TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Fred Varb

155186IAPB

INFRASTRUKTUUROBJEKTIDE TEISENDUS SOBIVATEKS POLÜGOONIDEKS

Bakalaureusetöö

Juhendaja: Margarita Spitšakova, Ph.D

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Fred Varb

20.05.2019

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua ja täiendada nõuetele vastav ja parameetriseeritav rakendusliides, mida on võimalik integreerida juba olemasoleva veebirakendusega. Töös põhjendatakse metoodika ja arhitektuuri valikut, tutvustatakse programmi sisendeid ja väljundeid, lõpuks valideeritakse rakendus erinevate võrdluste ja testide abil.

Töö tulemuseks on erinevaid geoandmetega faile polügoonideks konverteeriv rakendus, kus tulemiks saadud polügoonid vastavad ülesandes püstitatud nõuetele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 6 peatükki, 24 joonist, 1 tabelit.

Abstract

Conversion of infrastructure objects to eligible polygons

Sille is a web application, that helps users measure the land deformation of an infrastructure object by millimeter precision. To measure the land deformation via satellite, the client needs to draw the outlines of the given infrastucture in Sille application. Complicated objects can be difficult to draw in the application and for this reason, an API, that converts spatial data from files into polygons, is needed.

The goal of this bachelor's thesis is to expand an application program interface (API) created in subject „Tarkvaraarenduse meeskonnaprojekt", so that the API converts spatial data into polygons that meet the requirements. As an extention to the requirements, conversion of coordinates, axis switching and GML file reading is added.

To achieve these goals the author provides overview of metodics and architecture, explains how the files are read. Describes how coordinate systems can be switched, gives examples of how polygons are created and validated.

The thesis is in Estonian and contains 30 pages of text, 6 chapters, 24 figures, 1 table.

Lühendite ja mõistete sõnastik

GIS	*Geographic Information system*
API	*Application Program Interface*
LGPL	*GNU Lesser General Public License*
REST	*Representational State Transfer*
ISO	*International Organization of Standards*
JSON	*JavaScript Object Notation*
GML	*Geography Markup Language*
KML	*Keyhole Markup Language*
KMZ	*Keyhole Markup language Zipped*
SHP	*Shapefile*
XML	*Extensible Markup Language*
EPSG	*European Petroleum Survey Group*
UTM	*Universal Transverse Mercator*
WMS	*Web Map Service*
WKT	*Well-Known Text*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Sille on veebipõhine rakendus, kus kasutajal on võimalik tellida mingisuguse infrastruktuuriobjekti (näiteks maja või tee) deformeerimise jälgimise millimeetri täpsusega, Euroopa satelliidi Sentinel-1 abil [1]. Rakendus vajab analüüsiks objektide selgelt piiritletud maa-alasid polügoonidena. Selleks tuleb joonistada kaardirakendusse jälgitav piirkond ja esitada tellimus. Keerulisi objekte on raske ja aeganõudev kaardirakenduses joonistada. Failide importimise ja töötlemise süsteem aitaks kiirendada erinevate objektide joonistamist kaardirakenduses. [2]

Varasem rakendus on loodud aines Tarkvaraarenduse Meeskonnaprojekt, kuid aine raames loodud rakenduses puudub veakoodide haldus, tõlked, välisriikide poolt loodud failide lugemise tugi, koordinaattelgede vahetamise võimalus, parameetritele vastavus ja rakendus on aeglane.

Käesoleva bakalaureusetöö eesmärk on luua ja täiendada nõuetele vastav ja parameetriseeritav API ehk rakendusliides, mida on võimalik integreerida juba olemasoleva veebirakendusega. Rakendusliides peab aitama importida geoandmetega faile, töödelda geoobjekte parameetritega kirjeldatud polügoonideks ja väljastada polügoonid JSON formaadis.

Esimeses peatükis kirjeldatakse nõuded, püstitatakse küsimused ja selgitatakse panuse. Teises peatükis antakse ülevaade metoodikast, rakenduse arhitektuurist, failide lugemisest ja koordinaatsüsteemi vahetamisest. Kolmandas peatükis vaadeldakse joonest polügooni tegemist ja polügoonide kokku võtmist. Neljandas peatükis valideeritakse rakendus läbi automaattestide, parameetritele vastavuse, rakenduste kiiruste võrdluse ja veakoodide. Viimaks pakutakse võimalikke edasiarendusi.

1.1 Nõuded

Nõuded rakendusliidesele on kujunenud läbi ülesandepüstituse ja koosolekute ettevõttelt Datel [2]. Laiendused on autori poolt pakutud lahendused, mis aitaksid tulemuse kvaliteeti parandada.

Nõuded tarkvarale:

- realiseeritud API-na, programm peab töötama automaatselt;
- komponente peab olema võimalik lihtsasti integreerida teiste komponentidega;
- testitud ja kasutusjuhendiga;
- paigaldusjuhend teekide ja versioonidega.

Nõuded imporditavatele failidele:

- toetatud failiformaadid on vähemalt:
    - SHP
    - geoJson
    - KML
    - KMZ
- sisendformaatide arv ei tohi olla piiratud;
- imporditavale failile toimuks formaadikontroll ja valideerimine;
- probleemsete failide korral peab väljastama veateate, millel on nii inglise kui eesti keelne tõlke tugi;
- korrektse faili peab töötlema ümber polügoonideks.

Nõuded meetoditele:

- põhineb kommertskasutust võimaldaval vabavara teekidel;

- teostamiseks kasutatud eelistatult NodeJS või JAVA;
- asukoha riigi leidmiseks võib kasutada openstreetmap Nominatim API-t.

Nõuded konverteerimistele:

- punktidest tuleb teha ruudud, mille külje pikkus on X meetrit, olemasolev punkt jääb keskpunktiks;
- kui polügoonid on üksteisele lähemal kui X meetrit, tuleb objektid kokku ühendada;
- joonest tuleb teha polügoon X laiusega;
- joonest saadud polügoonile *convex hull'*i pole vaja teha. *Convex hull* tähendab, et polügoonist loodakse ringi laadne punktide kogum nii, et ükskõik millise kolme järjestikuse punkti valikul keskmine punk ei ole kahe äärmise suhtes ringi keskele poole;
- polügoon ei tohi sisaldada auku;
- enne polügooni, mis polnud joon, väljastamist tuleb teha *convex hull*, mis lihtsustab kujud.

Nõuded väljundile:

- väljastama peab polügoonid koos metaandmetega ja asukohaga JSON formaadis;
- väljundi alusel saab kuvada polügoonid (Joonis 1) [2];

Joonis 1. Datel'i poolt kirjeldatud väljund kaardil

Laiendused:

- GML/XML failist lugemine;
- koordinaatsüsteemi teisendamine;
- X ja Y telje vahetamise võimalus.

1.2 Küsimused

Nõuetest tulenevad uurimisküsimused on järgnevad:

Küsimus 1. Kuidas tõlkida veakoode nii, et kasutajaliidesesse saadetakse õiges keeles viga?

Küsimus 2. Kuidas testida *Spring* rakendust?

Küsimus 3. Kuidas teisendada koordinaatsüsteemi kasutaja poolse sisendiga?

Küsimus 4. Kuidas luua joonest polügoon?

Küsimus 5. Kuidas liita polügoone?

Küsimus 6. Kuidas lugeda GML failist geoobjekte?

1.3 Panus

Käesolev bakalaureusetöö on laiendus projektile, mis valmis aines “Tarkvaraarenduse meeskonnaprojekt” koostöös tudengitega Hugo Volk, Henrik Tomson, Daniel Smirnov, Valentin Stennikov, Martin Post ja juhendajaga Margarita Spitšakova. Lõputöö teema, nõuded ja eesmärgi pakkus välja Datel AS. Lõputöö autori poolt oli aine raames tehtud joone laiendamine polügooniks, osaliselt polügoonide kokku liitmine, raamistikude ja teekide valik, *Shape* failist lugemine, kasutajaliides, failide importimine, faili tüübi kontroll, teiste koodi parandamine ja ühildamine.

Lõputöö raames on juurdeehitusena tehtud veakoodide haldus, tõlge, GML faili lugemine, polügoonide liitmise algoritmi täiendamine, koordinaatsüsteemi vahetamine, x- ja y-telje muutmine, parameetrite lugemine *Spring* notatsioonile kohaselt ja testimine.

2 Algandmed

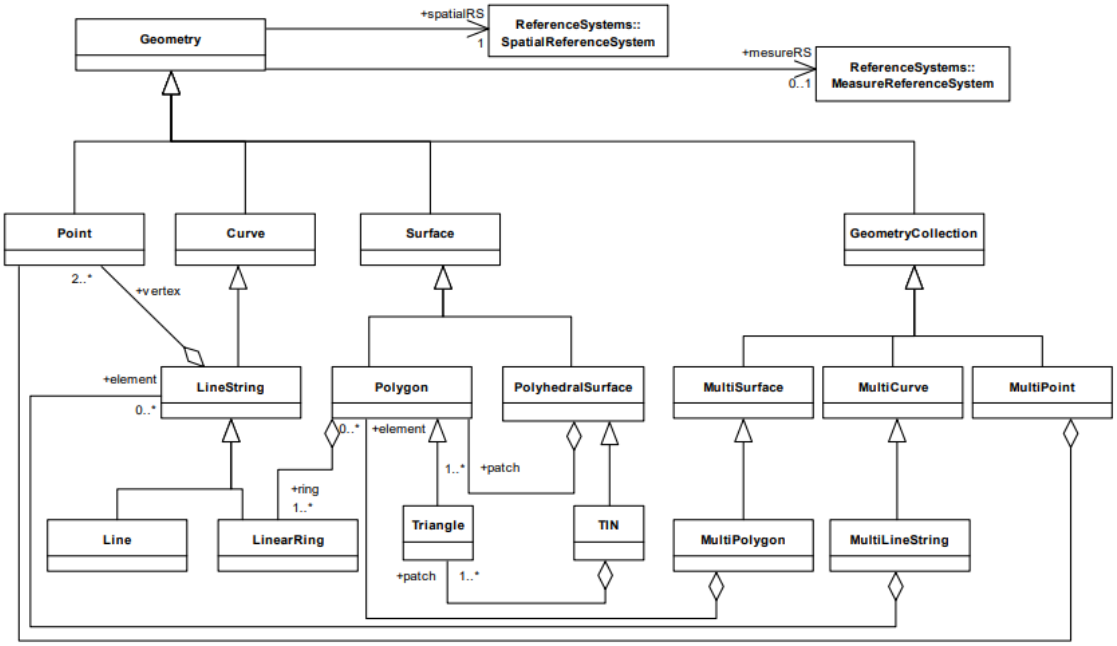
Rakenduse algandmeteks on erinevad failid (*Shape*, GML, geoJson, KML, KMZ). Kuna autori poolt on loodud GML ja *Shape* faili lugemine, mõeldud algne metoodika, arhitektuur ja koordinaatsüsteemi vahetamine, on neile pikem selgitus.

2.1 Metoodika

Üks tähtsaim kitsendus erinevate lahenduste kaalumisel on kommertskasutust võimaldavad vabad teegid. Teegid tuleb kasutusele võtta, sest pole mõistlik uuesti arendada terve komponentide kogu, mis on juba tegelikult arendatud. Kuna tegemist on geoandmetega, on kasutusele vaja võtta geograafilise infosüsteemi (GIS) teegid. Üheks selliseks geograafiliseks infosüsteemiks on *GeoTools*. [3] *GeoTools* on avatud lähtekoodiga LGPL litsentsiga teek, LGPL litsents võimaldab kommertskasutust ja ei nõua, et teeki kasutanud koodijupp oleks avaldatud. Avaldada tuleb kood sel juhul, kui modifitseerida LGPL litsensiga teegi lähtekoodi. [4] Teiseks vajalikuks geoandmete protsessoriks on *JTS Topology Suite,* mis võimaldab erinevaid kahe-dimensioonilisi geoteisendusi, nagu objektide ühendamine, laiendamine, lihtsustamine ja muu. *JTS* kasutab litsense *Eclipse Public License* [5] ja *Eclipse Distribution License* [6], mis võimaldavad kommertskasutust. [7] Mõlemad teegid on realiseeritud JAVA's, seega sobiv platvorm arendamiseks oleks JAVA. Veelgi enam võimaldab Spring realiseerida REST API'sid [8] ja parameetrite lugemist *properties* failist [9]. JAVA kasutamine on ka soovitatav, kui tulevikus soovitakse API'ga ühendada andmebaasi [10].

2.2 Arhitektuur

Et komponente (JAVA klasse) oleks võimalik siduda omavahel (mis on üks nõuetest), peaksid klasside sisendid ja väljundid olema sama andmetüüpi või sama struktuuriga. *JTS* ja *GeoTools* mõlemad kasutavad teisendusteks standardset *Simple Feature* objekti, mis on ISO 19125 standard ja kirjeldab enamuste kahe-dimensiooniliste objektide mudeleid. *Simple Feature* sisaldab endas muuhulgas ka geomeetria objekte, mille struktuur ISO 19125 järgi on kirjeldatud joonisel 2 [11].



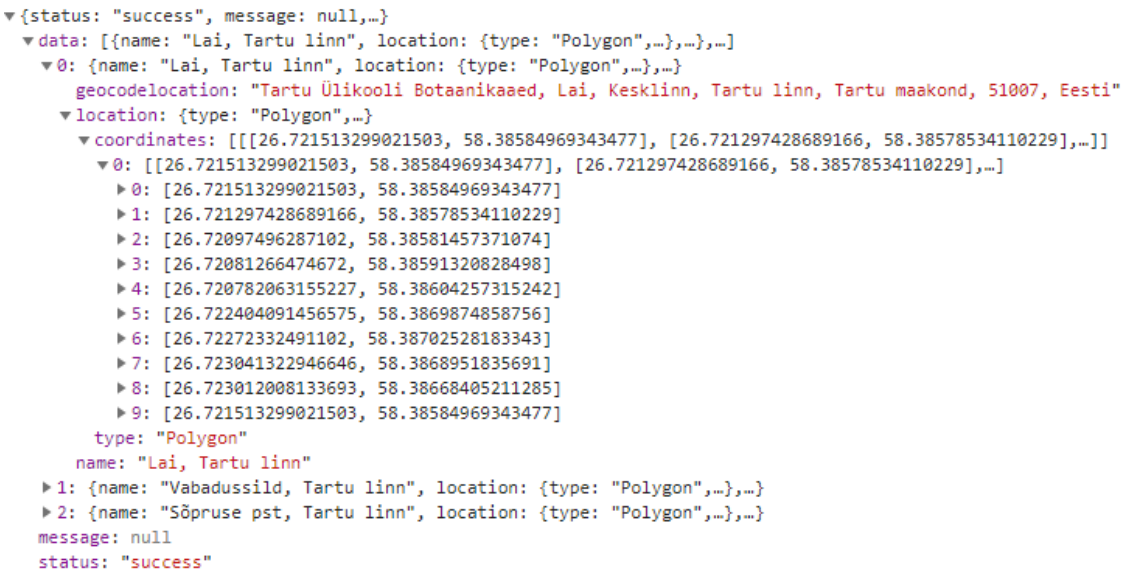
Joonis 2.Geomeetria klassi hierarhia

Simple Feature objekti geomeetria aitab kirjeldada kõiki võimalike geoobjekte, mida failist loetakse. Kuna *Simple Feature* on standardis kirjeldatud, siis on seda sobiv kasutada vaheklasside sisendiks ja väljundiks. Failide lugejad peaksid lugema faili tüüpi objekti ja tagastama massiivi *Simple Feature* objektidest. Väljundiks genereerib Spring vajaliku JSON formaadi klassist mille konstruktsioon on joonisel 3. [10]

```
public class Polygon{
    private String type;
    private String id;
    private String description;
    private String name;
    private String geoCodeLocation;
    private Object polygon;
}
```

Joonis 3. JAVA klass JSON genereerimiseks

Spring genereerib Joonisel 3 näidatud JAVA klassist vajaliku JSON formaadis väljundi, kui omastada *poygon* väljale geomeetria, mille väljastab *Simple Feature* (Joonis 4).



Joonis 4. JAVA Spring genereeritud JSON

2.3 *Shape* faili lugemine

ESRI *Shape* fail koosneb minimaalselt kolmest failis –

- .shp fail, kus hoitakse geomeetria objekti ennast;
- .shx fail, mida kasutatakse indekseerimiseks, et .shp failist sissekandeid leida;
- .dbf fail, mis koosneb *feature* atribuutidest.

Mittekohustuslikes failides hoitakse näiteks kahe-dimensiooniliste andmete võtmeid, projektsiooni, meta-datat ja muud. Tavaliselt hoitakse *Shape* faile koos, kõik komponendid on sama prefiksiga. [12]

Shape faili lugemiseks saab kasutada *GeoTools Shape* faili lugemiseks mõeldud pluginat. Kahjuks on plugina implementatsioonis keskendutud lokaalsetele falidele, see tähendab, et faili lugemiseks antakse ette faili asukoht, et algoritm saaks lugeda ka teisi faili selles asukohas [13]. Veebist lugemisel tekib probleem sellest, et üle veebi ei saa API võtta faili asukohast juurde vajalikku teavet. Vajaliku teabe lugemiseks *Shape* failist saab siiski teha lokaalsed ajutised failid:

```
//Create location for the temp files location
File directory = new File(System.getProperty("java.io.tmpdir"));
//Put file to temporary directory with the correct file name
File testFile = new File(directory, file.getOriginalFilename());
try {
    // Make OutPutStream to push shape file contents to the new location
    OutputStream outputStream = new FileOutputStream(testFile);
    outputStream.write(("").getBytes());
    outputStream.write(file.getBytes());
    outputStream.close();
} catch (IOException e) {
    e.printStackTrace();
}
// Return the new file, which is now local
return testFile;
```

Nüüd on *GeoTools*'i plugina jaoks *Shape* failid lokaalsed ja saab faile lugeda. Kuna lisa parameetrid *feature* atribuutidest pole vaja, siis tuleb importida ja lugeda minimaalselt .shp ja .shx faili korraga. Mõne faili juhul piisab ainult .shp faili lugemisest ka.

2.4 GML faili lugemine

GML fail, nagu KML failgi, põhineb XML notatsioonil. Kuna GML ja KML on XML-põhised, on võimalik andmetele peale vaadates juba aru saada, mis failis on, erinevalt *Shape* failist. GML failis on võimalik kirjeldada ka koordinaatsüsteem *srsName* nime all, mis annab võimaluse parsida andmetest *srsName* ja selle põhjal koordinaatsüsteemi teisendada [14]. Antud koodis muutuja *from* ongi koordinaatsüsteem, millest saaksime edasi projekteerida koordinaadid programmi poolt kasutatavasse koordinaatsüsteemi:

```
DefaultProjectedCRS defaultProj =
(DefaultProjectedCRS)((Geometry)simpleFeature.getProperty("SHAPE").getValue()).getUse
rData();
CoordinateReferenceSystem from = CRS.parseWKT(defaultProj.toWKT());
```

Sellist lahendust aga loodavas süsteemis ei kasutata, kuna kasutajale on antud võimalus ise sisestada soovitud koordinaatsüsteem, kust teisendada. GML ja KML faili lugemisel antakse XML süntaks analüüsiks *GeoTools* komponendile konfiguratsioon, mille järgi faili analüüsida. Konfiguratsioonis on kirjas erinevad objekti tüübid/struktuurid, mis failis leiduvad [15]. *GeoTools* parser loeb faili voo ja tagastab *SimpleFeatureCollection* objekti, millest saab individuaalseid *simple feature* objekte analüüsima hakata. Antud peatükiga on vastatud 6. uurimisküsimusele.

2.5 Koordinaatsüsteemi kirjeldamine

Õige koordinaatsüsteem faili sisse lugemisel on üks tähtsamaid teisenduse lähteandmeid. Näiteks punkti koordinaat võib olla mitu tuhat korda suurem sellest, mida süsteem eeldab. Et *GeoTools* suudaks enamus EPSG süsteeme teisendada, tuleb importida *maven*'i teek *gt-epsg-extention*. Selleks, et konverteerida faili õigesse koordinaatsüsteemi, on vaja esialgset koordinaatsüsteemi, kust teisendada. Lihtsam on lasta kasutajal valida kasutajaliideses, kuna igas failis ei ole struktuur päris sama. Massiiv erinevatest koordinaatsüsteemidest ja nende teisendustest on hallatud *International Association of Oil & Gas Producers* poolt, koordinaatsüsteemid on kirjeldatud EPSG koodidega ja alla laaditav epsg.org veebilehelt MS Access failina [16]. JAVA's saab lugeda MS Access faili *Ucanaccess* draiveriga ja maven impordiga, mis on kirjeldatud Lisas 2. Kui EPSG koodid on sisse loetud, saab need saata kasutajaliidesesse, et neist saaks valiku teha. *MathTransform* abil saab leida sobiva ülemineku ühelt koordinaatsüsteemilt teisele ja JTS teegi abil ülemineku teostada:

```
CoordinateReferenceSystem from = CRS.decode(EPSG);
CoordinateReferenceSystem target =
CRS.decode(constantReader.getCoordinateReferenceSystem());
MathTransform transform = CRS.findMathTransform(from, target);
com.vividsolutions.jts.geom.Geometry targetGeometry = JTS.transform((Geometry)
feature.getDefaultGeometry(), transform);
feature.setDefaultGeometry(targetGeometry);
```

Samuti võib kasutajal olla vajadus vahetada x- ja y-telge, kuna kõik failid ei kirjelda x- ja y-telge ühte moodi, näiteks *Shape* või GML faili lugemisel. Selle jaoks lisas autor kontrollerisse vastavad sisendid, kus *file* parameeter on kohustuslik:

```
@PostMapping("/upload")
public Object singleFileUpload(@RequestParam("file") MultipartFile[] file, String
epsg, Boolean flip) throws IOException,ParseException,
        ParserConfigurationException, SAXException, XMLStreamException {
    String EPSG = null;
    if (!epsg.isEmpty()) {
        EPSG = "EPSG:" + epsg;
        System.out.println(EPSG);
    }
    return selector.findExtensionAndReadFile(file, EPSG, flip);
}
```

EPSG koodidega ja eelnevate 2.5 punktis kirjeldatud koodinäidetega saab kirjeldada enamik koordinaatsüsteeme ja nende teisendust süsteemi siseseks koordinaatsüsteemiks, mis vastab 3. uurimisküsimusele.

Et iga kord ei peaks lugema kõiki koode *Ucanaccess* draiveri abil ja päringuga, on loodud koodidest XML fail, kuna ainult EPSG koodide hoidmiseks pole andmebaasi vaja. XML faili saab luua *JAXB* abil, kus on võimalik anda ette klassi sugune kontekst ja selle põhjal luua XML fail. Hiljem on sama faili põhjal võimalik lugeda kõik koodid uuesti XML failist välja. Failist lugemiseks ja faili kirjutamiseks on vastavalt *unmarshal* ja *marshal* funktsioonid. XML faili loomiseks eraldi importida pole midagi vaja, kuna vajalikud meetodid on vaikimisi JAVA's juba olemas. [17]

3 Teisendused

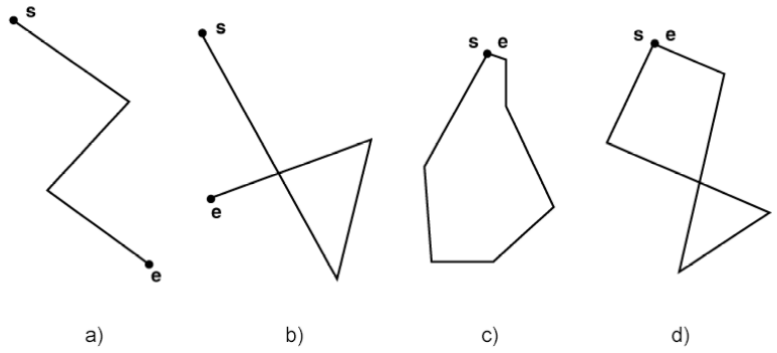
Tähtsamad teisendused on joonest polügooni genereerimine ja polügoonide kokku võtmine. Algoritmidest paremini aru saamiseks on esitatud pseudokoodid ja joonised. Peatükk 3.1 on vastus 4. uurimisküsimusele. Peatükk 3.2 on vastus 5. uurimisküsimusele.

3.1 Joonest polügoon

Failidest saadud jooned võivad koosneda *LineString*'idest või *MultiLineString*'idest.

3.1.1 Joonte ülevaade

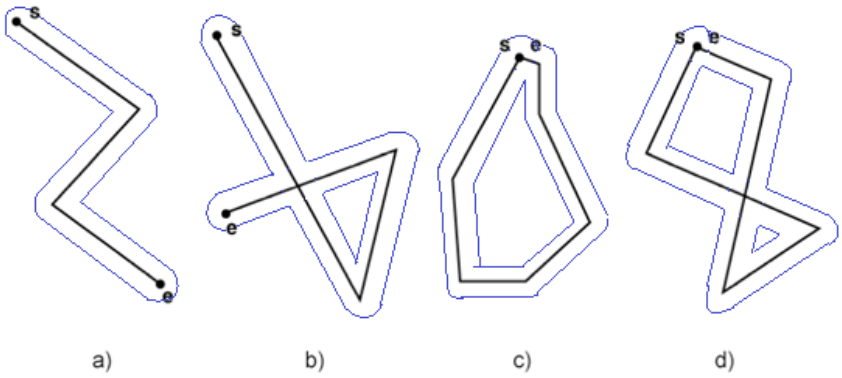
Jooned klassifitseeruvad suletud, lihtsateks ja mitte lihtsateks joonteks:



Joonis 5. Joonte liigid

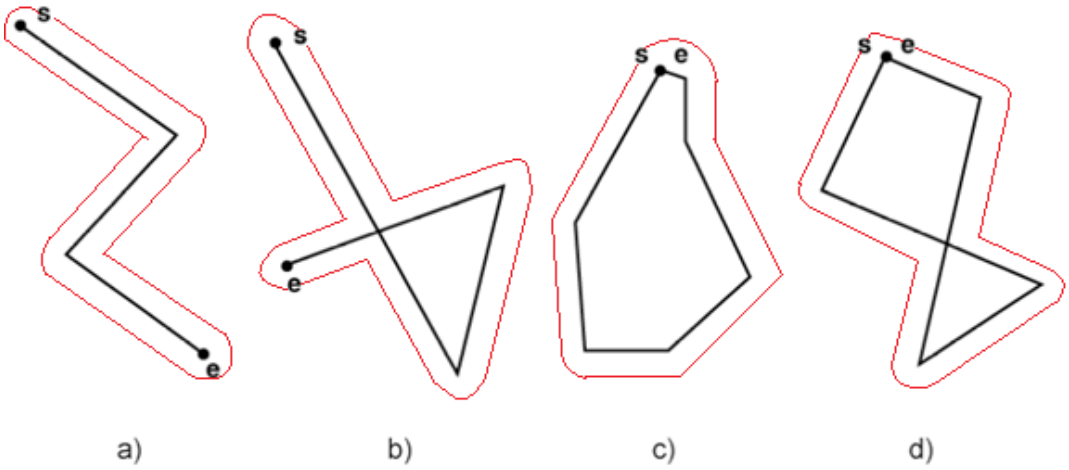
Joonisel 5 [11] märgistab s joone algust ja e joone lõppu. Antud joonisel on tähistatud a-sektsioonis lihtne *LineString*, joonisel pole ühtegi lõiget ega ühenduskohta. B-sektsioonis on mitte lihtne *LineString*, kuna esineb lõige. C-sektsioonis on lihtne suletud *LineString*. D-sektsioonis on mitte lihtne suletud *LineString* objekt [18].

Joonisel 5 näidatud *LineString*'ide polügoonideks tegemisel ei piisa ainult *GeoTools* geomeetria *buffer* funktsioonist. Tekkiv polügoon *buffer* funktsiooniga oleks ligikaudu sinise joonega märgitud:



Joonis 6. *Buffer* funktsiooni läbinud joon

Joonisel 6 [11] sinise joonega märgitud polügoonid pole antud ülesande nõuetele vastavad. Polügoonidel on augud sees, mida väikeste polügoonide juhul ei tohi sees olla. Aukude eemaldamiseks on geomeetria objektil võimalik ainult välisring saada ja tekiksid ligikaudu punase joonega märgitud polügoonid, mis asuvad joonisel 7.



Joonis 7. Joonest polügoon

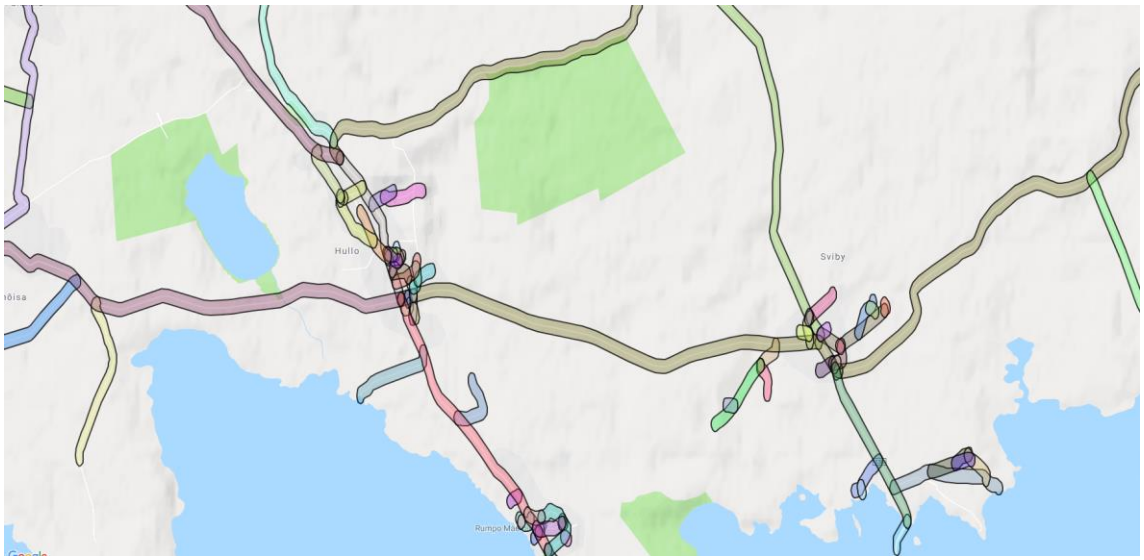
Joonisel 7 [11] märgitud polügoonid on ülesande lahendiks ainult siis, kui joonisel 6 tekkinud polügoonide sisemised kujundid (b, c ja d sektsioonis) on ette antud pindala piirangust väiksemad. Kui loodava polügooni sisemine ring on liiga suur, siis (jooniselt 6) suuremat polügooni ainult väliste piiridega (joonisel 7) ei looda ja lahend on ainult *buffer* rakendamine (joonisel 6). Et teada geomeetria objekti sisemise augu pindala, tuleb võrrelda ainult kõige suuremat geomeetria objektis leiduvat auku, teised on irrelevantsed.

3.1.2 Joonte teisendus

Kui jooned on liiga tihedalt, siis tuleks loodavad polügoonid kokku võtta ja samuti kontrollida sisemiste aukude pindalasid. Joonisel 8 on võimalik peale vaadates juba märgata, et esineb palju väga suuri ja problemaatilisi auke, mida algoritm ei tohiks üldistada ja ainult väliste piiride alusel polügooni luua.

Joonis 8. Muhu saare teed

Rakendades *buffer* funktsiooni antud joontele saab jooned konverteerida jooned polügoonideks (joonis 9). Lähemalt vaatame joonise 8 punasega tähistatud osa.



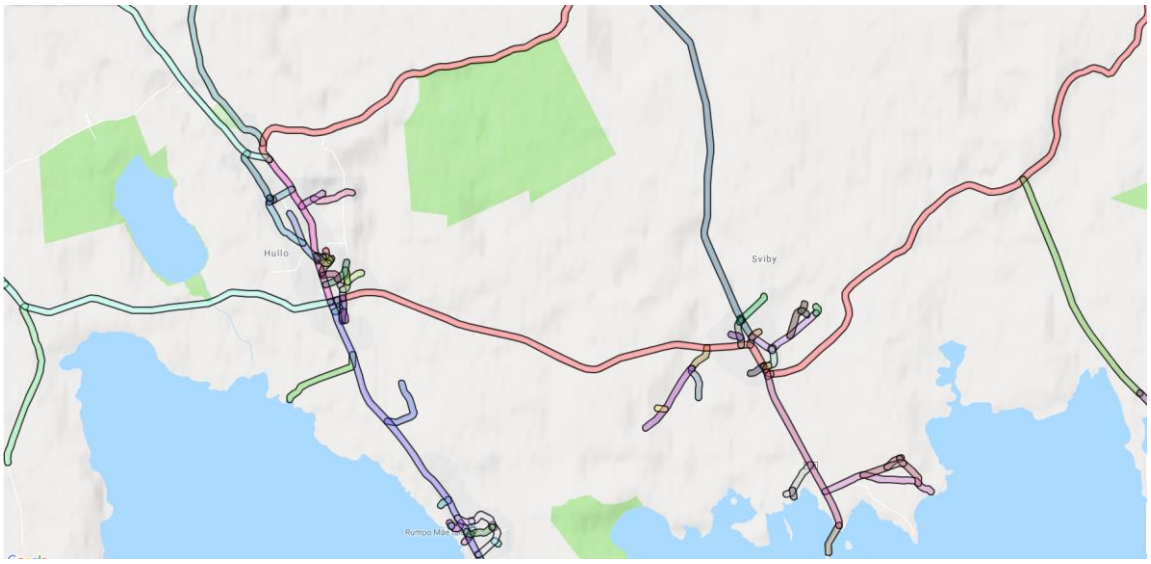
Joonis 9. Muhu saare polügoonid

Joonisel 9 jooned ei ole päris ühte moodi laiemaks tehtud – mõned jooned on laiemad kui teised ja vastupidi. Viga on tulenenud sellest, et meetrid on teisendatud WGS 84 süsteemi, kuna koordinaadid on selles süsteemis ja siis tehtud laiendamine. Sellisel viisil laiendamine ei ole ülesande lahenduse jaoks sobiv. Veel üheks võimaluseks laiendamiseks on koordinaatsüsteemi vahetamine meetripõhise süsteemi vastu, laiendada meetrite järgi ja siis konverteerida tagasi WGS 84 süsteemi:

```
String code = "AUTO:42001," + longitude + "," + latitude;
CoordinateReferenceSystem auto = CRS.decode(code);
MathTransform toTransform = CRS.findMathTransform(DefaultGeographicCRS.WGS84, auto);
MathTransform fromTransform = CRS.findMathTransform(auto,
DefaultGeographicCRS.WGS84);
line = JTS.transform(line, toTransform);
line = line.buffer(constantReader.getBUFFER_AMOUNT_METERS());
return JTS.transform(line, fromTransform);
```

Eelnevas koodijupis on kirjeldatud *code* osa *OpenGis* WMS notatsioonile sarnaselt. Geotools implementeerib WMS standardeid, mis on kirjeldatud *OpenGIS* implementatsiooni standardites. Üks neist on koordinaatsüsteemina *AUTO* kasutamine, millele järgnevad laiuskraad ja pikkuskraad, et konverteerida WGS 84 koordinaatsüsteemist UTM koordinaatsüsteemi koordinaadid [11]. UTM koordinaatsüsteemi puhul on Maa jaotatud kuuekümneks tsooniks pikkuskraadide 84º põhjast ja 80º lõunast, igaüks on 6 pikkuskraadi lai (kokku 360º). Iga tsoon jaotatakse 100000 m^2 pindalaga nelinurgaks. UTM süsteemis on ka x- ja y-telg aga seekord meetrites. Pikkus- ja laiuskraadiga saab defineerida punkti asukoha UTM

koordinaatsüsteemi võrgustiku tsoonis, mis võimaldab selles tsoonis täpset, meetritega laiendamist (joonis 10) [19].

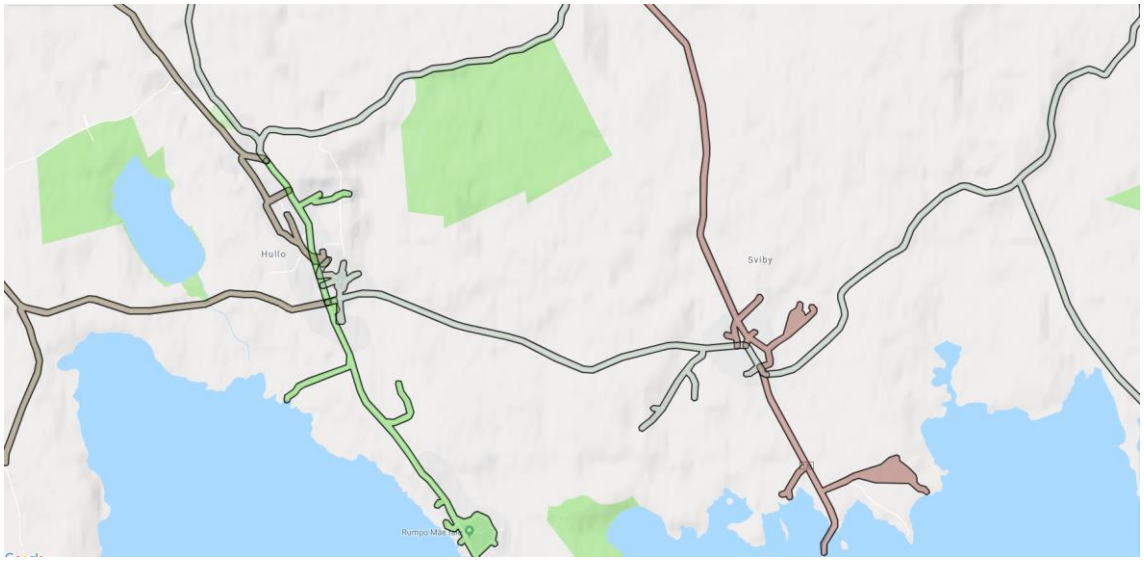


Joonis 10. Muhu saare polügoonid parandatud laiusega

Jooniselt 10 on näha, et kui enne *buffer* funktsiooni kasutamist joon viia UTM süsteemi ja hiljem tagasi WGS 84 süsteemi, siis ei esine joone laiendamises suuri erisusi joone laiuse osas. Joonisel 9 ja joonisel 10 esineb veel polügoonide kattumisi – liigandmed ülesande lahenduseks. Joontest laiendatud polügoonide kokku liitmiseks tuleb iga joone kaugust mõõta mõnest teisest joonest. Kui jooned on piisavalt lähedal, tuleb konstrueerida uus geomeetria neist. Loodud geomeetria kõige suurema augu pindala tuleb võrrelda ette antud väärtusega. Kui kõige suurema augu pindala on suurem kui lubatud, siis uut geomeetriat ei looda ja jääb geomeetria alles nii, nagu oli enne uue geomeetria loomist. Vastupidiselt, kui kõige suurema augu pindala on väiksem, kui lubatud, siis tuleb võtta uue geomeetria välise ringi ainult ja konstrueerida sellest polügoon. Joontest saadud polügoonide liitmise algoritmi pseudokood on järgmine:

```
1. FOR polygonA in bufferedLinesArr:
2.     FOR polygonB in bufferedLinesArr:
3.         IF polygonA.isWithinDistance(polygonB) and polygonA != polygonB:
4.             var newPolygon = union(polygonA, polygonB)
5.             var maximumArea = 0
6.             FOR innerPolygon in newPolygon:
7.                 newArea = area(innerPolygon)
8.                 IF newArea > maximumArea
9.                     maximumArea = newArea
10.             IF maximumArea < maximumAreaFromTask:
11.                 newPolygon = exteriorRing(newPolygon)
12.                 DELETE polygonA, polygonB
13.                 SAVE newPolygon
14.                 JUMP 1.
```

Algoritmi rakendamisel loodud polügoonidele on joonte failide ülesande lahenduseks joonis 11.



Joonis 11. Muhu saare joonte lõpplahendus

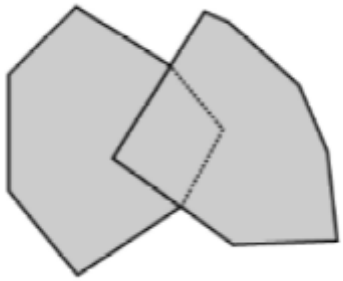
3.2 Polügoonide kokku liitmine

Tavaliste polügoonide kokku liitmine toimub sarnaselt joontest laiendatud polügoonide kokku liitmisele. Sarnaselt tuleb iga polügooni võrrelda teisega, kontrollida kaugust ja teha otsus, kas liita need kokku, lõigata või jätta samaks. Kokku liitmisel on suurematekst probleemideks polügoonide kuulumine üksteise sisse ja polügoonide osaline lõikumine. Kui polügoon kuulub täies ulatuses teise polügooni sisse, siis võib teise sisse jääva polügooni lahendusest välja jätta. Kui polügoon osaliselt lõikab teist polügooni, tuleb ühte neist lõigata, et ei tekiks ülekattumisi. Kui polügoon ei lõiku teise

polügooniga ega kuulu teise sisse ja mõlemad polügoonid on väiksemad maksimaalsest pindalast, võetakse polügoonid kokku ja tehakse *convex hull*.

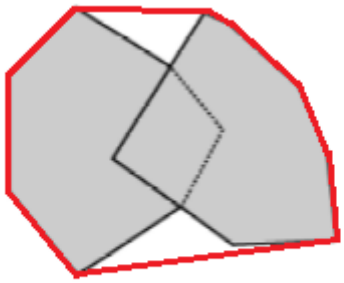
3.2.1 Üksikute polügoonide liitmine

Näide ülekattumisest on joonisel 12.



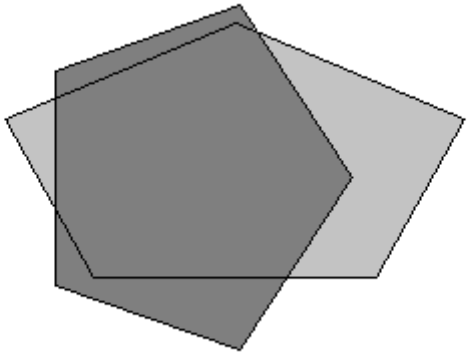
Joonis 12. Ülekattumise näide

Kui joonisel 12 [11] olevate polügoonide pindalad on väiksemad piirangust, saab antud polügoonid liita ja võtta *convex hull*, mis näeks välja nagu punasega märgitud ala joonisel 13 [11].



Joonis 13. Ülekattumise näite *convex hull*

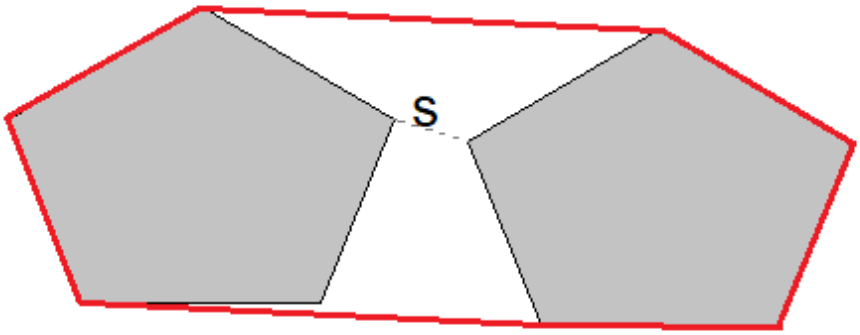
Joonisel 12 lõikuvad polügoonid ei tohi kokku ühendada kui vähemalt 1 neist on liiga suure pindalaga. Mitte kokku liitmisel tekib üks uus polügoon, mis joonisel 12 on vasakpoolne, piirjoonega tähistatud ala kustutatakse. Keerulisemate polügoonide lõikamisel võib tekkida ka mitu polügooni (joonis 14).



Joonis 14. Ülekattumise mitme polügooni näide

Joonisel 14 tekib lõikamisel mitu uut polügooni, mis on joonisel heledama tooniga. GeoTools loob lõikamisel mitme objekti tekkimisel *Polygon*'i või *MultiPolygon*'i asemel *GeometryCollection*'i.

Polügoonide kauguseks üksteisest võetakse kaks kõige lähimat punkti mõlemast polügoonist ja võrreldakse nende punktide kaugust teineteisest. Kui kaugus on soovitud piirides, tehakse alast *convex hull* (joonis 15).



Joonis 15. Kauguse piiranguga liitmine

Joonisel 15 tähistab S kaugust polügoonidest. Kui kaugus on väiksem kui maksimaalne kaugus, mis on ette määratud, liidetakse polügoonid ja võetakse *convex hull*, mida joonisel tähistab punane piirjoon.

3.2.2 Polügoonide liitmise Pseudokood

Algoritmis on kasutatud ära sorteerimist esimesel real, et parandada algoritmi efektiivsust ja keerukust. Sorteeritakse esimese elemendi kauguse järgi teistest. Kuna

iteratsioon algab esimesest elemendist, on sorteeritud massiiviga võimalik lühendada iteratsioonide arvu, kui liidetavad polügoonid on lähestikku.

```
1. polygonArr.sortByDistance(simpleFeatures)
2. FOR polygonA IN polygonArr:
3.     FOR polygonB IN polygonArr:
4.         IF isAllowedDistance(polygonA, polygonB):
5.             IF polygonA.covers(polygonB):
6.                 DELETE polygonB
7.                 JUMP 3.
8.             ELSE IF polygonB.covers(polygonA):
9.                 DELETE polygonA
10.                polygonA = polygonArr.next()
11.                JUMP 3.
12.            ELSE IF canMerge(polygonA, polygonB):
13.                newPolygon = merge(polygonA, polygonB)
14.                newPolygon.convexHull()
15.                DELETE polygonA, polygonB
16.                SAVE newPolygon
17.                JUMP 3.
18.            ELSE IF polygonA.overlaps(polygonB):
19.                newPolygons = polygonB.difference(polygonA)
20.                FOR newPolygon IN newPolygons:
21.                    SAVE newPolygon
22.                DELETE polygonB
23.                JUMP 2.
```

Antud algoritmis võib probleem olla suurte kaardus polügoonidega, kuna sisemiste aukude suurust või polügoonide pindala võrdlust ei toimu, sest nõuetes pole seda kirjeldatud.

4 Valideerimine

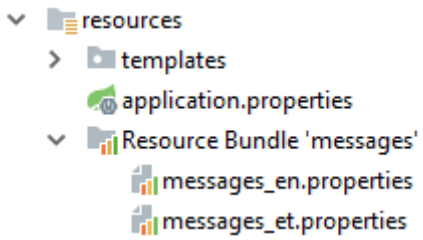
Tulemuste valideerimiseks ainult polügoonidele peale vaatamisest ei piisa. Tuli täiendada ja juurde kirjutada automaatteste. Testimiseks kasutatud failid olid lihtsustatud – näiteks punkti laiendamise testimiseks kasutas autor faili ühe punktiga. Polügoonide parameetritele vastavuse valideerimiseks tuleb vahetada parameetreid.

4.1 Tõlkimine

Vastavalt nõuetele, peab programm tõlkima veakoode. 1. uurimisküsimusele vastamiseks on vaja analüüsida erinevaid lahenduskäike. Erinevad vead võivad tekkida nii programmi jooksutamise käigus kui ka vastuolust nõuetega. Antud rakendusega on mitmeid viise kuidas saaks tõlkida. Üheks võimaluseks on saata string kujul veakood kasutajaliidesesse ja lasta kasutajaliidesel koodi alusel tõlkimist hallata. Kuna pole täpselt teada millise struktuuriga imporditava API kasutajaliides hakkab olema, siis selline lahendus ei ole universaalne. Teiseks võimaluseks on saata igas keeles tõlke kasutajaliidesesse, näiteks „error_EN“; „error_ET“. Igas keeles tõlke saatmine on autori arvates liigandmed. Parem lahendus oleks veebilehitseja poolt saadetava päringu *request header*'i kasutamine. *Header* sisaldab muuhulgas parameetrit *accept-language*, mis võimaldaks keele koodi alusel vea tõlgituna kasutajaliidesesse tagasi saata.

Et püüda kinni kasutajaliidesest saadetud *request header*, tuleb Spring notatsioonile kohaselt defineerida konfiguratsioon. *Spring* raamistikus on kirjeldatud klass *AcceptHeaderLocaleResolver*, mille abil saab teada, mis *request header*'i *accept-language* väärtus on. *AcceptHeaderLocaleResolver* klassi abil saab ka kirjeldada milliseid keeli üldse API aktsepteerib ja ülejäänu jaoks kasutatakse vaikeväärtus tõlget, mis antud rakenduses on inglise keeles. [20] [21]

Erinevad tõlked saab hoida *properties* failides. Selle jaoks tuleb teha sama algusega failid, näiteks „*messages_en.properties*“ ja „*messages_et.properties*“. Spring grupeerib need failid automaatselt ja paneb need *Resource Bundle 'messages'* sisse:



Joonis 16. *Resource Bundle*

Peale tõlkefailide loomist, saab defineerida *Bean*'i *ResourceBundleMessageSource*, mis baasnime „*messages*“ järgi teab, kust tõlke faile otsida koodi ja keelekoodi järgi. [20]

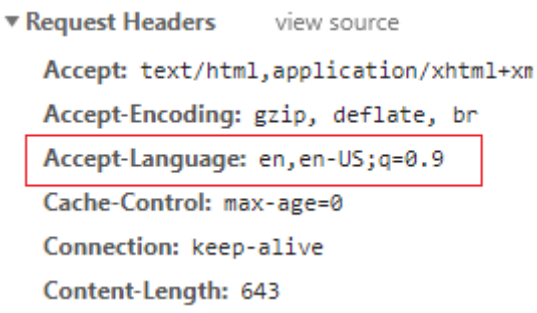
Igal pool kus tuleks tõlkida veakood visatakse JAVA poole pealt error koos veakoodiga:

```
throw new IllegalArgumentException("fail.message.gml.read");
```

Välja kutsuvas klassis saab nüüd selle veakoodi kinni püüda ja üritada otsida seda tõlke failidest, kus antud kood on kirjeldatud eri failides erinevalt:

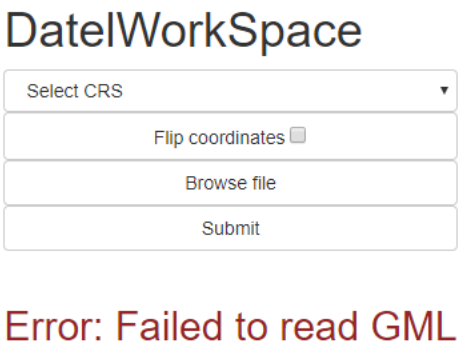
```
fail.message.gml.read=Failed to read GML
fail.message.gml.read=GML faili lugemine ebaõnnestus
```

Kasutajaliidese koha pealt võib näha Chrome *dev tools* alt saadetud päringut, kus muuhulgas on keelekoodid *accept-language* all:



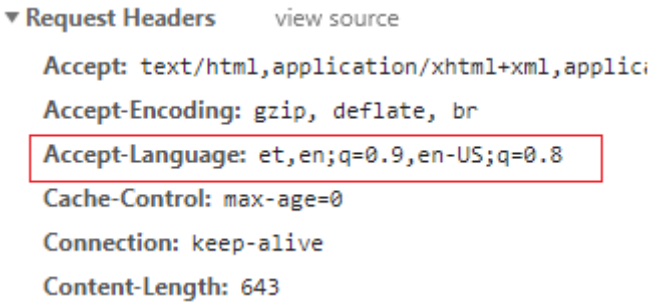
Joonis 17. *Accept-langage* inglise keeles

Accept language alusel vigase GML üles laadimise korral kuvatakse:



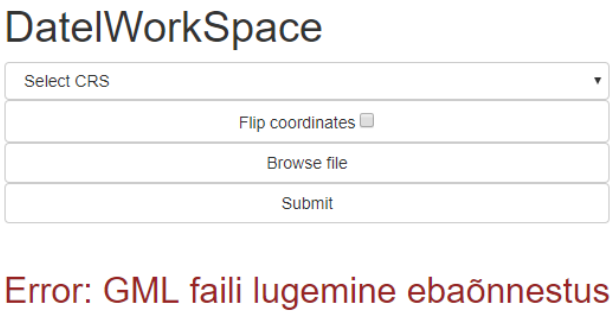
Joonis 18. Vigase GML faili inglise keelne tagastus

Kui lisada eesti keele brauseri seadete alt keelte hulka, märkida primaarseks ja vajadusel liigutada nimekirjas esimeseks, saadetakse *header*:



Joonis 19. *Accept-language* eesti keeles

Juurde on tekkinud nimekirja etteotsa „et“ keelekood. Joonisel 19 saadetud headeriga sama faili error on joonisel 20.



Joonis 20. Vigase GML faili eesti keelne tagastus

Võib esineda properties faili lugemises kodeeringu probleeme. Intellij programmis saab selle parandada määrates *properties* failile kodeeringu UTF-8.

4.2 Automaattestid

Automaattestide kirjutamisel *Spring* rakendusele tuli probleemiks *Spring* notatsiooniga defineeritud klasside kasutamine. Käesolev peatükk annab vastuse 2. uurimisküsimusele. Kui on defineeritud klass *@Service* või *@Configuration* notatsiooniga, siis tuli testi klassis kirjeldada *@ContextConfiguration* abil kõik testis kasutatava klassi sees olevad klassid mis olid *@Autowired* notatsiooniga ja klass ise. Kui testimiseks on *@Service* klass PointToPolygon ja PointToPolygon klassi sees on veel *@Autowired* ConstantReader klass, siis testi klassis tuleb PointToPolygon testimiseks kirjeldada klass:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {PointToPolygon.class, ConstantReader.class,
CalculateArea.class})
public class PointToPolygonTest {

    @Autowired
    private PointToPolygon pointToPolygon;

    //Tests here
}
```

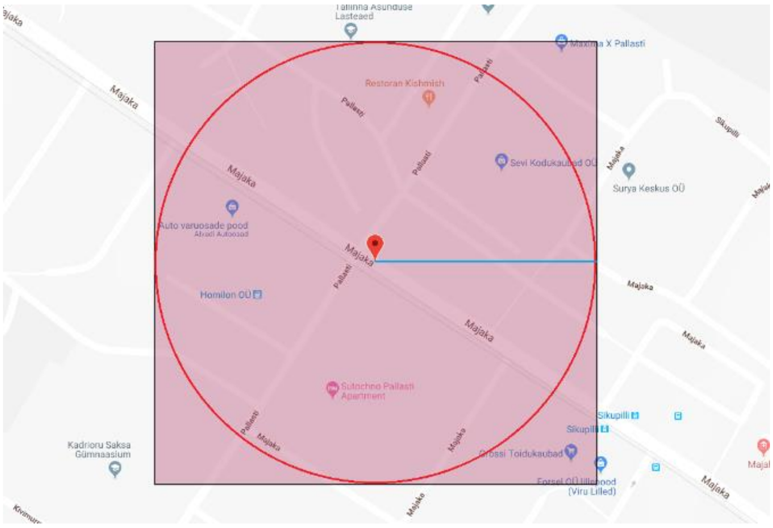
@ContextConfiguration defineerib klassi metaandmed, konfigureerib ApplicationContext'i, mida läheb vaja *bean*'ide otsimiseks, defineerib failide asukoha või klassid, mida läheb kasutatava klassi sisu kasutamise jaoks vaja. Kuna kasutajaliides saadab API'le *MultiPartFile* objekti, siis *springframework*'iga tuleb kaasa ka *MockMultiPartFile*, mille abil saab testida faile nii, nagu need kasutajaliidesest saadetakse. [22]

Testimise käigus ilmnes mitu probleemi. Esiteks WGS 84 süsteemi kirjeldamiseks ei piisa koodist CRS.decode(„EPSG:4326"), kuigi WGS 84 EPSG kood peaks olema EPSG:4326 [23]. Kui koordinaatsüsteemi kirjeldamiseks kasutada EPSG:4326, tekkisid väikesed nihked kauguste vahel, kus laius- ja pikkuskraad erinesid 200 meetri juures +- 50 meetrit, kuigi kasutati *buffer* funktsiooni, mis oleks pidanud igas suunas sama palju laiendama. Lahenduseks tuli kirjeldada WGS 84 *Well-Known Text*(WKT) formaadis:

```
GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM[
"Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.01745329251994328,AUTHORITY["
EPSG","9122"]],AUTHORITY["EPSG","4326"]]
```

Antud WKT formaadis teksti saab parsida kasutades CRS.parseWKT(), millega *buffer* laiendab ja GeodeticCalculator leiab polügoonide kaugust meetrites väga väikese veaga.

Teiseks probleemiks oli vale sisendi andmine *buffer* funktsioonile punkti laiendamisel, millelt võetakse väline ruut. Nimelt *buffer* laiendab punkti ringiks kindla raadiusega ja *envelope* on sellest vähim väline ruut [24]. Enne anti otse buffer funktsiooni sisse tegelikult radiuse (sinine joon joonisel 21) asemel parameetritest külje pikkus ehk diameeter.



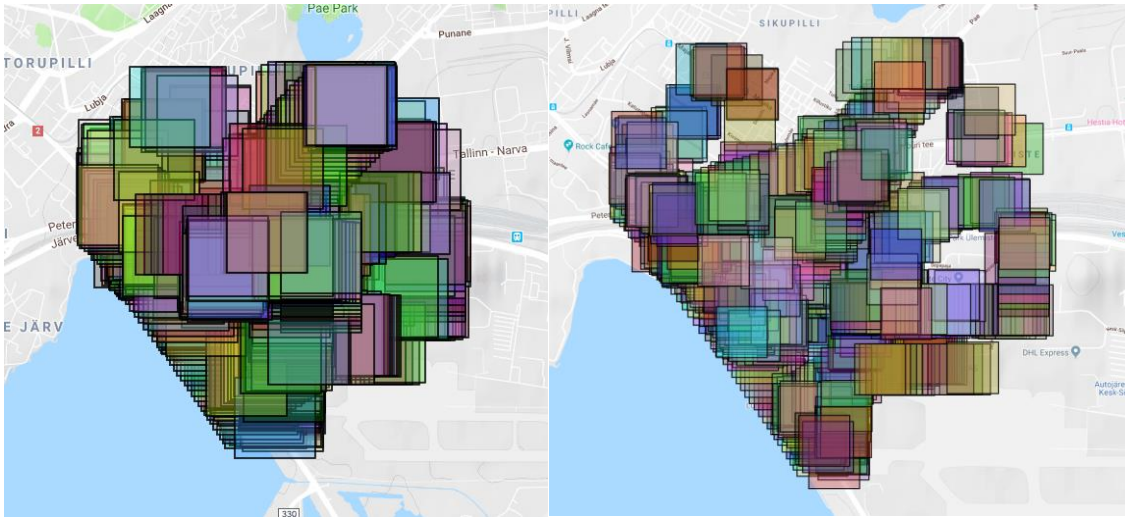
Joonis 21. Punkti *buffer* ja *envelope*

4.3 Parameetritele vastavuse võrdlus

Parameetritele vastavuse testimiseks tuleb vahetada parameetreid ja juba tegelikult visuaalselt on võimalik tuvastada, kas programmis erines polügoonide osas erisusi. Võrdluse üheks pooleks on „Tarkvaraarenduse meeskonnaprojekt“ raames tehtud rakendus ja teiseks pooleks laiendustega rakendus.

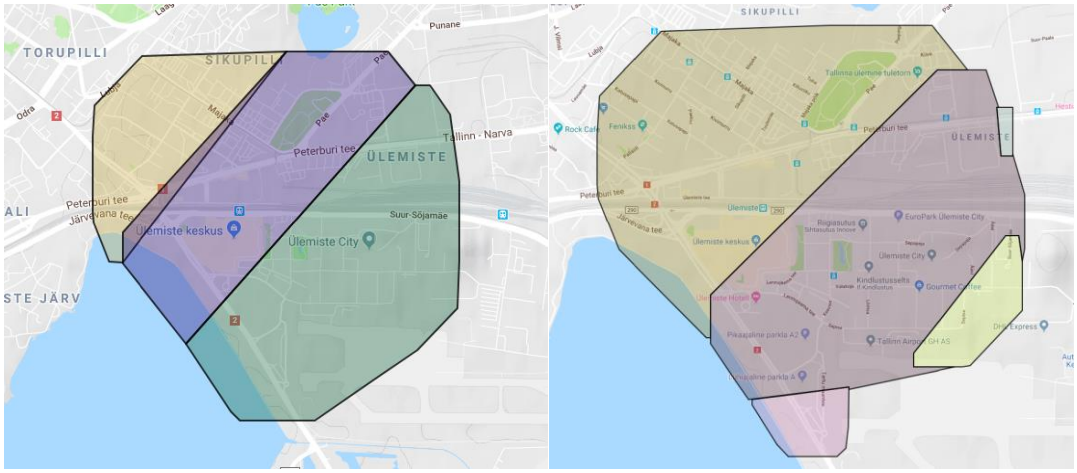
Alustuseks on mõlema rakenduse maksimum pindala polügoonide liitmiseks on 1000000 ruutmeetrit ja polügoonist laiendatud ruudu külje pikkus 200 meetrit. Lähteandmeteks on punktidega fail. Punkti laiendustega esineb probleem varasemal

rakendusel, mis on joonisel 22 vasakpoolne. Nimelt testimisel ilmes probleem punkti laiendusega, kus koostati punktist liiga suur nelinurkne polügoon.



Joonis 22. Punktide laienduse võrdlus

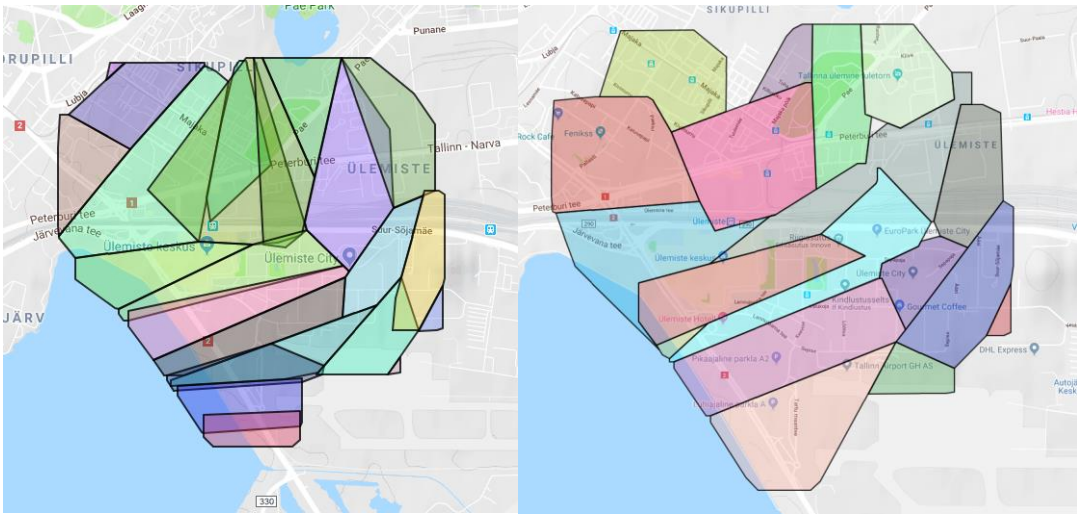
Järgnevalt on võrdluseks mõlema punkti faili lõpplahendus:



Joonis 23. Punktide lõpplahenduse võrdlus

Vasakpoolsemalt, ehk varasemast lahendusest on märgata, et polügoonid kattuvad, kuigi polügoonid näevad võrdsemad välja. Laiendustega lahendusel, ehk parempoolsel puudub ülekattumine aga polügoonid pole päris võrdsed.

Parameetritele reageerimise võrdluseks on mõlema rakenduse polügoonide liitmise maksimum pindala vähendatud 100000 ruutmeetri peale, nelinurga külje pikkus on sama. Esialgne rakendus üldse väiksema maksimum pindala peale ei töötanud ja tuli teha väikesed parandused ka esialgses rakenduses, et üldse tulemus kätte saada.



Joonis 24. Punktide lõpplahenduse võrdlus muudetud parameetriga

Joonisel 24 varasemal rakendusel (vasakpoolne) on ülekattumised, mis tekitaksid lõppkasutajale ebamugavust, kuna kõik mittesobivad polügoonid peaks lõppkasutaja kaardirakenduses käsitsi parandama. Laiendustega rakendusel, ehk parempoolsel puuduvad ülekattumised ja polügoon vastab nõuetele.

4.4 Rakenduste kiirused

Uue rakenduse paremuse mõõtmiseks võib võtta rakenduse kiiruse sama probleemi lahendamisel. Esimene mõõtetulemustes kasutatud rakendus on „Tarkvaratehnika meeskonnaprojekt“ raames tehtud rakendus. Teine rakendus on täienduste ja laiendustega, mis peaks olema lisaks uuele funktsionaalsusele ka paremini optimiseeritud. Et tulemusi ei mõjutaks rakenduse väline API, on eemaldatud mõõtmise ajal asukohtade otsimine. Mõõtetulemuste mõõteühikuks on millisekundid. Mõlemas rakenduses on kasutatud samu parameetreid.

Tabel 1. Rakenduse kiiruse võrdlus

Faili nimi	Polügoonide arv, mis võivad liituda	Alguses keskmine kiirus (ms)	Peale parandusi keskmine kiirus (ms)	Alguses parim kiirus (ms)	Peale parandusi parim kiirus (ms)
punktid_1.geojson	1026	13013,5	3529,8	11372	2817
jooned.geojson	0	878,8	927,5	705	705
ristmik_1.geojson	0	313,5	407,5	196	198
ristmik_2.geojson	0	222,1	298,1	100	111
hooned_1.geojson	167	403,2	410,1	188	129
hooned_2.geojson	998	1676,8	1286,0	1210	895

Tabelist 1 on näha, et punktide lugemine ja polügoonideks teisendamine paranes kõige rohkem. Veel võib märgata, et mida rohkem polügoone kuuluvad liitmisele, siis seda kiirem on täiendatud rakendus. Kuna ristmike ja joonte failid koosnesid teedest ja teid liideti välispiirjoonte kaudu, siis nende polügoonide liitmise arv on 0, kuna neid ei liidetud, kui polügoone, vaid kui jooni ja joonte liitmisele olid eraldi nõuded.

5 Võimalikud edasiarendused

Loodud rakendus asub aadressil https://gitlab.cs.ttu.ee/frvarb/iapb, vaatamata sellele, et käesoleva bakalaureuse töö raames valminud laiendustega rakendus testitud failide korral vastab nõuetele, on mõnda aspekti võimalik veel parandada.

Võimalikke polügoone, nende liitumise viise on palju ja kõiki neist pole testitud, seega on võimalik rakendust parandada genereerides keerulisemaid polügoone ja proovida testide kaudu neid liita. EPSG koode hoitakse praegu XML failis, mis tähendab, et muudatuste tegemisel peab *Ucanaccess* draiveriga lugema sisse MS Access failist soovitud EPSG koodid ja genereerima XML faili uuesti. ESPG koodide MS Access faili uuendatakse umbes 2 korda aastas [16]. EPSG koodide täiendamiseks on vähemalt 2 võimalust: lugeda EPSG koodid andmebaasist; teha ümber rakendus nii, et EPSG koode ei peaks kunagi lugema, kasutades *Shape* faili korral .dbf faili ja GML faili korral *srsName* välja.

Kuna ülesande püstituses soovitatud API on aeglane, on võimalik maksta ja saata kõik koordinaadid, millele soovitakse aadressi, korraga. Selleks on üheks paljudest näiteks *maplarge* [25].

6 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua ja täiendada nõuetele vastav parameetriseeritav API, mis aitab lugeda geoandmetega faile ja neid töödelda sobivateks polügoonideks. Täienduste hulka kuulusid koordinaatsüsteemi vahetamine, GML faili lugemine, testimine, tõlkimine, olemasoleva algoritmi parandamine, x- ja y-telje vahetamine ja muud täiendused.

Eesmärkide täitmiseks tutvuti kõigepealt rakenduse tööks vajalike algandmetega, seejuures teekide, raamistiku valiku, failide, koordinaatsüsteemide vahetuse, klasside struktuuri, klasside sisendite ja väljunditega. Seejärel kirjeldati joonest polügooni loomist, polügoonide kokku liitmist ja toodi lihtsamaid näiteid. Valideerimiseks kirjutati automaattestid *Spring* rakendusele kohaselt. Võrreldi varasema rakenduse ja bakalaureuse töö raames valminud täiendustega rakenduse parameetritele vastavust ja kiirust. Viimaks tehti veakoodide haldus ja tõlge vastavalt veebilehitsejast saadetud päisele.

Vastavalt valideerimisele, said töö eesmärgid täidetud. Valmis täiendatud API koos täiendatud kasutajaliidesega, kus lisaks teistele failidele, saab GML faili üles laadida ja tänu koordinaatsüsteemide vahetamise võimalusele, on nüüd ka võimalik laadida faile, mis pole ainult WSG84 süsteemis ehk ka välismaalt. Saadud failidest luuakse parameetritele vastavad polügoonid ja edastatakse need JSON formaadis.

Kasutatud kirjandus

[1] Datel AS / Ovela LLC, „Sille,“ [Võrgumaterjal]. Available: https://sille.space/en/about. [Kasutatud 6 Aprill 2019].

[2] Datel AS, *Infrastruktuuriobjektide maa-alade mitmik-importimine ja töötlemine,* Tallinn, 2018.

[3] O. S. G. Foundation, „GeoTools,“ [Võrgumaterjal]. Available: https://www.geotools.org/about.html. [Kasutatud 9 aprill 2019].

[4] I. Free Software Foundation, „GNU Lesser General Public License,“ [Võrgumaterjal]. Available: https://www.gnu.org/licenses/lgpl-3.0.en.html. [Kasutatud 17 aprill 2019].

[5] E. Foundation, „Eclipse Public License,“ [Võrgumaterjal]. Available: https://www.eclipse.org/legal/epl-v10.html. [Kasutatud 17 aprill 2019].

[6] E. Foundation, „Eclipse Distribution License,“ [Võrgumaterjal]. Available: https://www.eclipse.org/org/documents/edl-v10.php. [Kasutatud 9 aprill 2019].

[7] O. S. G. Foundation, „JTS Topology Suite,“ [Võrgumaterjal]. Available: https://www.osgeo.org/projects/jts/. [Kasutatud 4 aprill 2019].

[8] P. Software, „Spring,“ [Võrgumaterjal]. Available: https://spring.io/. [Kasutatud 9 aprill 2019].

[9] J. Hoeller, „Annotation Type Value,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotation/Value.html. [Kasutatud 9 aprill 2019].

[10] R. Paks, „JAVASCRIPT NODE.JS NING JAVA SPRING PLATVORMIL REALISEERITUD VEEBIRAKENDUSTE ANALÜÜS NING VÕRDLUS,“ Tallinn, 2018.

[11] O. G. C. Inc, „OpenGIS® Web Map Server Implementation Specification,“ [Võrgumaterjal]. Available: http://www.opengeospatial.org/standards/wms. [Kasutatud 13 aprill 2019].

[12] D. Formats, „ Sustainability of Digital Formats: Planning for Library of Congress Collections,“ [Võrgumaterjal]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000280.shtml. [Kasutatud 9 aprill 2019].

[13] O. S. G. Foundation, „Shapefile Plugin,“ [Võrgumaterjal]. Available: http://docs.geotools.org/stable/userguide/library/data/shape.html. [Kasutatud 11 aprill 2019].

[14] O. G. C. Inc., „Geography Markup Language (GML) simple features profile,“ [Võrgumaterjal]. Available: https://www.opengeospatial.org/standards/gml. [Kasutatud 11 aprill 2019].

[15] O. S. G. Foundation, „Geometry,“ [Võrgumaterjal]. Available: http://docs.geotools.org/stable/userguide/library/xml/geometry.html. [Kasutatud 11 aprill 2019].

[16] I. A. o. O. &. G. Producers, „EPSG home,“ [Võrgumaterjal]. Available: http://www.epsg.org/EPSGhome.aspx. [Kasutatud 9 aprill 2019].

[17] „Interface Marshaller,“ [Võrgumaterjal]. Available: https://docs.oracle.com/javase/8/docs/api/javax/xml/bind/Marshaller.html. [Kasutatud 28 aprill 2019].

[18] O. G. C. Inc., „OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1 : Common architecture,“ 28 mai 2011. [Võrgumaterjal]. Available: http://www.opengeospatial.org/standards/sfa. [Kasutatud 9 aprill 2019].

[19] J. P. Snyder, „Map projections: A working manual,“ [Võrgumaterjal]. Available: https://pubs.er.usgs.gov/publication/pp1395. [Kasutatud 13 aprill 2019].

[20] I. Kosandyak, „Spring Boot REST Internationalization,“ [Võrgumaterjal]. Available: https://blog.usejournal.com/spring-boot-rest-internationalization-9ab3fce2489. [Kasutatud 17 aprill 2019].

[21] R. S. Juergen Hoeller, „Class AcceptHeaderLocaleResolver,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/i18n/AcceptHeaderLocaleResolver.html. [Kasutatud 17 aprill 2019].

[22] Pivotal, „Testing,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html. [Kasutatud 20 04 2019].

[23] C. S. D. S. J. L. Howard Butler, „EPSG:4326,“ [Võrgumaterjal]. Available: https://spatialreference.org/ref/epsg/wgs-84/. [Kasutatud 20 aprill 2019].

[24] OSGeo, „Class Envelope,“ [Võrgumaterjal]. Available: https://locationtech.github.io/jts/javadoc/org/locationtech/jts/geom/Envelope.html. [Kasutatud 20 aprill 2019].

[25] MapLarge, „Reverse Geocoding Online and Reverse Geocoder API - GPS Coordinates to Address,“ MapLarge, [Võrgumaterjal]. Available: https://maplarge.com/reversegeocoding. [Kasutatud 16 Mai 2019].

[26] M. Amadei, „UCanAccess,“ [Võrgumaterjal]. Available: http://ucanaccess.sourceforge.net/site.html#home. [Kasutatud 9 aprill 2019].

[27] M. team, „EPSG:4326,“ [Võrgumaterjal]. Available: https://epsg.io/4326. [Kasutatud 9 aprill 2019].

Lisa 1 – nõuded Datel AS'ilt [2]

Infrastruktuuriobjektide maa-alade mitmik-importimine ja töötlemine

Probleem:

Selleks, et analüüsida objekte maismaal satelliit piltide abil vajab analüüsitulemeid kuvav rakendus nende objektide selgeid piiritletud maa-alasi (bounding box) (polügoonidena). Kõige lihtsam on lasta kasutajatel antud polügoone käsitsi joonistada või edastada need koordinaatidena. Paraku on antud teguviis kasutaja jaoks aeganõudev ning osutub eriti problemaatiliseks juhul kui analüüsida on vaja suurt hulka objekte.

Eesmärk:

Kuna polügoonide koostamine käsitsi (või üksikult mõne kaardirakenduse abil) on väga aeganõudev protsess võimaldaks loodav tarkvara importida kasutajate objekte suurel hulgal ja korraga (toetatud oleks levinumad formaadid näiteks: KML, geoJSON, SHP). Valminud tarkvara integreeritakse olemaseolevasse komponentide ahelasse ja see loob kasutaja jaoks eeldused paremaks kasutuskogemuseks – muutes rakenduse töö ka kliendimugavuse kiiremaks ja lihtsamaks.

Lahendus:

Lahendus oleks kui saaksime loodava tarkvarapaketi abil anda sisendi levinuimate geograafiliste andmete failiformaatide alusel. Minimaalselt peaksid olema toetatud KML, geoJSON, SHP formaadid. Toetatud sisendformaatide maksimaalne arv ei ole piiratud.

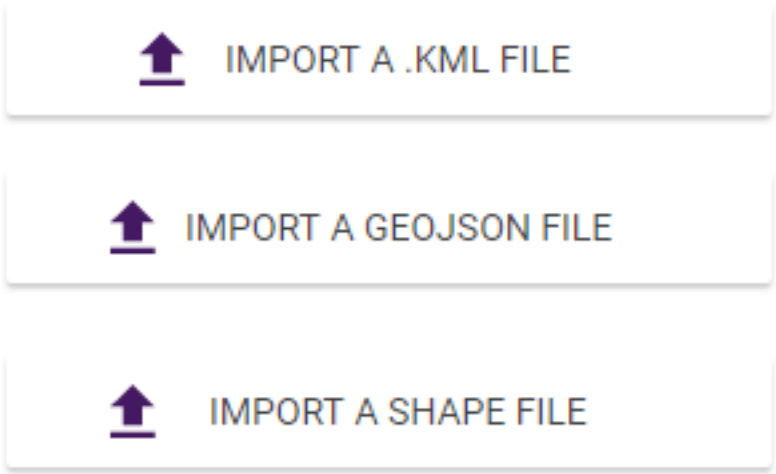
Peale kasutajaliidesest antud sisendi andmist korrektseks formaadis toimuks formaadi kontroll, valideerimine (erinevate veateadete kuvamine). Veateated peavad olema inglise keeles, lisaks peab olema võimalus lisada ka eestikeelsete tõlgete tugi.

Korrektse faili puhul töötleks loodav tarkvara imporditud faili ringi polügoonideks JSON formaadis ning tagastaks selle väljundina. Lisaks võiks loodav tarkvara geokodeerimise alusel tuvastada objekti asukoha riigi ning osariigi või maakonna tasemel (mida detailsem seda parem).

Kasutada võib ka vabalt valitud lahendusekäiku ja algoritmi.

Näide:

1. Satelliitseire analüüsitulemusi kuvav rakendus omab näiteks nuppu sisendi pakkumiseks loodavale tarkvarale:



2. Tarkvara töötleb faili ning kuvab vea korral vastava teate.

3. Vigade puudumisel tagastab JSON'i objekti(de) polügoonide ja *meta*-infoga


```
▼ location: {type: "Polygon",…}
  ▼ coordinates: [[[-76.41286520435112, 39.01797553358721], [-76.41044816478049, 39.01555642511512],…]]
    ▼ 0: [[-76.41286520435112, 39.01797553358721], [-76.41044816478049, 39.01555642511512],…]
      ▶ 0: [-76.41286520435112, 39.01797553358721]
      ▶ 1: [-76.41044816478049, 39.01555642511512]
      ▶ 2: [-76.3991646318291, 38.99838889355856]
      ▶ 3: [-76.39548226005414, 38.9956037283206]
      ▶ 4: [-76.31964284229508, 38.97735946080398]
      ▶ 5: [-76.3190737475331, 38.97947537319976]
      ▶ 6: [-76.33209037629224, 38.98262851486322]
      ▶ 7: [-76.33705247977768, 38.98484259057427]
      ▶ 8: [-76.3929843766052, 38.99813140060284]
      ▶ 9: [-76.39575005464084, 38.99984039030653]
      ▶ 10: [-76.40752226778855, 39.01731881078261]
      ▶ 11: [-76.41151274944541, 39.02021212532605]
      ▶ 12: [-76.41193336157231, 39.02039299578774]
      ▶ 13: [-76.41286520435112, 39.01797553358721]
    type: "Polygon"
  name: "Chesepeake Bay Bridge"
```

Näiteks:

4. Väljundi alusel peab olema võimalik kasutajale kuvada näiteks:

Nõuded:

1. Komponent peab olema programmselt kasutatav nö API-na, ilma inimese vahele sekkumiseta.
2. Komponendil peavad olema sisendid ja väljundid, mida on võimalik integreerida teiste komponentidega.
3. Tarkvara peab olema nõuetekohaselt testitud ja koos kasutusjuhendiga.
4. Tarkvaral peab olema paigaldusjuhend ja nimekiri kõikidest alamkomponentidest koos versiooninumbritega.
5. Tarkvara peab põhinema 100% kommertskasutust võimaldaval vabavara teekidel.

AS Datel soovib tulemina probleemi lahendava tarkvara, mis vastab nõuetele. Nõuetele vastav tarkvara võetakse reaalselt kasutusele.

Kasutatavad tehnoloogiad:

Eelistatult JAVA või NodeJS paketina *(NODE runtime package)* - NPM kaudu paigaldatav.

Töömaht, ajakava ja aste:

Maht: ~500h:

Ajakava: vaba ehk võib kohe alustada ja lõpetada hiljemalt juunis või juulis.

Aste: bakalaureus

Alusandmed:

Asukoha riigi leidmiseks võib kasutada vaba teeki:

https://wiki.openstreetmap.org/wiki/Nominatim

Vahendid:

Koolitus ja juhendamine

Lisa 2 – *Ucanaccess* draiveri kasutamine

Maven dependency:

```
<dependency>
    <groupId>net.sf.ucanaccess</groupId>
    <artifactId>ucanaccess</artifactId>
    <version>4.0.4</version>
</dependency>
```

JAVA kood:

```
public List<DtoEPSG> read() throws SQLException {
    try {
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        System.out.println("Loaded Ucanaccess");
    } catch (ClassNotFoundException e) {
        throw new
IllegalArgumentException("failed.message.ucanaccess");
    }
    System.out.println("Connecting");
    Connection conn = DbReader.getConnection();
    System.out.println("Connection established");
    Statement s = conn.createStatement();
    ResultSet rs = s.executeQuery("SELECT * FROM [Coordinate Reference
System]");
    List<DtoEPSG> result = new ArrayList<>();
    while (rs.next()) {
        DtoEPSG dto = new DtoEPSG();
        dto.setCode(rs.getString(1));
        result.add(dto);
    }
    return result;
}

private static Connection getConnection() throws SQLException {
    return
DriverManager.getConnection("jdbc:ucanaccess://C://Users//random//Desk
top//db//EPSG_v9_6.mdb", "", "");
}
```