Rapport d'analyse de sécurité du *Serveur Social*

Analyse fonctionnelle et technique

Projet : Serveur Social

Auteur :

AOUDIA Lilia

Date : 3 novembre 2025

Ce document présente une analyse de la sécurité du projet de « Serveur Social » couvrant la sécurité fonctionnelle (usages, modération, protection des usagers) et la sécurité technique (architecture, données, résilience).

Table des matières

1 Introduction **2**

2 Analyse de la sécurité fonctionnelle **2**

2.1 Messagerie privée *one-to-one* 2

2.2 Profils ouverts/fermés et abonnements 3

2.3 Discussions thématiques (topics, groupes) et commentaires 3

2.4 Modération et prévention des abus 3

3 Analyse de la sécurité technique **4**

3.1 Séparation des couches et contrôles transverses 4

3.2 Sécurité de la base de données et gestion des données 4

3.3 Contrôle d'accès, authentification et permissions 4

3.4 Résilience, performance et prévention des DoS 5

3.5 Chiffrement des communications et protection des données 5

3.6 Journalisation et supervision de la sécurité 5

4 Recommandations finales **6**

1 Introduction

Ce rapport présente une analyse de la sécurité du projet de serveur social tel que décrit dans le document `Serveur_Social.pdf`. Le projet vise à créer une plateforme sociale où des élus peuvent communiquer à grande échelle et interagir avec les citoyens. Les fonctionnalités prévues incluent notamment la publication de posts pour diffusion large, une messagerie privée *one-to-one*, des fils de discussion thématiques avec sous-thèmes (topics et groupes), la création de profils utilisateurs (avec alias et statut ouvert ou fermé), et un système d'abonnement aux utilisateurs ou aux fils de discussion. L'architecture backend proposée repose sur une API Django modulable, découpée en applications (`users`, `posts`, `comments`, `groups`, etc.) et en couches (modèles de données, services métiers, DTO, etc.).

L'objectif de ce rapport est d'évaluer le projet sous deux angles : sécurité fonctionnelle (protection des interactions sociales entre utilisateurs) et sécurité technique (robustesse de l'architecture backend et des mesures de protection technique). Pour chaque aspect, nous identifions les risques potentiels (harcèlement, spam, abus, attaques techniques, etc.) puis proposons des mesures concrètes d'atténuation.

2 Analyse de la sécurité fonctionnelle

La sécurité fonctionnelle concerne la façon dont la plateforme gère les interactions entre utilisateurs pour prévenir les abus : messagerie privée, profils ouverts/fermés, abonnements, discussions thématiques (topics, groupes) et commentaires. Chaque mécanisme présente des risques et appelle des garde-fous adaptés.

2.1 Messagerie privée *one-to-one*

La messagerie privée est sensible : elle peut servir au harcèlement ciblé, au spam ou au *phishing*. Il est donc crucial d'implémenter des contrôles d'interaction : blocage d'expéditeurs, signalement des abus à la modération. Pour les élus (fort volume entrant), prévoir des filtres additionnels (par ex. accepter les messages de certains utilisateurs uniquement, ou utiliser une boîte secondaire pour le reste).

Sur la confidentialité, privilégier au minimum le chiffrement fort des messages en transit et au repos (côté serveur). Un chiffrement de bout en bout (E2EE) peut être envisagé pour que seuls expéditeur et destinataire lisent le contenu, au prix de contraintes pour la modération. Une option pragmatique est d'offrir un mécanisme de signalement « sécurisé » qui permet, en cas d'abus, de transmettre le contenu aux modérateurs avec consentement/opt-in dédié.

2.2 Profils ouverts/fermés et abonnements

Chaque utilisateur peut créer un profil avec alias et définir une visibilité « ouverte » ou « fermée ». Un profil ouvert expose davantage l'utilisateur (harcèlement public, spam, scraping) ; un profil fermé réduit l'exposition. Mesures recommandées : paramètres de confidentialité granulaires (qui peut voir/contacter), approbation des abonnés sur profils fermés, blocage/sourdine d'abonnés indésirables, et vérification d'identité pour distinguer les comptes officiels d'élus (badge vérifié).

Les abonnements et notifications doivent rester maîtrisables par l'utilisateur (fréquence, opt-out facile) et résilients aux abus (CAPTCHA à l'inscription, vérification de l'identité).

2.3 Discussions thématiques (topics, groupes) et commentaires

Les threads publics sont exposés au débordement (harcèlement collectif, trolls, désinformation, spam/URLs malveillantes). Il faut une modération efficace : charte d'utilisation claire, outils de signalement, intervention humaine (suppression/masquage, avertissements, bannissements), et filtres automatiques de premier niveau (anti-spam, liste de mots interdits). Limiter la fréquence de publication, exiger un minimum d'ancienneté avant la création de nouveaux groupes ? et traiter les liens partagés (par ex. `mettre un flag` pour les nouveaux membres) permettent de préserver la qualité des échanges.

2.4 Modération et prévention des abus

La modération est le fil conducteur de la sécurité fonctionnelle. Recommandations clés :

— **Signalement & blocage** : interface simple de report, tri et délais de traitement raisonnables ; blocage utilisateur côté cible.

— **Équipe humaine & surveillance** : modérateurs identifiés, droits d'action, gradation des mesures (suppression, suspension, bannissement).

— **Politiques & sanctions** : charte claire, échelle de sanctions cohérente, information et droit de recours de l'utilisateur.

— **Anti-spam/faux comptes** : CAPTCHA, check d'identité notamment via la mairie ? , système de réputation/progression pour nouveaux comptes.

3 Analyse de la sécurité technique

D'après la documentation du projet, le backend Django est structuré en modules/applications (`comments`, `posts`, `friends`, `groups`, `subscriptions`, `tags`, `topics`, `users`) et en couches distinctes : `social_domains` (modèles), `social_services` (logique métier), `social_dto` (DTO), `common` (utils/permissions), `social_db` (DB), `social_api` (exposition API). Cette séparation est favorable à la maintenabilité et à l'implémentation de contrôles transverses de sécurité.

3.1 Séparation des couches et contrôles transverses

Centraliser les vérifications d'autorisations dans des services communs et des classes de permissions réduit les risques d'oubli au niveau des vues. Les DTO permettent de ne pas exposer les objets Django bruts : ne retourner que les champs strictement nécessaires (principe du moindre privilège). Tester systématiquement les chemins critiques (accès à une ressource d'autrui, élévation de privilèges).

3.2 Sécurité de la base de données et gestion des données

Mesures essentielles :

— **Durcissement d'accès** : DB non accessible depuis Internet ; comptes SQL à privilèges minimaux ; secrets en variables d'environnement.

— **Chiffrement des données sensibles** : mots de passe hachés (PBKDF2 [solide, par défaut dans Django]/Argon2 [top actuel : mémoire-dur, résistant aux GPU]) ; chiffrement disque ; chiffrement applicatif pour données personnelles/messages selon besoin.

— **Backups & intégrité** : sauvegardes régulières chiffrées, test de restauration ; contraintes d'intégrité référentielle cohérentes.

— **Prévention des fuites** : réponses API en liste blanche (whitelisting) des champs ; revue des schémas ; logs d'accès DB utiles et surveillés.

3.3 Contrôle d'accès, authentification et permissions

Pour une API REST, **nous recommandons d'utiliser des jetons (JWT/OAuth2 Bearer) avec expiration courte, rotation/renouvellement, et TLS**. Implémenter un modèle de rôles (citoyen, élu, modérateur/admin) et appliquer le principe du moindre privilège à la consultation et la modification. Doubler les vérifications : au niveau vues (contrôle global (qui peut appeler cette route ?) et au niveau services métiers (contrôle logique (qui peut faire cette action spécifique ?).

3.4 Résilience, performance et prévention des DoS

— **Performance** : limiter la taille des requêtes et des fichiers.

— **Temps réel** : si des connexions persistantes sont utilisées, appliquer un délai d'expiration simple pour les connexions inactives et limiter le nombre de connexions simultanées par utilisateur ou par IP.

— **Rate limiting** : mettre en place des seuils de requêtes raisonnables par IP (plus stricts sur la route de connexion), et définir une taille maximale de réponse côté API.

— **Monitoring perf** : suivre des indicateurs simples (latence moyenne, taux d'erreurs, charge serveur) et prévoir un comportement dégradé clair en cas de surcharge (par exemple renvoyer une erreur 503 plutôt que bloquer tout le service).

3.5 Chiffrement des communications et protection des données

— **TLS partout** : HTTPS obligatoire ; WebSockets en `wss://` ; redirection `HTTP`→`HTTPS`.

— **Stockage sécurisé** : fichiers utilisateurs non publics ; noms non devinables ; chiffrement applicatif si documents sensibles.

— **E2EE** : confidentialité maximale pour la messagerie privée (au prix de contraintes de modération).

— **Gestion des clés** : rotation planifiée ; durées de vie courtes pour tokens.

— **2FA** : activer l'authentification à deux facteurs pour comptes sensibles (élus, administrateurs).

3.6 Journalisation et supervision de la sécurité

— **Logs d'activité** : connexions (succès/échec), changements de mots de passe, créations de comptes, actions d'administration (horodatage, identifiant, IP).

— **Logs d'anomalies** : exceptions et erreurs serveur corrélées ; alertes si pics inhabituels (ex. erreurs 500).

— **Conservation** : stockage sécurisé (hors web), export vers SIEM, sauvegardes régulières, durées de rétention conformes.

— **Surveillance** : règles d'alerte (SIEM) sur *bruteforce*, DDoS, connexions suspectes (géographie inhabituelle), répétitions anormales d'appels API.

— **Mises à jour et tests** : patch management (Django/bibliothèques/OS), scans de vulnérabilités, tests d'intrusion ponctuels, tests unitaires de sécurité.

4 Recommandations finales

Sécurité fonctionnelle (usages et interactions)

- **Protection contre le harcèlement** : blocage utilisateur et signalement de contenu, traitement rapide par modération.
- **Vie privée** : paramètres fins ; profils fermés avec approbation ; contrôle de « qui peut me contacter ».
- **Modération proactive** : charte claire ; filtres automatiques d'appoint ; outils d'admin (suppression/bannissement).
- **Anti-spam/faux comptes** : CAPTCHA, double opt-in, réputation/progression, détection IPs massives.
- **Comptes officiels** : mécanisme de vérification/badge pour élus et personnalités publiques.

Sécurité technique (architecture et données)

- **AuthN/AuthZ strictes** : tokens JWT/OAuth2, HTTPS, permissions à tous les niveaux, 2FA pour comptes sensibles.
- **Chiffrement généralisé** : TLS, hachage robuste (PBKDF2/Argon2), chiffrement applicatif pour données sensibles/E2EE si requis.
- **Base de données protégée** : accès restreint, secrets hors code, intégrité/réplication/sauvegardes chiffrées.
- **Résilience & DoS** : rate limiting, scalabilité horizontale, quotas par utilisateur.
- **Journalisation & monitoring** : logs complets, SIEM, alertes, politique de patchs et tests sécurité continus.