Report: Public Transport Project

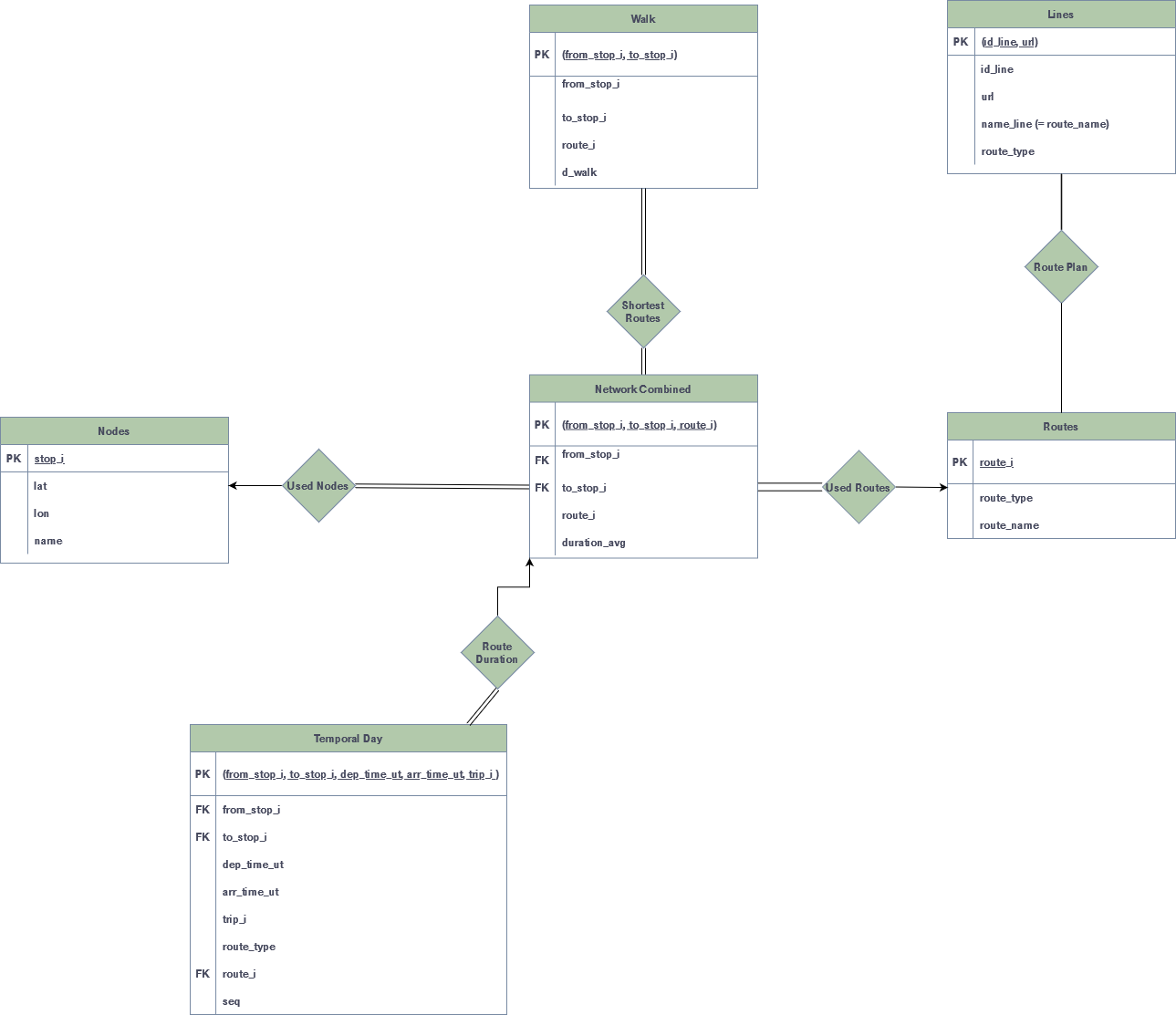
I – Functionalities list

A **requirements.txt** file has been generated to help the end-user install the modules necessary for this program. A **README.MD** file was written to clarify the installation and usage.

Given a dataset of public transport network data (.**csv** files containing network nodes and temporal data, .**geojson** files containing routes and stops names, all taken from <https://www.nature.com/articles/sdata201889>, as well as additional **.json** files taken from ) that is kept within a folder on the user’s computer, as well as a *PostgreSQL* server, our *PyQT5* application can:

* Prompt the user the path to the **data** folder (with a *FileDialog* that will be used to select the folder), as well as the login credentials necessary to connect to the postgres database (with a custom *QDialog* that has four entries: **user**, **password** (this one does not show plain text), **database** and **host** (the IP address which the postgres server runs on).
* Save the variables from previous step inside a “**params.json** file so they can be reused next time the program is launched. The json file, if already existing, is converted back into a set of variables by using the *json* module.
* Fill the postgres database with data obtained from the files within the **data** folder. The module *pandas* is used to convert the *.csv* files into a **DataFrame** (pandas’ way of storing tabular data). Said DataFrame is then used to create an equivalent *PostgreSQL* table. When necessary, we use the *psycopg2* module to run additional *SQL* requests that set the uniqueness of certain attributes in the table.
* Display a *QWindow* where the user can select a departure and a destination from two different Combo Boxes. Once both are selected, the user can press the “Go” button to launch the query that will give a path from the departure point to the arrival point. The user can also click the “Clear” button to clear the text within the Combo Boxes.
* Said window also features a *QTableWidget* that shows the detailed route the user must take to go from the departure point to the destination point on its first row, and the estimated time it would take on the second row. Clicking on a transport name (*RER E, BUS 165*…) within that table will make appear a message box, asking the user if they want to see said transport’s plan/map. If the user accepts, we will open the transport’s plan page within their default web browser. If there are multiple plans related to a transport name (often happens given how generic bus names are for example).
* Below that table, is a map (displayed with the *folium* module from a web page), which, when a route has been decided, shows all the stops of said route, linking each stop by a line, otherwise it shows all the stops that we know of on the map. The user can click a location on the map to select their departure and destination stops as well. All these actions are done by running *JavaScript* code with the *folium* module to interact with the map websites.
* Once the user has decided on a path to take, we use the *NetworkX* module to define the shortest path between the departure node and the arrival node. Given a set of *nodes* linked by *edges* (here routes), the module runs the *Dijkstra algorithm*, a graph known for its efficiency on finding the shortest path for road networks.

II – Entity-Relationship Diagram & Dependencies



*Fig.X: The* ***Entity-Relation Diagram***

The entities have been found with the following logic:

* The tables **nodes, combined, walk** and **temporal\_day** are based on the .*csv* files in the data folder.
* The table**route** is based on the .*geojson* file in the data folder.
* The table **line** is based on the *.json* files *fichers-horaires-et-plans* & *referentiel-des-lignes.*
* All the tables above have been imported by keeping the structure of their file:
* the keys for the tables based on the *.csv* files take the values of the first row within said files for each column that we need.
* The keys for the **route** table uses the *geojson* module to read the file as a dictionary and base the final table on the keys of said dictionary.
* The keys for the **line** table were generated in the same way, except that we merged both *json* dictionaries to only keep the data of transport lines that have an *URL* attached to them.

The **nodes** file stores data about all the stops in Paris. It contains the following keys:

* *stop\_i*: the ID of a stop
* lat: the latitude of a stop
* *lon*: the longitude of a stop
* *name*: the name of a stop

Une image contenant texte

Description générée automatiquement

*Fig.X: The* ***nodes*** *table*

**stop\_i** allows us to know the other columns, and, logically speaking, knowing the **latitude** and the **longitude** of a stop should give us both the **ID** and the **name** of it but it doesn’t mean the **ID** of a stop is dependent on its coordinates. In other words, the dependencies in this table are the following:

stop\_i -> lon, lat, name (primary key)

This means the **nodes** table is BCNF since stop\_i is a superkey of the table, which respects the condition ( for all dependencies a -> b in the list of functional dependcies of **routes**, a is superkey

The **routes** file stores the data of all the possible routes that exist in Paris, without worrying about the context (aka which stops are bound to said routes). It contains the following keys:

* *route\_i: the ID given to a route (*i.e a possible path between two stops by using a certain transport)
* *route\_type:* the transport ID that we use to go between two stops (0 is for trams, 1 is for metro, 2 is for trains, 3 is for buses… some other IDs exist but are not used in the Paris data, such as a Funicular ID or even a Boat ID)
* *route\_name:* the name that is given to said route (A, 14, T8…)

Une image contenant texte, équipement électronique, capture d’écran

Description générée automatiquement

*Fig.X: The* ***routes*** *table*

In this table, knowing **route\_i** gives **route\_type** and **route\_name**, making it a primary key. Knowing **route\_type** or **route\_**name alone does not allow to know any other key in the table, knowing (**route\_type,** **route\_name)** does NOT give **route\_i**, since we can have multiple transports of the same type and the same name from different places (Bus 116 for Epinay sur Orge and Bus 116 for Rosny-Bois-Perrier for example, are different bus lines)

This leaves us with the following dependency:

route\_i -> route\_type, route\_name

That means the **routes** table is in **BCNF** form, we only have one dependency and it’s a superkey of the table (we can derive it into route\_i -> route\_type; route\_i -> route\_name but route\_i would still be a superkey).

The **combined** file, as its name indicates, stores the data of all the possible routes one could have in Paris between two stops, no matter the transport used, instead of having to look up the routes for each transport separately (as it’s the case in any other network\_[mode].csv file). It contains the following keys:

* *from\_stop\_i*: the departure stop of a route
* *to\_stop\_i*: the arrival stop of a route
* *d:* the straight-line distance between two stops, **we won’t use it**
* *n\_vehicles*: the number of vehicles that took the rout, **we won’t use it**
* *duration\_avg:* the average duration of a trip between two stops on a certain route
* *route\_i: the ID given to a route (*i.e a possible path between two stops by using a certain transport)
* *route\_type*: the transport ID that we use to go between two stops, **we won’t use it** (we can get that value from the **routes** table)

The keys **from\_stop\_i, to\_stop\_i** and **route\_i** are *foreign:* **from\_stop\_i** and **to\_stop\_i** reference the **stop\_i** key from the **nodes** table, whereas **route\_i** references the eponym key from the **routes** table.

Une image contenant texte

Description générée automatiquement

*Fig.X: The* ***combined*** *table*

Knowing any key by itself isn’t enough to find the other values. For example, knowing **route\_i** doesn’t allow us to **from\_stop\_i** nor **to\_stop\_i**. There might me multiple routes that allow us to go from one stop to another, so the composite key (**from\_stop\_i, to\_stop\_**i) wouldn’t be enough to find **route\_**i. We can then then write the following dependencies:

From\_stop\_i, to\_stop\_i, route\_i -> duration\_avg

As we cannot derive this dependency, and it’s a superkey of the table, we can safely say that the table **combined** is in BCNF form.

The **temporal\_day** file contains the following columns:

* *from\_stop\_i*: the departure stop of a route
* *to\_stop\_i*: the arrival stop of a route
* *dep\_time\_ut*: the Unix time at which we take the route from the departure stop
* *arr\_time\_ut*: the Unix at which we are supposed to arrive to the arrival stop
* *route\_type*: the transport ID that we use to go between two stops, **we won’t use it**
* *trip\_I*: gives the ID given to a certain sequence of routes taken, **we won’t use it**
* *seq*: gives the sequence number within a trip, **we won’t use it**
* *route\_i: the ID given to a route (*i.e a possible path between two stops by using a certain transport)

Une image contenant texte, capture d’écran, moniteur

Description générée automatiquement

*Fig.X: The* ***temporal\_day*** *table*

This table is used to get the estimated duration of a trip between two stops. We can subtract the departure time from the arrival time to get a trip’s duration (in seconds).

This table has a composite *foreign key*, **(from\_stop\_i, to\_stop\_i, route\_i)**, which seems to reference **combined**’s primary key: both tables feature all the possible routes going from one stop to another within Paris, with the difference being that combined gives the average duration of a trip on said route, while **network\_day** gives the duration of a trip on said route depending on the current time (so it has way more values).

The *primary key* of this table is the composite key **(from\_stop\_i, to\_stop\_i, dep\_time\_ut, arr\_time\_ut, route\_i)**:

* we can go from a stop to multiple other stops, which means having **from\_stop\_i** or **to\_stop\_i** alone isn’t enough to know any other key in the table.
* the same route can be used to have a trip between multiple stops, so having **route\_i** does not allow us to know the departure or the destination stops.
* We can take multiple routes to go from a stop to another, so the composite key (**from\_stop\_i, to\_stop\_i)** isn’t sufficient to know other keys in the table.
* knowing the departure time alone does not allow us to know which route was taken, nor where it was from or to, neither when the arrival time would be. Thus, knowing **dep\_time\_ut** or **arr\_time\_ut** alone wouldn’t be of much use.
* The composite key (**dep\_time\_ut, arr\_time\_ut)** doesn’t tell much either
* we can take a route from one stop to another at different times so having **(from\_stop\_i, to\_stop\_i, route\_i)** alone wouldn’t be enough.
* We can also have multiple routes assigned to a same time so using **(from\_stop\_i, to\_stop\_i, dep\_time\_u, arr\_time\_ut)** wouldn’t suffice.
* We might be able to go from a stop to multiple others, during the same timelapse and using the same route. Same goes from a set of departure stops that would have the same arrival stop in the same timelapse using the same route, meaning that neither the **(from\_stop\_i, dep\_time\_ut, arr\_time\_ut, route\_i)** nor the (**to\_stop\_i, dep\_time\_ut, arr\_time\_u, route\_i)** are enough.

And so, it seems that the only dependency the **network\_day** table possesses is the combination of all the keys.

The functional dependencies are the following:

from\_stop\_i, to\_stop\_i, route\_i, dep\_time\_ut, arr\_time\_ut ->

from\_stop\_i, to\_stop\_i, route\_i, dep\_time\_ut, arr\_time\_ut (primary key)

Obviously, the **network\_day** table is in **BCNF** form since its functional dependencies are trivial. Since a -> b is trivial, this implies that the derived dependencies from a -> b will be trivial as well.

The **fiches-horaires-et-plans** and **referential-des-lignes** files store data about all the transport lines in Paris. It contains the following keys:

* *transportmode*: the type of transport related to a line; we can use it like **route\_type**
* *name\_line*: the name of the line
* *id\_line*: the internal ID of a line in the *IDF Mobilités* dataset, has NOTHING to do with any other ID of our tables, such as route\_i or stop\_i.
* *url*: the URL on which a plan related to the line is located.
* *type*: the type of document the url redirects to (can be a plan of the line or its schedule)**, we won’t use it** (as some schedules contain both the plan and the schedule, filtering the data would not be useful)
* *externalcode\_line****:*** external ID of a line in the IDF Mobilités dataset, **we won’t use it**
* *operatorref:* the ID of the line’s operator, **we won’t use it**
* *colourprint\_cmjn*: the colors of the line, in CMYK form, **we won’t use it**
* *textcolourprint\_hexa*: same, but in hexadecimal, **we won’t use it**
* *accessibility:* how accessible the line is, **we won’t use it**
* *audiblesigne\_available:* if the line has audible signs, **we won’t use it**

***etc, etc…***

Une image contenant texte

Description générée automatiquement

*Fig.X: the* ***lines*** *table*

It isn’t possible to know any other value in the table by knowing **route\_type** alone, same goes for the **url** (some plans show multiple lines at the same time) and **id\_line** ( the same **id\_line** can have multiple **url** values: one for the plan, the other for the schedule).

If we know the **url** and **id\_line**, we can find **route\_type** and **name\_line** (a certain transport line can only have one url associated to it)**.**

Knowing **(url, name\_line)** or even **(url, name\_line, route\_type)** isn’t enough to know **id\_line**, we can imagine the case of two lines “XX” lines of the same transport mode hosted on a main website where the url would redirect to the main page (example: the table contains two lines named *TER Hauts-de-France* which have the same *route\_type* and redirect to https://ter.sncf.com/hauts-de-france).

As such, the only dependency we have is the following:

id\_line, url -> route\_type, name\_line (primary key)

Once again, we can safely conclude this table is in **BCNF** form.

Now that we know the characteristics of our entity tables, we can derive the relations

III – Encountered Difficulties & Contributions per member