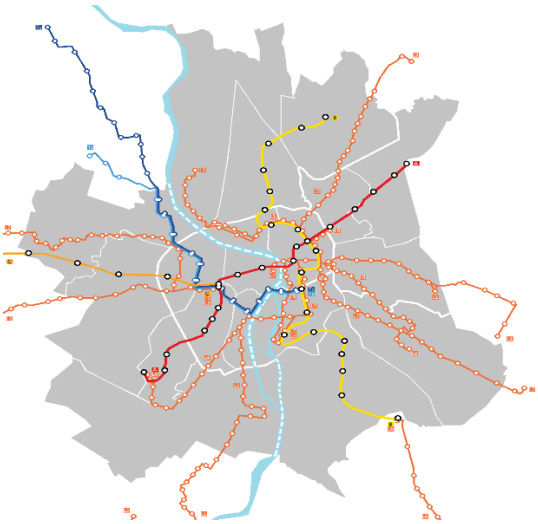
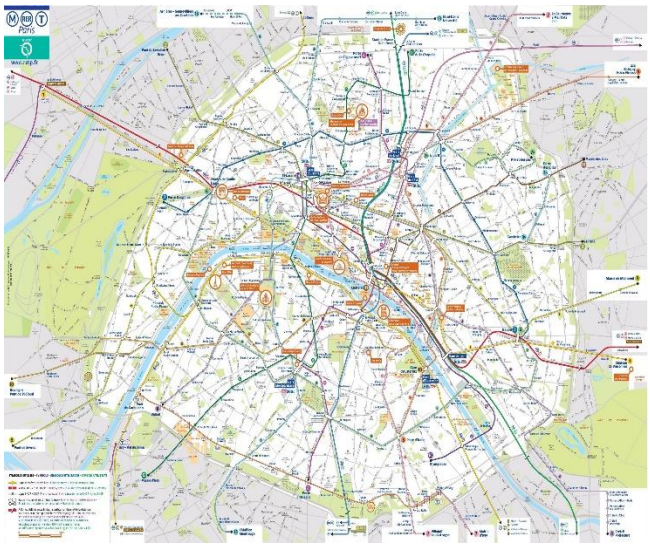
Licence Informatique – 3e année

Rapport de projet: CITYMAPPER





Réalisé par :

Melissa BENKHODJA

Dieunel MARCELIN

Amazigh ALLOUN

Sommaire :

Introduction:

Cet article décrit un effort collaboratif visant à concevoir et mettre en œuvre une application de gestion du trafic urbain pour les villes de Paris et de Toulouse. Ce projet porte sur une interface Python basée sur une base de données PostgreSQL et vise à fournir aux utilisateurs une solution simple d'utilisation pour planifier des déplacements sur les réseaux de transports publics.

Nous commençons par examiner la conception des bases de données, en nous concentrant sur les choix et les processus derrière la création de tables et la gestion des relations. Les sections suivantes décrivent en détail les requêtes SQL les plus importantes et démontrent leur utilisation dans l'utilisation de votre application.

Les défis auxquels vous êtes confrontés, des aspects techniques aux limitations des données, sont résolus grâce aux solutions proposées. Enfin, nous examinons les perspectives d'avenir du projet et les enseignements tirés de cette collaboration.

Cet article donne un aperçu d'un projet d’application collaborative de gestion du trafic urbain, mettant en évidence les décisions techniques, les défis surmontés et les leçons apprises.

1. Objectif du projet :

Le but de notre projet était de concevoir et mettre en œuvre une application, basée sur une interface en Python, dédiée à la gestion des transports dans différentes villes. Pour cela, nous avons utilisé PostgreSQL comme système de gestion de base de données, ainsi que des interfaces Python.

Pour notre application, nous avons conçu deux interfaces distinctes, permettant à l'utilisateur de choisir entre les villes de Paris et Toulouse.

L'application permet à l'utilisateur de :

1. Choisir la ville de son choix entre Paris et Toulouse.

2. Sélectionner le lieu de départ et d'arrivée soit directement sur la carte avec la souris, soit en l'écrivant.

3. Choisir le nombre de correspondances pour atteindre sa destination (3 au maximum).

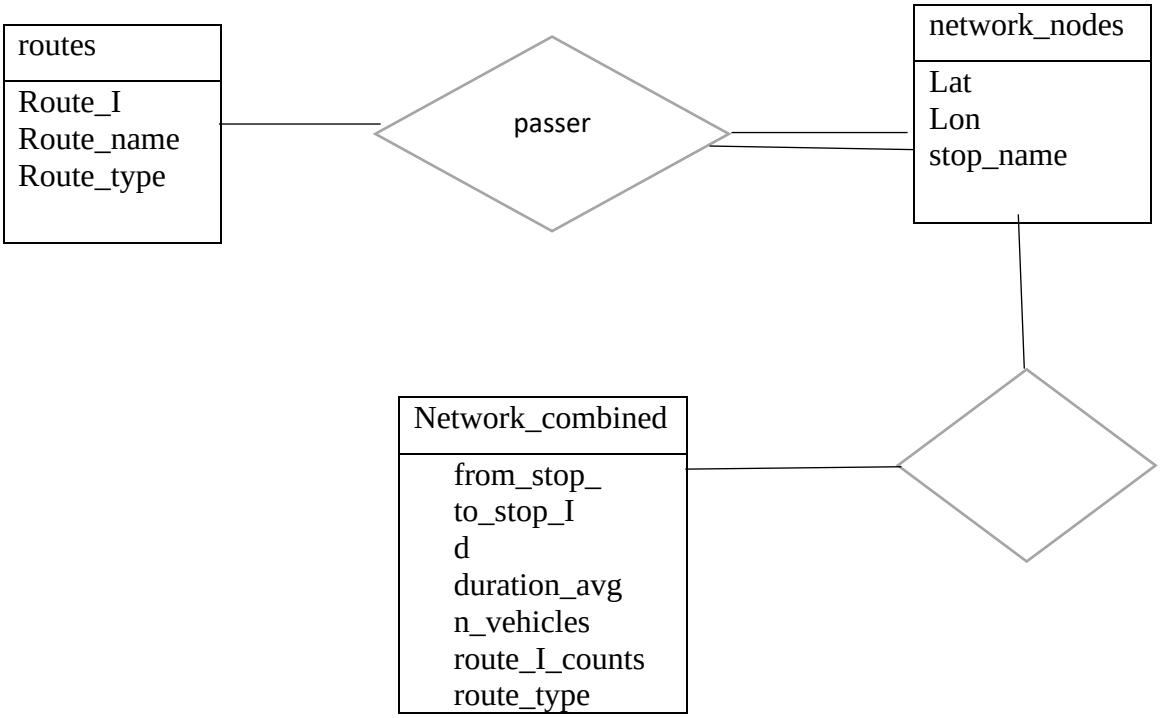
4. Visualiser le temps de trajet estimé.

5. Choisir le moyen de transport(bus,tram …) pour hops=1.

2. Conception de la base de données :

Nous avons recueilli les données à partir du site fourni, lequel contient plus de 25 villes parmi lesquelles nous avons sélectionné Paris et Toulouse. Nous avons téléchargé les fichiers correspondants à chaque ville et créé des tables distinctes pour chaque moyen de transport, notamment network_bus, network_tram, etc. De plus, une table network_combined a été mise en place pour identifier les trajets possibles entre deux stations en utilisant tous les moyens de transport disponibles. La table network_nodes sert à identifier la latitude et la longitude de chaque station. Enfin, une table routes a été créée pour regrouper tous les identifiants, noms et types de transports. Il est important de souligner que ces tables sont communes aux deux villes, Paris et Toulouse.

On a donc eu le diagramme ci-dessous :



Les étapes de création du diagramme sont les suivantes :

1. Création de trois tables principales : network_nodes, network_combined et routes, chacune avec ses propres attributs.
2. Établissement de deux relations .
3. La table network_nodes a la clé primaire de la table stops, ce qui en fait une entité faible.

Et pour créer les tables :

- Table network_nodes

```
CREATE TABLE network_nodes (
    stop_I INTEGER,
    lat NUMERIC(20, 30),
    lon NUMERIC(20, 30),
    stop_name TEXT,
    PRIMARY KEY (stop_I)
);
```

-Table network_combined

```
CREATE TABLE network_combined (
    from_stop_i INTEGER,
    to_stop_i INTEGER,
    d INTEGER,
    duration_avg TEXT,
    n_vehicles INTEGER,
    route_I_counts INTEGER,
    routes_type INTEGER,
    PRIMARY KEY (from_stop_I, to_stop_I, route_I_counts),
    FOREIGN KEY (route_I_counts) REFERENCES routes
);
```

```
-Create table routes (
route_I integer,
route_name text,
route_type integer,
PRIMARY KEY (route_I)
);
```

3. Dépendances et normalisation :

a- Dans la table network_nodes on a la dépendance suivante :

stop_I→ lat, lon, stop_name

b- Dans la table network_combined on a la dépendance suivante :

from_stop_I, to_stop_i, route_I_counts → d, n_vehicles, duration_avg, routes_type

d- Dans la table routes on a la dépendance suivante :

route_I → route_name, route_type

- ***BNCF OU 3NF***

➢ La table network_combined :

from_stop_I, to_stop_I, route_I_counts → d, n_vehicles, duration_avg, routes_type

Notation :R1(from_stop_I, to_stop_I, n, duration_avg , d, n_vehicles, route_I_counts, routes_type)

Considérons F l'ensemble des dépendances fonctionnelles

F = { from_stop_I, to_stop_I, route_I_counts → d, n_vehicles, duration_avg, routes_type}

Nous examinons si R1 satisfait la BCNF:

Calcul de la cloture de from_stop_I, to_stop_I, route_I_counts :

{ from_stop_I, to_stop_I, route_I_counts }+ = { from_stop_I, to_stop_I, route_I_counts}

Comme nous avons la dépendance from_stop_I, to_stop_I, route_I_counts → d, n_vehicles, duration_avg, routes_type,

{ from_stop_I, to_stop_I, route_I_counts }+ = { From_stop_I, to_stop_I, route_I_counts, d, n_vehicles, duration_avg, routes_type}.

Par conséquent, R1 est en BCNF.

➢ En ce qui concerne la table routes :

route_I → route_name, route_type

Soit R2 (Route_I ,route_name, route_type),

Considérons F l'ensemble des dépendances fonctionnelles

{ Route_I → route_name, route_type}

Vérifions si R2 est BCNF :

Calculons la cloture de Route_I: {Route_I}+ = { Route_I}

Vu qu'on a Route_I → route_name, route_type,

{Route_I}+ = { Route_I ,route_name, route_type }

Par conséquent, R2 est BCNF.

➢ La table network_nodes on a :

stop_I → lat, lon, stop_name

Considérons R3 (Stop_I, lat, lon, stop_name)

Considérons F l'ensemble des dépendances fonctionnelles

{stop_I → lat, lon, stop_name}

Vérifions si R3 est BCNF :

Calculons la cloture de stop_I :{stop_I}+ = {stop_I}

{stop_I}+ = {stop_I, lat, lon, stop_name},

Par conséquent, R3 est BCNF.

Et donc, R, R1, R2, R3 sont aussi en troisième forme normale (3NF).

4. Les requêtes SQL

4.1 Requête pour se connecter à la base de données :

Afin de se connecter à la base de donnée créer sur postgreSQL nous avons utilisé la fonction ci-dessous :

```
    def connect_DB(self):
        self.conn = psycopg2.connect(database="citympdb", user="melissa",
host="localhost", password="melissa23")
        self.cursor = self.conn.cursor()
```

4.2 Requête pour ajouter les noms des stations aux boutons from et to :

```
        self.cursor.execute("""SELECT distinct name FROM network_nodes ORDER BY
name""")
        self.conn.commit()
        rows = self.cursor.fetchall()

        for row in rows :
            self.from_box.addItem(str(row[0]))
            self.to_box.addItem(str(row[0]))
```

4.3. Sélectionner le point de départ et d'arrivée :

L'utilisateur a la possibilité de définir son point de départ et d'arrivée de deux manières différentes. Tout d'abord, il peut les spécifier en les écrivant directement, donc une option de saisie manuelle. Mais aussi, l'utilisateur peut choisir ces points en les sélectionnant directement sur la carte.

```
    def mouseClick(self, lat, lng):
        self.webView.addPointMarker(lat, lng)

        print(f"Clicked on: latitude {lat}, longitude {lng}")
        self.cursor.execute(""f" WITH stations AS (SELECT A.name, (A.lat - {lat})
* (A.lat - {lat}) + (A.lon - {lng})*(A.lon - {lng}) AS distance FROM network_nodes
AS A) SELECT T.name FROM (SELECT A.name, (A.lat - {lat}) * (A.lat - {lat}) +
(A.lon - {lng})*(A.lon - {lng}) AS distance FROM network_nodes as A) as T where
T.distance <= ALL (SELECT distance from stations) """)

        self.conn.commit()
        rows = self.cursor.fetchall()
        #print('Closest STATION is: ', rows[0][0])
        if self.startingpoint :
            self.from_box.setCurrentIndex(self.from_box.findText(rows[0][0],
Qt.MatchFixedString))
        else :
            self.to_box.setCurrentIndex(self.to_box.findText(rows[0][0],
Qt.MatchFixedString))
        self.startingpoint = not self.startingpoint
```

4.4 Choix de l'itinéraire par l'utilisateur :

Après avoir sélectionné le point de départ et la destination, l'utilisateur peut choisir le nombre de correspondances qu'il souhaite faire pour son trajet. Pour cela, nous avons ajouté un bouton "hops" qui permet à l'utilisateur de choisir jusqu'à un maximum de 3 correspondances.

1.hops1 :

```
        if _hops >= 1 and _route_type == 'All' :
            self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B.name,
NULL, NULL, NULL, NULL, A.d, A.duration_avg AS d_avg FROM (SELECT * FROM
network_combined,network_nodes,routes_toulouse WHERE network_combined.from_stop_I
= network_nodes.stop_I  AND routes_toulouse.route_I =
network_combined.route_I_counts) AS A, (SELECT * FROM
network_combined,network_nodes,routes_toulouse WHERE network_combined.to_stop_I =
network_nodes.stop_I AND routes_toulouse.route_I =
network_combined.route_I_counts) AS B WHERE A.name= $${_fromstation}$$ AND B.name=
$${_tostation}$$ AND A.route_I_counts = B.route_I_counts ORDER BY d_avg""")
            self.conn.commit()
            self.rows += self.cursor.fetchall()
```

2.hops2 :

```
        if _hops >= 2 and _route_type == 'All':
            self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B.name,
C.route_name,D.name, NULL, NULL, A.d + C.d, A.duration_avg + C.duration_avg as
d_avg FROM (SELECT *FROM network_combined, network_nodes, routes_toulouse WHERE
network_combined.from_stop_I = network_nodes.stop_I AND routes_toulouse.route_I =
network_combined.route_I_counts) AS A, (SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_I =
network_nodes.stop_I AND routes_toulouse.route_I =
network_combined.route_I_counts) AS B, (SELECT * FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_I =
network_nodes.stop_I  AND routes_toulouse.route_I =
network_combined.route_I_counts) AS C, (SELECT * FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.to_stop_I =
network_nodes.stop_I AND routes_toulouse.route_i =
network_combined.route_I_counts ) AS D WHERE A.name = $${_fromstation}$$  AND
D.name = $${_tostation}$$ AND A.route_I_counts = B.route_I_counts AND B.name =
C.name  AND C.route_i_counts = D.route_i_counts AND A.route_i_counts <>
C.route_i_counts  AND A.name <> B.name   AND B.name <> D.name ORDER BY d_avg""")
            self.conn.commit()
            self.rows += self.cursor.fetchall()
```

3.hops3 :

```
        if _hops >= 3 and _route_type == 'All' :
            self.cursor.execute(""f" SELECT distinct A.name, A.route_name,
B2.name, B2.route_name, C2.name, C2.route_name, D.name, A.d + B2.d + C2.d,
A.duration_avg + B2.duration_avg + C2.duration_avg as d_avg FROM (SELECT *FROM
network_combined, network_nodes, routes_toulouse WHERE
network_combined.from_stop_i = network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS A, (SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_i =
network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS B1,(SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_i =
network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS B2,(SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_i =
network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS C1, (SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_i =
network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS C2, (SELECT *FROM network_combined,
network_nodes, routes_toulouse WHERE network_combined.from_stop_i =
network_nodes.stop_i AND routes_toulouse.route_i =
network_combined.route_i_counts) AS D WHERE A.name = $${_fromstation}$$ AND
A.route_i_counts = B1.route_i_counts AND B1.name = B2.name AND B2.route_i_counts =
C1.route_i_counts AND C1.name = C2.name AND C2.route_i_counts = D.route_i_counts
AND D.name = $${_tostation}$$ AND A.route_i_counts <> B2.route_i_counts AND
B2.route_i_counts <> C2.route_i_counts AND A.route_i_counts <> C2.route_i_counts
AND A.name <> B1.name AND B2.name <> C1.name AND C2.name <> D.name ORDER BY
d_avg"""
```


Nous avons également inclus un bouton permettant à l'utilisateur de choisir le moyen de transport de son choix. Cependant, cette fonctionnalité n'est disponible que lorsque le nombre de correspondances (hops) est égal à 1. À cet effet, nous avons ajouté une condition pour chaque type de transport sélectionné.

Par ailleurs, nous avons intégré la fonctionnalité qui affiche la durée entre deux trajets en exploitant les informations stockées dans les tables network_temporal_week et network_temporal_day.

5.Gestion des deux villes :

Pour permettre la gestion de deux villes, Paris et Toulouse, dans notre projet, nous avons mis en place une boîte de dialogue au sein du fichier app.py.

Cette boîte de dialogue offre à l'utilisateur la possibilité de choisir la ville qu'il souhaite. Une fois que l'utilisateur a effectué son choix, il est automatiquement redirigé vers l'interface correspondante à la ville sélectionnée.

Plus en détail, pour chaque ville, nous avons créé des tables spécifiques dans notre base de données. Ces tables conservent la même structure et les mêmes attributs, seules les données qu'elles contiennent changent en fonction de la ville choisie.

```
if __name__ == "__main__":
    app = QApplication([])

    city_selector_dialog = CitySelectorDialog()
    if city_selector_dialog.exec_() == QDialog.Accepted:
        selected_city = city_selector_dialog.city_combobox.currentText()

        if selected_city == "Paris":
            Paris_window = ParisWindow()
            Paris_window.show()
        elif selected_city == "Toulouse":
            toulouse_window = ToulouseWindow()
            toulouse_window.show()

    sys.exit(app.exec_())
```

6.Difficultés et solutions techniques :

Nous avons rencontré quelques difficultés lors de la réalisation de notre projet, telles que :

1. La plupart des membres de l'équipe n'avaient que des connaissances de base en Python. Nous avons donc dû apprendre le langage sur le tas, ce qui a pris du temps et de l'énergie. Nous avons trouvé des ressources en ligne et en bibliothèque, et nous avons également demandé de l'aide à nos professeurs et à d'autres étudiants plus expérimentés.

2. L'insertion des données à partir des fichiers network_temporal_day.csv et network_temporal_day.csv, une tâche qui a pris énormément de temps.

3. L'utilisation des machines des salles de TP à distance qui, la plupart du temps, ne fonctionnaient pas correctement ou ne nous accordaient pas les droits nécessaires pour installer les paquets requis pour la réalisation de ce projet.

4. Les données que nous avons utilisées étaient très volumineuses. Notre espace de stockage local était insuffisant pour les stocker. Nous avons donc dû utiliser un service de stockage en nuage. Cela a ajouté une complexité supplémentaire au projet.

5. Pour la table network_combined, la séparation de route_I_counts et la récupération du premier élément ont posé un problème au début, mais nous avons fini par résoudre cette difficulté.

7. Perspectives et améliorations :

Si nous avions eu un peu plus de temps, voire simplement pour l'avenir, nous envisagerions quelques points :

- Nous aimerions améliorer l'interface graphique de notre projet afin de la rendre plus esthétique et plus intuitive.

- Nous souhaiterions également étendre la prise en charge du moyen de transport pour tous les hops.

- L'ajout d'autres villes à notre projet est également envisagé.

- L'intégration des temps réels des trajets constitue une amélioration que nous aimerions apporter.

- Enfin, la mise en place d'un historique des trajets déjà effectués serait une fonctionnalité pratique pour les utilisateurs. Cela leur permettrait de retrouver facilement les trajets qu'ils ont déjà effectués et de les comparer entre eux.

8.Evaluation individuelle et contributions au projet :

Chaque membre du groupe s'est occupé d'une tâche afin de mener à bien le projet.

Melissa s'est chargée de la mise en place de la base de données destinée à servir le projet. Elle a d'abord défini le modèle de données, puis elle a créé les tables et les relations entre elles. Elle a également développé les différentes interfaces en Python qui permettent d'accéder à la base de données.

Dieunel quant à lui s'est occupé de la mise en place des requêtes SQL. Il a rédigé les requêtes nécessaires à l'extraction des données de la base de données. Ces requêtes ont été optimisées pour être performantes et pour répondre aux besoins du projet.

Et Amazigh a pris en charge la rédaction du code Python nécessaire à l'analyse et au traitement des données dans la base de données, inspiré en partie des scripts vus en TP pour le parsing mais adapté aux besoins et contraintes engendrés par les particularités de ce projet.

Conclusion:

Le projet de développement d'une application de gestion des transports a été une expérience enrichissante et passionnante pour les équipes impliquées. Des défis tels que la complexité des données et les contraintes de temps ont été surmontés grâce à une étroite collaboration entre les membres de l'équipe. Cette collaboration a permis à chacun de partager ses connaissances et ses compétences, ce qui a été essentiel à la réussite du projet.

Les applications développées répondent aux besoins des utilisateurs en matière d'information et de planification de voyages. Les utilisateurs peuvent l'utiliser pour consulter les horaires des transports publics, trouver les itinéraires les plus rapides et les plus rentables et recevoir des notifications en cas de perturbations. Les améliorations de l'application incluent une esthétique améliorée, des fonctionnalités étendues, l'intégration de données en temps réel, et bien plus encore. Améliorer l'esthétique de votre application la rendra plus facile à utiliser et plus attrayante pour les utilisateurs. Les fonctionnalités étendues vous permettent de répondre aux besoins d'un plus large éventail d'utilisateurs. L'intégration des données en temps réel fournit aux utilisateurs des informations plus précises et plus actuelles.