API E-commerce Sécurisée

Documentation Technique & Sécurité

Implémentation d'un Backend RESTful Node.js robuste

Auteur :

Yacine Sehli

Étudiant B3 Cybersécurité

Développeur Web

19 novembre 2025

Table des matières

1. Introduction du Projet

1.1 Contexte

Dans le cadre du développement d'applications web modernes, la sécurité des données transactionnelles est primordiale. Ce projet, intitulé **API E-commerce Sécurisée**, vise à fournir une architecture Backend RESTful solide, capable de gérer des transactions de vente en ligne tout en résistant aux vecteurs d'attaques courants.

1.2 Objectifs Techniques

L'objectif principal n'était pas seulement de créer une API fonctionnelle, mais de prioriser la **Security by Design**. Le système a été conçu pour :

- Assurer l'intégrité des données entrantes.
- Protéger l'infrastructure contre les surcharges (DoS).
- Prévenir les injections de code (XSS, NoSQL).

2. Architecture Technique

2.1 Stack Technologique

Le projet repose sur un environnement JavaScript moderne et performant :
— **Environnement :** Node.js
— **Base de données :** MongoDB (NoSQL)
— **Architecture :** API RESTful

2.2 Documentation

Une documentation complète des endpoints est disponible via Swagger/OpenAPI, permettant une intégration fluide pour les développeurs Front-End.

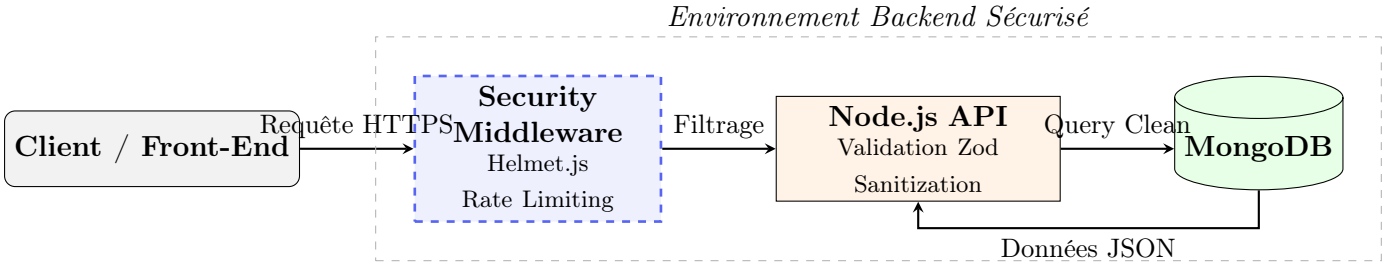

FIGURE 2.1 – Architecture du flux de données sécurisé

Analyse du Flux de Données Sécurisé

La Figure 2.1 illustre l'architecture de *défense en profondeur* mise en place pour l'API. Contrairement à une approche monolithique classique, le traitement des requêtes suit un pipeline de validation strict en trois étapes :

1. **La Barrière Périmétrique (Middleware) :** Avant même d'atteindre la logique métier, la requête traverse une première couche de sécurité. C'est ici que le *Rate Limiting* filtre les volumes anormaux (protection DDoS/Brute-force) et que *Helmet.js* verrouille les en-têtes HTTP.
2. **Le Contrôle d'Intégrité (API Layer) :** Une fois la connexion acceptée, les données (payload) sont inspectées par *Zod*. Si le schéma JSON ne correspond pas strictement aux attentes (types, formats), la requête est rejetée instantanément, évitant le traitement de données polluées.
3. **L'Assainissement Final (Storage Layer) :** Avant l'interaction avec MongoDB, une étape de *Sanitization* retire les caractères spéciaux dangereux (comme les opérateurs `$` utilisés dans les injections NoSQL).

Cette segmentation garantit que la base de données n'est jamais exposée directement aux entrées brutes de l'utilisateur, isolant ainsi le composant le plus critique de l'infrastructure.

3. Implémentations de Sécurité

Ce chapitre détaille les mesures défensives mises en place, constituant le cœur de la valeur ajoutée de ce projet.

3.1 Validation des Données (Zod)

L'une des premières lignes de défense est la validation stricte des entrées utilisateur.
— **Technologie :** Zod
— **Rôle :** Vérification des schémas de données avant tout traitement.
— **Bénéfice :** Rejette automatiquement toute requête malformée ou contenant des types de données inattendus, réduisant la surface d'attaque.

3.2 Sécurisation des En-têtes HTTP (Helmet.js)

Pour protéger l'application contre des attaques courantes liées au navigateur :
— **Technologie :** Helmet.js
— **Rôle :** Configuration automatique des en-têtes HTTP sécurisés.
— **Détails :** Mise en place de `X-XSS-Protection`, `X-Frame-Options` et `HSTS` pour forcer les connexions sécurisées.

3.3 Protection contre la Force Brute (Rate Limiting)

Afin de prévenir l'épuisement des ressources et les tentatives de devinette de mots de passe :
— **Mécanisme :** Rate Limiting
— **Application :** Limitation du nombre de requêtes autorisées par IP sur une fenêtre de temps donnée.
— **Cible :** Particulièrement strict sur les routes sensibles comme `/login` ou `/register`.

3.4 Assainissement des Requêtes (Sanitization)

L'utilisation de bases de données NoSQL comme MongoDB nécessite une vigilance particulière contre les injections d'opérateurs.
— **Action :** Nettoyage systématique des requêtes entrantes.
— **Objectif :** Empêcher les injections NoSQL (ex : envoi de `{$gt: ""}` pour contourner l'authentification).

4. Conclusion

Ce projet démontre la capacité à allier développement web performant et pratiques de cybersécurité rigoureuses. L'API E-commerce Sécurisée est une base solide pour tout projet commercial nécessitant fiabilité et protection des données utilisateurs.