Tartu Ülikool
Loodus- ja tehnoloogiateaduskond
Ökoloogia ja Maateaduste Instituut
Geograafia osakond

VEEBIPÕHISTE KAARDIRAKENDUSTE LOOMISE RAKENDUSLIIDESTE VÕRDLUS

Magistritöö geoinformaatika ja kartograafia erialal

Olga Troškina

Juhendaja: prof. Tõnu Oja

Kaitsmisele lubatud:

Juhendaja:

Osakonna juhataja:

Tallinn 2015

VEEBIPÕHISTE KAARDIRAKENDUSTE LOOMISE RAKENDUSLIIDESTE VÕRDLUS

Olga Troškina

Sisu lühikokkuvõte

Antud magistritöö käsitleb kartograafilisi JavaScript'i rakendusliideseid, mis on abivahenditeks interaktiivsete veebikaartide loomisel. Tuntumad ja konkureerivad rakendusliidesed veebikartograafia jaoks on OpenLayers, Leaflet, Google Maps API ja ArcGis API.

Alustades esmakordselt tööd kaardistamise teekidega, tuleb teha oluline otsus ja valida oma eesmärkidele, oskustele ja olemasolevatele ruumilistele andmetele paremini sobiv teek. Antud töö on keskendatud väljaselgitamisele, milline eespool mainitud rakendusliides võiks olla mingis osas sobivam või lihtsam erinevate ülesannete jaoks. Töös tutvustatakse ka arvutikartograafia põhilisi aspekte ja kaardirakenduse loomise peamisi etappe.

Võtmesõnad

- *Veebikaardistamine*
- *Lõppkasutaja programmeerimine*
- *Andmete visualiseerimine*
- *Rakendusliidesed*
- *JavaScript*
- *OpenLayers*
- *Leaflet*
- *Google Maps API*
- *ArcGIS API for JavaScript*

Sisukord

Sissejuhatus

Veebikartograafia Interneti piiramatute võimaluste abil on abivahend kartograafiliste produktide jagamiseks tuhandete inimestega. Veebikaartide tunnusjooneks on interaktiivsus, mis lisab kaartidele väärtust suumimise ja filtreerimise võimalustega ning kihtide kasutamisega ja multimeedia lisamisega. Selleks et veebikaardi luua on olemas palju erinevaid kaardistamise programmiliideseid (API) ning sobivaima valik nende vahel võib tekitada raskusi. Magistritöö eesmärgiks on parandada sobiva rakendusliidese valiku protsessi, et vältida teatud takistusi ja saavutada optimaalsust rakendusliidesega töötades.

Töö koosneb kuuest peatükist. Esimene peatükk keskendub arvutikartograafiale, selle arengu ajaloole ja trendidele. Siin tutvustatakse ka praktilises osas kasutatavaid kaardistamise teeke: Google Maps, OpenLayers, Leaflet ja ArcGis API, räägitakse ka selle areneva valdkonna väljakutsetest ja probleemidest, mis ei ole veel ületatud. Lisaks illustreeritakse rakenduse loomise protsessi, antakse lühike ülevaate veebikaardinduse juures kasutatavatest tehnoloogiatest. Teine peatükk tutvustab kasutatud tehnoloogiaid ja andmed. Kolmas peatükk keskendub uurimistöö metoodikale: räägitakse API'de võrdlemise kriteeriumitest ja võrdlemiseks kasutatavatest kahest lähenemisest: kvantitatiivsest ja kvalitatiivsest võrdlemismeetodist. Neljas osa annab ülevaate töö käigus loodud prototüüprakendustest ning kirjeldab kuidas konkreetse API'ga vajalik funktsionaalsus saavutatakse ja seletab JavaScript koodi vastavate osade olulisemad kohad. Seejärel tutvustatakse API'de võrdlemise tulemusi ning viimases osas on toodud saadud tulemuste arutelu ja diskussioon.

Kuna arendajad saavad valida paljude erinevate API'de vahel, on valiku otsuse tegemisel olulisemaks teguriks loodud rakendust silmas pidades otstarbekus ja antud valiku kasutamise tasuvus. Erinevatel API teekidel on hulk omavahel erinevaid ja unikaalseid funktsioone ja võimeid, seega on

väga oluline valida kaardirakenduse vajadustele paremini sobiv teek. Selle magistritöö võtmeküsimuseks on uurida veebikaardinduse tehnilisi võimalusi ja analüüsida erinevate tootjate Internetis töötavate kaardilahenduste loomise võimalusi ja lahendusi. Antud töö jaoks on suurest hulgast JavaScript'il põhinevatest kartograafilistest raamistikest valitud kõige tuntumad ja suurimad.

Antud magistritöö eesmärgiks on :

1. Luua näited, mis aitaks valida sobivaima keskkonna ja omandada selle kasutamise tehnilised võtted ning annaks üldised praktilised oskused interaktiivse veebipõhise kaardi loomiseks JavaScript'i rakendusliidese abil.

   Töö käigus sobilike näidisandmetega, mis on vabalt kättesaadavad ja representatiivsed Eesti kohta kaardi tegemiseks, tehakse iga teegiga valmis 5 veebikaardi prototüüpi ja 35 baas-näidet: WMS teenuse kasutamine ning vektorandmete lisamine kaardile WFS teenuse kaudu ja failist, kusjuures mõlemad on kolme erineva vektorobjektide arvuga.

2. Võrrelda erinevate raamistike keerukust, efektiivsust ning sobilikkust erinevate ülesannete lahendamisel.

   Prototüüprakendusi kasutatakse kvalitatiivseks empiiriliseks testimiseks rakendusliideste kasutatavuse võrdlemiseks ning näidisrakendused kasutatakse kvantitatiivseks testimiseks, kus võrdlemiskriteeriumiteks on API suurus, API kasutatavuse indeks ja kaardi kihtide visualiseerimise kiirus. Kahe meetodi kasutamine võimaldab anda ülevaate teekide võimalustest, plussidest ja miinustest.

1. Teoreetiline taust

Antud peatükkis antakse ülevaade arvutikartograafia ajaloost, tänapäeva arengu trendidest ja väljakutsetest ning tutvustatakse lühidalt töös käsitletud rakendusliideseid.

1.1 Mõistete ja lühendite register

AJAX

interaktiivsete veebirakenduste loomise meetod jooksva andmevahetusega brauseri ja veebiserveri vahel, nii et kasutaja iga liigutuse peale pole vaja kogu veebilehte uuesti alla laadida.

API

rakenduse programmeerimise liides.

CDN

sisuedastusvõrk, laiali jaotatud serverite võrk selleks, et sisule kiiremini juurde pääseda võrreldes sellega, kui sisu paikneks ainult ühes serveris.

CSS3

keel, millega kirjeldatakse veebilehe kujundust.

GeoJSON

kergekaaluline JSON andmevahetusformaat, mis on mõeldud geograafiliste andmete edastamiseks ja hoidmiseks.

HTML5

hüpertekst-märgistuskeel veebidokumentide loomiseks.

JavaScript

objektorienteeritud programmeerimiskeel, mida kasutatakse peamiselt veebilehtede skriptimiseks.

jQuery

vabavara JavaScript raamistik, mis võimaldab kirjeldada JavaScript funktsionaalsusi lühema koodiga, lihtsustab koodi kirjutamist.

JSON, JSONP

Javascript'i objektide standardid.

KML

XML-põhine failivorming geograafiliste andmete edastamiseks.

Mashup

rakendus, mis kasutab teiste loodud tehnoloogiaid ja/või ressursse, reeglina ühendades mitu erinevat.

OGC

OpenGIS konsortsium, rahvusvaheline organisatsioon, mis on loodud ruumiinfo- ja asukohapõhiste tehnoloogiate arendamiseks.

UI

kasutajaliides, mis kujutab endast käskude või menüüde komplekti, mille abil kasutaja saab programmiga suhelda.

WFS

teenus veebist geograafiliste vektorobjektide laadimiseks, mis võimaldab teha erinevaid päringuid.

WMS

teenus veebist asukohaga seotud raster kaardipilte laadimiseks.

XML

standardne üldotstarbeline märgistuskeel, mille eesmärgiks on struktureeritud info jagamine infosüsteemide vahel.

1.2 Arvutikartograafia ajalugu

Veebikartograafia on defineeritud kui kaartide projekteerimise, teostamise ja World Wide Web (WWW) keskkonda kohaletoimetamise protsess [21].

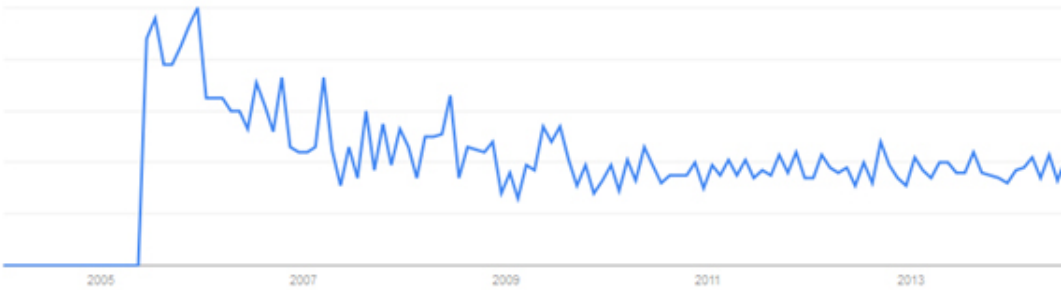
Sellest, kuidas saab kaarte ja muud geograafilist infot jagada nii arvutite vahel, organisatsiooni sees kui ka üldisemalt kogukonnale hakati mõtlema 1990. aastate keskpaigast [30]. Veel 8-9 aastat tagasi oli kaks peamist võimalust, kuidas visualiseerida andmeid *on-line* kaartide kujul: esimene võimalus oli manustada lihtne Google Maps oma veebilehele või teine võimalus investeerida Adobe Flash ja Microsoft Silverlight tehnoloogiatel põhinevatesse kaardistamisvahenditesse [26, 30]. Seega, enne 2005. aastat oli veebikaartide loomine kulukas ja keerukas, kuna vajas väga spetsialiseeritud tööriistu ja oskusi [6] ja juurdepääs kaartidele erinevatelt platvormidelt oli raske [26]. Kui sellistest visualiseerimise ja animatsiooni võimalustest loobuti HTML5 kasuks [30], on veebikaartide kasutamine tõusnud massiliseks [28], kuigi alguses kujutasid esimesed kaardid endast vaid staatilisi punkte kaardil [19] ning selliste kaartide massilist teket nimetati sel ajal „punase punkti palavikuks“ [4].

Umbes 2004. aastast Web 2.0 tekkega on teabevahetus ja avatus muutunud veebi põhimõtteks, mis mõjutas positiivselt veebipõhise kartograafia teket [11]. Web 2.0 (eesti keeles Veeb 2.0) on teise põlvkonna veebidisain, mis võimaldab inimestevahelist infovahetust ja koostööd sotsiaalvõrgustike, vikide, veebi-programmiliideste ning mitmesuguste veebiteenuste kaudu: veeb on muutunud kasutajate poolt isikliku teabe lisamist lubavaks [18]. Olid loodud soodsad tingimused tavainimeste osalemiseks endale vajalike kaartide loomises kasutades olemasolevaid tööriistu [8, 13]. Esialgu domineerisid veebikaardistusteenuste pakkujatena suured ettevõtted nagu Google, ESRI ja NASA, kuid FOSS4G (avatud lähtekoodiga georakenduste tarkvara) arenguga ja Open Source Geospatial Foundation (OSGeo) tekkega võib veebikaardistamisega tegeleda igaüks, kellel on arvuti [28] ja kes oskab programmeerida; algas aeg, mil kaartide tegemisega tegelesid nn programmeerijad-kartograafid [19]. Hakati rääkima „Veebikaardistamisest 2.0“ nagu uuest etapist kartograafia arengus [8, 13]. Veebikaardistamiseks 2.0

nimetatakse kõiki uusi lahendusi ja tehnoloogiaid, mis on seotud Internetiga ja geograafilise informatsiooniga, kaardi tegemise tehnikaid ja vahendeid, mis ei kuulu traditsioonilise GIS valdkonda.

Web 2.0 tehnoloogiad on olnud erilise tähtsusega, soodustades veebikaardindust, eelkõige AJAX parema kaardi sirvimiskogemuse jaoks ja API'd, mis võimaldasid alustada *mashup*'i tegemisega [8, 18]. HTML5, CSS3 ja teiste kaasaegsete veebitehnoloogiate tulekuga luuakse uued ja uued avatud lähtekoodiga, erinevatele platvormidele sobilikud kaardistamise tehnoloogiad, millega saab arendada stiilseid ja interaktiivseid kaarte.

Karograafiliste rakendusliideste areng algas 2005. aastal (*Joonis 1* näitab antud mõiste *Google*'s otsimise trendi) Google Maps API tekkega, millega on muutunud terve veebikaardistamine.



Joonis 1: *Google Trends otsingu „mapping API" graafik.*

Esimese kaardi *mashup*'i tegi aprillis 2005 valmis graafiline disainer Paul Rademacher, kes kombineeris reklaam- ja kuulutuste veebilehe eluasemete andmeid Google Maps'iga [16]. Sellest ajast alates tegelevad kaardi loomisega ka mitte professionaalsed kartograafid, ning kasutavad selleks tööristu, mis ei kuulu traditsioonilise GIS valdkonda [13].

Vastavalt *ProgrammableWeb* portaalile[1], on olemas kümned kaardistamise API'd. Kasutades erinevaid JavaScript rakendusliidesed ja tööriistu on võimalik mitu korda kiirendada veebiprojekti arendamist [32]. Selliste teekide kasutamise peamine eesmärk on rakenduste standardsete osade realiseerimise lihtsustamine jättes viimase teekide hooleks ning lubades seeläbi arendajatel keskenduda rakenduse põhilistele nõuetele.

[1] http://www.programmableweb.com/

Internetis töötavate kaardirakenduste loomiseks on võimalik valida paljude erinevate JavaScript'il põhinevate teekide vahel. Vanim nendest on Google Maps. Muud populaarsed võimalused on kasutada OpenLayers'i, Leaflet'i ja ArcGIS API't või teisi.

Veebikaardistamine on muutunud Interneti kasutajate seas populaarseks tegevuseks ja sellele saab leida mitu põhjust: kindlasti olulisim põhjus on konkurentsivõimeliste veebikaardistusteenuste ja kaardistamisraamistike teke, tasuta tehnoloogia teke ning nende uute veebitehnoloogiate kasutajate poolt vastuvõtmine tänu lihtsale ja odavale eksperimenteerimisvõimalusele, samuti ka suurenenud kaardistatud informatsiooni vajadus, ruumilisest visualiseerimisest huvitatud inimkond või üldine ruumiliste visualiseerimiste trend [11]. Põhjuseks on ka ruumiliste andmete suurenenud kättesaadavus ning rakendusliideste kasutamise lihtsus nende visualiseerimiseks. Lisaks on API kaudu kasutajatel juurdepääs väga kõrge resolutsiooniga taustakaartide allikale, sealhulgas tänavakaartidele, satelliitpiltidele, tänava fotodele jne [8] näiteks Google Maps API kaartidele ja ArcGIS ESRI kaartidele. Esitatuna dünaamiliselt ja interaktiivselt, annavad veebikaardid informatsioonile uue välimuse, mõnikord ka lisateabe otsuste tegemiseks ning aitavad oma loomingut kergesti ja kiiresti Interneti kaudu levitada [13]. Kaardi tegemiseks on nüüd vaja vaid natuke koodi kirjutamist ja tehnilist oskusteavet, kuid kulud ja õppimisvajadus on langenud märgatavalt [26] võimaldades palju suuremat kogukonda inimesi, kes võiksid luua ja jagada geograafilist teavet.

1.3 API'd

JavaScript API on JavaScript'i klasside ja omavahel seotud objektide kogum, kus iga objekt omab atribuute ja meetodeid [12], mida veebilehelt kutsutakse selleks, et ehitada interaktiivse kaardi erinevaid elemente.

Tuntud on järgmised veebikaardistamise vahendid/teegid.

1. Google Maps (alates 2005. aastast)

2. OpenLayers (alates 2005. aastast ning alates 2014. aastast versioon 3)

3. Leaflet (alates 2009. aastast)

4. ArcGis API for Javascript (alates 2008. aastast)

API'liidesed võivad olla avatud lähtekoodiga, nagu OpenLayers ja Leaflet, või varalised. Varaline tähendab, et API lähtekoodi ei saa alla laadida ja/või ei ole lubatud modifitseerida ja/või ei saa ilma litsentsi tasuta kasutada [30]. Avatud lähtekoodiga teekide eelis on võimalus erinevate tarnijate erinevate komponentide segamiseks ja sobitamiseks, kuna ei ole ranget kontrolli kaardilahenduse kõikide komponentide üle [27], aluskaartide kujunduse muutmise võimalus. Avatud lähtekoodiga teekide koodi saab vastavalt oma vajadustele modifitseerida, muuta või kasutada osaliselt. Varaliste API'de eelis on nende oma aluskaartide kiire kuvamine, kuna nad on salvestatud hiigelsuurtes serverites, intuitiivsem kasutajaliides, kuigi miinuseks on piiratud võimalused aluskaartide muutmiseks ja litsentsi küsimused, ka võib neid olla raskem kasutada juhul kui tahetakse kasutada teisi standardeid [18].

Võimalikud kaardistamise API'de komponendid, mida võib leida, on [16]:

1. Aluskaartide kogum, näiteks tänavakaart, maastikukaart, satelliitfoto.

2. Kaardistamise API'liides ise, tavaliselt JavaScript'i teek, mis aitab luua ja kontrollida baaskaarte ja temaatiliste andmetega pealiskihte.

3. Geokodeerimisteenus, mis leiab aadressidele vastavad geograafilised koordinaadid.

4. Dokumentide kogum, et aidata veebiarendajail API't kasutada, sealhulgas kasutusjuhendid, õpetused, dokumentatsioon.

5. Arendajate kogukond, kus nad saavad jagada oma küsimusi, tähelepanekuid ja kogemusi.

6. Õiguslikud aspektid: autoriõigused, serveri tasandil kokkulepped, teenuse tingimused, privaatsusdeklaratsioonid ja lõppkasutaja litsentsilepingud.

7. Tehnilise toe keskus, et aidata arendajatel lahendada konkreetsed probleemid.

Järgnevalt tutvustatakse magistritöös kasutatavaid rakendusliideseid.

1.3.1 Google Maps API

Google Maps[2] on vanim veebikaardistamise teekidest. Google Maps API avalikustati ametlikult juunis 2005. Selle varalise teegi tunnusjooneks on lai valik staatilisi ja dünaamilisi visualiseerimisviise. Unikaalsetest võimalustest on integratsioon *Google Earth*'iga ja *StreetView* kasutamine.

Kuna see on varaline teek on kohustuslik kasutada API't koos võtmega, mis võimaldab jälgida oma rakenduse kasutamist ja tagab, et ka Google teie rakendust jälgiks ja saaks teiega kontakteeruda kui kasutuslimiidid on ületatud. Google Maps API muutub tasuliseks, kui ületatakse kasutuspiirangud: 25000 või rohkem kaardilaadimist päevas rohkem kui 90 päeva järjest, rohkem kui 2500 geokodeerimist päevas jne. Samuti peab rakendus olema tasuta, vabalt ja avalikult kättesaadav. Suure võrguliiklusega lehekülgedel on kohustus osta *Maps for Work* litsents rakendusliidese kasutamiseks.

Uus versioon toetab nii traditsioonilisi veebilehitsejaid nagu Internet Explorer 7.0 +, Firefox 3.0 +, Safari 4 +, Chrome, kui ka mobiilseid brausereid. Statistiliste andmete järgi on Google Maps JavaScript API enamkasutatav kartograafiline API [9].

1.3.2 OpenLayers

OpenLayers API on avatud georuumilise konsortsiumi vaba ja tasuta projekt, mis ei sisalda kasutuspiiranguid.

Teek avalikustati 26. juunil 2006. Peatades 2.13[3] versiooni uuendamise 2014. aastal, oli loodud täiesti algusest uus API nimega OpenLayers 3[4], mille versioon 3.0 tuli välja 2014. aasta septembris. Praegu on mõlemad versioonid laialt kasutusel: versioon 2, kuna see on stabiilne, väga populaarne ja sisaldab kõiki põhilisi funktsioone ning versioon 3, kuna seda arendatakse lubades mõnesid uuenduslikke funktsioone.

[2] https://developers.google.com/maps/documentation/javascript/

[3] http://openlayers.org/two/

[4] http://openlayers.org/

Mõlemad töötavad järgmiste veebilehitsejate versioonidega: Mozilla v1.8, Firefox 1.0+, Internet Explorer 6.0+, Safari v2.0+, Opera v9.0+.

Mahuka funktsionaalsuse tõttu on OpenLayers'i lähtekood muutunud suureks ja seepärast on sellega loodud kaardid suhteliselt aeglased (vastavalt [30] andmetele). Samuti ei ole OpenLayers nii sobilik ja vastutulelik algajatele kui teised API'd [30], eriti kasutades OpenLayers 3 versiooni ja arvestades sellega, et dokumentatsioon ei ole veel täielik ja näidete maht on väike.

1.3.3 Leaflet

Leaflet[5] on käsitletatavatest rakendusliidestest uusim kaardistamise API. Esimest korda avalikustati see 2011. aastal. Viimane versioon 0.7 on pärit aastast 2013, kuigi see pidevalt areneb. Rakendusliides on saavutanud oma populaarsuse väga kiiresti olles kasutusel näiteks populaarsetel Foursquare, Pinterest ja Flickr saitidel [27].

Rakendusliides on kasutuspiiranguteta. Töötab Chrome, Firefox, Safari 5+, Opera 12+ ja IE 7-11 veebilehitsejatega.

Funktsionaalsus ei ole nii mitmekesine kui teistel API'del, kuid selle arvel on siin väike lähtekood. Näiteks ei ole sisseehitatud funktsiooni objektide klastrite moodustamiseks või kaardi peal hiirega objektide joonistamiseks. Kuid Leaflet'i üks olulisim aspekt on see, et ta pakub vaid kõige baasilisemat koodi, mille põhjal arendatakse muud pluginad ja tarkvaramoodulid, mis lisavad süsteemile teatud võimaluse, et platvorm oleks parem ja täiuslikum, seega ülalmainitud funktsionaalsusega saab Leaflet'i teeki täiendada. Selline ehitus lisab rakendusliidesele paindlikkust [31], moodulite kogu kasvab pidevalt ja kiiresti [28].

1.3.4 ArcGIS API for JavaScript

ArcGIS API for JavaScript[6] on ESRI varaline teek. Esimene rakendusliidese versioon avalikustati 2008. aastal, viimane uuendus versioonini 3.12 avalikustati detsembris 2014. ArcGIS API on ehitatud kasutades Dojo

[5] http://leafletjs.com/

[6] https://developers.arcgis.com/javascript/

JavaScript riistakomplekti. API'ga soovitatakse kasutada kaarte ArcGIS Online serverist või oma kaarte ArcGIS Server'ist. Rakendusliidese saab ka eestikeelseks lokaliseerida (kokku 26 erinevat keelt).

ArcGIS rakendusliidest saab kasutada tasuta ainult juhul, kui tegemist ei ole kommertsrakendusega või kui ei ole soovi kasutada ArcGIS serverit, kuna see on tasuline. Töötab Chrome, Firefox, Safari 5, ja IE 7+ veebilehitsejatega.

1.4 Väljakutsed

Veebikartograafia valdkond areneb veel ka tänapäeval ning kõik probleemid ja takistused ei ole ületatud. Näiteks faili suurus ja geomeetriline keerukus on oluline takistus veebi kartograafia jaoks. Kõikide vektorandmete kuvamine kaardil võtab aega ja veebilehitseja töötlemise võimsust. Isegi mõõdukalt suur vektorandmekogu võib sisaldada tuhandeid objekte, nii et suuremate kujundite joonistamine aeglustab kaardi kuvamist [26]. Siinjuures sõltub protsessi kiirus ka arvuti võimsusest, arvutis käivitatud protsesside kogusest, veebilehitseja poolt kasutatud arvuti mälust jne. Üheks probleemi lahenduseks on kuvamise resolutsioonist sõltuvalt geomeetriate lihtsustamine/generaliseerimine nii palju kui võimalik, eriti väiksemate mõõtkavade jaoks, et kiirendada faili ülekannet nii, et vektorandmete laadimisega ei aeglustuks rakenduse töö ja sujuv interaktiivsus oleks saavutatud [2, 30].

Ühelt poolt on puudujäägiks ka see, et tavaliselt kasutatakse *mashup*'i jaoks avatud lähtekoodiga meetodeid ruumiandmete ettevalmistamisel, sealhulgas GeoJSON, XML faile, Google Fusion Tables, CSV faile või KML faile, kuid arvestades ESRI toodete domineerimisega GIS alal, on kõige keerulisemad avalikud andmekogud tihti ESRI *Shape* faili formaadis. *Shapefile* on binaarne vorming ning seda ei saa otse API'de võimalustega veebikaardil kuvada: selle probleemi lahenduseks saab neid faile enne kasutamist veebirakenduses konverteerida üheks toetatud vorminguks näiteks QGIS tarkvara abil[7].

Sageli kuid mitte alati on *mashup*-kaardid lühikese elueaga ja mõeldud teatud unikaalsete vajadustega kasutajate grupi jaoks [1]. Ehk raskus on selles,

[7] http://www.qgis.org/

et kaardi kehtivusaeg väheneb kiiresti [20], kuid seda võib lahendada kasutades hea uuendamisvõimalusega andmebaase ja pöörates rohkem tähelepanu andmete ajalisele aspektile.

Veebikaartide arendajad ei pööra tihti suurt tähelepanu kartograafilistele põhireeglitele, nt loetavus ja kontrastsus võivad kannatada [18]. Nii tekivad kaardid, kus on raske infot eristada väga paljude erinevate kihtide olemasolu tõttu või kihtide liiga sarnase kujunduse tõttu või kaardid, millel puuduvad sellised kartograafilised elemendid nagu põhja-lõuna suuna nool, legend või mõõtkava. Ka sümbolite valik, mida saab veebilehitsejal näidata, on väike ja kuigi alati saab näidata graafilist elementi või pilti punktobjektide jaoks, ei ole väga keerukad jooned ja pindobjektide taustad võimalikud.

Märgistus (*labeling*) on veel üheks väljakutseks, sest kuigi brauserid võivad lihtsasti näidata teksti etteantud koordinaadi jaoks, puuduvad teksti paigutuse algoritmid. Parem on näidata objektile vastav tekst läbi interaktiivsuse, nt hüpikaknas või HTML `<div>` elemendis [30].

Lisaks eelnimetatud spetsiifilistele probleemidele on olemas ka hulk üldisemaid olulisi küsimusi, mis on uue veebikartograafia ajal tekkinud.

1.5 Arengu trendid

Pidevalt toimub API'de arendamine, et nende kasutamine oleks võimalikult efektiivne ja kasutatud tehnilised kontseptsioonid oleksid kõige paremad ning toetaksid uusi tehnoloogiaid. Pidevale arendamisele viitab ka OpenLayers'i uue API OpenLayers 3 avalikustamine 2014. aasta lõpus, kus on loodud uued funktsioonid võrreldes eelmise API'ga 2 viimase versiooniga 2.13.1, näiteks 3D andmete visualiseerimine.

Tänapäeval on kasutusel ka uued seadmed, nt puutetundlikud seadmed, millel on huvitav potentsiaal kartograafiliste teenuste arendamiseks [20]. Seega arvestavad uued API'de versioonid ka selle suunaga ja lisavad vastavad võimalused. Võrreldes PC-põhiste (personaalarvuti) rakendustega, on mobiilsetel rakendustel oma miinused, näiteks ekraani suurus ja resolutsioon,

kuid on mitmeid eeliseid, nagu näiteks info lõppkasutajate asukoha kohta ja seadme mobiilsus.

Veel üheks trendiks API'de arendamisel tänapäeval on üha rohkemate dünaamiliste kihtide lisamine API'de teekidesse. Näitena võib tuua liikluskihid, mis näitavad reaalajas liikluse informatsiooni linnapiirkondades või jalgrattakihid, mis pakuvad rattamarsruute ja spetsiifilisi kihte jalgratturitele [16].

1.6 Varasemad uuringud

Veebikaardistamise teekide võrdlusi on mõned uurijad ka varem teha proovinud ja nende tulemuste alusel tekib esialgne ettekujutus API'de erinevustest.

Näslund [15] võrdles omavahel kaardistamise teeke, kuid võrdluses olid praegu väga vähe kasutatav Microsoft Virtual Earth[8], Multimap, ViaMichelin[9] ja lisaks Google Maps. Võrreldes nende funktsioone, brauserite sobivust, geokodeerimise täpsust, kasutatavust ja arendajate toetust oli järelduseks võimetus valida nendest parimat, kuna igal oli oma puudused ja plussid. Võrreldi ka andmete mahtu, mida iga kaardi avamise korral veebilehitsejas alla laetakse.

Kennedy [25] võrdles kaardistamise raamistikke (Google Maps API 2.84, Yahoo! Maps API 3.7[10], Windows Live Local Search Maps, Virtual Earth API 5.0) samuti veel 2007. aastal eesmärgiga võrrelda API'de selliseid karakteristikuid nagu vastuse aeg, koormus, ehk võrgus liikuvate andmete hulk ajaühikus, API suurus ja visualiseerimise aeg sekundites. See test viidi läbi kasutades Mozilla Firefox *Firebug* võrgu jälgimise võimalust, iga kolme korra järel Firefox taaskäivitati ja selle vahemälu (mälu sagelikasutatavate andmete ajutiseks hoidmiseks [33]) tühistati. Tema metoodika on ka aluseks antud uuringule.

[8] http://www.microsoft.com/maps/choose-your-bing-maps-API.aspx
[9] http://dev.viamichelin.com/
[10] https://developer.yahoo.com/maps/flash/jsGettingStarted.html

Uuem uuring viidi läbi 2014. aastal [22], kus võrdluse all oli viis kaardistamise teeki: Google Maps JS API v3, Bing Maps JS API v7, ESRI ArcGIS JS API v3.6, Leaflet v0.6.4 ja Openlayers v2.13.1 ning võrdluse aluseks vektorandmete visualiseerimise kiirus. Pärast 210 testi iga API'ga oli keskmistatud andmete põhjal esikohal Google Maps, kelle kiirus oli suurim ning viimasel kohal oli OpenLayers.

Suur veebikaardistamise raamistiku valimise arutelu viidi läbi JiscG3 projekti jaoks [34], mille üheks järelduseks oli see, et vaatamata asjaolule, et kaardistamise raamistikke on väga palju, taandub miskipärast paljude geoveebi arendajate valik (ka mainitud artikli juhul) tihti valikule ainult OpenLayers ja Google Maps API vahel.

2011. aastal võrreldi Google Maps API ja OpenLayers'i sobivust akadeemilise uurimisgrupi projekti jaoks ning otsustati kasutada OpenLayers'it ja Google Maps'i koos, sooviga kasutada pigem täielikult avatud tehnoloogiaid ja mobiilseid veebirakenduste versioone [29]. Ka uuemad võrdlused on olemas, kus võrreldakse juba uut OpenLayers 3. Näiteks selle arendaja Hocevar [24] kirjutas artikli selle võrdlemisest Google Maps API v3 versiooniga veel enne rakendusliidese ametliku ilmumist, pöörates tähelepanu funktsionaalsuse võrdlemisele.

Põhjalikult uuriti kaardistamise rakendusliideseid 2013. aastal [5], kus pöörati tähelepanu nii dokumentatsioonile ja teoreetilistele andmetele, kui ka prototüüpidele võrreldes API'de kasutatavust. Võrreldi Google 3.7 – 3.9, ArcGIS 2.0 – 3.1, OpenLayers 2.3 – 2.12. teeke. Antud uuring on üks peamistest metoodika allikatest minu uurimistöö jaoks. Vektorkihtide ilmekuse võrdlemise aluseks on varem läbiviidud sarnane test [22], kus kasutati GeoJSON 40K, 10K, 5K, 2K, 1K ja 500 objektidega. Keskmise väärtuse saamiseks viidi iga test läbi 30 korda.

1.6.1 Varasemate uuringute järeldused

Olemasolevatest artiklitest ja uuringutest võib luua esimese ettekujutuse, mille poolest üks või teine API on teistest erinev.

Näiteks üks oluline erinevus Google Maps API ja OpenLayers'i ning Leaflet'i vahel on see, et esimesega koos on ka tema enda põhikaart kuid viimased nõuavad kolmanda osapoole kaarti, nt OpenStreetMap [24], kuigi see on ka eeliseks võimaldades kasutada erinevaid kaardi pakkujaid, mistahes allikast aluskaarti [34] ja miinuseks Google Mapi API jaoks, kuna isegi soovi korral kasutada oma WMS kihti, on Google aluskaart alati olemas veel ühe alumise kihina.

Google Maps API üheks eeliseks on ka see, et sellega tuleb kaasa ka tavaline Google Maps'i kasutajaliides, millega suurim hulk kasutajaid on juba harjunud ning nähes navigatsiooni ja kaardi stiili tunnevad end mugavamalt [30].

Kuivõrd Google Maps on varaline teek, on see väga põhjalikult dokumenteeritud ja suur arendajate kogukond pakub palju näiteid, kuigi teised teegid on sellega juba võrdsustumas, ning nt OpenLayers'il (siin räägin vanast versioonist, kuna uus OpenLayers 3 on veel liiga noor, et konkureerida näidete mahus) on ka suur ja mahukas dokumentatsioon ja hea funktsionaalsus. Kuid 2011. aastal [34] on veel väidetud, et OpenLayers'i dokumentatsioon ja koodi näidete kvaliteet võrreldes Google Maps'i omaga on madalam.

Google Maps'i miinuseks, olles varaline teek, on asjaolu, et rakendused, mis kasutavad Google Maps API't tasuta peavad olema avalikkusele kättesaadavad, maksimum kaardi laadimiste arv päevas on 25000. Organisatsioonid, mis ei vasta kriteeriumitele, ostavad *Google Maps API Work* litsentsi [30].

Lihtsuse ja kasutatavuse poolest on varasemate hinnangute alusel OpenLayers keerulisem [31, 34] selgeks teha kui Google Maps API ning esimesega töötamine nõuab sügavamat GIS mõistmist. Kuigi arvatakse, et see on tingitud sellest, et OpenLayers'i funktsionaalsus on mitmekesisem ja toetab OGC teenuste standardeid nagu WFS, WMS jne – OpenLayers rakendusliidest nimetatakse kõige täiuslikumaks GIS projektiks veebikaartide loomiseks [31]. Kuid samal ajal Santiago [31] väidab, et Leaflet'i koodi stiil on ilusam, kaasaaegse projekti tundega, mida on ilmselt sellepärast lihtsam kasutada.

Rääkides OGC teenuste standardite toetusest on oluline, et nii ArcGIS API kui ka Google Maps API eelistavad kasutada oma standardeid ja faile oma serveritest [24].

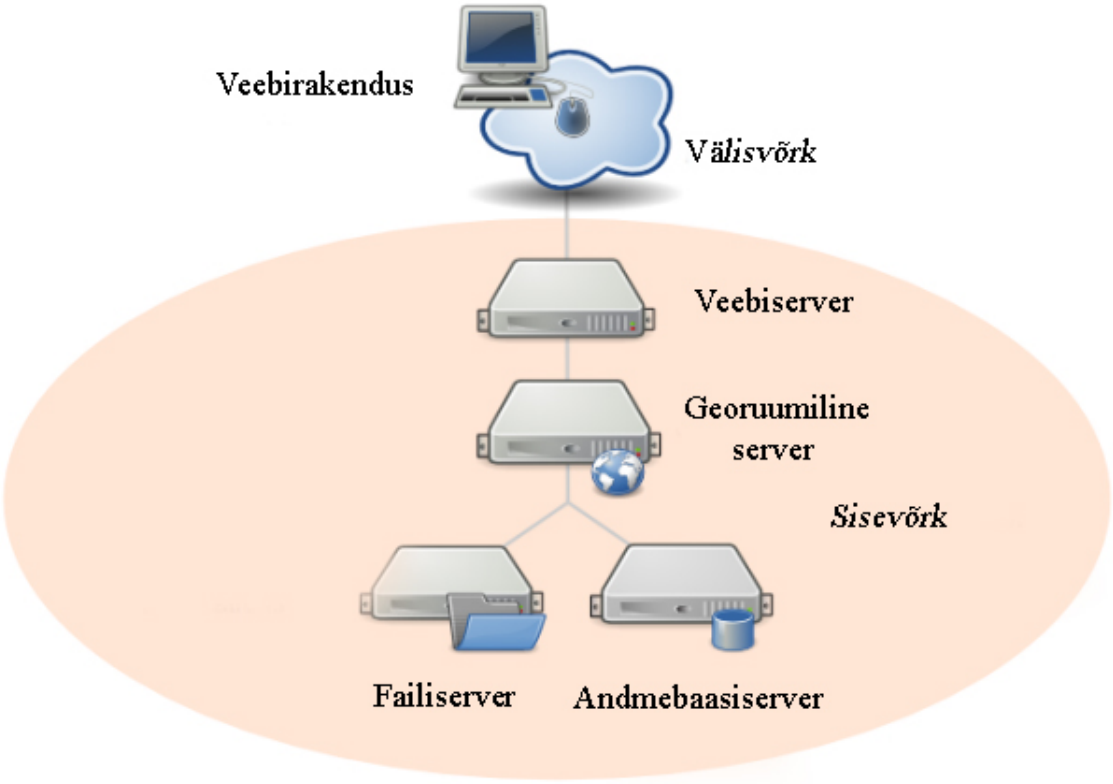
OpenLayers 2 kritiseeriti ka tema vaikimisi UI elementide kujunduse eest, mis on suhteliselt primitiivsed tänapäeva Web 2.0 maailmas [34]. Ka Santiago [31] võrreldes Leaflet'i ja OpenLayers 2 omavahel nimetas Leaflet'i UI elemente paremateks, täiuslikumateks ja värskemateks võrreldes OpenLayers'i rakendusliidese omadega.

1.7 Veebirakenduse struktuur

Mitmeid erinevaid termineid on kasutatud, et kirjeldada API kasutatavaid rakendusi. Kõige levinum on *"map mashups"*, kuna kasutatud API või taustakaart ühest andmeallikast (nt Google) rakenduse alusena ja selle pealolevad ruumilised andmed teisest andmeallikast segatakse ja luuakse erinevate andmeallikate kombinatsioon [20]. Erinevate andmeallikate koos kasutamine ühistele ja avatud standarditele tuginedes on Web 2.0 tervikuna keskne mõiste [7]. *Mashup*'i loomise protsess on n.n. lõppkasutaja programmeerimine, ehk selline tegevus, kus programmeerimisega tegelevad inimesed ei ole professionaalsed tarkvaraarendajad.

Tavaliselt saab veebirakenduse arhitektuuris eraldada kolm erinevat komponenti, tehnoloogiate kolm funktsionaalset kihti. Tüübi järgi on need järgmised: andmed, rakenduse loogika ja kasutajaliides (Schmidt [18] nimetab oma ettekandes neid vastavalt geoandmed, platvorm ja kasutajaliides). Neid võib ette kujutada ka kihtidena: esituskiht, loogiline kiht ja andmeallikate kiht. Sisu, funktsionaalsus ja esitusviis integreeritakse kokku kasutades erinevaid programmeerimiskeeli ja tehnoloogiaid (*Joonis 2*), kusjuures antud lõppkasutaja programmeerimise protsessis on ülioluline ka teadlikkus kontekstist [1].

Esituskiht on see, mida näeb kasutaja. Loogiline kiht on kõige olulisem kiht süsteemi jaoks, kuna see tegeleb andmete töötlemisega. Andmeallikate kiht pakub andmeid, sisu ja teeki, mis liidetakse kokku uueks rakenduseks.



Joonis 2: *Veebikaardirakenduse arhitektuur: andmeallikate kiht (failiserver ja andmebaasiserver), loogiline kiht (georuumiline server ja veebiserver) ja esituskihit (veebirakendus) [30].*

1.7.1 Geograafilised andmed

Välised andmed, mida kuvatakse digikaardil, võivad pärineda erinevatest allikatest ja olla erinevates vormides ja formaatides [7]. Andmetetüüp, mis valitakse *mashup*'i jaoks sõltub ka looja oskustest ning potentsiaalsete kasutajate prognoositud vajadustest [13].

Rakenduse projekteerimisel otsustatakse, kas vajalikud andmed salvestatakse failidena või andmebaasina. Failipõhine lähenemine on kergem ja lihtsam, kui andmestik pidevalt ei muutu [30] .

1.7.1.1 Vektorandmed

Vektorandmed koosnevad geomeetrilistest objektidest-punktidest, joontest ja polügoonidest ning on määratud nende koordinaatidega.

1.7.1.1.1 Andmed failist

Enam levinud avatud lähtekoodiga vektorandmete formaadid on GeoJSON, KML [26] või XML, JSON [1]. Kuna (Geo)JSON on mahult väiksem, mis tähendab veebirakenduse jaoks kiiremat laadimist, on see muutunud väga populaarseks [30]. GeoJSON on georuumiline andmevahetusformaat erinevate geograafiliste andmestruktuuride kodeerimiseks, mis põhineb JSON tehnoloogial. GeoJSON toetab järgmisi geomeetria tüüpe: punkt –*Point* (aadressid ja asukohad), joon – *LineString* (tänavad, kiirteed ja piirid), polügon- *Polygon* (riigid, maa-alad), *MultiPoint*, *MultiLineString* ja *MultiPolygon*. GeoJSON objekt võib sisaldada mistahes arvu liikmeid (nimi / väärtus paare). Näide kõige tavalisemast geoandmete esitamisviisist internetis, GeoJSON formaadis punkt objekt:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Binaarseid formaate (kahendvormingus salvestatud fail, mis erinevalt tekstifailist sisaldab suvalise sisuga baite, nt masinakoodi [33]), vaatamata nende laialdasele kasutamisele *desktop*-GIS süsteemides (näiteks *.shp* vorming), ei kasutata veebis [2, 7].

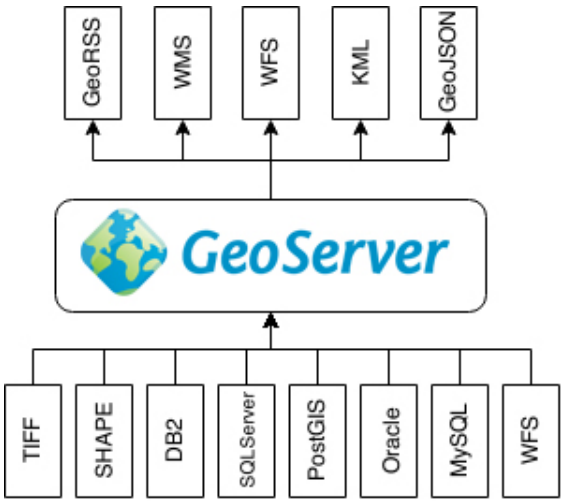
Pärast andmete lisamist tehakse need inimesele loetavamateks tavaliselt JavaScript'i abil, et brauseriaknas näidata. Kui andmete maht on suur, siis võib see ülesanne panna kasutajaliidese tarduma [1].

1.7.1.1.2 Andmed andmebaasidest

Kartograafilised veebirakendused on tavaliselt tehtud mingi kindla eesmärgiga ning nende andmed peavad olema ajakohased. Seega hoitakse andmeid tihti relatsioonilistes või objektorienteeritud andmebaasides, sest

nii on lihtne ja hea andmete uuendamise võimalus. Tendents on kasutada andmebaase spetsiaalsete ruumiliste laienditega. Nagu näiteks Oracle koos ruumilise laiendiga Oracle Spatial ja DB2 koos Spatial Extender'iga. Kasutatakse ka avatud lähtekoodiga andmebaase nagu PostgreSQL koos ruumilise laiendiga PostGIS, mis vastab OGC *„Simple Features for SQL“* spetsifikatsioonidele – seda kasutatakse üha rohkem [30]. Ruumilisi tehteid ja ruumilisi objekte toetavad ka Oracle Spatial, SpatiaLite (SQLite), DB2 Spatial, ESRI ArcSDE, Microsoft SQL Server Spatial ja MySQL Spatial. Nende ruumiliste andmebaaside kasutamises on kaks eelist: võimalus säilitada geomeetriaga koos ka atribuut-andmed ning väga kiire ruumiline analüüs lihtsate SQL käskude kasutamisega. Ruumiliste tehete hulk on piiratud, kuid puhvri leidmisi, lõikumisi ja ühendusi saab teha [20].

Toorandmete andmebaasidest kättesaamiseks kasutatakse veebiservereid nagu GeoServer, ArcGIS Server, MapGuide ja MapServer. Sellised veebiserverid kuuluvad rakenduse loogika juurde. Mainitud veebiserverid lihtsustavad andmete päringuid andmebaasidest, nad teevad seda automaatselt ise. Selline server esitab ruumiandmeid (kas andmebaaside tabelid või lihtsalt failid kõvakettalt) objektide kogumina ning võimaldab HTTP klientidel nendega operatsioonide tegemist: visualiseerida kaardina, loogilisi tehteid teha, pärida andmeid erinevates formaatides (*Joonis 3*).



Joonis 3: *GeoServeri ruumiliste andmete sisend- ja väljundvormingud.*

Vektorandmeid kasutatakse WFS formaadis. Põhiidee vektorandmete allalaadimiseks WFS kaudu on erinevate strateegiate kasutamine. Strateegiaks on viis kuidas koordinaate ja nendega seotud atribuute alla laaditakse – kas kõik korraga või mingi eeskirja järgi [30]. Tüüpiliselt sisaldavad serveris asuvad andmed väga suure koguse objekte, kuid ei ole kohustuslik, et neid on vaja kõiki korraga. Sellisel juhul võib kasutada erinevate strateegiate plusse, et objektide visualiseerimise ajal rakendus ei jookseks kokku ning töötlus ei katkeks ootamatult.

Soovitakse [23] visualiseerida võimalikult vähe objekte korraga.

1.7.1.2 Rasterandmed

Teiseks peab kaardil olema ka mingi rasterkujul aluskaart. See, mida me näeme kaardipildina on *„base tiles“* ehk on pildiruudukesed, mida pannakse kokku nii, et tekiks terviklik pilt. Kasutatava tehnoloogia nimetus on WMS. WMS teenust kasutatakse tavaliselt selleks, et tõmmata serverist rasterkujul olevaid andmeid JPEG, GeoTiff, PNG ja teistes formaatides. Kaartidel laetakse neid pildikesi asünkroonselt – kui kaader tsentreeritakse, valitakse asukoht, suumitakse sisse või välja [26] ja kuna need pildiosad on väikesed, siis laevad nad kiiremini kui terviklik pilt laeks: mõõdukalt kiire interneti ühendusega ei ole see protsess kasutaja jaoks isegi märgatav [16] ning kasutatavuse poolest on oluline, et allalaadimise ajal ei oota rakendus selle protsessi lõpuni vaid on võimalus rakendusega edasi töötada.

Ka WMS serverid kuuluvad rakenduse loogika juurde. Rakendusliideste funktsioonid WMS kihtide näitamiseks tegelevad päringu URL ehitamisega, lisades lingile kõik vajalikud parameetrid nt pildi vorming, mõõtkava jne [30].

1.7.2 API

Geograafiline JavaScript teek paneb põhiliselt kokku andmed ja pildid, ehk ta saab nendest ruumilistest andmetest aru ja visualiseerib neid objektidena kaardil ning varustab kaardi kõigi vajalike kontrollidega. Erinevad geograafilised JavaScript teegid pakuvad kõik veidi erinevaid kontrollereid

ja funktsioone kaardiga suhtlemiseks ning toimivad erinevalt erinevat tüüpi geograafiliste andmetega. API on üks osa rakenduse esituskihist.

Kaarditegemise API sisaldub ühes JavaScript failis, millele saab viidata lisades vastava rea HTML-faili `<head>` ossa. Mõnedel API'del on kohustuslik ka kaskaadlaadistik CSS. On kaks võimalust kuidas lisada API leheküljele. Üks võimalus on seda alla laadida ja postitada API oma serverisse, lisada projekti kausta, sellega vähendades koormust ja võimaldades API't kohandada. Niimoodi on ehitatud suur osa projekte, kuigi selle meetodi miinuseks on, et tuleb ise aeg ajalt teegi lähtekoodi uuendada [23]. Teine lähenemine on viidata vabalt kättesaadavale versioonile, mis on juba olemas teises API'de säilitamiseks mõeldud CDN serveris. Sel juhul tehakse veebirakenduse esialgsel avamisel veel üks päring antud serverile ja rakenduse avamise kiirus hakkab sõltuma ka selle serveri ülalolekuajast. Rakenduse esialgne avamine on natuke aeglasem [23].

API varustab veebipõhise kaardi kasutajaliidese interaktsiooni nuppudega, ehk erinevat tüüpi ja erineva suurusega objektide kogumiga, millega kaardi kasutajad kaardiga vastastikku toimivad, nt nupud ja funktsioonid panoraamimiseks ja suumimiseks või kihtide vahetamiseks ja sisse/välja lülitamiseks [15].

1.7.3 HTML ja CSS

HTML dokument määrab lõpprakenduse kasutajaliidese. Kasutajaliides, ehk UI on tähtis komponent iga *mashup*-kaardi juures, mille abil kasutajad ja rakendus omavahel suhtlevad. See on samuti n.ö. rakenduse esituskiht – see mida näeb kasutaja ja kus toimub lõppkasutaja ja süsteemi vaheline suhtlus. Lõppkasutajad suhtlevad tavaliselt rakendusega veebilehitseja abil.

Peaaegu kõik lehed, mis kasutavad veebikaardistusteeke on järgmise põhikonstruktsiooniga:

- Esiteks sisaldavad nad kindlasti viiteid JavaScript'i failidele, mis on HTML lehel `<script>` märgendi abil kirjutatud ja stiilimäärangutele, mis on `<link>` märgendi abil defineeritud.

- Teiseks, selleks, et kaarti veebilehel näha, tuleb lisada veel vaid üks rida HTML-dokumenti : `<div>` konteineri `<body>` ossa. Tavaliselt näeb see välja standartsena `<div id="map"></div>`, mis sisaldab kaardi elementi.

Selleks, et konteiner kaardiga oleks nähtav tuleb eraldi määrata selle pikkus ja laius CSS'iga. Vajalikud on ka API'de omad stiilimäärangud. Nendele saab viidata samuti kas CDN serveris olemasolevale versioonile, või projekti kaustas alla laetud versioonile [30]. CSS on kasutajaliidese vormistamise puhul olulisim komponent.

2. Kasutatud tehnoloogiad

Selle töö jaoks valiti ruumilised andmed, mis oleksid aktuaalsed Eesti kohta veebikaardi tegemisel, ehk andmed peaksid olema Eestis kasutatavas koordinaatsüsteemis. Selliseks on peamine Eestis kasutusel olev L-EST97 kaardiprojektsioon, milles koostatakse näiteks Eesti põhikaarti. L-EST97 kaardiprojektsiooni EPSG kood on 3301.

Samuti on eelduseks andmete vastavus avatud spetsifikatsioonidele ja standarditele, seega vektorandmetena otsustati kasutada WFS teenust ja GeoJSON faile ja rasterandmetena WMS teenust. Ning viimaseks kriteeriumiks on ülal nimetatud teenuste tasuta kasutamise võimalus. Ülalnimetatud kriteeriumitele vastavad teenused on Maaameti WMS ja ERMAS (elurikkuse, mulla ja maapõue andmesüsteemide geoinformaatiline arendus) projekti WFS teenused.

Seega on ülesandeks kasutada ERMAS projekti test WFS teenuseid (serveris on paigaldatud Geoserver 2.5.1, mis töötleb päringuid): PlutoF andmebaasi objektide unikaalseid asukohti (elurikkuse uuringud). Kasutatakse Maa-ameti koduleheküljelt *.shp* haldusjaotuse kaarti maakondade piiridega, mis QGIS[11] abil salvestati GeoJSON formaati ja lihtsustati Visvalingam'i algoritmiga tolerantsiga 1%. Aluskaartidena kasutatakse Maaameti WMS kihte. Maaameti WMS-teenus pakub väga palju erinevaid kaarte Eesti kohta ning nende kasutamise eeliseks on oluliselt detailsem info Eesti kohta, kui nt Google'i omadel.

Kaardirakenduste loomisel kasutatavad teegid: Google Maps API v3, Open Layers v3.0.0, Leaflet 0.7.3, OpenLayers 2.11 ja ArcGis API for JavaScript 3.12. Prototüüpide jaoks on samuti lisatud mõned teised JavaScript teegid: *jQuery*, *proj4* ja *proj4leaflet*.

[11] http://www.qgis.org/en/site/

Arvuti, millega viidi läbi kvantitatiivne testimine on järgmiste parameetritega: protsessor Intel(R) Core(TM) i5-2500 CPU 3.30GHz muutmäluga (RAM) 16,0 GB. Interneti ühenduse kiirus on 10/10 Mbit/sek.

3. Metoodika

Selle töö juures rakendati samu ülesandeid viie kaardistamise API'ga, mida varem loetleti: OpenLayers 2, OpenLayers 3, Leaflet, Google Maps ja ArcGIS API. Iga API juures on proovitud järgida häid tavasid (vaata [14]) ja kasutada lihtsamaid lahendusi, nagu on esmajärjekorras dokumenteeritud iga API veebilehel ja ametlikes näidetes. Enne testimist tehakse valmis viis identset veebirakenduse prototüübi, mis esindavad tüüpilist funktsionaalsust, mida võib kaardirakendustes leida. Nende JavaScript koodi analüüsitakse peatükis „Põhilised operatsioonid". Edasi võrreldakse lõplikke prototüüpe ja nende osi omavahel erinevate kriteeriumite järgi. Kasutatakse kahte lähenemist: kvantitatiivset ja kvalitatiivset võrdlemismeetodit, toetudes nii API dokumentatsioonile ja seal olevatele näitajatele, kui ka testimise tulemustele.

3.1 Kvantitatiivne hindamine

Kvantitatiivseks testimiseks iga rakendusliidese ja iga näidisrakendusega leitakse järgmised näitajad:

1. Kvantitatiivse testimise osana võrreldakse omavahel andmemahtu, mis laetakse alla veebikaardi avamisel. Kõikide API'de puhul mõõdetakse vajalike failide suurused kilobaitides (kB). Siin võrreldakse nii eraldi teekide failide suurust (*.js* ja *.css*), kuid ka nende projektsiooni muutmise teekide suurust, milleta üks või teine kaardistamise teek Eesti kaartidega ja projektsiooniga ei tööta. API failid laetakse sisuedastusvõrgu kaudu ning nende suurust jälgitakse veebilehitseja konsoolis.

   Hüpoteesiks on, et valitud rakendusliidesed ei erine oluliselt üksteisest allalaetava andmemahu poolest.

2. Järgmiseks kriteeriumiks on Rakendusliidese Kasutuse Indeks (*API Usage Index* (APIUI)) [5]. Arvestades prototüübi ja funktsioonide sama funktsionaalsusega viie teegiga võrreldakse vajalike päringute arvu API poole: konstruktorite, funktsioonide ja omaduste kutsed. Mida vähem päringuid API poole tehakse, et saavutada sama funktsionaalsust, seda ilmekam API on – s.t. väljendusrikkaim teek nõuab vähem päringuid. Eraldi loetakse päringud konstruktoritele, ehk sellistele meetoditele, mis kutsutatakse välja ainult objekti esmakordsel loomisel [33], funktsioonidele ja omadustele. Lisaks päringutele kaardistamise API poole loendatakse ka neid päringuid, mis on vajalikud projektsioonide teekiga suhtlemiseks ja *jQuery* poole päringuid.

   Hüpoteesiks on, et varaliste API'de (Google Maps ja ArcGIS) kutsete arv tuleb suurem OGC standardite kasutamisel (avatud WMS ja WFS standardid) ja laialt levinud JSON vormingu korral, kui selliste standarditega töötamiseks loodud vaba ja avatud lähtekoodiga tarkvara (FOSS) teekidel (OpenLayers ja Leaflet).

3. Kvantitatiivseks testimiseks võrreldakse ka JavaScript'i funktsioonide sooritamise kiirust. Iga võrdluses oleva funktsiooni sooritamise aeg mõõdetakse kasutades JavaScript'i meetodit aja mõõtmiseks `performance.now`[12], mis annab tulemuse ujukoma numbrina mikrosekundeteni. Koodi eri osade tulemuslikkuse võrdlusuuringutes on funktsiooni teostamise aeg üheks näitlikumaks indeksiks. Teste sooritatakse 30 korda kolme veebilehitsejaga usaldusväärse keskmise näitaja saamiseks. Kasutatakse järgmisi veebilehitsejaid: Firefox 36.0.1, Google Chrome 40.0.2214.115 ja Internet Explorer 11 11.0.9600.17631.

   Testimise aluseks on järgmine päring:

```
1 var start = performance.now();
2 var time = performance.now() - start;
3 console.log(time);
```

[12] https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

kusjuures aega mõõdetakse vektorandmete puhul siis, kui kõik vajalikud objektid on lisatud vektorkihti, kuna edaspidise ekraanil visualiseerimise kiirus sõltub veebilehitsejast ja sellest, kui suurt osa arvuti mälust ta kasutab ning vektorobjekti sümboli tüübist ja rasterkihi puhul mõõdetakse siis, kui rasterkiht on ekraanil visualiseeritud.

Iga API juures on võetud kõige paremini sobiv sündmus (programmi poolt avastatav tegevus või toiming [33]) selleks, et mõõta vektorkihi loomise kiirust ja rasterkihi visualiseerimise kiirust. Vastavad sündmused, mida kasutati testimiseks on toodud LISAs 1.

Teostatud funktsionaalsus võrdlemiseks on järgmine:

- WMS aluskaardi tõlgendamise kiirus. Testimiseks kasutatakse Maaameti MA-ALUS aluskaart.
- WFS objektide tõlgendamise kiirus. Testimiseks kasutatakse ERMAS punktobjektide WFS kihti järgmiste objektide kogustega: 300, 3000, 30 000.
- GeoJSON vektorobjektide tõlgendamise kiirus. Testimiseks kasutatakse samu objekte, mida kasutati WFS'i puhul *json* vormingus salvestatud faili kujul.

Esimese sammuna kontrollitakse andmete vastavust normaaljaotusele Shapiro-Wilk'i testiga, kuna iga testi tulemusi oli vähem kui 50 kirjet.

H_0**:** Andmed on normaaljaotusega.

H_1**:** Andmed erinevad normaaljaotusest.

Edasiseks sammuks otsustatakse, kas esineb erinevusi rakenduse kiiruses kasutades erinevaid veebilehitsejaid arvutil. Kasutatakse Kruskal-Wallis'e testi, et otsustada, kas jaotused on statistiliselt identsed või statistiliselt oluliselt erinevad. See test sobib normaaljaotuseta (vaata eelmist hüpoteesi), sõltumatute andmete korral ning juhul kui kirjeid on vähe (vähem kui 50) ning on enam kui kaks valimit.

Hüpoteesid on sõnastatud järgmiselt:

H_0: Erinevate veebilehitsejate kasutamisel vastav objektide lisamise kiirus ei erine.

H_1: Erinevate veebilehitsejate kasutamisel, erineb ka vastav objektide lisamise kiirus.

Edasi kontrollitakse hüpoteesi, et esinevad erinevused rakendusliideste vahel:

H_0: Rakendusliideste vahel ei ole funktsioonide kiiruses erinevusi.

H_1: Rakendusliidesed erinevad funktsioonide kiiruse poolest.

Erinevuste esinemist rakendusliideste vahel kontrollitakse samuti kasutades Kruskal-Wallis'e testi.

Selleks, et teha saadud andmete põhjal järeldusi API'de kohta tervikuna kasutatakse statistilist analüüsi. Kasutatakse saadud andmete alusel leitud hinnanguid ja keskmiste alusel tehtud järeldusi. Statistilise analüüsi testid viiakse läbi programmis R (kasutades RStudio 0.98.1103). Enne sõnastatakse hüpoteesid, ehk üldkogumi kohta tehtud oletused, ning seejärel tehakse vastavad arvutused.

3.2 Kvalitatiivne hindamine

Siin proovitakse kasutada mõistet kasutatavus: ulatus, millises kindlaksmääratud kasutajate grupid võivad kasutada toodet, et saavutada määratletud eesmärgid efektiivselt, tõhusalt ja rahuloluga [5]. Kasutatavus on mitmetahuline omadus, mida on raske hinnata usaldusväärselt, kuna see hõlmab mõnevõrra subjektiivseid aspekte nagu individuaalsed harjumused, kogemused ja üksikute inimeste/programmeerijate töötamise meetodid [17].

Kvalitatiivne testimine allub kasutatavuse testimise standarditele [3]: osalejad esindavad tegelikke kasutajaid ning teevad reaalseid ülesandeid, testija märgib ja salvestab nende tulemused ja märkused ja seejärel analüüsib saadud andmeid, et leida olemasolevad probleemid eesmärgiga kasutatavust

parandada, mis antud juhul tähendab eesmärgiga aidata arendajal valida sobivam teek.

Siin hinnatakse API kasutatavust testgrupi abil, millele antakse prototüüprakenduse ja ülesannete nimekiri, mida neil tuleb valmis teha kasutades veebilehitseja konsooli. Inimesed, kes testivad, peavad esindama kasutajate grupi liikmeid ja ei oma liiga suurt kogemust antud alal. Prototüüpide alusel loodud test viidi läbi aine „LOOM.02.019 Standardid ja kvaliteet geoinformaatikas“ raames, seega on testi osalejad eesmärgi jaoks representatiivsed.

Vaatamata sellele, et kaardistamise API'd pakuvad rikast funktsionaalsust, on testimiseks tavaliselt valik ülesandeid, mis on tavapärasemad ka tegeliku API'ga töö juures [3, 5]. Antud juhul tuleb testimiseks sooritada mõned lihtsad ülesanded, mille seas on nt suumimine, kihtide sisse ja välja lülitamine, stiili muutmine. Prototüüpe kasutatakse hindamiseks kuidas iga rakendusliides mõjutab lõpliku rakenduse keerukust [5].

Test küsimused ja ülesanded on toodud LISAs 2 ja selle testi vastused on antud tabelitena LISAs 3. Iga osaleja läbib testi üksinda. Juhend algab lühikese ülevaadega kogu protsessist ning sissejuhatusega teemasse. Testi lõppedes tuleb täita küsimustik, kus on palutud hinnata ülesannete täitmise raskust iga API jaoks arvestades kõikvõimalikke vajalikke informatsiooni allikaid, nt API dokumentatsioon, õpetused, näidised ja küsimused-vastused kasutajate foorumites, kuna kõik see mõjutab kasutatavust oluliselt [10]. Vastajad annavad ka oma subjektiivse hinnangu koodi elegantsusele ja avaldavad oma rahulolu, annavad tagasiside nende poolt tajutud kasutatavusele. Selleks, et saada ka mingi kvantitatiivne näitaja mõõdeti aega, mis kuulub testi tegemiseks [17].

3.2.1 Vastuste töötlemine

Edasi hinnatakse saadud vastuste alusel kasutatavust järgmiste subjektiivsete kriteeriumite järgi:

1. Omandamise lihtsus – kasutaja on võimeline süsteemi endale selgeks tegema ilma suure pingutuseta. Seda punkti võrreldakse kasutades

abistavate materjalide ulatust. Võtmesõnadeks on dokumentatsiooni ja näidete lihtsus, *online* õpetuste rohkus ja arendajate jaoks abistavad materjalid, mis lihtsustaks API kasutamist ning foorumid, kus sarnane probleem võib olla juba lahendatud.

2. Efektiivsus – kasutaja peaks olema võimeline oma ülesanded täitma kiiresti, suurendades seeläbi produktiivsust. Peamine on arendamise kiirus. Iga API jaoks kulunud aeg mõõdetakse ja salvestatakse.

3. Rahulolu – süsteemi toimimine peaks olema kasutaja jaoks meeldiv. Siin räägitakse erinevatest aspektidest, mis teegiga töötades silma torkavad ning aitavad või segavad arusaamist ja produktiivset töötamist, mis on selgelt vastuolus kasutaja ootustega või algse intuitsiooniga [17] ning ka visuaalne lõpliku rakenduse väljanägemine (nt vaikimisi kasutajaliidese elemendid).

4. Põhilised operatsioonid

Antud peatükk annab ülevaate testimise jaoks loodud prototüüpidest ja seletab põhilise funktsionaalsuse saavutamise printsiipe iga API jaoks. Järgnevalt on käsitletud sama funktsionaalsuse tagavaid näiteid viie teegi jaoks: projektsiooni defineerimise, kaardi loomise ja kaardikihtide lisamise näited. Täismahus töötav kood WMS, WFS ja GeoJSON andmetega kaardi loomiseks iga rakendusliidesega on toodud LISAs 4.

4.1 Projektsioonid

Tavaliselt kaardistamise teegid (ja veebikaardid nagu Google, Bing, OpenStreetMap ja teised) kasutavad *Web Mercator* projektisooni (koodiga EPSG:3857 või mõnikord ka EPSG:900913) ja toetuvad WGS 84 (EPSG:4326). Nende definitsioonid on järgmised:

```
1 proj4.defs("EPSG:3857","+proj=merc +a=6378137 +b=6378137 +
    lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +
    nadgrids=@null +wktext  +no_defs");
2 proj4.defs("EPSG:4326","+proj=longlat +datum=WGS84 +no_defs");
```

Eestis on kasutusel L-EST97 kaardiprojektsioon EPSG koodiga (SRID) 3301. Kaardiprojektsiooni parameetrite ja definitsiooni saamiseks saab kasutada koordinaatsüsteemide registrit[13].

Selleks, et oleks võimalus kasutada EPSG:3301 projektsiooni OpenLayers 2, OpenLayers 3 ja Google Maps'iga tuleb kasutada JavaScript'i teeki *proj4js*[14] kas endale manuaalselt alla laadides *proj4.js* faili või kasutades CDN sisuedastusvõrku. Sarnaselt lisatakse Leaflet'i kaardile ülal nimetatud teeki spetsiaalne versioon *proj4leaflet*[15].

[13] http://epsg.io/3301
[14] http://proj4js.org/
[15] https://github.com/kartena/Proj4Leaflet

```
<script type="text/javascript" src="http://cdnjs.cloudflare.
    com/ajax/libs/proj4js/2.2.1/proj4.js"></script>
```

Edasi on vaja defineerida L-EST97 projektsioon. *Proj4js* teeki kasutades tehakse seda niinimetatud *"proj4.defs"* objekti kaudu, defineerides projektsiooni parameetrid sõnena või lisades samasse teksti veel ühe JavaScript'i failina järgmiselt:

```
    <script type="text/javascript" src="http://epsg.io/3301.js
        "></script>
```

Faili sisu on Eesti koordinaatsüsteemi definitsioon *proj4js* jaoks.

```
    proj4.defs("EPSG:3301","+proj=lcc +lat_1=59.33333333333334
        +lat_2=58 +lat_0=57.51755393055556 +lon_0=24 +x_0
        =500000 +y_0=6375000 +ellps=GRS80 +towgs84
        =0,0,0,0,0,0,0 +units=m +no_defs");
```

ArcGIS API töötab projektsioonidega teisel põhimõttel, millest räägitakse edaspidi.

4.1.1 OpenLayers 2

OpenLayers'i versioonis 2 on palju eeldefineerituid projektsioone: EPSG:4326, CRS:84, urn:ogc:def:crs:EPSG:6.6:4326, EPSG:900913, EPSG:3857, EPSG:102113 ja EPSG:102100[16]. Vaikimisi kasutatakse EPSG:3857.

Eelnevalt defineeritud projektsiooni lisamine OpenLayers'i projektsioonina toimub järgmiselt:

```
var projection = new OpenLayers.Projection("EPSG:3301")
```

[16] http://openlayers.org/two/

4.1.2 OpenLayers 3

OpenLayers 3 toetab ilma defineerimata EPSG:4326 ja EPSG:3857. Vaikimisi kasutatakse EPSG:3857. Eeldefineeritud projektsiooni esindaja tehakse OpenLayers'i teegile kättesaadavaks pannes meetodile *ol.proj.addProjection* muutujaks projektsiooni koodi:

```
1       var projection = ol.proj.get('EPSG:3301');
```

Rakendusliides visualiseerib kaardi selles projektsioonis, mis on kasutaja poolt defineeritud. Edasi defineerides *map* objekti tuleb *view* objektil defineerida projektsioon, kuna vaikimisi on see EPSG:3857:

```
1     var view = new ol.View({
2         projection: "EPSG:3301"
3     });
```

4.1.3 Leaflet

Leaflet toetab samuti ainult EPSG:4326 ja EPSG:3857 projektsioone. WMS'i võetakse vastu vaikimisi vaid *Spherical Mercator* EPSG:3857 koordinaatidega, ka GeoJSON'i andmete puhul eeldatakse näha koordinaate EPSG:3857 koordinaatide süsteemis. Selleks, et kasutada teisi koordinaate tuleb ka siin kasutada *Proj4js* võimalusi, kuid siin ei saa kasutada *proj4js* otseselt nagu eelmistes näidetes vaid, kuna Leaflet on moodulitepõhine teek, tuleb kasutada vastavat moodulit, mis võimaldab *proj4js* kasutamist - *Proj4Leaflet*.

L.Proj.CRS konstruktori abil defineeritakse projektsioon:

```
1     var projection = new L.Proj.CRS('EPSG:3301',
2                     '+proj=lcc +lat_1=59.33333333333334 +lat_2
                          =58 +lat_0=57.51755393055556 +lon_0=24 +
                          x_0=500000 ' +
3                     '+y_0=6375000 +ellps=GRS80 +towgs84
                          =0,0,0,0,0,0,0 +units=m +no_defs',
4             {
5                 resolutions: [
6                     2048, 1024, 512, 256, 128,
```

```
7                   64, 32, 16, 8, 4, 2, 1, 0.5
8               ]
9           });
```

Võrreldes eelmise variandiga, teeb Leaflet teisendust defineeritud kaardiprojektsioonist, ehk meie juhul EPSG:3301 Eesti projektsioonist, geograafilistesse koordinaatidesse. See tähendab, et Leaflet visualiseerib kaardi oma koordinaatsüsteemis [23] ja tegeleb oma koordinaatsüsteemi koordinaatidega ja meie EPSG:3301 koordinaate vaid lennult transformeerib.

4.1.4 Google Maps API

Google Maps'i kaardi projektsiooniks on EPSG:3857 *Web Mercator*. Selleks, et punkte EPSG:3301 koordinaatsüsteemis näidata Google'i kaardil tuleb objektide koordinaadid esimese sammuna transformeerida EPSG:3857 koordinaatideks. Kuna siin tuleb koordinaadid ühest koordinaatsüsteemist teisendada teisse, siis eelnevalt tuleb need mõlemad defineerida:

```
1 var proj1 = '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +
    lat_0=57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +
    ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs',
2           proj2 = '+proj=longlat +ellps=WGS84 +datum=WGS84 +
                no_defs ';
```

Vastav funktsioon tegeleb koordinaatide ümberarvutamistega:

```
1 function convert(data) {
2           for (var i = 0; i < data.features.length; i++) {
3               var convert = proj4(proj1, proj2, data.features
                    [i].geometry.coordinates);
4               data.features[i].geometry.coordinates = convert
                    ;
5           }
6           return data
7       }
```

Antud juhul kasutatakse *proj4js* funktsiooni *proj4*, mis kasutab kolme argumenti: andmete projektsiooni, kaardi projektsiooni ja koordinaatide paari.

4.1.5 ArcGIS API

ArcGIS API vaikimisi projektsiooniks on samuti EPSG:3857, muu projektsiooni kasutamiseks päritakse projektsioonide definitsioone ArcGIS veebiteenuse serverist. Selleks et defineerida kaardi projektsiooni ArcGIS API's tuleb *map* objektil defineerida koordinaatide ulatust, milles mainitud koordinaatsüsteem hakkab kehtima kaardi koordinaatsüsteemina. Koordinaatsüsteemi defineeritakse ID abil (*wkid* – *„well-known ID“*).

```
1 extent: new Extent({"xmin": 300000, "ymin": 6.3e+06, "xmax":
    800000, "ymax": 6.7e+06, "spatialReference": {"wkid":
    3301}})
```

Kui ulatus ei ole defineeritud, siis kaardi projektsiooniks on esimese kaardile lisatud kihi projektsioon.

4.2 Kaart

Map (eesti keeles „kaart“, edasi kasutan *„map“*) on iga kaardistamise teegi peamine JavaScript'i klass, mis sisaldab veebikaardi kõiki aspekte. Kaart visualiseeritakse HTML elemendis, mida saab soovi korral kujundada, seega on vaja eeldefineerida `<div>` element:

```
1   <div id="map"></div>
```

Luues kaardi rakendusliidesega viidatakse antud elemendile veebilehitseja dokumendiobjektide mudelis.

Järgnevad antud kaardi objekti konstrueerimised viie API'ga, kusjuures kaardi omadusteks on pandud minimaalne arv omadusi, mis on vajalik, et kaart töötaks õigesti Eesti projektsiooniga, kusjuures projektsioon on eelnevalt defineeritud nagu eelmises osas kirjeldatud. Kaardi defineerimisel määratakse ka selle keskpunkt. *Map*'i klassi abil määratakse ühe konkreetse kaardi veebilehel. Selle klassi uue muutuja loomiseks kasutatakse *new* operaatorit.

4.2.1 OpenLayers 2

```
var map = new OpenLayers.Map('map',
          {
              projection: projection,
              maxResolution: "auto",
              maxExtent: new OpenLayers.Bounds(300000, 6.3e
                  +06, 800000, 6.7e+06),
              center: [550000, 6520000]
          });
```

4.2.2 OpenLayers 3

```
var map = new ol.Map({
      target: document.getElementById('map'),
      view: new ol.View({
          center: [550000, 6520000],
          projection: projection
      })
 });
```

4.2.3 Leaflet

```
var map = L.map('map', {
              crs: crs,
              center: L.latLng(58.66, 25.05),
              zoom: 2
          }
 );
```

4.2.4 Google Maps API

```
var  map = new google.maps.Map(document.getElementById('map'
    ),
              {
                  center: new google.maps.LatLng(59, 24)
              });
```

4.2.5 ArcGIS API

```
1  var map = new Map("map",
2                {
3                    extent: new Extent({"xmin": 300000, "ymin":
                         6.3e+06, "xmax": 800000, "ymax": 6.7e
                        +06, "spatialReference": {"wkid":
                        3301}}), center: new esri.geometry.Point
                        (550000, 6520000, new esri.
                        SpatialReference({ wkid: 3301 }))
4                });
```

4.3 Kaardikihid

Kaardikihid on veel üks GIS valdkonnas domineeriv nähtus. Kaks kõige kasulikumat OGC standardil põhinevat teenust on WMS ja WFS spetsifikatsioonid. Mõlemad tagastavad vastavalt kasutaja päringule läbi HTTP andmevahetusprotokolli geograafilist informatsiooni.

Kihtide defineerimine käib järgmiselt: defineeritakse uus *layer* objekt ja antakse talle kõik vajalikud argumendid: kõigepealt serveri internetiaadress ning vajaliku kihi nimetus serveris või faili asukoht. Kõikvõimalikud parameetrid defineeritakse võti/väärtus paaridena.

4.3.1 WMS

Antud näidetes kasutatakse vabalt kättesaadavat Maa-ameti WMS-teenust, mis pakub väga palju erinevaid kaarte Eesti kohta. Antud juhul kasutatakse aluskaarti nimega 'MA-ALUS'.

4.3.1.1 OpenLayers 2

OpenLayers.Layer.WMS on konstruktor OGC WMS teenuste kasutamiseks OpenLayers 2 kaardil. Kohustuslikud väljad on kihile antud nimetus, WMS serveri URL ja parameetritega objekt, kus kohustuslikuks on argument kihi

nimega, mis on antud kasutatava teenuse poolt. Juhul kui ei kasutata vaikimisi projektsiooni, tuleb ka seda eraldi mainida.

```
1    var wms = new OpenLayers.Layer.WMS(
2              "Maaamet aluskaart",
3              "http://kaart.maaamet.ee/wms/alus?",
4              {
5                  "LAYERS": 'MA-ALUS'
6              },
7              {
8                  projection: projection
9              })
```

4.3.1.2 OpenLayers 3

OpenLayers 3 defineerib kihi ja selle sisu allikad eraldi. *ol.source.ImageWMS* konstruktorit kasutatakse tavalise WMS serveri puhul. Siin on kohustuslikeks parameetriteks WMS serveri URL ja parameetrite objekt kihi nimega ning serveri versiooniga.

```
1  var wms = new ol.layer.Image({
2          source: new ol.source.ImageWMS({
3              url: 'http://kaart.maaamet.ee/wms/alus?',
4              params: {
5                  LAYERS: 'MA-ALUS',
6                  VERSION: '1.1.1'
7              }
8          })
9      });
```

4.3.1.3 Leaflet

Leaflet saab *L.tileLayer.wms* konstruktori abil ka ilma lisanditeta WMS kihti näidata. Kohustuslikeks parameetriteks on WMS serveri URL ja kihi nimi ning lisaks veel *continuousWorld* parameeter, mida tuleb tõeseks muuta rasterpiltide näitamiseks.

```
var wms = L.tileLayer.wms('http://kaart.maaamet.ee/wms/alus', {
        layers: 'MA-ALUS',
        continuousWorld: true
    });
```

4.3.1.4 Google Maps API

Google Maps programmiliidesega on WMS kihi lisamine palju raskem. Esiteks luuakse objekt omadustega:

```
var wmsOptions = {
            getTileUrl: WMSGetTileUrl2,
            tileSize: new google.maps.Size(256, 256)
        };
```

google.maps.ImageMapType konstruktori abil luuakse uus kiht, mis loomise käigus kasutab *wmsOptions* parameeteid:

```
        wmsMapType = new google.maps.ImageMapType(wmsOptions);
```

Seejärel tuleb loodud kiht lisada kaardile kõige ülemise kihina:

```
        map.overlayMapTypes.insertAt(0, wmsMapType);
```

Järgnevalt defineeritakse funktsioon, mis loeb WMS pilte. Siin tuleb määrata kõik vajalikud parameetrid WMS päringu lingi jaoks (vajalikke elemente saab kontrollida Maaameti lehel). Ka siin on oluline LAYERS parameeter, kusjuures võib märgata, et see erineb eelmistest näidetest. Google'i kaardiga saab kasutada ainult geograafiliste koordinaatidega (WGS84) rasterpilte.

```
function WMSGetTileUrl2(coord, zoom) {
            var proj = map.getProjection();
            var zfactor = Math.pow(2, zoom);
            var top = proj.fromPointToLatLng(
                    new google.maps.Point(coord.x * 256 /
                       zfactor, coord.y * 256 / zfactor));
            var bot = proj.fromPointToLatLng(
                    new google.maps.Point((coord.x + 1) * 256 /
                        zfactor, (coord.y + 1) * 256 / zfactor)
                    );
```

```
8             var deltaX = 0.0013;
9             var deltaY = 0.00058;
10            var bbox = (top.lng() + deltaX) + "," + (bot.lat()
                 + deltaY) + "," + (bot.lng() + deltaX) + "," + (
                 top.lat() + deltaY);
11            var url = 'http://kaart.maaamet.ee/wms/alus-geo?';
12            url += '&REQUEST=GetMap';
13            url += '&SERVICE=WMS'; //WMS teenus
14            url += '&VERSION=1.1.1'; //WMS versioon
15            url += '&LAYERS=MA-ALUS-GEO'; //WMS kiht
16            url += '&FORMAT=image/png'; //WMS formaat
17            url += '&SRS=EPSG:4326';
18            url += '&BBOX=' + bbox;
19            url += '&WIDTH=256';
20            url += '&HEIGHT=256';
21            return url;
22        }
```

4.3.1.5 ArcGIS API

WMSLayer klassi konstruktori abil, mis võtab parameetritena WMS serveri lingi ja kihi nimetuse, lisaks veel kihi ulatuse ning määrab ka antud kihi projektsiooni, lisatakse kaardile OGS WMS kihte.

```
1  var wms = new WMSLayer('http://kaart.maaamet.ee/wms/alus?', {
2             resourceInfo: {
3                 extent: new Extent(350000, 6370000, 750000,
                      6630000, {
4                      wkid: 3301
5                 }),
6                 layerInfos: new WMSLayerInfo({
7                 })
8             },
9             visibleLayers: ['MA-ALUS']
10        });
```

4.3.2 WFS

Alternatiiv WMS kaartide kasutamisele on kliendi poolel vektorkihi joonistamine. Selleks saab kasutada WFS teenust, mis vastavalt päringule tagastab toorandmed XML, JSON ja teistes formaatides, mida API visualiseerib kaardil. Tavaliselt on API'des klassid vektorkihtide joonistamiseks veebilehitsejal, kuigi need on erinevates API'des erinevate nimede all. Üksikvektorobjekti puhul kasutatakse tihti mõistet *"marker"* nt *Layer.Markers* (OpenLayers 2) või *"feature"* nt *ol.Feature* (OpenLayers 3). Komplekssemateks vektorkihtideks kasutatakse mõisteid nagu *FeatureGroup* (Leaflet), *FeatureLayer* (Esri), *Layer.Vector* (OpenLayers 2) jne [30].

Järgmiseks on näited WFS kihi lisamisest kaardile, kusjuures on määratud minimaalne arv kohustuslikke omadusi kihi töötamiseks. Siin toodud kood pärib vektorkihi 1000 punktobjektiga Tartu Ülikooli serverist[17].

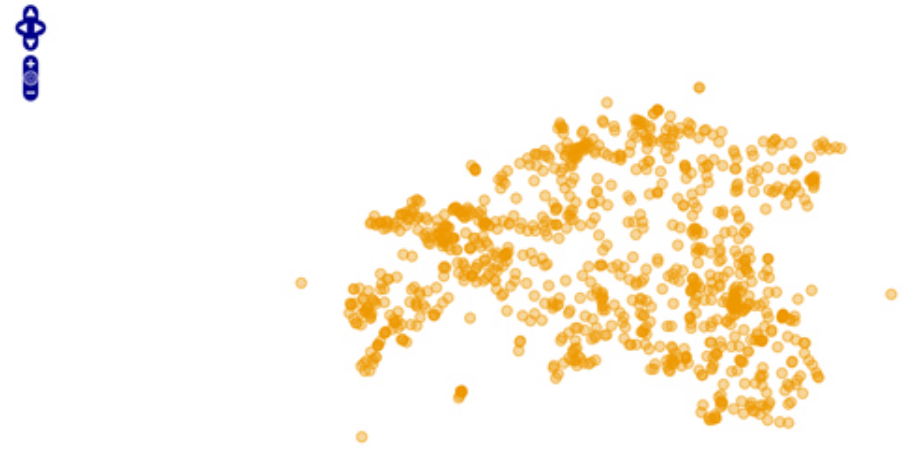
4.3.2.1 OpenLayers 2

OpenLayers.Protocol.WFS on konstruktor viie kohustusliku parameetriga, mille abil laetakse vajalikud andmed serverist (*Joonis 4*). Strateegia, ehk andmete serverist allalaadimise lähenemise, määramine on siin samuti kohustuslik; valitud strateegia küsib vaid need objektid, mis on kaardi vaatevälja sees. Parameeter *isBaseLayer* on kohustuslik juhul, kui antud kiht on ainus kaardil, vaikimisi on selle väärtus väär.

```
1   var wfs = new OpenLayers.Layer.Vector("WFS", {
2       isBaseLayer: true,
3       strategies: [new OpenLayers.Strategy.BBOX()],
4       protocol: new OpenLayers.Protocol.WFS({
5           url: 'http://loom-gis.geo.ut.ee:8080/geoserver/
                ermas/ows?service=WFS',
6           featureType: "testdata_geopnt",
7           featureNS: 'http://loom-gis.geo.ut.ee:8080/
                geoserver/ermas',
8           srsName: "EPSG:3301",
9           geometryName: "geometry",
```

[17] loom-gis.geo.ut.ee

```
            maxFeatures: 1000
        })
    });
```



Joonis 4: *OpenLayers 2 vaikimisi kujundusega 1000 punktobjektiga kaart.*

4.3.2.2 OpenLayers 3

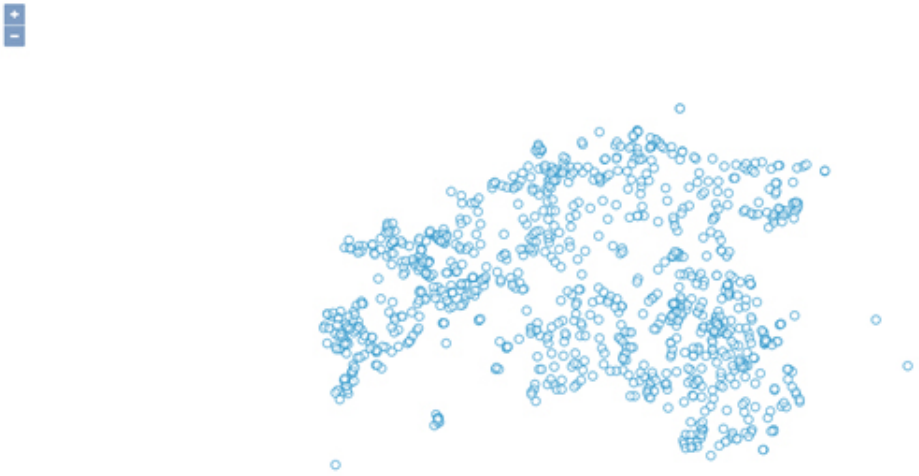
OpenLayers 3'ga andmete lugemiseks kaugserverist kasutatakse *ol.source.ServerVector* konstruktorit, mis kasutab *jQuery* AJAX päringu andmete laadimiseks.

```
var vectorSource = new ol.source.ServerVector({
        format: new ol.format.GeoJSON(),
        loader: function (extent, resolution, projection) {
            var url = 'http://loom-gis.geo.ut.ee:8080/geoserver
               /ermas/ows?service=WFS&' +
                   'version=1.1.0&request=GetFeature&typeName=
                      ermas:testdata_geopnt&' +
                   'outputFormat=text/javascript&
                      format_options=callback:loadFeatures&
                      maxFeatures=1000&' +
                   'srsname=EPSG:3301&EPSG:3301';
            $.ajax({
                url: url,
        dataType: 'jsonp'
            });
```

```
12          }
13      });
```

Vektorkihi loomise (*Joonis 5*) andmete allikaks kasutatakse serverist saadud vastuseid, mida *ol.source.ServerVector* funktsioon loeb ja transformeerib OpenLayers'i objektideks.

```
1  var wfs = new ol.layer.Vector({
2          source: vectorSource
3      });
4      var loadFeatures = function (response) {
5          vectorSource.addFeatures(wfs.readFeatures(response));
6    };
```

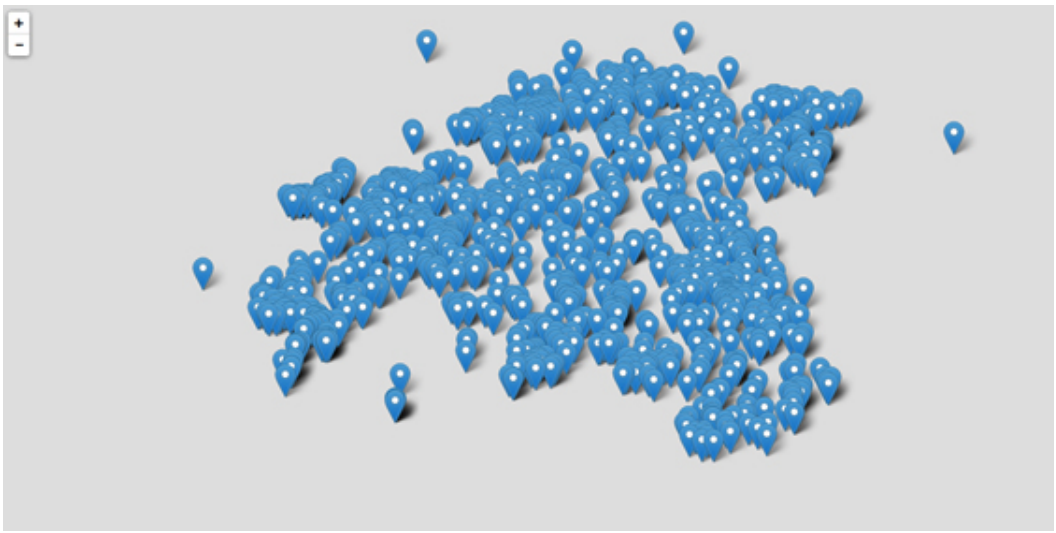


Joonis 5: *OpenLayers 3 vaikimisi kujundusega 1000 punktobjektiga kaart.*

4.3.2.3 Leaflet

Leafleti GeoJson suudab visualiseerida (*Joonis 6*) vaid EPSG:4326 projektsioonis olevaid faile, kuid WFS anmded on EPSG:3301 formaadis. Seega kui *jQuery* AJAX päringu abil andmed on kätte saadud, tuleb iga koordinaadi paar konverteerida sobivaks koordinaatsüsteemiks *proj4Leaflet* teegi abil. Konverteeritud andmed lisatakse kihti ja seejärel kiht lisatakse kaardile.

```
var geojsonLayer = new L.geoJson();
    function getJson(data) {
        for (var i = 0; i < data.features.length; i++) {
            var convert = proj4(proj1, proj2,  ata .features[i
               ].geometry.coordinates);
            data.features[i].geometry.coordinates = convert;
        }
        geojsonLayer.addData(data);
        geojsonLayer.addTo(map);
    }
    $.ajax({
        url: "http://loom-gis.geo.ut.ee:8080/geoserver/ermas/
           ows?service=WFS&version=1.0.0&request=GetFeature&
           typeName=ermas:testdata_geopnt&maxFeatures=1000&
           outputFormat=text/javascript",        dataType: '
           jsonp',
        jsonpCallback: 'parseResponse',
      success: getJson
    });
```

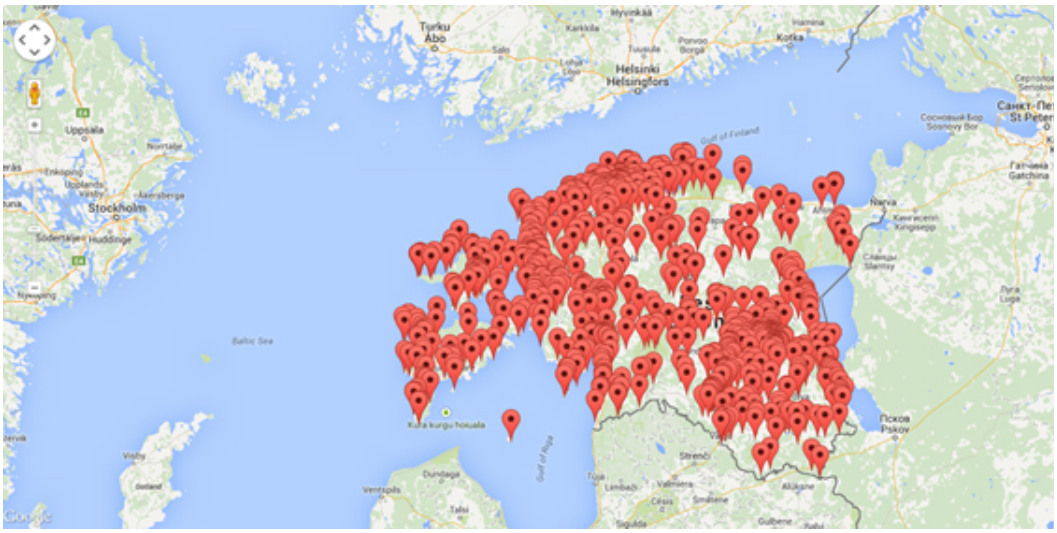


Joonis 6: *Leaflet vaikimisi kujundusega 1000 punktobjektiga kaart.*

4.3.2.4 Google Maps API

Erinevalt eelmistest rakendusliidesest Google Maps API varustab kaardi ka oma aluskaartidega, seega lisaks lisatud kihile (ka eelmises osas lisaks WMS kihile) on kaardil alati olemas veel üks kiht aluskaardiga (*Joonis 7*). Seda ei saa välja lülitada, kuid selleks, et tekiks mulje, et seda ei ole, saab seda ümber kujundada ja teha läbipaistvaks (*Joonis 8*). Seega antud korral on kaks joonist, kus esimene näitab lisatud WFS kihti vaikimisi aluskaardiga ja teine näitab WFS kihti, kui aluskaart on tehtud nähtamatuks.

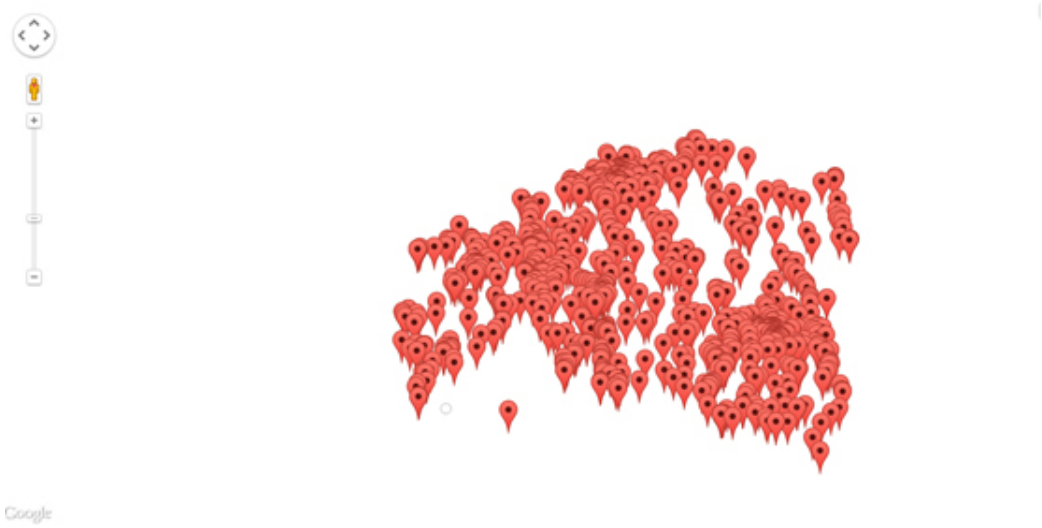
Kuna andmete koordinaate tuleb konverteerida, kasutatakse *proj4js* teeki ning konverteeritakse *jQuery* AJAX päringu vastused enne kihile lisamist.



Joonis 7: *Google Maps API vaikimisi kujundusega 1000 punktobjektiga kaart aluskaardiga.*

```
1  var wfs = new google.maps.Data();
2          wfs.setMap(map);
3          $.ajax({
4              url: "http://loom-gis.geo.ut.ee:8080/geoserver/
                   ermas/ows?service=WFS&version=1.0.0&request=
                   GetFeature&typeName=ermas:testdata_geopnt&
                   maxFeatures=1000&outputFormat=text/javascript",
5              dataType: 'jsonp',
6              jsonpCallback: 'parseResponse',
7              success: function (data) {
```

```
8                 for (var i = 0; i < data.features.length; i++)
                      {
9                      var convert = proj4(proj1, proj2, data.
                           features[i].geometry.coordinates);
10                     data.features[i].geometry.coordinates =
                           convert;
11                }
12                wfs.addGeoJson(data);
13            }
14        });
```



Joonis 8: *Google Maps API vaikimisi kujundusega 1000 punktobjektiga kaart peidetud aluskaardiga.*

4.3.2.5 ArcGIS API

ArcGIS API on varaline teek ja on rohkem suunatud oma varaliste andmete vormingute kasutamisele. Näiteks ArcGIS serveril on oma viis vektorandmete hoidmiseks nimetusega *"feature service"*, mille funktsioonid on aga WFS teenusega sarnased. ArcGIS omase vorminguga andmete lisamine serverist (ArcGis Online või ArcGis for Server) on lihtne ja kiire, kuid OGC WFS kasutamiseks tuleb rohkem koodi kirjutada. Esiteks tuleb defineerida *FeatureLayer* kihi parameetrid, mis on vajalikud ka kaardi projektsiooni määramiseks, kui see on ainus kiht kaardil.

```
var layerDefinition = {
            "extent": {
                "xmin": 350000,
                "ymin": 6370000,
                "xmax": 750000,
                "ymax": 6630000,
                "spatialReference": {
                    "wkid": 3301
                }
            },
            "fields": [
                {
                    "name": "OBJECTID",
                    "type": "esriFieldTypeOID"
                }
            ]
        };
        var featureCollection = {
            layerDefinition: layerDefinition
        };
        var wfs = new FeatureLayer(featureCollection);

        map.addLayers([wfs]);
```

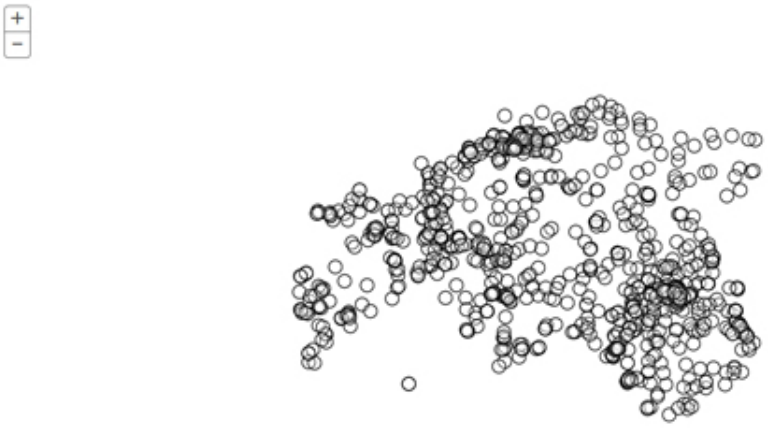
Serveri WFS teenuse link on antud vajalike parameetriga, mille abil edasi luuakse iga objekti kohta ArcGIS kaardi *Graphic* objekt (*Joonis 9*).

```
var layerUrl = 'http://loom-gis.geo.ut.ee:8080/geoserver/ermas/
    ows?service=WFS&' +
                'version=1.1.0&request=GetFeature&typeName=
                    ermas:testdata_geopnt&' +
                'outputFormat=text/javascript&format_options=
                    callback:loadFeatures&maxFeatures=1000&' +
                'srsname=EPSG:3301&EPSG:3301';
        var loadFeatures = function (response) {
            for (var i = 0; i < response.features.length; i++)
                {
                var geometry = new esri.geometry.Point(response
                    .features[i].geometry.coordinates[0],
```

```
                    response.features[i].geometry.coordinates
                    [1], new esri.SpatialReference({ wkid: 3301
                    }));
8                   var symbol = new esri.symbol.SimpleMarkerSymbol
                    ();
9                   var graphic = new esri.Graphic(geometry, symbol
                    );
10                  wfs.add(graphic)
11              }
12          };
13          window.loadFeatures = loadFeatures;
```

Päring ise tehakse *request* funktsiooni abil.

```
1   var layersRequest = request({
2               url: layerUrl,
3               callbackParamName: "loadFeatures"
4           });
```



Joonis 9: *ArcGIS API vaikimisi kujundusega 1000 punktobjektiga kaart.*

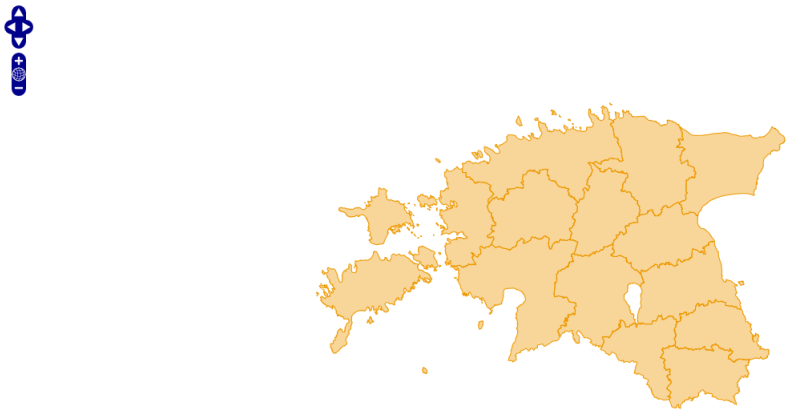
4.3.3 GeoJSON

Viimaseks on näited vektorkihi lisamisest failist. Vektorkiht võeti Maaameti kodulehelt *.shp* vormingus, lihtsustati kiiremaks kuvamiseks ekraanil ja salvestati GeoJSON vormingus. Seejärel lisati GeoJSON JavaScript'i faili objektina nimega *featurecollection*.

4.3.3.1 OpenLayers 2

Kasutades OpenLayers 2 rakendusliidest kaardi andmete visualiseerimiseks (*Joonis 10*) luuakse uus vektorkiht nimega „maakond", mis võib mitmetest erinevatest allikatest andmeid võtta. Antud juhul kasutatakse GeoJSON vormingus andmed, mis loetakse konstruktor *OpenLayers.Format.GeoJSON* abil, mis on vastavas vormingus andmete sõel. Vektorkihile on kohustuslik anda nimi.

```
1 var geojson_format = new OpenLayers.Format.GeoJSON();
2     var maakond = new OpenLayers.Layer.Vector('maakonnad');
3     maakond.addFeatures(geojson_format.read(featurecollection))
          ;
```

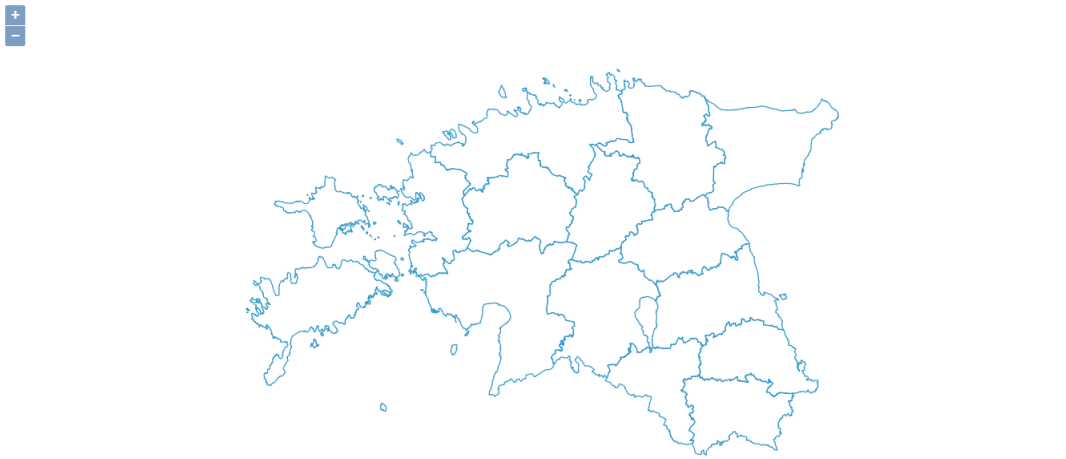


Joonis 10: *OpenLayers 2 vaikimisi kujundusega maakondade polügoonide kiht.*

4.3.3.2 OpenLayers 3

Vektorkihte (*Joonis 11*) esindab OpenLayers 3 rakendusliideses *ol.layer.Vector* konstruktor, millele tuleb ainult õige andmeallikas anda.

```
var maakond = new ol.layer.Vector({
        title: 'maakonnad',
        source: new ol.source.GeoJSON({
            object: featurecollection
        })
    });
```

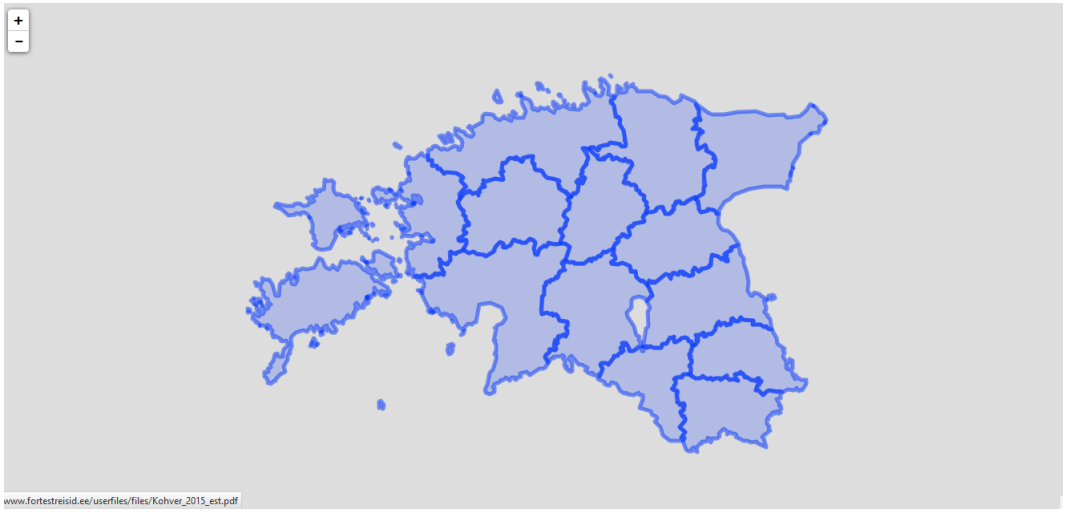


Joonis 11: *OpenLayers 3 vaikimisi kujundusega maakondade polügoonide kiht.*

4.3.3.3 Leaflet

Kui eelnevalt on defineeritud andmete projektsioon kasutatakse *L.Proj.GeoJSON* funktsiooni *proj4Leaflet*'i teegist, et lennult koordinaate transformeerida Leaflet rakendusliidesega. Funktsioon *pointToLayer* loob kihi vastavatest objektidest (*Joonis 12*).

```
var maakond = L.Proj.geoJson(featurecollection, {
        'pointToLayer': function (feature, latlng) {
            return L.marker(latlng);
        }
    });
```



Joonis 12: *Leaflet'i vaikimisi kujundusega maakondade polügoonide kiht.*

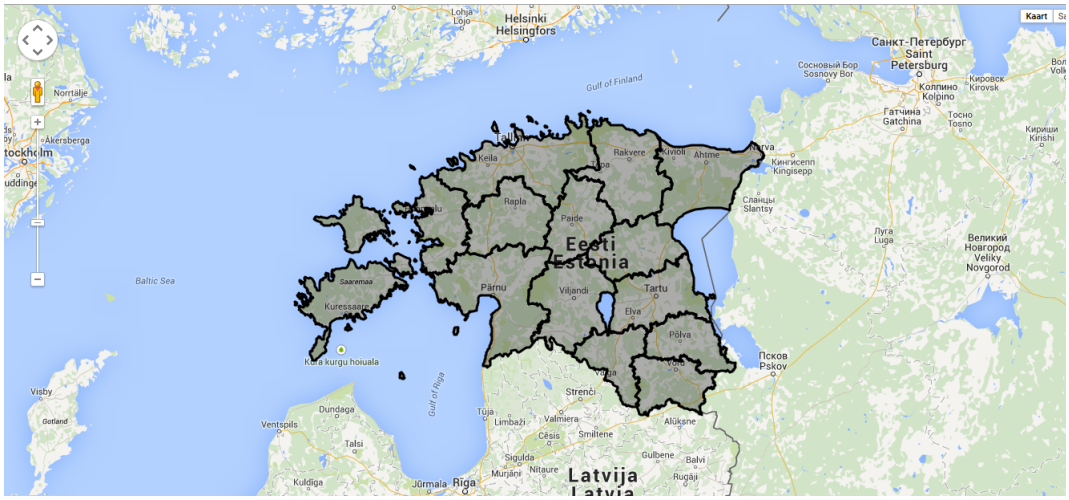
4.3.3.4 Google Maps API

Kuna Google Maps kaart on Merkaatori projektsioonis, siis selleks, et näidata polügoone Lambert'i konformsel koonilisel projektsioonil tuleb kasutada *proj4* teeki. Konverteerimise funktsioon on sarnane eelmises osas WFS punktide konverteerimise funktsiooniga, kuid siin arvestatakse sellega, et objektid on polügoonid ja multi-polügoonid.

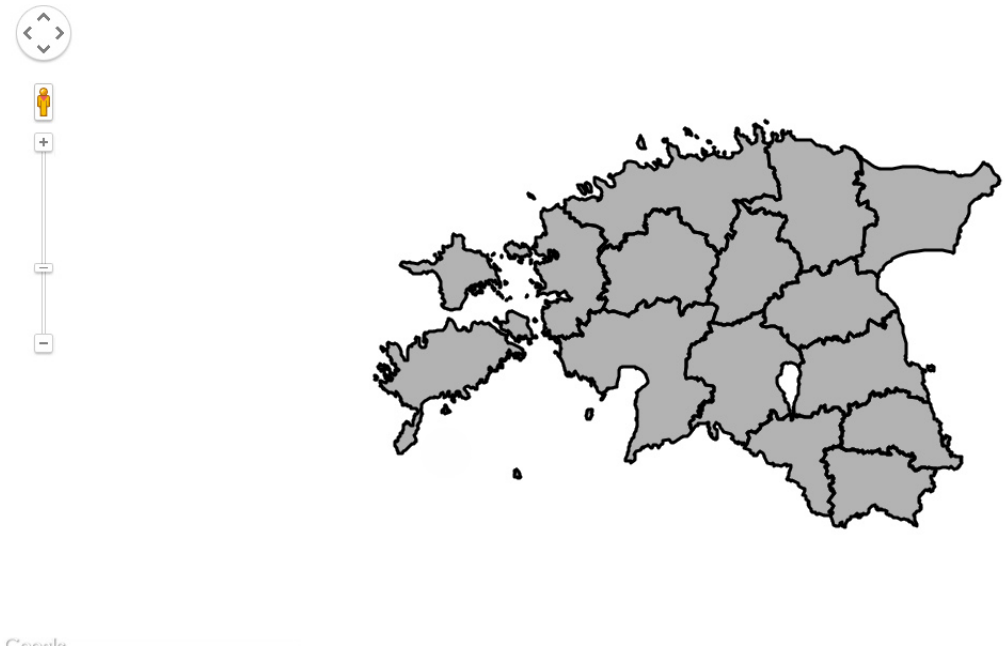
Luuakse *google.maps.Data* kiht (*Joonis 13*, *Joonis 14*) ning lisatakse sellele andmed meetodiga *addGeoJson()*.

```
1  var maakond = new google.maps.Data({map: null});
2          function convert(data) {
3              for (var i = 0; i < data.features.length; i++) {
4                  var polygons = data.features[i].geometry.
                       coordinates;
5                  if (polygons.length == 1) {
6                      var polygon111=[];
7                      for (var j = 0; j < polygons[0].length; j
                           ++) {
8                          var convert = proj4(proj1, proj2,
                               polygons[0][j]);
9                          polygons[0][j] = convert;//
10                     }
11                 } else {
12                     for (var j = 0; j < polygons.length; j++) {
```

```
13                          for (var m = 0; m < polygons[j].length;
                                m++) {
14                              for (var k = 0; k < polygons[j][m].
                                    length; k++) {
15                                  var convert = proj4(proj1,
                                        proj2, polygons[j][m][k]);
16                                  polygons[j][m][k] = convert;
17                              }
18                          }
19                      }
20                  }
21              }
22              return data;
23          }
24          maakond.addGeoJson(convert(featurecollection));
```



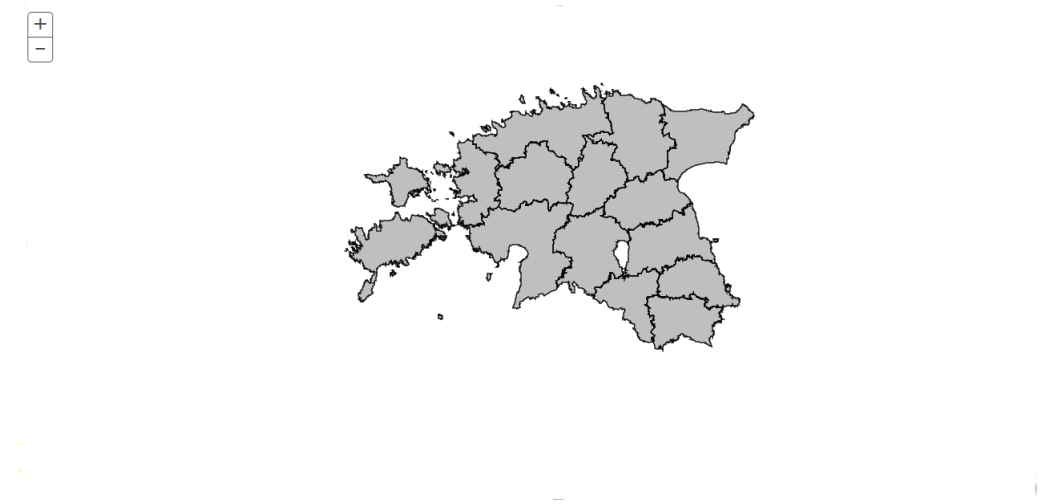
Joonis 13: *Google Maps API vaikimisi kujundusega maakondade polügoonide kiht aluskaardiga.*



Joonis 14: *Google Maps API vaikimisi kujundusega maakondade polügoonide kiht peidetud aluskaardiga.*

4.3.3.5 ArcGIS API

Sisseehitatud sõela GeoJSON vormingu jaoks, mis suudaks ka projektsiooni arvesse võtta, ESRI ArcGIS JS API ei sisalda (kuid ESRI on loonud oma varalise JSON-põhise geomeetria formaadi). Sellepärast kasutatakse isekirjutatud funktsiooni, mis vastavalt sellele kas objekt on polügon või multi-polügon, loob sellest ArcGIS rakendusliidesele omase geomeetria ning seejärel lisab vastav geomeetria kihile (*Joonis 15*).



Joonis 15: *ArcGIS API vaikimisi kujundusega maakondade polügoonide kiht.*

```
1  var maakond = new esri.layers.GraphicsLayer();
2          for (var i = 0; i < featurecollection.features.length;
               i++) {
3              var polygons = featurecollection.features[i].
                   geometry.coordinates;
4              if (polygons.length == 1) {
5                  var attr = {"id": featurecollection.features[i
                       ].id, "MKOOD": featurecollection.features[i
                       ].properties.MKOOD, "MNIMI":
                       featurecollection.features[i].properties.
                       MNIMI};
6                  var geometry = new esri.geometry.Polygon(
                       polygons, new esri.SpatialReference({ wkid:
                       3301 }));
7                  var symbol = new esri.symbol.SimpleFillSymbol()
                       ;
8                  var graphic = new esri.Graphic(geometry, symbol
                       , attr);
9                  maakond.add(graphic)
10             } else {
11                 for (var m = 0; m < polygons.length; m++) {
12                     var attr = {"id": featurecollection.
                           features[i].id, "MKOOD":
```

```
                    featurecollection.features[i].properties
                    .MKOOD, "MNIMI": featurecollection.
                    features[i].properties.MNIMI};
13                  var geometry = new esri.geometry.Polygon(
                    polygons[m], new esri.SpatialReference({
                     wkid: 3301 }));
14                  var symbol = new esri.symbol.
                    SimpleFillSymbol();
15                  var graphic = new esri.Graphic(geometry,
                    symbol, attr);
16                  maakond.add(graphic)
17              }
18          } }
```

5. Tulemused

Antud peatükk annab ülevaate testimise käigus saadud tulemustest.

5.1 Kvantitatiivse hindamise tulemused

5.1.1 API suurus

Antud tabel (*Tabel 1*) iseloomustab vajalikke resursse (kilobaitides (kB)), mis API päringuid tehes laetakse läbi sisuedastusvõrgu veebilehe avamisel. Iga rakendusliidese JavaScript'i lähtekoodiga tuleb kaasa ka CSS kaskaadlaadistik ning laetakse alla mõned vajalikud kasutajaliidese pildid *jpg/png* vormingus. Kuna tegemist on Eesti kohta kaartide tegemise võrdlusega, võetakse arvesse ka kohustuslik OpenLayers 2, OpenLayers 3 ja Google Maps API jaoks *proj4.js* teek ning Leaflet jaoks *proj4leaflet.js*. Omavahel võrreldakse vajalike ressursside summaarset mahtu.

	JavaScript	**Css**	**Pildid**	**proj4**	**Kokku**
OpenLayers 2.11	756,0	8,5	3,0	71,6	839,1
OpenLayers 3.2.1	449,3	3,6	1,1	71,6	525,6
Leaflet 0.7.3	122,0	9,9	2,5	79,8	214,2
Google Maps JavaScript API v3 3.018	455,8	9,0	476,7	71,6	1013,1
ArcGIS API for JavaScript 3.12	1278,2	229,0	6,0	0,0	1513,2

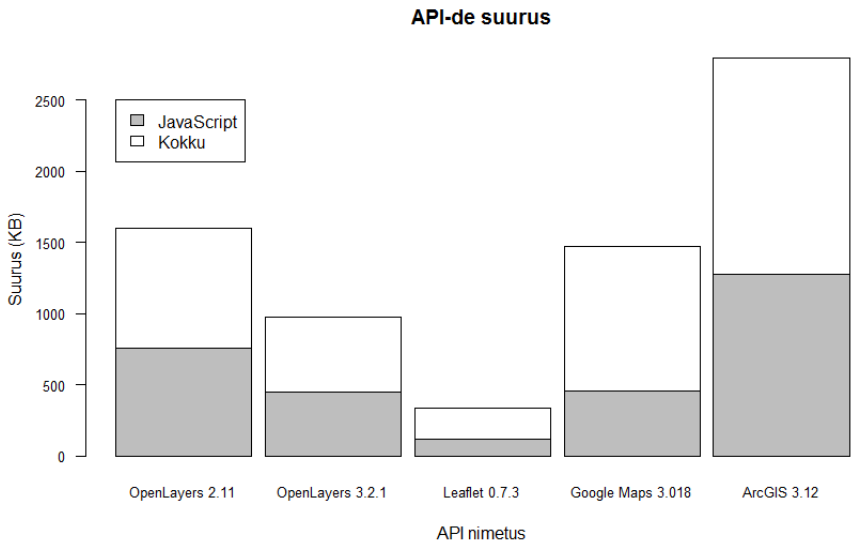
Tabel 1: *Kaardistamise rakendusega vajalik ressursside maht (kB)*

Nagu tabelist nähtub, on suurim API ArcGIS teek, milles nii JavaScript'i lähtekood kui ka rakendusliidese jaoks vajalik CSS fail on selles nimekirjas kõige mahukamad, moodustades kokku 1513,2 kilobaiti. Vaatamata sellele, et erinevalt teistest teekidest pole siin vaja kasutada projektsioonide teisendamise teeki *proj4*, erineb ArcGIS API summaarne suurus oluliselt ülejäänud teekidest ning on 7 korda mahukam väikseimast ja kõige kergekaalulisemast teegist –

Leaflet raamistikust. Nagu selle moto ja peamine eesmärk ütleb, Leaflet jääb kõige kompaktsemaks teegiks kaaludes ainult 214,2 kB ja olles rohkem kui 2 korda väiksem suuruse poolest teisest teegist (OpenLayers 3).

Keskmisest (638,640 kB) suuremad on veel OpenLayers versioon 2 kaaludes 839,1 kB ja Google Maps API 1013,1 kB, keskmisest väikem lisaks Leaflet teegile veel OpenLayers versioon 3 (526,6 kB).

Nagu *Joonis 16* näitab, on API suuruse võrdlemisel oluline eristada, kas võrreldakse ainult JavaScript'i faili või summaarselt kõiki vajalikke allalaetavaid faile. Võrreldes vaid JavaScript'i faili on Google Maps suurusega 455,8 kB 2 korda väiksem tema summaarsest suurusest, mis on tingitud allalaetavate pildifailide suurusest, kuna Google Maps kaardi initsialiseerimisel on nõutud ka Google'i aluskaardi kuvamine. Võrreldes ainult JavaScript'i faile muutub Google Maps API keskmisest väiksemaks (keskmiseks suuruseks on antud juhul 612,3 kB) ning on peaaegu sarnane OpenLayers 3 Javascript faili kaaluga.



Joonis 16: *API'de suurus (JavaScript ja kokku).*

Oluline on mainida, et mõne API puhul toodud JavaScript'i suurus ei ole maksimaalne. OpenLayers ja ArcGIS teegid sisalduvad kohe ühes failis. Aga Google Maps'i teek võib sõltuvalt olukorrast olla suurem, sest vajadusel võib lisada läbi URL'i parameetri kuni 7 täiendavat moodulit[18], mis tõstavad

[18] https://developers.google.com/maps/documentation/javascript/

funktsionaalsust. Samamoodi on ka Leaflet mooduliteõhine ja tema suurus võib oluliselt tõusta.

Tabeli ja joonise alusel püstitatud hüpotees, et erinevate rakendusliideste kasutamisel allalaetavate failide maht ei erine oluliselt üksteisest on tagasi lükatud ja vastu on võetud väide, et allalaetav ressursside kogus erineb ning kuna raskemate failide allalaadimine toimub pikemat aega ja see mõjutab rakenduse veebilehe kiirust, siis võib API suurus olla otsustavaks teguriks rakendusliidese valimisel.

5.1.2 API kasutatavuse indeks

Järgnevalt võrreldi tüüpiliste ülesannete lahendamiseks vajalikku arvu API kutseid. Antud parameetri väärtused on olulised, kuna väiksem väärtus viitab sellele, et kirjeldatud oli vähem erinevaid parameetreid ja kasutatud vähem erinevaid funktsioone ehk kirjutati vähem JavaScript koodi. Seega vajalik arv API kutseid korreleerub rakenduse arendamise kiirusega.

Funktsioonid, milles päringuid võrreldi on WMS kihi lisamine, WFS kihi lisamine ja vektorkihi lisamine GeoJSON failist. Arvesse võeti vajalikud konstruktorid ja funktsioonid ning kohustuslikud parameetrid ning lisaks need päringud teiste teekide poole, mis antud rakendusliidesega töötades on vajalikud (päringud *proj4* ja *jQuery* teekidele).

Järgnevalt on esitatud kolm tabelit, iga tabel kajastab erinevusi rakendusliideste poole kutsete arvus ühe konkreetse tegevuse teostamisel: WMS kihi lisamisel (*Tabel 2*), vektorkihi lisamisel failist (*Tabel 3*) ja WFS kihi lisamisel (*Tabel 4*).

WMS kutsed	**Konstruktor**	**Funktsioon**	**Parameeter**	**Kokku**
OpenLayers 2.11	4	3	10	17
OpenLayers 3.2.1	4	4	10	18
Leaflet 0.7.3	3	2	9	14
Google Maps JavaScript API v3 3.018	6	8	8	22
ArcGIS API for JavaScript 3.12	6	2	9	17

Tabel 2: *WMS kihi lisamiseks vajalik API kutsete arv.*

WMS kihi loomise käigus toimub kõige rohkem kutseid API poole kasutades Google Maps API't. Keskmine kutsete arv on 17,6 ning OpenLayers 2, OpenLayers 3 ja ArcGIS sellest oluliselt ei erine, kuid Google Maps 22 kutsega on selgelt viimasel kohal ning Leaflet 14 kutsega on antud juhul kõige ilmekam.

GeoJSON kutsed	**Konstruktor**	**Funktsioon**	**Parameeter**	**proj4**	**Kokku**
OpenLayers 2.11	5	5	8	0	18
OpenLayers 3.2.1	4	4	7	0	15
Leaflet 0.7.3	4	2	7	0	13
Google Maps JavaScript API v3 3.018	3	4	5	1	13
ArcGIS API for JavaScript 3.12	8	3	15	0	26

Tabel 3: *GeoJSON kihi lisamiseks vajalik API kutsete arv.*

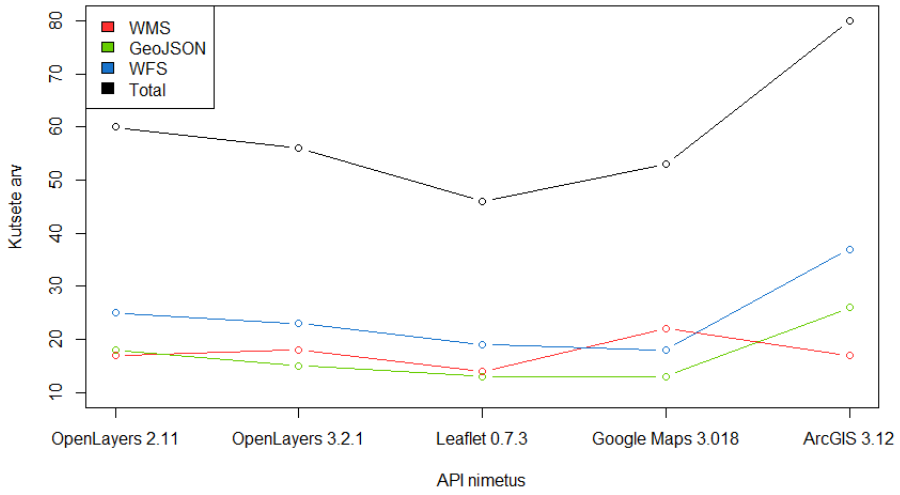
GeoJSON kihi lisamisel mõjutab keskmist selgelt ArcGIS'i rakendusliides, mis lahendab ülesannet minimaalselt 26 kutsega (keskmine on 17). Ülejaanutest nõuab vektorkihi failist moodustamiseks OpenLayers 2 täpsustamiseks rohkem parameetreid, kui ülejäänud kolm API't (OpenLayers 3, Leaflet ja Google Maps). Ka antud juhul nõuab varaline teek ArcGIS kõige rohkem API kutseid.

WFS kutsed	**Konstruktor**	**Funktsioon**	**Parameeter**	**jQuery**	**proj4**	**Kokku**
OpenLayers 2.11	6	3	16	0	0	25
OpenLayers 3.2.1	5	7	8	3	0	23
Leaflet 0.7.3	3	3	7	5	1	19
Google Maps JavaScript API v3 3.018	3	4	5	5	1	18
ArcGIS API for JavaScript 3.12	9	4	24	0	0	37

Tabel 4: *WFS kihi lisamiseks vajalik API kutsete arv.*

Kui vektorkiht listakse WFS serverist, siis on nõutud kutseid rohkem, kui kihi failist lisamise käigus, kuna tuleb moodustada korrektne päring andmete serverist allalaadimiseks. OpenLayers 3'e, Leaflet'i ja Google Maps'i puhul lisandub veel *jQuery* teegi kasutamine, mille abil teostatakse asünkroonne AJAX päring. Siin on kokku keskmiselt vaja 24 kutset. Ka siin mõjutab keskmist väärtust ArcGIS API, mis on kõige vähem ilmekas rakendusliides, kokku 37 kutsega (ja seda vaatamata sellele, et *jQuery* ja *proj4* teeke siin ei kasutata), millest suurem osa on parameetrite kutsed.

Vaatamata sellele, et väga suuri erinevusi ei esine, eristub igas testis teistest selgelt üks varalistest rakendusliidestest, kas Google Maps (WMS puhul) või ArcGIS API (WFS ja GeoJSON puhul). Seega võib püstitatud hüpoteesi õigeks pidada. Samuti võib väike erinevus rakendusliideste API kutsete arvude vahel muutuda märgatavaks lõpprakenduses, kus on vaja kasutada palju erinevaid funktsioone korraga. Näiteks ArcGIS API puhul võib kutsete arv oluliselt tõusta (*Joonis 17*), vähendades seeläbi arendamise kiirust.



Joonis 17: *API kutsete arv funktsiooniti ja summaarselt.*

5.1.3 Kihi objektide visualiseerimise kiirus

Iga rakendusliidese puhul mõõdeti kaardile kihi lisamise kiirus. *Tabel 5* näitab testi käigus saadud keskmisi väärtuseid. Detailne tabel, kus iga rakendusliidese jaoks on seitset testi (WMS, WFS 300, WFS 3000, WFS 30 000, GeoJSON 300, GeoJSON 3000 ja GeoJSON 30 000, kus number tähendab objektide arvu kaardil) testitud 30 korda kolme veebilehitsejaga (Firefox, Chrome ja Internet Explorer), on toodud LISAs 5. Statistiliste erinevuste leidmiseks kasutati täielikku tabelit.

	WMS	GeoJSON 300	GeoJSON 3000	GeoJSON 30000	WFS 300	WFS 3000	WFS 30000
OpenLayers 2.11	338,375	39,190	168,813	1322,697	390,443	2069,344	15632,558
OpenLayers 3.2.1	1240,364	134,846	1434,763	9115,325	327,629	1437,907	16004,640
Leaflet 0.7.3	537,863	152,665	1463,771	52892,237	315,665	2505,789	53150,445
Google Maps JavaScript API v3 3.018	5725,678	65,213	457,851	3458,461	234,962	1373,491	11934,262
ArcGIS API for JavaScript 3.12	660,491	367,727	1191,452	15995,676	475,277	1918,963	21213,501

Tabel 5: *Vektorobjektide visualiseerimise keskmine kiirus (millisekundites).*

Normaaljaotusele vastavuse hüpoteesi kontrolliti 105 korda (täieliku tabeli LISAs 5 iga veeruga). Kuna igal korral osutus $p<0,05$, tuleb andmed Shapiro-Wilk testide tulemusel lugeda normaaljaotusest erinevaks (olulisuse nivoo 0,05 ehk 5%). Hüpotees H_0 on tagasi lükatud ja vastu võetud on hüpotees H_1, et antud andmed erinevad statistiliselt oluliselt normaaljaotusest kõikide testide puhul.

Järgmist hüpoteesi (kas esineb erinevusi rakenduse kiiruses kasutades erinevaid veebilehitsejaid arvutil) kontrolliti iga rakendusliidesega iga testi puhul, st 7 korda iga rakendusliidesega, ehk kokku 35 korda. Hüpoteesi kontrolliti Kruskall-Wallis'e testiga. Iga testi puhul on saadud väärtus $p<0,05$, seega H_0 hüpotees on tagasi lükatud. Vastuvõetud on hüpotees H_1 ehk tulemuseks on statistiliselt oluline erinevus eri veebilehitsejate kasutamisel kõikide testide puhul.

Selgus, et WMS kihi lisamisel ei ole selget tendentsi, milline veebilehitseja näitab kõikide rakendusliideste puhul rasterkihti kiiremini, kuid iga API puhul on sisemine erinevus veebilehitsejate vahel selgelt eristatav. OpenLayers 2 raamistik näitab rasterkihti 1,5 korda kiiremini Chrome veebilehitsejas, kui Firefox veebilehitsejas. Internet Explorer on kiireim OpenLayers 3 puhul (1,3 korda kiirem kui Chrome ja 1,2 korda kiirem kui Firefox), Leaflet'i puhul (1,7 korda kiirem kui Firefox ja 1,2 korda kiirem kui Chrome) ning ArcGIS API puhul (1,2 korda kiirem kui Firefox ja 1,1 korda kiirem kui Chrome). Google

Maps'i kaart töötab kiiremini Firefox'i veebilehitsejas olles seal 1,2 korda kiirem kui Internet Explorer'i veebilehitsejas.

Testimise käigus on selgunud, et vektorobjektide kaardile lisamise puhul on Chrome kiiruse poolest selgelt esikohal. Vektorobjektide lisamise kiiruse võrdlemisel võeti arvesse järgmised testid: GeoJSON ning WFS 300, 3000 ja 30000 objektiga, ehk kokku kuus testi iga rakendusliidese jaoks, millest võeti keskmised väärtused. OpenLayers 3 ja Leaflet teekide puhul on ülalnimetatud veebilehitsejas funktsiooni kiirus märgatavalt suurem. OpenLayers 3 lisab objektid kihile Chrome veebilehitsejas 1,1 kuni 6,1 korda kiiremini, kui teistes veebilehitsejates (keskmiselt 2,4 korda kiiremini). Keskmiselt 2 korda kiiremini töötab vektorkiht Chrome's Leaflet'i rakendusliidese puhul, maksimaalne erinevus ulatub kuni 3,9 kordseni. Kiire lisamine toimub ka ArcGIS'i (keskmiselt 1,7 korda maksimumiga 2,8) ja OpenLayers 2 (keskmiselt 1,5 korda maksimumiga 2,4) puhul. Erinevused on märgatavamad testides rohkema arvu vektorobjektidega kaardil. Google API puhul selget tendentsi ei ole: neli korda kuuest testist osutus Chrome teistest veebilehitsejatest kiiremaks (keskmiselt 1,4 korda kiirem), kuid ühe testi puhul oli kiireim Firefox ja teise testi puhul Internet Explorer. Internet Explorer osutus keskmiselt aeglasemaks kõikide testide puhul: kokku 25-l korral 35-st võimalikust osutus Internet Explorer aeglaseimaks.

Selleks, et selgitada, kas rakendusliideste vahel on erinevusi või mitte, kasutati ainult Google Chrome veebilehitseja andmeid, kuna see on populaarseim veebilehitseja Eestis (seisuga juuli 2014[19]) ning kuna ta osutus eelmises testis paljudel juhtudel kiireimaks.

Hüpoteesi kontrollimiseks koostati vastavad tabelid WMS'i, WFS'i (3 tk) ja GeoJSON'i (3 tk) jaoks. LISAs 6 on toodud vastav tabel näitena WMS funktsiooni kiiruse võrdlemiseks API'de vahel kasutades Chrome andmeid (30 sõltumatut testi tulemust iga rakendusliidesega) ning *Tabel 6* näitab selle sama tabeli kokkuvõtet. Sellised tabelid koostati iga funktsiooni jaoks.

[19] http://en.wikipedia.org/wiki/Usage_share_of_web_browsers

WMS	**Miinimum**	**1 kvantiil**	**Mediaan**	**Keskmine**	**3 kvantiil**	**Maksimum**
OpenLayers 2	235,0	255,2	266,5	282,6	294,5	464,0
OpenLayers 3	1007,0	1066,0	1126,0	1371,0	1709,0	2255,0
Leaflet	426,0	427,5	482,5	502,8	522,5	704,0
Google Maps	5036,0	5439,0	5594,0	5893,0	5884,0	8336,0
ArcGIS	1899,0	1925,0	1956,0	1992,0	2012,0	2436,0

Tabel 6: *WMS funktsiooni kokkuvõttev tabel.*

WMS'i korral lükkab Kruskal-Wallis'e test H_0 hüpoteesi tagasi, kuna p<0,05 ning seega on vastu võetud väide, et rakendusliidesed erinevad antud funktsioonide kiiruse poolest. Kruskall-Wallis'e test näitab selgeid erinevusi erinevate API'de kasutamisel WMS'i funktsiooni kiiruste vahel ning kõikvõimalike API paaride vahel. WMS kihi lisamisega töötab kõige kiiremini OpenLayers 2 ning kõige aeglasemalt Google Maps (mis on oodatav käitumine vaadates eelnevalt vastavat koodi lõiku osas 4.3.1.4.) ning nende vahel on erinevus 20,84 korda.

Lisaks *Tabel 6* andmetele on LISAs 7 toodud sarnased kokkuvõtvad tabelid ka vektorobjektide lisamise kiirusest. Kuuest vektorobjektide kuvamise kiirust puutuvast testist kolm näitasid sarnasust OpenLayers 3 ja Google Maps API kiiruste vahel (GeoJSON 3000 testi tulemuseks väärtus p=0,8941, WFS 300 p=0,6361 ja WFS 30000 p=0,06899). Ühel korral näitasid ka OpenLayers 2 ja Leaflet'i mõõdetud kiirused sarnasust p väärtusega umbes 0,7. Kõik ülejäänud API'de paarid ja grupid näitasid selgeid erinevusi p väärtusega <0,05, mis tähendab et nullhüpotees on tagasi lükatud ning vastu on võetud väide, et rakendusliidesed erinevad funktsioonide teostamise kiiruse poolest.

Kui vaadata, kui kiiresti toimub kaardile andmete lisamine failist, siis kõigil kolmel korral on esikohal OpenLayers 2. Selle erinevus aeglaseima rakendusliidesega ArcGIS API ulatub üle 10 korra 300 objekti puhul ja üle 13 korra 30000 objekti puhul (erinevus on 10 sekundit), ehk erinevus suureneb veelgi objektide arvu suurenemisel. Ka Leaflet osutub aeglaseks ning ka tema erinevus kiirema OpenLayers 2 tulemustest suureneb objektide arvu suurenemisel: Leaflet on sellest 4, 7 ja 9 korda aeglasem vastavalt 300, 3000 ja 3000 objektiga kaardil. OpenLayers 2 kiiruse erinevus ülejäänud teekidest

ei ole nii suur, aga ikka ulatub kuni 5 kordseks (OpenLayers 3 30000 objekti puhul) ning see on väga märgatav suure objektide arvu puhul.

Kui võrrelda WFS funktsiooni kiirust, siis on tulemus eelmisest täiesti erinev. Siin on suure objektide arvu puhul esikohal Google Maps. Erinevus ulatub kuni 2 korrani, mis moodustab 11 sekundit erinevust kiiruses. 300 objektiga ei ole erinevus inimese jaoks silmaga nähtav, kuna maksimaalselt kestab vaid 174 millisekundit. Siin ei erine OpenLayers 3, Leaflet'i ja Google Maps'i kiirus palju. ArcGIS ja OpenLayers 2 on nendest aeglasemad. WFS funktsiooni puhul on API'de omavaheline erinevus protsentuaalselt väiksem, kui failist vektorkihi lisamise funktsiooni puhul, kuid võttes arvesse palju pikemad ajad WFS puhul on erinevus API'de vahel sekundites mõõdetuna natuke suurem.

Võib näha, et juhul kui andmed laetakse failist on laadimise aeg kiirem. Samal ajal selle meetodi miinuseks võrreldes WFS serveri kasutamisega on see, et tuleb oma andmed koodis uuendada. Kasutades WFS serverit näitab rakendus juhul, kui serveris on andmed uuendatud, kohe neid ka uuendatud kujul.

5.2 Kvalitatiivse hindamise tulemused

Kvalitatiivse testi osalejateks oli 8 geoinformaatika tudengit, kellest 7 on eelnevalt läbinud ülikoolis ühe programmeerimise kursuse, kus oli käsitletud *Python* programmeerimiskeelt. Vaatamata sellele on leitud õigete vastuste arv 25-st võimalikust kokku keskmiselt 4,5%. Ükski inimene ei ole varem kokku puutunud JavaScript'i keeles programeerimisega ega kasutanud rakendusliideseid veebikaardi loomise jaoks. Kolm inimest ei saanud üldse ühtegi vastust, 1 inimene leidis 3 õiget käsku ja 1 inimene leidis ühe õige käsu ka ühele raskematest ülesannetest. Vaatamata sellele, et testimise käigus saadud sisuliste vastuste arv on väike, oli antud põhjalik tagasiside ning detailsed kommentaarid kasutatavust iseloomustatavatele küsimustele.

Omandamise lihtust iseloomustatakse vajalike teadmiste allikate rohkusega ning testimise käigus anti tagasiside sellest, kust oli vajalik info otsitud.

Vastuseid prooviti otsida igalt poolt Internetist otsides. Otsiti ka ametlikust dokumentatsioonist, kui ka teistest kohtadest nt foorumitest. Paljud kasutasid ametlikke dokumentatsioone ning andsid oma hinnangu selle struktureerimisele ja loogilisusele. Näiteks OpenLayers 3 dokumentatsiooni nimetati hästi struktureerituks, kuid samal ajal Leaflet oma dokumentatsioon tundus asjalikum ja lihtsamini arusaadav. Kuid samal ajal oli rõhutatud, et kuna dokumentatsioonides esinevad näited või kasutamise *"how to"* koodijupid harva, siis on raske ettekujutada kuidas meetodit või parameetrit kasutada. Õiged vastused leiti pigem teistest allikatest ja näidetest ettekirjutatud koodijuppide abil ja katsetamise käigus.

Peamiseks probleemiks kõikide dokumentatsioonidega töötades oli nimetatud respondentide JavaScript'i kogemuse puudumine, JavaScript'i süntaksi põhimõtete/loogika, käskude defineerimise põhireeglite ja üldiste põhimõtete teadmise puudumine, mis takistab lakooniliselt kirjeldatud meetodite ja parameetrite kasutamist ilma nende kasutamisnäidete ja loogikat seletavate koodilõikudeta dokumentatsioonis.

Kasutatavuse juures oleva efektiivsuse näitaja iseloomustamiseks paluti mõõta iga API jaoks ülesannete lahendamisele kulunud aeg, ehk arendamise kiirust. Kahjuks ükski kvalitatiivse testimise respondent ei saanud kõikide rakendusliideste ülesannetega hakkama ja nende poolt määratletud aeg on suurema arvu kordades iga API jaoks ühesugune, mida nad kulusid proovimiseks ja katsetamiseks. Seega antud näitaja asemel saab efektiivsuse iseloomustamiseks kasutada testi sisuliste küsimuste vastused, mis on toodud LISAs 3 võrreldes antud käskude pikkust, mida võib pidada korreleeruvaks arendamise kiirusega, kuna arendajal tuleb erineva pikkusega käsud kirjutada.

Kokku oli ülesandeid viis (LISA 2): kihi lisamine kaardile, vektorobjektide arvu määramine, kujunduse muutmine, kaardivälja keskpunkti nihutamine ja suurendusaste muutmine. Esimesed käsud, mille ülesandeks on ainult kihi lisamine kaardile, ei erine üksteisest pikkuse poolest, kuid teise küsimusega on juba näha erinevused API'de vahel. Ülesandeks on leida kui palju vektorobjekte kihis on ning siin on Google Maps'i rakendusliidese koodipikkus vajaliku funktsionaalsuse saavutamiseks palju pikem, kui teiste API'de puhul.

Selle põhjuseks on sisseehitatud funktsiooni puudumine ning seepärast vajadus antud funktsionaalsus ise kirjutada. Kujunduse muutmise ülesande puhul ei olnud vajalikku sisseehitatud funktsiooni juba kolme rakendusliidese puhul: Leaflet, Google Maps ja ArcGIS API. Seega oli ka siin vaja võrreldes OpenLayers 2 ja 3'ga funktsionaalsuse saavutamiseks rohkem koodi kirjutada ja aega kulutada. Uue keskpunkti määramise ülesanne on tulemusena jälle pikem koodilõik Leaflet'i, Google Maps'i ja ArcGIS'i puhul. Leaflet'i ja Google Maps'i koodipikkus on tingitud vajadusest kasutada koordinaatide konverteerimist ning ArcGIS'i puhul tingitud üldisest funktsiooni pikemast süntaksist. Viimane ülesanne ei näita suuri erinvusi vastavate käskude pikkustes, kuna kaardi suurendusastme funktsioonid esinevad kõikides API'des ning on oma loogika järgi sarnased.

Seda arvesse võttes võib öelda, et mõlemad OpenLayers'i rakendusliidese versioonid nõuavad vähem arendamise aega, kuna sisaldavad rohkem sisseehitatud funktsioone mitmekülgse funktsionaalsuse saavutamiseks. Ning vastupidi, Google Maps'i ja Leaflet'i rakendusliidesed nõuavad sama funktsionaalsuse saavutamiseks rohkem koodikirjutamist, kuna ei sisalda kvalitatiivses testis vajaliku funktsionaalsuse saavutamiseks sisseehitatud käske. Selle põhjuseks Leaflet'i puhul võib olla rakendusliidese moodulitepõhine ehitus ja vajalike käsude olemasolu lisamoodulites ning Google Maps'i puhul rakendusliidese eelistus töötada teiste ruumiandmete vormingutega.

Kasutatavuse rahulolu näitaja üheks aspektiks oli rahulolu lõplike rakenduste vaikimisi väljanägemisega. Respondentide poolt anti tagasiside ka rakendusliideste abil loodud interaktiivsete kaartide vaikimisi kartograafilise kujunduse elementide kohta. Rakendusliideste interaktsiooni nuppudest ja vektorkihtide välimusest saadud tagasiside võtab kokku arvamus, et Openlayers 3 välimus on parim tänu asjalikele ja hästi kasutatavatele kaardinuppudele ja korrektsele punktandmete välimusele ning märgatud asjaolule, et aluskaardi kuvamine on siin oluliselt parem kui teiste rakendusliideste puhul. OpenLayers 2 ja ArcGIS API punktobjektide vaikimisi kujundus on samuti hea, kuid Google Maps'i ja Leaflet'i puhul oli punktide välimus punktide paljususe puhul (välja zoomitult) vastajate jaoks veidi liiga

kirju, mis on tingitud sellest, et nimetatud rakendusliidestega punktobjektide kujundamine toimub markeri pildi abil. Polügoonide kuvamine hinnati heaks OpenLayers 3 ja ArcGIS'i puhul, kuid teistel API'del olid maakonnapiirid juba liialt laiaks tehtud ja torkasid enim silma.

6. Arutelu

Enne rakendusliidese valikut tuleb kindlasti kõik rakenduse eesmärgid ja tehnilised nõuded selgeks teha ja langetada edasine valik kõiki aspekte arvestades. Näiteks võiks olla eesmärgiks luua rakendus, mis töötaks hästi mobiilsetel seadmetel ja oleks väga kiire. Pidades silmas kvantitatiivse testimise tulemusi saab järeldada, et esiteks peaks sellel juhul valikut mõjutama rakendusliidese suurus. See on esimene põhjus, mis saadud rakenduse kiirust mõjutab. Ei ole kahtlust, et selleks, et saavutada rakenduse head kiirust võiks suurimad ja mahukamad rakendusliidesed valikust ära jätta ning eelistada tuleb väiksemaid rakendusliideseid nagu Leaflet. Väike rakendusliides on soovitatav ka juhul, kui lõpliku kaardirakenduse tehniliseks nõudeks on töötamine aeglase interneti ühendusega.

Kutsete arv rakendusliidese poole on näitaja, mida võiks arvestada juhul kui eesmärgiks on uue kaardirakenduse kiire arendamine. Arendamise kiirusega korreleeruvaks saab nimetada ka kvalitatiivse testimise efektiivsuse mõõdet. Mõlemad näitajad iseloomustavad sama funktsionaalsuse saavutamiseks vajaliku koodi pikkust, kuid esimene nendest arvestab vaid päringuid API poole ja teine arvestab kokkuvõtvalt kogu keelelist konstruktsiooni käsu moodustamiseks. Siin on vaja mainida, et mõlemad näitajad sõltuvad näiteks sellest, milliseid andmeid kasutatakse rakenduse alusena. Nagu töös olevad näited demonstreerivad, on vähem API kutseid ja üldiselt vähem koodikirjutamist nõudvad OGC vabu standardeid kasutavad tasuta ja avatud lähtekoodiga OpenLayers ja Leaflet rakendusliidesed ning nendega võrreldes märgatavalt rohkem koodi nõuavad varalised teegid Google Maps ja ArcGIS. ArcGIS API erineb selgelt teistest WFS ja GeoJSON kihi lisamisel ning Google Maps WMS kihi lisamisel: kahjuks töötades avatud standarditega on ülalnimetatud rakendusliidesed ilmetud, ei ole nii väljendusrikkad, kui ülejäänud rakendusliidesed. Samuti on suurem arv mitmekülgseid sisseehitatuid funktsioone omaduseks, mis võib koodi pikkusele mõju avaldada,

kuna nõutud käsu puudumisel tekib vajadus seda iseseisvalt programmeerida, mis tavaliselt nõuab palju rohkem koodi kirjutamist ja arendamise aega. Suurema arvu erinevaid funktsioone võib leida mahukamas teegis ja see on veel üks põhjus API valikul võrrelda rakendusliideste suurust kilobaitides. Seega, kui eesmärgiks on raske funktsionaalsus ja mitmekülgsed ruumilised funktsioonid, siis võib oletada, et mahukam ja raskem teek võiks pakkuda ka suuremat hulka erinevaid mitmekülgseid funktsioone. Seda oletust tuleb kontrollida dokumentatsiooni alusel ning juhtumil, kui väide igal konkreetsel juhul leiab kinnitust, eelistada antud mahukat teeki. Näiteks OpenLayers 2 teek on üks mitmekülgsematest teekidest, kuid samal ajal ka üks raskematest ning sama puudutab ka ArcGIS rakendusliidest.

Lisaks näitasid testid, et erinevate veebilehitsejate vahel on erinevusi ühe ja sama kaardirakenduse kiiruses. Seda oleks hea mainida rakenduse kasutajatele, et nende kogemus oleks parem ja töö loodud rakendusega oleks kiirem. Kindlasti sõltuvad konkreetsed suurused antud testi puhul arvuti parameetritest ja interneti ühenduse kiirusest ning kasutatava veebilehitseja versioonist, kuid erinevuste protsendid jäävad igal juhul samadeks. Siin saab järeldada, et Chrome'i veebilehitseja kasutamine võib parandada rakenduse töökiirust, efektiivsust ja jõudlust ning kuna iga rakendusliidese jaoks kiiremaks osutus üks ja sama veebilehitseja, siis see ei mõju rakenduse jaoks API valikut.

Töö käigus selgus, et ühe ja sama ülesande täitmise kiirus erineb eri API'des. See oleks üks olulisemaid aspekte mida võtta arvesse API valikul. Näiteks kui ülesandeks on WMS kihi näitamine, võib selgelt aeglasema Google Maps API valikust ära jätta. Kui ülesandeks on näidata vektorobjektide kihti, siis vektorobjektide visualiseerimise kiiruse mõõtmise testi järgi eelistatakse Google Maps API't või Openlayers 2. Kui ülesandeks ei ole näidata tuhat objekti või rohkem, siis erinevus API'de vahel esineb, aga ei ole kasutajale silmaga märgatav, kuna tegemist on millisekundites mõõdetud suurustega. Kui aga vektorobjektide arv on väga suur ning ulatub kümnete tuhandeteni, muutub ooteaeg kasutajale märgatavaks, ning erinevused API'de vahel on samuti silmale nähtavad. OpenLayers 2 on palju kiirem teistest API'dest

kui räägitakse GeoJSON faili visualiseerimisest, kuid kasutades andmeallikana WFS serverit selline erinevus väheneb ja esikohal on Google Maps.

Lisaks sellele ajale, mis on toodud tulemustes (*Tabel 5*) vektorkihi lisamise kohta, lisandub veel aeg selleks, et visualiseerida kõik objektid ekraanil. Sel juhul võtab veel palju aega nt Leaflet'i punkt-objektide markeri ja Google'i markerite lisamine, kuna need lisatakse piltidena (Leaflet puhul isegi kaks pilti iga objekti kohta: marker ise ja tema vari eraldi piltidena). Kiirem protsess toimub OpenLayers 2, OpenLayers 3 ja ArcGIS API puhul, mis lisavad vaikimisi punktobjektide visualisatsiooni lennult graafikana HTML `<canvas>` elemendil. Seega WFS kihi puhul muutub Google Maps'i kihi lõplik visualiseerimine pikaks ja seega muutuvad API valimisel paremaks variandiks OpenLayers 2 või OpenLayers 3. See tähendab, et API valik sõltub visualiseeritavate objektide ja kihtide arvust ning tüüpidest.

Kõik ülalmainitu viib järelduseni, et vaatluse all olevad rakendusliideseid (OpenLayers, Leaflet, Google Maps ja ArcGIS API) saab kasutada sarnastel, kuid samal ajal erinevatel eesmärkidel. Näiteks Leaflet (ja niimoodi väitis ka Santiago [31]) ja Google Maps on ideaalsed lihtsate kaartide loomiseks, visualiseerimiseks markeritega või hüpikakendega, kuid OpenLayers ja ArcGIS sobivad rohkem GIS orienteeritud lahenduste loomiseks paljude ruumiliste funktsioonidega ning Leaflet on sobiv kasutamiseks mobiilsete kaardirakenduste osana.

Samuti tuleb arvestada milliseid ruumiandmete allikaid ja vorminguid kasutatakse. Varalised rakendusliidesed (Google Maps API ja ArcGIS API) on suunatud kasutama oma andmete vorminguid (vastavalt Google Fusion Table ja KML ning ArcGIS for Server vormingud), kuid takistuseks võivad olla litsentsitingimused. Samas tasuta ja avatud lähtekoodiga rakendusliidesed ülalmainitud andmete vormingutega ei tegele ning eelistavad avatud standardeid. Seega sõltub rakendusliidese valik ka olemasolevate ruumiliste andmete vormingust.

Peamiseks kvalitatiivseks kriteeriumiks, kuidas otsustada API valikut on kasutatavus, mis iseloomustab pigem subjektiivseid aspekte ja API omadusi, mis võivad kallutada valima üht või teist rakendusliidest. Antud kasutatavuse

testimist iseloomustab subjektiivsus arvestades kontseptuaalsete teadmiste erinevusi. Seda on väidetud ka varem [10] rõhutades iga API iseloomuliku eriterminoloogia teadmise olulisust, mis lihtsustab tarkvaraarendajale programeerimise protsessi ja vajalike näidete otsimist aidates vältida mitu korda päringut ümbersõnastamist enne, kui õige leitakse. Niimoodi osutub uue API selgeks õppimine lihtsamaks, kui eelnevalt oli juba proovitud sama töö teise rakendusliidesega ja on esmane ettekujutus kuidas kaardirakenduse protsess toimib. Samamoodi on ka lihtsam selgeks teha rakendusliidest, kus käskude ja konstruktorite nimetused on koostatud loomulikumas keeles.

Ko ja Riche [10] rõhutavad, et kontseptuaalsed teadmised on väga olulised. Niimoodi on geograafidel sügavam arusaam nt projektsioonidest ja nende teisendustest, programmeerijatel aga on lihtsam erinevad koodilõigud ja näited kokku panna. See tähendab, et kasutajate taust on ka oluline tegur kasutatavuse hindamisel. Antud uuring aga on suunatud kartograafidele, selleks et tõsta nende poolt uute tehnoloogiate kasutamist, seega kvalitatiivses testis osalejate grupp kuulub samasse kategooriasse, mis teeb tulemused ja järeldused antud kontekstis representatiivsemateks.

Täiesti korrektse lõpliku kartograafilise produkti saamiseks soovitatakse kartograafide ja programmeerijate koostööd. Näiteks soovitavad Quinn ja Dutton [30] arendajal võimalusel konsulteerida kartograafidega, et valida sobilikum kartograafiline kujundus, nt sobilikud joonte laiused, värvid, taustad ja sümbolid, et tulemusena tekkiv rakendus oleks nii kartograafia poolest kui ka tehnilisest poolest korrektne. Samamoodi soovitatakse ka kartograafidel konsulteerida arendajatega, et saavutada sujuv ja kiire rakendus, mis oleks võimalikult optimaalselt ehitatud.

See uuring ning ka eelnevad uuringud [10] on näidanud, et kõige tõhusamalt tõstavad kasutatavust selgitustega koodinäited. Kiire vastuse leidmise jaoks on oluline näidete rohkus, mis tähendab, et palju sõltub rakendusliidese populaarsusest ehk kasutajate arvust ja küsimuste ja vastuste arvust, mida kasutajad on küsinud varem. Ka dokumentatsiooniga veebilehe disain on olulise tähtsusega tegur. Mõnikord võib leida väga sarnase näite oma ülesandega, kuid tuleb lisada veel mõni parameeter ja selle õpetamiseks

on paras koht ametlik dokumentatsioon. Dokumentatsioonis olev lihtne otsinguvahend ja arusaadav konstruktsioon lihtsustavad vajaliku parameetri kiiret leidmist. Kuid puhtalt dokumentatsiooni kasutamine võib olla raskendatud inimestel, kellel puudub antud programeerimiskeeles arendamise kogemus ning nende jaoks on sobivamad pikemad näited. Varasemad uuringud [17] näitavad, et ideaaljuhul peab dokumentatsioon olema täpne ja üheselt mõistetav ning selles võiks kohe sisalduda koodilõigud ja õpetused.

Olles subjektiivne omadus, on kasutatavus raskesti hinnatav, seega antud uuringus püüti koostada test ja testi alusel küsimused nii, et kujuneks välja kasutatavuse optimaalne hinnang. Küsimused, mis on rühmitatud, et iseloomustada erinevaid kasutatavuse aspekte, olid koostatud nii, et avastada olemasolevad probleemid ning saadud vastused mõnevõrra iseloomustavad rakendusliideste probleemseid kohti.

Paljud kasutatavuse testi respondendid tegid ettepaneku ka Tartu Ülikoolis Geograafia ainekavas uue sissejuhatava aine loomiseks, mis tutvustaks kartograafiliste rakendusliideste (API) kasutamist ning muid kaasaaegseid võimalusi, mida pakub GIS ja programeerimise interaktsioon. Sellised programmeerimise teadmised pakuvad kartograafidele huvi kuna annavad palju vabadust ruumiliste andmete visualiseerimisel, lubavad palju interaktiivsust ja on paindlikud saadud tulemuse jagamisel kogukonnaga. Kuna kvalitatiivse testimise respondendid ei puutunud kartograafiliste rakendusliidestega varem kokku, on näitlik ka nende esimene mulje API'dest ning on oluline, et on avaldatud soovi selliseid rakendusliideseid tulevikus kasutada.

Kokkuvõte

Web 2.0 pakub suurepäraseid võimalusi luua ja jagada geograafilist informatsiooni interaktiivselt visualiseerides. Teadlased ja spetsialistid võivad saada tehnoloogia arengust kasu, ülikoolides tuleb ka tähelepanu pöörata uutele võimalustele kartograafias. Selliseks tehnoloogiaks on JavaScript'i kaardistamise rakendusliidesed. Praegu on kättesaadav lai valik vahendeid ja andmeid eri keerukusastmega, erinevate litsentsi tingimustega, mis koos huviga geoinformaatika vastu toob kaasa veebikartograafiaga laialdase tegelemise. Rakendusliidese valimisel on oluline uurida ülevaadet pakutud kasulikest võimalustest ja iga konkreetse ülesande jaoks arvestada API'de paindlikkusega püstitatud ülesannete lahendamisel.

Antud magistritöö kirjutati selleks, et anda ülevaade tuntumatest ja konkureerivatest JavaScript'i teekidest veebikartograafia jaoks: OpenLayers, Leaflet, Google Maps API ja ArcGis API. Eesmärgiks oli ka rakendusliideste tugevuste ja nõrkuste väljaselgitamine selleks, et alustades esimest korda tööd kaardistamise teekidega oleks võimalus teha õige otsus ja valida oma eesmärkidele, oskustele ja olemasolevatele ruumilistele andmetele paremini sobiv teek.

Töös on antud ülevaade põhilistest funktsioonidest, mida kaardirakenduse loomisel kohatatakse: WMS kihi lisamine, WFS kihi lisamine ja GeoJSON kihi lisamine. Ülalnimetatud ülesannete lahendamiseks vajalik kood on lahti seletatud, seega võib seda kasutada juhendina API'ga tööd alustades. Ülalnimetatud funktsioone ja rakendusliideseid võrreldi omavahel erinevate kvantitatiivsete näitajate abil, et mõõta raamistike tugevused ja nõrkused.

Projekti jaoks kartograafilise rakendusliidese valik võib olla põhjustatud paljude tegurite koosmõjust: API suurus ja loodava rakenduse üldine kiirus ning võimalus töötada mobiilsetel seadmetel, API poole kutsete arv ja rakenduse arendamise kiirus, kihi visualiseerimise kiirus ja valitud ruumiandmete standardid ning vormingud ning ruumiandmete kogus, API

kasutatavus ning arendaja programmeerimise- ja kartograafilised oskused ning teemakohased teadmised jne.

Nagu ka varasemad uuringud näitasid, sõltub API valik rakenduse tüübist ja eesmärgist, seega on saadud tulemused abiks otsuse tegemisel, kui eesmärgid on juba püstitatud. On oluline, et projekti tingimused ja vajadused mõjutavad erinevate API'de valikut. Olles kaardirakenduse arendaja, on oluline mõista rakenduse üldist struktuuri ja API funktsioonide mustreid. Selline teadmine aitaks kiiresti hakkama saada uue kaardistamise API'ga vastavalt vajadusele.

COMPARISON OF MAPPING APIs

Olga Troškina

Summary

Web cartography gives limitless possibilities to create and share interactive maps. There are a lot of different JavaScript mapping APIs, yet every one of them has its own benefits and disadvantages and has different properties making the most suitable choice between them difficult, but very important. This master thesis aims to investigate and analyze technical capabilities of the various libraries and improve the process of selection of suitable mapping API with a purpose of achieving optimal and cost-effective working process of creating a web map.

This work introduces the history of web cartography, its development trends, as well as problems and challenges. The process of creating web map is described along with a specific mapping APIs that are used in this work because of their popularity: Google Maps, OpenLayers, Leaflet and ArcGIS API. The paper gives an overview of the implementation of the most common features using APIs listed above and comments on the most important parts of the JavaScript code.

The aim of the thesis is to create examples to help choose the most suitable environment and acquire techniques and practical skills to create web-based map application. There are five prototype applications done and 35 functionality examples altogether: example of using WMS layer and vector layer examples with data sources from WFS or from file (both with different number of objects). Another aim of the paper is to compare complexity , effectiveness and appropriateness of these libraries' for various challenges. For that purpose qualitative and quantitative comparison was done.

Choosing the interface of the mapping application programming is a combination of various decisions based on the properties of the mapping APIs and purpose of the mapping application: API's size and application needed, overall speed and ability to work with mobile devices, number of API calls and speed of application development, layer visualization speed and selected spatial data standards and formats, API usability and developer's programming and cartographic skills as well as knowledge of needed terminology. These all can be important when choosing appropriate API for new project.

This study showes that the choice of maping APIs depends on the type of application and technical requirements, thus obtained results can be helpful in deciding if the targets are already set. Being a map application developer it is important to understand the overall structure of an application and an API functions patterns so, if needed, studying a new mapping API, that best suits new requirements, is easy.

Kasutatud kirjandus

1. Aghaee, S., Pautasso, C. 2010. Mashup development with HTML5. *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups* 10-18.

2. Bostock, M., Davies, J. 2013. Code as cartography. *The cartographic journal*. 50, 129-135.

3. Dumas, J. S., Redish, J. 1999. A practical guide to usability testing. Intellect Books, Exeter, England.

4. Erle, S., Gibson, R., Walsh, J. 2005. Mapping hacks: Tips & tools for electronic cartography. O'Reilly Media, Inc.

5. Fernandes A. I., Goulão M., Rodrigues A. 2013. A Comparison of Maps Application Programming Interfaces. *16th AGILE Conference on Geographic Information Science*. 1-13.

6. Gale, G. 2013. Push Pins, Dots, Customisation, Brands and Services: The Three Waves of Making Digital Maps. *The Cartographic Journal*. 50, 155-160.

7. Gibin, M., Singleton, A., Milton, R., Mateos, P., Longley, P. 2008. An Exploratory Cartographic Visualisation of London through the Google Maps API. *Appl. Spatial Analysis*. 1, 85–97.

8. Haklay, M., Singleton, A., Parker, C. Web mapping 2.0: The neogeography of the GeoWeb. *Geography Compass*. 2(6), 2011-2039.

9. Hu, S.,Dai, T. 2013. Online Map Application Development Using Google Maps API, SQL Database, and ASP.NET. *International Journal of Information and Communication Technology Research*. 3, 102-110.

10. Ko, A. J., Riche, Y. 2011. The role of conceptual knowledge in API usability. *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium* 173-176.

11. Lee, K. 2009. Technical Architecture for Land Monitoring Portal using Google Maps API and Open Source GIS. *17th International Conference on Geoinformatics.* 1-5.

12. Li, M. 2013. Architecture of Web GIS based on Web2.0. *IEEE Conference Anthology.* 1-4.

13. Liu, S., Palen, L. 2010. The New Cartographers: Crisis Map Mashups and the Emergence of Neogeographic Practice. *Cartography and Geographic Information Science.* 37(1), 69-90.

14. Martin, R. C. 2008. Clean code: a handbook of agile software craftsmanship. Pearson Education.

15. Näslund, M. 2007. Web-based mapping: An evaluation of four JavaScript APIs. *Final thesis in Computer and Information Science at Linköping Institute of Technology.*

16. Peterson, M. 2012. Online Maps with APIs and WebServices. Springer.

17. Piccioni, M., Carlo A. F., and Bertrand M. 2013. An empirical study of api usability. *Empirical Software Engineering and Measurement, ACM/IEEE International Symposium.* 5-14.

18. Schmidt, M., 2010. Web Mapping 2.0. What it is and how to use it.

19. Schmidt, M., Weiser, P. 2012. Web Mapping Services: Development and Trends. *Online Maps with APIs and WebServices. Springer Berlin Heidelberg.* 13-21.

20. Schnabel, O., Hurni, L. 2009. Cartographic Web Applications - Developments and Trends. *Proceedings of the 24rd International Cartographic Conference.*

21. Shekhar, S., Xiong, H. 2008. Encyclopedia of GIS. Springer.

Interneti allikad

22. Dincer, A. 2014. Web Mapping APIs' Vector Performance Comparison. [WWW] http://www.geowebdeveloper.com/2014/06/01/web-mapping-apis-vector-performance-comparison/ (10.02.2015)

23. Farkas, G. 2014. Web Mapping Tutorial. [WWW] http://webmappingtutorial.blogspot.com/ (10.04.2015)

24. Hocevar A. 2013. OpenLayers 3 Google Maps API Compared. [WWW]

25. Kennedy N. 2007. JavaScript Map API comparison. [WWW] http://www.niallkennedy.com/blog/2007/08/map-api-comparison.html (10.04.2015)

26. Larkin, S. 2012. Terms and Trends in DIY and Open Source Online Maps. [WWW] http://www.nten.org/articles/2012/terms-and-trends-in-diy-and-open-source-online-maps (10.04.2015)

27. Leler, W., Oliver, D. 2014. Guide to Google Maps API - and 6 great alternatives. [WWW] http://www.creativebloq.com/web-design/google-maps-api-7122779 (10.04.2015)

28. Lovelace, R. 2014. Testing web map APIs - Google vs OpenLayers vs Leaflet. [WWW] http://robinlovelace.net/software/2014/03/05/webmap-test.html (10.04.2015)

29. O'Brien O. 2011. Mapping APIs – Google vs OpenLayers [WWW] http://gemma.blogweb.casa.ucl.ac.uk/2011/06/mapping-apis-google-vs-openlayers/ (10.04.2015)

30. Quinn, S., Dutton. J.A. 2014. GEOG 585: Open Web Mapping. Online course, e-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University. [WWW] https://www.e-education.psu.edu/geog585/node/508 (10.04.2015)

31. Santiago, A. 2013. The OpenLayers fallen and Leaflet arise... sure??? [WWW] http://acuriousanimal.com/blog/2013/05/05/the-problem-with-openlayers/ (10.04.2015)

32. Thomas, H. 2013. 25 Useful JavaScript Libraries And Tools for Creating Interactive Maps. [WWW] http://www.instantshift.com/2013/08/26/useful-javascript-libraries-for-maps/ (10.04.2015)

33. Vallaste, H. E-teatmik: ingliskeelsete info- ja sidetehnoloogia terminite seletav sõnaraamat [WWW] http://vallaste.ee/ (10.04.2015)

34. Weber P. 2011. The JISC G3 project is setting out to bridge the learning Gap between the Geoweb and GIS. [WWW] http://jiscg3.blogspot.com/2011/06/decision-time-google-maps-vs-openlayers.html (10.04.2015)

LISA 1

1 OpenLayers 2.11

1.1 WMS

```
wms.events.register("loadend", wms, function () {
            var time = performance.now() - start;
            console.log(time);
        });
```

1.2 WFS

```
vectorLayer.events.register("loadend", vectorLayer, function () {
          var time = performance.now() - start;
            console.log(time);
       });
```

1.3 GeoJSON

```
vectorLayer.events.register("added", vectorLayer, function () {
          var time = performance.now() - start;
            console.log(time);
       });
```

2 OpenLayers 3.2.1

2.1 WMS

```
map.on('postrender', function(){
            var time=performance.now()-start;
            console.log(time);
        });
```

2.2 WFS/GeoJSON

```
map.on('postrender', function(){
            var time=performance.now()-start;
            console.log(time);
        });
```

3 Leaflet 0.7.3

3.1 WMS

```
wms.on("load",function() {
          var time = performance.now() - start;
          console.log(time);
      });
```

3.2 WFS/GeoJSON

```
var i=0;
map.on("layeradd",function() {
        i = i + 1;
        if (i == 300) {
            var time = performance.now() - start;
            console.log(time);
        }
    });
```

4 Google Maps JavaScript API v3 3.018

4.1 WMS

```
google.maps.event.addListener(wms, 'tilesloaded', function (evt) {
            var time = performance.now() - start;
            console.log(time)
        });
```

4.2 WFS/GeoJSON

```
i=0;
google.maps.event.addListener(vectorLayer, 'addfeature', function (evt) {
            i=i+1;
            if(i==300) {
                var time = performance.now() - start;
                console.log(time);
            }
        });
```

5 ArcGIS API for JavaScript 3.12

5.1 WMS

```
wms.on("update",function() {
            var time = performance.now() - start;
            console.log(time);
        });
```

5.2 WFS/GeoJSON

```
var i = 0;
vectorLayer.on("graphic-add", function () {
            i = i + 1;
            if (i == 300) {
                var time = performance.now() - start;
                console.log(time);
            }
        });
```

LISA 2

Sissejuhatus

See on JavaScript kaardistamise rakendusliideste testimise ülesannete leht. JavaScript rakendusliides ehk programmiliides ehk API (inglise keeles *Application Programming Interface*) on reeglistik, JavaScript klasside ja omavahel seotud objektide kogum, kus iga objekt omab atribuute ja meetodeid. Selle peamine eesmärk on rakenduste standardsete osade realiseerimise lihtsustamine. Veebikartograafia JavaScript raamistik paneb põhiliselt kokku andmed ja pildid, ehk ta saab ruumilistest andmetest aru ja visualiseerib neid objektidena kaardil ning varustab kaardi kõigi vajalike kontrollidega. See on abivahend interaktiivsete veebikaartide loomiseks.

Kuna arendajad saavad valida paljude erinevate rakendusliideste vahel, on oluline leida endale sobivam teek, leida selline API, et programmeerija saavutaks määratletud eesmärgid efektiivselt, tõhusalt ja rahuloluga – need parameetrid iseloomustavad kasutatavust.

Testimise eesmärgiks ongi hinnata API-de kasutatavust (*usability*), ehk seda kuivõrd arendajad saavutavad eesmärgid KIIRESTI ja RAHULOLUGA. Testimiseks tuleb sooritada mõned ülesanded kasutades viit rakendusliidest, peamine on teie põhjalik tagasiside. On olemas kümned kaardistamise API-d, kuid antud testi jaoks valiti viis tuntumat ja konkureerivat JavaScript teeki veebikartograafia jaoks: OpenLayers 2 ja 3, Leaflet, Google Maps API ja ArcGis API.

See test on üks osa magistritööst, mis on keskendatud väljaselgitamisele, milline eespool mainitud teek võiks olla mõnelt poolt sobivam või lihtsam antud ülesande jaoks. Töö keskendub API-de tugevuste ja nõrkuste väljaselgitamisele.

Vaatamata sellele, et kaardistamise API-d pakuvad rikast funktsionaalsust, on testimiseks tavaliselt valik ülesandeid, mis on tavapärasemad ka tegeliku API-ga töö jaoks. Antud juhul tuleb testimiseks sooritada mõned lihtsad ülesanded, mille seas on nt suumimine, kihtide sisse ja välja lülitamine, stiili muutmine. Prototüüpe kasutatakse hindamiseks kuidas iga rakendusliides mõjutab lõpliku rakenduse loomise keerukust.

Kasutada võib mistahes *interneti* allikaid: API-de dokumentatsioonid, foorumid, tutorial-õpetused jne, kuigi soovitus on proovida esmalt ise midagi välja mõelda, kuna paljud käsud on üheselt mõistetavad inglisekeelsed sõnad. Kuivõrd meid huvitab API kasutatavuse juures ka see, kui lihtsasti potentsiaalne kasutaja vajadusel juhiseid leiab, ei ole siinkohal ühtegi viidet juhendmaterjalidele antud.

Testi vastustes oodatakse subjektiivset hinnangut JavaScript koodi elegantsusele ja oma rahulolule, tagasisidet tajutavale kasutatavusele ja kõiki töö käigus tekkinud mõtteid, mis puudutavad vastuse otsingu protsessi, mõne veebilehe näiteid või dokumentatsiooni, prototüüprakenduse välimust jne. Järgnevalt esitatakse ülesannete nimekiri, mis tuleb valmis teha kasutades veebilehitseja konsooli.

Testimiseks on 5 prototüüprakendust:

1. OpenLayers 2
2. OpenLayers 3
3. Leaflet
4. Google Maps API
5. ArcGIS API for JavaScript

Testi tuleb seega teha 5 korda ja iga API jaoks täita ankeet ja hinnata API raskust.

Oluline info

Kõikidel test-rakendustel on sama ülesehitus:

“map” kaardi objekt

“base” Maa-ameti aluskaart

“vectorLayer” vector punktidega kiht

“maakond” maakondade polügoonide kiht

Test-rakendused asuvad aadressil:

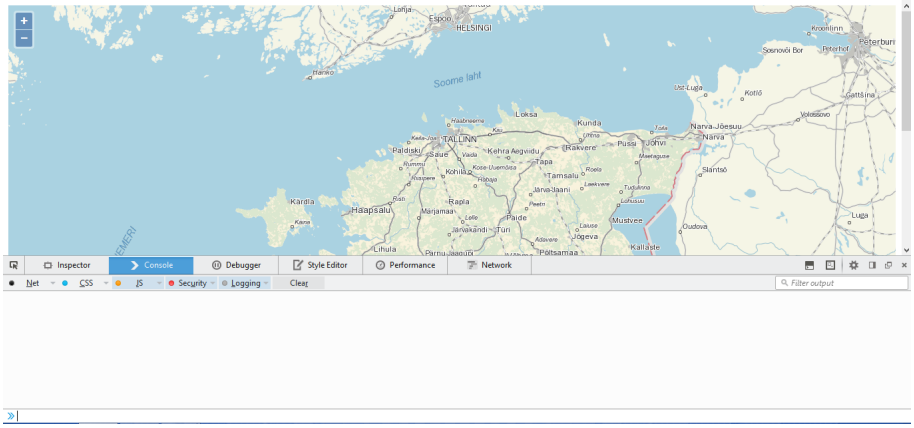
1. OpenLayers 2: `http://kodu.ut.ee/~anuket/test/ol3.html`
2. OpenLayers 3: `http://kodu.ut.ee/~anuket/test/ol2.html`
3. Leaflet: `http://kodu.ut.ee/~anuket/test/leaflet.html`
4. Google Maps API: `http://kodu.ut.ee/~anuket/test/gmap.html`
5. ArcGIS API for JavaScript: `http://kodu.ut.ee/~anuket/test/arcgis.html`

Test

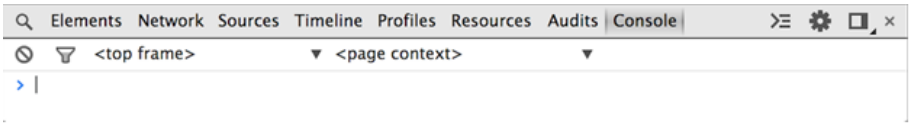
1. Ava JavaScript konsool.

Firefox Command - Option - K (Mac) or Control -Shift -K (Windows/Linux) *(Joonis 1)*

Chrome Command - Option - J (Mac) or Control -Shift -J (Windows/Linux) *(Joonis 2)*



Joonis 1: Firefox



Joonis 2: Chrome

2. **Tööta ühe API-ga korraga (tee ülesanded 1-5 lõpuni ja siis alusta teise API-ga).**
3. **Palun vaata kella, et iga API testi lõppedes saaks testi tegemiseks kulunud aja kirja panna (lk. 6 Küsimustiku I osa 1. küsimuse tabel).**
4. Leia vastused (koodilõigud) järgmistele küsimustele. **Täida tabel vastavate käskudega.** Anna ka põhjalikud kommentaarid iga ülesande kohta: kas vastus on lihtne, loogiline jne, mis tekitas raskusi.
5. Täida testi lõpus olev ankeet-küsimustik kui töö kõigi viie API-ga on lõpetatud.

Vastuste leht

1. Lisa **“vectorLayer”** kaardile.

API	Käsk
OpenLayers3	
OpenLayers2	
Leaflet	
Google Maps API	
ArcGis API for JavaScript	

KOMMENTAARID: .....
.....

2. Leia kui palju punkte on selles kihis.

API	Käsk
OpenLayers3	
OpenLayers2	
Leaflet	
Google Maps API	
ArcGis API for JavaScript	

KOMMENTAARID: .....
.....

3. Muuda Viljandimaa maakonna kujundust **“style** (id=”Vijandumaa”).

API	Käsk
OpenLayers3	
OpenLayers2	
Leaflet	
Google Maps API	
ArcGis API for JavaScript	

KOMMENTAARID: .....
.....

4. Muuda kaardi keskpunktiks punkt koordinaatidega **446500, 6562500** EPSG:3301 projektsioonis. Vihje: mitmed API'd tahavad koordinaate EPSG:4326 projektsioonis **(lat/lng!!!)**, selleks tuleb antud koordinaate konverteerida **proj4js** (https://github.com/proj4js/proj4js) abil, kus **proj1** on EPSG:4326 ja **proj2** on EPSG:3301. Keskpunkt on Hiiumaa lähedal.

API	**Käsk**
OpenLayers3	
OpenLayers2	
Leaflet	
Google Maps API	
ArcGis API for JavaScript	

KOMMENTAARID: .....
.....

4. Leia missugune ZOOM'i aste (number) on praegu ja pane selleks 3.

API	**Käsk**
OpenLayers3	
OpenLayers2	
Leaflet	
Google Maps API	
ArcGis API for JavaScript	

KOMMENTAARID: .....
.....

Küsimustik

OSA I: Palun vasta iga API jaoks eraldi kirjutades oma vastused tabeli vastavasse lahtrisse

1. Kui palju aega läks testi tegemiseks. Millele aeg kulus?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript
2. Kas rohkem infot leiti ametlikust dokumentatsioonist või teistest kohtadest? Kust täpsemalt? MIKS?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript
3. Mis nõudis kõige rohkem aega ülesannete täitmiseks? MIKS?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript
4. Kas tunned, et selle API-ga sinu edaspidine töö tuleb kiiremini? MIKS?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript
5. Kui õige vastus oli leitud, kas see tundus loogiline? MIKS?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript
6. Mis arvad vaikimisi kaardi nuppudest (zoom jne) ja vektor objektide vaikimisi välimusest? MIKS?
    - OpenLayers 2
    - OpenLayers 3
    - Leaflet
    - Google Maps API
    - ArcGIS API for Javascript

OSA II: Üldised küsimused pärast kõikide testide lõpetamist

Milline API tundus kõige lihtsam, milline kõige raskem. MIKS?

.....

.....

Kas viimastele küsimustele oli lihtsam vastata, kas käskude ülesehitus lõpuks oli arusaadavam? MIKS?

.....

.....

Palun vasta ka järgmistele küsimustele:

- Milline on sinu varasem programmeerimise kogemus?
- Kas oled kaardistamise raamistikega ka varem kokku puutunud? Kuidas?
- Kas soovid tulevikus neid kasutada? Millist?

LISA 3

1.

API	Käsk
OpenLayers3	map.addLayer(vectorLayer);
OpenLayers2	map.addLayer(vectorLayer);
Leaflet	vectorLayer.addTo(map);
Google Maps API	vectorLayer.setMap(map);
ArcGis API for JavaScript	map.addLayer(vectorLayer);

2.

API	Käsk
OpenLayers2	vectorLayer.getSource().getFeatures().length;
OpenLayers3	vectorLayer.features.length;
Leaflet	vectorLayer.getLayers().length;

Google Maps API	var count=0;
	vectorLayer.forEach(function (feature) {
	count=count+1
	});
	console.log(count);

ArcGis API for JavaScript	vectorLayer.graphics.length;

3.

API	Käsk
OpenLayers3	maakond.getSource().getFeatureById(”Viljandimaa”).setStyle(style2);
OpenLayers2	maakond.getFeatureByFid(”Viljandimaa”).style=style2;,maakond.redraw();

Leaflet	maakond.eachLayer(function (layer) {
	var id = layer.feature.id;
	if (id === ’Viljandimaa’) {
	layer.setStyle(style2);
	}
	});

Google Maps API	maakond.forEach(function (mk) {
	if(mk.id==”Viljandimaa”) {
	mk.setOptions(style2)
	}
	})

ArcGis API for JavaScript	dojo.forEach(maakond.graphics,function(graphic){
	if (graphic.attributes.id==’Viljandimaa’){
	graphic.setSymbol(style2)
	}
	})

4.

API	Käsk
OpenLayers3	map.getView().setCenter([446500, 6562500]);
OpenLayers2	map.setCenter((new OpenLayers.LonLat(446500, 6562500)));
Leaflet	map.setView([proj4(proj1, proj2, [446500, 6562500])[1], proj4(proj1, proj2, [446500, 6562500])[0]]);
Google Maps API	map.setCenter(new google.maps.LatLng(proj4(proj1, proj2, [446500, 6562500])[1], proj4(proj1, proj2, [446500, 6562500])[0]));
ArcGis API for JavaScript	map.centerAt(new esri.geometry.Point(446500, 6562500, new esri.SpatialReference({ wkid: 3301 })));

5.

API	Käsk
OpenLayers3	map.getView().getZoom(); map.getView().setZoom(3);
OpenLayers2	map.getZoom(); map.setZoom(3);
Leaflet	map.getZoom(); map.setZoom(3);
Google Maps API	map.getZoom(); map.setZoom(3);
ArcGIS API for JavaScript	map.getZoom(); map.setZoom(3);

LISA 4

1 OpenLayers 2.11

1.1 WMS

```
proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
    =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
    =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = new OpenLayers.Projection('EPSG:3301');
    var map = new OpenLayers.Map('map',
            {
                projection: projection,
                maxResolution: "auto",
                maxExtent: new OpenLayers.Bounds(350000, 6370000, 750000, 6630000),
                center: [550000, 6520000]
            });
    var wms = new OpenLayers.Layer.WMS(
            "Maaamet aluskaart",
            "http://kaart.maaamet.ee/wms/alus?",
            {
                "LAYERS": 'MA-ALUS'
            },
            {
                projection: projection
            }
    );
    map.addLayer(wms);
    map.zoomToMaxExtent();
    map.zoomTo(0);
```

1.2 WFS

```
  proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
      =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
      =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = new OpenLayers.Projection('EPSG:3301');
    var map = new OpenLayers.Map('map',
            {
                projection: projection,
                maxResolution: "auto",
                maxExtent: new OpenLayers.Bounds(300000, 6.3e+06, 800000, 6.7e+06),
                center: [550000, 6520000]
            });
    var wfs = new OpenLayers.Layer.Vector("vector layer", {
        isBaseLayer: true,
        strategies: [new OpenLayers.Strategy.BBOX()],
        protocol: new OpenLayers.Protocol.WFS({
            url: 'http://loom-gis.geo.ut.ee:8080/geoserver/ermas/ows?service=WFS',
            featureType: "testdata_geopnt",
            featureNS: 'http://loom-gis.geo.ut.ee:8080/geoserver/ermas',
            srsName: "EPSG:3301",
            geometryName: "geometry",
            maxFeatures: 300
        })
    });
    map.addLayer(wfs);
    map.zoomToMaxExtent();
    map.zoomTo(0);
```

1.3 GeoJSON

```
proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
    =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
    =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = new OpenLayers.Projection('EPSG:3301');
    var map = new OpenLayers.Map('map',
            {
                projection: projection,
                maxResolution: "auto",
                maxExtent: new OpenLayers.Bounds(300000, 6.3e+06, 800000, 6.7e+06),
                center: [550000, 6520000]
            });
    var geojson_format = new OpenLayers.Format.GeoJSON();
    var vectorLayer = new OpenLayers.Layer.Vector("GeoJSON",
            {
                isBaseLayer: true
            });
    vectorLayer.addFeatures(geojson_format.read(features));
    map.addLayer(vectorLayer);
    map.zoomToMaxExtent();
    map.zoomTo(0);
```

2 OpenLayers 3.2.1

2.1 WMS

```
proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
    =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
    =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = ol.proj.get('EPSG:3301');
    var map = new ol.Map({
        target: document.getElementById('map'),
        view: new ol.View({
            center: [550000, 6520000],
            projection: projection
        })
    });
    map.getView().setZoom(8);
    var wms = new ol.layer.Image({
        source: new ol.source.ImageWMS({
            url: 'http://kaart.maaamet.ee/wms/alus?',
            params: {
                LAYERS: 'MA-ALUS',
                VERSION: '1.1.1'
            }
        })
    });
    map.addLayer(wms);
```

2.2 WFS

```
 proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
     =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
     =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = ol.proj.get('EPSG:3301');
    var map = new ol.Map({
        target: document.getElementById('map'),
        view: new ol.View({
            center: [550000, 6520000],
            projection: projection
        })
    });
    map.getView().setZoom(8);
    var vectorSource = new ol.source.ServerVector({
        format: new ol.format.GeoJSON(),
        loader: function (extent, resolution, projection) {
```

```
            var url = 'http://loom-gis.geo.ut.ee:8080/geoserver/ermas/ows?service=WFS&'
                +
                    'version=1.1.0&request=GetFeature&typeName=ermas:testdata_geopnt&' +
                    'outputFormat=text/javascript&format_options=callback:loadFeatures&
                        maxFeatures=300&' +
                    'srsname=EPSG:3301&EPSG:3301';
            $.ajax({
                url: url,
        dataType: 'jsonp'
            });
        }
    });
    var vectorLayer = new ol.layer.Vector({
        source: vectorSource
    });
    var loadFeatures = function (response) {
        vectorSource.addFeatures(vectorSource.readFeatures(response));
    };
    map.addLayer(vectorLayer);
```

2.3 GeoJSON

```
proj4.defs("EPSG:3301", "+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
    =57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84
    =0,0,0,0,0,0,0 +units=m +no_defs");
    var projection = ol.proj.get('EPSG:3301');
    var map = new ol.Map({
        target: document.getElementById('map'),
        view: new ol.View({
            center: [550000, 6520000],
            projection: projection
        })
    });
    map.getView().setZoom(8);
    var vectorLayer = new ol.layer.Vector({
        source: new ol.source.GeoJSON({
            object: features
        })
    });
    map.addLayer(vectorLayer);
```

3 Leaflet 0.7.3

3.1 WMS

```
var crs = new L.Proj.CRS('EPSG:3301',
                    '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0
                        =57.51755393055556 +lon_0=24 +x_0=500000 ' +
                    '+y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs'
                        ,
            {
                resolutions: [
                    2048, 1024, 512, 256, 128,
                    64, 32, 16, 8, 4, 2, 1, 0.5
                ]
            });
    var map = L.map('map',
            {
                crs: crs
            }
    );
    map.setView([58.66, 25.05], 2);
    var wms = L.tileLayer.wms('http://kaart.maaamet.ee/wms/alus', {
        layers: 'MA-ALUS',
        continuousWorld: true
    });
    wms.addTo(map);
```

3.2 WFS

```
var proj1 = '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0=57.51755393055556 +lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs',
        proj2 = '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs ';
    var crs = new L.Proj.CRS('EPSG:3301',
        proj1,
        {
            resolutions: [
                2048, 1024, 512, 256, 128,
                64, 32, 16, 8, 4, 2, 1, 0.5
            ]
        });
    var map = L.map('map',
        {
            crs: crs,
            center: L.latLng(58.66, 25.05),
            zoom: 2
        }
    );
    map.setZoom(2);
    var geojsonLayer = new L.geoJson();
    function getJson(data) {
        for (var i = 0; i < data.features.length; i++) {
            var convert = proj4(proj1, proj2, data.features[i].geometry.coordinates);
            data.features[i].geometry.coordinates = convert;
        }
        geojsonLayer.addData(data);
    }
    $.ajax({
        url: "http://loom-gis.geo.ut.ee:8080/geoserver/ermas/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=ermas:testdata_geopnt&maxFeatures=300&outputFormat=text/javascript",
        dataType: 'jsonp',
        jsonpCallback: 'parseResponse',
        success: getJson
    });
    geojsonLayer.addTo(map);
```

3.3 GeoJSON

```
var crs = new L.Proj.CRS('EPSG:3301',
                    '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0=57.51755393055556 +lon_0=24 +x_0=500000 ' +
                    '+y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs',
            {
                resolutions: [
                    2048, 1024, 512, 256, 128,
                    64, 32, 16, 8, 4, 2, 1, 0.5
                ]
            });
    var map = L.map('map',
            {
                crs: crs
            }
    );
    map.setView([58.66, 25.05], 2);
    var vectorLayer = L.Proj.geoJson(features, {
        'pointToLayer': function (feature, latlng) {
            return L.marker(latlng);
        }
    });
    vectorLayer.addTo(map);
```

4 Google Maps JavaScript API v3 3.018

4.1 WMS

```
    var map;
    var wmsMapType;
    function initialize() {
map = new google.maps.Map(document.getElementById('map'),
                {
                    center: new google.maps.LatLng(59, 24)
                }
        );
        map.setZoom(7);
        var wmsOptions = {
            getTileUrl: WMSGetTileUrl2,
            tileSize: new google.maps.Size(256, 256)
        };
        wmsMapType = new google.maps.ImageMapType(wmsOptions);
        map.overlayMapTypes.insertAt(0, wmsMapType);
        function WMSGetTileUrl2(coord, zoom) {
            var proj = map.getProjection();
            var zfactor = Math.pow(2, zoom);
            var top = proj.fromPointToLatLng(
                    new google.maps.Point(coord.x * 256 / zfactor, coord.y * 256 /
                        zfactor));
            var bot = proj.fromPointToLatLng(
                    new google.maps.Point((coord.x + 1) * 256 / zfactor, (coord.y + 1) *
                         256 / zfactor));
            var deltaX = 0.0013;
            var deltaY = 0.00058;
            var bbox = (top.lng() + deltaX) + "," + (bot.lat() + deltaY) + "," + (bot.
                lng() + deltaX) + "," + (top.lat() + deltaY);
            var url = 'http://kaart.maaamet.ee/wms/alus-geo?';
            url += '&REQUEST=GetMap';
            url += '&SERVICE=WMS';
            url += '&VERSION=1.1.1';
            url += '&LAYERS=MA-ALUS-GEO';
            url += '&FORMAT=image/png';
            url += '&SRS=EPSG:4326';
            url += '&BBOX=' + bbox;
            url += '&WIDTH=256';
            url += '&HEIGHT=256';
            return url;
        }
    }
    google.maps.event.addDomListener(window, 'load', initialize);
```

4.2 WFS

```
var proj1 = '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0=57.51755393055556 +
    lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +
    no_defs',
            proj2 = '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs ';
    var map;
    function initialize() {
        map = new google.maps.Map(document.getElementById('map'),
                {
                    center: new google.maps.LatLng(59, 24)
                }
        );
        map.setZoom(7);
        var vectorLayer = new google.maps.Data({map: map});
        $.ajax({
            url: "http://loom-gis.geo.ut.ee:8080/geoserver/ermas/ows?service=WFS&version
                =1.0.0&request=GetFeature&typeName=ermas:testdata_geopnt&maxFeatures
                =300&outputFormat=text/javascript",
            dataType: 'jsonp',
            jsonpCallback: 'parseResponse',
            success: function (data) {
                for (var i = 0; i < data.features.length; i++) {
```

```
                    var convert = proj4(proj1, proj2, data.features[i].geometry.
                        coordinates);
                    data.features[i].geometry.coordinates = convert;
                }
                vectorLayer.addGeoJson(data);
            }
        });
    }
    google.maps.event.addDomListener(window, 'load', initialize);
```

4.3 GeoJSON

```
var proj1 = '+proj=lcc +lat_1=59.33333333333334 +lat_2=58 +lat_0=57.51755393055556 +
    lon_0=24 +x_0=500000 +y_0=6375000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +
    no_defs',
        proj2 = '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs ';
    var map;
    var vectorLayer;
    function initialize() {
        map = new google.maps.Map(document.getElementById('map'),
                {
                    center: new google.maps.LatLng(59, 24)
                }
        );
        map.setZoom(7);
        vectorLayer = new google.maps.Data();
        vectorLayer.setMap(map);
        function convert(data) {
            for (var i = 0; i < data.features.length; i++) {
                var convert = proj4(proj1, proj2, data.features[i].geometry.coordinates)
                    ;
                data.features[i].geometry.coordinates = convert;
            }
            return data
        }
        vectorLayer.addGeoJson(convert(features));

    }
    google.maps.event.addDomListener(window, 'load', initialize);
```

5 ArcGIS API for JavaScript 3.12

5.1 WMS

```
    var map;
    require([
        'esri/map', 'esri/layers/WMSLayer', 'esri/layers/WMSLayerInfo', 'esri/geometry/
            Extent',
        'dojo/_base/array', 'dojo/dom', 'dojo/dom-construct', 'dojo/parser',
        'dijit/layout/BorderContainer', 'dijit/layout/ContentPane', 'dojo/domReady!'
    ], function (Map, WMSLayer, WMSLayerInfo, Extent, array, dom, domConst, parser) {
        var map = new esri.Map("map",
                {center: new esri.geometry.Point(550000, 6520000, new esri.
                    SpatialReference({wkid: 3301}))});
        map.setZoom(4);
        var wmsLayer = new WMSLayer('http://kaart.maaamet.ee/wms/alus?', {
            resourceInfo: {
                extent: new Extent(350000, 6370000, 750000, 6630000, {
                    wkid: 3301
                }),
                layerInfos: new WMSLayerInfo({
                })
            },
            visibleLayers: ['MA-ALUS']
        });
        map.addLayers([wmsLayer]);
    });
```

5.2 WFS

```
var map;
   require([
       'esri/map', "esri/layers/FeatureLayer", "esri/SpatialReference", 'esri/layers/
           WMSLayer', 'esri/layers/WMSLayerInfo', 'esri/geometry/Extent',
       'dojo/_base/array', 'dojo/dom', 'dojo/dom-construct', 'dojo/parser', "esri/
           request",
       'dijit/layout/BorderContainer', 'dijit/layout/ContentPane', "dojo/_base/json" ,
           'dojo/domReady!'
   ], function (Map, FeatureLayer, SpatialReference, WMSLayer, WMSLayerInfo, Extent,
       array, dom, domConst, parser, request) {
       map = new Map("map",
               {
                   extent: new Extent({"xmin": 300000, "ymin": 6.3e+06, "xmax": 800000,
                        "ymax": 6.7e+06, "spatialReference": {"wkid": 3301}}), center:
                       new esri.geometry.Point(550000, 6520000, new esri.
                       SpatialReference({ wkid: 3301 }))
               });
       var layerDefinition = {
           "extent": {
               "xmin": 350000,
               "ymin": 6370000,
               "xmax": 750000,
               "ymax": 6630000,
               "spatialReference": {
                   "wkid": 3301
               }
           },
           "fields": [
               {
                   "name": "OBJECTID",
                   "type": "esriFieldTypeOID"
               }
           ]
       };
       var featureCollection = {
           layerDefinition: layerDefinition
       };
       var vectorLayer = new FeatureLayer(featureCollection);

       map.addLayers([vectorLayer]);
       var layerUrl = 'http://loom-gis.geo.ut.ee:8080/geoserver/ermas/ows?service=WFS&'
            +
               'version=1.1.0&request=GetFeature&typeName=ermas:testdata_geopnt&' +
               'outputFormat=text/javascript&format_options=callback:loadFeatures&
                   maxFeatures=300&' +
               'srsname=EPSG:3301&EPSG:3301';
       var loadFeatures = function (response) {
           for (var i = 0; i < response.features.length; i++) {
               var geometry = new esri.geometry.Point(response.features[i].geometry.
                   coordinates[0], response.features[i].geometry.coordinates[1], new
                   esri.SpatialReference({ wkid: 3301 }));
               var symbol = new esri.symbol.SimpleMarkerSymbol();
               var graphic = new esri.Graphic(geometry, symbol);
               vectorLayer.add(graphic)
           }
       };
       window.loadFeatures = loadFeatures;
       var layersRequest = request({
           url: layerUrl,
           callbackParamName: "loadFeatures"
       });
       map.setZoom(0);
   });
```

5.3 GeoJSON

```
var map;
   require([
       'esri/map', "esri/layers/FeatureLayer", "esri/tasks/FeatureSet", 'esri/layers/
           WMSLayer', 'esri/layers/WMSLayerInfo', 'esri/geometry/Extent',
       'dojo/_base/array', 'dojo/dom', 'dojo/dom-construct', 'dojo/parser',
       'dijit/layout/BorderContainer', 'dijit/layout/ContentPane', 'dojo/domReady!'
   ], function (Map, FeatureLayer, FeatureSet, WMSLayer, WMSLayerInfo, Extent, array,
       dom, domConst, parser) {
       var map = new esri.Map("map",
               {
                   center: new esri.geometry.Point(550000, 6520000, new esri.
                       SpatialReference({ wkid: 3301 }))
               });
       var layerDefinition = {
           "extent": {
               "xmin": 350000,
               "ymin": 6370000,
               "xmax": 750000,
               "ymax": 6630000,
               "spatialReference": {
                   "wkid": 3301
               }
           },
           "fields": [
               {
                   "name": "OBJECTID",
                   "type": "esriFieldTypeOID"
               }
           ]
       };
       var featureCollection = {
           layerDefinition: layerDefinition
       };
       var featureLayer = new esri.layers.FeatureLayer(featureCollection);
       map.addLayers([featureLayer]);
       for (var i = 0; i < points.features.length; i++) {
           var geometry = new esri.geometry.Point(points.features[i].geometry.
               coordinates[0], points.features[i].geometry.coordinates[1], new esri.
               SpatialReference({ wkid: 3301 }));
           var symbol = new esri.symbol.SimpleMarkerSymbol();
           var graphic = new esri.Graphic(geometry, symbol);
           featureLayer.add(graphic)
       }
       map.setZoom(0);
   });
```

LISA 5

1 OpenLayers 2.11

WMS			GeoJSON 300			GeoJSON 3000			GeoJSON 30 000		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
374,455	320,000	307,112	45,815	30,000	37,824	168,388	120,325	228,728	1353,318	861,000	1823,161
397,550	266,000	284,134	44,717	31,000	37,564	173,130	115,000	202,968	1141,675	810,000	1879,872
399,133	308,000	261,478	45,808	29,000	38,665	157,404	108,000	209,745	1313,841	790,000	1886,929
389,590	264,000	269,359	45,783	35,000	39,028	161,468	125,000	206,864	1280,772	800,000	1882,067
413,386	313,000	299,342	44,918	30,000	39,086	158,656	114,965	205,785	1328,832	878,000	1886,664
373,298	279,000	272,895	43,308	31,000	38,511	159,072	129,000	206,136	1127,685	795,000	1851,777
395,447	323,000	272,916	44,897	34,000	37,610	158,129	159,000	203,229	1325,878	781,000	1943,077
393,561	275,000	264,677	44,232	33,000	37,448	155,778	111,785	222,368	1379,081	770,000	1881,830
417,994	259,000	266,891	44,803	33,000	41,485	154,645	127,853	221,915	1336,402	772,000	1932,589
414,896	256,000	340,007	44,193	30,000	37,328	154,146	156,000	222,399	1312,952	820,000	1879,691
393,072	272,000	291,553	44,074	37,000	40,005	159,338	108,000	208,937	1182,839	727,000	1899,313
437,955	287,000	303,044	46,005	30,000	39,863	156,536	154,995	212,508	1338,292	726,000	1914,012
409,741	255,000	458,005	45,188	31,000	41,000	154,060	110,000	246,913	1192,650	708,000	1908,933
502,252	283,000	286,336	45,087	31,000	39,007	155,141	134,955	214,434	1170,826	778,000	1887,877
378,237	260,000	395,423	44,366	33,000	41,108	155,541	119,000	241,026	1232,348	769,000	1906,663
368,211	255,000	580,734	43,418	31,000	40,314	158,213	115,000	219,350	1378,010	768,000	1879,550
455,417	251,000	393,478	45,002	30,000	40,487	160,453	122,432	243,075	1215,901	784,000	1901,188
467,167	258,000	277,927	45,493	29,000	40,090	156,193	129,000	225,353	1248,999	786,000	1994,083
389,878	284,000	320,411	44,031	31,000	44,058	156,593	113,000	248,102	1261,680	784,000	1989,230
448,000	464,000	284,445	44,083	34,000	41,875	154,218	108,692	241,715	1248,600	770,000	1971,331
492,691	235,000	267,368	44,225	31,000	42,230	156,017	123,000	254,850	1347,179	853,000	1947,453
361,598	267,000	291,464	45,597	29,000	44,352	158,469	138,000	215,473	1161,038	809,000	1997,188
398,072	297,000	290,132	44,263	29,000	37,326	155,347	126,000	221,911	1305,730	839,000	1860,102
496,078	254,000	296,135	45,919	31,000	39,066	158,581	115,965	216,755	1206,145	708,000	1916,792
457,057	249,000	269,270	45,842	31,000	44,029	161,368	118,000	245,829	1254,569	777,000	1940,933
486,068	388,000	289,498	44,227	35,000	41,028	158,069	118,000	235,417	1310,483	815,000	1928,135
456,173	261,000	300,424	46,120	45,000	39,774	154,794	120,033	210,833	1252,640	779,000	1867,444
385,783	253,000	273,508	44,946	31,000	40,745	157,846	120,592	267,832	1279,835	797,000	1863,582
379,722	302,000	288,125	44,839	34,000	41,468	154,515	117,655	224,163	1288,086	750,000	1975,073
515,720	241,000	330,485	45,360	45,000	44,206	154,744	126,000	226,451	1239,504	740,000	2086,392

WFS 300			**WFS 3000**			**WFS 30 000**		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
396,671	348,318	401,1592007	1648,932	1655,845	2147,624629	13337,728	13187,104	15418,307
396,041	299,321	352,7023907	1625,746	1474,003	2500,523857	12332,649	11875,461	19306,834
368,229	350,280	367,0229738	1899,776	1516,372	2852,921625	14006,479	12490,698	21116,000
420,197	322,504	379,6203683	1802,268	1991,028	2294,96943	12115,404	12472,243	19803,323
371,519	633,750	325,6828823	1605,750	1909,465	3630,898379	14170,813	11587,683	19119,586
403,043	297,138	339,1275383	1680,842	1910,866	3111,475188	12617,024	12390,814	21718,470
386,004	362,160	345,6106736	1628,830	1502,978	2921,063813	14820,738	12157,873	21718,216
383,444	362,448	400,3058614	1733,142	1513,624	3003,967262	12890,321	14606,855	18698,217
377,568	345,436	454,2522519	1894,041	1479,376	2767,757738	16000,053	12469,418	22137,888
408,722	335,836	402,8690014	1473,675	1533,700	2586,492028	11572,361	13663,024	18987,608
391,241	369,673	425,6994387	1682,575	1459,970	2524,466544	16152,526	11527,167	23589,492
396,208	482,217	440,7472486	1717,883	1734,675	2578,353031	11668,497	11998,146	20188,168
383,574	394,257	421,4366774	1651,935	1683,718	2779,38049	12262,914	12258,082	20185,786
439,566	310,603	432,8311786	1666,731	1531,445	3433,362632	12585,552	11724,413	19875,347
405,700	358,336	401,382661	1621,669	1792,718	3531,02537	12109,448	12047,278	19735,483
439,609	346,059	404,0339681	1813,171	1499,326	2959,796668	14327,828	11663,854	21042,578
413,105	388,331	424,0773554	1634,443	1530,687	3239,010992	12120,367	12210,050	19551,459
398,242	324,717	458,8954181	1652,688	1563,606	2931,502294	11327,779	15001,324	19085,311
399,225	317,299	470,9506751	1763,531	1602,320	3218,035996	16489,427	12117,021	21998,089
407,019	359,557	438,8378835	1674,018	1560,115	3269,477955	12237,312	11478,202	20320,670
417,734	366,934	450,8566399	1679,173	1605,168	3225,82836	11298,849	12063,973	21526,377
380,089	344,777	427,2968834	1714,996	1483,191	2815,852626	12583,001	12076,713	23658,969
367,606	349,834	436,2761621	1655,864	1488,465	2821,487721	17260,712	13995,864	24538,082
384,029	343,297	450,623045	1674,069	1466,942	2755,916955	12693,162	11889,822	22187,828
435,874	343,332	455,6314204	1723,500	1556,537	3442,728212	17275,343	12257,625	22486,525
374,285	323,489	479,2005444	1696,877	1468,523	3029,78977	12339,454	11224,035	19825,238
385,346	340,231	400,7221364	1746,841	1492,614	2921,729802	13275,818	11441,440	20791,675
395,380	376,797	395,2440473	1599,119	1461,297	3045,504972	16978,394	12211,417	22722,419
404,786	324,521	495,79947	1663,326	1482,841	2925,203499	13997,501	11341,416	22462,911
353,218	331,042	425,201172	1670,487	1511,304	2816,15856	16172,215	14462,791	22221,845

2 OpenLayers 3.2.1

WMS			GeoJSON 300			GeoJSON 3000			GeoJSON 30 000		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
1240,981	1007,000	1104,491	154,245	80,127	172,109	2474,492	387,452	1340,704	8069,750	4399,200	14733,137
1135,640	1999,000	1051,133	153,677	82,000	175,522	2566,991	422,160	1326,276	8136,402	4381,202	15314,971
1707,776	1033,000	1044,800	148,863	71,530	153,865	2654,864	487,325	1395,711	8534,274	4272,852	14793,046
1194,868	1088,000	981,671	159,201	74,029	164,790	2658,172	411,420	1393,638	8088,792	4774,884	15464,677
1208,295	1781,000	1059,675	142,395	72,198	186,399	2485,479	418,369	1336,864	7881,829	4165,386	14712,350
1177,857	2255,000	1030,105	155,244	78,991	175,219	2528,215	411,190	1348,229	8540,169	4281,278	15011,889
1179,407	1999,000	1194,182	147,122	72,997	187,612	2487,196	402,199	1313,504	8514,469	4277,485	14622,974
1269,194	1983,000	1020,265	141,032	90,116	184,748	2563,406	400,195	1356,275	8062,653	3986,679	15406,510
1261,952	1052,000	1017,970	144,243	71,000	178,704	2427,472	414,194	1310,124	7932,711	3730,737	15136,396
1234,924	1262,000	1556,596	145,318	71,015	175,305	2402,419	421,961	1374,084	8219,076	3977,710	14775,928
1215,538	1083,000	1083,136	145,659	72,006	185,144	2544,881	434,529	1342,933	8080,978	4209,669	15096,731
1177,161	1158,000	1001,518	146,388	74,645	171,539	2465,638	420,469	1394,142	8062,513	3779,806	15228,556
1191,254	1627,000	993,052	147,157	80,972	184,247	2489,689	418,416	1344,902	8007,203	3664,151	14767,364
1191,254	1576,000	1007,854	146,681	83,850	172,697	2516,032	425,129	1364,046	8383,161	4318,203	15086,691
1606,422	1049,000	1038,009	152,749	75,322	182,887	2640,878	425,169	1359,270	8183,859	4103,657	14926,595
1190,100	1051,000	969,126	142,536	70,550	174,636	2539,342	420,240	1309,898	8232,443	4082,447	17606,694
1239,136	1562,000	1015,551	146,340	79,997	180,183	2623,541	418,000	1319,242	7910,448	4782,604	14881,683
1030,557	1736,000	1019,089	144,647	71,944	196,969	2607,168	411,196	1321,676	7867,111	3604,531	15053,235
1182,600	1053,000	1006,382	145,788	69,410	167,515	2662,030	435,490	1369,006	8149,720	3847,038	14896,572
1216,173	1094,000	1043,918	147,645	85,000	181,336	2602,543	425,585	1350,862	7828,303	4599,524	14691,957
1210,795	1065,000	1097,126	157,723	89,650	185,179	2584,126	415,366	1334,947	7836,021	4387,536	14496,606
1177,519	1752,000	1016,500	145,150	81,961	181,473	2401,014	415,589	1316,464	8035,463	4191,009	14846,116
1139,654	1069,000	1024,026	147,887	84,008	175,774	2561,670	395,496	1363,489	8051,091	4252,681	14821,103
1247,725	1079,000	991,142	144,331	71,069	191,570	2409,958	405,103	1341,492	7959,150	4296,118	15333,941
1840,318	1072,000	1037,684	145,820	74,620	174,900	2604,762	410,558	1347,074	7869,200	4102,423	14966,887
1698,314	1813,000	1051,618	147,439	74,000	176,953	2596,924	411,258	1321,819	8349,476	4816,842	14736,001
1219,427	1061,000	1098,728	143,004	84,012	181,975	2584,564	455,156	1346,551	7722,308	3739,871	15026,547
1990,069	1235,000	970,482	158,754	84,814	195,737	2402,354	402,127	1323,994	7843,060	4359,129	14739,709
1224,490	1095,000	1044,787	142,220	76,000	192,071	2577,199	422,310	1317,457	8597,321	4252,489	14881,672
1312,065	1436,000	1025,690	147,366	71,361	173,245	2555,133	423,235	1358,973	8140,551	4306,698	15289,390

WFS 300			**WFS 3000**			**WFS 30 000**		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
324,408	305,000	342,170	1529,654	992,217	1910,154	13255,010	10229,601	21268,249
401,920	281,000	345,163	1517,400	966,133	1868,599	13737,369	9938,218	22932,665
370,148	245,568	307,284	1534,182	913,169	2015,798	13956,877	10154,545	20748,127
347,974	318,000	315,673	1459,683	923,189	1953,662	15891,899	10259,010	21036,844
452,527	324,785	329,043	1646,526	928,165	1910,820	14245,098	9682,691	22783,336
450,966	282,000	341,497	1393,307	974,477	1935,649	16379,189	10307,786	21497,965
335,760	234,787	295,622	1384,257	885,465	1875,609	15847,210	10862,009	24510,987
429,438	277,000	311,008	1405,307	885,126	1964,769	15797,097	10003,862	22824,767
357,676	330,000	311,643	1417,520	913,165	1949,230	14379,114	11090,038	22289,116
369,135	300,559	316,295	1473,890	982,169	1886,505	16139,813	9857,646	22341,532
423,073	277,000	320,385	1487,309	945,169	2003,971	16477,236	10022,899	24041,434
410,617	317,000	299,082	1559,557	923,165	2022,156	16430,291	11039,019	20629,802
405,617	250,000	279,959	1442,532	888,100	1976,193	16055,076	9718,244	23029,834
421,814	322,754	323,916	1437,158	958,165	1973,623	14180,326	12211,125	21400,701
357,812	319,786	343,691	1452,130	868,169	1919,000	14368,581	10235,538	21293,311
380,462	247,000	291,492	1419,218	923,900	1822,793	15575,045	10999,000	21022,420
381,397	334,000	303,671	1427,464	924,198	1887,032	14615,440	9466,737	20644,359
416,255	258,987	307,175	1434,577	902,600	1932,784	15015,892	10625,327	22955,678
366,581	260,000	291,260	1433,275	899,132	1959,160	18260,234	11006,516	21198,222
397,673	254,000	301,355	1418,997	939,469	1970,748	16998,918	9129,989	21970,701
426,934	340,000	289,169	1397,620	994,490	2037,465	13855,126	10487,127	24694,881
372,788	270,000	307,889	1361,920	938,225	1830,819	15371,493	10344,798	23673,813
389,007	254,965	323,659	1408,377	965,350	1865,638	18335,970	10402,092	22383,476
397,447	278,078	316,397	1486,940	944,235	1953,649	14150,211	10351,983	21367,943
372,245	261,012	259,165	1441,048	902,528	1966,230	14584,263	9564,117	23148,215
370,176	290,190	431,989	1449,727	899,222	1941,784	15470,706	10371,481	23295,077
414,729	316,916	311,744	1450,227	910,220	1879,594	13995,136	10338,705	22922,667
373,702	293,251	331,450	1452,240	912,132	1861,896	19490,344	10359,227	21220,519
371,415	284,000	22,609	1426,564	915,640	1962,419	16423,242	11032,470	21483,197
363,165	316,000	318,676	1449,286	913,450	1945,138	15131,112	10515,628	20786,969

3 Leaflet 0.7.3

WMS			GeoJSON 300			GeoJSON 3000			GeoJSON 30 000		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
786,781	502,000	378,284	158,109	118,000	172,571	1227,611	759,985	1915,095	100122,446	6465,246	25710,240
616,977	426,000	392,102	160,873	118,978	168,177	1291,399	818,210	2195,282	102869,012	6394,848	37641,813
645,340	439,000	395,669	158,955	122,989	166,762	1279,581	792,000	2178,512	108885,084	7037,206	50268,952
640,573	474,000	394,912	162,055	124,116	200,890	1323,118	728,060	2268,138	115825,599	7595,769	50685,726
809,420	475,000	381,714	179,000	122,956	172,224	1268,502	936,890	2255,533	102589,235	6917,716	50980,976
621,180	528,000	951,635	170,810	115,967	159,659	1344,780	748,001	1987,357	102587,359	7895,528	29985,390
680,386	461,000	373,396	159,492	132,005	163,021	1373,723	836,971	2006,223	105872,236	7828,790	38129,350
937,224	466,000	376,857	155,562	125,470	171,841	1232,791	721,024	2103,207	102589,365	7525,096	50021,363
631,332	486,000	437,258	165,794	126,807	159,420	1394,272	950,017	2280,017	112587,359	7026,389	45258,354
573,814	704,000	392,378	153,615	125,167	166,701	1356,027	878,032	1893,003	117852,360	7502,930	45125,365
657,418	490,000	371,772	152,024	126,923	156,869	1370,902	896,547	2298,653	102587,222	7696,748	40025,203
654,418	453,000	368,897	156,988	132,970	172,276	1451,457	1002,610	2257,033	100028,369	7217,799	40158,230
885,908	474,000	382,919	169,199	124,032	170,024	1208,443	925,000	2311,259	116690,357	7742,577	38002,020
719,115	438,000	390,579	152,618	126,792	162,460	1306,581	894,390	2280,492	110177,258	8189,258	47258,330
694,165	472,000	452,174	154,180	124,007	160,857	1332,608	898,000	2298,721	102456,336	7171,142	50225,220
682,980	478,000	380,961	153,273	136,000	184,465	1211,038	948,003	2222,138	109229,058	8060,702	50987,983
675,425	506,000	369,433	165,599	125,017	201,442	1295,154	822,048	2277,665	115879,394	6789,025	50357,350
691,536	493,000	372,574	164,005	124,900	177,404	1228,009	873,961	2323,326	103582,002	6999,025	43358,350
661,849	481,000	381,960	160,891	126,072	167,988	1277,836	973,700	2236,477	105258,520	7085,250	38369,230
704,444	474,000	461,146	164,090	128,000	175,120	1260,915	690,500	2238,815	104785,000	7198,614	40363,335
853,915	531,000	371,571	159,744	135,036	162,518	1340,905	876,860	2309,048	110369,025	7199,369	40325,369
726,006	484,000	374,818	160,873	125,345	161,301	1288,764	882,000	2320,005	110022,369	7698,390	45969,025
685,885	478,000	377,678	176,742	126,214	166,547	1254,682	823,130	2290,916	107778,223	7320,520	40366,390
730,543	468,000	425,606	157,820	124,251	167,833	1274,818	920,597	2331,906	100155,334	7888,055	45999,025
614,175	552,000	363,454	157,773	124,997	183,603	1313,391	813,998	2290,253	100115,147	8022,980	40920,025
859,891	542,000	379,483	148,584	126,520	159,404	1350,615	957,460	2295,043	102966,330	8017,436	50202,200
586,860	656,000	372,397	171,697	128,000	165,245	1345,585	927,560	2299,564	103001,200	7663,495	50112,288
638,025	498,000	374,962	162,420	129,294	162,403	1294,550	836,058	2305,949	111963,215	7636,902	50111,983
718,296	559,000	383,369	169,726	131,000	171,189	1216,984	860,325	2302,366	117283,500	7748,181	45238,350
670,005	596,000	439,831	167,201	126,970	175,134	1354,073	859,335	2247,036	108389,390	7888,390	50223,180

WFS 300			**WFS 3000**			**WFS 30 000**		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
329,619	239,000	352,945	2191,725	2016,089	2699,265	64574,032	21100,528	34592,641
322,608	276,709	340,113	2456,296	3273,808	3925,319	69893,236	20530,335	35838,423
366,243	254,282	369,748	2458,300	1735,546	3754,876	78963,235	22082,509	63897,464
329,314	267,761	352,042	2204,158	1599,698	3779,355	74258,036	18381,804	61759,005
321,021	254,368	380,001	2108,437	1575,191	3179,306	89325,128	20982,402	62516,951
321,389	260,694	379,463	2598,753	1652,391	3800,157	60995,268	19462,386	62084,504
313,016	266,335	360,453	2165,198	1599,376	3752,061	78955,258	18930,119	62662,962
324,285	236,508	353,652	2070,741	1474,331	3719,703	69393,466	20272,628	60365,657
312,867	284,359	371,612	2169,693	1394,520	3840,268	90255,368	21174,591	66976,647
308,095	255,314	382,578	2047,549	1684,766	3766,219	88485,015	20036,448	61823,358
307,187	299,940	354,555	2041,924	1554,579	3888,857	88718,158	22168,418	61752,058
311,454	263,531	381,810	2165,688	1529,877	3255,418	97420,366	20447,842	61088,256
328,668	240,943	354,165	2190,356	1550,260	3473,109	78958,089	18660,560	62555,554
321,398	261,358	342,060	2560,390	1543,356	3782,236	68924,032	23403,774	62357,106
354,369	255,639	370,150	2498,208	1608,352	3745,359	90048,025	18636,777	62780,325
330,280	268,359	375,893	2111,356	1643,050	3702,002	75586,369	21863,962	63083,250
320,289	255,640	355,260	2133,260	1568,550	3800,238	78896,300	23244,636	62666,136
321,286	260,052	362,350	2143,300	1539,214	3715,256	74431,548	20024,402	62705,000
322,389	263,998	354,389	2089,356	1569,240	3688,610	80258,001	19533,945	62725,551
315,890	267,388	367,357	2103,356	1600,185	3825,236	85802,708	18759,098	62195,248
325,150	280,569	368,230	2050,368	1535,350	3755,238	84258,025	21130,859	61722,208
313,982	276,998	367,235	2083,390	1563,335	3899,238	70658,883	19266,632	61068,260
324,258	260,550	379,300	2112,326	1568,230	3730,250	68958,222	19289,552	61826,120
321,999	280,216	360,327	2204,130	1553,300	3760,265	98686,125	17747,540	61555,150
310,250	265,260	361,066	2105,133	1603,200	3800,565	68988,126	15280,737	61725,213
310,286	245,260	354,150	2028,399	1486,260	3625,230	78987,238	20891,633	61223,000
314,350	262,163	352,239	2075,853	1555,239	3905,350	77895,288	21240,864	65358,000
320,620	260,562	353,138	2068,690	1532,235	3777,136	98585,128	18434,415	59068,090
314,289	260,150	352,100	2209,230	1530,225	3689,266	68989,014	20675,888	58068,650
316,390	263,025	361,326	2167,526	1530,230	3702,585	69878,879	22792,170	59025,258

4 Google Maps JavaScript API v3 3.018

WMS			GeoJSON 300			GeoJSON 3000			GeoJSON 30 000		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
5391,730	5181,000	5536,274	70,130	52,348	60,458	361,782	258,953	439,592	3155,676	2240,410	5575,739
5683,979	5577,000	5499,869	60,001	54,819	74,645	384,096	464,617	458,308	3330,469	2256,465	4285,579
5743,092	5351,000	5683,033	63,665	65,158	62,002	390,327	268,308	502,696	3008,811	2256,655	5015,809
5655,412	7744,000	5777,460	66,716	58,177	87,710	376,429	426,307	518,259	2828,602	2280,312	4294,265
5439,435	5726,000	6253,667	64,551	68,799	65,853	412,377	466,564	554,721	2962,291	2269,713	5423,548
5414,238	5447,000	5778,480	61,663	57,153	67,172	371,541	329,166	506,271	3017,553	2394,885	5559,257
5618,084	6335,000	5943,680	65,147	64,511	69,966	403,323	291,733	602,115	2991,015	2280,024	4997,324
4827,206	5973,000	5155,076	66,995	68,164	69,949	410,383	450,353	547,597	3010,586	2316,271	5614,007
5169,488	5764,000	6446,423	86,727	65,258	73,546	389,102	283,253	526,142	3236,759	2359,553	5472,831
4764,679	5297,000	6406,149	62,289	53,750	75,793	392,524	280,825	598,960	3278,878	2302,267	4306,067
5666,657	5812,000	6877,664	77,500	66,344	66,017	368,009	362,013	618,081	3069,311	2331,537	4237,653
5077,391	5751,000	5756,509	66,452	56,355	71,038	412,172	366,651	526,459	3458,081	2295,524	5613,529
4860,764	7626,000	5676,948	66,576	56,429	72,540	387,441	463,084	595,967	3013,027	2230,297	5502,705
5069,914	8336,000	6078,522	66,927	57,264	67,392	388,335	321,062	625,460	3001,303	2269,883	5069,367
5398,046	7324,000	5636,123	69,357	53,050	69,260	373,845	329,235	512,619	3011,269	2347,662	5323,640
4939,743	5036,000	6046,923	61,359	55,055	71,066	362,390	458,536	639,231	2888,920	2307,691	5074,818
4909,880	5331,000	5589,650	62,268	60,156	71,005	370,260	450,268	612,881	2901,892	2307,529	5141,476
5104,960	5474,000	5980,761	67,269	58,529	73,029	380,558	463,266	660,781	3011,630	2282,846	5546,300
5221,532	5544,000	6480,451	65,658	57,058	69,215	378,268	356,566	628,202	3824,082	2338,274	4285,579
5405,661	6504,000	6255,771	64,265	56,158	69,020	371,550	435,566	689,932	2968,126	2317,603	4197,615
5141,719	5581,000	7510,575	70,260	57,000	69,900	382,158	451,250	615,831	2799,878	2234,851	5115,389
4901,404	5707,000	6849,096	67,055	56,259	70,782	379,558	462,250	630,265	2964,774	2406,150	5103,134
4807,569	5448,000	6931,903	65,158	56,998	71,058	383,138	399,255	703,486	3107,904	2270,983	5493,407
5857,753	5348,000	5717,189	66,138	57,020	71,953	389,260	462,150	637,274	3324,025	2328,873	5542,901
4867,535	5573,000	5565,221	66,238	60,150	73,553	387,558	430,146	635,150	3008,610	2272,722	4221,514
5068,667	5908,000	5672,309	66,989	61,080	73,558	388,138	431,150	647,625	3038,318	2259,303	4250,098
5718,569	5765,000	5735,065	68,108	58,050	68,268	375,238	445,250	602,264	3314,046	2297,911	4213,404
5730,902	5436,000	5735,065	63,158	56,058	65,598	400,258	420,565	662,584	3010,127	2287,891	5534,464
4713,731	5278,000	6292,268	65,080	56,998	68,560	391,050	425,468	620,566	3041,691	2353,744	5132,526
4960,409	5607,000	6528,751	63,820	58,058	69,565	389,003	425,188	658,229	3082,746	2466,976	4292,299

WFS 300			**WFS 3000**			**WFS 30 000**		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
325,844	277,079	237,765	1621,685	910,361	1633,784	9360,076	10766,680	9563,123
243,730	261,742	140,659	1310,432	1224,932	1655,286	10803,771	8748,901	16654,999
295,343	267,782	135,769	1353,821	1075,193	1605,703	14090,711	7905,428	9090,961
274,119	289,701	153,183	1731,432	761,429	1569,778	9136,669	9843,971	9852,248
256,500	283,567	145,516	1444,652	778,628	1901,743	14561,069	8351,852	11511,362
248,296	287,748	159,638	1582,253	822,656	1620,589	9786,575	8760,088	10679,806
267,974	296,297	189,799	1705,708	786,930	1740,956	10320,299	8366,531	11300,022
243,302	296,541	154,640	1085,515	808,273	1752,122	11557,894	9562,390	12148,662
255,801	274,862	146,665	1382,936	739,040	1754,921	11230,234	8861,073	12761,451
242,717	285,459	153,781	914,226	906,981	1801,420	10816,770	8679,055	13699,885
375,692	273,908	163,105	1599,209	872,505	1822,312	10065,431	8300,948	13355,414
272,056	273,666	147,771	1593,849	786,472	1865,932	9409,057	9003,203	9037,512
227,931	284,354	152,997	1607,606	746,756	1774,599	12386,647	7972,691	7134,707
239,247	268,289	156,057	1622,817	806,619	2011,363	12864,362	8306,143	11312,086
258,037	429,597	155,117	1417,947	728,342	1775,719	10844,332	8159,773	9360,522
244,526	272,632	151,138	1293,505	738,988	1765,157	15481,140	7888,478	18999,312
204,877	285,473	153,978	1665,831	737,871	1778,267	14080,479	10395,239	18482,826
253,330	280,469	148,277	1776,793	796,445	1822,574	26164,443	8355,605	15928,174
218,719	287,227	153,098	1638,962	752,120	1800,845	11452,588	8626,506	16564,927
258,791	279,174	157,442	989,545	816,902	1866,404	10206,277	8397,755	15301,788
221,617	257,148	156,061	1365,629	747,627	1849,365	28757,836	8139,910	27526,382
251,771	272,691	160,661	2571,315	750,544	1902,165	17039,876	8549,837	15067,531
245,457	282,082	257,970	1029,703	815,839	1791,954	21700,928	8074,838	14630,746
215,470	324,170	157,451	1321,936	726,803	1755,164	8658,522	8533,093	13551,119
251,374	261,679	165,233	1686,590	735,025	1785,265	18367,435	8072,244	14947,907
220,821	273,397	164,630	1655,415	798,600	1800,289	13781,459	8443,018	19512,761
243,527	260,561	159,830	1706,486	731,874	1850,160	9640,775	7972,855	10502,289
214,185	324,371	174,200	2510,787	829,042	1850,168	14231,449	8995,958	12306,943
286,437	273,337	166,814	1631,422	787,453	1562,166	16330,637	9187,142	14583,355
217,065	285,681	282,067	1335,727	696,917	1777,158	12757,154	8310,417	13298,201

5 ArcGIS API for JavaScript 3.12

WMS			GeoJSON 300		GeoJSON 3000			GeoJSON 30 000		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
1903,942	1958,000	1792,118	258,204	318,310	392,858	735,079	624,695	2387,922	11757,044	11787,546	21682,405
2102,887	1935,000	1648,729	271,619	308,639	457,046	848,784	720,823	1353,621	12531,147	10179,413	30063,268
2110,646	1925,000	1778,502	332,486	461,937	388,870	780,642	672,420	2439,241	12059,487	9892,149	21928,350
2022,442	1912,000	1786,889	278,311	314,265	484,953	812,788	667,030	2565,944	12730,198	10913,613	21974,629
2145,882	1909,000	1882,965	321,042	356,530	464,897	814,037	664,416	1411,386	23085,043	9839,675	30273,547
2118,086	1926,000	1842,593	287,309	320,297	461,411	791,238	730,964	2499,888	11736,491	10746,373	22448,830
2073,433	2001,000	1946,576	258,241	309,684	389,353	793,918	688,152	2408,418	13385,862	10345,724	22294,096
2236,924	2436,000	1802,618	322,554	418,416	499,178	790,720	669,152	2339,375	12780,540	9798,668	22586,326
2143,480	1936,000	1916,117	273,221	310,689	459,420	775,482	823,532	3300,372	13356,124	9887,098	29358,256
3017,982	1966,000	1841,986	265,760	311,200	488,255	784,717	666,726	2533,258	11555,087	9774,115	22035,529
2134,909	1998,000	1685,269	261,087	329,097	391,229	802,249	675,324	2403,341	12971,231	10134,209	22033,250
2104,482	1952,000	1668,553	328,940	331,974	479,047	787,819	821,260	2464,771	12187,022	10434,139	28996,350
2059,039	1928,000	1684,481	275,284	316,116	454,513	814,601	710,081	2329,311	11522,897	10068,042	23058,200
2033,887	2015,000	1892,345	268,796	333,131	389,710	798,598	811,523	2519,585	11574,452	10165,868	21039,600
2321,312	1995,000	1948,019	283,826	305,961	483,680	812,400	664,601	1421,659	11660,060	10085,312	30316,325
1991,976	2055,000	2780,321	302,291	322,882	477,086	800,432	662,475	2426,044	12201,005	11737,446	31025,635
1904,607	1925,000	1960,074	286,847	316,335	467,826	802,834	662,286	1370,132	12180,933	9882,131	29035,140
2042,106	1918,000	1733,426	269,866	331,288	408,113	882,589	694,139	2422,103	11784,871	12232,107	27065,858
2024,834	1955,000	1738,599	285,363	375,120	477,864	783,505	791,328	1421,659	11835,128	10801,880	21955,352
1987,659	1899,000	1698,878	289,262	327,466	636,044	786,507	809,325	2408,836	11691,275	10194,283	21358,250
2109,292	2122,000	1883,763	329,510	345,905	468,013	824,866	833,919	1302,250	12377,085	10329,666	29015,320
2407,996	2047,000	1694,723	266,446	330,459	463,883	809,140	912,042	1352,884	12432,254	10919,622	22035,240
2062,666	1921,000	1769,802	363,936	346,354	472,181	827,805	657,343	1338,054	11829,309	11024,711	22003,500
2056,761	2077,000	1710,591	271,550	319,474	468,931	812,432	725,549	1311,055	12774,705	10702,087	23238,001
2003,133	1985,000	2030,941	290,942	306,460	385,830	788,074	700,065	1280,473	12781,403	9945,802	30285,269
2091,939	2197,000	1789,202	264,975	586,589	493,411	807,101	677,358	2513,829	11787,149	10274,326	21944,985
2106,160	1945,000	1813,654	332,356	520,597	519,944	774,860	799,322	1355,731	12551,716	11234,766	23592,500
2223,594	1920,000	1708,706	282,842	315,975	483,241	823,252	676,756	2396,252	12628,054	10423,654	22306,265
2020,349	2031,000	1758,897	270,146	331,717	488,195	770,765	667,768	1282,189	13268,836	11030,197	28369,300
2110,703	1967,000	1713,188	262,944	531,145	390,444	1300,159	687,417	2565,944	11704,673	10985,916	21799,635

WFS 300			**WFS 3000**			**WFS 30 000**		
Firefox	Chrome	IE	Firefox	Chrome	IE	Firefox	Chrome	IE
347,577	421,060	733,825	1395,018	1101,098	3679,519	25938,830	9314,198	19424,150
450,948	397,223	582,044	1316,039	1114,661	3110,890	20996,111	11112,198	35315,817
353,533	422,134	561,213	1368,632	1317,623	3810,843	21158,285	10140,994	29419,525
457,400	408,034	791,130	1325,367	1338,111	3027,048	22701,304	13091,061	27652,427
390,321	434,469	578,965	1283,033	1359,051	2855,227	22344,633	10179,609	36909,346
393,982	591,737	598,063	1317,286	1126,659	2957,499	21553,184	11616,802	35198,738
378,465	472,942	568,870	1304,655	1104,913	4276,211	22716,164	10929,585	35565,242
373,819	418,750	585,697	1299,164	1151,163	2937,474	20743,810	10282,696	28308,278
435,528	408,162	599,525	1415,484	1237,739	2963,325	21471,513	11847,674	29071,759
351,451	419,693	574,308	1600,768	1342,630	2997,157	21393,343	10519,543	28681,307
429,597	451,019	509,266	1358,463	1168,307	2922,719	22079,889	11600,594	20012,156
392,045	536,905	587,408	1279,324	1253,016	3740,651	21010,260	10398,554	31028,352
365,712	413,155	548,578	1379,700	1371,296	3780,889	22797,487	10046,097	28398,524
427,839	404,550	585,791	1280,063	1313,437	3072,357	20990,086	10221,150	30952,369
414,990	465,750	584,744	1317,881	1115,463	2947,850	21260,044	11823,503	35048,254
349,865	398,843	560,980	1330,048	1120,142	2914,836	22351,377	11519,574	35569,258
388,712	412,508	692,844	1451,278	1118,507	3636,882	21729,697	10113,645	30258,360
358,088	424,731	579,102	1382,648	1361,851	2983,562	21296,850	9736,875	29125,350
374,994	458,896	522,396	1399,997	1138,758	3721,211	29558,229	9906,097	29346,350
363,176	403,414	504,790	1414,148	1319,235	2957,141	22175,903	12561,882	28681,307
381,243	418,842	589,760	1323,942	1132,209	2981,498	24365,609	11231,531	29669,356
476,002	392,801	563,068	1266,922	1221,160	2840,586	21219,975	11657,396	31258,300
372,578	407,559	589,337	1330,286	1224,415	2921,163	22034,697	11852,439	35223,360
419,626	604,837	566,574	1420,225	1266,701	2926,774	21208,921	9662,397	29936,360
425,529	405,632	578,120	1350,399	1314,092	2920,316	20843,461	12415,275	28216,873
387,907	465,272	551,708	1235,176	1185,518	2908,285	26477,066	10301,144	29857,350
382,147	415,110	546,463	1321,568	1112,799	4111,266	20759,750	9935,084	28966,342
378,218	406,927	706,895	1287,445	1119,121	2903,210	20394,804	11630,606	33258,350
428,003	408,708	571,540	1586,824	1111,634	2952,559	23269,705	9751,879	34288,365
367,651	413,989	741,307	1289,401	1156,732	2998,534	21879,302	11444,273	29008,937

LISA 6

OpenLayers2	OpenLayers3	Leaflet	Google Maps	ArcGIS
320,000	1007,000	502,000	5181,000	1958,000
266,000	1999,000	426,000	5577,000	1935,000
308,000	1033,000	439,000	5351,000	1925,000
264,000	1088,000	474,000	7744,000	1912,000
313,000	1781,000	475,000	5726,000	1909,000
279,000	2255,000	528,000	5447,000	1926,000
323,000	1999,000	461,000	6335,000	2001,000
275,000	1983,000	466,000	5973,000	2436,000
259,000	1052,000	486,000	5764,000	1936,000
256,000	1262,000	704,000	5297,000	1966,000
272,000	1083,000	490,000	5812,000	1998,000
287,000	1158,000	453,000	5751,000	1952,000
255,000	1627,000	474,000	7626,000	1928,000
283,000	1576,000	438,000	8336,000	2015,000
260,000	1049,000	472,000	7324,000	1995,000
255,000	1051,000	478,000	5036,000	2055,000
251,000	1562,000	506,000	5331,000	1925,000
258,000	1736,000	493,000	5474,000	1918,000
284,000	1053,000	481,000	5544,000	1955,000
464,000	1094,000	474,000	6504,000	1899,000
235,000	1065,000	531,000	5581,000	2122,000
267,000	1752,000	484,000	5707,000	2047,000
297,000	1069,000	478,000	5448,000	1921,000
254,000	1079,000	468,000	5348,000	2077,000
249,000	1072,000	552,000	5573,000	1985,000
388,000	1813,000	542,000	5908,000	2197,000
261,000	1061,000	656,000	5765,000	1945,000
253,000	1235,000	498,000	5436,000	1920,000
302,000	1095,000	559,000	5278,000	2031,000
241,000	1436,000	596,000	5607,000	1967,000
282,633	**1370,833**	**502,800**	**5892,800**	**1991,867**

LISA 7

WMS	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	235,0	255,2	266,5	282,6	294,5	464,0
OpenLayers 3	1007,0	1066,0	1126,0	1371,0	1709,0	2255,0
Leaflet	426,0	427,5	482,5	502,8	522,5	704,0
Google Maps	5036,0	5439,0	5594,0	5893,0	5884,0	8336,0
ArcGIS	1899,0	1925,0	1956,0	1992,0	2012,0	2436,0

GeoJSON 300	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	29,00	30,00	31,00	32,47	33,75	45,00
OpenLayers 3	69,41	71,96	74,98	77,31	81,99	90,12
Leaflet	116,00	124,10	125,80	126,20	127,70	136,00
Google Maps	52,35	56,28	57,11	58,74	60,15	68,80
ArcGIS	306,00	316,00	328,30	355,10	346,20	586,6

GeoJSON 3000	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	108,00	115,00	120,20	123,50	127,40	159,00
OpenLayers 3	387,50	411,20	418,20	418,90	423,00	487,30
Leaflet	690,50	819,20	875,40	861,70	923,90	1003,00
Google Maps	259,00	336,10	422,90	392,30	450,30	464,60
ArcGIS	624,70	667,20	687,80	718,90	776,20	912,00

GeoJSON 30000	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	708,00	769,20	782,50	784,80	806,80	878,00
OpenLayers 3	3605,00	4011,00	4253,00	4198,00	4349,00	4817,00
Leaflet	6395,00	7107,00	7560,00	7447,00	7809,00	8189,00
Google Maps	2230,00	2270,00	2297,00	2305,00	2331,00	2467,00
ArcGIS	9774,00	10070,00	10340,00	10530,00	10920,00	12230,00

WFS 300	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	297,10	326,30	345,70	358,40	362,40	633,80
OpenLayers 3	234,80	260,30	283,00	288,10	317,00	340,00
Leaflet	236,50	255,60	261,80	262,90	267,70	299,90
Google Maps	257,10	272,90	279,80	285,70	286,80	429,60
ArcGIS	392,80	407,70	416,90	436,80	446,90	604,80

WFS 3000	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	1460,00	1485,00	1524,00	1582,00	1604,00	1991,00
OpenLayers 3	868,20	904,50	923,20	927,70	944,90	994,50
Leaflet	1395,00	1536,00	1566,00	1639,00	1602,00	3274,00
Google Maps	696,90	741,00	786,70	807,20	816,60	1225,00
ArcGIS	1101,00	1119,00	1177,00	1211,00	1314,00	1371,00

WFS 30000	Miinimum	1 kvantiil	Mediaan	Keskmine	3 kvantiil	Maksimum
OpenLayers 2	11220,00	11760,00	12140,00	12400,00	12470,00	15000,00
OpenLayers 3	9130,00	10010,00	10340,00	10350,00	10600,00	12210,00
Leaflet	15280,00	19010,00	20360,00	20210,00	21160,00	23400,00
Google Maps	7888,00	8195,00	8420,00	8651,00	8836,00	10770,00
ArcGIS	9314,00	10120,00	10720,00	10890,00	11630,00	13090,00

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Olga Troškina
(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Veebipõhiste kaardirakenduste loomise rakendusliideste võrdlus,

   (lõputöö pealkiri)

   mille juhendaja on Tõnu Oja
   (juhendaja nimi)

   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 DSpacei 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, **25.05.2015**