Année 2022-2023

Institut GALILÉE

Rapport projet

UE BASE DE DONNÉES

City mapper PARIS

Projet réalisé par :

Nicolas CALCAR
Maurice Kaliva BORE
Aminata Aliou BA

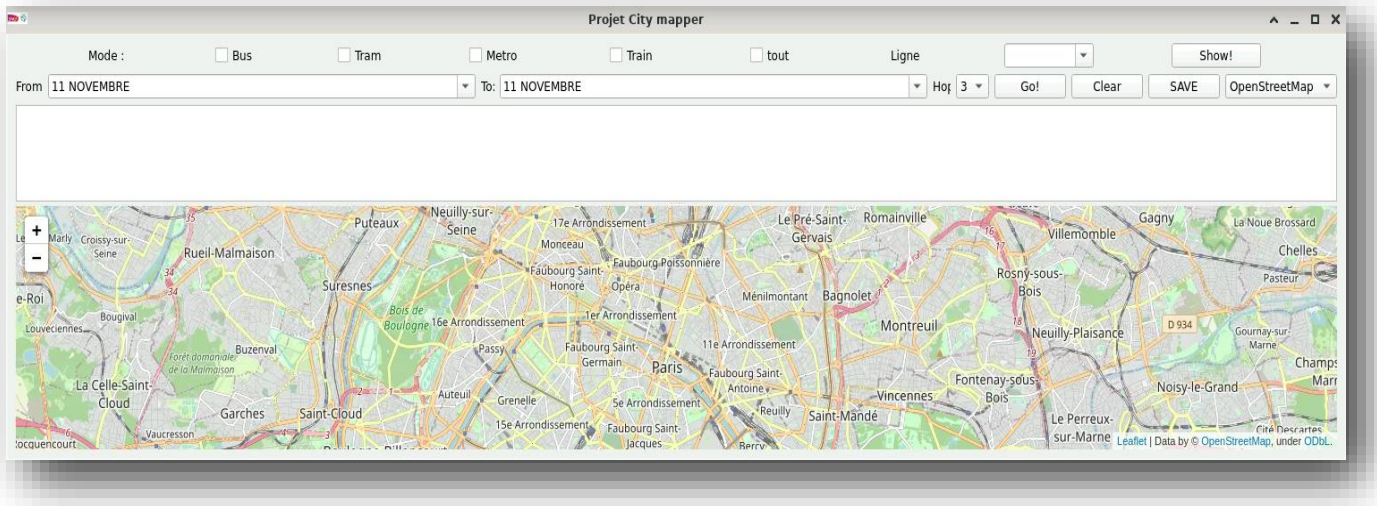
Enseignant :

Nabil MUSTAFA

Sommaire

1 Les fonctionnalités de l'application

L'application du city mapper possède plusieurs fonctionnalités que nous avons implémenté. Dans un premier temps, nous avons installé les modules nécessaires au fonctionnement de l'application. Nous avons installé les modules comme folium qui est un puissant outil de visualisation avec la conception de carte interactives. Nous avons installé psycopg2 qui nous permet d'utiliser la base de données PostgreSQL, nous avons installé PyQt5 et PyQtWebEngineWidgets qui nous permettent d'utiliser et de gérer des fenêtre. Une fois les modules installés, il nous faudra se rendre dans le répertoire du projet et de lancer le fichier python en utilisant le terminal et en tapant la commande « python3 projet_executable.py»



En lançant l'application nous accédons à une fenêtre composée de trois parties :

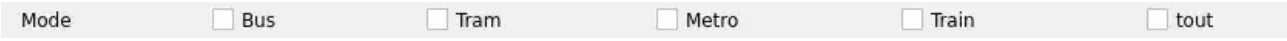
- Un panneau de contrôle
- Un espace où on obtiendra le résultat
- Une carte qui est centré à Paris.

Commençons aux fonctionnalités en rapport à l'esthétique de la fenêtre et au configuration des trajets.

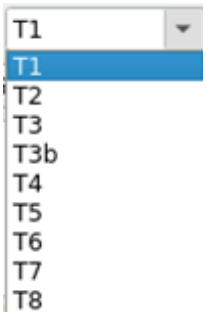
- Nous avons mis en place un logo de la SNCF et de la RATP qui se situe au niveau supérieur gauche de la fenêtre en tout petit et nous avons mis en place un titre pour notre fenêtre

Pour commencer, nous avons mis en place deux choix :

1) L’utilisateur peut rechercher toutes les gares qui desservent une ligne précise.

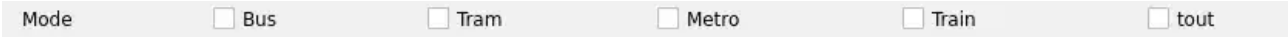


L’utilisateur doit cocher une case soit un bus, soit un tram, soit un métro, soit un train ou soit la case « tout ». Dès que le mode de transport est choisi, l’utilisateur peut choisir une ligne en fonction du mode de transport. Par exemple si on coche la case « Tram » on aura le choix entre les lignes qui ont pour type « Tram » donc la fenêtre afficheras :

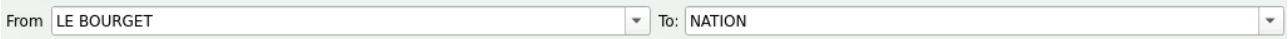


Si on choisit par exemple le T8, et qu’on appuie sur le bouton show. On aura pour affichage toutes les stations qui ont le T8 comme ligne.

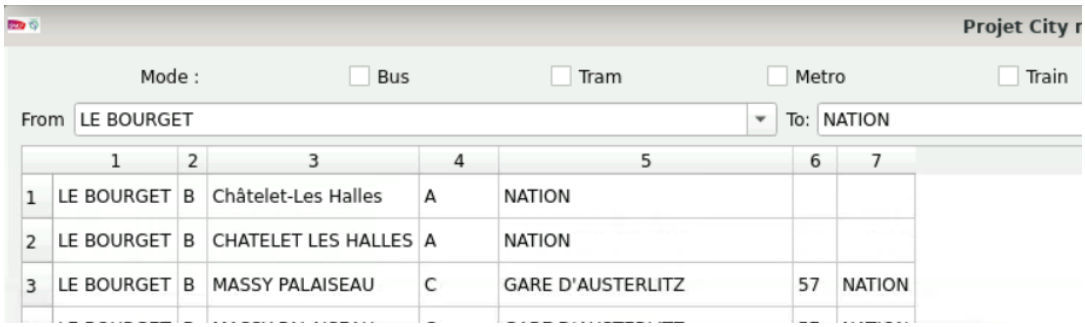
2) L’utilisateur peut choisir un trajet composé d’**un** mode de transport (le choix « tout » est par défaut si rien n’est coché) ce qui change les possibilité sur « from » et « to ».



L’utilisateur a la possibilité de rechercher tous les trajets qui ont pour le départ et la destination le choix de l’utilisateur.



Il a aussi le choix de demander un trajet avec un nombre maximal de ligne utiliser. Il a le choix d'utiliser entre 1 et 3 lignes au maximum exemple avec :

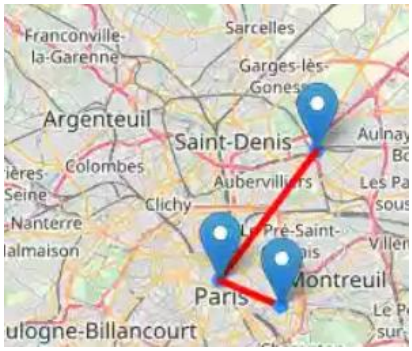


	1	2	3	4	5	6	7
1	LE BOURGET	B	Châtelet-Les Halles	A	NATION		
2	LE BOURGET	B	CHATELET LES HALLES	A	NATION		
3	LE BOURGET	B	MASSY PALAISEAU	C	GARE D'AUSTERLITZ	57	NATION

3) L'itinéraire s'affiche en appuyant sur le bouton « Go ! » qui va afficher le résultat en exécutant une requête SQL pour l'afficher dans la partie blanche sous la forme d'un tableau.

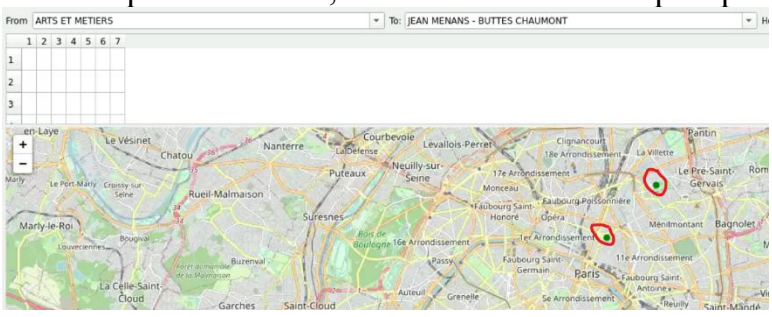
Si le chemin n'existe pas il écrit sur le terminal « AUCUN RESULTAT !!! »

4) Si on clique sur une case du tableau dans pour le trajet on peut créer un chemin sur la carte avec l'exemple du 2).



5) On a aussi implémenté une touche « Clear » qui permettra de restaurer les valeurs par défaut et de remettre les cases en non cochées.

6) Si on clique sur la carte, on trouve la station la plus proche de l'endroit cliqué.

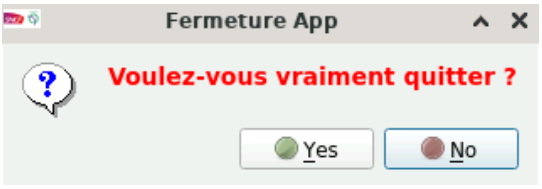


7) L'utilisateur a aussi après avoir créé un trajet dans l'application, la possibilité de sauvegarder les itinéraires dans un fichier en appuyant sur le bouton « SAVE »

ce qui crée un fichier « historique_chemin.txt » où seras indiqué la date et l'heure de la sauvegarde ainsi que les itinéraires comme dans l'exemple ci-dessous.

```
1 Fait le : 23/12/2022  à 21:27:07
2 LE BOURGET  |B  |Châtelet-Les Halles    |A  |NATION |
3 LE BOURGET  |B  |CHATELET LES HALLES    |A  |NATION |
4 LE BOURGET  |B  |MASSY PALAISEAU    |C  |GARE D'AUSTERLITZ  |57 |NATION |
```

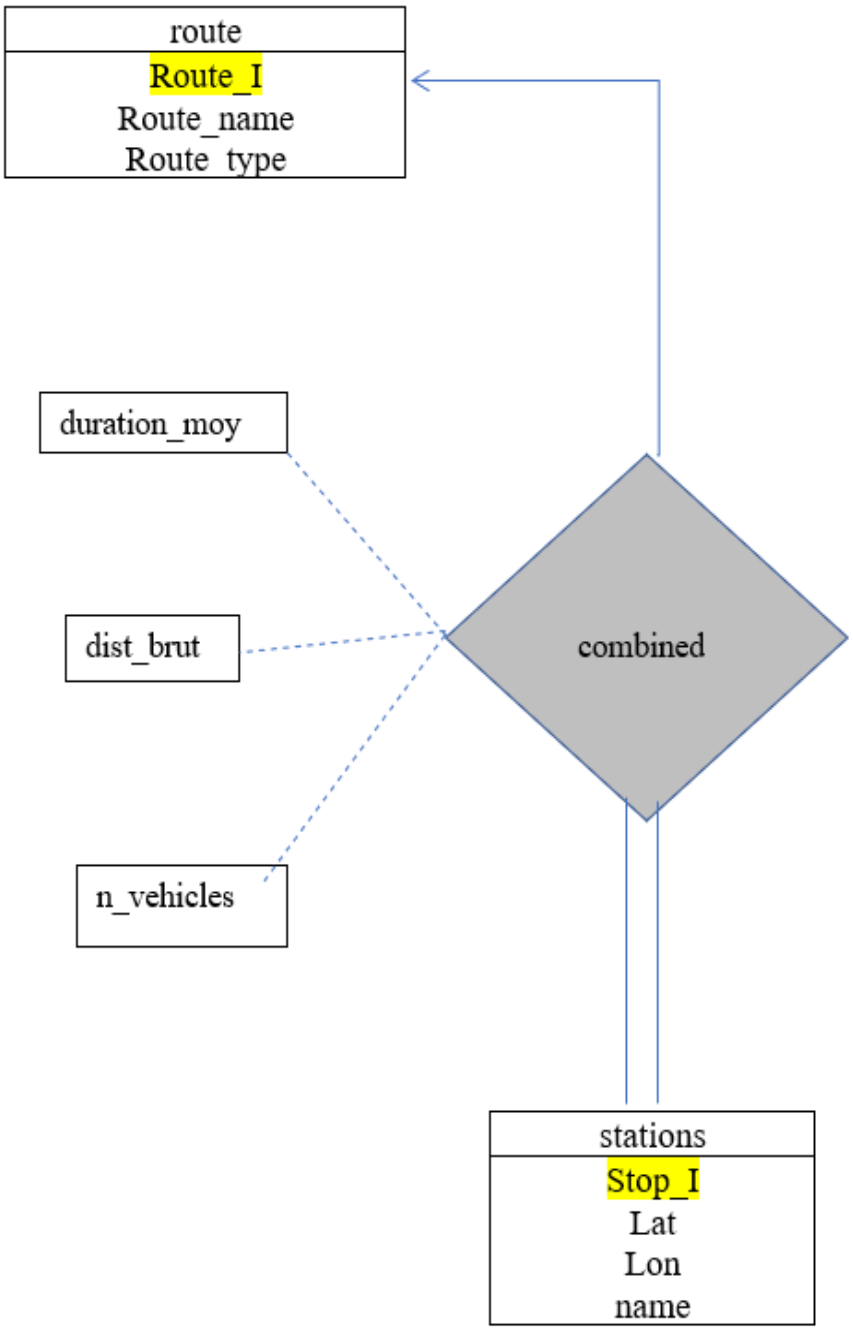
8) Nous avons créé un message de validation si l'utilisateur souhaite quitter l'application en cliquant sur la croix.



9) Pour finir, l'utilisateur a aussi la possibilité de changer le modèle de la carte selon son choix. La carte par défaut sera « openstreetmap » mais il aura aussi le choix d'utiliser stamen Terrain, stamentoner et cartodbposition.

2 Entity-Relationship

Entity Relation diagram



3 Passage de ER

Pour construire la base de données, nous avons utilisées les fichier csv fourni dans le dossier du projet. Pour obtenir les données de la table stations nous avons extrait les données des stations à l'aide d'un script Python depuis le fichier « network_nodes.csv » pour effectuer un parsing des données contenue dans le fichier pour créer un fichier « stations.sql » qui contient :

INSERT INTO station VALUES (stop_I, lat, lon, name) ;

Pour chaque ligne du fichier csv avec stop_I l'id de la station, lat qui est la latitude de la station, lon qui est la longitude de la station et name qui est le nom de la station. Nous avons créé un fichier « stations_schema.sql » qui contient les types de chaque attributs de la table station. Une fois les données récupérées, nous avions ajoutée et crée la table station dans la base de données PostgreSQL à l'aide des commandes :
« \i stations_schema.sql » et de « \i stations.sql ».

Pour obtenir les données de la table route nous avons extrait les données du fichier « routes.geojson » à l'aide d'un fichier script Python « create-routeI-routeName-routeTypefile.py »
Pour obtenir le fichier « create-routeI-routeName-routeTypefile.csv » et comme le paragraphe précédent nous avons utilisé un fichier pour le parsing pour créer un fichier « routes.sql » qui contient :

INSERT INTO routes VALUES (route_I, route_name, route_type) ;

Avec route_I qui est l'id de la ligne, route_name qui est le nom de la ligne et route_type qui est le type de la ligne (bus, train, tram, métro). Nous avons créé un fichier « route_sche.sql » qui contient les types de chaque attributs de la table routes. Une fois les données récupérées, nous avions ajoutée et crée la table routes dans la base de données PostgreSQL à l'aide des commandes :
« \i route_sche.sql » et de « \i route.sql ».

En regardant et en lisant la documentation autour des fichiers csv , nous nous sommes rendu compte que le fichier « network_combined.csv » est composé de l'ensemble des transports en commun (network_subway.csv, network_bus.csv, network_tram.csv, network_rail.csv), ce qui nous a facilité la mise en place de la table car nous pouvions utiliser uniquement le fichier « network_combined.csv » à la place des 4 autres tables séparément. La création a été plus difficiles car l'attribut route_I_counts est composée de deux variables dont route_I, nous avons donc utilisées un script python pour pouvoir séparer les deux variables et les attribuer aux bonne variables. Le fichier « combined.sql » contient :

INSERT INTO combined_projet VALUES (from_stop, to_stop, dist_brut, duration_moy, n_vehicles, route_I, route_type) ;

Avec from_stop et to_stop id des stations, dist_brut distance entre from_stop et to_stop, duration_moy temps moyen entre les stations, n_vehicles le nombre de véhicule entre les deux stations. Nous avons créé un fichier « combined_schema.sql » qui contient les types de chaque attributs de la table routes. Une fois les données récupérées, nous avions ajoutée et crée la table routes dans la base de données PostgreSQL à l'aide des commandes « \i combined_schema.sql » et de « \i combined.sql ».

Finalement à l'aide de toutes les commandes et des fichiers scripts Python on a réussi à passer de la forme fichier.csv à obtenir notre diagramme entité relation.

4 Liste des tables

Nous allons définir les tables utilisées dans notre projet.

Route		
route_I	route_name	route_type

La table route est composée des informations des lignes utilisés dans les transports en commun.
La table est composée pour chaque ligne de l'id de la ligne (route_I), du nom de la ligne (route_name) et du type de la ligne 0 pour le tramway, 1 pour le métro, 2 pour le trains et 3 pour le bus (route_type).

route_I et route_type sont de type INT et route_name est de type TEXT.

La candidate key est composée de route_I.

stations			
stop_I	Lat	lon	name

La table stations est composée des informations de chaque station utilisées dans les transports en commun.
La table est composée pour chaque station de l'id de la station (stop_I), de la latitude (lat), de la longitude (lon) et du nom de la station (name).

Stop_I est de type INT, lat et lon sont de type NUMERIC(10,5) et name est de type TEXT.

La candidate key est composée de stop_I.

Combined_projet						
from_stop	to_stop	dist_brut	duration_moy	n_vehicles	route_I	route_type

La table combined_projet est composée des informations de déplacements entre deux stations.
La table est composée de l'id de la station de départ (from_stop), de l'id de la destination (to_stop), de la distance brute entre les deux stations (dist_brut), de la durée de trajet moyen entre les deux stations (duration_moy), du nombre de véhicule qui traverse dans un temps donnée (n_vehicles), de l'id de la ligne utilisée (route_I) et du type de la ligne utilisé (route_type)

from_stop et to_stop sont en rapport avec la table stations, route_I et route_type sont en rapport avec la table route.

from stop, to_stop, dist_brut, n_vehicles, route_I, route_type sont de type INT et duration_moy est de type numeric(10,6).

La candidate key est composée de from_stop, to_stop, route_I, route_type.

5 Liste des dépendances

Nous allons définir les dépendances pour chaque table

- Pour la table stations

Comme expliqué précédemment la table stations est composée des attributs :

i. stop_I
ii. lat
iii. lon
iv. name

on a comme dépendances :

stop_I → lat
stop_I → lon
stop_I → name

Donc par addition on a aussi :
stop_I → lat,lon,name

Car comme expliqué précédemment on sait que stop_I est une superkey et que stop_I est une clé candidate.

- Pour la table route

Comme expliqué précédemment la table route est composée des attributs :

i. route_I
ii. route_name
iii. route_type

On a comme dépendances :

route_I → route_name
route_I → route_type

Donc par addition on a aussi :
route_I → route_name, route_type

Car comme expliqué précédemment on sait que route_I est une superkey et que route_I est une clé candidate.

- Pour la table combined_projet

Comme expliqué précédemment la table combined_projet est composée des attributs :

i. from_stop
ii. to_stop
iii. dist_brut
iv. duration_moy
v. n_vehicles
vi. route_I
vii. route_type

On a comme dépendances :

From_stop, to_stop → dist_brut

On a aussi :

Route_I → route_type

Donc :

from_stop, to_stop, route_I, route_type → from_stop, to_stop, dist_brut, duration_moy, n_vehicles, route_I, route_type

6 Table BCNF ou 3NF

Nous allons vérifier si les tables de notre projet sont BCNF ou 3NF.

- Pour la table stations nous savons avec la précédente partie que la dépendance:

Stop_I → lat, lon, name

donc stop_I est une clé primaire donc la table est bien de la forme BCNF.
Comme les attributs ne dépendent pas d'une sous-partie de la clé alors c'est de la forme 2NF.
Les attributs ont un lien entre la clé primaire donc c'est 3NF.

- Pour la table route nous savons aussi avec la précédente partie que la dépendance:

route_I → route_name, route_type

donc route_I est une clé primaire, donc la table route est bien de la forme BCNF.
Les attributs ne dépendent pas d'une sous-partie de la clé donc elle est bien de la forme 2NF.
Les attributs dépendent de la clé primaire directement donc la table est 3NF.

- Pour la table combined_projet nous savons que la dépendance :

from_stop, to_stop, route_I, route_type → from_stop, to_stop, dist_brut, duration_moy, n_vehicles, route_I, route_type

donc from_stop, to_stop, route_I, route_type est une clé primaire, donc la table combined_projet est bien de la forme BCNF.
Comme les attributs dépendent d'une sous-partie de la clé alors ce n'est pas de la forme 2NF et donc ce n'est pas de la forme 3NF.

7 Requêtes SQL

Nous allons afficher les requêtes SQL qui nous permettent d'utiliser les fonctionnalités présente dans l'application.

---

1)

```
SELECT DISTINCT name FROM combined_projet, stations WHERE stop_I = combined_projet.from_stop ORDER BY name
```

Cette requête permet d'afficher les stations de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique, elle est liée à la case « tout » dans le menu en utilisant la table combined_projet et stations.

2)

```
SELECT DISTINCT name FROM combined_projet, stations WHERE stop_I = combined_projet.from_stop AND route_type = 3 ORDER BY name
```

Cette requête permet d'afficher les stations de bus de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique, comme dans la base de données les bus ont pour route_type 3 alors dans la requête on cherche toutes les stations de bus à l'aide de la requête

3)

```
SELECT distinct route_name FROM route where route_type = 3 ORDER BY route_name
```

Cette requête permet d'afficher toutes les lignes de bus de la table route dans la case ligne dans l'ordre alphabétique et en enlevant tous les doublons.

4)

```
SELECT DISTINCT name FROM combined_projet, stations WHERE stop_I = combined_projet.from_stop AND route_type = 0 ORDER BY name
```

Cette requête permet d'afficher toutes les stations de tram de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique et en enlevant les doublons, les tram ont pour route_type 0.

5)

```
SELECT distinct route_name FROM route where route_type = 0 ORDER BY route_name
```

Cette requête permet d'afficher toutes les lignes de tram de la table route dans la case ligne dans l'ordre alphabétique et en enlevant les doublons.

6)

```
SELECT DISTINCT name FROM combined_projet, stations WHERE stop_I = combined_projet.from_stop AND route_type = 1 ORDER BY name
```

Cette requête permet d'afficher les stations de métro de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique et en enlevant les doublons, le métro à pour route_type 1

7)

```
SELECT distinct route_name FROM route where route_type = 1 ORDER BY route_name
```

Cette requête permet d'afficher les lignes de métro de la table route dans la case ligne dans l'ordre alphabétique et en enlevant les doublons.

8)

```
SELECT DISTINCT name FROM combined_projet, stations WHERE stop_I = combined_projet.from_stop AND route_type = 2 ORDER BY name
```

Cette requête permet d'afficher toutes les stations de train de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique et en enlevant les doublons, le train à pour route_type 2

9)

```
SELECT distinct route_name FROM route where route_type = 2 ORDER BY route_name
```

Cette requête permet d'afficher les lignes de train de la table route dans la case ligne dans l'ordre alphabétique et en enlevant les doublons.

Nous allons présenter les requêtes SQL pour représenter les chemins possibles du départ « from » vers la destination « to ».

_type = route.route_type

HOP 1 :

10)

```
with metros(geo_point_2d,nom_long,res_com) as (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i   order by stations.name ASC)
    select distinct  A.geo_point_2d,A.nom_long, A.res_com,B.geo_point_2d,  B.nom_long
    from metros as B, metros as A
    where A.nom_long = $${_fromstation}$$ AND B.nom_long =$${_tostation}$$ AND A.res_com = B.res_com
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 1 ligne de transport et en ayant la possibilité d'utiliser tous les moyens de transport disponible.

11)

```
with metros(geo_point_2d,nom_long,res_com) as (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i
and combined_projet.route_type = {_type} and route.route_type = {_type} order by stations.name ASC)
    select distinct  A.geo_point_2d,A.nom_long, A.res_com,B.geo_point_2d,  B.nom_long
    from metros as B, metros as A
    where A.nom_long = $${_fromstation}$$ AND B.nom_long =$${_tostation}$$ AND A.res_com = B.res_com
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 1 ligne de transport et en ayant la possibilité d'utiliser un seul moyen de transport disponible « _type ».

Hop 2 :

12)

```
with metros(geo_point_2d,nom_long,res_com) as   (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i   order by stations.name ASC)
   SELECT distinct A.geo_point_2d,A.nom_long, A.res_com,B.geo_point_2d,  B.nom_long, C.res_com,  D.geo_point_2d,D.nom_long
   FROM metros as A, metros as B, metros as C, metros as D
   WHERE A.nom_long =$${_fromstation}$$  AND D.nom_long =$${_tostation}$$ AND A.res_com = B.res_com AND B.nom_long = C.nom_long
   AND C.res_com = D.res_com AND A.res_com <> C.res_com AND A.nom_long <> B.nom_long AND B.nom_long <> D.nom_long
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 2 lignes de transport au maximum et en ayant la possibilité d'utiliser tous les moyens de transport disponible.

13)

```
with metros(geo_point_2d,nom_long,res_com) as   (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i and combined_projet.route_type = {_type}
and route.route_type = {_type}  order by stations.name ASC)
    SELECT distinct A.geo_point_2d,A.nom_long, A.res_com,B.geo_point_2d,  B.nom_long, C.res_com,  D.geo_point_2d,D.nom_long
    FROM metros as A, metros as B, metros as C, metros as D
    WHERE A.nom_long =$${_fromstation}$$  AND D.nom_long =$${_tostation}$$ AND A.res_com = B.res_com AND B.nom_long = C.nom_long
    AND C.res_com = D.res_com AND A.res_com <> C.res_com AND A.nom_long <> B.nom_long AND B.nom_long <> D.nom_long
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 2 lignes de transport au maximum et en ayant la possibilité d'utiliser un seul moyen de transport disponible « _type ».

HOP 3 :

14)

```
with metros(geo_point_2d,nom_long,res_com) as    (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i   order by stations.name ASC)
    SELECT distinct  A.geo_point_2d,A.nom_long, A.res_com,B1.geo_point_2d,  B2.nom_long, B2.res_com,C2.geo_point_2d,
    C2.nom_long, C2.res_com,D.geo_point_2d,  D.nom_long
    FROM metros as A, metros as B1, metros as B2, metros as C1, metros as C2, metros as D
    WHERE A.nom_long = $${_fromstation}$$  AND A.res_com = B1.res_com AND B1.nom_long = B2.nom_long AND B2.res_com = C1.res_com
    AND C1.nom_long = C2.nom_long AND C2.res_com = D.res_com AND D.nom_long =$${_tostation}$$ AND A.res_com <> B2.res_com
    AND B2.res_com <> C2.res_com AND A.res_com <> C2.res_com AND A.nom_long <> B1.nom_long AND B2.nom_long <> C1.nom_long
    AND C2.nom_long <> D.nom_long
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 3 lignes de transport au maximum et en ayant la possibilité d'utiliser tous les moyens de transport disponible.

15)

```
with metros(geo_point_2d,nom_long,res_com) as    (select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i and combined_projet.route_type = {_type}
and route.route_type = {_type}  order by stations.name ASC)
    SELECT distinct  A.geo_point_2d,A.nom_long, A.res_com,B1.geo_point_2d,  B2.nom_long, B2.res_com,C2.geo_point_2d,
    C2.nom_long, C2.res_com,D.geo_point_2d,  D.nom_long
    FROM metros as A, metros as B1, metros as B2, metros as C1, metros as C2, metros as D
    WHERE A.nom_long = $${_fromstation}$$  AND A.res_com = B1.res_com AND B1.nom_long = B2.nom_long AND B2.res_com = C1.res_com
    AND C1.nom_long = C2.nom_long AND C2.res_com = D.res_com AND D.nom_long =$${_tostation}$$ AND A.res_com <> B2.res_com
    AND B2.res_com <> C2.res_com AND A.res_com <> C2.res_com AND A.nom_long <> B1.nom_long AND B2.nom_long <> C1.nom_long
    AND C2.nom_long <> D.nom_long
```

Cette requête permet d'afficher tous les chemins possible de « _fromstation » à « _tostation » en utilisant 3 lignes de transport au maximum et en ayant la possibilité d'utiliser un seul moyen de transport disponible « _type ».

16)

```
WITH mytable (distance, name) AS (SELECT ( ABS(stations.lat-{lat}) + ABS(stations.lon-{lng}) ), stations.name FROM stations)
    SELECT A.name
    FROM mytable as A
    WHERE A.distance <=  (SELECT min(B.distance) FROM mytable as B)
```

Cette requête permet de rechercher la gare la plus proche selon l'endroit où on a cliqué sur la carte.

17)

```
with metros(geo_point_2d,nom_long,res_com) as
(select distinct CONCAT(stations.lat,',',stations.lon) as ma,stations.name,route.route_name
from stations,route,combined_projet
where stations.stop_i=combined_projet.from_stop and route.route_i=combined_projet.route_i
order by stations.name ASC) SELECT distinct nom_long FROM metros ORDER BY nom_long
```

Cette requête permet d'afficher les stations de la table combined_projet et stations dans les cases from et to dans l'ordre alphabétique, elle est liée à la case « tout » dans le menu en utilisant la table combined_projet et stations.

8 Les difficultés rencontrées au cours du projet

La principale difficulté rencontrée était sur l'utilisation du langage Python car nous avions des connaissances assez restreintes autour du langage de programmation. Avant de commencer à travailler sur le projet, nous avions commencé à comprendre le code utilisé dans les TP précédent pour pouvoir commencer à travailler sur des bonnes bases. Nous avons aussi commencé à travailler sur le parsing des fichiers csv qui était difficile à comprendre, en particulier le parsing du fichier « network_combined.csv » car nous avons eu des problèmes à split l'attribut route_I_count pour obtenir seulement l'attribut route_I.

L'installation des modules sur nos propres machines nous à donner des difficultés c'est pour cela que nous avons utilisé les machines de TP et nous nous sommes connectés sur la base de données PostgreSQL de l'université tout au long de la réalisation du projet.

La mise en place des requêtes SQL a été difficile car les id des stations sont différentes par exemple pour une même station on peut avoir des stations pour le bus et le train qui n'ont pas le même id car des trains peuvent arriver sur des coordonnées géographiques différentes et nous avions eu du mal à trouver un lien entre la base de données du TP en fonction des attributs utilisées dans nos requêtes et la base de données fournies pour le projet.

9 Contribution des membres de l'équipe

Voici les contributions des membres de l'équipe pour notre projet :

Aminata Aliou BA : Pour le projet, j'ai écrit les scripts Python pour le parsing des fichiers « network_combined.csv », « network_nodes.csv » et « network_routes.csv ». Je me suis aussi chargé de la rédaction du rapport du projet, j'ai aidé dans l'affichage des stations sur l'application.

Maurice Kaliva BORE : Pour le projet, j'ai mis en place les principales requêtes SQL pour la recherche de la station la plus proche et des itinéraires (stations, ligne à prendre) avec hop 1, 2 et 3. J'ai aussi aidé dans la mise en place des fonctionnalité de l'application.

Nicolas CALCAR : Pour le projet, j'ai mis en place les fonctionnalité de recherche de toutes les stations qui utilisent une ligne précise, de la recherche en fonction du mode de transport, j'ai travaillé sur l'esthétique de l'application avec la mise en place des check-boxes, des combo-boxes, des box-lay-out et du logo de l'application.

Contribution collective : Nous avons tous participé en réfléchissant sur le design et l'esthétique dans l'ensemble. Nous avons mis en place le diagramme Entity-relationship ensemble.