Nuutti Lainesalo

KARTTAKOMPONENTIN SUUNNITTELU JA TOTEUTUS WUUDIS-PALVELUUN

Opinnäytetyö
Tietojenkäsittelyn koulutus

2018

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Nuutti Lainesalo	Tradenomi (AMK)	Helmikuu 2018

Opinnäytetyön nimi

Karttakomponentin suunnittelu ja toteutus Wuudis-palveluun

38 sivua

Toimeksiantaja

MHG Systems Oy

Ohjaaja

Janne Turunen

Tiivistelmä

Tässä opinnäytetyössä suunnitellaan ja toteutetaan karttakomponentti mikkeliläisen MHG Systems Oy:n Wuudis-palveluun. Opinnäytetyö käsittelee karttakomponentin käyttöliittymän ja sen lähdekoodin toteutusta ja suunnittelua Angular 5 -ohjelmistokehyksen avulla. Myös siirtymistä AngularJs-ohjelmistokehyksestä Angular-ohjelmistokehykseen käsitellään toteutusvaiheessa.

Opinnäytetyö alkaa lähtökohdilla, jossa käydään läpi Wuudis-palvelun tämän hetkinen tilanne sekä uuden Wuudis-palvelun tavoitteet. Opinnäytetyössä käydään läpi käytettävät teknologiat ja käsitteet kuten Angular, Node.js ja Material Design.

Kun lähtökohdat ovat selitetty, aloitetaan suunnittelun ja toteutuksen esittely, alkaen rautalankamallintamisesta. Tässä opinnäytetyö käy lyhyesti läpi rautalankamallintamisen terminä sekä esittelee rautalankamallin uudesta karttakomponentista. Seuraavana aloitetaan opinnäytetyön tekninen toteutus, jossa esittelemme jokaisen karttakomponentin suunnittelun ja toteutuksen. Opinnäytetyö esittelee myös valmiita komponentteja sekä koodillisesti että kuvallisesti.

Opinnäytetyö onnistuu pääsemään tavoitteeseensa ja vastaa toimeksiantajan antamia vaatimusmäärittelyjä. Lopputuloksena uusi karttakomponentti on kehitetty siten, että sen ominaisuudet vastaavat pääpiirteittäin nykyisen karttakomponentin ominaisuuksia. Käymme myös läpi nykyisen karttakomponentin tilanteen sekä eri tapoja toteuttaa komponentteja.

Asiasanat

ohjelmointi, karttapalvelut, metsätilat, JavaScript, Angular

Author (authors)	Degree	Time
Nuutti Lainesalo	Bachelor of Business Administration	February 2018

Thesis title

Designing and implementing a map component for Wuudis

38 pages

Commissioned by

MHG Systems Ltd

Supervisor

Janne Turunen

Abstract

The objective of this thesis was to design and implement a map component using the Angular framework. The component was commissioned by MHG Systems Ltd for their Wuudis forest management service. The thesis investigated planning and creating a map component using OpenLayers and the Angular 5 web application platform. It also touched the subject of migrating from AngularJs to Angular 5.

The thesis began by explaining the premise in which it surveyed the status of Wuudis and the requirements for the next version. In addition, it explained the technologies and concepts to be used in creating the map component. Terms such as the Angular platform, Node.js runtime and the Material Design language were briefly touched upon.

After the groundwork was done the thesis started on explaining the process of design and implementation. Beginning with planning and designing, it explained what a wireframe model was. Then the actual process of implementation was started by going through the design and implementation of every map component. It also shows completed components.

The thesis managed to reach its objective. The new map component managed to meet the requirements set by the client and paved way for further its development. It approximately had the same features as the map component in the current Wuudis service. The thesis also presented different ways to implement the component.

Keywords

programming, map services, forest management, JavaScript, Angular

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on suunnitella ja toteuttaa karttakomponentti MHG Systems Oy:n Wuudis-palveluun. MHG Systems Oy on mikkeliläinen yritys, joka tarjoaa ratkaisuja bioenergialiiketoimintaan. Wuudis on SaaS-palvelu, jolla metsänomistaja tai metsätoimija visualisoi ja hallitsee metsävaratietojaan. Palvelu on saatavissa kahdessa eri paketissa; Wuudis ja Wuudis Pro, joista jälkimmäinen on kuukausimaksullinen ja suunnattu puuntuottajille, yrityksille ja organisaatioille.

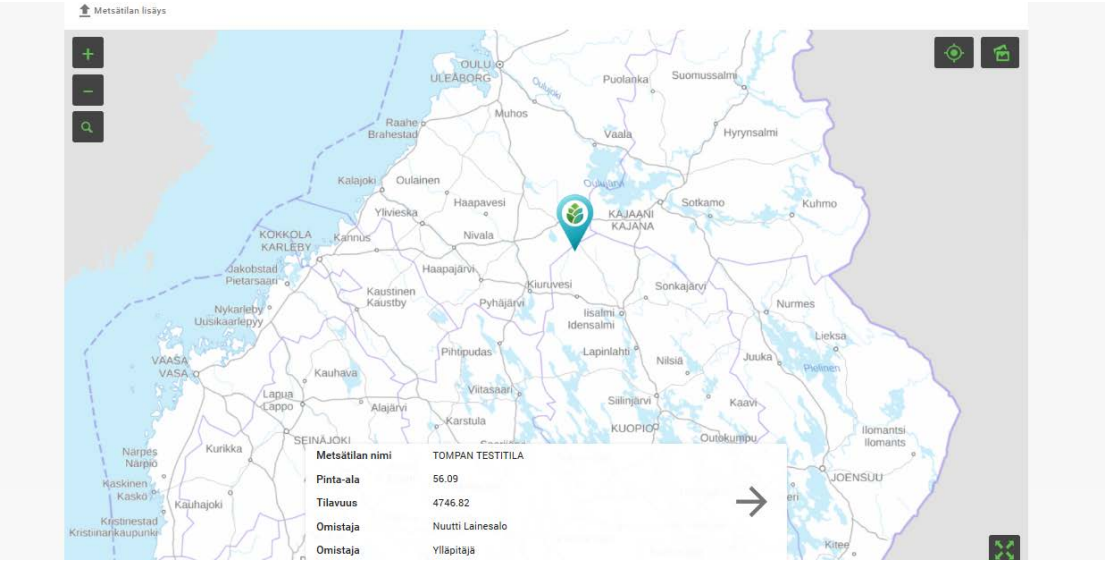
Raportti alkaa toisessa luvussa, jossa esittelen tämänhetkisen Wuudis-palvelun karttakomponentin tilan, toimeksiantoni ja uuden Wuudis-palvelun tilan. Käyn läpi myös toimeksiantajan antamat vaatimukset. Itse toteutus alkaa kolmannessa luvussa. Kerron lyhyesti käytettävistä teknologioista ja niiden käyttöönotoista luvuissa 3.1 ja 3.2. Käyn läpi lyhyesti myös projektissa käytettävän Material Designin ja sen käyttöönoton kappaleessa 3.3.

Luvussa 3.4 käyn läpi rautalankamallin käsitteenä, esittelen rautalankamallia Wuudiksesta ja kerron parannusehdotuksia vanhaan karttaan. Luvusta 3.5 eteenpäin aloitan ohjelmakoodin kirjoittamisen, alkaen metsätilanäkymän luonnista. Käyn läpi Angularin *router*-moduulin sekä komponenttirakenteen. Tämän jälkeen kerron OpenLayers-kirjastosta ja kartan luomisesta Angular-ympäristössä.

Alkaen luvusta 3.7, aloitan kartan alikomponenttien luomisen, lähtien metsätilojen listauksesta. Luvussa 3.8 kerron Angularin *pipe*-ominaisuudesta ja hakutoiminnallisuuden toteuttamisesta. Luvuissa 3.9-3.11 käyn läpi tiedon esittämistä infopaneeli-komponentin ja listojen avulla. Viimeisessä kolmannen luvun osassa käyn läpi jQuery-kirjaston käyttöönoton Angular-projektissa sekä kartta-asetusten valikon. Neljännessä luvussa käyn lyhyesti läpi lopputulosta, karttakomponentin nykyistä tilannetta sekä ajatuksiani aiheesta.

2 LÄHTÖKOHDAT

Nykyisen Wuudiksen selainpuoli on toteutettu Googlen AngularJs -ohjelmistokehyksellä. Minulle annettiin opinnäytetyön aiheeksi tehdä karttakomponentti uuteen Wuudis-palveluun, joka on toteutettu Angular 4 -ohjelmistokehyksellä. Alkutekijöikseni meninkin tarkastelemaan nykyistä Wuudiksen karttaa. Se koostuu karkeasti jaoteltuna kahdesta eri osasta: Metsätilat-näkymä (kuva 1), jossa kartalla näkyvät kaikki käyttäjän metsätilat, joihin hänellä on pääsyoikeus, sekä yksittäisestä metsätilanäkymästä, jossa kartalle piirtyvät yhden metsätilan kuviot, havainnot ja muut karttaobjektit.



Kuva 1 Metsätilat-näkymä

Kartta tarjoaa monia eri vaihtoehtoja sen käyttämiseen kuten esimerkiksi taustakartan vaihtamisen (esim. OpenStreetMaps tai Maanmittauslaitoksen taustakartta), kiinteistörajojen ja -tunnusten näyttö, teemakarttojen vaihto sekä kuvionpiirtotyökalut.

Toimeksiantajan vaatimukset uudelle karttakomponentille olivat vähäiset. Vaatimuksina oli mm. Angular 4 -ohjelmistokehyksen käyttö. Muuten sain vapaat kädet suunnitella uudenlaisen karttakomponentin, kunhan hyväksytin ideat toimeksiantajalla.

3 SUUNNITTELU JA TOTEUTUS

Tutustuttuani Wuudiksen tämän hetkiseen tilanteeseen, laitoin työympäristöni käyttökuntoon. Ohjelmointiin otin käyttöön JetBrains Webstorm IDE -ohjelmointiympäristön. Se on maksullinen, JavaScriptin ja TypeScriptin kirjoittamiseen tarkoitettu kehitysympäristö. Se sisältää myös kehittyneen tuen moderneille ohjelmistokehyksille, kuten Angularille. (Jetbrains, 2018.) Valitsin Webstormin, koska sen IDE-ominaisuudet tulevat varmasti hyödyllisiksi isoimmissa projekteissa, kuten uudessa Wuudiksessa. Tämän jälkeen asensin Node.js -runtimen, ja latasin versionhallinnasta uuden Wuudiksen projektin ja asensin sen käyttämällä ”npm install” -komentoa.

3.1 Node.js ja Node Package Manager

Node.js on Google Chromen V8 -JavaScript-moottorin päälle rakennettu avoimen lähdekoodin runtime, joka mahdollistaa JavaScriptin käyttämisen mm. palvelinpään ohjelmoinnissa. Sen mukana tulee myös paketinhallintaohjelma npm, jolla voi ladata JavaScript-moduuleja mm. npmjs.com-sivustolta. (About Node.js, 2017.)

Node.js:n suosion syynä on mahdollisuus käyttää samaa ohjelmointikieltä sekä selain- että palvelinpään ohjelmoinnissa hälventäen rajaa em. ohjelmointikuntien välillä. Käytännössä se tarkoittaa sitä, että sama ohjelmoija voi työskennellä sekä selain- että palvelinkoodin kanssa. (Aguilar L. 2014.) Tämä yhdistettynä npm:n uudelleenkäytettäviin ja kaikkien saatavissa oleviin komponentteihin antaa kehittäjälle tehokkaat työkalut verkkosovellusten kehittämiseen.

Yleensä jokaisen node.js-sovelluksen mukana tulee tiedosto nimeltä package.json (kuva 2). Tiedosto sisältää metadataa paketista, kuten esimerkiksi sen nimen, lähteen, ohjeet sen käyttämiseen ja riippuvuudet.

```
{
  "name": "package-json-esimerkki",
  "version": "0.0.1",
  "description": "Esimerkki package.json-tiedostosta",
  "main": "server.js",
  "license": "MIT",
  "dependencies": {
    "express": "^4.16.2",
    "mongoose": "^4.13.6"
  }
}
```

Kuva 2 Esimerkki package.json-tiedostosta

Esimerkiksi ajamalla komennon ”npm install”, hakee npm automaattisesti asennettavan paketin riippuvuudet, tarkistaa tarvittaessa ympäristön (esim. vastaako node.js versio paketin vaatimaa versiota) ja ajaa mahdolliset paketin luojan asettamat skriptit hyödyntämällä package.jsonia (npm Documentation, 2017.)

3.2 Angular CLI

Jotta pystyisin suorittamaan uuden Wuudiksen kehitysversion paikallisesti minun tietokoneelleni, piti minun ensin asentaa Angular CLI. Se on Angular-sovelluksien kehittämiseen tarkoitettu komentoriviohjelma. Ohjelmalla voi luoda uuden Angular-projektin ja sen kehikon, generoida uusia komponentteja, sekä testata sovellusta paikallisesti kehittäessä.

Angular CLI:n asennus tapahtuu hakemalla se ensin npm-paketinhallinnasta ja asentamalla sen globaalisti. Käytännössä tämä tarkoittaa sitä, että Angular CLI toimii siis ikään kuin mikä muu tahansa komentoriviohjelma, joka on saatavilla kaikkialla. Se asennetaan seuraavalla komennolla: ”npm install -g @angular/cli”. Tämän jälkeen se on käytettävissä ng-komennolla (kuva 3). (Angular CLI, 2017.)

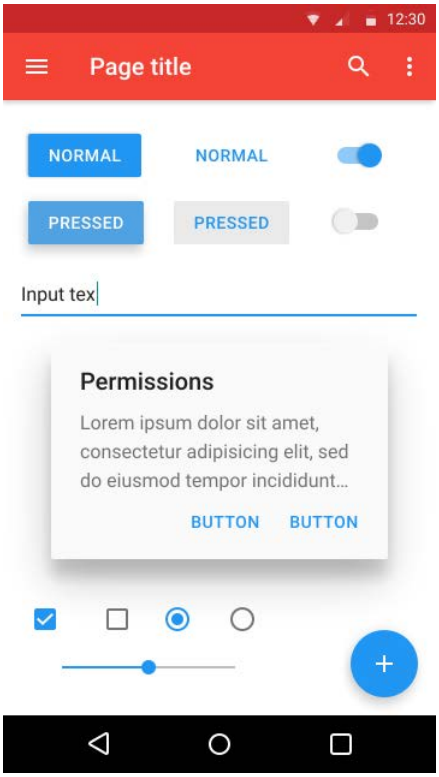
```
$ ng new esimerkki-angular-sovellus
Installed packages for tooling via npm.
Successfully initialized git.
Project 'esimerkki-angular-sovellus' successfully created.
$ cd esimerkki-angular-sovellus
$ ng serve
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2017-12-13T12:50:28.975Z
Hash: 300a5320416d86324c36
Time: 5675ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 20.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 553 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 34 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.14 MB [initial] [rendered]

webpack: Compiled successfully.
```

Kuva 3 Uuden projektin luonti ja käynnistys

Uusi Wuudis on kehitetty Angular-ohjelmistokehyksellä. Se on Googlen kehittämä ja ylläpitämä, avoimen lähdekoodin ohjelmistokehys jolla voidaan kehittää alustariippumattomia verkkosovelluksia. Se käyttää Microsoftin julkaisemaa avoimen lähdekoodin TypeScript-ohjelmointikieltä, joka perustuu JavaScriptiin. (Angular Docs, 2017).

3.3 Material Design ja sen käyttöönotto

Material Design on Googlen kehittämä muotokieli, joka julkaistiin vuoden 2014 Google I/O -konferenssissa. Muotokieli painottaa käyttöliittymän hierarkiaa, tarkoituksenmukaisuutta ja yhdenmukaisuutta. Käyttöliittymä koostuu ikään kuin ”virtuaalisesta paperista”, joista käyttöliittymä rakennetaan tasoittain (kuva 4). Material Design myös painottaa kaikkien elementtien ja animaatioiden tarkoituksenmukaisuutta. Näillä periaatteilla Google on rakentanut muotokielen, joka lainaa paljon oikeasta maailmasta, antaen työkalut selkeän käyttöliittymän luontiin. (Material Design Guidelines, 2017.)



Kuva 4 Esimerkki Material Designistä

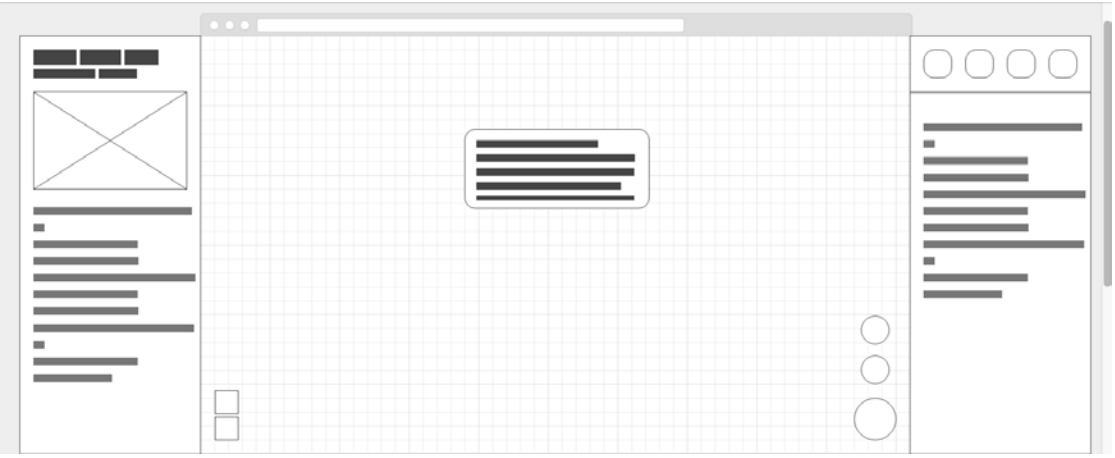
Angular Materialin käyttöönotto Angular-projektissa tapahtuu ensin asentamalla paketit *@angular/material* ja *@angular/cdk* NPM-paketinhallinnasta. Jotta kaikki Material-komponentit toimisivat kunnolla, on myös suositeltavaa asentaa *@angular/animations*-paketti. Nyt kun Angular Materialin kaikki riippuvuudet on täytetty, voidaan se ottaa käyttöön *NgModulissa*.
Wuudiksen tapauksessa Materialin käyttöönotto tapahtuu *shared*-moduulissa *app*-moduulin sijasta. *Shared*-moduulia käytetään yleensä projektin laajuisten komponenttien, direktiivien ja pipejen jakamisessa. (Angular Material Documentation, 2017.)

Jotta esimerkiksi Angular Materialin nappi-komponenttia voitaisiin käyttää projektissa, tulee se ensin ottaa käyttöön käyttämällä *import*-ominaisuutta *(import { MatButtonModule } from '@angular/material'*). Tämän jälkeen lisätään *MatButtonModule NgModulen imports*- ja *exports*-taulukkoon. Nyt nappia voidaan käyttää projektinlaajuisesti html-tiedostoissa lisäämällä button-elementtiin *mat-button*-direktiivi. (Angular Material Documentation, 2017.)

3.4 Karttakomponentin suunnittelu

Nyt kun kehitysympäristöni oli käyttökunnossa, olin valmis aloittamaan kehityksen, alkaen käyttöliittymän suunnittelusta. Päätin hyödyntää rautalankamallintamista. Se on tapa hahmotella käyttöliittymän ulkoasua ja elementtien sijoittelua nopeasti väliaikaisilla ”rautalankaelementeillä”. Rautalankamalli on kaksiulotteinen esitys sivun käyttöliittymästä, joka keskittyy tilan käytön hahmotteluun sekä sisällön priorisointiin (U.S Department of Health & Human Services, s.a.). Mallilla on helppo esittää oma idea käyttöliittymistä muille, sekä muokata sitä tarvittaessa. Rautalankamallit ovat myös hyvä tapa luoda ”kokoamisohjeet” käyttöliittymälle, joka helpottaa itse käyttöliittymän konkreettista luomista.

Käytin mallintamiseen wireframe.cc-nimistä selaimen kautta toimivaa ohjelmaa. Se on erittäin yksinkertainen rautalankamallinnusohjelma, josta mallit voi jakaa joko linkkinä tai ladata kuvina. Mallinsin kolme erilaista käyttöliittymää kartalle, joista sain kehitysideoita, ja päädyin lopulliseen rautalankamalliratkaisuun (kuva 5).



Kuva 5 Lopullinen rautalankamalli

Ideanani oli siirtää kaikki mahdollinen, ei aina tarvittavissa oleva tieto pois kartan päältä piilotettaviin sivupaneeleihin, jotka tuodaan näkyviin tarvittaessa. Kartalla olisi aina näkyvissä vain karttaohjaimet (suurennus ja pienennys), sekä kartan käytetyin funktio (esimerkiksi kartan keskittäminen metsätilan päälle, tai oikean sivupaneelin avaaminen) Material *Designin Fast Action Button* (FAB) -painike-elementin avulla.

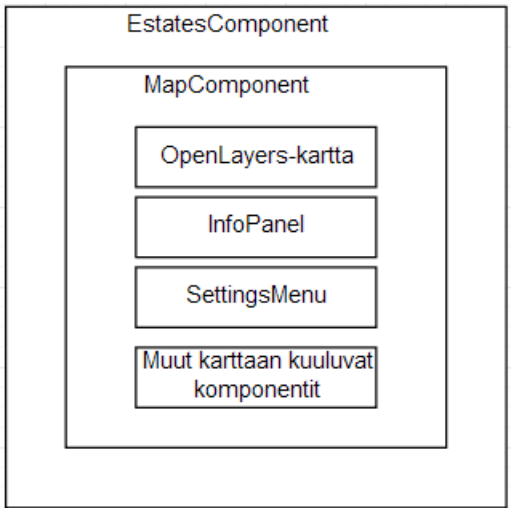
Klikkaamalla jotain karttamerkkiä kuten metsätilaa, avattaisiin ensin kartan päällä ”leijuva” nopea infoelementti, ja tarvittaessa liu’utetaan vasemmanpuoleinen sivupaneeli näkyviin. Tässä vasemmanpuoleisessa sivupaneelissa (info-panel) näkyisi avatun kohteen laajemmat tiedot, sekä mahdollinen kuva tai muu media. Elementti sisältäisi myös kaikkien metsätilojen listauksen sekä hakukentän. Oikeanpuoleisessa paneelissa olisi taas saatavilla kartan asetukset (settings-menu), johon kuuluvat mm. karttatason vaihto (esim. maastokartta tai satelliittikuva) ja karttamerkkien näkyvyys.

Yksi suuri muutos vanhan Wuudiksen metsätilanäkymään on kartan koko; se tulisi olemaan selaimen kokoinen täysleveä ja -korkea elementti. Ainoa elementti, joka ei kuulu karttaan tulisi olemaan yläreunassa sijaitseva navigaatiopalkki. Tällä saataisiin poistettua vanhan Wuudiksen kartan sivun vertikaalin navigoimisen tarve ja maksimoitaisiin tilankäyttö. Kartta hyödyntäisi siis aina kaiken tilan selaimesta. Toinen iso muutos oli idea yhdistää kaikkien metsätilojen selaus ja yksittäisen metsätilan katsominen yhteen näkymään. Vanhassa Wuudiksessa metsätilan avaaminen avasi uuden sivun, josta pääsi tarkastelemaan tiettyä yksittäistä metsätilaa.

3.5 Metsätilanäkymän luonti

Tällä hetkellä Wuudiksen kehitysversion kartta sijaitsi etusivulla omassa kortissaan. Ensitöikseni siirsin kartan omalle sivulleen. Koska Angular-sovellukset ovat yleensä SPA:ta eli ”yhden sivun applikaatioita” (Single page app), ei uuden sivun lisääminen ole yhtä yksinkertaista kuin uuden html-sivun luonti ja siihen linkittäminen. Sen sijaan Angular käyttää *router*-nimistä moduulia, joka vaihtaa *index*-sivun sisällä olevan *router-outlet*-elementin sisältöä. Tällä saadaan aikaan verkkosovellus tavallisen sivuston sijasta, joka mahdollistaa mm. sivujen vaihtamisen ilman niiden uudelleenlataamista ja komponenttien uudelleenkäytettävyyden. (Angular Docs, 2017.)

Generoin Angular CLI-komentoriviohjelman avulla ensin uuden komponentin *EstatesComponent*, joka suomennettuna tarkoittaa metsätilat-komponenttia. Tämä komponentti tulisi sisältämään karttakomponentin ja muut siihen liittyvät komponentit (kuva 6).



Kuva 6 Havainnekuva EstatesComponentin rakenteesta

Lisäsin routes-taulukkoon uuden reitin, jonka *path* eli polku oli *estates*, ja joka vei komponenttiin *EstatesComponent*. Lisäsin tähän objektiin muitten reittien mukaisesti *canActivate*-ominaisuuteen *AuthGuard*-luokan, joka evää pääsyn kirjautumattomilta käyttäjiltä ja ohjaa ne *login*-reitille (kuva 7).

```
const routes: Routes = [
    { path: '', redirectTo: '/home', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },
    { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
];
```

Kuva 7 Esimerkki routerin reiteistä

Lopputuloksena kartta oli nyt omalla reitillään *estates*-komponentissa, johon sisään kirjautunut käyttäjä pystyi navigoimaan laittamalla osoitteen perään ”/estates/”.

3.6 OpenLayers ja kartan luonti

OpenLayers on avoimen lähdekoodin JavaScript-kirjasto, joka mahdollistaa karttapalvelun toteuttamisen selainympäristössä. Se on lisensoitu BSD 2 -lauseke-lisenssille (BSD 2-clause license), joka antaa melkein rajoittamattoman vapauden OpenLayersin käyttämiseen, kunhan BSD-tekijänoikeushuomautus on sisällytetty tuotteeseen kokonaisuudessaan.

OpenLayers on erittäin monikäyttöinen ja joustava karttakirjasto, jolla on mahdollista toteuttaa hyvinkin kehittyneitä karttasovelluksia. Wuudis-palvelun monet ominaisuudet kuten esimerkiksi mahdollisuus käyttää monia eri lähteitä karttatasoina (esim. Maanmittauslaitoksen taustakartat tai Bing-karttojen satelliittikuvat) tai vektoridatan piirto kartalle (kuviorajojen piirtäminen) ovat hyviä esimerkkejä OpenLayersin monikäyttöisyydestä.

Uuden OpenLayers-kartan luonti on hyvin yksinkertaista käyttämällä new-avainsanaa JavaScriptissä (kuva 8).

```
var map = new ol.Map({
    layers: [
        new ol.layer.Tile({
            source: new ol.source.OSM()
        })
    ],
    target: 'map',
    controls: ol.control.defaults({
        attributionOptions: ({
            collapsible: false
        })
    }),
    view: new ol.View({
        center: [0, 0],
        zoom: 2
    })
});
```

8 OpenLayers-kartan luonti JavaScriptissä

Nyt kun kartta oli omalla sivullaan, pystyin aloittamaan kartan työstämisen. Jotta kartan ”päälle” luotavien elementtien kehitys olisi helpompaa, aloitin siirtämällä kartan mallin mukaisesti selainikkunan kokoiseksi. Annoin OpenLayers-kartalle tyyliluokan ”map”, ja lisäsin sen määrittelyn *map*-komponentin scss-tiedostoon. Tyylittely onnistui muuten hyvin; leveyden asettaminen sataan prosenttiin ja marginaalien poistaminen toimi niin kuin pitikin, mutta korkeuden kanssa oli ongelma: karttaa joutui vierittämään pystysuunnassa. Syynä tähän oli sovelluksessa aina näkyvissä olevat header- ja footer-elementit, jotka sisälsivät mm. navigaation.

Yritin etsiä tähän pitkään ratkaisua, joka löytyikin sitten opinnäytetyön toteutuksen loppuvaiheessa. CSS-tyylikieli sisältää *calc*-funktion, joka mahdollistaa

laskutoimitusten tekemisen CSS-tyylitiedostossa. Tämän olemassaolon tiedostamisen jälkeen ratkaisu oli yksinkertainen: kartan korkeus on selaimen korkeus, miinus *footerin* ja *headerin* korkeus.

3.7 Metsätilat ja listaus

Nyt kun kartta oli valmiina, aloitin karttojen elementtien kehittämisen, alkaen metsätilojen listauksesta. Päätin käyttää vasemmanpuoleisen paneelin, estate-listin toteutukseen Angular Materialin Sidenav -komponenttia. Se on selaimen ulkopuolelta liukuva kokopitkä elementti. Sidenav eli sivupaneeli sisältäisi metsätilojen listauksen lisäksi myös infopaneelin, joka näyttäisi tietoa valitusta OpenLayers-featuresta, kuten esimerkiksi metsätilasta.

Komponenttiajattelun mukaisesti päätin sijoittaa *sidenav*-komponentin sisällön omaan *estate-list* komponenttiinsa. Tässä kuitenkin törmäsin heti ongelmaan; *sidenav* ei toiminut niin kuin sen olisi pitänyt. Kartan päälle pitäisi ilmestyä modaali, jota ei kuitenkaan tapahtunut. *Sidenavin* sisältö ei myöskään ikinä kadonnut, vaan oli selvästi näkyvissä selaimen ulkopuolella, aiheuttaen vierityspalkkien ilmestymisen. Aloin tutkimaan ongelmaa, ja ongelma oli yleinen. Vika johtui siitä, kun *sidenavin* ”mat-drawer-container” -osa oli eri komponentissa kuin sen ”mat-drawer”-osa. Kun Materialin sidenav-komponentin osat eivät pystyneet kommunikoimaan keskenään kapseloinnin (eng. *view encapsulation*) takia, meni sen toiminnallisuus rikki. Vian olisi voinut todennäköisesti kiertää, mutta se olisi vaatinut paljon komponenttikoodin ja teemoituksen ylikirjoitusta. Myös päivitykset olisivat saattaneet rikkoa sen tulevaisuudessa.

Kapseloinnilla eli *view encapsulationilla* tarkoitetaan sitä, kuinka komponenttien CSS-tyylit eivät ”vuoda” muihin komponentteihin, vaan ovat eristetty toisistaan. Oletusarvoisesti Angular käyttää emuloitua kapselointia, joka eristää komponenteissa määritellyt tyylit, mutta sallii tyylien käytön pää-HTML:stä. Muita vaihtoehtoja ovat natiivi kapselointi, joka ei salli tyylien käyttöä muualta, sekä sen pois käytöstä ottaminen, joka sallii komponenttien välisen tyylien jakamisen. Tätä ei kuitenkaan suositella, sillä jo keskisuurissa projekteissa tyylien ylläpidosta tulee hankalaa.

Etsiessäni vaihtoehtoja sivupaneelille tajusin, että se voisi olla käyttäjää turhauttava elementti tässä tapauksessa. Jos käyttäjä haluaisi esimerkiksi selata metsätiloja läpi valitsemalla niitä kartasta, aukeaisi joka ”klikkauksella” sivupaneeli. Tämä tarkoittaisi sitä, että kun käyttäjä haluaa valita toisen metsätilan, tulisi hänen ensin sulkea sivupaneeli klikkaamalla karttaa, tämän jälkeen odottaa sulkemisanimaatio loppuun ja klikata jotain metsätilaa uudelleen. Tässä käyttäjälle tulisi yksi ylimääräinen klikkaus per metsätila, sekä kaksi animaatiota, jotka lukittavat käyttöliittymän hetkeksi.

Hylkäsin koko ajatuksen sivupaneelin käytöstä ja aloin miettimään muita vaihtoehtoja. Päädyin käyttämään Material Designin ehkäpä tunnetuinta elementtiä, korttia (card). Material Designin virallinen dokumentaatio kuvailee kortin olevan arkki ”materiaa”, joka toimii ikään kuin lähtökohtana yksityiskohtaisempaan tietoon (Material Design guidelines, 2017). Tämä toimii hyvin infopaneelin käyttötarkoituksessa: nopeasti silmäiltävää tietoa tiivistettynä. Metsätilojen listausten eli *estate-listin* tapauksessa kortti ei ole välttämättä se elementti, mitä Google suosittelee. Karttakäyttöliittymällä on kuitenkin hieman erilaiset tarpeet ja vaatimukset, joten Material Designin seuraaminen täydellisesti tässä tapauksessa tulisi vain tielle.

Estate-list-komponentti tulisi olemaan kaksiosainen; hakukenttä ja metsätilalistaus. Hakukenttä tulisi olemaan aina näkyvissä, ja jonka aktivoimalla näytettäisiin haetut metsätilat (ilman hakutermiä näytettäisiin kaikki metsätilat). Haetut metsätilat näytettäisiin kortin sisällä, ja yksi metsätila näytettäisiin *sheet*-elementtinä. *Sheet* eli arkkielementti on yleensä listoissa käytettävä, kortti-elementin tapainen kokoleveä elementti (Material Design guidelines, 2017). Yhdessä listaelementissä näytettäisiin esimerkiksi metsätilan nimi, kiinteistötunnus ja omistaja. Klikkaamalla sitä keskistettäisiin kartta metsätilaan, ja avattaisiin *info-panel*, jossa näytettäisiin metsätilan tarkemmat tiedot.

Generoin uuden komponentin käyttämällä Angular CLI:tä ja lisäsin sen karttakomponenttiin tavallisen HTML-tagin tavoin. Nyt karttakomponentin HTML-tiedostossa oli juuri luotu *map-estate-list*-tagi, sekä *div*-tagi johon kartta piirretään. Jotta komponentti ei jäisi kartan alle, muokkasin sen *z-index*-tyyliarvoa korkeammaksi*. Z-index*-tyyliarvolla voidaan määritellä html-elementtien taso;

mitä korkeampi arvo, sitä ”korkeammalla” se on. Tämän jälkeen lisäsin komponenttiin kaksi Angular Materialin *mat-card*-komponenttia, joista ensimmäiseen lisäsin input-kentän hakusanalle, toiseen listaelementin, johon kaikki metsätilat listattaisiin.

Metsätilat haetaan komponenttikoodissa käyttämällä *RxJS* -JavaScript-kirjaston *subscribe*-metodia (kuva 9). RxJs-kirjasto mahdollistaa reaktiivisen ohjelmoinnin, helpottaen asynkronisen koodin luomista. Se mahdollistaa *observable-observer*-tyylisen ohjelmoinnin. (RxJS API Document. 2017.)

```
this.subscribers.estatesChange = this.estateService.whenEstatesChanged().subscribe(newEstates => {
    this.estateList = <Array<Estate>> newEstates
})
```

Kuva 9 Subscribellä datan hakeminen

Tilaamalla *whenEstatesChanged()*-metodin palauttaman *observablen*, *estate-list*-komponentilla on käytössään aina tuorein haettu data. Angularin *ngFor*-direktiivillä on helppo tulostaa metsätilat käyttöliittymään. Antamalla ngFor-direktiiville arvon ”let estate of estateList” ja lisäämällä sen johonkin elementtiin, tulostetaan elementti for-lauseen tapaisesti, jossa yksittäisen metsätilan tiedot ovat käytössä ”estate”-objektissa.

Koska Material Angularin list-komponentti ei vastannut ulkoasullisesti tai toiminnallisesti sitä mitä halusin, loin listaelementin itse. Lisäämällä Angular Materialin button-elementille mukautetun tyylin, sain luotua siitä sheet-elementin tapaisen komponentin. Käyttämällä *ngFor*-direktiiviä tähän elementtiin, sain metsätilat listattua.

Lisäsin metsätilaelementtiin tapahtumakäsittelijän, joka kuuntelee ”click”-tapahtumaa, joka suorittaa komponenttikoodissa olevan metodin *previewEstate*. Tämä taas kutsuu *map-servicessä* olevaa saman nimistä metodia (kuva 10). Tällä metodilla on kaksi tehtävää; ensiksi se keskittää kartan valittuun metsätilaan, jonka jälkeen se ”simuloi” metsätilan klikkauksen. Tämä tehdään ensin hakemalla metsätilan *feature*, jonka jälkeen se lisätään valituiden *featureiden* taulukkoon käyttämällä JavaScriptin push-metodia. Sen jälkeen asetetaan metsätila *map-servicen selectFeatureEventin* subjektiksi.

```
public previewEstate(estate: Estate) {
    /* Center map on estate */
    let vectorSource = this.vectorSourceService.getEstateVectorSource()
    let estateFeature = this.getFeatureForItem(estate, vectorSource)
    this.centerViewOnSelectedFeature(estateFeature)

    /* Do selection events */
    this.estateInteractionSelect.getFeatures().push(estateFeature)
    this.setSelectEventFeatureItem(estate)
}
```

Kuva 10 Map-servicen previewEstate-metodi

OpenLayersin *feature* on vektoriobjekti geograafisille elementeille. Yksi feature sisältää yleensä yhden karttaobjektin, tässä tapauksessa metsätilan geometriset tiedot. Se voi myös sisältää esimerkiksi sen tyylitykset tai sen nimen. (OpenLayers Documentation, 2018)

Angularissa *servicen* tehtävä on tarjota muille komponenteille uudelleenkäytettäviä, koko sovelluksenlaajuisia ohjelmakoodikirjastoja. Esimerkiksi kaikki API-kutsut on hyvä paketoida *serviceksi*. Yleisiä muita käyttökohteita niille ovat esimerkiksi käyttäjän todentaminen tai komponenttien välinen kommunikointi. (Lavin J 2014, 26.)

Nyt kun käyttäjä avaa metsätilan listauksen kautta, käyttäytyy se samalla tavalla kuin karttaa klikkaamalla. Seuraavaksi aloin tekemään metsätilan hakukenttää ja siinä käytettävää Angularin *pipeä*.

3.8 Geneerinen hakutoiminnallisuus

AngularJs:ä oli sisäänrakennettu filter, jolla pystyi suodattamaan taulukon halutuiden arvojen mukaisesti. Tämä kuitenkin poistettiin käytöstä Angular 2 -julkaisussa, sillä Angularin kehittäjätiimi koki, että suodatinta väärinkäytettiin jatkuvasti, joka johti AngularJs:än hitauteen. Kuitenkin juuri tällaista toiminnallisuutta olisin tarvinnut metsätilojen suodatukseen, joten päätin toteuttaa samanlaisen suodattimen itse käyttämällä Angularin pipe -ominaisuutta.

Pipe yksinkertaistettuna ottaa sisääntulona dataa ja palauttaa sen halutussa muodossa. Yleisin esimerkki pipen käytöstä on päivämäärät; date-pipe muuntaa päivämäärä-objektit ihmisluettavaan muotoon (kuva 11).

```
<p matLine class="info-value">{{observation.created | date:'dd.MM.yyyy'}}</p>
```

Kuva 11 Esimerkki date-pipen käytöstä

Tarvittavan geneerisen suodattimen tulisi toimia samalla tavalla: sisääntulona suodatettava lista, hakusana ja hakukohteet, joka palauttaa suodatetun listan. Aloitin generoimalla uuden pipen käyttämällä angular-cli-komentoriviohjelmaa komennolla ”ng g pipe list-search”.

@Pipe-dekoraattorissa oleva *name*-ominaisuus määrittää sen, millä nimellä direktiiviä kutsutaan itse sovelluksessa. Pipeä voisi käyttää seuraavanlaisesti: ”estateList | listSearchPipe:argumentit”. Tämä syöttää pipen sisällä olevan *transform*-metodin parametreina esimerkki-datan, ja mahdolliset argumentit. *Transform*-metodissa tehdään esimerkki-datalle tarvittava muunnos, ja palautetaan sen *returnilla*.

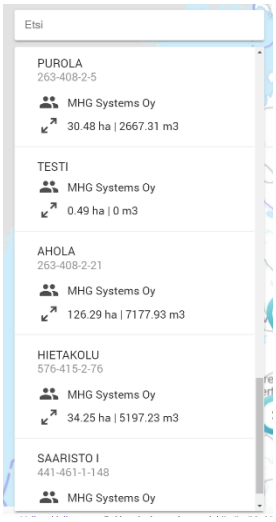
Muutin *transform*-metodin siten, että se ottaa kolme parametria: *listItems*, joka sisältää suodatettavat kohteet, *searchString*, joka sisältää käyttäjän antaman hakusanan, sekä *searchTargets*, jossa haettavat kohteet ovat eroteltu pilkulla. Seuraavaksi otin käyttöön JavaScriptin *filter*-metodin. *Filter* ketjutetaan pilkulla suodatettavaan taulukkoon, ja sille annetaan parametreina ehtolause. Jos ehtolause palautuu tosi-arvolla, lisätään taulukon nykyinen arvo tulokset-taulukkoon. Tämä ehto käydään läpi kaikille taulukon arvoille, *forEach*-metodin tapaisesti.

List-search-pipen tapauksessa ehtolause olisi seuraavanlainen: jos käyttäjän antama hakusana (searchString) vastaa joidenkin suodatettavien tulosten (listItems) hakukohteiden (searchTargets) arvoista, on ehto tosi. Aloitin tekemällä searchTargets-arvosta taulukon käyttämällä split-metodia. Tämän jälkeen käyttämällä for-lausetta tarkastellaan, jos jokin hakukohteiden arvoista vastaa annettua hakusanaa käyttämällä includes-metodia. *Includes* on EcmaScript 6 mukana tuoma metodi, jolla voidaan tarkastella, jos taulukko sisältää parametrissa annetun arvon. Metodi palauttaa joko totuusarvomuuttujan tosi tai epätosi, riippuen löytyikö arvo taulukosta. Tämän arvon mukaisesti filter joko lisää tai on lisäämättä läpikäytävää arvoa tulostaulukkoon.

Otin pipen käyttöön estate-list komponentin hakukentässä. Annoin suodatettavaksi listaksi kaikki metsätilat, ja hakukohteiksi metsätilan arvot ”name” ja ”propertyIdentifier” eli kiinteistötunnus. Huomasin kuitenkin sen käyttäytymisessä jotain outoa: haettu metsätila näytetään ainoastaan, jos hakusana ilmenee sekä nimessä että kiinteistötunnuksessa. Tämä ilmeni siinä, kuinka ainoa metsätila, jota pystyi hakemaan, oli testimetsätila ”Ahola 3”, jonka kiinteistötunnus oli ”333-333…” hakusanalla ”3”. Koska 3 esiintyi sekä sen nimessä, että sen kiinteistötunnuksessa, vastasi se list-search-pipen ehtoja.

Korjasin asian siten, että lisäsin filterin sisälle taulukon ”results”. Tämän jälkeen muutin includes-kohtaa siten, että se lisää results-taulukkoon totuusarvomuuttujan. Lopuksi tarkastelin, jos results-taulukko sisältää tosi-arvon käyttämällä includes-metodia, ja palautin sen tuloksen. Nyt *pipe* toimi niin kuin sen pitikin, ja sen käyttöönotto *estate-list*-komponentissa oli valmis (kuva 12).

```
export class ListSearchPipe implements PipeTransform {
    transform(listItems: any[], searchString: string, searchTargets: any): any {
        if (!listItems) return []
        if (!searchString) return listItems

        searchTargets = searchTargets.split(',')
        searchString = searchString.toLowerCase()

        return listItems.filter(filteredItems => {
            let results: any[] = []
            for(let i = 0; i < searchTargets.length; i++) {
                let searchTarget = searchTargets[i]
                if(filteredItems[searchTarget] != null) {
                    results.push(filteredItems[searchTarget].toString().toLowerCase().includes(searchString))
                }
            }
            // If either of the includes is true, return true.
            if(results.includes(true)) return true
            else return false
        })
    }

}
```



Kuva 12 Lopullinen list-search-pipe ja estate-list-komponentti

Nyt metsätilat olivat listattu nopeassa ja helposti silmäiltävässä muodossa. Niiden hakeminen oli myös helppoa ja nopeaa, sillä kaikki tämä tehdään selainkoodissa, joten tiedon lähettäminen ja vastaanottaminen palvelinpuolelta oli tarpeetonta.

3.9 Valitun kohteen tiedon näyttö

Wuudiksen kartta sisältää monia erilaisia asioita, joilla kaikilla on erilaisia attribuutteja; metsätilalla on esimerkiksi sen puuston arvo, kuviosta taas halutaan tietää sen saavutettavuus ja havainnoista halutaan nähdä kuvia. Kuitenkin ne

tulisi näyttää samalla tavalla ja niiden pitäisi käyttäytyä samanlaisesti, jotta käyttäjän ei tarvitse opetella kolmea eri tapaa käydä läpi informaatiota.

Alkuperäisessä rautalankamallissa tämä ei ollut niin suuri ongelma; sivupaneeli toimii vain alustana muille komponenteille. Se voi olla niin leveä kuin on tarve, sillä se piilotetaan kuitenkin aina näytön ”ulkopuolelle”. Tällä tapaa tiedon näyttö olisi paljon vapaampaa ja esim. seliteteksteille olisi enemmän tilaa. Infopaneelissa on kuitenkin vähemmän tilaa käytössä, ja sen tulisi vastata em. ongelmaan.

Infopaneelin tulisi olla samankaltainen kuin estate-list-komponentti; ikään kuin sen ”sisarkomponentti”. Molemmat seuraisivat samanlaista tyyliä ja veisivät saman tilan karttakäyttöliittymästä. Ne eivät kuitenkaan voisi esiintyä samanaikaisesti, jottei käyttöliittymä tulisi liian vilkkaaksi. Tämän infopaneelin tehtävänä olisi näyttää nopeaa, silmäiltävää tietoa. Toisin sanoen, sen ei tarvitse sisältää mahdollisuutta muokata tietoa tai näyttää yksityiskohtia valitusta asiasta.

Generoin uuden komponentin käyttämällä Angular CLI -komentoriviohjelmaa kuten aikaisemminkin, ja lisäsin siihen samankaltaisen rakenteen ja tyylin kuin estate-list-komponentissa. Käyttämällä RxJS-kirjaston subscribe-metodia, tilasin *selectFeatureEvent*-subjektin map-servicestä. Tilaus päivittyy aina, kun käyttäjä on valinnut jonkin *featuren* kartasta. Käyttämällä JavaScriptin *instanceof*-operaattoria, tarkastelin jos valittu *feature* oli haluttua tyyppiä. Tämän jälkeen haettiin valitun *featuren* lisätiedot; esimerkkinä metsätilalle sen arvo tai havainnolle sen kuvat.

Seuraavaksi aloin suunnittelemaan infopaneelin ulkoasua, alkaen metsätilan tietojen näytöstä. Päätin toteuttaa Google Maps -karttapalvelun kaltaisen avainarvoparilistauksen (kuva 13), jossa avainarvo on ikoni selitetekstin sijasta. Ikonien hyöty on siinä, että tieto on nopeasti silmäiltävää ja ne säästävät paljon tilaa verrattuna selitetekstiin.

Kuva 13 Google Mapsin avainarvoparilistaus Mikkelin kampuksesta

Sopivia listauksessa käytettäviä ikoneita etsiessäni huomasin, kuinka Angular Materialin oletuksena käytettävä Googlen Material Icons -kirjaston ikonivalikoima oli suppea. Se sisältää vain yleisimmät verkkosovelluksissa käytettävät ikonit, joten metsään liittyviä ikoneita ei juuri löytynyt. Angular Material tukee ikonikirjaston vaihtoa, joten vaihdoin sen *materialdesignicons.com*-ikonikirjastoon. Se sisältää sekä Googlen Material Designin ikonit, että käyttäjien itsetehdyt ikonit, jotka seuraavat ulkoasultaan alkuperäisiä Googlen ikoneja. Materialdesignicons.com-ikonikirjastoon sisältääkin yli tuplasti ikoneita verrattuna Googlen omaan ikonikirjastoon.

Latasin materialdesignicons.com-kirjaston *assets*-kansioon *svg*-muodossa. Tämän jälkeen käytin *import*-lausetta *shared*-moduulissa tuodakseni *MatIconRegistry*- sekä *DomSanitizer*-moduulin käytettäväksi. *DomSanitizerin bypassSecurityTrustResourceUrl*-metodi ohittaa Angularin XSS-hyökkäyksen estäjän merkkaamalla resurssin luotettavaksi. Nyt käyttämällä *MatIconRegistryn addSvgIconSet*-metodia, pystyin lisäämään uudet ikonit projektissa käytettäväksi (kuva 14). (Angular Docs, 2018.)

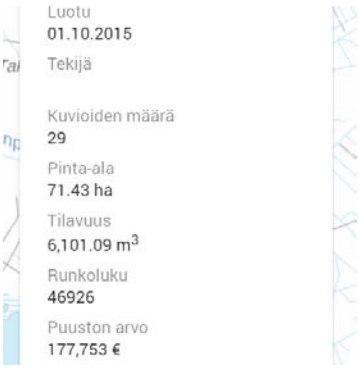
```
export class SharedModule {
    constructor(matIconRegistry: MatIconRegistry, domSanitizer: DomSanitizer) {
        // Add materialdesignicons.com-icons to iconRegistry and mark it as trusted source
        matIconRegistry.addSvgIconSet(domSanitizer.bypassSecurityTrustResourceUrl('./assets/mdi.svg'))
    }
}
```

Kuva 14 Uuden ikonikirjaston käyttönotto

Nyt kun laajennettu ikonivalikoima oli käytössä, jatkoin infopaneelin metsätilaosion toteuttamista. Infopaneelin rakenne olisi kaksiosainen; otsikko-osa, jossa olisi valitun kohteen otsikko ja jokin hyvin olennainen tieto, sekä tieto-osa, jossa näytettäisiin ”nopea tieto” valitusta kohteesta. Tieto-osan rakennetta vaihdettaisiin sen mukaan, mitä tietoa haluttaisiin näyttää.

Listasin metsätilan näytössä mm. sen luontipäivän ja luojan, johon normaalit Material Designin ikonit riittivät hyvin. Metsätilan alueen tai puuston arvon näyttöön tulivat taas materialdesignicons.com-kirjaston ikonit hyödyksi. Sain kuitenkin viikkopalaverissa tästä tyylistä palautetta; muut eivät olleet varmoja ja joutuivat arvaamaan, mikä oli minkäkin ikonin tarkoitus. Olin aluksi hieman ymmälläni tästä; miten Google Maps voi käyttää tätä tyyliä ilman, että käyttäjät joutuisivat arvailemaan ikonien tarkoitusta. Syy oli kuitenkin yksinkertainen: Wuudiksen metsätilan tieto on paljon enemmän erikoistunutta tietoa, joten käyttäjät eivät ole oppineet ikonien merkitystä esimerkiksi muista verkkopalveluista. Toinen syy tähän oli se, että puuta kuvaava ikoni voi tarkoittaa esimerkiksi puuston määrää, sen laatua, arvoa, tyyppiä jne. Ikonit ovat liian ”laajoja” kuvaamaan yksityiskohtaista tietoa tässä tapauksessa.

Todetessani, ettei ikonien käyttö ole tässä tilanteessa millään tavalla käytännöllistä, muutin kaikki selitteet tekstiksi. Vaikka tekstiselitteet eivät välttämättä ole yhtä hienoja kuin ikonit, ovat ne kuitenkin paljon selkeämpiä ja käytännöllisiä (kuva 15).



Kuva 15 Tekstiselitteet infopaneelissa

Myöhemmin tutkiessani tätä ongelmaa, yksi MHG:n työntekijöistä mainitsi minulle käsitteen ”mystery meat navigation”. Tämän termin oli kehittänyt Vincent Flanders vuonna 1998. Sillä viitataan nappeihin tai linkkeihin, joiden tarkoituksen käyttäjä itse joutuu selvittämään. (Siang, T. 2017) Vaikka ”mystery meat navigation” -termillä viitataan nappeihin ja linkkeihin, voidaan sitä soveltaa tähän ongelmaan: käyttäjä joutuu arvaamaan ja selvittämään mikä on minkäkin ikonin tarkoitus.

3.10 Metsätilan operaatiot ja havainnot

Sain viikkokokouksessa kehitysehdotuksen liittyen metsätilojen selailuun: infopaneelista tulisi nähdä valitun metsätilan tai kuvion operaatiot ja havainnot. Wuudiksessa on ominaisuus, jolla käyttäjä voi lisätä jonkinlaisen operaation metsätilalle, ikään kuin tehtävälistalle. Operaatio voi olla vaikkapa vuoden päästä tehtävä hakkuu. Tällä ominaisuudella voidaan luoda eräänlainen ”aikajana” metsänhoitotehtävistä. Havainnot ovat Wuudiksen mobiilisovelluksella tehtäviä merkintöjä, joihin voidaan liittää esimerkiksi kuva ja sijainti. Havannoilla käyttäjä voi merkata itselleen ylös vaikkapa myrskyssä kaatuneen puun. Metsätiloja selatessa nämä tiedot olisi hyvä saada helposti esille.
Päätin lisätä infopaneelin sisältöosaan välilehtituen. Niitä tulisi olemaan kolme, joissa tulisi olemaan tämän hetkinen infopaneelin tieto-osa, operaatiot ja havainnot. Välilehti eli *tab* on Material Designin komponentti, joka mahdollistaa sisällönhallinnan helposti (Material Design guidelines, 2017). Operaatioista ja havainnoista listattaisiin esimerkiksi viisi oleellisinta nimikettä. Uuden Wuudiksen käyttöliittymäkehittäjä suositteli toteuttamaan listauksen tekemällä geneerisen listakomponentin. Listakomponentti ottaisi sisääntulona *(input*) listattavat asiat, listan rajoittavan lukumäärän ja listausjärjestyksen.

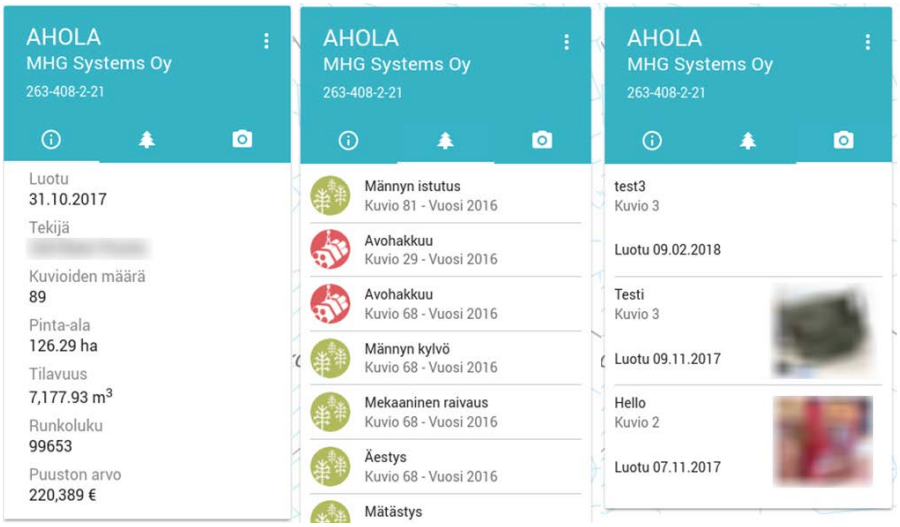
Generoin uuden komponentin ja lisäsin sille aluksi tarvittavat *input*-ominaisuudet. Angularissa *input* on yksi tapa kommunikoida komponenttien välillä. Listakomponentin tapauksessa sillä on kolme *input*-direktiiviä, joiden arvot määritellään sen isäntäkomponentista eli infopaneelista käsin. Listakomponentissa näihin arvoihin saadaan ote käyttämällä @input-ominaisuutta.

Loin uuden mallin eli *modelin*, jonka avulla listaukselle saataisiin tarvittava data oikeassa muodossa. Tämän jälkeen lista lajitellaan käyttämällä JavaScriptin *sort*-metodia listausjärjestyksen mukaisesti joko uusimmasta vanhimpaan tai päinvastoin.

Nyt listaus on käytössä komponentinlaajuisesti muuttujassa, ja se voidaan piirtää. Listaukseen käytin Angular Material 2 -komponenttikirjaston *mat-grid-list*-komponenttia. *Grid-list* on kaksiulotteinen listanäkymä, jolla voidaan järjestellä

taulukkosoluja ruudukkopohjaiseen sommitteluun (Angular Material Documentation, 2017). Lisäämällä ngFor-direktiivin *grid-listiin* tulostin listauksen komponentin pintaan.

Lisäsin infopaneelin metsätilan latausmetodiin kaksi uutta metodikutsua listattaville asioille. Operaatiot ja havainnot haetaan omista *serviceistään* aina kun metsätila tai kuvio on valittu. Tämän jälkeen listat muunnetaan vastaamaan aikaisemmin luomaani mallia ja syötetään listakomponenteille käyttämällä input-direktiiviä. Nyt infopaneeli näyttää metsätilasta sen perustiedot, operaatiot sekä havainnot helposti silmäiltävässä muodossa (kuva 16).



Kuva 16 Valmis infopaneeli metsätilaa esikatsellessa

Infopaneelissa olevat listaukset ovat myös uudelleenkäytettäviä, joten infopaneelin jatkokehitys esimerkiksi listauksen osalta pitäisi olla helppoa ja selkeää.

3.11 Yksittäisen metsätilan näkymä

Tämän hetkisessä Wuudiksessa kartta on jaettu kahteen eri näkymään: metsätilojen selaus ja yksittäisen metsätilan näkymä. Nämä näkymät olivat kahdella eri reitillä, joten metsätilan avaaminen ja sulkeminen aiheutti jatkuvaa kartan uudelleenlataamista. Tämä taas nostaa palvelinkustannuksia ja käyttäjä joutuu odottamaan sivun lataamista.
Yksi Wuudiksen työntekijöistä ehdotti ratkaisua, jossa kaikki metsätiloihin liittyvä olisi samalla kartalla. Metsätilojen selaus sekä yksittäisen metsätilan näkymä tapahtuisi siis samalla sivulla, ilman uudelleenlatauksia. Tämä ratkaisu

on monelta kannalta edeltäjäänsä parempi: kartan käytös on ”sovellusmaisempi”, seuraten ”single page app” -sovellusrakennetta. Myös samojen komponenttien, kuten infopaneelin hyödyntäminen on helpompaa.

Tämä ratkaisu on kuitenkin monimutkaisempi toteuttaa kuin nykyisessä Wuudiksessa; metsätilaa avatessa kartta pitäisi aina pyyhkiä, siihen pitäisi ladata uudet karttamerkit, ja metsätilasta poistuttaessa kartta pitäisi palauttaa alkuperäiseen muotoonsa.

Metsätilan avaamiseen suunnittelin seuraavanlaista tapahtumasarjaa: kun käyttäjä avaa metsätilan, keskitetään kartta, poistetaan nykyiset karttamerkit ja käyttöliittymäelementit, mitä yksittäisessä metsätilanäkymässä ei tarvita. Tämän jälkeen aloitetaan latausprosessi, ja näytetään ladatut karttamerkinnät sitä mukaan, milloin lataukset valmistuvat.

Aloitin tekemällä stand-serviceen uuden metodin *getLocalEstateStandList*, joka hakee valitulle metsätilalle sitä vastaavat kuviot. Kuvio on osa metsätilaa, ikään kuin palapelin palanen, josta kokonainen metsätila koostuu. Se ensin tarkistaa, ovatko valitun metsätilan kuviot saatavilla selaimen sessiomuistissa, ja jos ei, niin hakee sen Wuudiksen palvelimelta. Kuviolistaus käsitellään map-servicessä, jossa jokaisen kuvion geometria muunnetaan polygoniksi ja lisätään uuteen vektorilähteeseen. Tätä vektorilähdettä käytetään uuden karttatason luonnissa.

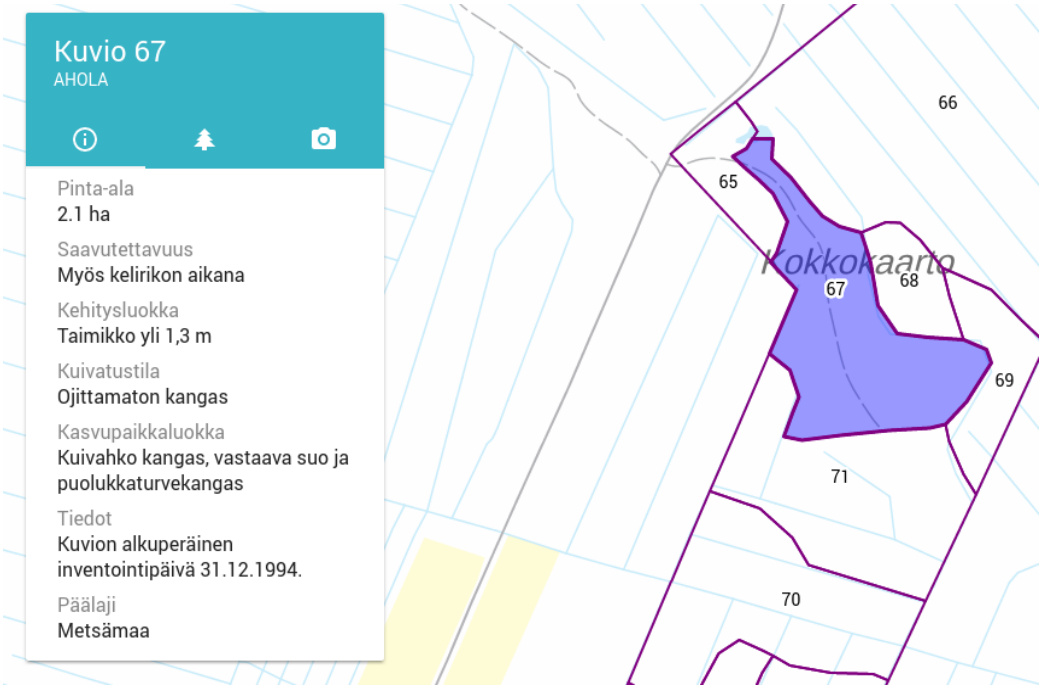
Loin estates-komponenttiin uuden metodin *openEstate*, joka aloittaa metsätilan avausprosessin. Tässä metodissa tehdään itse avausprosessi, joka pyyhkii koko kartan tyhjäksi, lataa uudet karttamerkinnät ja lisää niille tapahtumakäsittelijät. Loin myös sille vastakappaleeksi metodin *closeEstate,* joka pyyhkii kartan ja suorittaa vastaavat toiminnot metsätilat-näkymälle. Tilaamalla *map-serviceen* lisäämäni *activeEstateChanged-observablen*, suoritetaan *openEstate*-metodi.

Nyt kun tarvittava koodi metsätilojen välillä liikkumiseen oli valmis, aloin keskittyä sen käyttöliittymän luontiin. Törmäsin kuitenkin heti tässä ongelmaan: metsätilan avaamiselle ei tuntunut olevan luontevaa paikkaa, joka kuitenkin seuraisi jollain tasolla joko Material Designin ohjeita tai toimisi kartassa hyvin.

Vaihtoehtoina oli upottaa jonkinlainen avauspainike infopaneeliin, lisätä karttaan tapa avata metsätila tai korvata hakupalkki jonkinlaisella avausmekanismilla. Paras ratkaisu mielestäni olisi yhdistää nämä kaksi tapaa: metsätilan voisi avata sekä kartasta että infopaneelista. Tällä tavoin käyttäjällä olisi aina jonkinlainen luonnollinen tapa avata metsätila: listauksesta avaaminen tapahtuisi luonnollisesti infopaneelista klikkaamalla, kun taas kartasta metsätilaa valitessa sen avaus olisi myös luonnollisesti kartalla. Avaaminen olisi siis aina lähellä käyttäjän hiirtä, riippumatta valintareitistä. Mekanismin sijoittaminen kartalle osoittautui kuitenkin oletettua hankalemmaksi ja tässä vaiheessa opinnäytetyötä en nähnyt järkeväksi tehdä hakupalkkia uudestaan, joten päätin sijoittaa avauksen väliaikaisesti infopaneelin valikkoon.

Infopaneelista metsätilan avauksen valitseminen kutsuu metodia, joka muuttaa *map-servicessä* olevan, aikaisemmin mainitun *activeEstateChanged-observablen* aiheeksi auki olevan metsätilan. Tämä laukaisee *estates*-komponentissa olevan *openEstate*-metodin, joka avaa metsätilan.

Lisäsin infopaneeliin tarvittavat käsittelyt ja ulkoasut yhden kuvion tietojen, operaatioiden ja havaintojen näyttämiseksi. Metsätilan ja kuvion tietojen rakenne on samankaltainen, joten tämä oli helppo toteuttaa. Nyt valitsemalla kuvion, näytetään sen tiedot samantapaisesti kuin metsätilasta (kuva 17).



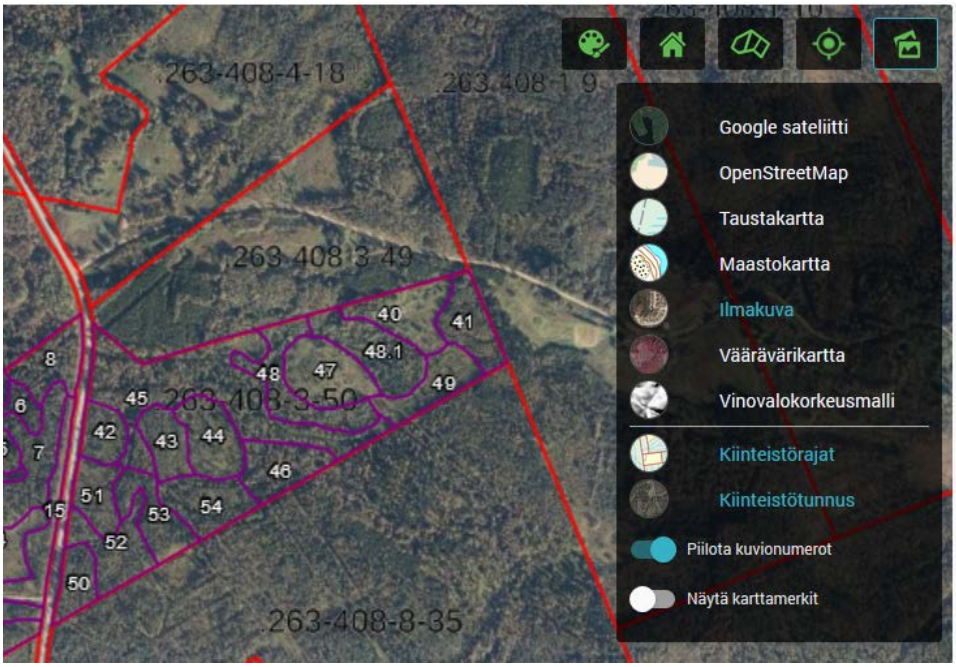
Kuva 17 Kuvion valinta metsätilanäkymässä

Infopaneeli toimii nopeasti ilman latausaikoja, joka mahdollistaa nopeasti kuvioiden läpikäymisen. Tieto näytetään myös helposti silmäiltävässä muodossa, joten käyttäjän on helppo saada kuvioista haluamansa tieto helposti esille.

3.12 Kartta-asetusten valikko

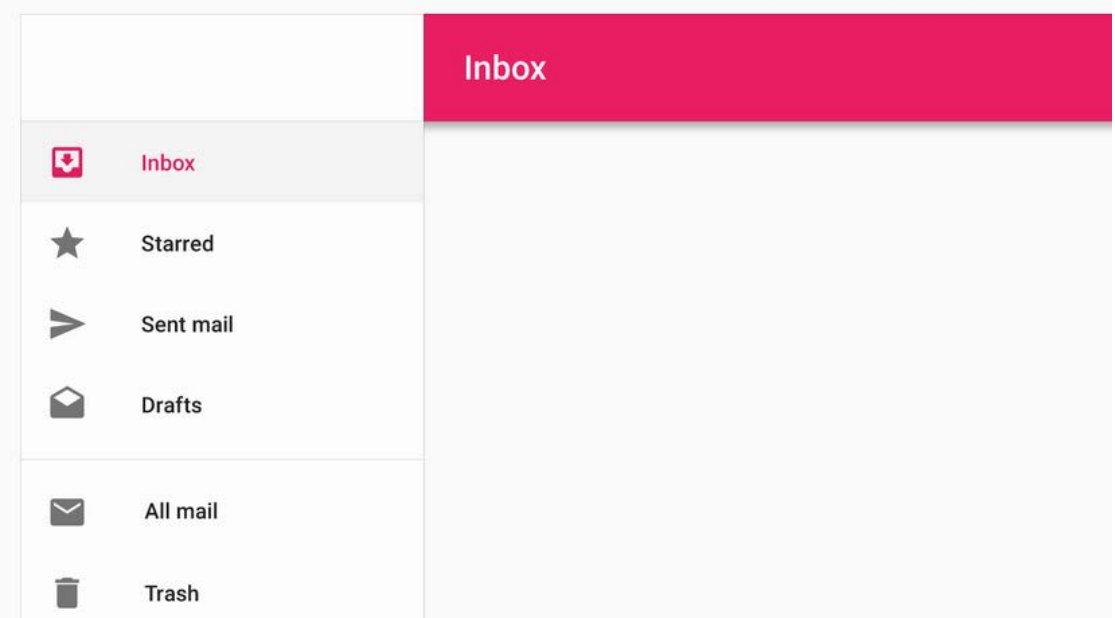
Yksi tärkeimmistä Wuudiksen karttatyökaluista on karttatasojen vaihtaminen. Nopeasti karttatasojen välillä vaihtaminen on olennainen osa palvelun käyttöä, joten sen vaihtaminen pitäisi olla nopeaa ja selkeää.

Nykyisessä Wuudiksessa karttatyökalut on sijoitettu nappien taakse, ja ne ovat aina näkyvissä kartan oikeassa yläkulmassa. Uutena käyttäjänä ratkaisu tuntui minusta hieman sekavalta. Nappuloiden ikonit eivät juuri kertoneet mitä niiden alta löytyy ja sen alla saattoi olla joko jokin toiminto tai valikko (kuva 18).



Kuva 18 Nykyisen Wuudiksen karttatyökalujen hallinta

Uuden Wuudiksen koko ikkunan tilan vievä kartta mahdollistaa sivupaneelien käytön. Sivupaneeli on yksi Material Designin ominaisista elementeistä. Se on käytännössä selaimen ”ulkopuolella” piilossa oleva valikko, joka kutsuttaessa animoidaan liukumaan näytön sivulle (kuva 19) (Material Design guidelines, 2017). Tämä ratkaisu voittaa valikkonappulat mahdollisessa tilankäytössä. Sen sijaan että valinnat oltaisiin hajotettu monien eri nappuloiden alle, voidaan ne sijoittaa yhden valikon alle.



Kuva 19 Material.io-sivustolla oleva esimerkki sivupaneelista

Ennen kuin aloin suunnittelemaan tarkemmin sivuvalikon ulkoasua, päätin tutkia, mikä olisi hyvä tapa toteuttaa se. Ensimmäiseksi kokeilin toteuttaa valikon Angular Materialin omalla sidenav-komponentilla. Törmäsin siinä kuitenkin samaan ongelmaan kuin infopanel-komponentin kanssa: sidenav-komponentti ei toimi, jos se on hajotettu eri komponentteihin. Sen käyttöönotto olisi vaatinut paljon alkuperäisen komponenttikoodin kiertämistä käyttämällä mukautettuja tyylejä sekä ylikirjoittamalla sen JavaScriptiä. Todetessani tämän huonoksi ratkaisuksi päätin etsiä jonkin toisen kolmannen osapuolen komponentin, jolla sivuvalikon toteutus onnistuisi helpommin. Huomasin kuitenkin, että suuri osa juuri tämän kaltaisista UI-plugineista ovat toteutettu jQuery-kirjastolla. Angulariin sisäänrakennettu, kevennetty versio jQuerystä (jQLite) ei kuitenkaan riitä tyydyttämään tarvittavia riippuvuuksia.

jQuery on ilmainen avoimen lähdekoodin JavaScript-kirjasto. Se on maailman käytetyin JavaScript-kirjasto (Usage of JavaScript libraries for websites, 2018). Se on tunnettu parhaiten sen kyvystä manipuloida verkkosivun DOM-mallia. Tämä tarkoittaa sitä, että jQueryllä voidaan helposti luoda, muunnella, poistaa ja lukea verkkosivun elementtejä. Parhaiden periaatteiden mukaisesti Angular-projektiin ei suositella jQueryn lisäämistä. Se nostaa huomattavasti riippuvuuksien kokoa ja voi aiheuttaa ongelmia Angularin elinkaarille, jos jQuery manipuloi DOMia. Jälkimmäisen ei pitäisi koitua kuitenkaan ongelmaksi, sillä jQuery tuotaisiin vain pluginin riippuvuuksien tyydyttämiseksi.

Sivuvalikon toteutus päätettiin tehdä SlideReveal-pluginin avulla. Se on Github-käyttäjä *nnattawat* tekemä helposti kustomoitava sivuvalikko-plugin. Jotta sen voisi asentaa, piti jQuery ottaa ensin käyttöön. Asensin jQueryn npm-paketinhallinnasta komennolla ”npm install jquery”. Jotta TypeScriptin tyypitysvaatimukset saataisiin tyydytettyä, asensin myös jQueryn tyypitykset komennolla ”npm install @types/jquery”. Seuraavaksi otin jQueryn käyttöön lisäämällä sen .angular-cli.json-tiedoston scripts-taulukkoon. Tämä käskee Webpack-moduuliniputusohjelmaa sisällyttämään jQueryn ng build -komennolla luodussa ohjelmassa. Nyt jQuery on määritetty projektinlaajuisesti käytettäväksi $-muuttujalla.

Seuraavaksi latasin SlideReveal-pluginin projektin assets-kansioon ja lisäsin sen .angular-cli.json-tiedostoon jQueryn tavoin. Tässä kuitenkin törmäsin ongelmaan; projektin build epäonnistuu, sillä jQueryllä ei ole slideReveal-ominaisuutta. Tämä ongelma syntyy siitä, kun SlideReveal luodaan koodilla $(.settings-sidenav’).slideReveal(). Vaikka koodi itsessään on oikein ja toimisi tavallisella JavaScriptillä, TypeScript vaatii vahvan tyypityksen. Yritin aluksi etsiä tyypityksiä definitelytyped.org-sivulta, mutta en löytänyt niitä. Tämän jälkeen yritin luoda tyypitykset käyttämällä dts-gen -komentorivisovellusta, mutta törmäsin tässä ongelmaan; slideReveal-ominaisuutta ei silti löytynyt jQuerylle.

Hetken tutkimisen jälkeen tajusin, missä vika piilee: jQueryllä ei ole ominaisuutta slideReveal. Tämä ei tarkoita välttämättä sitä, että vika olisi siinä, ettei SlideRevealille ole tyypityksiä, vaan siinä, että jQuerylle ladatuissa tyypityksissä (@typings/jquery) ei ole määritetty em. ominaisuutta. Korjasin ongelman lisäämällä typings.d.ts-tiedostoon jQuery-interfacen, ja siihen ominaisuuden slideReveal:

```
interface JQuery {
    slideReveal(options?: any): any
}
```

Kuva 20 slideRevealin lisääminen jQuery-interfaceen

Tämä ratkaisi ongelman; nyt jQueryllä on olemassa slideReveal-ominaisuus, ja ng build -komento suoriutui virheettä.

Nyt kun SlideReveal ja jQuery toimivat ja ovat käytössä projektissa, oli seuraavana vuorossa itse sivupaneelin luonti. SlideReveal ei sisällä minkäänlaista valmista sivupaneelin teemoitusta, joten se tulee luoda itse. Tämä on hyvä asia yhtenäisen ulkoasun kannalta, sillä se mahdollistaa Angular Material -kirjaston käyttämisen. Ensimmäiseksi loin uuden komponentin map-settings-menu, joka tulisi sisältämään itse sivupaneelin. Tämän jälkeen sijoitin komponentin map-komponenttiin erillisen ”settings-sidenav”-divin sisään, ja tyylitin sen mukailemaan Material Designin teemoitusta.

Nyt map-komponentilla on saatavilla ”map-settings-menu”-tyyliluokka, jonka avulla SlideReveal voi tehdä siitä sivupaneelin. Tein tämän map-komponentin TypeScript-tiedostoon kuvan 21 mukaisesti.

```
private createSettingsSidebar() {
    this.sidebar = $('.settings-sidenav').slideReveal({
        position: 'right',
        width: '280px',
        top: '65px',
        overlay: false,
        push: false,
        hide: () => {
            this.mapSettingsMenu.clearSelection()
        }
    })
}
```

Kuva 21 Sivupaneelin luonti

Ensin sivupaneeli laitetaan muuttujaan, jotta se voidaan esimerkiksi sulkea kutsumalla muuttujan ”sidebar” slideReveal-ominaisuutta, ja antamalla sille parametriksi ”hide”: ”this.sidebar.slideReveal(”hide”)”. Tämän jälkeen valitaan elementti, jolla on tyyliluokka settings-sidenav ja tehdään siitä SlideReveal-sivupaneeli. SlideReveal konfiguroidaan antamalla sille parametrissa JavaScript-objekti, joka sisältää esimerkiksi sen leveyden, sijainnin ja tiettyjen tapahtumien (kuten esimerkiksi sulkemisen) aikaan suoritettavat funktiot.

Jotta sivupaneelin saisi auki, aloin toteuttamaan hallintapainikkeita. Jokaiselle sivupaneelin välilehdelle tulisi olemaan oma ”kirjanmerkki”, joka on kiinnitetty sivupaneliin. Tällä tavalla sivuvalikosta saisi suoraan avattua haluamansa välilehden auki ilman, että käyttäjän tarvitsisi ensin avata sivupaneeli ja sen jälkeen navigoida halutulle sivulle (kuva 22). Tämän toteutin käyttämällä Angular Materialin *tab-group*-komponenttia. Jonkin kirjanmerkin valitseminen avaisi sivupaneelin ja asettaisi halutun välilehden *tab-groupin* tämänhetkiseksi sivuksi.

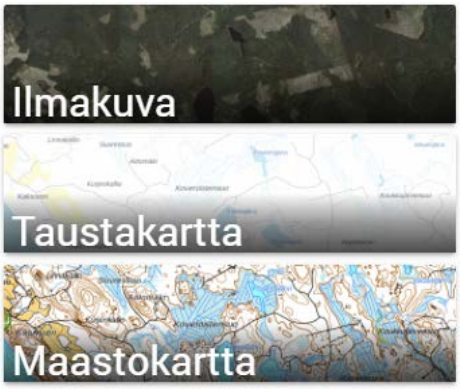
Mat-tab-group sisältää *inputin selectedIndex*, jolla voidaan määrittää aktiivinen välilehti antamalla sille välilehden indeksinumero, alkaen nollasta.

Loin kirjanmerkit käyttämällä tavallista div-elementtiä ja teemoittamalla ne Material Designin tyyliseksi. Lisäsin elementin sisään tavallisen painike-elementin, jotta kirjanmerkit käyttäytyisivät samalla tavalla kuin muutkin painikkeet. Lisäsin kirjanmerkeille *click*-tapahtumakäsittelijän, jolla suoritetaan komponenttikoodissa oleva *openSidebar*-metodi. Tämä metodi hoitaa välilehden avaamisen halutulle sivulle. Lisäsin myös tarvittavat käsittelyt sivupaneelin sulkemiselle ja muille hallintapainikkeille, kuten kartan suurennus- ja pienennyspainikkeille.

Kuva 22 Valmiin sivupaneelin "kirjanmerkit"

Nyt kun sivupaneelin hallinta oli valmis, aloin toteuttamaan alikomponentteja. Välilehdet koostuisivat alikomponenteista, jolloin niiden kehitys jatkossakin olisi selkeää. Niitä tulisi olemaan kolme: karttataso-, karttaobjekti- ja teemakarttavalikko. Aloitin karttatasovalikon toteuttamisesta. Vanhassa karttatasovalikossa karttojen ikonit olivat pieniä, ja en itse saanut niistä heti selvää. Kokeilin ratkaisua, jossa painikkeilla olisi taustakuvana esimerkki kyseisestä karttatasosta. Karttatasojen näyttöön ja vaihtoon tarkoitettu koodi oli jo olemassa entuudestaan, joten uusien painikkeiden toteutus oli vain uuden ulkoasun toteuttamista. Tein painikkeet käyttämällä Material Angularin painike-elementtiä, ja lisäsin oman teemoituksen sen päälle. Lopputuloksena sain hyvin kuvaavat painikkeet, jotka kuitenkin seurasivat muiden painikkeiden käyttäytymistä: valinta-animaatiot toimivat kuten muuallakin ja ne olivat näppäimistönavigoitavia (kuva 23).

Toteutin teemakarttavalikon samaan tyyliin. Myös teemakarttojen koodi oli valmiina, joten jäljelle jäi vain ulkoasun toteutus. Käytin Material Designin avatar-listaelementtiä. Se on normaali listaelementti, jossa ennen listauksen nimeä on esimerkiksi ikoni tai käyttäjäkuva.



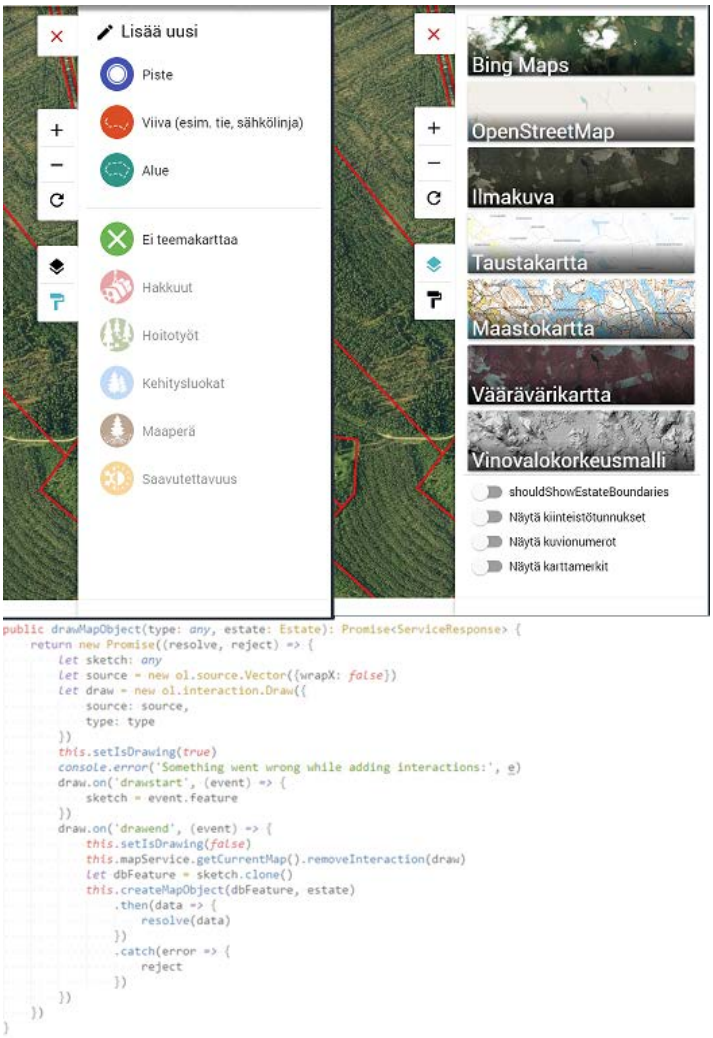
Kuva 23 Valmiit karttatasopainikkeet

Seuraavaksi generoin karttaobjektivalikkokomponentin. Karttaobjekti on merkintä kartalla. Esimerkiksi käyttäjä voi merkata viivan karttaan, joka kuvaisi sähkölinjojen sijaintia metsätilalla. Tämän komponentin luonti oli paljon haastavampaa kuin karttatasovalikon. Karttaobjektitukea uudessa Wuudiksessa ei ollut vielä toteutettu ja nykyisen AngularJs-version koodeja ei pystynyt soveltamaan nykyiseen Angular 5 -ratkaisuun.

Karttaobjektien haku, tallennus ja poisto toimi samalla tavalla kuin muissakin *serviceissä* käyttämällä REST-kutsuja. Karttaobjektien piirto osoittautui haastavammaksi. Loin uuden *servicen map-draw-service*, jossa itse piirtäminen tapahtuisi. Koitin aluksi soveltaa nykyisen Wuudiksen piirto-*servicen* metodeja, mutta ne eivät toimineet sovelluksen rakenteen eroavaisuuden, Angular- ja OpenLayers-versioeron takia suoraan. Onnistuin kuitenkin muuttamaan piirtologiikkaa siten, että se toimi uudessa Wuudiksessa.

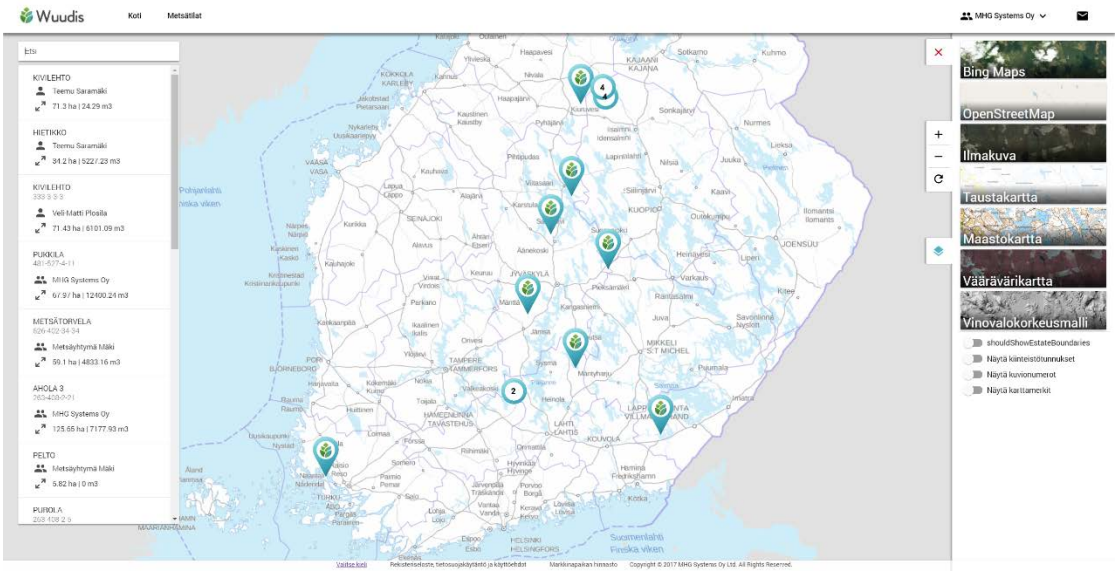
Karttaobjektin piirto tapahtuu käyttämällä OpenLayersin Draw-interaktiota. Interaktio (interaction) OpenLayers-kirjastossa tarkoittaa jotain käyttäjän toimintaa, joka vaikuttaa karttaan (OpenLayers Documentation, 2018). Sen käyttö on suhteellisen yksinkertaista. Aluksi luodaan uusi Draw-interaktio, jolle annetaan vektorilähde, johon piirto tapahtuu ja piirron tyyppi (esimerkiksi viiva). Sen jälkeen interaktio lisätään kartalle, joka näkyy käyttäjälle piirron alkami-

sena. Kuuntelemalla interaktion *drawend*-tapahtumaa, lopetetaan ja tallennetaan piirretty objekti. Piirto lopetetaan poistamalla draw-interaktio kartasta. Uusi ”hahmotelma” muunnetaan karttaobjektiksi, ja käyttämällä *map-object-servicen* tallennusominaisuutta lisätään uusi objekti tietokantaan. Myös uusi kuvio luodaan samalla tavalla (kuva 24).



Kuva 24 Uuden karttaobjektin hahmotelman piirto sekä valmis asetusvalikko

Lisäsin karttaobjektimenuun listauksen mahdollisista karttaobjektimalleista. Näitä klikkaamalla aloitetaan uuden karttaobjektin piirto. Karttaobjektia piirtäessä komponentti tuo näkyviin valikon, josta voi valita karttaobjektin värin ja nimetä sen. Saatuani sivupaneelin ja sen alikomponentin valmiiksi, oli kaikki karttaan kuuluvat pääelementit valmiit (kuva 25).



Kuva 25 Lopullinen karttakomponentti

Kartan lopullinen käyttöliittymä seurasi osittain Material Design -tyylikieltä. Vaikka alkuperäinen suunnitelmani seurata sitä tarkasti ei onnistunut kartan kaltaisen uniikin käyttöliittymäelementin tapauksessa, oli lopputulos kuitenkin sen kaltainen. Karttakomponentti onnistui mielestäni hyödyntämään Material Design -tyylikielen ulkoasua käytännöllisessä käyttöliittymäsuunnittelussa.

4 PÄÄTÄNTÖ

Karttakomponentti oli nyt saavuttanut pääpiirteittäin saman tilan kuin nykyisen Wuudiksen kartta. Uusi komponentti sisältää työkalut datan läpikäymiseen sekä kartassa että listassa. Karttamerkkien toiminnallisuus on myös alustavasti valmis ja kaikenlaisten *featureiden* näyttäminen on kartassa. Kuitenkaan julkaistavassa tilassa karttakomponentti ei tällä hetkellä ole. Käyttöliittymässä on pientä viilaamista ja kaikki suunnitellut ominaisuudet eivät olleet valmiita. Nämä eivät kuitenkaan kuuluneet minun opinnäytetyön aihealueeseen, joten omalta osaltani kartta oli valmis. Karttaa on kuitenkin jo jatkokehitetty ja em. asiat ovat varmasti jo työn alla.

Näin jälkikäteen tekisin monta eri asiaa eri tavalla. Varsinkin alkupään koodini olisi voinut olla komponenttiajattelun mukaisempaa ja siistimpää. Myös infopaneelin olisin toteuttanut samaan tyyliin kuin oikeanpuoleisen asetusvalikon alkuperäisen rautalankamallin mukaisesti. Olisin myös halunnut suorittaa käyttöliittymän käyttökokemuksen testausta jollain ulkopuolisella henkilöllä.

Opinnäytetyön tekeminen oli suuri oppimiskokemus itselleni. Opin mm. Angular-ohjelmistokehyksen käytön sekä uudelleenkäytettävän ja siistin koodin kirjoittamisen. Vaikka näin jälkikäteen karttakomponentti kuulostaa pieneltä osa-alueelta koko sovelluksesta, oli se kuitenkin kokonaisuutena minulle iso. Opin paljon uutta ja olisinkin halunnut sisällyttää opinnäytetyöhöni enemmän oppimiskokemuksia, mitä siihen olisi järkevästi mahtunut. Aihe oli minulle kaiken kaikkiaan mielenkiintoinen ja sopivan haastava.

LÄHTEET

About Node.js. 2017. Node.js Foundation. WWW-dokumentti. Saatavissa: https://nodejs.org/en/about/ [viitattu 2.12.2017]

Aguilar L. 2014. Why is Node.js so Popular? WWW-dokumentti. Saatavissa: https://dzone.com/articles/why-nodejs-so-popular [viitattu 3.12.2017]

Angular CLI. 2017. Angular Team. WWW-dokumentti. Saatavissa: https://github.com/angular/angular-cli/wiki [viitattu 3.12.2017]

Angular Docs. 2017. Angular Team. WWW-dokumentti. Saatavissa: https://angular.io/docs [viitattu 28.11.2017]

Angular Material Documentation. 2017. Google. WWW-dokumentti. Saatavissa: https://material.angular.io/ [viitattu 12.1.2018]

JetBrains s.a. WebStorm features. WWW-dokumentti. Saatavissa: https://www.jetbrains.com/webstorm/features/ [viitattu 15.11.2017]

Lavin J. 2014. AngularJS Services. Birmingham: Packt Publishing.

Material Design Guidelines. 2017. Google. WWW-dokumentti. Saatavissa: https://material.io/guidelines [viitattu 12.1.2018]

npm Documentation. 2017. npm, Inc. WWW-dokumentti. Saatavissa: https://docs.npmjs.com/ [viitattu 2.12.2017]

OpenLayers Documentation. 2018. WWW-dokumentti. Saatavissa: https://openlayers.org/en/latest/doc/ [viitattu 16.12.2018]

RxJS API Document. 2017. WWW-dokumentti. Saatavissa: http://reactivex.io/rxjs/ [viitattu 22.12.2017]

Siang, T. 2017. Material Design and the Mystery Meat Navigation Problem. WWW-dokumentti. Saatavissa: https://medium.freecodecamp.org/material-design-and-the-mystery-meat-navigation-problem-65425fb5b52e [viitattu 3.1.2018]

Usage of JavaScript libraries for websites. 2018. W3Techs. WWW-dokumentti. Saatavissa: https://w3techs.com/technologies/overview/javascript_library/all [viitattu 2.1.2018]

Wireframing s.a. U.S. Department of Health & Human Services. WWW-dokumentti. Saatavissa: https://www.usability.gov/how-to-and-tools/methods/wireframing.html [viitattu 14.1.2018]