DOCUMENTATION TECHNIQUE DU CODE POUR NOTRE PROJET WEB

Groupe N*8

Membres du Groupe

- MBOCK Eliza Lolita (chef de graoupe)
- NTSAM Alain
- NASSER Ismael
- GUEMTUE Étienne Varnel
- MAVOUNGOU Smith
- ESSOMBA Aloys

Sommaire

Introduction

Cette documentation technique a pour objectif de fournir une vue d'ensemble complète du projet, en détaillant ses aspects architecturaux, ses choix technologiques, et les étapes nécessaires pour son installation et son utilisation. Elle est structurée en quatre parties principales :

La première partie qui est la description de l'architecture du projet présentes les technologies utilisées pour le backend, le frontend et la base de données, tout en expliquant les raisons qui nous ont poussé a faire de tel choix. Elle aborde également la composition des modules et services du projet, ainsi que la structure des API pour la communication entre les différentes couches.

La deuxième partie étant le Guide d'installation et d'utilisation expliques les étapes pour installer et configurer le projet, aussi bien en local qu'en environnement de production.

La troisième partie étant les Schémas de la base de données fournit une représentation graphique de la structure de la base de données, décrivant les tables, les relations entre elles, ainsi que les données indispensables au bon fonctionnement du projet.

La dernière partie étant la charte graphique décrivant la police d’écriture, la palette de couleur utilisé et l’ensemble des styles et des éléments graphiques présent dans la page.

Objectifs de ce Document

- **Clarifier l'Architecture du Projet**
Expliquer les choix technologiques et la structure générale de l'application.

- **Guider l'Installation**
Fournir des instructions faciles à suivre pour installer et configurer le projet.

- **Visualiser les Données**
Présenter des schémas de la base de données pour comprendre les relations entre les tables.

- **Assurer la Cohérence Visuelle**
Définir les polices et les couleurs pour garantir une présentation uniforme de l'application.

- **Faciliter le Développement**
Servir de référence pour les développeurs afin de soutenir l'ajout de nouvelles fonctionnalités.

Description de l'architecture fonctionnelle

Dans cette première partie, nous explorerons l'architecture fonctionnelle du projet, en mettant en lumière les choix technologiques qui ont été adoptés. Nous détaillerons les différentes couches de l'application, les modules qui la composent, ainsi que les services interconnectés.

Choix des technologies

Technologie Backend : PHP

Dans le cadre de notre projet "Mon Miam-Miam", nous avons opté pour PHP comme technologie backend. Ce choix stratégique repose sur plusieurs facteurs clés qui garantissent à la fois la robustesse et l'efficacité de notre application. Voici les raisons qui ont motivé cette décision :

- **La simplicité** : le langage PHP est un langage facile a apprendre, a installer et a appréhender. De plus il contient il existe de nombreuses ressources pour les débutants.
- **Intégration avec les Bases de Données** : PHP est très efficace pour interagir avec des bases de données, notamment MySQL, ce qui en fait un choix courant pour les applications web. Il propose des extensions comme PDO (PHP Data Object) pour une gestion sécurisée et flexible des bases de données.
- **Support de la Programmation Orientée Objet** : PHP supporte la programmation orientée objet, ce qui permet d'organiser le code de manière modulaire et de réutiliser les composants.
- **Communauté Active** : PHP dispose d'une communauté active qui fournit une assistance, des tutoriels et des mises à jour régulières. Cela facilite la résolution des problèmes et le partage de connaissances.
- **Hébergement Abordable** : Les serveurs web qui supportent PHP sont largement disponibles et souvent moins coûteux que d'autres solutions, rendant le déploiement d'applications PHP économique.
- **Sécurité :** Bien que la sécurité dépende largement de la manière dont le code est écrit, PHP offre des fonctionnalités pour aider à sécuriser les applications, telles que la protection contre les injections SQL et la gestion des sessions.

Technologies Frontend

REACTJS

Au cours de ce projet, nous avons opté pour React.js comme technologie principale pour le développement de notre application. React.js, une bibliothèque JavaScript développée par Facebook, est reconnue pour sa capacité à créer des interfaces utilisateur dynamiques et réactives. Utiliser React pour le développement d'applications web présente de nombreux avantages tels que :

- **Composants Réutilisables** : React repose sur une architecture basée sur des composants, permettant de créer des morceaux de code réutilisables. Cela facilite la gestion et la maintenance du code, ainsi que la collaboration entre les développeurs.

- **Performance Élevée** : Grâce à son Virtual DOM, React optimise les mises à jour de l'interface utilisateur en réduisant les opérations coûteuses de manipulation du DOM réel. Cela améliore la performance des applications, en rendant l'expérience utilisateur plus fluide.

- **Unidirectionnel des Données** : React suit un flux de données unidirectionnel, ce qui facilite le suivi et la gestion des données dans l'application. Ce modèle aide à rendre le code plus prévisible et à réduire les erreurs.

- **Écosystème Riche** : React dispose d'un écosystème vaste, comprenant des bibliothèques et des outils comme React Router pour la gestion des routes, Redux pour la gestion de l'état, et bien d'autres. Cela permet d'intégrer facilement des fonctionnalités avancées.

- **Communauté Active** : Avec une communauté active et dynamique, React bénéficie de nombreux supports, ressources, et mises à jour régulières. Les développeurs peuvent facilement trouver des solutions à leurs problèmes ou des exemples de code.

- **Adaptabilité** : React peut être utilisé pour développer une variété d'applications, allant des simples applications web aux applications mobiles via React Native, ce qui permet de partager des logiques de code entre différentes plateformes.

- **Facilité d'Apprentissage** : Sa syntaxe basée sur le JavaScript moderne et sa documentation claire facilitent l'apprentissage pour les développeurs, même pour ceux qui n'ont pas une expérience approfondie en développement web.

- **Tests Efficaces** : React est conçu de manière à faciliter les tests, permettant d'écrire des tests unitaires et d'intégration pour les composants, ce qui contribue à la qualité et à la stabilité de l'application.

Cependant, comme toute technologie, React.js présente certains inconvénients. Parmi ceux-ci, on peut noter la courbe d'apprentissage initiale, notamment pour les développeurs qui découvrent le concept des composants et de l'état (state management). De plus, la rapidité d'évolution de la bibliothèque peut rendre certaines fonctionnalités obsolètes ou dépréciées, nécessitant une mise à jour régulière des compétences des développeurs. Enfin, le référencement SEO peut être un défi, car le contenu généré dynamiquement peut ne pas être indexé correctement par les moteurs de recherche.

Néanmoins, ces défauts sont souvent comblés par des solutions adaptées. Par exemple, l'utilisation de bibliothèques complémentaires telles que Redux ou Context API pour la gestion de l'état permet de simplifier le flux de données au sein de l'application. De plus, des outils comme Vite peuvent être intégrés pour améliorer le référencement et la performance des applications Vite, en offrant un rendu côté serveur. Grâce à ces ajustements, nous avons pu tirer parti des points forts de React.js tout en atténuant ses limitations, ce qui a grandement contribué à la réussite de notre projet.

VITE

Vite est un bundler de développement JavaScript moderne et léger, conçu pour offrir une expérience de développement rapide et efficace. Il utilise des techniques avancées telles que le chargement à la demande (code splitting) et l'importation de modules ES pour réduire les temps de démarrage et améliorer la réactivité pendant le développement. Vite propose également un rechargement à chaud (hot module replacement) instantané, ce qui permet aux développeurs de voir les changements en temps réel sans avoir à recharger manuellement la page. Grâce à sa configuration minimale et à son intégration avec des Framework populaires comme Vue.js et React, Vite est devenu un choix prisé pour les projets modernes de développement web. Voici quelques raisons de privilégier Vite :

1. **Temps de Démarrage Ultra-Rapide** : Vite utilise un serveur de développement basé sur des modules ES natifs, ce qui réduit considérablement le temps de démarrage de l'application. Cela permet aux développeurs de commencer à travailler presque instantanément.
2. **Rechargement à Chaud Amélioré** : Vite offre un rechargement à chaud extrêmement rapide. Lorsqu'un fichier est modifié, seul ce fichier est recompilé et rechargé, contrairement à Webpack qui peut nécessiter une recompilation de l'ensemble du projet. Cela améliore considérablement l'expérience de développement.
3. **Configuration Minimale** : Vite nécessite très peu de configuration initiale pour démarrer un projet. Les configurations par défaut sont souvent suffisantes pour la plupart des projets, ce qui permet de se concentrer sur le développement plutôt que sur la configuration.
4. **Prise en Charge des Modules ES** : Vite utilise les modules ES natifs dans le navigateur, ce qui permet un chargement plus efficace des dépendances. Cela simplifie la gestion des modules et réduit le besoin de transpilation lors du développement.
5. **Optimisation des Builds de Production** : Vite utilise Rollup en arrière-plan pour les builds de production, offrant des optimisations avancées et un bundling efficace. Cela garantit que les applications React sont légères et performantes en production.
6. **Support de TypeScript et JSX** : Vite prend en charge TypeScript et JSX par défaut, simplifiant la configuration et permettant aux développeurs de tirer parti des fonctionnalités modernes de JavaScript sans effort supplémentaire.
7. **Plugins Riches** : Vite dispose d'un écosystème de plugins en pleine expansion qui permet d'ajouter facilement des fonctionnalités supplémentaires, telles que le support de CSS, les transformations de code, et bien plus.
8. **Expérience de Développement Améliorée** : Avec ses fonctionnalités comme l'autocomplétion rapide, les suggestions de code et le support intégré des fichiers statiques, Vite offre une expérience de développement plus fluide et agréable.
9. **Communauté Croissante** : Bien que Vite soit relativement nouveau, il bénéficie d'une communauté active et d'un support continu, ce qui assure son évolution et sa pertinence dans le paysage des outils de développement.

BOOSTRAP

Bootstrap est un Framework frontend populaire conçu pour faciliter le développement d'interfaces web réactives et esthétiques. Créé par Twitter, il offre une vaste bibliothèque de composants préconçus, des grilles flexibles et des styles CSS prédéfinis, permettant aux développeurs de créer rapidement des sites web modernes et fonctionnels. Grâce à sa structure modulaire et à sa compatibilité multi-navigateurs, Bootstrap simplifie la conception

d'applications web, garantissant une expérience utilisateur cohérente sur divers appareils et résolutions d'écran. Que vous soyez débutant ou développeur expérimenté, Bootstrap vous aide à gagner du temps tout en améliorant la qualité visuelle et fonctionnelle de vos projets web.

- **Gain de Temps** : Bootstrap propose une multitude de composants prêts à l'emploi, tels que des boutons, des formulaires, des modal et des barres de navigation. Cela permet de créer rapidement des interfaces sans avoir à partir de zéro.
- **Responsive Design** : Le Framework est conçu pour être réactif dès le départ, ce qui signifie que les applications et sites créés avec Bootstrap s'adaptent automatiquement à différentes tailles d'écran, garantissant ainsi une expérience utilisateur optimale sur les mobiles, tablettes et ordinateurs.
- **Grille Flexible** : Bootstrap offre un système de grille puissant qui facilite la mise en page d'éléments sur une page. Ce système permet de créer des mises en page complexes tout en maintenant une structure claire et intuitive.
- **Consistance Visuelle** : En utilisant Bootstrap, les développeurs s'assurent que les éléments de l'interface utilisateur sont cohérents et harmonisés, ce qui améliore la convivialité et l'esthétique générale du site.
- **Personnalisation Facile** : Bien que Bootstrap offre des styles par défaut, il permet également une personnalisation facile. Les développeurs peuvent adapter les couleurs, les polices et les styles selon les besoins spécifiques de leur projet, tout en conservant la structure de base.
- **Large Écosystème** : Bootstrap bénéficie d'une communauté active qui propose de nombreux thèmes, plugins et extensions. Cela permet d'enrichir les projets avec des fonctionnalités supplémentaires sans trop d'efforts.
- **Documentation Complète** : Bootstrap est accompagné d'une documentation détaillée et bien structurée, ce qui facilite l'apprentissage et l'utilisation du Framework, même pour les débutants.
- **Compatibilité Multi-Navigateurs** : Bootstrap est testé sur les principaux navigateurs, garantissant que les applications s'affichent correctement et fonctionnent de manière homogène, quel que soit le navigateur utilisé par l'utilisateur.
- **Support des Normes** : En respectant les standards web, Bootstrap aide les développeurs à créer des sites conformes aux meilleures pratiques en matière d'accessibilité et de compatibilité.
- **Adoption Générale** : Étant l'un des Framework front-end les plus populaires, Bootstrap est largement adopté dans l'industrie, ce qui signifie qu'il est facile de trouver des ressources, des exemples de code et une assistance communautaire.

Technologie pour la Base de Données : MariaDB/MySql

Le choix d'un système de gestion de base de données (SGBD) est crucial pour assurer la performance, la scalabilité et la fiabilité d'une application. Dans cette optique, MariaDB a été sélectionné pour plusieurs raisons. Tout d'abord, en tant que fork de MySQL, MariaDB offre une compatibilité élevée avec les applications existantes utilisant MySQL, facilitant ainsi la migration. De plus, il propose des fonctionnalités avancées, telles que des performances améliorées, une sécurité renforcée et un support pour les requêtes complexes.

En outre, MariaDB bénéficie d'une communauté active qui contribue à son développement constant et à l'ajout de nouvelles fonctionnalités. Sa capacité à gérer de grandes quantités de

données tout en maintenant une rapidité d'accès et de traitement en fait un choix idéal pour notre projet. Cette section détaillera les avantages spécifiques de MariaDB et comment il répond aux exigences de notre application. L'utilisation de MariaDB ou MySQL pour la gestion de bases de données dans un projet présente plusieurs avantages :

- **Open Source**
  MariaDB et MySQL sont des systèmes de gestion de bases de données relationnelles (SGBDR) open source, ce qui permet un accès libre au code source. Cela favorise la transparence, la communauté de développeurs et la possibilité d'adapter le logiciel aux besoins spécifiques.
- **Performance et Scalabilité**
  Ces bases de données sont conçues pour offrir de bonnes performances, même avec de grandes quantités de données. Elles sont capables de gérer un nombre élevé de connexions simultanées, ce qui est essentiel pour les applications à fort trafic.
- **Fiabilité et Robustesse**
  MariaDB et MySQL sont connus pour leur stabilité et leur fiabilité. Ils incluent des mécanismes de sauvegarde et de restauration, ainsi que des fonctionnalités de gestion des transactions qui garantissent l'intégrité des données.
- **Facilité d'Utilisation**
  Avec une documentation exhaustive et une large communauté, il est facile pour les développeurs de trouver des ressources et du support. De plus, des outils comme phpMyAdmin simplifient la gestion et l'administration des bases de données.
- **Compatibilité**
  MariaDB est conçu comme un fork de MySQL, ce qui signifie qu'il est largement compatible avec les applications et outils conçus pour MySQL. Cela facilite la migration ou l'intégration de solutions existantes.
- **Fonctionnalités Avancées**
  MariaDB propose des fonctionnalités avancées telles que des moteurs de stockage variés, des optimisations de requêtes, et des capacités de réplication. MySQL, quant à lui, continue d’évoluer avec des mises à jour régulières ajoutant de nouvelles fonctionnalités.
- **Support pour les Transactions**
  Les deux systèmes prennent en charge les transactions ACID (Atomicité, Cohérence, Isolation, Durabilité), ce qui est crucial pour garantir la fiabilité des opérations critiques sur les données.
- **Sécurité**
  MariaDB et MySQL offrent des fonctionnalités de sécurité robustes, telles que l'authentification par mot de passe et le chiffrement des données, ce qui est essentiel pour protéger les informations sensibles.

Modules et Services

Cette partie aborde les différents modules et services qui composent le projet, en mettant l'accent sur leurs fonctionnalités et leurs interactions.

Fonctionnalités des Modules

MODULES

- **Module Utilisateur** : Gère l'authentification et la gestion des profils utilisateurs. Il permet aux utilisateurs de s'inscrire, de se connecter, et de modifier leurs informations personnelles.
- **Module Menu** : Responsable de l'affichage et de la gestion des produits disponibles. Il inclut des fonctionnalités comme l'ajout, la modification, et la suppression de produits, ainsi que la recherche et le filtrage.
- **Module Commandes** : Gère le processus de commande, y compris la création de nouvelles commandes, le suivi des statuts, et la gestion des paiements.
- **Module Panier** : Permet aux utilisateurs d'ajouter des produits à leur panier, de modifier les quantités, et de visualiser le total avant de finaliser leur achat.
- **Module Administration** : Offrant des fonctionnalités de gestion pour les administrateurs, il permet la supervision des utilisateurs, des commandes, et des produits, ainsi que la génération de rapports.
- **Module Parrainage** : Ce module permet aux utilisateurs de parrainer d'autres personnes et de bénéficier de récompenses (réductions, points, etc.) en fonction du nombre de parrainages réussis. Génération de liens de parrainage gestion des récompenses, Tableau de bord des parrainages et Suivi des conversions
- **Module Employé** : Ce module est conçu pour la gestion des employés au sein d'une entreprise. Il peut inclure des fonctionnalités pour suivre les tâches, les horaires, et la performance des employés.
- **Module Réclamation** : Ce module permet aux utilisateurs de déposer des réclamations ou des plaintes concernant les produits ou services, avec un suivi des réponses et des résolutions. Soumission de réclamations, suivi des réclamations, assignation d'un employé et historique.
- **Module Paiement :** Permet de gérer les paiements via différentes méthodes (carte bancaire, Mobile Money, etc.) avec un suivi des transactions pour chaque utilisateur. Choix du mode de paiement, historique des transactions, intégration avec des passerelles de paiement
- **Module Promotions & Evènements :** Permet de gérer des campagnes promotionnelles. Création de codes promotionnels et suivi des utilisations.
- **Module Points de Fidélité :** Permet de récompenser les utilisateurs en leur attribuant des points de fidélité à chaque achat. Accumulation de points**,** utilisation des points, Tableau de bord de fidélité

Interaction entre les modules

MODULE UTILISATEUR

- **Interaction avec le Module Menu** : Lorsqu'un utilisateur se connecte ou s'inscrit, il peut accéder au menu pour consulter et ajouter des produits au panier.
- **Interaction avec le Module Commandes** : L'utilisateur peut consulter ses commandes passées et suivre leur statut après s'être authentifié.
- **Interaction avec le Module Panier** : Seul un utilisateur authentifié peut créer, gérer ou modifier un panier.
- **Interaction avec le Module Administration** : Les administrateurs ont accès à la gestion des utilisateurs et peuvent consulter, suspendre ou supprimer des comptes.

- **Interaction avec le Module Parrainage** : Chaque utilisateur a un lien de parrainage unique qu'il peut partager. Les récompenses dépendent des inscriptions réussies à travers ce lien.
- **Interaction avec le Module Réclamation** : Un utilisateur peut soumettre une réclamation sur un produit ou une commande depuis son compte.
- **Interaction avec le Module Points de Fidélité** : Les points de fidélité accumulés sont liés au profil de l'utilisateur et à ses commandes passées.

MODULE MENU

- **Interaction avec le Module Utilisateur** : Les utilisateurs peuvent parcourir les produits, ajouter des produits au panier, et appliquer des filtres personnalisés selon leurs préférences.
- **Interaction avec le Module Commandes** : Les produits sélectionnés dans le menu sont directement transférés dans une commande une fois l'achat validé.
- **Interaction avec le Module Administration** : Les administrateurs peuvent ajouter, modifier ou supprimer des produits dans le menu.
- **Interaction avec le Module Panier** : Les utilisateurs ajoutent des produits du menu dans leur panier avant de passer commande.
- **Interaction avec le Module Promotions & Evènements** : Des promotions peuvent s'appliquer directement aux produits visibles dans le menu.
- **Interaction avec le Module Points de Fidélité** : Les produits achetés depuis le menu rapportent des points de fidélité après validation de la commande.

MODULE COMMANDES

- **Interaction avec le Module Utilisateur** : Les utilisateurs passent des commandes depuis leur compte, où ils peuvent suivre leur progression.
- **Interaction avec le Module Panier** : Une commande est créée à partir des articles présents dans le panier de l'utilisateur.
- **Interaction avec le Module Paiement** : La commande est validée et suivie après un paiement réussi.
- **Interaction avec le Module Administration** : Les administrateurs peuvent consulter et gérer les commandes (expédition, statut, remboursements).
- **Interaction avec le Module Réclamation** : Les utilisateurs peuvent déposer des réclamations liées à une commande spécifique.
- **Interaction avec le Module Points de Fidélité** : Une commande terminée permet d'accumuler des points de fidélité pour l'utilisateur.
- **Interaction avec le Module Promotions & Evènements** : Les commandes peuvent inclure des réductions si des promotions sont appliquées

MODULE PANIER

- **Interaction avec le Module Utilisateur** : Un utilisateur peut ajouter, modifier ou supprimer des produits de son panier avant de passer une commande.
- **Interaction avec le Module Menu** : Les produits du menu sont ajoutés au panier selon la sélection de l'utilisateur.
- **Interaction avec le Module Commandes** : Lorsque l'utilisateur valide son panier, une commande est créée à partir des articles du panier.

- **Interaction avec le Module Paiement** : Après validation du panier, l'utilisateur passe au paiement pour finaliser la commande.

MODULE ADMINISTRATION

- **Interaction avec le Module Utilisateur** : Les administrateurs peuvent superviser les utilisateurs (création, suppression, gestion des permissions).
- **Interaction avec le Module Menu** : Gestion et mise à jour des produits visibles dans le menu (ajout, suppression, modification).
- **Interaction avec le Module Commandes** : Les administrateurs peuvent superviser et gérer les commandes des utilisateurs.
- **Interaction avec le Module Employé** : Gestion des tâches, horaires et performances des employés depuis l'interface d'administration.
- **Interaction avec le Module Réclamation** : Les administrateurs peuvent gérer et assigner des réclamations aux employés appropriés pour résolution.
- **Interaction avec le Module Promotions & Evènements** : Les administrateurs peuvent créer et superviser des promotions qui apparaîtront dans le module Menu.
- **Interaction avec le Module Points de Fidélité** : Gestion des barèmes de points de fidélité et des récompenses offertes aux utilisateurs.

MODULE PARRAINAGE

- **Interaction avec le Module Utilisateur** : Les utilisateurs peuvent partager leur lien de parrainage et recevoir des récompenses pour chaque parrainage réussi.
- **Interaction avec le Module Points de Fidélité** : Des points de fidélité peuvent être attribués en fonction des parrainages réussis.
- **Interaction avec le Module Administration** : Les administrateurs peuvent superviser et gérer les parrainages (vérification, récompenses).

MODULE EMPLOYÉ

- **Interaction avec le Module Administration** : Les administrateurs peuvent assigner des tâches aux employés et suivre leur progression et performance.
- **Interaction avec le Module Commandes** : Les employés peuvent être assignés pour gérer l'expédition ou la préparation des commandes.
- **Interaction avec le Module Réclamation** : Les employés peuvent être assignés pour résoudre les réclamations déposées par les utilisateurs

MODULE RÉCLAMATION

- **Interaction avec le Module Utilisateur** : Les utilisateurs peuvent soumettre des réclamations liées aux produits ou services.
- **Interaction avec le Module Commandes** : Les réclamations peuvent être liées à des commandes spécifiques.
- **Interaction avec le Module Administration** : Les administrateurs peuvent consulter les réclamations et les assigner aux employés.
- **Interaction avec le Module Employé** : Les réclamations assignées sont traitées par les employés responsables.

MODULE PAIEMENT

- **Interaction avec le Module Commandes** : Les commandes ne sont validées qu'après un paiement réussi.
- **Interaction avec le Module Panier** : Une fois le panier validé, l'utilisateur est redirigé vers la page de paiement pour finaliser son achat.
- **Interaction avec le Module Administration** : Les administrateurs peuvent suivre les transactions et résoudre les problèmes liés aux paiements.
- **Interaction avec le Module Promotions & Evènements** : Les promotions appliquées dans le panier peuvent affecter le montant total à payer.

MODULE PROMOTIONS & EVÈNEMENTS

- **Interaction avec le Module Utilisateur** : Les utilisateurs peuvent appliquer des codes promotionnels ou bénéficier d'évènements de réduction sur leurs achats.
- **Interaction avec le Module Menu** : Les produits du menu peuvent bénéficier de promotions spéciales.
- **Interaction avec le Module Commandes** : Les promotions appliquées lors du processus de commande affectent le total à payer.
- **Interaction avec le Module Panier** : Les utilisateurs peuvent appliquer des promotions avant de finaliser leur commande.
- **Interaction avec le Module Points de Fidélité** : Certaines promotions peuvent offrir des points de fidélité supplémentaires.

MODULE POINTS DE FIDÉLITÉ

- **Interaction avec le Module Utilisateur** : Les utilisateurs accumulent des points de fidélité en fonction de leurs achats et parrainages.
- **Interaction avec le Module Commandes** : Chaque commande réussie permet d'accumuler des points de fidélité.
- **Interaction avec le Module Panier** : Les utilisateurs peuvent choisir d'utiliser leurs points de fidélité pour bénéficier de réductions sur leurs achats.
- **Interaction avec le Module Administration** : Les administrateurs peuvent superviser les points accumulés et les règles de fidélité.

Structures des API

La structure d'une API repose sur plusieurs aspects importants, notamment les points d'entrée (endpoints), les méthodes et réponses API, et la gestion des erreurs. Ces éléments définissent comment les clients interagissent avec l'API pour envoyer et recevoir des données, et comment les erreurs sont traitées en cas de problèmes.

STRUCTURE DE L'API DU CODE ETUDIANT.PHP

Points d'entrée (endpoints)

1. **GET /etudiants**
   - Description : Récupérer la liste de tous les étudiants.
   - Méthode : `GET`
   - Réponse : Liste des étudiants.
   - Autorisation : Seuls les administrateurs peuvent accéder à cet endpoint.

2. **POST /etudiants/{id}**
    - Description : Mettre à jour les informations d'un étudiant spécifique (nom, prénom, photo).
    - Méthode : `POST`
    - Paramètres :
        - `id` (int) : Identifiant de l'étudiant.
        - `nom_etudiant` (string) : Nom de l'étudiant.
        - `prenom_etudiant` (string) : Prénom de l'étudiant.
        - `photo` (fichier) : Photo de l'étudiant (optionnel).
    - Réponse : Confirmation de la mise à jour.
    - Autorisation : Seuls les administrateurs peuvent effectuer cette opération.

3. **DELETE /etudiants/{id}**
    - Description : Supprimer un étudiant spécifique.
    - Méthode : `DELETE`
    - Paramètres :
        - `id` (int) : Identifiant de l'étudiant.
    - Réponse : Confirmation de la suppression.
    - Autorisation : Seuls les administrateurs peuvent effectuer cette opération.

Méthodes et réponses API

1. **GET /etudiants**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "data": [
    {"id": 1, "nom": "Dupont", "prenom": "Jean", "photo":
"uploads/dupont.jpg"},
    {"id": 2, "nom": "Durand", "prenom": "Alice", "photo":
"uploads/durand.jpg"}
  ]
}
```

    - **Réponse en cas d'erreur (500 Internal Server Error)** :

```
{
  "success": false,
  "message": "Erreur lors de la récupération des clients."
}
```

2. **POST /etudiants/{id}**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Étudiant mis à jour avec succès."
}
```

    - **Réponse en cas d'erreur (500 Internal Server Error)** :

```
{
  "success": false,
  "message": "Erreur lors de la mise à jour de l'étudiant."
```

```
}
```

3. **DELETE /etudiants/{id}**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Étudiant supprimé avec succès."
}
```

   - **Réponse en cas d'erreur (500 Internal Server Error)** :

```
{
  "success": false,
  "message": "Erreur lors de la suppression de l'étudiant."
}
```

Gestion des erreurs

Les erreurs peuvent se produire à plusieurs niveaux :

1. **403 Forbidden** : Si l'utilisateur n'a pas le bon rôle (seuls les administrateurs peuvent accéder à ces endpoints).
   - Réponse :

```
{
  "success": false,
  "message": "Accès refusé."
}
```

2. **500 Internal Server Error** : Si une erreur survient lors de l'exécution des requêtes SQL (par exemple, échec de la récupération des données, de la mise à jour, ou de la suppression).
   - Réponse :

```
{

  "success": false,
  "message": "Erreur lors de [l'action concernée]."
}
```

STRUCTURE DE EDITEREMPLOYER.PHP

Points d'entrée (endpoints)

1. **POST /employes/{id}**
   - Description : Mettre à jour les informations d'un employé spécifique (nom, prénom, et photo en option).
   - Méthode : `POST`
   - Paramètres :
     - `id` (int) : Identifiant de l'employé (obligatoire).
     - `nom_employe` (string) : Nom de l'employé (optionnel).
     - `prenom_employe` (string) : Prénom de l'employé (optionnel).
     - `photo` (fichier) : Photo de l'employé (optionnel).
   - Réponse : Confirmation de la mise à jour.

Méthodes et réponses API

1. **POST /employes/{id}**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Employé mis à jour."
}
```

   - **Réponse en cas d'erreur de validation (400 Bad Request)** :

```
{
  "success": false,
  "message": "ID d'employé manquant."
}
```

   - **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "success": false,
  "message": "Erreur de connexion : [message d'erreur PDO]."
}
```

Gestion des erreurs

1. **400 Bad Request** : Lorsque l'ID de l'employé est manquant ou invalide.
   - Réponse :

```
{
  "success": false,
  "message": "ID d'employé manquant."
}
```

2. **500 Internal Server Error** : Si une erreur survient lors de la connexion à la base de données ou de l'exécution de la requête SQL.
   - Réponse :

```
{
  "success": false,
  "message": "Erreur de connexion : [message d'erreur PDO]."
}
```

STRUCTURES EMPLOYE.PHP

Points d'entrée (Endpoints)

1. **GET /employee.php**
   - Description : Récupérer la liste des employés avec le rôle "Employer".
   - Méthode : GET
   - Réponse : Retourne un tableau contenant tous les employés avec le rôle "Employer".

2. **POST /employee.php**
   - Description : Ajouter un nouvel employé avec un rôle "Employer".
   - Méthode : POST
   - Paramètres :
     - `email` (string) : L'email de l'employé (obligatoire).

        - `password` (string) : Le mot de passe de l'employé (obligatoire).
    - Réponse : Confirmation de l'ajout de l'employé.

3. **PUT /employee.php**
    - Description : Modifier les informations d'un employé existant (email et mot de passe).
    - Méthode : `PUT`
    - Paramètres :
        - `id` (int) : L'ID de l'employé (obligatoire).
        - `email` (string) : L'email de l'employé (obligatoire).
        - `password` (string) : Le mot de passe (facultatif).
    - Réponse : Confirmation de la modification de l'employé.

4. **DELETE /employee.php**
    - Description : Supprimer un employé en fonction de son ID.
    - Méthode : `DELETE`
    - Paramètres :
        - `id` (int) : L'ID de l'employé à supprimer (obligatoire).
    - Réponse : Confirmation de la suppression de l'employé.

2. Méthodes et réponses API

1. GET /employee.php

- **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id": 1,
    "email": "employe@example.com",
    "role": "Employer"
  },
  {
    "id": 2,
    "email": "autre@example.com",
    "role": "Employer"
  }
]
```

- **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "error": "Erreur lors de l'exécution de la requête"
}
```

2. POST /employee.php

- **Corps de la requête** :

```
{
  "email": "nouveau@example.com",
  "password": "motdepasse"
}
```

- **Réponse en cas de succès (201 Created)** :

```
{
  "success": "Employé ajouté avec succès"
}
```

- **Réponse en cas de données manquantes (400 Bad Request)** :

```
{
  "error": "Données manquantes pour l'ajout"
}
```

- **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "error": "Erreur lors de l'ajout de l'employé"
}
```

3. PUT /employee.php

- **Corps de la requête** :

```
{
  "id": 1,
  "email": "modifie@example.com",
  "password": "nouveaumotdepasse"
}
```

- **Réponse en cas de succès (200 OK)** :

```
{
  "success": "Employé modifié avec succès"
}
```

- **Réponse en cas de données manquantes (400 Bad Request)** :

```
{
  "error": "ID ou données manquantes pour la modification"
}
```

- **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "error": "Erreur lors de la modification de l'employé"
}
```

4. DELETE /employee.php

- **Corps de la requête** (paramètres envoyés via `php://input`) :

```
{
  "id": 1
}
```

- **Réponse en cas de succès (200 OK)** :

```
{
  "success": "Employé supprimé avec succès"
}
```

- **Réponse en cas d'employé non trouvé (404 Not Found)** :

```
{
  "error": "Employé non trouvé"
}
```

- **Réponse en cas de données manquantes (400 Bad Request)** :

```
{
  "error": "ID manquant pour la suppression"
}
```

- **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "error": "Erreur lors de la suppression de l'employé"
}
```

STRUCTURE DE L'API EFFACEREMPLOYER.PHP

2.3 Structure des API

Points d'entrée (endpoints)

POST /employes/delete

- Description : Supprimer un employé de la base de données via son identifiant.
- Méthode : `POST`
- Paramètres :
  - `id` (int) : Identifiant de l'employé à supprimer (obligatoire).
- Réponse : Confirmation de la suppression.

Méthodes et réponses API

POST /employes/delete

- **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Employé supprimé."
}
```

- **Réponse en cas d'erreur de validation (400 Bad Request)** :

```
{
  "success": false,
  "message": "ID d'employé manquant."
}
```

- **Réponse en cas d'erreur de serveur (500 Internal Server Error)** :

```
{
  "success": false,
  "message": "Erreur de connexion : [message d'erreur PDO]."
}
```

Gestion des erreurs

1. **400 Bad Request** : Lorsque l'ID de l'employé est manquant ou invalide.
   - Réponse :

```
{
  "success": false,
  "message": "ID d'employé manquant."
}
```

2. **500 Internal Server Error** : Si une erreur survient lors de la connexion à la base de données ou de l'exécution de la requête SQL.
   - Réponse :

```
{
  "success": false,
  "message": "Erreur de connexion : [message d'erreur PDO]."
}
```

STRUCTURE API MENUADMIN.PHP

Point d'entrée principal : `/produits`

1. **GET /produits**
   - Description : Récupérer tous les produits disponibles dans la base de données.
   - Méthode : `GET`
   - Paramètres : Aucun
   - Réponse :
   -

```
{
  "success": true,
  "produits": [
    {
      "id": 1,
      "title": "Produit A",
      "description": "Description du produit A",
      "imgSrc": "imageA.jpg"
    },
    {
      "id": 2,
      "title": "Produit B",
      "description": "Description du produit B",
      "imgSrc": "imageB.jpg"
    }
  ]
}
```

2. **POST /produits**
   - **Description** : Ajouter un nouveau produit dans la base de données.
   - **Méthode** : `POST`

- **Paramètres** (via formulaire) :
  - `title` (string) : Le nom du produit (obligatoire)
  - `description` (string) : La description du produit (obligatoire)
  - imgSrc (fichier image) : Image associée au produit (obligatoire)
- **Réponse en cas de succès** :

```
{
  "success": true,
  "produit": {
    "id": 3,
    "title": "Produit C",
    "description": "Description du produit C",
    "imgSrc": "imageC.jpg"
  }
}
```

- **Réponse en cas d'erreur de validation** :

```
{
  "success": false,
  "message": "Données manquantes"
}
```

3. **PUT /produits/{id}**
   - **Description** : Mettre à jour les informations d'un produit spécifique.
   - **Méthode** : `PUT`
   - **Paramètres** (via body `x-www-form-urlencoded`) :
     - `title` (string) : Le nom du produit (obligatoire)
     - `description` (string) : La description du produit (obligatoire)
   - **Réponse en cas de succès** :

```
{
  "success": true
}
```

   - **Réponse en cas d'erreur de validation** :

```
{
  "success": false,
  "message": "Données manquantes"
}
```

   - **Réponse en cas d'erreur serveur** :

```
{
  "success": false,
  "message": "Erreur lors de la mise à jour du produit:
[message d'erreur]"
}
```

4. **DELETE /produits/{id}**
   - **Description** : Supprimer un produit de la base de données.
   - **Méthode** : `DELETE`
   - **Paramètres** :
     - `id` (int) : L'identifiant du produit (obligatoire, via URL)

- **Réponse en cas de succès** :

```
{
  "success": true
}
```

- **Réponse en cas d'erreur serveur** :

```
{
  "success": false,
  "message": "Erreur lors de la suppression du produit:
[message d'erreur]"
}
```

STRUCTURE FONCTIONPANIER.PHP

2.3 Structure des API

Points d'entrée (endpoints)

1. **POST /commandes**
   - Description : Créer une nouvelle commande.
   - Méthode : `POST`
   - Paramètres :
     - `id_etudiant` (int) : Identifiant de l'étudiant (obligatoire).
     - `statut` (string) : Statut de la commande (obligatoire).
     - `total` (float) : Montant total de la commande (obligatoire).
     - `type_service` (string) : Type de service (obligatoire).
     - `commentaire` (string) : Commentaire pour la commande (optionnel).
     - `points_utilises` (int) : Points utilisés (optionnel).
   - Réponse : ID de la nouvelle commande.

2. **GET /commandes/{id}**
   - Description : Obtenir les informations d'une commande par son ID.
   - Méthode : `GET`
   - Paramètres :
     - `id` (int) : Identifiant de la commande (obligatoire).
   - Réponse : Détails de la commande.

3. **PUT /commandes/{id}/statut**
   - Description : Mettre à jour le statut d'une commande.
   - Méthode : `PUT`
   - Paramètres :
     - `id` (int) : Identifiant de la commande (obligatoire).
     - `statut` (string) : Nouveau statut de la commande (obligatoire).
   - Réponse : Confirmation de la mise à jour.

4. **POST /commandes/{id}/details**
   - Description : Ajouter un détail à une commande.
   - Méthode : `POST`
   - Paramètres :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).
     - `id_menu` (int) : Identifiant du produit du menu (obligatoire).
     - `quantité` (int) : Quantité commandée (obligatoire).

        - `sous_total` (float) : Montant partiel (obligatoire).
    - Réponse : Confirmation de l'ajout.

5. **GET /commandes/{id}/details**
    - Description : Obtenir les détails d'une commande.
    - Méthode : `GET`
    - Paramètres :
        - `id` (int) : Identifiant de la commande (obligatoire).
    - Réponse : Détails des produits de la commande.

6. **GET /produits/{id}**
    - Description : Obtenir un produit du menu par son ID.
    - Méthode : `GET`
    - Paramètres :
        - `id` (int) : Identifiant du produit du menu (obligatoire).
    - Réponse : Détails du produit.

7. **GET /produits**
    - Description : Obtenir tous les produits du menu.
    - Méthode : `GET`
    - Réponse : Liste des produits.

8. **PUT /commandes/{id}/total**
    - Description : Mettre à jour le total d'une commande.
    - Méthode : `PUT`
    - Paramètres :
        - `id` (int) : Identifiant de la commande (obligatoire).
        - `total` (float) : Nouveau montant total (obligatoire).
    - Réponse : Confirmation de la mise à jour.

9. **GET /panier/{id_etudiant}**
    - Description : Obtenir le panier d'un étudiant.
    - Méthode : `GET`
    - Paramètres :
        - `id_etudiant` (int) : Identifiant de l'étudiant (obligatoire).
    - Réponse : Contenu du panier.

Méthodes et réponses API

1. **POST /commandes**
    - **Réponse en cas de succès (201 Created)** :

```
{
  "id_commande": 123
}
```

    - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
  "error": "Échec de l'insertion de la commande."
}
```

2. **GET /commandes/{id}**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "id": 123,
  "id_etudiant": 1,
  "statut": "En attente",
  "total": 29.99,
  "type_service": "Livraison",
  "commentaire": "Aucune",
  "points_utilises": 0
}
```

    - **Réponse en cas de commande non trouvée (404 Not Found)** :

```
{
  "error": "Commande non trouvée."
}
```

3. **PUT /commandes/{id}/statut**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "message": "Statut de la commande mis à jour."
}
```

    - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
  "error": "Échec de la mise à jour du statut."
}
```

4. **POST /commandes/{id}/details**
    - **Réponse en cas de succès (201 Created)** :

```
{
  "message": "Détail de commande ajouté."
}
```

    - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
  "error": "Échec de l'ajout du détail de commande."
}
```

5. **GET /commandes/{id}/details**
    - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id_commande": 123,
    "id_menu": 1,
    "quantité": 2,
    "sous_total": 19.98
  },
  {
```

```
    "id_commande": 123,
    "id_menu": 2,
    "quantité": 1,
    "sous_total": 10.01
  }
]
```

- **Réponse en cas de détail non trouvé (404 Not Found)** :

```
{
  "error": "Détails de commande non trouvés."
}
```

6. **GET /produits/{id}**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "id": 1,
  "nom": "Produit 1",
  "description": "Description du produit 1",
  "prix": 9.99
}
```

   - **Réponse en cas de produit non trouvé (404 Not Found)** :

```
{
  "error": "Produit non trouvé."
}
```

7. **GET /produits**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id": 1,
    "nom": "Produit 1",
    "description": "Description du produit 1",
    "prix": 9.99
  },
  {
    "id": 2,
    "nom": "Produit 2",
    "description": "Description du produit 2",
    "prix": 10.00
  }
]
```

8. **PUT /commandes/{id}/total**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "message": "Total de la commande mis à jour."
}
```

   - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
```

```
  "error": "Échec de la mise à jour du total."
}
```

9. **GET /panier/{id_etudiant}**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id_commande": 123,
    "id_menu": 1,
    "quantité": 2,
    "sous_total": 19.98
  },
  {
    "id_commande": 123,
    "id_menu": 2,
    "quantité": 1,
    "sous_total": 10.01
  }
]
```

   - **Réponse en cas de panier vide (404 Not Found)** :

```
{
  "error": "Panier vide."
}
```

Gestion des erreurs

1. **400 Bad Request** : Lorsqu'une requête est malformée ou que des paramètres requis sont manquants.
   - Réponse :

```
{
  "error": "Échec de l'insertion de la commande."
}
```

2. **404 Not Found** : Lorsque l'ID spécifié pour une commande, un détail ou un produit ne correspond à aucun enregistrement.
   - Réponse :

```
{
  "error": "Commande non trouvée."
}
```

3. **500 Internal Server Error** : En cas d'erreur serveur lors de l'exécution des requêtes.
   - Réponse :

```
{
  "error": "Erreur serveur."
}
```

STRUCTURES PROMOTION.PHP

2.3 Structure des API

Points d'entrée (endpoints)

1. **POST /commandes**
   - **Description** : Créer une nouvelle commande.
   - **Méthode** : `POST`
   - **Paramètres** :
     - `id_etudiant` (int) : Identifiant de l'étudiant (obligatoire).
     - `statut` (string) : Statut de la commande (obligatoire).
     - `total` (float) : Montant total de la commande (obligatoire).
     - `type_service` (string) : Type de service (obligatoire).
     - `commentaire` (string) : Commentaire sur la commande (optionnel).
     - `points_utilises` (int) : Points utilisés (optionnel).
   - **Réponse** :
     - En cas de succès : ID de la commande créée.
     - En cas d'erreur : Message d'erreur.

2. **GET /commandes/{id_commande}**
   - **Description** : Obtenir les informations d'une commande par son ID.
   - **Méthode** : `GET`
   - **Paramètres** :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).
   - **Réponse** :
     - En cas de succès : Détails de la commande.
     - En cas d'erreur : Message d'erreur.

3. **PUT /commandes/{id_commande}/statut**
   - **Description** : Mettre à jour le statut d'une commande.
   - **Méthode** : `PUT`
   - **Paramètres** :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).
     - `statut` (string) : Nouveau statut (obligatoire).
   - **Réponse** :
     - En cas de succès : Confirmation de la mise à jour.
     - En cas d'erreur : Message d'erreur.

4. **POST /commandes/{id_commande}/details**
   - **Description** : Ajouter un détail à une commande.
   - **Méthode** : `POST`
   - **Paramètres** :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).
     - `id_menu` (int) : Identifiant du produit du menu (obligatoire).
     - `quantité` (int) : Quantité du produit (obligatoire).
     - `sous_total` (float) : Montant sous-total (obligatoire).
   - **Réponse** :
     - En cas de succès : Confirmation de l'ajout.
     - En cas d'erreur : Message d'erreur.

5. **GET /commandes/{id_commande}/details**
   - **Description** : Obtenir les détails d'une commande.
   - **Méthode** : `GET`
   - **Paramètres** :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).

- **Réponse** :
  - En cas de succès : Liste des détails de la commande.
  - En cas d'erreur : Message d'erreur.

6. **GET /produits/{id_menu}**
   - **Description** : Obtenir un produit du menu par son ID.
   - **Méthode** : `GET`
   - **Paramètres** :
     - `id_menu` (int) : Identifiant du produit (obligatoire).
   - **Réponse** :
     - En cas de succès : Détails du produit.
     - En cas d'erreur : Message d'erreur.

7. **GET /produits**
   - **Description** : Obtenir tous les produits du menu.
   - **Méthode** : `GET`
   - **Réponse** :
     - En cas de succès : Liste de tous les produits.
     - En cas d'erreur : Message d'erreur.

8. **PUT /commandes/{id_commande}/total**
   - **Description** : Mettre à jour le total d'une commande.
   - **Méthode** : `PUT`
   - **Paramètres** :
     - `id_commande` (int) : Identifiant de la commande (obligatoire).
     - `total` (float) : Nouveau total (obligatoire).
   - **Réponse** :
     - En cas de succès : Confirmation de la mise à jour.
     - En cas d'erreur : Message d'erreur.

9. **GET /panier/{id_etudiant}**
   - **Description** : Obtenir le panier d'un étudiant.
   - **Méthode** : `GET`
   - **Paramètres** :
     - `id_etudiant` (int) : Identifiant de l'étudiant (obligatoire).
   - **Réponse** :
     - En cas de succès : Liste des détails du panier.
     - En cas d'erreur : Message d'erreur.

Méthodes et réponses API

1. **POST /commandes**
   - **Réponse en cas de succès (201 Created)** :

```
{
  "id_commande": 123
}
```

   - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
  "error": "Échec de l'insertion de la commande."
}
```

2. **GET /commandes/{id_commande}**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "id": 123,
  "id_etudiant": 456,
  "statut": "en cours",
  "total": 150.00,
  "type_service": "livraison",
  "commentaire": "Aucune",
  "points_utilises": 10
}
```

   - **Réponse en cas d'erreur (404 Not Found)** :

```
{
  "error": "Commande non trouvée."
}
```

3. **PUT /commandes/{id_commande}/statut**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "message": "Statut mis à jour."
}
```

   - **Réponse en cas d'erreur (404 Not Found)** :

```
{
  "error": "Commande non trouvée."
}
```

4. **POST /commandes/{id_commande}/details**
   - **Réponse en cas de succès (201 Created)** :

```
{
  "message": "Détail de commande ajouté."
}
```

   - **Réponse en cas d'erreur (400 Bad Request)** :

```
{
  "error": "Échec de l'ajout du détail de commande."
}
```

5. **GET /commandes/{id_commande}/details**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id": 1,
    "id_commande": 123,
```

```
    "id_menu": 789,
    "quantité": 2,
    "sous_total": 30.00
  }
]
```

- **Réponse en cas d'erreur (404 Not Found)** :

```
{
  "error": "Détails de commande non trouvés."
}
```

6. **GET /produits/{id_menu}**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "id": 789,
  "nom": "Burger",
  "description": "Un burger savoureux.",
  "prix": 15.00
}
```

   - **Réponse en cas d'erreur (404 Not Found)** :

```
{
  "error": "Produit non trouvé."
}
```

7. **GET /produits**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id": 789,
    "nom": "Burger",
    "description": "Un burger savoureux.",
    "prix": 15.00
  },
  {
    "id": 790,
    "nom": "Pizza",
    "description": "Une pizza délicieuse.",
    "prix": 20.00
  }
]
```

8. **PUT /commandes/{id_commande}/total**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "message": "Total mis à jour."
}
```

   - **Réponse en cas d'erreur (404 Not Found)** :

```
{
```

```
  "error": "Commande non trouvée."
}
```

9. **GET /panier/{id_etudiant}**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "id": 1,
    "id_commande": 123,
    "id_menu": 789,
    "quantité": 2,
    "sous_total": 30.00
  }
]
```

   - **Réponse en cas d'erreur (404 Not Found)** :

```
{
  "error": "Panier non trouvé."
}
```

STRUCTURES STATCOMMANDES.PHP

2.3 Structure des API

Points d'entrée (endpoints)

1. **GET /statistiques/revenu/mois**
   - Description : Récupérer le revenu total du mois actuel.
   - Méthode : `GET`
   - Réponse : Revenu total du mois actuel.

2. **GET /statistiques/commandes/mois**
   - Description : Récupérer le nombre total de commandes du mois actuel.
   - Méthode : `GET`
   - Réponse : Nombre total de commandes du mois actuel.

3. **GET /statistiques/revenu/jour**
   - Description : Récupérer le revenu total du jour actuel.
   - Méthode : `GET`
   - Réponse : Revenu total du jour actuel.

4. **GET /statistiques/mois**
   - Description : Récupérer les statistiques des revenus et du nombre de commandes par mois pour l'année actuelle.
   - Méthode : `GET`
   - Réponse : Statistiques des revenus et du nombre de commandes pour chaque mois.

Méthodes et réponses API

1. **GET /statistiques/revenu/mois**
   - **Réponse en cas de succès (200 OK)** :

```
json
Copy code
{
  "revenu_mois": montant
}
```

- **Exemple de réponse** :

```
json
Copy code
{
  "revenu_mois": 1500
}
```

2. **GET /statistiques/commandes/mois**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "nombre_commandes": nombre
}
```

3. **GET /statistiques/revenu/jour**
   - **Réponse en cas de succès (200 OK)** :

```
{
  "revenu_jour": montant
}
```

4. **GET /statistiques/mois**
   - **Réponse en cas de succès (200 OK)** :

```
[
  {
    "mois": 1,
    "nombre_commandes": 10,
    "revenus": 1000
  },
  ...
  {
    "mois": 12,
    "nombre_commandes": 0,
    "revenus": 0
  }
]
```

   - **Exemple de réponse** :

```
[
  {"mois": 1, "nombre_commandes": 10, "revenus": 1000},
  {"mois": 2, "nombre_commandes": 5, "revenus": 500},
  {"mois": 3, "nombre_commandes": 0, "revenus": 0},
  ...
  {"mois": 12, "nombre_commandes": 0, "revenus": 0}
]
```

Gestion des erreurs

1. **500 Internal Server Error** : Si une erreur survient lors de la connexion à la base de données ou de l'exécution des requêtes SQL.
    - Réponse :

```
{
  "success": false,
  "message": "Erreur de connexion : [message d'erreur PDO]."
}
```

STRUCTURE SIGNADMIN.PHP

2.3 Structure des API

Points d'entrée (endpoints)

1. **POST /auth/login**
    - **Description** : Authentifie un utilisateur administrateur avec son email et mot de passe.
    - **Méthode** : `POST`
    - **Paramètres** :
        - `email` (string) : L'email de l'utilisateur (obligatoire).
        - `password` (string) : Le mot de passe de l'utilisateur (obligatoire).
    - **Corps de la requête** (JSON) :

```
{
  "email": "utilisateur@example.com",
  "password": "votre_mot_de_passe"
}
```

    - **Réponses** :
        - **200 OK** : Connexion réussie.
        - **400 Bad Request** : Requête invalide (champs manquants ou format incorrect).
        - **404 Not Found** : Utilisateur non trouvé.
        - **401 Unauthorized** : Mot de passe incorrect.

Méthodes et réponses API

1. **POST /auth/login**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Connexion réussie"
}
```

    - **Réponse en cas de champ manquant (400 Bad Request)** :

```
{
  "success": false,
  "message": "Veuillez remplir tous les champs."
}
```

    - **Réponse en cas de format d'email invalide (400 Bad Request)** :

```
{
  "success": false,
  "message": "Format d'email invalide."
}
```

- **Réponse en cas d'utilisateur non trouvé (404 Not Found)** :

```
{
  "success": false,
  "message": "Utilisateur non trouvé."
}
```

- **Réponse en cas de mot de passe incorrect (401 Unauthorized)** :

```
{
  "success": false,
  "message": "Mot de passe incorrect."
}
```

- **Réponse en cas de méthode invalide (405 Method Not Allowed)** :

```
{
  "success": false,
  "message": "Requête invalide."
}
```

Gestion des erreurs

1. **400 Bad Request** : Lorsque des champs requis sont manquants ou ont un format incorrect.
   - Réponse :

```
{
  "success": false,
  "message": "Veuillez remplir tous les champs."
}
```

2. **404 Not Found** : Lorsque l'utilisateur avec l'email fourni n'existe pas dans la base de données.
   - Réponse :

```
{
  "success": false,
  "message": "Utilisateur non trouvé."
}
```

3. **401 Unauthorized** : Lorsque le mot de passe fourni est incorrect.
   - Réponse :

```
json
Copy code
{
  "success": false,
  "message": "Mot de passe incorrect."
}
```

4. **405 Method Not Allowed** : Si la méthode HTTP utilisée n'est pas autorisée pour cet endpoint.
    - Réponse :

```
{
  "success": false,
  "message": "Requête invalide."
}
```

STRUCTURE DATABASE.PHP

2.3 Structure des API

Points d'entrée (endpoints)

1. **POST /auth/login**
    - **Description** : Authentifie un utilisateur administrateur avec son email et mot de passe.
    - **Méthode** : `POST`
    - **Paramètres** :
        - `email` (string) : L'email de l'utilisateur (obligatoire).
        - `password` (string) : Le mot de passe de l'utilisateur (obligatoire).
    - **Corps de la requête (JSON)** :

```
{
  "email": "utilisateur@example.com",
  "password": "votre_mot_de_passe"
}
```

Réponses :

- **200 OK** : Connexion réussie.
- **400 Bad Request** : Requête invalide (champs manquants ou format incorrect).
- **404 Not Found** : Utilisateur non trouvé.
- **401 Unauthorized** : Mot de passe incorrect.

Méthodes et réponses API

1. **POST /auth/login**
    - **Réponse en cas de succès (200 OK)** :

```
{
  "success": true,
  "message": "Connexion réussie"
}
```

    - **Réponse en cas de champ manquant (400 Bad Request)** :

```
{
  "success": false,
  "message": "Veuillez remplir tous les champs."
}
```

    - **Réponse en cas de format d'email invalide (400 Bad Request)** :

```
{
  "success": false,
  "message": "Format d'email invalide."
}
```

- **Réponse en cas d'utilisateur non trouvé (404 Not Found)** :

```
{
  "success": false,
  "message": "Utilisateur non trouvé."
}
```

- **Réponse en cas de mot de passe incorrect (401 Unauthorized)** :

```
{
  "success": false,
  "message": "Mot de passe incorrect."
}
```

- **Réponse en cas de méthode invalide (405 Method Not Allowed)** :

```
{
  "success": false,
  "message": "Requête invalide."
}
```

Gestion des erreurs

1. **400 Bad Request** : Lorsque des champs requis sont manquants ou ont un format incorrect.
   - **Réponse** :

```
{
  "success": false,
  "message": "Veuillez remplir tous les champs."
}
```

2. **404 Not Found** : Lorsque l'utilisateur avec l'email fourni n'existe pas dans la base de données.
   - **Réponse** :

```
{
  "success": false,
  "message": "Utilisateur non trouvé."
}
```

3. **401 Unauthorized** : Lorsque le mot de passe fourni est incorrect.
   - **Réponse** :

```
{
  "success": false,
  "message": "Mot de passe incorrect."
}
```

4. **405 Method Not Allowed** : Si la méthode HTTP utilisée n'est pas autorisée pour cet endpoint.
    - **Réponse** :

```
{
  "success": false,
  "message": "Requête invalide."
}
```

Guide d'installation et d'utilisation

3.1. PREREQUIS

Avant de commencer l'installation et l'utilisation de cette API, assurez-vous d'avoir les éléments suivants installés et configurés sur votre système :

- **PHP** (version 7.4 ou supérieure)
- **Serveur Web** (Apache, Nginx ou tout autre serveur compatible)
- **MySQL** ou **MariaDB** pour la base de données
- **Composer** pour la gestion des dépendances PHP
- **Postman** pour tester les API
- **Docker** (facultatif mais recommandé pour l'environnement de développement et de production)

3.2. INSTALLATION EN LOCAL

Étapes d'installation

1. **Cloner le dépôt du projet** : Utilisez `git` pour cloner le dépôt dans votre répertoire local.

```
bash
Copy code
git clone https://github.com/mon-depot/mon-projet-api.git
```

2. **Naviguer dans le répertoire du projet** :

```
bash
Copy code
cd mon-projet-api
```

3. **Configurer l'environnement** : Créez un fichier `.env` en copiant le fichier `.env.example` et en modifiant les variables selon vos besoins :

```
bash
Copy code
cp .env.example .env
```

Assurez-vous de définir les bonnes valeurs pour les éléments suivants :

- `DB_HOST` : L'hôte de la base de données (par exemple, `localhost`).
- `DB_DATABASE` : Le nom de la base de données.
- `DB_USERNAME` : Le nom d'utilisateur de la base de données.
- `DB_PASSWORD` : Le mot de passe de la base de données.

4. **Installer les dépendances PHP** : Utilisez **Composer** pour installer les dépendances du projet :

```
bash
Copy code
composer install
```

5. **Initialiser la base de données** : Créez la base de données et appliquez les migrations (si des fichiers de migration sont disponibles) :

```
bash
Copy code
php artisan migrate
```

Commandes nécessaires

- **Démarrer le serveur de développement** :

```
bash
Copy code
php -S localhost:8000 -t public
```

- **Testez les routes avec Postman** :
  - Créez des requêtes dans **Postman** pour interagir avec l'API.
  - Exemple : Pour tester le point d'entrée `POST /auth/login`, configurez une requête `POST` avec les informations suivantes :
    - **URL** : `http://localhost:8000/auth/login`
    - **Body** (format `JSON`) :

```
json
Copy code
{
  "email": "admin@example.com",
  "password": "monmotdepasse"
}
```

3.3. DEPLOIEMENT EN PRODUCTION

Configuration de l'environnement

1. **Mettre à jour les variables d'environnement** : Mettez à jour les variables dans le fichier `.env` pour correspondre à votre environnement de production. Assurez-vous que les paramètres de base de données, d'hôtes, et de sécurité sont configurés correctement.
2. **Optimiser les performances** : Exécutez les commandes suivantes pour optimiser l'application avant le déploiement :

```
bash
Copy code
```

```
php artisan config:cache
php artisan route:cache
php artisan optimize
```

3. **Configurer le serveur Web** : Configurez **Apache** ou **Nginx** pour pointer vers le répertoire `public/` de votre application PHP.

Commandes de déploiement

- **Exécuter des migrations en production** :

```
bash
Copy code
php artisan migrate --force
```

- **Relancer les services** (avec Docker si utilisé) : Si vous utilisez Docker pour votre application, vous pouvez redémarrer l'application en production en exécutant :

```
bash
Copy code
docker-compose down
docker-compose up -d
```

3.4. GESTION DES DEPENDANCES ET MISE A JOUR

Postman :

- **Créer des collections de requêtes** : Utilisez **Postman** pour organiser les différentes requêtes API dans une **collection**. Cela vous permettra de tester rapidement toutes les fonctionnalités et les points d'entrée de l'API, et de sauvegarder les réponses.
- **Tests automatisés** : Vous pouvez ajouter des tests dans Postman pour valider automatiquement les réponses des API. Exemple de test dans l'onglet **Tests** de Postman :

```
javascript
Copy code
pm.test("Statut est 200", function () {
    pm.response.to.have.status(200);
});
```

Docker :

1. **Créer un environnement Docker** :

   Utilisez **Docker** pour gérer vos environnements de développement et de production avec un fichier `docker-compose.yml`. Voici un exemple de configuration :

```
yaml
Copy code
version: '3'
services:
  web:
    image: php:7.4-apache
    volumes:
      - .:/var/www/html
```

```
    ports:
      - "8080:80"
    environment:
      - DB_HOST=db
      - DB_DATABASE=zeducspace
      - DB_USERNAME=root
      - DB_PASSWORD=root
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: zeducspace
      MYSQL_USER: root
      MYSQL_PASSWORD: root
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

2. **Lancer Docker** :
   - Pour démarrer vos conteneurs Docker :

```
bash
Copy code
docker-compose up -d
```

3. **Mettre à jour les dépendances** :
   - En cas de mise à jour des dépendances ou des changements dans les configurations, vous pouvez exécuter les commandes suivantes :

```
bash
Copy code
composer update
docker-compose down
docker-compose up --build -d
```

4. Schémas de la base de données

4.1. Structure des tables

Voici la structure des principales tables de la base de données. Ces tables contiennent les données nécessaires à la gestion des utilisateurs, des employés, des rôles et autres informations liées à l'application.

1. TABLE UTILISATEUR

Attribut	Taper	Description
id	int	Identifiant unique de l'utilisateur (clé primaire).
nom	varchar(255)	Nom complet de l'utilisateur.
email	varchar(255)	Adresse email utilisée pour la connexion.
mot_de_passe	varchar(255)	Mot de passe sécurisé pour l'authentification.
role	énumération	Rôle de l'utilisateur : Étudiant, Employé, Administrateur, Gérant.
numero_telephone	varchar(15)	Numéro de téléphone de l'utilisateur.
adresse	varchar(255)	Adresse utilisée pour la livraison.

Attribut	Taper	Description
points_fidelite	int	Nombre de points de fidélité accumulés par l'utilisateur.
parrain_id	int, nullable	Référence à l'utilisateur qui a parrainé cet utilisateur (nullable).
date_inscription	date	Date d'inscription de l'utilisateur dans le système.

2. TABLE COMMANDE

Attribut	Taper	Description
id	int	Identifiant unique de la commande (clé primaire).
user_id	int	Référence à l'utilisateur (étudiant) qui passe la commande.
montant_total	décimal(10,2)	Montage total de la commande.
statut	énumération	Statut de la commande : en cours, validée, livrée, annulée.
date_commande	date et heure	Date et heure à laquelle la commande a été passée.
commentaire	texte, nullable	Commentaires laissés par l'étudiant lors de la commande.
points_utilises	int	Nombre de points de fidélité utilisés pour cette commande.

3. TABLEAU ARTICLE

Attribut	Taper	Description
id	int	Identifiant unique de l'article (clé primaire).
nom	varchar(255)	Nom de l'article (ex : "Pizza Margherita").
description	texte	Description de l'article (ex : ingrédients, caractéristiques).
prix	décimal(10,2)	Prix unitaire de l'article.
quantite_disponible	int	Quantité d'articles disponibles (pour gestion des stocks).

4. TABLEAU DETAILS_COMMANDE

Ce tableau repose sur les **commandes** aux **articles** commandés, en gérant les détails comme la quantité et le prix.

Attribut	Taper	Description
id	int	Identifiant unique du détail de commande (clé primaire).
commande_id	int	Référence à la commande concernée (clé étrangère).
article_id	int	Référence à l'article commandé (clé étrangère).
quantite	int	Quantité de cet article dans la commande.
prix_unitaire	décimal(10,2)	Prix de l'article au moment de la commande.

5. TABLEAU RÉCLAMATION

Attribut	Taper	Description
id	int	Identifiant unique de la réclamation (clé primaire).
user_id	int	Référence à l'utilisateur (étudiant) ayant déposé la réclamation.

Attribut	Taper	Description
commande_id	int	Référence à la commande concernée par la réclamation.
description	texte	Détails de la réclamation.
statut	énumération	Statut de la réclamation : soumise, en cours, résolue.

6. TABLEAU PARRAINAGE

Cette table gère les relations de parrainage entre étudiants.

Attribut	Taper	Description
id	int	Identifiant unique du parrainage (clé primaire).
parrain_id	int	Référence à l'utilisateur parrain (clé étrangère).
filleul_id	int	Référence à l'utilisateur filleul (clé étrangère).
date_parrainage	date et heure	Date du parrainage.
points_gagnes	int	Points de fidélité gagnés via le parrainage.

7. TABLEAU PROMOTION

Attribut	Taper	Description
id	int	Identifiant unique de la promotion (clé primaire).
nom	varchar(255)	Nom de la promotion (ex : "Promo Étudiant -10%").
description	texte	Détails de la promotion.
date_debut	date et heure	Date de début de la promotion.
date_fin	date et heure	Date de fin de la promotion.

8. TABLEAU STATISTIQUES

Attribut	Taper	Description
id	int	Identifiant unique des statistiques (clé primaire).
date	date et heure	Date de l'enregistrement des statistiques.
total_commandes	int	Nombre total de commandes effectuées durant la période donnée.
total_ventes	décimal(10,2)	Montant total des ventes durant la période donnée.

Relations entre les tables

1. **UTILISATEUR** :
    - **1-N** avec **COMMANDE** (Un utilisateur peut passer plusieurs commandes).
    - **1-N** avec **RÉCLAMATION** (Un utilisateur peut déposer plusieurs réclamations).
    - **1-N** avec **PARRAINAGE** (Un utilisateur peut être parrain ou filleul dans plusieurs relations de parrainage).
2. **COMMANDE** :

   - **1-N** avec **DETAILS_COMMANDE** (Une commande peut contenir plusieurs articles).
3. **ARTICLE** :
   - **N-N** avec **COMMANDE** (Un article peut apparaître dans plusieurs commandes via **DETAILS_COMMANDE**).
4. **PARRAINAGE** :
   - **1-N** avec **UTILISATEUR** (Un utilisateur peut parrainer plusieurs filleuls).

Clés primaires et étrangères

- **Clés primaires** : Chaque table dispose d'une clé primaire (`id`).
- **Clés étrangères** :
  - `user_id` dans **COMMANDE**, **RÉCLAMATION**, et **PARRAINAGE** fait référence à l'ID d'un utilisateur.
  - `commande_id` dans **DETAILS_COMMANDE** et **RÉCLAMATION** fait référence à l'ID d'une commande.
  - `article_id` dans **DETAILS_COMMANDE** fait référence à l'ID d'un article.

Contraintes et Index

Contraintes

Les contraintes garantissent l'intégrité des données et la cohérence des relations entre les tables. Voici les principales contraintes appliquées sur chaque table :

1. Table UTILISATEUR

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire qui assure l'unicité de chaque utilisateur.
UNIQUE (email)	`email` doit être unique pour chaque utilisateur afin d'éviter les doublons.
NOT NULL (nom, email, mot_de_passe, role, numero_telephone)	Ces champs ne peuvent pas être vides (contraintes de non-nullité).
FOREIGN KEY (parrain_id)	Référence à l'utilisateur parrainé (nullable) dans la même table `UTILISATEUR`.

2. Table COMMANDE

Contrainte	Description

PRIMARY KEY (id)	`id` est la clé primaire pour identifier chaque commande.
FOREIGN KEY (user_id)	Référence à l'utilisateur dans la table `UTILISATEUR`.
NOT NULL (user_id, montant_total, statut, date_commande)	Ces champs doivent être remplis pour qu'une commande soit valide.

3. Table ARTICLE

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire qui identifie chaque article.
NOT NULL (nom, prix, quantite_disponible)	Ces champs doivent obligatoirement être remplis pour assurer une gestion correcte des stocks.

4. Table DETAILS_COMMANDE

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire pour chaque détail de commande.
FOREIGN KEY (commande_id)	Référence à la commande dans la table `COMMANDE`.
FOREIGN KEY (article_id)	Référence à l'article dans la table `ARTICLE`.
NOT NULL (commande_id, article_id, quantite, prix_unitaire)	Ces champs doivent être remplis pour lier correctement les articles à la commande.

5. Table RÉCLAMATION

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire pour chaque réclamation.
FOREIGN KEY (user_id)	Référence à l'utilisateur dans la table `UTILISATEUR`.
FOREIGN KEY (commande_id)	Référence à la commande dans la table `COMMANDE`.
NOT NULL (user_id, commande_id, description, statut)	Ces champs sont obligatoires pour une gestion efficace des réclamations.

6. Table PARRAINAGE

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire pour chaque relation de parrainage.
FOREIGN KEY (parrain_id)	Référence à l'utilisateur parrain dans la table `UTILISATEUR`.
FOREIGN KEY (filleul_id)	Référence à l'utilisateur filleul dans la table `UTILISATEUR`.
NOT NULL (parrain_id, filleul_id, date_parrainage, points_gagnes)	Tous ces champs doivent être remplis pour valider la relation de parrainage.

7. Table PROMOTION

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire pour chaque promotion.
NOT NULL (nom, date_debut, date_fin)	Ces champs sont obligatoires pour gérer les périodes de promotion.

8. Table STATISTIQUES

Contrainte	Description
PRIMARY KEY (id)	`id` est la clé primaire pour chaque enregistrement statistique.
NOT NULL (date, total_commandes, total_ventes)	Ces champs sont nécessaires pour le calcul et la conservation des statistiques.

Index

Les index sont utilisés pour accélérer les requêtes, surtout celles impliquant des recherches fréquentes ou des jointures entre plusieurs tables.

UTILISATEUR :

- **Index sur `email`** : Permet d'accélérer les recherches d'utilisateurs par email, surtout pour la connexion.

  ```
  sql
  Copy code
  CREATE INDEX idx_utilisateur_email ON UTILISATEUR(email);
  ```

- **Index sur `parrain_id`** : Améliore les performances des requêtes qui impliquent des relations de parrainage.

```
sql
Copy code
CREATE INDEX idx_utilisateur_parrain ON
UTILISATEUR(parrain_id);
```

COMMANDE :

- **Index sur `user_id`** : Optimise les requêtes qui récupèrent les commandes d'un utilisateur.

```
sql
Copy code
CREATE INDEX idx_commande_user ON COMMANDE(user_id);
```

- **Index sur `statut`** : Accélère les requêtes de recherche ou de filtrage par statut de commande.

```
sql
Copy code
CREATE INDEX idx_commande_statut ON COMMANDE(statut);
```

ARTICLE :

- **Index sur `nom`** : Améliore les performances lors de la recherche d'articles par leur nom.

```
sql
Copy code
CREATE INDEX idx_article_nom ON ARTICLE(nom);
```

DETAILS_COMMANDE :

- **Index sur `commande_id`** : Optimise les jointures lors de la récupération des articles d'une commande.

```
sql
Copy code
CREATE INDEX idx_details_commande_id ON
DETAILS_COMMANDE(commande_id);
```

- **Index sur `article_id`** : Accélère la recherche d'articles spécifiques dans les commandes.

```
sql
Copy code
CREATE INDEX idx_details_article_id ON
DETAILS_COMMANDE(article_id);
```

RÉCLAMATION :

- **Index sur `user_id`** : Permet de récupérer rapidement les réclamations d'un utilisateur.

```
sql
```

```
Copy code
CREATE INDEX idx_reclamation_user ON RECLAMATION(user_id);
```

- **Index sur `commande_id`** : Optimise les requêtes liées à des réclamations spécifiques à une commande.

```
sql
Copy code
CREATE INDEX idx_reclamation_commande_id ON
RECLAMATION(commande_id);
```

PARRAINAGE :

- **Index sur `parrain_id` et `filleul_id`** : Améliore les performances des requêtes impliquant les relations de parrainage.

```
sql
Copy code
CREATE INDEX idx_parrainage_parrain ON PARRAINAGE(parrain_id);
CREATE INDEX idx_parrainage_filleul ON PARRAINAGE(filleul_id);
```