Rapport d'analyse de sécurité du *Serveur de Vote*

Analyse fonctionnelle et technique

Projet : Serveur de Vote

Auteur :

LESTIENNE Jules

Date : 4 novembre 2025

Ce document présente une analyse de la sécurité du projet de Serveur de Vote couvrant la sécurité fonctionnelle (usages, exposition d'information) et la sécurité technique (architecture, données, résilience).

Table des matières

1 Introduction **2**

2 Compréhension fonctionnelle **2**

2.1 Objectif du système 2

2.2 Structure logicielle 2

2.3 Principaux endpoints 2

3 Analyse des risques **3**

3.1 Risques fonctionnels et de confidentialité 3

3.2 Risques techniques 3

3.3 Risques liés à la base Neo4J 3

4 Recommandations de sécurité à l'étape de conception **3**

4.1 Authentification et gestion des sessions 3

4.2 Contrôle d'accès et autorisation 4

4.3 Validation et filtrage des entrées 4

4.4 Protection des données et conformité RGPD 4

4.5 Sécurisation de Neo4J 4

4.6 Résilience et monitoring 4

4.7 Sécurisation de la donnée 4

5 Synthèse **5**

1 Introduction

Ce document propose une analyse de sécurité du *Serveur de Vote*, un composant central du projet de démocratie liquide visant à permettre le vote, la délégation et la consultation des résultats de manière sécurisée et transparente.

L'objectif de ce rapport est de :

— Comprendre l'architecture fonctionnelle et logicielle du serveur ;
— Identifier les surfaces d'exposition et les menaces potentielles ;
— Formuler des recommandations de sécurité à l'étape de la conception.

2 Compréhension fonctionnelle

2.1 Objectif du système

Le serveur de vote gère un système de démocratie liquide où chaque utilisateur peut voter pour d'autres membres dans différents domaines (ex. : tech, design, communication). Les votes sont stockés dans une base orientée graphe **Neo4J** afin de modéliser les relations entre votants et votés. Le serveur expose une API REST sécurisée par **JWT** (bearer token) pour les opérations de création, consultation, suppression et agrégation des votes.

2.2 Structure logicielle

L'architecture suit une approche modulaire inspirée de la *Clean Architecture* :

— **app/** : configuration serveur (sécurité, base de données, CORS, lancement) ;
— **api/** : contrôleurs REST pour les différentes routes ;
— **core/** : logique métier (modèles, services, règles, mappers) ;
— **db/** : couche d'accès Neo4J (entités, repositories, scripts) ;
— **tests/** : tests unitaires et d'intégration (incluant la sécurité).

2.3 Principaux endpoints

— **/votes** : création et suppression de votes ;
— **/votes/by-voter/{id}** : votes émis par un utilisateur ;
— **/votes/for-user/{id}** : votes reçus par un utilisateur ;
— **/results**, **/elected** : résultats agrégés et liste des élus ;
— **/publication**, **/threshold** : paramètres de confidentialité et de publication ;
— **/election/{id}/validate** : validation d'une élection (restreinte à certains rôles).

Tous les endpoints sont protégés par un schéma d'authentification **bearerAuth (JWT)**.

3 Analyse des risques

3.1 Risques fonctionnels et de confidentialité

— **Fuite de données sociales** : les relations votant/voté peuvent révéler des préférences ou affinités : restreindre strictement l'accès aux données de votes individuels.
— **Violation de la confidentialité** : le paramètre `publishVotes` doit être géré par consentement explicite (RGPD).
— **Identification indirecte** : agrégations ou statistiques peuvent permettre la réidentification d'un utilisateur.

3.2 Risques techniques

— **Validation du token JWT** : absence de vérification de la signature ou de la correspondance entre le `userId` et le token.
— **Injection Cypher** : si les requêtes Neo4J sont construites dynamiquement, risque d'injesction.
— **Manque de contrôle d'accès** : endpoints sensibles comme `/election/validate` nécessitent un contrôle par rôle (RBAC).
— **IDOR (Insecure Direct Object Reference)** : possibilité d'accéder aux données d'un autre utilisateur via modification de l'ID dans l'URL.
— **DoS (Denial of Service)** : requêtes statistiques ou agrégées volumineuses.
— **CORS ou HTTPS mal configuré** : risque de fuite via requêtes cross-origin.

3.3 Risques liés à la base Neo4J

— Accès direct à la base possible si le port Neo4J est exposé ;
— Compromission potentielle si les comptes Neo4J ne sont pas isolés ;
— Risque d'intégrité si les relations entre utilisateurs ne sont pas validées (auto-vote, duplication).

4 Recommandations de sécurité à l'étape de conception

4.1 Authentification et gestion des sessions

— Utiliser des **JWT à durée de vie courte** (ex. : 15 minutes) avec mécanisme de *refresh token*.
— Vérifier systématiquement la signature, l'audience (`aud`), l'émetteur (`iss`) et la date d'expiration.
— Implémenter une **liste de révocation** pour les tokens invalidés.
— Journaliser toutes les actions critiques (vote, suppression, validation d'élection).

4.2 Contrôle d'accès et autorisation

- Mettre en uvre un **contrôle d'accès basé sur les rôles (RBAC)** :
  - **Utilisateur** : gestion de ses propres votes ;
  - **Admin / comité** : validation des élections ;
  - **Lecteur public** : accès uniquement aux statistiques agrégées.
- Vérifier que l'identifiant du token correspond à l'utilisateur dans l'URL.
- Restreindre les endpoints d'administration et de validation.

4.3 Validation et filtrage des entrées

- Appliquer une validation stricte sur les champs : `userId`, `domain`, `reason`.
- Mettre en place une **liste blanche** pour les domaines valides.
- Utiliser des **requêtes Cypher paramétrées** pour toute interaction avec Neo4J.
- Limiter la fréquence des appels (*rate limiting*) sur les endpoints de vote.

4.4 Protection des données et conformité RGPD

- Chiffrer les identifiants et les relations sensibles dans la base.
- Séparer les données publiques et privées dans les réponses API.
- Garantir le droit à l'effacement (`DELETE /votes`) et à la portabilité.
- Obtenir le consentement explicite avant toute publication automatique.

4.5 Sécurisation de Neo4J

- Utiliser un compte Neo4J dédié avec droits minimaux (lecture/écriture restreinte).
- Ne pas exposer le port Neo4J en externe.
- Vérifier la cohérence des graphes (interdire les auto-votes ou les cycles incohérents).

4.6 Résilience et monitoring

- Limiter la taille des réponses et imposer une pagination.
- Ajouter un cache (ex. : Redis) pour les statistiques.
- Restreindre le endpoint `/health` à des informations minimales.
- Activer les en-têtes de sécurité HTTP (CSP, HSTS, X-Frame-Options).

4.7 Sécurisation de la donnée

- Chiffrement des communications (*data in-transit*) via TLS 1.3 afin de garantir la confidentialité et l'intégrité des échanges entre clients, API et base de données.
- Chiffrement de la donnée stockée (*data at-rest*) à l'aide d'algorithmes robustes (AES-256) pour prévenir toute fuite en cas de compromission du stockage.

— Chiffrement de la donnée traitée et protection des votes (*chiffrement homomorphique*, enclaves de calcul sécurisées, *confidential computing*) afin d'assurer la confidentialité même durant les traitements.
— Classification et identification du niveau de sensibilité des données : distinguer les données publiques (statistiques agrégées), personnelles (identité, préférences) et critiques (vote individuel) afin d'appliquer des mesures de protection proportionnées.

5 Synthèse

Les principaux axes de sécurisation sont les suivants :

— **Authentification robuste** : JWT courts, signés et régulièrement renouvelés.
— **Autorisation granulaire** : correspondance stricte entre le token et lutilisateur, gestion des rôles et des privilèges.
— **Protection et classification de la donnée** : chiffrement systématique des communications et des données stockées, identification du niveau de sensibilité (publique, personnelle, critique) pour adapter le niveau de protection.
— **Intégrité des votes** : validation des règles métiers et contrôle de cohérence du graphe pour éviter les fraudes et doublons.
— **Durcissement de la configuration serveur** : politiques CORS strictes, usage obligatoire de HTTPS, contrôle daccès basé sur les rôles (RBAC) et journalisation des actions sensibles.

Cette architecture modulaire offre une base solide, mais la confidentialité des votes, la protection différentielle des données et la vérification de lintégrité devront être particulièrement soignées lors de la mise en uvre afin de garantir une sécurité de bout en bout.