Architecture Boutique

I - Présentation

Aujourd'hui, nous discuterons de la conception d'une architecture robuste pour une boutique en ligne qui gère actuellement 5 000 produits et envisage une expansion à plusieurs millions de produits dans les années à venir. Notre approche repose sur un modèle de micro-services pour assurer une évolutivité et une maintenance efficaces, ainsi qu'un bus de messages pour une communication asynchrone et décentralisée entre les différents services. L'utilisation de bases de données adaptées aux grandes volumétries permet de gérer efficacement l'augmentation des données tout en améliorant les performances du système. Des mécanismes de cache améliorent la réactivité du site, tandis que des systèmes de paiement sécurisés assurent la protection des transactions financières. Notre objectif est de créer une expérience utilisateur fluide et personnalisée, tout en garantissant une sécurité maximale des données.

II - Référencement des caractéristiques architecturales

- **Performance** : Capacité à gérer rapidement les requêtes des utilisateurs pour une expérience fluide, surtout pendant les pics de trafic.
- **Scalabilité** : Aptitude du système à augmenter sa capacité de traitement et de stockage pour accompagner la croissance du nombre de produits et des transactions.
- **Sécurité** : Protection des données personnelles et financières des utilisateurs tout au long des processus d'authentification, de paiement et de gestion de compte.
- **Personnalisation** : Offrir des recommandations et promotions ciblées basées sur l'historique d'achat et de navigation des utilisateurs.

III - Décisions d'architecture

Micro-services

Avantages :

- Flexibilité de développement et de déploiement
- Scalabilité
- Performance
- Gestion des pics de trafic

Inconvénients :

- Complexité de gestion
- Infrastructure avancée

1. Choix du Back-End

a. Analyse des besoins

Nous aurons besoin d'un serveur qui devra correspondre à notre analyse des besoins .

(Cf : Caractéristiques Architecturales)

b. Découper le Back-End

Chaque service gère un domaine spécifique :

- Présentation de la boutique
- Gestion des utilisateurs
- Traitement des commandes
- Gestion d'inventaire
- Recommandation des produits
- Gestion des paiements
- Un service = Un serveur

c. Guider le choix technologique

- Langages et Frameworks
- Gestion des performances et de la charge (Synchrones, Asynchrones)
- Compatibilité avec les bases de données
- Support pour les API

d. Notre choix :

- **Node.js** : Léger, rapide, idéal pour des services nécessitant une faible latence.

2. Choix du type de base de données : NoSQL

Notre choix : Utiliser une base de données NoSQL, comme MongoDB ou Cassandra, pour gérer le catalogue de produits et les interactions des utilisateurs.

Justification :

- **Scalabilité** : Les bases de données NoSQL offrent une excellente scalabilité horizontale, ce qui est essentiel pour gérer l'augmentation exponentielle du nombre de produits et de transactions. Elles permettent d'ajouter plus de serveurs facilement pour distribuer la charge.
- **Performance** : Elles sont conçues pour de grandes volumétries de données et des accès rapides, grâce à leurs mécanismes de stockage optimisés pour des requêtes simples et des écritures rapides.

- **Flexibilité des schémas** : La flexibilité de leurs schémas est idéale pour le catalogue de produits qui peut évoluer, ajoutant de nouveaux attributs sans perturber le système existant.

3. Stratégie de mise en cache : Cache distribué

Notre choix : Implémenter un système de cache distribué, comme Redis pour cacher les données de produits fréquemment accédées.

Justification :

- **Amélioration des performances** : En stockant les données fréquemment demandées en mémoire, les temps de réponse sont considérablement réduits, ce qui est crucial pour l'expérience utilisateur lors de la navigation et de la recherche de produits.
- **Réduction de la charge sur la base de données** : Le cache minimise le nombre de requêtes directes à la base de données, prolongeant ainsi sa durabilité et réduisant les coûts d'infrastructure nécessaires pour gérer les pics de charge.

4. Gestion de la personnalisation des recommandations : Apprentissage automatique

Notre choix : Déployer des modèles d'apprentissage automatique pour personnaliser les recommandations de produits basées sur le comportement des utilisateurs.

Justification :

- **Engagement utilisateur** : En fournissant des recommandations pertinentes, on augmente la probabilité de conversion des visiteurs en acheteurs récurrents.
- **Exploitation des données** : Utiliser les données collectées (historique de navigation, achats précédents, préférences) pour affiner continuellement les recommandations, créant une expérience d'achat dynamique et adaptée.

5. Approche de sécurité pour les paiements : Intégration d'un service de paiement tiers

Notre choix : Utiliser des solutions de paiement de tiers sécurisées, telles que Stripe ou PayPal, qui respectent les normes **PCI DSS***.

** Les normes **PCI DSS (Payment Card Industry Data Security Standard)** sont un ensemble de directives et de mesures de sécurité établies pour protéger les donnéesdes cartes de paiement et réduire les fraudes. Elles obligent toutes les entités traitantes, stockant ou transmettant des informations de carte de crédit à maintenir un environnement sécurisé.*

Justification :

- **Sécurité des données** : Ces services sont spécialisés dans la sécurisation des transactions et la protection des données financières, ce qui réduit le risque de violation de données.
- **Conformité réglementaire** : Ils assurent une conformité avec les normes internationales de sécurité des paiements, déchargeant la boutique de la gestion complexe de la conformité PCI DSS.

6. Choix d'un bus de messages : Architecture orientée événements

Notre choix : Adopter une architecture orientée événements en intégrant **un bus de messages*** comme **Kafka*** pour gérer la communication asynchrone entre les services.

Un **bus de messages est un système qui facilite la communication entre différentes parties d'une application en permettant à des composants logiciels de s'envoyer des messages de manière asynchrone. Cela signifie que les composants n'ont pas besoin d'interagir directement les uns avec les autres, mais peuvent échanger des informations via un canal central qui transporte les messages d'un point à un autre.*

***Kafka** est un exemple de technologie qui implémentent un bus de messages :*

- ***Kafka** est un système de messagerie distribué, orienté flux, qui est conçu pour gérer de grands volumes de données en temps réel. Il est souvent utilisé pour construire des applications qui nécessitent une analyse des données en temps réel ou qui intègrent des systèmes en nécessitant une grande capacité de transmission.*

Justification :

- **Découplage des services** : Permet une communication efficace entre les micro-services sans créer de dépendances directes entre eux, facilitant ainsi la maintenance et l'évolutivité.
- **Fiabilité et intégrité des données** : Le bus de messages garantit que les données sont synchronisées entre les services, même en cas de défaillance d'un

composant, par le biais de mécanismes de reprise sur erreur et de transactions persistantes.

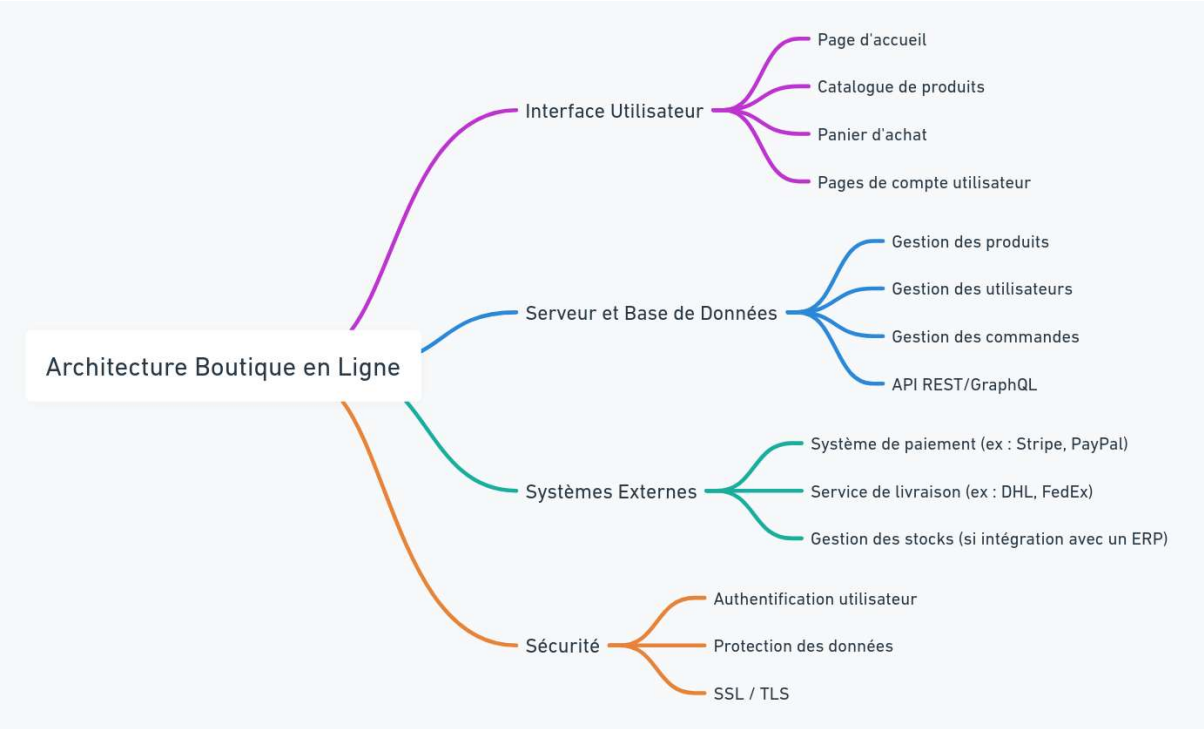
7. Hébergement

L'hébergement joue un rôle crucial dans la performance, la disponibilité, et la scalabilité d'une boutique en ligne. Voici un guide détaillé pour choisir et configurer un hébergement adapté à une architecture en micro-services.

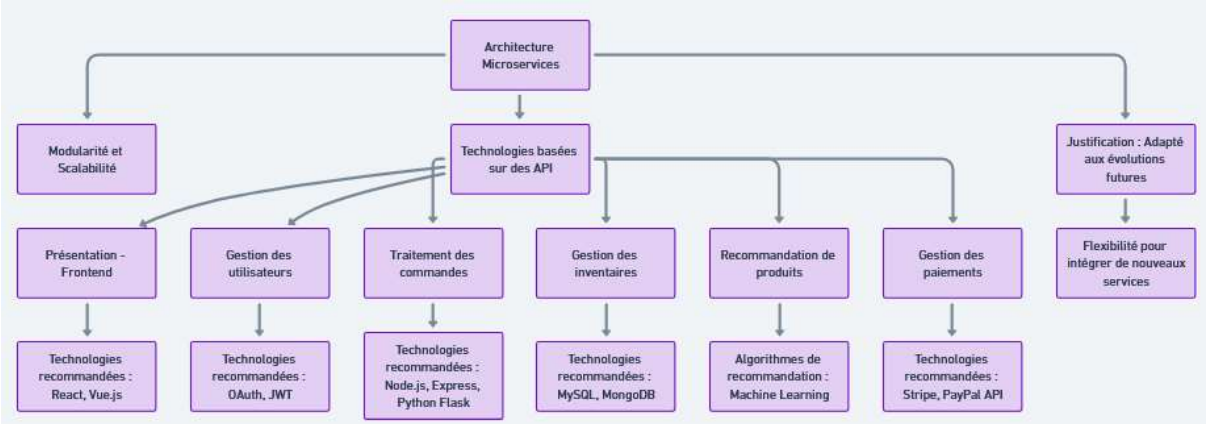
Notre choix : Simple, nous allons utiliser AWS pour leur large éventail de services en micro-services.

IV - Schéma des composants

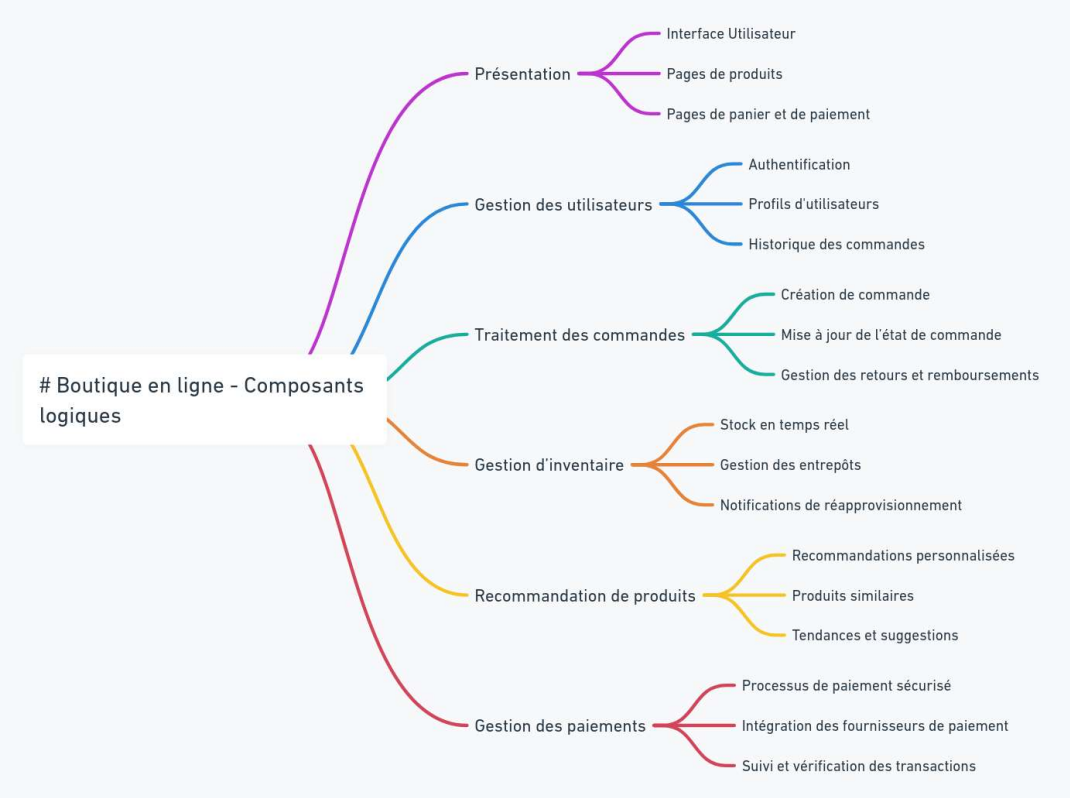
a. Schéma architectural :



b. Diagramme choix architecture :



c. Diagramme de composants logiques :



V – Conclusion

La conception architecturale mise en place pour cette boutique en ligne combine **robustesse**, **réactivité** et **sécurité** optimale. En adoptant une architecture de micro-services et un bus de messages, nous garantissons une évolutivité et une résilience élevées, essentielles pour accompagner la croissance rapide de l'entreprise.

L'utilisation de bases de données NoSQL et de systèmes de cache distribué assure des performances accrues, avec des temps de réponse rapides, cruciaux pour une expérience utilisateur fluide. Enfin, l'intégration de solutions de **paiement sécurisées** conforme aux normes protège les transactions financières, renforçant la confiance des clients. Cette architecture est donc parfaite pour soutenir les objectifs à long terme de la boutique tout en assurant une **gestion efficace et sécurisée des données utilisateur** et des transactions.

Document réalisé par :

LASSAL Shun
CHATELAIN Dylan
GROSSI Julia
LAFOSSE Manon
VITRAT Clement
FREGONESE Tom
CHAMANIER Enzo