Conception d’un système de gestion de boutique en ligne

Groupe : Jopcelin, Olfa, Mezen, David

Référencement des caractéristiques architecturales :

Sécurité : La sécurisation des achats est essentielle pour une application de boutique en ligne. L’objectif étant de protéger les données sensibles (personnel, information de paiement) et se protéger des cyberattaques.

Performance & Flexible : Il faut que l’application soit fluide et rapide, avec des temps de réponse optimaux et les mises à jour doivent être simples et efficaces.

Disponibilité & Déployabilité & Robustesse : L'application doit fonctionner 24/7 sans incidents et permettre une mise en ligne rapide pour limiter au maximum les interruptions.

Scalabilité : la scalabilité est indispensable pour gérer l’afflux d’utilisateurs lors des soldes ou autres, ainsi que l’augmentation du nombre de produits assurant une expérience fluide quel que soit le volume de trafic ou de données.

Personnalisation de l'expérience utilisateur & Utilisabilité : La boutique doit offrir une navigation fluide, avec une interface réactive, et avec une navigation optimisée. Un algorithme de recommandation devrait être également intégré pour augmenter les chances d'achat et la satisfaction client.

Maintenabilité : L'architecture doit être suffisamment flexible pour intégrer de nouvelles fonctionnalités, tout en permettant des évolutions futures sans perturber le fonctionnement existant.

Création de décisions d’architecture :

Choix de base de données : NoSQL MongoDB

Justification : Car on a un grand flux de données

Impact : Une base NoSQL offre une meilleure performance et scalabilité horizontale, mais pourrait manquer de rigueur pour des données nécessitant de fortes relations.

Sécurité des données : JWT

Justification : L'authentification et l'autorisation permettent de sécuriser les échanges sans nécessité de stockage de sessions sur le serveur. Chaque utilisateur obtient un jeton unique et sécurisé après s'être connecté.

Impact : Diminue la circulation des identifiants utilisateurs pour réduire les risques. Mais il faut un système de renouvellement pour prévenir les déconnexions lorsque le jeton expire.

Technologies de front-end et de back-end : Front (React) / Back (Springboot)

Justification :

React : Plus flexible et plus rapide que d’autres ce qui est Idéal pour créer des interfaces réactives et intuitives. Grâce à son architecture basée sur des composants réutilisables, React optimise les performances et facilite la maintenance.

Spring Boot : Il garantit une sécurité renforcée grâce à ses outils intégrés (Spring Security), et il offre aussi une solution robuste et performante idéal pour les projets à grande échelle.

Impact : Séparation claire entre front et back, facilitant le développement, le test, et la maintenance. Cette architecture permet aux équipes de travailler en parallèle sur les deux parties et améliore la scalabilité du projet et permet facilement la communication entre les deux, par une API REST. Mais il faut gérer deux environnements distincts avec des configuration parfois différentes cela augmente la charge de travail des développeurs.

Choix de modèles architectural : MVC

Justification : Il permet une organisation claire et modulaire du code et en plus il est pris en charge par Spring Boot, qui offre des outils efficaces pour sécuriser et gérer l'application.

Impact : Le code est mieux organisé, ce qui simplifie les modifications et les mises à jour. Mais nécessite une configuration et une organisation de projet plus rigoureuses pour bien structurer chaque couche.

Gestion de la personnalisation et des recommandations : Algo de recommandation

Justification : améliore l'expérience utilisateur en proposant des produits pertinents basés sur l'historique de navigation et d'achats.

Impact : Augmente les interactions avec les utilisateurs, et rends la boutique plus attractive, mais il peut proposer des articles non-pertinents.

Choix du style architectural et justification :

Choix : Microservice

Justification : L'architecture microservices a été choisie pour cette boutique en ligne en raison de nos choix de caractéristiques architecturaux (scalabilité, maintenabilité, déployabilité etc...). Cette architecture permet également une flexibilité pour ajouter de nouvelles fonctionnalités sans perturber le système, vu que chaque microservice est développé indépendamment des autres. Cependant, les microservices entraînent une complexité accrue, notamment en raison de la gestion des communications entre services et des transactions distribuées. De plus, les coûts peuvent augmenter, et la gestion des API et des tests devient plus complexe à mesure que le nombre de services augmente. Malgré tout ça, l'architecture microservices reste adaptée aux besoins de la boutique en ligne.

Schéma des composants logiques :