# Sustav za profiliranje i otkrivanje podataka

**Ivić, Mate**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 652

# SUSTAV ZA PROFILIRANJE I OTKRIVANJE PODATAKA

Mate Ivić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 652

# SUSTAV ZA PROFILIRANJE I OTKRIVANJE PODATAKA

Mate Ivić

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# DIPLOMSKI ZADATAK br. 652

| | |
|---|---|
| Pristupnik: | **Mate Ivić (0036523235)** |
| Studij: | Računarstvo |
| Profil: | Programsko inženjerstvo i informacijski sustavi |
| Mentor: | prof. dr. sc. Boris Vrdoljak |

Zadatak: **Sustav za profiliranje i otkrivanje podataka**

Opis zadatka:

U sklopu diplomskog rada potrebno je izgraditi sustav za profiliranje i otkrivanje podataka, koji će za podatke iz relacijske baze odrediti tip podatka, jedinstvenost te minimalnu, maksimalnu ili prosječnu vrijednost ako se radi o brojčanim atributima. Profiliranje podataka podrazumijeva analizu sadržaja i strukture atributa u relacijama. Treba uzeti u obzir povezanosti među relacijskim podacima, također i za slučajeve kad nema eksplicitnog refencijalnog integriteta. U određenim slučajevima potrebno je takve povezanosti otkriti heurističkim automatskim ili poluautomatskim metodama. Sustav za profiliranje i otkrivanje podataka treba dinamički generirati SQL upite i pomoću njih dohvaćati iz Kaggle baze podataka rezultata utrka Formule 1 sve potrebne statistike te pomoću generiranih SQL upita implementirati automatske i poluautomatske metode za otkrivanje relacijskih veza između tablica.

Rok za predaju rada: 28. lipnja 2024.

## Sadržaj

# Uvod

Živimo u digitalnom dobu gdje se svakodnevno stvara i pohranjuje ogromna količina novih podataka u raznim sektorima i industrijama. U toj golemoj količini podataka lako se izgubiti. Zamislite bazu podataka s tisuću redaka, što je uobičajeno u praksi. „Ručno“ pregledavanje takvih tablica je iznimno vremenski neefikasno i gotovo nemoguće, a traženje specifičnih veza ili posebnosti u tim podacima dodatno otežava zadatak. Stoga je automatizacija takvih postupaka neophodna, omogućavajući sustavima da profiliraju i otkrivaju podatke s minimalnom ljudskom intervencijom. Automatizacijom ovog postupka analiza i obrada velikih količina podataka može se izvršiti mnogo brže, pružajući korisnicima korisne uvide i informacije. U tom kontekstu, sustav za profiliranje i otkrivanje podataka postaje ključni alat za transformaciju gomile sirovih podataka u vrijedne spoznaje.

Moj sustav za profiliranje i otkrivanje podataka dizajniran je upravo s tom svrhom - da analizira nekoliko relacija baza podataka ili čak cijele baze podataka koristeći automatske ili poluautomatske metode. Sustav je osmišljen da identificira povezanost relacija na temelju vjerojatnosti, čak i kada referencijalni integritet nije eksplicitno naveden. Na primjer, u potpuno nepoznatim relacijama trebao bi s određenom vjerojatnošću odrediti primarne ključeve, strane ključeve (i na koju se relaciju odnose), tip podataka, jedinstvenost te minimalne, maksimalne ili prosječne vrijednosti brojčanih atributa.

Za testiranje sustava koristi se baza podataka s rezultatima utrka Formule 1, koja je prikladna za demonstraciju svih implementiranih funkcionalnosti, kao i prednosti i nedostataka sustava.

Prvo poglavlje opisat će općeniti proces profiliranja podataka i njegove ključne značajke. Drugo poglavlje bavi se specifikacijom zahtjeva. Objasnit će se koji su ciljevi ovog sustava te će se navesti kako su ti ciljevi ostvareni. Treće poglavlje pruža uvid u tehničke značajke te korištene alate tijekom rada na sustavu. Četvrto poglavlje prikazuje samu implementaciju sustava, te nekoliko istaknutih rješenja koji su sustav učinili funkcionalnim. Peto poglavlje pruža kratke upute za korištenje programa, nakon čega u šestom poglavlju slijede ograničenja sustava i smjerovi u kojima su moguća poboljšanja.

# 1. Profiliranje podataka

Profiliranje podataka je proces pregledavanja i čišćenja podataka kako bi se bolje razumjelo kako su strukturirani podaci te kako bi se održavali standardi kvalitete podataka unutar organizacije. Glavna svrha je steći uvid u kvalitetu podataka metodama pregleda i sažimanja podataka, a zatim i vrednovati njihovo stanje. Posao obično obavljaju podatkovni inženjeri koji koriste niz poslovnih pravila i analitičkih algoritama.

Profiliranje podataka procjenjuje podatke na temelju čimbenika kao što su točnost, dosljednost i pravodobnost. Rezultat je najčešće nešto jednostavno poput statistike, brojeva ili vrijednosti u obliku stupca, ovisno o skupu podataka. Profiliranje podataka može se koristiti za projekte koji uključuju skladištenje podataka ili poslovnu inteligenciju, a najkorisnije je za velike podatke. Profiliranje podataka je važan proces koji prethodi obradi i analitici podataka.[1]

## 1.1. Kako funkcionira profiliranje podataka

Tvrtke integriraju različite softvere ili aplikacije kako bi osigurale da su skupovi podataka pripremljeni na odgovarajući način i da se efektivno može izvršiti proces uklanjanja loših podataka. Konkretno, možete odrediti koji izvori imaju ili stvaraju probleme s kvalitetom podataka, što u konačnici utječe na ukupni poslovni operativni i financijski uspjeh. Ovaj proces također će izvršiti potrebnu procjenu kvalitete podataka.

Prvi korak profiliranja podataka je prikupljanje izvora podataka i povezanih metapodataka za analizu, što često može dovesti do otkrivanja odnosa stranog ključa. Sljedeći korak je čišćenje podataka kako bi se, između ostalog, osigurala jedinstvena struktura i uklonilo dupliciranje. Nakon što su podaci očišćeni, softver za profiliranje podataka vratit će statistiku za opis skupa podataka koja može uključivati stvari kao što su srednja vrijednost, minimalna/maksimalna vrijednost i učestalost.[1]

## 1.2. Prednosti i izazovi profiliranja podataka

Većina velikih tvrtki imaju veliku količinu podataka, ali kvaliteta tih podataka je isto važna i tada profiliranje podataka dolazi na scenu. Kada imate standardizirane podatke koji su precizno oblikovani, oni ostavljaju malo ili nimalo šanse za nezadovoljne klijente ili pogrešnu komunikaciju.

Izazovi su većinom sistemske prirode jer ako, na primjer, podaci nisu svi na jednom mjestu, vrlo ih je teško locirati. Ali s ugradnjom određenih podatkovnih alata i aplikacija to ne bi trebao biti problem. Profiliranje podataka može ponuditi pregled podataka na visokoj razini za razliku od bilo kojeg drugog alata. Neke od ključnih prednosti profiliranja podataka su:

- **Točnija analitika**: Kompletno profiliranje podataka osigurat će bolju kvalitetu i vjerodostojnije podatke. Ispravno profiliranje podataka može pomoći da se bolje shvati odnos između različitih skupova podataka i izvora te podrži postupke upravljanja podacima.
- **Održavanje informacije centraliziranom**: Ispitivanjem i analizom podataka putem profiliranja podataka može se očekivati da će kvaliteta podataka biti mnogo veća i podaci bolje organizirani. Pregledom izvornih podataka otklonit će se pogreške i istaknuti područja s najviše problema. Zatim će proizvesti uvid i organizaciju koja centralizira podatke na najbolji mogući način.

Izazovi profiliranja podataka obično proizlaze iz složenosti posla koji je uključen. Neki od najvećih izazova su:

- **Cijena i dugotrajnost procesa**: Profiliranje podataka može biti izuzetno složeno prilikom implementacije uspješnog programa, prvenstveno zbog ogromne količine podataka koje organizacija prikuplja. Ovaj proces može biti vrlo skup i dugotrajan, zahtijevajući angažman stručnjaka za analizu rezultata i donošenje odluka bez adekvatnih alata.
- **Decentraliziranost podataka**: Kako bi se započeo proces profiliranja podataka tvrtka treba svoje podatke na jednom mjestu, što često nije slučaj. Ako se podaci nalaze na različitim mjestima i tvrtka nema obučenog stručnjaka za podatke, može postati vrlo teško profilirati podatke o tvrtki kao cjelini.[1]

# 2. Specifikacija zahtjeva

## 2.1. Analiza zahtjeva

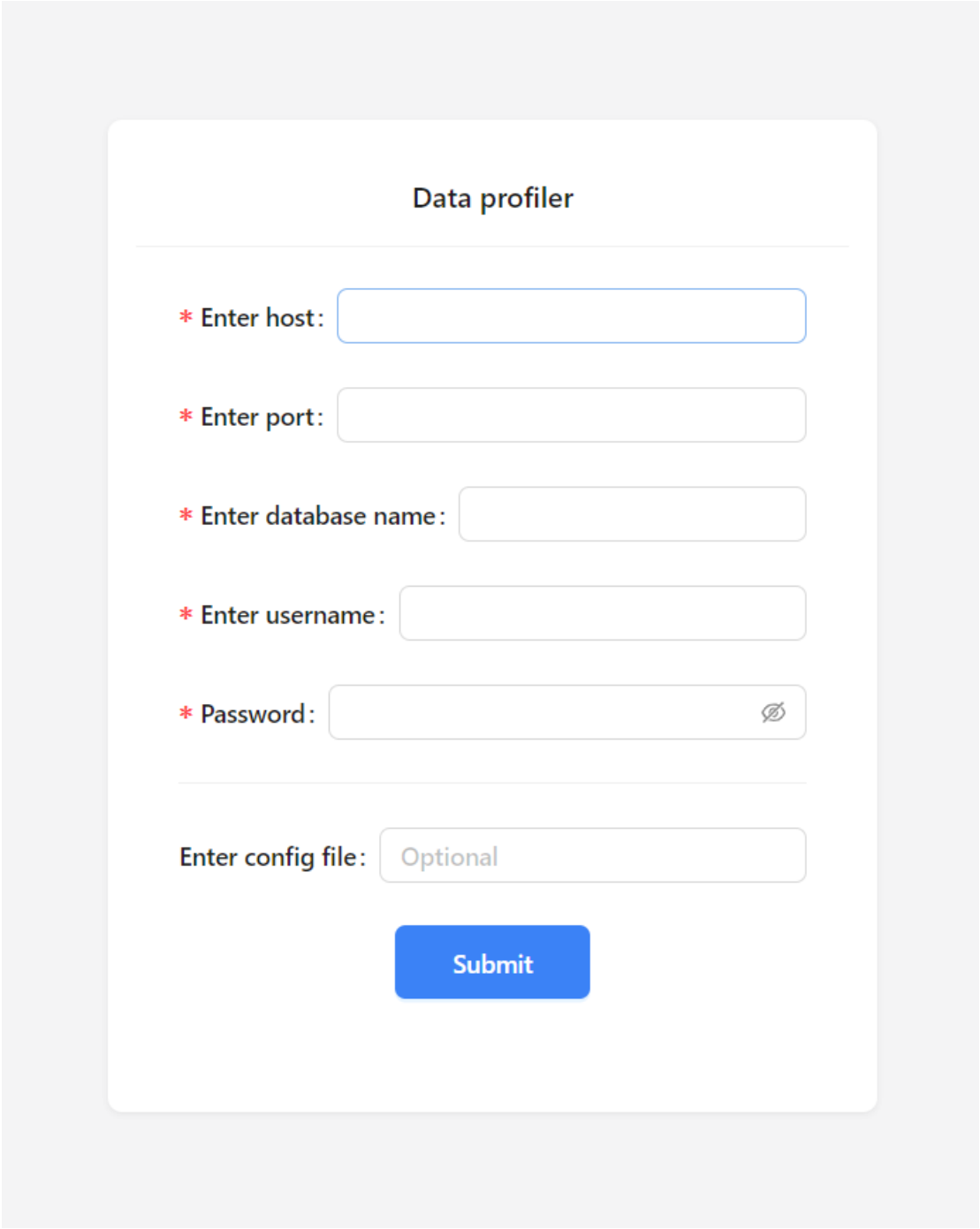
Prije izgradnje samog sustava potrebno je odrediti tko će ga koristiti, te samim time prilagoditi sustav zahtjevima i potrebama korisnika. Sustav je namijenjen „data inženjerima“, koji će iz gomile podataka profiliranjem pronaći korisne informacije te koristeći te informacije napraviti kvalitetnu pripremu podataka. Pretpostavka je da se svi potrebni podaci za profiliranje nalaze u istoj bazi podataka. Dakle, ključni zahtjevi korisnika su:

- Mogućnost povezivanja sustava na postojeću bazu podataka, na kojoj se vrši profiliranje
- Prikaz potencijalnog primarnog ključa za svaku relaciju
- Prikaz potencijalnih stranih ključeva u relaciji, te na koju relaciju bi se on referencirao
- Prikaz osnovnih mjera atributa (tip podatka, jedinstvenost, broj null vrijednosti te minimalna, maksimalna ili prosječna vrijednost ako se radi o brojčanim atributima)
- Prikaz najčešćih vrijednosti za svaki atribut svake relacije i broj njihovih pojavljivanja u stupcu
- Otkrivanje značenja pojedinog atributa (npr. prepoznavanje atributa koji prikazuje datum ili OIB, bez da je eksplicitno navedeno u imenu atributa)
- Otkrivanje da vrijednosti atributa odgovaraju nekom drugom tipu (npr. ako je neki atribut tipa znakovni niz, a u svakom retku se nalazi broj)
- Mogućnost određivanja koje relacije baze podataka će se uzeti u obzir pri profiliranju, a koje ne
- Jednostavno i intuitivno korisničko sučelje

## 2.2. Realizacija zahtjeva

Nakon navođenja zahtjeva korisnika potrebno je što više tih zahtjeva realizirati u projektu. U nastavku je ukratko objašnjeno na koji će način svaki od gore navedenih zahtjeva biti realiziran u stvarnom sustavu, tako da svaki odlomak daje kratki opis kako ispuniti gore navedeni zahtjev.

Nakon pokretanja programa korisnik će preko korisničkog sučelja morati unijeti podatke potrebne za pristup ciljanoj bazi podataka. Od korisnika se traži da unese korisničko ime, lozinku, poslužitelja (engl. *host*), vrata (engl. *port*), ime baze podataka te opcionalno konfiguracijsku datoteku u kojoj bi odabrao koje relacije i atributi će se uzeti u obzir pri profiliranju. Nakon unosa podataka, sustav će pokušati pristupiti toj bazi podataka te započeti s profiliranjem. Na Slika 2.1 vidi se kako izgleda forma za unos podataka potrebnih za pristup bazi podataka, što je ujedno i početni zaslon aplikacije.




Slika 2.1 – forma za unošenje podataka za pristup bazi podataka

Tijekom svog rada sustav će za svaku relaciju baze podataka s pomoću algoritma pokušati pronaći primarni ključ. Program će prikazati onaj atribut koji statistički ima najveće šanse biti primarni ključ, te naravno zadovoljava uvjete primarnog ključa kao što su npr. jedinstvenost i ne sadržavanje null redaka. Ako sustav nije pronašao nijedan atribut relacije

kojeg smatra kao kandidata za primarni ključ, korisniku će poručiti da ta relacija nema kandidata za primarni ključ.

Sustav će također pokušati pronaći kandidate za strane ključeve, te s određenom vjerojatnošću tvrditi na koju relaciju se taj strani ključ referencira. Izračun za dobivanje stranih ključeva i njihovih vjerojatnosti vršit će se koristeći heurističke metode. Ako je sustav pronašao više atributa na koje bi se strani ključ mogao referencirati, sustav će ponuditi sve atribute uz izračunatu vjerojatnost pripadnosti za svaki atribut, tako da korisnik bude svjestan koji su mogući kandidati za strani ključ i njihove vjerojatnosti, te se dubljom analizom korisnika može odlučiti koji kandidat najbolje odgovara.

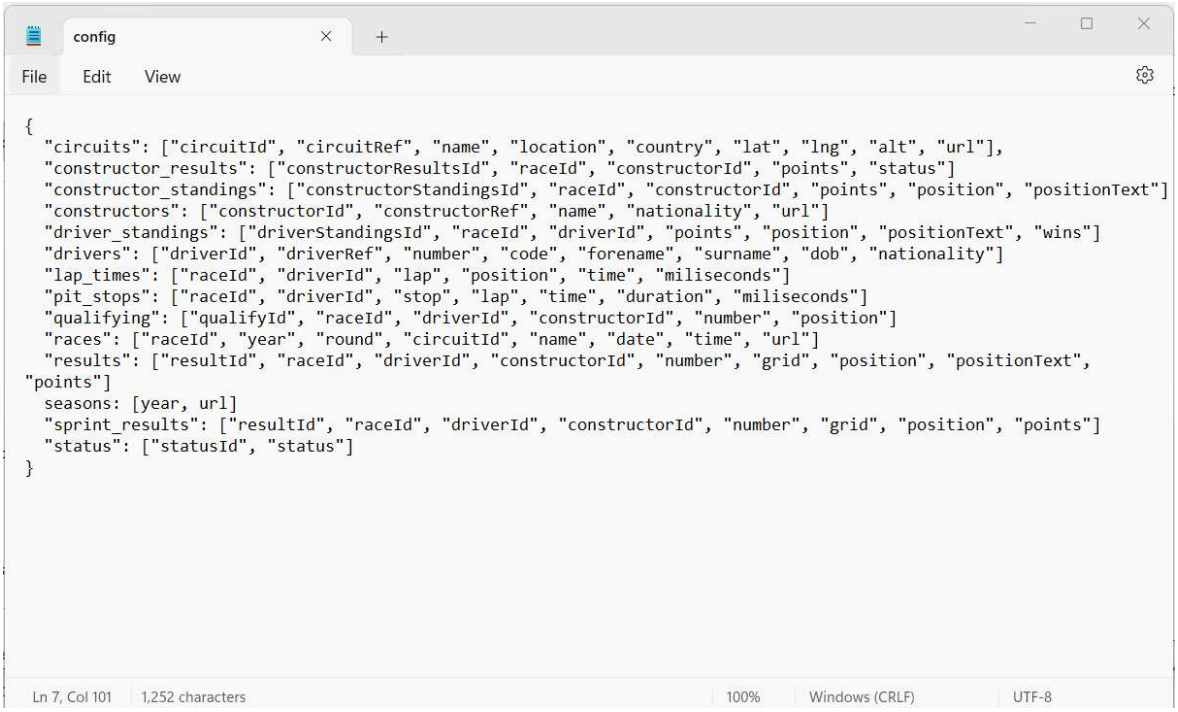
Za svaki atribut u svakoj relaciji program će ispisati koji je tip podatka, te ako je podatak brojčana vrijednost ispisat će neke osnovne mjere kao što su minimalna vrijednost, maksimalna vrijednost i aritmetička sredina. Također ako je tip podatka znakovni niz program će kroz razne uzorke i kontrolu zadnje znamenke (u slučaju za OIB) pokušati prepoznati radi li se možda o datumu ili OIB-u, te ako postoji velika vjerojatnost za to napomenuti korisniku u završnom izvještaju. Valja naglasiti da program neće uzimati u obzir ime atributa jer to i jest cilj sustava, odrediti veze između atributa i relacija te profilirati podatke bez da zna išta o njima.

Prilikom kreiranja izvještaja sustav će za svaki profilirani atribut relacije ispisati najčešće vrijednosti tog atributa (najviše 10), te ukupan broj pojavljivanja te vrijednosti u stupcu atributa. Ovaj podatak je dosta koristan pri profiliranju jer korisnik na temelju najčešćeg sadržaja stupca može jednostavnije otkriti što taj stupac zapravo predstavlja.

Ako se u bazi podataka slučajno desi da tip atributa ne odgovara njegovoj stvarnoj vrijednosti (npr. tip atributa je znakovni niz, a sve vrijednosti u tom stupcu su brojevi) program će primijetiti tu razliku te napomenuti korisniku. Na kraju će korisniku u izvještaju biti dostupni tip atributa u bazi podataka i tip atributa na temelju pregledavanja i analiziranja svih redaka tog atributa u tablici.

Program će moći i primiti konfiguracijsku datoteku u JSON formatu koja će diktirati koje relacije i koji atributi u relaciji će se uzeti u obzir prilikom profiliranja podataka. Konfiguracijsku datoteku je potrebno postaviti u „config“ direktorij unutar projekta, jer na tom mjestu sustav pronalazi dostupne konfiguracijske datoteke. Konfiguracijske datoteke će se unositi nakon unosa podataka potrebnih za pristup bazi podataka. Važno je naglasiti da je unos konfiguracijske datoteke potpuno opcionalan. Ako se konfiguracijska datoteka

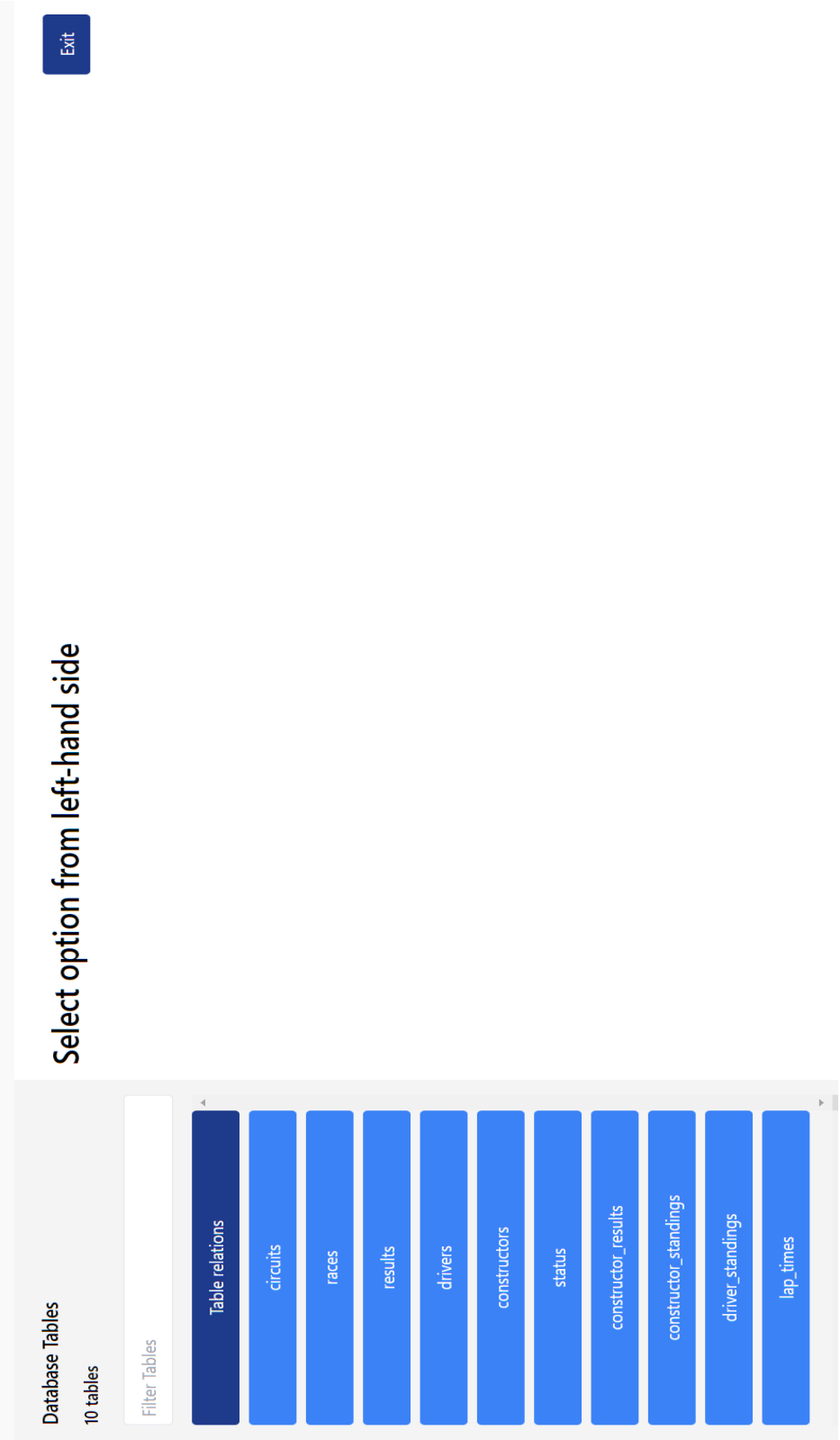
ne navede program će pri izračunu u obzir uzeti sve atribute svih relacija unutar baze podataka. Slika 2.2 prikazuje kako konfiguracijska datoteka može izgledati. Ona mora biti u JSON formatu te naziv svake značajke („circuits“, „constructor_results“…) predstavlja naziv relacije u bazi podataka, a pripadajuća vrijednost je prikazana u obliku polja koje govori koji će se atributi te relacije uzeti u obzir pri radu programa.




Slika 2.2 – primjer konfiguracijske datoteke

Nakon spajanja na bazu podataka i opcionalnog odabira konfiguracijske datoteke, korisniku se prikazuje korisničko sučelje koje pita korisnika da odabere opciju na lijevoj strani kako bi dobio izvještaj. Sučelje je prikazano na Slika 2.3. Korisniku će s lijeve strane u listi s mogućnošću filtera biti prikazane sve tablice koje je odabrao za profiliranje, te klikom na bilo koju od njih ispisuje se izvještaj za pojedinu tablicu na desnoj strani sučelja. Na vrhu liste, ispod tražilice, nalazi se gumb „Table relations“. Klikom na taj gumb korisniku se ispisuje izvještaj koji u obzir uzima sve tablice i njihovo međudjelovanje, tako se na primjer u tom izvještaju prikazuju predikcije stranih ključeva. Dakle, prikaz će biti grupiran po relacijama, što znači da će se prikazati rezultati programa za onu relaciju koju je korisnik izabrao, ili ako je izabrao prikaz „Table relations“ dobit će predikcije stranih

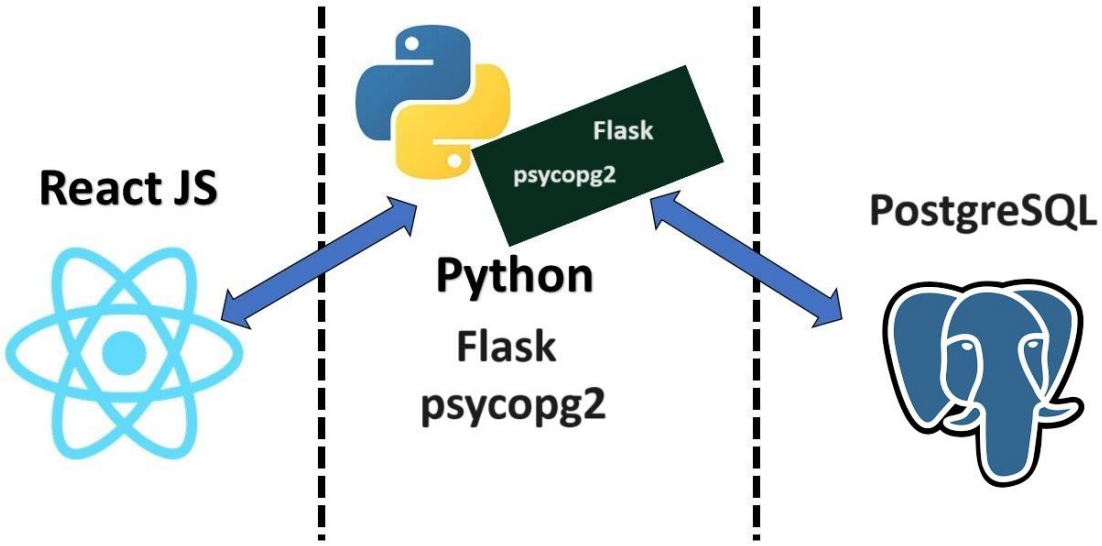
ključeva. Cilj je završni izvještaj učiniti što preglednijim, a program što jednostavnijim za korištenje i što otpornijim na pogreške.




Slika 2.3 – početni prikaz korisničkog sučelja nakon spajanja na bazu podataka

# 3. Korišteni alati i tehnologije

Izrada ovako složenog projekta zahtjeva korištenje i snalaženje u nekoliko različitih alata i tehnologija. Da bih izgradio sustav morao sam povezati program pisan u programskom jeziku Python s PostgreSQL bazom podataka u kojoj su se nalazile informacije o rezultatima Formule 1. Kako bih to uspio, morao sam koristiti Pythonovu biblioteku za stvaranje konekcije s bazom podataka psycopg2. Koristeći Pythonovu biblioteku Flask stvorio sam poslužitelj koji potrebne podatke šalje sa serverske na klijentsku stranu aplikacije. Za izradu klijentske strane aplikacije sam koristio React.js. Prikaz opisane arhitekture vidljiv je na Slika 3.1. Cjelokupni projekt pisan je u uređivaču izvornog koda Visual Studio Code. Skup podataka s rezultatima Formule 1 je dostupan na Kaggle stranici[11]. Važno je naglasiti da sustav funkcionira s bilo kojom PostgreSQL bazom podataka, te da je skup podataka s rezultatima Formule 1 korišten samo kako bi se demonstrirale i testirale funkcionalnosti sustava. Upiti na bazu podataka su se izvodili koristeći standardizirani jezik za upravljanje relacijskim bazama podataka, SQL. Rezultati tih upita su se prosljeđivali u Python skriptu te bi se na temelju tih rezultata raznim metodama izračunavala vjerojatnost povezanosti atributa i relacija u bazi podataka, te provodila daljnja analiza potrebna za finalni rezultat koji će biti ispisan na korisničkom sučelju.




Slika 3.1 – arhitektura sustava

## 3.1. Programski jezik Python

Python je dinamički tipizirani, interpretirani programski jezik visoke razine. Python je trenutno jedan od najpopularnijih programskih jezika u svijetu, a koristi se u različitim industrijama i područjima razvoja softvera. U usporedbi s drugim popularnim jezicima, Python se ističe jednostavnom i čitljivom sintaksom, zbog čega ima široku primjenu u raznim industrijama. Python je interpretirani jezik, što znači da se izvršava liniju po liniju, interpretirajući izvorni kod u stvarnom vremenu, što olakšava brži razvoj i testiranje sustava. Varijable nisu strogo tipizirane, što omogućuje automatsko određivanje tipova podataka tijekom izvršavanja programa, pružajući fleksibilnost i brži razvoj. Možda najveća prednost mu je bogata standardna biblioteka koja se neprekidno razvija i nadograđuje, tako da za većinu problema Python već ima biblioteku ili modul.

Upravo jedna od tih biblioteka mi je pomogla pri povezivanju PostgreSQL baze podataka s Python programom. Koristio sam biblioteku psycopg2 kako bih se povezao na bazu podataka i radio upite na bazu, te na kraju dobivao rezultate u programu. Ispod su prikazane tri linije kôda s pomoću kojih sam se spojio na bazu podataka te izvršio upit koji mi vraća ukupan broj svih vozača Formule 1. Na ovaj način je moguće izvesti bilo kakav upit na bazu podataka kroz Python skriptu.

```
engine =
create_engine(f'postgresql+psycopg2://{db_config["user"]}:{db
_config["password"]}@{db_config["host"]}:{db_config["port"]}/
{db_config["database"]}')
query = 'select count(*) from drivers;'
df = pd.read_sql(query, engine)
```

Za stvaranje Python web poslužitelja koristio sam Pythonovu biblioteku Flask. Flask je biblioteka za Python koja omogućava brzo i jednostavno razvijanje web aplikacija i aplikacijskih sučelja. Popularan je zbog svoje fleksibilnosti i jednostavnosti, pružajući osnovne alate potrebne za razvoj, dok programerima ostavlja slobodu izbora dodatnih komponenti prema potrebama projekta. Slika 3.2 prikazuje kreiranje web poslužitelja koristeći Flask, te primjer dvije funkcije koje se aktiviraju kad korisnik zatraži url za koji su te funkcije predviđene.

```python
app = Flask(__name__)
CORS(app)

@app.route('/getTableInfo/<table>', methods=['GET'])
def getTableInfo(table):
    global Engine
    global TablesAndColumns
    returnValue = []
    if table == 'relations':
        multipleTableAction = FindForeignKeyAction(Engine, TablesAndColumns)
        multipleTableAction.act()
        returnValue += multipleTableAction.printResults()
    else:
        oneTableActions = [FindPrimaryKeyAction(Engine), FindNumberOfDistinctAction(Engine),
        for action in oneTableActions:
            action.act(table, TablesAndColumns[table])
            returnValue += action.printResults()

    response = jsonify({'result': returnValue})
    response.headers.add('Access-Control-Allow-Origin', '*')
    return response

@app.route('/getDBTables', methods=['GET'])
def getDBTables():
    global Engine
    global TablesAndColumns
    if (len(TablesAndColumns.keys()) > 0):
        response = jsonify({'tables': list(TablesAndColumns.keys())})
    else:
        response = jsonify({'tables': []})
    response.headers.add('Access-Control-Allow-Origin', '*')
    return response
```

Slika 3.2 – stvaranje web poslužitelja koristeći Flask biblioteku u Pythonu

## 3.2. PostgreSQL

PostgreSQL je moćan, besplatan, objektno-relacijski sustav za upravljanje bazama podataka otvorenog koda. Ovaj sustav podržava širok spektar naprednih značajki kao što su transakcijska kontrola, integritet podataka, proširivost, pune vanjske ključeve, indeksiranje, prozori funkcija, upiti s pomoću različitih vrsta podataka (npr. tekstualni, geografski), podrška za JSON, XML i mnoge druge. PostgreSQL je poznat po svojoj pouzdanosti, stabilnosti i podršci za ACID (engl. *Atomicity, Consistency, Isolation, Durability*) principa, što ga čini popularnim izborom za aplikacije koje zahtijevaju visoku razinu konzistentnosti i pouzdanosti podataka, poput financijskih aplikacija ili sustava za upravljanje poslovanjem.[1]

Nad kreiranom bazom podataka izvršavaju se dinamički složeni upiti pisani u SQL-u. Nakon toga, dobiveni rezultati se koriste u programu za izračunavanje i dobivanje finalnog produkta. Prikaz jedne od relacija pohranjene u PostgreSQL bazu podataka vidljiv je na Slika 3.3.

public.results/DiplProjekt/postgres@PostgreSQL 13

Query Editor | Query History

```
1  SELECT * FROM public.results
2  ORDER BY resultid ASC
```

Data Output | Explain | Messages | Notifications

| | resultid [PK] integer | raceid integer | driverid integer | constructorid integer | _number integer | grid integer | finalposition integer | points integer | laps integer | fastestlap integer |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 18 | 1 | 1 | 22 | 1 | 1 | 10 | 58 | 39 |
| 2 | 2 | 18 | 2 | 2 | 3 | 5 | 2 | 8 | 58 | 41 |
| 3 | 3 | 18 | 3 | 3 | 7 | 7 | 3 | 6 | 58 | 41 |
| 4 | 4 | 18 | 4 | 4 | 5 | 11 | 4 | 5 | 58 | 58 |
| 5 | 5 | 18 | 5 | 1 | 23 | 3 | 5 | 4 | 58 | 43 |
| 6 | 6 | 18 | 6 | 3 | 8 | 13 | 6 | 3 | 57 | 50 |
| 7 | 7 | 18 | 7 | 5 | 14 | 17 | 7 | 2 | 55 | 22 |
| 8 | 8 | 18 | 8 | 6 | 1 | 15 | 8 | 1 | 53 | 20 |
| 9 | 9 | 18 | 9 | 2 | 4 | 2 | [null] | 0 | 47 | 15 |
| 10 | 10 | 18 | 10 | 7 | 12 | 18 | [null] | 0 | 43 | 23 |
| 11 | 11 | 18 | 11 | 8 | 18 | 19 | [null] | 0 | 32 | 24 |
| 12 | 12 | 18 | 12 | 4 | 6 | 20 | [null] | 0 | 30 | 20 |
| 13 | 13 | 18 | 13 | 6 | 2 | 4 | [null] | 0 | 29 | 23 |
| 14 | 14 | 18 | 14 | 9 | 9 | 8 | [null] | 0 | 25 | 21 |
| 15 | 15 | 18 | 15 | 7 | 11 | 6 | [null] | 0 | 19 | 18 |
| 16 | 16 | 18 | 16 | 10 | 20 | 22 | [null] | 0 | 8 | 8 |
| 17 | 17 | 18 | 17 | 9 | 10 | 14 | [null] | 0 | 0 | [null] |

Slika 3.3 – prikaz tablice rezultata

## 3.3. Kaggle

Skup podataka se preuzet sa službenih stranica Kagglea[11]. Kaggle je platforma za natjecanje u znanosti o podacima i online zajednica znanstvenika koji se bave podacima i praktičara strojnog učenja pod Googleom LLC. Kaggle omogućuje korisnicima da pronađu i objave skupove podataka, istraže i izgrade modele u mrežnom okruženju podatkovne znanosti, rade s drugim podatkovnim znanstvenicima i inženjerima strojnog učenja te sudjeluju u natjecanjima za rješavanje izazova podatkovne znanosti.[3]

Za potrebe testiranja i demonstracije izabran je skup podataka koji je vezan za Formulu 1. Naime, on prati sve informacije o utrkama, vozačima, konstruktorima, krugovima, vremenima krugova i prvenstvima Formule 1 od 1950. do 2023. Izabran je baš taj skup zato što je interesantna tematika te je prikladan potrebama projekta.

## 3.4. React.js

React.js je besplatna JavaScript biblioteka otvorenog koda koja služi za izgradnju korisničkih sučelja temeljenih na komponentama. React se može koristiti za razvoj jednostraničnih, mobilnih ili poslužiteljskih aplikacija. Budući da se React bavi samo korisničkim sučeljem i učitavanjem komponenata, React aplikacije se često oslanjaju na biblioteke za usmjeravanje i druge biblioteke koje pružaju funkcionalnosti na klijentskoj strani aplikacije. Ključna prednost Reacta je ta što ponovno učitava samo one dijelove stranice koji su promijenjeni, izbjegavajući nepotrebno ponovno učitavanje nepromijenjenih elemenata stranice.[4]

Slika 3.4 prikazuje dohvat podataka web poslužitelja koristeći Reactovu funkciju „fetch“, nakon čijeg izvođenja je varijabla „tables“ postavljena na vrijednosti dohvaćene s poslužitelja. Nakon postavljanja varijabla se koristi za prikaz imena tablica u korisničkom sučelju.

```
const Results = () => {
    const navigate = useNavigate();
    const [filterInput, setFilterInput] = useState('');
    const [selectedTable, setSelectedTable] = useState('');
    const [tableAnalitics, setTableAnalitics] = useState([]);

    const [tables, setTables] = useState([]);
    const [filteredTables, setFilteredTables] = useState([]);

    useEffect(() => {
        const getTables = async () => {
            const res = await fetch("http://127.0.0.1:5000/getDBTables",
                {
                    method: "GET",
                });
            const data = await res.json();
            return data.tables;
        }
        getTables().then(tables => {
            if (tables.length === 0) {
                navigate("/");
            } else {
                setFilteredTables(tables);
                setTables(tables);
            }
        });
    }, [])
```

Slika 3.4 – dohvat podataka sa servera u React komponenti

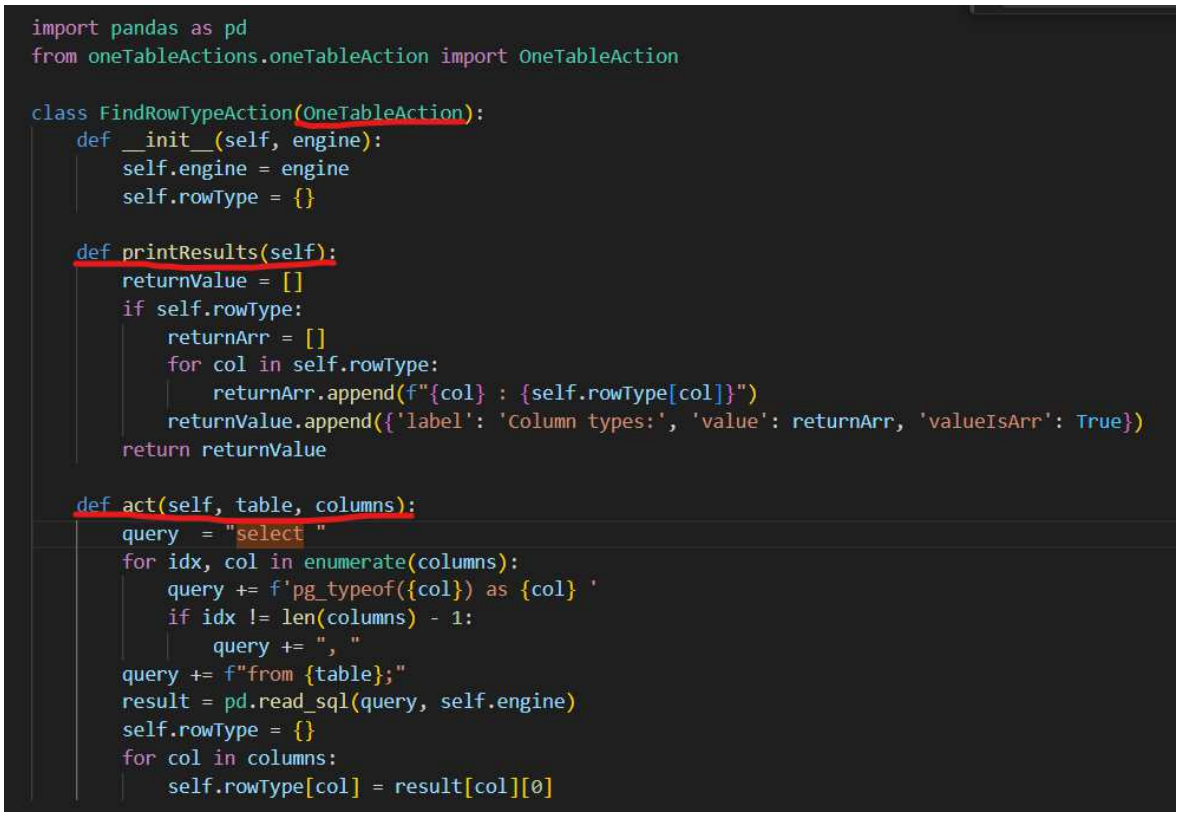
# 4. Implementacija sustava

Sustav je implementiran u programskom jeziku Python koristeći objektno orijentiranu paradigmu, koliko god to programski jezik Python dopušta. Pri implementaciji se koristilo nekoliko učestalih oblikovnih obrazaca u programiranju, od kojih je najvažnije istaknuti oblikovni obrazac strategija. Strategiju koristimo kad treba dinamički mijenjati ponašanje neke komponente (konteksta). Ponašanje zadajemo odabirom postupka, bez potrebe za mijenjanjem izvornog kôda konteksta. Obrazac je posebno koristan ako imamo više nezavisnih obitelji postupaka. Korištenjem strategije pospješuje se nadogradivost bez promjene (dodavanje novih postupaka), inverzija ovisnosti (kontekst ovisi o apstraktnom sučelju), te ortogonalnost (razdvajanje postupaka i konteksta).[5]

Konkretni primjer korištenja oblikovnog obrasca strategija u Pythonu vidljiv je na Slika 4.1, gdje je kreirana apstraktna klasa „OneTableAction“ i ona ima dvije metode koje njezina djeca moraju naslijediti: „printResult“ i „act“. Nakon toga, na Slika 4.2 vidimo kreiranje konkretne klase koja nasljeđuje apstraktnu klasu te onda mora implementirati njezine metode (na slici podvučeno crvenom bojom).

```
1  class OneTableAction:
2      def act(self, table, columns):
3          pass
4
5      def printResults(self):
6          pass
7
```

Slika 4.1 – apstraktna klasa „OneTableAction“

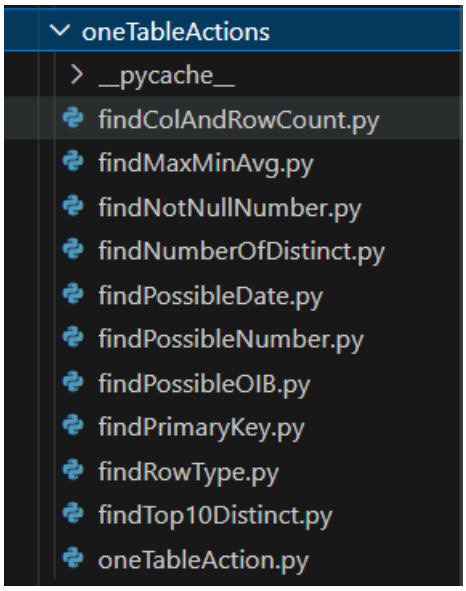


```python
import pandas as pd
from oneTableActions.oneTableAction import OneTableAction

class FindRowTypeAction(OneTableAction):
    def __init__(self, engine):
        self.engine = engine
        self.rowType = {}

    def printResults(self):
        returnValue = []
        if self.rowType:
            returnArr = []
            for col in self.rowType:
                returnArr.append(f"{col} : {self.rowType[col]}")
            returnValue.append({'label': 'Column types:', 'value': returnArr, 'valueIsArr': True})
        return returnValue

    def act(self, table, columns):
        query  = "select "
        for idx, col in enumerate(columns):
            query += f'pg_typeof({col}) as {col} '
            if idx != len(columns) - 1:
                query += ", "
        query += f"from {table};"
        result = pd.read_sql(query, self.engine)
        self.rowType = {}
        for col in columns:
            self.rowType[col] = result[col][0]
```

Slika 4.2 – konkretna klasa „FindRowTypeAction“

## 4.1. Akcije nad jednom tablicom

Implementaciju sustava i datoteke koje čine sustav se mogu podijeliti na dvije osnovne obitelji: datoteke koje su zadužene za akcije nad jednom tablicom baze podataka i tablice koje su zadužene za akcije nad više tablica baze podataka. Prva vrsta datoteka sadrži programski kod koji obavlja analizu i profiliranje tablice pojedinačno, a te se akcije pozivaju za svaku tablicu baze podataka. Na primjer, kad god korisnik klikne na neku od tablica u korisničkom sučelju pozvat će se niz akcija nad tom tablicom. Neke od tih akcija mogu biti pronalazak primarnog ključa, pronalazak broja jedinstvenih vrijednosti, provjera može li se ikoji tekstualni atribut tablice prikazati kao datumski tip itd. Primjer datoteka za rad nad jednom tablicom može se vidjeti na Slika 4.3 pod direktorijem „oneTableActions“. Ako pogledate imena datoteka uvidjet ćete da su to akcije koje su zadužene za obavljanje funkcionalnosti koje omogućuju realizaciju zahtjeva, kao što je navedeno u poglavlju Realizacija zahtjeva.




Slika 4.3 - struktura datoteka koje vrše akciju nad jednom tablicom

Kao primjer implementacije akcije nad jednom tablicom uzet ću akciju pronalaženja primarnog ključa. Akcija započinje tako da se pošalje upit na bazu podataka, a upit vraća omjer broja jedinstvenih vrijednosti koje nisu null i broja ukupnih vrijednosti za svaki stupac. Taj omjer se koristi kao pokazatelj može li taj stupac biti primarni ključ. Ako dobiveni omjer iznosi točno 1, to znači da su sve vrijednosti tog stupaca različite, te da se taj stupac dodaje u kandidate za primarni ključ. Dodatna funkcionalnost na ovu akciju je provjeravanje ima li ikoji atribut relacije slučajno kreiran indeks u bazi podataka. Tako se može sa sigurnošću reći da je takav atribut kandidat za primarni ključ, pa zato nije loše provjeriti postoje li indeksi. Dakle, ako postoje kandidati za primarni ključ kontrolira se ima li ikoji od njih kreiran indeks. Ako ima, dodaje se u novu listu kandidata s indeksom. Metoda „printResult“ samo vraća pronađene primarne ključeve u formatu pogodnom za prikaz na sučelju. Slika 4.4 prikazuje odsječak programskog koda koji obavlja gore navedene radnje.

```python
from oneTableActions.oneTableAction import OneTableAction

class FindPrimaryKeyAction(OneTableAction):
    def __init__(self, engine):
        self.engine = engine
        self.candidates = []
        self.candidatesWithIndex = []

    def printResults(self):
        retValue = []
        if len(self.candidates) > 0:
            retValue.append({'label': ["Candidates for primary key (100% uniqueness):"], 'va
            if len(self.candidatesWithIndex) > 0:
                retValue.append({'label': "Candidates for primary key (100% uniqueness AND h
        else:
            retValue.append({'label': "Candidates for primary key (100% uniqueness):", 'valu
        return retValue

    def act(self, table, columns):
        query  = "select "
        for idx, col in enumerate(columns):
            query += f'count(distinct {col}) * 1.0 / count(*) as {col} '
            if idx != len(columns) - 1:
                query += ", "
        query += f"from {table};"
        result = pd.read_sql(query, self.engine).values[0]
        self.candidates = []
        self.candidatesWithIndex.clear()
        for idx, perc in enumerate(result):
            if perc == 1.0:
                self.candidates.append(columns[idx])

        if (len(self.candidates) > 0):
            self.__checkForIndexes(table)


    def __checkForIndexes(self, table):
        query = f"select tablename, indexdef from pg_indexes where tablename = '{table}'"
        result = pd.read_sql(query, self.engine)
        candidatesWithIdxTemp = set()
        for index, row in result.iterrows():
            if "unique index" in row["indexdef"].lower():
                for candidate in self.candidates:
                    if candidate.lower() in row["indexdef"].lower():
                        candidatesWithIdxTemp.add(candidate)
        self.candidatesWithIndex = list(candidatesWithIdxTemp)
```

Slika 4.4 – programski kod za pronalazak primarnog ključa

## 4.2. Akcije nad više tablica

Druga vrsta datoteka sadrži programski kod koji se poziva samo jednom, ali on obuhvaća akcije nad svim tablicama baze podataka (ili onima koje su spomenute u konfiguracijskoj

datoteci). Kod u drugoj vrsti datoteka sadrži kompleksniju logiku i više linija koda, ali ima manje datoteka jer sustav nema previše akcija u koje ubraja sve tablice baze podataka. U ovoj verziji sustava jedina akcija nad više tablica je pronalazak potencijalnog stranog ključa.

Akcija pronalaska stranog ključa počinje tako da se iterira kroz sve tablice te se za svaku tablicu pronalaze kandidati za primarni ključ. Kandidati za primarni ključ se pronalaze koristeći akciju opisanu u poglavlju 4.1. Nakon toga, za svaki atribut (u nastavku primarni ključ) prolazi se kroz sve ostale tablice i sve ostale retke te se pregledava je li taj drugi stupac (u nastavku kandidat) podskup stupca primarnog ključa. Da bi se kandidat proglasio podskupom primarnog ključa on mora biti podskup vrijednosti primarnog ključa u mjeri barem 98 %. Namjerno nije postavljena mjera za podskup na 100 % jer je ostavljeno 2 % kao mjesto za pogrešku u bazi podataka. Kako bi se ovaj dugotrajan i zahtjevan posao vremenski optimizirao, kandidat se neće uzimati u obzir ako nakon dvadesetine iteracija usporedbe vrijednosti primarnog ključa i kandidata mjera podskupa vrijednosti kandidata nije veća od 50 %. Ako taj uvjet nije zadovoljen nije moguće ni da je taj kandidat podskup vrijednosti primarnog ključa u mjeri većoj od 98 %, te se kandidat može odbaciti, a vremenski resursi uštedjeti. Ako je pak kandidat podskup primarnog ključa u mjeri većoj od 98 %, tada se računa još jedna mjera korisna korisniku pri analiziranju koji od kandidata je stvarno strani ključ. Ta mjera je omjer broja jedinstvenih vrijednosti kandidata i broja vrijednosti primarnog ključa, te će se u nastavku nazivati mjera preklapanja. Za svakog kandidata koji je potencijalno strani ključ se računa mjera preklapanja jer ona govori koliki postotak vrijednosti primarnog ključa je kandidat koristio. Ako je mjera preklapanja jako malena, velika je šansa da taj kandidat nije zapravo strani ključ, iako su u vrijednosti podskup vrijednosti primarnog ključa. Potencijalni strani ključevi i opisane mjere se dodaju u polje koje se kasnije mapira i šalje putem web servera do korisničkog sučelja. Slika 4.5 prikazuje programski kod koji izvršava opisani proces pronalaženja potencijalnih stranih ključeva. Za pronalazak se poziva metoda „act“. Radi jednostavnosti mapiranje rezultata i ostatak klase nisu prikazani.

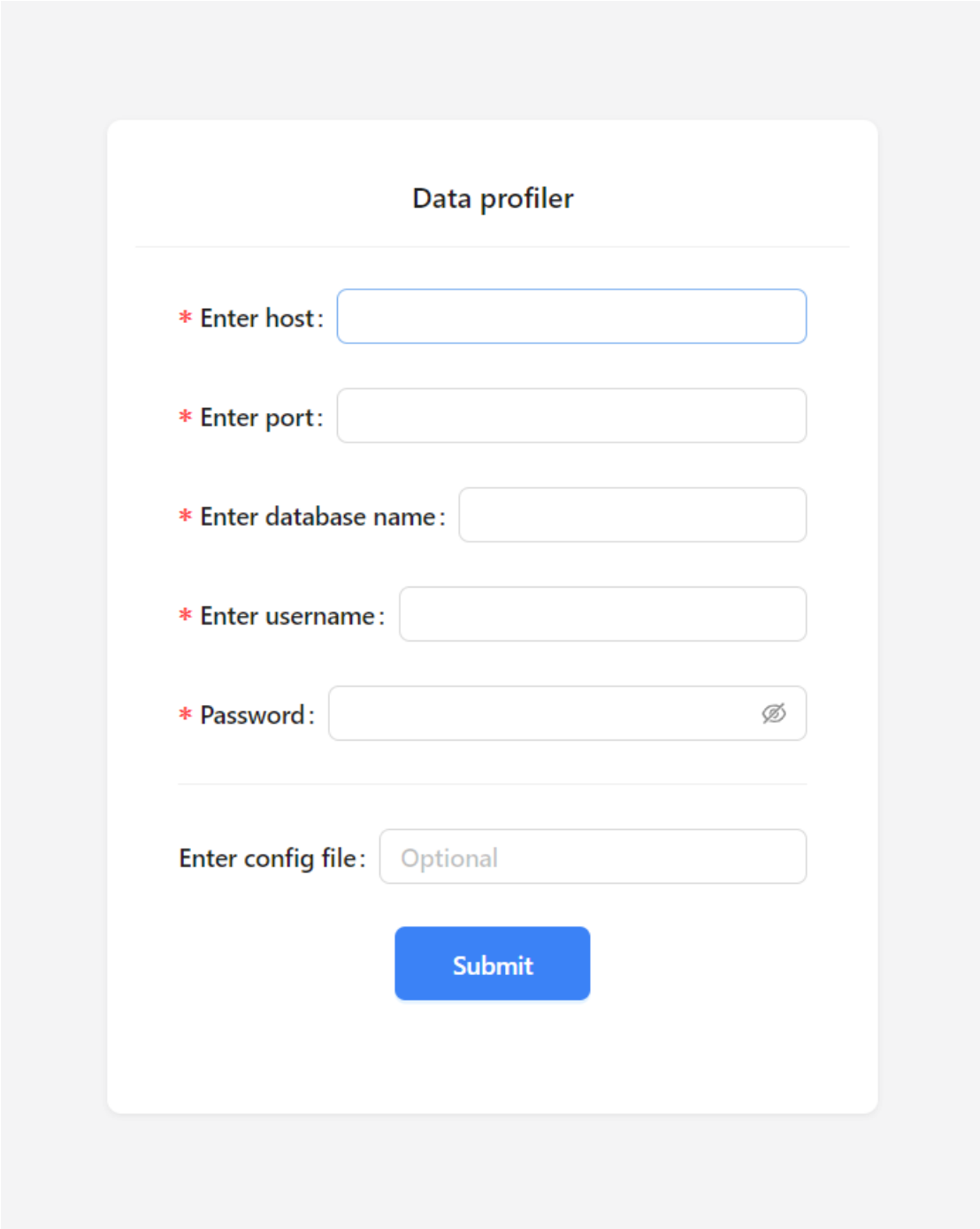
```
def act(self):
    primaryKeyFinder = FindPrimaryKeyAction(self.engine)
    self.candidates = []
    for table in self.tablesWithColumns:
        primaryKeyFinder.act(table, self.tablesWithColumns[table])
        for primaryKey in primaryKeyFinder.candidates:
            primaryKeyElements = pd.read_sql("select " + primaryKey + " from " + table, self.engine).iloc[:, 0].astype(str).to_list()
            for otherTable in self.tablesWithColumns:
                if otherTable != table:
                    self.__findPossibleForeignKey(table, primaryKey, primaryKeyElements, otherTable, self.tablesWithColumns[otherTable])


def __findPossibleForeignKey(self, aimedTable, aimedColumn, columnPattern, candidateTable, candidateColumns):
    columnElementsWithNullsNum = pd.read_sql("select count(*) num from " + candidateTable, self.engine).values[0][0]
    for column in candidateColumns:
        columnElementsWithoutNullsNum = pd.read_sql("select count(*) num from " + candidateTable + " where " + column + " is not null", self.engine).values[0][0]
        columnElements = pd.read_sql("select distinct(" + column + ") from " + candidateTable + " where " + column + " is not null", self.engine).iloc[:, 0].astype(str).to_list()
        numOfOverlaying = 0

        for idx, columnElement in enumerate(columnElements):
            if columnElement in columnPattern:
                numOfOverlaying += 1
            if (idx > len(columnElements) / 20) and (numOfOverlaying / idx < 0.5):
                break

        if (numOfOverlaying / len(columnElements)) > self.possibitityMargin:
            notNullPercentage = columnElementsWithoutNullsNum / columnElementsWithNullsNum
            possibility = numOfOverlaying / len(columnElements)
            overlap = (len(columnElements) / len(columnPattern)) * notNullPercentage
            if overlap > self.overlapMargin:
                self.candidates.append([aimedTable, aimedColumn, candidateTable, column, possibility, overlap])
```

Slika 4.5 – implementacija pronalaska stranog ključa

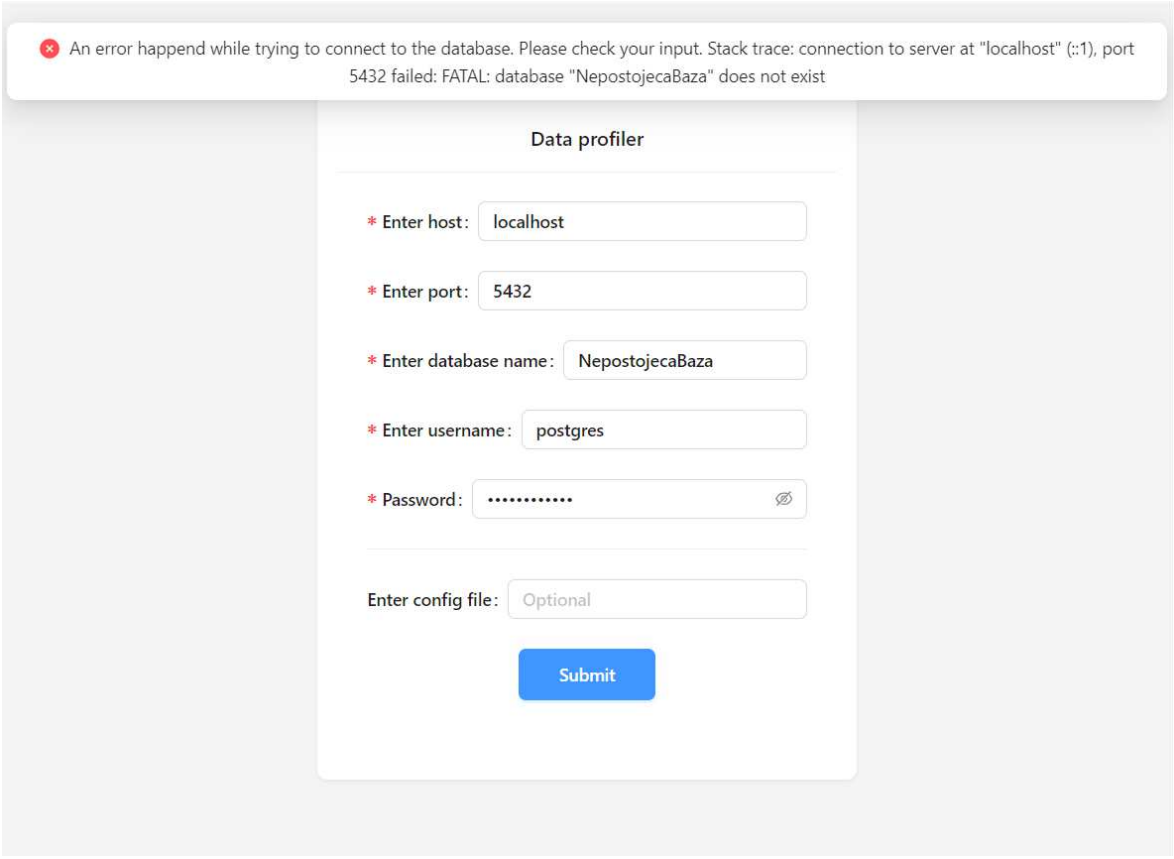
# 5. Upute za korištenje

## 5.1. Spajanje na bazu podataka, unos konfiguracijske datoteke

Nakon pokretanja sustav prikazuje formu za unos podataka za spajanje na bazu podataka. Početni zaslon vidljiv je na Slika 5.1. Nakon ispunjavanja forme, sustav se pokuša spojiti na unesenu bazu podataka, a ako ne uspije o tome će obavijestiti klijenta. Slika 5.2 prikazuje poruku nakon pogrešnog unosa. Iz poruke se može iščitati da se radi o pogrešno unesenom imenu baze podataka. Namjerno sam ostavio ispis izvoda zato što smatram da će sustav koristiti ljudi koji su upoznati s računalnim inženjerstvom te vjerujem da će im izvod stoga biti korisniji za uočiti pogrešku pri unosu podataka.

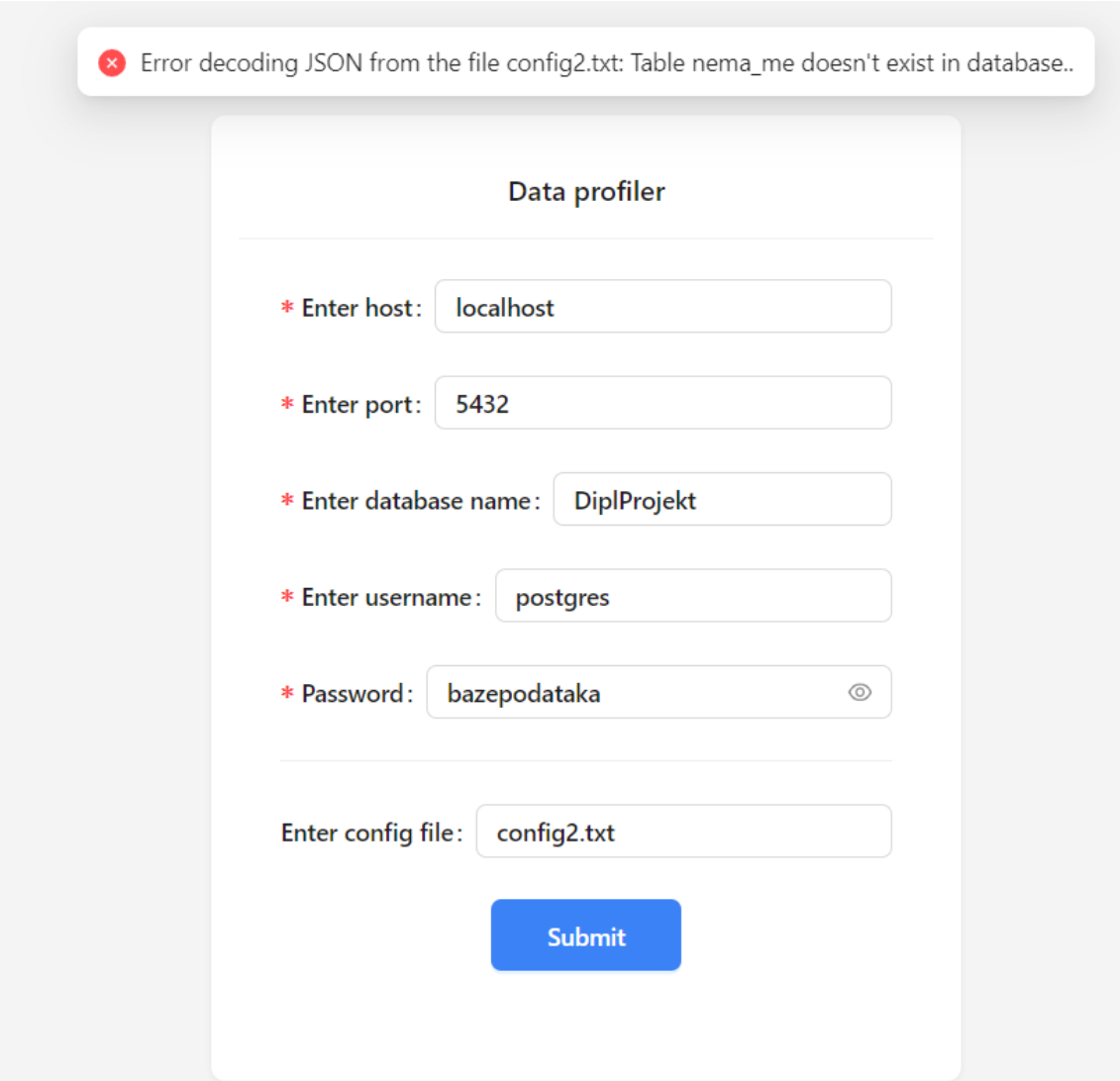



Slika 5.1 – početni zaslon aplikacije




Slika 5.2 – unos nepostojeće baze podataka

Ako korisnik želi dodati konfiguracijsku datoteku u formu, najprije treba pripremiti datoteku. Pod pripremiti smatra se osigurati da je tekst datoteke zapisan u JSON formatu i postaviti datoteku u direktorij „config“, koji se nalazi u korijenskom direktoriju projekta. Za osiguravanje JSON formata mogu se koristiti razni alati s interneta, na primjer JSON validator[10] za osiguravanje ispravnog JSON formata. Ako pak JSON format nije ispravan ili konfiguracijska datoteka sadrži neke relacije ili atribute koji ne postoje u bazi podataka, sustav će korisniku dojaviti i opisati grešku. Slika 5.3 prikazuje slučaj kada konfiguracijska datoteka sadrži relaciju „nema_me“, koja ne postoji u bazi podataka.

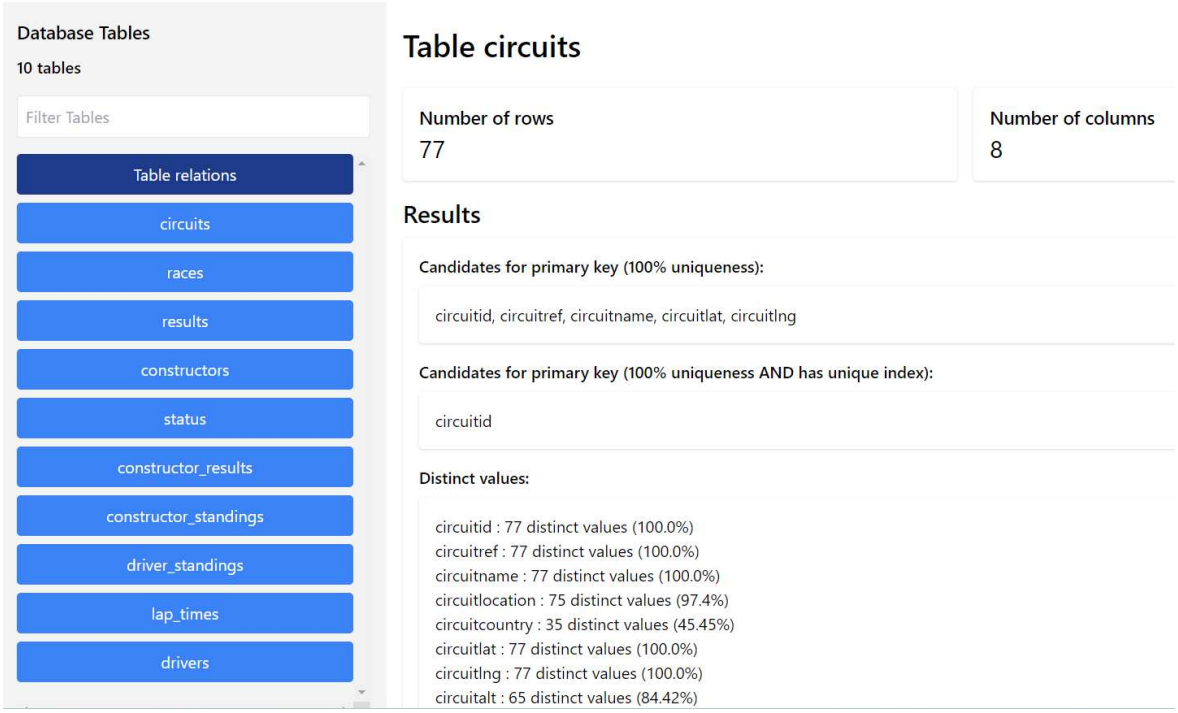



Slika 5.3 – greška u konfiguracijskoj datoteci

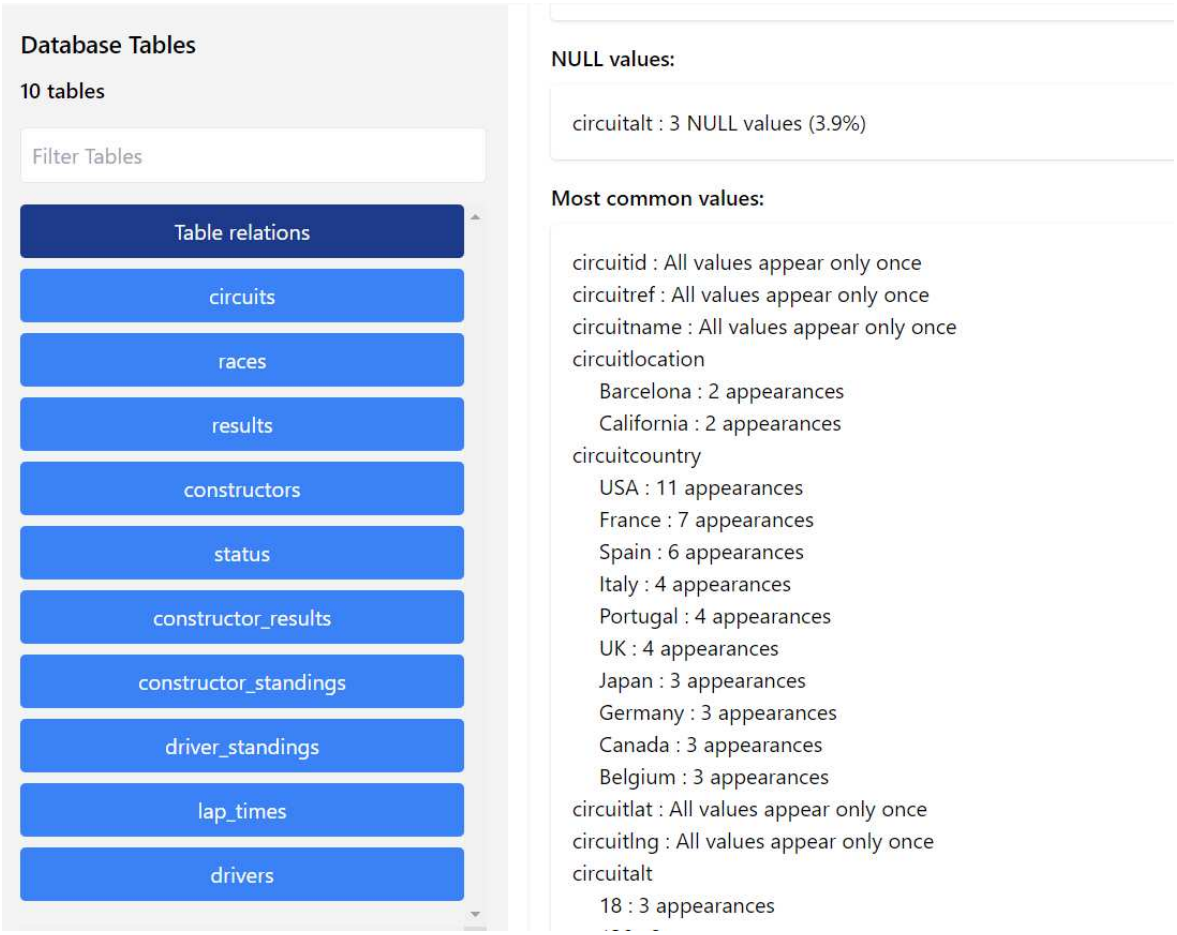
## 5.2. Profiliranje podataka

Nakon uspješnog spajanja na bazu podataka korisniku se s lijeve strane prikazuju sve tablice (u slučaju da konfiguracijska datoteka nije navedena) ili one tablice koje su spomenute u konfiguracijskoj datoteci. Korisnik mora kliknuti na jednu od opcija s lijeve strane kako bi započeo profiliranje i kreirao izvještaj. Slika 5.4, Slika 5.5 i Slika 5.6 prikazuju zaslon nakon korisnikovog klika na tablicu „circuits“. Redom su prikazani kandidati za primarni ključ, broj i postotak jedinstvenih vrijednosti za svaki atribut, postotak null vrijednosti za svaki atribut (ako postoje), najčešće vrijednosti i broj njihovog

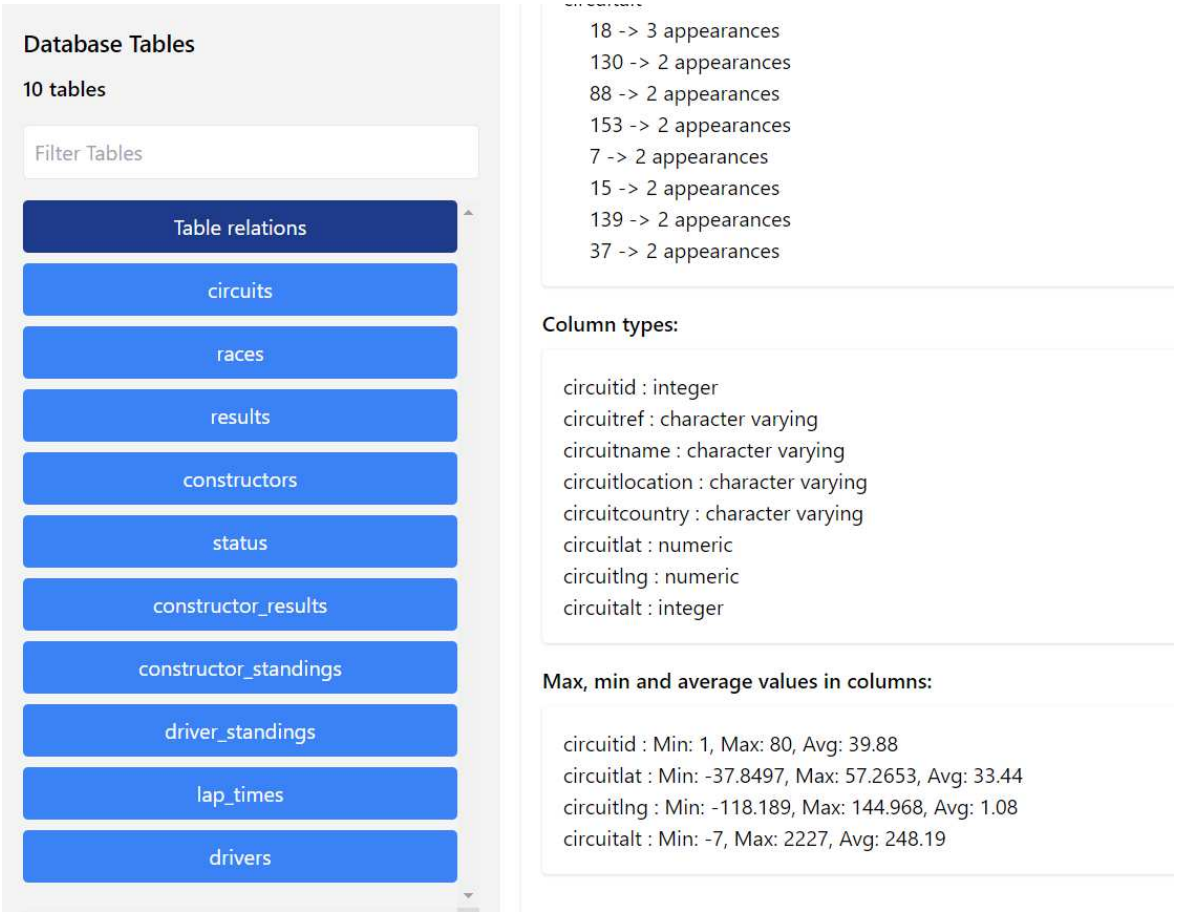
pojavljivanja za svaki atribut, tipovi atributa te minimum, maksimum i srednja vrijednost brojčanih atributa.




Slika 5.4 – izvještaj za tablicu „circuits“ (1. dio)

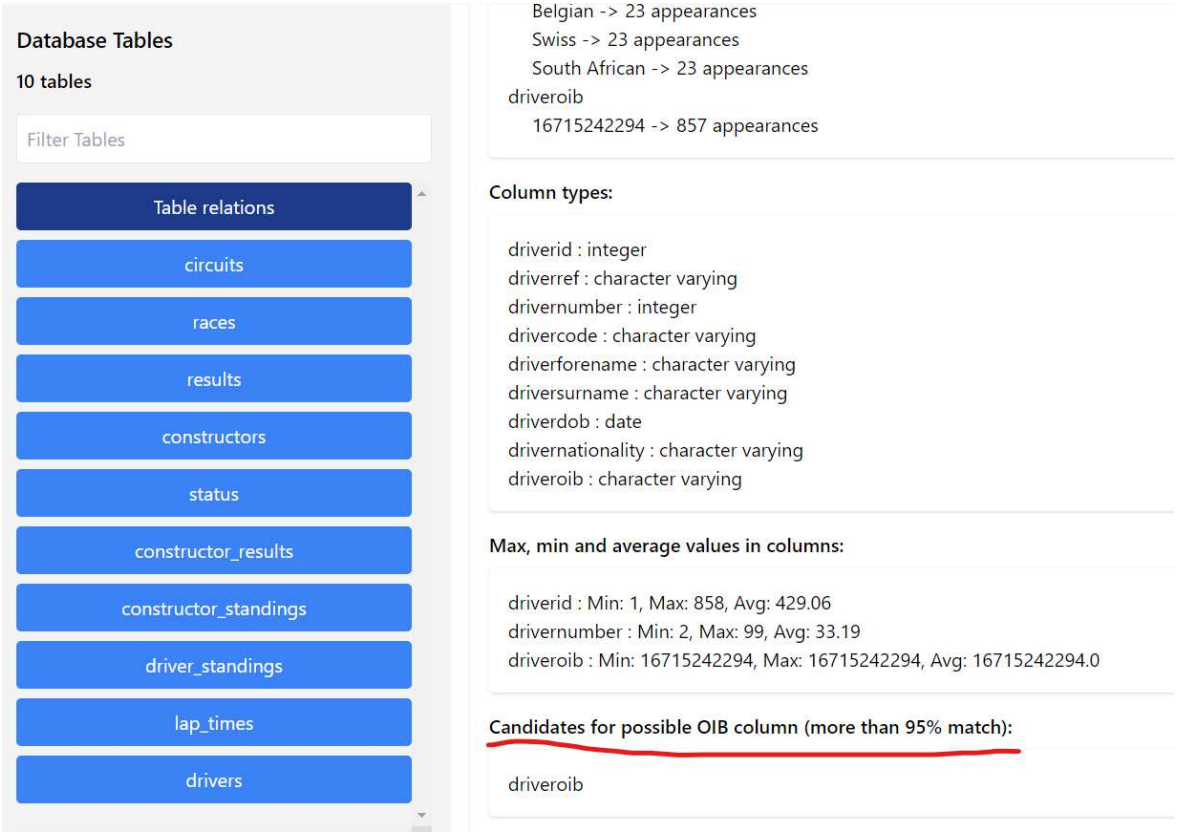



Slika 5.5 - izvještaj za tablicu „circuits“ (2. dio)

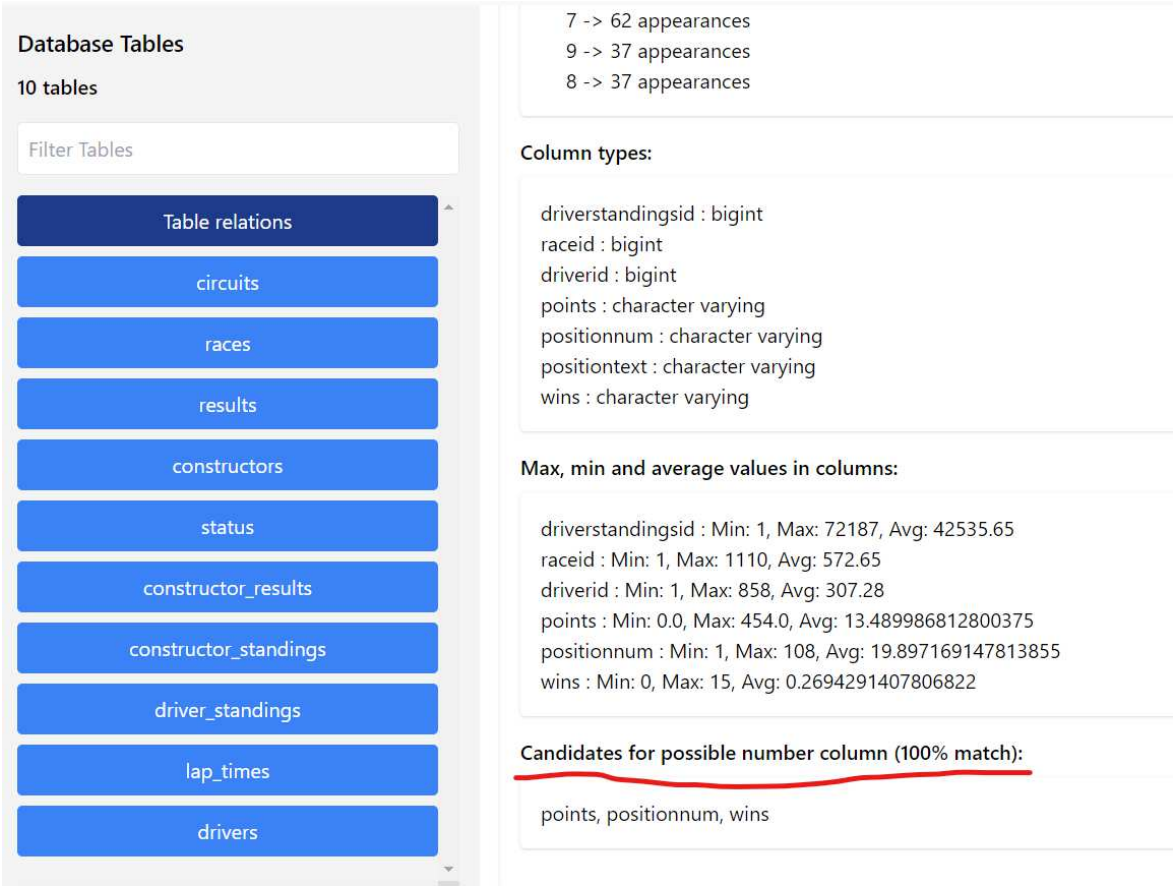



Slika 5.6 - izvještaj za tablicu „circuits“ (3. dio)

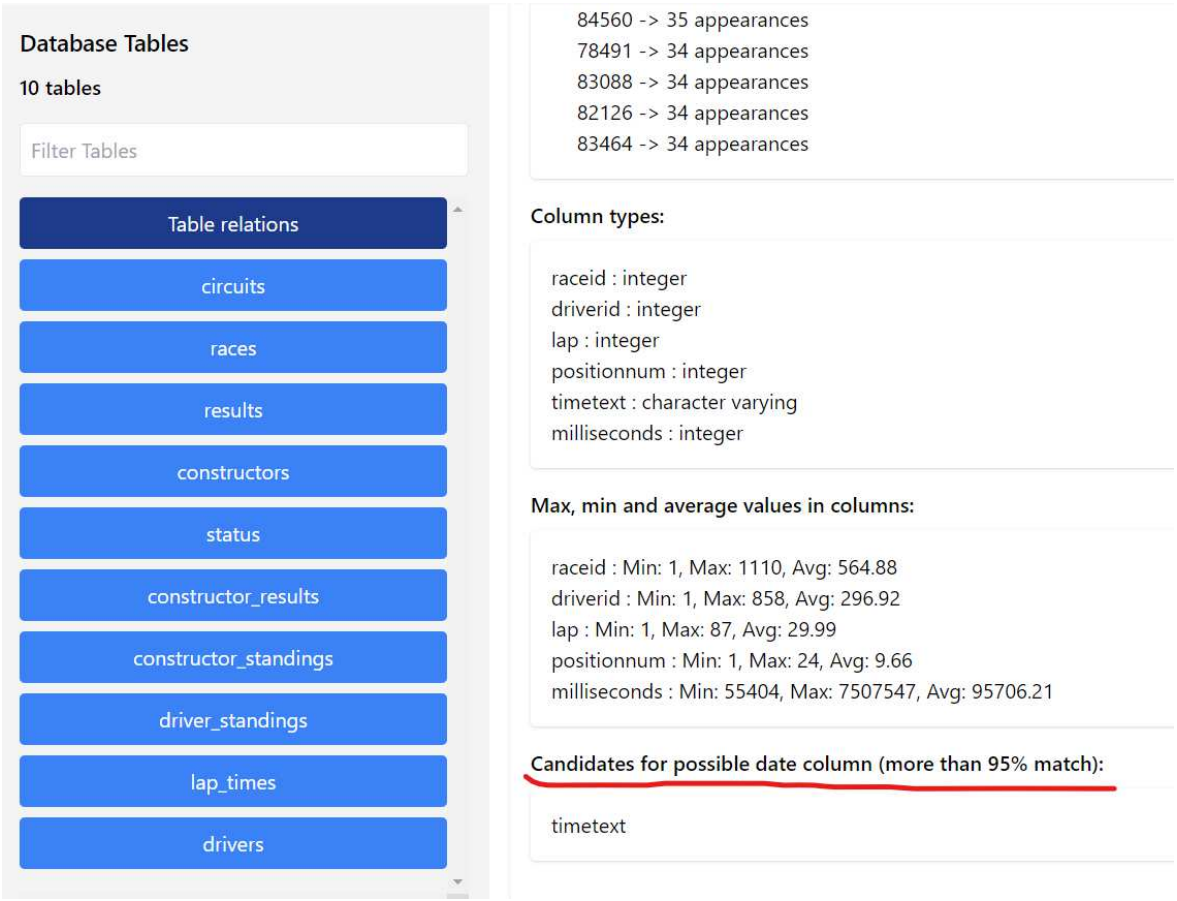
Sustav također ima mogućnost prepoznavanja OIB-a iz tekstualnog stupca, prepoznavanja datuma iz tekstualnog stupca, te uočavanja da je neki tekstualni stupac zapravo reprezentacija brojeva u tekstualnom obliku. Slika 5.7 prikazuje slučaj kada sustav u tablici „drivers“ pronađe stupac „driveroib“ kao potencijalni stupac koji prikazuje OIB-e vozača (crveno podebljano). Slika 5.8 prikazuje slučaj kada sustav dojavi da je pronašao sve brojčane vrijednosti u tekstualnim stupcima „points“, „positionnum“ i „wins“ (crveno podebljano). Slika 5.9 prikazuje slučaj kada sustav za tekstualni stupac posumnja da se radi o datumu (crveno podebljano).




Slika 5.7 – sustav pronalazi OIB-e iz tekstualnog stupca

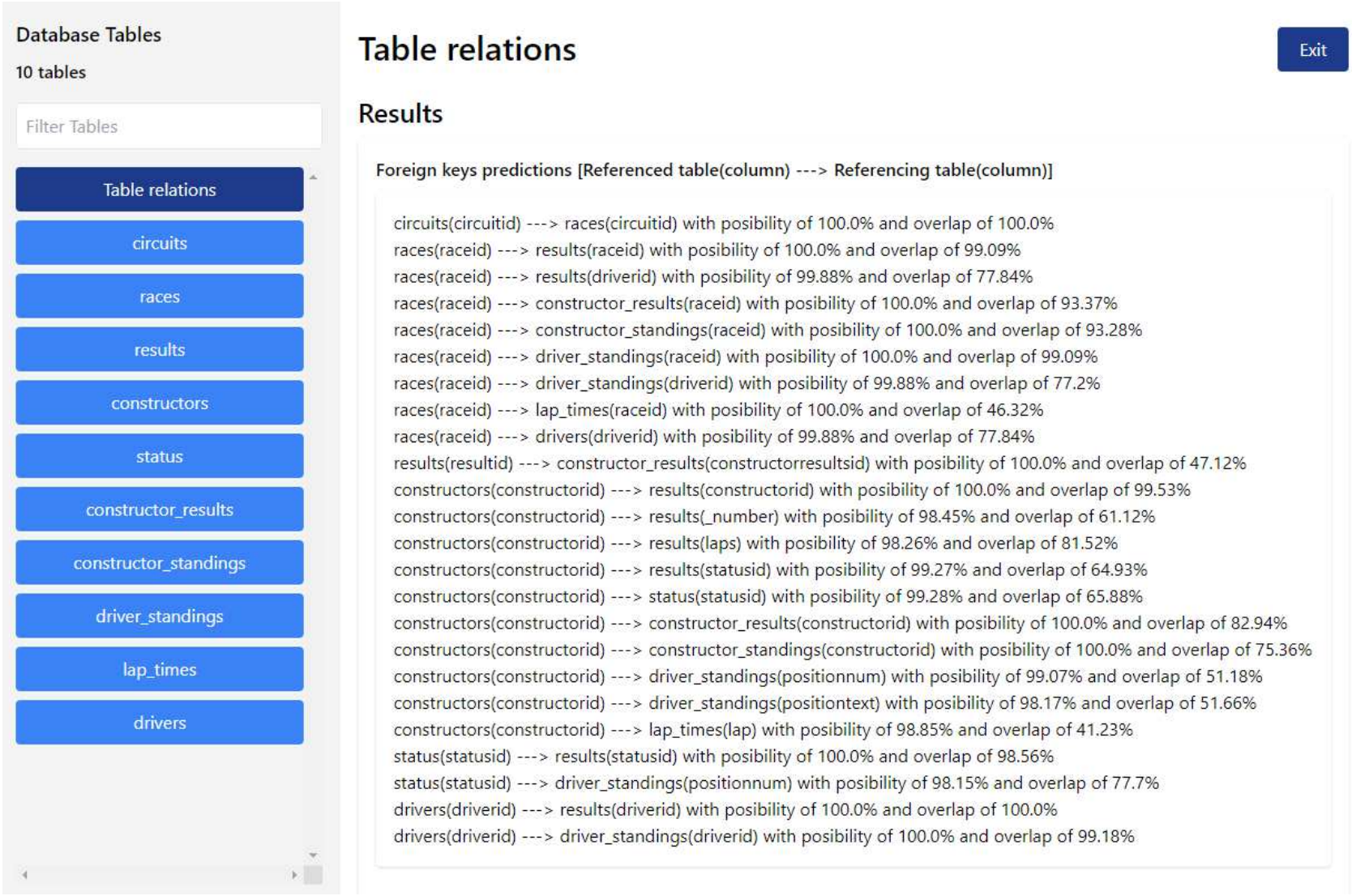



Slika 5.8 – sustav pronalazi brojčane vrijednosti iz tekstualnog stupca




Slika 5.9 – sustav pronalazi datume iz tekstualnog stupca

Korisnik na lijevoj strani uvijek ima dostupnu opciju poda nazivom „Table relations“. Ovo je opcija koja poziva operacije nad više tablica i u ovom slučaju pronalazi moguće strane ključeve unutar baze podataka. Klikom na tu opciju korisnik će morati malo pričekati (ovisno o veličini baze podataka) nakon čega dobiva izvještaj o potencijalnim stranim ključevima. Predikcije stranih ključeva su napisane tako da se najprije navede referencirana tablica i njezin atribut, pa nakon toga referencirajuća tablica i jedan od njezinih atributa, te dvije mjere važne korisniku za bližu analizu (mjera podskupa i mjera preklapanja), objašnjene u poglavlju 4.2. Prikaz potencijalnih stranih ključeva u izvještaju vidljiv je na Slika 5.10.




Slika 5.10 – sustav pronalazi potencijalne strane ključeve u bazi podataka

# 6. Ograničenja i moguća poboljšanja

Izrađeni sustav posjeduje brojne funkcionalnosti i ispunjava zahtjeve klijenta, no potrebno je biti svjestan njegovih nedostataka i ograničenja. Postoje funkcionalnosti koje sustav trenutno ne može obaviti i aspekti koji bi se mogli dodati u budućim nadogradnjama kako bi se povećala njegova funkcionalnost, poboljšala kvaliteta i korisnicima pružilo još više korisnih opcija.

Prvo i najvažnije ograničenje sustava je to što ne može pronaći kompozitni primarni ključ. Kad bi se ova funkcionalnost implementirala, ona bi mnogostruko povećala složenost i vrijeme izvedbe profiliranja, jer je potrebno pregledati sve kombinacije svih stupaca tablice (i tako za svaku tablicu) kako bi se utvrdilo postoje li uvjeti za kompozitni ključ. Svakako smatram da bi, unatoč vremenskoj složenosti, prva nadogradnja na ovaj sustav trebala biti akcija za pronalazak kompozitnog ključa i njezina optimizacija.

Drugo ograničenje sustava je to što se profiliranje tablice vrši svaki put kad korisnik u korisničkom sučelju klikne na tu tablicu. Ovaj problem je najviše vremenske prirode zato što ako korisnik zatraži profiliranje neke druge tablice pa se nakon toga vrati na prvotnu, morat će ponovo čekati da se profiliranje obavi, što u slučajevima većih tablica (više od milijun redaka) može trajati nekoliko minuta. Ovo ograničenje je najviše vezano za korisničko iskustvo dok koristi aplikaciju, te bi se moglo riješiti tako da svaka akcija pamti svoje izvještaje, te ako ima zapamćen izvještaj da pošalje njega umjesto da radi ponovno profiliranje. Ovo je svakako funkcionalnost koja bi bila među prvim nadogradnjama sustava.

Jedno od mogućih poboljšanja aplikacije bi moglo biti mogućnost preuzimanja izvještaja u određenom formatu (.pdf, .csv, .xls itd.). Smatram da bi korisniku bilo korisno preuzeti izvještaj za svaku tablicu, ili za sve tablice skupno, jer bi tako mogao pohraniti te podatke na svoje računalo i imati ih uvijek dostupnima.

Drugo moguće poboljšanje vezano je za konfiguracijsku datoteku. U trenutnoj verziji aplikacije konfiguracijsku datoteku kreira korisnik. Iako sustav ima implementiranu validaciju konfiguracijske datoteke, dosta se tu posla ostavlja korisniku, čime se povećava

mogućnost namjerne ili slučajne pogreške. Ne bi bilo loše kad bi korisnik imao opciju unutar aplikacije odabrati relacije i atribute koje želi profilirati, a aplikacija mu sama kreira konfiguracijsku datoteku. Tako bi manje posla palo na korisnika, korisnik bi jednostavnije kreirao konfiguracijske datoteke i sigurno bi se smanjila mogućnost pogreške u nekom postotku.

# Zaključak

Korištenjem navedenih alata i tehnologija, uspješno je izgrađen sustav za profiliranje podataka. Sustav zadovoljava sve korisničke zahtjeve, ima jednostavno korisničko sučelje i intuitivan je za korištenje.

Iako postoji prostor za napredak, kao što je proširenje opcija profiliranja, optimizacija performansi aplikacije ili poboljšanje korisničkog iskustva, smatram da aplikacija pokriva temeljne metode profiliranja. Profiliranjem baza podataka ovim sustavom moguće je otkriti zanimljive informacije, korisne uvide o tablicama baze podataka te brzo pronaći međusobne odnose tablica i strane ključeve.

Implementacijom ovakvog sustava, tvrtke mogu značajno smanjiti rizike povezane s nekvalitetnim podacima i poboljšati pouzdanost svojih analitičkih i operativnih aktivnosti. Tako benefiti profiliranja podataka izlaze na vidjelo u praksi te profiliranje podataka postaje neizostavan korak u svakodnevnom poslovanju i dugoročnoj strategiji razvoja.

# Literatura


[1] What is Data Profiling, https://www.ibm.com/topics/data-profiling, 12.06.2024.

[2] PostgreSQL: About, https://www.postgresql.org/about/, 25.03.2024.

[3] A Beginner's Guide to Kaggle for Data Science, https://www.makeuseof.com/beginners-guide-to-kaggle/, 25.03.2024.

[4] React (JavaScript library), https://en.wikipedia.org/wiki/React_(JavaScript_library), 10.06.2024.

[5] Nastavni materijali kolegija Oblikovni obrasci u programiranju, Fakultet elektrotehnike i računarstva, 2023

[6] PostgreSQL documentation, https://www.postgresql.org/docs/, 17.04.2024.

[7] Ant design components, https://ant.design/components/overview/, 05.06.2024.

[8] Welcome to Flask – Flask Documentation (3.0.x), https://flask.palletsprojects.com/en/3.0.x/, 01.06.2024.

[9] Introduction to Psycopg2 module in Python, https://www.geeksforgeeks.org/introduction-to-psycopg2-module-in-python/, 20.04.2024.

[10] JSON Lint: JSON Online Validator and Formatter, https://jsonlint.com/, 15.06.2024.

[11] Formula 1 World Championship (1950 - 2023), https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020?resource=download&select=pit_stops.csv, 17.06.2024.

# Sažetak


## Sustav za profiliranje i otkrivanje podataka

Tema ovog diplomskog rada bila je izrada sustava za profiliranje i otkrivanje podataka. Sustav funkcionira tako da korisnik upiše podatke za pristup bazi podataka, a sustav izvrši profiliranje tablica unutar baze podataka. Neke od funkcionalnosti koje sustav pokriva su: pronalazak najučestalijih vrijednosti u stupcu, pronalazak primarnog ključa, prepoznavanje brojčane vrijednosti, datuma ili OIB-a unutar tekstualnog stupca itd. Sustav također ima sposobnost pronalaska potencijalnih stranih ključeva unutar baze podataka, uz prikaz vjerojatnosti i mjera korisnih za daljnju analizu. Sustav može primiti konfiguracijsku datoteku te na temelju nje odabrati samo određene relacije i atribute u bazi podataka. Korisnik se koristi sustavom putem jednostavnog i intuitivnog sučelja.



*Ključne riječi: profiliranje podataka, otkrivanje podataka, primarni ključ, strani ključ, najčešće vrijednosti stupca, PostgreSQL baza podataka, Python, React.js*

# Summary


## System for profiling and data discovery

The topic of this thesis was the creation of a system for profiling and data discovery. The system works so that the user enters data to access the database, and the system performs profiling of the tables within the database. Some of the functionalities covered by the system are: finding the most frequent values in a column, finding the primary key, recognizing a numeric value, date or OIB within a text column, etc. The system also has the ability to find potential foreign keys within the database, displaying probabilities and measures useful for further analysis. The system can receive a configuration file then based on it, select only certain relations and attributes in the database. The user uses the system through a simple and intuitive interface.



*Keywords: data profiling, data discovery, primary key, foreign key, most common column values, PostgreSQL database, Python, React.js*