Tallinna Ülikool

Informaatika Instituut

Kaardirakenduse loomine Leaflet'i teegiga

Õppematerjal

Seminaritöö

Autor: Elari Roop

Õpperühm: IF-11

Juhendaja: Jaagup Kippar

Autor: …… " ..... " 2014

Juhendaja: …..... " ..... " 2014

Instituudi direktor: …..... " ..... " 2014

Tallinn 2014

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

SISUKORD

SISSEJUHATUS

Javascriptil põhinevate kaardirakenduste loomise võimalused on viimastel aastatel oluliselt kasvanud, lubades teha kõike, kujutlusvõime piirideni. Kaardirakendustel on Javascriptis suur arengupotentsiaal ja eelise annab töötamine kõikidel levinud brauseritel operatsioonisüsteemist sõltumata, mis tähendab seda, et ühes keeles kirjutatud rakendus toimib peaaegu kõikidel seadmetel.

Internetis töötavate kaardirakenduse loomiseks on võimalik valida paljude erinevate tootjate vahel. Põhjaliku ülevaate erinevatest teekidest ja nende võimalustest annab lisas 3 olev tabel (Donohue, 2012).

Google Maps, Bing Maps, OpenLayer ja Leaflet on suurimad Javascriptil põhinevad raamistikud ja vaid kaks viimast nendest on vabavaralised. OpenLayers pakub väga palju võimalusi ja suurt näidete varamut. Aastate jooksul on funktsionaalsuse lisamisel lähtekood muutunud suureks ja sellega loodud kaardid aeglased. Leaflet ei võimalda suurel hulgal erinevaid võimalusi, kuid väike lähtekood annab alust loota palju kiiremale lehe laadimise ajale võrreldes OpenLayers´iga. Leaflet on uudne ja kasutab *CSS3* võimalusi animeerimiseks.

Käesoleva töö teema valik tuleneb asjaolust, et Eesti keeles ei leidu autorile teadaolevalt materjale ega uurimusi, mis Leafleti käsitleks. Eesmärgiks oli luua õppematerjal, mis annaks praktilisi oskuseid interaktiivse veebipõhise kaardi loomiseks Eesti ja Soome näitel. Õppematerjali peab olema ülevaatlik ja kompaktne ja valminud näide töötama mobiilsetel seadmetel ja ei tohi olla aeglane. Selgitaks, kuidas kaardiandmeid lihtsustada ja töödelda ning anda ülevaade Leafleti võimalustest ja tõsta üldist teadlikkust kaardirakenduste võimaluste kohta.

Esimeses peatükis tutvustatakse Leafleti, miks selle populaarsus on kiirelt kasvanud ja millised on raamistiku eelised ja puudused. Kirjeldatakse kaardikihtide ning projektsioonide tuge, tutvustatakse sisse-ehitatud funktsioone ja lisasid (*plugins*), mis tavaliselt kasutust leiavad.

Teises peatükis on käsitletud andmete ettevalmistamist, andmekogudes olevaid andmeid ja nende kasutusvõimalusi. Kaardiandmete transformeerimist ja lihtsustamist. Andmete töötlemist ning teisendamist vajalikesse vormingutesse. Maa-ameti WMS-serverist (*Web Map Service*) kaartide laadimist, kaardikihtide info küsimist ja üksikutelt objektidelt info pärimist Leafleti abil.

Neljandas peatükis antakse ülevaade õppematerjali struktuurist, õppematerjali testimisest, tagasisidest ning analüüsitakse saadud tulemust.

Lisa 1 sisaldab valminud õppematerjali ja Lisa 2 andmete lihtsustamise näidet. Lisas 3 on toodud veebis töötavate kaardirakenduste võrdlus.

Õppematerjal sisaldab ka väga palju erinevaid faile ja valminud näiteid, mida on saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse vastavalt teemadele.

MÕISTETE JA LÜHENDITE REGISTER

CRS (Coordinate Reference System) – Koordinaatide referentssüsteem

JSON, JSONP (*Javascript Object Notation, JSON with padding)* – Javascripti objektide standardid.

GeoJSON – Geograafilise informatsiooni hoidmiseks mõeldud JSON formaat.

WMS – *Web Map Service*, teenus veebist rasterkihtide laadimiseks.

WFS – *Web Feature Service,* võimaldab teha erinevaid päringuid ja saadab andmeid ka vektorkujul.

Marker – visuaalne abivahend punkti tähistamiseks kaardil.

BBOX (*Bounding Box),*– Nelinurk, mille piirides geograafiline objekt asub.

API (*Application Programming Interface)* – Rakenduse programmeerimise liides.

Open Data - Open data on praktika mille puhul tehakse kättesaadavaks info kõigile ilma ligipääsu, patentide ja autoriõiguse piiranguteta (OpenData.ee, 2014).

AJAX *(Asynchronous JavaScript and XML)* – Asünkroonne Javascript ja XML.

Tile layer – Pilditükkidest koosnev kaardikiht

Polügoon (*Polygon, Multi-polygon*)- Hulknurk või hulknurkade kogum matemaatikas

1. LEAFLETI TUTVUSTUS

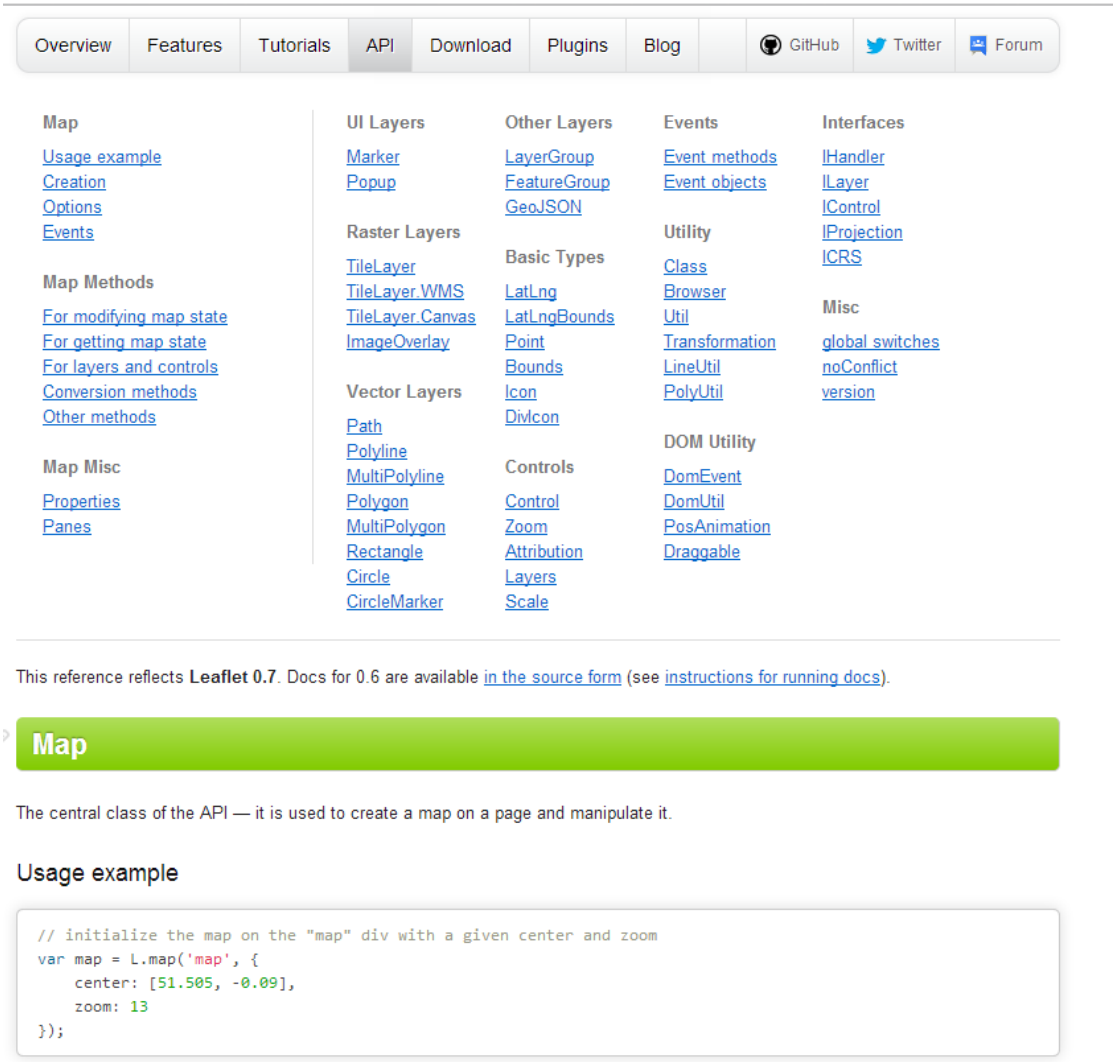
Leaflet on vabavaraline Javascripti teek (*Library*), mis võimaldab luua väga erinevaid kaardirakendusi personaalarvutitele ja mobiilsetele seadmetele. Võrreldes tuntud konkurentidega, nagu Google Maps ja OpenLayer, on Leaflet kogunud viimastel aastatel väga suurt populaarsust oma avatuse, väiksuse ja modulaarse ülesehitusega. Leafleti lähtekood on 123 KB, mis on OpenLayeri koodist kuus korda väiksem, võimaldades sellega kiiremat veebilehe laadimise aega.

Leaflet võimaldab kasutada mitmeid erinevaid kaardikihte, mis jaotuvad vektor- ja rasterkihtideks. Vektorkihid jagunevad GeoJSON kihtideks, mis võivad omakorda sisaldada väiksemaid geomeetrilisi kihte nagu polügoone ja ringe. Rasterkihid jagunevad WMS- ja *Tile* kihtideks, kus andmed on piltide kujul. Kõiki kihte saab lisada kihtide gruppidesse nende kergemaks haldamiseks.

Leafletil on hetkel üle saja arendaja ja selle visiooniks on, et kõikidest järgnevatest versioonidest hakatakse funktsionaalsust eemaldama kuni sellest saab tuum, mille kõik lisafunktsionaalsused hakkavad töötama läbi erinevate lisamoodulite (*plugins*). (Agafonkin, High Performance Data Visualizations in JavaScript, 2013)

Leafleti eelisteks võib pidada väga head dokumentatsiooni, kõik saadaolevad komponendid on hästi grupeeritud ja ühele lehele mahutatud. Iga komponendi juures on välja toodud esimesena lühikirjeldus, seejärel koodinäide kasutamisest ja selle atribuudid ja omadused. (Joonis 1). Mainimata ei saa ka jätta lihtsa API olemasolu, kus lihtsaima kaardi saab teha juba mõne rea koodiga.

Töös kasutati Leafleti viimast kättesaadavat stabiilset versiooni numbriga 0.7.2 ja lähitulevikus on oodata ka versioone 0.8 ja 1.0 (Agafonkin, Leaflet, 2014). Versioonis 0.8 plaanitakse hõlbustada erinevate projektsioonide kasutamist ja seetõttu ei pruugi näited järgnevatel versioonidel töötada.



Joonis 1 Leafleti API dokumentatsioon (Agafonkin, Leaflet, 2014)

Leafleti nõrkuseks võib pidada asjaolu, et ta ei toeta vaikimisi mitmeid erinevaid kaardi projektsioone, sealhulgas ka Eesti kaarte, mis on salvestatud EPSG:3301 süsteemis. Nende nende kasutamiseks tuleb kaardid transformeerida Leafletile sobivaks. Probleeme tekitab ka osadel juhtudel veateadete mittekuvamine ja keeruline on vea põhjust välja selgitada. Spetsiifilisemate probleemidega on uudsuse ja vähese kasutajaskonna tõttu raske lahendusi leida.

1.1. Leafleti API

Leafleti kasutamiseks tuleb HTML failis määrata Leafleti CSS ja Javascript failide asukohad (Koodinäide 9). Kõik Leafleti funktsioonid on koodis kergesti äratuntavad, sest algavad tähega L (Koodinäide 1). Leafleti API pakub paljusid erinevaid võimalusi ja kõiki kirjeldada pole mõistlik, välja on toodud õppematerjalis kasutatud funktsioonid.

```
L.funktsiooni_nimi();
```

Koodinäide 1 Leafleti objekti kasutamise näide

1.1.1. L.map

L.map ülesandeks on uue kaardi loomine ja kohustuslikeks parameetriteks on HTML elemendi *id*, mille sisse kaart paigutatakse, ning kaardi omadused (Koodinäide 2).

```
var map = new L.Map('map', {
      center: new L.LatLng(58.59, 25.48), // keskpunkti koordinaadid
      zoom: 7, // Suurenduse aste
      crs:crs, // kaardi projektsioon
      layers:[myLayer], // lisatav kaardikiht
});
```

Koodinäide 2 Kaardi loomine ja selleks vajalikud parameetrid

Kaardi omadustest on kohustuslikud vaid keskpunkti koordinaadid ja suurenduse aste, kuid nende määramiseks võib kasutada ka sisseehitatud funktsioone. Kaardi keskpunkt ja suurenduse aste arvutatakse välja myLayer kihi piiridest ja paigutatakse kaardi keskele maksimaalse suurenduse astmega, millega kiht ei välju kaardi piiridest (Koodinäide 3).

```
map = new L.map('map', {
  crs:crs, // kaardi projektsioon
  layers:[ myLayer], // lisatav kaardikiht
});

map.fitBounds(myLayer.getBounds()); //kaardi mahutamine kihi
piiridesse
```

Koodinäide 3 Dünaamiliselt kaardi keskpunkti ja suurenduse astme määramine

1.1.2. L.geoJSON

L.geoJSON objektiga on võimalik kaardile lisada vektorkujul andmeid, mis on GeoJSON formaadis. L.geoJSON kasutamiseks on vaja parameetrina anda objekt, mille sees GeoJSON andmed asuvad. Selleks võib teha päringu, mis tagastab GeoJSON formaadis andmed või luua uus objekt, mille sees asuvad GeoJSON kujul andmed. (Koodinäide 4).

```
var omavalitsused={
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "ONIMI": "Kiili vald", "MNIMI":
"Harju maakond" }, "geometry": { "type": "Polygon", "coordinates": [
[ [ 24.767396604767875, 59.359862764662928 ], […] ] } }
]};

var myLayer = L.geoJson(omavalitsused);
```

Koodinäide 4 L.geoJSON kasutamise näide

1.1.2.1. onEachFeature

Tegemist on L.geoJSON omadusega, mis võimaldab igale *Feature* objektile, antud näites igale omavalitsusele, lisada funktsioone, mida objekti loomisel välja kutsutakse (Koodinäide 5).

```
myLayer = L.geoJson(omavalitsused, {
      onEachFeature: function(feature, layer){
            layer.setStyle(defaultStyle(feature)); //värvi määramine
            layer.on('mouseover', function(e){ // mouseover event
                  var popup= L.popup().setLatLng([e.latlng.lat+0.05,
                  e.latlng.lng]).setContent(popupContent).openOn(map)
                  ;
            });
      }
});
```

Koodinäide 5 onEachFeature kasutamise näide

Sellisel viisil on võimalik igale *Feature* objektile lisada erinevaid omadusi, näiteks määrata nende värvi vastavalt objekti omadustele ja käitumist erinevate sündmuste (*events)* korral.

1.1.3. L.tileLayer.wms

L.tileLayer.wms abil on võimalik WMS-teenust pakkuvast serverist kaardikihte PNG või JPEG formaadis kuvada. Kasutamiseks tuleb teada, milliseid KRS-i tüüpe teenus toetab ja kohustuslike parameetritena tuleb anda WMS-teenuse aadress, kihtide nimed, formaat ja läbipaistvus (Koodinäide 6).

```
var aluskaart= L.tileLayer.wms('http://kaart.maaamet.ee/wms/alus', {
  layers: 'HALDUSPIIRID',
  format: 'image/png',
  continuousWorld: true,
  transparent:true,
});
```

Koodinäide 6 L.tileLayer.WMS kasutamise näide

1.1.4. L.layerGroup

L.layerGroup loob võimaluse kasutajatel erinevaid kihte sisse ja välja lülitada. L.control.layers lisab Leaflet kaardile ikooni kihtide haldamiseks ja lisab valikusse kõik L.layerGroup all defineeritud kihtide nimed. Kihte on kahte liiki *baseLayers* ja *overLays*. Esimesed moodustavad kaardi aluskihid ja neid saab vahetada, aga mitte välja lülitada. *OverLays* moodustavad pealmise kihi, mida saab ükshaaval sisse ja välja lülitada (Koodinäide 7).

```
var theLayers = new L.layerGroup([orto, myLayer]);
var baseLayers = {
    "Ortofoto": orto,
};
var overlays = {
  "Omavalitsused": myLayer
};
L.control.layers(baseLayers, overlays).addTo(map);
```

Koodinäide 7 L.layerGroup kasutamise näide

1.1.5. L.popup

L.popup abil on võimalik kaardil avada hüpikaknaid vastavalt ette antud koordinaatidele. Parameetritena on vajalikud koorinaadid, hüpikaknas kuvatav sisu ning kaardi objekt, millel hüpikaken avada (Koodinäide 8).

```
var popup= L.popup().setLatLng([e.latlng.lat+0.05,
e.latlng.lng]).setContent(popupContent).openOn(map);
```

Koodinäide 8 L.Popup kasutamise näide

1.2.Leafleti lisamoodulid

Leafleti teeb mitmekülgseks modulaarne ülesehitus, mis võimaldab funktsionaalsuse puudumisel kasutada või ise luua erinevaid lisamooduleid. Neid on Leafleti kodulehel palju ja siinkohal mõned tähtsamad koos selgitustega:

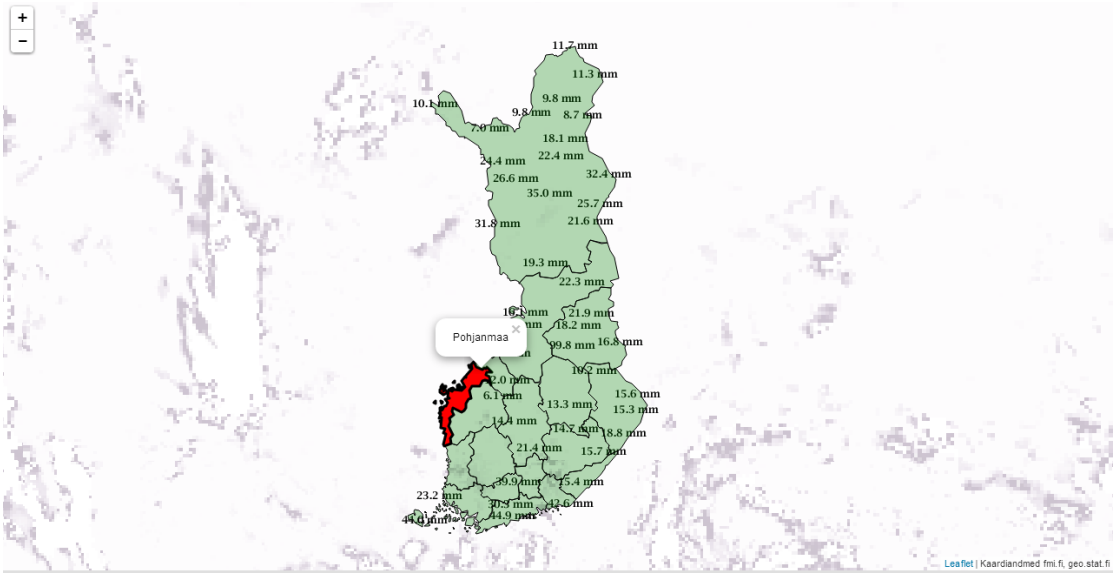
- Proj4Leaflet (Kartena AB, 2013) – võimaldab kasutada Leafleti poolt toetamata projektsioone, sealhulgas ka Eesti omasid. Siinkohal tasub märkida, et L.GeoJSON funktsiooniga kasutatavaid andmeid sellega ümber transformeerida pole võimalik.
- Leaflet-ajax (Metcalf, 2014) – JSON ja JSONP failide laadimiseks AJAX-i abil.
- Simplify.js (Agafonkin, Simplify, 2014) – andmete lihtsustamiseks, vähendab hulknurga joonistamiseks kasutatavate punktide arvu. Hulknurga kuju visuaalselt hinnates oluliselt ei muutu.

2. ANDMETE OTSIMINE JA ETTEVALMISTAMINE

Õppematerjalis toodud näidete jaoks vajalikud andmete hankimiseks kasutati Maa-ameti ja Statistikaameti kodulehekülgi. Maa-ameti kodulehte omavalitsuste kaardiandmete leidmiseks ja Statistikaameti lehte kaardiandmete sidumiseks teiste andmetega, näiteks rahvastikuandmetega.

2.1. Kaardirakenduste loomise tingimused

Kõige paremaks lahenduseks on, kui vektorkujul andmeid saab JSON või XML formaadis pärida otse algallikast. Andmed oleks sellisel juhul kõige uuemad ja nende töötlemine Leafletis automaatselt teostatav. Vektorandmed pakuvad ka rohkem võimalusi nendega manipuleerimiseks, näteks värvi muutmiseks ja suurendamiseks, mis *Tile Layer* korral pole võimalik. Heaks näiteks on siinkohal Soome, kus juba 2003. aastal alustati tööd nüüd juba *Open Data* nime all tuntud põhimõtetega (Lehtonen, 2011). Soome riiklike institutsioonide serveritest saab teha väga erinevaid päringuid, näiteks haldusjaotusest, satelliidi pilte pilvisusest ja viimase kuu sademeid (Joonis 2).



Joonis 2 Õppematerjalis toodud WFS- ja WMS-teenuse kasutamine Soome näitel

Eestis on samuti *Open Data* põhimõtet aktsepteeritud, kuid hetkel toimub kontseptsioonide ettevalmistamine (OpenData.ee, 2014).

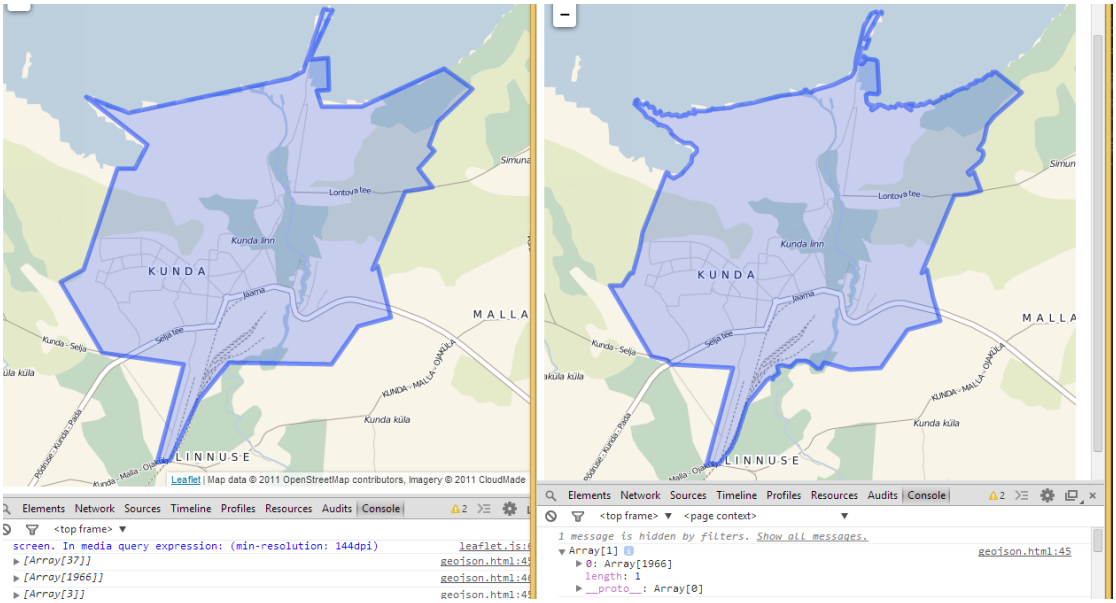
Maa-ameti kodulehelt kaartide laadimiseks vektrokujul GeoJSON või XML formaadis hetkel võimalused puuduvad, sest kuigi Maa-ametil on WFS-server olemas, pole see mõeldud avalikuks kasutamiseks. Hetkel toimub *INSPIRE* direktiivi elluviimine (Keskkonnaministeerium, 2013), mille tulemusel võib lähiajal võimalikuks saada ka vektorkujul andmete pärimine ja siis oleks mõistlik teemat edasi uurida.

2.2. Kaardiandmete hankimine ja töötlemine

Maa-ameti lehel on mitmeid erinevaid kaarte, mida on võimalik alla laadida ja kasutada. Õppematerjalis kasutatakse omavalitsuste kaarti, sest see on lihtsasti mõistetav, terve Eesti pind on kaetud ja selle abil on võimalik näidata, kuidas objekte teatud kriteeriumite alusel rühmitada.

Leaflet Maa-ameti poolt pakutavaid formaate ei toeta ja andmete kuvamiseks tuleb need kõigepealt transformeerida sobivasse koordinaatide referentssüsteemi. Seda on võimalik teha kasutades vabavaralist programmi nimega QGIS, mis võimaldab andmeid Leafletile sobivasse GeoJSON formaati salvestada.

Tekkinud fail on veebis kuvamiseks liiga suur, maht umbes 60 MB. Selle laadimisel kasutas brauser gigabaidi mälu, mida on liiga palju. Andmete kiiremaks laadimiseks on võimalik kasutada serveripoolset pakkimist, mille tulemusel vähendab saadetava faili suurus üle viie korra. Andmete lihtsustamiseks on võimalik kasutada eelpool mainitud lisamoodulit nimega Simplify.js. Lihtsustamise tulemusi on näha joonisel (Joonis 3), kus vasakpoolne linn on lihtsustatud ja joonistamiseks on kasutatud 37 punkti, parempoolsel aga pole lihtsustamist tehtud ja see koosneb 1966 punktist.



Joonis 3 Objektide lihtsustamine kasutades Simplify.js moodulit

Eelnev viis on automaatseks lihtsustamiseks väga efektiivne väikeste või siis täpsete andmete kuvamiseks ja selle näide on toodud lisas 2. Probleemiks kujunes aga asjaolu, et sedavõrd suurte andmete lihtsustamine nõudis lehe laadimisel liiga palju aega ja lihtsustamisel tekkisid omavalitsuste piiride vahele tühimikud, sest Simplify.js kasutab lihtsustamiseks Douglas-Peuckeri algoritmi, ning see ei võimalda selliste aukude tekkimist vältida.

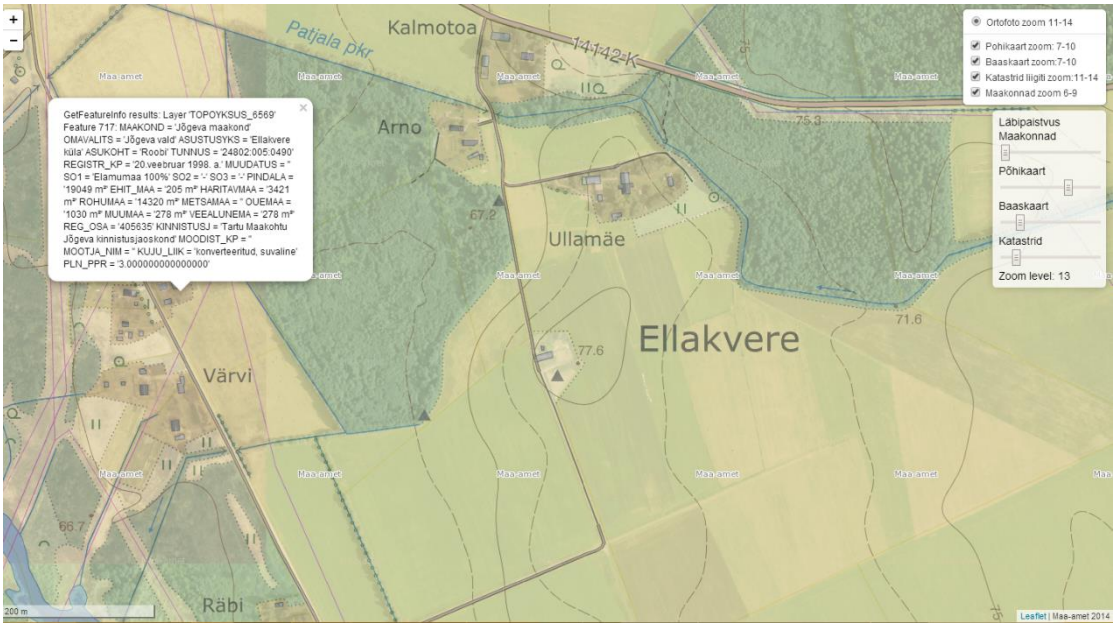
Lahenduseks oleks kasutada modifitseeritud Visvalingam algoritmi (Bloch, https://github.com, 2014), sest see suudab kõrvuti olevate objektide piire säilitada. Parim leitud vabavaralistest lehekülgedest, on veebisait nimega Mapshaper (Bloch, MapShaper, 2013), kuhu õige projektsiooniga salvestatud GeoJSON formaadis faili üles laadides saab määratleda lihtsustamise tolerantsi ja valminud faili kohe alla laadida. Selliselt valmis õppematerjalis kasutatav **omavalitsused.json** fail.

Statistikaameti kodulehelt andmeid otsides selgus, et sealt pole võimalik andmeid pärida kindla veebiaadressi poole pöördudes. Rahvaarv omavalitsuste kaupa laeti alla *CSV* formaadis ja kasutades programmi QGIS oli seda võimalik importida ning geomeetriata GeoJSON failiks salvestada. GeoJSON formaat sai valitud seetõttu, et selle andmed on Javascripti objektid ja neid on kerge töödelda.

2.3. Maa-ameti WMS-teenus

WMS-teenuse kasutamise tugi on Leafletil olemas ja kuna selle tööpõhimõte on *Tile Layer* omaga väga sarnane võib neid koos vaadata. WMS-teenust kasutatakse tavaliselt selleks, et tõmmata serverist rasterkujul olevaid andmeid. Maa-ameti WMS-teenus pakub väga palju erinevaid kaarte Eesti kohta, näiteks elektriliinide kaart või talunimedega kaart.

Õppematerjalis kasutati mitmeid erinevaid kaardikihte korraga, et näidata, kuidas on Leafletis võimalik erinevaid kaardikihte kuvada ja neid sisse- ja välja lülitada. Kasutusel oli ka WMS-teenuse võimalus küsida kihte läbipaistvatena ja kuvada selliselt mitu kihti üksteise peale, võimaldades kasutajal iga kihi läbipaistvust eraldi muuta (Joonis 4).



Joonis 4 WMS kihtide näide

2.3.1. WMS-teenuse GetFeatureInfo päring

Maa-ameti WMS-teenus võimaldab ka osadelt kihtidelt *GetFeatureInfo* päringuid teha ja õppematerjalis loodud näites saab hiirekliki korral küsida katastri kohta käivat informatsiooni, mille tulemusel tehakse Maa-ameti serverisse päring ja kuvatakse saadud andmed tekkivasse hüpikaknasse (*Popup*) (Joonis 4).

3. ÕPPEMATERJALI ÜLESEHITUS, TESTIMINE JA ANALÜÜS

Õppematerjali loomise vajadus tulenes eestikeelsete materjalide puudumisest Leafleti kohta ja ka üldiselt on veebipõhiste kaardirakenduste kohta võrdlemisi vähe informatsiooni. Lisaks sellele on autori arvates paljud kaardirakendused aeglased ja halva kasutajaliidesega.

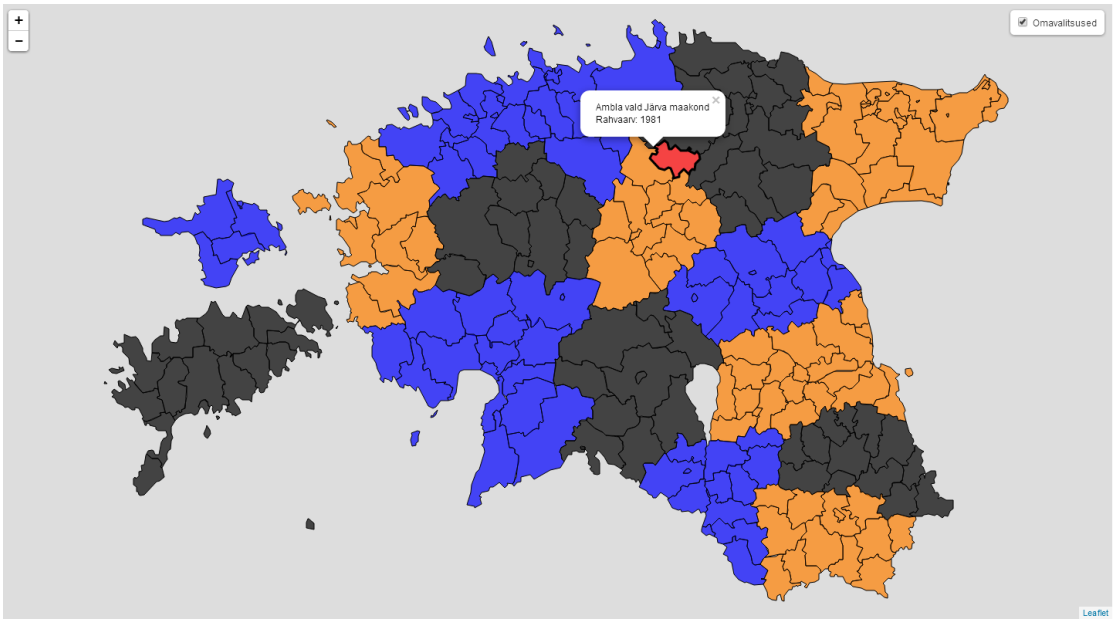
3.1. Õppematerjali nõuded

Õppematerjal on suunatud informaatika üliõpilastele ja eeldab algteadmisi programmeerimisest ja aine „Programmeerimise põhikursus“ läbimist. Suurem osa tööst puudutab Javascripti ja PHP-d. Materjalis käsitletatakse ka XML-ja JSON-failide struktuure ja seetõttu peaks olema sooritatud või sooritamisel aine „XML rakendused“. Materjali läbimiseks tuleks arvestada hinnanguliselt viie tunniga.

3.2. Õppematerjali ülesehitus

Valminud õppematerjal koosneb kolmest põhiosast ja lisadest. Esimene peatükk käsitleb andmete ettevalmistamist ja töötlemist ning mille võib vajaduse korral ära jätta, sest valminud andmed näite jaoks on failina lisatud.

Teine peatükk näitab kuidas Leafletis GeoJSON faili kaardile kuvada, andmete filtreerimist ja kujundamist vastavalt objekti omadustele. Lisaks veel objekti omaduste muutmist vastavalt hiire sündmustele (*mouse Events*), erinevate GeoJSON failide sisu ühendamist ja kaardikihtide haldust. Valminud näide vektorkihtide kasutamisega (Joonis 5) .



Joonis 5 GeoJSON näide

Kolmandas peatükis selgitatakse Maa-ameti WMS-teenuse võimalusi, uuritakse, millised kaardid on serveris saadaval ja millistes formaatides neid alla saab laadida. Pärast seda toimub WMS-kihtide lisamine, kihtide haldamise ja läbipaistvuse lisamine. Viimaseks toimub WMS-teenuse *GetFeatureInfo* päringu loomine, mis võimaldab andmeid pärida vastavalt klikitud koorinaatidele.

Õppematerjali lisas on ülesandeid ja näiteid, mis õppematerjali sisse ei mahtunud ning mõeldud iseseisvaks jätkamiseks teema vastu huvi tundvatele inimestele. Käsitletakse objektide otsimist kaardilt, pindalade arvutamist, erinevate projektsioonide korraga kuvamise võimalusi ja WFS- ja WMS-teenuste kasutamist Soome näitel. Sellele lisaks veel *WebWorker'*ite kasutamist andmete laadimiseks teises lõimes (*thread*), mis võimaldab rakendust andmete laadimise ajal edasi kasutada.

Õppematerjal sisaldab ka väga palju erinevaid faile ja valminud näiteid, mida on saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse.

3.3. Õppematerjali testimine

Õppematerjali oli võimalik testida kursuse „XML rakendused“ raames, mille eeldusaineks on „Programmeerimise põhikursus“. Testitav materjal sobis kursuse raamidesse, sest õppematerjalis tutvustati ja kasutati XML ja JSON formaate.

Esimest osa sellel korral ei proovitud, sest programmide paigaldamine oleks liiga palju aega võtnud ja see ei läinud otseselt kursuse teemaga kokku, pakkudes ehk rohkem huvi geograafidele. Ettevalmistuse osa põhiideed selgitati tudengitele lahti sissejuhatava osana ja põhirõhk oli osa lõpus tekkinud GeoJSON faili struktuuri selgitamisega, et seda järgmises osas kasutama hakata.

Teises osas alustati lihtsa näitega, omavalitsuste kaardile kuvamisega, ja seejärel lisati iga sammuga juurde kindel osa funktsionaalsust. Õppematerjali osa läbiti probleemideta esimese loengu lõpuks ja kõik tudengid said oma variandi tööle.

Kolmas osa oli kõige keerulisem ja sellega esines ka enim probleeme. Esimeseks suuremaks probleemiks kujunes asjaolu, et õppematerjali PDF-failiks konverteerides kadusid mõningad sümbolid ära ja osad read poolitati ning need ei töötanud enam. Paratamatult tulid sisse mõned trükivead ja asjaolu, et PHP faili sisule veebi kaudu ligi ei pääse.

3.4. Õppematerjali tagasiside ja analüüs

Peale tunni lõppu küsisime tudengitelt tagasisidet ja paremaks ning huvitavamaks peeti GeoJSON näidet. Esimeses iseseisva ülesande said mõned tudengid paari minutiga valmis, teistele valmistas see aga raskuseid. Tudengid, kes kiiremini valmis said, ütlesid, et neil on mõningast kogemust Javascriptiga ja ülesanne oli väga lihtne. Õppermaterjalist arusaamise eelduseks oleks seega algteadmiste omamine Javascripti programmeerimiskeelest.

WMS-kihtidega ülesandeks tuli tudengil iseseisvalt XML failist üles otsida kaardikihi nimi ja lisada see oma kaardile. Probleemina ilmnes asjaolu, et teise kaardikihi lisamisel

ei pruukinud see samade suurendusastmete (*Zoom level*) juures töötada ja selle vältimiseks asendati õppematerjalis aluskiht teise kihiga, mida on võimalik vaadata kõikide suurendusastmete juures.

Tagasiside põhjal sai materjali parendatud ja algselt kahes erinevas näites olnud sisu ühte suuremasse näitesse koondada.

KOKKUVÕTE

Javascriptil põhinevatel kaardirakendustel on tulevikus palju potentsiaali, võimaldades luua kiireid ja reaalajas töötavaid süsteeme.

Paremate kaardirakenduste eelduseks on, et kõik riigiasutused looks võimalused veebraadressi kaudu päringute tegemiseks ja tagastavad andmeid JSON formaadis. Maa-ameti WMS näites polnud võimalik kasutada WFS-teenust ja seda tutvustati õppematerjalis hoopis Soome näitel.

Valminud õppematerjal annab ülevaate, mida kaardirakendustes teha on võimalik, kasutades vektor- ja rasterandmetega ümberkäimiseks vajalike funktsioonide ja võimaluste tutvustamist. Õppematerjali esimeses osas käsitletakse andmete ettevalmistamist, teises osas käsitletakse vektorkujul olevaid andmeid ning kolmandas raster- ja vektorandmeid koos. Õppematerjali lisas on toodud näited, mida kaardirakendustega veel on võimalik teha. GeoJSON formaati toetavad paljud rakendused ja valminuid GeoJSON faile on võimalik Google Maps, OpenLayers jpt rakenduste juures kasutada ning loodetavasti lihtsustab ka nende jaoks vajalike õppematerjalide loomist.

Materjali testimine andis informatsiooni tekkinud probleemide kohta ja aitas õppematerjali paremini süstematiseerida ja lihtsustada, mille tulemusel ühendati algselt kahe erineva näite sisu.

Kõiki õppematerjalis olevaid faile ja näiteid saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse vastavalt teemadele. Lõpliku näidet saab vaadata aadressilt: http://elariroop.com.hostinghood.com/leaflet/wms_lopp/.

KASUTATUD KIRJANDUS

Agafonkin, V. (2013). High Performance Data Visualizations in JavaScript. *TopConf*. Tallinn.

Agafonkin, V. (2014). Kasutamise kuupäev: 26. 02 2014. a., allikas Leaflet: http://leafletjs.com/reference.html

Agafonkin, V. (26. 02 2014. a.). *Simplify*. Allikas: GitHub: http://mourner.github.io/simplify-js/

Bloch, M. (2013). *MapShaper*. Kasutamise kuupäev: 26. 02 2014. a., allikas MapShaper: http://mapshaper.org/

Bloch, M. (2014). *https://github.com*. Kasutamise kuupäev: 27. 02 2014. a., allikas https://github.com/mbloch/mapshaper/wiki/Simplification-Tips

Clarck, R. (2013). *Github*. Kasutamise kuupäev: 26. 02 2014. a., allikas https://gist.github.com/rclark/6908938

Donohue, R. W. (2012). *University of Wisconsin-Madison.* Kasutamise kuupäev: 27. 02 2014. a., allikas http://geography.wisc.edu/cartography/slides/emerging-web-mapping-donohue-et-al-nacis2012.pdf

Kartena AB. (2013). *GitHub*. Kasutamise kuupäev: 02. 02 2014. a., allikas Proj4Leaflet: https://github.com/kartena/Proj4Leaflet

Keskkonnaministeerium. (20. 06 2013. a.). *http://www.envir.ee/*. Kasutamise kuupäev: 27. 02 2014. a., allikas http://www.envir.ee/orb.aw/class=file/action=preview/id=1199555/Eduard+Pukkonen+INSPIRE+direktiivi+elluviimine+%26%238211%3B+andmete+harmoniseerimine,+teenuste+arendamine+ja+maakatte+andmestiku+uuendamine,+eelm%E4%E4ratletud+projekt.pdf

Lehtonen, P. (22. 06 2011. a.). *Open data in Finland - Public sector perspectives on open data.* Kasutamise kuupäev: 26. 02 2014. a., allikas http://virtual.vtt.fi/: http://virtual.vtt.fi/virtual/nextmedia/Deliverables-2011/D3.2.1.2._Hyperlocal_Open%20data%20in%20Finland.pdf

Maaamet. (16. 02 2012. a.). *Maaamet*. Allikas: http://geoportaal.maaamet.ee/est/Ruumiandmete-infrastruktuur/OpenGIS-p67.html

Metcalf, C. (26. 02 2014. a.). *GitHub*. Allikas: Leaflet-ajax: https://github.com/calvinmetcalf/leaflet-ajax

OpenData.ee. (27. 02 2014. a.). *Open Data Estonia.* Allikas: http://www.opendata.ee/hetkeolukord-eestis/

OpenData.ee. (27. 02 2014. a.). *OpenData.ee*. Allikas: http://www.opendata.ee/mis-on-open-data/

LISA 1 ÕPPEMATERJAL

Tallinna Ülikool

Elari Roop

Kaardirakenduse loomise õppematerjal Leaflet teegiga

Tallinn 2014

Sisukord

SISSEJUHATUS

Õppematerjal käsitleb kaardirakenduse loomist Leafleti teegiga. Eesmärgiks oli luua õppematerjal, mis annaks praktilisi oskuseid interaktiivse veebipõhise kaardi loomiseks Eesti ja Soome näitel. Õppematerjali peab olema ülevaatlik ja kompaktne ja valminud näide töötama ka mobiilsetel seadmetel ega tohi olla aeglane. Selgitaks, kuidas kaardiandmeid lihtsustada ja töödelda ning anda ülevaade Leafleti võimalustest ja tõsta üldist teadlikkust kaardirakenduste võimaluste kohta.

Õppematerjal on suunatud informaatika üliõpilastele ja eeldab algteadmisi programmeerimisest ja aine „Programmeerimise põhikursus“ läbimist. Suurem osa tööst puudutab Javascripti ja PHP-d. Materjalis käsitletatakse ka XML-ja JSON-failide struktuure ja seetõttu peaks olema sooritatud või sooritamisel aine „XML rakendused“.

Teine peatükk käsitleb, kuidas Leafletis GeoJSON faili kaardile kuvada, andmete filtreerimist ja kujundamist vastavalt objekti omadustele. Lisaks veel objekti omaduste muutmist vastavalt hiire sündmustele (*mouse Events*), erinevate GeoJSON failide sisu ühendamist ja kaardikihtide haldust.

Kolmandas peatükis tutvustatakse Maa-ameti WMS-teenuse võimalusi, uuritakse millised kaardid on serveris saadaval ja millistes formaatides neid alla saab laadida. Pärast seda toimub WMS-kihtide lisamine, kihtide haldamise lisamine, läbipaistvuse lisamine ja viimaseks *GetFeatureInfo* päringu loomine.

Õppematerjali lisas on ülesandeid ja näiteid, mis õppematerjali sisse ei mahtunud, ning mõeldud iseseisvaks jätkamiseks teema vastu huvi tundvatele inimestele. Käsitletakse objektide otsimist kaardilt, pindalade arvutamist, erinevate projektsioonide korraga kuvamise võimalusi ja WFS- ja WMS-teenuste kasutamist Soome näitel. Sellele lisaks veel *WebWorkers* kasutamine andmete laadimiseks teises lõimes (*thread*), mis võimaldab rakendust andmete laadimise ajal edasi kasutada.

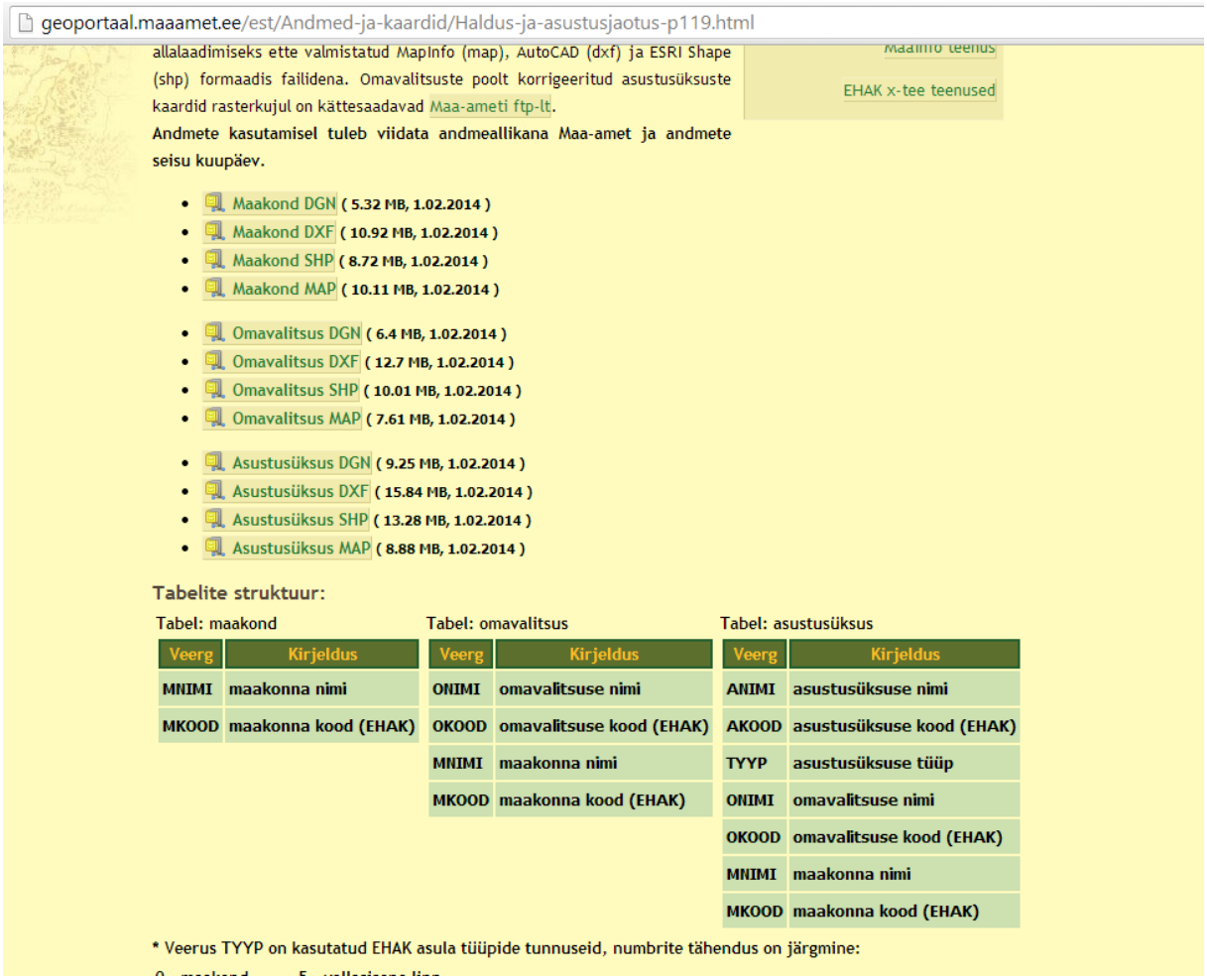
Õppematerjal sisaldab ka väga palju erinevaid faile ja valminud näiteid, mida on saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse vastavalt teemale.

1. ANDMETE ETTEVALMISTAMINE

Kõige enam kaardiandmeid on Eestis vabalt saadaval Maa-ameti kodulehel. Alla saab laadida erinevaid kaarte alates haldusjaotusest kuni mullakaartideni välja (Joonis 6).

Joonis 6 Maa-ameti kaardiserver

Haldus ja asutusjaotusele vajutades avaneb aken erinevate tarkvaraprogrammide jaoks mõeldud kaartidele (Joonis 7).



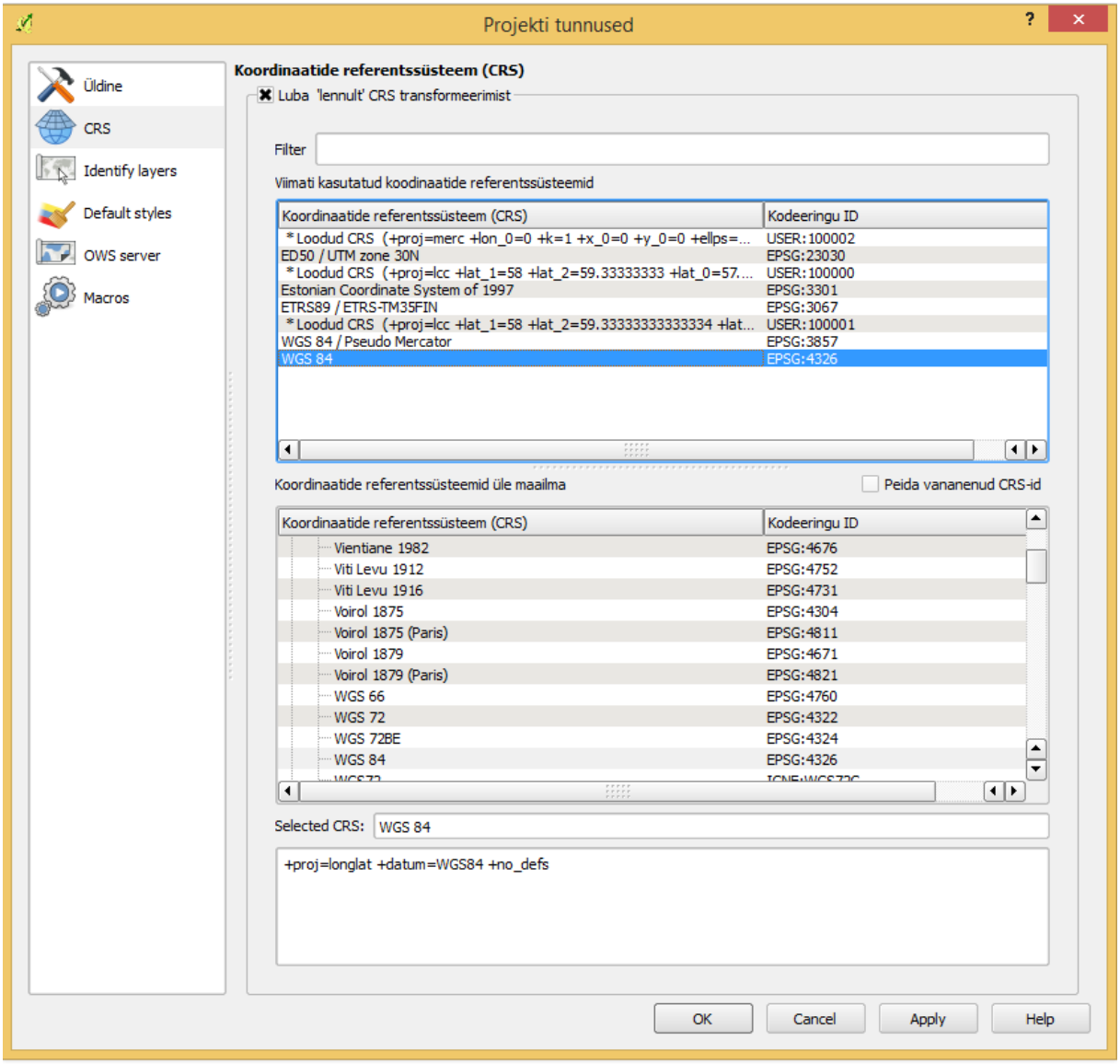
Joonis 7 Omavalitsuste kaardid

Alla tuleb laadida Omavalitsuse SHP fail ja see lahti pakkida. Tekkinud kaustas on näha *.DBF ja *.SHP faile. DBF faili on salvestatud andmetega koos käivad andmed andmebaasi kujul ja vajadusel saab seda SQL andmebaasi importida. Excelis avades on võimalik näha, milline info andmetega kaasas käib. SHP faili on salvestatud koordinaadid. Nende failide avamiseks tuleb alla laadida vabavaraline programm nimega QGIS aadressilt http://www.qgis.org/en/site/.

Enne kui QGIS-iga fail avada tuleb muuta sobivaks koordinaatide referentssüsteem, sest Leafleti GeoJson suudab lugeda vaid EPSG:4326 projektsioonis olevaid faile ja Maa-ametist saadud kaart on EPSG:3301 formaadis, seega tuleb enne QGIS-is lubada lennult transformeerimist. Vajutades nupule paremal all nurgas avaneb meile aken, kus saame muuta transformatsiooni (Joonis 8).

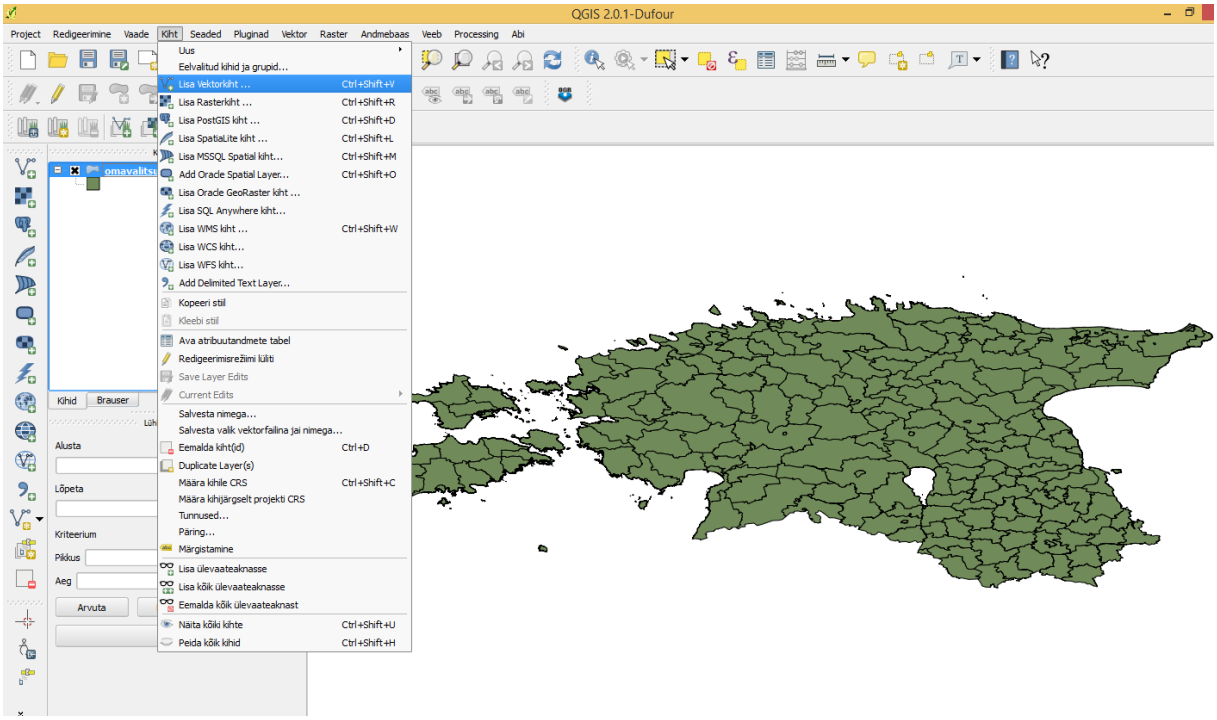
Joonis 8 Transformeerimise nupp

Sealt tuleb valida EPSG:4326 ja vajutada OK nupule (Joonis 9).



Joonis 9 'Lennult' transformeerimine

Pärast seda võib avada Maa-ametist tõmmatud kaardi valides Kiht-> Lisa vektorkiht ja valida alla laetud SHP faili asukoha. OK nupule vajutades avaneb Eesti kaart, mis on nüüd õigesse koordinaatide referentssüsteemi teisendatud (Joonis 10).

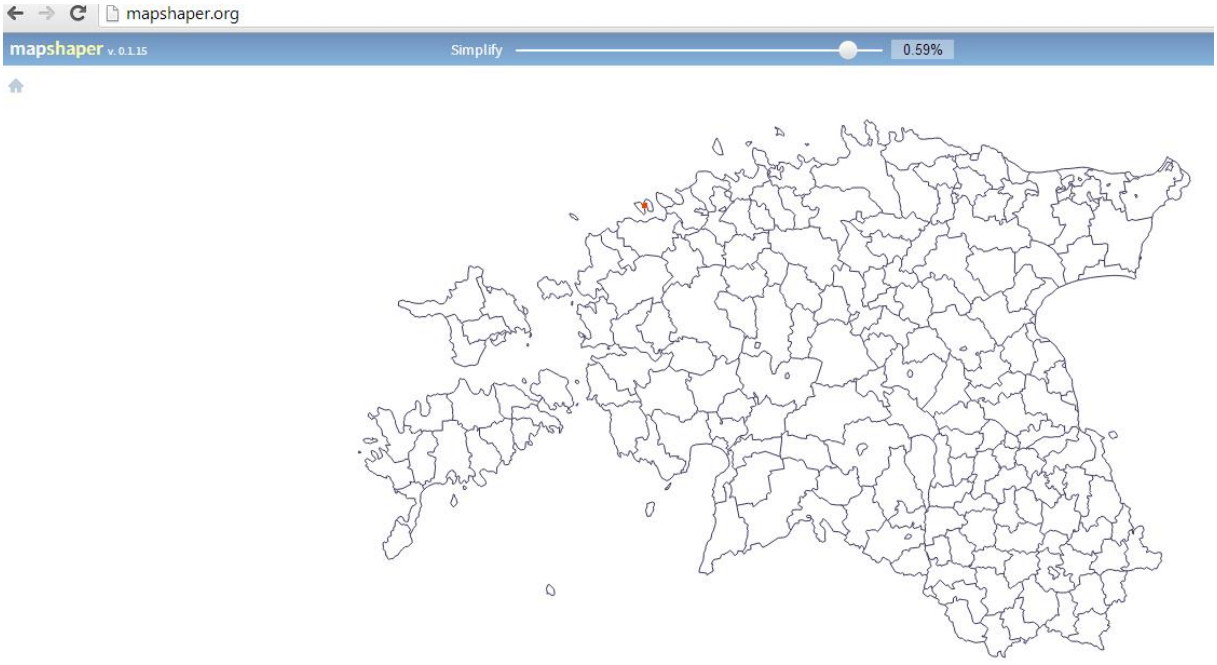


Joonis 10 Vektorkihi lisamine ja lisatud kihi välimus QGIS-is

Seejärel võiks faili kohe GeoJSON formaati salvestada, kuid fail tuleb ligikaudu 60 MB ja see on veebis töötamise jaoks liiga suur. QGIS-il on olemas funktsioon geomeetria lihtsustamiseks ja sellega on võimalik faili suurust vähendada, kuid sellega kaasneb probleem, et osade omavalitsuste vahele tekivad tühimikud. QGIS kasutab lihtsustamiseks Douglas ja Peucker algoritmi, ning piisavalt suure tolerantsiga lihtsustamise korral võivad omavalitsuste piirid nihkuda ja kaardile tekivad soovimatud tühimikud (Joonis 11).

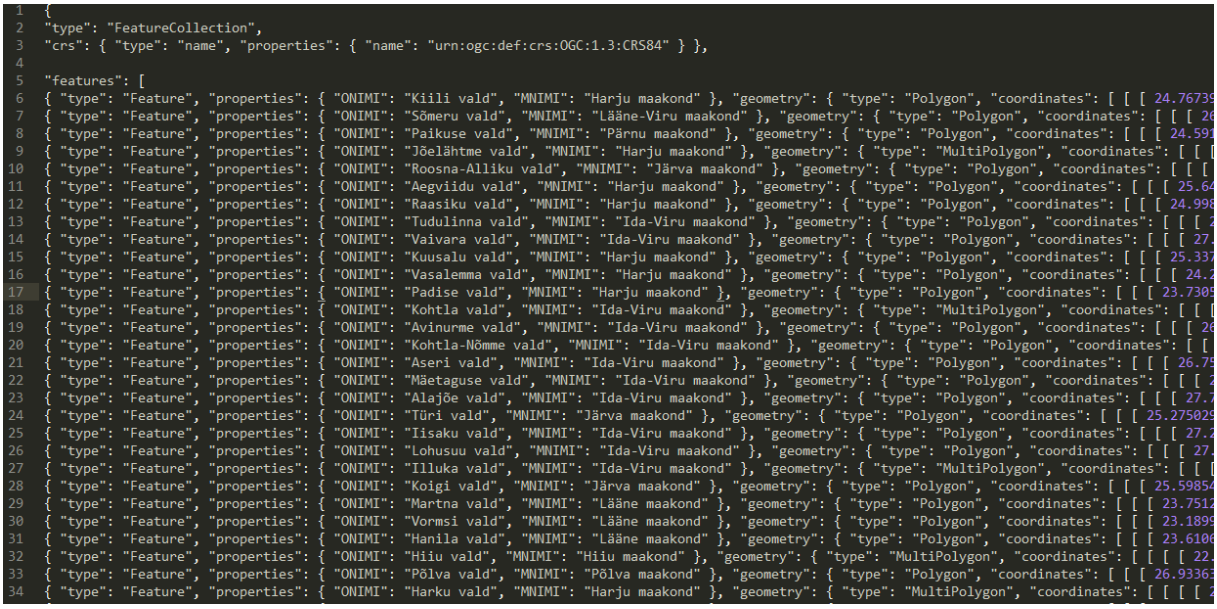
Joonis 11 punktide lihtsustamine QGIS-is ja tühimikud omavalitsuste vahel

Parem oleks kasutada modifitseeritud Visvalingam-i algoritmi, sest see suudab kõrvuti olevate objektide piire säilitada. Ainuke vabavaraline lehekülg, mis ma selle jaoks leidsin on MapShaper, (Bloch, MapShaper, 2013) (Joonis 12) kuhu oma õiges projitseeringus salvestatud SHP-faili üles laadides saame kohe visuaalselt näha kuidas seda lihtsustatakse. Lihtsustamise tolerantsiks peaks veebi jaoks valima kusagil 0,5-1%, vastasel juhul tuleb kas fail liiga suur veebis töötamiseks või kui 0,5% allapoole minna, siis hakkavad kaardilt kaduma linnad ja saared.



Joonis 12 Lihtsustamine Mapshaperit kasutades

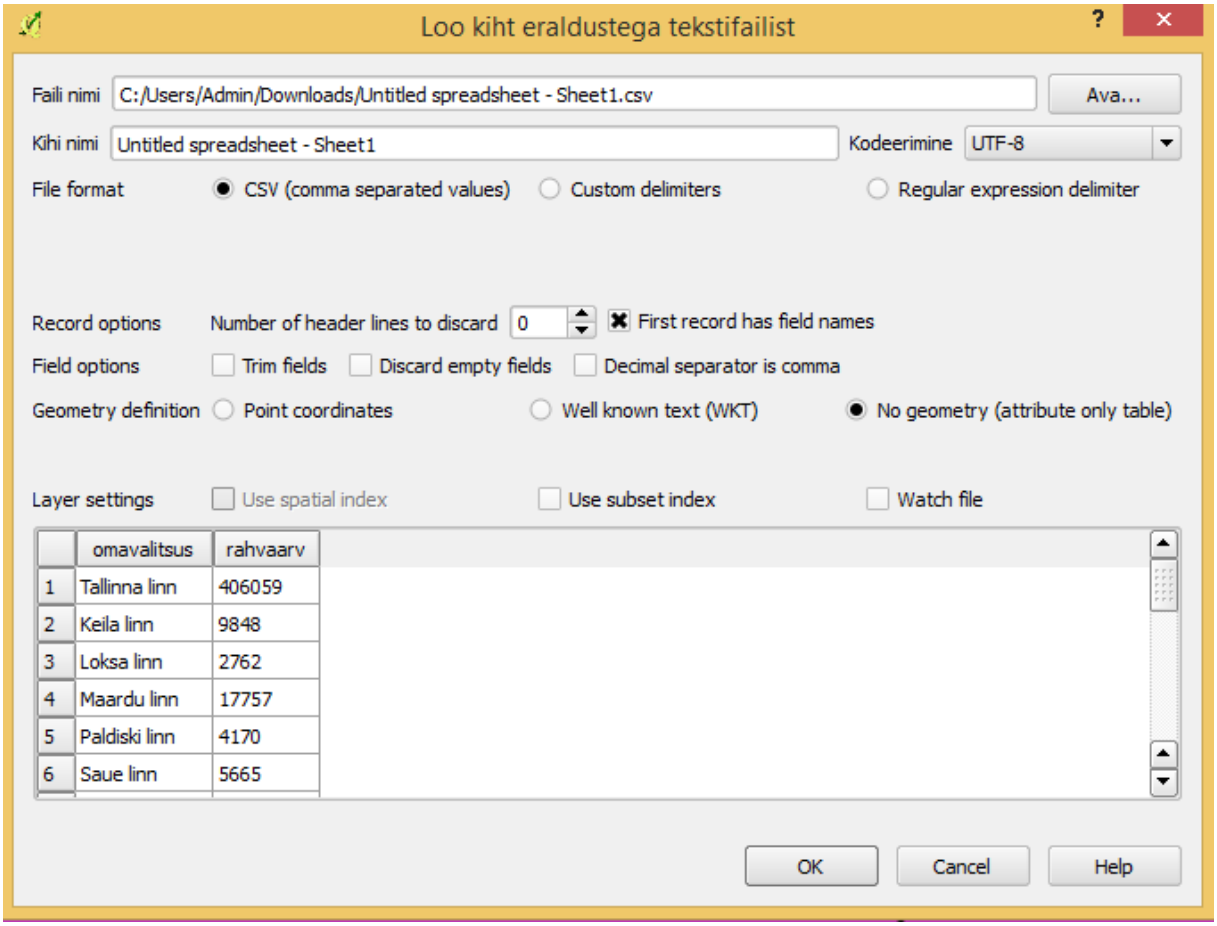
Alla tuleb laadida SHP fail, sest antud lehekülg kaotab lihtsustades ka kõik objekti omadused, kuid vähemasti ei jää enam omavalitsuste piiride vahele tühimikke. Allalaetud kaustast tuleb kustutada *.dbf fail ja asendada see Maa-ameti kodulehelt saadud *.dbf failiga. Avades shp faili uuesti QGIS-iga saab selle GeoJSON formaati salvestada valikust Kiht-> Salvesta nimega valikust GeoJSON formaat. Saadud GeoJSON fail tuli kõigest 537 KB ja seda avades on näha, et tegemist on Javascripti objektiga, mis teeb selle edaspidise töötlemise Javascriptis väga mugavaks. (Joonis 13)



Joonis 13 GeoJSON faili struktuur

Kui saadud andmed on ikka liiga suured võib Mapshaper.org saidilt alla laadida ka TopoJSON formaadis faili, mis asendab koordinaadid neile vastavate kaartega ja selle tulemusel väheneb andmemaht veelgi. TopoJSON formaati oskab lugeda Leafleti plugin nimega L.TopoJSON.

Kaardile teistsugust informatsiooni pannes, mis ei sisalda koordinaate on võimalik Statistikaameti kodulehelt tõmmata andmed rahvaarvu kohta omavalitsuste kaupa. Statistikaameti kodulehelt JSON faile päringuga küsida ei saa, ja alla tuleb laadida *.CSV fail. Seejärel tuleb importida QGIS-i valides Kiht-> Add delimited text layer (Joonis 14).



Joonis 14 CSV faili importimine QGIS-si

Seejärel tuleb vaadata, et andmed õigetesse tulpadesse läheks, ning vajutada ’No Geometry’ nuppu ja seejärel võib ka selle faili salvestada GeoJSON formaati.

2. GEOJSON FORMAADI KUVAMINE LEAFLETIS

Leafletil on sisseehitatud tugi GeoJSON formaadi kuvamiseks ja selle eelduseks on see, et andmed on EPSG:4326 projitseeringus. Lihtsaima näite jaoks võime kaardiandmed laadida HTML-i otse skriptina ja siis peame geojson faili ümber nimetama JSON failiks ning lisama failis kõige ette muutuja nime var omavalitsused. Kopeerime kaustast „leaflet“ kausta nimega geojson_algus.

Seejärel tuleb luua index.html fail ja lisada sinna HTML dokumendi kuvamiseks vajaminev info ja Leafleti kasutamiseks tuleb see skriptina laadida. (Koodinäide 9)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>GeoJSONi kasutamine</title>
    <link rel="stylesheet" href="css/style.css" />
    <link rel="stylesheet" href="css/leaflet.css" />
  </head>
  <body>
    <div id="map"></div>
    <script src="js/leaflet.js"></script>
    <script src="js/proj4-compressed.js"></script>
    <script src="js/proj4leaflet.js"></script>
    <script src="omavalitsused.json"></script>
    <script src="script.js"></script>
  </body>
</html>
```

Koodinäide 9 index.html faili sisu

Seejärel tuleks luua uus fail nimega script.js kus kõigepealt loome kihi, mille sisse laeme omavalitsused failist. (Koodinäide 10)

```
var myLayer = L.geoJson(omavalitsused);
```

Koodinäide 10 omavalitsuste laadimine failist

Pärast seda on võimalik luua kaart ja määrata vaikimisi nähtav kaardikiht (Koodinäide 11).

```
map = new L.map('map', {
      layers:[myLayer]
});
```

Koodinäide 11Kaardi loomine ja vaikimisi kihi lisamine

Seejärel on võimalik luua kihtide haldus kasutades L.LayerGroup funktsiooni ja lisades loodud kiht *Overlay* kihtide gruppi, et seda saaks sisse ja välja lülitada. Baaskihte saab lisada *baseLayers* alla ja neid välja lülitada ei saa (Koodinäide 12).

```
var theLayers = new L.LayerGroup([myLayer]);
var baseLayers = { };
var overlays = {
  "Omavalitsused": myLayer,
};
L.control.layers(baseLayers, overlays).addTo(map);
```

Koodinäide 12 Kihtide grupi loomine

Käsuga *fitBounds()* saab kaardile öelda millistes piirides see olema peaks ja *myLayer.getBounds()* tagastab kaardikihi piirid, sellisel viisil ei pea käsitsi suurenduse astet ja kaardi keskpunkti kaardi laadimisel määrama ja kaart mahutatakse piiridesse vastavalt andmetele (Koodinäide 13).

```
map.fitBounds(myLayer.getBounds());
```

Koodinäide 13 Kaardi piiride määramine

Kaardile on võimalik lisada ka meetermõõdustikus mõõtkava, kasutades objekti *control* funktsiooni *scale()*. (Koodinäide 14)

L.control.scale({imperial:false, maxWidth:250}).addTo(map);

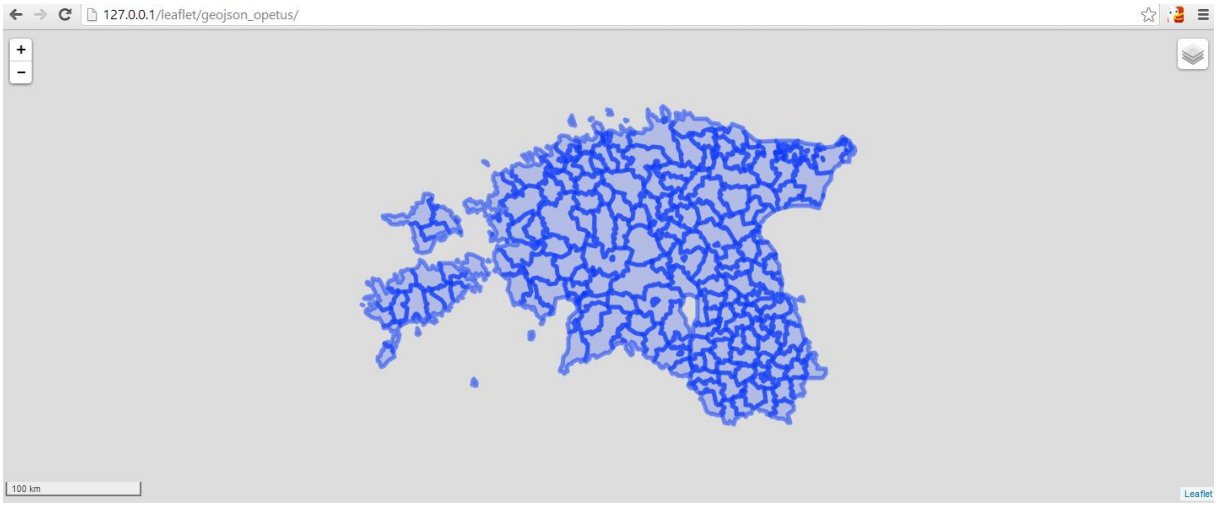
Koodinäide 14 Mõõtkava lisamine

Faili script.js kood peale muudatuste tegemist (Koodinäide 15).

```
var myLayer= L.geoJson(omavalitsused);
var theLayers = new L.layerGroup([myLayer]);

map = new L.map('map', {
     layers:[myLayer],
});
var baseLayers={};
var overLay={
     "Omavalitsused": myLayer,
}
L.control.layers(baseLayers, overLay).addTo(map);
map.fitBounds(myLayer.getBounds());
```

Koodinäide 15 script3.js faili sisu

Selle käivitamisel brauseris avaneb omavalitsuste kaart, kus kaart on paigutatud ekraani keskele ning võimalik on sisse ja välja suurendada. Kihtide nupule vajutades saab kaardil olevaid kihte sisse ja välja lülitada. (Joonis 15)



Joonis 15 Omavalitsuste kaart vektorkujul

2.1.GeoJSON objektide käsitlemine ja andmete filtreerimine

Nüüd on meil kaart vektorkujul olemas ja saame sinna erinevat funktsionaalsust juurde lisada. Omavalitsuste JSON failis on kõik omavalitsused eraldi objektidena ja L.GeoJSONil on olemas funktsionaalsus *onEachFeature* mida kasutades on võimalik lisada igale objektile *mouseListener*, mis hiire liikumise korral teada annab millise omavalitsuse kohal ollakse. (Koodinäide 16)

```
myLayer = L.geoJson(omavalitsused, {
  onEachFeature: function(feature, layer){
    var feat= feature.properties;

    var popupContent =  (feat.ONIMI +" " +feat.MNIMI);

    layer.on('mouseover',  function(e) {
      var popup = L.popup().setLatLng([e.latlng.lat+0.05,
            e.latlng.lng]).setContent(popupContent).openOn(map);
    });
  }
});
```

Koodinäide 16 Mousover funktsiooni lisamine

Selle käivitamisel näeme, et kui liikuda hiirega üle omavalitsuste, siis kuvatakse maakonna ja omavalitsuse nimi. Samuti võib sinna lisada ka filtreerimise reegleid, mis määravad millist värvi millise nimega objekte kujutatakse. Selle jaoks tuleb lisada funktsioon, mis tagastab vastava omavalitsuse värvi. (Koodinäide 17)

```
function getObjectColor(prop){
      return      prop.MNIMI == 'Harju maakond' ? "blue" :
                  prop.MNIMI == 'Jõgeva maakond' ? "blue" :
                  prop.MNIMI == 'Lääne-Viru maakond' ? "004020" :
                  prop.MNIMI == 'Ida-Viru maakond' ? "#FF8000" :
                  prop.MNIMI == 'Tartu maakond' ? "#FF8000" :
                  prop.MNIMI == 'Valga maakond' ? "blue" :
                  prop.MNIMI == 'Viljandi maakond' ? "004020" :
                  prop.MNIMI == 'Pärnu maakond' ? "blue" :
                  prop.MNIMI == 'Rapla maakond' ? "004020" :
                  prop.MNIMI == 'Saare maakond' ? "004020" :
                  prop.MNIMI == 'Hiiu maakond' ? "blue" :
                  prop.MNIMI == 'Lääne maakond' ? "#FF8000" :
                  prop.MNIMI == 'Järva maakond' ? "#FF8000" :
                  prop.MNIMI == 'Põlva maakond' ? "004020" :
                  "#FF8000";
      }
```

Koodinäide 17 Funktsioon värvi tagastamiseks vastavalt maakonna nimele

Seejärel tuleb teha teine funktsioon, mis siis vastavalt värvile omavalitsuse ära värvib (Koodinäide 18).

```
function defaultStyle(feature) {
            return{
              weight: borderWeight,
              opacity: 1,
              fillOpacity: 0.7,
              color: 'black',
              fillColor: getObjectColor(feature.properties)
          };
}
```

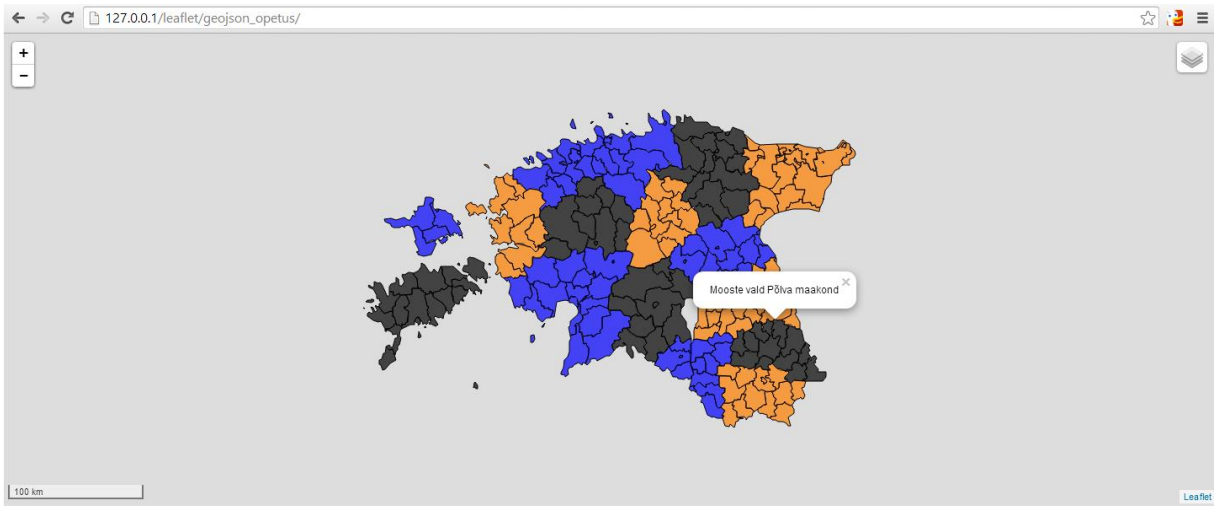
Koodinäide 18 Stiili lisamine

OnEachFeature alla tuleb lisada rida *defaultStyle* väljakutsumiseks (Koodinäide 19).

```
layer.setStyle(defaultStyle(feature));
```

Koodinäide 19 Stiili määramine

Tulemuseks on kaart, kus kõik maakonnad on värvitud erinevat värvi (Joonis 16).



Joonis 16 Maakondade värvimine filtritega

2.2.Erinevates GeoJSON failides oleva sisu ühendamine

Järgmisena võime index.html faili lisada skriptina Statistikaameti rahvaarvu sisaldava JSON faili ja see tuleb lisada enne script3.js faili. Rahvastik.json fail tuleb samuti muutujaks teha lisades kõige ette „*var rahvastik=*“ (Koodinäide 20).

```
    <script src="rahvastik.json"></script>
```

Koodinäide 20 Rahvastiku andmete lisamine

Kuna mõlemad failid on meil laetud *Javascripti* objektidena saame neid mõlemaid lihtsa *for*-tsükliga läbida ja kirjutada rahvastiku failis olevad rahvaarvud nendele vastavate omavalitsuste järele (Koodinäide 21). Siinkohal võib märkida, et omavalitsusreformi tõttu on osad omavalitsused liitunud ja uuemate omavalitsuste rahvaarvu Statistikaametil veel pole.

```
for(var i in omavalitsused.features){
  for(var j in rahvastik.features){
    if(omavalitsused.features[i].properties.ONIMI==
            rahvastik.features[j].properties.omavalitsus){
      omavalitsused.features[i].properties.Rahvastik =
            rahvastik.features[j].properties.rahvaarv;
    }
  }
}
```

Koodinäide 21 JSON failide sisu ühendamine

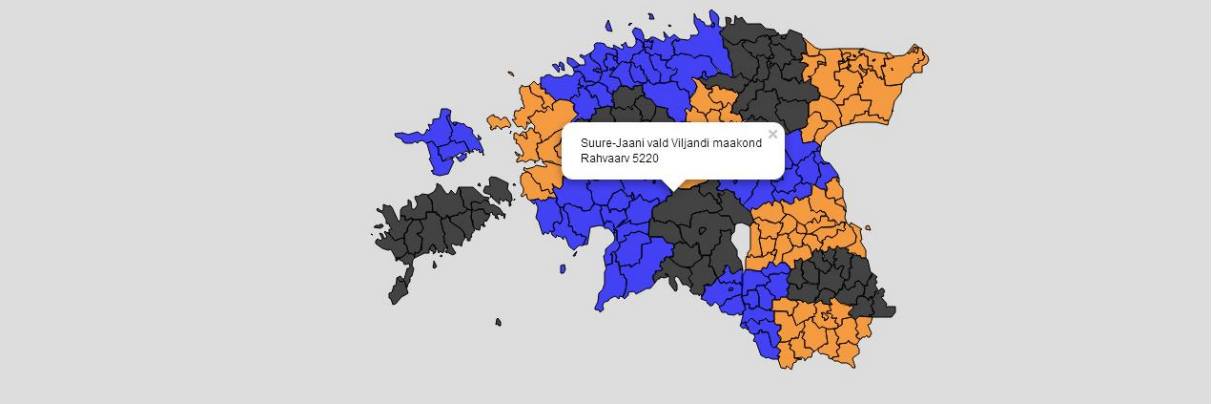
Seejärel piisab vaid *onEachFeature* juures *popupContent* muutmisest, et hiirega peale liikudes oleks näha ka rahvaarvu (Koodinäide 22).

```
    var popupContent =  (feat.ONIMI +" " +feat.MNIMI+" </br> Rahvaarv
"+ feat.Rahvastik);
```

Koodinäide 22 Rahvaarvu kuvamine hüpikaknal

Lõplik faili script.js kood asub kaustas leaflet/geojson_lopp kaustas.

Lõplik pilt valminud näitest koos rahvaarvuga (Joonis 17).



Joonis 17 Info kuvamine koos rahvaarvuga

Ülesanne 1

Lisa kaardile funktsioon, mis muudab hiire all oleva omavalitsuse punaseks ja hiire väljaliikumisel taastab endise värvi.

Probleemide korral leiab lõpliku näite leiab leaflet/geojson_lopp kaustast.

3. WMS MAA-AMETI SERVERIGA

WMS ehk *Web Map Service* võimaldab teha veebiserverisse päringuid ja saadab vastu pildi küsitud piirkonnast. Maa-ameti WMS-serveri aadress on http://kaart.maaamet.ee/wms/alus?. WMS serveri võimaluste ja kaartide teadasaamiseks tuleb teha *GetCapabilities* päring kujul:

http://xgis.maaamet.ee/wms-pub/alus?version=1.1.1&service=WMS&request=GetCapabilities (Maaamet, 2012)

Vastuseks tuleb pikk xml faili serveri võimalustest nagu toetatavad pildifailiformaadid ja serveris olevate kaartide nimed ning nende omadused. Esimene vajalik osa on näha millises vormingus pilte saame serverist pärida, selle jaoks otsime failist üles <GetMap> elemendi ja sealt on näha, et toetatakse png, jpeg, gif ja tiff faile. (Koodinäide 23) Veebis olevate kaartide jaoks on png formaat parim, sest see on väikese mahuga ja toetab läbipaistvust.

```
<GetMap>
      <Format>image/png</Format>
      <Format>image/png; mode=24bit</Format>
      <Format>image/png; mode=32bit</Format>
      <Format>image/jpeg</Format>
      <Format>image/gif</Format>
      <Format>image/vnd.wap.wbmp</Format>
      <Format>image/tiff</Format>
      <DCPType>
        <HTTP>
          <Get><OnlineResource xmlns:xlink=
                  "http://www.w3.org/1999/xlink" xlink:href=
                  "http://xgis.maaamet.ee/wms-pub/alus?"/></Get>
          <Post><OnlineResource xmlns:xlink=
                  "http://www.w3.org/1999/xlink" xlink:href=
                  "http://xgis.maaamet.ee/wms-pub/alus?"/></Post>
        </HTTP>
      </DCPType>
    </GetMap>
```

Koodinäide 23 GetMap parameetrid (Maaamet, 2012)

Järgmisena on vaja teada saada missuguseid kaarte server pakub ja millised on nende projektsioonid ja piirid. Seda on võimalik samas xml failis allapoole kerides <Layer> elemendi sees (Koodinäide 24). Selle sisust on näha, et kihi nimi on of10000 ja kasutatav koordinaatide referentssüsteem on EPSG:3301.

```
<Layer>
    <Title>Ortofoto</Title>
    <Layer queryable="0" opaque="1" cascaded="0">
        <Name>of10000</Name>
        <Title>ORTOFOTO</Title>
        <SRS>EPSG:3301</SRS>
        <LatLonBoundingBox minx="21.5928" miny="57.4533"
                  maxx="28.2755" maxy="59.8517" />
          <BoundingBox SRS="EPSG:3301"
                minx="365000" miny="6.375e+06"
                maxx="740000" maxy="6.635e+06" />
        <ScaleHint min="0" max="8.98399304309669" />
    </Layer>
```

Koodinäide 24 Kihi parameetrid (Maaamet, 2012)

Kõikidel kihtidel on piirid nimega BoundingBox, mis antud näite korral on 365000, ja see number on selline just kasutatava projektsiooni tõttu. Leaflet aga ei toeta sellist projektsiooni ja seetõttu tuleb selle kasutamiseks kasutada pluginat nimega Proj4leaflet, mis teeb kaardi vaatamise võimalikuks ka Leafletiga. BoundingBoxi minimaalseid ja maksimaalseid väärtuseid ületades saadab server meile kas musta pildi või veateate. Veateade tuleb samuti xml failina ja sisaldab tavaliselt kas infot, et piire on ületatud või teatab üsna eksitavalt et selline kiht puudub sootuks (Koodinäide 25) .

```
<?xml version='1.0' encoding="UTF-8" standalone="no" ?>
<!DOCTYPE ServiceExceptionReport SYSTEM
"http://schemas.opengis.net/wms/1.1.1/exception_1_1_1.dtd">
<ServiceExceptionReport version="1.1.1">
<ServiceException code="LayerNotDefined">
msWMSLoadGetMapParams(): WMS server error. Invalid layer(s) given in
the LAYERS parameter.
</ServiceException>
</ServiceExceptionReport>
```

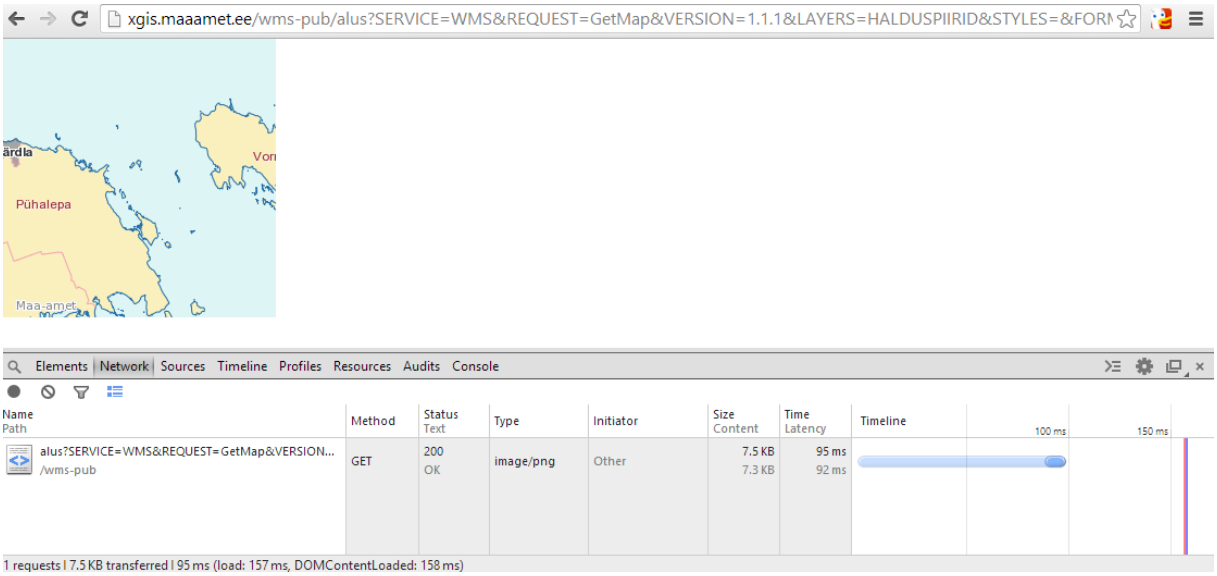
Koodinäide 25 Veateate kood

Eduka päringu jaoks tuleb saata serveri poolt toetatav versioon, kaardikihi nimi, pildi formaat, pildi kõrgus ja laius, kaardi SRS ja *Bounding Boxi* neli väärtust (Koodinäide 26).

```
xgis.maaamet.ee/wms-
pub/alus?SERVICE=WMS&REQUEST=GetMap&VERSION=1.1.1&LAYERS=HALDUSPIIRID
&STYLES=&FORMAT=image%2Fpng&TRANSPARENT=true&HEIGHT=256&WIDTH=256&CON
TINUOUSWORLD=true&SRS=EPSG%3A3301&BBOX=425983.99999999645,6520831.999
999799,458751.9999999978,6553599.999999814
```

Koodinäide 26 Näidispäring

Brauseriga sellist päringut Maa-ameti serverisse tehes saab vastuseks png faili küsitud asukohast. Vastuseks ei saadeta tervet Eesti kaarti, vaid 256 x 256 pikslit mõõdus pildi. (Joonis 18)



Joonis 18 Näidispäringu vastus

Leaflet oskab õnneks ise vaadata millist tükki tuleb millise suurenduse astme juures laadida ja iga kaardi laadimisega tõmmatakse POST või GET meetodiga umbes 20 sellist tükki. WMS-i kasutamiseks tuleb Leafletiga luua TileLayer ja anda serveri aadress ja vastavad parameetrid, et see oskaks neid Maa-ameti serverist laadida. Nagu näha, siis siin pole koordinaate ega mõõtmeid vaja anda, sest seda teeb Leaflet automaatselt. Kohustulikud parameetrid on siin vaid kihi nimi ja failiformaat mida soovime saada. Kõik mis algab L. tähistab Lealeti ja .addTo(map) tähendab, et kiht lisatakse kaardile ja tehakse nähtavaks . (Koodinäide 27)

```
var aluskaart= L.tileLayer.wms('http://kaart.maaamet.ee/wms/alus', {
  layers: 'HALDUSPIIRID',
  format: 'image/png',
  maxZoom: 14,
  minZoom: 3,
  continuousWorld: true,
  attribution: 'Maa-amet 2014',
  transparent:true,
  opacity:opacity,
});
```

Koodinäide 27 Maa-ameti WMS kihi kuvamise näide

Maa-ameti kaardid on EPSG:3301 KRS-iga aga Leaflet suudab töötada vaid EPSG:3857-ga. Selleks, et kaarti näha tuleb kasutada Leafleti pluginat nimega Proj4Leaflet, mis transformeerib kaardid lennult õigesse projitseeringusse. Esimene parameeter näitab millises projitseeringus kaardid tulevad, teine näitab kuidas koordinaate ümber arvutada ja resolutsioonid näitavad milliste suurendusastmete korral kihid nähtavad on (Koodinäide 28). Antud muutuja tuleb script.js failis kõige ette lisada, sest kaardi transformeerimine peab toimuma enne kihi laadimist.

```
var crs = new L.Proj.CRS('EPSG:3301','+proj=lcc
+lat_1=59.33333333333334 +lat_2=58 +lat_0=57.51755393055556 +lon_0=24
+x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m
+no_defs',
  {
    resolutions: [
      8192, 4096, 2048, 1024, 512, 256, 128,
      64, 32, 16, 8, 4, 2, 1, 0.5
    ], origin: [0, 0]
  });
```

Koodinäide 28 Projektsiooni muutmine

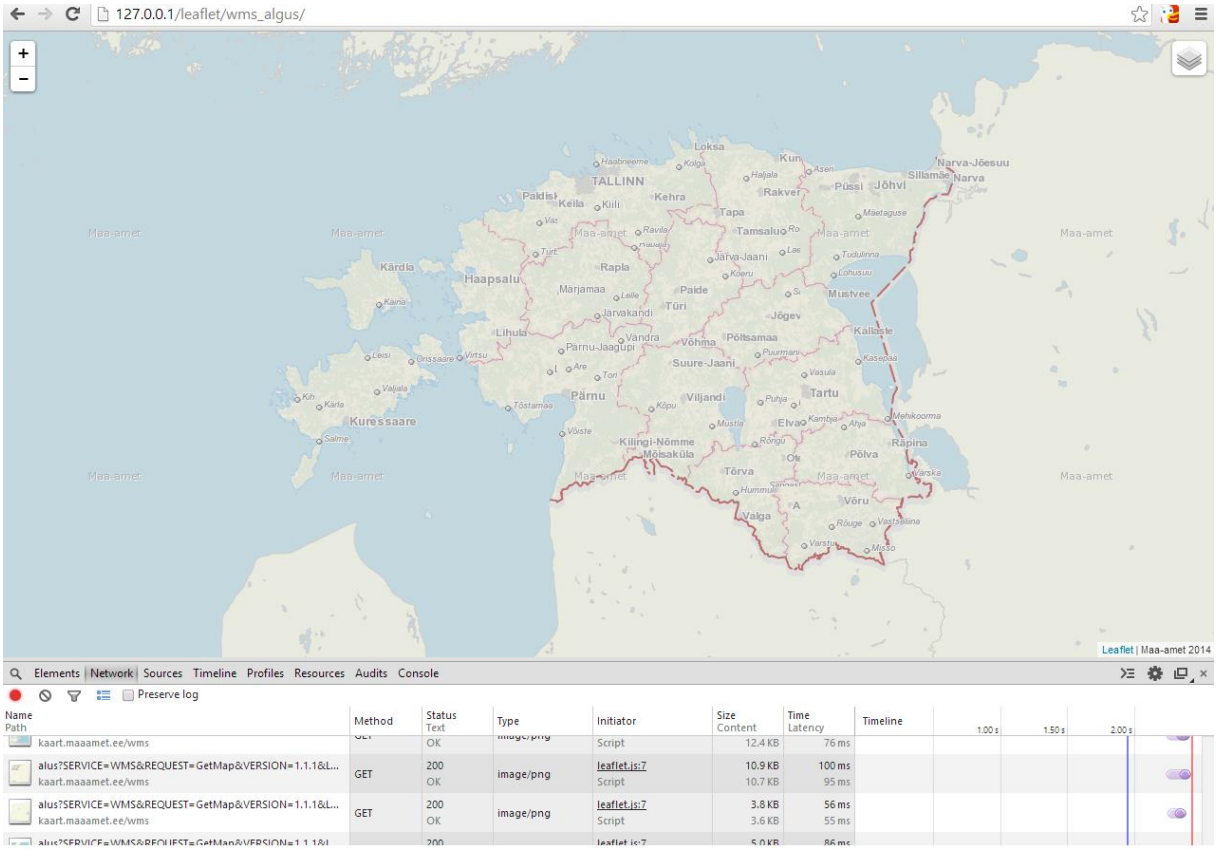
Eelnevat teades on võimalik lihtsaim WMS-kihtidega kaart valmis teha. Selleks tuleb GeoJSON näites alustatud Index.html faili sisusse lisada proj4-compressed.js ja proj4leaflet.js failid ning index.html fail peab välja nägema selline :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>WMS-i kasutamine</title>
    <link rel="stylesheet" href="css/style.css" />
    <link rel="stylesheet" href="css/leaflet.css" />
  </head>
  <body>
    <div id="map"></div>
    <script src="js/leaflet.js"></script>
    <script src="js/proj4-compressed.js"></script>
    <script src="js/proj4leaflet.js"></script>

    <script src="omavalitsused.json"></script>
    <script src="rahvastik.json"></script>
    <script src="script.js"></script>
  </body>
</html>
```

Joonis 19 Index.html faili sisu

Script.js faili tuleb lisada kõige ette eelpool(Koodinäide 28) mainitud CRS ja seejärel aluskaardi kiht. Probleemide tekkimise korral saab töötavat koodi vaadata ’script1.js’ failist

Nüüd peaks faili brauseriga avades nägema juba täiesti töötavat WMS kaarti nagu (Joonis 18). Võrgu osa vaadates näeme milliste päringutega pildid alla laeti ja nende formaati ning suurust. Kihtide nupule vajutades saab kihte sisse ja välja lülitada.



Joonis 20 Maa-ameti WMS kaart Leafletis

Ülesanne 2 Proovi lisada Maa-ameti serverist veel üks kiht nimega ’of10000’, lisa see *baseLayerite* alla ja seadista minZoom 11 peale ja maxZoom 14 peale. Proovi kas tehtu hakkab tööle.

Lahendust saab näha Koodinäide 29.

```
var orto= L.tileLayer.wms('http://xgis.maaamet.ee/wms-pub/alus', {
  layers: 'of10000',
  format: 'image/png',
  maxZoom: 14,
  minZoom: 11,
  continuousWorld: true,
  attribution: 'Maa-amet 2014',
  opacity:1,
});
Var baseLayers={ "Ortofoto zoom 11-14": orto}
```

Koodinäide 29 Ortofoto kihi lisamine

3.1. Erinevate kihtide haldamine

Kui soovida korraga mitut erinevat kaardikihti kasutada või neid vahetada, siis on kõige parem kasutada L.LayerGroup() funktsiooni. Selliselt on võimalik moodustada kihtide rühm ning neid kaardile lisada ning sealt eemaldada. Kihtide rühma lisamine ei tähenda, et need kaardile lisatakse, vaid tekitatakse eraldi paneel, kust saab kihte sisse ja välja lülitada (Koodinäide 30).

```
var theLayers = new L.LayerGroup([pohi, baas,
      topokataster, aluskaart]);
var baseLayers = {
    "Ortofoto zoom 11-14": orto,
  };

var overlays = {
  "Aluskaart:3-14": aluskaart,
  "Pohikaart zoom:11-14": pohi,
  "Baaskaart zoom:7-10": baas,
  "Katastrid tüüpide kaupa zoom:11-14": topokataster
};

L.control.layers(baseLayers, overlays).addTo(map);
```

Koodinäide 30 Kihtide kontrollimine

L.control.layers lisab kihtide haldamise kaardile, kuid mitte ühtegi kaardikihti. Vaikimisi kaardikihi lehe käivitumisel saab määrata kaardi parameetrite alt, kus layers all saab öelda millist kihti kohe näidatakse (Koodinäide 31).

```
var map = new L.Map('map', {
    continuousWorld: true,
    worldCopyJump: false,
    crs: crs,
    layers:[ aluskaart],
  });
```

Koodinäide 31 Vaikimisi näidatava kihi määramine

Kuna mitme kihi korral ei paista kihid läbi võib kihtidele lisada funktsiooni läbipaistvuse muutmiseks. Mõistlik oleks sedasama Leafleti kihtide vahetamise akent kasutada, kuid see juba ’kuulab’ hiireklikki ja teist lisada on ebamõistlikult keeruline. Seega tuleb lisada HTML-i read, et muuta kihtide läbipaistvust (Koodinäide 32).

```
<div class="opacity" id="opacity" >Läbipaistvus
      Piirid <input type='range' name='HALDUSPIIRID' min='1'
            max='100' value='100' onchange='changeOpacity(this.value,
            this.name)'>
      Põhikaart <input type='range' name='pohi_vr2' min='1'
            max='100' value='100' onchange='changeOpacity(this.value,
            this.name)'>
      Baaskaart <input type='range' name='BAASKAART' min='1'
            max='100' value='100' onchange='changeOpacity(this.value,
            this.name)'>
      Katastrid <input type='range' name='TOPOYKSUS_6569'
            min='1' max='100' value='100'
            onchange='changeOpacity(this.value, this.name)'>
      <div id="zoom">Zoom level: 6</div>
</div>
```

Koodinäide 32Index.html faili sisu

Script.js faili tuleb lisada funktsioon, mis muudab kihi läbipaistvust ja kuna kaardikihid on nähtavad vaid teatud suurendusastmete juures, siis võiks kasutajale ka näidata millise suurendusastme juures hetkel ollakse ning joonlaua kilomeetrites (Koodinäide 33).

```
function changeOpacity(val, name){
  opacity=val/100;
  var theLayer=theLayers.getLayers();
  for (var i in theLayer) {
    var layerName;
    try{
      layerName=theLayer[i].wmsParams.layers;
    }catch(error){}

    if(layerName==name){
      theLayer[i].setOpacity(opacity);
    }
  };
}

map.on("zoomend", function(){
    zoomLev = map.getZoom();
    document.getElementById("zoom").innerHTML="Zoom level: "+zoomLev;
});
L.control.scale({imperial:false, maxWidth:250}).addTo(map);
```

Koodinäide 33 Funktsioon kihi läbipaistvuse muutmiseks ja kihi suurenduse teavitamiseks

3.2. WMS getFeatureInfo service

Maa-ameti wms server pakub ka võimalusi teha andmepäringuid erinevatelt kihtidelt. GetCapabilities XML failist näeme et igale kihile on antud muutuja *queryable*, mis näitab seda kas kihilt on võimalik andmeid küsida või mitte (Koodinäide 34).

```
<Layer queryable="1" opaque="1" cascaded="0">
```

Koodinäide 34 Päringut teha võimaldavad kihid

Kokku on maa-ameti kaartide hulgas vaid kaks kaarti, millelt on võimalik päringuid teha. Ühel on kõik katastriinfoga seonduv ja teine on metaandmete kiht. GetFeatureInfo on GetMap päringuga üsna sarnane ja lisanduvad kihi nimi ja koordinaadid mille järgi andmeid pärida (Koodinäide 35).

```
http://xgis.maaamet.ee/wms-pub/alus?
      VERSION=1.1.1&
      REQUEST=GetFeatureInfo&
      service=WMS&
      version=1.1.1&
      layers=TOPOYKSUS_6569&
      styles=&
      srs=EPSG%3A3301&
      format=image%2Fpng&
      bbox=513024.0000000007,6487551.999999784,513536.00000000076,648
      8063.999999784&
      width=256&
      height=256&
      query_layers=TOPOYKSUS_6569&
      info_format=text/plain&
      feature_count=50&
      x=353&
      y=145&
      exceptions=application%2Fvnd.ogc.se_xml
```

Koodinäide 35 GetFeatureInfo päring.

Leafletil ei ole sisseehitatud funktsiooni GetFeatureInfo päringu tegemiseks ja selle jaoks tuleb teha ise AJAX päring. Kuna javascripti AJAX päring ei ole mõistlik teise domeeni sooritada tuleb päringu osa teha PHP-ga. Selle jaoks tuleb luua samasse kausta fail nimega „get.php“ ja kopeerida järgnev sisu (Koodinäide 36).

```
<?php
if(isset($HTTP_RAW_POST_DATA))
{
      header('Content-Type: text/html; charset=utf-8');
      $data=json_decode(stripslashes($HTTP_RAW_POST_DATA),
            true);
      $data = file_get_contents($data['data']);
      echo(utf8_encode($data));
}
?>
```

Koodinäide 36 Päringu tegemiseks vajaliku get.php faili sisu

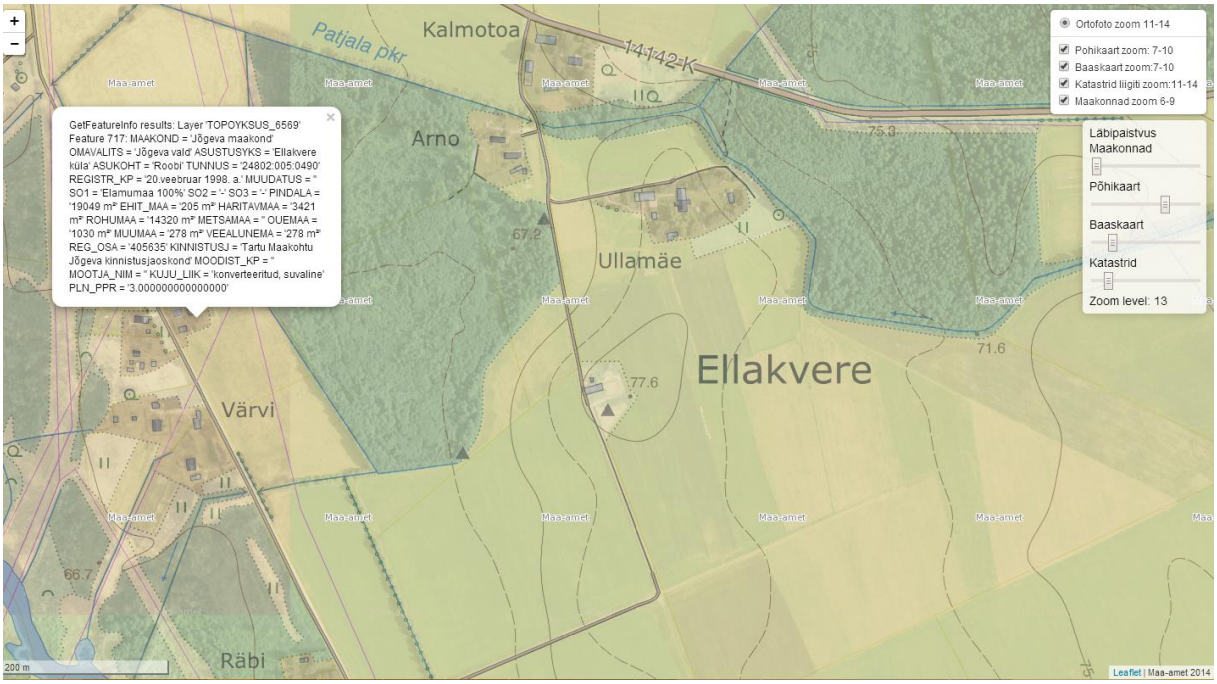
Faili script.js tuleb lisada funktsioonid getFeatureInfo saamiseks. Alustuseks peab ’kuulama’ hiireklikki ja seejärel hankima klikitud punkti koordinaadid, ning siis saab teha päringu AJAX-it kasutades, kuvades saadud andmed hüpikaknaga ekraanile (Koodinäide 37)((Clarck, 2013))

```
map.on('click', function(e, layer){
  var info = getFeatureInfoUrl(e.latlng);
  var url= 'get.php';
  var xhr=new XMLHttpRequest();
  xhr.open("POST", url, true);
  var data= {data: info};

  xhr.setRequestHeader("Accept","text/plain");
  xhr.setRequestHeader('Content-Type','application/json;
           charset=UTF-8');
  xhr.send(JSON.stringify(data));

  xhr.onloadend = function () {
      makePopup(e.latlng, xhr.responseText);
  };

  xhr.onerror = function () {
     console.log("Laadimine ebaõnnestus!");
  };
});
function makePopup(coords, text){
  var popup =
L.popup().setLatLng(coords).setContent(text).openOn(map);

}

function  getFeatureInfoUrl (latlng) {
        size = map.getSize();
        var futureBounds = map.getBounds();
        var crs = map.options.crs;
        var seLatLng = futureBounds.getSouthEast();
        latlng= crs.project(latlng);
        var se = crs.project(seLatLng);
        var bbox = ''+latlng.x+','+se.y+',
            '+se.x+','+latlng.y+'';
        url= 'http://xgis.maaamet.ee/wms-pub/alus?';
        params = {
          request: 'GetFeatureInfo',
          service: 'WMS',
          version:'1.1.1',
          layers: 'TOPOYKSUS_6569',
          srs: 'EPSG:3301',
          format: 'image/png',
          bbox: bbox,
          height: size.y,
          width: size.x,
          x: '10',
          y: '10',
          query_layers: 'TOPOYKSUS_6569',
          info_format: 'text/plain'
        };
    return url + L.Util.getParamString(params, url, true);
  };
```

Koodinäide 37 GetFeatureInfo päringu jaoks vajalikud funktsioonid

Lõplik tulemus peaks välja nägema selline, kus saab mitmel kihil olevaid andmeid korraga vaadata ja klikkides avaneb aken katastri tunnustega. Nagu näha allpool olevalt

pildilt pakuvad Maa-ameti kaardid Eesti kohta oluliselt rohkem infot kui Google või Bingi omad, kõige kasulikumaks võib pidada just talunimede ja katastritunnuse näitamist. (Joonis 21)



Joonis 21 Lõplik näide

Lõplik faili script.js kood asub kaustas leaflet/wms_lopp.

LISA 1 ISESEISVAD HARJUTUSI

- Proovi objektide otsimist ja nende peale sisse *zoomimist* vastavalt otsingusõnale. Näide asub aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet/geojson/with_functions.html

- Proovi objektide pindala leida kasutades objektide koordinaate Näide asub aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet/geojson/with_functions.html

- Proovi andmete laadimist *WebWorkerit* kasutades. Näide asub aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet/geojson/with_functions.html

- Proovi objektide lihtsustamist. Näide asub aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet/simplify

- Proovi teha päringuid WFS-serverist. Näide asub Näide asub aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet/WFS+WMS

LISA 2 LIHTSUSTAMINE KASUTADES SIMPLIFY.JS MOODULIT

Töötav näide on aadressil http://www.tlu.ee/~elariroo/simplify/simplify.html.

Näide vajab töötamiseks ka leflet.js, simplify.js ja GeoJSON faili koos andmetega ja konsooli aknas näidatakse ka lihtsustamise tulemusi.

LISA 3 ERINEVATE KAARDIRAKENDUSTE VÕRDLUS

Tumedad ruudu tähistavad täielikult toetatud funktsionaalsust, heledad aga osalist tuge.