Université de Yaoundé I

Faculté des Sciences

Département d'Informatique

Projet d'Application Web de E-commerce

Liste des participants :

AKONO NOAH CAROLE JESSICA 22T2892
DASSI MANDJO LEA JUSTINE 22W2164
DOSSIVIL RIVOIRE VIANEY 22U2116
NJUMBE SONIA TCHOMGUEM 22T2945
KOMADJOU TCHAPTCHE IMELDA SHELVIE 22T2956

Sous la supervision de : **Dr JIOMEKONG**

Année Académique 2024/2025

Contents

1 Présentation de Asso **2**
- 1.1 Interface utilisateur intuitive . . . 2
- 1.2 Système de recherche avancé . . . 2
- 1.3 Gestion des produits . . . 2
- 1.4 Sécurisation des transactions . . . 2
- 1.5 Suivi des commandes . . . 2
- 1.6 Gestion du panier utilisateur . . . 3
- 1.7 Gestion du fournisseur . . . 3
- 1.8 Gestion de livraison . . . 3
- 1.9 Gestion des recommandations: . . . 3

2 Analyse UML **4**
- 2.1 Diagramme de cas d'utilisation . . . 4
- 2.2 Diagramme d'état de transition . . . 5
- 2.3 Diagramme de classe . . . 5
- 2.4 Diagramme de séquence . . . 6

3 Architecture Physique de l'Application **8**

4 Architecture Logique de l'Application **10**
- 4.1 Pourquoi les Microservices pour l'implementation de l'Application? . . . 10
- 4.2 Tableau des technologies utilisées dans chaque Microservice . . . 11

5 ANNEXES **12**
- 5.1 API Utilisateur . . . 12
- 5.2 API du Fournisseur . . . 14
- 5.3 API des Produits . . . 14
- 5.4 API de Livraison . . . 15
- 5.5 API des Paiements . . . 15
- 5.6 API de recommandation des produits . . . 17
- 5.7 API des Commandes . . . 17

1 Présentation de Asso

Asso est une plateforme de e-commerce innovante conçue pour simplifier l'expérience d'achat en ligne. Avec une interface conviviale et des fonctionnalités avancées, Asso permet aux utilisateurs de découvrir, comparer et acheter des produits facilement, tout en bénéficiant d'une expérience personnalisée.

1.1 Interface utilisateur intuitive

- **Design réactif :** s'adapte à tous les appareils (ordinateurs, tablettes, smartphones).
- **Navigation fluide :** Menus clairs et catégories bien définies pour une exploration aisée.

1.2 Système de recherche avancé

- **Filtres de recherche :** Permet aux utilisateurs de trier par catégorie et sous-catégorie.
- **Suggestions personnalisées :** Recommandations basées sur les préférences et l'historique d'achat.

1.3 Gestion des produits

- **Ajout facile des produits :** Interface pour l'administrateur de l'application (Supermarché, boutique) permettant d'ajouter et gérer les produits rapidement.
- **Descriptions détaillées :** Chaque produit est accompagné de photos de haute résolution et de descriptions complètes.

1.4 Sécurisation des transactions

- **Paiement sécurisé :** Intégration de protocoles de sécurité avancés pour protéger les données des utilisateurs.
- **Multiples méthodes de paiement :** Acceptation de cartes de crédit, MoMo et Orange Money.

1.5 Suivi des commandes

- **Notifications en temps réel :** Mises à jour sur l'état de la commande envoyées par email.
- **Historique des commandes :** Les utilisateurs peuvent consulter l'historique de leurs commandes.

1.6 Gestion du panier utilisateur

- **Ajout rapide :** Les utilisateurs peuvent ajouter des produits à leur panier en un clic depuis la page de description personnalisée des produits.
- **Suppression facile :** Option pour retirer des articles du panier avec une simple action, garantissant une flexibilité totale.
- **Mise à jour des quantités :** Les utilisateurs peuvent ajuster le nombre d'unités d'un produit directement depuis le panier.
- **Alerte stock :** Notifications lorsque la quantité demandée dépasse le stock disponible.

1.7 Gestion du fournisseur

- **Ajout de fournisseur :** L'administrateur peut ajouter un fournisseur de produit et les gérer.

1.8 Gestion de livraison

- **Ajout des livreurs :** L'administrateur peut ajouter un livreur de produit et les gérer.
- **Notifier les livreurs :** L'administrateur peut notifier un livreur pour signaler une livraison.

1.9 Gestion des recommandations:

- **Basé sur les avis des utilisateurs** : Recommande des éléments en fonction des préférences d'utilisateurs.

2 Analyse UML

2.1 Diagramme de cas d'utilisation

Voici une brève description du diagramme de cas d'utilisation. Ce diagramme illustre les interactions entre les utilisateurs et le système.

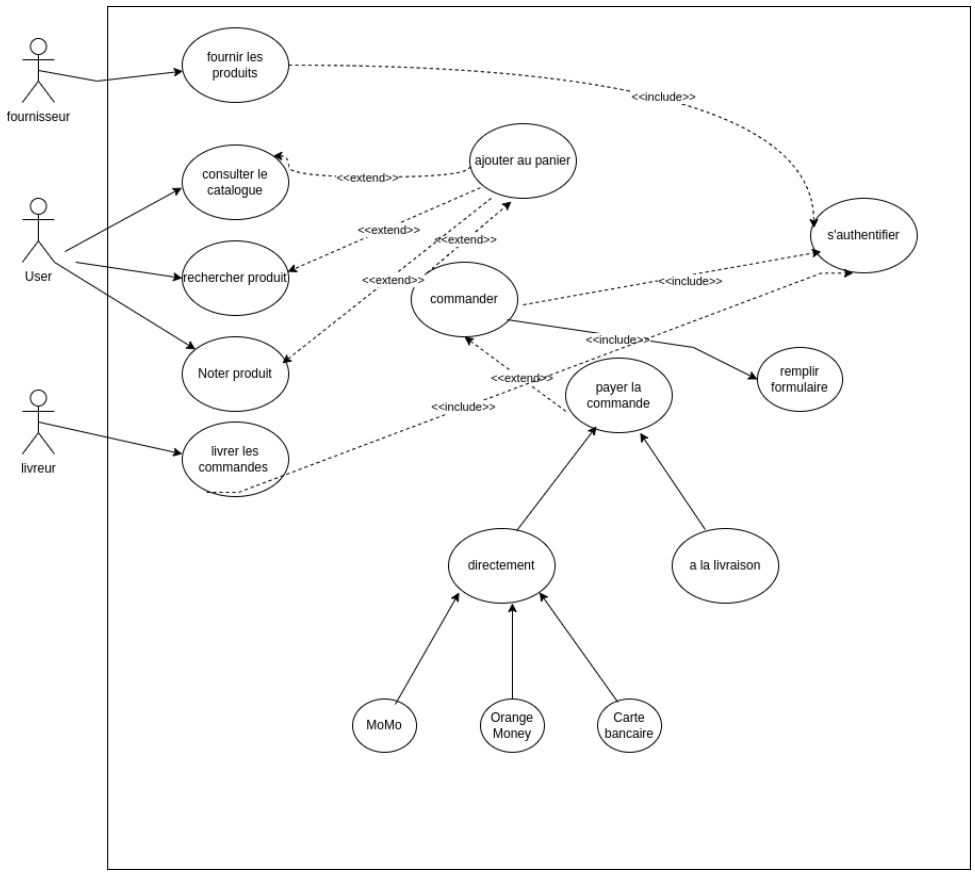

Figure 1: Diagramme de cas d'utilisation

2.2 Diagramme d'état de transition

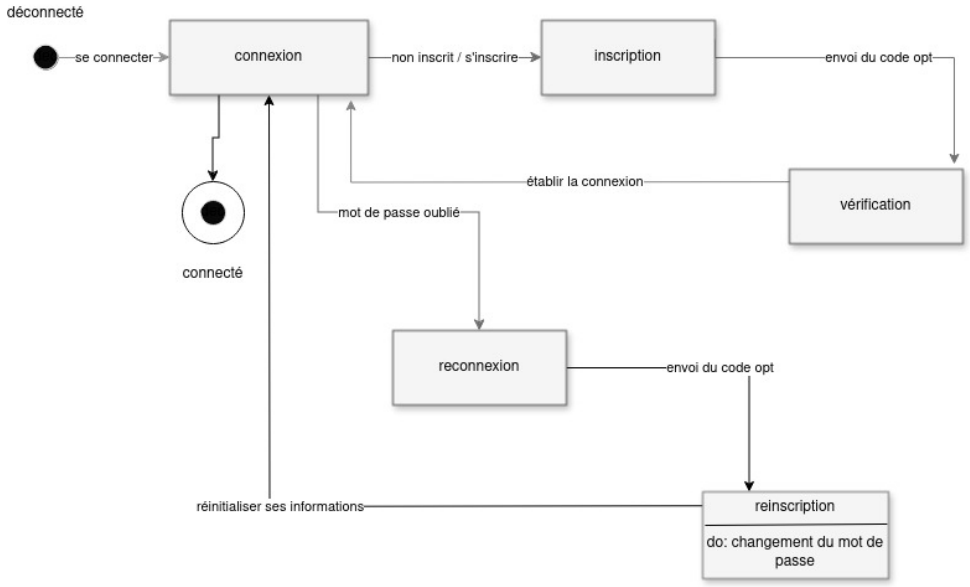

Figure 2: Diagramme d'état de transition de l'utilisateur

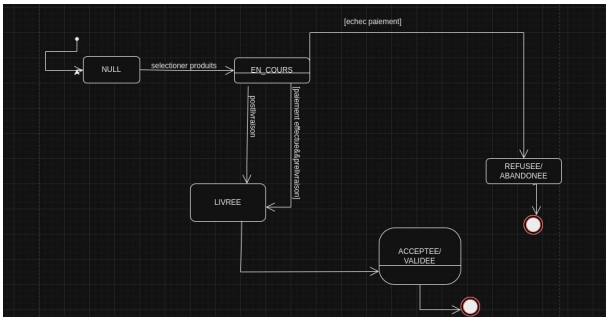

Figure 3: Diagramme état-transition de commande

2.3 Diagramme de classe

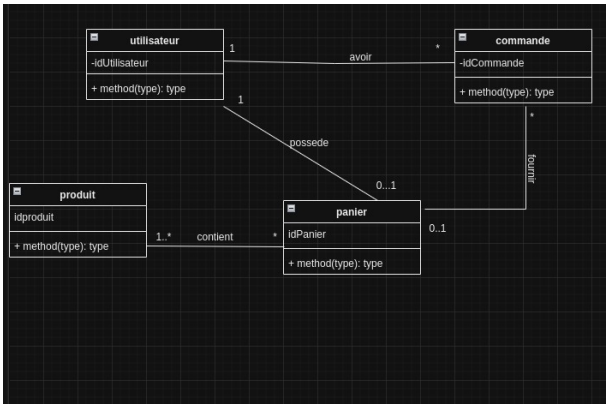

Figure 4: Diagramme de classe des commandes

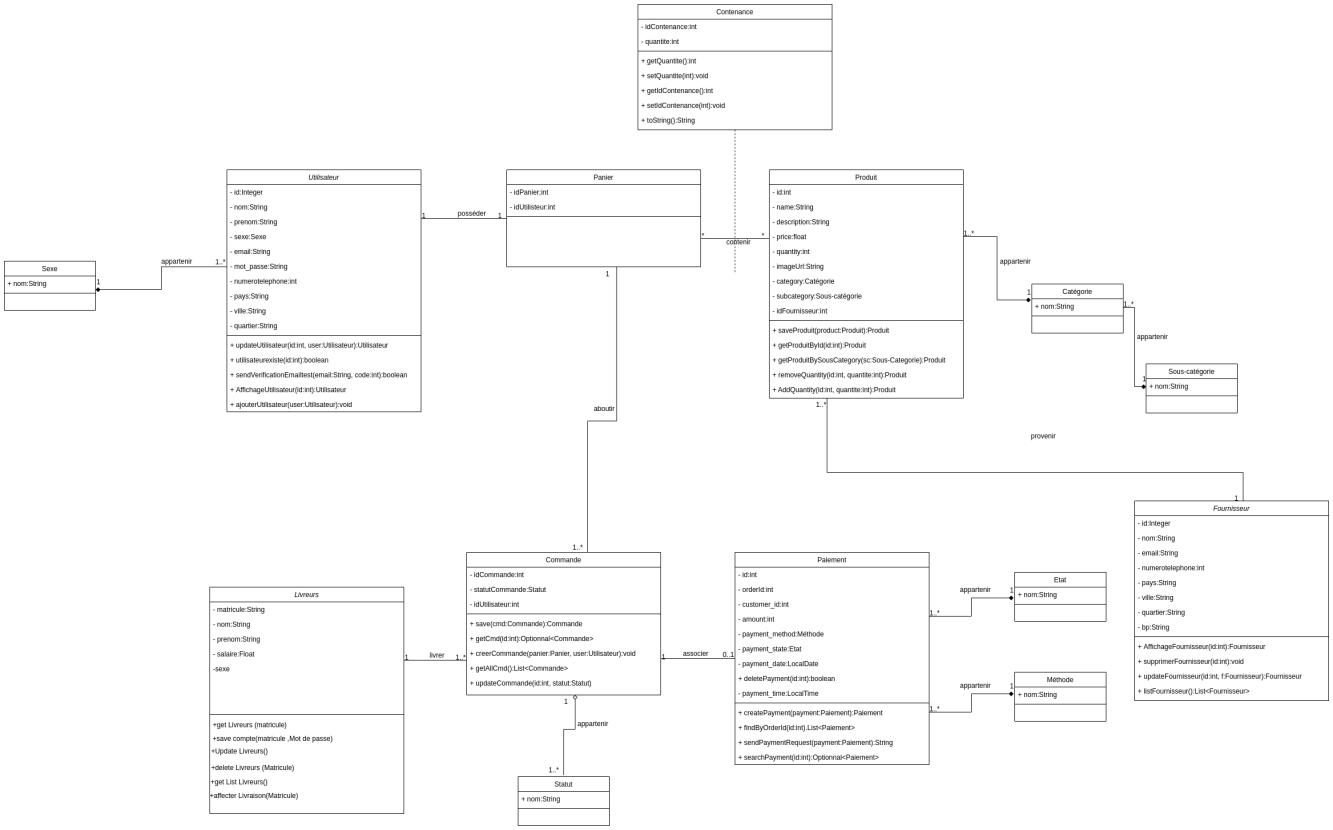

Figure 5: Diagramme de classe général

2.4 Diagramme de séquence

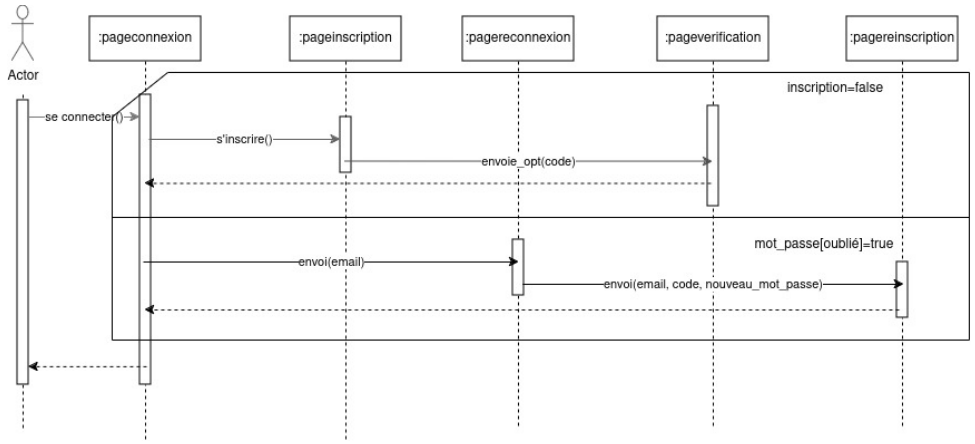

Figure 6: Diagramme de séquence de l'utilisateur

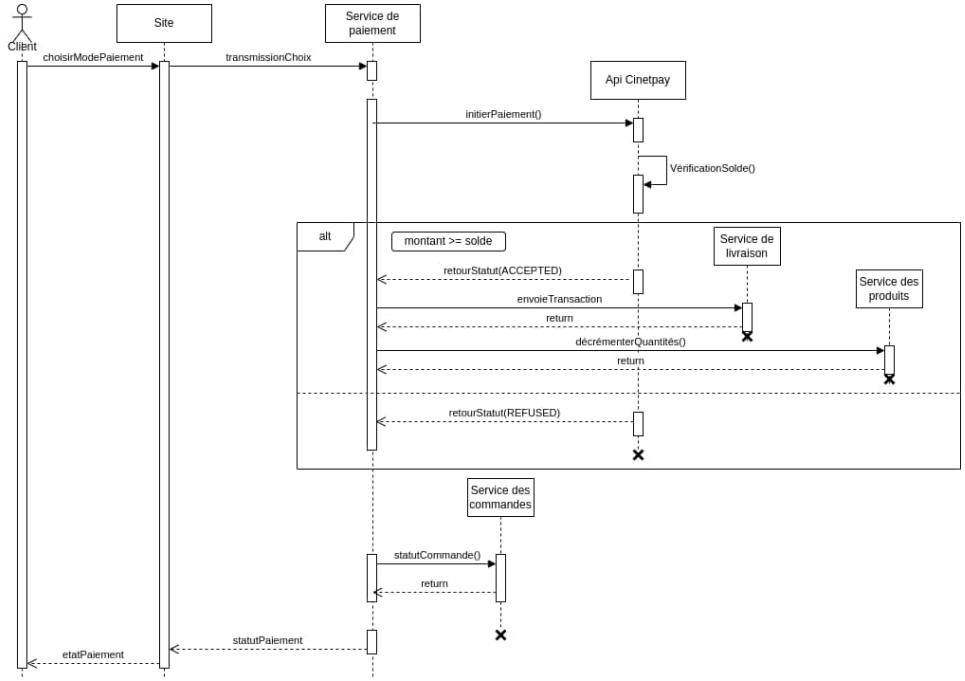

Diagramme de séquence du microservice de gestion des paiements

Figure 7: Diagramme de séquence des paiements

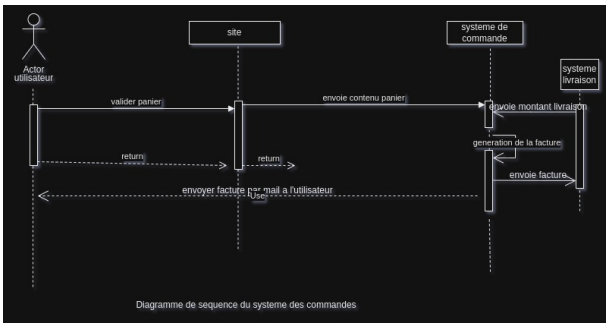

Figure 8: Diagramme de séquence des commandes

3 Architecture Physique de l'Application

L'architecture physique de notre application est conçue pour être modulaire et évolutive, facilitant ainsi la maintenance et l'ajout de nouvelles fonctionnalités. Elle se compose de plusieurs éléments clés :

- **Front-end :** Le front-end est l'interface utilisateur de l'application, développée en utilisant des technologies modernes telles que React et Bootstrap. Il est responsable de l'interaction avec les utilisateurs et de la présentation des données. Le front-end communique avec les services back-end via des API, garantissant une expérience utilisateur fluide et réactive.

- **API Gateway :** L'API Gateway agit comme un point d'entrée unique pour toutes les requêtes provenant du front-end. Elle gère le routage des demandes vers les différents microservices, tout en fournissant des fonctionnalités telles que l'authentification, la gestion des sessions et la limitation de débit, pour se faire nous avons configuré un serveur Eureka pour la découverte des microservices. Cela permet de centraliser la gestion des API et d'améliorer la sécurité de l'application.

- **Microservices :** L'architecture repose sur plusieurs microservices, chacun étant responsable d'un domaine fonctionnel spécifique. Ces microservices communiquent entre eux via des appels API, ce qui permet une séparation des responsabilités et une indépendance dans le développement. Les microservices de notre application incluent :

  - **Microservice de Gestion des Produits :** Ce service est chargé de la gestion du catalogue de produits. Il permet d'ajouter, de modifier et de supprimer des produits, ainsi que de gérer les informations associées, telles que les prix et les descriptions.
  - **Microservice de Gestion des Utilisateurs :** Ce service gère l'authentification et l'autorisation des utilisateurs. Il traite les inscriptions, les connexions et les profils utilisateurs, assurant ainsi la sécurité et la gestion des données personnelles.
  - **Microservice de Gestion des Fournisseurs :** Ce composant est responsable de la gestion des relations avec les fournisseurs. Il permet de suivre les informations sur les fournisseurs, les produits fournis et les conditions de commande.
  - **Microservice de Gestion des Commandes :** Ce service gère l'ensemble du processus de commande, du panier d'achat à la confirmation de la commande. Il s'assure que les commandes sont traitées correctement et que les utilisateurs reçoivent des mises à jour sur l'état de leurs commandes.
  - **Microservice de Gestion des Livraisons :** Ce service est responsable de la planification et du suivi des livraisons. Il interagit avec les livreurs pour garantir que les produits sont livrés aux utilisateurs dans les délais impartis.
  - **Microservice de Gestion des Paiements :** Le microservice de gestion des paiements est une composante essentielle d'une architecture de

e-commerce. Il est responsable de toutes les opérations liées au traitement des paiements, garantissant ainsi que les transactions sont effectuées de manière sécurisée, efficace et fiable. Ce microservice communique avec d'autres services de l'application, tels que la gestion des commandes et la gestion des utilisateurs.

– **Microservice de Gestion des Recommandations :** système de recommandation est une API dédiée à l'optimisation de l'expérience utilisateur dans le domaine du commerce en ligne. Ce système est conçu pour récupérer, trier et renvoyer les produits en fonction de leurs notes, permettant ainsi aux utilisateurs de découvrir facilement les produits les mieux évalués.

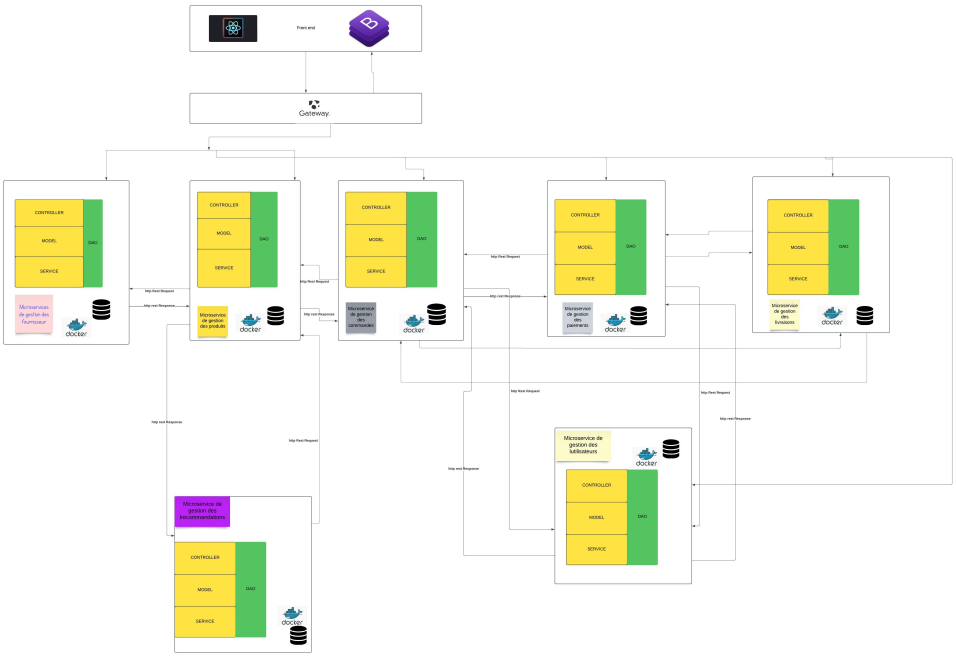

Figure 9: Représentation graphique de l'architecture de notre application

4 Architecture Logique de l'Application

Dans un environnement de e-commerce, les pics de trafic peuvent survenir à des moments critiques, comme pendant les promotions ou les périodes de fête. Une application monolithique peut rencontrer des difficultés à gérer une augmentation soudaine du nombre d'utilisateurs, ce qui peut entraîner des ralentissements, des erreurs ou même des temps d'arrêt. Cela peut nuire à l'expérience utilisateur, provoquer une perte de ventes et nuire à la réputation de la marque.

4.1 Pourquoi les Microservices pour l'implementation de l'Application?

L'architecture microservices permet de décomposer l'application en plusieurs services indépendants, chacun responsable d'une fonctionnalité spécifique (gestion des utilisateurs, gestion des produits, traitement des paiements, etc.). Cette approche offre plusieurs avantages :

- **Scalabilité Indépendante** : Chaque microservice peut être mis à l'échelle individuellement en fonction des besoins de charge, permettant une utilisation optimale des ressources.
- **Déploiement Rapide et Fréquent** : Les équipes peuvent déployer des mises à jour de manière autonome sans impacter l'ensemble du système, ce qui réduit le temps de mise sur le marché.
- **Résilience** : Si un microservice échoue, les autres services peuvent continuer à fonctionner, ce qui augmente la fiabilité et la disponibilité de l'application entière.
- **Technologies Variées** : Les équipes peuvent choisir les meilleures technologies adaptées à chaque microservice, favorisant l'innovation et l'optimisation des performances.
- **Facilité de Maintenance** : Les microservices, étant de plus petite taille et indépendants, sont généralement plus faciles à comprendre, à tester et à maintenir.
- **Développement Agile** : Les équipes peuvent travailler en parallèle sur différents services, ce qui favorise une approche agile et itérative du développement.
- **Amélioration des Performances** : Les microservices peuvent être optimisés individuellement, ce qui peut conduire à de meilleures performances globales de l'application.
- **Meilleure Gestion des Échecs** : La décomposition en services indépendants permet de mieux gérer les échecs, en isolant les problèmes et en minimisant leur impact sur l'ensemble du système.
- **Facilité d'Intégration** : Les microservices facilitent l'intégration avec d'autres systèmes et services, y compris des applications tierces ou des services cloud.
- **Évolutivité des Équipes** : Les équipes peuvent être organisées autour de services spécifiques, permettant une spécialisation et une expertise plus poussées.

4.2 Tableau des technologies utilisées dans chaque Microservice

Microservice	**Langage**	**Base de Données**	**Framework**
Gestion des Produits	Java	mySQL	Spring boot
Gestion des Utilisateurs	Java	PosgreSQL	Spring boot
Gestion des Fournisseurs	Java	PosgreSQL	Spring Boot
Gestion des Commandes	Java	mySQL	Spring boot
Gestion des Livraisons	Python	SQLite	Django
Gestion des Paiements	Java	PosgreSQL	Spring boot
Gestion des Recommandations	Java	PosgreSQL	Spring boot

Table 1: Informations sur les Microservices

5 ANNEXES

5.1 API Utilisateur

Endpoints principaux

- `POST /Utilisateurs/inscription` : Permet à un utilisateur d'enregistrer ses informations.
- `POST /Utilisateurs/verification` : Permet à un utilisateur de s'inscrire après avoir renseigné un code OTP qu'il aura reçu par son mail.
- `POST /Utilisateurs/connexion` : Permet à un utilisateur de se connecter.
- `POST /Utilisateurs/reinscription` : Permet à un utilisateur de réinitialiser son mot de passe.
- `POST /Utilisateurs/reconnexion` : Permet à un utilisateur de notifier l'adresse email si son mot de passe a été oublié.
- `GET /Utilisateurs` : Récupère tous les utilisateurs.
- `GET /Utilisateurs/{id}` : Récupère les informations d'un utilisateur par son ID.
- `PUT /Utilisateurs/{id}` : Met à jour les informations d'un utilisateur.
- `DELETE /Utilisateurs/{id}` : Supprime un utilisateur.

Exemple détaillé de chaque endpoint

Inscription (Register) Requête :

- Méthode : `POST`
- URL : `/Utilisateurs/inscription`
- Corps de la requête (JSON) :

```
{
    "nom": "Zara",
    "prenom": "selena",
    "email": "Zatulia@gmail.com",
    "mot_passe": "pourri",
    "numerotelephone": 678921475,
    "pays": "Cameroun",
    "ville": "yaoundé",
    "quartier": "Acacias",
    "bp": "Acac"
}
```

Traitement côté backend :

1. Vérification des données reçues (validation des champs, email unique).
2. Génération d'un code OTP.
3. Envoi du code à l'email reçu.

Réponse (succès) :

- Code HTTP : 201 Created
- Corps :

```
{
    "message": "Un email de vérification a été envoyé."
}
```

Réponse (échec) :

- Code HTTP : 401 Unauthorized
- Corps :

```
{
    "message": "Cet email est déjà utilisé."
}
```

Vérification Requête :

- Méthode : `POST`
- URL : `/Utilisateurs/verification`
- Corps de la requête (JSON) :

```
{
    "email": "Zatulia@gmail.com",
    "code": 365894
}
```

Traitement côté backend :

1. Hachage du mot de passe avec bcrypt lors de l'enregistrement dans la BD.
2. Vérification du code OTP.
3. Enregistrement des données dans la base de données.

Réponse (succès) :

- Code HTTP : 201 Created
- Corps :

```
{
    "message": "Utilisateur enregistré avec succès.",
    "utilisateur": utilisateur
}
```

Réponse (échec) :

- Code HTTP : 401 Unauthorized
- Corps :

```
{
    "message": "Code de vérification incorrect."
}
```

5.2 API du Fournisseur

Endpoints principaux

- `POST /Fournisseurs` : Permet à un fournisseur de s'inscrire.
- `GET /Fournisseurs` : Récupère tous les fournisseurs.
- `GET /Fournisseurs/{id}` : Récupère les informations d'un fournisseur par son ID.
- `PUT /Fournisseurs/{id}` : Met à jour les informations d'un fournisseur.
- `DELETE /Fournisseurs/{id}` : Supprime un fournisseur.

5.3 API des Produits

Endpoints Principaux

- `PUT /produitService/removeQuantity/id` : Retire une quantité de produit spécifiée.
- `PUT /produitService/addQuantity/id` : Ajoute une quantité de produit spécifiée.
- `POST /produitService/updateProduit` : Met à jour les informations d'un produit.
- `POST /produitService/saveProduit` : Enregistre un nouveau produit.
- `GET /produitService/getProduit/id` : Récupère les informations d'un produit par son ID.
- `GET /produitService/getAllsubCategory/subcategory` : Récupère tous les produits d'une sous-catégorie spécifiée.
- `GET /produitService/getAllProduits` : Récupère tous les produits.
- `GET /produitService/getAllCategory/category` : Récupère tous les produits d'une catégorie spécifiée.

- `DELETE /produitService/deleteProduct/id` : Supprime un produit par son ID.

Schéma de Produit

Le schéma de l'objet `Produit` contient les propriétés suivantes :

- **id** (integer) : Identifiant unique du produit.
- **name** (string) : Nom du produit.
- **description** (string) : Description du produit.
- **price** (float) : Prix du produit.
- **quantity** (integer) : Quantité disponible.
- **category** (string) : Catégorie du produit (alimentaire, vêtement, cosmétique).
- **imageUrl** (string) : URL de l'image du produit.
- **subCategory** (string) : Sous-catégorie du produit (homme, femme, enfant, aucune).
- **idFournisseur** (integer) : Identifiant du fournisseur.

5.4 API de Livraison

5.5 API des Paiements

Endpoints Principaux

- **/payments/id**
  - *GET* : Rechercher un paiement avec son identifiant.
  - *PUT* : Mettre à jour des informations d'un paiement.
- **/payments/sendpayment**
  - *POST* : Envoyer les informations d'un paiement ayant le statut ACCEPTED.
- **/payments/savepayment**
  - *POST* : Enregistrer les informations d'un paiement.
- **/payments**
  - *GET* : Récupérer les informations sur tous les paiements initiés.
- **/payments/findbyorderid/order_id**
  - *GET* : Rechercher un paiement avec l'identifiant de la commande.
- **/cinetpaypayments/user_id/order_id**

  - *GET* : Récupération des informations d'un utilisateur et de sa commande à partir de leurs identifiants.

- **/cinetpaydeliverypayments/user_id/order_id**
  - *GET* : Récupérer les informations d'une commande et de l'utilisateur ayant choisi le mode de paiement à la livraison.

Schémas de Données

- **Payment**
  - **id** : Identifiant du paiement (integer)
  - **orderId** : Identifiant de la commande (integer)
  - **customer_id** : Identifiant du client (integer)
  - **amount** : Montant du paiement (integer)
  - **payment_method** : Méthode de paiement (string)
  - **payment_state** : État du paiement (string)
  - **payment_date** : Date du paiement (date)
  - **payment_time** : Heure du paiement (LocalTime)

- **CinetPayPaymentDto**
  - **apikey** : Clé API (string)
  - **amount** : Montant (integer)
  - **currency** : Devise (string)
  - **description** : Description (string)
  - **channels** : Canaux (string)
  - **transactionId** : Identifiant de transaction (string)
  - **customerId** : Identifiant du client (string)
  - **customerName** : Nom du client (string)
  - **customerSurname** : Prénom du client (string)
  - **customerPhoneNumber** : Numéro de téléphone (string)
  - **customerEmail** : Email du client (string)
  - **customerAddress** : Adresse (string)
  - **customerCity** : Ville (string)
  - **customerZipCode** : Code postal (string)
  - **siteId** : Identifiant du site (string)
  - **idPaiement** : Identifiant de paiement (string)
  - **returnUrl** : URL de retour (string)
  - **notifyUrl** : URL de notification (string)
  - **customerCountry** : Pays du client (string)
  - **customerState** : État du client (string)

5.6 API de recommandation des produits

Endpoints Principaux

- **/recommandations/saverecommandation**
  - *POST* : Sauvegarder une note.
  - **Description :** Enregistre en base de données la note d'un produit pour un utilisateur donné.
- **/recommandations**
  - *GET* : Retourne toutes les recommandations.
  - **Description :** Retourne les 15 produits ayant obtenu la meilleure moyenne de votes.
- **/recommandations/ratings/productId**
  - *GET* : Obtenir la note d'un produit.
  - **Description :** Retourne la moyenne des notes d'un produit en entrant son identifiant.

Schémas de Données

- **id** : Identifiant de la recommandation (integer)
- **userId** : Identifiant de l'utilisateur (integer)
- **productId** : Identifiant du produit (integer)
- **rating** : Note attribuée (integer, entre 1 et 5)

5.7 API des Commandes

Récupérer toutes les commandes

- **GET:** `/commande/historique-commandes`
- **Réponse:** 200 OK - Liste des commandes.

Récupérer une commande par ID

- **GET:** `/commande/historique-commandes/{id}`
- **Réponse:** 200 OK - Commande; 404 Not Found - Non trouvée.

Créer une commande

- **POST:** `/commande/historique-commandes/enregistrerCmd`
- **Corps:**

```
{
  "idPanier": int,
  "moyenPaiement": string,
  "statut": string
}
```

- **Réponse:** 201 Created - Commande créée.

Récupérer les commandes d'un utilisateur

- **GET:** `/commande/historique-commandes/utilisateur/{idUtilisateur}`
- **Réponse:** 200 OK - Liste des commandes de l'utilisateur.

Récupérer toutes les contenances

- **GET:** `/commande/contenances`
- **Réponse:** 200 OK - Liste des contenances.

Récupérer une contenance par ID

- **GET:** `/commande/contenances/{id}`
- **Réponse:** 200 OK - Contenance; 404 Not Found - Non trouvée.

Créer une contenance

- **POST:** `/commande/contenances`
- **Corps:**

```
{
  "idPanier": int,
  "idProduit": int,
  "quantite": int
}
```

- **Réponse:** 201 Created - Contenance créée; 404 Not Found - Panier non trouvé.

Mettre à jour une contenance

- **PUT:** `/commande/contenances/{id}`
- **Corps:**

```
{
  "idProduit": int,
  "quantite": int
}
```

- **Réponse:** 200 OK - Contenance mise à jour; 404 Not Found - Non trouvée.

Supprimer une contenance

- **DELETE:** `/commande/contenances/{id}`
- **Réponse:** 204 No Content - Contenance supprimée.

Table 2: Tableau de suivi des objectifs

Semaine	**Objectifs fixés**	**Réalisations**	**Difficultés**	**Solutions apportées**	**Scrum Master**
14 octobre	- Installation et configuration de l'environnement de développement - Identification des besoins et conception des diagrammes - Réalisation des opérations de CRUD de chaque microservice	Tous les objectifs definis initialement	Conception des diagrammes assez complexe	Redefinition des besoins	Dossivil Rivoire Vianey
28 octobre	- Revoir la conception architecturale - Finaliser le Back-end de chaque microservice - Mettre chaque microservice dans un docker tout en assurant l'indépendance des microservices - Tests individuels	- Gestion des sessions - Mise sur pied d'un système de notifications sychrone -Intégration de la traduction des pages	-Difficultés pour dockeriser les microservices -Difficulté pour gérer la résilience aux pannes	Documentation et tutoriels	Njumbe Sonia Tchomguem

Suite à la page suivante

Semaine	Objectifs fixés	Réalisations	Difficultés	Solutions apportées	Scrum Master
11 novembre	-Concevoir les maquettes de l'application -Dockeriser et push les microservices dans le repository -Débuter la réalisation du front end	-Conception et réalisation des maquettes de l'application - Début du front-end	- Pas de maitrise des outils de conception des maquettes et du framework React js	Documentation sur Réact et figma	Akono Noah Carole Jessica
25 novembre	- Clôturer les tests d'intégration - Débuter les tests de communication avec la Gateway - Finaliser le cahier des charges et la présentation - Concevoir l'API de recommandation et intégrer à l'application	- Réalisation de la présentation -Réalisation du systeme de recommandations -Finalisation du front-end des pages pour utilisateur -Rendre le front-end responsive	-Permettre au front-end d'interagir avec le back-end	Documentation sur ce sujet	Dassi Mandjo Lea Justine
9 décembre	-Test systeme et deploiement de l'application -Faire communiquer le front-end avec le back-end				Komadjou Tchaptche Imelda Shelvie