TP1 - Conception d’un système de gestion de boutique en ligne

Table des matières

1. Contexte

Contexte de la boutique en ligne

1.1. Catalogue et produits :

Volumétrie actuelle : La boutique propose actuellement une sélection de **5 000 produits.**

Objectif de croissance : D'ici deux ans, le catalogue devrait atteindre **plusieurs millions de produits,** et le système doit être prêt à gérer ce volume.

Mise à jour fréquente des produits : Des modifications, telles que des changements de prix, de description et de disponibilité, peuvent se produire à tout moment.

1.2. Expérience utilisateur :

Personnalisation : L'interface doit proposer des recommandations de produits et des promotions personnalisées pour chaque utilisateur, basées sur leur historique de navigation et d'achat.

Réactivité : Le système doit offrir des temps de réponse rapides, notamment lors de la consultation du catalogue, l'ajout au panier, et le passage de commande.

Gestion des comptes utilisateurs : Le système doit permettre aux utilisateurs de créer un compte, de suivre leurs commandes et de gérer leurs informations personnelles de manière sécurisée.

1.3. Volumes de transactions et trafic :

Trafic estimé : Initialement modéré, mais avec des pics potentiels pendant les périodes de soldes et de fêtes.

Objectif de disponibilité : La boutique doit être accessible en continu, avec une tolérance de quelques minutes de temps d'arrêt maximum par mois.

Paiements : Le système doit gérer plusieurs options de paiement (carte de crédit, PayPal, cartes cadeaux) et garantir la sécurité des transactions.

1.4. Perspectives d'évolution :

D'ici trois ans, l'entreprise prévoit **d'intégrer des points de vente physiques** avec un inventaire synchronisé.

La boutique souhaite également proposer des services supplémentaires, tels que **des notifications de réapprovisionnement** et **des abonnements mensuels** pour certains produits.

2. Référencement des caractéristiques architecturales

Pour la réalisation de ce projet de boutique en ligne, nous avons identifié une multitudes de caractéristiques importantes qui ont un impacte sur l’architecture du projet. Ci-dessous la liste des caractéristiques détaillées :

2.1. Robustesse

La boutique doit pouvoir gérer des volumes de transactions ou bien de trafic croissant sans défaillance durant des périodes de pics d’activités. Par exemple lors des périodes de soldes ou bien de fêtes qui cause des pics d’activités. Une architecture robuste doit obligatoirement garantir aucun dysfonctionnement majeur durant les pics d’activités.

2.2. Disponibilité et Fiabilité

La boutique doit être accessible en continue avec une tolérance faible au temps d’arrêt de fonctionnement avec quelques minutes par mois d’arrêt. Dans la mesure où l’arrêt peut entraîner une perte de vente mais également nuire à la réputation de la boutique.

2.3. Scalabilité

Le catalogue doit évoluer pour passer de 5000 produits à plusieurs millions de produits dans deux ans. Le système doit également être capable de s’adapter au flux croisant d’utilisateurs. Il doit donc être capable de gérer un nombre important de requêtes simultanées toute en restant performant.

2.4. Sécurité

Le système doit gère des payements (carte de crédit, Google pay, Apple pay, etc…), des informations utilisateurs sensibles donc le système doit protéger et sécuriser les données mais également lutter contre les attaques et fuite de données.

2.5. Extensibilité

Le système doit pouvoir intégrer de nouvelle fonctionnalité et évoluer pour répondre aux besoins des utilisateurs mais également aux évolution du secteur de la vente en ligne. Dans trois ans, il faudra synchroniser l’inventaire de la boutique en ligne avec les futurs points de ventes physiques. L’architecture doit faciliter l’ajout de nouveaux services sans perturber l’existant.

2.6. Portabilité

Le système doit être déployable sur plusieurs environnements (cloud ou sur site). Cela garantit une plus grande flexibilité technique. La boutique pourrait avoir besoin de migrer son infrastructure ou de s'adapter à un nouveau fournisseur cloud à l'avenir.

2.7. Agilité

L’équipe doit pouvoir apporter des modifications rapidement pour répondre aux évolutions des besoins métier (ex : nouvelles promotions, mises à jour de produits). Les prix et descriptions des produits changent fréquemment. Le système doit permettre des mises à jour rapides sans altérer les performances.

3. Création de décisions d’architecture

D’après les caractéristiques architecturale, nous avons pue prendre des décisions architecturales. Ci-dessous la liste des différentes décisions architecturales qui nous à poussé pour les choix technologiques.

3.1. Décision : Choix d'une base de données pour catalogue produits

- **Titre :** Utilisation d'une base NoSQL pour le catalogue des produits.
- **Contexte :**
  - Le catalogue doit gérer des millions de produits.
  - Les mises à jour fréquentes nécessitent de la flexibilité.
- **Décision :** Utiliser MongoDB pour le catalogue des produits.
- **Options considérées :**
  - MongoDB : Flexible, performant pour de grandes volumétries, scalabilité horizontale.
  - PostgreSQL : Solide et fiable, mais plus rigide pour les mises à jour fréquentes.

- **Justification :** MongoDB permet de scaler horizontalement et de gérer des documents complexes, ce qui est idéal pour un catalogue évolutif.
- **Conséquences :**
  - Positif : Facilité de gestion d'un grand volume de produits.
  - Négatif : Nécessite une équipe formée sur NoSQL.

3.2. Décision : Gestion de la sécurité des paiements

- **Titre :** Utilisation d'un fournisseur de paiement sécurisée.
- **Contexte :** Garantir la sécurité des transactions financières.
- **Décision :** Intégrer Stripe comme fournisseur de paiement externe.
- **Options considérées :**
  - Fournisseur Stripe : Sécurisé, conforme PCI-DSS, API simple à intégrer.
  - Fournisseur PayPal : Réputé, mais des frais plus élevés pour l'entreprise.
- **Justification :** Stripe offre une meilleure flexibilité avec un coût plus faible pour les transactions.
- **Conséquences :**
  - Positif : Paiements sécurisés et conformité garantie.
  - Négatif : Dépendance vis-à-vis d'un fournisseur externe.

3.3 Décision : Choix d'une architecture microservices

- **Titre** : Mise en place d'une architecture adaptée.
- **Contexte** : La boutique en ligne doit évoluer pour gérer un catalogue de plusieurs millions de produits, des pics de trafic et l'ajout futur de nouvelles fonctionnalités comme l'intégration de points de vente physiques.
- **Décision** : Adopter une architecture microservices
- **Options considérées** :
  - Architecture microservice :
  - Architecture monolithique :
- **Justification** : Une architecture microservices garantit la scalabilité, la modularité et la flexibilité nécessaires pour accompagner la croissance du système tout en facilitant l'ajout de nouvelles fonctionnalités.
- **Conséquences** :
  - Positif : Évolutivité, résilience, déploiements indépendants des composants.

- Négatif : Complexité initiale et coûts supplémentaires au début du projet.

3.4 Décision : Stratégie de mise en cache

- **Titre** : Utilisation de Redis pour la mise en cache des produits.
- **Contexte** : Les temps de réponse doivent être rapides pour la consultation du catalogue pour éviter de faire attendre le client.
- **Décision** : Intégrer Redis comme solution de mise en cache.
- **Options considérées** :
  - Redis : Faible latence, haute disponibilité.
  - Memcached : Bonne performance, mais fonctionnalités limitées.
- **Justification** : Redis offre des performances élevées et permet de gérer des structures complexes.
- **Conséquences** :
  - Positif : Amélioration des temps de réponse.
  - Négatif : Complexité supplémentaire dans l'architecture.

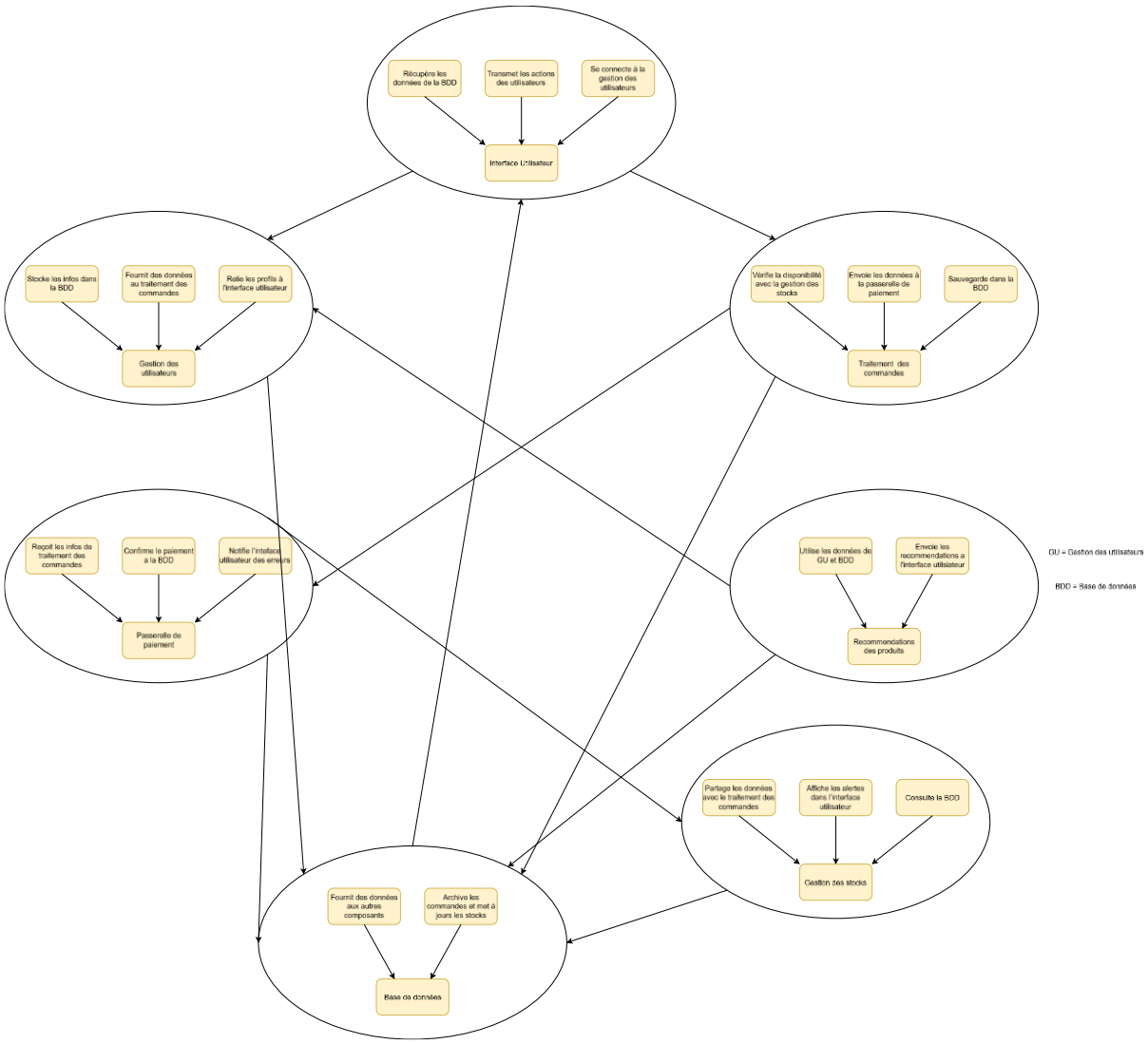
3.5 Décision : Choix de la base de données pour l'authentification et la gestion des utilisateurs

- **Titre** : Utilisation d'une base de données relationnelle pour l'authentification et les comptes utilisateurs.
- **Contexte** : Le système doit gérer les comptes utilisateurs de manière sécurisée, fiable et performante. Ces données nécessitent cohérence, intégrité et durabilité pour garantir que les utilisateurs puissent interagir avec la boutique de manière fluide et sécurisée soit profiter des principes ACID.
- **Décision** : Utilisations d'une base de données PostgreSQL, une base de données relationnelle, pour stocker les informations d'authentification et les données critiques des utilisateurs.
- **Options considérées** :
  - SQLite (Relationnelle légère)
  - PostgreSQL (Relationnelle)
- **Justification** : PostgreSQL a été choisi pour la gestion de l'authentification et des comptes utilisateurs en raison de sa capacité à garantir la cohérence et l'intégrité

des données grâce aux propriétés ACID. Il offre également des performances élevées pour les requêtes transactionnelles complexes et une évolutivité robuste pour accompagner la croissance du nombre d'utilisateurs et de transactions.

- **Conséquences** :
    - Positif : Intégrité et cohérence garanties pour les données critiques. Sécurité renforcée pour les opérations d'authentification.
    - Négatif : Nécessite une configuration initiale plus lourde.

4. Schéma des composants logiques



5. Choix du style architectural et justification

Pour concevoir le système de gestion de la boutique en ligne, nous avons choisi d'adopter une architecture microservices. Ce choix repose sur les exigences identifiées, notamment la scalabilité, la performance, la robustesse et l’extensibilité du système.

En premier lieu, la boutique prévoit une forte croissance de son catalogue, passant de 5 000 produits à plusieurs millions d’ici deux ans. Une architecture microservices permet de scaler horizontalement chaque service de manière indépendante pour répondre à cette augmentation de volume et aux pics de trafic, notamment pendant les périodes critiques comme les soldes ou les fêtes.

Ensuite, la performance et la réactivité sont des critères essentiels pour garantir une bonne expérience utilisateur. Grâce à une architecture modulaire, les services critiques comme la consultation du catalogue, l’ajout au panier ou la finalisation des commandes peuvent être optimisés individuellement. De plus, des solutions telles que la mise en cache (par exemple, avec Redis) permettront de réduire les temps de réponse.

L’extensibilité constitue également un point clé pour l’évolution future du système. L’entreprise prévoit d’intégrer des points de vente physiques avec un inventaire synchronisé et d’ajouter des fonctionnalités telles que les notifications de réapprovisionnement ou les abonnements mensuels. L’architecture microservices facilite l’ajout de nouvelles fonctionnalités ou l’évolution des composants existants sans perturber l’ensemble du système.

En matière de robustesse, cette architecture garantit que les défaillances sont isolées. Par exemple, si un microservice comme le moteur de recommandations rencontre un problème, les autres services tels que la gestion des paiements ou des commandes restent fonctionnels, assurant ainsi une disponibilité élevée du système.

Enfin, la sécurité est renforcée grâce à l’isolation des services sensibles. Des mesures spécifiques peuvent être appliquées, notamment sur les services critiques comme la gestion des paiements pour assurer la sécurité des transactions et des données.

Cependant, il est important de noter que l’architecture microservices présente quelques inconvénients. Sa mise en place est plus complexe qu’une architecture monolithique, car elle nécessite des outils d’orchestration (comme un API Gateway ou un orchestrateur comme Kubernetes) et une communication inter-services robuste. Cette complexité s’accompagne également de coûts accrus pour la gestion et le monitoring des microservices. Enfin, les communications réseau entre services peuvent introduire une légère latence.

En conclusion, malgré ces défis, l’architecture microservices est le style le plus adapté pour répondre aux exigences et aux contraintes du projet. Elle offre une scalabilité, une performance et une flexibilité essentielles pour accompagner la croissance de la boutique en ligne et garantir une évolutivité à long terme.

