

# **Livrable 1 : Liv'In Paris**

Antoine Groussard, Lucie Grandet, Tristan Limousin

1er Mars 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Modélisation de la Base de Données</b>	<b>4</b>
2.1	Schéma Entité-Association . . . . .	5
2.2	Création de la Base de Données . . . . .	6
2.2.1	Initialisation de la Base de Données . . . . .	6
2.2.2	Création des Rôles . . . . .	6
2.2.3	Création des Stations de Métro . . . . .	6
2.2.4	Création de la Table des Utilisateurs . . . . .	7
2.2.5	Gestion des Rôles des Utilisateurs . . . . .	7
2.2.6	Création de la Table des Clients . . . . .	8
2.2.7	Création de la Table des Cuisiniers . . . . .	8
2.2.8	Création de la Table des Plats . . . . .	9
2.2.9	Création de la Table des Commandes . . . . .	9
2.2.10	Création de la Table des Évaluations . . . . .	10
2.2.11	Création de la Table des Connexions entre Stations de Métro . . . . .	10
2.3	Peuplement de la Base de Données . . . . .	11
2.3.1	Ajout des Rôles . . . . .	11
2.3.2	Ajout des Stations de Métro . . . . .	11
2.3.3	Ajout des Utilisateurs . . . . .	11
2.3.4	Récupération des Identifiants des Utilisateurs . . . . .	12
2.3.5	Attribution des Rôles aux Utilisateurs . . . . .	12
2.3.6	Ajout du Cuisinier et du Client . . . . .	12
2.3.7	Ajout des Plats . . . . .	13
2.3.8	Récupération des Identifiants des Commandes . . . . .	13
2.3.9	Ajout des Commandes . . . . .	14
2.4	Requêtes SQL de Consultation . . . . .	15

2.4.1	Récupération des Utilisateurs et de Leur Station de Métro . . . . .	15
2.4.2	Affichage de Tous les Utilisateurs . . . . .	15
2.4.3	Récupération des Utilisateurs et de Leurs Rôles . . . . .	15
<b>3</b>	<b>Modélisation et Implémentation du Graphe</b>	<b>16</b>
3.1	Mise en Contexte . . . . .	16
3.2	Construction du Graphe . . . . .	16
3.2.1	Définition du Modèle . . . . .	16
3.2.2	Création et Manipulation du Graphe . . . . .	17
3.2.3	Parcours et Analyse du Graphe . . . . .	17
3.3	Visualisation du Graphe avec l'IA . . . . .	17
3.3.1	Objectif de la Visualisation . . . . .	17
3.3.2	Outils et Techniques Utilisées . . . . .	18
3.3.3	Résultats et Interprétation . . . . .	18
3.4	Conclusion . . . . .	18

# Chapter 1

## Introduction

**Liv’In Paris** est une application permettant le partage de repas entre habitants de Paris. Elle repose sur une base de données relationnelle pour gérer les utilisateurs (clients et cuisiniers), les plats proposés, les commandes et les évaluations. Cette structure assure l’intégrité des données et facilite les requêtes et analyses. Pour optimiser les livraisons, l’application utilise une modélisation de graphe représentant le réseau de transport parisien, notamment les stations de métro et leurs connexions. Cela permet de calculer les itinéraires les plus courts entre un cuisinier et un client en utilisant des algorithmes de graphes tels que Dijkstra ou Bellman-Ford. Ce rapport présente la modélisation de la base de données, les scripts SQL associés, ainsi que l’implémentation du graphe en C# pour optimiser les trajets de livraison.

## Chapter 2

# Modélisation de la Base de Données

La base de données de **Liv’In Paris** est conçue pour structurer efficacement les informations relatives aux utilisateurs, aux plats, aux commandes et aux livraisons. Elle repose sur un modèle relationnel garantissant l’intégrité des données et facilitant les requêtes. Ce chapitre détaille le schéma Entité-Association, la structure des tables, ainsi que les scripts SQL permettant la création et le peuplement de la base.

## 2.1 Schéma Entité-Association

Le schéma suivant représente la base de données du projet :

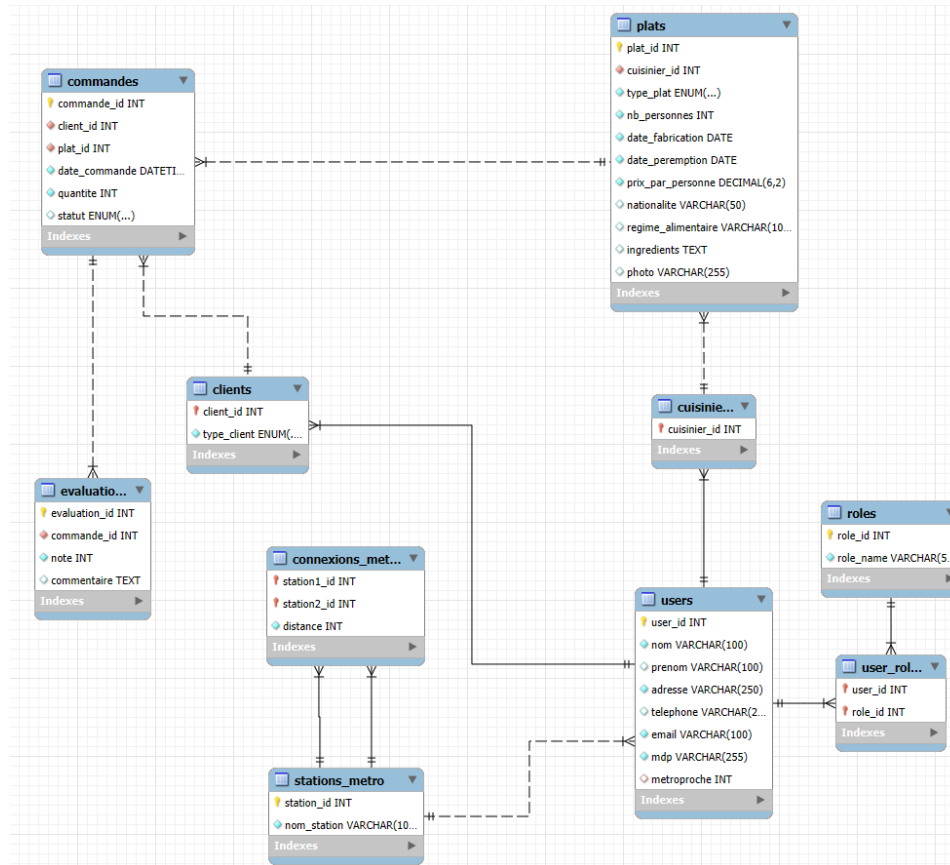


Figure 2.1: Schéma Entité-Association

## 2.2 Création de la Base de Données

### 2.2.1 Initialisation de la Base de Données

Avant de créer les tables, on s'assure que l'ancienne base n'existe plus et on en crée une nouvelle :

```
DROP DATABASE IF EXISTS livinparis;  
CREATE DATABASE livinparis;  
USE livinparis;
```

### 2.2.2 Création des Rôles

La table `roles` définit les différents rôles possibles pour les utilisateurs :

```
CREATE TABLE IF NOT EXISTS roles (  
    role_id INT AUTO_INCREMENT PRIMARY KEY,  
    role_name VARCHAR(50) NOT NULL  
);
```

### 2.2.3 Création des Stations de Métro

La table `stations_metro` enregistre les différentes stations de métro :

```
CREATE TABLE IF NOT EXISTS stations_metro (  
    station_id INT AUTO_INCREMENT PRIMARY KEY,  
    nom_station VARCHAR(100) NOT NULL  
);
```

### 2.2.4 Création de la Table des Utilisateurs

Les utilisateurs peuvent être des clients ou des cuisiniers, et chaque utilisateur est associé à une station de métro proche :

```
CREATE TABLE IF NOT EXISTS users (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    nom VARCHAR(100) NOT NULL,  
    prenom VARCHAR(100),  
    adresse VARCHAR(250) NOT NULL,  
    telephone VARCHAR(20),  
    email VARCHAR(100) NOT NULL UNIQUE,  
    mdp VARCHAR(255) NOT NULL,  
    metroproche INT,  
    FOREIGN KEY (metroproche)  
        REFERENCES stations_metro(station_id)  
        ON DELETE CASCADE  
);
```

### 2.2.5 Gestion des Rôles des Utilisateurs

Les utilisateurs peuvent avoir plusieurs rôles, stockés dans une table de jointure :

```
CREATE TABLE IF NOT EXISTS user_roles (  
    user_id INT NOT NULL,  
    role_id INT NOT NULL,  
    PRIMARY KEY (user_id, role_id),  
    FOREIGN KEY (user_id) REFERENCES users(user_id)  
        ON UPDATE CASCADE ON DELETE CASCADE,  
    FOREIGN KEY (role_id) REFERENCES roles(role_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```



### 2.2.6 Création de la Table des Clients

Les clients peuvent être des particuliers ou des entreprises :

```
CREATE TABLE IF NOT EXISTS clients (  
    client_id INT PRIMARY KEY,  
    type_client ENUM('PARTICULIER','ENTREPRISE')  
        NOT NULL,  
    FOREIGN KEY (client_id) REFERENCES users(user_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

### 2.2.7 Création de la Table des Cuisiniers

Les cuisiniers sont des utilisateurs ayant un rôle spécifique :

```
CREATE TABLE IF NOT EXISTS cuisiniers (  
    cuisinier_id INT PRIMARY KEY,  
    FOREIGN KEY (cuisinier_id)  
        REFERENCES users(user_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

### 2.2.8 Création de la Table des Plats

Les plats sont créés par les cuisiniers et ont plusieurs attributs :

```
CREATE TABLE IF NOT EXISTS plats (  
    plat_id INT AUTO_INCREMENT PRIMARY KEY,  
    cuisinier_id INT NOT NULL,  
    type_plat ENUM('ENTREE','PLAT_PRINCIPAL','DESSERT')  
        NOT NULL,  
    nb_personnes INT NOT NULL,  
    date_fabrication DATE NOT NULL,  
    date_peremption DATE NOT NULL,  
    prix_par_personne DECIMAL(6,2) NOT NULL,  
    nationalite VARCHAR(50),  
    regime_alimentaire VARCHAR(100),  
    ingredients TEXT,  
    photo VARCHAR(255),  
    FOREIGN KEY (cuisinier_id)  
        REFERENCES cuisiniers(cuisinier_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

### 2.2.9 Création de la Table des Commandes

Les commandes sont passées par des clients pour acheter des plats :

```
CREATE TABLE IF NOT EXISTS commandes (  
    commande_id INT AUTO_INCREMENT PRIMARY KEY,  
    client_id INT NOT NULL,  
    plat_id INT NOT NULL,  
    date_commande DATETIME  
        NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    quantite INT NOT NULL,  
    statut ENUM('EN_COURS','LIVRE','ANNULE')  
        DEFAULT 'EN_COURS',  
    FOREIGN KEY (client_id)  
        REFERENCES clients(client_id)  
        ON UPDATE CASCADE ON DELETE CASCADE,  
    FOREIGN KEY (plat_id) REFERENCES plats(plat_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

### 2.2.10 Création de la Table des Évaluations

Les clients peuvent évaluer les plats commandés :

```
CREATE TABLE IF NOT EXISTS evaluations (  
    evaluation_id INT AUTO_INCREMENT PRIMARY KEY,  
    commande_id INT NOT NULL,  
    note INT NOT NULL CHECK (note BETWEEN 1 AND 5),  
    commentaire TEXT,  
    FOREIGN KEY (commande_id)  
        REFERENCES commandes(commande_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

### 2.2.11 Création de la Table des Connexions entre Stations de Métro

Les connexions entre les stations permettent d'établir les trajets optimaux :

```
CREATE TABLE IF NOT EXISTS connexions_metro (  
    station1_id INT NOT NULL,  
    station2_id INT NOT NULL,  
    distance INT NOT NULL,  
    PRIMARY KEY (station1_id, station2_id),  
    FOREIGN KEY (station1_id)  
        REFERENCES stations_metro(station_id)  
        ON UPDATE CASCADE ON DELETE CASCADE,  
    FOREIGN KEY (station2_id)  
        REFERENCES stations_metro(station_id)  
        ON UPDATE CASCADE ON DELETE CASCADE  
);
```

## 2.3 Peuplement de la Base de Données

### 2.3.1 Ajout des Rôles

```
INSERT INTO roles (role_name)
VALUES
    ('CUISINIER'),
    ('CLIENT');
```

### 2.3.2 Ajout des Stations de Métro

```
INSERT INTO stations_metro (nom_station)
VALUES ('Republique'), ('Cardinet');
```

### 2.3.3 Ajout des Utilisateurs

```
INSERT INTO users
    (nom, prenom, adresse, telephone,
     email, mdp, metrop proche)
VALUES
    ('Dupond', 'Marie',
     'Rue de la Republique, 75011 Paris',
     '1234567890', 'Mdupond@gmail.com',
     'hashedmdp',
     (SELECT station_id FROM stations_metro
      WHERE nom_station = 'Republique')),

    ('Durand', 'Medhy', 'Rue Cardinet, 75017 Paris',
     '1234567890', 'Mdurand@gmail.com',
     'hashedmdp',
     (SELECT station_id FROM stations_metro
      WHERE nom_station = 'Cardinet'));
```

### 2.3.4 Récupération des Identifiants des Utilisateurs

```
SET @user_cuisinier =  
    (SELECT user_id FROM users  
     WHERE email = 'Mdupond@gmail.com');  
  
SET @user_client =  
    (SELECT user_id FROM users  
     WHERE email = 'Mdurand@gmail.com');
```

### 2.3.5 Attribution des Rôles aux Utilisateurs

```
INSERT INTO user_roles (user_id, role_id)  
VALUES  
    (@user_cuisinier,  
     (SELECT role_id FROM roles  
      WHERE role_name = 'CUISINIER')),  
  
    (@user_client,  
     (SELECT role_id FROM roles  
      WHERE role_name = 'CLIENT'));
```

### 2.3.6 Ajout du Cuisinier et du Client

```
INSERT INTO cuisiniers (cuisinier_id)  
VALUES (@user_cuisinier);  
  
INSERT INTO clients (client_id, type_client)  
VALUES (@user_client, 'PARTICULIER');
```

### 2.3.7 Ajout des Plats

```
INSERT INTO plats
    (cuisinier_id, type_plat, nb_personnes,
     date_fabrication, date_peremption,
     prix_par_personne, nationalite,
     regime_alimentaire, ingredients, photo)
VALUES
    (@user_cuisinier, 'PLAT_PRINCIPAL', 10,
     '2025-01-10', '2025-01-15',
     6.00, 'Francaise', 'Indifferent',
     'Raclette_250g,Pommes_de_terre_200g,
     Jambon_200g,Cornichon_3p', 'raclette.jpg'),

    (@user_cuisinier, 'DESSERT', 5,
     '2025-01-10', '2025-01-15',
     6.00, 'Indifferent', 'Vegetarien',
     'Fraise_100g,Kiwi_100g,Sucre_10g',
     'salade_fruits.jpg');
```

### 2.3.8 Récupération des Identifiants des Commandes

```
SET @client_id =
    (SELECT client_id FROM clients
     WHERE client_id =
        (SELECT user_id FROM users
         WHERE email = 'Mdurand@gmail.com'));

SET @plat_raclette_id =
    (SELECT plat_id FROM plats
     WHERE ingredients LIKE '%Raclette%');

SET @plat_salade_id =
    (SELECT plat_id FROM plats
     WHERE ingredients LIKE '%Fraise%');
```

### 2.3.9 Ajout des Commandes

```
INSERT INTO commandes
    (client_id, plat_id, date_commande,
     quantite, statut)
VALUES
    (@client_id, @plat_raclette_id, NOW(), 2,
     'EN_COURS'),

    (@client_id, @plat_salade_id, NOW(), 3,
     'EN_COURS');
```

## 2.4 Requêtes SQL de Consultation

Ce chapitre présente quelques requêtes SQL permettant d’extraire des informations de la base de données **Liv’In Paris**. Des requêtes mono tables nous sont demandées, mais étant donné la construction de la base de données, il nous est paru plus judicieux de faire des jointures entre tables pour avoir des requêtes un minimum utiles et pertinents malgré sa faible population.

### 2.4.1 Récupération des Utilisateurs et de Leur Station de Métro

La requête suivante permet d’obtenir la liste des utilisateurs avec la station de métro la plus proche associée :

```
SELECT u.nom, u.prenom, s.nom_station
FROM users u
JOIN stations_metro s
ON u.metroproche = s.station_id;
```

Cette requête effectue une jointure entre la table **users** et la table **stations\_metro** en associant chaque utilisateur à sa station de métro via l’attribut **metroproche**.

### 2.4.2 Affichage de Tous les Utilisateurs

La requête suivante permet d’afficher l’ensemble des utilisateurs enregistrés dans la base de données :

```
SELECT * FROM users;
```

Elle retourne toutes les colonnes de la table **users**, permettant ainsi d’avoir une vue complète des utilisateurs inscrits.

### 2.4.3 Récupération des Utilisateurs et de Leurs Rôles

La requête suivante permet d’afficher les utilisateurs avec leurs rôles respectifs :

```
SELECT u.nom, u.prenom, r.role_name
FROM users u
JOIN user_roles ur ON u.user_id = ur.user_id
JOIN roles r ON ur.role_id = r.role_id;
```

Elle effectue une jointure entre les tables **users**, **user\_roles** et **roles** pour récupérer le rôle attribué à chaque utilisateur.



## Chapter 3

# Modélisation et Implémentation du Graphe

### 3.1 Mise en Contexte

L’optimisation des trajets de livraison est un élément central de l’application **Liv’In Paris**. Pour assurer des livraisons efficaces, un **graphe** est utilisé afin de modéliser le réseau de transport parisien. Ce graphe représente :

- Les **stations de métro** comme des **nœuds**.
- Les **connexions entre stations** comme des **liens**.

Chaque utilisateur (cuisinier ou client) est associé à une station de métro proche, ce qui permet de calculer des trajets optimaux en utilisant des algorithmes de graphes tels que **Dijkstra** ou **Bellman-Ford**. Cette section détaille la construction, l’implémentation et la visualisation du graphe.

### 3.2 Construction du Graphe

#### 3.2.1 Définition du Modèle

Le graphe est structuré à l’aide de deux représentations :

- Une **matrice d’adjacence** pour gérer les connexions entre les stations.
- Une **liste d’adjacence** pour un parcours plus rapide des voisins de chaque nœud.

Chaque station est un **nœud** identifié par un **ID unique**, et chaque connexion est un **lien** reliant deux stations.

### 3.2.2 Création et Manipulation du Graphe

Le graphe est construit à partir d'un fichier contenant les connexions entre les stations. Les étapes principales sont :

1. Lecture des données et création des objets **Noeud** et **Lien**.
2. Remplissage de la **matrice d'adjacence** et de la **liste d'adjacence**.
3. Ajout de méthodes pour manipuler le graphe :
  - Ajouter un **nouveau lien** entre deux stations.
  - Afficher la **matrice d'adjacence** et la **liste des connexions**.

### 3.2.3 Parcours et Analyse du Graphe

Deux algorithmes classiques sont implémentés :

- **Parcours en largeur (BFS)** : utilisé pour trouver les **chemins les plus courts**.
- **Parcours en profondeur (DFS)** : utilisé pour vérifier la **connectivité** et détecter les **cycles**.

Ces algorithmes permettent de répondre aux questions suivantes :

- Existe-t-il un chemin entre deux stations ?
- Le graphe est-il **connexe** ?
- Le graphe contient-il des **cycles** ?

## 3.3 Visualisation du Graphe avec l'IA

### 3.3.1 Objectif de la Visualisation

La visualisation du graphe permet :

- De **mieux comprendre** les connexions entre les stations.
- D'identifier les **trajets optimaux** entre cuisiniers et clients.
- De vérifier l'**efficacité des algorithmes** appliqués au graphe.

### 3.3.2 Outils et Techniques Utilisées

Pour représenter le graphe de manière interactive :

- La bibliothèque **SkiaSharp** est utilisée pour générer un affichage graphique.
- Une **IA générative** permet d'améliorer la visualisation des connexions et trajets.

Les stations sont représentées sous forme de **cercles** et les connexions sous forme de **lignes**. Des couleurs sont utilisées pour distinguer :

- Les **stations initiales** (points de départ des trajets).
- Les **destinations** (points d'arrivée).
- Les **trajets optimisés** pour la livraison.

### 3.3.3 Résultats et Interprétation

Grâce à cette visualisation :

- Les trajets optimaux sont identifiables en un coup d'œil.
- Les chemins inutiles peuvent être supprimés pour améliorer l'efficacité des livraisons.
- Les performances du graphe peuvent être analysées et améliorées.

## 3.4 Conclusion

La modélisation du graphe permet d'optimiser les trajets et d'améliorer l'expérience utilisateur de **Liv'In Paris**. Les prochaines étapes consisteront à tester le système sur un ensemble plus vaste de stations et à intégrer d'autres paramètres comme les conditions de circulation pour encore affiner les trajets optimaux.