Audit d'un projet informatique

Projet : DEMPERM

Équipe d'audit : Securite
Date : 18 janvier 2026

Table des matières

1 Résumé exécutif

1.1 Objectif

Ce rapport présente les conclusions de l'audit de sécurité et d'architecture de la plateforme **DEMPERM**. La mission avait pour objectif d'analyser le code source des clients (Web, Android, iOS) et des serveurs (Social, Vote) ainsi que la robustesse du protocole de vote. L'audit vise à identifier les risques pesant sur la confidentialité des données, l'intégrité du scrutin et la disponibilité du service.

1.2 Synthèse des résultats

L'audit a révélé un niveau de risque global **CRITIQUE**. Si l'architecture basée sur des micro-services et la conteneurisation est saine, l'implémentation de la sécurité présente des lacunes fondamentales.

— **Principaux risques :**
  — **Vote non sécurisé :** Le protocole actuel ne garantit ni le secret du vote (lien direct votant-vote en base), ni l'intégrité cryptographique des bulletins (stockés en clair).
  — **Exposition de secrets :** De multiples identifiants critiques (Clés Firebase, mots de passe Neo4j par défaut) sont présents en clair dans le code source.
  — **Communications non chiffrées :** L'usage prédominant de HTTP au lieu de HTTPS expose l'ensemble des données (y compris les tokens d'authentification) à des interceptions.
— **Points forts :** Séparation architecturale nette entre le serveur social et le serveur de vote. Usage de Docker facilitant la reproductibilité des environnements.
— **Axes prioritaires :**
  1. Chiffrement obligatoire des flux (HTTPS) et épinglage de certificats (Pinning).
  2. Externalisation immédiate des secrets (Vault ou Variables d'environnement).
  3. Refonte du stockage des votes pour inclure des preuves cryptographiques (signature) et garantir l'anonymat.

1.3 Vue d'ensemble du risque

Composant	Critique	Élevé	Moyen	Faible	Total
Client Web	1	2	1	0	4
Client Android	4	8	1	0	13
Client iOS	0	2	2	0	4
Serveur Social	1	2	1	1	5
Serveur de Vote	1	3	1	1	6
Protocole	2	3	3	1	9
TOTAL	**9**	**20**	**9**	**3**	**41**

TABLE 1 – Distribution des constats par composant et sévérité.

2 Contexte et périmètre

2.1 Contexte

Le projet DEMPERM est une plateforme hybride alliant réseau social et système de vote électronique. Dans ce contexte, la confiance des utilisateurs repose sur deux piliers antagonistes : l'identification forte pour éviter les fraudes, et l'anonymat strict pour protéger le secret du vote. L'application doit également gérer des interactions sociales classiques (posts, commentaires) avec une haute disponibilité.

2.2 Périmètre audité

L'audit a couvert l'intégralité du code source fourni :

- **Clients Front-end :**
  - Web : React / TypeScript.
  - Android : React Native avec Expo.
  - iOS : Swift (Code natif).
- **Back-end :**
  - Serveur Social : Microservices Node.js, MongoDB, Elasticsearch.
  - Serveur de Vote : Python (Django), Base de données graphe Neo4j.
- **Protocole :** Algorithmes de validation, de stockage et de comptage des votes.
- **DevOps :** Configurations Docker et fichiers d'environnement.

2.3 Hors périmètre

- La sécurité physique des infrastructures d'hébergement.
- La configuration interne des services tiers (console Google Cloud/Firebase), seule l'intégration client a été vérifiée.
- Les tests d'intrusion en boîte noire (Black Box) sur l'environnement de production.

3 Méthodologie

3.1 Approche

- **Revue de code (White Box) :** Analyse statique manuelle et automatisée pour détecter les failles logiques, les mauvaises pratiques de codage et les secrets codés en dur.
- **Revue d'architecture :** Analyse des flux de données, des surfaces d'attaque des conteneurs et de la ségrégation des réseaux.
- **Analyse des dépendances (SCA) :** Vérification des versions des bibliothèques (npm, pip) face aux bases de données de vulnérabilités (CVE).
- **Analyse protocolaire :** Vérification mathématique et logique des propriétés du scrutin (unicité, intégrité, secret).

3.2 Échelle de sévérité

- **Critique :** Compromission totale du système, perte massive de données, ou manipulation avérée du vote (Score > 12).
- **Élevé :** Impact fort sur la sécurité, exploitation probable (ex : DoS facile, accès non autorisé) (Score 9-12).
- **Moyen :** Impact modéré, nécessite des conditions spécifiques ou impacte une partie limitée du système (Score 5-8).
- **Faible :** Non-respect des bonnes pratiques, risque mineur ou théorique (Score < 5).

3.3 Outils et références

- **Outils :** `npm audit`, analyseurs statiques de code, scripts de recherche de motifs (secrets).
- **Standards :** OWASP Mobile Top 10, OWASP Web Top 10, Bonnes pratiques de sécurité Docker.

4 Client Web

4.1 Surface d'attaque

— Authentification / sessions (cookies, JWT, refresh tokens)

— API calls, CORS, CSRF

— Entrées utilisateur (XSS, injection, upload)

— Stockage local (localStorage, IndexedDB)

4.2 Constats

4.2.1 Paquets Node vulnérables

ID	WEB-01
Sévérité	Critique
Composant	Client Web
Impact	Le paquet *react-router-dom* dépend de *react-router*, la version installé étant la 7.9.6, elle est vulnérable aux attaques CSRF et XSS si on utilise certains composants
Reproduction / Preuves	Lancer la commande *npm audit*.
Recommandation	Mettre à jour le paquet, au moins à la version 7.12.0

4.2.2 Versions de paquets non fixées

ID	WEB-02
Sévérité	Élevé
Composant	Client Web
Impact	Vu que les versions des paquets Node ne sont pas fixes, une nouvelle version pourrait s'installer automatiquement et casser l'application, ou faire apparaître des nouveaux bugs ou failles de sécurité
Reproduction / Preuves	Lire le fichier *package.json* et remarquer les "carets" (^) à côté des numéros de versions, indiquant que des mises à jour "mineures" sont acceptées.
Recommandation	Figer les versions, ou ne permettre que des nouveaux "patches" avec le "tilde" (∼)

4.2.3 Clés API en dur

ID	WEB-03
Sévérité	Élevé
Composant	Client Web
Impact	Le fichier de configuration Firebase contient des clés d'API et des identifiants de projet codés en dur directement dans le code source. Bien que les clés Firebase soient souvent considérées comme publiques, il est recommandé de ne pas les exposer dans le dépôt Git pour éviter les abus de quota ou le ciblage par des bots.
Reproduction / Preuves	Lire le fichier *src/client/web/src/firebase-config.ts*
Recommandation	Utilisez des variables d'environnement (fichiers .env) via Vite (import.meta.env.VITE)

4.2.4 Stockage des tokens

ID	WEB-04
Sévérité	Modéré
Composant	Client Web
Impact	Dans le composant de connexion, le token d'identité (ID Token) est récupéré et passé à une fonction *setCredentials*. Si setCredentials stocke ce token dans le localStorage (pratique courante mais risquée), l'application est vulnérable aux attaques XSS (Cross-Site Scripting). Tout script mal- veillant injecté dans la page pourrait exfiltrer ce token.
Reproduction / Preuves	Lire les fichiers *src/client/web/src/app/auth/login.tsx* et *src/client/-web/src/shared/auth.ts*
Recommandation	Privilégiez le stockage des tokens sensibles en mémoire ou via des cookies HttpOnly sécurisés gérés par le backend, si possible.

4.2.5 Risques et conséquences

— Exécution de scripts malveillant (XSS) : vol de données sensibles

— Envoie de requêtes vers des sites dans lesquels la victime pourrait être authentifiée (CSRF) : exécution forcée d'actions non désirées sans le savoir, par exemple faire un virement bancaire

— Abus d'usage des clés API par des personne tiers : abus de quota ou ciblage des bots

— Instabilité des paquets : des nouvelles vulnérabilités et bugs apparaissent

4.3 Recommandations globales

— Faire un audit des paquets régulièrement, afin d'éviter de rester sur une version présentant des failles de sécurité. Pour cela, une "scheduled pipeline" (GitLab) ou un "schedule workflow" (GitHub) pourrait être utilisé : un job ou une GitHub action qui lance la commande *npm audit* pourrait être lancé toutes les semaines ou tous les jours.

— Faire attention à quelles données sont exposées/accessibles, il est conseillé de montrer aussi peu d'information que possible.

— Fixer les versions, il est important de savoir exactement quelles bibliothèques on utilise.

5 Client Android

5.1 Surface d'attaque

5.1.1 Environnement technique

— **Framework** : React Native avec Expo

— **Authentification** : Firebase Authentication

— **Stockage local** : AsyncStorage (non chiffré par défaut)

— **Communication réseau** : API REST en HTTP non chiffré

— **Backend** : API sociale (port 8000), API vote (port 8080)

5.1.2 Composants critiques analysés

— **Gestion des secrets** : Configuration Firebase, tokens JWT

— **Authentification** : Flux de connexion, gestion de session

— **Communications** : Appels API, interception réseau possible
— **Validation** : Entrées utilisateur, données reçues
— **Logging** : Exposition d'informations sensibles

5.2 Constats

5.2.1 Vulnérabilités critiques (Score 13-16)

▲ **VULN-001** : Secrets Firebase hardcodés dans `firebaseConfig.ts`
▲ **VULN-002** : Stockage d'authentification non sécurisé avec AsyncStorage
▲ **VULN-003** : Tokens JWT statiques et expirants dans `social_api_config.ts` et `call_api.tsx`
▲ **VULN-004** : Communication HTTP non chiffrée sur tous les endpoints API

5.2.2 Vulnérabilités modérées (Score 5-8)

▲ **VULN-013** : Pas de validation de complexité de mot de passe (`connection.tsx`)

5.2.3 Synthèse des vulnérabilités

Table 2 – Synthèse et reclassification des vulnérabilités - Client Android

ID	Description	Proba.	Impact	Score	Niveau
VULN-001	Secrets Firebase hardcodés	4	4	16	Critique
VULN-002	Stockage auth non sécurisé	4	4	16	Critique
VULN-003	Tokens JWT statiques	3	4	16	Critique
VULN-004	HTTP non chiffré	4	4	16	Critique
VULN-005	AsyncStorage données sensibles	4	3	12	Élevé
VULN-006	Absence validation entrées	3	3	12	Élevé
VULN-007	Messages d'erreur précis	3	2	12	Élevé
VULN-008	Pas de rate limiting	2	3	12	Élevé
VULN-009	Logs sensibles production	3	3	12	Élevé
VULN-010	Duplication configuration	2	2	12	Élevé
VULN-011	Injection côté client	3	3	12	Élevé
VULN-012	Pas de sanitization données	3	3	12	Élevé
VULN-013	Validation complexité faible	4	2	8	Modéré

Table 3 – Matrice de risque avec niveau de criticité

Impact \ Proba.	1	2	3	4
1	Faible	Faible	Faible	Moyen
2	Faible	VULN-010	VULN-007	VULN-013
3	Faible	VULN-008	VULN-005 VULN-006 VULN-009 VULN-011 VULN-012	Moyen
4	Faible	Moyen	VULN-003	VULN-001 VULN-002 VULN-004

5.2.4 Analyse des impacts pour DemPerm

— **Intégrité démocratique compromise** : Tokens statiques + HTTP → votes falsifiables
— **Confidentialité des données** : Communications interceptables → fuite des débats privés
— **Disponibilité du service** : Secrets Firebase exposés → risque de suppression de la base
— **Confiance des utilisateurs** : Usurpation d'identité possible → perte de crédibilité

5.3 Recommandations globales

5.3.1 Corrections prioritaires (immédiates)

1. **Remplacer HTTP par HTTPS** : Pour toutes les communications API
2. **Externaliser les secrets** : Variables d'environnement pour Firebase
3. **Supprimer les tokens statiques** : Implémenter l'authentification réelle
4. **Sécuriser le stockage** : Remplacer AsyncStorage par SecureStore

5.3.2 Améliorations structurelles (court terme)

— **Validation des entrées** : Implémenter `validator.js` pour tous les formulaires
— **Gestion des erreurs** : Messages génériques, pas de logs sensibles en production
— **Rate limiting** : Limiter les tentatives de connexion (5 max)
— **Sanitization données** : Valider les données reçues du backend

5.3.3 Feuille de route de correction

Table 4 – Feuille de route de sécurisation

Délai	Actions	Vulnérabilités concernées
24-48h	HTTPS + secrets externalisés	VULN-001, VULN-004
1-2 sem.	Authentification réelle	VULN-002, VULN-003
2-4 sem.	Validation + stockage sécurisé	VULN-005 à VULN-012
Prochain trimestre	Bonnes pratiques avancées	VULN-013 + prévention

5.3.4 Métriques de suivi

— **Score de sécurité initial** : 31% (4/13 vulnérabilités critiques)
— **Objectif court terme** : <2 vulnérabilités critiques
— **Objectif moyen terme** : 0 vulnérabilité critique, <4 vulnérabilités élevées

6 Client iOS

6.1 Surface d'attaque

— Authentification / sessions (JWT, access tokens, refresh tokens)
— Appels API HTTPS (MITM, validation TLS)
— Entrées utilisateur (formulaires, champs de connexion)
— Stockage local (Keychain, UserDefaults, cache)

6.2 Constats

6.2.1 Stockage potentiellement non sécurisé des tokens

ID	IOS-01
Sévérité	Élevé
Composant	Client iOS
Impact	Les tokens d'authentification sont récupérés lors de la connexion et utilisés pour authentifier les appels API. Si ces tokens sont stockés dans *UserDefaults* ou dans un stockage non chiffré, ils peuvent être compromis en cas de jailbreak, d'accès physique à l'appareil ou d'analyse dynamique de l'application.
Reproduction / Preuves	Analyser les fichiers du dossier *auth* et vérifier l'utilisation de *UserDefaults.standard.set* ou d'un stockage persistant non sécurisé pour les tokens.
Recommandation	Stocker les tokens sensibles exclusivement dans le *Keychain iOS* avec une politique d'accessibilité restrictive, et les supprimer explicitement lors de la déconnexion.

6.2.2 Absence de pinning TLS

ID	IOS-02
Sévérité	Élevé
Composant	Client iOS
Impact	Les appels réseau utilisent HTTPS, mais sans mécanisme de *certificate pinning*. Un attaquant capable d'installer un certificat malveillant sur l'appareil pourrait intercepter ou modifier les communications réseau, notamment sur des réseaux Wi-Fi publics.
Reproduction / Preuves	Analyser la configuration de *URLSession* dans le dossier *api* et constater l'absence de validation personnalisée du certificat serveur.
Recommandation	Mettre en place un *TLS certificate pinning* via *URLSessionDelegate*, en validant l'empreinte du certificat ou la clé publique du serveur.

6.2.3 Gestion insuffisante des erreurs API

ID	IOS-03
Sévérité	Modéré
Composant	Client iOS
Impact	Les erreurs retournées par l'API (codes HTTP 4xx/5xx) peuvent être affichées ou loggées telles quelles. Cela peut révéler des informations internes sur le backend ou la logique d'authentification, exploitables par un attaquant.
Reproduction / Preuves	Forcer des erreurs réseau ou d'authentification et observer les messages affichés à l'écran ou dans les logs applicatifs.
Recommandation	Mettre en place une gestion centralisée des erreurs avec des messages génériques côté interface utilisateur et limiter les informations exposées.

6.2.4 Présence de données sensibles dans les logs

ID	IOS-04
Sévérité	Modéré
Composant	Client iOS
Impact	Des logs de debug (*print*, *NSLog*) peuvent contenir des données sensibles telles que des tokens, des identifiants utilisateur ou des réponses API complètes. En environnement de production, ces logs peuvent être récupérés lors d'analyses ou de crashs.
Reproduction / Preuves	Rechercher l'utilisation de *print()* ou de logs non conditionnés par un flag de debug dans les dossiers *api* et *auth*.
Recommandation	Supprimer les logs sensibles, utiliser des logs conditionnés (*#if DEBUG*) et ne jamais journaliser de données d'authentification.

6.2.5 Risques et conséquences

— Vol de tokens d'authentification : usurpation de compte

— Interception ou modification des communications réseau (MITM)

— Fuite de données sensibles via les logs applicatifs

— Exposition d'informations internes facilitant des attaques ciblées

6.3 Recommandations globales

— Centraliser la gestion de l'authentification et du stockage des tokens.

— Utiliser exclusivement le *Keychain iOS* pour les données sensibles.

— Mettre en place un mécanisme de *TLS certificate pinning* pour sécuriser les communications réseau.

— Réduire la verbosité des logs en production et auditer régulièrement le code client iOS.

— Appliquer le principe du moindre privilège en limitant strictement les données accessibles côté client.

7 Serveur Social

7.1 Surface d'attaque

7.1.1 Environnement technique

— **Framework** : Service REST (architecture microservices)

— **Base de données** : MongoDB, Elasticsearch, Neo4j

— **Authentification** : Tokens JWT émis par le service d'authentification

— **Communication** : API HTTP inter-services

— **Environnement d'exécution** : Conteneurs Docker

7.1.2 Composants critiques analysés

— **Logique métier** : Gestion des posts, interactions sociales, relations utilisateurs

— **Points d'accès API** : Endpoints REST exposés aux clients

— **Autorisation** : Vérification des droits utilisateur sur les ressources

— **Communication inter-services** : Appels vers les services Auth et Timeline

— **Accès aux bases de données** : Requêtes MongoDB, Elasticsearch et Neo4j

7.2 Constats

7.2.1 Vulnérabilités critiques (Score 13-16)

▲ **VULN-SOC-001** : Absence de contrôle d'autorisation strict lors des opérations sensibles (suppression ou modification de contenus), permettant à un utilisateur authentifié d'agir sur des ressources ne lui appartenant pas.

7.2.2 Vulnérabilités élevées (Score 9-12)

▲ **VULN-SOC-002** : Absence de validation stricte des entrées utilisateur (contenu des posts, paramètres de requêtes), exposant le service à des injections NoSQL ou Elasticsearch.

▲ **VULN-SOC-003** : Absence de limitation de taux (rate limiting) sur les endpoints exposés, rendant le service vulnérable aux abus, au spam et aux attaques par déni de service applicatif.

7.2.3 Vulnérabilités modérées et faibles (Score < 9)

▲ **VULN-SOC-004** : Exposition excessive d'informations dans les réponses API (identifiants internes, métadonnées non nécessaires).

▲ **VULN-SOC-005** : Journalisation potentielle de données sensibles (identifiants utilisateurs, contenu détaillé des requêtes) dans les logs applicatifs.

7.2.4 Synthèse des vulnérabilités

Table 5 – Synthèse et reclassification des vulnérabilités - Serveur Social

ID	Description	Proba.	Impact	Score	Niveau
VULN-SOC-001	Autorisation insuffisante	4	4	16	Critique
VULN-SOC-002	Validation des entrées absente	3	3	9	Élevé
VULN-SOC-003	Absence de rate limiting	3	3	9	Élevé
VULN-SOC-004	Sur-exposition des données API	2	2	4	Modéré
VULN-SOC-005	Logs contenant des données sensibles	2	2	4	Faible

Table 6 – Matrice de risque avec niveau de criticité

Impact \ Proba.	**1**	**2**	**3**	**4**
1	Faible	Faible	Faible	Moyen
2	Faible	VULN-SOC-004 VULN-SOC-005	Moyen	Moyen
3	Faible	Moyen	VULN-SOC-002 VULN-SOC-003	Élevé
4	Faible	Moyen	Élevé	VULN-SOC-001

7.2.5 Analyse des impacts pour DemPerm

— **Atteinte à l'intégrité des données** : Un défaut d'autorisation (VULN-SOC-001) permet la suppression ou la modification frauduleuse de contenus.

— **Dégradation du service** : L'absence de rate limiting (VULN-SOC-003) facilite le spam et les attaques par déni de service.

— **Perte de confiance des utilisateurs** : La persistance de contenus malveillants et les fuites de données nuisent à la crédibilité de la plateforme.

7.3 Recommandations globales

7.3.1 Corrections prioritaires (immédiates)

1. **Mettre en place des contrôles d'autorisation stricts** (VULN-SOC-001) pour toutes les actions sensibles.
2. **Limiter l'exposition des endpoints** (VULN-SOC-003) via un mécanisme de rate limiting par utilisateur ou par IP.

7.3.2 Améliorations structurelles (court terme)

— **Valider strictement les entrées utilisateur** (VULN-SOC-002) via des schémas et contraintes de type.
— **Réduire les données exposées par les API** (VULN-SOC-004) en utilisant des DTO adaptés.
— **Nettoyer les logs applicatifs** (VULN-SOC-005) pour éviter toute fuite de données sensibles.

7.3.3 Feuille de route de correction

Table 7 – Feuille de route de sécurisation du serveur social

Délai	Actions	Vulnérabilités concernées
Immédiat	Contrôles d'autorisation + rate limiting	VULN-SOC-001, VULN-SOC-003
1-2 sem.	Validation des entrées + réduction des réponses API	VULN-SOC-002, VULN-SOC-004
2-4 sem.	Revue et durcissement des logs	VULN-SOC-005

7.3.4 Métriques de suivi

— **Score de sécurité initial** : Préoccupant (1 vulnérabilité critique, 2 élevées).
— **Objectif court terme** : 0 vulnérabilité critique.
— **Objectif moyen terme** : Réduction significative de la surface d'attaque et durcissement global du service.

8 Serveur de Vote

8.1 Surface d'attaque

8.1.1 Environnement technique

— **Framework** : Django avec Django REST Framework
— **Base de données** : Neo4j (base de données orientée graphe)
— **Authentification** : Firebase (via tokens JWT vérifiés côté serveur)
— **Environnement d'exécution** : Conteneur Docker avec Python 3.10
— **Serveur applicatif** : Serveur de développement Django (`manage.py runserver`)

8.1.2 Composants critiques analysés

— **Logique métier** : Service de validation des votes (`vote_validation_service.py`)
— **Configuration** : `settings.py`, `docker-compose.yml`, `Dockerfile`
— **Authentification** : `security_config.py` (vérification des tokens Firebase)
— **Points d'accès API** : `api_urls.py`, `vote_controller.py`
— **Gestion des secrets** : Accès à la base de données (`neo4j_config.py`), clés Django et Firebase

8.2 Constats

8.2.1 Vulnérabilités critiques (Score 13-16)

▲ **VULN-VOTE-001** : Identifiants de la base de données Neo4j par défaut et codés en dur dans `docker-compose.yml` et `neo4j_config.py` (`neo4j:password`).

8.2.2 Vulnérabilités élevées (Score 9-12)

▲ **VULN-VOTE-002** : Utilisation du serveur de développement Django en production via la commande `CMD` du `Dockerfile`, qui n'est ni performant, ni sécurisé.

▲ **VULN-VOTE-003** : Absence de contrôle d'accès sur le point d'API de validation forcée (`GET /api/votes/validate/force`), exposant une fonctionnalité lourde à tous les utilisateurs authentifiés et créant un risque de Déni de Service (DoS).

▲ **VULN-VOTE-004** : Configuration de sécurité par défaut permissive dans `settings.py` (`SECURE_SSL_REDIRECT`, `SESSION_COOKIE_SECURE` désactivés), sans configuration HTTPS dans l'environnement Docker.

8.2.3 Vulnérabilités modérées et faibles (Score < 9)

▲ **VULN-VOTE-005** : Clé secrète Django (`SECRET_KEY`) faible et prédictible si la variable d'environnement n'est pas définie.

▲ **VULN-VOTE-006** : Procédures de la base de données Neo4j excessivement permissives (`apoc.*`, `gds.*`), augmentant la surface d'attaque en cas d'injection Cypher.

8.2.4 Synthèse des vulnérabilités

Table 8 – Synthèse et reclassification des vulnérabilités - Serveur de Vote

ID	Description	Proba.	Impact	Score	Niveau
VULN-VOTE-001	Identifiants DB par défaut	4	4	16	Critique
VULN-VOTE-002	Serveur dev en production	4	3	12	Élevé
VULN-VOTE-003	DoS sur l'API de validation	3	3	9	Élevé
VULN-VOTE-004	Configuration HTTPS absente	3	3	9	Élevé
VULN-VOTE-005	`SECRET_KEY` Django faible	3	2	6	Modéré
VULN-VOTE-006	Procédures Neo4j permissives	2	2	4	Faible

Table 9 – Matrice de risque avec niveau de criticité

Impact \ Proba.	1	2	3	4
1	Faible	Faible	Faible	Moyen
2	Faible	VULN-VOTE-006	VULN-VOTE-005	Moyen
3	Faible	Moyen	VULN-VOTE-003 VULN-VOTE-004	VULN-VOTE-002
4	Faible	Moyen	Élevé	VULN-VOTE-001

8.2.5 Analyse des impacts pour DemPerm

— **Prise de contrôle totale des données** : Les identifiants par défaut (VULN-VOTE-001) permettent à un attaquant de lire, modifier ou supprimer l'intégralité de la chaîne de votes.

— **Indisponibilité du service** : L'utilisation du serveur de développement (VULN-VOTE-002) et l'exposition d'une API de validation non restreinte (VULN-VOTE-003) rendent le service vulnérable aux dénis de service, paralysant le processus démocratique.

— **Perte de crédibilité** : La combinaison de ces failles rend le système inutilisable pour un cas d'usage réel et sape la confiance que les utilisateurs pourraient avoir dans la sécurité de la plateforme.

8.3 Recommandations globales

8.3.1 Corrections prioritaires (immédiates)

1. **Changer les identifiants de base de données** (VULN-VOTE-001) : Utiliser des mots de passe robustes et les gérer via des secrets Docker ou des variables d'environnement sécurisées.
2. **Utiliser un serveur WSGI de production** (VULN-VOTE-002) : Remplacer `runserver` par Gunicorn ou uWSGI et le configurer derrière un reverse proxy comme Nginx.
3. **Restreindre l'accès à l'API de validation** (VULN-VOTE-003) : Mettre en place une permission (ex : `IsAdminUser`) pour que seuls les administrateurs puissent appeler ce point d'API.

8.3.2 Améliorations structurelles (court terme)

— **Activer HTTPS** (VULN-VOTE-004) : Mettre en place un reverse proxy (Nginx) pour gérer le SSL/TLS et activer les paramètres de sécurité Django (`SECURE_SSL_REDIRECT`, etc.).

— **Sécuriser la `SECRET_KEY` Django** (VULN-VOTE-005) : S'assurer que la variable d'environnement `DJANGO_SECRET_KEY` est toujours définie en production avec une valeur forte et unique.

— **Réduire les permissions de la base de données** (VULN-VOTE-006) : Configurer des rôles Neo4j avec des permissions limitées pour l'application, au lieu d'utiliser un accès administrateur sans restriction.

8.3.3 Feuille de route de correction

Table 10 – Feuille de route de sécurisation du serveur de vote

Délai	Actions	Vulnérabilités concernées
Immédiat	Changer identifiants DB + Sécuriser `SECRET_KEY`	VULN-VOTE-001, VULN-VOTE-005
1-2 sem.	Remplacer `runserver` par Gunicorn/Nginx + HTTPS	VULN-VOTE-002, VULN-VOTE-004
2-4 sem.	Restreindre l'API de validation + permissions DB	VULN-VOTE-003, VULN-VOTE-006

8.3.4 Métriques de suivi

— **Score de sécurité initial** : Préoccupant (1 vulnérabilité critique, 3 élevées).

— **Objectif court terme** : 0 vulnérabilité critique, 0 vulnérabilité élevée.

— **Objectif moyen terme** : Remédiation de toutes les vulnérabilités identifiées.

9 Protocole

9.1 Rapport d'audit : Analyse des risques du protocole de vote

9.2 Résumé exécutif

9.2.1 Objectif et architecture

La base de code implémente un protocole de vote côté serveur reposant sur une base de données graphe Neo4j. Les votes sont modélisés sous forme de relations `(:User)-[VOTED]->(:User)` avec des propriétés telles que `domain`, `createdAt`, `processed`, `valid`, `current`, `count` et `cycle`.

La logique protocolaire est répartie dans les fichiers suivants :

— `src/serveur/vote/db/repository/vote_repository.py`

— `src/serveur/vote/core/services/vote_validation_service.py`

— `src/serveur/vote/core/services/vote_service.py`

— `src/serveur/vote/core/services/result_service.py`

— `src/serveur/vote/api/vote_controller.py`

9.2.2 Évaluation globale du risque

Niveau de risque global : ÉLEVÉ. Plusieurs vulnérabilités critiques compromettent l'intégrité du vote, la confidentialité des bulletins et la fiabilité du dépouillement. On observe notamment une invalidation aléatoire des votes, l'absence totale de protections cryptographiques, une dépendance à des identifiants internes de la base de données et des incohérences dans le modèle d'état.

9.2.3 Constats critiques majeurs

— Invalidation aléatoire des votes lors de la phase de validation.

— Absence de bulletins cryptographiquement protégés (pas de signature, pas de chiffrement).

— Absence totale de secret du vote (liens directs votant → cible).

— Bugs logiques et fautes de frappe impactant la suppression et le comptage.

— Dépendance aux identifiants internes Neo4j (`elementId`).

— États mutables (`current`, `processed`) favorisant les conditions de course.

9.3 Méthodologie

9.3.1 Périmètre

Audit limité à la couche protocolaire :

— Logique de vote

— Structures de données

— Validation et dépouillement

— Garanties cryptographiques et de consensus

Les interfaces utilisateur, le déploiement et l'infrastructure n'ont pas été audités sauf impact direct sur les garanties du protocole.

9.3.2 Méthodes

— Revue statique du code source

— Traçage des flux logiques entre services et dépôts

— Modélisation des attaques plausibles sur le protocole

9.4 Constats détaillés

9.4.1 Invalidation aléatoire des votes dans le flux de validation

ID	P001
Sévérité	Critique
Composant	Protocole
Impact	Atteinte directe à l'intégrité et au déterminisme du scrutin
Reproduction / Preuves	La fonction `process_daily_votes()` valide environ 80% des votes et rejette aléatoirement les autres après mélange.
Recommandation	Supprimer toute logique aléatoire. La validation doit être déterministe et fondée sur des critères vérifiables. Toute détection de fraude doit être séparée et auditée.

9.4.2 Absence totale de protection cryptographique des bulletins

ID	P002
Sévérité	Critique
Composant	Protocole
Impact	Votes falsifiables, absence d'authenticité et de non-répudiation
Reproduction / Preuves	Les votes sont stockés en clair sans signature, nonce ou chiffrement.
Recommandation	Introduire des bulletins signés cryptographiquement avec nonces et horodatage. Envisager chiffrement homomorphe, signatures aveugles ou mix-nets.

9.4.3 Absence de secret du vote

ID	P003
Sévérité	Critique
Composant	Protocole
Impact	Violation du secret du scrutin, coercition possible
Reproduction / Preuves	Relations directes votant $\rightarrow$ cible stockées en base.
Recommandation	Séparer identité et bulletin via engagements cryptographiques ou protocoles de vote anonymes standard.

9.4.4 Incohérence de propriété (`process` vs `processed`)

ID	P004
Sévérité	Élevé
Composant	Protocole
Impact	Suppression et comptage incorrects
Reproduction / Preuves	Erreur de nom de propriété dans les requêtes Neo4j.
Recommandation	Corriger vers `processed` partout et ajouter des tests unitaires.

9.4.5 Dépendance aux identifiants internes Neo4j (`elementId`)

ID	P005
Sévérité	Élevé
Composant	Protocole
Impact	Fragilité, non-reproductibilité, dépendance moteur
Reproduction / Preuves	Utilisation de `elementId()` comme identifiant métier.
Recommandation	Introduire des UUID stables stockés comme propriétés explicites.

9.4.6 Complexité et incohérences potentielles du recalcul des scores

ID	P006
Sévérité	Élevé
Composant	Protocole
Impact	Résultats incohérents, conditions de course
Reproduction / Preuves	Recalculs GDS non atomiques avec mutations en place.
Recommandation	Utiliser des snapshots immuables et promouvoir les résultats de manière atomique.

9.4.7 Absence de journaux d'audit infalsifiables

ID	P007
Sévérité	Moyen
Composant	Protocole
Impact	Audit post-élection non fiable
Reproduction / Preuves	Statistiques stockées sans signature ni chaînage.
Recommandation	Mettre en place un journal append-only signé (Merkle, hash chain).

9.4.8 Absence de protection contre les attaques Sybil

ID	P008
Sévérité	Moyen
Composant	Protocole
Impact	Manipulation du scrutin par création massive de comptes
Reproduction / Preuves	Aucune garantie d'unicité des votants.
Recommandation	Introduire un registre de votants vérifiés ou des identités uniques.

9.4.9 Conditions de course liées aux flags `current/processed`

ID	P009
Sévérité	Moyen
Composant	Protocole
Impact	Votes dupliqués ou perdus
Reproduction / Preuves	Transitions d'état réparties sur plusieurs transactions.
Recommandation	Imposer des transitions atomiques et des contraintes d'unicité.

9.4.10 Contrôles métier insuffisants côté serveur

ID	P010
Sévérité	Faible
Composant	Protocole
Impact	Votes hors période ou non autorisés
Reproduction / Preuves	Peu de vérifications sur les fenêtres de vote et l'éligibilité.
Recommandation	Forcer toutes les règles métier dans la couche service.

9.5 Matrice de risques

— P001–P003 : **Critique**

— P004–P006 : **Élevé**

— P007–P009 : **Moyen**

— P010 : **Faible**

9.6 Recommandations spécifiques au protocole

— Supprimer toute invalidation aléatoire.

— Introduire des bulletins cryptographiquement signés.

— Garantir le secret du vote par des mécanismes éprouvés (mix-nets, chiffrement homomorphe).

— Utiliser des UUID stables.

— Rendre le dépouillement atomique et déterministe.

— Ajouter des journaux d'audit infalsifiables.

— Mettre en place un registre de votants robuste.

— Tester les invariants du protocole sous concurrence.

9.7 Conclusion

En l'état, le protocole de vote n'est **pas adapté à un usage électoral réel**. La combinaison de votes traçables, d'absence de cryptographie, d'invalidation aléatoire et de défauts d'implémentation expose le système à des manipulations graves affectant l'intégrité, la confidentialité et la vérifiabilité du scrutin.

Actions prioritaires immédiates :

1. Supprimer l'invalidation aléatoire.
2. Cesser le stockage direct des bulletins en clair.
3. Corriger les incohérences d'état et d'identifiants.
4. Introduire des preuves cryptographiques et des journaux infalsifiables.