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 ka asjaolu, et Javascript töötab kõikidel levinud brauseritel operatsioonisüsteemist sõltumata, mis tähendab seda, et ühes keeles kirjutatud rakendus toimib peaaegu kõikidel seadmetel.

Google Maps, OpenLayer ja Leaflet on suurimad Javascriptil põhinevad raamistikud ja autor valida viimase, sest see on vabavaraline, kiire ja uudne kasutades näiteks *CSS3* ja *HTML5* võimalusi. Leafletiga puutus autor kokku konverentsil TopConf, kus raamistiku looja Vladimir Agafonkin tutvustas selle võimalusi

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 näitel, olles samaaegselt ülevaatlik ja kompaktne. Selgitaks kuidas kaardiandmeid lihtsustada ja töödelda, anda ülevaade Leafleti võimalustest ja kaardiandmete ühendamisest teiste andmetega.

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 ettevalmistamise protsessi ning uuritakse millistest allikatest andmeid otsida, milleks neid transformeerida ja lihtsustada. Näidatakse erinevaid võimalusi andmete töötlemiseks ning teisendamist vajalikesse vormingutesse.

Kolmandas peatükis käsitletakse 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.

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

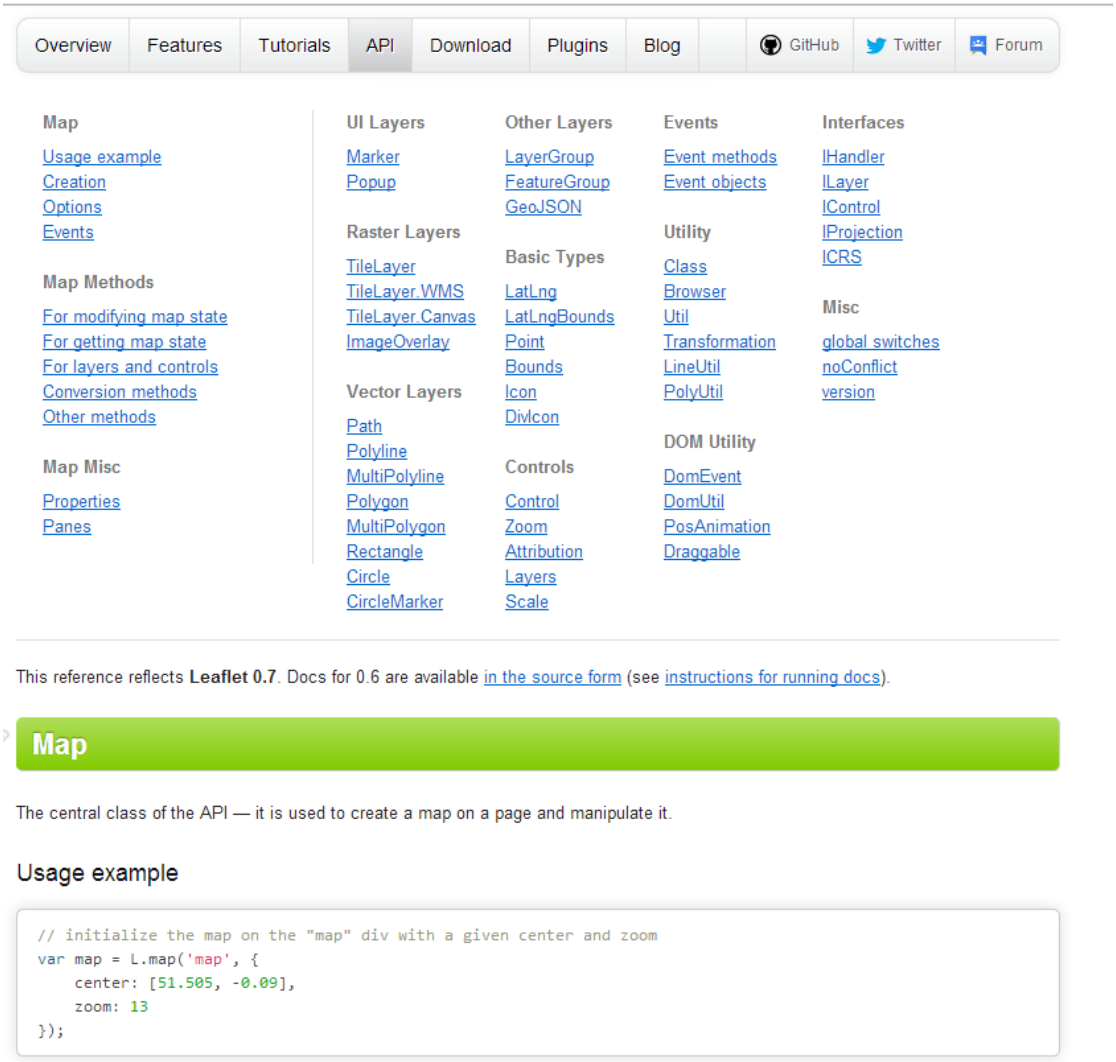
1. LEAFLETI TUTVUSTUS

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.



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 lisamoodulid

Leafleti teeb mitmekülgseks modulaarne ülesehitus, mis tähendab, et teatud funktsionaalsuse puudumisel on võimalik kasutada või ise luua erinevaid

lisamooduleid. Neid on Leafleti kodulehel palju ja siinkohal mõned tähtsamad koos selgitustega:

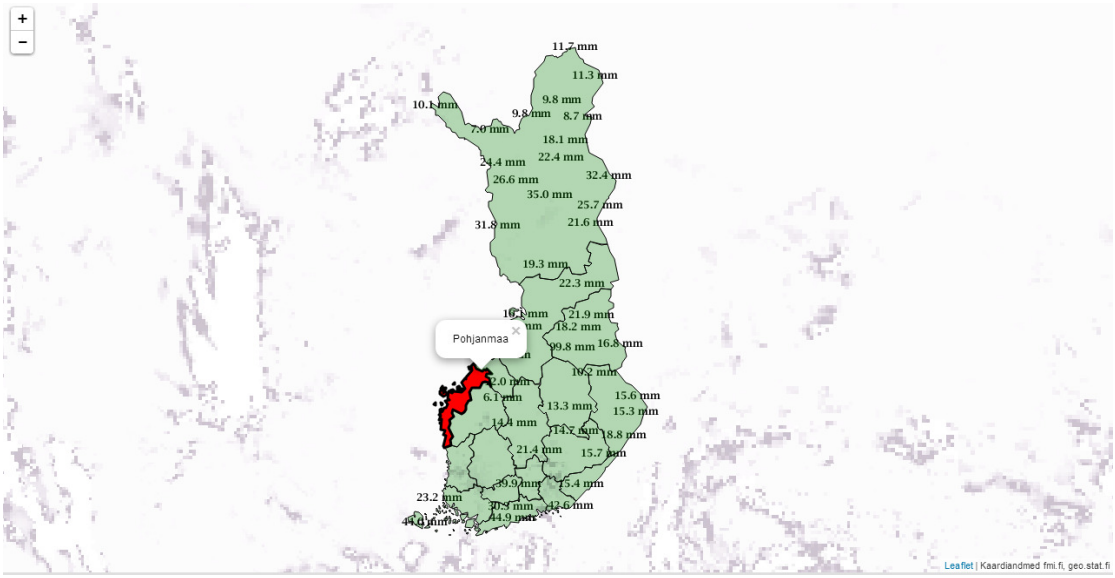
- Proj4Leaflet (Kartena AB, 2014) - võimaldab kasutada Leafleti poolt toetamata projektsioone, sealhulgas ka Eesti omasid.
- Leaflet-ajax (Metcalf, 2014) – JSON ja JSONP failide laadimiseks AJAX-i abil.
- Simplify.js (Agafonkin, Simplify, 2014) – Andmete lihtsustamiseks, vähendab punktide arvu jättes kuju ligikaudselt samaks.

2. ANDMETE OTSIMINE JA ETTEVALMISTAMINE

Andmete hankimiseks kasutatakse Maa-ameti ja Statistikaameti koduleheküIgi. 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. Heaks näiteks on siinkohal Soome, kus juba 2003. aastal alustati tööd nüüd juba *Open Data* nime all tuntud põhimõtetega (Lehtonen, 2014). Soome riiklike institutsioonide serveritest saab teha väga erinevaid päringuid, näiteks haldusjaotusest, satelliidi pilte pilvisusest ja viimase kuu sademeid (Joonis2).



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 (Maa-

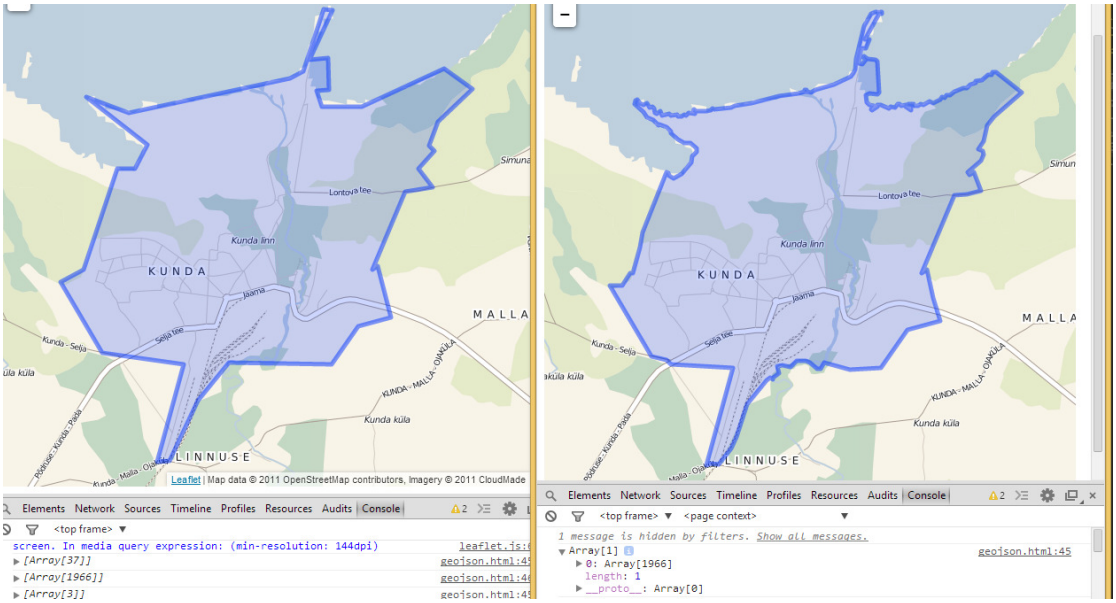
amet, 2014), mille tulemusel võib lähiajal võimalikuks saada ka vektorkujul andmete pärimine ja siis oleks mõistlik teemat edasi uurida.

2.2. Kaardiandmete hankimine ja töötlemine

Maa-ameti lehel on lai valik 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 ilmselgelt 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 3, kus vasakpoolne linn on lihtsustatud ja joonistamiseks on kasutatud 37t punkti parempoolsel aga pole lihtsustamist tehtud ja see koosneb 1966st 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 ikka liiga palju aega ja lihtsustamisel tekkisid omavalitsuste vahele augud, sest Simplify.js kasutab lihtsustamiseks Douglas-Peuckeri algoritmi, ning see ei võimalda selliste aukude tekkimist vältida.

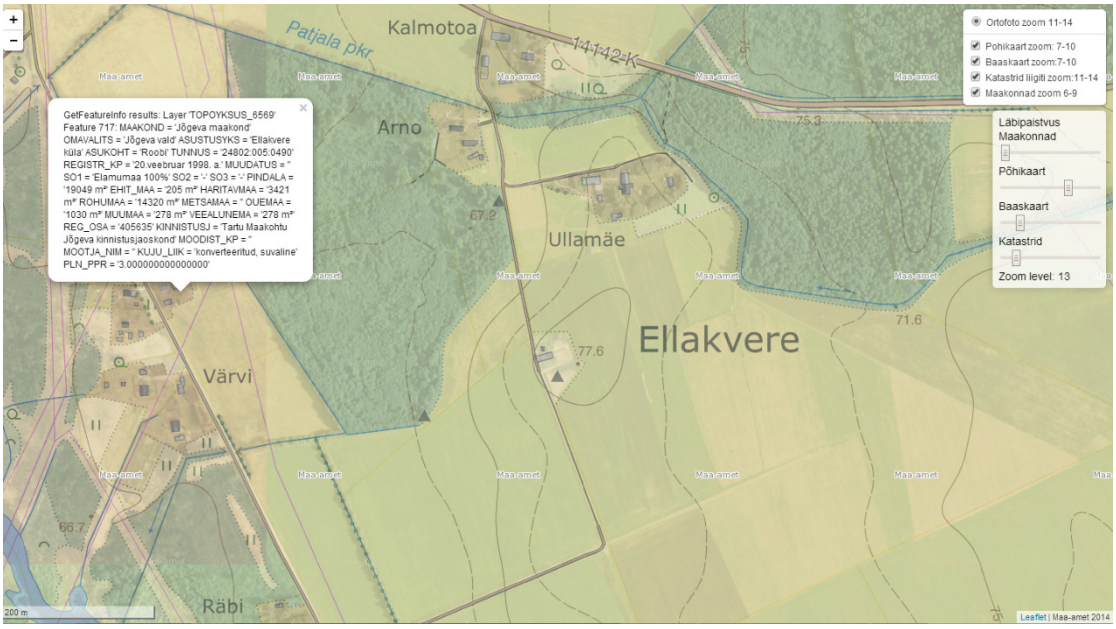
Lahenduseks oleks kasutada modifitseeritud Visvalingam algoritmi (Bloch, 2014), sest see suudab kõrvuti olevate objektide piire säilitada. Ainuke vabavaraline lehekülg, mis autor selle jaoks leidis on http://mapshaper.org/ kuhu õige projektsiooniga salvestatud GeoJSON formaadis faili üles laadides saab 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.

3. MAA-AMETI WMS-TEENUS

WMS-teenuse kasutamise tugi on Leafletil olemas ja kuna selle tööpõhimõte on *Tile* kihi 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 mitmeid kaardikihte kuvada ja kuidas 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 nagu näha joonisel 3.



Joonis 4WMS kihtide näide

3.1. WMS-teenuse GetFeatureInfo päring

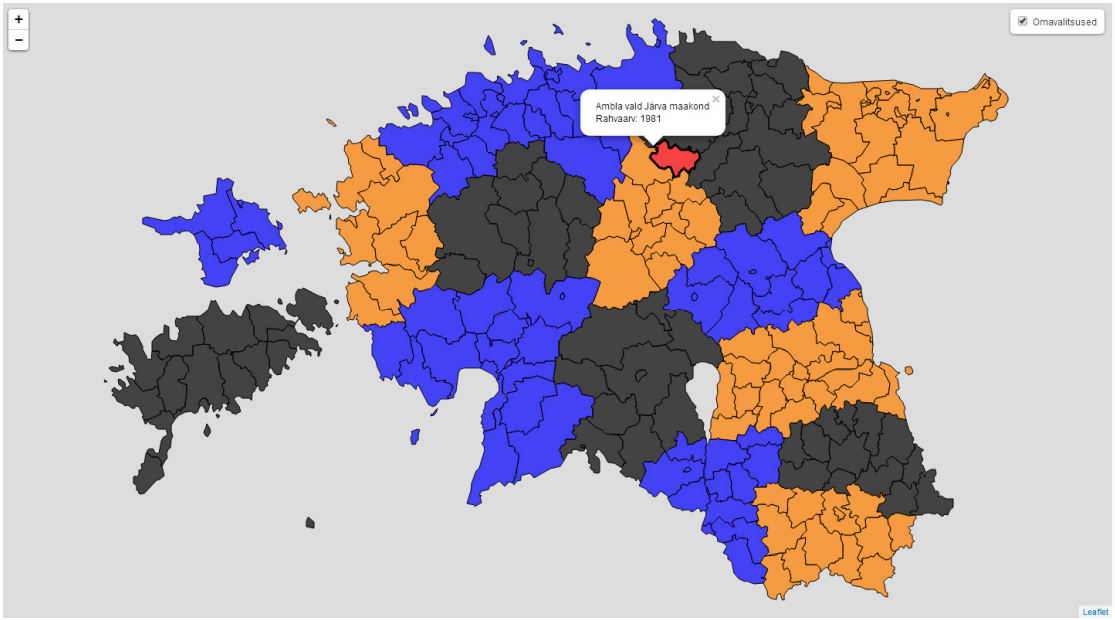
Maa-ameti WMS-teenus võimaldab ka osadelt kihtidelt *GetFeatureInfo* päringuid teha ja õppematerjalis loodud näites saab hiirekliki korral küsida katastri kohta käivat informatsiooni, mille tulemusel tehakse Maa-ameti serverisse päring ja kuvatakse saadud andmed tekkivasse hüpikaknasse (*Popup*) (Joonis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

4. ÕPPEMATERJALI ÜLESEHITUS, TESTIMINE JA ANALÜÜS

4.1. Õppematerjali ülesehitus

Valminud õppematerjal koosneb erinevatest osadest. Esimene osa käsitleb andmete ettevalmistamist ja töötlemist ning mille võib vajaduse korral ära jätta, sest valminud andmed näite jaoks on valmiskujul 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. Lõplik näide on näha joonisel 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.

4.2. 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 järmises osas kasutama hakata.

Teises osas alustati lihtsa näitega omavalitsuste kaardile kuvamiseks ja seejärel lisati iga sammuga juurde kindel osa funktsionaalsust. Õppematerjali läbiti probleemideta esimese tunni lõpuks ja kõik tudengid suutsid oma variandi tööle saada.

Kolmas osa oli kõige keerulisem ja sellega esines ka kõige rohkem probleeme. Esimeseks suuremaks probleemiks kujunes asjaolu, et õppematerjali PDF-failiks konverteerides kadusid osad sümbolid ära ja osad read poolitati ning need ei töötanud enam. Paratamatult tulid sisse ka mõned näpuvead ja ka materjalis olid mõned vead ja asjaolu, et PHP faili sisule veebi kaudu ligi ei pääse, mis nüüdseks on parandatud.

4.3. 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 tagastatavad 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 ja kolmandas raster- ja vektorandmeid koos. Õppematerjali lisas on toodud näited, mida huvitavat kaardirakendustega veel on võimalik teha.

Materjali testimine andis palju erinevat informatsiooni tekkinud probleemide kohta ja aitas seda paremini süstematiseerida ja lihtsustada, mis viis lõpuks algselt kahe erineva näite sisu ühendamiseni.

Kõiki õppematerjalis olevaid faile ja näiteid saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse vastavalt teemadele.

KASUTATUD KIRJANDUS

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

Agafonkin, V. (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. (27. 02 2014. a.). *https://github.com.* Allikas: https://github.com/mbloch/mapshaper/wiki/Simplification-Tips

Kartena AB. (26. 02 2014. a.). *GitHub.* Allikas: Proj4Leaflet: https://github.com/kartena/Proj4Leaflet

Lehtonen, P. (27. 02 2014. a.). *Open data in Finland - Public sector perspectives on open data* . Allikas: http://virtual.vtt.fi/: http://virtual.vtt.fi/virtual/nextmedia/Deliverables-2011/D3.2.1.2._Hyperlocal_Open%20data%20in%20Finland.pdf

Maa-amet. (27. 02 2014. a.). *http://www.envir.ee/.* 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

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

Valminud õppematerjal koosneb erinevatest osadest, millest esimene käsitleb andmete ettevalmistamist ja töötlemist ning mille võib vajaduse korral ära jätta, sest valminud andmed näite jaoks on valmiskujul 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.

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 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.

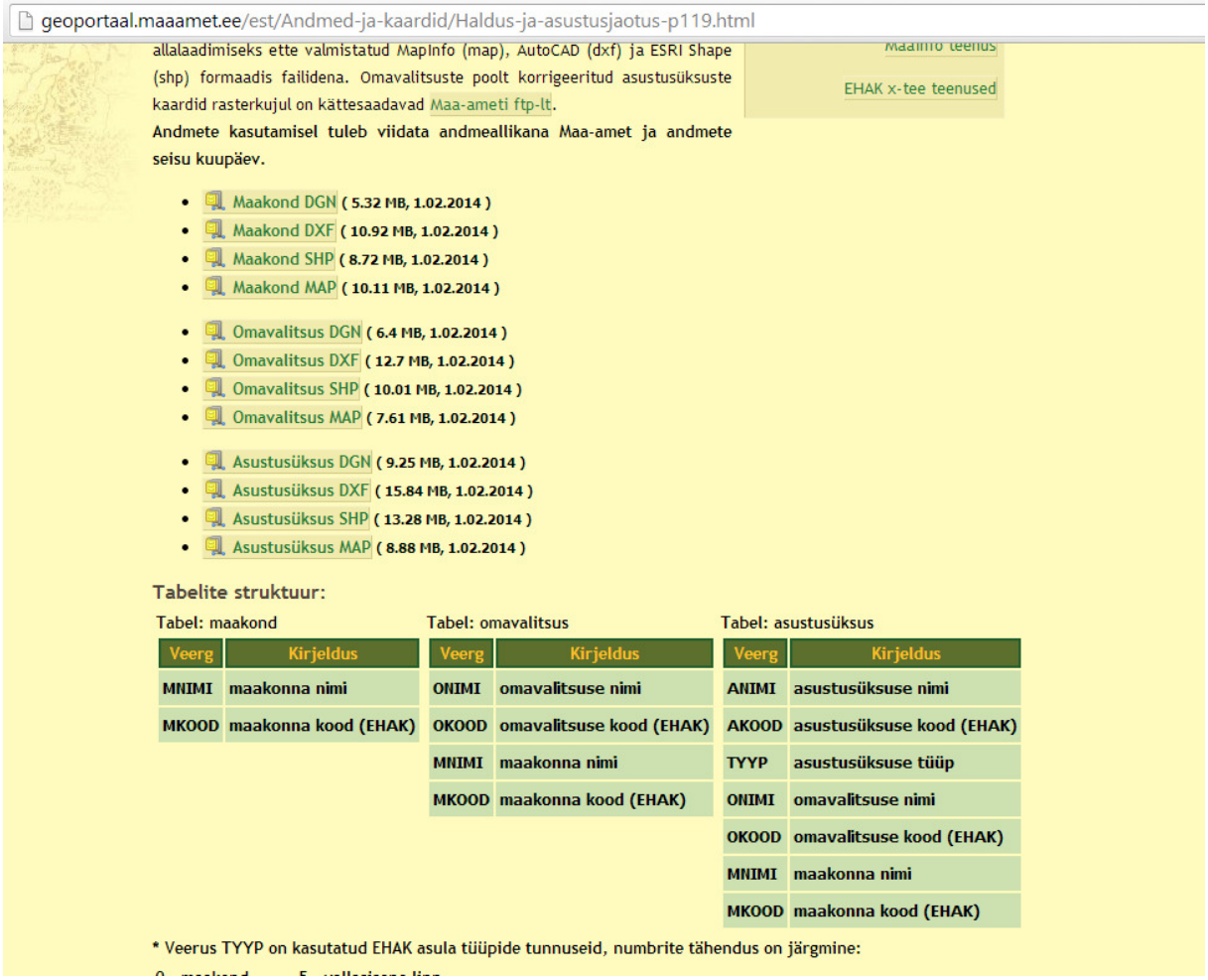
Õppematerjal sisaldab ka väga palju erinevaid faile ja valminud näiteid, mida on saab vaadata aadressil http://greeny.cs.tlu.ee/~elariroo/leaflet. Näited on jaotatud erinevatesse kaustadesse vastavalt teemale.

1. ANDMETE ETTEVALMISTAMINE

Kõige enam kaardiandmeid on Eestis vabalt saadaval Maa-ameti kodulehel. Alla saab laadida erinevaid kaarte alates haldusjaotusest kuni mullakaartideni välja. (Joonis 1)

Joonis 6 Maa-ameti kaardiserver

Haldus ja asutusjaotusele vajutades avaneb aken erinevate tarkvaraprogrammide jaoks mõeldud kaartidele. (Joonis 2)

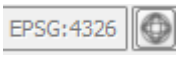


Joonis 7 Omavalitsuste kaardid

Tõmbame alla Omavalitsus SHP faili ja pakime selle lahti. Avades tekkinud kausta näeme *.DBF ja *.SHP faile. DBF faili on salvestatud andmetega koos käivad andmed andmebaasi kujul ja vajadusel saame selle SQL andmebaasi importida. Excelis avades näeme, milline info andmetega kaasas käib. SHP faili on salvestatud koordinaadid. Nende failide avamiseks peame tõmbama vabavaralise programmi nimega QGIS aadressilt http://www.qgis.org/en/site/.

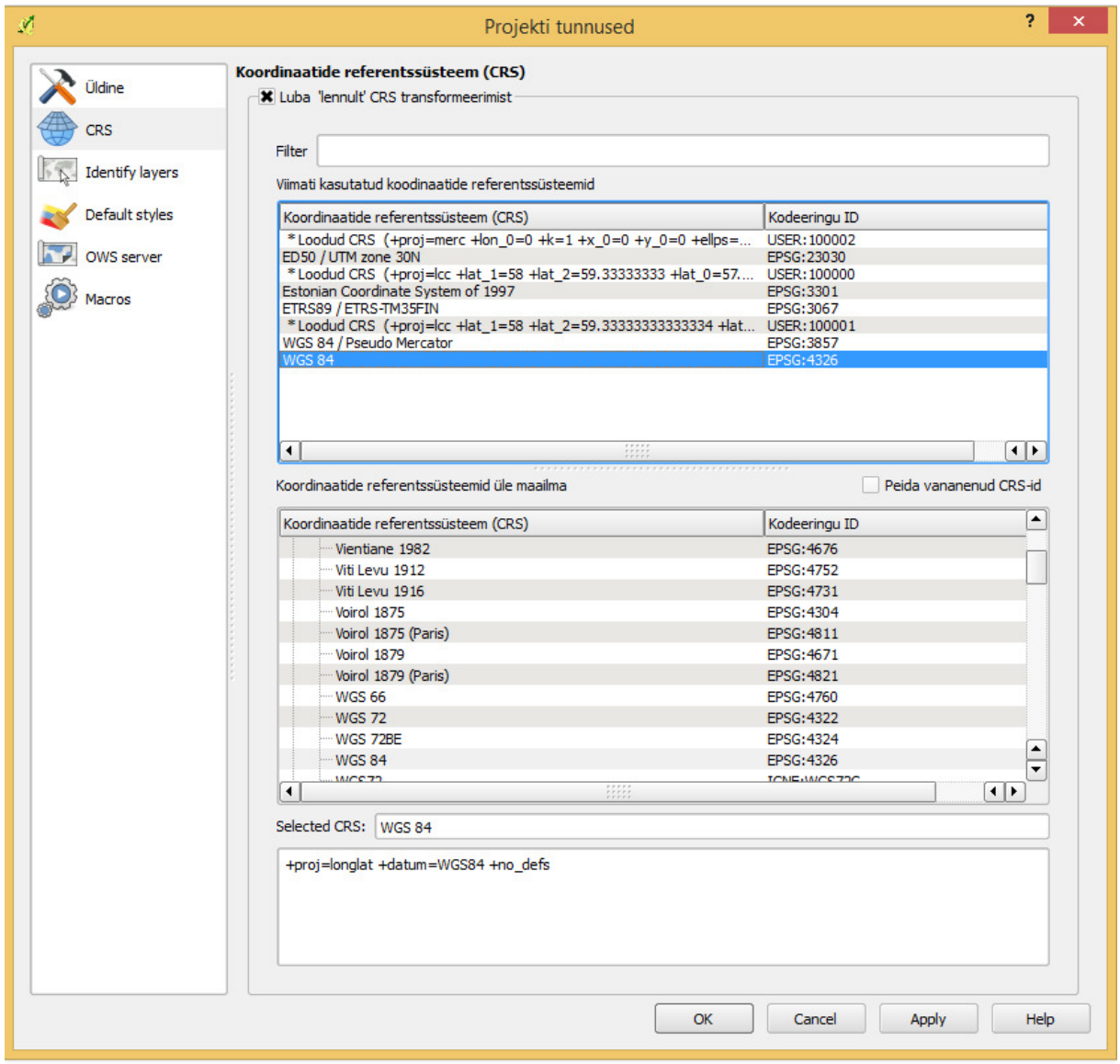
Enne kui QGIS-iga fail avada tuleb muuta sobivaks koordinaatide referentssüsteem, sest Leafleti GeoJson suudab lugeda vaid EPSG:4326 projektsioonis olevaid faile ja Maa-ametist saadud kaart on EPSG:3001 formaadis, seega peame enne QGIS-is lubama

lennult transformeerimist. Vajutades nupule paremal all nurgas avaneb meile aken, kus saame muuta transformatsiooni. (Joonis 3)



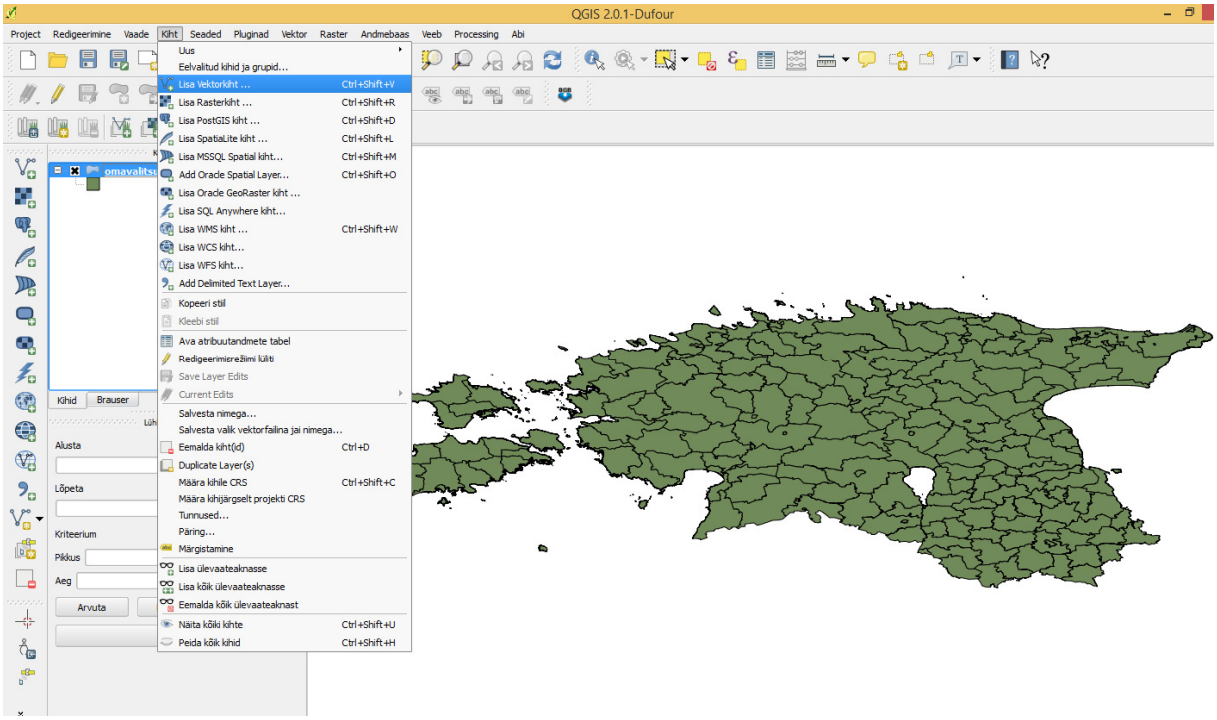
Joonis 8 Transformeerimise nupp

Sealt peame valima EPSG:4326 ja vajutama OK. (Joonis 4)



Joonis 9 'Lennult' transformeerimine

Pärast seda võime avada Maa-ametist tõmmatud kaardi valides Kiht-> Lisa vektorkiht ja valides tõmmatud SHP faili asukoha. OK nupule vajutades avaneb meile Eesti kaart, mis on nüüd õigesse koordinaatide referentssüsteemi teisendatud. (Joonis 5)



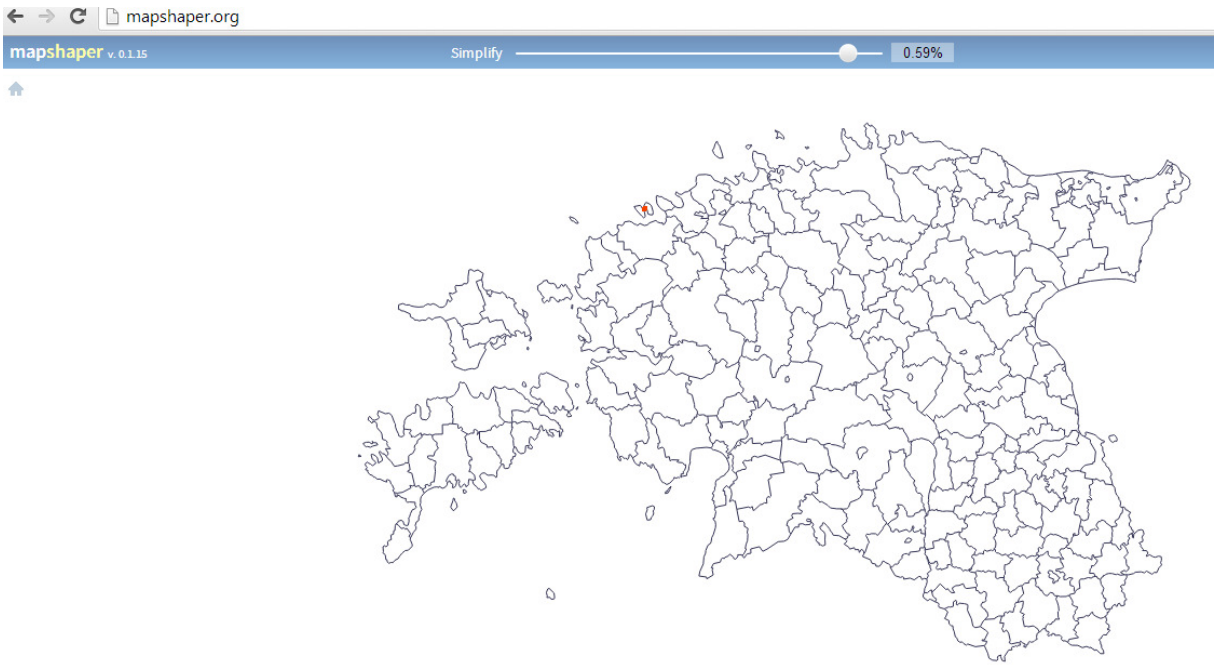
Joonis 10 Vektorkihi lisamine

Seejärel võiksime faili kohe GeoJSON formaati salvestada, kuid see fail tuleb ligikaudu 60 MB ja see on veebis töötamise jaoks liiga suur. QGIS-il on olemas funktsioon geomeetria lihtsustamiseks ja sellega õnnestus faili suurus saada umbes 1 MB peale, kuid sellega kaasnes selline probleem, et osade tükkide vahele tekkisid augud. See tekib seetõttu, et QGIS kasutab lihtsustamiseks Douglas ja Peucker algoritmi, ning see ei võimalda selliste aukude tekkimist suurte lihtsustamiste põhjal vältida. (Joonis 6)

Joonis 11 punktide lihtsustamine QGIS-is ja augud omavalitsuste vahel

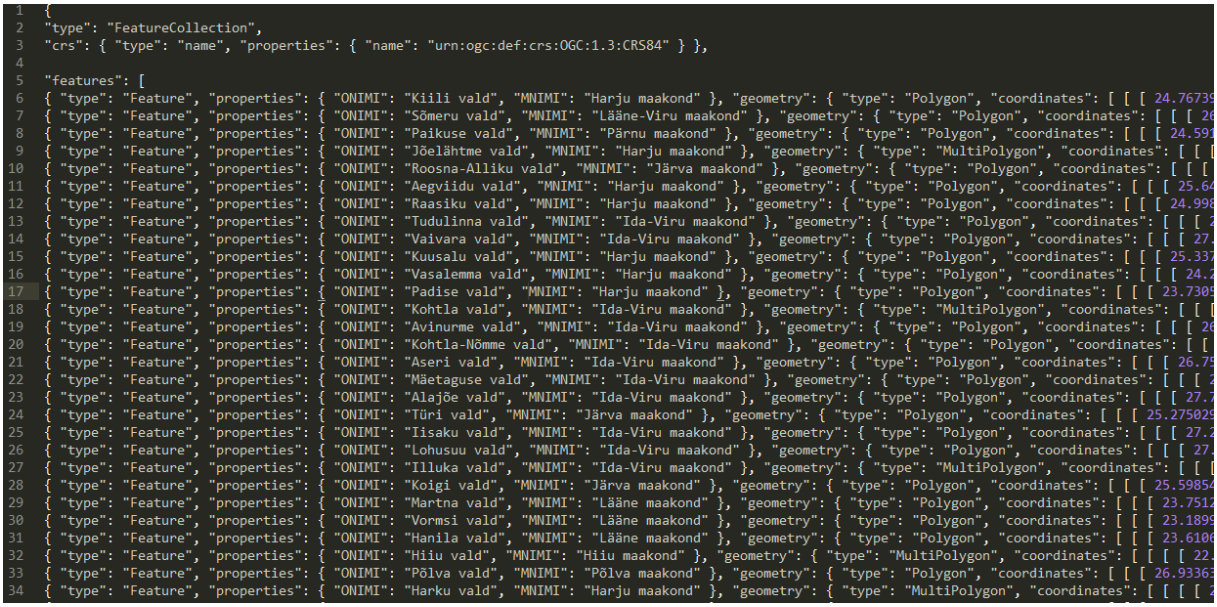
Parem oleks kasutada modifitseeritud Visvalingam-i algoritmi, sest see suudab kõrvuti olevate objektide piire säilitada. Ainuke vabavaraline lehekülg, mis ma selle jaoks

leidsin on http://mapshaper.org/ (Joonis 7) 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

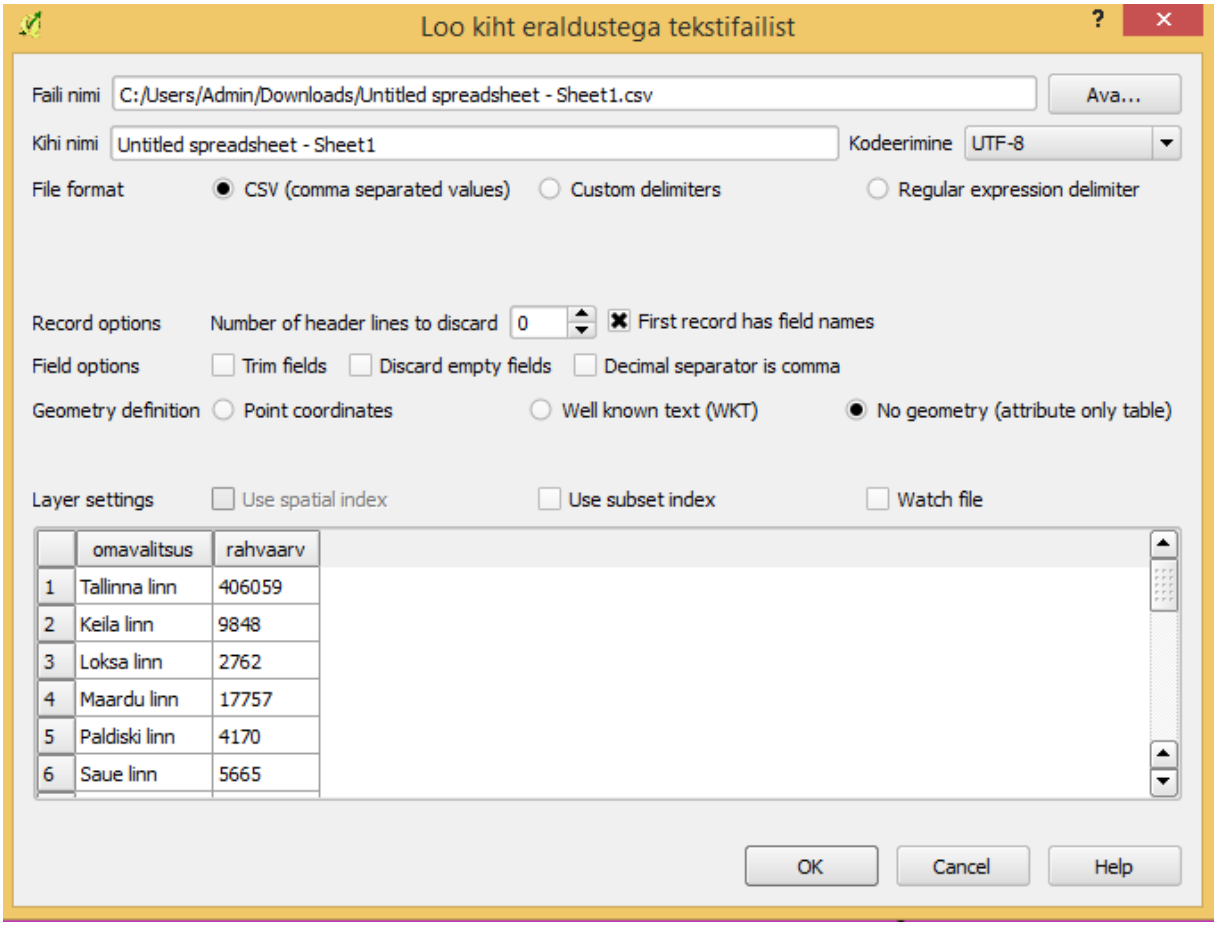
Alla tuleb laadida SHP fail, sest antud lehekülg kaotab lihtsustades ka kõik objekti omadused, kuid vähemasti ei jää enam auke kaardile. Allalaetud kaustast kustutame *.dbf faili ja asendame selle Maa-ameti kodulehelt saadud *.dbf failiga. Siis avame shp faili uuesti QGIS-iga ja nüüd saame selle lõpuks GeoJSON formaati salvestada valides Kiht-> Salvesta nimega ja valime GeoJSON formaadi. Saadud GeoJSON fail tuli kõigest 537 KB ja seda avades näeme, et tegemist on sisuliselt Javascripti objektiga, mis teeb selle edaspidise töötlemise Javascriptis väga mugavaks. (Joonis 8)



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

Järgmiseks soovime kaardile panna ka teistsugust informatsiooni, mis ei sisalda koordinaate. Selle jaoks otsustasin Statistikaameti kodulehelt tõmmata andmed rahvaarvu kohta omavalitsuste kaupa. Kuna Statistikaameti kodulehelt JSON faile alla laadida ei saa, siis tõmbasin alla *.CSV faili ja importisin selle QGIS-i valides Kiht-> Add delimited text layer. (Joonis 9)



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 aadressilt http://greeny.cs.tlu.ee/~elariroo/leaflet kausta nimega geojson_algus.

Seejärel tuleks luua index.html fail ja anda lisada sinna HTML dokumendi kuvamiseks vajaminev info ja Leafleti kasutamiseks tuleb see skriptina laadida. (Koodinäide 1)

```
<!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 1 index.html faili sisu

Seejärel tuleks luua uus fail nimega script.js kus kõigepealt loome kihi, mille sisse laeme omavalitsused failist. (Koodinäide 2)

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

Koodinäide 2 omavalitsuse laadimine failist

Pärast seda saab luua kaardi ja määrata vaikimisi nähtava kaardikihi. (Koodinäide 3)

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

Koodinäide 3Kaardi loomine ja vaikimisi kihi lisamine

Seejärel tekitame kihtide halduse kasutades L.LayerGroupi ja lisame loodud kihi *Overlay* kihtide gruppi, et seda saaks sisse ja välja lülitada. Baaskihte saab lisada *baseLayers* alla ja neid välja lülitada ei saa. (Koodinäide 4)

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

Koodinäide 4 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 5)

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

Koodinäide 5 Kaardi piiride määramine

Lõpuks võib lisada kaardile ka meetermõõdustikus mõõtkava. (Koodinäide 6)

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

Koodinäide 6 Mõõtkava lisamine

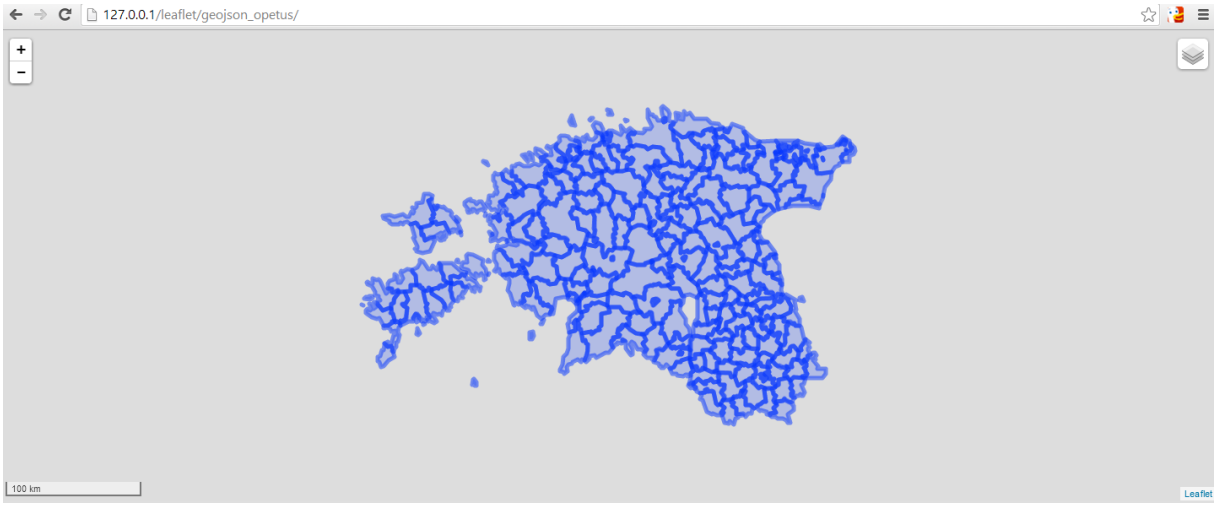
Faili script.js kood peale muudatuste tegemist. (Koodinäide 7)

```
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 7 script3.js faili sisu

Selle käivitamisel brauseris avaneb omavalitsuste kaart, kus kaart on paigutatud ekraani keskele ja saame ka sisse ja välja suurendamist kasutada. Kihtide nupule vajutades saame ka kaarti sisse ja välja lülitada.



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 8)

```
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 8 Mousover funktsiooni lisamine

Selle käivitamisel näeme, et kui liigume hiirega üle omavalitsuste, siis kuvatakse meile 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 meile vastava omavalitsuse värvi. (Koodinäide 9)

```
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 9 Funktsioon värvi tagastamiseks vastavalt maakonna nimele

Seejärel tuleb teha teine funktsioon, mis siis vastavalt värvile omavalitsuse ära värvib. (Koodinäide 10)

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

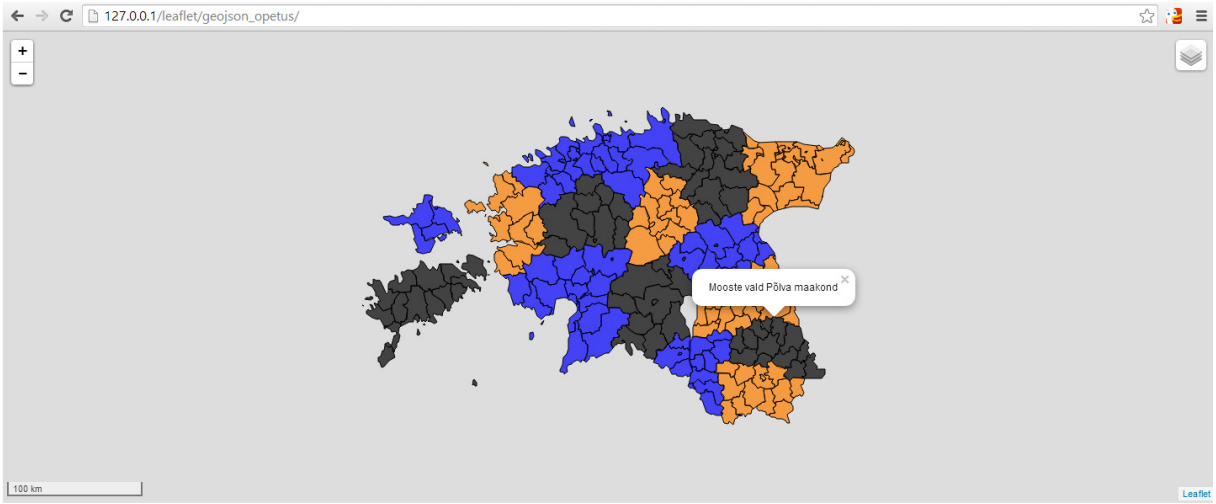
Koodinäide 10 Stiili lisamine

OnEachFeature alla lisame rea *defaultStyle* väljakutsumiseks. (Koodinäide 11)

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

Koodinäide 11 Stiili määramine

Tulemuseks on kaart, kus kõik maakonnad on värvitud erinevat värvi.



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 12)

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

Koodinäide 12 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 13) 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 13 JSON failide sisu ühendamine

Seejärel piisab vaid *onEachFeature* juures *popupContent* muutmisest, et näeksime hiirega peale liikudes ka rahvaarvu. (Koodinäide 14)

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

Koodinäide 14 Rahvaarvu kuvamine hüpikaknal

Lõplik faili script.js kood (Koodinäide 15):

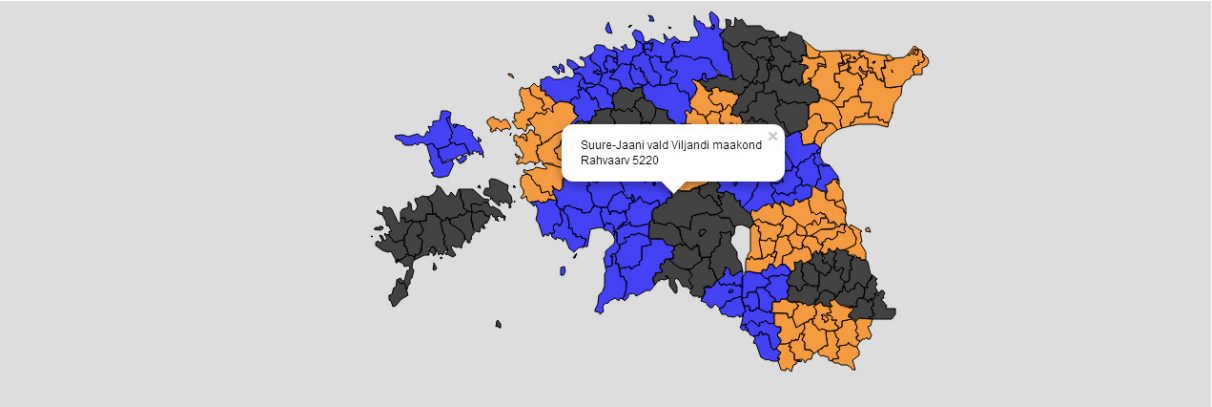
```
for(var i in omavalitsused.features){
  for(var j in rahvastik.features){
    if(omavalitsused.features[i].properties.ONIMI==
          rahvastik.features[j].properties.omavalitsus){
      omavalitsused.features[i].properties.Rahvastik =
          rahvastik.features[j].properties.rahvaarv;
    }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

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

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

```
    layer.on('mouseover',  function(e) {
      var popup = L.popup().setLatLng([e.latlng.lat+0.05,
          e.latlng.lng]).setContent(popupContent).openOn(
     map);
    });
  }
});
function defaultStyle(feature) {
    return{
          weight: 1,
          opacity: 1,
          fillOpacity: 0.7,
          color: 'black',
          fillColor: getObjectColor(feature.properties)
      };
}
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";
  }
var theLayers = new L.LayerGroup([myLayer]);
 map = new L.Map('map', {
    continuousWorld: true,
    worldCopyJump: false,
    layers: [ myLayer]
  });
var baseLayers = {};
var overlays = {
  "Omavalitsused": myLayer,
};

map.fitBounds(myLayer.getBounds());

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

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

Koodinäide 15 Lõplik kood

Lõplik pilt koos rahvaarvuga:



Joonis 17 Info kuvamine koos rahvaarvuga

Ülesanne 1

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

Hättajäämise korral leiab lõpliku näite leiab 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, 2014)

Vastuseks saame pika 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>** tagi ja näeme et toetatakse png, jpeg, gif ja tiff faile. (Koodinäide 16) 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 16 GetMap parameetrid (Maaamet, 2014)

Järgmisena on vaja teada saada missuguseid kaarte server pakub ja millised on nende projektsioonid ja piirid. Seda näeme samas xml failis allapoole kerides (Koodinäide 17). Näeme, 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 17 Kihi parameetrid (Maaamet, 2014)

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 18)

```
<?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>
```

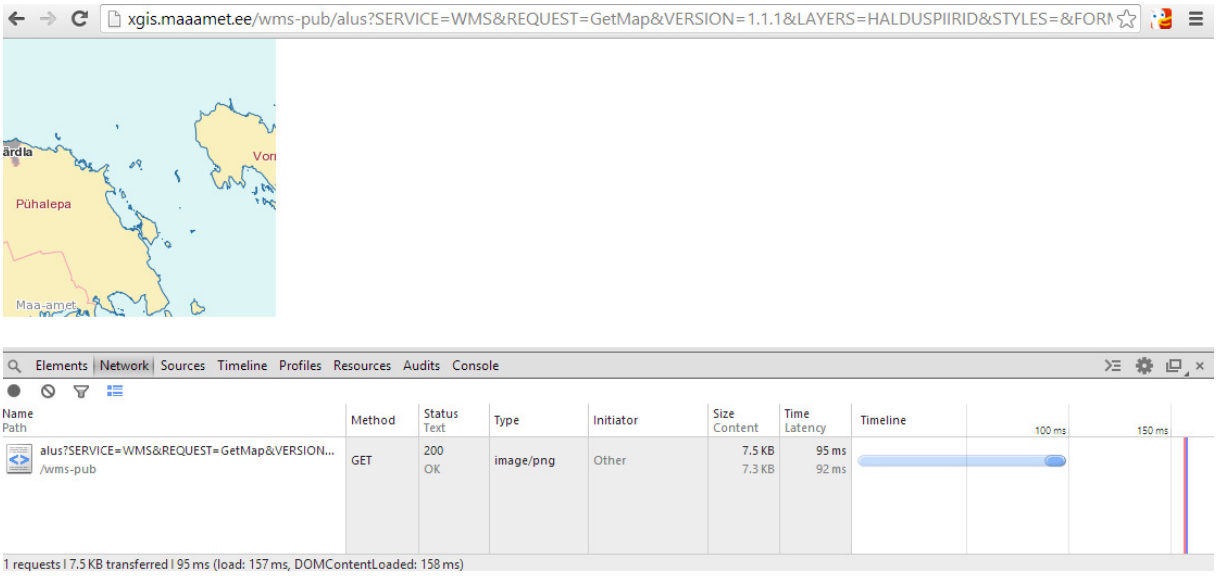
Koodinäide 18 Veateate kood

Eduka päringu jaoks tuleb meil öelda serveri poolt toetatav versioon, kaardikihi nimi, pildi formaat, pildi kõrgus ja laius, kaardi SRS ja *Bounding Boxi* neli väärtust. (Koodinäide 19)

```
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&CONTINUOUSWORLD=true&SRS=EPSG%3A3301
&BBOX=425983.99999999645,6520831.999999799,458751.9999999
978,6553599.999999814
```

Koodinäide 20 Näidispäring

Tehes brauseriga sellise päringu Maa-ameti serverisse saame vastuseks png faili omalt poolt küsitud asukohast. Näeme ka seda, et ei saa vastuseks tervet Eesti kaarti, vaid 256 x 256 pikslit mõõdus pildi. (Joonis 13)



Joonis 18 Näidispäringu vastus

Leaflet oskab seda 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 tarvis 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 lisame kihi kaardile ja teeme selle nähtavaks. . (Koodinäide 20)

```
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 21 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 kaardid nähtavad on. (Koodinäide 21)

```
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 22 Projektsiooni muutmine

Eelnevat teades saame juba kõige lihtsama WMS kaardi valmis teha. Jätkame GeoJSON näites alustatu jätkamisega. Index.html faili sisusse lisame proj4-compressed.js ja proj4leaflet.js failid skriptina ning index.html fail peaks lõpuks 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 ja faili tuleb lisada kõige ülesse eelpool mainitud CRS, siis aluskaardi kiht ja lõpuks peab script.js välja nägema selline:

```
var opacity=0.5;

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]
  });


var aluskaart=
L.tileLayer.wms('http://kaart.maaamet.ee/wms/alus', {
  layers: 'HALDUSPIIRID',
  format: 'image/png',
  maxZoom: 14,
```

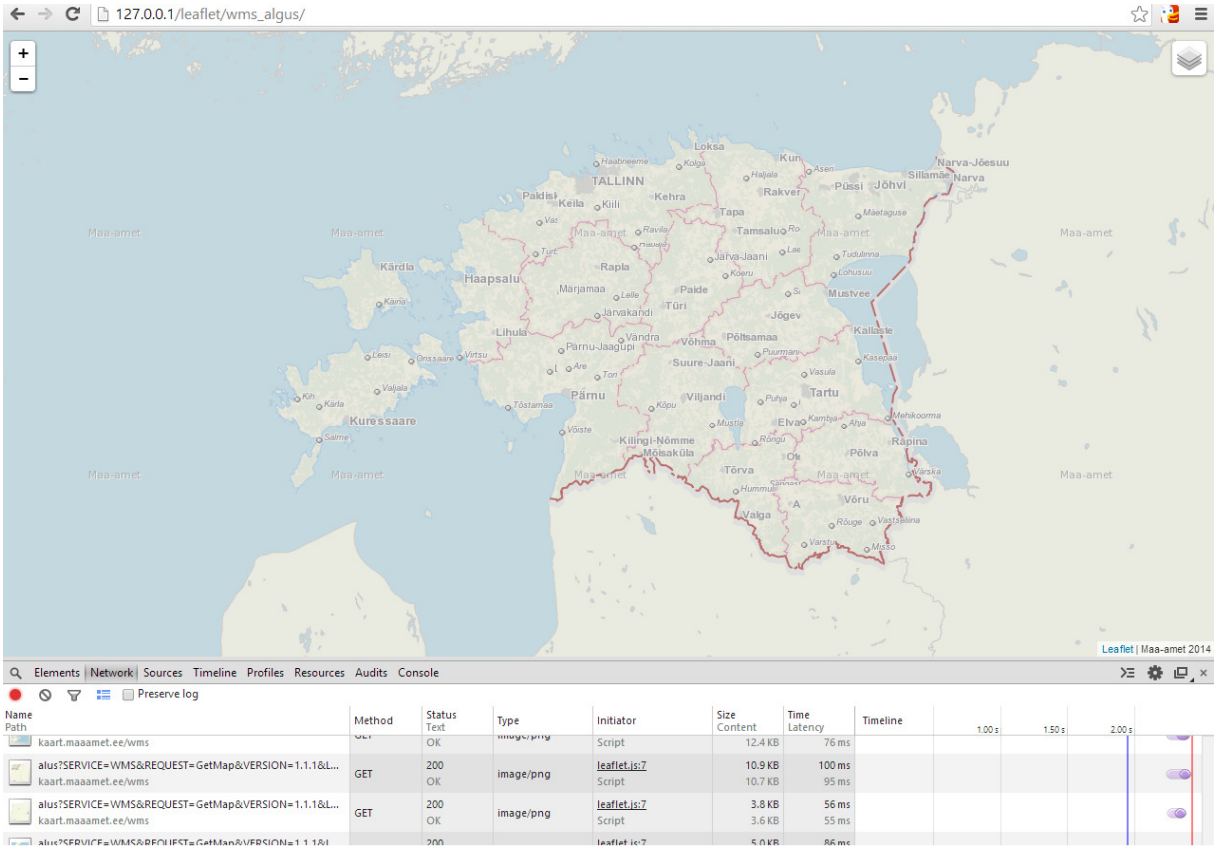
```
  minZoom: 3,
  continuousWorld: true,
  attribution: 'Maa-amet 2014',
  transparent:true,
  opacity:opacity,
});

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;
    }

  }
}

var myLayer= L.geoJson(omavalitsused,
{
  onEachFeature: function(feature, layer){
    layer.setStyle(defaultStyle(feature));
    var feat = feature.properties;
    var popupContent = feat.ONIMI+" "+feat.MNIMI+"</br>
Rahvaarv: "+feat.Rahvastik;
    layer.on('mouseover', function(e){
      layer.setStyle(highlightStyle(feature));
      var popup= L.popup().setLatLng([e.latlng.lat+0.05,
e.latlng.lng]).setContent(popupContent).openOn(map);
    });
```

```
    layer.on('mouseout',  function(e) {
          layer.setStyle(defaultStyle(feature));
      });

  }
});
function highlightStyle(feature) {
    return{
          weight: 3,
          opacity: 1,
          fillOpacity: 0.7,
          color: 'black',
          fillColor: 'red'
      };
}

function defaultStyle(feature){
  return {
    weight:1,
    opacity:1,
    fillOpacity:0.7,
    color: 'black',
    fillColor: getObjectColor(feature.properties)
  }
}
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";
}
var theLayers = new L.layerGroup([myLayer]);

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

var baseLayers={};
var overLay={
  "Omavalitsused": myLayer,
  "Aluskaart": aluskaart
}
L.control.layers(baseLayers, overLay).addTo(map);
map.fitBounds(myLayer.getBounds());
```

Joonis 20 script.js faili sisu

Nüüd peaks faili brauseriga avades nägema juba täiesti töötavat WMS kaarti nagu joonisel 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 21 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 koodinäites 23

```
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 23 Ortofoto kihi lisamine

3.1. Erinevate kihtidega tegutsemine

Kui soovime korraga mitut erinevat kaardikihti kasutada või neid vahetada, siis on kõige mõistlikum 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.

```
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);
```

Joonis 22

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.

```
var map = new L.Map('map', {
    continuousWorld: true,
```

```
    worldCopyJump: false,
    crs: crs,
    layers:[ aluskaart],
  });
```

Joonis 23

Kuna mitme kihi korral ei paista kihid läbi võime 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 lisame HTML-i mõned read, et muuta kutsuda välja kihtide läbipaistvust.

```
<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>
```

Joonis 24

Script.js faili lisame funktsiooni, 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.

```
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);
```

S

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.

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

Koodinäide 24 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.

```
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=513024.0000000007,6487551.999999784,513536.0000
      0000076,6488063.999999784&
      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 25 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 otsustasin tuleb päringu osa teha PHP-ga. Selle jaoks loome faili get.php ja kopeerime sinna koodinäite 26 sisu.

```
<?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 26 Päringu tegemiseks vajaliku php faili sisu

Faili script.js lisame funktsioonid getFeatureInfo saamiseks. Alustuseks peame 'kuulama' hiireklikki ja seejärel hankima klikitud punkti koordinaadid, ning siis saame teha päringu AJAX-it kasutades kuvades saadud andmed hüpikaknaga ekraanile. (Koodinäide 27)

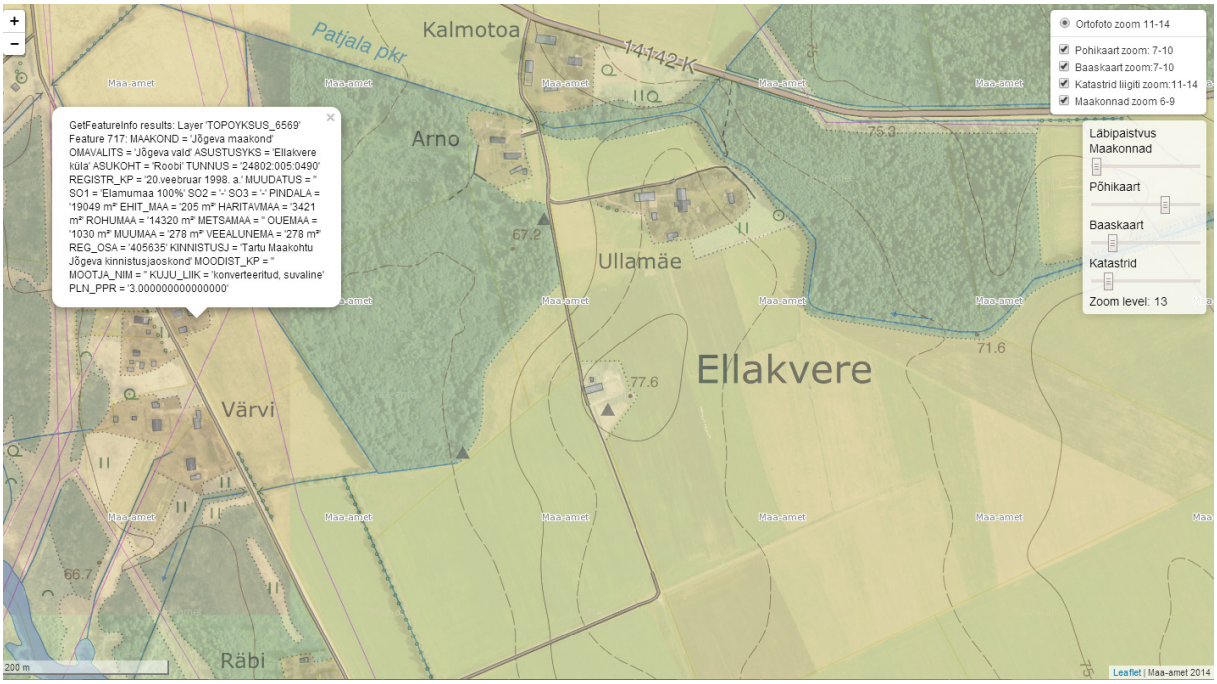
```
map.on('click', function(e, layer){
  var info = getFeatureInfoUrl(e.latlng);
  var url= 'get.php';
  var xhr=new XMLHttpRequest();
  xhr.open("POST", url, true);
  var data= {data: info};

  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 27 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 näha pakuvad Maa-ameti kaardid Eesti kohta oluliselt rohkem infot kui Google või Bingi omad, kõige kasulikumaks võib pidada just talunimede ja katastritunnuse näitamist. (Joonis 22)



Joonis 25 Lõplik näide

Lõplik faili script.js kood asub kaustas wms_lopp ja on toodud ka koodinäites 28:

```
var opacity=0.5;
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]
  });

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,
});


var pohi= L.tileLayer.wms('http://xgis.maaamet.ee/wms-
pub/alus', {
  layers: 'pohi_vr2',
  format: 'image/png',
  maxZoom: 14,
  minZoom: 11,
  opacity:opacity,
```

```
  attribution: 'Maa-amet 2014',
  continuousWorld: true,
});

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 baas= L.tileLayer.wms('http://xgis.maaamet.ee/wms-
pub/alus', {
  layers: 'BAASKAART',
  format: 'image/png',
  maxZoom: 10,
  minZoom: 7,
  continuousWorld: true,
  attribution: 'Maa-amet 2014',
  transparent:true,
  opacity:opacity,
});

var topokataster=
L.tileLayer.wms('http://xgis.maaamet.ee/wms-pub/alus', {
  layers: 'TOPOYKSUS_6569',
  format: 'image/png',
```

```
  maxZoom: 14,
  minZoom: 11,
  continuousWorld: true,
  attribution: 'Maa-amet 2014',
  transparent:true,
  opacity:opacity,
});


var theLayers = new L.LayerGroup([pohi, baas,
topokataster,aluskaart, orto]);
 map = new L.Map('map', {
    crs: crs,
    continuousWorld: true,
    worldCopyJump: false,
    layers: [ aluskaart]
  });

map.setView([58.8, 26.5], 4);
var baseLayers = {
    "Ortofoto zoom 11-14": orto,
  };

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

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);

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

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("Ebaõnnestus!");
    };
});

function makePopup(coords, text){
  var popup =
L.popup().setLatLng(coords).setContent(text).openOn(map);

}

function  getFeatureInfoUrl (latlng) {
    // Construct a GetFeatureInfo request URL given a
point
        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'
        };

    console.log(url + L.Util.getParamString(params, url,
true));
    return url + L.Util.getParamString(params, url,
true);
  };
```

Koodinäide 28 Lõplik kood

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.

```
<!DOCTYPE html>
<html>
<head>
     <title>Simplify.js näide</title>
     <meta charset="utf-8" />
     <meta name="viewport" content="width=device-width,
initial-scale=1.0">
     <link rel="stylesheet" href="leaflet.css" />
</head>
<body>
     <div id="map" style="width: 800px; height:
600px"></div>
     <script src="leaflet.js"></script>
     <script src="simplify.js"></script>
     <script src="eraldised.js"></script><!--
omavalitsused geojson formaadis -->
     <script>


          var cloudmade =
L.tileLayer('http://{s}.tile.cloudmade.com/{key}/997/256/
{z}/{x}/{y}.png', {
               maxZoom: 18,
               attribution: 'Map data &copy; 2011
OpenStreetMap contributors, Imagery &copy; 2011
CloudMade',
               key: 'BC9A493B41014CAABB98F0471D759707'
          });


          var map = L.map('map').addLayer(cloudmade);
```

```
          var simplifiedArray=asulad;

          for (var objekt in asulad.features) {

     if(asulad.features[objekt].geometry.type=="Polygon")
{
                    var SimplifiedPointsArray=
[getTips(asulad.features[objekt].geometry.coordinates)];


     simplifiedArray.features[objekt].geometry.coordinate
s=SimplifiedPointsArray;
               }
          };

          var geoSimplifiedArray =
L.geoJson(simplifiedArray,{style:style});

          function style(feature) {
              return {
                  border:false,
                  weight: 2,
                  opacity: 1,
                  color: 'black',
                  fillcolor: 'red',
                  dashArray: '3',
                  fillOpacity: 0.7
              };
          }
          map.addLayer(geoSimplifiedArray);

          function getTips(coords){
               var points=[];
               for (var i = coords[0].length - 1; i >= 0;
i--) {
                    points.push({x:coords[0][i][0] ,
y:coords[0][i][1]});
               };
```

```
                var tolerance =0.001;
                var highQuality=false;
                var simplifiedPoints= simplify(points,
tolerance,highQuality);
                var jsonSimplified=[];
                console.log("Lihtsustatud "+
points.length+" punktilt "+simplifiedPoints.length+"
punktile");
                if(simplifiedPoints.length<=4){
                     // muuda tolerantsi
                     return coords;
                }
                else{
                     for (var i = simplifiedPoints.length -
1; i >= 0; i--) {
                          if(simplifiedPoints[i].x &&
simplifiedPoints[i].y){

     jsonSimplified[i]=[[simplifiedPoints[i].x],
[simplifiedPoints[i].y]];
                          }else{
                               //  Vajab funktsiooni
multipolügoonide lihtsustamiseks
                          }
                     }
                }
          return jsonSimplified;
          }
          map.setView([58, 26], 6);
     </script>
</body>
</html>
```