Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Ökoloogia ja maateaduste instituut

Geograafia osakond

Magistritöö geoinformaatikas

KAARDIRAKENDUSTE TEGEMINE LEAFLETI JA POSTGISI NÄITEL

Risto Ülem

Juhendaja: prof. Tõnu Oja

Kaitsmisele lubatud:

Juhendaja: /allkiri, kuupäev/

Osakonna juhataja: /allkiri, kuupäev/

Tartu 2018

Kaardirakenduste tegemine Leafleti ja PostGISi näitel

Lühikokkuvõte:

Käesoleva magistritöö eesmärgiks oli uurida, kuidas käib veebikaartide tegemine, millist tarkvara ja tarkvara konfiguratsiooni veebikaartide tegemiseks kasutatakse. Töö käigus tehti mitu erinevat kaardirakendust, et uurida kuidas käib mõnede enamlevinud veebikaardi komponentide programmeerimine (teekonna arvutus, aadressotsing, kasutaja poolt kaardile joonistamine ja lihtsa teemakaardi tegemine).

Selleks valiti sobiv tarkvara pakett – WAMP server, mis sisaldab Apache veebiserverit, PHP-d ja laiendeid PostgreSQL/PostGIS andmebaasiga ühendumiseks. Andmebaasiks valiti PostgreSQL koos PostGIS laiendiga. Kaardi API-ks valiti Leaflet, kuna see on kõige lihtsamini õpitav. Mõne rakenduse jaoks oli kasutusel ka GeoServer.

Märksõnad: Leaflet, PostGIS, GeoServer, veebikaart, kaardirakendus geoinformaatika

CERCS:

P510 Füüsiline geograafia, geomorfoloogia, mullateadus, kartograafia, klimatoloogia

P160 Statistika, operatsioonianalüüs, programmeerimine, finants- ja kindlustusmatemaatika

Web mapping with Lealfet and PostGIS

Abstract:

The aim of this master's thesis was to find out how web mapping works, what kind of software and software configuration are used for web mapping. Several different web maps were developed, to learn how to make most common web map components (routing, address search, client-side drawing on the map, and making a simple thematic map).

A software package was selected for this purpose, which includes WAMP server which comes with Apache web server, PHP and PostgreSQL/PostGIS database extensions. PostgreSQL with PostGIS extension was selected for the database. Lealfet was choosen for mapping API because of it's small learning curve. For some applications GeoServer was in use.

Keywords: Leaflet, PostGIS, GeoServer, WebGIS, programming, geoinformatics

CERCS:

P510 Physical geography, geomorphology, pedology, cartography, climatology

P160 Statistics, operations research, programming, actuarial mathematics

Sisukord

Sissejuhatus

Esimesed veebikaardid tehti juba 1990. aastatel ja sellest ajast peale on veebikaardid muutunud tavapärasemaks kui paberkaardid, pakkudes mitmekülgsemaid vahendeid maailmas toimuvate protsesside visualiseerimiseks. Sellegipoolest ei õpetata Tartu Ülikoolis geograafidele piisaval määral kaardirakenduste tegemist ega veebikaartide kujundamist ja võib-olla ei peagi, kuna enamasti on see ikkagi programmeerijate ülesanne ning internetis on kõik vajalik informatsioon veebikaartide tegemiseks huvilistele kättesaadav.

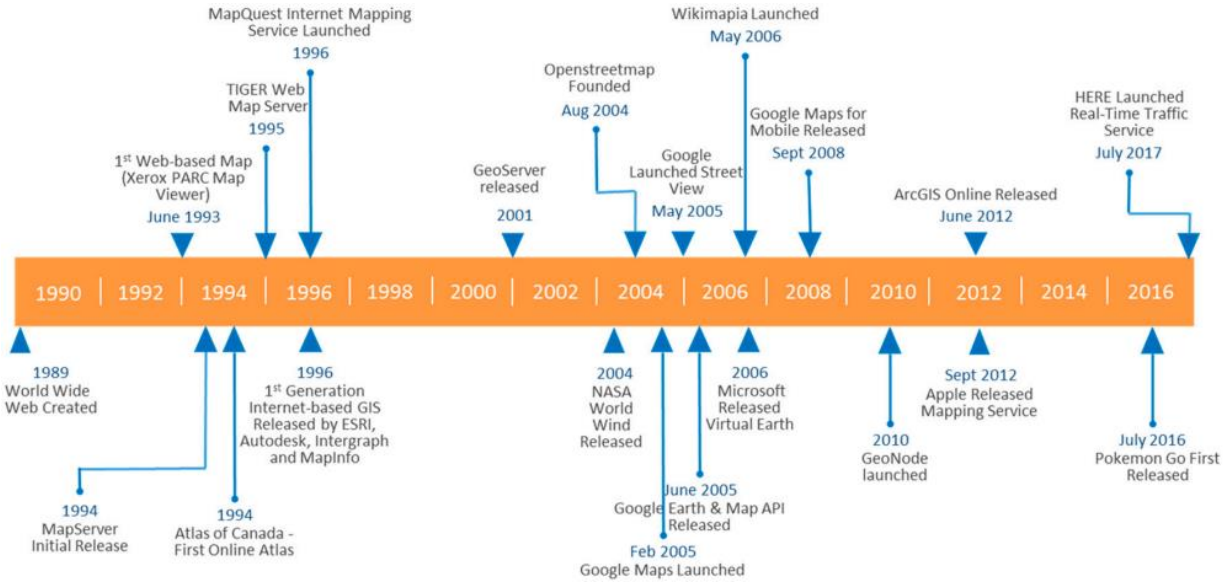
Siiski võiks vähemalt geoinformaatikute võimete pagasisse kuuluda lihtsamate kaardirakenduste tegemise oskus, et vajadusel kasvõi arendajatega paremat koostööd teha. Aluskaartide tegemine on siiski endiselt kartograafide õlgadel, kes erinevalt paberkaardi tegemisest peavad veebikaardi puhul oskama orienteeruda kolmemõõtmelises süsteemis, mis muudab veebikaardi kujundamise keerulisemaks ja nõuab kogenud kartograafi oskuseid. Inimgeograafidelegi tuleks kaardirakenduse tegemise oskus kasuks näiteks inimeste mobiilsust uurides, kus uurija koostab lihtsa kaardirakenduse, kuhu uuritavad saavad märkida oma liikumisteid jms.

Olga Troškina magistritöös (Troškina, 2015) tudengite hulgas läbiviidud küsitlus näitab, et huvi kaardirakenduste tegemise vastu on suur. Siinkohal on tegelikult ka tudengitele vastu tuldud ja kaks teemakohast ainet on õppekavasse lisandunud. Nendes ainetes ei käsitleta siiski kaardirakenduse tegemist terviklikult, vaid paari üksikut osa sellest. Vastavateemalist materjali on küll veebis tohutult palju, kuid selles orienteerumine ja endale vajaliku ülesleidmine võib osutuda keeruliseks ja liiga aeganõudvaks väljakutseks. Samas eestikeelset materjali napib.

Sellest tulenevalt ongi töö eesmärk uurida, mida on kaardirakenduse tegemiseks vaja ja miks ning tuua praktilisi näiteid, kuidas kaardirakendusi teha ning läbi selle anda panus eesti keelsesse geoinformaatika õpetamisse ja arengusse.

1. Veebikaardi ajalugu

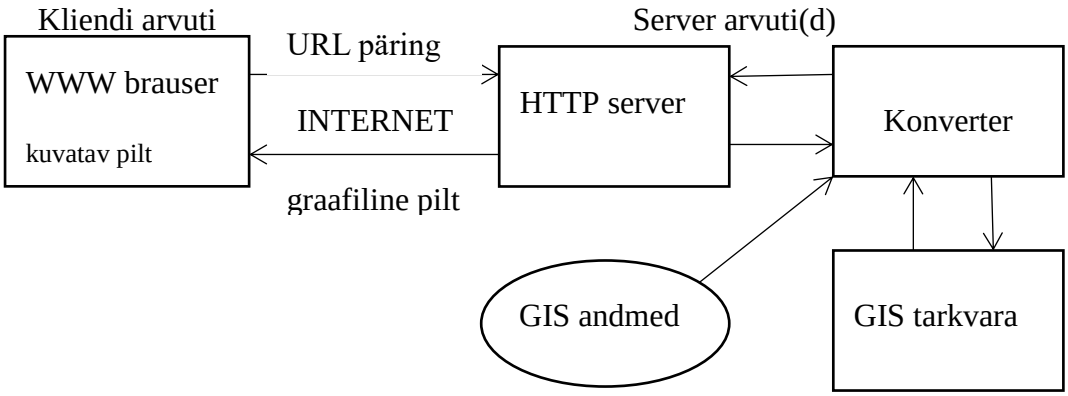
Enne interneti leiutamist kasutati geograafilise info töötlemiseks ja vaatamiseks lauaarvutitele installitud GIS tarkvara. Andmete jagamine ja nähtavaks tegemine suurele vaatajaskonnale toimus vaid paberil. Interneti tulekuga aastal 1989 ja kiire levikuga 1993. aastast, on hakanud arenema ka veebikaardid. Esimesed veebikaardid tulid 1993. aastal, eesotsas Xeroxi Map Serveriga Palo Alto uurimiskeskusest, 1994. aastal Virtual Tourist ja 1995. aastal TIGER (Topologically Integrated Geographic Encoding and Referencing database) Mapping Service'ist. Alates 1996. aastast on tulnud turule juba väga palju erinevaid tegijaid (Blewe, 1997, Veenendaal *et. al* 2017). Joonisel 1 on Veenendaal *et. al* (2017) välja toonud mõned olulisemad sündmused, mis kirjeldavad veebikaartide ajalugu.



Joonis 1. Ajajoon mõnede tähtsamate sündmustega veebikaartide ajaloos (Veenendaal *et. al* 2017).

Veebikaardid jagati kaheks: staatilised ja dünaamilised. Staatilised kaardid olid tavalised rasterformaadis pildid, mis enamasti olid lihtsalt skaneeritud paberkaardid või eelnevalt GIS programmidega tehtud. Need salvestati serverisse ja kuvati kasutajatele veebilehe koosseisus HTML <img> siltide vahel piltidena. Esimesed dünaamilised veebikaardid olid väga aeglased, väikese lahutusega ja korraga sai neid kasutada väike hulk inimesi. Iga päring saadeti läbi serveri GIS programmi, mis võttis andmebaasist vajalikud andmed vajalikus ulatuses ja genereeris nendest kaardi, mis saadeti seejärel tagasi brauserisse (joonis 2). Igakord, kui kasutaja kaardil midagi muutis (suumis, lülitas kihte sisse välja, liigutas

kaardiakent), korrati tegevust uuesti. Rakenduse kiirus sõltus serveri võimekusest andmeid töödelda ja edastada, mistõttu võis rakendus paljude päringute korral väga aeglaseks jääda või kokku joosta. Seetõttu oli korraga töös mitu serverit, mille vahel päringuid jagati, siiski võttis päringu töötlemine paljude kasutajate puhul aega mitu sekundit. Teise võimalusena võeti kasutusele pluginad, mis installiti brauserile ning mille abil delegeeriti osa serveri tööst veebilehitsejale. Serverist saadi ainult andmed, töötluse tegi ära plugin. Selline seadistus muutis kaardirakenduse töö kiiremaks. (Blewe, 1997)



Joonis 2. Dünaamiliste veebikaartide üldistatud klient-server arhitektuur (Blewe, 1997).

Juba kaardirakenduste algusaegadel soovisid geoinformaatikud kasutada veebis tervet GISi funktsionaalsust (päringud, puhvrite tegemine, kattumiste leidmine, ümberklassifitseerimine ja palju muud). See oli just võimalik tänu serveris olevale GIS tarkvarale ja brauseritele lisatud pluginatele. (Blewe, 1997)

Veebikaartide kiirus ja lahutus paranes kõvasti 2005. aastal, kui Google võttis aluskaardina kasutusele eelgenereeritud kaardipüramiidid. Kaardipüramiid on igas suumiastmes valmis tehtud kaardimosaiik raster- või vektorkujul, mis asub serveris ja vastavalt suumiastmele ja asukohale kuvatakse brauseris osa sellest mosaiigist. Ühte kaarditükki kutsutakse kaardi tailiks (ingl k *tile*). Selline lähenemine oli murranguline, kuna enam ei pidanud iga päringut serveris eraldi töötlema, vaid veebilehitsejale saadeti lihtsalt eelnevalt valmis tehtud pilt kaardist. See võimaldas kaarti kuvada paljudele kasutajatele korraga väga kiiresti ja on tänapäeval väga levinud veebikaartide näitamise viis. (Quinn, 2017)

Seega on veebikaart muutunud läbi aja staatilisest pildist interaktiivseks kliendi-serveri suhtlusel põhinevaks rakenduseks (Brovelli *et. al* 2016), mida saab sisse-välja suumida, nihutada, mille kaudu toimuvad päringud serveri ja kliendi vahel ja/või tegeleda ruumiandmetöötlusega.

Veebikaardi definitsioon on siiski jäänud häguseks, nii inglise kui eesti keeles on kasutusel mitu sarnase/sama tähendusega sõna: veebikaart, kaardirakendus (*map application*), veebi kaardistamine (*web mapping*, *online mapping*), interaktiivsed kaardid (*interactive maps*), veebi-GIS (*WebGIS*). Erinevate autorite definitsioonidest tulevad siiski välja ühised jooned, enamasti on kasutusel mõisted nagu kaart, andmebaas, ruumiandmed, visualiseerimine, kasutajaga suhtlemine, GIS, geoinfotarkvara ja veeb (Hess, 2002; Neumann, 2008; Techopedia, 2018; Veenendaal, 2017; Yang *et al.* 2005).

2. Veebikaadi arhitektuur ja komponendid

Internet põhineb kliendi-serveri suhtlusel. Interneti kasutaja pärib läbi kliendi (enamasti veebilehitseja) serverist andmed, server saadab andmed kliendile tagasi ning klient kuvab need. Klient-server süsteemis eristatakse kolme tüüpi arhitektuuri (Agrawal ja Gupta, 2017; Agrawal ja Gupta, 2014; Karnatak, 2016). Esiteks õhuke klient, kus andmetöötluse teeb ära server, mis saadab tulemused kliendile (veebilehitsejale), mille ülesandeks on ainult andmete kuvamine. Selle vastand on paks klient, kus andmed päritakse serverist ja töödeldakse kliendi poolt. Kolmas variant on nende kahe hübriid, kus andmed tulevad serverist ning osa töötlusest teeb ära server ja osa klient. Paksu kliendi eeliseks õhukese ees on, et see vähendab koormust serverile, mis võimaldab kokku hoida riistvara kulutustelt. Paksu kliendi kasutamine muutub järjest populaarsemaks, kuna tavakasutajate seadmete arvutusvõimsused lähevad järjest paremaks ning internetikiirused järjest suuremaks ning see võimaldab suunata rohkem andmetöötlust serverist kliendile. Samas, kui tegemist on väga suurte andmehulkadega, on mõistlikum kasutada õhukest klienti, kuna brauserid võivad lihtsalt hätta jääda väga suurte andmehulkade või väga keerulise andmetöötlusega. (Agrawal ja Gupta, 2017) Kõiki geoinformaatika andmetöötlusvõtteid ei suudagi tänapäeval selleks kasutusel olevad kaardi APId (ingl k *Application Programming Interface*) läbi kliendi teha (Farkas, 2017).

2.1. Serveri komponendid

Kui kliendi pool on enamasti ainult brauser, siis serveri poole saab jaotada veel mitmeks komponendiks, mis võivad asuda ühes ja samas arvutis või paikneda kõik eraldi arvutites. Internetipõhistel rakendustel peab kindlasti olema veebiserver (nt Apache, Nginx, IIS), mis suhtleb läbi HTTP(S) protokolli kliendiga. Andmed saadakse andmebaasiserverist (nt PostgreSQL, MySQL, Oracle) või failiserverist. Kaardirakenduste puhul võib olla kasutusel veel kaardiserver ehk GIS server (nt GeoServer, MapServer), mille kaudu tuleb näiteks aluskaart ja teised kaardiandmed. Eksisteerida võiks ka aplikatsioonserver, mis tänapäeval on enamasti integreeritud veebiserverisse või kaardiserverisse ja mille eesmärgiks on kirjeldada rakenduse käitumise loogika ning eraldada see loogika kliendist ja andmebaasist (Karnatak, 2016; Rao, Vinay, 2009).

2.2. Kasutajaliidese komponendid

Kasutajaliidese komponendid saab jagada kolmeks: aluskaart, teemakihid ja interaktiivsed elemendid. Aluskaart on enamasti taustainfoks ja annab kaardile geograafilise konteksti ning

seda kuvatakse enamasti kaardipüramiidina. Teemakihid määravad ära kaardi kasutusotstarbe ja eesmärgi ning need kuvatakse enamasti vektorkujul, kuna siis on kiiresti muutuvat teemainfot lihtsam uuendada. Interaktiivsete elementide kaudu käib suhtlus kaardirakendusega, näiteks kihtide sisse-välja lülitamine, infoakende kuvamine, vormid, mille kaudu saab pärida infot andmebaasist jne. Nende programmeerimine on kõige ajamahukam. Kaardi API-d nagu Leaflet ja Open Layers pakuvad võimalusi, kuidas lihtsasti teha kõige elementaarseimaid lahendusi. Interaktiivsete elementide kasutusmugavus ja loogilisus määrab paljuski ära terve rakenduse kasutusmugavuse. (Quinn, 2017)

2.3. Kaardi API-d

Rakendusliidesed ehk API-d (ingl k *Application Programming Interface*) on tarkvarad, mis on tehtud selleks, et programmide kirjutamine oleks lihtsam. Näiteks sisaldavad kaardistamise APId juba vajalikke klasse aluskaardi ja kihtide jaoks. Kaardi API-sid on väga palju ning paljud neist on kas täiesti tasuta kasutatavad või siis mingi piirini tasuta. Populaarsemad tasuta APId on näiteks Leaflet, OpenLayers, Tangram ja Cesium. Lisaks on igal suuremal kaardistamisega tegeleval firmal (Google Maps, ESRI, Mapbox, Carto jne) oma API-d. Nende kasutamiseks on vaja API võtit, mille abil jälgitakse API kasutamist. Selleks, et API-sid kasutada, on kaks võimalust. Need saab kas oma arvutisse/serverisse alla laadida, mis võimaldab teha lähtekoodis muudatusi, või viidates API-le läbi sisuedastusvõrgu, mis asub kellegi teise serveris. Selleks tuleb lihtsalt lisada koodi API dokumentatsioonis viidatud veebilehe aadress.

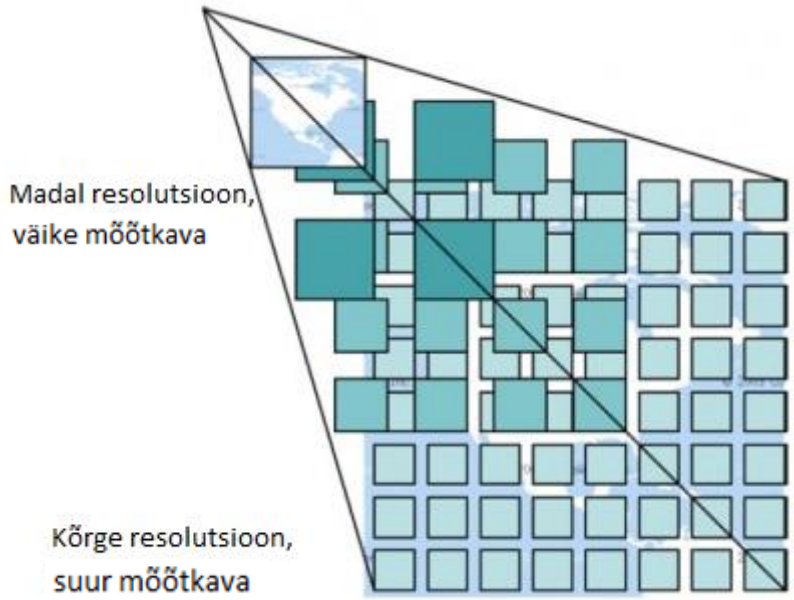
2.4. Kaardipüramiid/aluskaart

Nagu eelnevalt mainitud, on kaardipüramiid (ingl k *tiled web map, slippy map, cached map*) igas suumiastmes valmis tehtud kaardimosaiik raster- või vektorkujul ning kaardipüramiidi kasutamine teeb veebikaardi WMS-iga võrreldes palju kiiremaks. Ühte kaardimosaiigi tükki kutsutakse kaarditailiks (ingl k *map tile*). Kaarditailid on enamasti 256x256 piksli suurused (aga ka 64x64, 128x128, 512x512 või ristkülikukujulised (Untera, 2012)) ning need paigutatakse üksteise kõrvale, nii et jääb mulje nagu oleks tegemist ühes tükis oleva kaardiga. Tailid salvestatakse arvuti vahemällu (ingl k *cache*) ning on sealt ülikiiresti kättesaadavad, see muudabki kaardi kasutuse kiireks ja sujuvaks.

Teatavasti on veebikaardid suumitavad ning sisse suumimisel kaardi detailsusaste suureneb, nähtusi tekib juurde, mis tähendab, et igal suumiastmel peavad olema erineva kujundusega

tailid. Võrreldes tavaliste paberkaartidega nõuab see kartograafidelt suuremat pingutust, kuna iga suumiaste tuleb kujundada erinevalt (värvid, leppemärgid, joonestiilid, kaardikirjad). Selle asemel, et lihtsalt nähtusi mingil suumiastmel juurde lisada, saab ja olekski mõistlik muuta ja vahetada nähtuste kujundust erinevatel suumiastmetel, olenevalt sellest, mida kaardil parasjagu näidatakse. Kasutada saab ka põnevaid üleminekuefekte ühelt suumiastmelt teisele. Lisaks sellele on vaja täiesti teistsugust stiili hübriidkaardile, et kaardiinfo paistaks ortofoto pealt välja.

Kaardipüramiidis on tailid organiseeritud koordinaatsüsteemi järgi, igal tailil on z-koordinaat, mis näitab suumiastet, ning x- ja y-koordinaat, mis näitavad, kus tail kaardipüramiidis asub. Suumiastme suurenedes (aste null on terve maailm; mida suurem aste, seda suurem detailsus) jagunevad kaarditailid järjest väiksemateks (joonis 3).

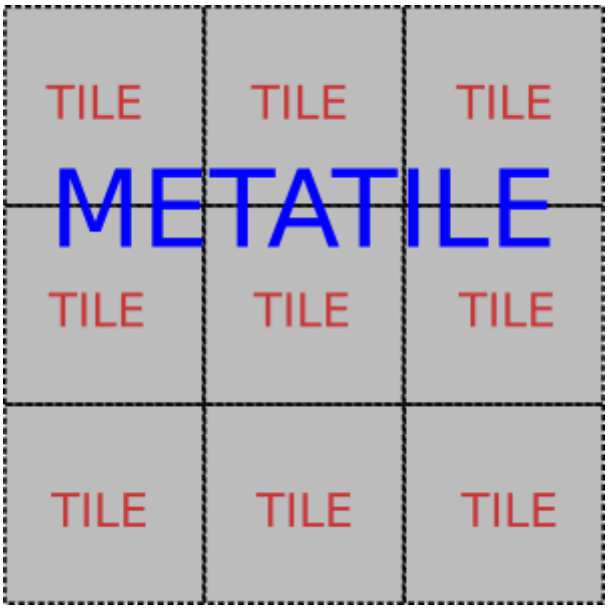


Joonis 3. Näide kaardipüramiidist (QGIS, 2017).

Kaarditaile hoiustatakse arvuti kõvakettal või serveris kaustades, kusjuures igal standardil või tootjal on oma hoiustamise süsteem. Näiteks võib taile jagada suumiastmete järgi kaustadesse, kus iga alamkaust tähistab ühte veergu ja igas veeru kaustas on tailid ridade järgi nummerdatud (Open Street Map, 2017). Neid taile saab seejärel pärida läbi veebiaadressi. Näiteks OSM kaarditailid asuvad aadressil http://{s}.tile.osm.org/{z}/{x}/{y}.png. Taile saab hoiustada ka andmebaasis, näiteks SQLite, mille eeliseks on väiksem andmemaht, kuid kiiruse poolest on failisüsteemis tailide hoiustamine siiski natuke kiirem (Untera, 2012).

Kaarditailide tegemiseks on mitmeid võimalusi (Quinn, 2017). Üks võimalus on terve oma kaart vaikimisi tailideks genereerida, jättes tähelepanuta, et eksisteerivad suured alad, kus suumiastme muutudes kaardipilt ei muutu, näiteks suured tühjad ookeani- või kõrbealad. See muudab kaardipüramiidi põhjendamatult suureks. Teiseks võimaluseks on jätta nende kohtade peal väiksemates suumiastmetes tailid tegemata, hoides nii mahtu ja aega kokku. Kolmandaks võimaluseks on panna nende kohtade peale nii-öelda „tühjad“ tailid mingisuguse kirjaga, näiteks „info puudu“ ning hoida sellega veel rohkem mälu kokku. Neljandaks variandiks on genereerida tailid ainult kõige rohkem vaadatavate piirkondade kohta (nt linnad) ja ülejäänud tailid genereerida siis, kui kasutaja neid piirkondi vaatab. Muidugi tuleb arvestada, et jooksvalt tailide genereerimine muudab konkreetses kohas kaardi aeglaseks. Tailidele saab panna ka aegumise kuupäeva, misjärel kaardiserver järgmisel päringul uued tekitab.

Tailide genereerimiseks kasutatakse tihti metataile, mis on üks suur tail, mis koosneb mitmest väikesest (joonis 4). Selle peamisi eesmärke on vältida taili servas nähtustele topelt kirjete panemist (joonis 5).



Joonis 4. Metatail (GeoWebCache, 2017).

Joonis 5. Topeltkirjad taili servas (GeoWebCache, 2017).

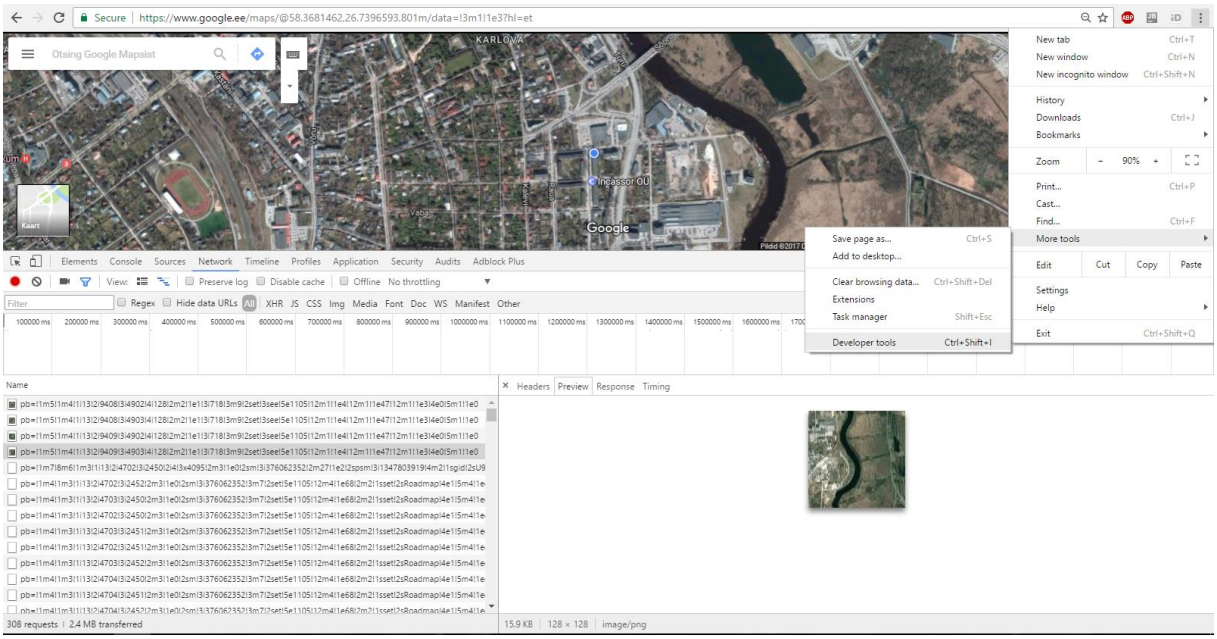
2.4.1. Rastertailid

Rastertailid on vektorandmetest genereeritud pildifailid, enamasti PNG või JPG formaadis. Kuna need on lihtsalt rasterpildid, siis ei saa nendega midagi muud peale vaatamise teha. Rastertailide genereerimine võtab väga palju aega ja rasterkaardipüramiid on mahult väga suur. Näiteks OSM-i rasterkaardipüramiid terve Eesti kohta on umbes 30 GB[1] ja terve maailma kohta 54-340 TB[2] (suurus sõltub kujundusest ja sellest, mida mingil suumiastmel näidatakse). Seetõttu kasutatakse neid aluskaardina, mille peale pannakse vektorteemakihid, mis on sisse-välja lülitatavad ja kiiremini uuendatavad. Kuna aluskaardi info ei vanane nii kiiresti kui teemakihtide info, on selline lähenemine täiesti vastuvõetav. Rastertailid võtavad väga palju ruumi ja nende genereerimine on aeglane, mistõttu ei saa ja pole ka mõistlik neid väga tihti uuendada. Näiteks kasutades programmi TileMill arvutiga Lenovo Thinkpad 64 bit i7-4710MQ 2.50 GHz 8GB RAM võtab kogu Eesti kohta rastertailide tegemine aega kuskil 10 päeva. (sisendandmeteks olid majad ja teed OSM-st, Maa-ameti poolt pakutavad avaandmed mõõtkavas 1:250000 ja haldusjaotuse *shapefailid*). Rastertailid on kõige populaarsem viis aluskaartide tegemiseks, kuna enamus tarkvarasid toetab neid. Rasterkaardipüramiididele on loodud ka mitmeid standardeid, millest täpsema ülevaate on teinud Untera (2012):

[1]http://tools.geofabrik.de/calc/#type=geofabrik_standard&bbox=21.336402,57.487752,28.236061,59.968233
[2]https://help.openstreetmap.org/questions/9766/how-big-is-your-map-size-for-the-entire-world

- TMS (*Tile Map Standard*), mis on OSGeo (*Open Source Geospatial Foundation*) poolt loodud.
- WMTS (*Web Map Tile Standard*), mis on OGC (*Open GIS Consortium*) poolt loodud.
- *Quadkey (Quadtree keys),* mida kasutab Microsofti Bing Maps.

Selleks, et näha, milline on veebikaardi tail, saab kasutada brauseris arendaja tööriistu või inspekteerimise võimalust. Näiteks avades Google Chrome'is arendaja tööriistad ja vaadates samal ajal Google Mapsi, avaneb selline vaade nagu joonisel 6. Tehes lahti „*Network*" vahelehe ja avades sealt ühe musta ruuduga (ortofoto) tähistatud lingi, ongi näha üks tail (joonis 6).



Joonis 6. Arendaja tööriistade kasutamine Google Chrome'is kaarditaili vaatamiseks.

Rastertailide kujundamiseks ja genereerimiseks on palju erinevaid tarkvarasid. Üks nendest on avatud lähtekoodiga JavaScriptil, C++ või Pythonil põhinev teek Mapnik. Firma Mapbox on loonud Mapnikul põhineva graafilise kasutajaliidesega programmi TileMill, mis on väga mugav ja väga palju võimalusi pakkuv kaardipüramiidi kujundamise ja genereerimise programm. TileMill on tasuta ja kõigile vabalt alla laetav. TileMill kasutab Mapboxi poolt tehtud spetsiaalselt kaardi kujundamise jaoks loodud keelt CartoCSS, mis on väga sarnane tavalisele CSS-ile ning seetõttu lihtsasti kasutatav ja ära õpitav. TileMill toetab enamusi laialt levinud andmetüüpe (shapefile, geoJSON, CSV, KML, GeoTIFF), kui ka andmete otse SQLite või PostGIS andmebaasist sisse toomist. Väljundformaadiks on PNG, SVG või PDF ning ka Mapboxi enda formaat MBTiles, mis on tegelikult lihtsalt SQLite andmebaas tailide

hoidmiseks (MBTiles Specification, 2017). MBTiles formaati kasutades tuleb tähele panna, et see võib sisaldada nii raster- kui vektortaile, kuna Mapbox on läinud rastertailidelt üle vektortailidele, aga on failiformaadi samaks jätnud. Osad serverid (TileStache, TileStream, TileCloud) oskavad MBTiles formaati kasutada otse, aga MBTiles formaati on võimalik ka lahti pakkida tavaliseks piltidest koosnevaks kaardipüramiidiks, et kasutada taile otse oma arvuti kõvakettalt või ükskõik millisest serverist. Selleks saab kasutada käsuviibapõhist programmi MButil.

Taile saab genereerida ka GeoServeris ja MapServeris. GeoServer kasutab tailide tegemiseks sisseehitatud GeoWebCache'i teeki ja tailid kujundatakse SLD-d (*Styled Layer Descriptor*) kasutades, mis on XMLi-põhine märgistuskeel (GeoServer, 2017). Võrreldes CartoCSS'iga on SLD keerulisem ja ei paku ka nii palju võimalusi. MapServer kasutab MapCache'i tailide tegemiseks ja kujundamiseks *style*-faili, mis on küll lihtne, kuid pakub samuti vähem võimalusi.

Loomulikult saab kaardipüramiidi teha ESRI ArcMap'is. ArcMap (versioon 10) genereerib kaardipüramiidi kaardiaknas olevatest kihtidest, selle kujundusega, mis neil seal parasjagu on. Seda, mis suumiastmes mingit kihti näidatakse, saab sättida, klikkides *Table of Contents* aknas parema klahviga kihi nimel ja valides hüpikaknas *Visible Scale Range.* Kaardipüramiidi tegemiseks tuleb kõigepealt *Customize* menüüst minna *ArcMap options* aknasse ja *Sharing* vahekaardi alt sisse lülitada *ArcGIS Runtime Tools*. See tekitab *File* menüüsse *Share As* nupu juurde valiku *Tile Package*. Avanevas aknas saab valida tailide formaadi ja suumiastmed ja koha, kuhu tailid salvestatakse – kas ArcGIS Online kontole, mis peab olema organisatsiooni konto, et neid hiljem kasutada saaks, või arvuti kõvakettale. Väljundiks on TPK (*Tile package*) fail, mis sisaldab kokku pakitud taile ning mida saab kasutada ArcGIS serveris. Kui tahta vaadata TPK faili sisse, saab selle lahti pakkida WinRAR'i või mõne muu arhiveerimistarkvaraga või kasutades ArcMap'i tööriista *Extract Package*. Lahtipakitud kaustas on teiste hulgas ka kaust nimega *_alllayers*, mis sisaldab endas kaardipüramiidi jagatuna z, x, y skeemi järgi kaustadesse, kuid pildifailide asemel on hoopis *Compact cache* formaadis .bundle ja .bundlx failid, mis sisaldavad endas kokkupakitud pildifaile binaarkujul. Eesmärgiks on vähendada kaardipüramiidi mahtu ja suurendada tailide genereerimise kiirust (ArcGIS, 2017). Kuna TPK on ESRI enda formaat, siis saab seda kasutada ainult ESRI tarkavaraga ja ainult läbi serveri. Selleks, et TPK failist teha pildiformaadis kaardipüramiid on loodud Pythoni vabavaraline teek Tpkutils.

QGIS'is on kaardipüramiidi tegemine ilmselt eelpool mainitud viisidest kõige lihtsam ja kuna QGIS on vabavaraline tarkvara, ei ole ka failiformaatidega mingit muret. Selleks on loodud plugin QMetaTiles, mis teeb kaardipüramiidi kaardiaknas olevast sisust ja kihtide kujundusest PNG formaadis, salvestades nad ZIP formaadis kokkupakituna kettale. Kihtide suumiastmeid saab sättida sarnaselt ArcGIS'ile kihi nimel avaneva hüpikakna valikus „Määra kihi nähtavuse skaala".

2.4.2. Vektortailid

Vektortailid sarnanevad paljuski rastertailidele, kuid nagu nimigi ütleb, sisaldavad nad vektorandmeid (binaarformaadis), mitte rasterpilti. Kuna vektorandmed on mahult väiksemad kui rasterandmed, võtavad need kordades vähem kettaruumi ja nende tegemine võtab vähem aega. Näiteks OSM'i andmetest tehtud vektorkaardipüramiid Eesti kohta on umbes 140 MB ja terve maailm ligikaudu 44 GB. Vektortailide väikese suuruse tõttu on kaardi käsitlemine brauseris veelgi kiirem kui rastertailide korral, ning kui aluskaardi andmed muutuvad, siis saab väga kiiresti tekitada uue kaardipüramiidi omamata väga suurt arvutusvõimsust.

Teadupoolest ei sisalda vektorandmed infot kujunduse kohta, need kujundatakse kliendi (brauser, ArcMap, QGIS) poolt, mis tähendab, et ka vektortailid ei ole eelnevalt kujundatud. Seetõttu saab ühele kaardipüramiidile luua mitu erinevat stiili.

OGC'l ega ka OSGeol ei ole vektortailide standardit nagu rastertailide puhul, küll aga on Mapbox avaldanud oma avatud formaadiga vektortailide spetsifikatsiooni (Mapbox, 2016), mis on muutumas mitteametlikuks standardiks. Näiteks 2015. aasta juulis avaldas ESRI, et võtab oma tarkvaras kasutusele Mapbox'i spetsifikatsiooni, mitte ei loo enda oma (ESRI, 2015).

Vektortailide tegemiseks saab samuti kasutada Mapbox'i tarkvara: internetipõhine Mapbox Studio või kui tahta tailid salvestada enda arvuti kõvakettale, siis sobib selleks Mapbox Studio Classic. Mapbox'i vektortailid on nagu rastertailidki salvestatud enamasti MBTiles formaadis ning kuna see on avatud formaat on paljudel tarkvaradel võimekus seda kasutada. Vajadusel saab selle siiski ka lahti pakkida, kasutades MBUtil'i. Mapbox Studio Classic võimaldab kohe teha ka tailidele vastava stiilifaili, kasutades CartoCSS kujundamiskeelt, ja selle üles laadida Mapbox'i kontole või salvestada enda arvutisse TM2 faililaiendiga kausta. Hea vahend stiilifailide tegemiseks on vabavaraline programm Maputnik. Maputnikul on olemas nii

veebipõhine kui ka desktop-versioon. Desktop-versioon on tegelikult kohalik server, mida saab kasutada veebibrauseris localhosti abil. Maputnikuga loodud stiilid on JSON formaadis ning neid saab lihtsasti kasutada iga kaardi API-ga. Vektortailide kujundamiseks ei ole vaja tegelikult eraldi tarkvara – stiilifaili saab teha ka tavalise tekstiredaktoriga ning tulemus salvestada JSON formaadis.

ESRI ArcMap'i hetkel väljas olevad versioonid ei toeta vektortailide tegemist, küll aga on neid võimalik teha ArcGIS Pro'ga. Selleks on kaks võimalus, kasutades tööriista *Create Vector Tile Package* või siis läbi *Share* menüü. Jällegi saab kaardipüramiidi salvestada otse ArcGIS Online kontole või siis oma arvuti kõvakettale. Kujundus võetakse otse kaardiaknast ja pannakse failina kaardipüramiidile kaasa. Kõvakettale salvestades tekib VTPK laiendiga fail, mille saab *Extract package* tööriistaga lahti pakkida, ning tulemuseks on PBF (p*rotocol buffers*) laiendiga tailid (ArcGIS Pro, 2017). ESRIl on ka oma vektortailide kujundamise keskkond[3], mis on ArcGIS Online põhine, mis tähendab, et sinna peab sisse logima ja seal saab modifitseerida ainult oma ArcGIS Online kontol olevaid kaardipüramiidide stiilifaile.

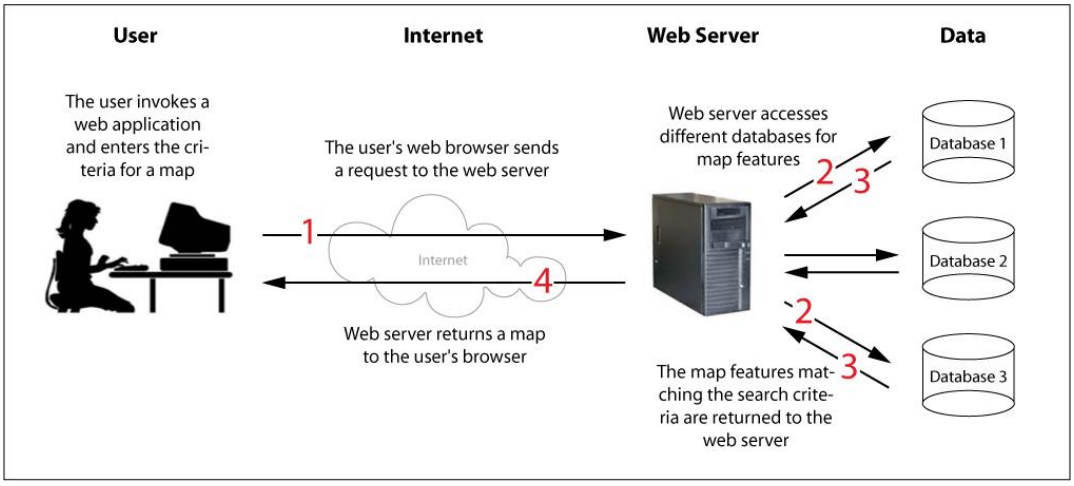
Geoserver toetab vektortailide tegemist läbi GeoWebCache'i, selleks tuleb panna ainult linnuke vastava variandi ette, samamoodi nagu rastertaile tehes. Mapserveri praegune versioon 7.0 vektortailide genereerimist ei toeta, küll aga plaanitakse see lisada versiooni 7.2 (MapServer, 2017).

2.5. WMS

WMS ehk *Web Map Service* on OGC poolt kirjeldatud standardne ruumiandmete vaatamise teenus. WMSi puhul teostatakse läbi kaardiserveri päring andmebaasi ning saadud andmed kuvatakse kas brauseri või mõne GIS tarkvara abil ekraanile (joonis 7). WMS'iga saab esitada terve aluskaardi, kuid see ei ole veebikaardi puhul väga otstarbekas, kuna erinevalt kaardipüramiidist, tuleb iga kord, kui kaarti liigutatakse, saata läbi serveri uus päring andmebaasi ning see teeb kaardi väga aeglaseks. Küll aga on WMSi hea kasutada kiiresti muutuva teemainfo korral, sest iga muudatus, mis tehakse andmebaasi, kajastub koheselt ka kaardil. Ehk siis aluskaart on tavaline kaardipüramiid ja teemainfo tuleb läbi WMSi. WMSi saab kasutada nii raster- (näiteks Maa-ameti WMS) kui ka vektorandmetega. Kaks

[3] http://esri.github.io/ArcGIS-vectortile-style-editor/#

populaarsemat kaardiserverit selle jaoks on GeoServer ja MapServer, mis lisaks muudele failitüüpidele toetavad ka *shapefile'*e.



Joonis 7. WMS'i tööpõhimõte (CartouCHe, 2012).

3. Veebilehtede loomiseks kasutatavad keeled

Veebilehtede loomise oskus on veebikaartide tegemise eelduseks, kuna loodavad kaardid paigutatakse veebilehele. HTML, CSS, Javascript ja PHP on neli väga levinud komponenti, millega veebilehti tehakse ja kõik nad on tasuta. Veebilehti saab luua ja vaadata oma arvutis. Selleks, et oma veebilehti teistele näidata, on aga vaja veebiserverit ja domeeninime. Veebilehtede ja -kaartide koodi kirjutamiseks saab kasutada tavalist tekstiredaktorit, näiteks Notepadi, salvestades faili vajaliku laiendiga (.html, .css, .js, .php). Selleks, et nende tegemine oleks lihtsam, on loodud spetsiaalseid tarkvarasid (Atom, Notepad++, Sublime Text, Brackets). Veebilehtede ja –kaartide tegemiseks ei pea tingimata ise koodi kirjutama. Selleks on paljud teenusepakkujad (ESRI, Mapbox, Google) loonud veebikeskkonnad, kus saab valmistehtud mallide abil veebikaarte teha. Koodi kirjutamise oskus annab aga paindlikkuse teha erilahendusi.

HTML (*Hyper Text Markup Language*) on hüperteksti markeerimiskeel, mille abil tehakse valmis veebilehtede struktuur (HTML Introduction, 2018).

CSS (*Cascading Style Sheets* – kaskaadilaadistik) on stiliseerimiskeel, mille abil määratakse veebilehtede kujundus: värvid, teksti suurus ja font, kõrgused, laiused, paigutus (CSS Introduction 2018).

JavaScript on objektorienteeritud programmeerimiskeel, millega saab veebilehe teha interaktiivseks (JavaScript Introduction, 2018) – panna asju liikuma hiirekliki peale, menüüd avanema ja palju muud, ka aluskaardi ja kaardikihtide brauseris nähtavaks tegemine käib JavaScriptiga.

PHP (*Hypertext Preprocessor*) on serveripoolne skriptimiskeel, mida kasutatakse dünaamiliste veebilehtede loomiseks. Veebilehe kood ja sisu genereeritakse reaalajas läbi andmebaasi PHP-d kasutades (PHP 5 Introduction, 2018). Kaardirakenduste puhul saab näiteks PHP-d kasutades pärida andmebaasist kaardiandmeid ja neid seejärel brauseris näidata.

4. Töös kasutatavate tarkvarade valik

Tehnoloogia muutub pidevalt ja tarkvarasid, mille vahel valida, on väga palju ning neid tuleb pidevalt juurde. Seetõttu on keeruline leida sellist tarkvara, mille õpetus ja kasutus säiliks pikemat aega adekvaatsena. Sellele probleemile otsisid lahendust ka Roth *et.al* (2014) oma uurimuses, kus nad proovisid leida võimalusi, kuidas pidevalt arenevas veebikaartide tegemise tehnoloogiates järge pidada, et kasutada saadud tulemusi ülikoolis kaardirakenduste tegemise õpetamiseks. Peamiselt keskendusid nad kaardi API-dele ja leidsid, et 16-st API-st kõige sobilikum ülikooli kursuse jaoks on Leaflet, kuna seda on kõige lihtsam õppida. Sellel on olemas hea dokumentatsioon, tugev toetus kaardirakenduste arendajate kogukonna poolt ning see on pidevalt arenev, mistõttu on lootust, et Leaflet peab vastu tehnoloogia survele ja leiab kasutust ka edaspidi. Samadele järeldustele jõudis ka Farkas (2017).

Ruumiandmebaasidest on ülevaate teinud Nair *et. al* (2015), kes leidsid, et kolme populaarsema vabavaralise andmebaasi (PostgreSQL/PostGIS, MySQL, SpatialLite) seast sobib PostgreSQL/PostGIS kõige laiema valiku rakenduste jaoks, mistõttu on see ka väga populaarne. PostgreSQL/PostGIS on kasutusel ka Tartu Ülikoolis, kus seda õpetatakse ka üliõpilastele, seega on antud töös võetud kasutusele just PostgreSQL/PostGIS. Selleks, et PostgreSQL-st teha ruumiandmebaas, on vaja PostGIS laiendust, mille saab installida koos PostgreSQL-ga.

Veebiserveriks osutus WAMP serverisse integreeritud vabavaraline Apache, kuna see on kõige populaarsem veebiserver olnud juba pikka aega (Netcraft, 2018). WAMP server on Windowsi veebiarenduse keskkond, mis sisaldab kahte töös vajaminevat tarkvara – lisaks Apache-le ka PHP-d, mida kasutatakse andmebaasiga suhtlemiseks, ning olemas on ka PHP laiendus, et ühenduda PostgreSQL baasiga. WAMP server on ka lihtsasti installitav. Kõik see teeb sellest hea valiku alustavale veebiprogrammeerijale. Linuxi ja MAC OS kasutajad saavad kasutada sarnastel põhimõtetel tehtud LAMP ja MAMP või XAMPP serverit. Järgnevas peatükis olevad praktilised näited põhinevad siiski ainult Windowsi operatsioonisüsteemil.

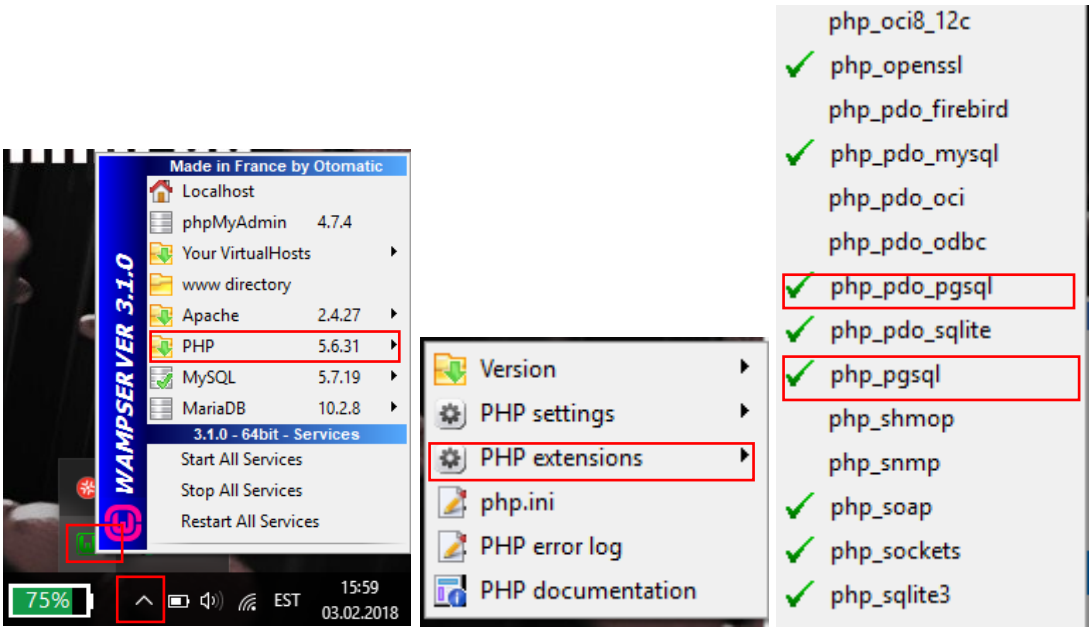
Kaardiserveritest kasutatakse töös GeoServerit, kuna ka seda juba õpetatakse Tartu Ülikoolis ja on seega tudengitele tuttav. GeoServer on ka mõnevõrra lihtsam, kuna ei pea kirjutama Mapfaili nagu MapServeris, kogu töö saab tehtud graafilise kasutajaliidese kaudu.

Samasugust tarkvarapaketti on kasutanud ka Brovelli *et. al* (2016) ja Brovelli *et. al* (2014), seega ei ole see sobilik ainult õppimiseks vaid sobib ka päriselt kasutusse minevate rakenduste jaoks.

5. Praktilised näited

Praktiliste näidete tegemisel on osalt aluseks võetud Roth *et. al* (2014) uurimuses välja toodud oskused. Käesoleva töö autori poolt on lisatud suhtlus andmebaasiga, kasutajate poolt andmete lisamine andmebaasi läbi rakenduse ja lühima tee leidmine ühest punktist teise, kuna need on väga levinud kaardirakenduse osad.

Alustuseks tuleks oma arvutisse installida WAMP server (http://www.wampserver.com/en/#download-wrapper), Geoserver (http://geoserver.org/) ja soovi korral ka Leaflet (http://leafletjs.com/download.html), salvestada kausta C:\wamp64\www\leaflet_naidised\src\leaflet. Tuleb jälgida, et WAMP serveri ja Geoserveri pordid oleksid erinevad. Leafleti kasutamiseks on kaks võimalust, kas oma arvuti kõvakettalt või läbi sisuedastusvõrgu, viidates koodis internetiaadressile, kus Leaflet asub. WAMP server tuleb käivitada ja sisse lülitada PHP laiendid php_pdo_pgsql ja php_pgsql, et saaks ühenduda PostgreSQL baasiga (joonis 8). Samuti oleks vaja mõnda vabalt valitud tekstiredaktorit (Notepad, Notepad++, Sublime Text, Brackets, Atom vms) kuhu koodi kirjutada. Andmete töötlemiseks ja andmebaasi viimiseks kasutatakse QGIS-i.



Joonis 8. PHP laiendite sisse lülitamine WAMP serveris.

5.1 Leafletiga aluskaartide lisamine

Selleks, et Leafleti kasutada, tuleb kõigepealt HTML koodi lisada Leafleti teek ja stiilifail. Tükkhaaval täiendatav kood on täispikkuses ära toodud lisas 1. Koodi read on nummerdatud nii, et kui keegi tahab koodi kaasa kirjutada, siis iga järgnev koodi lõik tuleks sisestada HTML dokumendis numbriga näidatud reale.

```
 1 <!DOCTYPE html>
 2 <html>
 3  <head>
 4    <title>Minu kaart</title>
 5    <meta charset="utf-8" />
 6    <meta name="viewport"
           content="width=device-width, initial-scale =1.0">
 7    <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
 8    <script src="src/leaflet/leaflet.js"></script>
 9  </head>
10
11  <body>
12  </body>
13 </html>
```

Leafleti lisamine sisuedastusvõrgu kaudu näeks välja järgmine:

```
 7    <link rel="stylesheet"
 8          href="https://unpkg.com/leaflet@1.3.1/dist/leaflet.css"
 9          integrity="sha512-Rksm5RenBEKSKFjgI3a41vrjkw4EVPlJ3+OiI65vTjIdo
9brlAacEuKOiQ5OFh7cOI1bkDwLqdLw3Zg0cRJAAQ==" crossorigin=""/>
10    <script src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js"
11            integrity="sha512-/Nsx9X4HebavoBvEBuyp3I7od5tA0UzAxs+j83KgC8P
U0kgB4XiK4Lfe4y4cgBtaRJQEIFCW+oC506aPT2L1zw==" crossorigin=""></script>
```

Järgmiseks tuleks HTML dokumendis reserveerida kaardile koht, seda saab teha kasutades *div* silte ja omastades sellele *div*'ile ID. Seejärel on aeg JavaScripti käes. Selleks, et kaarti veebilehel näidata, tuleb luua kaardi objekt (Leafleti puhul *L.map*), ja seejärel kaardikihi objekt (*L.tileLayer*), milleks praegu on Open Street Map'i (OSM) rasterkaardipüramiid koos viitega (*attribution*) OSM'ile. Lõpuks tuleb see kiht lisada kaardile (*.addTo(kaart)*), *kaart.setView* abil saab määrata kaardi keskpunkti ja suumiastme.

```
12    <div id="kaart"></div>
13
14    <script type="text/javascript">
15      const kaart = L.map('kaart');
16      const OSM = L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
17        attribution: '&copy; <a href="http://osm.org/copyright">
OpenStreetMap</a> contributors'
18       }).addTo(kaart);
```

```
19        kaart.setView({ lat: 58.7, lng: 25.0 }, 8);
20    </script>
```

Selleks, et kaart nähtavale tuleks, peab määrama *div* objektile stiili, kasutades identifikaatorit. Alloleva koodi korral tuleb kaart terve HTML lehe suurune. Selleks, et lahti saada valgetest äärtest ja kerimisribast, tuleb omistada sama stiil ka tervele HTML failile ja *body* osale.

```
 9    <style>
10      #map, html, body {
11        width:100%;
12        height:100%;
13        margin: 0;
14      }
15    </style>
16  </head>
```

Tihti on veebikaartidel ühe kihina olemas ka satelliidipilt või ortofoto. ESRI pakub oma sateliidipiltide aluskaarti (lisaks teistele aluskaartidele) tasuta kasutamiseks ja ka siin töös on see kasutusele võetud. Kuna pildid on saadud paljudelt organisatsioonidelt, siis on ka viidete loend (*pildid_attr*) päris pikk[4].

```
28      const esri ='<a href="http://www.esri.com/">Esri</a>';
29      const pildid_attr = 'i-cubed, USDA, USGS, AEX, GeoEye, Getmapping,
Aerogrid, IGN, IGP, UPR-EGP,\and the GIS User Community';
30
31      const sat = L.tileLayer ('http://server.arcgisonline.com/ArcGIS/
rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}', {
32        attribution: '&copy; ' + esri + ', ' + pildid_attr
33      });
```

Selleks, et saaks kihte sisse ja välja lülitada, tuleb lisada kihiloend ja sinna omakorda kaks praegu olemasolevat kihti. Kihiloendi kujundust saab samuti muuta, kirjutades üle leaflet.css failis oleva kujunduse enda omaga või modifitseerides otse leaflet.css faili.

```
35      L.control.layers({
36        Kaart: OSM,
37        Satelliit: sat
38     }, {}, {collapsed: false}).addTo(kaart);
39    </script>
```

Nüüd saab testida, kas veebileht töötab. Kood tuleks salvestada .html laiendiga WAMP serveri www kausta, näiteks: C:\wamp64\www\leaflet_naidised\kaart.html. Kui kõik läks hästi, peaks brauserisse kirjutades http://localhost:8081/leaflet_naidised/kaart.html tulema

[4] https://leanpub.com/leaflet-tips-and-tricks/read#leanpub-auto-mapquest-open-aerial

nähtavale joonisel 9 näidatud pilt, kus saab valida kahe aluskaardi vahel, kaarti nihutada ja suumida. Kui sellist pilti ei näe, tuleks viga tuvastada brauseri arendaja tööriistu kasutades.

Joonis 9. Leafleti ja WAMP serveriga tehtud kaart kahe aluskaardiga.

5.2 Vektorkaardipüramiidi lisamine

Vektortailide lisamise näitamiseks kasutatakse võimalust salvestada kaardipüramiid oma arvuti kõvakettale. Vektortaile selle töö raames tehtud ei ole, vaid need on alla laetud Open Map Tiles veebilehelt[5] .mbtails formaadis, mis on üks enam levinud viise vektortailide hoiustamiseks. Vektortaile Leafleti põhifunktsionaalsus ei toeta, küll aga on praeguse kõige uuema versiooni jaoks loodud moodul Leaflet.VectorGrid, mis saab hakkama GeoJSON, TopoJSON ja ka PBF formaadis vektortailidega. Faililaiend MBTiles'i puhul on tegu SQLite andmebaasiga, mis hoiustab kaardipüramiidi taile .pbf formaadis kokkupakituna. Selleks, et tailid lahti pakkida, kasutatakse käsuviibapõhist programmi mbutil[6]. Käsuviibale sisestatakse järgnev käsk, kus tuleb näidata pythoni asukoht, mbutil asukoht, tailide formaat, tailide asukoht ja koht kuhu tailid salvestada:

```
C:\Users\Laptop\AppData\Local\Programs\Python\Python36-32\python.exe C:\mbutil\mb-util --image_format=pbf C:\Users\Laptop\Documents\kaart\estonia.mbtiles C:\wamp64\www\leaflet_naidised\andmed\vektor_aluskaart
```

[5] https://openmaptiles.com/downloads/dataset/osm/europe/estonia/#5.85/58.77/24.533

[6] https://github.com/mapbox/mbutil

Väljundiks on kaustadesse jaotatud kaardipüramiid, igas kaustas on .pbf laiendiga tailid. Enne kasutamist tuleb need veel omakorda lahti pakkida. Selleks kasutatakse käsuviibapõhist programmi GZIP[7]. GZIP tuleb lisada ka keskonnamuutujatesse, et seda saaks käsuviibaga kasutada. Selleks tuleb minna Juhtpaneel→ Kõik juhtpaneeli üksused → Süsteem → Täpsemad süsteemisätted → Keskkonnamuutujad →lisada muutujale "*path*" juurde GZIP-i asukoht (näiteks C:\Program Files (x86)\GnuWin32\bin). Seejärel liikuda käsuviibal asukohta, kuhu kaardipüramiid sai lahti pakitud ja sisestada järgnev käsk, mis käib läbi kõik kaustad ja pakib nendes olevad tailid lahti:

```
gzip -d -r -S .pbf *
```

Nüüd omistatakse tailidele uuesti .pbf faililõpp, et *Leaflet.VectorGrid* moodul teaks, millega tegemist on. Selleks sisestatakse käsureale järgnev käsk:

```
for /R %G in (*.) DO REN "%G" "%~nG.pbf"
```

Vektortailide näitamiseks tuleb lisada Leaflet.vectorGrid teek.

```
9       <script src="https://unpkg.com/leaflet.vectorgrid@latest/dist/
Leaflet.VectorGrid.bundled.js"></script>
```

Seejärel defineeritakse muutuja *vektorTailid*, millele omistatakse *L.vectorGrid.protobuf* funktsioon, millele lisatakse .pbf formaadis vektortailide asukoht. Kuna vektortailidel pole kujundust, siis tuleb real 40 omistada ka stiil.

```
37      const url = "andmed/vektor_aluskaart/{z}/{x}/{y}.pbf";
38      const vektorTailid = L.vectorGrid.protobuf(url, {
39        attribution: '<a href="https://openmaptiles.org/">&copy;
OpenMapTiles</a>, <a href="http://www.openstreetmap.org/copyright">&copy;
OpenStreetMap</> contributors',
40         vectorTileLayerStyles: vektorTailidStiil,
41        });
```

Stiil tuleb aga eelnevalt luua, selleks võib teha uue faili, näiteks stiil.js (lisa 2), kus on iga nähtuseklassile omistatud kujundus. Stiilifailile tuleb viidata HTML dokumendis.

```
10       <script src="stiil.js"></script>
```

Kuna tegemist on vektoriga, saab sellega teha päringuid, näiteks kuvada hiirega tee peale klikkides tee nime. Selleks tuleb alustuseks antud moodulis teha vektortailid interaktiivseks, lisades vastava rea vektorkaardipüramiidi kihile (*interactive: true*). Selleks, et klikkimisel

[7] http://gnuwin32.sourceforge.net/packages/gzip.htm

ilmuks hüpikaken, tuleb lisada käsitleja, mis on näidatud järgneva koodilõigu 42. real. Hüpikakna sisu on määratud real 43, antud juhul näitab nähtuse nime, kui see on nähtusel olemas (teed, tänavad, jõed). *SetLatLng* ütleb, et hüpikaken peaks ilmuma sinna, kuhu hiirega klikiti ja *.openOn(kaart)* avab selle kaardil.

```
40          interactive: true
41        }).on('click', function(e) {
42          L.popup()
43          .setContent(e.layer.properties.name)
44          .setLatLng(e.latlng)
45          .openOn(kaart);
46        });
```

Uus aluskaart lisatakse ka kihiloendisse.

```
50          Vektor: vektorTailid,
```

Nüüd klikkides kaardil, näiteks Tallinn-Tartu maanteel, ilmub hüpikaken vastava sisuga ja stiilifailis muudatusi tehes muutub ka aluskaart.

5.3 Teemakihtide lisamine

Pärast aluskaartide lisamist lisatakse kaardile teemainfo kihid. Kõige lihtsam viis selleks on kasutada geoJSON andmeformaati, mille saab HTML dokumenti linkida. Antud näites on kasutatud Keskkonnaregistrist võetud infot (nimi, koordinaadid) looduskaitseliste objektide kohta (rändrahnud, puud, allikad, pinnavormid), mis on salvestatud Exceli faili ning QGISi abil eksporditud geoJSON formaati. Lisame selle info HTML dokumenti, nimega objektid.js, mis asub *andmed*-nimelises kaustas. GeoJSON-i fail nimega objektid.js lingitakse HTML dokumenti.

```
11        <script src="andmed/objektid.js"></script>
```

Selleks, et seda faili saaks kasutada, tuleb objektid.js faili alguses öelda, et kogu faili sisu on muutuja.

```
1 var objektid ={
2 "type": "FeatureCollection",
3 "crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:EPSG::4180" } },
4 "features": [
5 { "type": "Feature", "properties": { "nimi": "Aasumägi", "X": 6539645,
    "y": 558110, "tyyp": "pinnavorm" }, "geometry": { "type": "Point",
    "coordinates": [ 25.010894454618807, 58.991843566257103 ] } },
6 ...
```

Kuna kaardile kantavaid objekte on palju, siis grupeeritakse objektid väiksemas suumiastmes kokku. Selleks kasutatakse moodulit *MarkerCluster*, mis tuleb samuti siduda HTML dokumendiga. *MarkerCluster* pakub väga palju huvitavaid võimalusi punktide klasterdamiseks, kuid siinkohal on piirdutud kõige põhilisemaga.

```
 8    <link rel="stylesheet"
href="src/marker_cluster/MarkerCluster.Default.css">
```

```
11    <script src="src/marker_cluster/leaflet.markercluster.js"></script>
```

HTML dokumendis omistatakse igale looduskaitselise objekti tüübile oma ikoon, millega seda objekti kaardil kujutada.

```
52      var ikoonid = {
53        rändrahn: L.icon({
54          iconSize: [32, 21],
55          iconAnchor: [16, 21],
56          popupAnchor:  [0, -21],
57          iconUrl: 'pildid/kivi.png'
58        }),
59        allikas: L.icon({
60          iconSize: [32, 32],
61          iconAnchor: [16, 32],
62          popupAnchor:  [0, -32],
63          iconUrl: 'pildid/allikas.png'
64        }),
65        puu: L.icon({
66          iconSize: [30, 30],
67          iconAnchor: [15, 30],
68          popupAnchor: [0, -30],
69          iconUrl: 'pildid/puu.png'
70        }),
71        pinnavorm: L.icon({
72          iconSize: [38, 21],
73          iconAnchor: [19, 21],
74          popupAnchor: [19, -21],
75          iconUrl: 'pildid/pinnavorm.png'
76        })
77      };
```

Igale objektile hüpikakna lisamiseks luuakse funktsioon *onEachFeaturePopup* ning hüpikakna sisuks saab objekti nimi (*feature.properties.nimi*).

```
79      function onEachFeaturePopup(feature, layer) {
80        layer.bindPopup(feature.properties.nimi);
81      }
```

Seejärel tehakse uus geoJSON kiht ja omistatakse sellele äsja loodud ikoonid ja funktsioon. Funktsioon *onEachFeature* real 90 on Leafleti enda funktsioon, mida on hea kasutada hüpikakende tegemiseks, sest seda kutsutakse välja iga nähtuse korral. Ridadel 85-86 näidatakse, milline ikoon millise nähtusega tuleb siduda.

```
83        const loodusObjekti = new L.GeoJSON(objektid, {
84          pointToLayer: function (feature, latlng) {
85            marker = new L.marker(latlng, {
86              icon: ikoonid[feature.properties.tyyp]
87            });
88            return marker;
89          },
90          onEachFeature: onEachFeaturePopup
91        });
```

Järgnevad read tekitavad markerite klastri, lisavad geoJSON kihi sinna klastrisse ja lõpuks lisatakse see kiht kaardile.

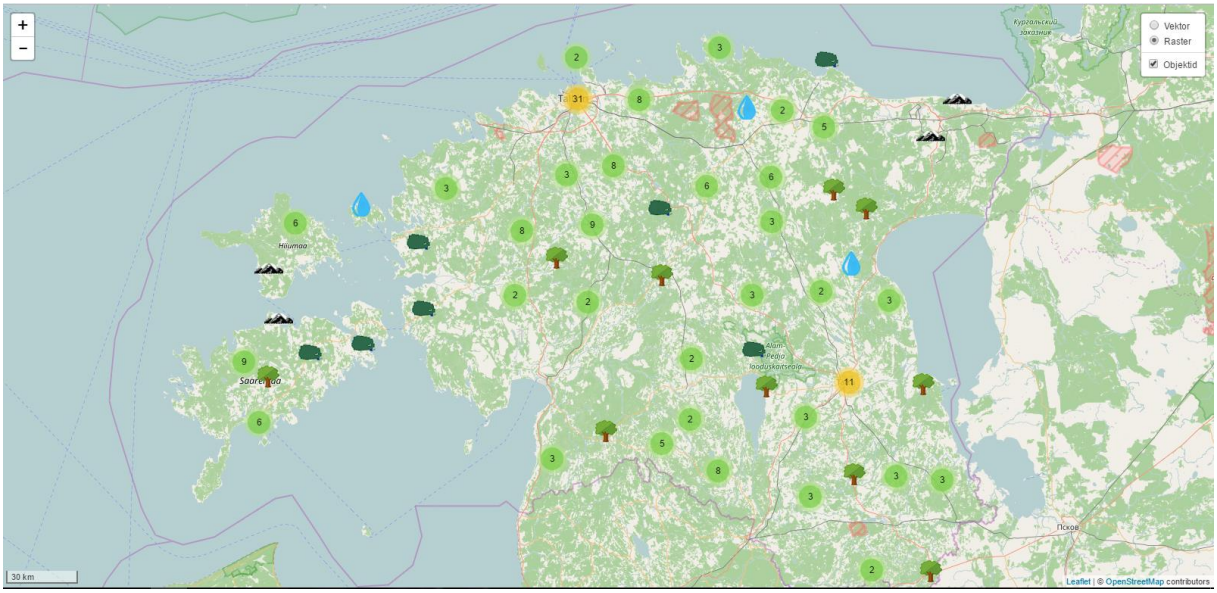
```
93      const klastrid = L.markerClusterGroup({
94        showCoverageOnHover: false,
95        maxClusterRadius: 50
96      });
97
98      klastrid.addLayer(loodusObjektid);
99      kaart.addLayer(klastrid);
```

Selleks, et teemakihi sisselülitamisel ei kaoks aluskaart ära, peab kihiloendit natuke ümber tegema. Selleks tehakse eraldi kihiloend aluskaartide ja teemainfo jaoks.

```
101       const teemainfo = {
102         Objektid: markers
103       };
104
105       const aluskaardid = {
106         Vektor: estoniaVectorLayer,
107         Raster: OSMLayer,
108         Satelliit: sat
109       };
110
111       const kihiloend = L.control.layers(aluskaardid, teemainfo)
.addTo(map);
112
113     </script>
```

Kaardil peaks olema ka mõõtkava. Sulgudesse on lisatud *imperial: false*, et mõõtkava oleks kilomeetrites. Tulemus on näha joonisel 10.

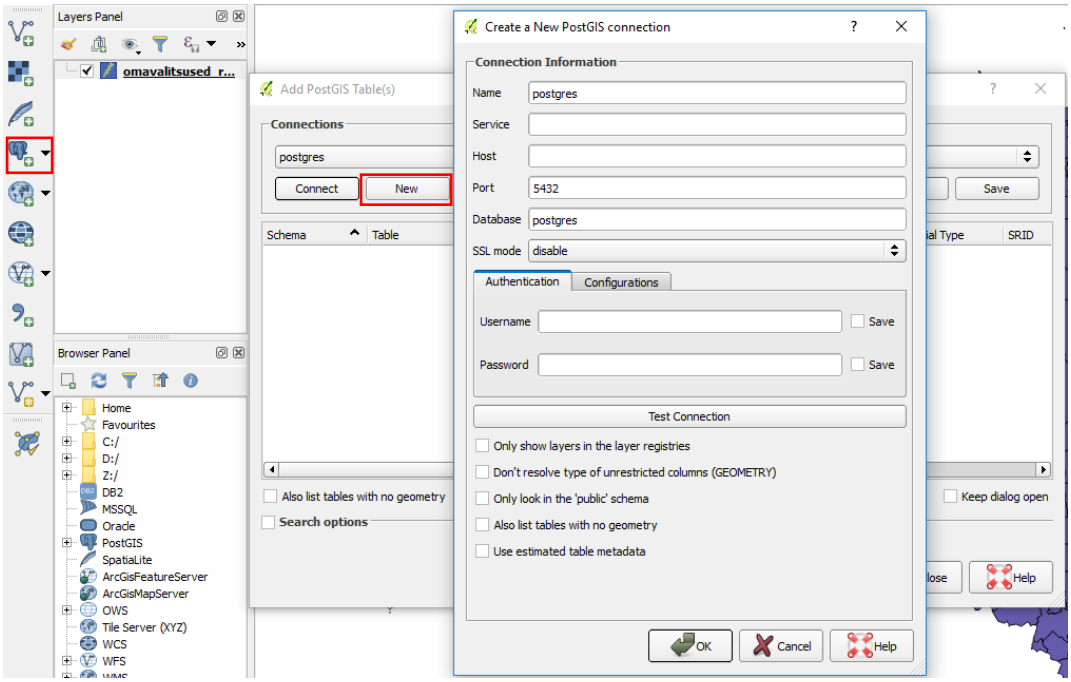
```
113       L.control.scale({imperial: false}).addTo(kaart);
```



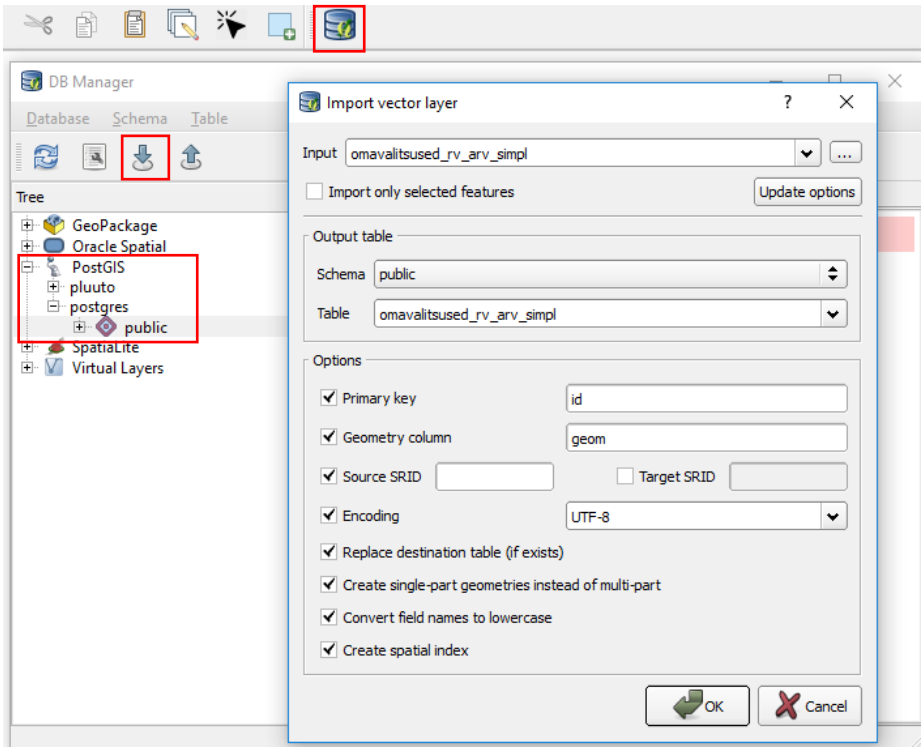
Joonis 10. Leafletiga tehtud kaart, looduskaitseliste objektide kohta.

5.4 WMS kihi lisamine

WMS kihi lisamiseks tehti horopleetkaart Eesti omavalitsuste rahvaarvu jaotusest. Selleks seoti Maa-ametist alla laetud Eesti haldusjaotuse shapefail ja statistikaametist saadud omavalitsuste rahvaarvud. Saadud kiht viidi QGIS-i abil PostgreSQL/PostGIS andmebaasi. Selleks loodi ühendus PostgreSQL/PostGIS baasiga (joonis 11) ning seejärel lisati kiht baasi (joonis 12).

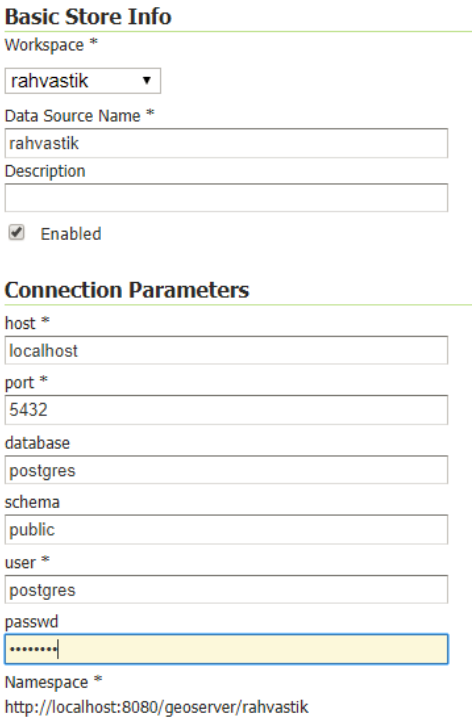


Joonis 11. QGIS-i ühendamine PostgreSQL/PostGIS andmebaasiga.



Joonis 12. Läbi QGIS-i PostgreSQL/PostGIS andmebaasi kihi lisamine.

PostgreSQL/PostGIS andmebaas seoti omakorda Geoserveriga. Selleks lisati uus töökaust nimega „rahvastik“ ja uus PostGIS andmeladu (joonis 13). Rahvastiku kiht kujundati rahvaarvu järgi klassidesse kasutades SLD-d (*Styled Layer Descriptor*) (lisa 3).



Joonis 13. Geoserveri ühendamine PostgreSQL/PostGIS andmebaasiga.

Veebikaardile lisati OSM-i aluskaart ja seejärel rahvastiku kiht WMS-na GeoServerist, kus „rahvastik“ on töökausta nimi ja „omavalitsused_rv_arv_simpl“ on kihi nimi. Kogu kood on toodud lisas 4.

```
31      const wmsKiht = L.tileLayer.wms(
'http://localhost:8082/geoserver/rahvastik/wms?',{
32        layers: 'rahvastik:omavalitsused_rv_arv_simpl',
33        format: 'image/png',
34        transparent: true
35      }).addTo(kaart);
```

Kaardile lisatakse ka legend, mis näitab, mida mingi värv tähendab.

```
37      function getColor(d) {
38        return d > 500000 ? '#800026' :
39               d > 95000  ? '#BD0026' :
40               d > 40000  ? '#E31A1C' :
41               d > 18000  ? '#FC4E2A' :
42               d > 8000   ? '#FD8D3C' :
43               d > 3000   ? '#FEB24C' :
44                            '#FED976';
45      }
46
47      const legend = L.control({position: 'bottomright'});
48      legend.onAdd = function(kaart){
49
50        const div = L.DomUtil.create('div', 'legend');
51        const vaartused = [0, 3000, 8000, 18000, 40000, 95000];
52
53        for (var i = 0; i < vaartused.length; i++) {
54          div.innerHTML +='<i style="background:'+getColor(vaartused[i]+1
)+'"></i>'+vaartused[i]+(vaartused[i+1]?'&ndash;'+vaartused[i+1]+' in
<br><br>':'+ in');
55        }
56          return div;
57      };
58      legend.addTo(kaart);
```

Ning kui omavalitsuste kiht on välja lülitatud, siis peaks ka legend olema välja lülitatud.

```
60      kaart.on('overlayadd', function(eventLayer){
61        if (eventLayer.name === 'Rahvastik'){
62          kaart.addControl(legend);
63        }
64      });
65
66      kaart.on('overlayremove', function(eventLayer){
67        if (eventLayer.name === 'Rahvastik'){
68          kaart.removeControl(legend);
69        }
```

```
70        });
```

Kihid lisatakse kihiloendisse ja kaardile lisatakse mõõtkava.

```
72      const teemainfo = {
73        Rahvastik: wmsKiht,
74      };
75
76      const aluskaardid = {
77        Kaart: OSM,
78      };
79
80      const kihiloend = L.control.layers(aluskaardid, teemainfo,
{collapsed: false}).addTo(kaart);
81
82      L.control.scale({imperial: false}).addTo(kaart);
```

Seejärel kujundatakse legendikast

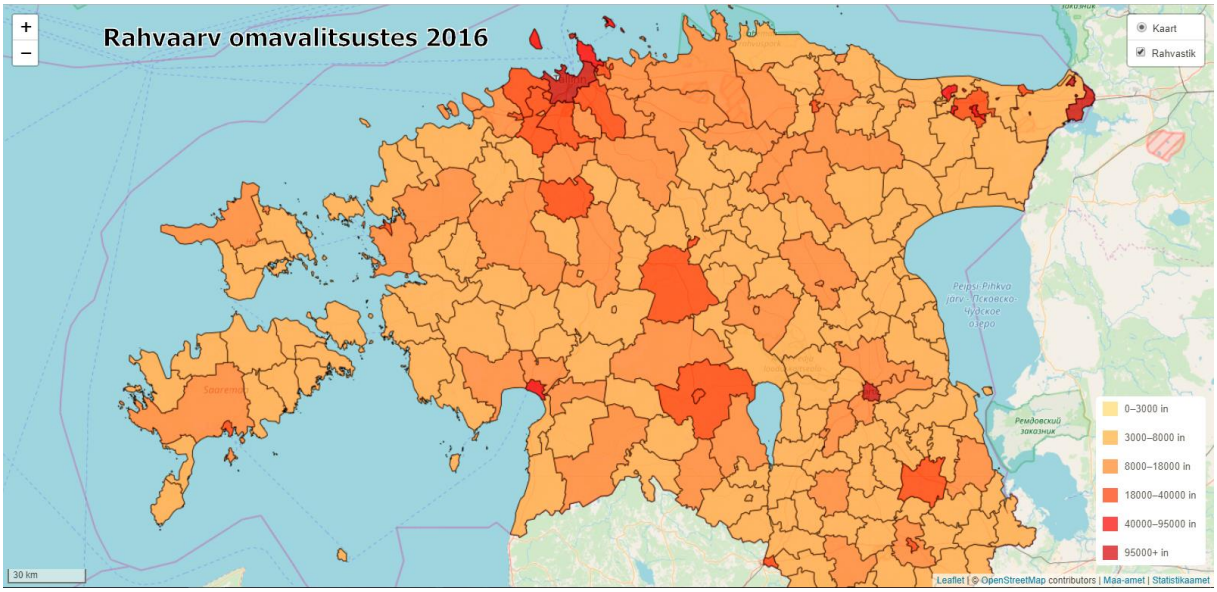
```
15        .legend {
16           padding: 6px 8px;
17           line-height: 18px;
18           color: #555;
19           background: white;
20           border-radius: 5px;
21         }
22         .legend i {
23           width: 18px;
24           height: 18px;
25           float: left;
26           margin-right: 8px;
27           opacity: 0.7;
28         }
```

Kaardile lisatakse ka pealkiri ja kujundatakse vastavalt.

```
45      <div id="pealkiri">
46        Rahvaarv omavalitsustes 2016
47      </div>
```

```
31      #pealkiri{
32        position: absolute;
33        z-index: 10;
34        left: 120px;
35        top: 20px;
36        font-family: "Tahoma";
37        font-size: 30px;
38        font-weight: bold;
39        -webkit-text-stroke: 1px white;
40      }
```

Tulemus on näha joonisel 14 ning kui andmebaasis muuta mõne omavalitsuse rahvaarvu või kuju, siis muutub see kohe ka kaardil. Kaardil sisse suumides tulevad nähtavale ka omavalitsuste nimed.



Joonis 14. Leafletiga tehtud kaart, kus on omavalitsuste kiht värvitud rahvaarvu järgi.

5.5 Aadressotsing

Üks veebikaardi peaaegu et lahutamatu osa on aadressotsing ehk geokodeerimine, mis võib olla kahepidine. Kasutaja sisestab aadressi, millele otsitakse vaste andmebaasist või siis kasutaja klikib kaardile ja talle näidatakse sellele kohale vastavat aadressi või lähimat aadressi. Viimast varianti nimetatakse tagurpidi geokodeerimiseks (ingl k *reverse geocoding*).

Leafletile on loodud hulgaliselt pluginaid, mis seda võimekust pakuvad. Portaalis Stackoverflow.com soovitatakse enamasti Leaflet Control Search või Lealfet Control Geocoder pluginaid. Viimane toetab paljusid andmepakkujaid (Google, OSM, Bing jne) ja tagurpidi geokodeerimist, heaks küljeks võib veel lugeda, et otsingut saab piirata konkreetse riigi või piirkonnaga. Leaflet Control Search võimaldab aga kasutada väga lihtsasti ka oma andmed näiteks geoJSON formaadis või enda andmebaasi. Mõlema plugina kohta on hulgaliselt häid näiteid. Muidugi saab ka otse ESRI[8] või Mapzeni[9] otsingumootorit kasutada ilma vahe-pluginata.

[8] https://github.com/Esri/esri-leaflet-geocoder
[9] https://mapzen.com/documentation/search/

Järgnevalt aga ei kasutata olemasolevaid pluginaid, vaid ehitatakse otsingumootor ise. Lisateegina kaasatakse jQuery Autocomplete[10], mille abil saab tulemusi kuvada lennult, kui kasutaja midagi otsingukasti sisestab. Aadressandmetena kasutatakse Maaameti poolt avalikuks tehtud aadressandmeid, mis on allalaaditavad INSPIRE'i lehelt[11] maakondade kaupa shape-formaadis. Need on kokku liidetud üheks, arvutatud X ja Y koordinaadid eraldi veergudesse ning viidud PostgreSQL/PostGIS andmebaasi. Terve kood on toodud lisas 5.

Kõigepealt lisatakse jQuery ja jQuery-ui (kasutajaliidese) teegid.

```
 8    <script src="http://code.jquery.com/jquery-1.12.4.js"></script>
 9    <script src="http://code.jquery.com/ui/1.12.1/jquery-ui.js">
</script>
```

Seejärel lisatakse otsingukast *body* osa algusesse kohe peale kaardi *div*'i.

```
23     <form id="otsing">
24       <input type="text" id="otsi" name="addr" placeholder="otsi...">
25     </form>
```

Otsingukast paigutatakse kaardi vasakule üles nurka.

```
18      #otsing{
19        position: absolute;
20        z-index: 10;
21        left: 100px;
22        top: 20px;
23      }
```

Seejärel on järg JavaScripti ja jQuery käes. Lisatakse *autocomplete* funktsioon, kus „#otsi" tähistab eelnevalt tehtud otsingukasti ID-d, „otsing.php" on PHP fail, mis tagastab aadressid ja mille tegemiseni kohe ka jõutakse, vastete otsimine hakkab pärast kolmanda tähe trükkimist ja et ressurssi kuluks vähem, peab pärast trükkimist ootama pool sekundit enne, kui vasted kuvatakse. „*append to: '#otsing'*" ütleb, et vastete nimekiri ilmuks otsingukasti külge. See aga ei kuva veel leitud aadressi kaardil. *Select* atribuut saab aru, kui mingi aadress nimekirjast on valitud ja sellele omistatud funktsiooni abil saab kontrollida, mis edasi juhtub.

```
42      $(function() {
43        $("#otsi").autocomplete({
44          source: "php/otsing.php",
45          minLength: 3,
46          delay: 500,
47          appendTo: '#otsing',
48          select: function(event, ui) {
```

[10] https://jqueryui.com/autocomplete/

```
49            }
50          });
51        });
```

Funktsiooni sees kasutatakse AJAX'it[12] (Asynchronous JavaScript And XML), et teha uus päring andmebaasi, kus sisendiks on eelnevalt valitud vaste nimekirjast (*data:{addr: ui.item.label}*) ja väljundiks on selle aadressi koordinaadid. Päringu tegemiseks luuakse fail „koordinaat.php", väljundandmeformaat on JSON.

```
49              $.ajax({
50                url: "php/koordinaat.php",
51                dataType: "json",
52                data:{addr: ui.item.label},
53                success: function(data){
54                }
55              });
```

Kui andmebaasist on vaste leitud (rida 53), siis kuvatakse selle koha peale hüpikaken, mille sisuks on saadud aadress ning kaart suumitakse saadud kohani suumiastmel 15.

```
54                  popup = L.popup().setLatLng(data[0].loc)
55                                   .setContent(data[0].nimi)
56                                   .openOn(kaart);
57                  kaart.flyTo(data[0].loc, 15);
```

Selleks, et saadud vastete nimekiri ei ulatuks lõpmatuseni, muudatakse CSS'iga vastete kuvamise nimekirja vormingut ja eemaldatakse kiri, mis ütleb, mitu vastet leiti.

```
49          .ui-autocomplete{
50            max-height: 200px;
51            overflow-y: scroll;
52            overflow-x: hidden;
53            background: white;
54          }
55          .ui-helper-hidden-accessible {display:none}
```

Faili „otsing.php" abil päritakse PostgreSQL/PostGIS andmebaasist aadressandmed. Selleks peab olema WAMP serveris *php-pdo-pgsql* ja *php-pgsql* laiendid sisse lülitatud, et saaks ühenduse PostgreSQL andmebaasiga.

Kõigepealt defineeritakse muutujad hosti, pordi, andmebaasi nime, kasutajanime ja salasõna jaoks. Seejärel luuakse ühendus andmebaasiga ja testitakse, kas ühendus saavutati.

[11] http://inspire.maaamet.ee/inspire-teenused/aadressid-(ad)

```php
<?php

  $host        = "host = localhost";
  $port        = "port = 5432";
  $dbname      = "dbname = postgres";
  $credentials = "user = postgres password=postgres";

  $yhendus = pg_connect( "$host $port $dbname $credentials"  );
  if(!$yhendus) {
    echo "Error : Ei saa andmebaasiga ühendust ";
  }
```

Seejärel kontrollitakse, kas otsingukasti on midagi sisestatud („*term*“ on jQuery *autocomplete* teegi sisene sõna, mis tähistab otsingusõna) ja kui on, siis teostatakse SQL'i kasutades andmebaasist päring, kus „full_addre“ on veeru nimi, „aadressid“ tabeli nimi ja '%".$otsi."%' näitab, et vaste peaks olema sarnane otsitava sõnaga. Selline SQL päring ei ole kindlasti kõige optimaalsem viis tekstilise otsingu teostamiseks (näiteks on otsing praegu tõstutundlik), kuid praegusel juhul sellest piisab. Rohkem infot, kuidas tekstilist SQL otsingut teha, leiab PostgreSQL dokumentatsioonist[13].

```php
  if (!empty($_GET['term'])) {
    $otsi = $_GET['term'];
    $paring = "SELECT full_addre FROM aadressid WHERE full_addre LIKE
'%".$otsi."%'";
    $valjund = pg_query($yhendus, $paring);
```

While-tsükli abil lisatakse kõik saadud tulemused järjendisse ja *echo* väljastab JSON formaadis listi JQuery *autocomplete* teegile. Kaks viimast rida kustutavad päringu tulemuse ja sulgevad andmebaasi.

```php
    while($rida = pg_fetch_assoc($valjund)){
       $andmed[] = $rida['full_addre'];
     }
     echo json_encode($andmed);

     pg_free_result($valjund);
     pg_close($yhendus);
   }
?>
```

Koordinaat.php faili loogika on suhteliselt samasugune. Seekord otsitakse ühte kindlat vastet, mille kasutaja on eelnevalt vastete nimekirjast valinud. Koordinaatide veerust on vaja ka x- ja y-koordinaate.

[12] https://www.w3schools.com/xml/ajax_intro.asp

```php
<?php
  $host        = "host = localhost";
  $port        = "port = 5432";
  $dbname      = "dbname = postgres";
  $credentials = "user = postgres password=postgres";

  $yhendus = pg_connect( "$host $port $dbname $credentials"  );
  if(!$yhendus) {
    echo "Error : Ei saa andmebaasiga ühendust ";
  }

  $otsi = $_GET['addr'];
  $paring = "SELECT full_addre, xcoord, ycoord FROM aadressid WHERE full_addre='".$otsi."'";
  $valjund = pg_query($yhendus, $paring);
```

Väljundi vormistame geoJSON formaadis, et hiljem oleks lihtsam seda töödelda.

```php
  $geojson = array();
  $rida = pg_fetch_assoc($valjund);
  $feature = array(
    'loc' => array($rida['xcoord'], $rida['ycoord']),
    'nimi' => $rida['full_addre']);

  array_push($geojson, $feature);
  echo json_encode($geojson);

  pg_free_result($valjund);
  pg_close($yhendus);
?>
```

Sellega on aadressotsingu tegemine valmis. Antud lahendus ei ole küll kõige kiirem ja ilmselt mitte kõige optimaalsem, kuid sobib demonstreerimaks, kuidas väga lihtsate vahenditega teha kaardile aadressotsingut.

Tagurpidi geokodeerimine

Kõige lihtsam viis tagurpidi geokodeerimiseks punktandmete korral on kasutada lähima naabri meetodit ehk kui klikitakse kaardil, siis otsitakse andmebaasist lähim aadress/punkt klikitud kohale. See meetod ei ole kindlasti ideaalne, kuna katastriüksused, mille järgi aadresspunktid on Maa-ametis ilmselt genereeritud, ei ole kõik ühesuured ja ühe kujuga ning seetõttu võib anda „valesid“ tulemusi. Ideaalis võiks kasutada katastriüksuste polügoone ja võrrelda, millise polügooni sisse klikitud punkt langeb, kuid need ei ole Eestis avalikud

[13] https://www.postgresql.org/docs/current/static/textsearch.html

andmed. Kuna töös on kasutusel ainult Eesti aadressid, siis Eesti maismaa piiridest välja klikkides ei tohiks tulemust saada (kasutusel on ju lähima naabri meetod). Selle vältimiseks lisatakse Eesti-kujuline polügoon ja võrreldakse, kas punkt on polügooni sees või mitte. Eesti polügoon pannakse küll koodile kaasa, kuid ei lisata kaardile.

```
10    <script src="data/eesti.js"></script>
```

```
72    const eesti_polygon = L.geoJson(eesti);
```

Kontrollimaks, kas punkt langeb polügooni sisse, kasutatakse Mapbox'i tehtud teeki Leaflet-Pip.

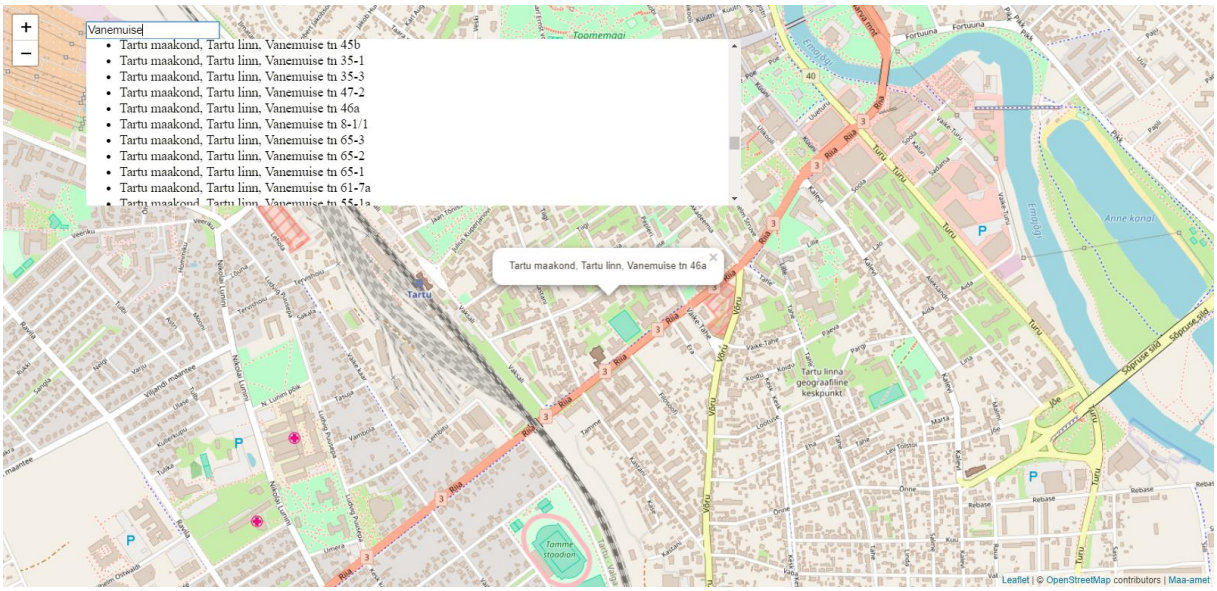
```
10    <script src="https://unpkg.com/@mapbox/leaflet-pip@1.1.0/leaflet-pip.
js"></script>
```

Kõigepealt luuakse uus muutuja, mis hakkab sisaldama leitud aadressi hüpikakent. Kaardil klikkimise tuvastamiseks luuakse „kuulaja" ning kui kaardil on juba üks hüpikaken, siis see eemaldatakse. Saadud koordinaadid salvestatakse *lat* ja *lng* muutujatesse. Muutuja „tulemus" kontrollib, kas punkt jääb Eesti maismaa sisse.Kui jääb, tehakse AJAXi päring andmebaasi, kus sisendiks on klikitud punkti koordinaadid. Edasi luuakse klikitud kohale hüpikaken aadressiga ja vajadusel suumitakse kaart sisse.

```
 78    var popup;
 79    var tagurpidi_otsing = kaart.on('click', function(ev) {
 80      if (popup) {
 81        kaart.removeLayer(popup);
 82      }
 83      var lat = ev.latlng.lat;
 84      var lng = ev.latlng.lng;
 85      var muutuja = leafletPip.pointInLayer([lng, lat], eesti_polygon);
 86      if(muutuja.length > 0){
 87        $.ajax({
 88          type: "get",
 89          url: "php/tagurpidi_otsing.php",
 90          dataType: "json",
 91          data: {lat: lat, lng: lng},
 92          success: function(data){
 93            popup = L.popup().setLatLng([lat, lng])
 94                             .setContent(data[0].nimi)
 95                             .openOn(kaart);
 96            kaart.flyTo([lat, lng], 15);
 97            }
 98        });
 99      }
100    });
```

Tagurpidi_otsing.php on jällegi sarnane kahele eelmisele PHP failile. Erinevus on ainult SQL päringus. „ORDER BY“ järjestab leitud punktid, „<->“ tähendab punktide vahemaade võrdlust ja „LIMIT“ ütleb, mitu punkti valitakse. Tulemus on näha joonisel 15.

```
<?php
  $host        = "host = localhost";
  $port        = "port = 5432";
  $dbname      = "dbname = postgres";
  $credentials = "user = postgres password=postgres";

  $yhendus = pg_connect( "$host $port $dbname $credentials"  );
  if(!$yhendus) {
    echo "Error : Ei saa andmebaasiga ühendust";
  }
  $lat = $_GET['lat'];
  $lng = $_GET['lng'];
  $paring = "SELECT full_addre FROM aadressid
             ORDER BY geom <-> 'SRID=3301;POINT(".$lng."
".$lat.")'::geometry
             LIMIT 1;";

  $geojson = array();
  $valjund = pg_query($yhendus, $paring);
  $rida = pg_fetch_assoc($valjund);

  $feature = array(
    'nimi' => $rida['full_addre']);

  array_push($geojson, $feature);
  echo json_encode($geojson);

  pg_free_result($valjund);
  pg_close($yhendus);
?>
```



Joonis 15. Näide aadressotsingust.

5.6 Teekonnaarvutus

Lühima/kiireima tee leidmine kahe või enama punkti vahel on aadressotsingu kõrval väga levinud veebikaardi osa. Leafletile on selleks tehtud mitu moodulit, millest töös kasutatakse Leaflet Routing Machine moodulit, mis kasutab teekonna arvutusteks Open Street Map'i teedevõrgustikku. Leaflet Routing Machine toetab nii geokodeerimist kui ka tagurpidi geokodeerimist. Aadressotsinguna on väga mugav kasutada Leaflet Control Geocoder'it, kuna see on lihtasti Leaflet Routing Machine'i integreeritav ning võimaldab kasutada mitut erinevat teenusepakkujat.

Kõigepealt lisatakse Leaflet Routing Machine ja Leaflet Control Geocoder teegid (vt. Terve kood lisa 7).

```
8  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
leaflet-routing-machine/3.2.8/leaflet-routing-machine.min.css"/>
```

```
10  <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-routing-
machine/3.2.8/leaflet-routing-machine.min.js"></script>
11  <script src="https://unpkg.com/leaflet-control-geocoder/dist/Control.
Geocoder.js"></script>
```

Tagurpidi geokodeerimise jaoks luuakse funktsioon, mis kaardile klikides tekitab lähtepunkti ja sihtpunkti valimise nupud.

```
30    function createButton(silt, konteiner) {
31      var nupp = L.DomUtil.create('button', '', konteiner);
32      nupp.setAttribute('type', 'button');
```

```
33      nupp.innerHTML = silt;
34      return nupp;
35    }
```

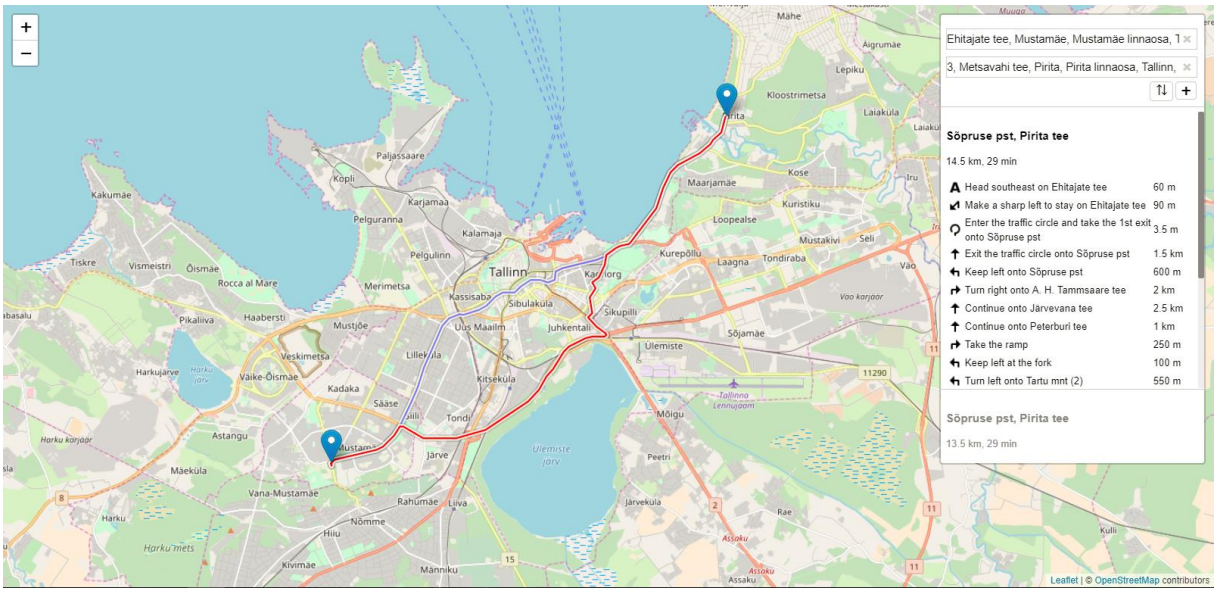
Kaardile klikkides avaneb hüpikaken, kus sees on eelpool defineeritud funktsiooniga tehtud nupud. Nuppudele vajutades (read 47-53) asendatakse olemasoleva teekonna punkt uue punktiga (read 48 ja 53).

```
37    kaart.on('click', function(e) {
38      var konteiner = L.DomUtil.create('div'),
39          lahteNupp = createButton('Lähtepunkt', konteiner),
40          sihtNupp = createButton('Sihtpunkt', konteiner);
41
42      L.popup()
43        .setContent(konteiner)
44        .setLatLng(e.latlng)
45        .openOn(kaart);
46
47      L.DomEvent.on(lahteNupp, 'click', function() {
48        routingMachine.spliceWaypoints(0, 1, e.latlng);
49        kaart.closePopup();
50      });
51
52      L.DomEvent.on(sihtNupp, 'click', function() {
53        routingMachine.spliceWaypoints(routingMachine.getWaypoints()
.length - 1, 1, e.latlng);
54        kaart.closePopup();
55      });
56    });
```

Leaflet Routing Machine'i kasutamiseks luuakse uus objekt *routingMachine*, mis saab esialgseks sisendiks listi tühjade koordinaatidega, aadressotsinguks kasutatakse Nominatimit, parameetriga *routeWhileDragging* saab öelda, kas juba välja arvutatud teekonda saab uusi punkte lisada ja parameeteri *reverseWaypoints* abil saab määrata, kas teekonna algust ja lõppu saab ümber keerata, parameeteri *showAlternatives* abil saab määrata, kas näidatakse ka alternatiivseid teekondi ja parameetriga *altLineOptions* saab määrata alternatiivsete teekondade värvi ja seda kas nendele saab lisada uusi punkte. Pilt teekonnaarvutusest on toodud joonisel 16.

```
58    const routingMachine = L.Routing.control({
59      waypoints: [L.latLng(), L.latLng()],
60      geocoder: L.Control.Geocoder.nominatim(),
61      routeWhileDragging: true,
62      reverseWaypoints: true,
63      showAlternatives: true,
64      altLineOptions:{
```

```
65          addWaypoints: true,
66          styles: [
67            {color: 'black', opacity: 0.15, weight: 9},
68            {color: 'white', opacity: 0.8, weight: 6},
69            {color: 'blue', opacity: 0.5, weight: 2}
70            ]
71          },
72      }).addTo(kaart);
```



Joonis 16. Teekonnaarvutus kasutades Leaflet Routing Machine'i.

5.7 Kaardile joonistamine

Tihti on vaja kaardile midagi joonistada, mõõta pikkuseid ja pindalasid või siis märkida mingeid punkte. Levinud on rakendused, kus elanikke kutsutakse üles panustama linna planeerimisse just läbi sellise veebilehe, kus nad saavad kaardile märkida kohti, kus võiks midagi paremini teha. Või tahab keegi märkida kaardile oma teekonna, selle siis oma arvutisse laadida ja hiljem GPS seadmesse lisada ning selle järgi looduses liikuda. Kogu seda funktsionaalsust aitab saavuta Leaflet'ile tehtud teek Leaflet Draw, mis lisab kaardile joonistamiseks vajaliku menüü. Järgnevalt tehaksegi rakendus, kus kasutaja saab lisada kaardile punkte, pindu ja jooni ning kirjutada nendele lisaja nime ja kommentaari. Lisatud nähtused salvestatakse PostgreSQL/PostGIS andmebaasi.

Kõigepealt lisatakse Lealfet Draw teek (terve kood on toodud lisas 6).

```
7     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
leaflet.draw/1.0.2/leaflet.draw.css"/>
```

```
9     <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/
1.0.2/leaflet.draw.js"></script>
```

Seejärel luuakse nähtuste grupp, kuhu hakatakse lisama joonistatud nähtuseid.

```
32    var joonistatudNahtused = new L.FeatureGroup();
33    kaart.addLayer(joonistatudNahtused);
```

Siis lisatakse menüü joonistamise nuppudega, mida on võimalik päris palju muuta. Näiteks võtta mõni nupp menüüst ära, praegusel juhul editeerimise, ringi ikooni lisamise ja ringi joonistamise nupud. Polügooni ja joone puhul saab määrata kas lubada lõikumist või mitte. Kui tahta mõõta midagi, siis saab öelda, et näidatakse pikkuseid ja pindalasid. Lisaks saab määrata igale objektile värvi, mis vaikimisi on sinine, ja punktile leppemärgi.

```
35    var joonistamiseMenuu = new L.Control.Draw({
36      edit: false,
37      draw:{
38        polygon: {
39          metric: true,
40          allowIntersection: false,
41          showArea: true
42        },
43        rectangle:{
44          metric: true,
45          showArea: true
46        },
47        circle:false,
48        circlemarker:false,
49        polyline:{
50          repeatMode: true,
51          feet: false,
52          nautic: false,
53          allowIntersection: false,
54          showLenght: true
55        },
56      }
57    });
```

Nuppudele antakse eestikeelsed nimetused ja menüü lisatakse kaardile (rida 85).

```
59    L.drawLocal.draw.toolbar.buttons.polygon = 'Lisa pind';
60    L.drawLocal.draw.toolbar.buttons.polyline = 'Lisa joon';
61    L.drawLocal.draw.toolbar.buttons.rectangle = 'Lisa nelinurk';
62    L.drawLocal.draw.toolbar.buttons.marker = 'Lisa punkt';
63
64    L.drawLocal.draw.handlers.polygon.tooltip.start = 'Joonistamiseks
kliki kaardile';
65    L.drawLocal.draw.handlers.polyline.tooltip.start = 'Joonistamiseks
kliki kaardile';
67    L.drawLocal.draw.handlers.rectangle.tooltip.start = 'Joonistamiseks
kliki kaardile ja hoia hiirt all';
```

```
68    L.drawLocal.draw.handlers.marker.tooltip.start = 'Lisamiseks kliki
kaardile';
69
70    L.drawLocal.draw.handlers.polygon.tooltip.cont = '';
71    L.drawLocal.draw.handlers.polyline.tooltip.cont = '';
72
73    L.drawLocal.draw.handlers.polygon.tooltip.end = 'Lõpetamiseks kliki
esimesel punktil';
74    L.drawLocal.draw.handlers.polyline.tooltip.end = 'Lõpetamiseks tee
topeltklikk';
75    L.drawLocal.draw.handlers.simpleshape.tooltip.end = 'Lõpetamiseks
lase hiir lahti';
76
77    L.drawLocal.draw.toolbar.actions.title = '';
78    L.drawLocal.draw.toolbar.finish.title = '';
79    L.drawLocal.draw.toolbar.undo.title = '';
80
81    L.drawLocal.draw.toolbar.actions.text = 'Katkesta';
82    L.drawLocal.draw.toolbar.finish.text = 'Lõpeta';
83    L.drawLocal.draw.toolbar.undo.text = 'Võta tagasi';
84
85    kaart.addControl(joonistamiseMenuu);
```

Real 94 kirjeldatakse, mis juhtub, kui Leaflet Draw'd kasutades kaardile midagi joonistatakse. Kõigepealt tehakse uus kiht kuhu lisatakse joonistatud nähtus, uus kiht lisatakse joonistatud nähtused gruppi.Seejärel tehakse nähtusele hüpikaken, mis on kirjeldatud real 87 ja hüpikaken avaneb kohe kui nähtus on joonistatud. Hüpikaknasse saab kirjutada nime ja kirjelduse.

```
87    var popup = '<form id="ankeet">'+
88                  '<b>Nimi</b></br><input id="nimi" type="text"
style="width:300px"></br></br>'+
89                  '<b>Kirjeldus</b></br>'+
90                  '<textarea id="kirjeldus" rows="6" cols="40"
maxlength="254"></textarea>'+
91                  '<input id="submit" type="submit" value="Saada"> '+
92                 </form>';
93
94    kaart.on('draw:created', function (f) {
95      var kiht = f.layer;
96      joonistatudNahtused.addLayer(kiht);
97      kiht.bindPopup(popup, {closeButton: false}).openPopup();
98
```

Selleks, et andmed baasi saada kasutatakse AJAX't ja PHP-d. Kõigepealt salvestatakse hüpikakna sisu muutujatesse ja loodud kiht salvestatakse GeoJSON formaati, kust omakorda salvestatakse geomeetria eraldi muutujasse. Ridadel 107-112 saadetakse nimi, kirjeldus ja geomeetria saada_andmed.php failile. Seejärel eemaldatakse loodud kiht kaardilt ja

salvestatakse lisatud nähtus kihti andmed_baasist (mis tehakse järgmises lõigus) koos kirjeldusega.

```
 99      $("#ankeet").submit(function(e){
100        e.preventDefault();
101        var nimi =$("#nimi").val();
102        var kirjeldus = $('#kirjeldus').val();
103        var json = kiht.toGeoJSON();
104        var json_loetav = JSON.stringify(json);
105        var geomeetria = json_loetav.slice(45).slice(0,-1);
106
107        $.ajax({
108          type:"get",
109          url: "php/saada_andmed.php",
110          dataType: "json",
111          data:{nimi: nimi, kirjeldus: kirjeldus, json:geomeetria},
112          success:function(data){
113            console.log(data)
114          }
115        });
116        kaart.removeLayer(kiht);
117        json.properties.kirjeldus = kirjeldus;
118        andmed_baasist.addData(json);
219      });
220    });
```

Teiste kasutajate lisatud andmed peaksid kaardirakenduse avamisel kohe nähtavale ilmuma. Selleks luuakse uus GeoJSON kiht kuhu lisatakse veebilehe avamisel baasist andmed. Igale nähtusele lisatakse ka hüpikaken koos kirjeldusega.

```
87    var andmed_baasist = new L.geoJson(null, {
88      onEachFeature: function(feature,layer){
89        layer.bindPopup(feature.properties.kirjeldus);
90      }
91    }).addTo(kaart);
```

Lehe avamisel andmete baasist kätte saamiseks kasutatakse jällegi AJAX’t ja PHP’d. Saadud andmed salvestatakse GeoJSON formaati ja lisatakse eelmises lõigus tehtud andmed_baasist kihile.

```
 93    $.ajax({
 94      type:"get",
 95      url: "php/andmed_baasist.php",
 96      dataType: "json",
 97      success:function(data){
 98        var andmed = data.jsonb_build_object;
 99        andmed = JSON.parse(andmed);
100        andmed_baasist.addData(andmed);
```

```
101       }
102     });
```

Saada_andmed.php failis võrreldakse, kas saadetud nähtus on punkt, pind või joon ja vastavalt sellele lisatakse nähtus vastavasse andmebaasi tabelisse (tabelid peavad enne olemas olema).

```
 1 <?php
 2   $host        = "host = localhost";
 3   $port        = "port = 5432";
 4   $dbname      = "dbname = postgres";
 5   $credentials = "user = postgres password=postgres";
 6
 7   $yhendus = pg_connect( "$host $port $dbname $credentials"  );
 8   if(!$yhendus) {
 9     echo "Error : Unable to open database\n";
10   }
11
12   if($_GET['nimi'] && $_GET['kirjeldus'] &&$_GET['json']){
13     $nimi = $_GET['nimi'];
14     $kirjeldus = $_GET['kirjeldus'];
15     $json = $_GET['json'];
16
17     if(strpos($json, 'Point')!== false){
18       $paring = "INSERT INTO sisestatud_punktid (nimi, kirjeldus, geom)
VALUES
19               ('".$nimi."','".$kirjeldus."',ST_SetSRID(ST_GeomFromGeoJSON
('".$json."'),3301))";
20       $valjund = pg_query($yhendus, $paring);
21       $tulemus = pg_affected_rows($valjund);
22       if ($tulemus == 1) {
23         echo ("Kirje edukalt lisatud");
24       } else {
25         echo ("Kirjet ei lisatud");
26       };
27       pg_free_result($valjund);
28     };
29
30     if(strpos($json, 'LineString')!== false){
31       $paring = "INSERT INTO sisestatud_jooned (nimi, kirjeldus, geom)
VALUES
32               ('".$nimi."','".$kirjeldus."',ST_SetSRID(ST_GeomFromGeoJSON
('".$json."'),3301))";
33       $valjund = pg_query($yhendus, $paring);
34       $tulemus = pg_affected_rows($valjund);
35       if ($tulemus == 1) {
36         echo ("Kirje edukalt lisatud");
37       } else {
38         echo ("Kirjet ei lisatud");
```

```
39        };
40        pg_free_result($valjund);
41      };
42
43      if(strpos($json, 'Polygon')!== false){
44        $paring = "INSERT INTO sisestatud_pinnad (nimi, kirjeldus, geom)
VALUES
45              ('".$nimi."','".$kirjeldus."',ST_SetSRID(ST_GeomFromGeoJSON
('".$json."'),3301))";
46        $valjund = pg_query($yhendus, $paring);
47        $tulemus = pg_affected_rows($valjund);
48        if ($tulemus == 1) {
49          echo("Kirje edukalt lisatud");
50        } else {
51          echo ("Kirjet ei lisatud");
52        };
53        pg_free_result($valjund);
54      };
55    };
56    pg_close($yhendus);
57  ?>
```

Andmed_baasist.php algus ja lõpp on samad, mis eelmistel PHP failidel. Erinevus on real 11 kus tehakse SQL päring andmebaasi ning päritakse andmeid kolmelt kihil (sisestatud_punktid, sisestatud_jooned ja sisestatud_pinnad). Seejärel salvestatakse saadud tulemused nähtuste gruppi (*FeatureCollection*) ja saadetakse brauserile.

```
 1 <?php
 2   $host        = "host = localhost";
 3   $port        = "port = 5432";
 4   $dbname      = "dbname = postgres";
 5   $credentials = "user = postgres password=postgres";
 6
 7   $yhendus = pg_connect( "$host $port $dbname $credentials"  );
 8   if(!$yhendus) {
 9     echo "Error : Unable to open database\n";
10   }
11   $paring = "SELECT jsonb_build_object(
12             'type',     'FeatureCollection',
13             'features', jsonb_agg(feature)
14             )
15             FROM (
16               SELECT jsonb_build_object(
17                 'type',       'Feature',
18                 'geometry',   ST_AsGeoJSON(geom)::jsonb,
19                 'properties', to_jsonb(row) - 'geom'
20               ) AS feature
21               FROM (
```

```
22                 SELECT kirjeldus, geom FROM sisestatud_punktid UNION
23                 SELECT kirjeldus, geom FROM sisestatud_jooned UNION
24                 SELECT kirjeldus, geom FROM sisestatud_pinnad) row)
features";
25
26   $valjund = pg_query($yhendus, $paring);
27   $rida = pg_fetch_assoc($valjund);
28
29   echo json_encode($rida);
30   pg_free_result($valjund);
31   pg_close($yhendus);
32 ?>
```

Tihti saab kasutaja enda joonistatud nähtuseid veebikaardilt ka alla laadida. Selleks tehakse kõigepealt nupp, kust seda teha.

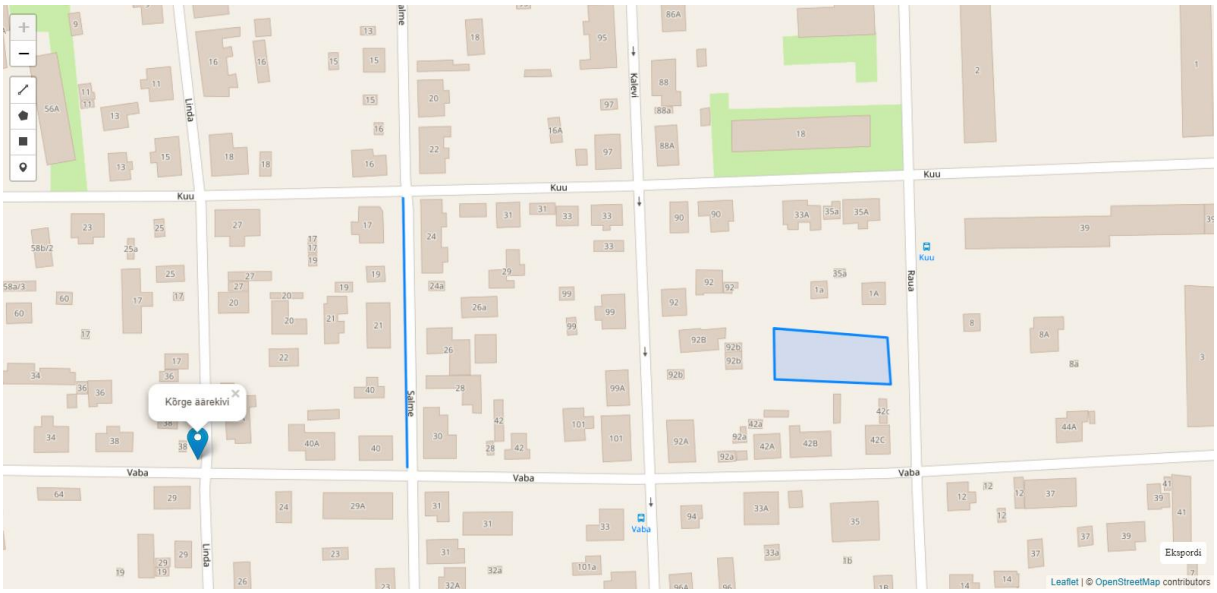
```
23  <a href='#' id='ekspordi'>Ekspordi</a>
```

Nupp stiliseeritakse, et seda oleks paremini näha ja paigutatakse ekraani alla paremasse nurka.

```
19    #ekspordi {
20      position: absolute;
21      bottom:70px;
22      right:10px;
23      z-index:100;
24      background:white;
25      color:black;
26      padding:6px;
27      border-radius:4px;
28      font-family: 'Helvetica Neue';
29      cursor: pointer;
30      font-size:12px;
31      text-decoration:none;
32      bottom:30px;
33    }
```

Eksport nupule klikkides salvestatakse kasutaja lisatud andmed GeoJSON formaati ja laetakse alla. Lõplik tulemus rakendusest on näha joonisel 17.

```
155    document.getElementById('ekspordi').onclick = function(e) {
156      var data = andmed_baasist.toGeoJSON();
157      var convertedData = 'text/json;charset=utf-8,'+encodeURIComponent
(JSON.stringify(data));
158      document.getElementById('ekspordi').setAttribute('href', 'data:' +
convertedData);
159      document.getElementById('ekspordi').setAttribute('download',
'data.geojson');
160    };
```



Joonis 17. Kaardirakendus, kus kasutaja saab kaardile joonistada.

Kokkuvõte

Veebikaardid on arenenud tavalistest piltidest HTML lehel interaktiivseteks rakendusteks, mille abil saab kasutajatele näidata maailmas toimuvaid protsesse uutel ja innovatiivsetel viisidel. Läbi kaardirakenduste saab inimesi panna rohkem osalema ühiskonnas ja viia kurssi maailmas toimuvaga, kuna läbi interneti jõuavad kaardid suurema hulga inimesteni kui paberkaartide puhul. Seetõttu oligi antud töö eesmärgiks uurida, kuidas käib kaardirakenduste tegemine ja milliseid oskuseid ja vahendeid selleks vaja on. Töö annab panuse eesti keelsesse geoinformaatika haridusse ja aregnule, kuna sisaldab ka praktilisi näiteid kaardirakenduste tegemise ja soovitusi arhitektuuri valiku ja komponentide kohta.

Praktiliste näidete tegemiseks valiti kaardi API-ks Leaflet, kuna seda on algajal veebikaardi tegijal kõige lihtsam omandada. Andmebaasiks valiti PostgreSQL/PostGIS, kuna see pakub kõige rohkemaid võimalusi ruumiandmetega töötamiseks ning see on kasutusel ka Tartu Ülikoolis geograafia osakonnas, mistõttu juba tuttav geoinformaatikutele. Serverina kasutati WAMP serverit, mis sisaldab endas praegusel hetkel kõige populaarsemat ja vabavaralist veebiserverit Apache, PHP-d ning laiendeid PostgreSQL baasiga ühendumiseks. Sellise arhitektuuri kasutamine teeb algajale kaardirakenduste programmeerimisega alustamise lihtsaks, kuna üks programm sisaldab enamuse vajaminevast tarkvarast. Kaardiserverina kasutati ka Tartu Ülikooli geograafia osakonnas juba tuttavat Geoserverit.

Töö käigus tehti neli erinevat kaardirakendust, uurimaks kuidas programmeerida enamlevinud veebikaardi komponente (aluskaartide lisamine, teemakaardi tegemine, WMS kihi tegemine, aadressotsing, teekonnaarvutus, kaardile joonistamine). Kõikide rakenduste kood on koos seletustega töös välja toodud.

Web mapping with Leaflet and PostGIS

Risto Ülem

Summary

Web maps have developed from pictures in web pages to interactive applications, through which it is possible, by new ways, to show the processes that happen in our world. It is possible to reach more people through web maps then through paper maps and make them be part of what is happening in society and in the world. Despite that web mapping is not taught thoroughly for geoinformatics in the University of Tartu geography department.

The aim of this master's thesis was to find out how web mapping works, what kind of software is needed for it and what kind of software configuration are used for web mapping. Several different web maps were developed, to learn how to make most common web map components (routing, address search, client side drawing on the map, and making a simple thematic map) and to provide examples for future generations of web mapping enthusiasts for them to quick start with web mapping.

A software package was selected for this purpose for students from University of Tartu Geography department. The package includes WAMP server which comes with Apache web server (most populaar and open source web server), PHP (to communicate with databse) and PotgreSQL/PostGIS database extensions. PostgreSQL with PostGIS extension was selected for the database due to its big scope of spatial data functions and it is already in use in University of Tartu Geography department, so that students are familiar with it. Lealfet was choosen for mapping API because of its small learning curve. For some applications GeoServer was in use. GeoServer was choosen also because it is being taught in the Geography department of The University of Tartu and because of the simplicity of its graphical user interface.

Tänuavaldused

Tänan oma juhendajat, Tõnu Oja, kannatlikkuse ja õpetussõnade eest. Samuti Brita Vibot keelekorrektuuri ja abi eest.

Kasutatud kirjandus

Agrawal, S., Gupta, R. D. 2017. Web GIS and its architecture - a review. Arabian Journal of Geosciences, 10:518.

Agrawal, S., Gupta, R. D. 2017. Development and Comparision of Open Source Based Web Gis Frameworks on Wamp and Apache Tomcat Web Servers. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, volume XL-4.

ArcGIS Pro. 2017. ArcGIS Pro Online Help. http://pro.ArcGIS.com/en/pro-app/tool-reference/data-management/create-vector-tile-package.htm (14.04.2017)

Blewe, B., 1997. GIS Online: Information retrieval, Mapping, and the Internet. On Word Press, Santa Fe.

Brovelli M. A., Minghini M., Zamboni G. 2016. Public participation in GIS via mobile applications. ISPRS Journal of Photogrammetry and Remote Sensing. 114, 306-315.

Brovelli M. A., Minghini M., Zamboni G. 2015. Public Participation GIS: a FOSS architecture enabling field-data collection. Jnternational Journal of Digital Earth, 8:5, 345-363.

CartouCHe. 2012. What is WMS and how does it work? http://www.e-cartouche.ch/content_reg/cartouche/webservice/en/html/wms_learningObject1.html (15.04.2017).

CSS Introduction. 2018. W3schools.com. https://www.w3schools.com/css/css_intro.asp (10.03.2018).

ESRI. 2015. ArcGIS Blog. https://blogs.esri.com/esri/ArcGIS/2015/07/20/vector-tiles-preview/ (09.04.2015)

Farkas, G. 2017. Applicability of open-source web mapping libraries for building massive WebGIS clients. Journal of Geographical Systems, 19, 273-295.

GeoServer. 2017. GeoServer User Manual. http://docs.geoserver.org/stable/en/user/index.html (01.04.2017).

GeoWebCache. 2017. GeoWebCache 1.11.0. http://geowebcache.org/docs/current/index.html (01.04.201

JavaScript Introduction. 2018. W3schools.com. https://www.w3schools.com/js/js_intro.asp (10.03.2018).

Hess, S. 2002. GRASS on the Web. In Proceedings of the Open Source GIS-GRASS Users Conference 2002, Trento, Italy, 11–13 September 2002; pp. 1–14.

JavaScript Introduction. 2018. W3schools.com. https://www.w3schools.com/css/css_intro.asp (10.03.2018).

Karnatak, H. C. 2017. Concept and Applications of Web GIS and Geo-Web Services - Technology and Applications. Indian Institute of Remote Sensing, Indian Space Research Organization. Dehradun, India.

Mapbox. 2016. Vector Tile Specification. https://github.com/mapbox/vector-tile-spec/tree/master/2.1 (04.09.2016).

MBTiles Specification. 2017. https://github.com/mapbox/mbtiles-spec (01.04.2017).

Nair, R., Chauhan, R., Vats, M. 2015. Comparative Analasys of Open Soure Spatial database Systems. International Journal of Innovative Computer Science & Engineering. Volume 2, Issue 6.

Netcraft. 2017. December 2017 Web Server Surveys. https://news.netcraft.com/archives/2017/12/26/december-2017-web-server-survey.html (03.02.2018).

Neumann, A. 2008. Web Mapping and Web Cartography. Teoses Encyclopedia of GIS, toim: Shekhar, S., Xiong, H., Springer: Berliin, Saksamaa, lk 1261–1270.

Open Street Map. 2017. Slippy map tilenames. http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames (22.03.2017).

PHP 5 Introduction. 2018. W3schools.com. https://www.w3schools.com/php/php_intro.asp (10.03.2018).

QGIS. 2017. Documentation QGIS 2.6 https://docs.qgis.org/2.6/en/docs/user_manual/working_with_ogc/ogc_client_support.html (22.03.2016)

Quinn, S., 2017. GEOG 585: Open Web Mapping. John A. Dutton e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University. https://www.e-education.psu.edu/geog585/ (28.01.2018).

Rao, S., Vinay, S. 2009. Choosing the right GIS framework for an informed Enterprise Web GIS Solution. CIESIN Columbian University & NASA. New York, USA. 17.12.2009.

Roth, R. E., Donohue, R. G., Sack, C. M. 2014. A process for Keeping Pace with Evolving Web Mapping Technologies. Cartographic Perspectives, 78, 25-52

Techopedia. 2018. Web Mapping. https://www.techopedia.com/definition/15584/webmapping (05.03.2018).

Troškina, O., 2014. Veebipõhiste kaardirakenduste loomise rakendusliideste võrdlus. Magistritöö geoinformaatikas ja kartograafias. Tartu Ülikool, Tartu.

Untera, M., 2012. Veebipõhise kaardikomponendi loomine puuteekraaniga mobiiltelefonidele. Bakalaureusetöö geoinformaatikas ja kartograafias. Tartu Ülikool, Tartu.

Veenendaal B., Brovelli M. A., Li S., 2017. Review of Web Mapping: Eras, Trends and Directions. International Journal of Geo-information, 6, 317.

Yang C. P., Wong W. D., Yang R., Kafatos M., Li Q. 2005 Performance-improving techniques in web-based GIS, International Journal of Geographical Information Science, 19:3, 319-342.

Lisad

Lisa 1. Leafleti kaardirakenduse kood aluskaartide ja punkt objektide näitamiseks

```
<!DOCTYPE html>
<html>
    <head>
      <title>Minu kaart</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
    <link rel="stylesheet" href="src/marker_cluster/MarkerCluster.Default.css">
    <script src="src/leaflet/leaflet.js"></script>
    <script src="https://unpkg.com/leaflet.vectorgrid@latest/dist/Leaflet.VectorGrid.bundled.js"></script>
    <script src="src/marker_cluster/leaflet.markercluster.js"></script>
    <script src="vektor_tailid_stiil.js"></script>
    <script src="andmed/objektid.js"></script>
    <style>
      #kaart, html, body {
        width:100%;
        height:100%;
        margin: 0;
      }
    </style>
  </head>

  <body>
    <div id="kaart"></div>

    <script type="text/javascript">
            const kaart = L.map('kaart');
            const OSM = L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
        attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
      }).addTo(kaart);
      kaart.setView({ lat: 58.7, lng: 25.0 }, 8);

      const esri ='<a href="http://www.esri.com/">Esri</a>';
      const pildid_attr = 'i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP,\and the GIS User Community';

      const sat = L.tileLayer ('http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}', {
        attribution: '&copy; '+esri+', '+pildid_attr
      });
```

```
 39
 40       const url = "andmed/vektor_aluskaart/{z}/{x}/{y}.pbf";
 41       const vektorTailid = L.vectorGrid.protobuf(url, {
 42         attribution: '<a href="https://openmaptiles.org/">&copy;
OpenMapTiles</a>, <a href="http://www.openstreetmap.org/copyright">&copy;
OpenStreetMap</> contributors',
 43         vectorTileLayerStyles: vektorTailidStiil,
 44         interactive: true
 45       }).on('click', function(e) {
 46         L.popup()
 47         .setContent(e.layer.properties.name)
 48         .setLatLng(e.latlng)
 49         .openOn(kaart);
 50       });
 51
 52       var ikoonid = {
 53         rändrahn: L.icon({
 54           iconSize: [32, 21],
 55           iconAnchor: [16, 21],
 56           popupAnchor:  [0, -21],
 57           iconUrl: 'pildid/kivi.png'
 58         }),
 59         allikas: L.icon({
 60           iconSize: [32, 32],
 61           iconAnchor: [16, 32],
 62           popupAnchor:  [0, -32],
 63           iconUrl: 'pildid/allikas.png'
 64         }),
 65         puu: L.icon({
 66           iconSize: [30, 30],
 67           iconAnchor: [15, 30],
 68           popupAnchor: [0, -30],
 69           iconUrl: 'pildid/puu.png'
 70         }),
 71         pinnavorm: L.icon({
 72           iconSize: [38, 21],
 73           iconAnchor: [19, 21],
 74           popupAnchor: [19, -21],
 75           iconUrl: 'pildid/pinnavorm.png'
 76         })
 77       };
 78
 79       function onEachFeaturePopup(feature, layer) {
 80         layer.bindPopup(feature.properties.nimi);
 81       }
 82
 83       const loodusObjektid = new L.GeoJSON(objektid, {
 84         pointToLayer: function (feature, latlng) {
 85           marker = new L.marker(latlng, {
```

```
 86              icon: ikoonid[feature.properties.tyyp]
 87            });
 88            return marker;
 89          },
 90          onEachFeature: onEachFeaturePopup
 91        });
 92
 93        const klastrid = L.markerClusterGroup({
 94          showCoverageOnHover: false,
 95          maxClusterRadius: 50
 96        });
 97
 98        klastrid.addLayer(loodusObjektid);
 99        kaart.addLayer(klastrid);
100
101        const teemainfo = {
102          Objektid: klastrid,
103        };
104
105        const aluskaardid = {
106          Vektor: vektorTailid,
107          Raster: OSM,
108          Satelliit: sat
109        };
110
111        const kihiloend = L.control.layers(aluskaardid,
teemainfo).addTo(kaart);
112
113        L.control.scale({imperial: false}).addTo(kaart);
114
115      </script>
116
117    </body>
118 </html>
```

Lisa 2. Stiilifail stiil.js

```
const vektorTailidStiil = {
  water: {
    fill: true,
    weight: 1,
    fillColor: '#06cccc',
    color: '#06cccc',
    fillOpacity: 0.2,
    opacity: 0.4,
  },
  admin: {
    weight: 1,
    fillColor: 'pink',
    color: 'pink',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  waterway: {
    weight: 1,
    fillColor: '#2375e0',
    color: '#2375e0',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  landcover: {
    fill: true,
    weight: 1,
    fillColor: '#53e033',
    color: '#53e033',
    fillOpacity: 0.2,
    opacity: 0.4,
  },
  landuse: {
    fill: true,
    weight: 1,
    fillColor: '#e5b404',
    color: '#e5b404',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  park: {
    fill: true,
    weight: 1,
    fillColor: '#84ea5b',
    color: '#84ea5b',
    fillOpacity: 0.2,
    opacity: 0.4
  },
```

```
  boundary: {
    weight: 1,
    fillColor: '#c545d3',
    color: '#c545d3',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  aeroway: {
    weight: 1,
    fillColor: '#51aeb5',
    color: '#51aeb5',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  transportation: {
    weight: 0.5,
    fillColor: '#f2b648',
    color: '#f2b648',
    fillOpacity: 0.2,
    opacity: 0.4,
      dashArray: [4, 4]
  },
  building: {
    fill: true,
    weight: 1,
    fillColor: '#2b2b2b',
    color: '#2b2b2b',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  water_name: {
    weight: 1,
    fillColor: '#022c5b',
    color: '#022c5b',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  transportation_name: {
    weight: 1,
    fillColor: '#bc6b38',
    color: '#bc6b38',
    fillOpacity: 0.2,
    opacity: 0.4
  },
  place: {
    weight: 1,
    fillColor: '#f20e93',
    color: '#f20e93',
    fillOpacity: 0.2,
```

```
 97     opacity: 0.4
 98   },
 99   housenumber: {
100     weight: 1,
101     fillColor: '#ef4c8b',
102     color: '#ef4c8b',
103     fillOpacity: 0.2,
104     opacity: 0.4
105   },
106   poi: {
107     weight: 1,
108     fillColor: '#3bb50a',
109     color: '#3bb50a',
110     fillOpacity: 0.2,
111     opacity: 0.4
112   }
113 };
```

Lisa 3. Rahvastiku kihi stiil SLD-s

```
 1 <?xml version="1.0" encoding="ISO-8859-1"?>
 2 <StyledLayerDescriptor version="1.0.0"
 3   xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd"
 4   xmlns="http://www.opengis.net/sld"
xmlns:ogc="http://www.opengis.net/ogc"
 5   xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 6
 7   <NamedLayer>
 8     <Name>rahvastik</Name>
 9     <UserStyle>
10       <Title>omavalitsused rahvaarvu järgi</Title>
11       <FeatureTypeStyle>
12        <Rule>
13          <PolygonSymbolizer>
14           <Stroke>
15             <CssParameter name="stroke">#000000</CssParameter>
16           </Stroke>
17          </PolygonSymbolizer>
18        </Rule>
19        <Rule>
20          <Name>0-3000</Name>
21          <ogc:Filter>
22            <ogc:PropertyIsLessThan>
23              <ogc:PropertyName>el_arv</ogc:PropertyName>
24              <ogc:Literal>3000</ogc:Literal>
25            </ogc:PropertyIsLessThan>
26          </ogc:Filter>
27          <PolygonSymbolizer>
28            <Fill>
29              <CssParameter name="fill">#FEB24C</CssParameter>
30              <CssParameter name="fill-opacity">0.8</CssParameter>
31            </Fill>
32          </PolygonSymbolizer>
33        </Rule>
34        <Rule>
35          <Name>3000-8000</Name>
36          <ogc:Filter>
37            <ogc:And>
38              <ogc:PropertyIsGreaterThanOrEqualTo>
39                <ogc:PropertyName>el_arv</ogc:PropertyName>
40                <ogc:Literal>3000</ogc:Literal>
41              </ogc:PropertyIsGreaterThanOrEqualTo>
42              <ogc:PropertyIsLessThan>
43                <ogc:PropertyName>el_arv</ogc:PropertyName>
44                <ogc:Literal>8000</ogc:Literal>
```

```
            </ogc:PropertyIsLessThan>
          </ogc:And>
        </ogc:Filter>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#FD8D3C</CssParameter>
            <CssParameter name="fill-opacity">0.8</CssParameter>
          </Fill>
        </PolygonSymbolizer>
      </Rule>
      <Rule>
        <Name>8000-18000</Name>
        <ogc:Filter>
          <ogc:And>
            <ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:PropertyName>el_arv</ogc:PropertyName>
              <ogc:Literal>8000</ogc:Literal>
            </ogc:PropertyIsGreaterThanOrEqualTo>
            <ogc:PropertyIsLessThan>
              <ogc:PropertyName>el_arv</ogc:PropertyName>
              <ogc:Literal>18000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:And>
        </ogc:Filter>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#FC4E2A</CssParameter>
            <CssParameter name="fill-opacity">0.8</CssParameter>
          </Fill>
        </PolygonSymbolizer>
      </Rule>
      <Rule>
        <Name>18000-40000</Name>
        <ogc:Filter>
          <ogc:And>
            <ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:PropertyName>el_arv</ogc:PropertyName>
              <ogc:Literal>18000</ogc:Literal>
            </ogc:PropertyIsGreaterThanOrEqualTo>
            <ogc:PropertyIsLessThan>
              <ogc:PropertyName>el_arv</ogc:PropertyName>
              <ogc:Literal>40000</ogc:Literal>
            </ogc:PropertyIsLessThan>
          </ogc:And>
        </ogc:Filter>
        <PolygonSymbolizer>
          <Fill>
            <CssParameter name="fill">#E31A1C</CssParameter>
            <CssParameter name="fill-opacity">0.8</CssParameter>
```

```
            </Fill>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <Name>40000-95000</Name>
          <ogc:Filter>
            <ogc:And>
              <ogc:PropertyIsGreaterThanOrEqualTo>
                <ogc:PropertyName>el_arv</ogc:PropertyName>
                <ogc:Literal>40000</ogc:Literal>
              </ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:PropertyIsLessThan>
                <ogc:PropertyName>el_arv</ogc:PropertyName>
                <ogc:Literal>95000</ogc:Literal>
              </ogc:PropertyIsLessThan>
            </ogc:And>
          </ogc:Filter>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#BD0026</CssParameter>
              <CssParameter name="fill-opacity">0.8</CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <Name>95000+</Name>
          <ogc:Filter>
            <ogc:PropertyIsGreaterThan>
              <ogc:PropertyName>el_arv</ogc:PropertyName>
              <ogc:Literal>95000</ogc:Literal>
            </ogc:PropertyIsGreaterThan>
          </ogc:Filter>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#BD0026</CssParameter>
              <CssParameter name="fill-opacity">0.8</CssParameter>
            </Fill>
          </PolygonSymbolizer>
        </Rule>
        <Rule>
          <Name>Labels</Name>
          <MaxScaleDenominator>1000000</MaxScaleDenominator>
          <TextSymbolizer>
           <Geometry>
            <ogc:Function name="centroid">
             <ogc:PropertyName>geom</ogc:PropertyName>
            </ogc:Function>
           </Geometry>
          <Label>
```

```
143             <ogc:PropertyName>onimi</ogc:PropertyName>
144           </Label>
145           <Font>
146             <CssParameter name="font-family">Arial</CssParameter>
147             <CssParameter name="font-size">14</CssParameter>
148             <CssParameter name="font-style">normal</CssParameter>
149             <CssParameter name="font-weight">bold</CssParameter>
150           </Font>
151           <LabelPlacement>
152             <PointPlacement>
153               <AnchorPoint>
154                 <AnchorPointX>0.5</AnchorPointX>
155                 <AnchorPointY>0.5</AnchorPointY>
156               </AnchorPoint>
157             </PointPlacement>
158           </LabelPlacement>
159           <Fill>
160             <CssParameter name="fill">#FFFFFF</CssParameter>
161           </Fill>
162           <VendorOption name="autoWrap">60</VendorOption>
163           <VendorOption name="maxDisplacement">150</VendorOption>
164         </TextSymbolizer>
165         </Rule>
166     </FeatureTypeStyle>
167     </UserStyle>
168   </NamedLayer>
169 </StyledLayerDescriptor>
```

Lisa 4. Kaardi „Rahvaarvu tihedus omavalitsuste kaupa“ kood

```
<!DOCTYPE html>
<html>
<head>
  <title>Rahvaarv omavalitsuste kaupa</title>
  <meta charset="utf-8" />
    <meta name="viewport"
          content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
    <script src="src/leaflet/leaflet.js"></script>
    <style>
      #kaart, html, body {
        position: relative;
        z-index: 1;
        width:100%;
        height:100%;
        margin: 0;
      }
      .legend {
        padding: 6px 8px;
        line-height: 18px;
        color: #555;
        background: white;
        border-radius: 5px;
      }
      .legend i {
        width: 18px;
        height: 18px;
        float: left;
        margin-right: 8px;
        opacity: 0.7;
      }
      #pealkiri{
        position: absolute;
        z-index: 10;
        left: 120px;
        top: 20px;
        font-family: "Tahoma";
        font-size: 30px;
        font-weight: bold;
        -webkit-text-stroke: 1px white;
      }
    </style>
</head>
<body>
  <div id="pealkiri">
    Rahvaarv omavalitsustes 2016
  </div>
```

```
 47
 48   <div id="kaart"></div>
 49
 50     <script type="text/javascript">
 51
 52       const kaart = L.map('kaart');
 53
 54       const OSM =
L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
 55         attribution: '&copy; <a href="http://osm.org/copyright">
OpenStreetMap</a> contributors | <a href="http://"maaamet.ee">Maa-amet</a>
| <a href="http://"stat.ee">Statistikaamet</a>'
 56       }).addTo(kaart);
 57       kaart.setView({ lat: 58.7, lng: 25.0 }, 8);
 58
 59       const wmsKiht =
L.tileLayer.wms('http://localhost:8082/geoserver/rahvastik/wms?',{
 60         layers: 'rahvastik:omavalitsused_rv_arv_simpl',
 61         format: 'image/png',
 62         transparent: true
 63       }).addTo(kaart);
 64
 65       function getColor(d) {
 66         return d > 500000 ? '#800026' :
 67                d > 95000  ? '#BD0026' :
 68                d > 40000  ? '#E31A1C' :
 69                d > 18000  ? '#FC4E2A' :
 70                d > 8000   ? '#FD8D3C' :
 71                d > 3000   ? '#FEB24C' :
 72                             '#FED976';
 73       }
 74
 75       const legend = L.control({position: 'bottomright'});
 76       legend.onAdd = function (kaart) {
 77
 78         const div = L.DomUtil.create('div', 'legend');
 79         const vaartused = [0, 3000, 8000, 18000, 40000, 95000];
 80
 81         for (var i = 0; i < vaartused.length; i++) {
 82           div.innerHTML +='<i style="background:'+getColor(vaartused[i]
+1)+'"></i>'+vaartused[i]+(vaartused[i+1]?'&ndash;'+vaartused[i+1]+' in
<br><br>':'+ in');
 83         }
 84           return div;
 85       };
 86       legend.addTo(kaart);
 87
 88       kaart.on('overlayadd', function(eventLayer){
 89         if (eventLayer.name === 'Rahvastik'){
```

```
 90            kaart.addControl(legend);
 91          }
 92        });
 93
 94        kaart.on('overlayremove', function(eventLayer){
 95          if (eventLayer.name === 'Rahvastik'){
 96            kaart.removeControl(legend);
 97          }
 98        });
 99
100        const teemainfo = {
101          Rahvastik: wmsKiht,
102        };
103
104        const aluskaardid = {
105          Kaart: OSM,
106        };
107
108        const kihiloend = L.control.layers(aluskaardid, teemainfo,
{collapsed: false}).addTo(kaart);
109
110        L.control.scale({imperial: false}).addTo(kaart);
111
112      </script>
113 </body>
114 </html>
```

5. Leafleti kaardirakenduse kood aadressotsingu tegemiseks

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4   <title>Aadressotsing</title>
 5   <meta name="viewport" content="width=device-width, initial-scale=1.0"
charset="UTF-8" >
 6   <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
 7   <script src="src/leaflet/leaflet.js"></script>
 8   <script src="http://code.jquery.com/jquery-1.12.4.js"></script>
 9   <script src="http://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
10   <script src="https://unpkg.com/@mapbox/leaflet-pip@1.1.0/leaflet-
pip.js"></script>
11   <script src="andmed/eesti.js"></script>
12   <style>
13     #kaart, html, body {
14       position: relative;
15       z-index: 1;
16       width:100%;
17       height:100%;
18       margin: 0;
19     }
20     #otsing{
21       position: absolute;
22       z-index: 10;
23       left: 100px;
24       top: 20px;
25     }
26     .ui-autocomplete{
27       max-height: 200px;
28       overflow-y: scroll;
29       overflow-x: hidden;
30       background: white;
31     }
32     .ui-helper-hidden-accessible {display:none}
33   </style>
34 </head>
35 <body>
36   <div id="kaart"></div>
37
38   <form id="otsing">
39     <input type="text" id="otsi" placeholder="otsi...">
40   </form>
41
42   <script type="text/javascript">
43
44     const kaart = L.map('kaart');
45
```

```
 46     const OSM = L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
 47       attribution: '&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors | <a
href="http://"maaamet.ee">Maa-amet</a>'
 48     }).addTo(kaart);
 49     kaart.setView({ lat: 58.7, lng: 25.0 }, 8);
 50
 51     $(function() {
 52         $("#otsi").autocomplete({
 53           source: "php/otsing.php",
 54           minLength: 3,
 55           delay: 500,
 56           appendTo: '#otsing',
 57           select: function(event, ui) {
 58             $.ajax({
 59               url: "php/koordinaat.php",
 60               dataType: "json",
 61               data:{addr: ui.item.label},
 62               success: function(data){
 63                 popup = L.popup().setLatLng(data[0].loc)
 64                                  .setContent(data[0].nimi)
 65                                  .openOn(kaart);
 66                 kaart.flyTo(data[0].loc, 15);
 67               }
 68             });
 69           }
 70         });
 71       });
 72
 73     const eesti_polygon = L.geoJson(eesti);
 74
 75     var popup;
 76
 77     var tagurpidi_otsing = kaart.on('click', function(ev) {
 78       if (popup) {
 79          kaart.removeLayer(popup);
 80       }
 81       var lat = ev.latlng.lat;
 82       var lng = ev.latlng.lng;
 83       var results = leafletPip.pointInLayer([lng, lat], eesti_polygon);
 84       if(results.length > 0){
 85         $.ajax({
 86           type: "get",
 87           url: "php/tagurpidi_otsing.php",
 88           dataType: "json",
 89           data: {lat: lat, lng: lng},
 90           success:function(data){
 91             popup = L.popup().setLatLng([lat, lng])
 92                              .setContent(data[0].nimi)
```

```
                                .openOn(kaart);
            kaart.flyTo([lat, lng], 15);
            }
        });
      }
    });

  </script>
</body>
</html>
```

Lisa 6. Leafleti kaardirakenduse kood kaardile joonistamiseks

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4     <title>kaardile joonistamine</title>
 5     <meta name="viewport" content="width=device-width, initial-
scale=1.0" charset="UTF-8">
 6   <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
 7   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
leaflet. draw/1.0.2/leaflet.draw.css"/>
 8   <script src="src/leaflet/leaflet.js"></script>
 9   <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet.draw/
1.0.2/leaflet.draw.js"></script>
10   <script src="http://code.jquery.com/jquery-1.12.4.js"></script>
11   <style>
12     #kaart, html, body {
13       position: relative;
14       z-index: 1;
15       width:100%;
16       height:100%;
17       margin: 0;
18     }
19   </style>
20 </head>
21 <body>
22     <div id="kaart"></div>
23
24     <script type="text/javascript">
25
26     const kaart = L.map('kaart');
27
28     const OSM = L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
29       attribution: '&copy; <a href="http://osm.org/copyright">
OpenStreetMap</a> contributors'
30     }).addTo(kaart);
31     kaart.setView({ lat: 58.7, lng: 25.0 }, 8);
32
33     var joonistatudNahtused = new L.FeatureGroup();
34     kaart.addLayer(joonistatudNahtused);
35
36     var joonistamiseMenuu = new L.Control.Draw({
37       edit: false,
38       draw:{
39         polygon: {
40           metric: true,
41           allowIntersection: false,
42           showArea: true
43         },
```

```
 44         rectangle:{
 45           metric: true,
 46           showArea: true
 47         },
 48         circle:false,
 49         circlemarker:false,
 50         polyline:{
 51           repeatMode: true,
 52           feet: false,
 53           nautic: false,
 54           allowIntersection: false,
 55           showLenght: true
 56         },
 57       }
 58     });
 59
 60     L.drawLocal.draw.toolbar.buttons.polygon = 'Lisa pind';
 61     L.drawLocal.draw.toolbar.buttons.polyline = 'Lisa joon';
 62     L.drawLocal.draw.toolbar.buttons.rectangle = 'Lisa nelinurk';
 63     L.drawLocal.draw.toolbar.buttons.marker = 'Lisa punkt';
 64
 65     L.drawLocal.draw.handlers.polygon.tooltip.start = 'Joonistamiseks
kliki kaardile';
 66     L.drawLocal.draw.handlers.polyline.tooltip.start = 'Joonistamiseks
kliki kaardile';
 67     L.drawLocal.draw.handlers.rectangle.tooltip.start = 'Joonistamiseks
kliki kaardile ja hoia hiirt all';
 68     L.drawLocal.draw.handlers.marker.tooltip.start = 'Lisamiseks kliki
kaardile';
 69
 70     L.drawLocal.draw.handlers.polygon.tooltip.cont = '';
 71     L.drawLocal.draw.handlers.polyline.tooltip.cont = '';
 72
 73     L.drawLocal.draw.handlers.polygon.tooltip.end = 'Lõpetamiseks kliki
esimesel punktil';
 74     L.drawLocal.draw.handlers.polyline.tooltip.end = 'Lõpetamiseks tee
topeltklikk';
 75     L.drawLocal.draw.handlers.simpleshape.tooltip.end = 'Lõpetamiseks
lase hiir lahti';
 76
 77     L.drawLocal.draw.toolbar.actions.title = '';
 78     L.drawLocal.draw.toolbar.finish.title = '';
 79     L.drawLocal.draw.toolbar.undo.title = '';
 80
 81     L.drawLocal.draw.toolbar.actions.text = 'Katkesta';
 82     L.drawLocal.draw.toolbar.finish.text = 'Lõpeta';
 83     L.drawLocal.draw.toolbar.undo.text = 'Võta tagasi';
 84
 85     kaart.addControl(joonistamiseMenuu);
```

```
 86
 87     var andmed_baasist = new L.geoJson(null, {
 88       onEachFeature: function(feature,layer){
 89            layer.bindPopup(feature.properties.kirjeldus);
 90       }
 91     }).addTo(kaart);
 92
 93     $.ajax({
 94       type:"get",
 95       url: "php/andmed_baasist.php",
 96       dataType: "json",
 97       success:function(data){
 98         var andmed = data.jsonb_build_object;
 99         andmed = JSON.parse(andmed);
100         andmed_baasist.addData(andmed);
101       }
102     });
103
104     var popup = '<form id="ankeet">'+
105                   '<b>Nimi</b></br><input id="nimi" type="text"
style="width:300px"></br></br>'+
106                   '<b>Kirjeldus</b></br>'+
107                   '<textarea id="kirjeldus" rows="6" cols="40"
maxlength="254"></textarea>'+
108                   '<input id="submit" type="submit" value="Saada"> '+
109                 '</form>';
110
111     kaart.on('draw:created', function (f) {
112       var kiht = f.layer;
113       joonistatudNahtused.addLayer(kiht);
114       kiht.bindPopup(popup, {closeButton: false}).openPopup();
115
116       $("#ankeet").submit(function(e){
117         e.preventDefault();
118         var nimi =$("#nimi").val();
119         var kirjeldus = $('#kirjeldus').val();
120         var json = kiht.toGeoJSON();
121         var json_loetav = JSON.stringify(json);
122         var geomeetria = json_loetav.slice(45).slice(0,-1);
123
124         $.ajax({
125           type:"get",
126           url: "php/saada_andmed.php",
127           dataType: "json",
128           data:{nimi: nimi, kirjeldus: kirjeldus, json:geomeetria},
129           success:function(data){
130             console.log(data)
131           }
132         });
```

```
133         kaart.removeLayer(kiht);
134         json.properties.kirjeldus = kirjeldus;
135         andmed_baasist.addData(json);
136       });
137     });
138 
139   </script>
140 
141 </body>
142 </html>
```

Lisa 7. Leafleti kaardirakenduse kood teekonnaarvutuse tegemiseks

```
<!DOCTYPE html>
<html>
<head>
  <title>navigeerimine</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="src/leaflet/leaflet.css"/>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/
leaflet-routing-machine/3.2.8/leaflet-routing-machine.min.css"/>
  <script src="src/leaflet/leaflet.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/leaflet-routing-
machine/3.2.8/leaflet-routing-machine.min.js"></script>
  <script src="https://unpkg.com/leaflet-control-geocoder/dist/Control
.Geocoder.js"></script>
  <style>
    #kaart, html, body {
      width:100%;
      height:100%;
      margin: 0;
    }
  </style>
</head>
<body>
  <div id="kaart"></div>

  <script type="text/javascript">
    const kaart = L.map('kaart');
    const OSM = L.tileLayer("http://{s}.tile.osm.org/{z}/{x}/{y}.png",{
      attribution: '&copy; <a href="http://osm.org/copyright">
OpenStreetMap</a> contributors'
    }).addTo(kaart);
    kaart.setView({ lat: 58.7, lng: 25.0 }, 8);

    function createButton(silt, konteiner) {
      var nupp = L.DomUtil.create('button', '', konteiner);
      nupp.setAttribute('type', 'button');
      nupp.innerHTML = silt;
      return nupp;
    }

    kaart.on('click', function(e) {
      var konteiner = L.DomUtil.create('div'),
          lahteNupp = createButton('Lähtepunkt', konteiner),
          sihtNupp = createButton('Sihtpunkt', konteiner);

      L.popup()
          .setContent(konteiner)
```

```
44            .setLatLng(e.latlng)
45            .openOn(kaart);
46
47        L.DomEvent.on(lahteNupp, 'click', function() {
48            routingMachine.spliceWaypoints(0, 1, e.latlng);
49            kaart.closePopup();
50        });
51
52        L.DomEvent.on(sihtNupp, 'click', function() {
53            routingMachine.spliceWaypoints(routingMachine.getWaypoints()
.length - 1, 1, e.latlng);
54            kaart.closePopup();
55        });
56      });
57
58      const routingMachine = L.Routing.control({
59        waypoints: [L.latLng(), L.latLng()],
60        geocoder: L.Control.Geocoder.nominatim(),
61        routeWhileDragging: true,
62        reverseWaypoints: true,
63        showAlternatives: true,
64        altLineOptions:{
65          addWaypoints: true,
66          styles: [
67            {color: 'black', opacity: 0.15, weight: 9},
68            {color: 'white', opacity: 0.8, weight: 6},
69            {color: 'blue', opacity: 0.5, weight: 2}
70            ]
71          },
72      }).addTo(kaart);
73
74    </script>
75 </body>
76 </html>
```

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Risto Ülem,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kaardirakenduste tegemine Leafleti ja PostGISi näitel“, mille juhendaja on Tõnu Oja,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, 01.06.2018