# Document de Synthèse

**Lien Git** : https://github.com/miangaly-rafiri/EDBB

## 1. Schéma de la Base de Données:

### Entités et Relations :

- Categories : Contient les catégories de produits.
  - id (INT, PK, Auto-incrémenté)
  - nom (VARCHAR(100), Unique)

- Fournisseurs : Contient les informations des fournisseurs.
  - id (INT, PK, Auto-incrémenté)
  - nom (VARCHAR(100))
  - prenom (VARCHAR(100))
  - adresse (VARCHAR(100))
  - numero (VARCHAR(100))

- Produits : Contient les informations des produits.
  - id (INT, PK, Auto-incrémenté)
  - nom (VARCHAR(100))
  - prix_unitaire (FLOAT)
  - quantite_stock (INT)
  - categorie_id (INT, FK vers Categories)
  - fournisseur_id (INT, FK vers Fournisseurs)

- Clients : Contient les informations des clients.
  - id (INT, PK, Auto-incrémenté)
  - nom (VARCHAR(100))
  - prenom (VARCHAR(100))
  - adresse (VARCHAR(100))
  - numero (VARCHAR(100))

- Commandes : Contient les informations des commandes.
  - id (INT, PK, Auto-incrémenté)
  - client_id (INT, FK vers Clients)
  - date_commande (DATETIME)
  - statut (ENUM("En cours", "Livrée"))

- Lignes_Commande : Contient les lignes de commande associées à chaque commande.
  - id (INT, PK, Auto-incrémenté)
  - commande_id (INT, FK vers Commandes)
  - produit_id (INT, FK vers Produits)
  - quantite (INT)
  - prix_unitaire_applique (FLOAT)

Relations :

- Un produit appartient à une catégorie (Produits.categorie_id → Categories.id).
- Un produit est fourni par un fournisseur (Produits.fournisseur_id → Fournisseurs.id).
- Une commande est passée par un client (Commandes.client_id → Clients.id).
- Une ligne de commande est associée à une commande (Lignes_Commande.commande_id → Commandes.id).
- Une ligne de commande concerne un produit (Lignes_Commande.produit_id → Produits.id).

## 2. Liste des Endpoints de l'API

**Produits :**

- GET /produits : Récupérer tous les produits.
  - Exemple de réponse JSON :

```
[
  {
    "id": 1,
    "nom": "Produit A",
    "prix_unitaire": 10.99,
    "quantite_stock": 100,
    "categorie_id": 1,
    "fournisseur_id": 1
  }
]
```

- POST /produits : Ajoute un nouveau produit.
  - Paramètres :

```
{
  "nom": "Produit B",
  "prix_unitaire": 15.99,
  "quantite_stock": 50,
  "categorie_id": 2,
  "fournisseur_id": 2
}
```

- PUT /produits/:id : Met à jour un produit existant.
    - Paramètres :

```
{
  "nom": "Produit B",
  "prix_unitaire": 15.99,
  "quantite_stock": 50,
  "categorie_id": 2,
  "fournisseur_id": 2
}
```

- DELETE /produits/:id : Supprime un produit.

**Clients :**
- GET /clients : Récupère tous les clients.
- POST /clients : Ajoute un nouveau client.
- PUT /clients/:id : Met à jour un client existant.
- DELETE /clients/:id : Supprime un client.

**Commandes :**
- GET /commandes : Récupère toutes les commandes.
- POST /commandes : Ajoute une nouvelle commande.
- PUT /commandes/:id : Met à jour une commande existante.
- DELETE /commandes/:id : Supprime une commande.

**Categories :**
- GET /categories : Récupère toutes les catégories.
- POST /categories : Ajouté une nouvelle catégorie.
- PUT /categories/:id : Met à jour une catégorie existante.
- DELETE /categories/:id : Supprime une catégorie.

**Fournisseurs :**
- GET /fournisseurs : Récupérer tous les fournisseurs.
- POST /fournisseurs : Ajoute un nouveau fournisseur.
- PUT /fournisseurs/:id : Met à jour un fournisseur existant.
- DELETE /fournisseurs/:id : Supprime un fournisseur.

**Lignes_Commande :**
- GET /lignes_commande : Récupérer toutes les lignes de commande.
- POST /lignes_commande : Ajouté une nouvelle ligne de commande.
- PUT /lignes_commande/:id : Met à jour une ligne de commande existante.
- DELETE /lignes_commande/:id : Supprime une ligne de commande.

## 3. Audit V1 et Améliorations en V2:

***Problèmes détectés en V1 :***

1. Injection SQL : Les requêtes SQL étaient construites en concaténant des chaînes de caractères, ce qui rendait l'application vulnérable aux injections SQL.
2. Absence de vérifications métier : Aucune vérification n'était effectuée pour s'assurer de la cohérence des données (ex. : stock suffisant, produit existant dans le catalogue).
3. Requêtes non paramétrées : Les requêtes SQL étaient directement injectées dans la chaîne de requête, ce qui posait des problèmes de sécurité et de maintenance.

***Solutions adoptées en V2 :***

1. Requêtes paramétrées : Toutes les requêtes SQL ont été modifiées pour utiliser des paramètres, ce qui empêche les injections SQL.
    - Exemple en V1 :

```
const query = `INSERT INTO Produits (nom, description, prix) VALUES ('${produit.nom}', '${produit.description}', ${produit.prix})`;
```

***Exemple en V2 :***

```
const query = `INSERT INTO Produits (nom, description, prix) VALUES (?, ?, ?)`;
const [results] = await connection.query(query, [produit.nom, produit.description, produit.prix]);
```

1. Vérifications métier : Des vérifications ont été ajoutées pour s'assurer de la cohérence des données.
    - Exemple : Avant d'ajouter une commande, vérifier que le produit existe et que le stock est suffisant.
    - Exemple : Avant de supprimer un produit, vérifier qu'il n'est pas utilisé dans une commande.

2. Amélioration des endpoints : De nouveaux endpoints ont été ajoutés pour permettre des recherches plus complexes (ex. : recherche de commandes par client, date, statut, etc.).

- Exemple :

```
app.get('/commandes/recherche', async (req, res) => {
    const { client_id, start, end, statut, produit_id } = req.query;
    let query = `SELECT * FROM Commandes c JOIN Lignes_Commande lc
ON lc.commande_id = c.id WHERE 1=1`;
    let queryParams = [];
    if (client_id) {
        query += ` AND c.client_id = ?`;
        queryParams.push(client_id);
    }
    // ...
});
```

3. Gestion des stocks : Un endpoint a été ajouté pour récupérer les produits dont le stock est faible.

Exemple :

```
app.get('/produits/stock-faible', async (req, res) => {
    const { seuil } = req.query;
    const query = `SELECT * FROM Produits WHERE stock < ?`;
    const [results] = await connection.query(query, [seuil]);
    res.send(results);
});
```

# Conclusion :

La version 2 de l'API apporte des **améliorations significatives en termes de sécurité, de robustesse et de fonctionnalités**. Les requêtes paramétrées et les vérifications métier ont permis de sécuriser l'application et de garantir l'intégrité des données. De nouveaux endpoints ont également été **ajoutés pour répondre à des besoins plus complexes, comme la recherche de commandes ou la gestion des stocks.**