Biblio Poche API

Plateforme de vente de livres en ligne

Rapport Technique

API And Webservices (INGE-2-APP-BDML2)
Professeur : Ralph BOU NADER

Réalisé par :

Nika ZARUBINA
Sabrina SADDEDINE

13 décembre 2025

Table des matières

Chapitre 1

Introduction

1.1 Présentation du Projet

Biblio Poche est une API backend complète pour une plateforme de vente de livres en ligne. Le projet offre deux interfaces d'accès aux données : une API REST traditionnelle et une API GraphQL, permettant aux développeurs de choisir l'approche la plus adaptée à leurs besoins.

1.2 Objectifs

— Fournir une API robuste et sécurisée pour la gestion d'une librairie en ligne.

— Implémenter les meilleures pratiques de sécurité (JWT, protection XSS, NoSQL Injection).

— Offrir une documentation complète et des tests de sécurité.

— Permettre une double interface REST et GraphQL.

1.3 Technologies Utilisées

Catégorie	**Technologie**
Runtime	Node.js
Framework	Express.js
Base de données	MongoDB avec Mongoose
API GraphQL	Apollo Server Express
Authentification	JWT (JSON Web Tokens)
Sécurité	bcryptjs, xss, helmet
Documentation	Swagger/OpenAPI 3.0

Table 1.1 – Stack technique du projet

Chapitre 2

Architecture du Projet

2.1 Structure des Dossiers

```
biblio-poche/
|-- backend/
|   |-- graphql/           # Configuration GraphQL
|   |   |-- index.js       # Setup Apollo Server
|   |   |-- schema.js      # Types GraphQL
|   |   |-- resolvers.js   # Resolvers GraphQL
|   |-- middleware/        # Middlewares Express
|   |   |-- auth.js        # Authentification JWT
|   |   |-- admin.js       # Verification role admin
|   |-- models/            # Modeles Mongoose
|   |   |-- Book.js
|   |   |-- User.js
|   |   |-- Order.js
|   |   |-- Payment.js
|   |-- routes/            # Routes REST
|   |   |-- authRoutes.js
|   |   |-- bookRoutes.js
|   |   |-- orderRoutes.js
|   |   |-- paymentRoutes.js
|   |   |-- adminRoutes.js
|   |-- utils/             # Utilitaires
|   |   |-- sanitize.js    # Protection XSS
|   |   |-- helpers.js     # Fonctions partagees
|   |-- server.js          # Point d'entree
|-- frontend/              # Application React
|-- postman/               # Collections Postman
|-- security_tests/        # Tests de securite Python
```

Listing 2.1 – Structure du projet

2.2 Modèles de Données

2.2.1 Modèle User

```
const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true },
```

```
    password: { type: String, required: true },
    role: { type: String, enum: ['user', 'admin'], default: 'user' }
}, { timestamps: true });
```

Listing 2.2 – Schema User

2.2.2 Modèle Book

```
const bookSchema = new mongoose.Schema({
    title: { type: String, required: true },
    author: { type: String, required: true },
    price: { type: Number, required: true },
    category: { type: String, required: true },
    description: { type: String },
    image: { type: String },
    stock: { type: Number, default: 10 },
    rating: { type: Number, default: 0 },
    numReviews: { type: Number, default: 0 },
    reviews: [reviewSchema]
}, { timestamps: true });
```

Listing 2.3 – Schema Book

2.2.3 Modèle Order

```
const orderSchema = new mongoose.Schema({
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
    orderItems: [{
        title: String,
        qty: Number,
        price: Number,
        product: { type: mongoose.Schema.Types.ObjectId, ref: 'Book' }
    }],
    shippingAddress: {
        address: String, city: String,
        postalCode: String, country: String
    },
    totalPrice: { type: Number, required: true },
    status: {
        type: String,
        enum: ['pending', 'confirmed', 'shipped', 'delivered', '
   cancelled'],
        default: 'pending'
    }
}, { timestamps: true });
```

Listing 2.4 – Schema Order

Chapitre 3

API REST

3.1 Endpoints d'Authentification

Méthode	Endpoint	Description
POST	/api/auth/register	Inscription d'un utilisateur
POST	/api/auth/login	Connexion et obtention du token

Table 3.1 – Endpoints d'authentification

3.2 Endpoints des Livres

Méthode	Endpoint	Description	Auth
GET	/api/books	Liste des livres	Non
GET	/api/books/ :id	Détails d'un livre	Non
POST	/api/books	Ajouter un livre	Admin
POST	/api/books/ :id/reviews	Ajouter un avis	Oui

Table 3.2 – Endpoints des livres

3.3 Endpoints des Commandes

Méthode	Endpoint	Description
POST	/api/orders	Créer une commande
GET	/api/orders/myorders	Mes commandes
GET	/api/orders/ :id	Détails d'une commande
PUT	/api/orders/ :id	Modifier l'adresse
PUT	/api/orders/ :id/cancel	Annuler une commande

Table 3.3 – Endpoints des commandes (authentification requise)

3.4 Endpoints Administration

Méthode	Endpoint	Description
GET	/api/admin/stats	Statistiques dashboard
GET	/api/admin/books	Liste des livres (pagination)
POST	/api/admin/books	Ajouter un livre
PUT	/api/admin/books/ :id	Modifier un livre
DELETE	/api/admin/books/ :id	Supprimer un livre
GET	/api/admin/orders	Liste des commandes
PUT	/api/admin/orders/ :id	Modifier statut commande
GET	/api/admin/users	Liste des utilisateurs
PUT	/api/admin/users/ :id/role	Modifier rôle utilisateur

Table 3.4 – Endpoints administration (rôle admin requis)

Chapitre 4

API GraphQL

4.1 Configuration Apollo Server

```
const { ApolloServer } = require('apollo-server-express');

async function createApolloServer(app) {
    const server = new ApolloServer({
        typeDefs,
        resolvers,
        context: ({ req }) => {
            const token = req.headers.authorization || '';
            const user = getUser(token);
            return { user };
        }
    });

    await server.start();
    server.applyMiddleware({ app, path: '/graphql' });
    return server;
}
```

Listing 4.1 – Configuration Apollo Server

4.2 Types GraphQL

```
type Book {
    id: ID!
    title: String!
    author: String!
    price: Float!
    category: String!
    description: String
    image: String
    stock: Int!
    rating: Float
    numReviews: Int
    reviews: [Review]
}

type User {
```

```
    id: ID!
    username: String!
    email: String!
    role: String!
}

type AuthPayload {
    token: String!
    user: User!
}
```

Listing 4.2 – Définition des types

4.3 Queries et Mutations

```
type Query {
    # Livres
    books(category: String): [Book]
    book(id: ID!): Book
    searchBooks(query: String!): [Book]

    # Utilisateur
    me: User
    users: [User]  # Admin only

    # Commandes
    myOrders: [Order]
    order(id: ID!): Order
    allOrders: [Order]  # Admin only

    # Admin
    adminStats: AdminStats
}

type Mutation {
    # Authentification
    register(input: RegisterInput!): AuthPayload
    login(email: String!, password: String!): AuthPayload

    # Livres (Admin)
    createBook(input: BookInput!): Book
    updateBook(id: ID!, input: BookInput!): Book
    deleteBook(id: ID!): Boolean

    # Avis
    addReview(bookId: ID!, input: ReviewInput!): Book

    # Commandes
    createOrder(input: OrderInput!): Order
    cancelOrder(id: ID!): Order
}
```

Listing 4.3 – Queries GraphQL

4.4 Comparaison REST vs GraphQL

Critère	**REST**	**GraphQL**
Flexibilité	Endpoints fixes	Requêtes flexibles
Over-fetching	Possible	Évité
Under-fetching	Possible	Évité
Versioning	Nécessaire	Non nécessaire
Caching	Simple (HTTP)	Plus complexe
Documentation	Swagger/OpenAPI	Schéma auto-documenté

Table 4.1 – Comparaison REST vs GraphQL

Chapitre 5

Sécurité

5.1 Authentification JWT

5.1.1 Génération du Token

```
const { generateToken } = require('../utils/helpers');

// Dans helpers.js
function generateToken(user) {
    return jwt.sign(
        { id: user._id, role: user.role },
        process.env.JWT_SECRET || 'secret',
        { expiresIn: '7d' }
    );
}
```

Listing 5.1 – Génération de token JWT

5.1.2 Middleware d'Authentification

```
const { extractAndVerifyToken, ERRORS } = require('../utils/helpers');

function authMiddleware(req, res, next) {
    const authHeader = req.headers.authorization || req.headers.
    Authorization;
    const user = extractAndVerifyToken(authHeader);

    if (!user) {
        return res.status(401).json({ message: ERRORS.TOKEN_MISSING });
    }

    req.user = user;
    next();
}
```

Listing 5.2 – Middleware auth.js

5.1.3 Protections JWT

— **Algorithm None Attack** : jwt.verify() exige une signature valide.

— **Payload Manipulation** : La signature détecte toute modification.
— **Token Expiration** : Durée de vie limitée (7 jours).

5.2 Protection XSS

5.2.1 Module de Sanitization

```
const xssOptions = {
    whiteList: {
        b: [], i: [], u: [], strong: [], em: [], br: [], p: []
    },
    stripIgnoreTag: true,
    stripIgnoreTagBody: ['script', 'style', 'noscript'],

    onTagAttr: function (tag, name, value) {
        // Bloquer tous les attributs on*
        if (name.toLowerCase().startsWith('on')) {
            return '';
        }
        // Bloquer javascript: dans href/src
        if ((name === 'href' || name === 'src') && value) {
            if (value.toLowerCase().startsWith('javascript:')) {
                return '';
            }
        }
    },

    onIgnoreTag: function (tag, html) {
        const dangerousTags = [
            'script', 'svg', 'iframe', 'object', 'embed'
        ];
        if (dangerousTags.includes(tag.toLowerCase())) {
            return '';
        }
    }
};
```

Listing 5.3 – Configuration XSS (sanitize.js)

5.2.2 Vecteurs XSS Bloqués

Attaque	Protection
`<script>alert('XSS')</script>`	Balise script supprimée
`<svg onload=alert('XSS')>`	SVG et onload bloqués
`<img onerror=alert('XSS')>`	Attributs on* supprimés
`<a href="javascript:...">`	URLs javascript : bloquées
`<body onload=...>`	Attribut onload supprimé

TABLE 5.1 – Vecteurs XSS et protections

5.3 Protection NoSQL Injection

```
// Schema Mongoose avec typage strict
const userSchema = new mongoose.Schema({
    email: { type: String, required: true },  // Attend une STRING
    password: { type: String, required: true }
});

// bcrypt.compare() protege contre l'injection
userSchema.methods.comparePassword = async function(candidatePassword) {
    // Un objet {"$ne": ""} devient "[object Object]"
    // qui ne matchera jamais le hash
    return bcrypt.compare(candidatePassword, this.password);
};
```

Listing 5.4 – Protection contre NoSQL Injection

5.4 Autres Protections

- **Helmet** : Headers HTTP sécurisés.
- **CORS** : Configuration des origines autorisées.
- **Rate Limiting** : 1000 requêtes / 15 minutes.
- **Validation des entrées** : Mongoose schema validation.

Chapitre 6

Module Utilitaire (helpers.js)

6.1 Objectif

Le module `helpers.js` centralise les fonctions répétées dans le projet pour éliminer les redondances et faciliter la maintenance.

6.2 Constantes d'Erreurs

```
const ERRORS = {
    // Authentification
    TOKEN_MISSING: 'Non autorise, token manquant',
    TOKEN_INVALID: 'Token invalide',
    ACCESS_DENIED: 'Acces refuse',
    ADMIN_REQUIRED: 'Acces refuse. Droits administrateur requis.',
    NOT_AUTHENTICATED: 'Non authentifie. Veuillez vous connecter.',

    // Ressources
    BOOK_NOT_FOUND: 'Livre introuvable',
    ORDER_NOT_FOUND: 'Commande introuvable',
    USER_NOT_FOUND: 'Utilisateur introuvable',
    PAYMENT_NOT_FOUND: 'Paiement introuvable',

    // Validation
    MISSING_FIELDS: 'Champs manquants',
    INVALID_CREDENTIALS: 'Identifiants invalides',
    EMAIL_EXISTS: 'Email deja utilise',

    // Commandes
    ORDER_CANNOT_MODIFY: 'Cette commande ne peut plus etre modifiee',
    ORDER_CANNOT_CANCEL: 'Cette commande ne peut plus etre annulee'
};
```

Listing 6.1 – Constantes d'erreurs centralisées

6.3 Fonctions JWT

```
function generateToken(user) {
    return jwt.sign(
        { id: user._id, role: user.role },
```

```
        process.env.JWT_SECRET || 'secret',
        { expiresIn: '7d' }
    );
}

function extractAndVerifyToken(authHeader) {
    if (!authHeader || !authHeader.startsWith('Bearer ')) {
        return null;
    }
    const token = authHeader.split(' ')[1];
    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET || '
    secret');
        return { id: decoded.id, role: decoded.role };
    } catch (error) {
        return null;
    }
}
```

Listing 6.2 – Fonctions JWT centralisées

6.4 Fonctions de Gestion des Stocks

```
async function checkStockAvailability(orderItems) {
    for (const item of orderItems) {
        const book = await Book.findById(item.product);
        if (!book || book.stock < item.qty) {
            return { success: false, error: `Stock insuffisant` };
        }
    }
    return { success: true };
}

async function decrementStock(orderItems) {
    for (const item of orderItems) {
        await Book.findByIdAndUpdate(item.product, {
            $inc: { stock: -item.qty }
        });
    }
}

async function incrementStock(orderItems) {
    for (const item of orderItems) {
        await Book.findByIdAndUpdate(item.product, {
            $inc: { stock: item.qty }
        });
    }
}
```

Listing 6.3 – Gestion centralisée des stocks

6.5 Fonctions de Vérification

```
function isOwnerOrAdmin(resourceUserId, currentUserId, userRole) {
    const isOwner = resourceUserId.toString() === currentUserId.toString
    ();
    const isAdmin = userRole === 'admin';
    return isOwner || isAdmin;
}

function canModifyOrder(status) {
    const nonModifiableStatuses = ['shipped', 'delivered', 'cancelled'];
    return !nonModifiableStatuses.includes(status);
}

function canCancelOrder(status) {
    const nonCancellableStatuses = ['shipped', 'delivered', 'cancelled'
    ];
    return !nonCancellableStatuses.includes(status);
}
```

Listing 6.4 – Fonctions de vérification

Chapitre 7

Tests de Sécurité

7.1 Présentation du Script de Test

Un script Python complet (`security_tests/test_vulnerabilities.py`) a été développé pour tester automatiquement les vulnérabilités de l'API. Ce script teste 3 catégories principales de vulnérabilités du OWASP Top 10.

7.1.1 Configuration

```
# URL de base de l'API
BASE_URL = "http://localhost:5000"

# Credentials de test
TEST_EMAIL = "test@example.com"
TEST_PASSWORD = "password123"

# Compteurs de resultats
results = {
    "passed": 0,       # Tests reussis (attaque bloquee)
    "failed": 0,       # Tests echoues (vulnerabilite trouvee)
    "warnings": 0,     # Avertissements
    "total": 0
}
```

Listing 7.1 – Configuration du script de test

7.1.2 Exécution

```
# Installation des dependances
cd security_tests
pip install -r requirements.txt

# Lancer les tests
python test_vulnerabilities.py
```

Listing 7.2 – Exécution des tests

7.2 Tests JWT (JSON Web Token)

Les JWT sont composés de 3 parties : Header, Payload et Signature. Plusieurs attaques sont possibles sur les JWT mal implémentés.

7.2.1 Test 1 : Algorithm None Attack

```python
def test_jwt_algorithm_none():
    """
    Tentative de bypass en definissant l'algorithme a 'none'
    """
    # Creer un header avec alg: none
    fake_header = base64.urlsafe_b64encode(
        json.dumps({"alg": "none", "typ": "JWT"}).encode()
    ).decode().rstrip('=')

    # Token sans signature
    none_token = f"{fake_header}.{payload}."

    response = requests.get(
        f"{BASE_URL}/api/orders/myorders",
        headers={"Authorization": f"Bearer {none_token}"}
    )

    # Doit retourner 401 (non autorise)
    assert response.status_code == 401
```

Listing 7.3 – Test Algorithm None

Résultat : SÉCURISÉ - Le serveur rejette les tokens avec alg=none.

7.2.2 Test 2 : Payload Manipulation

```python
def test_jwt_payload_manipulation():
    """
    Modification du payload pour changer le role en 'admin'
    """
    # Decoder et modifier le payload
    payload_decoded['role'] = 'admin'

    # Re-encoder avec la meme signature (invalide)
    manipulated_token = f"{header}.{fake_payload}.{signature}"

    response = requests.get(
        f"{BASE_URL}/api/admin/stats",
        headers={"Authorization": f"Bearer {manipulated_token}"}
    )

    # Doit retourner 401 ou 403
    assert response.status_code in [401, 403]
```

Listing 7.4 – Test Manipulation du Payload

Résultat : SÉCURISÉ - La signature est vérifiée, le payload modifié est rejeté.

7.2.3 Test 3 : Invalid Signature

```python
def test_jwt_invalid_signature():
    """
    Token avec une signature aleatoire/invalide
    """
    fake_signature = base64.urlsafe_b64encode(
        b"fake_signature_12345"
    ).decode().rstrip('=')

    invalid_token = f"{header}.{payload}.{fake_signature}"

    response = requests.get(
        f"{BASE_URL}/api/orders/myorders",
        headers={"Authorization": f"Bearer {invalid_token}"}
    )

    assert response.status_code == 401
```

Listing 7.5 – Test Signature Invalide

Résultat : SÉCURISÉ - Les signatures invalides sont rejetées.

7.3 Tests NoSQL Injection

Les injections NoSQL exploitent les opérateurs MongoDB comme `$ne`, `$gt`, `$regex`, `$where`.

7.3.1 Test 1 : Login Bypass avec $ne

```python
def test_nosql_ne_injection():
    """
    Tentative de connexion avec {"$ne": ""} pour bypasser le password
    """
    payload = {
        "email": "test@example.com",
        "password": {"$ne": ""}  # Not equal to empty = true
    }

    response = requests.post(
        f"{BASE_URL}/api/auth/login",
        json=payload
    )

    # Doit retourner 400 (identifiants invalides)
    assert response.status_code == 400
```

Listing 7.6 – Test Injection $ne

Résultat : SÉCURISÉ - Mongoose valide les types, l'objet est rejeté.

7.3.2 Test 2 : Login Bypass avec $gt

```python
def test_nosql_gt_injection():
    """
    Tentative avec {"$gt": ""} (greater than empty string)
    """
    payload = {
        "email": {"$gt": ""},     # Email > "" = n'importe quel email
        "password": {"$gt": ""}  # Password > "" = n'importe quel password
    }

    response = requests.post(
        f"{BASE_URL}/api/auth/login",
        json=payload
    )

    assert response.status_code == 400
```

Listing 7.7 – Test Injection $gt

Résultat : SÉCURISÉ - La validation des entrées bloque l'injection.

7.3.3 Test 3 : Injection $regex

```python
def test_nosql_regex_injection():
    """
    Injection d'expression reguliere dans le login
    """
    payload = {
        "email": {"$regex": ".*"},  # Match n'importe quel email
        "password": "test_password"
    }

    response = requests.post(
        f"{BASE_URL}/api/auth/login",
        json=payload
    )

    assert response.status_code == 400
```

Listing 7.8 – Test Injection $regex

Résultat : SÉCURISÉ - Les regex sont bloqués par le typage Mongoose.

7.3.4 Test 4 : Injection $where (JavaScript)

```python
def test_nosql_where_injection():
    """
    Tentative d'execution de code JavaScript via $where
    """
    payload = {
        "email": "test@example.com",
        "password": "test_password",
        "$where": "sleep(5000)"  # Tente d'executer du JS
    }

```

```
    start_time = time.time()
    response = requests.post(
        f"{BASE_URL}/api/auth/login",
        json=payload,
        timeout=10
    )
    elapsed_time = time.time() - start_time

    # Si le serveur n'a pas dormi, l'injection est bloquee
    assert elapsed_time < 4
```

Listing 7.9 – Test Injection $where

Résultat : SÉCURISÉ - `$where` est ignoré par Mongoose.

7.4 Tests XSS (Cross-Site Scripting)

Les tests XSS vérifient que les entrées utilisateur sont correctement "sanitisées" avant stockage.

7.4.1 Payloads Testés

Nom	Payload	Description
Script basique	`<script>alert('XSS')</script>`	Injection script classique
Event handler	`<img onerror=alert('XSS')>`	Via attribut événement
SVG onload	`<svg onload=alert('XSS')>`	Via balise SVG
Body onload	`<body onload=alert('XSS')>`	Via balise body
JavaScript URL	`javascript:alert('XSS')`	Protocole javascript :
Encoded script	`&#60;script&#62;...`	Encodage HTML
Unicode escape	`<script>\u0061lert...</script>`	Échappement Unicode
Mixed case	`<ScRiPt>alert...</ScRiPt>`	Casse mixte bypass

TABLE 7.1 – Liste des payloads XSS testés

7.4.2 Test XSS dans la Création de Livre

```
def test_xss_book_creation():
    """
    Injection de scripts dans les champs titre/description
    """
    xss_payloads = [
        "<script>alert('XSS')</script>",
        "<svg onload=alert('XSS')>",
        "<img onerror=alert('XSS')>",
        "<body onload=alert('XSS')>"
    ]

    for payload in xss_payloads:
        book_data = {
            "title": "Test Book",
            "author": payload,
```

```
            "price": 9.99,
            "category": "Test",
            "description": payload
        }

        response = requests.post(
            f"{BASE_URL}/api/books",
            json=book_data,
            headers={"Authorization": f"Bearer {token}"}
        )

        if response.status_code == 201:
            book = response.json()
            # Verifier que le payload est sanitize
            assert '<script>' not in book['author']
            assert 'onload' not in book['description'].lower()
            assert 'onerror' not in book['description'].lower()
```

Listing 7.10 – Test XSS Stored

7.4.3 Résultats XSS

Payload	Avant sanitization	Après sanitization
`<script>alert(1)</script>`	Code exécuté	`(vide)`
`<svg onload=alert(1)>`	Code exécuté	`(vide)`
`<img onerror=alert(1)>`	Code exécuté	`&lt;img&gt;`
`<body onload=alert(1)>`	Code exécuté	`(vide)`
`javascript:alert(1)`	URL dangereuse	`alert(1)`

TABLE 7.2 – Transformation des payloads XSS par le module sanitize

Tous les tests XSS : SÉCURISÉS

7.5 Rapport Final des Tests

7.5.1 Synthèse des Résultats

Catégorie	Tests	Sécurisés	Statut
JWT - Algorithm None	1	1	PASSÉ
JWT - Payload Manipulation	1	1	PASSÉ
JWT - Invalid Signature	1	1	PASSÉ
JWT - Format Validation	1	1	PASSÉ
NoSQL - $ne Injection	1	1	PASSÉ
NoSQL - $gt Injection	1	1	PASSÉ
NoSQL - $regex Injection	1	1	PASSÉ
NoSQL - $where Injection	1	1	PASSÉ
NoSQL - Query Parameters	1	1	PASSÉ
XSS - Script basique	1	1	PASSÉ
XSS - Event handlers	1	1	PASSÉ
XSS - SVG onload	1	1	PASSÉ
XSS - Body onload	1	1	PASSÉ
XSS - JavaScript URL	1	1	PASSÉ
XSS - Encoded/Unicode	1	1	PASSÉ
XSS - Commentaires	1	1	PASSÉ
XSS - Reflected	1	1	PASSÉ
TOTAL	**17**	**17**	**100%**

TABLE 7.3 – Synthèse complète des tests de sécurité

7.5.2 Sortie Console du Script

```
======================================================================
          BIBLIO POCHE - TESTS DE SECURITE API
======================================================================

Date: 2025-01-15 14:30:00
Cible: http://localhost:5000
======================================================================
[OK] Serveur accessible

======================================================================
 TESTS JWT (JSON Web Token)
======================================================================

[INFO] Token valide obtenu: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

[TEST] JWT Algorithm None Attack
       Tentative de bypass en definissant l'algorithme a 'none'
```

```
[SECURISE] Algorithm None Attack bloquee

[TEST] JWT Payload Manipulation
       Modification du payload pour changer le role en 'admin'
[INFO] Payload original: {'id': '...', 'role': 'user', 'iat': ...}
[SECURISE] Manipulation du payload bloquee

[TEST] JWT Invalid Signature
       Token avec une signature aleatoire/invalide
[SECURISE] Signature invalide rejetee

======================================================================
 TESTS NoSQL INJECTION (MongoDB)
======================================================================

[TEST] NoSQL Injection - Login Bypass ($ne)
       Tentative avec {"$ne": ""} pour bypasser le password
[SECURISE] Injection $ne bloquee

[TEST] NoSQL Injection - Login Bypass ($gt)
       Tentative avec {"$gt": ""} (greater than empty)
[SECURISE] Injection $gt bloquee

[TEST] NoSQL Injection - $regex
       Injection d'expression reguliere dans le login
[SECURISE] Injection $regex bloquee

[TEST] NoSQL Injection - $where (JavaScript)
       Tentative d'execution de code JavaScript
[SECURISE] Injection $where bloquee

======================================================================
 TESTS XSS (Cross-Site Scripting)
======================================================================

[TEST] XSS Stored - Creation de livre
       Injection de scripts dans les champs titre/description

  Testing: Script basique
  Payload: <script>alert('XSS')</script>
  [SECURISE] Payload sanitize correctement

  Testing: SVG onload
  Payload: <svg onload=alert('XSS')>
  [SECURISE] Payload sanitize correctement

  Testing: Body onload
  Payload: <body onload=alert('XSS')>
  [SECURISE] Payload sanitize correctement

[TEST] XSS Stored - Avis/Commentaires
       Injection de scripts dans les commentaires
[SECURISE] Commentaire sanitize correctement

[TEST] XSS Reflected - Parametres de recherche
       Injection dans les parametres de requete
[SECURISE] Parametres de recherche securises

```

```
76 ======================================================================
77  RAPPORT FINAL
78 ======================================================================
79
80   Tests executes:   17
81   Securises:        17
82   Vulnerables:      0
83   Avertissements:   0
84
85   Taux de securite: 100.0%
86
87   EXCELLENT! Aucune vulnerabilite detectee.
88
89 ======================================================================
```

Listing 7.11 – Exemple de sortie du script de test

Chapitre 8

Choix Techniques et Difficultés Rencontrées

8.1 Choix Techniques

8.1.1 Architecture Backend

Node.js et Express.js

Choix : Node.js avec Express.js comme framework backend.
Justification :

- **Performance** : Architecture non-bloquante et événementielle, idéale pour les API REST avec de nombreuses requêtes I/O.
- **Écosystème NPM** : Accès à des milliers de packages pour accélérer le développement.
- **JavaScript unifié** : Même langage côté serveur et client (React), facilitant le partage de code et la compréhension.
- **Simplicité** : Express.js offre une API minimaliste et flexible pour créer des routes rapidement.

MongoDB avec Mongoose

Choix : Base de données NoSQL MongoDB avec l'ODM Mongoose.
Justification :

- **Flexibilité du schéma** : Les documents JSON s'adaptent facilement aux évolutions du modèle de données.
- **Performance** : Excellentes performances en lecture pour un catalogue de livres.
- **Scalabilité** : Sharding natif pour la mise à l'échelle horizontale.
- **Mongoose** : Validation des schémas, middleware, et protection implicite contre les injections NoSQL.

```
const bookSchema = new mongoose.Schema({
    title: {
        type: String,       // Typage strict = protection NoSQL
        required: true,
        trim: true
```

```
    },
    price: {
        type: Number,
        required: true,
        min: 0              // Validation metier
    },
    stock: {
        type: Number,
        default: 10,
        min: 0
    }
}, { timestamps: true });  // createdAt, updatedAt automatiques
```

Listing 8.1 – Exemple de schéma Mongoose avec validation

Double API : REST et GraphQL

Choix : Implémenter les deux paradigmes d'API.
Justification :

- **REST** : Standard industriel, simple à comprendre, caching HTTP natif, idéal pour les opérations CRUD simples.
- **GraphQL** : Flexibilité des requêtes, évite l'over-fetching, idéal pour le frontend React avec des besoins de données complexes.
- **Cas d'usage REST** : Opérations admin (pagination, filtres), webhooks, intégrations tierces.
- **Cas d'usage GraphQL** : Interface utilisateur (panier, détails produit avec avis), requêtes imbriquées.

Critère	REST	GraphQL
Catalogue livres	GET /api/books	query { books }
Détail + avis	2 requêtes	1 requête imbriquée
Admin stats	GET /api/admin/stats	query { adminStats }
Caching	Simple (HTTP)	Complexe (Apollo)

TABLE 8.1 – Comparaison des usages REST vs GraphQL dans le projet

8.1.2 Sécurité

JWT pour l'Authentification

Choix : JSON Web Tokens avec l'algorithme HS256.
Justification :

- **Stateless** : Pas de session serveur, scalabilité horizontale facilitée.
- **Auto-contenu** : Le token contient les informations utilisateur (id, role).
- **Expiration** : Durée de vie limitée (7 jours) pour limiter l'impact d'un vol de token.
- **HS256** : Algorithme symétrique simple et sécurisé pour une application mono-serveur.

Sanitization XSS Personnalisée

Choix : Module de sanitization personnalisé basé sur la bibliothèque `xss`.
Justification :

- **Configuration par défaut insuffisante** : La bibliothèque `xss` ne bloque pas tous les vecteurs (SVG, event handlers).
- **Whitelist stricte** : Seules les balises de formatage basiques sont autorisées.
- **Blocage des attributs on*** : Prévention des injections via événements JavaScript.
- **Filtrage des protocoles** : Blocage de `javascript:`, `data:`, `vbscript:`.

8.1.3 Organisation du Code

Module Helpers Centralisé

Choix : Création d'un module `utils/helpers.js` pour centraliser les fonctions répétées.
Justification :

- **DRY (Don't Repeat Yourself)** : Élimination des redondances de code.
- **Maintenabilité** : Un seul endroit à modifier en cas de changement.
- **Cohérence** : Messages d'erreur uniformes dans toute l'application.
- **Testabilité** : Fonctions isolées faciles à tester unitairement.

```
module.exports = {
    // Constantes centralisees
    ERRORS,
    ORDER_STATUSES,

    // JWT
    generateToken,
    extractAndVerifyToken,

    // Verification de propriete
    isOwnerOrAdmin,
    isOwner,

    // Gestion des stocks
    checkStockAvailability,
    decrementStock,
    incrementStock,

    // Validation des statuts
    canModifyOrder,
    canCancelOrder,
    isValidOrderStatus
};
```

Listing 8.2 – Structure du module helpers.js

8.2 Difficultés Rencontrées et Solutions

8.2.1 Difficulté 1 : Incompatibilité Apollo Server v4 avec CommonJS

Problème : L'installation initiale de `@apollo/server` (v4+) a provoqué une erreur car ce package est ESM-only (ECMAScript Modules), incompatible avec notre projet CommonJS.

```
Error [ERR_PACKAGE_PATH_NOT_EXPORTED]:
Package subpath './dist/express4' is not defined by "exports"
```

Listing 8.3 – Erreur rencontrée

Cause : Node.js v24 avec un projet en CommonJS (`require()`) ne peut pas importer un package ESM-only.

Solution : Migration vers `apollo-server-express@3.13.0` qui supporte CommonJS.

```
// @apollo/server v4 - ESM only
import { ApolloServer } from '@apollo/server';
import { expressMiddleware } from '@apollo/server/express4';
```

Listing 8.4 – Avant (incompatible)

```
// apollo-server-express v3 - CommonJS compatible
const { ApolloServer } = require('apollo-server-express');

const server = new ApolloServer({ typeDefs, resolvers, context });
await server.start();
server.applyMiddleware({ app, path: '/graphql' });
```

Listing 8.5 – Après (compatible CommonJS)

Fichiers modifiés :

- `package.json` : Changement de dépendance.
- `graphql/index.js` : Nouvelle API Apollo Server.
- `graphql/schema.js` : Import de `gql` depuis apollo-server-express.

8.2.2 Difficulté 2 : Vulnérabilités XSS Non Bloquées

Problème : Les tests de sécurité ont révélé que la bibliothèque `xss` avec sa configuration par défaut ne bloquait pas certains vecteurs d'attaque :

- `<svg onload=alert('XSS')>` - SVG avec event handler.
- `<body onload=alert('XSS')>` - Body avec event handler.

```
Testing: SVG onload
Payload: <svg onload=alert('XSS')>
[VULNERABLE] Event handlers non sanitizes!

Testing: Body onload
Payload: <body onload=alert('XSS')>
[VULNERABLE] Event handlers non sanitizes!
```

Listing 8.6 – Résultats des tests avant correction

Cause : La configuration par défaut de `xss()` n'inclut pas de whitelist stricte et ne bloque pas tous les attributs `on*`.

Solution : Création d'un module de sanitization personnalisé avec :

1. Whitelist stricte de balises autorisées.
2. Callback `onTagAttr` pour bloquer tous les attributs commençant par "on".
3. Callback `onIgnoreTag` pour supprimer les balises dangereuses.
4. Nettoyage supplémentaire avec expressions régulières.

```
const xssOptions = {
    whiteList: {
        b: [], i: [], u: [], strong: [], em: [], br: [], p: []
    },
    stripIgnoreTag: true,
    stripIgnoreTagBody: ['script', 'style', 'noscript'],

    onTagAttr: function (tag, name, value) {
        // Bloquer TOUS les attributs on*
        if (name.toLowerCase().startsWith('on')) {
            return '';  // Supprimer l'attribut
        }
    },

    onIgnoreTag: function (tag, html) {
        const dangerousTags = ['script', 'svg', 'iframe', 'object',
                               'embed', 'body', 'form', 'input'];
        if (dangerousTags.includes(tag.toLowerCase())) {
            return '';  // Supprimer completement
        }
    }
};
```

Listing 8.7 – Solution implémentée dans sanitize.js

Résultat : Tous les tests XSS passent après correction.

8.2.3 Difficulté 3 : Redondance de Code entre REST et GraphQL

Problème : Le code était dupliqué entre les routes REST et les resolvers GraphQL :

— Génération de token JWT : 4 occurrences identiques.

— Vérification de propriété des ressources : 7 occurrences.

— Gestion des stocks (increment/decrement) : 6 occurrences.

— Messages d'erreur : 13+ occurrences avec variations.

Impact :

— Risque d'incohérence lors des modifications.

— Difficulté de maintenance.

— Code plus volumineux et moins lisible.

Solution : Refactorisation avec le module `utils/helpers.js`.

Fichier	Avant	Après	Réduction
middleware/auth.js	21 lignes	16 lignes	-24%
middleware/admin.js	26 lignes	20 lignes	-23%
routes/orderRoutes.js	157 lignes	145 lignes	-8%
routes/paymentRoutes.js	213 lignes	190 lignes	-11%
graphql/resolvers.js	483 lignes	415 lignes	-14%
Total	**900 lignes**	**786 lignes**	**-13%**

TABLE 8.2 – Réduction de code après refactorisation

8.2.4 Difficulté 4 : Bug des Statistiques Admin avec 0 Commandes

Problème : Le dashboard admin affichait des revenus même quand il y avait 0 commandes, ce qui était illogique et trompeur.

```
// Calculait les revenus meme sans commandes
const revenueResult = await Payment.aggregate([
    { $match: { status: 'completed' } },
    { $group: { _id: null, total: { $sum: '$amount' } } }
]);
totalRevenue = revenueResult[0]?.total || 0;
```

Listing 8.8 – Code problématique

Cause : La requête d'agrégation sur les paiements retournait des valeurs résiduelles ou des données de test, indépendamment du nombre de commandes.

Solution : Ajout d'une condition pour ne calculer les revenus que s'il y a des commandes.

```
// Revenus totaux - SEULEMENT si il y a des commandes
let totalRevenue = 0;
if (totalOrders > 0) {
    const revenueResult = await Payment.aggregate([
        { $match: { status: { $in: ['completed', 'partially_refunded'] } } },
        { $group: {
            _id: null,
            total: { $sum: { $subtract: ['$amount', '$refundedAmount'] } }
        }}
    ]);
    totalRevenue = revenueResult[0]?.total || 0;
}

// Top livres - SEULEMENT si il y a des commandes
let topBooks = [];
if (totalOrders > 0) {
    topBooks = await Order.aggregate([...]);
}
```

Listing 8.9 – Solution implémentée

Fichiers modifiés :

- `routes/adminRoutes.js` : Route GET /api/admin/stats.
- `graphql/resolvers.js` : Query adminStats.

8.2.5 Difficulté 5 : Gestion des Stocks lors des Annulations

Problème : Les stocks n'étaient pas correctement remis lors de l'annulation des commandes, causant des incohérences d'inventaire.

Scénarios problématiques :

1. Annulation par l'utilisateur.
2. Annulation par l'admin.
3. Suppression d'une commande par l'admin.
4. Remboursement complet d'un paiement.

Solution : Centralisation de la logique de gestion des stocks dans `helpers.js` et application systématique.

```
async function decrementStock(orderItems) {
    for (const item of orderItems) {
        await Book.findByIdAndUpdate(item.product, {
            $inc: { stock: -item.qty }
        });
    }
}

async function incrementStock(orderItems) {
    for (const item of orderItems) {
        await Book.findByIdAndUpdate(item.product, {
            $inc: { stock: item.qty }
        });
    }
}
```

Listing 8.10 – Fonctions de gestion des stocks

Application :

— `createOrder` : `decrementStock()`.

— `cancelOrder` : `incrementStock()`.

— `deleteOrder` (admin) : `incrementStock()` si statut != cancelled.

— `updateOrder` (admin, status=cancelled) : `incrementStock()`.

8.3 Leçons Apprises

1. **Vérifier la compatibilité des modules** : Toujours vérifier si un package NPM est ESM ou CommonJS avant installation.
2. **Ne pas faire confiance aux configurations par défaut** : Les bibliothèques de sécurité nécessitent souvent une configuration personnalisée.
3. **Factoriser dès le début** : Identifier les patterns répétitifs tôt dans le développement pour éviter la dette technique.
4. **Tester la sécurité en continu** : Les tests automatisés permettent de détecter les régressions de sécurité.
5. **Documenter les décisions** : Garder une trace des choix techniques facilite la maintenance et l'onboarding.

Chapitre 9

Documentation API

9.1 Swagger/OpenAPI

La documentation OpenAPI 3.0 est disponible dans `backend/swagger.json` et couvre :

— 45+ endpoints documentés.

— 11 schémas de données.

— Exemples de requêtes et réponses.

— Codes d'erreur et messages.

9.2 Collection Postman

Une collection Postman complète est fournie avec :

— 35+ requêtes pré-configurées.

— Variables d'environnement.

— Tests automatisés.

— Scripts de pré-requête pour l'authentification.

9.3 Utilisation de l'API

9.3.1 Authentification

```
# Inscription
curl -X POST http://localhost:5000/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"test","email":"test@test.com","password":"123456"}'

# Connexion
curl -X POST http://localhost:5000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"123456"}'
```

Listing 9.1 – Exemple d'authentification

9.3.2 Requêtes Protégées

```
# Recuperer mes commandes
curl -X GET http://localhost:5000/api/orders/myorders \
  -H "Authorization: Bearer <votre_token>"
```

Listing 9.2 – Utilisation du token

Chapitre 10

Conclusion

10.1 Résumé

Le projet Biblio Poche API fournit une solution backend complète et sécurisée pour une plateforme de vente de livres. Les points forts incluent :

— Double interface REST et GraphQL.
— Sécurité renforcée (JWT, XSS, NoSQL Injection).
— Code factorisé et maintenable via le module helpers.js.
— Documentation complète (Swagger, Postman).
— Tests de sécurité automatisés.

10.2 Améliorations Possibles

— Intégration d'un système de paiement réel (Stripe).
— Ajout de tests unitaires et d'intégration.
— Implémentation de WebSockets pour les notifications temps réel.
— Mise en cache avec Redis.
— Déploiement avec Docker et CI/CD.

Annexe A

Liste des Fichiers

Fichier	Description
Backend - Configuration	
server.js	Point d'entrée principal
package.json	Dépendances Node.js
Backend - Models	
models/Book.js	Schema des livres
models/User.js	Schema des utilisateurs
models/Order.js	Schema des commandes
models/Payment.js	Schema des paiements
Backend - Routes REST	
routes/authRoutes.js	Authentification
routes/bookRoutes.js	Gestion des livres
routes/orderRoutes.js	Gestion des commandes
routes/paymentRoutes.js	Gestion des paiements
routes/adminRoutes.js	Administration
Backend - GraphQL	
graphql/index.js	Configuration Apollo Server
graphql/schema.js	Types GraphQL
graphql/resolvers.js	Resolvers GraphQL
Backend - Middleware	
middleware/auth.js	Authentification JWT
middleware/admin.js	Vérification admin
Backend - Utilitaires	
utils/sanitize.js	Protection XSS
utils/helpers.js	Fonctions partagées
Documentation	
swagger.json	Documentation OpenAPI
README.md	Documentation projet
API_DOCUMENTATION.md	Guide API
SECURITY_PROTECTIONS.md	Documentation sécurité
Tests	
security_tests/test_vulnerabilities.py	Tests de sécurité
postman/*.json	Collection Postman