



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Filippo Mininno 10667429**

Group Number: **Gruppo Singolo**

Academic Year: 2023-2024

Contents

Contents	i
1 Introduction	1
2 Dataset	3
2.1 Field description	3
2.2 Neo4j implementation	6
2.2.1 Schema	6
2.2.2 Creation queries	7
3 Queries	11
3.1 Highest Airport	11
3.2 Countries with higher number of operating airlines	11
3.3 Top 5 Airports by number of destinations	12
3.4 Biggest airlines by countries served	12
3.5 Biggest airlines by destination offered	13
3.6 Most competed routes	13
3.7 Most flight-dependant countries	14
3.8 Under-served routes	14
3.9 Shortest Path	15
3.10 Shortest Path with condition	16
4 Extra	17
List of Figures	19

1 | Introduction

The objective of this project is to realize a chat-based travel assistant, able to give information about scheduled flights and their routes. The chat assistant is based on openAI technologies and the data fed to the assistant is stored in a graph database. The technology chosen for the graph DB is Neo4j.

Neo4j has been chosen because, thanks to its native functionalities, it makes easy to obtain complex information, like a shortest path between two airports. Another reason on why to use a graph database is that this technology is better suited to handle the relations between data and the proposed dataset is rich of relations in this sense (A flight route is operated by an airline which is based in a country, the flight departs from Airport A, in country AC and arrives in Airport B, in country BC...)

2 | Dataset

The chosen dataset can be found at <https://openflights.org/data.php>. Openflight is an (old) opensource project whos objective was to build a tool that let user map, search and calculate statistics about flights operated by different airlines. With time they also gathered data about airlines and their operated routes. The dataset is composed of 3 different csv files:

- *countries.dat* that contains data about countries.
- *airports-extended.dat* that contains data about airports, train stations and ferry terminals.
- *routes.dat* that contains data about flight routes.

My idea was to use flights routes data to identify the optimal itinerary between two given cities. Unfortunately the routes.dat does not contain any information about flight prices and travel times, so the best itinerary could be computed only as the one that involves the less amount of stops. This itinerary has been computed using Neo4j function *shortesPath()*.

Using the other data available it was also possible to obtain interesting information about the aviation market.

The same opensource project offers also a dataset of flight equipment (planes) but I decided not to use it since the total entries of the first three csv file are already big enough (in total they are composed of about 80000 rows).

2.1. Field description

Here is the ER diagram:

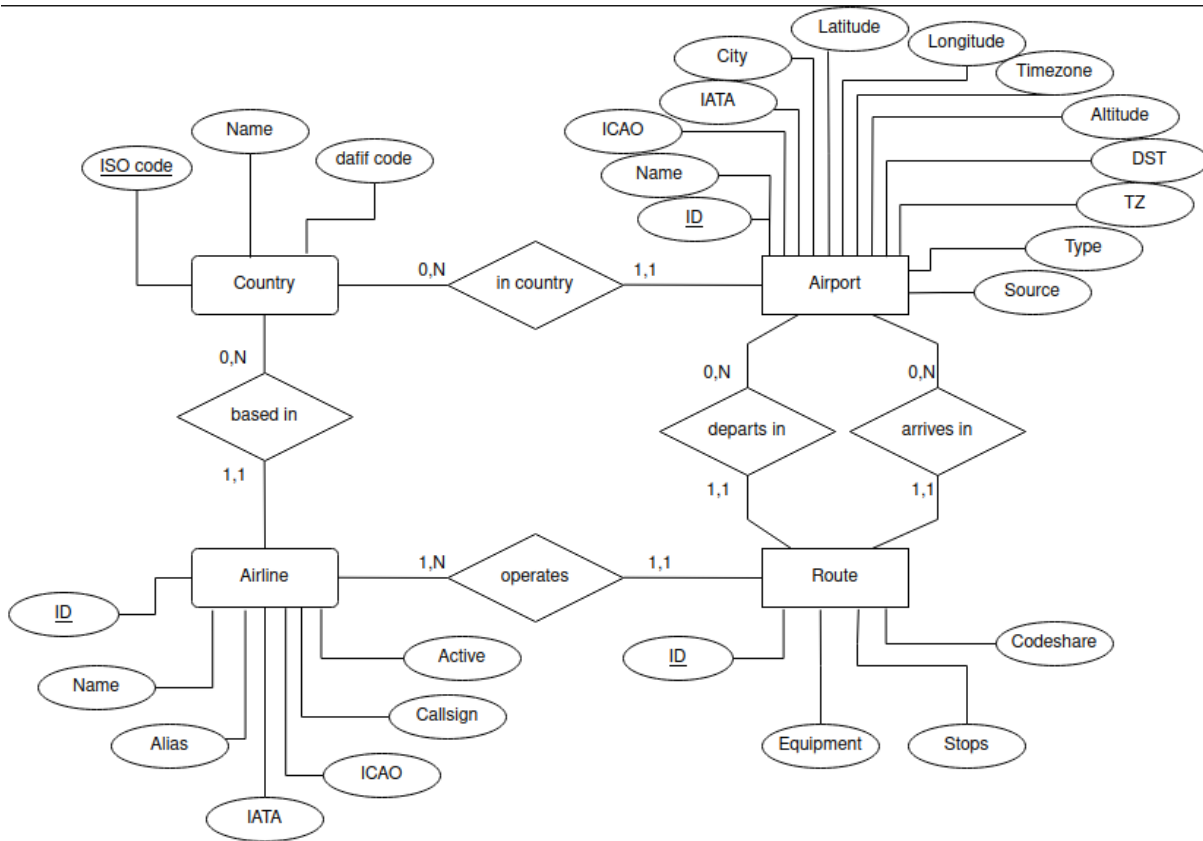


Figure 2.1: ER Diagram

For each entity I will now describe the value of its fields.

Airport Entity

- ID: Unique OpenFlights identifier for the airport.
- Name: Name of airport.
- IATA: 3-letter IATA code.
- ICAO: 4-letter ICAO code.
- City: Main city served by airport
- Latitude: Latitude of the airport in decimal degrees.
- Longitude: Longitude of the airport in decimal degrees.
- Timezone: Hours offset from UTC. Fractional hours are expressed as decimals.
- Altitude: altitude of the airport in feet.

- DST: Daylight savings time. One of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown).
- TZ: Timezone in "tz" (Olson) format
- Type: Type of the airport. Value "airport" for air terminals, "station" for train stations, "port" for ferry terminals and "unknown" if not known.
- Source: Source of this data. "OurAirports" for data sourced from OurAirports, "Legacy" for old data not matched to OurAirports (mostly DAFIF), "User" for unverified user contributions.

Route

- ID: Unique ID of the Route
- Equipment: 3-letter codes for plane type(s) generally used on this flight, separated by spaces.
- Stops: Number of stops on this flight ("0" for direct)
- Codeshare: "Y" if this flight is a codeshare.

Airline

- ID: Unique OpenFlights identifier for this airline.
- Name: Name of the Airline.
- Alias: Alias of the airline. For example, All Nippon Airways is commonly known as "ANA".
- IATA: 2-letter IATA code.
- ICAO: 3-letter ICAO code.
- Callsign: Airline callsign.
- Country: Country or territory where airport is located.
- Active: "Y" if the airline is or has until recently been operational, "N" if it is defunct.

Country

- ISO Code: Full name of the country or territory.

- Name: Unique two-letter ISO 3166-1 code for the country or territory.
- daff code: FIPS country codes as used in DAFIF. Obsolete and primarily of historical interest.

2.2. Neo4j implementation

2.2.1. Schema

Here is the schema of the graph database. The nodes and the relation between them are the same of the ER diagram but I added a new relation called *[:TO]*. This relation connects node of type *airport* to other nodes of the same type. The relation contains just one attribute called *RouteID*: it is an integer value that creates a bijection between a *[:TO]* relation and a *(:Route)* node: a new *[:TO]* relation is created between two airports whenever there is a *Route* that connects the airports. The *[:TO]* relations means that there is a path between two given airports. Why was this done? This may seem not necessary, but it allows to easily apply the shortest path function between two given airport without specifying the pattern of the path. In this way if I want to get the shortest path between the airports of MXP (Milano Malpensa) and LAX (Los Angeles):

```
1 MATCH p=shortestPath((:Airport {IATA:"MXP"})-[:TO*..5]->(:Airport {IATA:
   "LAX"}))
2 RETURN ...
```

However, even if the new relation allowed more flexibility in path-finding, it made a little bit more difficult to get a path where the route respect a given condition (see query *Shortest path with condition*)

Another option that would have made easier querying path with a given condition on the *Route* was to get rid of the *(:Route)* nodes and keep all the data in the *[:TO]* relation as attributes. I decided to keep the *(:Route)* nodes because it is useful to relate it with nodes of other types. Getting rid of the *(:Route)* node means moving all route data into the *[:TO]* relation that relates two airports: in this scenario if I want to make a query that involves airlines and their operated routes I cannot rely on any relationship between this two entities and to make the query I would lose all the benefit of using a graph database. For this reason I decided to keep both *[:TO]* relation and *(:Route)* node.

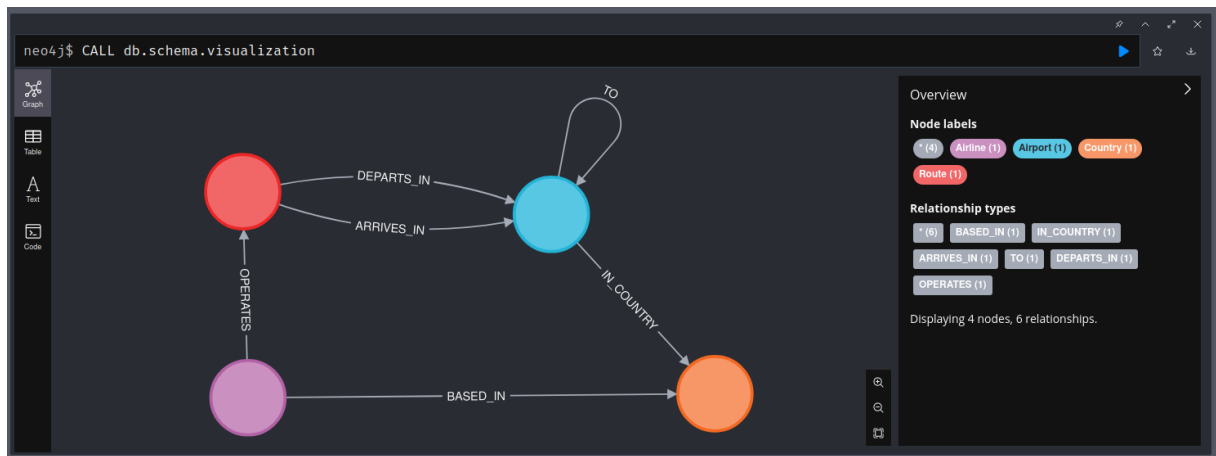


Figure 2.2: Neo4j DB schema

2.2.2. Creation queries

The project has been performed both on Neo4j Aura and on an instance of Neo4j database installed on a local machine. Here are reported the creation queries:

Loading Countries :

```
1 LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/
  FilippoIspanico/SMBUD_project/master/data/countries.dat" AS row
2 CREATE (:Country {Name: row.Name, Iso_code: row.Iso_code, Dafif_code:
  row.Dafif_code})
```

Loading Airports :

```
1 LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/
  FilippoIspanico/SMBUD_project/master/data/airports-extended.dat' AS
  line
2 MERGE (country:Country {Name: line.Country})
3 CREATE (airport:Airport {
4   AirportID: toInteger(line.AirportID),
5   Name: line.Name,
6   City: line.City,
7   IATA: line.IATA,
8   ICAO: line.ICAO,
9   Latitude: toFloat(line.Latitude),
10  Longitude: toFloat(line.Longitude),
11  Altitude: toInteger(line.Altitude),
12  Timezone: line.Timezone,
13  DST: line.DST,
```

```

14         TzDatabaseTimezone: line.TzDatabaseTimezone,
15         Type: line.Type,
16         Source: line.Source
17     })
18 CREATE (airport)-[:IN_COUNTRY]->(country)

```

Loading Airlines :

```

1     LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/
    FilippoIspanico/SMBUD_project/master/data/airlines.dat" AS row
2 WITH row
3 WHERE row.Country IS NOT NULL AND row.Country <> ""
4 MERGE (country:Country {Name: row.Country})
5 WITH row, country
6 CREATE (airline:Airline {
7     AirlineID : toInteger(row.AirlineID),
8     Name : row.Name,
9     Alias : row.Alias,
10    IATA : row.IATA,
11    ICAO : row.ICAO,
12    Callsign : row.Callsign,
13    Active : row.Active
14 })
15 CREATE (airline)-[:BASED_IN]->(country)

```

Loading routes :

```

1     CALL apoc.periodic.iterate(
2     "LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/
    FilippoIspanico/SMBUD_project/master/data/routes_id.dat' AS row
3 WITH row
4 WHERE row.SourceAirportID IS NOT NULL AND row.SourceAirportID <> ''
5 AND row.DestinationAirportID IS NOT NULL AND row.DestinationAirportID <>
    ''
6 AND row.AirlineID IS NOT NULL AND row.AirlineID <> ''
7 MERGE(a0:Airport {AirportID:toInteger(row.SourceAirportID)})
8 MERGE(a1:Airport {AirportID:toInteger(row.DestinationAirportID)})
9 MERGE(airline:Airline {AirlineID:toInteger(row.AirlineID)})
10 CREATE (a0)-[:TO {
11     RouteID:toInteger(row.RouteID)
12     }]->(a1)
13
14 CREATE (r:Route {
15

```

```
16     RouteID:toInteger(row.RouteID),
17     Airline:row.Airline,
18     AirlineID:row.AirlineID,
19     Codeshare:row.Codeshare,
20     Stops:row.Stops,
21     Equipment:row.Equipment
22 })
23
24 CREATE (a0)-[:DEPARTS_IN]-(r)-[:ARRIVES_IN]->(a1)
25 CREATE (airline)-[:OPERATES]->(r)
26 "
27 {batchSize:2, parallel:false}
28 )
```

Here the call to the APOC routine was necessary because Aura gave a problem on limits of the free memory.

3 | Queries

The text of all the queries can also be found in the file *queries.md* in the project repository (https://github.com/FilippoIspanico/SMBUD_project)

3.1. Highest Airport

The objective of this query is to find the commercial airport at highest altitude.

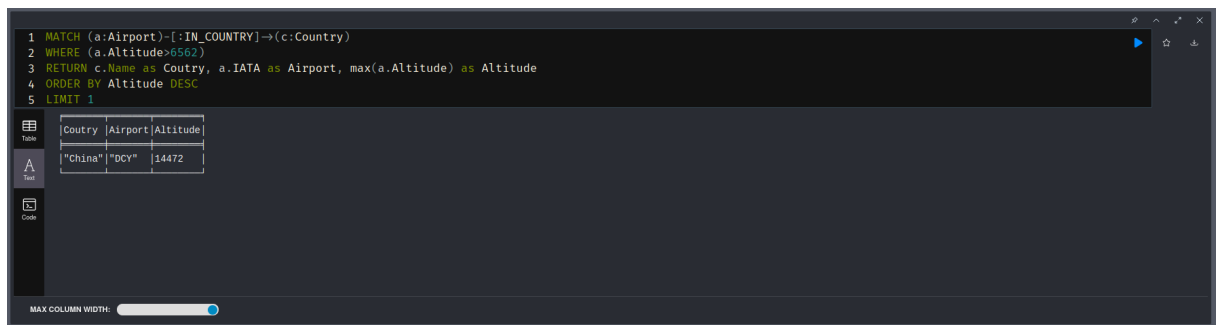
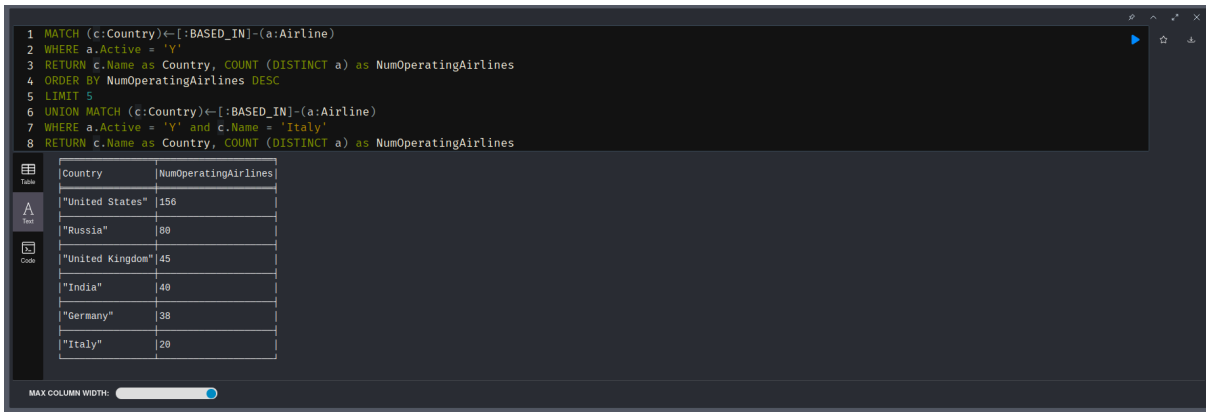


Figure 3.1: The Highest commercial airport in the world

Note that in this query the WHERE condition is unnecessary.

3.2. Countries with higher number of operating airlines

This query has the objective to identify the top 5 countries with the highest number of operating airlines. Has been inserted also the entry of Italy for comparison.



```

1 MATCH (c:Country)←[:BASED_IN]-(a:Airline)
2 WHERE a.Active = 'Y'
3 RETURN c.Name as Country, COUNT (DISTINCT a) as NumOperatingAirlines
4 ORDER BY NumOperatingAirlines DESC
5 LIMIT 5
6 UNION MATCH (c:Country)←[:BASED_IN]-(a:Airline)
7 WHERE a.Active = 'Y' and c.Name = 'Italy'
8 RETURN c.Name as Country, COUNT (DISTINCT a) as NumOperatingAirlines

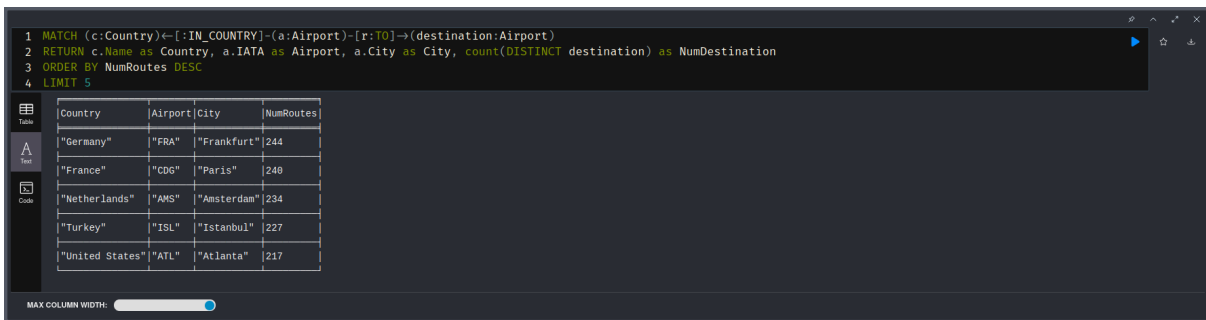
```

Country	NumOperatingAirlines
"United States"	156
"Russia"	88
"United Kingdom"	45
"India"	40
"Germany"	38
"Italy"	20

Figure 3.2: Operating airlines per Country

3.3. Top 5 Airports by number of destinations

This query has the objective of identifying the top 5 airports by number of destination offered



```

1 MATCH (c:Country)←[:IN_COUNTRY]-(a:Airport)-[r:TO]→(destination:Airport)
2 RETURN c.Name as Country, a.IATA as Airport, a.City as City, count(DISTINCT destination) as NumDestinations
3 ORDER BY NumDestinations DESC
4 LIMIT 5

```

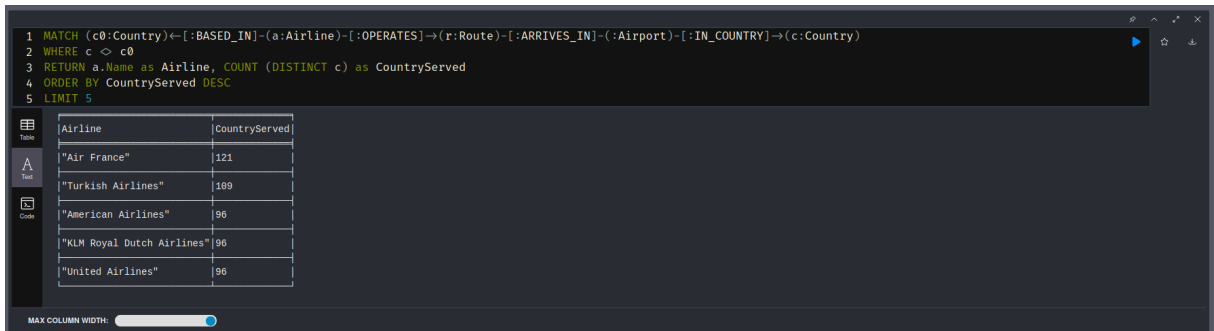
Country	Airport	City	NumDestinations
"Germany"	"FRA"	"Frankfurt"	244
"France"	"CDG"	"Paris"	240
"Netherlands"	"AMS"	"Amsterdam"	234
"Turkey"	"ISL"	"Istanbul"	227
"United States"	"ATL"	"Atlanta"	217

Figure 3.3: Top 5 airports by number of destination offered

The following two queries are thought for studying the airlines by two different metrics: the number of countries served and the number of destinations offered.

3.4. Biggest airlines by countries served

The objective of this query is to identify the biggest airlines by the number of countries served.



```

1 MATCH (c0:Country)-[:BASED_IN]-(a:Airline)-[:OPERATES]->(r:Route)-[:ARRIVES_IN]-(a1:Airport)-[:IN_COUNTRY]-(c:Country)
2 WHERE c <> c0
3 RETURN a.Name as Airline, COUNT (DISTINCT c) as CountryServed
4 ORDER BY CountryServed DESC
5 LIMIT 5


```

Airline	CountryServed
"Air France"	121
"Turkish Airlines"	109
"American Airlines"	96
"KLM Royal Dutch Airlines"	96
"United Airlines"	96

Figure 3.4: Top 5 Airlines by number of countries served

3.5. Biggest airlines by destination offered

The objective of this query is to identify the biggest airlines by the number of destination offered.



```

1 MATCH (a:Airline)-[:OPERATES]->(r:Route)-[:ARRIVES_IN]-(a1:Airport)
2 RETURN a.Name as Airline, COUNT (DISTINCT a1) as NumDestinations
3 ORDER BY NumDestinations DESC
4 LIMIT 5

```

Airline	NumDestinations
"American Airlines"	435
"United Airlines"	432
"Air France"	378
"KLM Royal Dutch Airlines"	361
"Delta Air Lines"	352

Figure 3.5: Top 5 Airlines by number of destinations offered

The next query will also analyze the airline market by looking at what are the most competed routes between airlines

3.6. Most competed routes

The objective of this query is to identify the most competed routes: competed in this context means the routes that are operated by the most number of airlines. This search has been limited to all routes that originate from airports in Spain. To facilitate easy interpretation of the results, in addition to the routes' beginning and ending airport, the list of competing airlines is also returned.

```

1 MATCH (r:Route)-[:OPERATES]-(airline:Airline {Active: "Y"})
2 MATCH (:Country {Name : "Spain"})-[:IN_COUNTRY]-(a0:Airport)-[:DEPARTS_IN]-(r)-[:ARRIVES_IN]-(a1:Airport)
3 RETURN a0.City as Origin, a1.City as Destination, count(DISTINCT airline) as OpAirlines, COLLECT(DISTINCT airline.Name) as AirlineNames
4 ORDER BY OpAirlines DESC
5 LIMIT 5

```

Origin	Destination	OpAirlines	AirlineNames
"Arrecife"	"London"	10	["Air Bourbon", "Thomsonfly", "Transavia France", "easyJet", "Ryanair", "Thomas Cook Airlines", "Condor Flugdienst", "British Airways", "Iberia Airlines", "Norwegian Air Shuttle"]
"Barcelona"	"New York"	9	["KLM Royal Dutch Airlines", "Iberia Airlines", "US Airways", "Delta Air Lines", "American Airlines", "Finnair", "Air France", "British Airways", "Qatar Airways"]
"Palma de Mallorca"	"London"	9	["Norwegian Air Shuttle", "Thomas Cook Airlines", "British Airways", "Transavia France", "easyJet", "Iberia Airlines", "Thomsonfly", "Air Bourbon", "Ryanair"]
"Madrid"	"New York"	9	["KLM Royal Dutch Airlines", "Air Europa", "Delta Air Lines", "American Airlines", "Air France", "British Airways", "Finnair", "US Airways", "Iberia Airlines"]
"Tenerife"	"London"	9	["Air Bourbon", "Thomsonfly", "Transavia France", "easyJet", "Ryanair", "British Airways", "Norwegian Air Shuttle", "Iberia Airlines", "Thomas Cook Airlines"]

Figure 3.6: Top 5 competed routes that departs from Spain

3.7. Most flight-dependant countries

Given that the aviation industry contributes significantly to greenhouse gas emissions, it could be interesting to look into which nations rely most heavily on this mode of transportation. The goal of this query is to determine the top 5 countries in terms of the number of flights used for internal travel (when the origin and destination airport are in the same nation).

```

1 MATCH (c:Country)-[:IN_COUNTRY]-(a:Airport)-[:TO]-(a2:Airport)-[:IN_COUNTRY]-(c)
2 RETURN c.Name as Country, count(DISTINCT r) as NumInternalRoutes
3 ORDER BY NumInternalRoutes DESC
4 LIMIT 5

```

Country	NumInternalRoutes
"United States"	10640
"China"	7046
"India"	3826
"Brazil"	1199
"Russia"	942

Figure 3.7: Top 5 flight dependant countries

3.8. Under-served routes

The objective of this query is to find routes that may be under-served. We look at Airports that are not connect with a direct flight but are only connected by flights that make one stop in an intermediate airport. Since the time complexity of this query is big, we limit the search at the US market.

```

1 MATCH(c:Country {Name: "United States"})←[:IN_COUNTRY]-(A:Airport)-[ab:TO]→(B:Airport)-[bc:TO]→(C:Airport)-[:IN_COUNTRY]→(c)
2 MATCH (B)-[:IN_COUNTRY]→(c)
3 WHERE NOT (A)-[:IN_COUNTRY]→(c) AND A < C
4 WITH A, B, C, COUNT(ab) as AB, COUNT(bc) as BC
5 RETURN A.Name as from, A.IATA as from_IATA, B.Name as stop, B.IATA as stop_IATA, C.Name as to, C.IATA as to_IATA, AB as IntRoutes, BC as FinRoutes, AB + BC
   as TOT
6 ORDER BY TOT
7 LIMIT 5

```

from	from_IATA	stop	stop_IATA	to	to_IATA	IntRoutes	FinRoutes	TOT
"Metropolitan Oakland International Airport"	"OAK"	"Boise Air Terminal/Gowen Field"	"BOI"	"Sacramento International Airport"	"SME"	1	1	2
"Metropolitan Oakland International Airport"	"OAK"	"Boise Air Terminal/Gowen Field"	"BOI"	"San Francisco International Airport"	"SFO"	1	1	2
"Metropolitan Oakland International Airport"	"OAK"	"Boise Air Terminal/Gowen Field"	"BOI"	"Norman Y. Mineta San Jose International Airport"	"SJC"	1	1	2
"Portland International Airport"	"PDX"	"Boise Air Terminal/Gowen Field"	"BOI"	"Lewiston Nez Perce County Airport"	"LWS"	1	1	2
"Metropolitan Oakland International Airport"	"OAK"	"Boise Air Terminal/Gowen Field"	"BOI"	"Lewiston Nez Perce County Airport"	"LWS"	1	1	2

Figure 3.8: Top 5 under-served routes in the US

The next queries will make the most out of the graph architecture of the database to identify the shortest path between two given airports. The first query will search for the shortest path without any condition, while the second query will introduce condition that may correspond to user request, like: "What is the shortest way to go from Milan (MXP) to London with easyJet?"

3.9. Shortest Path

This query answers the question: "What is shortest way to go from A to B?" . Given the restriction of the data (no information about travel times or price) here the shortest path between two airports is identified by the path that involves the less number of stops: for this reason to compute the user-requested path is used the neo4j built-in function `shortestPath`.

```

1 MATCH p=shortestPath((:Airport {IATA:"LMP"})-[:TO*..5]→(:Airport {IATA: "USH"}))
2 UNWIND relationships(p) AS r
3 MATCH (a1:Airport)-[r]→(a2:Airport)
4 MATCH (airline:Airline)-[:OPERATES]→(:Route {RouteID: r.RouteID})
5 RETURN a1.IATA AS From, a2.IATA AS To, airline.Name AS Airline, r.RouteID as RouteID

```

From	To	Airline	RouteID
"LMP"	"PMO"	"Flybaboo"	24392
"PMO"	"FCO"	"Ryanair"	27620
"FCO"	"EZE"	"Aerolineas Argentinas"	11460
"EZE"	"USH"	"Aerolineas Argentinas"	11458

Figure 3.9: Shortest path between Lampedusa (IT) and Ushuaia (ARG)

3.10. Shortest Path with condition

This query answer the question: "What is the shortest way to go from A to B with my favorite airline?" Like before, the shortest path is the one that involves the less number of stops.

```

1 MATCH (airline:Airline {Name:"easyJet"})-[:OPERATES]->(r:Route)
2 WITH COLLECT(DISTINCT r.RouteID) as RList
3 MATCH p=shortestPath((a0:Airport {IATA: "MXP"})-[:TO*..5]->(a1:Airport {IATA:"ARN"}))
4 WHERE all(r in relationships(p) WHERE r.RouteID IN RList)
5 UNWIND relationships(p) AS r
6 MATCH (a2:Airport)-[r]->(a3:Airport)
7 MATCH (airline:Airline)-[:OPERATES]->(r:Route {RouteID: r.RouteID})
8 RETURN a2.IATA as From, a3.IATA as To, airline.Name as Airline, r.RouteID as RouteID

```

From	To	Airline	RouteID
"MXP"	"BDS"	"easyJet"	55186
"BDS"	"GVA"	"easyJet"	54470
"GVA"	"ARN"	"easyJet"	54798

Figure 3.10: Shortest path between Milan and Stockholm using only easyJet flights.

Due to the schema of the graph db, this query is not straightforward. Computing the shortest path between two airports without any condition is easily done by looking at the `[:TO]` relations, however those relations don't contain any information about the flight, but just an ID of a `(:Route)` node. The `(:Route)` node contains all the information about the flight (or is related to other relevant nodes). So if we want impose a condition about the flight, like in the case of this query we have to:

1. match all `(:Route)` nodes that satisfy the requested condition and extract their IDs.
2. Apply `shortestPath` function and impose that the `[:TO]` RouteID must be in the list of the previously extracted IDs.

Here, for comparison, is reported the shortest path between the same airports without any conditions:

```

1 MATCH p=shortestPath((:Airport {IATA:"MXP"})-[:TO*..5]->(Airport {IATA: "ARN"}))
2 UNWIND relationships(p) AS r
3 MATCH (a1:Airport)-[r]->(a2:Airport)
4 MATCH (airline:Airline)-[:OPERATES]->(r:Route {RouteID: r.RouteID})
5 RETURN a1.IATA AS From, a2.IATA AS To, airline.Name AS Airline, r.RouteID as RouteID

```

From	To	Airline	RouteID
"MXP"	"LIS"	"Alitalia"	13551
"LIS"	"ARN"	"TAP Portugal"	53687

Figure 3.11: Shortest path between Milan and Stockholm.

4 | Extra

As mentioned in the introduction, for the extra work I am proposing a chat-based assistant. Thanks to the design of the database is possible to obtain the shortest path between two airports quite easily. Now, I would like to make this information easily accessible to the user in a chat-based environment. To do so I have developed a simple Telegram bot. The bot uses an openAI assistant to interact to the user and, when needed, it calls a function that queries the Neo4j database with the data given by the user. To keep things simple the bot will be able to make just one type of query to the database (of course the parameters of the query may vary). The chosen query is the previously seen *shortest path query*.

All the system is designed in python and its implementation can be found in my GitHub repository at https://github.com/FilippoIspanico/SMBUD_project. Here is a simple component diagram that illustrates the system:

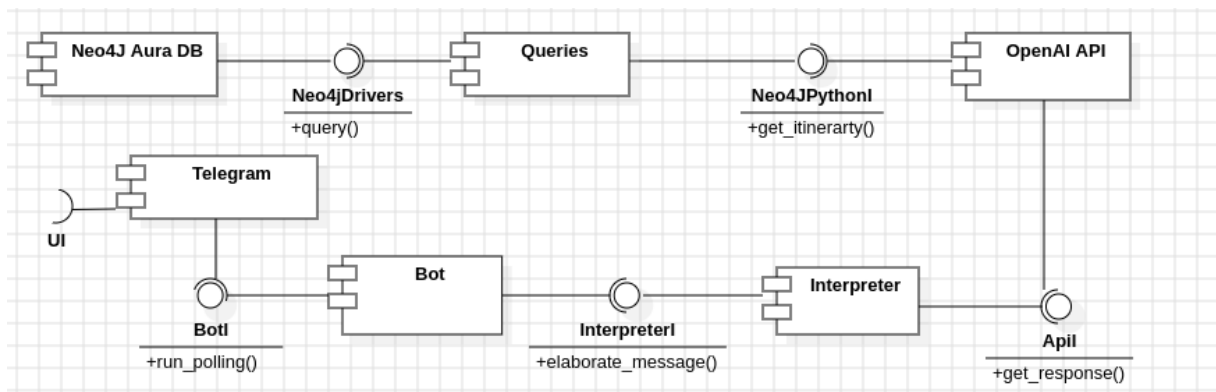


Figure 4.1: Component diagram.

There are three main component of the system:

- Bot component. It is implemented in the file `scripts/bot.py`. Here it is configured the connection with telegram API to send and receive messages.
- Interpreter component. In the file `scripts/interpreter.py` is configured the interaction with the openAI API. It is also defined a function that elaborates a user message.

- Queries components. In the file *scripts/queries.py* is configured the interaction with the Neo4j database using Neo4j Python drivers. The query text is hard coded into a function called *get_itinerary*.

Some remarks:

- The python scripts are currently running on an Azure virtual machine.
- The Neo4j database that is queried by the bot is an instance of a Neo4j Aura database. Aura was chosen because is offered as a cloud service: in this way it was not necessary to install Neo4j server on the Azure Virtual Machine, so that the hardware architecture of the VM was kept as cheap as possible.
- The choice of deploying the system on Azure is justified by the fact that students are granted with free credits. Telegram has been chosen over Whatsapp because its API calls are free. However, to interact with Telegram's API, has been used a python interface built by the community that can be found here.

The bot can be accessed and used through this link: t.me/FlightAI_bot (to use it is necessary to be registered to Telegram)

List of Figures

2.1	ER Diagram	4
2.2	Neo4j DB schema	7
3.1	The Highest commercial airport in the world	11
3.2	Operating airlines per Country	12
3.3	Top 5 airports by number of destination offered	12
3.4	Top 5 Airlines by number of countries served	13
3.5	Top 5 Airlines by number of destinations offered	13
3.6	Top 5 competed routes that departs from Spain	14
3.7	Top 5 flight dependant countries	14
3.8	Top 5 under-served routes in the US	15
3.9	Shortest path between Lampedusa (IT) and Ushuaia (ARG)	15
3.10	Shortest path between Milan and Stockholm using only easyJet flights. . .	16
3.11	Shortest path between Milan and Stockholm.	16
4.1	Component diagram.	17

