GeeKingdom

Plateforme E-commerce Geek Moderne

« Where Geeks Find Their Kingdom »

Rapport Technique

Développement d'API REST et Site E-commerce

Version : 1.5.0
Date : 14 décembre 2025
Réalisé par : Sedik BEMESSAOUD, Camille DOMMERGUE

Technologies :

Java 17 Spring Boot React 18 MySQL 8.0

Projet de Développement d'API et Site E-commerce

Année Académique 2025-2026

Table des matières

Chapitre 1

Présentation Générale du Projet

1.1 Introduction

GeeKingdom est une plateforme e-commerce complète dédiée aux produits geek (jeux vidéo, manga, culture pop). Le projet combine des API internes robustes avec l'intégration d'une API externe de géolocalisation (OpenStreetMap).

L'architecture permet une séparation claire entre le frontend (React) et le backend (Spring Boot), facilitant la maintenance et les évolutions futures. Ce projet a été réalisé dans le cadre d'un exercice académique visant à maîtriser le développement d'applications web modernes avec des API REST.

1.2 Contexte du Projet

Dans le contexte actuel du commerce en ligne, les plateformes e-commerce doivent répondre à des exigences élevées en termes de :

— **Performance** : Temps de réponse rapides pour une expérience utilisateur fluide

— **Sécurité** : Protection des données personnelles et des transactions

— **Évolutivité** : Capacité à gérer une croissance du nombre d'utilisateurs

— **Maintenabilité** : Code propre et documenté pour faciliter les évolutions

GeeKingdom a été conçu pour répondre à ces exigences tout en proposant une expérience utilisateur moderne et intuitive.

1.3 Objectifs du Projet

— **Site web e-commerce** : Navigation dans les catégories, affichage des produits, gestion du panier et finalisation des commandes

— **API internes** : Gestion des produits, commandes, utilisateurs, stocks, paiements, avis et livraisons

— **API externe** : Intégration de la géolocalisation des points de retrait via OpenStreetMap

— **Documentation API** : Documentation complète permettant à des développeurs tiers d'utiliser les API

1.4 Fonctionnalités Principales

1.4.1 Gestion des Utilisateurs

Le module utilisateurs permet l'inscription et la connexion sécurisées via JWT, la gestion complète du profil (informations personnelles, adresse, préférences), l'accès à l'historique des commandes et un système de rôles différenciés (client et administrateur).

1.4.2 Catalogue Produits

Le catalogue offre une navigation intuitive par catégories, un système de recherche et filtres avancés, des fiches produits détaillées avec images multiples, une gestion des stocks en temps réel et un système de notation avec avis clients modérés.

1.4.3 Panier et Commandes

Le processus d'achat comprend un panier persistant (synchronisé entre localStorage et API), un processus de commande en plusieurs étapes, le support de multiples méthodes de paiement et un suivi des livraisons en temps réel.

1.4.4 Points de Retrait

L'intégration géographique permet la recherche de points de retrait par ville ou par géolocalisation automatique, l'affichage sur carte interactive OpenStreetMap, le calcul des distances avec la formule de Haversine et l'affichage des horaires d'ouverture.

1.4.5 Administration

L'interface d'administration offre la gestion complète des produits et catégories, le suivi et la gestion des commandes, la gestion des stocks avec alertes de rupture, la modération des avis clients et une interface GraphQL pour les opérations avancées.

TABLE 1.1 – Récapitulatif des Fonctionnalités par Module

Module	**Fonctionnalités Clés**
Utilisateurs	Inscription/connexion JWT, profil complet, historique, rôles
Produits	Catalogue, catégories, recherche, fiches détaillées, avis
Commandes	Panier persistant, processus en étapes, paiements multiples
Livraisons	Suivi temps réel, points de retrait géolocalisés
Administration	CRUD complet, stocks, modération, GraphQL

Chapitre 2

Technologies Utilisées

2.1 Stack Technique Globale

Le projet GeeKingdom repose sur une stack technique moderne et éprouvée, combinant des technologies côté serveur et côté client.

TABLE 2.1 – Stack Technologique Complet

Couche	**Technologie**	**Version**	**Rôle**
Backend	Java	17	Langage principal
	Spring Boot	3.5.6	Framework applicatif
	Spring Security	6.x	Sécurité et auth
	Spring Data JPA	3.x	Accès données
Frontend	React	18.x	Interface utilisateur
	React Router	6.x	Navigation SPA
	Context API	-	Gestion d'état
Base de données	MySQL	8.0	SGBD relationnel
	Docker	Latest	Conteneurisation
Sécurité	JWT (JJWT)	0.11.5	Tokens auth
	BCrypt	-	Hashage mots de passe
Outils	Maven	3.x	Build backend
	Node.js	18+	Runtime frontend
	Git	Latest	Versioning

2.2 Backend : Spring Boot

2.2.1 Justification du Choix

Spring Boot a été choisi pour plusieurs raisons :

— **Auto-configuration** : Réduction drastique du code de configuration grâce aux conventions par défaut

— **Starter Dependencies** : Gestion simplifiée des dépendances avec des "starters" pré-configurés

— **Serveur Embarqué** : Tomcat intégré permettant un déploiement simplifié
— **Écosystème Riche** : Spring Security, Spring Data JPA, Spring GraphQL disponibles
— **Production-Ready** : Métriques, health checks et actuators intégrés

2.2.2 Modules Spring Utilisés

— **spring-boot-starter-web** : Contrôleurs REST et serveur embarqué
— **spring-boot-starter-data-jpa** : ORM Hibernate et repositories
— **spring-boot-starter-security** : Authentification et autorisation
— **spring-boot-starter-validation** : Validation des données entrantes
— **spring-boot-starter-graphql** : API GraphQL pour l'administration

2.3 Frontend : React

2.3.1 Architecture React

Le frontend est développé en React 18 avec une architecture basée sur les composants fonctionnels et les hooks modernes.

— **Context API** : Gestion d'état globale pour l'authentification (AuthContext) et le panier (CartContext)
— **React Router 6** : Navigation SPA avec routes protégées
— **Custom Hooks** : Réutilisation de la logique métier
— **CSS Modules** : Styles encapsulés par composant

2.3.2 Organisation des Composants

L'application React est organisée en plusieurs dossiers :

— **components/** : Composants réutilisables (Navbar, ProductCard, MapIframe, Loader, etc.)
— **context/** : Contextes React pour l'état global
— **pages/** : Pages de l'application correspondant aux routes
— **services/** : Service API centralisé avec gestion automatique des tokens
— **styles/** : Fichiers CSS globaux et variables

2.4 Base de Données : MySQL

2.4.1 Choix de MySQL

MySQL 8.0 a été choisi pour sa robustesse, ses performances et sa large adoption. La base de données est conteneurisée avec Docker pour garantir un environnement reproductible.

2.4.2 Avantages de Docker

— Environnement identique sur toutes les machines de développement

— Initialisation automatique avec scripts SQL

— phpMyAdmin inclus pour l'administration visuelle

— Isolation complète du système hôte

Chapitre 3

Architecture du Projet

3.1 Architecture 3-Tiers

L'architecture de GeeKingdom suit le pattern **3-tiers** classique avec une séparation claire des responsabilités :

1. **Couche Présentation** : Application React (port 3000)
2. **Couche Métier** : API Spring Boot (port 8080)
3. **Couche Données** : MySQL (port 3306)

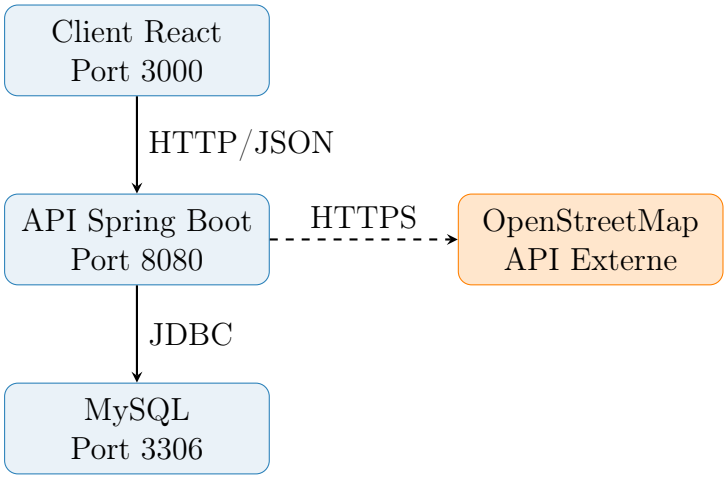

FIGURE 3.1 – Architecture globale de GeeKingdom

3.2 Architecture Backend

3.2.1 Pattern MVC

Le backend suit le pattern **MVC** (Model-View-Controller) adapté aux API REST :

— **Controllers** : Points d'entrée REST, validation des requêtes, délégation aux services

— **Models** : Entités JPA annotées, mappées aux tables MySQL

— **Repositories** : Interfaces Spring Data avec requêtes personnalisées

— **Security** : Filtres de sécurité (JWT, Rate Limit, API Key)

3.2.2 Liste des Contrôleurs

TABLE 3.1 – Contrôleurs REST de l'Application

Contrôleur	**Responsabilité**
UtilisateurController	Authentification et profils
ProduitController	Gestion du catalogue
CategorieController	Gestion des catégories
PanierController	Gestion des paniers
CommandeController	Gestion des commandes
PaiementController	Traitement des paiements
AvisController	Gestion des avis clients
StockController	Gestion des stocks
LivraisonController	Suivi des livraisons
PointRetraitController	Points de retrait
MouvementStockController	Historique des stocks
DetailCommandeController	Lignes de commande

3.3 Architecture Frontend

3.3.1 Structure des Dossiers

L'application React suit une organisation modulaire :

— **src/components/** : Composants UI réutilisables

— **src/context/** : AuthContext (authentification) et CartContext (panier)

— **src/pages/** : Composants de page (Home, Products, Checkout, etc.)

— **src/services/api.js** : Service centralisé pour les appels API

— **src/styles/** : Fichiers CSS

3.3.2 Gestion de l'État

L'état global est géré via Context API :

— **AuthContext** : Utilisateur connecté, token JWT, fonctions login/logout

— **CartContext** : Contenu du panier, fonctions add/remove/update

3.4 Flux d'Authentification

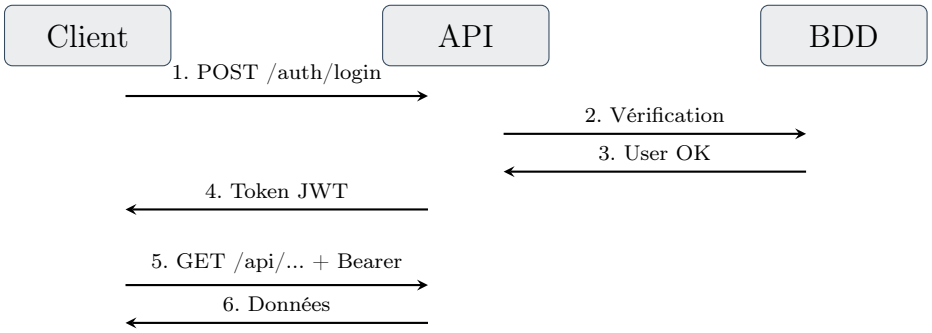

FIGURE 3.2 – Flux d'authentification JWT

Chapitre 4

Base de Données

4.1 Schéma Relationnel

La base de données `geekingdom_db` contient 12 tables interconnectées gérant l'ensemble du cycle e-commerce : utilisateurs, produits, commandes, paiements, livraisons et avis.

4.1.1 Diagramme de la Base de Données

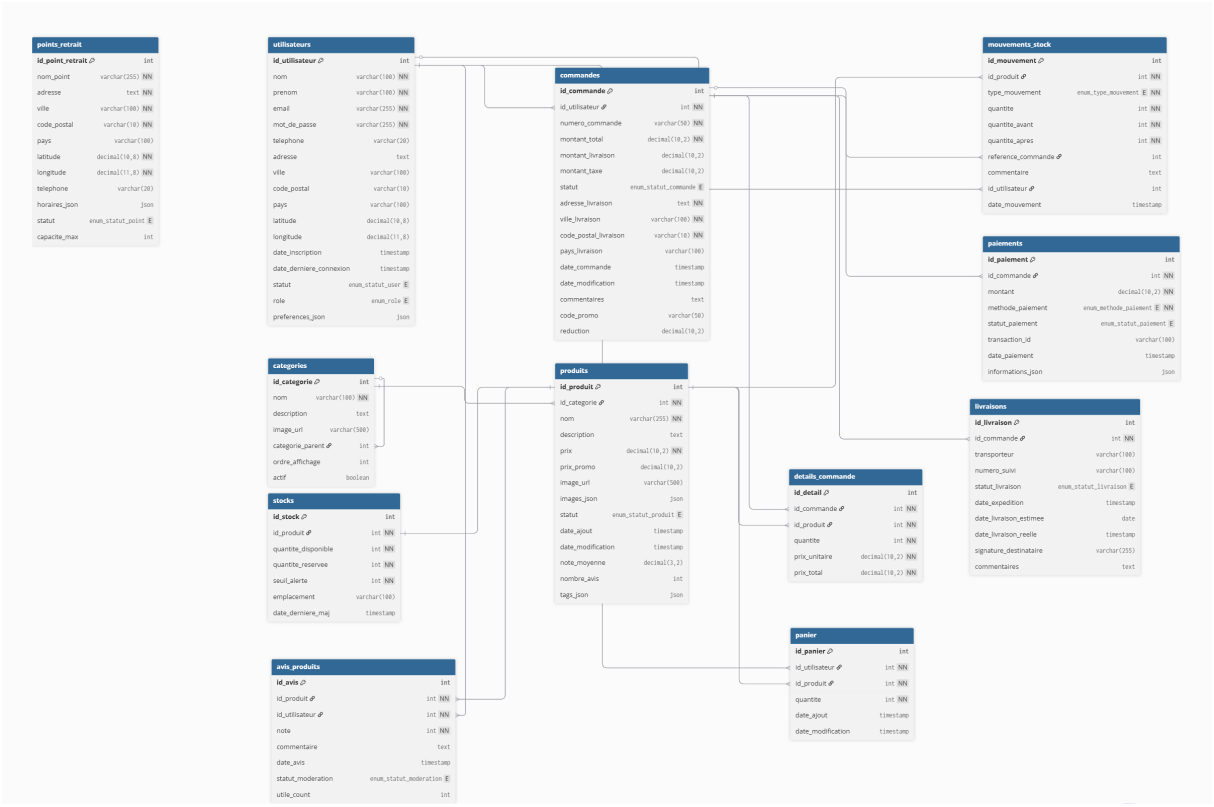

FIGURE 4.1 – Diagramme de la base de données GeeKingdom

4.2 Description des Tables

TABLE 4.1 – Description des Tables de la Base de Données

Table	Description
utilisateurs	Comptes utilisateurs avec informations personnelles, coordonnées GPS, rôles (client/admin) et préférences JSON
categories	Catégories de produits avec support de sous-catégories (parent)
produits	Catalogue complet avec prix, descriptions, images, notes moyennes
stocks	Quantités disponibles et réservées par produit, seuils d'alerte
mouvements_stock	Historique complet des entrées, sorties et ajustements de stock
commandes	Commandes clients avec statuts, adresses et montants
details_commande	Lignes de commande (produit, quantité, prix unitaire)
paiements	Transactions avec méthode, statut et ID de transaction
livraisons	Suivi avec transporteur, numéro de suivi et dates
panier	Paniers temporaires des utilisateurs connectés
avis_produits	Avis clients avec notes (1-5) et système de modération
points_retrait	Points de retrait avec coordonnées GPS et horaires JSON

4.3 Relations entre Tables

Les principales relations de la base de données sont :

- **utilisateurs** → commandes : Un utilisateur peut avoir plusieurs commandes (1 :N)
- **utilisateurs** → panier : Un utilisateur a un panier avec plusieurs produits (1 :N)
- **utilisateurs** → avis_produits : Un utilisateur peut rédiger plusieurs avis (1 :N)
- **categories** → produits : Une catégorie contient plusieurs produits (1 :N)
- **produits** → stocks : Un produit a une entrée de stock (1 :1)
- **produits** → details_commande : Un produit peut apparaître dans plusieurs commandes (1 :N)
- **commandes** → details_commande : Une commande contient plusieurs lignes (1 :N)
- **commandes** → paiements : Une commande peut avoir plusieurs paiements (1 :N)
- **commandes** → livraisons : Une commande a une livraison (1 :1)
- **stocks** → mouvements_stock : Un stock a un historique de mouvements (1 :N)

4.4 Types Énumérés

La base utilise plusieurs types ENUM pour garantir l'intégrité des données :

TABLE 4.2 – Types Énumérés de la Base de Données

Enum	**Valeurs**
statut_utilisateur	actif, inactif, suspendu
role	client, vendeur, admin
statut_produit	disponible, rupture, arrêté, précommande
statut_commande	en_attente, confirmée, en_preparation, expédiée, livrée, annulée
methode_paiement	carte_bancaire, paypal, virement, espèces, chèque
statut_paiement	en_attente, réussi, échoué, remboursé
statut_livraison	en_attente, en_transit, en_cours, livrée, échec, retournée
type_mouvement	entrée, sortie, réservation, libération, ajustement
statut_moderation	en_attente, approuvé, rejeté

Chapitre 5

Gestion des Comptes et Sécurité

5.1 Système de Rôles

GeeKingdom implémente un système de rôles à deux niveaux principaux :

TABLE 5.1 – Permissions par Rôle

Rôle	**Permissions**
Client	Navigation catalogue, gestion panier, passage commandes, rédaction avis, consultation historique, recherche points de retrait
Admin	Toutes les permissions client + gestion utilisateurs, gestion produits/catégories, gestion stocks, modération avis, accès GraphQL admin

5.2 Mécanismes de Sécurité

5.2.1 Hashage des Mots de Passe

Les mots de passe sont hashés avec **BCrypt** avant stockage. BCrypt inclut automatiquement un salt unique pour chaque mot de passe, rendant les attaques par rainbow tables inefficaces.

5.2.2 Authentification JWT

Le système utilise des JSON Web Tokens (JWT) pour l'authentification stateless :

— **Algorithme** : HS256 (HMAC-SHA256)

— **Expiration** : 1 heure

— **Claims** : email (subject), rôle, date d'émission, date d'expiration

— **Stockage client** : localStorage

5.2.3 Chaîne de Filtres

L'API utilise trois filtres de sécurité exécutés dans l'ordre :

1. **RateLimitFilter** : Limite à 5 requêtes par minute par IP pour prévenir les abus
2. **ApiKeyFilter** : Vérifie la clé API pour les endpoints /api/admin/* et /api/external/*
3. **JwtFilter** : Valide le token Bearer et extrait les informations utilisateur

5.2.4 Endpoints Publics

Les endpoints suivants sont accessibles sans authentification :

— `POST /auth/register` - Inscription

— `POST /auth/login` - Connexion

— `GET /api/produits/**` - Consultation du catalogue

— `GET /api/categories/**` - Liste des catégories

— `GET /api/avis/produit/**` - Avis des produits

— `GET /api/points-retrait/**` - Points de retrait

5.3 Protection CORS

La configuration CORS autorise les requêtes cross-origin depuis le frontend React :

— Origines autorisées : `http://localhost:3000`

— Méthodes autorisées : GET, POST, PUT, DELETE, OPTIONS

— Headers autorisés : Authorization, Content-Type

— Credentials : activés pour les cookies

Chapitre 6

Documentation des API

6.1 Vue d'Ensemble

L'API REST de GeeKingdom expose plus de 80 endpoints organisés par ressource. Toutes les réponses sont au format JSON.

6.2 Endpoints d'Authentification

TABLE 6.1 – API Authentification

Endpoint	**Méthode**	**Description**	**Auth**
/auth/register	POST	Inscription utilisateur	Non
/auth/login	POST	Connexion, retourne JWT	Non
/auth/profile/{id}	GET	Consulter profil	Oui
/auth/profile/{id}	PUT	Modifier profil	Oui

6.3 Endpoints des Ressources

TABLE 6.2 – API Ressources Principales

Ressource	**Préfixe**	**Opérations Disponibles**
Catégories	/api/categories	Liste, détail, création, modification, suppression
Produits	/api/produits	Liste (avec filtre catégorie), détail, CRUD complet
Panier	/api/panier	Panier par utilisateur, ajout, modification quantité, suppression, vidage
Commandes	/api/commandes	Liste, par utilisateur, par statut, par numéro, création, modification
Paiements	/api/paiements	Liste, par commande, création, traitement, remboursement, statistiques
Avis	/api/avis	Par produit, par utilisateur, CRUD, approbation, rejet, utile
Stocks	/api/stocks	Par produit, ruptures, alertes, approvisionnement, réservation, libération
Livraisons	/api/livraisons	Par commande, par suivi, création, expédition, livraison
Points retrait	/api/points-retrait	Actifs, par ville, proximité GPS, CRUD, changement statut

6.4 Codes de Réponse HTTP

TABLE 6.3 – Codes HTTP Utilisés

Code	**Statut**	**Utilisation**
200	OK	Requête réussie
201	Created	Ressource créée avec succès
400	Bad Request	Données invalides ou manquantes
401	Unauthorized	Token manquant ou invalide
403	Forbidden	Permissions insuffisantes
404	Not Found	Ressource non trouvée
429	Too Many Requests	Rate limit dépassé
500	Internal Server Error	Erreur serveur

Chapitre 7

Intégration de l'API Externe

7.1 OpenStreetMap

7.1.1 Présentation

OpenStreetMap (OSM) est un projet collaboratif de cartographie libre. L'intégration dans GeeKingdom permet :

— L'affichage des points de retrait sur une carte interactive via iframe

— La géolocalisation des utilisateurs via l'API Navigator du navigateur

— Le calcul des distances entre l'utilisateur et les points de retrait

7.1.2 Intégration via Iframe

L'affichage de la carte utilise l'export embed d'OpenStreetMap. Le composant React `MapIframe` génère dynamiquement l'URL avec les coordonnées et un marqueur.

7.2 Calcul des Distances

7.2.1 Formule de Haversine

La formule de Haversine calcule la distance entre deux points sur une sphère (la Terre) :

$$d = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{7.1}$$

Où :

— $R = 6371$ km (rayon moyen de la Terre)

— ϕ_1, ϕ_2 : latitudes des deux points en radians

— λ_1, λ_2 : longitudes des deux points en radians

— $\Delta\phi = \phi_2 - \phi_1$

— $\Delta\lambda = \lambda_2 - \lambda_1$

7.2.2 Implémentation

La formule est implémentée :

— **Côté Java** : Méthode `distanceFrom()` dans l'entité `PointRetrait`

— **Côté SQL** : Requête native pour filtrer les points dans un rayon donné

7.3 Géolocalisation Utilisateur

La géolocalisation utilise l'API Navigator du navigateur (`navigator.geolocation`). L'utilisateur doit autoriser l'accès à sa position. En cas de refus ou d'indisponibilité, la recherche par ville reste disponible.

Chapitre 8

Outils de Documentation et Test

8.1 Test avec Postman

8.1.1 Configuration

1. Créer une collection "GeeKingdom API"
2. Configurer l'environnement avec `base_url = http://localhost:8080`
3. Ajouter une variable `token` (vide initialement)

8.1.2 Workflow de Test

1. Tester l'inscription : `POST {{base_url}}/auth/register`
2. Tester la connexion : `POST {{base_url}}/auth/login`
3. Stocker le token automatiquement via script Post-Response
4. Utiliser le token : Header `Authorization: Bearer {{token}}`

8.2 Test avec cURL

cURL permet de tester rapidement les endpoints depuis le terminal. Les exemples de commandes sont fournis en annexe.

8.3 Interface GraphQL

L'endpoint `/graphql` expose des queries et mutations pour l'administration :

— **Queries** : Liste des utilisateurs, produits, commandes avec filtres

— **Mutations** : Suspension utilisateur, changement de rôle, mise à jour en masse

L'interface GraphiQL est accessible à `/graphiql` pour les tests interactifs.

8.4 Swagger

Implémentation d'une documentation interactive avec Swagger pour une meilleure prise en main de l'API. L'interface Swagger UI est accessible à `/swagger-ui.html`.

Chapitre 9

Défis Rencontrés et Solutions

9.1 Défis Techniques

TABLE 9.1 – Défis Techniques et Solutions

Défi	**Problème**	**Solution**
CORS	Blocage des requêtes entre React (port 3000) et Spring Boot (port 8080)	Configuration CorsConfigurationSource avec origines, méthodes et headers autorisés
Gestion JWT	Tokens expirés ou invalides non gérés proprement, erreurs 500	Filtre JwtFilter avec try-catch et réponses JSON d'erreur appropriées (401)
Stock concurrent	Race conditions lors de commandes simultanées sur le même produit	Transactions avec @Transactional, réservation de stock avant validation
Géolocalisation	Calcul de distance précis sur une sphère	Implémentation de Haversine en Java et requête SQL native avec HAVING
Rate Limiting	Protection contre les abus sans infrastructure Redis	ConcurrentHashMap avec timestamps et nettoyage périodique en mémoire

9.2 Défis Fonctionnels

TABLE 9.2 – Défis Fonctionnels et Solutions

Défi	**Problème**	**Solution**
Panier persistant	Synchronisation entre localStorage (non connecté) et API (connecté)	Double stockage avec priorité API si authentifié, fusion au login
Modération avis	Filtrage du contenu inapproprié sans modération manuelle constante	Système de statuts (en_attente, approuvé, rejeté) avec file de modération admin
Suivi commandes	Mise à jour des statuts en cascade (commande, paiement, livraison)	Workflow défini avec transitions valides uniquement
Multi-paiements	Support de plusieurs méthodes avec logiques différentes	Enum de méthodes avec traitement générique et informations JSON flexibles
Images produits	Gestion de plusieurs images par produit	Champ images_json (tableau JSON) en plus de image_url principale

9.3 Solutions

1. **Sécurité dès le début** : Intégrer JWT, validation et CORS dès la conception évite des refactorisations coûteuses
2. **Tests API réguliers** : Postman est indispensable pour valider chaque endpoint pendant le développement
3. **Docker simplifie** : La conteneurisation de MySQL élimine les problèmes d'environnement entre développeurs
4. **Documentation continue** : Documenter pendant le développement, pas après, garantit une documentation à jour
5. **Gestion d'erreurs** : Des messages d'erreur clairs en JSON facilitent le débogage côté frontend

Chapitre 10

Améliorations Futures

— Intégration Swagger/OpenAPI pour documentation interactive auto-générée
— Pagination des listes de produits et commandes (page, size, sort)
— Implémentation de filtres avancés (prix, note, disponibilité)
— Tests unitaires (JUnit) et d'intégration (MockMvc) automatisés
— Validation renforcée des entrées avec messages d'erreur détaillés
— Cache Redis pour produits fréquemment consultés et sessions
— Système de recommandation basé sur l'historique d'achat (collaborative filtering)
— Notifications email avec templates (confirmation commande, suivi livraison)
— Intégration paiement réel (Stripe ou PayPal)
— Système de codes promo et réductions
— Upload d'images produits avec stockage cloud (S3)
— Architecture microservices avec découpage par domaine
— Orchestration Kubernetes pour scalabilité horizontale
— Authentification OAuth2 avec providers externes (Google, Facebook)
— Authentification à deux facteurs (2FA) par SMS ou authenticator
— Application mobile React Native
— Dashboard analytics admin avec graphiques et KPIs
— Système de chat support client en temps réel (WebSocket)

Conclusion

Le projet **GeeKingdom** représente une implémentation complète d'une plateforme e-commerce moderne, démontrant la maîtrise des technologies web actuelles.

Réalisations Principales

— **Architecture robuste** : Séparation claire 3-tiers avec React, Spring Boot et MySQL

— **API REST complète** : Plus de 80 endpoints documentés couvrant tout le cycle e-commerce

— **Sécurité multicouche** : JWT, BCrypt, rate limiting et gestion des rôles

— **Intégration externe réussie** : OpenStreetMap pour la géolocalisation des points de retrait

— **Base de données structurée** : 12 tables avec relations cohérentes et contraintes d'intégrité

Compétences Acquises

Ce projet a permis de développer des compétences en :

— Développement d'API REST avec Spring Boot

— Développement frontend avec React et hooks

— Modélisation de base de données relationnelle

— Sécurisation d'applications web

— Intégration d'API tierces

— Conteneurisation avec Docker

Perspectives

Les améliorations identifiées (cache, recommandations, paiements réels, microservices) tracent la voie vers une plateforme de production capable de gérer un volume important d'utilisateurs et de transactions.

Annexe A

Exemples de Code Backend

A.1 Contrôleur d'Authentification

Listing A.1 – UtilisateurController.java - Extrait

```java
@RestController
@RequestMapping("/auth")
public class UtilisateurController {

    @Autowired
    private UtilisateurRepository utilisateurRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody Utilisateur
        utilisateur) {
        // Verification email unique
        if (utilisateurRepository.existsByEmail(utilisateur.getEmail
            ())) {
            return ResponseEntity.badRequest()
                .body(Map.of("error", "Email deja utilise"));
        }

        // Hashage BCrypt du mot de passe
        String hashedPassword = passwordEncoder.encode(utilisateur.
            getMotDePasse());
        utilisateur.setMotDePasse(hashedPassword);

        // Valeurs par defaut
        utilisateur.setDateInscription(LocalDateTime.now());
        utilisateur.setStatut(Utilisateur.Statut.actif);
        utilisateur.setRole(Utilisateur.Role.client);

        Utilisateur saved = utilisateurRepository.save(utilisateur);
        return ResponseEntity.ok(Map.of(
```

```
30              "message", "Utilisateur cree avec succes",
31              "id", saved.getIdUtilisateur()
32          ));
33      }
34
35      @PostMapping("/login")
36      public ResponseEntity<?> login(@RequestBody Map<String, String>
            body) {
37          String email = body.get("email");
38          String password = body.get("password");
39
40          Utilisateur user = utilisateurRepository.findByEmail(email).
                orElse(null);
41
42          if (user != null && passwordEncoder.matches(password, user.
                getMotDePasse())) {
43              // Generation du token JWT avec role
44              String token = JwtUtil.generateToken(
45                  email,
46                  "ROLE_" + user.getRole().name().toUpperCase()
47              );
48
49              // Construction de la reponse
50              Map<String, Object> response = new HashMap<>();
51              response.put("token", token);
52              response.put("user", Map.of(
53                  "id", user.getIdUtilisateur(),
54                  "nom", user.getNom(),
55                  "prenom", user.getPrenom(),
56                  "email", user.getEmail(),
57                  "role", user.getRole()
58              ));
59
60              return ResponseEntity.ok(response);
61          }
62
63          return ResponseEntity.status(401)
64              .body(Map.of("error", "Identifiants invalides"));
65      }
66  }
```

A.2 Utilitaire JWT

Listing A.2 – JwtUtil.java

```
1   public class JwtUtil {
2
3       private static final String SECRET = "
            votre_cle_secrete_256_bits_minimum";
```

```
4       private static final Key KEY = Keys.hmacShaKeyFor(SECRET.getBytes
            ());
5       private static final long EXPIRATION_TIME = 60 * 60 * 1000; // 1
            heure
6
7       public static String generateToken(String username, String role)
            {
8           Map<String, Object> claims = new HashMap<>();
9           claims.put("role", role);
10
11          return Jwts.builder()
12                  .setClaims(claims)
13                  .setSubject(username)
14                  .setIssuedAt(new Date(System.currentTimeMillis()))
15                  .setExpiration(new Date(System.currentTimeMillis() +
                        EXPIRATION_TIME))
16                  .signWith(KEY, SignatureAlgorithm.HS256)
17                  .compact();
18      }
19
20      public static boolean validateToken(String token) {
21          try {
22              Claims claims = Jwts.parserBuilder()
23                      .setSigningKey(KEY)
24                      .build()
25                      .parseClaimsJws(token)
26                      .getBody();
27              return claims.getExpiration().after(new Date());
28          } catch (Exception e) {
29              return false;
30          }
31      }
32
33      public static Key getSigningKey() {
34          return KEY;
35      }
36  }
```

A.3 Filtre JWT

Listing A.3 – JwtFilter.java

```
1   @Component
2   public class JwtFilter extends OncePerRequestFilter {
3
4       @Override
5       protected void doFilterInternal(HttpServletRequest request,
6               HttpServletResponse response, FilterChain filterChain)
7               throws ServletException, IOException {
```

```
8
9           // Ignorer les endpoints publics
10          String path = request.getServletPath();
11          if (path.startsWith("/auth")) {
12              filterChain.doFilter(request, response);
13              return;
14          }
15
16          // Recuperer le header Authorization
17          String authHeader = request.getHeader("Authorization");
18
19          if (authHeader == null || !authHeader.startsWith("Bearer "))
                {
20              filterChain.doFilter(request, response);
21              return;
22          }
23
24          String token = authHeader.substring(7);
25
26          if (!JwtUtil.validateToken(token)) {
27              response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
28              response.setContentType("application/json");
29              response.getWriter().write("{\"error\": \"Token invalide
                    \"}");
30              return;
31          }
32
33          try {
34              Claims claims = Jwts.parserBuilder()
35                  .setSigningKey(JwtUtil.getSigningKey())
36                  .build()
37                  .parseClaimsJws(token)
38                  .getBody();
39
40              String username = claims.getSubject();
41              String role = (String) claims.get("role");
42
43              UsernamePasswordAuthenticationToken authentication =
44                  new UsernamePasswordAuthenticationToken(username,
                        null,
45                          Collections.singletonList(new
                                SimpleGrantedAuthority(role)));
46              SecurityContextHolder.getContext().setAuthentication(
                    authentication);
47
48          } catch (Exception e) {
49              response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
50              response.getWriter().write("{\"error\": \"Erreur token\"}
                    ");
51              return;
52          }
```

```
53
54          filterChain.doFilter(request, response);
55      }
56  }
```

A.4 Calcul de Distance (Haversine)

Listing A.4 – PointRetrait.java - Methode distanceFrom

```
1  public double distanceFrom(BigDecimal lat, BigDecimal lon) {
2      double R = 6371; // Rayon de la Terre en km
3
4      double lat1 = Math.toRadians(this.latitude.doubleValue());
5      double lat2 = Math.toRadians(lat.doubleValue());
6      double deltaLat = Math.toRadians(lat.doubleValue() - this.
           latitude.doubleValue());
7      double deltaLon = Math.toRadians(lon.doubleValue() - this.
           longitude.doubleValue());
8
9      double a = Math.sin(deltaLat/2) * Math.sin(deltaLat/2) +
10                Math.cos(lat1) * Math.cos(lat2) *
11                Math.sin(deltaLon/2) * Math.sin(deltaLon/2);
12
13     double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
14
15     return R * c; // Distance en kilometres
16 }
```

Annexe B

Exemples de Code Frontend

B.1 Service API

Listing B.1 – api.js - Service centralise

```
const API_BASE_URL = 'http://localhost:8080';

const apiRequest = async (endpoint, options = {}) => {
  const token = localStorage.getItem('token');

  const config = {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      ...(token && { 'Authorization': 'Bearer ' + token }),
      ...options.headers,
    },
  };

  const response = await fetch(API_BASE_URL + endpoint, config);

  if (!response.ok) {
    if (response.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    throw new Error('HTTP error! status: ' + response.status);
  }

  return response.json();
};

// API Produits
export const produitsAPI = {
  getAll: (categorieId) => {
    const url = categorieId
      ? '/api/produits?categorie=' + categorieId
```

```
      : '/api/produits';
    return apiRequest(url);
  },
  getById: (id) => apiRequest('/api/produits/' + id),
  create: (data) => apiRequest('/api/produits', {
    method: 'POST',
    body: JSON.stringify(data),
  }),
  update: (id, data) => apiRequest('/api/produits/' + id, {
    method: 'PUT',
    body: JSON.stringify(data),
  }),
  delete: (id) => apiRequest('/api/produits/' + id, {
    method: 'DELETE',
  }),
};

// API Authentification
export const authAPI = {
  register: (data) => apiRequest('/auth/register', {
    method: 'POST',
    body: JSON.stringify(data),
  }),
  login: (data) => apiRequest('/auth/login', {
    method: 'POST',
    body: JSON.stringify(data),
  }),
};
```

B.2 Contexte d'Authentification

Listing B.2 – AuthContext.jsx

```
import React, { createContext, useState, useContext, useEffect } from
    'react';
import { authAPI } from '../services/api';

const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Verifier le token au chargement
    const token = localStorage.getItem('token');
    const savedUser = localStorage.getItem('user');
    if (token && savedUser) {
      setUser(JSON.parse(savedUser));
```

```
    }
    setLoading(false);
  }, []);

  const login = async (email, password) => {
    const response = await authAPI.login({ email, password });
    localStorage.setItem('token', response.token);
    localStorage.setItem('user', JSON.stringify(response.user));
    setUser(response.user);
    return response;
  };

  const logout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    setUser(null);
  };

  const isAuthenticated = () => !!user;

  const hasRole = (role) => user?.role === role;

  return (
    <AuthContext.Provider value={{
      user, login, logout, isAuthenticated, hasRole, loading
    }}>
      {children}
    </AuthContext.Provider>
  );
}

export const useAuth = () => useContext(AuthContext);
```

B.3 Composant Carte OpenStreetMap

Listing B.3 – MapIframe.jsx

```
import React from 'react';

function MapIframe({ latitude = 48.8566, longitude = 2.3522, zoom =
    15 }) {
  // Calcul de la bounding box
  const delta = 0.005;
  const bbox = (longitude - delta) + ',' + (latitude - delta) + ',' +
               (longitude + delta) + ',' + (latitude + delta);

  const mapSrc = 'https://www.openstreetmap.org/export/embed.html' +
                 '?bbox=' + bbox + '&layer=mapnik' +
                 '&marker=' + latitude + ',' + longitude;
```

```
12
13    return (
14      <div className="map-container">
15        <iframe
16          width="100%"
17          height="300"
18          frameBorder="0"
19          scrolling="no"
20          src={mapSrc}
21          title="Carte OpenStreetMap"
22          loading="lazy"
23        />
24        <small>
25          <a
26            href={'https://www.openstreetmap.org/?mlat=' + latitude +
27                  '&mlon=' + longitude + '#map=' + zoom + '/' +
28                  latitude + '/' + longitude}
29            target="_blank"
30            rel="noopener noreferrer"
31          >
32            Voir sur OpenStreetMap
33          </a>
34        </small>
35      </div>
36    );
37  }
38
39  export default MapIframe;
```

Annexe C

Exemples de Requêtes SQL

C.1 Création des Tables

Listing C.1 – Structure table utilisateurs

```sql
CREATE TABLE utilisateurs (
    id_utilisateur INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    mot_de_passe VARCHAR(255) NOT NULL,
    telephone VARCHAR(20),
    adresse TEXT,
    ville VARCHAR(100),
    code_postal VARCHAR(10),
    pays VARCHAR(100) DEFAULT 'France',
    latitude DECIMAL(10,8),
    longitude DECIMAL(11,8),
    date_inscription TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    statut ENUM('actif', 'inactif', 'suspendu') DEFAULT 'actif',
    role ENUM('client', 'vendeur', 'admin') DEFAULT 'client',
    INDEX idx_email (email),
    INDEX idx_role (role)
);
```

C.2 Requête de Proximité

Listing C.2 – Recherche points de retrait par proximite

```sql
SELECT *,
    (6371 * acos(
        cos(radians(:latitude)) * cos(radians(latitude)) *
        cos(radians(longitude) - radians(:longitude)) +
        sin(radians(:latitude)) * sin(radians(latitude))
    )) AS distance
```

```
7  FROM points_retrait
8  WHERE statut = 'actif'
9  HAVING distance <= :rayon
10 ORDER BY distance;
```

Annexe D

Commandes cURL de Test

Listing D.1 – Tests API avec cURL

```
# ========================================
# AUTHENTIFICATION
# ========================================

# Inscription
curl -X POST http://localhost:8080/auth/register \
  -H "Content-Type: application/json" \
  -d '{
    "nom": "Dupont",
    "prenom": "Jean",
    "email": "jean@example.com",
    "motDePasse": "password123"
  }'

# Connexion
curl -X POST http://localhost:8080/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "jean@example.com",
    "password": "password123"
  }'

# ========================================
# PRODUITS (endpoints publics)
# ========================================

# Liste des produits
curl http://localhost:8080/api/produits

# Produits par categorie
curl http://localhost:8080/api/produits?categorie=1

# Detail d'un produit
curl http://localhost:8080/api/produits/1

```

```
# ========================================
# REQUETES AUTHENTIFIEES
# ========================================

# Remplacer <TOKEN> par le token JWT recu au login

# Mes commandes
curl http://localhost:8080/api/commandes/utilisateur/1 \
  -H "Authorization: Bearer <TOKEN>"

# Ajouter au panier
curl -X POST http://localhost:8080/api/panier \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{
    "idUtilisateur": 1,
    "idProduit": 5,
    "quantite": 2
  }'

# Creer une commande
curl -X POST http://localhost:8080/api/commandes \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{
    "idUtilisateur": 1,
    "adresseLivraison": "123 Rue Example",
    "villeLivraison": "Paris",
    "codePostalLivraison": "75001"
  }'
```

Annexe E

Configuration

E.1 application.properties

Listing E.1 – Configuration Spring Boot

```
# Base de donnees MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/geekingdom_db
spring.datasource.username=geekingdom_user
spring.datasource.password=Api_Bdml_2025

# JPA / Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Serveur
server.port=8080
spring.application.name=API_GeeKingdom

# GraphQL
spring.graphql.graphiql.enabled=true
spring.graphql.graphiql.path=/graphiql
```

E.2 docker-compose.yml

Listing E.2 – Configuration Docker

```
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: geekingdom_mysql
    restart: always
    environment:
```

```
9         MYSQL_ROOT_PASSWORD: root_password_change_me
10        MYSQL_DATABASE: geekingdom_db
11        MYSQL_USER: geekingdom_user
12        MYSQL_PASSWORD: Api_Bdml_2025
13      ports:
14        - "3306:3306"
15      volumes:
16        - mysql_data:/var/lib/mysql
17        - ./init-scripts:/docker-entrypoint-initdb.d
18
19    phpmyadmin:
20      image: phpmyadmin:latest
21      container_name: geekingdom_phpmyadmin
22      restart: always
23      environment:
24        PMA_HOST: mysql
25        PMA_PORT: 3306
26      ports:
27        - "8081:80"
28      depends_on:
29        - mysql
30
31  volumes:
32    mysql_data:
```

Annexe F

Liens Utiles

— **Documentation Spring Boot** : https://docs.spring.io/spring-boot/
— **Documentation React** : https://react.dev/
— **Documentation MySQL** : https://dev.mysql.com/doc/
— **OpenStreetMap** : https://www.openstreetmap.org/
— **JWT.io** : https://jwt.io/
— **Postman** : https://www.postman.com/
— **dbdiagram.io** : https://dbdiagram.io/
— **Docker** : https://www.docker.com/