Livrable 1 : Liv'In Paris

Antoine Groussard, Lucie Grandet, Tristan Limousin

1er Mars 2025

Contents

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 :

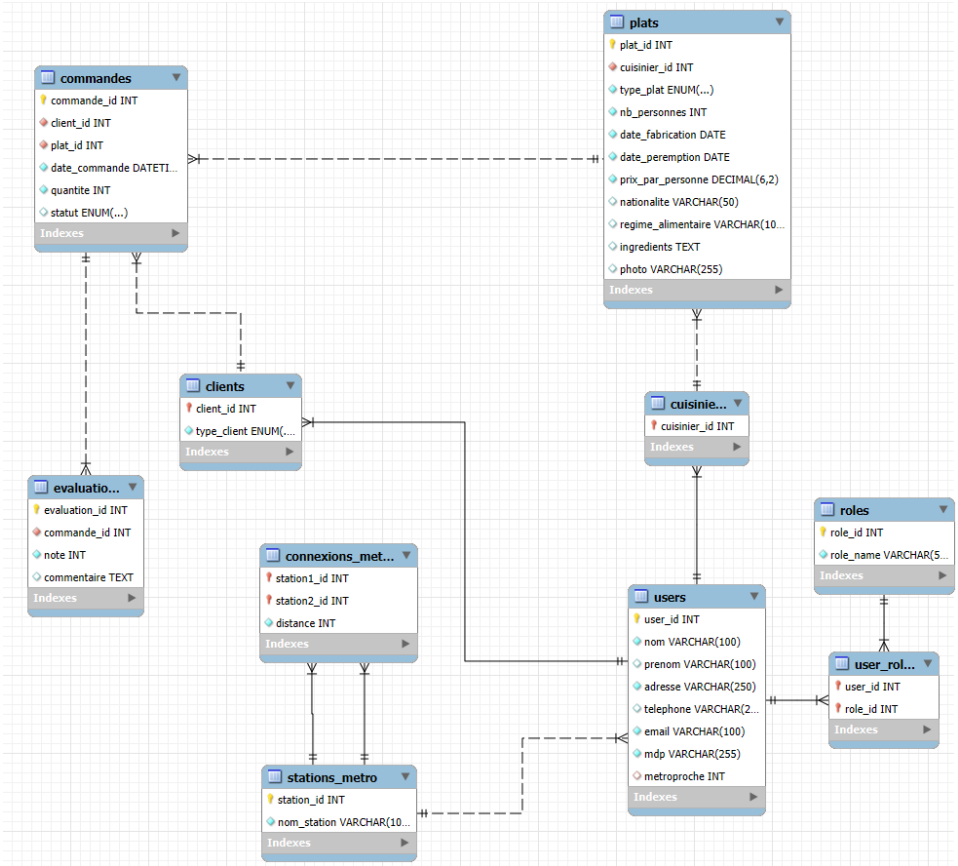

Figure 2.1: Schéma Entité-Association

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, metroproche)
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 requetages 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

Dans le cadre de ce projet, 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, et l'étude de l'exemple fournis.

3.2 Construction du Graphe

3.2.1 Structure générale

Ce graphe est structuré de tel sorte :

- 34 nœuds identifiés de 1 à 34.
- Matrice d'adjacence de taille 35×35 (indexation à partir de 1, l'index 0 est inutilisé).

- Ce Graphe est non orienté. Les relations sont bidirectionnelles.

Bien que la matrice soit dimensionnée pour accueillir des ID jusqu'à 34, l'index 0 reste inutilisé. La symétrie de la matrice confirme le caractère non orienté du graphe, où chaque relation entre nœuds est bidirectionnelle.

3.2.2 Création et Manipulation du Graphe

Les classes Noeud, Lien et Graphe constituent le socle de la modélisation. Chaque Noeud est identifié par un entier unique (Id) et contient une liste d'adjacence recensant ses voisins directs. La méthode AjouterVoisin garantit des connexions bidirectionnelles, reflétant la réciprocité des relations. La classe Lien formalise explicitement une arête entre deux nœuds, bien que cette abstraction soit optionnelle dans un graphe non orienté. Enfin, la classe Graphe agrège l'ensemble des nœuds et utilise une matrice d'adjacence (symétrique) ainsi que des listes d'adjacence pour une double représentation des connexions. Cette dualité permet d'optimiser les opérations : les listes facilitent l'accès rapide aux voisins, tandis que la matrice offre une vue globale des relations.

3.2.3 Parcours et Analyse du Graphe

Deux algorithmes de parcours ont été implémentés :

Le parcours en largeur (BFS) explore les nœuds niveau par niveau. Exécuté depuis le nœud 1, il a visité tous les nœuds dans l'ordre 1,2,3,4,. . . ,34, confirmant l'accessibilité complète du graphe.

Le parcours en profondeur (DFS) a révélé un ordre de visite différent (26,25,32,. . . ,2), mettant en évidence des chemins imbriqués.

La méthode EstConnexe() a validé que le graphe est entièrement connexe : aucun nœud n'est isolé. Par ailleurs, la détection de cycles via Contient-Cycle() a identifié au moins une boucle, comme le cycle $2 \to 1 \to 3 \to 2$. Cette détection repose sur un marquage tricolore (Blanc pour les nœuds non visités, Jaune pour ceux en cours de traitement, Rouge pour les traités), une approche classique en DFS pour repérer les retours vers des nœuds actifs.

3.3 Visualisation du Graphe à l'aide de l'IA

3.3.1 Première Tentative avec ChatGPT

Pour implémenter la visualisation du graphe, nous avons initialement sollicité ChatGPT, qui a proposé une solution basée sur **SkiaSharp**, une bibliothèque graphique moderne. Bien que cette approche semblait prometteuse, elle s'est rapidement heurtée à des limites pratiques. La syntaxe suggérée, notamment l'utilisation de déconstructions modernes (`var (id, pos)`), s'averrant incompatible avec la version de C# utilisée dans notre projet (.NET Framework 4.7.2).

Par ailleurs, l'absence de connexion explicite entre le contrôle graphique (`Panel`) et la logique de rendu empêchait tout affichage, même avec un code structurellement correct. Enfin, l'intégration de SkiaSharp nécessitait des configurations supplémentaires (ajout de packages NuGet, gestion de fenêtres spécifiques) qui complexifiaient inutilement l'architecture.

3.3.2 Corrections Clés avec DeepSeek

Face à ces défis, DeepSeek a apporté des solutions pragmatiques et adaptées à nos besoins spécifiques, en privilégiant System.Drawing.

- Résolution des Erreurs de Syntaxe :

La déconstruction moderne des KeyValuePair, proposée par ChatGPT, générait des erreurs de compilation. DeepSeek a remplacé cette approche par une boucle classique.

```
foreach (KeyValuePair<int, Point> kvp in _positions)
{
    int id = kvp.Key;
    Point pos = kvp.Value;
    g.FillEllipse(brush, pos.X - radius, pos.Y - radius, 2 * radius, 2 * ra
}
```

- Optimisation Spatiale et Centrage Dynamique

Pour résoudre le problème des nœuds mal centrés, DeepSeek a revu le calcul des positions en intégrant une marge dynamique et un rayon adaptatif :

```
private void CalculateNodePositions(int width, int height)
{
    int margin = 70;
```

```
    int effectiveRadius = (Math.Min(width, height) - margin) / 2;
    // ...
}
```

- Amélioration de la Lisibilité :

Les IDs des nœuds, initialement désalignés, ont été centrés grâce à 'StringFormat' :

```
StringFormat format = new StringFormat()
{
    Alignment = StringAlignment.Center,
    LineAlignment = StringAlignment.Center
};
g.DrawString(id.ToString(), font, brush, rect, format);
```

3.3.3 Résultats Final

Grâce aux corrections apportées par DeepSeek, la visualisation du graphe est désormais satisfaisant. L'application affiche les nœuds de manière claire et harmonieuse, répartis sur un cercle. Les liens entre les nœuds, mettent en valeur la structure du graphe sans surcharger l'affichage.

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.

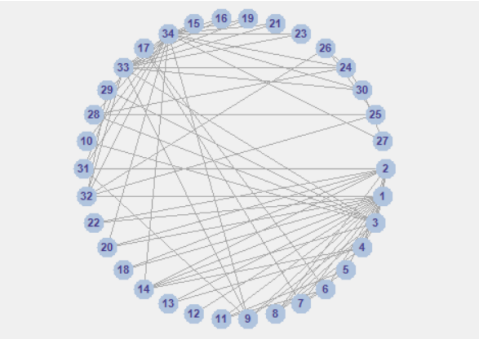

Figure 3.1: Visualisation du Graphe