

SKYGRAPH: PRONAĐI SVOJ IDEALNI ODMOR U EUROPI

Seminar iz kolegija Napredne baze podataka

Sadržaj

1. UVOD.....	2
2. PRIKUPLJANJE I PRIPREMANJE PODATAKA	3
2.1. Podaci o aerodromima i letovima.....	3
2.2. Podaci o gradovima	5
3. POSTAVLJANJE BAZE U MEMGRAPH	7
3.1. Instalacija Memgrapha	7
3. 2. Kreiranje vrhova i bridova	8
4. UPITI NA BAZU	12
4.1. Prikaz podataka baze	12
4.2. Lista svih gradova (s ili bez vlastitog aerodroma)	13
4.3. Pronalazak najbližeg aerodroma za gradove polaska i odredišta.....	14
4.4. Najjeftiniji direktni let.....	15
4.5. Najjeftiniji let s mogućim presjedanjima i najviše 2 dana ukupnog putovanja	16
4.5. Vremenski najkraći let s mogućim presjedanjima i najviše 1 danom putovanja	17
5. ZAKLJUČAK	18
6. LITERATURA	18

1. UVOD

U ovom seminaru opisat ću cijeli postupak kreiranja grafovske baze podataka koji uključuje prikupljanje i pripremu podataka, stvaranje grafovske baze na javno dostupnoj platformi Memgraph i konverziju podataka u vrhove i bridove sa određenim atributima, a zatim i izvršenje smislenih upita nad kreiranom bazom. Za demonstraciju sam odabrala podatke o letovima i gradovima jer je ova domena prirodno povezana s grafovskim strukturama. Zračni promet zapravo funkcionira kao mreža gdje su aerodromi čvorovi, a letovi veze među njima, što se savršeno uklapa u koncept grafovske baze podataka.

Ovakav izbor ima nekoliko prednosti. Prvo, podaci o letovima su po svojoj prirodi međusobno povezani - jedan let spaja dva aerodroma, a više letova može stvarati kompleksne rute. U grafovskoj bazi takve veze se jednostavno prikazuju, dok bi u klasičnoj relacijskoj bazi bilo potrebno složeno povezivanje tablica. Drugo, grafovske baze su odlične za upite vezane uz pronalaženje puteva i povezanosti, što je ključno kod pretraživanja letova. Na primjer, možemo lako postaviti upit koji traži sve moguće rute od Zagreba do Porta s najviše 2 presjedanja, što bi u SQL-u bilo puno kompliciranije. Treće, model je fleksibilan i lako se može nadograđivati - dodavanje novih letova, aerodroma ili atributa poput cijene, vremena putovanja ili sezonske dostupnosti ne zahtijeva promjenu strukture baze.

Ideja za ovaj projekt došla je iz mog osobnog interesa za putovanja i čestog korištenja aplikacija poput Google Flights i Skyscanner. Zanimalo me kako u pozadini rade sustavi koji mogu brzo pronaći najbolje rute između različitih lokacija, uzimajući u obzir brojne faktore poput cijene, trajanja leta i broja presjedanja.

2. PRIKUPLJANJE I PRIPREMANJE PODATAKA

2.1. Podaci o aerodromima i letovima

Za izradu grafovske baze podataka Skygraph bilo je potrebno prikupiti kvalitetne i relevantne podatke o letovima između različitih aerodroma. Prilikom istraživanja dostupnih izvora podataka susrela sam se s nekoliko izazova. Naime, podaci o letovima, a posebno oni koji uključuju trenutne cijene, vremena letova i raspoložive rute rijetko su dostupni u strukturiranom obliku bez naknade. Prvi pokušaj bio je pretraživanje javno dostupnih skupova podataka na stranicama poput Kaggle-a, ali oni nisu sadržavali aktualne podatke o letovima s potrebnim atributima. Zatim, većina komercijalnih API-ja aviokompanija i platformi za usporedbu letova primarno je orijentirana na B2B suradnju s turističkim agencijama i drugim pružateljima usluga, što ih čini nedostupnima za individualne akademske projekte, a besplatni probni periodi ne bi bili dovoljni za potrebe projekta.

Nakon razmatranja svih dostupnih opcija, odlučeno je implementirati vlastito rješenje web scrapinga platforme Google Flights, koja predstavlja jedan od najobuhvatnijih izvora podataka o letovima. Za implementaciju je korištena biblioteka "fast-flights" iz GitHub repozitorija (<https://github.com/AWeirdDev/flights>), koja omogućuje dohvaćanje podataka s Google Flights platforme te je bazirana na dekodiranju Base64-kodiranih Protobuf stringova kojima Google Flights interno prenosi podatke o letovima.

Za potrebe projekta, ručno je kreirana CSV datoteka s popisom 35 odabranih aerodroma diljem Europe za koje će se prikupljati podaci o letovima. Svaki zapis u datoteci sadrži IATA kod aerodroma, grad u kojem se nalazi i pripadajuću državu.



```
nodesAirports.csv > data
1 code,city,country
2 FCO,Rome,Italy
3 LHR,London,United Kingdom
4 AMS,Amsterdam,Netherlands
5 ZAG,Zagreb,Croatia
6 FRA,Frankfurt,Germany
7 VIE,Vienna,Austria
8 ATH,Athens,Greece
9 MUC,Munich,Germany
10 CPH,Copenhagen,Denmark
11 LIS,Lisbon,Portugal
12 DUB,Dublin,Ireland
13 OSL,Oslo,Norway
14 BUD,Budapest,Hungary
15 BGY,Milano,Italy
```

Slika 1: Prvih nekoliko redaka CSV datoteke s podacima o aerodromima

Osnovna funkcionalnost biblioteke "fast-flights" proširena je dodatnom Python skriptom koja je omogućila sistematizirano prikupljanje podataka o letovima između 35 prethodno odabranih europskih aerodroma u razdoblju od 4. do 10. kolovoza 2025.godine, te njihovo spremanje u CSV datoteku. Datoteka sadrži sljedeće podatke za svaki let:

- departureCity (grad polaska)
- departureAirport (IATA kod aerodroma polaska)
- destinationCity (grad slijetanja)
- destinationAirport (IATA kod aerodroma slijetanja)
- company (aviokompanija koja obavlja let)
- departureTimestamp (vrijeme polaska u timestamp formatu)
- arrivalTimestamp (vrijeme slijetanja u timestamp formatu)
- duration (trajanje leta u minutama)
- price (cijena leta u eurima)

	A	B	C	D	E	F	G	H	I	
1	departureCity	departureAirport	destinationCity	destinationAirport	company	departureTimestamp	arrivalTimestamp	duration	price	Upiti i veze
2	Rome	FCO	London	LHR	British Airways	4.8.2025 18:00	4.8.2025 19:35	155	222	Upiti Veze
3	Rome	FCO	London	LHR	British Airways	4.8.2025 16:50	4.8.2025 18:25	155	296	Broj upita: 2
4	Rome	FCO	London	LHR	British Airways	4.8.2025 20:30	4.8.2025 22:05	155	296	pathsFlights
5	Rome	FCO	London	LHR	British Airways	4.8.2025 11:20	4.8.2025 13:10	170	400	Broj učitanih redaka: 45.563. Sljedeći broj pogrešaka: 129.
6	Rome	FCO	London	LHR	British Airways	4.8.2025 12:40	4.8.2025 14:30	170	400	Pogreške u upitu pathsFlights
7	Rome	FCO	London	LHR	British Airways	4.8.2025 15:30	4.8.2025 17:10	160	400	Samo veza.
8	Rome	FCO	London	LHR	British Airways	4.8.2025 16:50	4.8.2025 18:25	155	296	
9	Rome	FCO	London	LHR	British Airways	4.8.2025 20:30	4.8.2025 22:05	155	296	
10	Rome	FCO	London	LHR	British Airways	4.8.2025 11:20	4.8.2025 13:10	170	400	
11	Rome	FCO	London	LHR	British Airways	4.8.2025 12:40	4.8.2025 14:30	170	400	
12	Rome	FCO	London	LHR	British Airways	4.8.2025 15:30	4.8.2025 17:10	160	400	
13	Rome	FCO	London	LHR	British Airways	5.8.2025 20:35	5.8.2025 22:10	155	166	
14	Rome	FCO	London	LHR	British Airways	5.8.2025 21:05	5.8.2025 22:45	160	166	
15	Rome	FCO	London	LHR	British Airways	5.8.2025 16:00	5.8.2025 17:40	160	185	
16	Rome	FCO	London	LHR	British Airways	5.8.2025 6:55	5.8.2025 8:40	165	197	
17	Rome	FCO	London	LHR	British Airways	5.8.2025 16:50	5.8.2025 18:30	160	197	
18	Rome	FCO	London	LHR	British Airways	5.8.2025 12:45	5.8.2025 14:35	170	210	
19	Rome	FCO	London	LHR	British Airways	5.8.2025 11:50	5.8.2025 13:40	170	247	
20	Rome	FCO	London	LHR	British Airways	5.8.2025 7:15	5.8.2025 9:00	165	378	
21	Rome	FCO	London	LHR	British Airways	5.8.2025 20:35	5.8.2025 22:10	155	166	
22	Rome	FCO	London	LHR	British Airways	5.8.2025 21:05	5.8.2025 22:45	160	166	
23	Rome	FCO	London	LHR	British Airways	5.8.2025 6:55	5.8.2025 8:40	165	197	
24	Rome	FCO	London	LHR	British Airways	5.8.2025 16:50	5.8.2025 18:30	160	197	
25	Rome	FCO	London	LHR	British Airways	5.8.2025 12:45	5.8.2025 14:35	170	210	
26	Rome	FCO	London	LHR	British Airways	5.8.2025 11:50	5.8.2025 13:40	170	247	
27	Rome	FCO	London	LHR	British Airways	5.8.2025 7:15	5.8.2025 9:00	165	378	
28	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 10:25	5.8.2025 13:00	155	180	
29	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 17:25	5.8.2025 19:55	150	180	
30	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 12:45	5.8.2025 15:10	145	184	
31	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 6:30	5.8.2025 8:50	140	180	
32	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 19:40	5.8.2025 22:00	140	180	
33	Rome	FCO	Amsterdam	AMS	ITA	5.8.2025 8:30	5.8.2025 11:00	150	202	
34	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 10:25	5.8.2025 13:00	155	180	
35	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 17:25	5.8.2025 19:55	150	180	
36	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 6:30	5.8.2025 8:50	140	180	
37	Rome	FCO	Amsterdam	AMS	KLM	5.8.2025 19:40	5.8.2025 22:00	140	180	
38	Rome	FCO	Amsterdam	AMS	ITA	5.8.2025 8:30	5.8.2025 11:00	150	202	
39	Rome	FCO	Amsterdam	AMS	ITA	4.8.2025 14:35	4.8.2025 17:05	150	177	
40	Rome	FCO	Amsterdam	AMS	KLM	4.8.2025 12:45	4.8.2025 15:10	145	180	
41	Rome	FCO	Amsterdam	AMS	KLM	4.8.2025 17:25	4.8.2025 19:55	150	184	
42	Rome	FCO	Amsterdam	AMS	KLM	4.8.2025 19:40	4.8.2025 22:00	140	180	

Slika 2: dio CSV datoteke sa podacima o letovima prikazan u Excel-u

Na Slici 2 vidimo da je prikupljeno podataka o 45 563 letova, za njih 129 utvrđena je pogreška jer za njih nisu bili dostupni podaci o cijeni odnosno u CSV-u na pripadnim mjestima zapisano je „Price unavailable“. Prilikom unošenja podataka u grafovsku bazu ti letovi čija cijena nije poznata bit će izostavljeni.

2.2. Podaci o gradovima

Kako bi korisnici mogli lakše pretraživati letove koristeći nazive gradova, a ne samo aerodrome, u bazu su dodani i podaci o gradovima koji nemaju vlastite aerodrome. Tako korisnik može tražiti letove iz manjeg grada npr. Čakovca do Pariza, a sustav će automatski pronaći letove iz najbližeg aerodroma, u ovom slučaju Zagreba.

Za prikupljanje podataka o gradovima korišten je javno dostupni skup podataka sa stranice [simplemaps.com](https://simplemaps.com/data/world-cities) (<https://simplemaps.com/data/world-cities>), koji sadrži opsežne informacije o svjetskim gradovima uključujući broj stanovnika i geografske koordinate korištene u ovom projektu. Najprije je ručno kreiran popis gradova i njihovih najbližih aerodroma (od onih 35 odabranih) u obliku liste sa Slike 3.

```
cities.py > ...
1  Cities = [
2      {
3          "city": "Varazdin",
4          "country": "Croatia",
5          "nearest_airport_code": "ZAG",
6          "nearest_airport": "Zagreb",
7          "airport_country": "Croatia"
8      },
9      {
10         "city": "Cakovec",
11         "country": "Croatia",
12         "nearest_airport_code": "ZAG",
13         "nearest_airport": "Zagreb",
14         "airport_country": "Croatia"
15     },
16     {
17         "city": "Maribor",
18         "country": "Slovenia",
19         "nearest_airport_code": "ZAG",
20         "nearest_airport": "Zagreb",
21         "airport_country": "Croatia"
22     },
23 ]
```

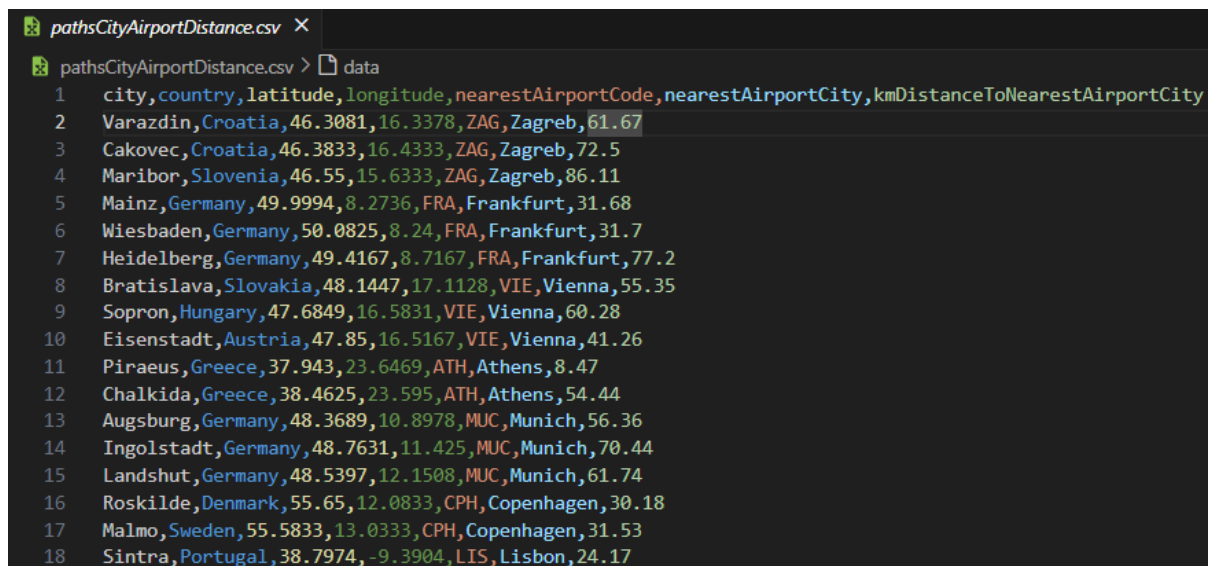
Slika 3: Lista rječnika u Pythonu sa popisom gradova i njima najbližih aerodroma

Zatim, pomoću dodatne Python skripte dobivena je datoteka nodesCities.csv s donje Slike 4 koja uz sve informacije iz liste Cities sa Slike 3 sadrži i broj stanovnika grada iz datoteke worldcities.csv.

```
nodesCities.csv
nodesCities.csv > data
1  name,population,country,nearestAirportCode,nearestAirportCity,nearestAirportCountry
2  Varazdin,43782,Croatia,ZAG,Zagreb,Croatia
3  Cakovec,27122,Croatia,ZAG,Zagreb,Croatia
4  Maribor,96211,Slovenia,ZAG,Zagreb,Croatia
5  Mainz,222889,Germany,FRA,Frankfurt,Germany
6  Wiesbaden,285522,Germany,FRA,Frankfurt,Germany
7  Heidelberg,162960,Germany,FRA,Frankfurt,Germany
8  Bratislava,475503,Slovakia,VIE,Vienna,Austria
9  Sopron,98479,Hungary,VIE,Vienna,Austria
10 Eisenstadt,14476,Austria,VIE,Vienna,Austria
```

Slika 4: Datoteka nodesCities.csv sa informacijama o gradovima

Na kraju, za svaki grad izračunata je zračna udaljenost do grada njegovog najbližeg aerodroma primjenom Haversine formule koja računa najkraći put između dvije točke na sferi koristeći njihove geografske koordinate. Rezultati izračuna udaljenosti pohranjeni su u zasebnu CSV datoteku `pathsCityAirportDistance.csv` čiji je dio vidljiv na Slici 5.



```
pathsCityAirportDistance.csv > data
1  city,country,latitude,longitude,nearestAirportCode,nearestAirportCity,kmDistanceToNearestAirportCity
2  Varazdin,Croatia,46.3081,16.3378,ZAG,Zagreb,61.67
3  Cakovec,Croatia,46.3833,16.4333,ZAG,Zagreb,72.5
4  Maribor,Slovenia,46.55,15.6333,ZAG,Zagreb,86.11
5  Mainz,Germany,49.9994,8.2736,FRA,Frankfurt,31.68
6  Wiesbaden,Germany,50.0825,8.24,FRA,Frankfurt,31.7
7  Heidelberg,Germany,49.4167,8.7167,FRA,Frankfurt,77.2
8  Bratislava,Slovakia,48.1447,17.1128,VIE,Vienna,55.35
9  Sopron,Hungary,47.6849,16.5831,VIE,Vienna,60.28
10 Eisenstadt,Austria,47.85,16.5167,VIE,Vienna,41.26
11 Piraeus,Greece,37.943,23.6469,ATH,Athens,8.47
12 Chalkida,Greece,38.4625,23.595,ATH,Athens,54.44
13 Augsburg,Germany,48.3689,10.8978,MUC,Munich,56.36
14 Ingolstadt,Germany,48.7631,11.425,MUC,Munich,70.44
15 Landshut,Germany,48.5397,12.1508,MUC,Munich,61.74
16 Roskilde,Denmark,55.65,12.0833,CPH,Copenhagen,30.18
17 Malmo,Sweden,55.5833,13.0333,CPH,Copenhagen,31.53
18 Sintra,Portugal,38.7974,-9.3904,LIS,Lisbon,24.17
```

Slika 5: Datoteka koja sadrži informacije o zračnoj udaljenosti grada i grada njemu najbližeg aerodroma

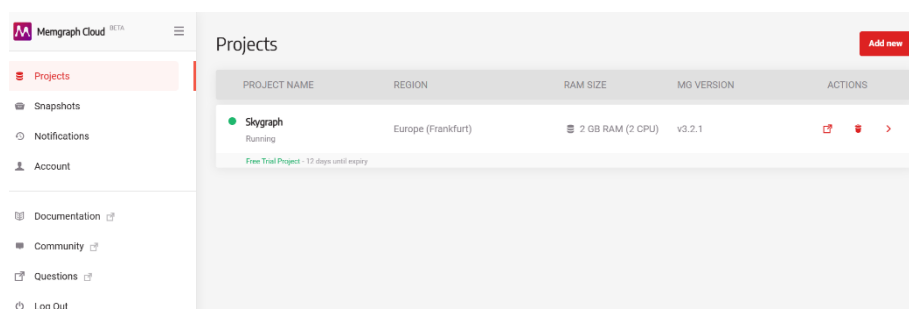
Sljedeći korak je ubacivanje svih podataka iz kreiranih datoteka u grafovsku bazu podataka u Memgraphu.

3. POSTAVLJANJE BAZE U MEMGRAPH

Za pohranu i upravljanje podacima o letovima i gradovima odabrana je grafovska baza podataka Memgraph. Osim što je to hrvatski proizvod, glavni razlog odabira Memgrapha bila je njegova in-memory arhitektura koja odgovara prirodi Skygraph projekta koji zahtijeva brzo pretraživanje velikog broja letova i izračun optimalnih puteva u realnom vremenu. Prema službenoj dokumentaciji (<https://memgraph.com/blog/neo4j-vs-memgraph>), ključna razlika između Memgrapha i drugih grafovskih baza podataka poput Neo4j-a leži u pristupu pohrani podataka. Dok Neo4j koristi on-disk pohranu koja je prikladna za velike količine podataka kojima se ne pristupa često, Memgraph je implementiran kao potpuno in-memory rješenje fokusirano na procesiranje streaming podataka i izračune u realnom vremenu. Ova arhitektura omogućuje brzu analizu velikih grafova i daje odgovore u najkraćem mogućem vremenu bez problema s performansama, što je idealno za sustav pretraživanja letova.

3.1. Instalacija Memgrapha

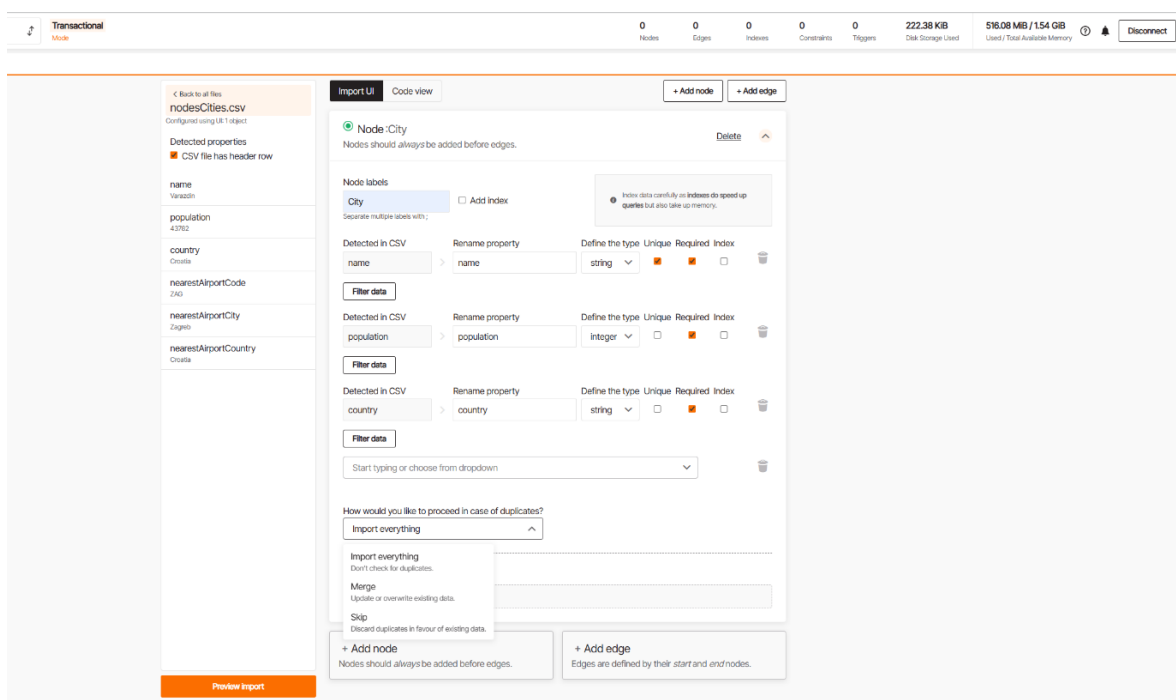
Memgraph nudi tri osnovna pristupa instalaciji i korištenju. Prvi način je lokalna instalacija kojom se Memgraph instalira izravno na računalo korisnika korištenjem Docker kontejnera (za Windows, macOS i Linux), putem Linux paketa, kroz WSL za Windows korisnike ili kompajliranjem izvornog koda na svom računalu. Važno je napomenuti da za lokalnu instalaciju računalo mora zadovoljavati hardverske i softverske preduvjete koji su detaljno navedeni u službenoj dokumentaciji (<https://memgraph.com/docs/getting-started/install-memgraph>), poput minimalnih zahtjeva na RAM memoriju i procesor, posebno ako se radi s velikom grafovskom bazom. Druga opcija je registracija i pokretanje Memgraph Cloud instance s 14-dnevnom probnom verzijom. U ovom slučaju korisnik dobiva pristup i Memgraph Cloud-u (usluga u oblaku koja sadrži samu grafovsku bazu podataka) i Memgraph Lab-u (web sučelju za komunikaciju s bazom) direktno u web pregledniku bez bilo kakve instalacije. Treći je hibridni pristup koji kombinira prednosti lokalnog i cloud pristupa. Korisnik može instalirati Memgraph Lab lokalno kao desktop aplikaciju, a istovremeno ga povezati s Memgraph instancom u oblaku (Memgraph Cloud) ili na drugom računalu. Ovakav pristup omogućuje lokalno optimiziranu vizualizaciju podataka uz fleksibilnost oblak infrastrukture. Za potrebe ovog projekta odlučila sam se za pristup Memgraph Cloudu i Memgraph Lab-u u web pregledniku, odnosno za 14-dnevnu besplatnu probnu verziju.



Slika 6: Besplatna probna instanca u Memgraph Cloud-u

3. 2. Kreiranje vrhova i bridova

Kada se u web sučelju Memgraph Lab odabere opcija u glavnom izborniku Import, predlažu se 3 osnovne opcije za uvoz podataka. Prva je CSV file import – unos podataka iz CSV datoteka, druga je Connect Kafka or Pulsar stream – povezivanje sa streaming izvorima podataka, a treća CYPHERL file import – unos podataka kroz Cypher upite u datoteci. Za unos vrhova tipa :Airport, :City i bridova tipa :NEAREST_AIRPORT odabrala sam prvu opciju CSV file import. Memgraph Lab nudi intuitivan Import UI u kojem se jednostavno definiraju elementi grafovske strukture. Na Slici 7 vidimo kreiranje vrha tipa :City iz nodesCities.csv. Potrebno je definirati ime čvora (City). Zatim se dodaju atributi tako da se povežu sa stupcima iz CSV-a ("Detected in CSV") i daju im nazivi koji će se koristiti u bazi ("Rename property"). Za svaki atribut potrebno je odabrati tip podatka (string, integer, float, itd.), je li atribut obavezan (Required), je li vrijednost jedinstvena (Unique) te hoće li se na atributu postaviti indeks (Index) za brže pretraživanje. Na donjoj slici kreiran je vrh tipa :City s atributima name, population i country koji su povezani s istoimenim stupcima iz CSV-a. Atributi name i country su tipa string, a population integer, te su svi označeni kao Required, dok je name dodatno postavljen kao Unique. Dodatno, potrebno je odabrati kako se ponašati u slučaju duplikata. Ponuđene su 3 opcije Import everything – nema provjere duplikata, Merge – ažurira ili zamijeni postojeće podatke te Skip – odbaci duplikate i zadrži postojeće podatke. Budući da sam pripremom podataka osigurala da ne postoje duplikati odabrala sam 1. opciju koja je najbrža.



Slika 7: Import UI za unos vrhova tipa :City

Također, ovaj UI automatski generira Cypher kod za uvoz podataka koji je vidljiv odabirom opcije „Code view“ na Slici 8.

The screenshot shows the Transactional UI interface. On the left, a sidebar lists detected properties for 'nodesCities.csv': name (Varaždin), population (43782), country (Croatia), nearestAirportCode (ZAG), nearestAirportCity (Zagreb), and nearestAirportCountry (Croatia). The main area is titled 'Import UI' and 'Code view'. It displays a Cypher query with the following structure:

```
1 // #1 Setup (in case of an edit, keep these comments)
2
3
4 // #2 Create constraints (in case of an edit, keep these comments)
5 CREATE CONSTRAINT ON (n:city) ASSERT EXISTS (n.name);
6 CREATE CONSTRAINT ON (n:city) ASSERT n.name IS UNIQUE;
7 CREATE CONSTRAINT ON (n:city) ASSERT EXISTS (n.population);
8 CREATE CONSTRAINT ON (n:city) ASSERT EXISTS (n.country);
9
10 // #3 Create indexes (in case of an edit, keep these comments)
11
12
13
14 // #4 Import nodes (in case of an edit, keep these comments)
15 LOAD CSV FROM $file WITH HEADER IGNORE BAD AS row
16 CREATE (:city) {
17   'name': CASE row.name WHEN '' THEN null ELSE row.name END,
18   'population': toInteger(row.population),
19   'country': CASE row.country WHEN '' THEN null ELSE row.country END
20 };
21
22
23
24 // #5 Import relationships (in case of an edit, keep these comments)
25
26
27 // #6 Cleanup (in case of an edit, keep these comments)
28
29
```

Below the code, a warning message states: "Changes made in the Import UI are visible in the Code view, but changes to the code will discard the configuration made in the Import UI. By editing the code, you are committing to write the complete import code yourself."

Slika 8: Generiranje kod za kreiranje vrha tipa :City

Analogno sam kreirala vrhove tipa :Airport iz datoteke nodesAirports.csv, atributi ovog vrha su code – IATA kod aerodroma, city – grad u kojem se aerodrom nalazi i country – država u kojoj se nalazi. Svi atributa su tipa string i Required, a code je dodatno i Unique.

The screenshot shows the Transactional UI interface for the 'pathsCityAirportDistance.csv' import. On the left, a sidebar lists detected properties: city (Varaždin), country (Croatia), latitude (45.7639), longitude (16.2701), nearestAirportCode (ZAG), nearestAirportCity (Zagreb), and kmDistanceToNearestAirportCity (61.67). The main area is titled 'Import UI' and 'Code view'. It displays a configuration panel for the 'NEAREST_AIRPORT' edge type. The panel includes fields for 'Start node' (City) and 'End node' (Airport), both with 'name' and 'code' properties. The 'Edge properties' section shows 'kmDistanceToNearestAirportCity' as a 'float' property. The 'Filter data' section is empty. The 'How would you like to proceed in case of duplicates?' dropdown is set to 'Import everything'.

Slika 9: Kreiranje brida :NEAREST_AIRPORT u Import UI

Na Slici 9 vidimo proces kreiranja brida tipa `:NEAREST_AIRPORT` iz datoteke `pathCityAirportDistance.csv`. Za kreiranje ovog brida u Import UI-u potrebno je prvo odabrati `Start node`, u ovom slučaju tipa `:City`. Za taj početni čvor definiramo uvjet povezivanja – postojeći vrh u bazi tipa `:City` povezuje se sa zapisom u CSV datoteci kada je vrijednost atributa `"name"` jednaka vrijednosti u stupcu `"city"` CSV datoteke (`"Select property from .csv"`). Nakon toga, potrebno je odabrati `End node`, u ovom slučaju tipa `:Airport`. Ovdje definiramo uvjet da se postojeći vrhovi tipa `:Airport` povezuju sa zapisima u CSV datoteci kada je vrijednost atributa `"code"` vrha jednaka vrijednosti jednaka vrijednosti u stupcu `"nearestAirportCode"` u CSV datoteci. Novokreiranom bridu `:NEAREST_AIRPORT` dodajemo atribut `"kmDistanceToNearestAirport"` tipa `float` koji predstavlja zračnu udaljenost između grada i grada kojem pripada njegov najbliži aerodrom. Vrijednost ovog atributa preuzima se iz istoimenog stupca u CSV datoteci. Na donjoj Slici 10 vidljiv je Cypher kod koji je Import UI izgenerirao iz odabranih opcija.

The screenshot shows the Memgraph Import UI interface. On the left, under 'Detected properties', a list of CSV columns is shown: `city`, `country`, `latitude`, `longitude`, `nearestAirportCode`, `nearestAirportCity`, and `kmDistanceToNearestAirportCity`. The 'Code view' tab on the right displays the following Cypher code:

```

1 // #1 Setup (in case of an edit, keep these comments)
2
3
4 // #2 Create constraints (in case of an edit, keep these comments)
5
6
7 // #3 Create indexes (in case of an edit, keep these comments)
8
9
10 // #4 Import nodes (in case of an edit, keep these comments)
11
12
13 // #5 Import relationships (in case of an edit, keep these comments)
14 LOAD CSV FROM $file WITH HEADERS IGNORE 45 row
15 MATCH (n:city) {
16   name : GGC row.'city' WHEN "" THEN null ELSE row.'city' END
17 }
18 MATCH (n:airport) {
19   code : GGC row.'nearestAirportCode' WHEN "" THEN null ELSE row.'nearestAirportCode' END
20 }
21 CREATE (n)-[:NEAREST_AIRPORT] {
22   kmDistanceToNearestAirportCity : tofloat(row.'kmDistanceToNearestAirportCity')
23 }->(n)
24
25 // #6 Cleanup (in case of an edit, keep these comments)
26
27
28
29

```

At the bottom, a message states: "Changes made in the Import UI are visible in the Code view, but changes to the code will discard the configuration made in the Import UI. By editing the code, you are committing to write the complete import code yourself."

Slika 10: Cypher kod za kreiranje brida `:NEAREST_AIRPORT`

Na pretpregledu unosa (Preview import) za svaku csv datoteku vidljivi su tipovi bridova i vrhova sa njihovim pripadnim atributima koje smo iz njihovih podataka kreirali te jednostavnim klikom na Start Import ubacujemo napravljeno u grafovsku bazu podataka.

Bridove tipa `:FLIGHT_TO` odlučila sam dodati pisanjem custom Cypher upita sa Slike 11. Datoteku `pathsFlights.csv` spremila sam na Google disk, zatim preuzela te koristila link za preuzimanje za pristup njenim podacima. Donji upit nakon učitavanja datoteke, sprema kodove polazišnog i odredišnog aerodroma kao zasebne varijable `depCode` i `destCode`. Sa uvjetom nakon `WHERE` filtriraju se samo redovi s ispravnim cijenama, izostavljajući one s nedostajućim podacima ili null vrijednostima. Zatim pronalazi postojeće vrhove tipa `:Airport` koji odgovoraju kodovima iz CSV datoteke. Na kraju, kreira brid između ta dva postojeća vrha s pripadajućim atributima; `company` – aviokompanija koja obavlja let, `departureTimestamp` – vrijeme polaska u timestamp obliku, `arrivalTimestamp` – vrijeme slijetanja u timestamp formatu, `durationMinutes` – trajanje leta u minutama te `priceEur` – cijena karte za let u eurima. Pri tome koristi funkcije za konverziju tipova podataka; `localDateTime()` koja vraća tip podatka koji predstavlja datum i lokalno vrijeme, `toInteger()` za trajanje leta u minutama i `toFloat()` za cijenu karte, budući da su u CSV datoteci svi podaci pohranjeni kao stringovi.

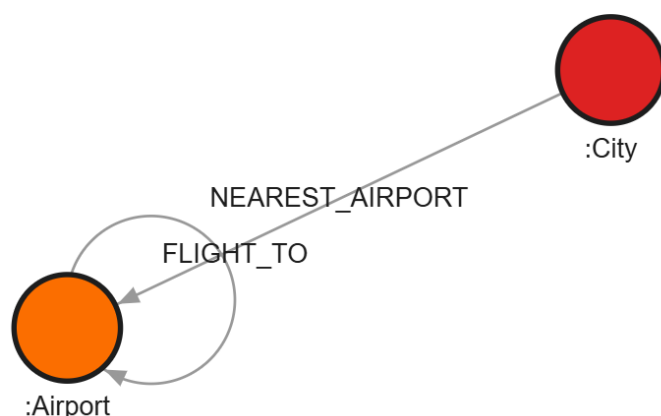
```

Cypher editor | Graph Style editor | Parameters
1 // Edge :FLIGHT_TO
2
3 LOAD CSV FROM "https://drive.usercontent.google.com/..." WITH HEADER IGNORE BAD AS row
4 WITH
5     row,
6     row.departureAirport AS depCode,
7     row.destinationAirport AS destCode
8 WHERE row.price IS NOT NULL AND row.price <> "Price unavailable"
9 MATCH (n:Airport { code: depCode })
10 MATCH (m:Airport { code: destCode })
11 CREATE (n)-[:FLIGHT_TO {
12     company: row.company,
13     departureTimestamp: localDateTime(row.departureTimestamp),
14     arrivalTimestamp: localDateTime(row.arrivalTimestamp),
15     durationMinutes: toInteger(row.duration),
16     priceEur: toFloat(row.price)
17 }]->(m);

```

Slika 11: Cypher query za unos bridova tipa `:FLIGHT_TO`

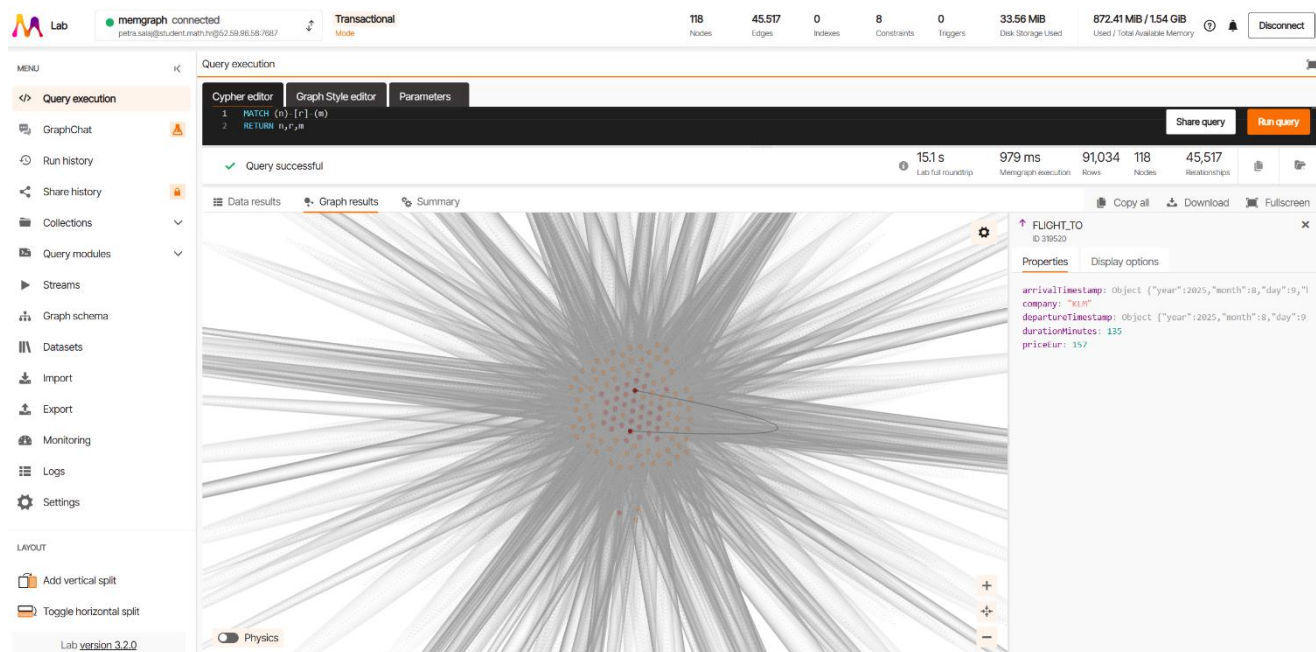
Sada kada su uneseni svi vrhovi i bridovi, moguće je generirati shemu grafovske baze Skygraph prikazanu na Slici 12. Iako se za nerelacijske baze pa tako i grafovske često govori da su bez sheme (schema-less), one zapravo koriste fleksibilnu shemu koja se formira kroz tipove vrhova i bridova.



Slika 12: Shema grafovske baze Skygraph

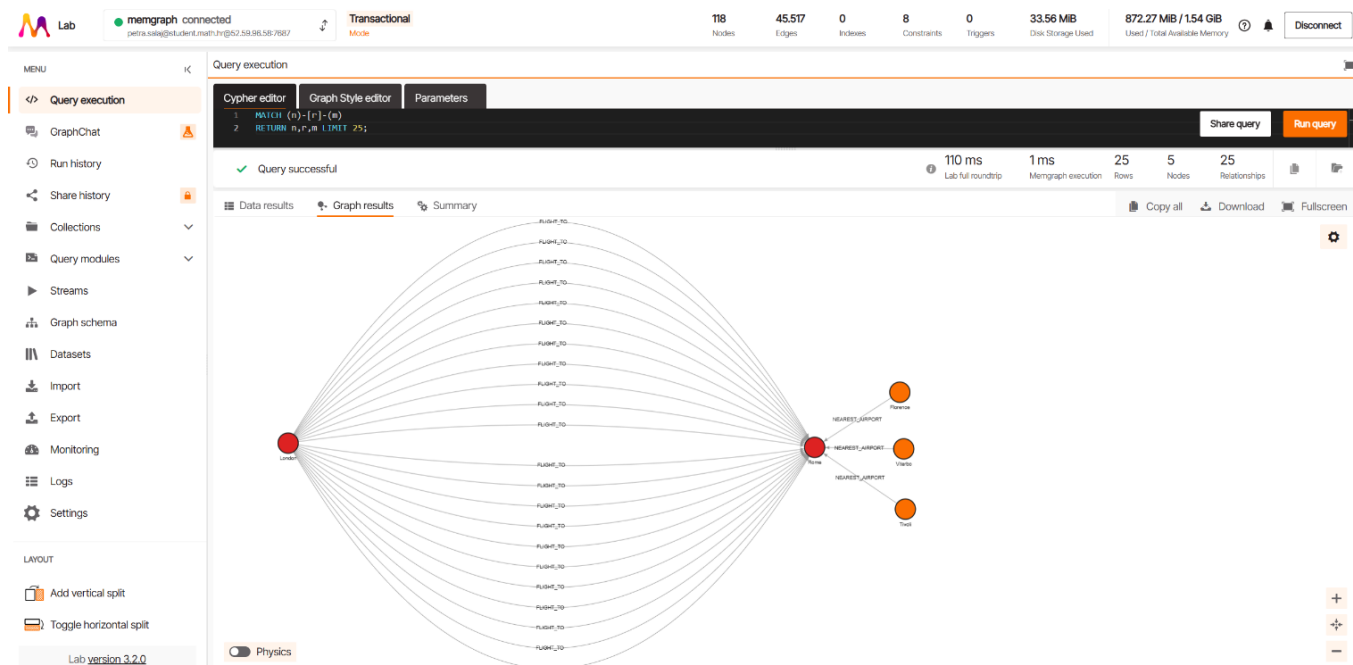
4. UPITI NA BAZU

4.1. Prikaz podataka baze



Slika 13: Cijeli graf

Na Slici 13 prikazani su rezultati prvog upita koji izgleda poput zvijezde jer sadrži svih 118 vrhova i svih 45 517 bridova u bazi podataka, što stvara vrlo gustu vizualizaciju.



Slika 14: dio grafa

Na Slici 14 prikazani su rezultati drugog upita koji zbog ograničenja pokazuje samo dio grafa. Vidljivi su letovi iz Londona u Rim te gradovi Firenca, Viterbo i Tivoli kojima je aerodrom u Rimu najbliži.

4.2. Lista svih gradova (s ili bez vlastitog aerodroma)

The screenshot shows a Cypher query editor interface. The query is as follows:

```
1 MATCH (n)
2 RETURN CASE
3   WHEN n:City THEN n.name
4   WHEN n:Airport THEN n.city
5   ELSE null
6 END AS result;
```

The query was successful. The results table shows 118 rows, 0 nodes, and 0 relationships. The results are as follows:

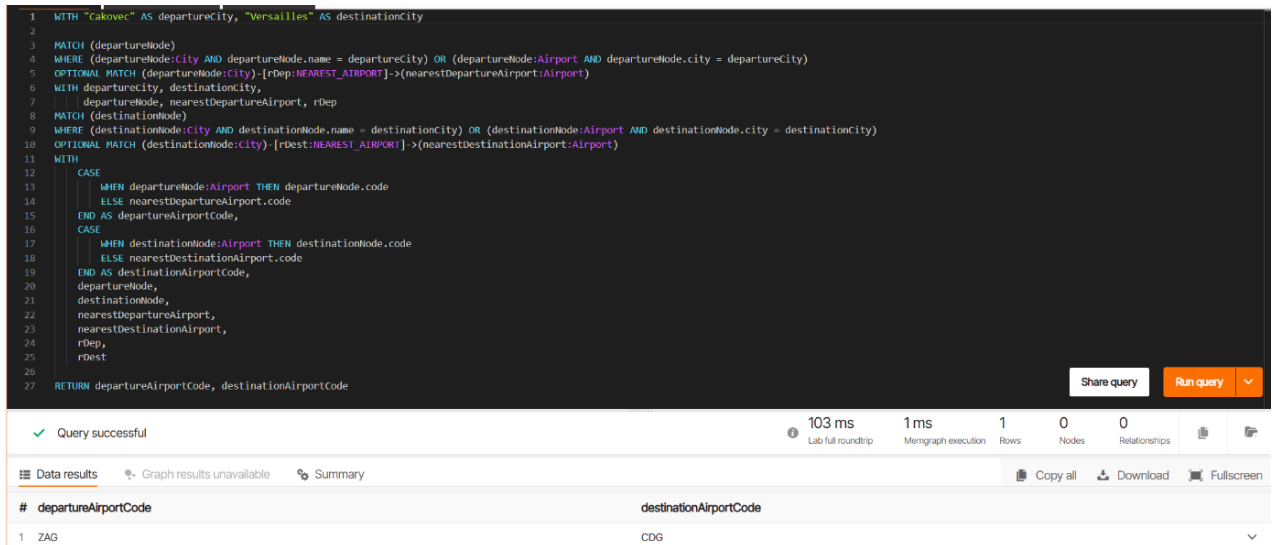
#	result
1	Rome
2	London
3	Amsterdam
4	Zagreb
5	Frankfurt
6	Vienna
7	Athens
8	Munich
9	Copenhagen

Slika 15: Upit za ispis liste svih gradova

Svrha upita je ispis svih gradova u bazi podataka tako što se izdvajaju imena gradova iz čvorova tipa :City i čvorova tipa :Airport. Upit pretražuje sve čvorove u bazi podataka bez filtriranja. Koristi CASE uvjet kako bi izdvojio imena gradova ovisno o tipu čvora. Kad je riječ o čvoru tipa :City, upit izdvaja vrijednost iz svojstva name. Kod čvora tipa :Airport, upit izdvaja vrijednost iz svojstva city. Upit ignorira sve ostale tipove čvorova vraćajući za njih NULL vrijednost. U kontekstu aplikacije, ovaj upit osigurava početni popis gradova koje korisnici mogu odabrati kao polazište ili odredište u sučelju aplikacije za pretraživanje letova.

Na Slici 15 vidimo da je u ovoj bazi podataka spremljeno 118 gradova.

4.3. Pronalazak najbližeg aerodroma za gradove polaska i odredišta



```
1 WITH "Čakovec" AS departureCity, "Versailles" AS destinationCity
2
3 MATCH (departureNode)
4 WHERE (departureNode:City AND departureNode.name = departureCity) OR (departureNode:Airport AND departureNode.city = departureCity)
5 OPTIONAL MATCH (departureNode:City)-[rDep:NEAREST_AIRPORT]->(nearestDepartureAirport:Airport)
6 WITH departureCity, destinationCity,
7      departureNode, nearestDepartureAirport, rDep
8 MATCH (destinationNode)
9 WHERE (destinationNode:City AND destinationNode.name = destinationCity) OR (destinationNode:Airport AND destinationNode.city = destinationCity)
10 OPTIONAL MATCH (destinationNode:City)-[rDest:NEAREST_AIRPORT]->(nearestDestinationAirport:Airport)
11 WITH
12     CASE
13         WHEN departureNode:Airport THEN departureNode.code
14         ELSE nearestDepartureAirport.code
15     END AS departureAirportCode,
16     CASE
17         WHEN destinationNode:Airport THEN destinationNode.code
18         ELSE nearestDestinationAirport.code
19     END AS destinationAirportCode,
20     departureNode,
21     destinationNode,
22     nearestDepartureAirport,
23     nearestDestinationAirport,
24     rDep,
25     rDest
26
27 RETURN departureAirportCode, destinationAirportCode
```

Query successful

103 ms Lab full roundtrip 1 ms Memgraph execution 1 Rows 0 Nodes 0 Relationships

Data results Graph results unavailable Summary Copy all Download Fullscreen

#	departureAirportCode	destinationAirportCode
1	ZAG	CDG

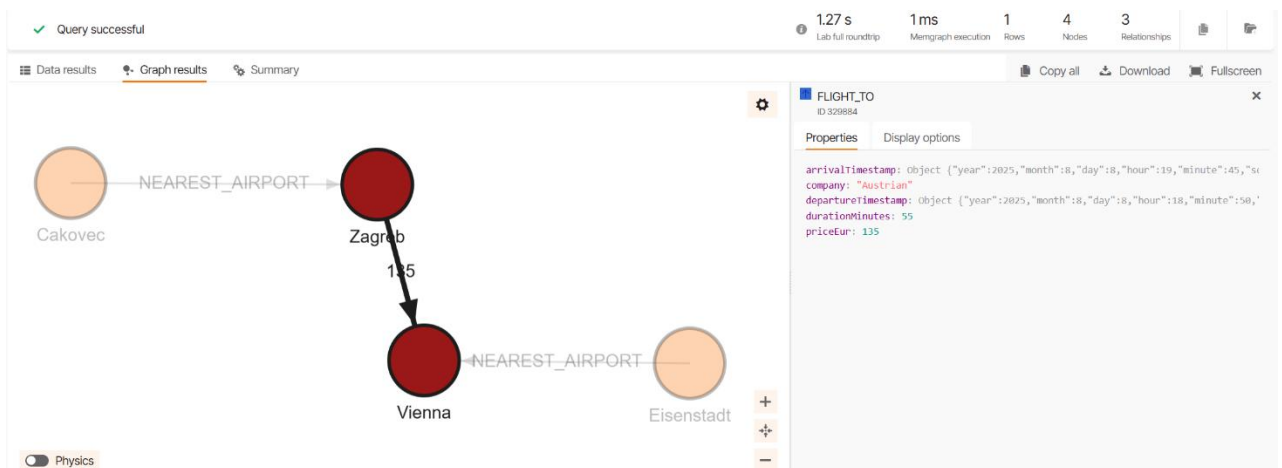
Slika 16: Upit za pronalaženje kodova zračnih luka za gradove polaska i odredišta

Svrha upita je pronalaženje kodova zračnih luka za polazišne i odredišne gradove, uzimajući u obzir slučajeve kada gradovi imaju vlastite zračne luke ili koriste najbližu zračnu luku. Upit započinje s imenima gradova ("Čakovec" i "Versailles") koje bi korisnik unio. Za svaki grad upit traži ili :City čvor s odgovarajućim imenom ili :Airport čvor gdje svojstvo city odgovara traženom gradu. Ako pronađe City čvor, upit slijedi vezu NEAREST_AIRPORT do najbliže zračne luke. Koristeći CASE uvjete, upit određuje konačne kodove zračnih luka - ako je čvor tipa :Airport, koristi njegov kod direktno, inače koristi kod najbliže zračne luke. U kontekstu aplikacije, ovaj upit prevodi korisnički odabrane gradove u kodove zračnih luka potrebne za pretraživanje letova, rješavajući uobičajeni scenarij kada korisnici putuju iz/u gradove bez vlastite zračne luke. Sa Slike 16 vidimo da je za navedene gradove Čakovec i Versailles, rezultat ZAG (Zračna luka Franjo Tuđman u Zagrebu) i CDG (Zračna luka Charles de Gaulle u Parizu).

4.4. Najjeftiniji direktni let

```
Cypher editor | Graph Style editor | Parameters
1 WITH "Čakovec" AS departureCity, "Eisenstadt" AS destinationCity
2
3 MATCH (departureNode)
4 WHERE (departureNode:City AND departureNode.name = departureCity) OR (departureNode:Airport AND departureNode.city = departureCity)
5 OPTIONAL MATCH (departureNode:City)-[rDep:NEAREST_AIRPORT]->(nearestDepartureAirport:Airport)
6 WITH departureCity, destinationCity,
7  departureNode, nearestDepartureAirport, rDep
8 MATCH (destinationNode)
9 WHERE (destinationNode:City AND destinationNode.name = destinationCity) OR (destinationNode:Airport AND destinationNode.city = destinationCity)
10 OPTIONAL MATCH (destinationNode:City)-[rDest:NEAREST_AIRPORT]->(nearestDestinationAirport:Airport)
11 WITH
12   CASE
13     WHEN departureNode:Airport THEN departureNode.code
14     ELSE nearestDepartureAirport.code
15   END AS departureAirportCode,
16   CASE
17     WHEN destinationNode:Airport THEN destinationNode.code
18     ELSE nearestDestinationAirport.code
19   END AS destinationAirportCode,
20   departureNode,
21   destinationNode,
22   nearestDepartureAirport,
23   nearestDestinationAirport,
24   rDep,
25   rDest
26
27 // Use the airport codes from the first part - CHEAPEST DIRECT FLIGHT
28 MATCH path=(n {code: departureAirportCode})-[:FLIGHT_TO]->(m {code: destinationAirportCode})
29 RETURN
30   departureNode,
31   CASE WHEN rDep IS NOT NULL THEN [departureNode, rDep, nearestDepartureAirport] ELSE [] END AS departureToAirportPath,
32   destinationNode,
33   CASE WHEN rDest IS NOT NULL THEN [destinationNode, rDest, nearestDestinationAirport] ELSE [] END AS destinationToAirportPath,
34   nodes(path) AS directFlightNodes,
35   relationships(path) AS directFlightRelationship,
36   r.durationMinutes AS flightDuration,
37   r.priceEur AS flightPrice,
38   r.departureTimestamp AS departureTime,
39   r.arrivalTimestamp AS arrivalTime
40 ORDER BY r.priceEur ASC
41 LIMIT 1;
```

Slika 17: Cypher kod upita za najjeftiniji direktni let

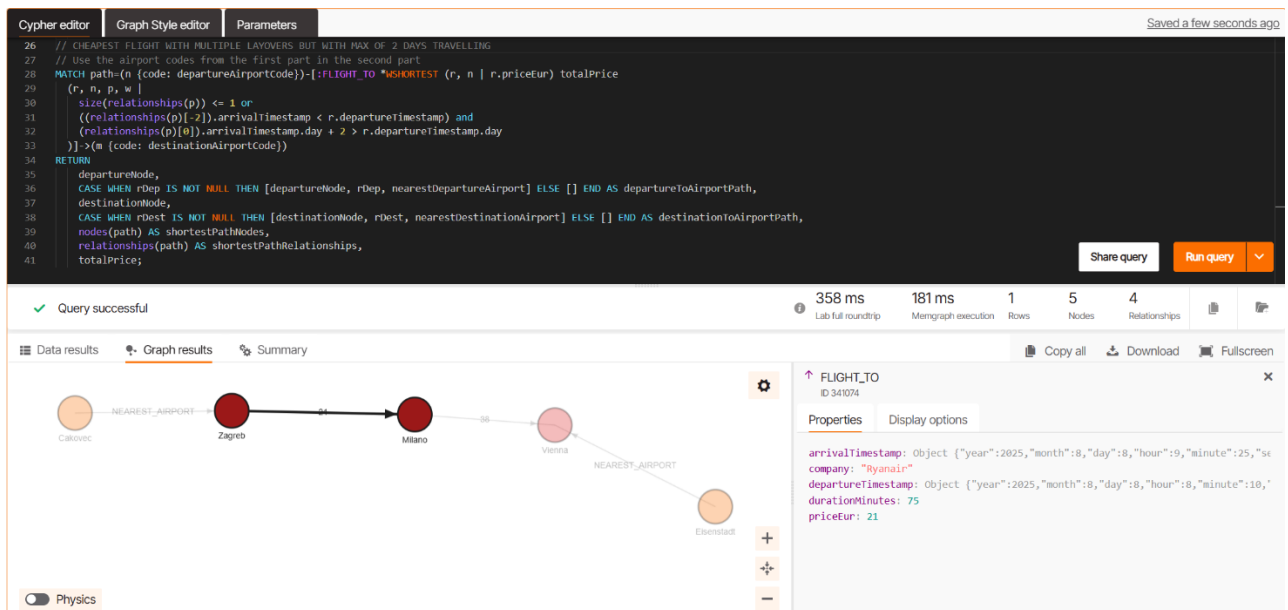


Slika 18: Rezultat upita za najjeftiniji direktni let

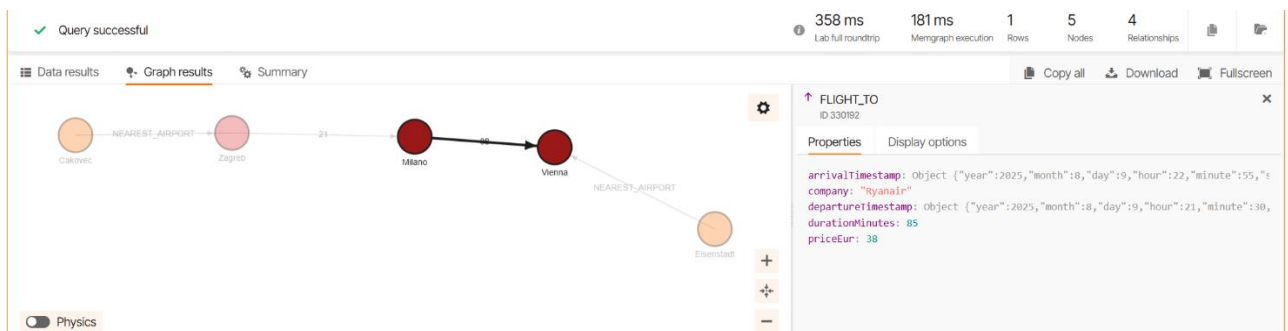
Svrha ovog upita je pronalaženje najjeftinijeg izravnog leta između dva grada (Čakovec i Eisenstadt), uključujući detalje o povezivanju za gradove koji nemaju vlastite zračne luke. Upit prvo koristi podupit za pronalaženje odgovarajućih kodova zračnih luka za oba grada, a zatim pronalazi izravni let između njih. Vraća sveobuhvatne informacije o polaznim i odredišnim čvorovima, putevima do najbližih zračnih luka, te detaljima leta (vremenu polijetanja, slijetanja, cijeni, trajanju i aviokompaniji). U kontekstu aplikacije, ovaj upit implementira uobičajeni scenarij pretraživanja letova s filtrom "bez presjedanja" i sortiranjem po cijeni.

Rezultat za zadane gradove je let od Zagreba do Beča 8.8.2025. s vremenom polaska u 18:50 po cijeni od 135 eura.

4.5. Najjeftiniji let s mogućim presjedanjima i najviše 2 dana ukupnog putovanja



Slika 19: Cypher upit i rezultat za najjeftiniji let s mogućim presjedanjima i max 2 dana putovanja

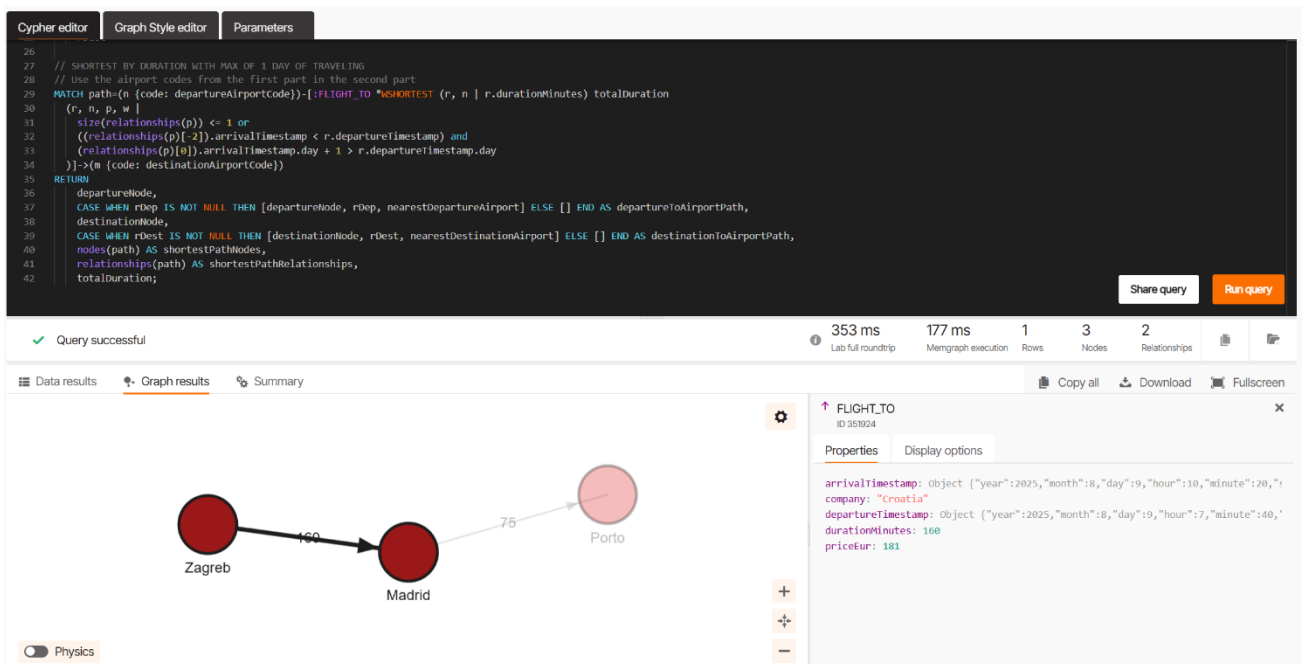


Slika 20: Detalji rezultata upita za najjeftiniji let s mogućim presjedanjima i max 2 dana putovanja

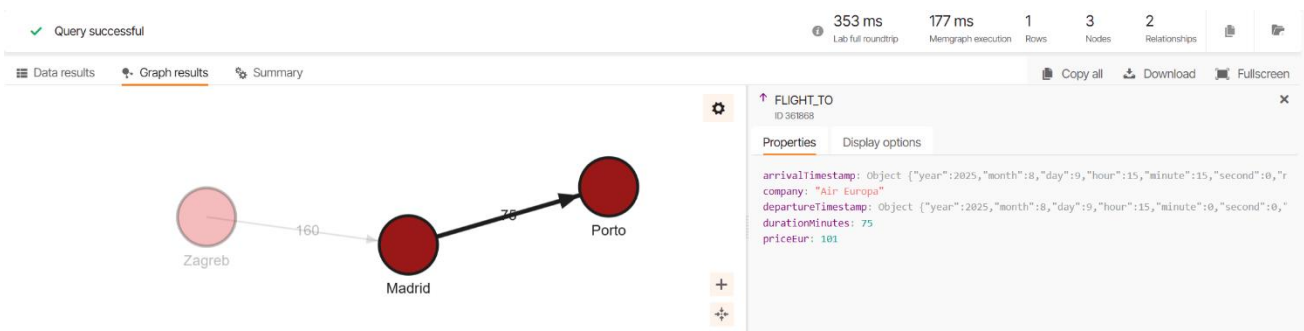
Svrha ovog upita je pronalaženje najjeftinijeg leta između dva grada (Čakovec i Eisenstadt), omogućavajući neograničen broj presjedanja uz uvjet da ukupno vrijeme putovanja ne prelazi 2 dana. Upit koristi podupit za pronalaženje odgovarajućih kodova zračnih luka te implementira algoritam za težinski najkraći put (težine su cijene u eurima) kako bi pronašao najjeftiniju rutu. Primjenjuje ograničenja koja osiguravaju realistične veze: dolazak prethodnog leta mora biti prije polaska sljedećeg, a ukupno vrijeme putovanja ne smije prelaziti 2 dana. U kontekstu aplikacije, ovo je fleksibilnije pretraživanje letova za korisnike koji prioritziraju cijenu nad udobnošću putovanja.

Rezultat upita za zadane gradove je ruta s jednim presjedanjem u Milano: polazak iz Zagreba 8.8. u 8:10, dolazak u Milano u 9:25, te polazak iz Milana za Beč idući dan (9.8.) u 9:21. Ukupna cijena svih letova je 59 eura, što je više od duplo manje od cijene direktnog leta.

4.5. Vremenski najkraći let s mogućim presjedanjima i najviše 1 danom putovanja



Slika 21: Cypher upit za pronalaženje vremenski najkraćeg leta s max putovanjem od 1 dana



Slika 22 : Rezultat upita za pronalaženje vremenski najkraćeg leta s max putovanjem od 1 dana

Svrha ovog upita je pronalaženje najbrže rute između Zagreba i Porta minimiziranjem ukupnog trajanja leta, dopuštajući presjedanja uz osiguravanje da vrijeme putovanja ne prelazi 1 dan. Upit koristi podupit za pronalaženje odgovarajućih kodova zračnih luka, te implementira algoritam za težinski najkraći put (težina je trajanje leta u minutama) kako bi pronašao rutu s minimalnim ukupnim trajanjem leta. U kontekstu aplikacije, ovaj upit je optimiziran za putnike koji prioritiziraju minimiziranje vremena putovanja uz strogo ograničenje jednodnevnog putovanja, što je posebno korisno za poslovna putovanja.

U ovom konkretnom slučaju, kada ne postoji direktni let iz Zagreba u Porto, rezultat upita pokazuje rutu koja uključuje let iz Zagreba u Madrid 9.8. u 7:40, te zatim let iz Madrida za Porto istog dana u 15:00, s ukupnim trajanjem leta od 235 minuta odnosno 3 sata i 55 minuta.

5. ZAKLJUČAK

U ovom radu uspješno je demonstrirana implementacija ključnih komponenti za aplikaciju nalik Google Flights-u, korištenjem Memgraph grafovske baze podataka i Cypher upita. Razvijeni sustav omogućuje korisničko iskustvo koje prati prirodni tijek korištenja aplikacije za pretraživanje letova - od početnog prikaza svih dostupnih gradova za polazak i odredište, do naprednog pretraživanja optimalnih ruta prema odabranim kriterijima (cijeni, trajanju leta, ukupnom trajanju putovanja).

Implementacija pokazuje prednosti grafovskih baza podataka nad tradicionalnim relacijskim bazama kada je riječ o modeliranju kompleksnih međusobno povezanih podataka i izvođenju algoritama za pretraživanje optimalnih puteva. Memgraph se pokazao kao moćan alat koji omogućava intuitivno modeliranje i efikasno pretraživanje podataka o letovima, slično načinu na koji funkcioniraju komercijalne aplikacije.

U budućim verzijama, sustav bi se mogao proširiti detaljnim informacijama o povezanosti aerodroma s drugim oblicima prijevoza (vlak, autobus, metro...), čime bi aplikacija postala sveobuhvatniji alat za planiranje putovanja.

6. LITERATURA

<https://github.com/AWeirdDev/flights>

<https://simplemaps.com/data/world-cities>

<https://memgraph.com/docs>