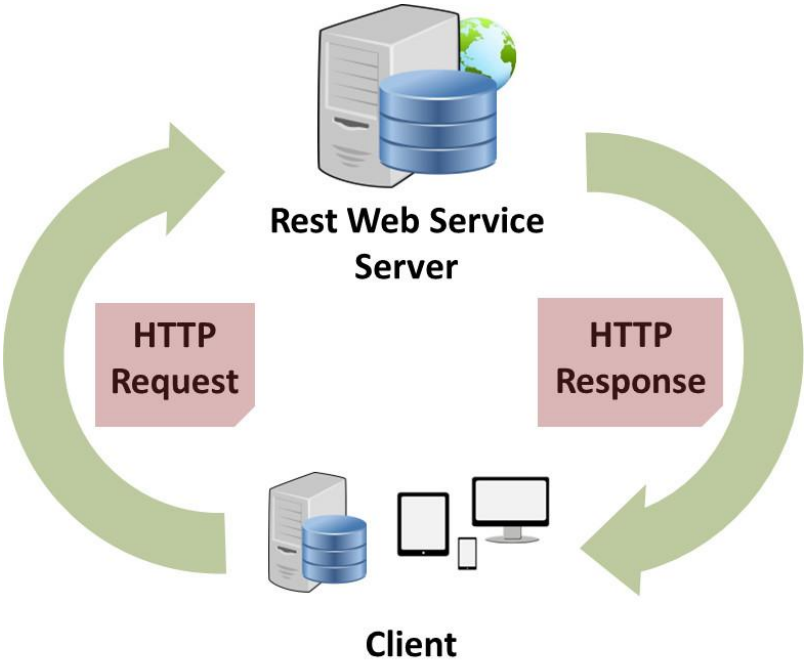


Fait par :	Professeur	Date
Romain REN **Jacq VENGADESAN**	**Ralph BOU NADER**	04/12/2025

Introduction

Dans le cadre de ce projet, il a été demandé de concevoir et de développer un site de commerce en ligne moderne pour l'année 2025. Ce projet a pour objectif principal de mettre en œuvre une architecture complète de type full-stack, intégrant un frontend, un backend et une base de données, tout en respectant les bonnes pratiques actuelles de développement web et d'API.

Le site e-commerce développé permet aux utilisateurs de consulter un catalogue de produits, de gérer un panier, de passer des commandes, de suivre leurs achats et d'interagir avec la plateforme via un système d'avis. Une attention particulière a été portée à la création d'API internes robustes, sécurisées et documentées, ainsi qu'à l'intégration d'une API externe de géolocalisation afin d'enrichir l'expérience utilisateur. Le projet inclut également un système de recommandations de produits, correspondant aux nouveaux usages attendus pour un site e-commerce en 2025.

Objectifs du projet

Les objectifs principaux de ce projet sont les suivants :

- Concevoir une application e-commerce complète couvrant l'ensemble du parcours utilisateur, de l'inscription jusqu'au suivi des commandes.
- Développer une API REST permettant la gestion des utilisateurs, des produits, des commandes, des paiements, des livraisons et des avis clients.
- Mettre en place un système d'authentification sécurisé basé sur des jetons JWT.
- Intégrer une API externe de géolocalisation afin de proposer des points de retrait proches de l'utilisateur.
- Implémenter un système de recommandations de produits basé sur l'historique d'achat, la popularité et les avis clients.
- Documenter l'ensemble des API à l'aide de Swagger et valider leur fonctionnement à l'aide de Postman.

Architecture globale de l'application

Architecture côté backend

Le backend est construit autour du framework Express. Il constitue le cœur fonctionnel de l'application et gère les règles métier, la sécurité et les échanges avec la base de données.

L'organisation interne du backend repose sur une structure modulaire :

- Un point d'entrée principal (index.js) qui initialise le serveur Express, configure les middlewares globaux (CORS, parsing JSON, sécurité) et monte les différentes routes.
- Un dossier routes qui regroupe l'ensemble des routes API, organisées par domaine fonctionnel (authentification, produits, commandes, avis, recommandations, géolocalisation).
- Un dossier middlewares contenant les middlewares de sécurité, notamment le middleware d'authentification JWT et les contrôles de rôles et de permissions.
- Un dossier utils dédié aux fonctions utilitaires, telles que la validation des données ou la gestion de règles communes.
- Une configuration Swagger permettant de générer automatiquement la documentation de l'API.
- Un point d'entrée GraphQL (graphql.js) qui offre une alternative à l'API REST pour certaines opérations.

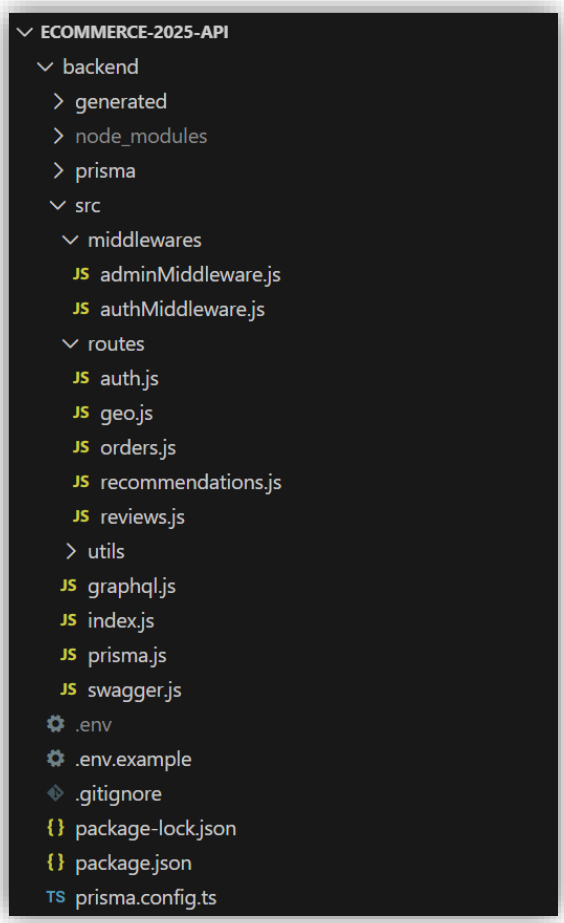
Cette organisation permet de séparer clairement les responsabilités et de limiter les dépendances entre les différentes parties du code.

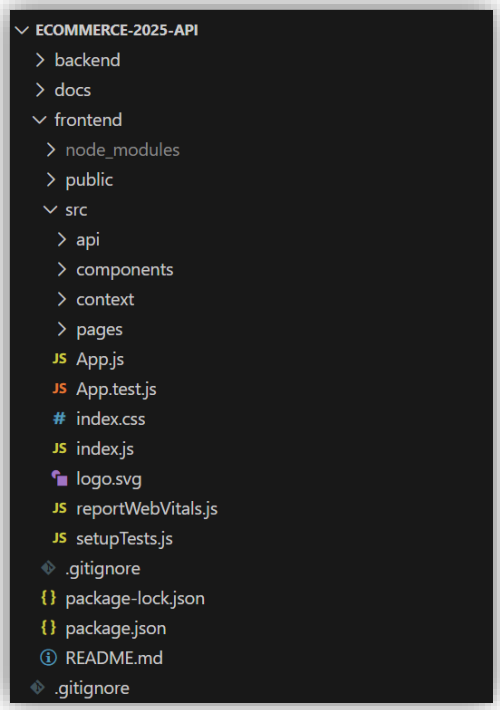
Architecture côté frontend

Le frontend est développé avec React et repose sur une architecture en composants. Chaque page de l'application correspond à un composant React distinct (connexion, inscription, catalogue, panier, commandes, profil, géolocalisation, recommandations).

L'état global du panier est géré via un contexte React, ce qui permet de partager les données du panier entre plusieurs composants sans duplication de logique. Les appels vers le backend sont centralisés dans un client Axios configuré pour ajouter automatiquement le token JWT dans les en-têtes HTTP lorsque l'utilisateur est authentifié.

Le frontend ne contient aucune logique métier sensible. Toutes les validations critiques et règles de sécurité sont implémentées côté backend, ce qui garantit que les règles ne peuvent pas être contournées par un utilisateur malveillant.





Documentation et outils annexes

La documentation de l'API est générée automatiquement grâce à Swagger. Elle est accessible via une route dédiée et permet de visualiser l'ensemble des endpoints, leurs paramètres et leurs réponses. Swagger constitue un outil essentiel pour les développeurs souhaitant comprendre ou consommer l'API.

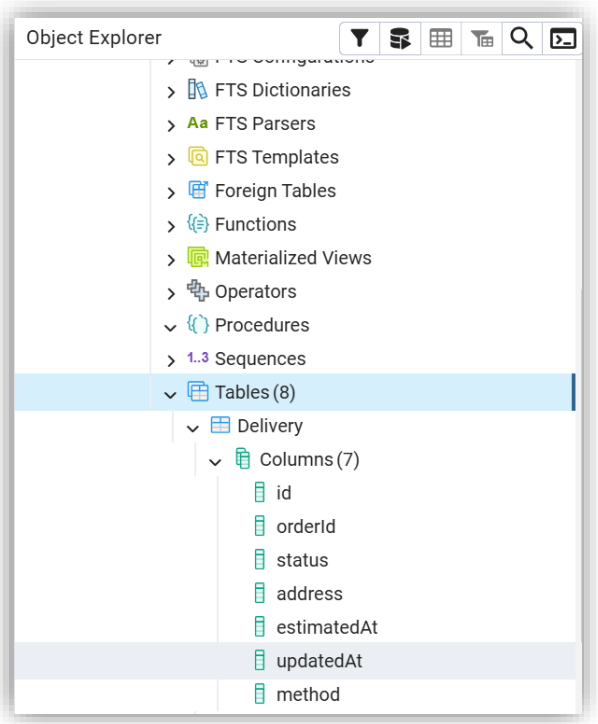
Les tests des endpoints ont été réalisés à l'aide de Postman. Cet outil a permis de vérifier le bon fonctionnement des routes, la gestion des erreurs et la sécurité des accès, notamment pour les routes protégées par authentification.

Base de données et gestion des données

Choix de la base de données

Le projet utilise **PostgreSQL** comme système de gestion de base de données. Ce choix s'explique par la nature relationnelle des données d'un site e-commerce. En effet, une application de ce type manipule de nombreuses entités fortement liées entre elles, telles que les utilisateurs, les produits, les commandes, les articles commandés, les paiements, les livraisons et les avis clients.

PostgreSQL offre une excellente gestion des relations, des contraintes d'intégrité et des transactions, ce qui est essentiel pour garantir la cohérence des données, notamment lors de la création ou de l'annulation de commandes.



Utilisation de Prisma ORM

L'accès à la base de données est assuré par **Prisma ORM**. Prisma agit comme une couche d'abstraction entre le code backend et la base PostgreSQL. Il permet de manipuler les données sous forme d'objets JavaScript plutôt que d'écrire directement des requêtes SQL.

Prisma présente plusieurs avantages dans le cadre de ce projet :

- Il centralise la définition des modèles de données dans un fichier unique (schema.prisma).
- Il génère automatiquement un client typé facilitant l'écriture de requêtes.
- Il gère les migrations de la base de données.
- Il protège l'application contre les injections SQL grâce à l'utilisation de requêtes paramétrées.

L'utilisation de Prisma améliore la lisibilité du code, réduit les risques d'erreurs et facilite la maintenance et l'évolution du schéma de données.

Schéma de données

Le schéma de la base de données est défini dans le fichier schema.prisma. Il décrit l'ensemble des modèles utilisés par l'application, leurs champs, leurs types, leurs relations et leurs contraintes.

Les principaux modèles sont les suivants :

Utilisateur (User)
Le modèle utilisateur représente les comptes clients de la plateforme. Il contient un identifiant unique, une adresse email unique, un mot de passe hashé, un nom, un rôle (utilisateur ou administrateur) et une date de création. Un utilisateur peut être lié à plusieurs commandes et plusieurs avis.

Produit (Product)
Le modèle produit représente les articles disponibles à la vente. Il contient un nom, une description, un prix, un stock, une catégorie et une date de création. Un produit peut être associé à plusieurs articles de commande et à plusieurs avis clients.

Commande (Order)
Le modèle commande représente une transaction passée par un utilisateur. Il contient un statut (en attente, annulée, etc.), un montant total et une date de création. Une commande est liée à un utilisateur, à plusieurs articles de commande, et possède une relation un-à-un avec un paiement et une livraison.

Article de commande (OrderItem)
Ce modèle sert de table de liaison entre les commandes et les produits. Il permet de stocker la quantité commandée et le prix du produit au moment de l'achat, ce qui garantit la cohérence des données même si le prix du produit change ultérieurement.

Avis (Review)
Le modèle avis permet aux utilisateurs de laisser une note et un commentaire sur un produit. Il est lié à un utilisateur et à un produit. Les avis sont utilisés à la fois pour l'affichage côté frontend et pour alimenter le système de recommandations.

Paiement (Payment)
Le modèle paiement représente l'état du paiement associé à une commande. Il contient un montant, un statut (en cours, annulé, etc.) et une date de création. Chaque commande possède au maximum un paiement associé.

Livraison (Delivery)
Le modèle livraison permet de suivre l'état de la livraison d'une commande. Il contient un statut, un mode de livraison (livraison à domicile ou point de retrait), une adresse éventuelle et une date estimée de livraison.

Relations entre les modèles

Les relations entre les modèles sont conçues pour refléter fidèlement le fonctionnement d'un site e-commerce :

- Un utilisateur peut passer plusieurs commandes.
- Une commande appartient à un seul utilisateur.
- Une commande contient plusieurs articles de commande.
- Un article de commande relie une commande à un produit.
- Un produit peut apparaître dans plusieurs commandes.
- Un produit peut recevoir plusieurs avis.
- Un utilisateur peut publier plusieurs avis.
- Chaque commande possède un paiement unique et une livraison unique.

Ces relations garantissent une structure cohérente et facilitent les requêtes complexes, telles que la récupération de l'historique des commandes d'un utilisateur avec le détail des produits, du paiement et de la livraison.

Gestion des transactions et de la cohérence des données

Lors de la création d'une commande, plusieurs opérations critiques doivent être effectuées de manière atomique :

- vérification de la disponibilité du stock,
- création de la commande,
- création des articles de commande,
- création des enregistrements de paiement et de livraison,
- mise à jour du stock des produits.

Ces opérations sont réalisées à l'aide de **transactions Prisma**, ce qui garantit que l'ensemble des actions est exécuté complètement ou annulé en cas d'erreur. Cette approche est essentielle pour éviter les incohérences, telles qu'une commande créée sans mise à jour du stock.

Développement des API internes (REST)

Présentation générale des API

Le cœur fonctionnel de l'application repose sur un ensemble d'API internes développées avec Node.js et Express. Ces API exposent les fonctionnalités principales du site e-commerce et servent d'interface entre le frontend React, la base de données PostgreSQL et les services externes.

Les routes sont organisées par domaines fonctionnels (authentification, produits, commandes, avis, recommandations, géolocalisation), ce qui facilite la lisibilité, la maintenance et l'évolution du code. Chaque route applique des règles de validation, de sécurité et de contrôle d'accès adaptées à son usage.

API d'authentification et de gestion des utilisateurs

Les API d'authentification permettent la gestion complète du cycle de vie d'un utilisateur.

L'inscription permet de créer un nouveau compte utilisateur. Les données saisies sont validées côté serveur, et le mot de passe est systématiquement hashé à l'aide de l'algorithme bcrypt avant d'être stocké en base de données.

La connexion permet à un utilisateur authentifié de recevoir un jeton JWT. Ce jeton contient l'identifiant de l'utilisateur et une date d'expiration. Il est ensuite utilisé pour accéder aux routes protégées de l'application.

Une fois connecté, l'utilisateur peut consulter son profil, le modifier (nom, email ou mot de passe) ou supprimer son compte. Toutes ces opérations nécessitent un jeton JWT valide.

La déconnexion est gérée via un mécanisme de révocation de jeton, basé sur l'identifiant unique du token (jti), afin d'empêcher son utilisation ultérieure.

Ces routes garantissent que seules les personnes authentifiées peuvent accéder ou modifier leurs propres données.

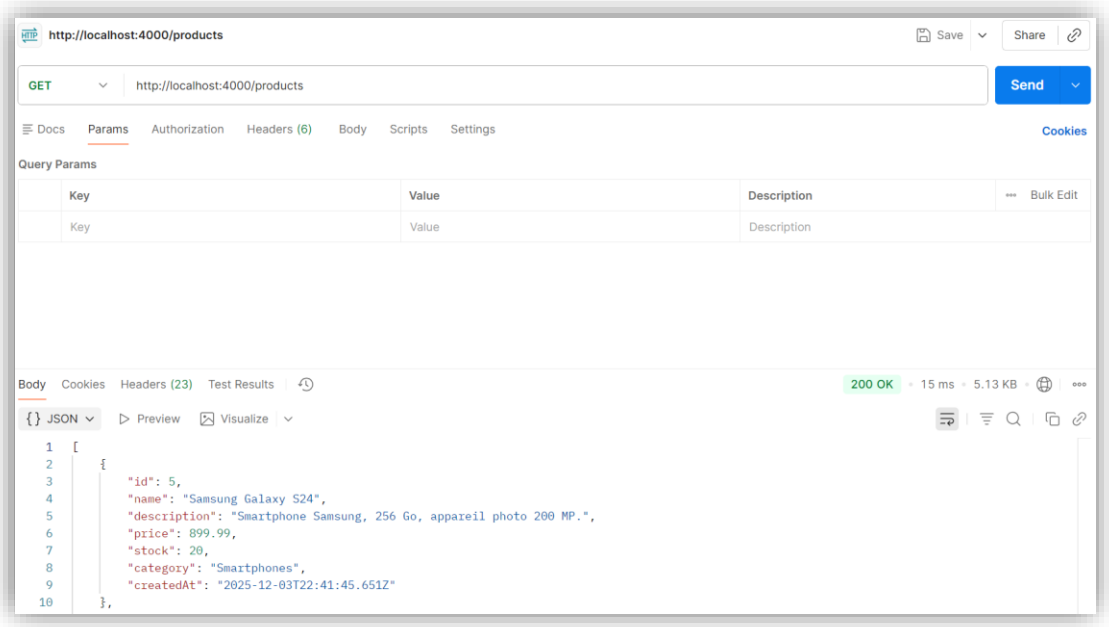
API de gestion des produits

Les API de gestion des produits permettent d'exposer le catalogue de produits au public et d'offrir des fonctionnalités d'administration sécurisées.

Les routes publiques permettent à tout utilisateur, authentifié ou non, de consulter la liste des produits et le détail d'un produit spécifique. Ces routes sont utilisées par le frontend pour afficher le catalogue et les pages de détail.

Les routes de création, modification et suppression de produits sont protégées par une authentification et un contrôle de rôle administrateur. Cette séparation garantit que seuls les administrateurs peuvent gérer le contenu du catalogue.

Les données reçues lors de la création ou de la mise à jour d'un produit sont validées côté serveur afin d'éviter les incohérences (prix négatif, stock invalide, champs manquants).



API de gestion des commandes

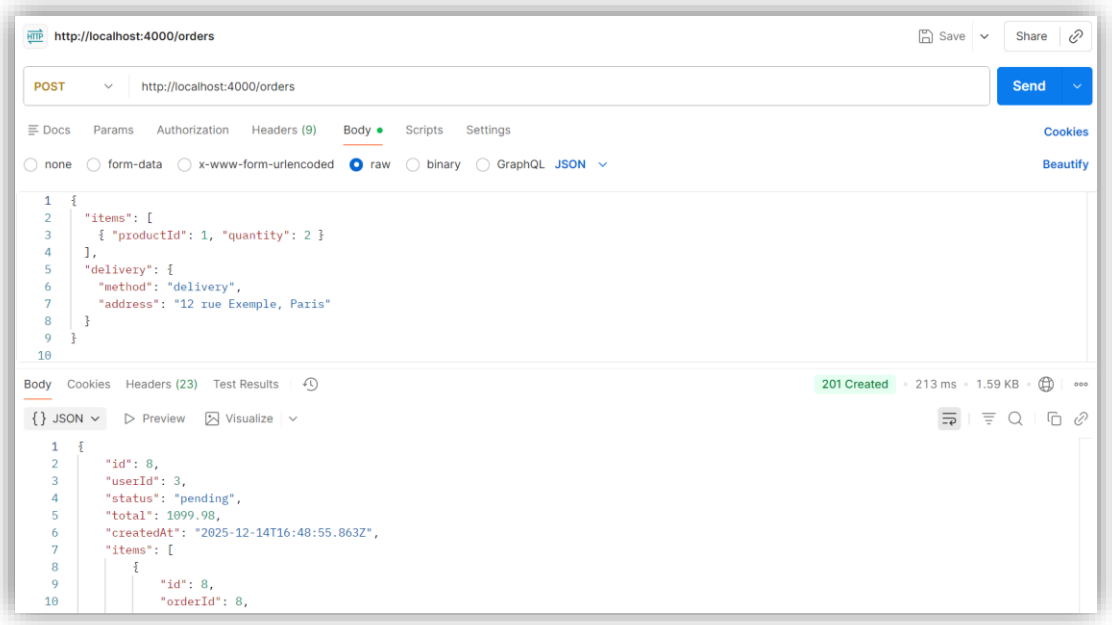
Les API de commandes constituent un élément central du projet e-commerce.

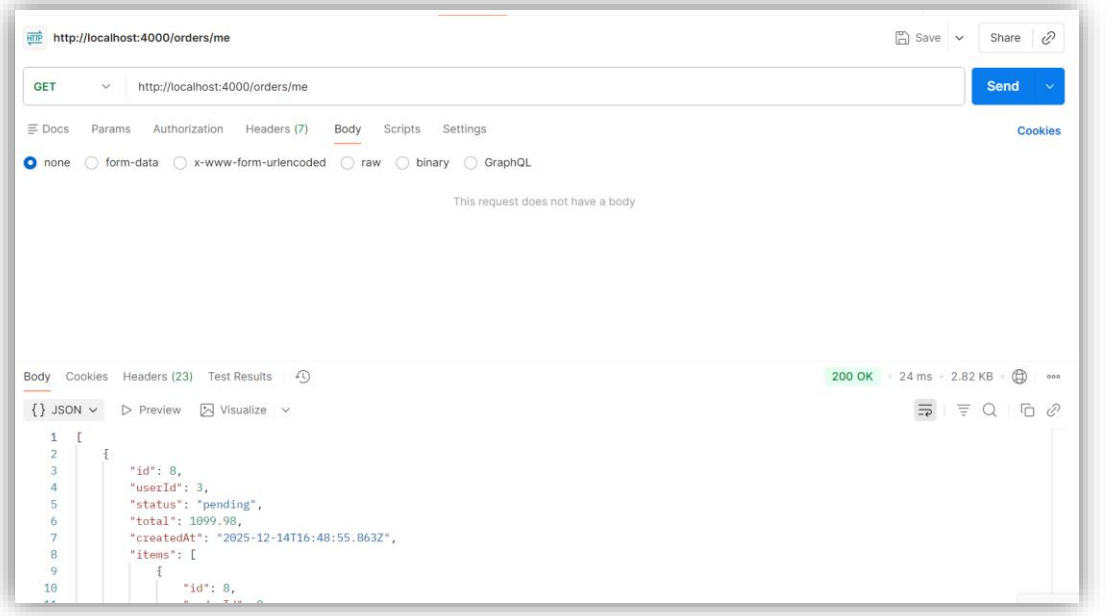
Un utilisateur authentifié peut créer une commande en envoyant la liste des produits souhaités et les informations de livraison. Lors de cette opération, le backend vérifie la disponibilité du stock, calcule le montant total et crée simultanément les enregistrements liés à la commande, aux articles de commande, au paiement et à la livraison.

Un utilisateur peut consulter l'historique de ses commandes ainsi que le détail d'une commande spécifique. L'accès est strictement limité au propriétaire de la commande ou à un administrateur.

Les administrateurs disposent d'une route permettant de consulter l'ensemble des commandes et de mettre à jour leur statut. Cette fonctionnalité permet de simuler le suivi d'une commande (préparation, expédition, livraison).

L'annulation d'une commande est également prise en charge, sous certaines conditions de statut. Cette opération entraîne la mise à jour du stock des produits ainsi que l'état du paiement et de la livraison.





API de gestion des avis clients

Le système d'avis permet aux utilisateurs authentifiés de laisser une note et un commentaire sur un produit.

Les avis sont associés à un utilisateur et à un produit, ce qui garantit leur traçabilité. Un utilisateur ne peut supprimer que ses propres avis, ce qui est contrôlé côté serveur.

Les avis peuvent être consultés publiquement par produit et sont utilisés à la fois pour informer les futurs acheteurs et pour alimenter le système de recommandations.

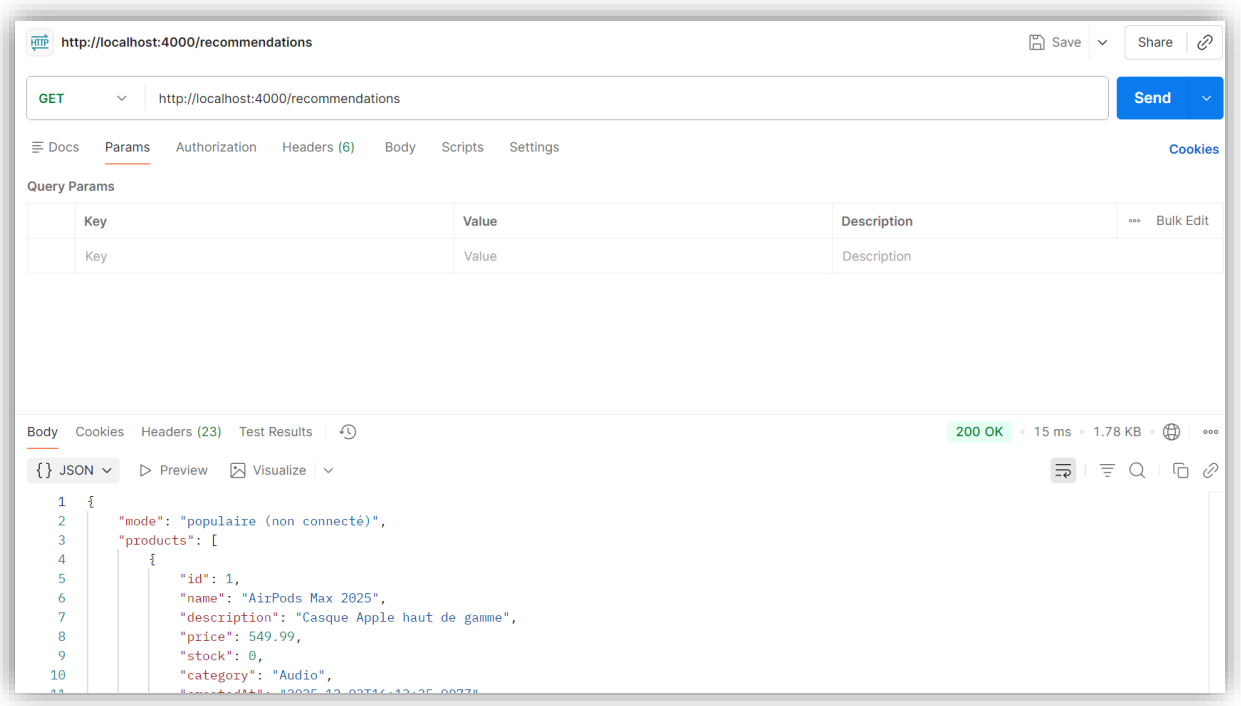
API de recommandations de produits

L'application propose une API de recommandations destinée à enrichir l'expérience utilisateur.

Cette API peut être appelée avec ou sans authentification. Lorsqu'un utilisateur est connecté, les recommandations sont basées sur son historique d'achat et sur des produits similaires (catégorie, type de produit, popularité). Les produits déjà achetés sont exclus des recommandations afin d'éviter de proposer des articles redondants.

Pour les utilisateurs non connectés, l'API retourne des recommandations basées sur la popularité globale et les produits les mieux notés.

Cette API peut également prendre en compte la localisation de l'utilisateur (latitude et longitude) afin de proposer des recommandations contextuelles, notamment en lien avec les points de retrait disponibles à proximité.

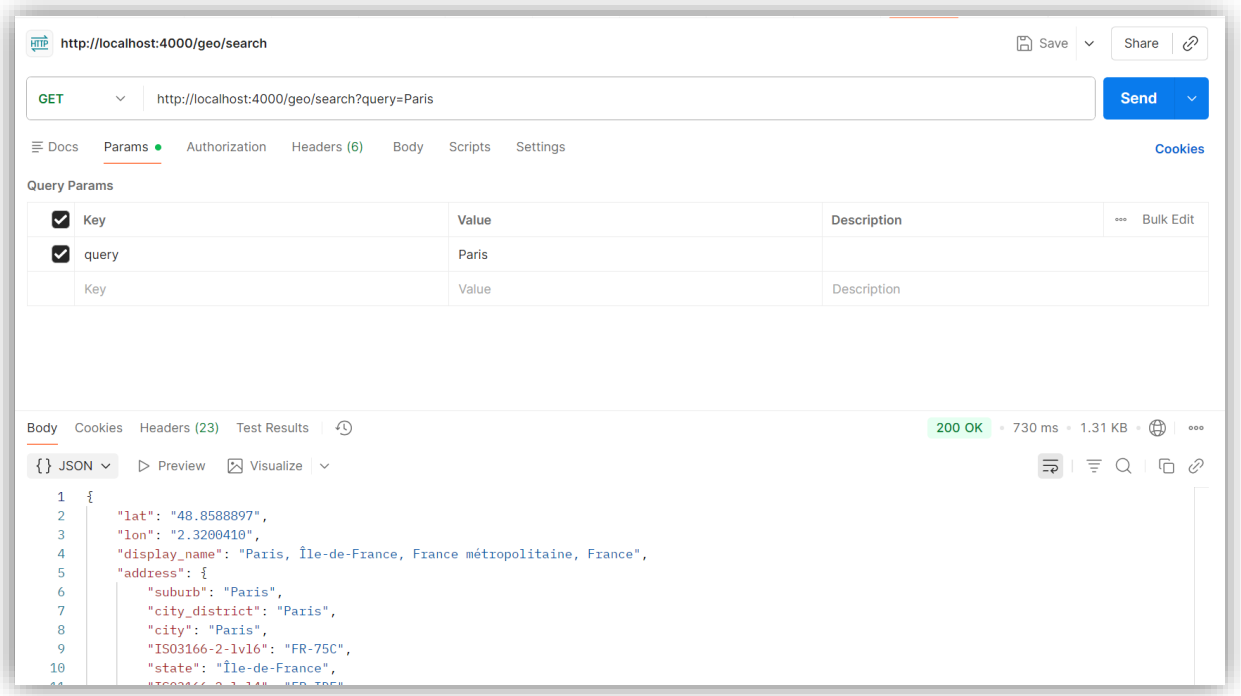


API de géolocalisation (intégration externe)

Les API de géolocalisation servent de passerelle vers le service externe OpenStreetMap Nominatim.

Une première route permet de transformer une adresse textuelle en coordonnées géographiques. Une seconde route permet de rechercher des points de retrait ou lieux d'intérêt autour d'une position donnée.

Le backend agit comme intermédiaire afin de contrôler les paramètres envoyés, d'ajouter les en-têtes requis par l'API externe, de gérer les délais de réponse et de traiter proprement les erreurs éventuelles.



Gestion des erreurs et des réponses

L'ensemble des API respecte des conventions de réponse cohérentes. Les codes HTTP sont utilisés de manière appropriée afin de distinguer les succès (200, 201), les erreurs de validation (400), les accès non autorisés (401), les permissions insuffisantes (403) et les ressources inexistantes (404).

Sécurité et authentification de l'application

Authentification basée sur JSON Web Tokens (JWT)

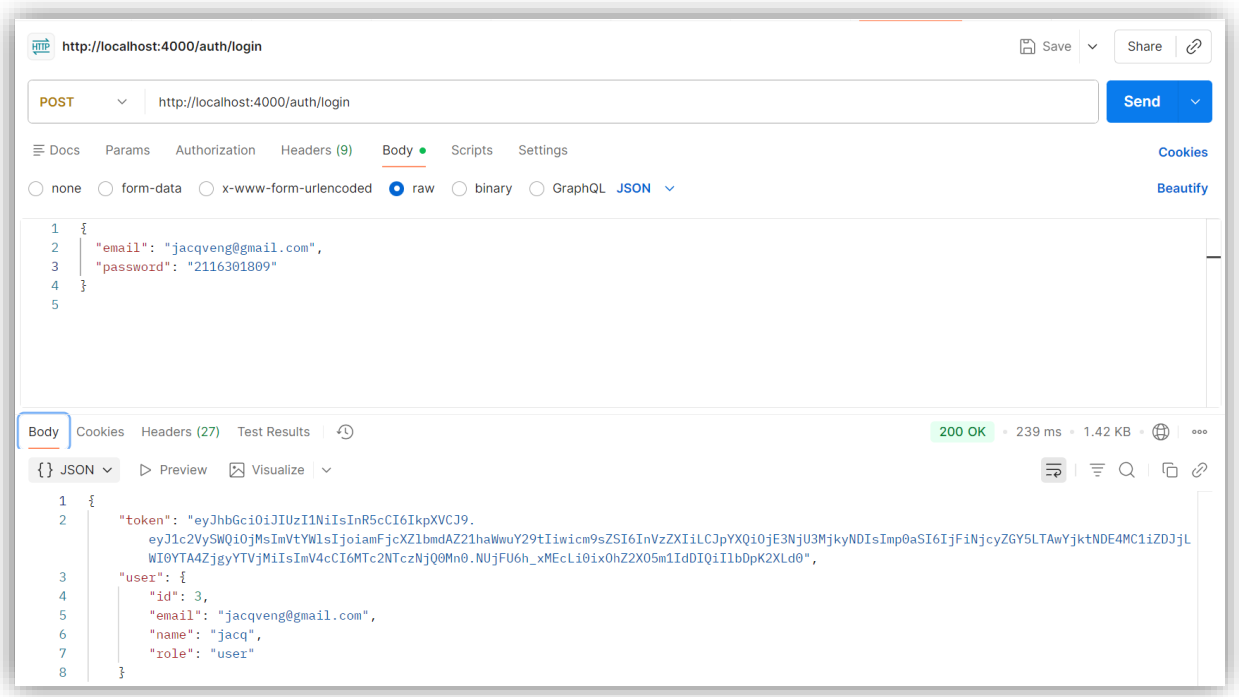
L'authentification des utilisateurs repose sur l'utilisation de JSON Web Tokens (JWT). Lorsqu'un utilisateur se connecte avec des identifiants valides, le backend génère un token signé à l'aide d'un secret (JWT_SECRET). Ce token contient l'identifiant de l'utilisateur ainsi qu'une date d'expiration.

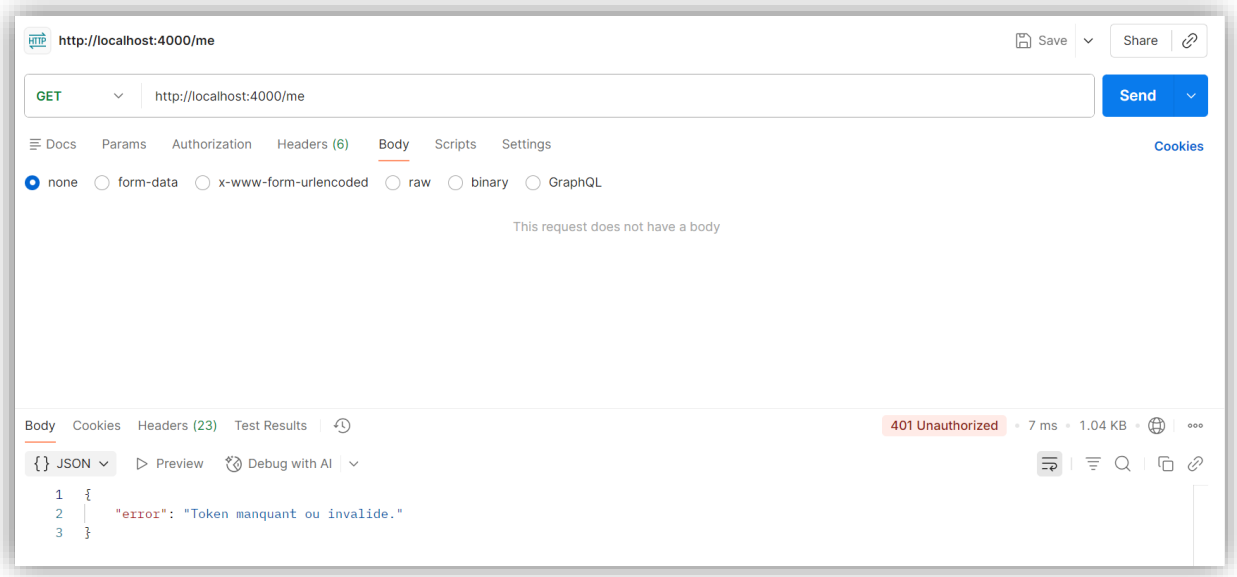
Le token JWT est ensuite transmis au frontend, qui le stocke côté client. Pour chaque requête nécessitant une authentification, le frontend inclut ce token dans l'en-tête HTTP Authorization sous la forme d'un Bearer token.

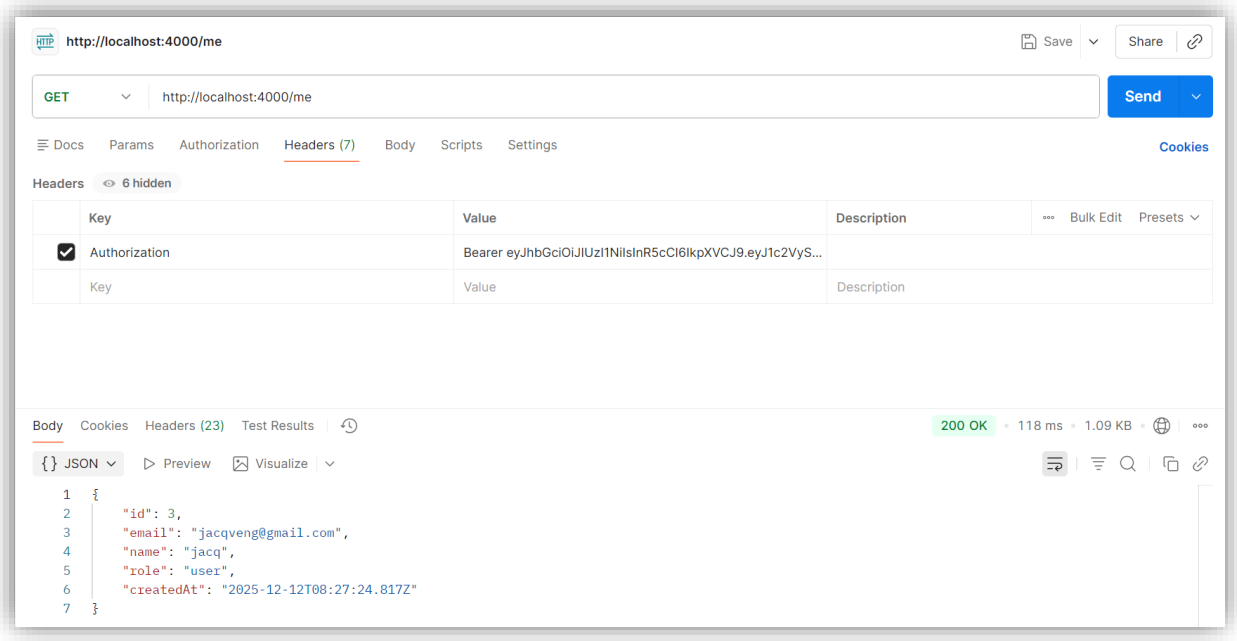
Côté backend, un middleware d'authentification intercepte les requêtes protégées et vérifie :

- la présence du token,
- la validité de la signature,
- la non-expiration du token,
- l'existence de l'utilisateur associé.

Si l'une de ces vérifications échoue, l'accès à la ressource est refusé.







Gestion de la déconnexion et révocation des tokens

Afin de gérer la déconnexion, un mécanisme de révocation des tokens a été mis en place. Chaque JWT contient un identifiant unique (jti). Lorsqu'un utilisateur se déconnecte, cet identifiant est ajouté à une liste de tokens révoqués.

Lors de chaque requête authentifiée, le middleware vérifie que le token présenté n'est pas présent dans cette liste. Cette approche permet d'empêcher l'utilisation d'un token après une déconnexion, même s'il n'a pas encore expiré.

Dans le cadre de ce projet, cette liste est maintenue en mémoire. Une amélioration possible pour un environnement de production consisterait à persister cette liste dans un système externe tel que Redis.

Hashage des mots de passe

Les mots de passe ne sont jamais stockés en clair dans la base de données. Lors de l'inscription ou de la modification du mot de passe, celui-ci est hashé à l'aide de l'algorithme bcrypt.

Le hashage garantit que, même en cas d'accès non autorisé à la base de données, les mots de passe des utilisateurs ne peuvent pas être exploités directement. Lors de la connexion, le mot de passe fourni est comparé au hash stocké, sans jamais exposer le mot de passe original.

Gestion des rôles et des autorisations

Le projet met en place un système de rôles afin de différencier les utilisateurs classiques des administrateurs. Ce rôle est stocké dans le modèle utilisateur et vérifié via des middlewares dédiés.

Les routes sensibles, telles que la création, la modification ou la suppression de produits, sont réservées aux utilisateurs disposant du rôle administrateur. De la même manière, certaines routes permettent uniquement au propriétaire d'une ressource d'y accéder ou de la modifier, comme c'est le cas pour les commandes et les avis.

Ces contrôles garantissent qu'un utilisateur ne peut ni accéder aux données d'un autre utilisateur ni effectuer des actions pour lesquelles il n'a pas les permissions nécessaires.

Validation des données côté serveur

Toutes les données reçues par l'API sont validées côté serveur avant d'être traitées. Cette validation concerne notamment :

- les formats d'email,
- la longueur et la complexité des mots de passe,
- les bornes numériques (prix, quantité, notes),
- la présence des champs obligatoires.

Cette étape permet d'éviter les incohérences de données, de limiter les erreurs applicatives et de réduire les risques d'exploitation via des entrées malformées.

Protection contre les attaques courantes

Plusieurs mécanismes contribuent à renforcer la sécurité globale de l'application :

- l'utilisation de Prisma ORM protège contre les injections SQL grâce à des requêtes paramétrées,
- la limitation de la taille des requêtes JSON empêche l'envoi de charges excessives,
- la configuration de CORS limite les origines autorisées à accéder à l'API,
- des middlewares de sécurité (tels que helmet) permettent d'ajouter des en-têtes HTTP sécurisés.

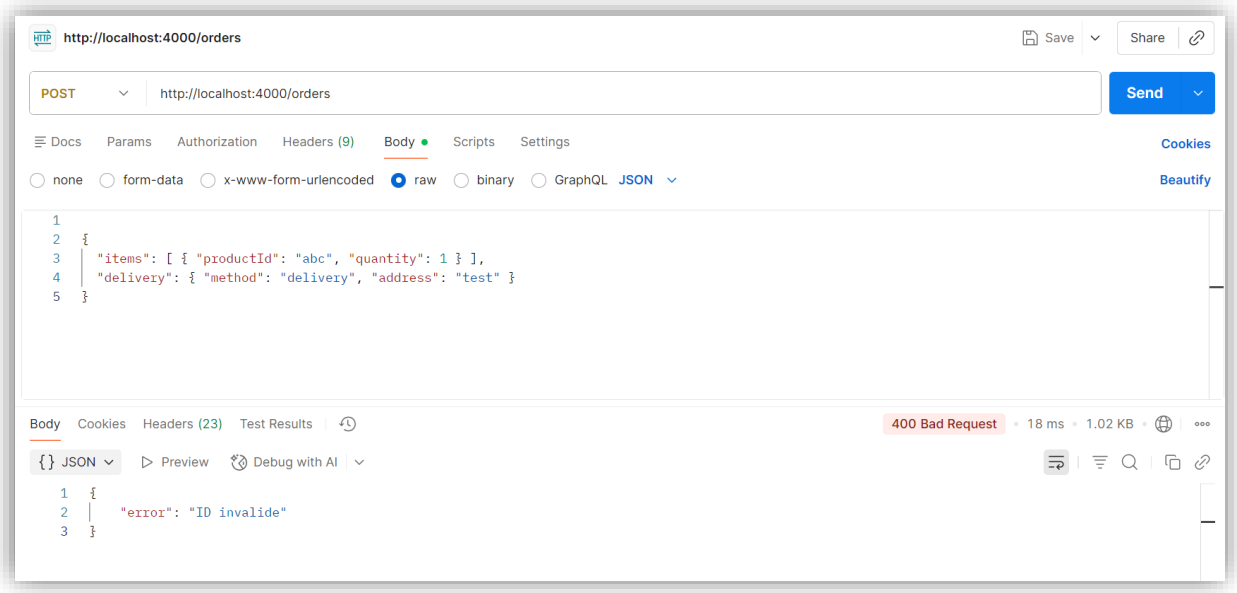
Ces mesures ne remplacent pas un audit de sécurité complet, mais constituent une base solide pour une application e-commerce de niveau universitaire ou pré-production.

Test d'injection SQL

Afin de vérifier la résistance de l'API face aux tentatives d'injection SQL, plusieurs requêtes malveillantes ont été simulées via Postman. Un test a notamment consisté à fournir une valeur non numérique ("abc") à la place d'un identifiant de produit lors de la création d'une commande.

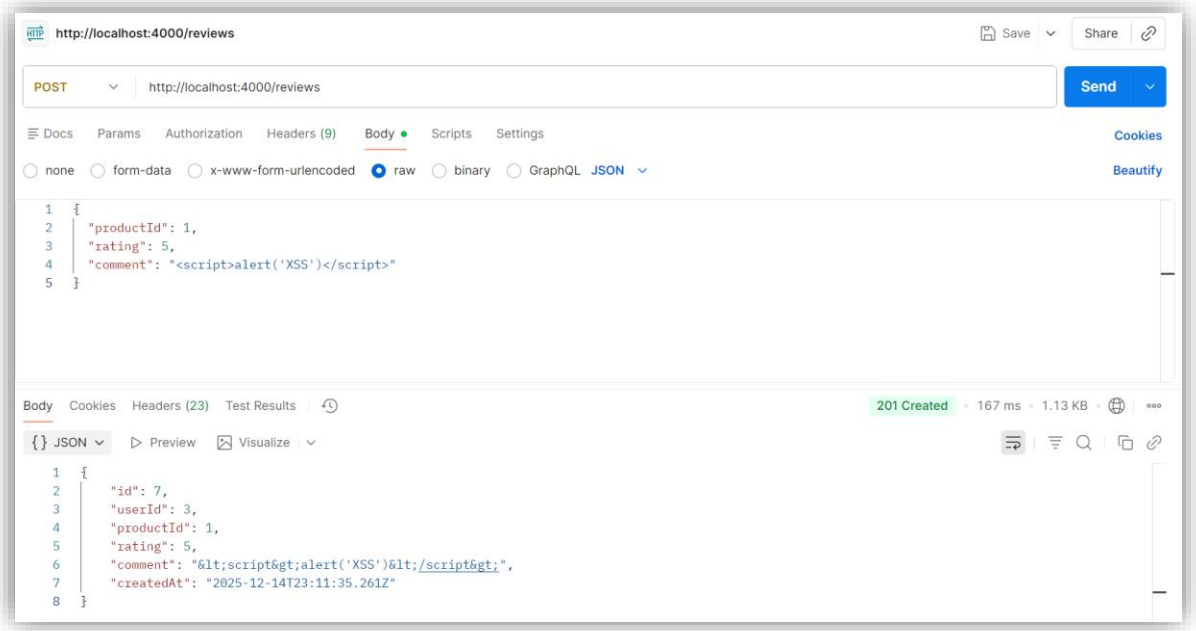
L'API a correctement rejeté la requête avec un code HTTP 400 (Bad Request) et un message d'erreur indiquant un identifiant invalide. Ce comportement démontre la présence de validations côté serveur empêchant toute donnée non conforme d'atteindre la base de données.

De plus, l'utilisation de Prisma ORM garantit l'emploi de requêtes SQL paramétrées, éliminant ainsi les risques d'injection SQL. Ces tests confirment que l'API est protégée contre ce type d'attaque.



Test XSS (Cross-Site Scripting)

Afin de vérifier que l'application n'exécute pas de code injecté via les champs texte, un commentaire contenant un script (<script>alert('XSS')</script>) a été envoyé lors de la création d'un avis produit. L'API a accepté la requête, mais la valeur retournée dans la réponse est échappée (par exemple <script>...</script>), ce qui signifie que le contenu est traité comme une chaîne de caractères et non comme du code HTML/JavaScript. Ce test confirme que les données utilisateur affichées ne permettent pas l'exécution de scripts, réduisant fortement le risque d'attaque XSS.



Documentation des API avec Swagger

La documentation des API est un élément central du projet. Elle permet de présenter clairement l'ensemble des endpoints du backend, leur fonctionnement, leurs paramètres et les règles de sécurité associées. Elle facilite également l'utilisation de l'API par des développeurs tiers et la maintenance du projet.

Pour ce projet, la documentation est réalisée avec **Swagger (OpenAPI)**. Cet outil a été choisi car il permet de générer automatiquement une documentation à partir du code, propose une interface graphique interactive et est largement utilisé dans le milieu professionnel. Swagger complète l'utilisation de Postman : Postman est utilisé pour les tests approfondis, tandis que Swagger sert principalement à la compréhension, à la démonstration et à la présentation de l'API.

Swagger est intégré côté backend à l'aide des bibliothèques swagger-jsdoc et swagger-ui-express. La configuration globale définit les informations principales de l'API (titre, version, description, serveurs disponibles) ainsi que le mécanisme de sécurité basé sur l'authentification JWT. Les annotations Swagger sont directement intégrées dans le code des routes, ce qui permet de maintenir la documentation synchronisée avec l'implémentation réelle.

La documentation est accessible via l'URL suivante :
http://localhost:4000/api-docs

L'interface Swagger regroupe les endpoints par fonctionnalités (authentification, produits, commandes, avis, géolocalisation, recommandations). Pour chaque route, elle indique la méthode HTTP, l'URL, la description, les paramètres attendus, les réponses possibles et les règles de sécurité. Les routes protégées précisent clairement l'obligation de fournir un token JWT dans l'en-tête Authorization.

Enfin, Swagger propose la fonctionnalité **"Try it out"**, permettant de tester directement les routes depuis le navigateur. Cette fonctionnalité a été utilisée pour effectuer des tests rapides, vérifier le bon fonctionnement des endpoints et démontrer l'API lors de la présentation du projet.

Système de recommandations de produits

Le projet intègre également un système de recommandations de produits, répondant aux nouveaux usages attendus pour un site de commerce en ligne moderne.

Le système de recommandations est accessible via une API dédiée. Cette API peut être appelée avec ou sans authentification, ce qui permet d'adapter les recommandations au contexte de l'utilisateur.

Recommandations pour les utilisateurs connectés

Lorsqu'un utilisateur est authentifié, les recommandations sont basées sur son historique d'achat. Le système analyse les produits précédemment commandés et propose des articles similaires, en tenant compte de critères tels que la catégorie, le type de produit ou des règles métiers spécifiques.

Les produits déjà achetés par l'utilisateur sont exclus des recommandations afin d'éviter de proposer des articles redondants. Cette approche améliore la pertinence des suggestions et favorise la découverte de nouveaux produits.

Recommandations pour les utilisateurs non connectés

Pour les utilisateurs non authentifiés, le système propose des recommandations basées sur des indicateurs globaux, tels que la popularité des produits et les meilleures notes laissées par les clients.

Cette approche permet de fournir des recommandations pertinentes même en l'absence de données personnelles, tout en respectant la confidentialité des utilisateurs.

Prise en compte de la localisation

Le système de recommandations peut également exploiter la localisation de l'utilisateur lorsqu'elle est disponible. Les informations géographiques permettent de proposer des points de retrait proches, ce qui renforce la personnalisation de l'expérience utilisateur.

Cette combinaison entre recommandations de produits et géolocalisation illustre l'intégration de fonctionnalités innovantes adaptées aux attentes des plateformes e-commerce modernes.

Conclusion générale

Ce projet avait pour objectif de concevoir et de développer une application de commerce en ligne moderne répondant aux attentes fonctionnelles et techniques d'un site e-commerce en 2025. À travers la mise en place d'un frontend en React, d'un backend en Node.js avec Express et d'une base de données PostgreSQL, l'application couvre l'ensemble du parcours utilisateur, de l'inscription à la gestion et au suivi des commandes.

Le développement des API internes a permis de structurer clairement les fonctionnalités clés du projet : gestion des utilisateurs, gestion des produits, gestion des commandes, suivi des paiements et des livraisons, gestion des avis clients, recommandations de produits et

intégration d'une API externe de géolocalisation. L'utilisation de Prisma ORM a assuré une modélisation cohérente des données, une gestion fiable des relations et une protection efficace contre les injections SQL.

La sécurité de l'application repose sur des mécanismes éprouvés tels que l'authentification par jetons JWT, le hashage des mots de passe, la gestion des rôles et des autorisations, ainsi que la validation systématique des données côté serveur. Ces choix garantissent un niveau de sécurité adapté au périmètre du projet et conforme aux bonnes pratiques actuelles du développement web.

La documentation de l'API via Swagger apporte une réelle valeur ajoutée en rendant les endpoints facilement compréhensibles et testables. Les tests réalisés avec Postman ont permis de valider le bon fonctionnement des routes, la gestion des erreurs et la robustesse des mécanismes de sécurité. L'intégration de l'API externe OpenStreetMap Nominatim illustre la capacité du projet à s'appuyer sur des services tiers afin d'enrichir l'expérience utilisateur.

Enfin, le système de recommandations et la prise en compte de la géolocalisation démontrent une volonté d'aller au-delà des fonctionnalités e-commerce classiques en proposant des usages plus avancés et personnalisés. Bien que certaines améliorations puissent être envisagées pour un déploiement en production, telles que l'ajout de tests automatisés, l'utilisation de tokens de rafraîchissement ou l'optimisation des recommandations par apprentissage automatique, le projet constitue une base solide, cohérente et extensible.

En conclusion, ce projet répond pleinement aux objectifs fixés, tant sur le plan fonctionnel que technique. Il illustre la mise en œuvre concrète d'une architecture e-commerce moderne, sécurisée et documentée, et constitue une réalisation représentative des compétences attendues dans le cadre de ce cursus.