

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 073

**Automatizirana procjena prostorne
točnosti skupova podataka
opažanja u prirodi**

Dominik Kniewald

Zagreb, srpanj 2023.

SADRŽAJ

1. Uvod	1
2. Geoparsiranje podataka	3
2.1. Koncept	3
2.2. Primjena	4
3. Arhitektura i dizajn sustava	6
3.1. Baza Podataka	7
3.2. Dijagram komponenti	10
3.3. Dijagram razmještaja	11
3.4. Dijagram aktivnosti	12
4. Implementacija	13
4.1. Poslužiteljska strana	13
4.1.1. Integracija Baze Podataka	13
4.1.2. Poslužitelj	14
4.1.3. Kreiranje Geonames indeksa	17
4.2. Klijentska strana	19
4.2.1. React Komponente	19
4.2.2. Virtualni DOM	19
4.2.3. React aplikacija	20
4.3. Postavljanje aplikacije za javnu upotrebu	26
4.3.1. Virtualni stroj	26
4.3.2. Postavljanje klijentske strane	27
4.3.3. Postavljanje poslužiteljske strane	28
5. Korištene tehnologije	29
5.1. Poslužiteljska strana	29
5.1.1. REST API	30

5.1.2. JSON i XML formati podataka	30
5.2. Klijentska strana	31
5.2.1. HTML i CSS	32
5.3. Postavljanje aplikacije za javnu upotrebu	33
6. Korištenje aplikacije	35
6.1. Geoparsiranje excel podataka	35
6.2. Geoparsiranje teksta	37
6.3. Pretraga mjesta	38
7. Testiranje i evaluacija aplikacije	39
8. Zaključak	41
Literatura	42

1. Uvod

U vremenu digitalizacije i brzog protoka informacija, sposobnost pravilnog i točnog geopozicioniranja bioloških opažanja postaje sve važnija. U Hrvatskoj se prikuplja ogroman broj bioloških podataka zbog svoje jako velike biološke raznolikosti. S obzirom na brojnost i raznolikost taksonomskih baza podataka nedostatak kvalitetne kontrole i ispravnosti prostornih podataka predstavlja ozbiljan izazov. Razloga za pogreške koje su vezane za prostorne podatke može biti više, od poteškoća s ručnim unosom koordinata do manjka iskustva istraživača u korištenju GPS opreme (Filip Varga, 2023).

Podaci koji se prikupljaju obično će u sebi uključivati informacije poput datuma, vremena, lokacije, vrste, broja jedinki, ponašanja i drugih relevantnih podataka. Prikupljanje tih podataka predstavlja složen proces u kojem su kvaliteta i točnost izuzetno važni. Postoje brojni izazovi s kojima se stručnjaci mogu susresti prilikom rada s ovom vrstom podataka, a među najčešćima su nedostatak standardizacije, greške u identifikaciji, dupli unosi te poteškoće u bilježenju i čuvanju informacija.

Osiguranje kvalitete podataka iz bioloških opažanja može se provesti na više načina. Prije svega, standardizacija protokola za prikupljanje podataka omogućuje dosljedno prikupljanje podataka, što olakšava kasniju analizu. Obuka promatrača ključna je za smanjenje pogrešaka i nejasnoća, a provjera podataka od treće strane pri unosu može otkriti potencijalne greške. Uz to, korisni su alati i softveri za upravljanje podacima koji mogu automatski provjeravati valjanost i konzistenciju podataka.

Automatizirani alat za ocjenu prostorne točnosti značajan je zbog povećanja kvalitete prikupljenih podataka, a samim time omogućuje se i učinkovitija upotreba tih podataka u različitim primjenama. Ovakav alat bi znanstvenicima mogao omogućiti bolje shvaćanje i preciznije predviđanje distribucije endemičnih i ugroženih vrsta. Također se mogu smanjiti rizici koje invazivne vrste mogu predstavljati za biološku raznolikost.

Geoparsing na hrvatskom jeziku predstavlja izazov zbog nekoliko faktora. Morfološka složenost hrvatskog jezika često otežava prepoznavanje toponima, budući da riječi mijenjaju oblik ovisno o gramatičkom kontekstu. Osim toga, može biti teško razumjeti kontekst zbog mogućnosti različitih sintaktičkih interpretacija i upotrebe raz-

ličityh naziva za istu lokaciju. Također, postojeći alati za geoparsing često su optimizirani za engleski, što znači da resursi za hrvatski jezik mogu biti ograničeni.

Cilj ovog rada bio je istražiti i predložiti mogućnost razvoja takvog alata. Njegova svrha je s jedne strane pomoći učiniti skupove podataka o biološkim opažanjima točnijim i pouzdanijim, a s druge strane omogućiti istraživačima i znanstvenicima pristup kvalitetnim i točnim prostornim podacima.

2. Geoparsiranje podataka

Geokodiranje je proces uzimanja nedvosmislene strukturirane reference lokacija, kao što su poštanske adrese ili konkretne numeričke koordinate kako bi se identificirala lokacija na Zemlji. Obrnuto geokodiranje je proces pretvaranja geografskih koordinati u čovjeku razumljive lokacije, na primjer naziv mjesta ili adresa.

Geoparsiranje ide korak dalje od geokodiranja tako što uzima dvosmislene reference iz nestrukturiranih komunikacijskih kanala i pretvara tu referencu u geografsku lokaciju. Ako se naiđe na pojam kao što je „Al Hamra”, što je naziv za više mjesta uključujući gradove u Jemenu i Siriji gleda se kontekst ili se traži neka fraza koja će specificirati lokaciju. Također se mogu geoparsirati reference lokacija iz drugih oblika medija, na primjer audio sadržaj u kojem govornik spominje mjesto. Dvije primarne upotrebe geografskih koordinata izvedenih iz nestrukturiranog sadržaja su iscrtavanje dijelova sadržaja na kartama i pretraživanje sadržaja pomoću karte.

S geografskim koordinatama značajke se mogu mapirati te unijeti u geografske informacijske sustave. Geografski informacijski sustav (GIS) je vrsta baze podataka koja sadrži geografske podatke u kombinaciji s alatima za upravljanje, analizu i vizualizaciju tih podataka. GIS pruža mogućnost povezivanja prethodno nepovezanih informacija. Sve lokacije se mogu zabilježiti kroz datum i vrijeme nastanka uključujući s koordinatama koji predstavljaju zemljopisnu dužinu, širinu i nadmorsku visinu. Sve lokacije i reference bi trebale biti povezane te se pomoću tog skupa informacija dobiva neko stvarno područje. Ovo je jedna od ključnih karakteristika koja je počela otvarati nove puteve znanstvenih istraživanja i proučavanja.

2.1. Koncept

Geoparser je alat koji služi za identifikaciju i pridruživanje geografskih koordinata geografskim entitetima (poput imena gradova, država, rijeka, planina itd.) koji se nalaze unutar nekog teksta. Alat je često dizajniran da funkcionira kao cjevovod tako da je rezultat jednog procesa dio ulaza za idući proces. Ovakav dizajn omogućava fleksi-

bilnost pri odabiru nekih unutarnjih funkcionalnosti, na primjer ako korisnik želi iskoristiti svoj program za tokeniziranje teksta jednostavno može zamijeniti s postojećom funkcionalnošću te će sve ostalo nastaviti raditi nezavisno.

Cjevovod alata za geoparesiranje uključuje sljedeće korake:

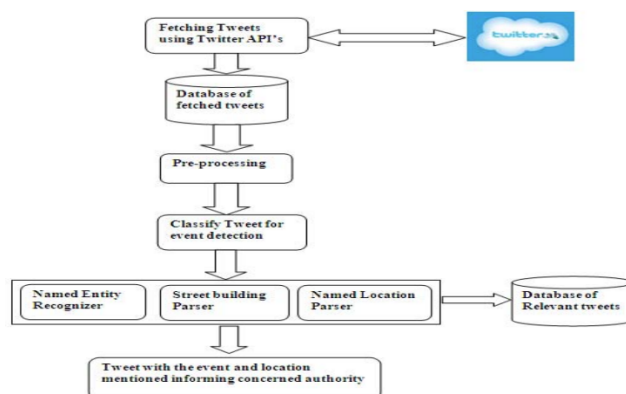
1. Tokenizacija - Razbijanje teksta na pojedinačne riječi ili "tokene". Na primjer, rečenica "Ja živim u New Yorku" bi se razbila na tokene: ["Ja", "živim", "u", "New", "Yorku"]
2. Prepoznavanje imenovanih entiteta (NER - Named Entity Recognition) - Identifikacija imenovanih entiteta, uključujući geografske entitete, u tekstu. Na primjer, u rečenici "Ja živim u New Yorku", "New York" bi bio prepoznat kao imenovani entitet.
3. Normalizacija imenovanih entiteta - Mapiranje imenovanih entiteta na standardizirane forme. Na primjer, "NYC", "New York City" i "The Big Apple" bi sve bile mapirane na "New York".
4. Geokodiranje - Povezivanje imenovanih geografskih entiteta s konkretnim geografskim koordinatama.
5. Rezolucija prostorne neodređenosti - Rješava slučajeve kada je jedno ime mjesta povezano s više mogućih lokacija uzevši u obzir kontekst.

2.2. Primjena

Geoparsiranje podataka može imati razne primjene, jedna od njih je identifikacija lokacija zločina i nesreća pomoću geoparsiranja tweetova. Zbog velike popularnosti Twittera kao platforme i dostupnosti Interneta i mobitela svugdje nerijetko se događa da je ljudi objave na Twitteru neki događaj koji su proživjeli ili kojemu su svjedočili. Zato pomoću prepoznavanja takvih situacija moguće je puno brže obavijestiti nadležne ustanove kako bi mogli na vrijeme reagirati. Uzevši u obzir sve navedeno napravljen je program koji upravo to i radi. Program dohvaća tweetove pomoću Twitterovog API-ja koji se spremaju u bazu podataka. Iz te baze podataka se uzimaju tweetovi koji se procesiraju te klasificiraju u grupu ovisno o vrsti događaja pomoću naivnog Bayesovog klasifikatora. Slika 2.1 prikazuje usporedbu različitih klasifikatora, a Slika 2.2 cijeli tijek procesa od objave tweeta do mogućnosti obavještavanja nadležnih institucija (Dhavase, 2014).

Classifier Name	Correct Classification percentage
ZeroR	44.2623 %
FilteredClassifier	85.3659 %
MultiScheme	41.4634 %
NaiveBayesMultinomialtext	93.4426 %

Slika 2.1: Preciznost naivnog Bayesovog klasifikatora (Dhavase, 2014)



Slika 2.2: Dijagram toka (Dhavase, 2014)

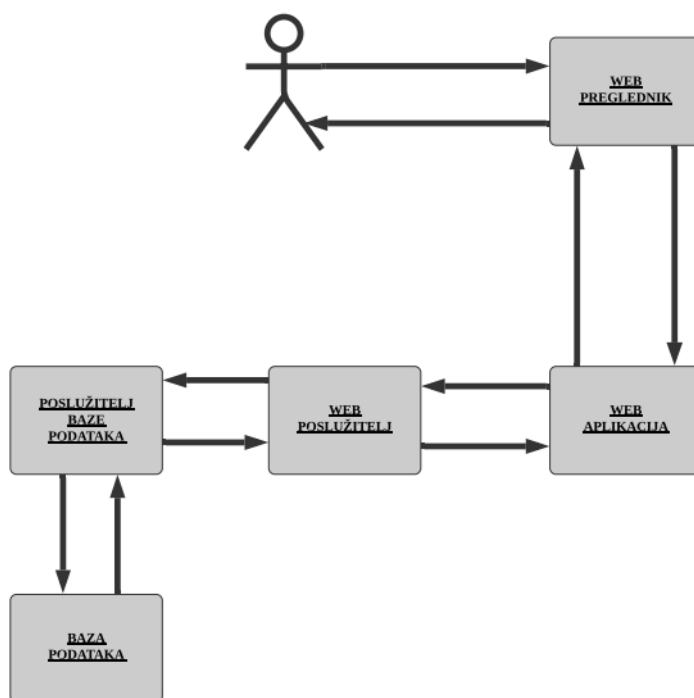
Također se velika primjena nalazi u geoparsiranju povijesnih zapisa. Naime, razumijevanje lokacije je ključan dio svakog povijesnog istraživanja i to vrlo precizno automatizirano geografsko referenciranje omogućuje povjesničarima otkriti informacije koje se odnose na regije izvan naziva mjesta kojeg traže. Kad bi sve digitalne zbirke bile georeferencirane pružila bi se mogućnost da se udruživanjem različitih resursa dođe do željenih rezultata (Rayson, 2013).

U svrhu ovog rada geoparsiranje se koristi u automatizaciji procjena prostorne točnosti skupova podataka opažanja u prirodi. Zbog velikog broj zapisa koji nisu uvijek prikupljeni od profesionalnih osoba postoji potreba za provjerom postojećih podataka u bazama podataka kako bi se eliminirali krivi zapisi.

3. Arhitektura i dizajn sustava

Arhitektura koja je korištena pri izradi aplikacije je klijent-poslužitelj te se sastoji od:

- Poslužitelj Weba
- Aplikacije Weba
- Baza podataka
- Poslužitelj baze podataka



Slika 3.1: Arhitektura sustava

Prikaz web stranice, uključujući sve podatke i zahtijevane medije, omogućuje preglednik weba. Preglednik weba prezentira stranicu na ljudski čitljiv način i pruža korisniku alat za slanje zahtjeva poslužitelju weba.

Sljedeća ključna funkcija je interakcija korisnika s aplikacijom, što omogućuje poslužitelj weba koji je svojevrsni motor web aplikacije. Komunikacija se odvija putem HTTP protokola koji se koristi za prijenos informacija i interakciju na internetu.

Web aplikacija se pokreće putem poslužitelja weba koji obrađuje korisničke zahtjeve. Na temelju tih zahtjeva, poslužitelj weba komunicira s bazom podataka i vraća podatke koji se zatim prikazuju korisniku. Za vizualni prikaz odgovoran je poslužitelj weba koji generira HTML dokument kao odgovor, dok je za čitljiv prikaz korisniku zadužen preglednik weba.

Poslužitelj baze podataka ispunjava ulogu komunikacije s bazom podataka. Dakle, on zaprima i obrađuje zahtjeve za podacima poslane iz web aplikacije koje mu proslijeđuje poslužitelj weba. Ovisno o vrsti zahtjeva, u odgovorima se mogu nalaziti zatraženi podatci ili informacije o uspješnosti nekih operacija.

Aplikacija se dijeli na klijentsku i poslužiteljsku stranu. Poslužiteljska strana se bavi komunikacijom s bazom podataka, dohvatom i obradom istih, izrađena je koristeći Python specifično uz pomoć Python biblioteke Mordecai koji obrađuje posao geoparsiranja. S druge strane, klijentska strana obuhvaća prikaz korisničkog sučelja i slanje zahtjeva za podacima prema poslužitelju, izrađen je koristeći React.js.

Velika prednost ove arhitekture je što omogućava nezavisni razvoj klijentske i poslužiteljske strane, pružajući veliku fleksibilnost i učinkovitost. Također, jedan dio sustava ne ovisi o drugom, što znači da se svaka komponenta može zamijeniti drugačijom implementacijom bez utjecaja na ukupnu funkcionalnost sustava.

3.1. Baza Podataka

Mordecai, kao biblioteka za geoparsiranje, koristi `geonames_index` - veliki geografski indeks koji je sastavni dio Geonames baze podataka.

Geonames baza podataka je besplatna i otvorena baza podataka koja sadrži geografske informacije o svijetu. Obuhvaća preko 11 milijuna geografskih imena, uključujući imena zemalja, regija, gradova pa čak i značajnih toponima. Osim imena, svaki unos u bazu podataka sadrži dodatne informacije poput geografske širine i dužine, nadmorske visine, populacije i mnogo više. Također je alat testiran sa bazom podataka koja se temelji na nacionalnom registru geografskih imena.

Mordecai koristi Geonames bazu podataka za stvaranje `geonames_indexa`, koji omogućuje brzu pretragu unutar ovog ogromnog skupa podataka. Ovaj indeks je ključan za rad Mordecai-a, jer omogućuje da se svako prepoznato geografsko ime može precizno locirati.

Geonames indeks je specifična struktura podataka koja omogućava brzo i efikasno pretraživanje geografskih informacija koje su potrebne za proces geoparsiranja. Izrađen je tako da optimizira brzinu i učinkovitost pretrage. To se postiže kroz specifičan način organizacije podataka unutar indeksa. Na primjer, podaci mogu biti indeksirani po imenima lokacija, po geografskim koordinatama ili po drugim relevantnim atributima. Ova vrsta indeksa omogućava vrlo brzo dohvaćanje podataka za svaku lokaciju koja je identificirana tijekom procesa geoparsiranja.

Izrada Geonames indeksa uključuje nekoliko koraka. Prvo, podaci iz Geonames baze podataka se preuzimaju i učitavaju u alat za rad s bazama podataka koji je u ovom slučaju Elasticsearch. Zatim se izvršava niz operacija kojima se podaci transformiraju i organiziraju u oblik koji je pogodan za indeksiranje.

Elasticsearch funkcionira na temelju koncepta indeksiranja. Kad Elasticsearch primi podatke za indeksiranje, analizira te podatke i stvara obrnuti indeks koji omogućuje brzu pretragu. Obrnuti indeks je struktura podataka koja povezuje svaku jedinstvenu riječ sa svim lokacijama na kojima se ta riječ pojavljuje. Podaci su indeksirani uz prethodnu definiciju sheme kako bi se poboljšale performanse i kvaliteta pretrage. Zahtjevi za pretragu mogu biti jednostavni, korištenjem samo ključnih riječi, ili složeni, koristeći Elasticsearch-ov Query DSL, jezik za definiranje upita. To omogućava korisnicima da izrađuju složene upite, uključujući upite za pretragu punog teksta, filtriranje, sortiranje i tako dalje.

```
1 {
2     "settings" : {
3         "number_of_shards" : 1,
4         "number_of_replicas" : 1
5     },
6     "mappings" : {
7         "properties" : {
8             "geonameid" : {"type" : "keyword", "index": "true"},
9             "name" : {"type" : "text"},
10            "asciiname" : {"type" : "text"},
11            "alternativenames" : {"type" : "text", "similarity" : "
                boolean",
12                                "norms": false},
13            "coordinates" : {"type" : "geo_point"},
14            "feature_class" : {"type" : "keyword", "index": "true"},
15            "feature_code" : {"type" : "keyword", "index": "true"},
16            "country_code3" : {"type" : "keyword", "index": "true"},
17            "admin1_code" : {"type" : "keyword", "index": "true"},
18            "admin1_name" : {"type" : "keyword", "index": "true"},

```

```

19         "admin2_code" : {"type" : "keyword", "index": "true"},
20         "admin2_name" : {"type" : "keyword", "index": "true"},
21         "admin3_code" : {"type" : "keyword", "index": "true"},
22         "admin4_code" : {"type" : "keyword", "index": "true"},
23         "population" : {"type" : "long"},
24         "alt_name_length": {"type": "long"},
25         "modification_date" : {"type" : "date", "format": "date"}
26     }
27 }
28 }

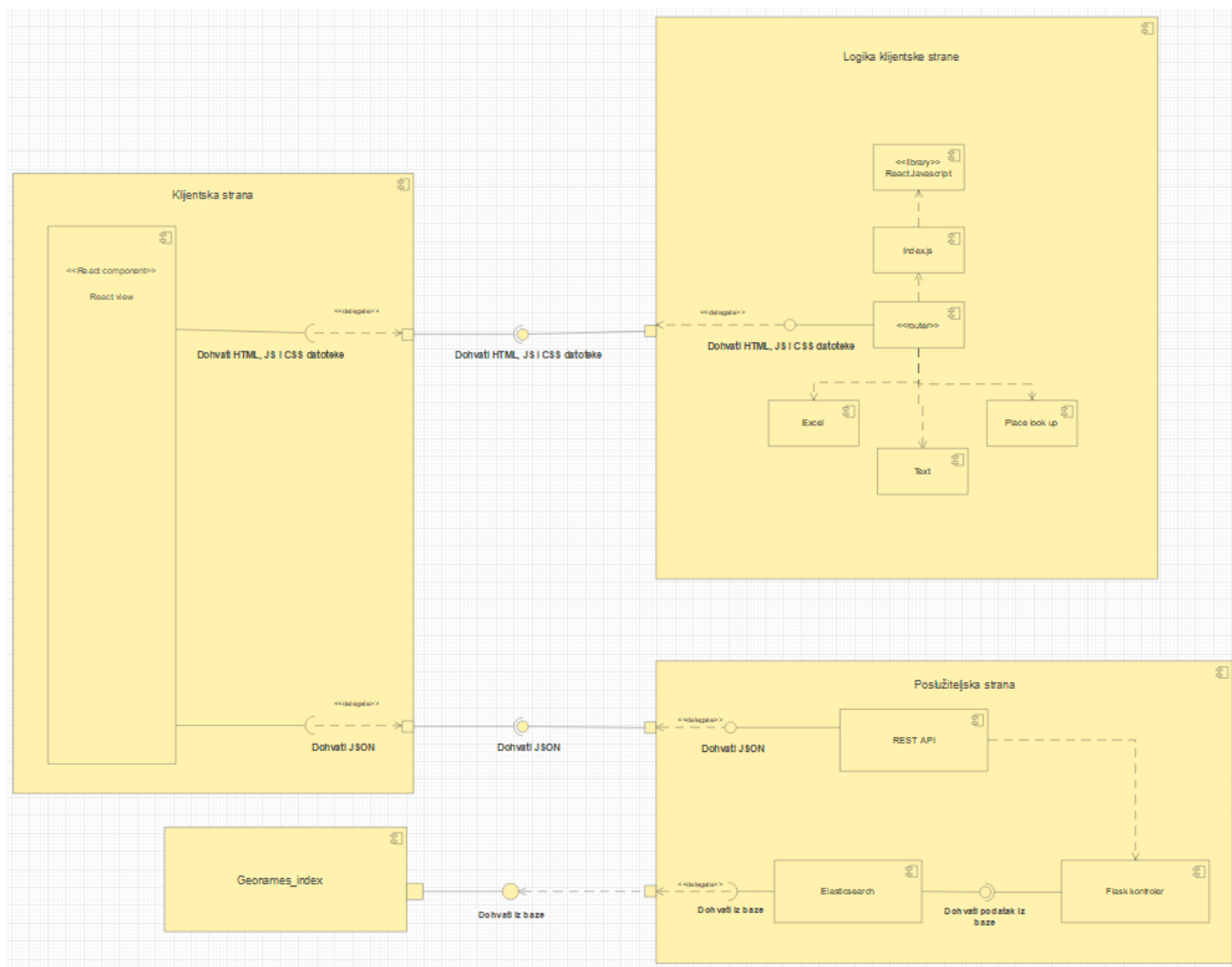
```

Geonames mapiranje 3.1: Definicija sheme

Kad je Geonames indeks izrađen, može se koristiti za pretragu lokacija unutar teksta. Na primjer, kad Mordecai identificira ime grada unutar teksta, može koristiti Geonames indeks da brzo dohvati geografske koordinate tog grada. Ovo omogućava Mordecai-u da vrlo brzo i točno odredi geografske lokacije koje su spomenute unutar teksta.

3.2. Dijagram komponenti

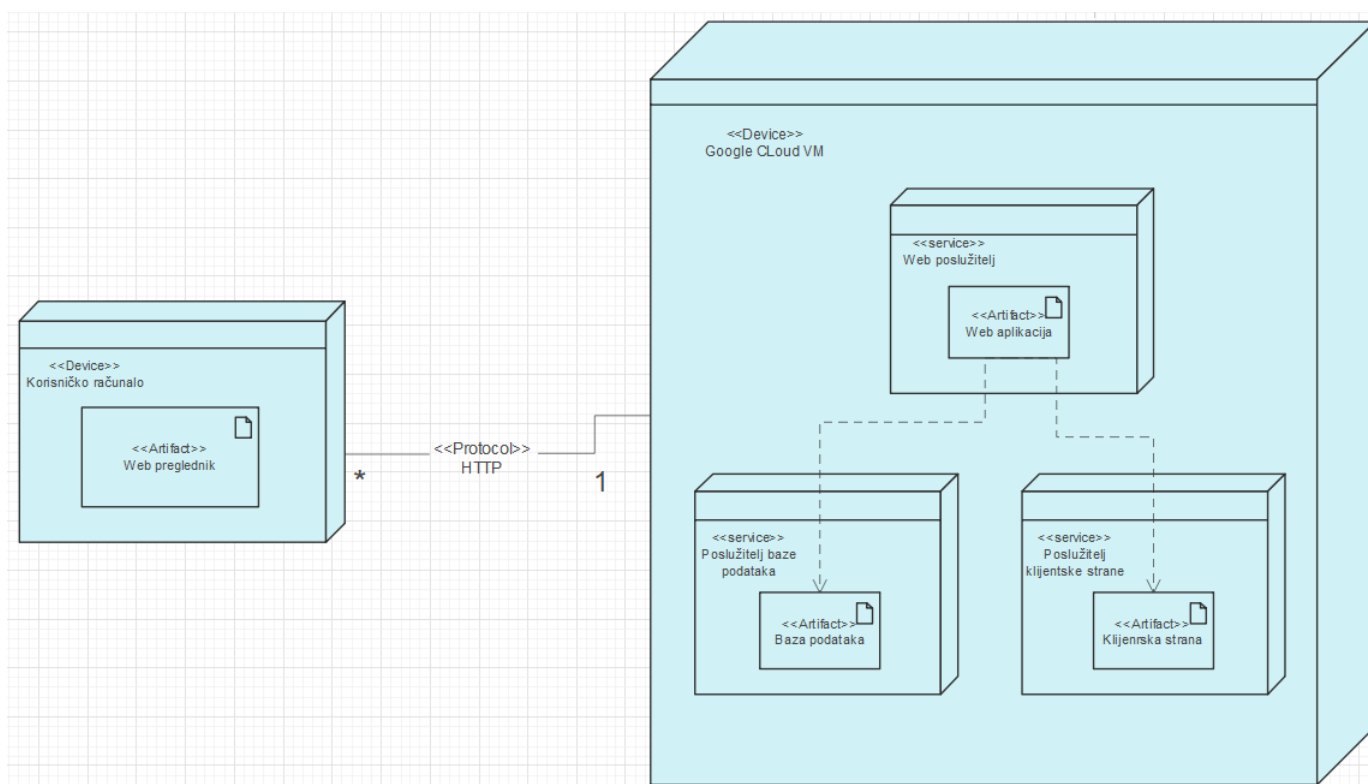
Dijagram komponenti prikazuje međuovisnost i organizaciju komponenti. Kao što prikazuje Slika 3.2 sustavu se pristupa preko nekoliko sučelja s različitim ulogama. Prvo sučelje naziva se "Dohvati HTML, JS i CSS datoteke" te ono služi za dohvaćanje datoteka potrebnih korisničkoj strani kako bi korisniku prikazao potrebno. Komponenta router služi kako bi klijentska strana znala kakav će prikaz biti. Na klijentskoj strani se nalaze JavaScript datoteke logički raspoređene te nazvane po ulogama u sustavu. Sve navedene JavaScript datoteke ovise o React biblioteci. Sučelje "Dohvati JSON" služi za dohvat JSON-a. Rest API poslužuje komponentu Flask dok Elasticsearch služi za dohvat podataka pomoću upita. To se ostvaruje pomoću sučelja za dohvat podataka iz baze. React pomoću svih navedenih sučelja kontinuirano komunicira s poslužiteljskom stranom te tako prikazuje potrebno korisniku.



Slika 3.2: Dijagram komponenti sustava

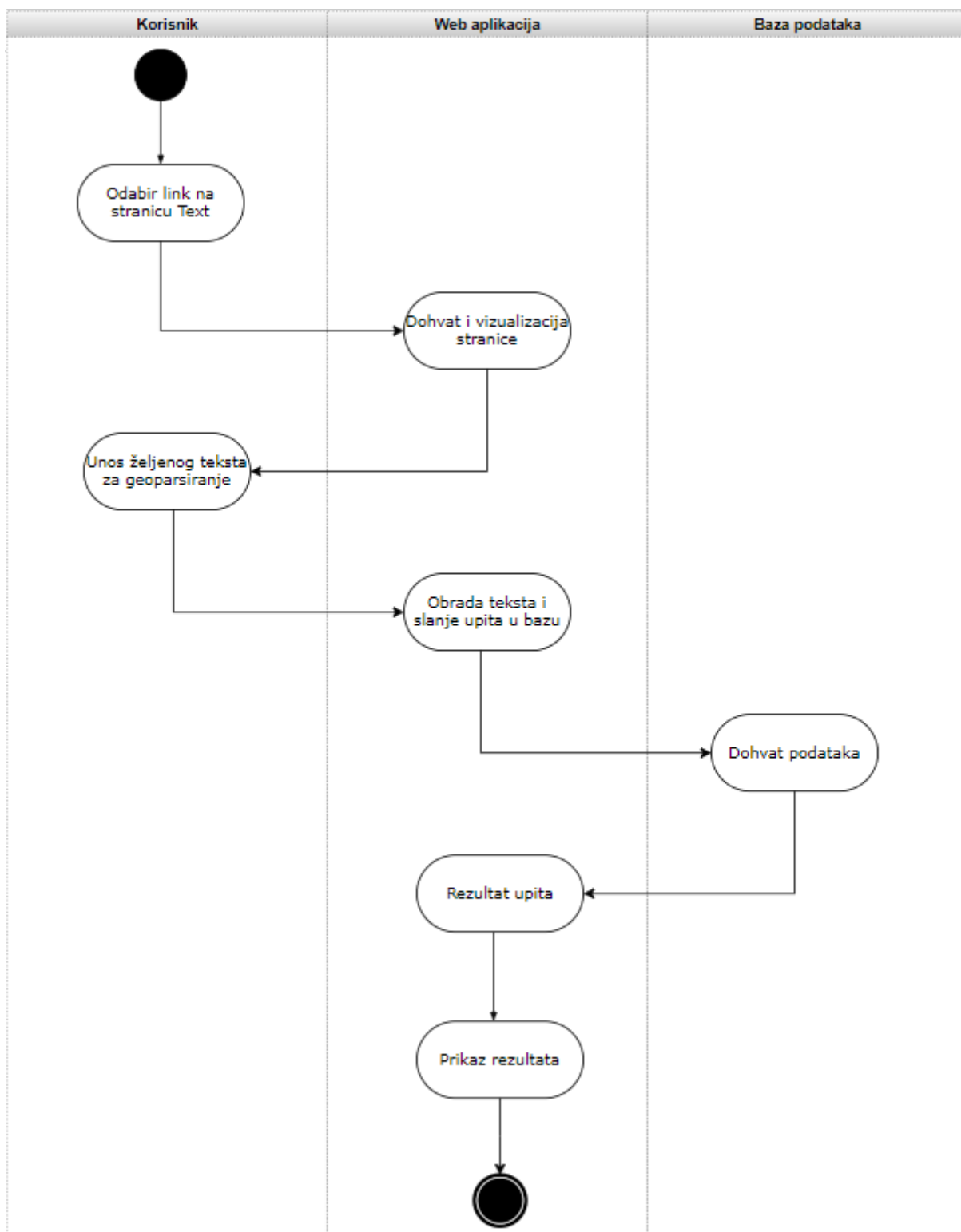
3.3. Dijagram razmještaja

Dijagram razmještaja prikazuje generalnu topologiju sustava koji se koristi kako bi resursi bili optimalno raspoređeni. Slika 3.3 predstavlja statički pogled na razmještaj sklopovskih i programskih komponenata. Na virtualnoj mašini se nalaze poslužitelj weba, poslužitelj baze podataka i poslužitelj za klijentsku stranu. Klijenti koriste preglednik web kako bi pristupili web aplikaciji. U bazi podataka sadržane su sve potrebne informacije koje korisnik može zahtijevati. Sustav je baziran na arhitekturi "klijent – poslužitelj". Komunikacija između poslužiteljskog računala i klijentskog računala se omogućuje HTTP protokolom.



Slika 3.3: Dijagram razmještaja sustava

3.4. Dijagram aktivnosti



Slika 3.4: Dijagram aktivnosti za geoparsiranje određenog teksta

4. Implementacija

Realizirani sustav temelji se na klijent-poslužitelj arhitekturi, implicirajući jasno razgraničenje između dvije primarne komponente sustava. Klijentska strana, kako joj samo ime govori, usmjerena je na interakciju s korisnikom. Omogućuje korisniku da ostvaruje komunikaciju s aplikacijom putem implementiranih funkcionalnosti. Uz to, jedan od ključnih zadataka klijentske strane je pretvorba programskog koda, koji je za većinu korisnika nečitljiv, u sučelje koje omogućuje jednostavnu i razumljivu interakciju.

S druge strane je poslužiteljska strana. Ovaj dio sustava zadužen je za provedbu aplikacijske logike. To znači da kad korisnik kroz sučelje, napravi neki zahtjev, primjerice pritiskom na određeni gumb, zadatak poslužitelja je da obradi pristigli zahtjev i vrati odgovarajući odgovor s podacima natrag klijentskoj strani.

4.1. Poslužiteljska strana

Poslužiteljski dio je realiziran kao REST API koristeći Python. Postavljena je veza s Geonames indeksom unutar Elasticsearcha, gdje su smješteni svi bitni podaci za rad sustava.

API se sastoji od definiranih ruta gdje svaka ruta komunicira s Geonames indeksom putem Elasticsearch upita. Rezultati tih upita se potom transformiraju u JSON format i šalju se kao odgovor klijentskoj strani.

Ovakav pristup omogućuje efikasnu i brzu obradu i dostavu podataka, koristeći moć Elasticsearcha za obradu i pretragu velikih skupova podataka unutar Geonames indeksa.

4.1.1. Integracija Baze Podataka

Za potrebu ovog projekta, umjesto tradicionalne SQL baze podataka, korišten je Geonames indeks u kombinaciji s Elasticsearch-om.

Za ovu integraciju, korišten je Docker, popularna platforma za kontejnerizaciju aplikacija, kako bi se postavilo Elasticsearch okruženje. Docker omogućuje izolaciju aplikacije i njenih ovisnosti u samodostatne jedinice koje se mogu izvršavati na bilo kojem sustavu koji podržava Docker, čime se olakšava postavljanje i distribucija aplikacija.

Korištenjem Dockera, Elasticsearch je bio postavljen u svojem kontejneru, a Python program je komunicirao s Elasticsearchom kroz Docker.

```
1     es = Elasticsearch([{'host': 'localhost', 'port': 9200}])
2     search_request = {
3         "query": {
4             "bool": {
5                 "must": [
6                     {"wildcard": {"name": query + "*"}},
7                     {"match": {"country_code3": "HRV"}}
8                 ]
9             }
10        }
11    }
12
13    response = es.search(index="geonames", body=search_request)
14    return jsonify(response['hits']['hits'])
```

Isječak programskog koda iz Server.py 4.1: Poziv prema bazi podataka

4.1.2. Poslužitelj

Prvo je potrebno učitati sve potrebne biblioteke koje se koriste u implementaciji ovog poslužitelja. Ovaj poslužitelj je implementiran uz pomoć Flask klase, koja predstavlja web aplikacijski okvir napisan u Pythonu.

U isječku programskog koda 4.2 deseti redak sadrži naredbu `app = Flask(__name__)` koja stvara instancu web aplikacije. Kad se pokrene aplikacija, Python dodjeljuje ime `__main__` skripti koja se izvršava. Dakle, ako aplikacija pokreće kao glavna skripta, `__name__` će biti `__main__`, ali ako se uveze skripta koja sadrži tu naredbu iz neke druge skripte `__name__` će biti postavljen na ime skripte. Upotreba `__name__` u Flasku služi za određivanje korijenske putanje aplikacije kako bi kasnije mogao pronaći druge datoteke koje su relativne u odnosu na korijen aplikacije. Također se koristi za postavljanje evidencije razvojnog servera.

U jedanaestom retku se nalazi naredba `CORS(app, origins='http://34.154.166.250')`, inače za vrijeme razvijanja aplikacije koristi se samo `CORS(app)` koja koristi za omo-

gućavanje CORS-a (engl. *Cross-Origin Resource Sharing*) u Flask aplikaciji. CORS je sigurnosni mehanizam koji omogućava web aplikacijama da dozvole zahtjeve s drugih domena, što je uobičajeno blokirano politikom istog izvora (engl. *Same-Origin Policy*). Ova politika ograničava način na koji skripte s jedne web stranice mogu pristupiti resursima druge web stranice. U produkcijskom okruženju često se želi ograničiti domene koje mogu pristupiti vašoj aplikaciji, tako da se postavi CORS politika samo za pouzdane domene. To je postignuto upravo `CORS(app, origins='http://34.154.166.250')` gdje je navedena domena s koje se dopuštaju zahtjevi, jer je to upravo domena virtualne mašine na kojoj radi cijela aplikacija.

U trinaestom retku se nalazi zadnja stvar koju je potrebno inicijalizirati za potpunu funkcionalnost poslužitelja. Naredbu `geo = Geoparser(spacy.load('hr_core_news_lg'))` se može podijeliti na dva dijela. Popularna biblioteka spaCy se koristi za obradu prirodnog jezika u Pythonu. Kao argument pri inicijalizaciji geoparsera se predaje model za hrvatski jezik koji se koristi za razne zadatke obrade prirodnog jezika, kao što su lematizacija, dijeljenje teksta na rečenice, prepoznavanje entiteta, itd. Drugi dio je sama inicijalizacija Geoparser objekta koji služi kao alat za obradu teksta i identifikaciju entiteta vezanih za geografske lokacije. Sve je to spremljeno u varijablu `geo` koja će se kasnije koristiti za izvlačenje geografskih informacija iz hrvatskog teksta.

```
1     from flask import Flask, request, jsonify
2     from flask_cors import CORS
3     from mordecai import Geoparser
4     from elasticsearch import Elasticsearch
5     import spacy
6     import numpy as np
7     import pandas as pd
8     from pyproj import Transformer
9
10    app = Flask(__name__)
11    CORS(app, origins='http://34.154.166.250')
12
13    geo = Geoparser(spacy.load('hr_core_news_lg'))
```

Isječak programskog koda iz Server.py 4.2: Postavljanje početnih vrijednosti

Sada kada su odrađene sve pripreme moguće je napisati konkretnu krajnju točku komunikacije. U isječku programskog koda 4.3 nalazi se primjer krajnje točke komunikacije koja je namapirana na putanju `"/upload_excel"` i GET HTTP metodu. Svaki takav poziv se izvodi asinkrono što znači da jednom pozvana ova metoda neće zablokirati cijeli program dok se cijela ne izvrši, već će se krenuti izvoditi u pozadini, a

program će se nastaviti izvoditi dalje. To je jako bitan dio u implementaciji, jer omogućava da više korisnika poziva API u isto vrijeme. Kada metoda završi s obradom zahtjeva poslat će odgovor s traženim informacijama.

```
1 @app.route('/upload_excel', methods=['GET'])
2 def upload_file():
3     if 'file' not in request.files:
4         return 'No file part', 400
5     file = request.files['file']
6     text = request.form.get('text')
7     x = request.form.get('x')
8     y = request.form.get('y')
9     if file.filename == '':
10        return 'No selected file', 400
11
12    data = []
13    try:
14        for entry in get_excel_data(file, text, x, y):
15            geoparsed_text = parse_float(geo.geoparse(entry[0]))
16            diff_lat, diff_lon = calculate_difference(entry[1],
17                                                    geoparsed_text)
18            data.append({
19                "requested": entry,
20                "geoparsed": geoparsed_text,
21                "difference": (diff_lat, diff_lon)
22            })
23        return jsonify(data)
24    except Exception as e:
25        return jsonify({'error': str(e)}), 500
```

Isječak programskog koda iz Server.py 4.3: Krajnja točka komunikacije upload_excel

Ova krajnja točka prima parametre kroz JSON koji sadrži ključeve file, text, x i y. U parametru file je spremljena excel datoteka koja će se kasnije obraditi. U preostala tri parametra je korisnik bio obavezan predati imena stupaca u excel tablici iz kojih će se uzimati podaci. Konkretno se traži ime stupca s tekstom koje se želi geoparsirati te stupci u kojima su spremljene koordinate. Nakon učitanih svih parametara kreće obrada same tablice. U retku četrnaest se nalazi funkcija get_excel_data(file, text, x, y) koja će učitati sve retke tablice i staviti u listu. Za svaki redak liste se obradi tekst koji se geoparsira te se u JSON koji se šalje kao odgovor sprema tekst koji je trebalo obraditi, obrađen tekst i razliku između očekivanih koordinata i koordinata dobivenim geoparsiranjem.

U primjeru 4.4 može se vidjeti primjer JSON rezultata koji će biti dobiven geoparsiranjem. U ovom slučaju geoparser je prepoznao Vransko jezero kao geografsku lokaciju te vratilo detaljne informacije o istoj. Neke od informacija koje se dobiju je zemlja u kojoj se lokacija nalazi, vrsta lokacije te ono najbitnije njene koordinate.

```
1      {
2          "country_conf": 0.9048774242401123,
3          "country_predicted": "HRV",
4          "geo": {
5              "admin1": "NA",
6              "country_code3": "HRV",
7              "feature_class": "L",
8              "feature_code": "PRK",
9              "geonameid": "7874378",
10             "lat": "43.89195",
11             "lon": "15.57037",
12             "place_name": "Vransko Jezero Nature Park"
13         },
14         "spans": [
15             {
16                 "end": 128,
17                 "start": 114
18             }
19         ],
20         "word": "Vransko jezero"
21     }
```

Primjer 4.4: Rezultat geoparsiranja

4.1.3. Kreiranje Geonames indeksa

Geonames indeks se ponaša kao baza podataka koja nije kao klasična relacijska baza podataka. Zbog svoje vrste drugačije se kreira i populira. Napravljena je skripta koja obavlja tu zadaću, a na početku skripte se kreira direktorij u kojem će se spremati podaci te se pokreće docker kontejner sa Elasticsearchom kako bi se mogla kreirati baza (Halterman, 2017).

```
1      dir_path="$PWD/geonames_index"
2
3      if [ -d "$dir_path" ]; then
4          rm -r "$dir_path"
5      fi
6
```

```

7     mkdir "$dir_path"
8
9     container_id=$(docker run -d -p 127.0.0.1:9200:9200 -e "
        discovery.type=single-node" -v $PWD/geonames_index/:/usr/
        share/elasticsearch/data elasticsearch:7.10.1)

```

Isječak koda iz skripte `create_index.sh` 4.5: Inicijalizacija

Budući da Geonames baza posjeduje jako veliki broj podataka, skinut će se podaci o Hrvatskim lokacijama od tamo zajedno sa još dvije datoteke koje će pomoći pri populaciji podataka u bazu. Prije nego krene obrada tih podataka dodatno se postavlja shema zbog koje će se poboljšati performanse i kvaliteta pretrage. Još se postavljaju neke informacije koje će Elasticsearchu govoriti kako da raspoređuje podatke s obzirom na preostalu memoriju. Nakon svih ovih priprema poziva se Python skripta koja obrađuje podatke dobivene s Geonames stranice te ih populira u bazu (Halterman, 2017).

```

1     wget https://download.geonames.org/export/dump/HR.zip
2     wget https://download.geonames.org/export/dump/admin1CodesASCII.
        txt
3     wget https://download.geonames.org/export/dump/admin2Codes.txt
4
5     unzip HR.zip
6
7     curl -XPUT 'localhost:9200/geonames' -H 'Content-Type:
        application/json' -d @geonames_mapping.json
8
9     curl -X PUT "localhost:9200/_cluster/settings" -H 'Content-Type:
        application/json' -d'
10    {
11        "transient": {
12            "cluster.routing.allocation.disk.watermark.low": "10gb",
13            "cluster.routing.allocation.disk.watermark.high": "5gb",
14            "cluster.routing.allocation.disk.watermark.flood_stage": "4
                gb",
15            "cluster.info.update.interval": "1m"
16        }
17    }
18    '
19
20    python geonames_elasticsearch_loader.py

```

Isječak koda iz skripte `create_index.sh` 4.6: Dohvat i populacija podataka (Halterman, 2017)

4.2. Klijentska strana

Klijentska strana je strana sustava s kojom korisnik ima interakciju. Glavna zadaća je vizualizacija podataka i omogućavanje interakcije korisniku s aplikacijom. Za implementaciju je korišten React.js.

Za pisanje klijentske strane koristi se HTML, CSS i najvažnije JavaScript. Projekt je strukturiran po stranicama, dok se svaka stranica sastoji od komponenti. Velika prednost Reacta je mogućnost korištenja jednom napisane komponente na više mjesta.

4.2.1. React Komponente

Jedan od glavnih razloga za odabir Reacta za razvoj klijentske strane je njegova upotreba komponenti pri izradi. To su cjeline koda koje su samostalne i ponovno upotrebљive. U njima se nalazi markup i logika te ne ovise o drugim stvarima u sustavu. Ova neovisnost omogućava jednostavnu izmjenu izgleda stranice koja koristi komponente, jer izmjenom jedne komponente ostale ostaju nepromijenjene. U svrhu lakšeg razumijevanja, komponentu se može shvatiti kao funkciju koja prima parametre (podaci koji se trebaju prikazati), a kao izlaz se vraća React element koji opisuje kako će se prikazati na zaslonu.

Svaka komponenta bi trebala odrađivati samo jednu funkcionalnost, odnosno ako jedna komponenta ima više funkcionalnosti bila bi dobra praksa podijeliti tu komponentu na više manjih komponenti. Time se osigurava neovisnost jedne komponente prema drugoj i veća šansa da se ista komponenta iskoristi na drugom mjestu.

4.2.2. Virtualni DOM

DOM (engl. *Document Object Model*) je sučelje koje pretvara HTML i XML u strukturirano stablo s čvorovima gdje svaki čvor predstavlja dio dokumenta. Za izmjenu DOMa se koristi JavaScript no treba biti oprezan, jer svaka izmjena podataka usporava rad programa zato što se ne može samo izmijeniti sadržaj tog jednog elementa već je potrebno generiranje cijelog novog DOMa.

React je taj problem riješio implementacijom virtualnog DOMa. To je zapravo kopija DOMa koja se stvara kada se dogodi promjena sadržaja na stranici (novi unos). Virtualni DOM se napravi kada se pozove metoda `setState()` kojom se mijenja stanje podataka koji se vizualiziraju. React koristi dva virtualna DOMa, jedan u kojem se čuva stanje sa starim vrijednostima i jedan koji se generira s novim vrijednostima. Time se osigurava brz i efikasan rad sustava. Kada se završi generiranje virtualnog

DOMa s novim podacima React vidi promjene te koristi taj novi DOM pri generiranju pravog završnog DOMa (Codecademy, 2021).

4.2.3. React aplikacija

Kod generiranja HTML dokumenta komponenta App je postavljena kao korijen aplikacije. Komponenta App u sebi sadrži sve druge komponente u aplikaciji.

```
1 const root = ReactDOM.createRoot(document.getElementById('root'));
2 root.render(
3   <React.StrictMode>
4     <App />
5   </React.StrictMode>
6 );
```

Isječak programskog koda iz index.js 4.7: Postavljanje komponente App kao root

Komponenta App je glavna komponenta u aplikaciji. Ona u sebi enkapsulira sve ostale stranice, a samim time i sve komponente koje se nalaze na tim stranicama. U Reactu je svaku komponentu potrebno uvesti kako bi se mogla koristiti. App u svojem stanju ima spremljen url s kojim se poziva API za dohvat podataka. U render() dijelu se piše ono iz čega će React generirati HTML koji će se prikazati. Svaka stranica će prikazati komponente header i footer te ovisno o tome koja putanja je pozvana prikazat će se pripadajuća komponenta.

```
1 import React from 'react';
2 import './App.css';
3 import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
4 import Header from './components/header/header.component.jsx';
5 import Homepage from './pages/homepage/homepage.component.jsx';
6 import Excel from './pages/excel/excel.component.jsx';
7 import Footer from './components/footer/footer.component.jsx';
8 import Geoparsetext from './pages/geoparsetext/geoparsetext.component.jsx';
9 import Placelookup from './pages/placelookup/placelookup.component.jsx';
10
11 class App extends React.Component {
12
13   constructor() {
14     super();
15     this.state = {
```



```

16     backendURL: "http://34.154.166.250:8000"
17   }
18 }
19
20 render() {
21   return (
22     <Router>
23       <div className="App">
24         <div className="app-container">
25           <Header/>
26           <div className='page-container'>
27             <Routes>
28               <Route path="/" element={<Homepage />} />
29               <Route path="/excel" element={<Excel backendURL={
30                 this.state.backendURL />} />
31               <Route path="/geoparsetext" element={<Geoparsetext
32                 backendURL={this.state.backendURL />} />
33               <Route path="/placelookup" element={<Placelookup
34                 backendURL={this.state.backendURL />} />
35             </Routes>
36           </div>
37           <Footer/>
38         </div>
39       </div>
40     </Router>
41   )
42 }
43
44 export default App;

```

Kod iz App.js 4.8: App komponenta

Komponenta Header u sebi sadrži tri poveznice na druge stranice. U poveznicu je upisana ista putanja koju prepoznaje komponenta App koja onda prikazuje tu određenu komponentu.

```

1 import React from 'react'
2 import './header.styles.scss'
3 import { Link } from 'react-router-dom'
4 import MyGif from '../assets/globe.gif'
5
6 class Header extends React.Component {
7

```

```

8     render() {
9         return (
10             <div className='header-container'>
11                 <Link className='logo-container' to='/>
12                     <img src={MyGif} alt="Logo" className='logo' />
13                 </Link>
14
15                 <div className='link-container'>
16                     <Link to='/excel'>Excel </Link>
17                     <Link to='/geoparsetext'>Text </Link>
18                     <Link to='/placelookup'>Place Lookup </Link>
19                 </div>
20             </div>
21         )
22     }
23 }
24 export default Header

```

Isječak programskog koda iz header.component.jsx 4.9: Header komponenta



Excel Text Place Lookup

Slika 4.1: Izgled zaglavlja aplikacije

Komponenta na kojoj se preda excel datoteka s dodatnim informacijama predaje te podatke API-ju koji vraća JSON s obrađenim podacima. Taj proces se odvija u handle-Submit funkciji koja prvo provjerava jesu li predani svi potrebni parametri. Nakon što prođu sve provjere šalje se zahtjev na backend te ako je status odgovora 200 spremaju se podaci u stanje komponente te se vizualiziraju korisniku.

```

1 handleSubmit = (event) => {
2     event.preventDefault();
3     if (!this.isFormValid()) {
4         this.setState({ errorMessage: "All fields are required"
5             });
6         return;
7     }
8     const formData = new FormData();
9     formData.append('file', this.state.selectedFile);
10    formData.append('text', this.state.text);
11    formData.append('x', this.state.x);
12    formData.append('y', this.state.y);

```

```

12
13     this.setState({ isLoading: true, elapsedTime: 0 });
14
15     this.timer = setInterval(() => {
16         this.setState((prevState) => ({
17             elapsedTime: prevState.elapsedTime + 1,
18         }));
19     }, 1000);
20
21     axios.get(`${this.state.backendURL}/upload_excel`, formData)
22         .then(response => {
23             clearInterval(this.timer);
24             this.setState({
25                 results: response.data,
26                 isLoading: false,
27                 elapsedTime: 0,
28             });
29         })
30         .catch(error => {
31             clearInterval(this.timer);
32             console.log(error);
33             // handle the error as needed
34             this.setState({
35                 isLoading: false,
36                 elapsedTime: 0,
37             });
38         });
39 }

```

Isječak koda iz excel.component.jsx 4.10: Dohvat podataka pomoću APIa

Na komponenti na kojoj se obrađuje excel tablica može se vidjeti efikasnost i jednostavnost korištenja manjih komponenti unutar jedne. Neovisno o predanom broju podataka oni će se prikazati pomoću iste komponente kojoj će se samo predati drugi parametri pri stvaranju komponente.

```

1     <div className="results-container">
2         <h1>Results</h1>
3         <div className='results-header-container'>
4             <h2>User data</h2>
5             <h2>Geoparsed data</h2>
6             <h3>Text</h3>
7             <h3>Coordinates</h3>
8             <h3>Place Name</h3>

```

Excel Upload

Choose File

01_FCD table.xlsx

locality

htsr

htsr

Process File

Results

User data		Geoparsed data						
Text	Coordinates	Place Name	Country	Region	Lat	Lon	Difference	Success
Otok Brač, Vidova gora 2, 14.6.2017.	43.2800, 16.6207	Otok Brač	HRV	Splitsko-Dalmatinska	43.3200	16.6372	0.0400, 0.0165	✓
otok Hvar, Hum, 5.7.2017.	43.1311, 16.6833	Grad Hvar	HRV	Splitsko-Dalmatinska	43.1726	16.4455	0.0415, 0.2378	✓
rijeka Cetina, kanjon 2, 6.7.2017.	43.4506, 16.6980	Cetina	HRV	Šibensko-Kniniska	43.9686	16.4331	0.5180, 0.2649	✓
Otok Silba	44.3778, 14.6920	Silba	HRV	Zadarska	44.3738	14.6964	0.0041, 0.0043	✓
Poluotok Pelješac, selo Ruskovići, 200 m nadmorske visine	42.9851, 17.1737	Pelješac Bridge	HRV	Dubrovačko-Neretvanska	42.9310	17.5352	0.0541, 0.3615	✓
Poluotok Pelješac, selo Podgorje, iznad Orebića, 200 m nadmorske visine	42.9802, 17.1486	Pelješac Bridge	HRV	Dubrovačko-Neretvanska	42.9310	17.5352	0.0492, 0.3866	✓
Bast-Sv. Ilija 1, 422 mnv, Biokovo, blizu staze do vrha	43.3551, 16.9923	Biokovo	HRV	NA	43.3300	17.0700	0.0251, 0.0777	✓

Slika 4.2: Vizualizacija obrade podataka iz excela

```

9      <h3>Country</h3>
10     <h3>Region</h3>
11     <h3>Lat</h3>
12     <h3>Lon</h3>
13     <h3>Difference</h3>
14     <h3>Success</h3>
15   </div>
16   {results.map(result => {
17     const geo = result.geoparsed.length > 0 ? result.
      geoparsed[0].geo : null;
18     return (
19       <ResultsMetaDataContainer
20         text={result.requested[0]}
21         coordinates={result.requested[1]}

```

```

22         placeName={geo ? geo.place_name : 'N/A'}
23         country={geo ? geo.country_code3 : 'N/A'}
24         region={geo ? geo.admin1 : 'N/A'}
25         lat={geo ? geo.lat : 'N/A'}
26         lon={geo ? geo.lon : 'N/A'}
27         difference={result.difference}
28     />
29 );
30 }}}
31 </div>

```

Isječak koda iz excel.component.jsx 4.11: Upotreba manjih komponenti za efikasniji prikaz

Komponenta koja prikazuje jedan obrađen tekst se razlikuje od ostalih u tome što je napravljena s `const`, a ne kao klasa. Razlika je u tome što klasa u sebi ima mogućnost spremanja stanja i dinamičku izmjenu podataka. Ovo je funkcionalna komponenta jer samo generira React element s predanim parametrima, ali i dalje se može odraditi dio neke jednostavne logike.

```

1  const ResultsMetaDataContainer = ({ text, coordinates, placeName,
    country, region, lat, lon, difference }) => {
2      const differenceValid = !(difference === null || difference[0]
        === null || difference[1] === null || isNaN(parseFloat(
        difference[0])) || isNaN(parseFloat(difference[1])));
3      let avgDifference = differenceValid ? (parseFloat(difference[0])
        + parseFloat(difference[1])) / 2 : 1;
4      let isSuccessful = avgDifference < 0.55 ? true : false;
5
6      return (
7          <div className='results-metadata-container'>
8              <p className='results-grid-content'>{text}</p>
9              <p className='results-grid-content'>{coordinates[0].
                toFixed(4) + ",\n" + coordinates[1].toFixed(4)}</p>
10             <p className='results-grid-content'>{placeName}</p>
11             <p className='results-grid-content'>{country}</p>
12             <p className='results-grid-content'>{region}</p>
13             <p className='results-grid-content'>{isNaN(parseFloat(
                lat)) ? lat : parseFloat(lat).toFixed(4)}</p>
14             <p className='results-grid-content'>{isNaN(parseFloat(
                lon)) ? lon : parseFloat(lon).toFixed(4)}</p>
15             <p className='results-grid-content'>
16                 {difference === null || difference[0] === null ||
                    difference[1] === null || isNaN(parseFloat(
                    difference[0])) || isNaN(parseFloat(difference

```

```

        [1])) ? "N/A" : parseFloat(difference[0]).toFixed
        (4) + ", " + parseFloat(difference[1]).toFixed(4)
    }
17     </p>
18     <p className='results-grid-content'>{isSuccessful ? <
        FaCheckCircle color="green" /> : <FaTimesCircle color
        ="red" />}</p>
19 </div>
20 )
21 }

```

Isječak koda iz results-metadata-container.jsx 4.12: Model komponente za jedan podatak

4.3. Postavljanje aplikacije za javnu upotrebu

Postavljanje aplikacije za javnu upotrebu ključan je dio svakog razvojnog procesa. Cilj je osigurati da aplikacija bude dostupna korisnicima na stabilan, siguran i efikasan način. U ovom slučaju, za hosting aplikacije korišten je Google Cloud, jedan od vodećih pružatelja cloud usluga. Google Cloud pruža širok spektar usluga i mogućnosti, uključujući skalabilne virtualne strojeve, podršku za razne tehnologije i niz alata za upravljanje i praćenje performansi.

4.3.1. Virtualni stroj

Virtualni strojevi (VM) su osnova infrastrukture na Google Cloud. Pri kreiranju nove instance virtualne mašine treba odraditi iduće korake:

1. Ime instance - naziv instance trebao bi biti jednostavan za prepoznavanje, a da pritom ispravno opisuje njen zadatak ili ulogu
2. Regija i zona - odabir regije utječe na brzinu i kvalitetu veze
3. Vrsta stroja - ovisno o potrebama aplikacije Google Cloud nudi strojeve s različitim kombinacijama CPU-a, memorije (RAM) i diskovnog prostora
4. Slika operativnog sustava - moguće je odabrati između različitih distribucija Linuxa ili Windowsa
5. Veličina diska i tip - Google Cloud koristi trajno diskovno pohranjivanje, ali može se odabrati i SSD za bolje performanse

6. Firewall - mogućnost omogućavanja HTTP i HTTPS prometa ili dodavanja nekog drugog pravila

Kako je virtualni stroj novi sustav potrebno je instalirati sve potrebne stvari koje aplikacija koristi kako bi uspješno radila.

4.3.2. Postavljanje klijentske strane

Za postavljanje klijentske strane koristi se Nginx (engl. *Engine X*), moćan HTTP i reverse proxy server. Kao web server, Nginx poslužuje statički sadržaj korisnicima koji ga zahtijevaju. To može biti bilo što, od HTML i CSS datoteka do slika i JavaScript datoteka. Kada se Nginx koristi kao obrnuti proxy server, on prima zahtjeve od klijenta i prosljeđuje ih na odgovarajući poslužiteljski server. Nakon što poslužiteljski server obradi zahtjev i vrati odgovor, Nginx preuzima odgovor i prosljeđuje ga natrag klijentu. Ova uloga može biti korisna u mnogim slučajevima, poput omogućavanja SSL enkripcije, load balancinga, skrivanja podataka o unutrašnjem mrežnom okruženju ili pomoći pri upravljanju velikim brojem istovremenih veza.

Nakon instalacije Nginx na virtualnu mašinu potrebno ga je konfigurirati da poslužuje ovu aplikaciju. Zato što je korišten React za kreiranje klijentske strane potrebno je kreirati statičke datoteke aplikacije što se radi naredbom "npm run build" u direktoriju gdje se nalazi projekt React. Statičke datoteke aplikacije će biti dostupne u direktoriju build koji je potrebno prebaciti na virtualnu mašinu te konfigurirati Nginx da poslužuje ove datoteke.

```
1     server {
2         listen 80;
3         server_name domain;
4
5         location / {
6             root /build;
7             index index.html;
8         }
9
10        location /api {
11            proxy_pass http://localhost:8000;
12            proxy_set_header Host $host;
13            proxy_set_header X-Real-IP $remote_addr;
14            proxy_set_header X-Forwarded-For
15                $proxy_add_x_forwarded_for;
16            proxy_set_header X-Forwarded-Proto $scheme;
```

```
16         }  
17     }
```

Primjer 4.13: Konfiguracija Nginx-a

4.3.3. Postavljanje poslužiteljske strane

Za postavljanje poslužiteljske strane sustava koristi se Gunicorn (engl. *Green Unicorn*), WSGI (engl. *Web Server Gateway Interface*) HTTP server za Python web aplikacije. Gunicorn je izuzetno snažan i fleksibilan server, ali je ujedno i vrlo jednostavan za korištenje. Svrha Gunicorna je da posluži kao međusloj između servera weba i same web aplikacije. Serveri weba kao što su Nginx ili Apache mogu biti vrlo efikasni u rukovanju HTTP zahtjevima, ali nisu specijalizirani za izvršavanje Python koda. S druge strane, Python web aplikacije mogu obraditi zahtjeve weba i izvršiti odgovarajući Python kod, ali one nisu nužno optimizirane za efikasno rukovanje velikim brojem zahtjeva. Sve ovo Gunicorn radi efikasno i pouzdano, osiguravajući da web aplikacije mogu izvršavati Python kod brzo i bez prekida, dok se istovremeno efikasno rukuje velikim brojem zahtjeva. On ima ključnu ulogu u osiguravanju visokih performansi i pouzdanosti Python web aplikacija kad su izložene velikom prometu. Nakon instalacije Gunicorna samo se potrebno pozicionirati u direktorij gdje se nalazi Python poslužitelj i pozvati naredbu "gunicorn server:app" koja će pokrenuti poslužiteljsku stranu aplikacije.

5. Korištene tehnologije

Poslužiteljska strana aplikacije je razvijena koristeći PyCharm, integrirano razvojno okruženje (IDE) koje se specijaliziralo za Python, a razvijen je od strane tvrtke JetBrains. PyCharm pruža mnoge značajke koje olakšavaju razvoj Python aplikacija, uključujući inteligentno uređivanje koda, automatizaciju refaktoriranja, grafički debugger i podršku za razne Python web frameworks, poput Django i Flask.

Klijentska strana je napisana u Visual Studio Codeu - source-code editor koji je razvijen od tvrtke Microsoft. Zaštićen je pod MIT licencom. VSC podržava programiranje u raznim jezicima kao što su Java, JavaScript, Go, Node.js and C++. Velika prednost je što se mogu instalirati proširenja koja mogu pružiti podršku za nove jezike, teme ili debuggere. Također, podržava rad s JSON, CSS i HTML tipovima podataka što omogućuje razvoj web-stranica i web-aplikacija.

5.1. Poslužiteljska strana

Poslužiteljska strana aplikacije implementirana je koristeći Python i Flask.

Python je visoko razinski, objektno orijentirani programski jezik koji se odlikuje jednostavnom sintaksom, lako čitljivim kodom i širokom podrškom za različite paradigme programiranja. Pythonova standardna biblioteka je bogata i raznolika, ali pruža i pristup velikom broju vanjskih biblioteka, dostupnih putem Python Package Indexa (PyPI). Ova kombinacija čini Python idealnim izborom za veliki broj razvojnih projekata, uključujući razvoj weba, znanstveno računanje, strojno učenje i mnogo toga.

Flask je mikro web radni okvir napisan u Pythonu. Nudi osnovne funkcionalnosti za izgradnju web aplikacija, uključujući upravljanje zahtjevima, preusmjerenje, sjednice i obradu predložaka. Flask je dizajniran da bude skalabilan, što znači da se njegove funkcionalnosti mogu lako proširiti koristeći različita proširenja. Također, pruža podršku za povezivanje s različitim vrstama baza podataka.

U kombinaciji, Python i Flask pružaju snažno, fleksibilno i skalabilno rješenje za razvoj poslužiteljske strane web aplikacije.

5.1.1. REST API

API (engl. *Application Programming Interface*) je skup logički povezanih komunikacijskih točaka koje pružaju uslugu. Svaka krajnja točka komunikacije je definirana s adresom, putanjom, HTTP metodom, parametrima poziva i odgovorom. Usluge koje pruža API mogu se koristiti pozivima skripte, preko preglednika ili uređaja itd.

REST (engl. *REpresentational State Transfer*) je stil programske arhitekture za nadogradnju raspodijeljenih sustava. Sustavi koji prate principe REST-a su: otvoreni, skalabilni, nadogradivi i jednostavni. Ovi sustavi imaju svojstvo slojevitosti što znači da se u postojeću implementaciju mogu dodavati nove funkcionalnosti (slojevi) s velikom lakoćom. Glavna značajka koja izdvaja REST API od drugih je HATEOAS-URI koji služi za preusmjerenje s jednog resursa na drugi. To omogućava dinamičko kretanje po strukturi API-a (Ivana Bosnić, 2020).

Kvaliteta REST APIa se ocjenjuje pomoću Richardsonovog modela zrelosti:

1. "Swamp": GET i POST, apsolutno svi resursi su dio jedne velike kolekcije, jedan URI
2. Resursi: pristup resursima pomoću jedinstvenog brojanog identifikatora
3. Glagoli/HTTP metode: za iste radnje koristimo iste HTTP metode i njihove specifičnosti, pravilna upotreba HTTP kodova (200, 400, 404...)
4. Hipermedijske kontrole: HATEOAS-poveznice za dinamičko pristupanje resursima

5.1.2. JSON i XML formati podataka

JSON (engl. *JavaScript Object Notation*) je tekstualni format zapisa podataka koji je baziran na sintaksi JavaScripta te je najefektivniji u aplikacijama koje same koriste JavaScript. Dizajniran je tako da bude lagano čitljiv čovjeku dok i dalje ostaje jednostavan za parsiranje računalu. JSON se ponaša kao običan string zbog čega se vrlo jednostavno šalje putem mreže. Kad se želi pristupiti tim podacima potrebno je pretvoriti taj JSON string nazad u JavaScript objekt. Postoje gotove metode koje rade to jako efikasno (contributors MDNs, 2021).

XML (engl. *eXtensible Markup Language*) je proširivi jezik za označavanje. XML-om se zapisuju dokumenti i podaci u tekstualnom formatu. Koncept jezika je da podatke enkapsulira u oznake koje im daju dodatno značenje. Svojom strukturom podsjeća na HTML oznake, ali s jednom bitnom razlikom, naime HTML ima mali broj

zadanih oznaka koje se mogu koristiti dok se u XML može upisati bilo što proizvoljno (W3C, 2015).

JSON i XML su i dalje aktualni u upotrebi te oboje imaju svoje prednosti i mane. Velika prednost JSONa je jednostavnost njegove strukture i iznimno lagana čitljivost i zbog toga je ljudima koji tek kreću učiti o ovim stvarima jednostavnije i brže savladati JSON. JSON također ima mogućnost spremanja polja. S druge strane, XML pruža višu razinu sigurnosti te nudi mogućnost slanja slika, grafova i drugih kompleksnih struktura koje JSON ne podržava (Pedamkar, 2018).

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "James Kirk",
        "age" : 40,
      },
      {
        "name" : "Jean-Luc Picard",
        "age" : 45,
      },
      {
        "name" : "Wesley Crusher",
        "age" : 27,
      }
    ]
  }
}
```

Slika 5.1: Razlika u strukturi XMLa i JSONa (Hoi, 2017)

5.2. Klijentska strana

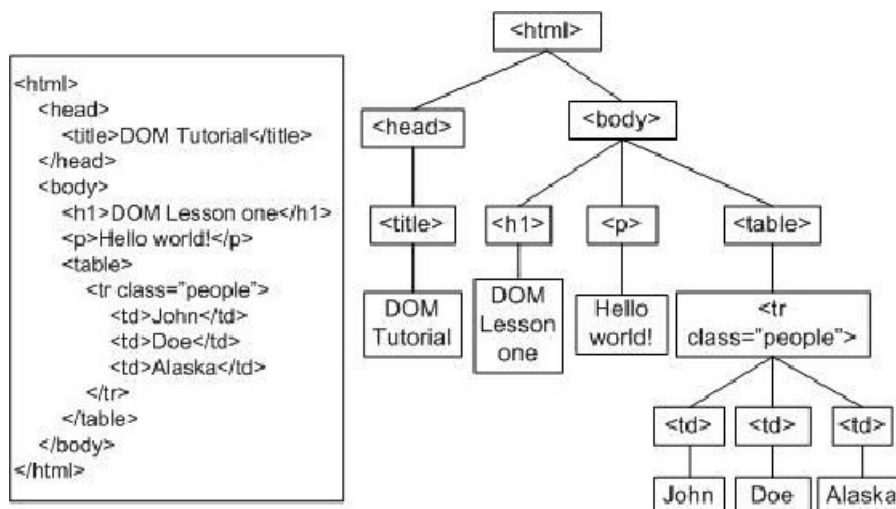
React, biblioteka za izradu korisničkih sučelja, i JavaScript, sveprisutan programski jezik, činili su temelj za izgradnju klijentske strane aplikacije.

React je JavaScript biblioteka posebno dizajnirana za izgradnju sučelja korisnika ili, točnije, UI komponenti. Biblioteka je održavana od strane Facebooka uz pomoć široke zajednice programera i organizacija. React se često koristi kao temelj za izgradnju jednostraničnih aplikacija ili stranica weba, no za kompleksnije projekte često se integrira s dodatnim bibliotekama poput Reduxa i React Routera. React se specifično usredotočuje na prikaz podataka u DOM, što omogućuje stvaranje dinamičkih i interaktivnih korisničkih sučelja.

S druge strane, JavaScript je sveprisutan programski jezik koji je postao standard za razvoj web aplikacija. Kao visokorazinski jezik, JavaScript podržava brojne programske paradigme, uključujući objektno-orijentirano programiranje, deklarativno programiranje i funkcionalno programiranje. S obzirom na svoju široku upotrebu i podršku, JavaScript se koristi za razvoj različitih vrsta aplikacija, kako na klijentskoj tako i na poslužiteljskoj strani, uključujući, ali ne ograničavajući se na, web aplikacije, mobilne aplikacije, desktop aplikacije i igre.

5.2.1. HTML i CSS

HTML (engl. *HyperText Markup Language*) je jezik koji se koristi za izradu svake web stranice. On služi kao opisnik za web stranice o tome kako prikazati naše podatke. U HTMLu postoje već definirane oznake koje se moraju koristiti pri izradi, jer su to pregledniku weba pregledniku upute što prikazati korisniku. HTML nije programski jezik jer nema mogućnost vršenja nikakvih matematičkih operacija već služi samo za prikaz. Struktura jedne HTML datoteke je stablo koje uvijek počinje sa `<html>` oznakom (contributors MDN, 2021b).



Slika 5.2: Primjer HTML strukture (Nasraoui, 2008)

CSS (engl. *Cascading Style Sheet*) je jezik kojim se opisuje izgled i format dokumenata napisanih markup jezikom. Počeo se koristiti sa svrhom odvajanja uređivanja izgleda i sadržaja samog dokumenta. Bez CSSa je potrebno na svaku oznaku pisati kako će biti oblikovan i kako će izgledati taj element. Taj posao vrlo brzo postaje repetativan, a razvijanjem ikakvog većeg sustava smanjuje se preglednost koda i samim time održavanje cijelog sustava. Jednom napisan CSS u jednoj datoteci sa .css nastavkom se koristi na svim mjestima, odnosno nije potrebno svaki put pisati kakve će boje, veličine itd. biti neki element. Primjer jednog CSS dokumenta koji opisuje kako će izgledati svaki naslov s h1 oznakom ili svaki odlomak prikazuje Slika 5.3 (contributors MDN, 2021a).

```
h1 {
  font-family: courier, courier-new, serif;
  font-size: 20pt;
  color: blue;
  border-bottom: 2px solid blue;
}
p {
  font-family: arial, verdana, sans-serif;
  font-size: 12pt;
  color: #6B6BD7;
}
.red_txt {
  color: red;
}
```

Slika 5.3: Primjer CSS dokumenta

5.3. Postavljanje aplikacije za javnu upotrebu

Postavljanje aplikacije za javnu upotrebu implementirano je koristeći Google Cloud VM, s dodatkom NGINX i Unicorn za efikasan web hosting.

Google Cloud VM (Virtual Machine), dio je Google Cloud Platforme (GCP), koja pruža skalabilne, sigurne i visoko dostupne virtualne mašine koje se pokreću na infrastrukturi Googlea. Google Cloud VM omogućuje korisnicima odabir između velikog broja strojeva prilagođenih različitim potrebama, s različitim konfiguracijama CPU-a, memorije, pohrane i mrežnih kapaciteta.

Unicorn je Python WSGI HTTP poslužitelj za UNIX. On je lagan, kompatibilan s velikim brojem web aplikacijskih radnih okvira i izuzetno jednostavan za konfiguraciju. Unicorn djeluje kao posrednik između aplikacije i interneta, obrađujući sve dolazne zahtjeve.

NGINX je snažan poslužitelj web, reverzni proxy, kao i poslužitelj za IMAP/POP3 e-poštu. Koristi se za distribuciju klijentskih zahtjeva prema aplikacijskim poslužiteljima, u ovom slučaju Gunicornu, što rezultira povećanjem performansi i efikasnosti aplikacije.

Google Cloud VM se koristi za hosting aplikacija, izvođenje velikih računalnih zadataka i serviranje web sadržaja. Pruža fleksibilnost, skalabilnost i efikasnost, omogućavajući korisnicima da optimiziraju svoju infrastrukturu i smanje troškove. Uz to, Google Cloud VM pruža i niz značajki za upravljanje, nadzor i automatizaciju infrastrukture.

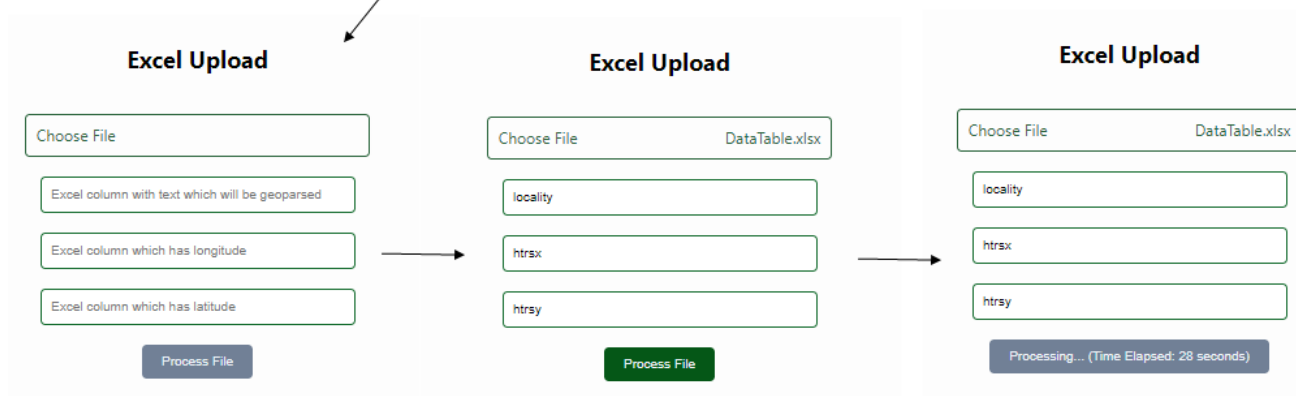
6. Korištenje aplikacije

U idućim poglavljima su pokazani primjeri korištenja aplikacije.

6.1. Geoparsiranje excel podataka

Na Slici 6.1 se može vidjeti primjer korištenja stranice koja podržava prijenos podataka iz excel datoteke. Prvo je potrebno unijeti datoteku te potrebne parametre kao što su imena stupaca u tablici gdje se nalazi tekst koji se želi geoparsirati, geografska dužina i širina. Nakon pravilno unesenih podataka potrebno je pritisnuti gumb "Process file" te pričekati rezultate. Ako nešto nije dobro uneseno od strane korisnika prikazat će se prateća poruka (Slika 6.2).

A	B	C	D	E	F	G
author	year	locality	htrsx	htrsy	idp	source
Jasprica, N.; Terzi, M.	2017	Otok Brač, Vidova gora 2, 14.6.2017.	509799.5679	4793354.003	10	literatura
Jasprica, N.; Terzi, M.	2017	otok Hvar, Hum, 5.7.2017.	514913.6704	4776827.03	10	literatura
Jasprica, N.; Terzi, M.	2017	rijeka Cetina, kanjon 2, 6.7.2017.	516025.6588	4812317.983	10	literatura
Perinčić, B.; Franić, K.; Marčević, Š.; Radović, I.; Židovec, V.	2016	Otok Silba	355926.1452	4916903.904	10	literatura
Jasprica, N.; Škvorc, Ž.; Dolina, K.; Rušić, M.; Kovačić, S.; Franjić, J.	2015	Poluotok Pelješac, selo Ruskovići, 200 m nadmorske visine	554943.8013	4760812.453	11	literatura



Slika 6.1: Primjer geoparsiranja podataka iz excel datoteke

Excel Upload

Choose File

HR.txt

Please upload a valid Excel file (.xls or .xlsx)

Slika 6.2: Primjer poruke pogreške

Nakon što se svi podaci obrade na stranici će se prikazati rezultati. Slika 6.3 prikazuje primjer takvog ispisa. U rezultatima su prikazani sljedeći podaci: tekst i koordinate koje je korisnik predao te ime, država, županija, geografska dužina i širina pronađene lokacije. Također je prikazana razlika između predanih i predviđenih koordinata, kao i vizualan prikaz uspješnosti.

Results								
User data		Geoparsed data						
Text	Coordinates	Place Name	Country	Region	Lat	Lon	Difference	Success
Otok Brač, Vidova gora 2, 14.6.2017.	43.2800, 16.6207	Otok Brač	HRV	Šplitsko-Dalmatinska	43.3200	16.6372	0.0400, 0.0165	✔
otok Hvar, Hum, 5.7.2017.	43.1311, 16.6833	Grad Hvar	HRV	Šplitsko-Dalmatinska	43.1726	16.4455	0.0415, 0.2378	✔
rijeka Cetina, kanjon 2, 6.7.2017.	43.4506, 16.6980	Cetina	HRV	Šibensko-Kniniska	43.9686	16.4331	0.5180, 0.2649	✔
Otok Silba	44.3778, 14.6920	Silba	HRV	Zadarska	44.3738	14.6964	0.0041, 0.0043	✔
Poluotok Peješač, selo Ruskovici, 200 m nadmorske visine	42.9851, 17.1737	Peješač Bridge	HRV	Dubrovačko-Neretvanska	42.9310	17.5352	0.0541, 0.3615	✔
Poluotok Peješač, selo Podgorje, iznad Orebića, 200 m nadmorske visine	42.9802, 17.1486	Peješač Bridge	HRV	Dubrovačko-Neretvanska	42.9310	17.5352	0.0492, 0.3866	✔
Bast-Sv. Ilija 1, 422 mrv, Blokovo, blizu staze do vrha	43.3551, 16.9923	Blokovo	HRV	NA	43.3300	17.0700	0.0251, 0.0777	✔
otok Krk, jače obršten peširjak, pokusna ploha površine 100 m2, 87 m nadmorske visine	45.1972, 14.5847	Grad Krk	HRV	Primorsko-Goranska	45.0283	14.5723	0.1689, 0.0124	✔
otok Čiovo, Gospa od Prizidnice (Dalmacija, Šplitsko-dalmatinska županija) - nalaz 25.04.2006.	43.4842, 16.3656	Otok Čiovo	HRV	Šplitsko-Dalmatinska	43.4981	16.3006	0.0139, 0.0650	✔
Otok Logorun (Šibenski amipelag; Sjeverna Dalmacija), nenaseljen, u neposrednoj blizini naselja Tribunj (Vodice, Šibenik), Podatke unio: M. Magajne, 18.07.2009.	43.7424, 15.7511	N/A	N/A	N/A	N/A	N/A	N/A	✖

Slika 6.3: Primjer rezultata

6.2. Geoparsiranje teksta

Na Slici 6.4 se može vidjeti primjer korištenja stranice koja podržava obradu bilo kojeg teksta. Prvo je potrebno unijeti željeni tekst te pritisnuti gumb "Geoparse" i pričekati rezultate.

The image shows two side-by-side screenshots of a web interface titled "Geoparse Text". The left screenshot shows the input state with a text area containing the placeholder "Enter text here to geoparse..." and a green "Geoparse" button below it. An arrow points to the right screenshot, which shows the output state. The text area now contains a paragraph of Croatian text: "Hrvatska je predivna zemlja s bogatom kulturnom baštinom. Zagreb je jako lijep, ali mi je Split ljepši, ipak mi je Vransko jezero bilo najbolje koje se nalazi gdje i Biograd na moru." The "Geoparse" button remains at the bottom right.

Slika 6.4: Primjer geoparsiranja teksta

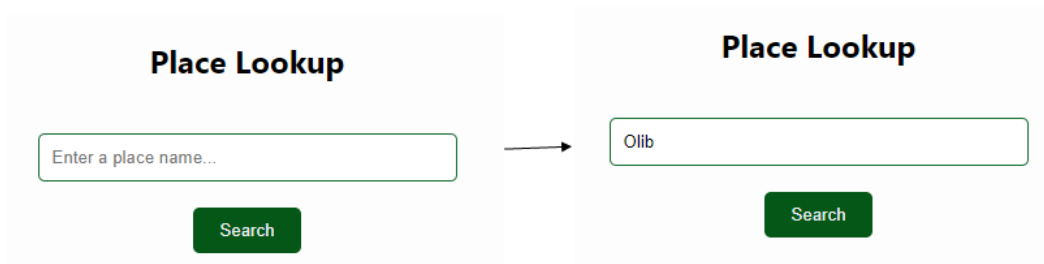
Nakon što je tekst obrađen na stranici će se prikazati rezultati (Slika 6.5). Korisnik može vidjeti koju riječ je alat prepoznao kao lokaciju uz informacije o toj lokaciji kao što su ime, država, županija, geografska dužina i širina.

Results					
Key name	Place Name	Country	Region	Lat	Lon
Hrvatska	Republic of Croatia	HRV	NA	45.1667	15.5000
Zagreb	Grad Zagreb	HRV	City of Zagreb	45.8131	15.9773
Split	Split-Dalmatia	HRV	Splitsko-Dalmatinska	43.1667	16.5000
Vransko jezero	Vransko Jezero Nature Park	HRV	NA	43.8920	15.5704
Biograd	Biograd na Moru	HRV	Zadarska	43.9433	15.4519

Slika 6.5: Primjer rezultata geoparsiranja teksta

6.3. Pretraga mjesta

Na Slici 6.6 ispod se može vidjeti primjer korištenja stranice koja podržava ručnu pretragu mjesta. Ova stranica se koristi kada korisnik želi ručno pretražiti podatke o nekoj lokaciji, kada geoparser ne uspije prepoznati lokaciju u tekstu ili podacima. Prvo je potrebno unijeti željenu lokaciju te pritisnuti gumb "Search" i pričekati rezultate.



The diagram illustrates the 'Place Lookup' form in two states. On the left, the form is titled 'Place Lookup' and contains a text input field with the placeholder text 'Enter a place name...' and a green 'Search' button below it. An arrow points to the right, where the same form is shown after a search. The text input field now contains the word 'Olib', and the 'Search' button remains below it.

Slika 6.6: Primjer pretrage mjesta

Nakon pretrage na stranici će se prikazati rezultati (Slika 6.7). Korisnik može vidjeti koje lokacije sadrže željenu riječ uz te informacije o toj lokaciji kao što su ime, država, županija, geografska dužina i širina.

Results				
Place Name	Country	Region	Lat	Lon
Otok Olib	HRV	County of Zadar	44.3803	14.7903
Luka Olib	HRV	N/A	44.3833	14.7667
Olib	HRV	County of Zadar	44.3750	14.7819

Slika 6.7: Primjer rezultata pretrage mjesta

7. Testiranje i evaluacija aplikacije

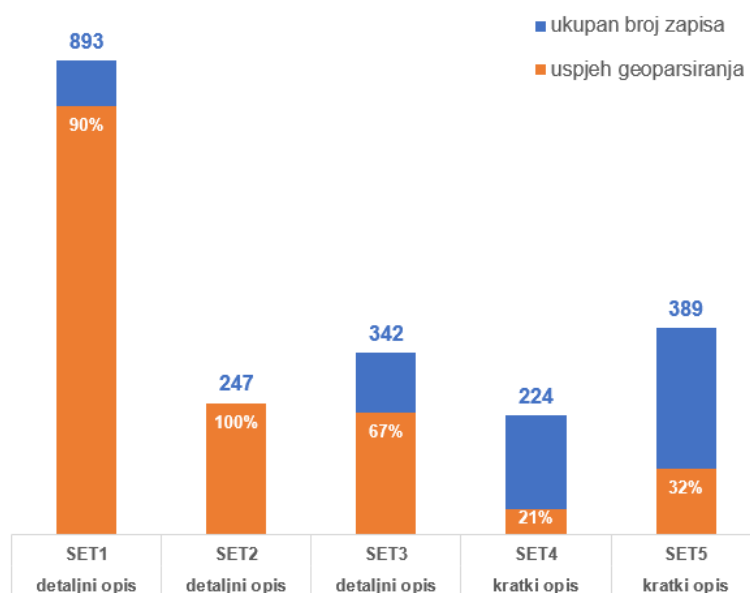
Aplikacija je testirana pomoću pet testnih setova podataka različitih veličina i geografskih koordinatnih sustava (Slika 7.1). Korišteni setovi podataka za testiranje potječu iz raznih bioloških istraživanja, praćenja stanja određenih vrsta kroz građanske inicijative ili nacionalnih baza za botaničke podatke.

OZNAKA SETA PODATAKA	SET PODATAKA	DETALJNOST OPISA LOKACIJA	BROJ ZAPISA	GEOGRAFSKI KOORDINATNI SUSTAV
SET1	lokacije rasta dalmatinskog buhača u prirodi	detaljni opis	893	HTRS96 (EPSG:3765)
SET2	zabilježene invazivne biljne vrste na području Zagrebačke županije	detaljni opis	247	HTRS96 (EPSG:3765)
SET3	lokacije uzorkovanja riječnih rakova	detaljni opis	342	HTRS96 (EPSG:3765)
SET4	lokacije sakupljanja tradicijskih kultivara graha	kratki opis	224	WGS84 (EPSG:4326)
SET5	zabilježeni nalazi krijesnica	kratki opis	389	WGS84 (EPSG:4326)

Slika 7.1: Osnovne karakteristike testnih setova podataka korištenih za testiranje alata

Također, za potrebe geoparsiranja testirane su dvije baze geografskih imena, Geonames baza za Hrvatsku i Registar geografskih imena (DGU, 2023), službeni popis toponima za područje RH koji broji preko 120 000 službenih hrvatskih geografskih imena. Uspješnost geoparsiranja uvelike je varirala ovisno o setu podataka od 21% (SET4) do 100 % (SET2). Viši postotak uspjeha geoparsiranja zabilježen je u setovima podataka koji su sadržavali detaljnije opise (više toponima u opisu lokacije) kao što je vidljivo na Slici 7.2. Također, korištenjem registra geografskih imena kao baze toponima s kojima se uspoređuju stvarni podaci ostvaren je neznatno bolji uspjeh geoparsiranja (1-4 % više zapisa uspješno geoparsirano). Bitno je napomenuti da u dva

testna seta podataka (SET2 i SET4) nije bilo razlike što se tiče izvora toponima korištenih za usporedbu.



Slika 7.2: Rezultati testiranja alata na pet testnih datasetova

Najmanja udaljenost između referentnih toponima i stvarnih podataka zabilježena je za SET2 (u prosjeku 1014 m). Ovaj set podataka imao je detaljne opise lokacija, uključujući i adrese s kućnim brojevima, za razliku od ostalih setova podataka koji su sadržavali ili samo imena gradova, mjesta ili sela (SET4 i SET5) ili detaljne opise lokacija van naseljenih mjesta, uz prirodne toponime kao što su rijeke i planine (SET1 i SET3). Najveće udaljenosti između referentnih toponima i stvarnih podataka zabilježene su upravo za SET1 i SET3 (24 794 m i 40 231 m u prosjeku).

Detaljnijim pregledom rezultata također je utvrđeno da neki zapisi koji nisu bili uspješno geoparsirani, a postoje u referentnim registrima (primjer otočić Obonjan u Zadarskom arhipelagu) ili zapisi koji su geoparsirani neprecizno (primjer Huma na Hvaru koji je geoparsiran i geokodiran kao grad Hvar). Ovi rezultati upućuju na potrebu da se sam postupak geoparsiranja dodatno prilagodi prije puštanja alata u širu upotrebu.

8. Zaključak

U sklopu ovog rada je razvijena aplikacija čija je svrha procjena prostorne točnosti skupova podataka opažanja u prirodi. Aplikacija koristi kombinaciju Reacta na klijentskoj strani i Pythona na poslužiteljskoj strani što omogućuje korisnicima da interaktivno i efikasno rade s geoprostornim podacima. Implementacija aplikacije na Google Cloud VM osigurava javnu dostupnost čime je olakšano i poboljšano geoprostorno istraživanje za stručnjake i druge zainteresirane.

Kako je aplikacija napravljena na klijent-poslužitelj arhitekturi moguće su lagane izmjene i dodavanje novih funkcionalnosti neovisno jedno o drugome. Samim time razvoj aplikacije se lagano može nastaviti te se može unaprijediti dodavanjem novih funkcionalnosti razvijanjem samog područja ovisno o potrebama korisnika.

Ovim diplomskim radom je napravljen korak prema kvalitetnijem i preciznijem sustavu koji koristi geoprostorne podatke. Iz ovog rada mogu proizaći nove mogućnosti i ideje za unaprijeđenje sustava.

LITERATURA

Codecademy. React: The virtual dom. <https://www.codecademy.com/articles/react-virtual-dom>, 2021.

contributors MDN. Css: Cascading style sheets. <https://developer.mozilla.org/en-US/docs/Web/CSS>, 2021a.

contributors MDN. Html: Hypertext markup language. <https://developer.mozilla.org/en-US/docs/Web/HTML>, 2021b.

contributors MDNs. Working with json. <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>, 2021.

DGU. Registar geografskih imena. <https://geoportal.nipp.hr/geonetwork/srv/hrv/catalog.search#/metadata/3195b3bd-e3ee-4220-89b6-1fa53f435dd6>, 2023.

Nikhil Dhavase. Location identification for crime disaster events by geoparsing twitter. <https://ieeexplore.ieee.org/document/7092336>, 2014.

Dragica Šlamon Filip Varga, Ana Kuveždić Divjak. Towards development of a tool for the automated assessment of the spatial accuracy of nature observation datasets. <https://www.bib.irb.hr/1253059>, 2023.

Andrew Halterman. Mordecai: Full text geoparsing and event geocoding. <https://github.com/openeventdata/mordecai>, 2017.

Sunny Hoi. Json vs xml picture. <https://www.1337pwn.com/json-vs-xml-format-use-api/>, 2017.

Igor Čavrak Ivana Bosnić. Web api, rest. Kolegij Otvoreno računarstvo, Fakultet elektrotehnike i računarstva, 2020.

Olfa Nasraoui. Dom tree of an example web page picture. https://www.researchgate.net/figure/Dom-Tree-of-An-Example-Web-Page_fig2_221417012, 2008.

Priya Pedamkar. Json vs xml. <https://www.educba.com/json-vs-xml/>, 2018.

Paul Rayson. Customising geoparsing and georeferencing for historical texts. <https://ieeexplore.ieee.org/document/6691671>, 2013.

W3C. Extensible markup language (xml). <https://www.w3.org/XML/>, 2015.

Automatizirana procjena prostorne točnosti skupova podataka opažanja u prirodi

Sažetak

S obzirom na veliki broj geoprostornih podataka koji postoje i koji se unose, ima previše mjesta za pogrešku. Zbog toga je izašla potreba za ovakvom aplikacijom koja će moći brzo i efikasno provjeriti postojeće i buduće zapise. Aplikacija je bazirana na arhitekturi klijent-poslužitelj koji su razvijeni neovisno jedno o drugome te bazi podataka u kojem su spremljeni podaci. Aplikacija je javno dostupna i omogućuje više vrsta provjere podataka te ih također korisniku vizualizira.

Ključne riječi: Geoparser, Mordecai, Python, React, JavaScript, Google Cloud

Automated assessment of the spatial accuracy of observation data sets in nature

Abstract

Given the large amount of geospatial data that exists and is being entered, there is too much room for error. This is why there was a need for such an application that will be able to quickly and efficiently check existing and future records. The application is based on the client-server architecture, which were developed independently of each other, and the database in which the data is stored. The application is publicly available and enables several types of data verification and also visualizes it for the user.

Keywords: Geoparser, Mordecai, Python, React, JavaScript, Google Cloud