttämä data. Nämä asiat tuleekin määritellä sitten, kun prototyyppiä aletaan jatkokehittämään tiettyyn asiakasprojektiin.

Ennen työn aloittamista tällainen tapa jättää moni asia avoimeksi tuntui jopa hyvältä. Ajattelin kehitysprosessin helpottuvan, kun ei tarvinnut tiukkaan noudattaa ennalta määrättyä suunnitelmaa. Kävi kuitenkin päinvastoin ja kehitysprosessi vaikeutui. Esimerkiksi sovelluksen tarvitsema back-endistä saapuvan datan muoto olisi ollut tärkeä tietää, jotta front-end oltaisiin voitu suunnitella oikeanlaiseksi. Tämän työn puitteissa

kävikin ilmi, miten rajoittavaa modulaarisuus voi olla sovelluksen kehitykselle. Sovellukselle ei voitu toteuttaa erityisen tarkkaan eri ominaisuuksia, koska nämä ominaisuudet eivät välttämättä olisi sopinut kaikkiin pääprojekteihin, joihin varaosavalitsinmoduulia oltaisiin liittämässä.

Sovelluksen kehitysprosessin aikana kävi myös selväksi, että sovellus ei olisi käytännöllinen, eikä välttämättä edes tarpeellinen autojen varaosien valintaan. Aluksi oltiin suunniteltu, että kuvan päältä klikkaamalla voisi valita osia. Autoissa on kuitenkin niin paljon eri kokoisia osia, että pienimmät osat, kuten jarrupalat eivät välttämättä edes näy kuvassa, saati sitten, että niitä voisi kuvan päältä klikkaamalla valita. Osia on myös päällekkäin autossa, eikä sovellus voisi klikkauksen perusteella mitenkään tietää mitä päällekkäistä osaa käyttäjä yrittää valita. Tämän prototyypin osalla jouduttiinkin tyytymään siihen, että käyttäjä valitsee perinteiseltä listalta osan, joka sitten korostetaan kuvassa.

Koen kuitenkin, että tämä työ opetti minulle paljon; ehkä jopa enemmän, kuin vastaava projekti, joka ei olisi samoista ongelmista kärsinyt. Tärkeimpänä opetuksena pidän ehdottomasti määrittelyn tarkkuuden hyödyllisyyttä. Jos projektin päämäärää ei ole riittävän hyvin suunniteltu, käy kehitysprosessi vaikeaksi, ellei jopa mahdottomaksi.

Kenties jokseenkin valitettavaa tämän työn osalta oli, ettei työllä ollut varsinaista asiakasta. Tästä johtuen moduulia ei päästy liittämään osaksi isompaa projektia. Näin ollen tästä oleellisesta modulaarisen sovelluskehitysprosessin vaiheesta ei saatu tämän työn puitteissa kokemusta. Prototyypin pohjalta on kuitenkin helppo lähteä rakentamaan valmista moduulia mahdollista tulevaa asiakasprojektia varten.

Lähteet

Dierx, P. 2016. A Beginner's Guide to npm – the Node Package Manager. SitePoint 30.3.2016. https://www.sitepoint.com/beginners-guide-node-package-manager/

Hettler, T. 2014. Using source maps with Sass 3.3. The Sass Way 7.6.2014. Viitattu 7.5.2017. http://thesassway.com/intermediate/using-source-maps-with-sass

Korhonen, S. 2015. Solteqin perustaja jättää yrityksen. Tivi 25.9.2015. Viitattu 31.1.2017. http://www.tivi.fi/Kaikki_uutiset/solteqin-perustaja-jattaa-yrityksen-6000113

Laakso, H. 2012. Solteq ostaa Aldata Solution Finlandin. Tivi 20.3.2012. Viitattu 31.1.2017. http://www.tivi.fi/Arkisto/2012-03-20/Solteq-ostaa-Aldata-Solution-Finlandin-3190849.html

Liiketoiminta-alueet. 2016. Kuvaus Solteqin liiketoiminta-alueista. Viitattu 30.1.2017. https://www.solteq.com/fi/sijoittajat/solteq-sijoituskohteena/liiketoiminta-alueet/

Node.js. 2017. Node.js:n kotisivusto. Viitattu 1.5.2017. https://nodejs.org/en/

Salminen, P. 2015. Solteq ostaa jyväskyläläisen Descomin. Keskisuomalainen 19.6.2015. Viitattu 31.1.2017. http://www.ksml.fi/talous/Solteq-ostaa-jyv%C3%A4skyl%C3%A4l%C3%A4isen-Descomin/365565

Solteq Oyj (STQ). 2017. Kauppalehden osaketiedot Solteqin osakkeista. Viitattu 31.1.2017. http://www.kauppalehti.fi/5/i/porssi/porssikurssit/osake/?klid=1202

SVG 1.1 Concepts. 2011. SVG:n tärkeimpien käsitteiden selitteet. Viitattu 26.4.2017. https://www.w3.org/TR/SVG11/concepts.html

SVG (basic support). 2017. Can I use –sivuston luettelo SVG:tä tukevista selaimista. Viitattu 10.4.2017. http://caniuse.com/#feat=svg

Using SVG as an Alternative To Imagemaps. 2013. Vertailu SVG:n ja imagemap-tekniikan välillä. Viitattu 24.4.2017. http://thenewcode.com/696/Using-SVG-as-an-Alternative-To-Imagemaps

Vuosikertomus 2015. 2016. Solteqin vuoden 2015 vuosikertomus. Viitattu 30.1.2017. https://www.solteq.com/fi/sijoittajat/tiedotteet-ja-julkaisut/vuosikertomukset-ja-tilinpaatokset/vuosikertomus-2015/

XML Namespaces. N.d. Ohjeita ja tietoa liittyen XML nimiavaruuksiin. Viitattu 10.5.2017. https://www.w3schools.com/xml/xml_namespaces.asp