Modèle d'authentification

Modèle d'authentification

> *Dernière mise à jour : 2026-05-10*

Mécanisme

- **Mots de passe** : hashés `argon2id` (lib `argon2`)
- **Sessions** : token SHA256 stocké en mémoire (`Map<token, {userId, expiresAt}>`)
- **Cookie** : `session=<token>`, HTTP-only, SameSite=Lax, Secure si HTTPS, durée 7 jours
- **Pas de JWT** : sessions stateful en RAM

Flow

Register

- `POST /api/auth/register { username, email, password }`
- Validation Zod : username 3-30 chars `[A-Za-z0-9_]`, password ≥8 chars, email format
- Hash argon2id du password
- Insertion `users` avec `role='pending'`
- Création session immédiate → cookie 7j
- Notification admin SMTP (si `SMTP_HOST` configuré)

Login

- `POST /api/auth/login { username, password }`
- Rate-limit : **5 tentatives par IP** avant **15 min de cooldown** (en mémoire)
- `verify(hash, input)` argon2

- Si OK : nouveau token aléatoire (`crypto.randomBytes(32).toString('hex')`)
- Stockage `Map.set(token, {userId, expiresAt})`
- Set-Cookie session=`<token>` HttpOnly Secure SameSite=Lax Max-Age=604800

Vérification (middleware)

- `requireAuth` : lit `c.req.cookie('session')`, vérifie `Map.get(token)`, populate `c.set('user', {...})`
- `requireConfirmed` : 403 si `role === 'pending'`
- `requireAdmin` : 403 si `role !== 'admin'`
- `requireCanAddGames` : 403 si `canAddGames === false` (admins bypass)

Logout

- `POST /api/auth/logout` : `Map.delete(token)` + clear cookie

Sessions in-memory : conséquences

- ✅ Pas de table SQL session, pas de Redis : simple
- ✅ Pas de problème de réplication (single instance)
- ⚠️ **Restart container = toutes les sessions perdues** — tout le monde doit se reloguer
- ⚠️ Pas de scaling horizontal (state n'est pas partagé)

C'est un compromis assumé pour un projet single-instance. Si un jour besoin de scaling :

- Migrer sessions vers SQLite (table `sessions`) ou Redis
- Garder le même format token (compat backward)

Email confirmation

Pas de magic-link auto. Le token de confirmation existe mais le user reste en `pending` tant qu'un admin ne le confirme manuellement (`/admin` → Users → Confirmer).

Pourquoi : on filtre les bots / curieux qui découvrent l'URL publique. Limite 100% les comptes non sollicités.

Reset password

- Bouton `/admin` → "Renvoyer mail de reset" → `POST /api/admin/send-password-reset/:userId`
- Génère token reset (one-shot) → mail avec lien `/reset-password?token=...`
- `/reset-password` : nouveau password → hash argon2 → update DB → token consommé

⚠ **Nécessite SMTP configuré**. Sans `SMTP_HOST`, le reset par mail est silencieusement skippé. Fallback : reset manuel via SQL (cf. `user/depannage/reset-mot-de-passe.md`).

Premier admin (boot)

Au tout premier démarrage, l'app vérifie si la table `users` est vide. Si oui :

- Crée un admin avec `username = FIRST_ADMIN_USERNAME`, `password = hash(FIRST_ADMIN_PASSWORD)`, `role = 'admin'`, `canAddGames = true`

Au boot suivant, si l'admin existe et que `FIRST_ADMIN_PASSWORD` a changé : le password n'est **pas** réinitialisé (pour éviter de l'écraser à chaque restart). Pour forcer le reset, supprimer la ligne SQL puis restart.

Pas de 2FA

Pas implémenté. Si tu veux ajouter :

- TOTP (Time-based One-Time Password) via `otpauth` lib
- Stocker le secret TOTP chiffré dans `users` table
- Step supplémentaire après le login password
- Probablement overkill pour un usage personnel

Pas de SSO / OAuth

Pas d'intégration Google / GitHub / etc. Tu pourrais ajouter mais c'est un trou de complexité pour peu de bénéfice (pas de partage de comptes avec d'autres apps).

Revision #1
Created 2026-05-10 15:20:18 UTC by thymon
Updated 2026-05-10 15:20:18 UTC by thymon