**Rapport Technique — Backend Sécurisé (Spring Boot)**

**1. Objectif du projet**

Ce projet a pour objectif la mise en place d’une API REST sécurisée en Java avec Spring Boot, respectant les principes de l'intégration et la sécurisation des bases de données. Il couvre les aspects suivants :

* Authentification et autorisation JWT
* Chiffrement des données sensibles
* Séparation des rôles (USER / ADMIN)
* Stockage et vérification des tokens d'actualisation
* Sécurité des endpoints

**2. Stack technique**

* **Java 17**
* **Spring Boot 3.4.5**
* **Spring Security**
* **Spring Data JPA**
* **PostgreSQL (via Render)**
* **JWT (jjwt)**
* **BCryptPasswordEncoder**
* **AES pour chiffrement**
* **spring-dotenv** (lecture du fichier .env)

**3. Sécurité implémentée**

**3.1. Authentification JWT**

* Génération d'un token d'accès (JWT signé)
* Génération d’un **refresh token** stocké en base (lié à l’utilisateur)
* Système de **logout** supprimant le refresh token

**3.2. Données sensibles chiffrées**

* Le champ **username** et **email** sont chiffrés en AES
* La clé de chiffrement est chargée via .env

**3.3. Mot de passe sécurisé**

* Haché avec **BCryptPasswordEncoder**

**3.4. Rôles & autorisation**

* Deux rôles : USER, ADMIN
* Les routes sont protégées avec @PreAuthorize
* Un admin peut modifier le rôle d’un utilisateur

**4. Fonctionnalités backend développées**

**AuthController**

* POST /auth/register : inscription avec données chiffrées et rôle USER
* POST /auth/login : connexion avec retour accessToken + refreshToken
* POST /auth/refresh : renouvellement du token
* POST /auth/logout : suppression du refresh token

**UserController**

* GET /users/me : voir ses infos
* PUT /users/me : modifier son profil
* PATCH /users/password : changer de mot de passe
* DELETE /users/me : supprimer son propre compte

**AdminController**

* POST /admin/change-role : changer le rôle d’un utilisateur
* GET /admin/users : voir tous les utilisateurs
* GET /admin/users/search : rechercher par username
* PATCH /admin/users/{id}/enable : activer un utilisateur
* PATCH /admin/users/{id}/disable : désactiver un utilisateur
* DELETE /admin/users/{id} : supprimer un utilisateur

**5. Sécurité des requêtes**

* Tous les endpoints sensibles sont protégés par JWT
* Les rôles sont validés avec @PreAuthorize
* Limitation des accès aux routes selon rôle :
  + /auth/\* → public
  + /users/\* → USER ou ADMIN
  + /admin/\* → ADMIN uniquement

**6. Sécurité supplémentaire (prévue ou recommandée)**

* ✅ Blocage temporaire après plusieurs tentatives échouées (LoginAttemptService)
* 🔒 Ajout possible d’un **filtre anti-XSS** (non encore intégré)
* 🔒 Limiteur de taille des requêtes HTTP à ajouter (RequestSizeFilter)
* 🔒 Ajout d’un **système de logs/audit** recommandé pour les actions critiques

**7. Configuration externalisée**

Aucune clé n’est codée en dur. Toutes les infos sensibles sont chargées depuis un fichier .env grâce à spring-dotenv.

Exemple :

DB\_HOST=...

DB\_USER=...

DB\_PASS=...

JWT\_SECRET=...

ENC\_SECRET\_KEY=...

**8. Résultat**

Le backend est fonctionnel, sécurisé, et respecte les exigences du TP. Il fournit un point de départ solide pour une application sécurisée en production.

Tous les endpoints peuvent être testés via Postman avec les tokens retournés au login.

**9. Améliorations non réalisées mais souhaitées**

Voici les fonctionnalités de sécurité que j'aurais aimé ajouter dans le cadre de ce projet, mais qui n'ont pas été mises en œuvre par manque de temps :

* 🔒 Mise en place d'un **filtre anti-XSS** avec nettoyage des entrées utilisateur (ex : avec Jsoup)
* 🔒 Ajout d'une **limite de taille des requêtes HTTP** (anti DOS) sur les endpoints /auth/\*
* 🔒 Vérification du **User-Agent / IP** lors du renouvellement des tokens pour éviter le vol de refresh token
* 🔒 Configuration complète d’un **Reverse Proxy HTTPS (SSL)** en local pour simuler un environnement de prod
* 🧪 Intégration d’un **framework de tests de sécurité automatisés** (Zap, OWASP, etc.)