UNIVERSITÉ / ÉCOLE

Faculté des Sciences et Techniques

MODULE

Sécurité des Applications Web Modernes

Rapport de Projet

Application E-Commerce Full Stack

Gestion de Produits avec Authentification Sécurisée

Réalisé par :
Reda El Hadfi

Encadré par :
Pr. [Nom du Professeur]

Année Universitaire 2025-2026

Remerciements

Je tiens à exprimer mes sincères remerciements à mon professeur, **Pr. [Nom]**, pour son encadrement, ses conseils précieux et son soutien tout au long de ce projet. Ses orientations m'ont permis d'approfondir mes connaissances en sécurité des applications web modernes.

Je remercie également toute l'équipe pédagogique pour la qualité de la formation dispensée et les connaissances transmises qui m'ont été indispensables pour mener à bien ce projet.

Enfin, je remercie tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Résumé

Français

Ce rapport présente la conception et la réalisation d'une application web Full Stack de gestion de produits e-commerce. Le projet met en œuvre les meilleures pratiques de sécurité des applications web modernes, incluant l'authentification sécurisée par JWT, le hashage des mots de passe avec bcrypt, la gestion des rôles (RBAC), et la protection contre les vulnérabilités OWASP Top 10 (injection NoSQL, XSS, CSRF).

L'application est développée avec une architecture client-serveur en trois tiers : un frontend React.js moderne et responsive, un backend Node.js/Express robuste et sécurisé, et une base de données MongoDB Atlas hébergée dans le cloud. Elle offre des fonctionnalités complètes de CRUD, recherche avancée, filtrage multi-critères, pagination, upload de fichiers et un tableau de bord administrateur avec statistiques en temps réel.

Le projet a été déployé en production sur des services cloud (Render pour le backend, Vercel pour le frontend), démontrant ainsi une maîtrise complète du cycle de développement d'une application web moderne et sécurisée.

Mots-clés : Full Stack, React, Node.js, MongoDB, Sécurité Web, JWT, REST API, CRUD, E-Commerce

English

This report presents the design and implementation of a Full Stack e-commerce product management web application. The project implements best practices in modern web application security, including secure JWT authentication, bcrypt password hashing, role-based access control (RBAC), and protection against OWASP Top 10 vulnerabilities (NoSQL injection, XSS, CSRF).

The application is developed with a three-tier client-server architecture : a modern and responsive React.js frontend, a robust and secure Node.js/Express backend, and a MongoDB Atlas database hosted in the cloud. It offers complete CRUD functionalities, advanced search, multi-criteria filtering, pagination, file upload, and an admin dashboard with real-time statistics.

The project has been deployed to production on cloud services (Render for backend, Vercel for frontend), demonstrating complete mastery of the development cycle of a modern and secure web application.

Keywords : Full Stack, React, Node.js, MongoDB, Web Security, JWT, REST API, CRUD, E-Commerce

Table des matières

Table des figures

Chapitre 1

Introduction et Contexte

1.1 Présentation du Projet

Dans le cadre du module **Sécurité des Applications Web Modernes**, ce projet vise à concevoir et développer une application web Full Stack complète de gestion de produits e-commerce. L'objectif principal est de mettre en pratique les concepts de sécurité avancés enseignés durant la formation, tout en démontrant une maîtrise des technologies web modernes.

L'application permet aux utilisateurs de consulter un catalogue de produits avec des fonctionnalités avancées de recherche et de filtrage, tandis que les administrateurs disposent d'un système complet de gestion (CRUD) des produits et d'un tableau de bord statistique.

1.2 Problématique

Les applications web e-commerce sont constamment ciblées par des attaques malveillantes (injection SQL/NoSQL, XSS, CSRF, vol de sessions). Il est donc crucial d'implémenter des mécanismes de sécurité robustes dès la conception pour protéger les données des utilisateurs et garantir l'intégrité du système.

La problématique centrale de ce projet est : *Comment développer une application web moderne, performante et sécurisée qui respecte les standards de sécurité OWASP et offre une expérience utilisateur optimale ?*

1.3 Objectifs du Projet

Les objectifs de ce projet sont multiples :

1. **Sécurité** : Implémenter une authentification sécurisée (JWT), un hashage des mots de passe (bcrypt), une gestion des rôles, et protéger contre les vulnérabilités OWASP.
2. **Fonctionnalités** : Développer un système CRUD complet avec recherche, filtrage, tri, pagination et upload de fichiers.
3. **Architecture** : Concevoir une architecture client-serveur en trois tiers avec séparation claire des responsabilités.
4. **Interface** : Créer une interface utilisateur moderne, responsive et intuitive avec TailwindCSS.
5. **Déploiement** : Déployer l'application en production sur des services cloud (Render, Vercel, MongoDB Atlas).
6. **Documentation** : Rédiger une documentation technique complète et professionnelle.

1.4 Cahier des Charges Résumé

Le cahier des charges du projet définit les exigences fonctionnelles et non-fonctionnelles suivantes :

1.4.1 Exigences Fonctionnelles

— **Authentification** : Inscription, connexion, déconnexion avec JWT
— **Gestion des utilisateurs** : Deux rôles (admin, user) avec permissions différentes
— **Gestion des produits** : CRUD complet (Créer, Lire, Modifier, Supprimer)
— **Recherche avancée** : Par nom, description, catégorie, prix
— **Filtrage** : Par catégorie, plage de prix, disponibilité en stock
— **Tri** : Par prix, nom, date de création
— **Pagination** : Navigation par pages (12 produits/page)
— **Upload** : Téléchargement d'images pour les produits
— **Dashboard** : Statistiques pour les administrateurs

1.4.2 Exigences Non-Fonctionnelles

— **Sécurité** : Protection contre OWASP Top 10, chiffrement des données sensibles
— **Performance** : Temps de réponse < 2 secondes
— **Scalabilité** : Architecture permettant la montée en charge
— **Maintenabilité** : Code propre, modulaire et documenté
— **Responsive** : Compatible mobile, tablette et desktop
— **Disponibilité** : Application accessible 24/7 en production

1.5 Méthodologie de Travail

Le développement du projet a suivi une approche **itérative et incrémentale** avec les étapes suivantes :

1. **Analyse et conception** : Définition des besoins, modélisation de la base de données, conception de l'architecture
2. **Développement backend** : Création des modèles, controllers, middlewares et routes
3. **Développement frontend** : Création des composants React, pages et gestion d'état
4. **Tests** : Tests manuels avec Postman, tests d'intégration
5. **Déploiement** : Mise en production sur les plateformes cloud
6. **Documentation** : Rédaction de la documentation technique

Cette méthodologie a permis de livrer un produit fonctionnel à chaque itération, tout en maintenant un code de qualité et sécurisé.

Chapitre 2

Analyse et Conception

2.1 Analyse des Besoins

2.1.1 Besoins Fonctionnels

L'analyse des besoins a permis d'identifier les fonctionnalités essentielles à implémenter :

ID	Fonctionnalité	Priorité
F1	Inscription et authentification des utilisateurs	Haute
F2	Gestion des rôles (admin/user)	Haute
F3	CRUD complet des produits (admin)	Haute
F4	Consultation des produits (public)	Haute
F5	Recherche textuelle dans produits	Moyenne
F6	Filtrage par catégorie et prix	Moyenne
F7	Tri des résultats	Moyenne
F8	Pagination des résultats	Moyenne
F9	Upload d'images produits	Moyenne
F10	Dashboard avec statistiques (admin)	Basse

TABLE 2.1 – Liste des besoins fonctionnels

2.1.2 Identification des Acteurs

Le système interagit avec deux types d'acteurs principaux :

— **Utilisateur simple (user)** : Peut consulter les produits, effectuer des recherches et des filtrages, mais ne peut pas modifier le catalogue.

— **Administrateur (admin)** : Dispose de tous les droits de l'utilisateur simple, plus la possibilité de créer, modifier et supprimer des produits, ainsi que d'accéder au tableau de bord statistique.

2.2 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation ci-dessous illustre les interactions entre les acteurs et le système :

[INSÉRER DIAGRAMME DE CAS D'UTILISATION]

Acteurs : Utilisateur, Administrateur
Cas d'utilisation : Inscription, Connexion, Consulter produits, Rechercher, Filtrer, Créer produit, Modifier produit, Supprimer produit, Voir statistiques

FIGURE 2.1 – Diagramme de cas d'utilisation du système

2.3 Modélisation de la Base de Données

2.3.1 Schéma Relationnel

La base de données MongoDB est conçue avec deux collections principales :

[INSÉRER SCHÉMA DE BASE DE DONNÉES]

Collection Users :
_id, name, email, password (hashed), role, createdAt, updatedAt

Collection Products :
_id, name, description, price, category, stock, image, createdBy (ref : Users), createdAt, updatedAt

Relation : One-to-Many (User → Products)

FIGURE 2.2 – Schéma de la base de données MongoDB

2.3.2 Description des Collections

Collection Users

Champ	Type	Description
_id	ObjectId	Identifiant unique (auto-généré)
name	String	Nom complet de l'utilisateur (2-50 caractères)
email	String	Email unique, validé par regex
password	String	Mot de passe hashé avec bcrypt (salt rounds : 10)
role	String	Rôle (enum : "user", "admin"), défaut : "user"
createdAt	Date	Date de création (auto-généré)
updatedAt	Date	Date de dernière modification (auto-généré)

TABLE 2.2 – Structure de la collection Users

Collection Products

Champ	Type	Description
_id	ObjectId	Identifiant unique (auto-généré)
name	String	Nom du produit (3-100 caractères)
description	String	Description détaillée (10-1000 caractères)
price	Number	Prix en euros (minimum 0.01)
category	String	Catégorie (enum : 7 valeurs prédéfinies)
stock	Number	Quantité en stock (entier 0)
image	String	Chemin de l'image uploadée
createdBy	ObjectId	Référence vers l'utilisateur créateur
createdAt	Date	Date de création (auto-généré)
updatedAt	Date	Date de dernière modification (auto-généré)

TABLE 2.3 – Structure de la collection Products

2.4 Contraintes de Sécurité

Plusieurs contraintes de sécurité ont été intégrées au niveau de la conception :

— **Validation des données** : Validation stricte à deux niveaux (client et serveur)
— **Hashage des mots de passe** : Utilisation de bcrypt avec 10 salt rounds
— **Authentification stateless** : JWT avec expiration de 30 jours
— **Contrôle d'accès** : Middleware d'autorisation basé sur les rôles
— **Protection injection** : Utilisation de Mongoose avec validation de schémas
— **Protection XSS** : Sanitization avec Express Validator et échappement React
— **Upload sécurisé** : Validation du type MIME, limite de taille (5MB)

Chapitre 3

Architecture et Technologies

3.1 Architecture Globale

L'application suit une architecture **client-serveur en trois tiers** qui sépare clairement les responsabilités :

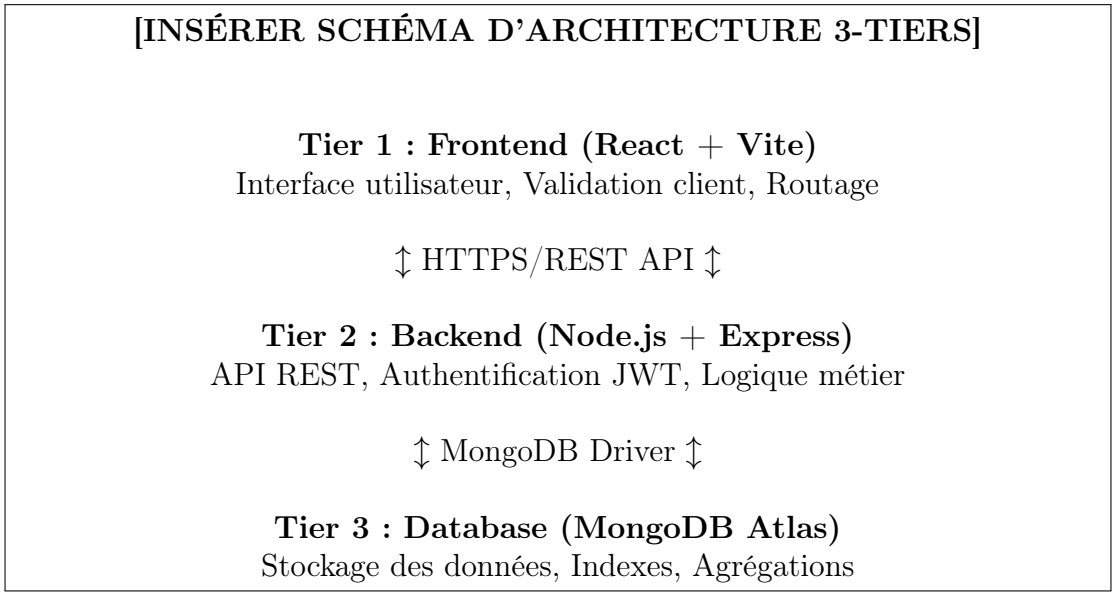

FIGURE 3.1 – Architecture client-serveur en trois tiers

3.1.1 Tier 1 : Couche Présentation (Frontend)

Technologies : React.js 18.3.1, Vite 7.3.1, TailwindCSS 3.4.1
Responsabilités :

- Interface utilisateur interactive et responsive
- Gestion de l'état de l'application (Context API)
- Validation côté client (React Hook Form)
- Communication avec le backend via Axios
- Routage côté client (React Router v6)

3.1.2 Tier 2 : Couche Logique Métier (Backend)

Technologies : Node.js, Express.js 5.2.1, Mongoose 9.1.5
Responsabilités :

- API RESTful avec endpoints sécurisés
- Authentification JWT et autorisation RBAC

— Validation des données (Express Validator)
— Logique métier et règles de gestion
— Gestion des fichiers uploadés (Multer)
— Interaction avec la base de données

3.1.3 Tier 3 : Couche Données (Database)

Technologies : MongoDB Atlas (Cloud)
Responsabilités :

— Stockage persistant des données
— Indexation pour performance
— Agrégations pour statistiques
— Backup automatique
— Chiffrement au repos et en transit

3.2 Stack Technologique

3.2.1 Justification des Choix

Le choix des technologies a été guidé par plusieurs critères :

Technologie	Rôle	Justification
React 18	Frontend Library	Composants réutilisables, Virtual DOM, Large écosystème
Vite	Build Tool	HMR rapide, Build optimisé, Configuration minimale
TailwindCSS	Styling	Utility-first, Responsive, Purge CSS automatique
Node.js	Backend Runtime	JavaScript full-stack, Asynchrone, NPM
Express	Web Framework	Minimaliste, Middleware, Large communauté
MongoDB	Database NoSQL	Flexible, Scalable, Cloud-ready
JWT	Authentication	Stateless, Self-contained, Standard
Bcrypt	Password Hashing	Salt automatique, Résistant brute force

TABLE 3.1 – Technologies utilisées et justifications

3.2.2 Avantages de l'Architecture Choisie

Séparation des préoccupations :

— Développement parallèle frontend/backend possible
— Facilite la maintenance et les tests
— Permet le remplacement d'une couche sans impacter les autres

Scalabilité :

— Frontend et backend peuvent être scalés indépendamment
— MongoDB Atlas offre le sharding horizontal

— Possibilité d'ajouter un load balancer

Sécurité :

— Séparation physique des couches

— Backend non exposé directement (API only)

— Chiffrement des communications (HTTPS)

3.3 Structure du Code

3.3.1 Structure Backend

```
backend/
|-- src/
|   |-- config/
|   |   |-- database.js      # Connexion MongoDB
|   |   +-- seed.js          # Peuplement BD
|   |-- models/
|   |   |-- User.js          # Schema utilisateur
|   |   +-- Product.js       # Schema produit
|   |-- controllers/
|   |   |-- authController.js
|   |   +-- productController.js
|   |-- middleware/
|   |   |-- auth.js          # JWT verification
|   |   |-- validator.js     # Validation
|   |   +-- upload.js        # Multer config
|   +-- routes/
|       |-- authRoutes.js
|       +-- productRoutes.js
|-- uploads/                 # Fichiers uploades
|-- server.js                # Point d'entree
|-- package.json
+-- .env                     # Variables d'environnement
```

Listing 3.1 – Arborescence du backend

3.3.2 Structure Frontend

```
frontend/
|-- src/
|   |-- components/
|   |   |-- Navbar.jsx
|   |   |-- Footer.jsx
|   |   |-- ProductCard.jsx
|   |   +-- ProtectedRoute.jsx
|   |-- pages/
|   |   |-- Home.jsx
|   |   |-- Login.jsx
|   |   |-- Register.jsx
|   |   |-- Products.jsx
|   |   |-- ProductDetail.jsx
|   |   |-- ProductForm.jsx
|   |   +-- Dashboard.jsx
```

```
|    |-- context/
|    |    +-- AuthContext.jsx
|    |-- utils/
|    |    +-- axios.js
|    |-- App.jsx
|    |-- main.jsx
|    +-- index.css
|-- package.json
+-- .env
```

Listing 3.2 – Arborescence du frontend

Chapitre 4

Fonctionnalités Détaillées

4.1 Authentification et Autorisation

4.1.1 Inscription des Utilisateurs

L'application permet aux nouveaux utilisateurs de créer un compte avec validation en temps réel.

Processus d'inscription :

1. L'utilisateur remplit le formulaire (nom, email, mot de passe, rôle)
2. Validation côté client (React Hook Form)
3. Envoi des données au backend
4. Validation côté serveur (Express Validator)
5. Hashage du mot de passe avec bcrypt
6. Création de l'utilisateur dans MongoDB
7. Génération d'un token JWT
8. Redirection vers le dashboard

FIGURE 4.1 – Page d'inscription avec validation

4.1.2 Connexion Sécurisée

La connexion utilise JWT pour une authentification stateless et sécurisée.

Mécanisme de connexion :

— Saisie email et mot de passe
— Recherche de l'utilisateur par email
— Comparaison du mot de passe avec bcrypt.compare()
— Génération d'un token JWT (expiration 30 jours)
— Stockage du token dans localStorage
— Ajout automatique du token aux requêtes (Axios interceptor)

[CAPTURE : PAGE DE CONNEXION]

Montrer le formulaire avec gestion des erreurs

FIGURE 4.2 – Page de connexion

4.1.3 Gestion des Rôles (RBAC)

L'application implémente un système de contrôle d'accès basé sur les rôles :

Rôle	Permissions	Restrictions
user	— Consulter les produits — Rechercher et filtrer — Voir les détails	— Ne peut pas créer de produits — Pas d'accès au dashboard
admin	— Toutes permissions "user" — Créer des produits — Modifier des produits — Supprimer des produits — Accès au dashboard	— Responsabilité complète sur le catalogue

TABLE 4.1 – Permissions par rôle

4.2 Gestion des Produits (CRUD)

4.2.1 Création de Produit (Admin)

Les administrateurs peuvent ajouter de nouveaux produits avec upload d'image.

[CAPTURE : FORMULAIRE CRÉATION PRODUIT]

Montrer tous les champs : nom, description, prix,
catégorie, stock, upload d'image avec aperçu

FIGURE 4.3 – Formulaire de création de produit

Validation implémentée :

— Nom : 3-100 caractères, obligatoire
— Description : 10-1000 caractères, obligatoire
— Prix : Nombre > 0, obligatoire
— Catégorie : 7 valeurs prédéfinies, obligatoire
— Stock : Entier 0, obligatoire
— Image : JPEG/PNG/GIF/WEBP, max 5MB, optionnel

4.2.2 Liste des Produits

La page principale affiche tous les produits dans une grille responsive.

[CAPTURE : LISTE DES PRODUITS]

Montrer la grille avec cartes produits,
barre de recherche, filtres et pagination

FIGURE 4.4 – Liste des produits avec filtres

4.2.3 Modification de Produit (Admin)

Les administrateurs peuvent modifier les informations d'un produit existant.

[CAPTURE : FORMULAIRE MODIFICATION PRODUIT]

Montrer le formulaire pré-rempli avec les données existantes

FIGURE 4.5 – Formulaire de modification de produit

4.2.4 Suppression de Produit (Admin)

La suppression nécessite une confirmation pour éviter les erreurs.

[CAPTURE : MODAL CONFIRMATION SUPPRESSION]

Montrer la boîte de dialogue avec boutons Annuler/Confirmer

FIGURE 4.6 – Confirmation de suppression

4.2.5 Détails d'un Produit

Une page dédiée affiche toutes les informations d'un produit.

[CAPTURE : PAGE DÉTAILS PRODUIT]

Montrer l'image grande taille, toutes les informations,
prix, stock, boutons Modifier/Supprimer (si admin)

FIGURE 4.7 – Page de détails d'un produit

4.3 Recherche et Filtrage Avancés

4.3.1 Recherche Textuelle

La recherche s'effectue simultanément dans le nom et la description des produits.
Implémentation :

— Recherche insensible à la casse

— Utilisation de regex MongoDB

— Recherche en temps réel (debounce 300ms)

— Mise en évidence des résultats

[CAPTURE : RÉSULTATS DE RECHERCHE]

Montrer la barre de recherche avec résultats filtrés

FIGURE 4.8 – Résultats de recherche

4.3.2 Filtrage Multi-Critères

Les utilisateurs peuvent combiner plusieurs filtres :

— **Par catégorie** : Dropdown avec 7 catégories

— **Par plage de prix** : Sliders min/max

— **Par disponibilité** : Checkbox "En stock uniquement"

[CAPTURE : PANNEAU DE FILTRES]

Montrer tous les filtres : catégorie, prix, stock

FIGURE 4.9 – Panneau de filtres

4.3.3 Tri des Résultats

Plusieurs options de tri sont disponibles :

Option	Description
Prix croissant	Du moins cher au plus cher
Prix décroissant	Du plus cher au moins cher
Nom A-Z	Ordre alphabétique
Nom Z-A	Ordre alphabétique inverse
Plus récents	Date de création décroissante
Plus anciens	Date de création croissante

TABLE 4.2 – Options de tri disponibles

4.3.4 Pagination

La pagination permet de naviguer efficacement dans un grand catalogue.
Caractéristiques :

— 12 produits par page (configurable)
— Numéros de pages cliquables
— Boutons Précédent/Suivant
— Affichage "Page X sur Y"
— Compteur total de résultats
— Conservation des filtres entre les pages

[CAPTURE : CONTRÔLES DE PAGINATION]

Montrer les boutons de navigation et numéros de pages

FIGURE 4.10 – Système de pagination

4.4 Upload de Fichiers

4.4.1 Upload d'Images Produits

L'application permet l'upload sécurisé d'images pour les produits.
Sécurité implémentée :

— Vérification du type MIME (pas juste l'extension)
— Limite de taille : 5 MB
— Types autorisés : JPEG, JPG, PNG, GIF, WEBP
— Renommage automatique (timestamp + random)
— Stockage dans dossier dédié (`/uploads`)

[CAPTURE : UPLOAD D'IMAGE AVEC APERÇU]

Montrer le bouton d'upload et l'aperçu de l'image sélectionnée

FIGURE 4.11 – Upload d'image avec aperçu

4.5 Dashboard Administrateur

4.5.1 Statistiques Globales

Le dashboard affiche des statistiques en temps réel sur le catalogue.
Métriques affichées :

— Nombre total de produits

— Valeur totale du stock (prix × quantité)
— Prix moyen des produits
— Stock total (somme des quantités)
— Nombre de produits en rupture de stock
— Nombre de produits à stock faible (< 10 unités)

[CAPTURE : DASHBOARD ADMINISTRATEUR]

Montrer les cartes de statistiques et les graphiques

FIGURE 4.12 – Dashboard administrateur

4.5.2 Distribution par Catégorie

Un tableau affiche la répartition des produits par catégorie.

[CAPTURE : DISTRIBUTION PAR CATÉGORIE]

Montrer le tableau avec nombre de produits et valeur par catégorie

FIGURE 4.13 – Distribution des produits par catégorie

4.5.3 Alertes Stock

Le dashboard signale les produits nécessitant un réapprovisionnement.

— **Rupture de stock** : Badge rouge pour stock = 0
— **Stock faible** : Badge orange pour stock < 10
— Liste déroulante des produits concernés
— Action rapide vers la page de modification

Chapitre 5

Interface Utilisateur et Expérience Utilisateur

5.1 Design System

5.1.1 Palette de Couleurs

L'application utilise une palette de couleurs moderne et cohérente :

Couleur	Code Hex	Utilisation
Bleu primaire	#3B82F6	Boutons principaux, liens
Violet secondaire	#8B5CF6	Accents, gradients
Vert succès	#10B981	Messages de succès, badges positifs
Rouge danger	#EF4444	Erreurs, suppressions
Gris neutre	#6B7280	Texte secondaire, bordures

TABLE 5.1 – Palette de couleurs de l'application

5.1.2 Typographie

— **Police principale** : Inter, system-ui, sans-serif
— **Tailles** : Adaptatives de 0.875rem (14px) à 2.25rem (36px)
— **Poids** : Normal (400), Medium (500), Bold (700)

5.2 Responsive Design

L'application est entièrement responsive avec trois breakpoints principaux :

Appareil	Largeur	Disposition
Mobile	$<$ 640px	1 colonne, menu burger
Tablette	640px - 1024px	2 colonnes, menu condensé
Desktop	$>$ 1024px	3-4 colonnes, menu complet

TABLE 5.2 – Breakpoints responsive

[CAPTURE : VERSION MOBILE + DESKTOP CÔTE À CÔTE]

Montrer la même page sur mobile et desktop

FIGURE 5.1 – Comparaison responsive mobile/desktop

5.3 Navigation

5.3.1 Barre de Navigation

La navbar est présente sur toutes les pages et s'adapte selon l'état de connexion.
Éléments affichés :

— Logo de l'application (lien vers accueil)
— Lien "Produits"
— Si connecté : Lien "Dashboard" (admin uniquement)
— Si connecté : Menu utilisateur avec nom et déconnexion
— Si non connecté : Boutons "Connexion" et "Inscription"

[CAPTURE : BARRE DE NAVIGATION]

Montrer la navbar avec menu utilisateur ouvert

FIGURE 5.2 – Barre de navigation

5.3.2 Page d'Accueil

La page d'accueil présente l'application avec un design moderne.
Sections :

— Hero section avec gradient animé
— Description de l'application
— Bouton CTA "Voir les Produits"
— Liste des fonctionnalités principales

[CAPTURE : PAGE D'ACCUEIL]

Montrer le hero avec gradient et CTA

FIGURE 5.3 – Page d'accueil

5.4 Animations et Transitions

L'application utilise des animations subtiles pour améliorer l'UX :

— **Hover effects** : Scale et shadow sur les cartes produits
— **Fade-in** : Apparition progressive des éléments au chargement
— **Slide-in** : Modales qui glissent depuis le haut
— **Transitions fluides** : 300ms avec easing naturel

5.5 Feedback Visuel

5.5.1 États de Chargement

Plusieurs indicateurs informent l'utilisateur durant les opérations asynchrones :

— Spinners animés lors du chargement des données
— Skeleton loaders pour les cartes produits
— Texte "Chargement..." explicite
— Désactivation des boutons durant les requêtes

5.5.2 Messages de Succès et d'Erreur

Messages de succès :

— Toast notifications (coin supérieur droit)
— Icône verte avec checkmark
— Auto-dismiss après 3 secondes

Messages d'erreur :

— Affichage sous les champs concernés
— Couleur rouge avec icône d'alerte
— Bordure rouge sur les inputs invalides

[CAPTURE : NOTIFICATIONS SUCCÈS/ERREUR]

Montrer les toast notifications

FIGURE 5.4 – Notifications utilisateur

Chapitre 6

Backend et API REST

6.1 Architecture Backend

Le backend suit une architecture MVC (Model-View-Controller) adaptée aux API REST :

- **Models** : Définition des schémas de données (User, Product)
- **Controllers** : Logique métier et traitement des requêtes
- **Routes** : Définition des endpoints et liaison aux controllers
- **Middleware** : Authentification, autorisation, validation

6.2 Endpoints Principaux

6.2.1 Routes d'Authentification

Méthode	Endpoint	Description
POST	/api/auth/register	Inscription d'un nouvel utilisateur
POST	/api/auth/login	Connexion et génération du JWT
GET	/api/auth/me	Récupération du profil utilisateur

TABLE 6.1 – Routes d'authentification

6.2.2 Routes des Produits

Méthode	Endpoint	Description	Accès
GET	/api/products	Liste avec filtres et pagination	Public
GET	/api/products/ :id	Détails d'un produit	Public
POST	/api/products	Création d'un produit	Admin
PUT	/api/products/ :id	Modification d'un produit	Admin
DELETE	/api/products/ :id	Suppression d'un produit	Admin
GET	/api/products/stats/dashboard	Statistiques	Admin

TABLE 6.2 – Routes des produits

6.3 Tests avec Postman

L'API a été testée de manière exhaustive avec Postman.

[CAPTURE : TESTS POSTMAN]

Montrer une requête POST de création de produit
avec le token JWT dans les headers et la réponse JSON

FIGURE 6.1 – Tests API avec Postman

6.4 Sécurité de l'API

6.4.1 Authentification JWT

Toutes les routes protégées nécessitent un token JWT valide dans le header :

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

6.4.2 Validation des Données

Express Validator assure la validation stricte des entrées :

- Validation des types de données
- Vérification des formats (email, prix, etc.)
- Sanitization (trim, escape)
- Messages d'erreur explicites

6.4.3 Protection contre les Vulnérabilités

Mesures implémentées :

- **Injection NoSQL** : Validation avec Mongoose, pas de `$where`
- **XSS** : Sanitization avec Express Validator
- **CSRF** : Tokens JWT dans headers (pas de cookies)
- **Rate Limiting** : À implémenter pour production
- **CORS** : Configuration stricte des origines autorisées

Chapitre 7

Déploiement et Tests

7.1 Architecture de Déploiement

L'application est déployée sur trois services cloud distincts :

Composant	Service	URL
Frontend	Vercel	https ://[votre-app].vercel.app
Backend	Render	https ://[backend].onrender.com
Database	MongoDB Atlas	Cloud (Europe)

TABLE 7.1 – Services de déploiement

7.2 Configuration du Déploiement

7.2.1 Backend sur Render

Configuration :

— **Build Command** : `npm install`
— **Start Command** : `npm start`
— **Environment** : Node
— **Instance Type** : Free (avec limitations)

Variables d'environnement :

— NODE_ENV=production
— MONGODB_URI=[connection string Atlas]
— JWT_SECRET=[secret sécurisé]
— CLIENT_URL=[URL frontend Vercel]

7.2.2 Frontend sur Vercel

Configuration :

— **Framework** : Vite
— **Build Command** : `npm run build`
— **Output Directory** : `dist`

Variables d'environnement :

— VITE_API_URL=[URL backend Render]

7.3 Application Déployée

[CAPTURE : APPLICATION EN PRODUCTION]

Montrer l'URL de production dans la barre d'adresse
et l'application fonctionnelle

FIGURE 7.1 – Application déployée en production

7.4 Tests Réalisés

7.4.1 Tests Fonctionnels

Tous les scénarios utilisateurs ont été testés :

Test	Description	Résultat
T1	Inscription avec email valide	Réussi
T2	Inscription avec email existant	Erreur 409
T3	Connexion avec credentials valides	Réussi
T4	Connexion avec mauvais mot de passe	Erreur 401
T5	Création produit par admin	Réussi
T6	Création produit par user	Erreur 403
T7	Recherche de produits	Réussi
T8	Filtrage par catégorie	Réussi
T9	Upload image > 5MB	Erreur 413
T10	Pagination	Réussi

TABLE 7.2 – Résultats des tests fonctionnels

7.4.2 Tests de Sécurité

Test	Description	Résultat
S1	Accès route protégée sans token	Erreur 401
S2	Accès route admin par user	Erreur 403
S3	Token JWT expiré	Erreur 401
S4	Injection NoSQL dans login	Bloqué
S5	Upload fichier .exe	Bloqué
S6	XSS dans champ nom	Échappé

TABLE 7.3 – Résultats des tests de sécurité

7.4.3 Tests de Performance

— **Temps de réponse API** : < 500ms (moyenne)
— **Temps de chargement page** : < 2 secondes
— **Lighthouse Score** : 90+ (Performance)

7.5 Monitoring

7.5.1 Outils de Monitoring

— **Render Dashboard** : Logs en temps réel, utilisation CPU/RAM
— **Vercel Analytics** : Nombre de visiteurs, temps de chargement
— **MongoDB Atlas** : Métriques de la base de données

Chapitre 8

Conclusion et Perspectives

8.1 Bilan du Projet

Ce projet de développement d'une application web Full Stack de gestion de produits e-commerce a permis de mettre en pratique l'ensemble des concepts de sécurité des applications web modernes enseignés dans le module.

8.1.1 Objectifs Atteints

Tous les objectifs fixés en début de projet ont été atteints :

- **Authentification sécurisée** : JWT, bcrypt, expiration des tokens
- **Autorisation granulaire** : RBAC avec deux rôles distincts
- **CRUD complet** : Toutes les opérations implémentées
- **Recherche et filtrage** : Fonctionnalités avancées opérationnelles
- **Interface responsive** : Compatible mobile, tablette et desktop
- **Protection OWASP** : Injection, XSS, CSRF traités
- **Déploiement cloud** : Application accessible en production
- **Documentation complète** : 3768 lignes de documentation technique

8.1.2 Compétences Acquises

Ce projet a permis de développer et consolider de nombreuses compétences :
Compétences Techniques :

- Maîtrise de React.js et de l'écosystème frontend moderne
- Développement d'API REST sécurisées avec Node.js/Express
- Conception et modélisation de bases de données NoSQL
- Implémentation des standards de sécurité OWASP
- Déploiement et configuration de services cloud
- Utilisation de Git et GitHub pour le versioning

Compétences Transversales :

- Analyse et conception d'applications complexes
- Gestion de projet et planification
- Résolution de problèmes techniques
- Rédaction de documentation technique
- Veille technologique

8.2 Défis Rencontrés

8.2.1 Défis Techniques

Plusieurs défis techniques ont été rencontrés et surmontés :

1. **Compatibilité Mongoose 9.x** : Adaptation des hooks pre-save (suppression du callback `next()`)
2. **TailwindCSS v4** : Incompatibilité initiale, résolution par downgrade vers v3.4.1
3. **Fast Refresh React** : Warnings ESLint, solution par séparation des hooks
4. **CORS en production** : Configuration stricte des origines autorisées
5. **Render Free Tier** : Gestion du sleep après 15 minutes d'inactivité

8.2.2 Solutions Apportées

Chaque défi a été résolu de manière méthodique :

— Lecture approfondie de la documentation officielle
— Recherche sur GitHub Issues et Stack Overflow
— Tests itératifs et débogage systématique
— Refactoring du code quand nécessaire

8.3 Perspectives et Améliorations Futures

Bien que l'application soit fonctionnelle et sécurisée, plusieurs améliorations peuvent être envisagées :

8.3.1 Fonctionnalités Utilisateur

— **Panier d'achat** : Ajout de produits, gestion des quantités, calcul du total
— **Système de commandes** : Historique, statuts (en cours, livrée), suivi
— **Paiement en ligne** : Intégration Stripe ou PayPal
— **Profil utilisateur** : Modification des informations, avatar
— **Favoris** : Sauvegarder des produits pour plus tard
— **Système de notation** : Notes et avis sur les produits
— **Notifications email** : Confirmation de commande, promotions

8.3.2 Fonctionnalités Administrateur

— **Gestion des utilisateurs** : Liste, modification, suppression, statistiques
— **Gestion des commandes** : Validation, expédition, annulation
— **Statistiques avancées** : Charts avec Chart.js, chiffre d'affaires, produits populaires
— **Export de données** : CSV, PDF pour rapports
— **Promotions** : Codes promo, réductions, ventes flash

8.3.3 Améliorations Techniques

— **Tests automatisés** : Jest, Supertest, React Testing Library
— **CI/CD** : GitHub Actions pour tests automatiques et déploiement
— **Documentation API** : Swagger/OpenAPI pour documentation interactive
— **Multi-langue** : i18next pour internationalisation (FR/EN/AR)
— **Mode sombre** : Toggle light/dark theme
— **PWA** : Progressive Web App avec offline support
— **WebSockets** : Notifications temps réel avec Socket.io
— **GraphQL** : Alternative à REST pour requêtes flexibles
— **Microservices** : Séparation en services indépendants pour scalabilité

8.3.4 Améliorations Sécurité

— **2FA** : Authentification à deux facteurs (SMS, Google Authenticator)
— **Captcha** : Protection contre bots (Google reCAPTCHA)
— **Rate Limiting** : Protection contre brute force et DDoS
— **Audit logs** : Traçabilité complète des actions administrateurs
— **Security headers** : Helmet.js avec CSP stricte
— **Encryption** : Chiffrement des données sensibles au repos
— **Penetration Testing** : Tests d'intrusion professionnels

8.4 Conclusion Finale

Ce projet de développement Full Stack a été une expérience enrichissante qui a permis de consolider les connaissances théoriques en sécurité des applications web modernes par une mise en pratique concrète.

L'application développée démontre une maîtrise des technologies web actuelles (React, Node.js, MongoDB) et l'implémentation effective des meilleures pratiques de sécurité (OWASP Top 10, JWT, bcrypt, RBAC).

Le déploiement réussi en production sur des services cloud professionnels (Render, Vercel, MongoDB Atlas) valide la qualité et la robustesse de l'architecture mise en place.

Ce projet constitue une base solide pour de futurs développements et peut servir de référence pour la création d'applications web sécurisées et performantes.

« La sécurité n'est pas un produit, mais un processus. »
— Bruce Schneier

Bibliographie

[1] OWASP Foundation. (2021). *OWASP Top Ten 2021*.
`https://owasp.org/www-project-top-ten/`

[2] Mozilla Developer Network. (2026). *Web Security*.
`https://developer.mozilla.org/en-US/docs/Web/Security`

[3] Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)*. RFC 7519.
`https://datatracker.ietf.org/doc/html/rfc7519`

[4] Meta. (2026). *React Documentation*.
`https://react.dev`

[5] Express.js Team. (2026). *Express.js Guide*.
`https://expressjs.com`

[6] MongoDB Inc. (2026). *MongoDB Manual*.
`https://docs.mongodb.com`

[7] OpenJS Foundation. (2026). *Node.js Documentation*.
`https://nodejs.org/en/docs/`

[8] Provos, N., & Mazières, D. (1999). *A Future-Adaptable Password Scheme*.
USENIX Annual Technical Conference.

Annexe A

Code Source Important

A.1 Schéma Mongoose User

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Le nom est obligatoire'],
    trim: true,
    minlength: 2,
    maxlength: 50
  },
  email: {
    type: String,
    required: [true, 'L\'email est obligatoire'],
    unique: true,
    lowercase: true,
    match: [/^\S+@\S+\.\S+$/, 'Email invalide']
  },
  password: {
    type: String,
    required: [true, 'Le mot de passe est obligatoire'],
    minlength: 6
  },
  role: {
    type: String,
    enum: ['user', 'admin'],
    default: 'user'
  }
}, { timestamps: true });

// Hash du mot de passe avant sauvegarde
UserSchema.pre('save', async function() {
  if (!this.isModified('password')) return;
  this.password = await bcrypt.hash(this.password, 10);
});

// M thode de comparaison du mot de passe
UserSchema.methods.comparePassword = async function(candidatePassword) {
  return await bcrypt.compare(candidatePassword, this.password);
};

// Exclure le mot de passe des r ponses JSON
UserSchema.methods.toJSON = function() {
  const obj = this.toObject();
  delete obj.password;
  return obj;
};
```

```
48
49 module.exports = mongoose.model('User', UserSchema);
```

Listing A.1 – Modèle User avec bcrypt

A.2 Middleware d'Authentification JWT

```
1  const jwt = require('jsonwebtoken');
2  const User = require('../models/User');
3
4  exports.protect = async (req, res, next) => {
5    let token;
6
7    // Extraction du token depuis le header Authorization
8    if (req.headers.authorization?.startsWith('Bearer')) {
9      token = req.headers.authorization.split(' ')[1];
10   }
11
12   if (!token) {
13     return res.status(401).json({
14       success: false,
15       message: 'Acc s non autoris . Veuillez vous connecter.'
16     });
17   }
18
19   try {
20     // V rification et d codage du token
21     const decoded = jwt.verify(token, process.env.JWT_SECRET);
22
23     // R cup ration de l'utilisateur
24     req.user = await User.findById(decoded.id).select('-password');
25
26     if (!req.user) {
27       return res.status(401).json({
28         success: false,
29         message: 'Utilisateur introuvable'
30       });
31     }
32
33     next();
34   } catch (error) {
35     return res.status(401).json({
36       success: false,
37       message: 'Token invalide ou expir '
38     });
39   }
40 };
41
42 // Middleware d'autorisation par r le
43 exports.authorize = (...roles) => {
44   return (req, res, next) => {
45     if (!roles.includes(req.user.role)) {
46       return res.status(403).json({
47         success: false,
48         message: `Le r le "${req.user.role}" n'est pas autoris `
49       });
```

```
50      }
51      next();
52    };
53  };
```

Listing A.2 – Middleware de protection des routes

Annexe B

Commandes de Déploiement

B.1 Commandes Git

```
# Initialisation
git init
git add .
git commit -m "feat: Initial commit"

# Cr ation repository GitHub
gh repo create fullstack-ecommerce --public

# Push
git remote add origin https://github.com/username/repo.git
git branch -M main
git push -u origin main
```

Listing B.1 – Workflow Git

B.2 Déploiement Backend (Render)

```
# Via Render Dashboard:
# 1. New Web Service
# 2. Connect GitHub repository
# 3. Configuration:
#    - Build Command: npm install
#    - Start Command: npm start
#    - Environment: Node
#
# 4. Environment Variables:
NODE_ENV=production
MONGODB_URI=mongodb+srv://user:pass@cluster.mongodb.net/db
JWT_SECRET=your_secret_here
CLIENT_URL=https://frontend.vercel.app
```

Listing B.2 – Configuration Render

B.3 Déploiement Frontend (Vercel)

```
# Via Vercel Dashboard:
# 1. New Project
# 2. Import Git Repository
# 3. Configuration:
#    - Framework: Vite
#    - Build Command: npm run build
#    - Output Directory: dist
```

```
#     - Root Directory: frontend
#
# 4. Environment Variables:
VITE_API_URL=https://backend.onrender.com/api
```

Listing B.3 – Configuration Vercel