Authentification NeuralES (Web + API)

Ce document explique le fonctionnement complet de l'authentification entre le front (neurales-web) et le backend (FastAPI), en utilisant un access token JWT en memoire cote front et un refresh token JWT stocke dans un cookie HttpOnly.

Objectif

- Minimiser l'exposition du token d'acces (pas de localStorage).
- Permettre un renouvellement automatique via un refresh token HttpOnly.
- Simplifier l'utilisation cote front (Axios ajoute le Bearer automatiquement).

Vue d'ensemble

1. L'utilisateur soumet le formulaire de connexion.
2. Le backend valide les identifiants.
3. Le backend renvoie un access token (JSON) et depose un refresh token (cookie HttpOnly).
4. Le front stocke l'access token en memoire et l'ajoute dans le header Authorization.
5. Au demarrage, le front appelle /auth/refresh pour regenerer un access token.
6. En logout, le front appelle /auth/logout et le backend supprime le cookie refresh.

Endpoints utilises

- POST /auth/login : { email, password } -> { access_token, token_type } + Set-Cookie refresh_token.
- POST /auth/refresh : cookie refresh_token -> { access_token, token_type } + rotation cookie.
- POST /auth/logout : supprime le cookie refresh_token.
- GET /auth/me : Authorization: Bearer -> profil utilisateur.

Cote front (neurales-web)

Le store Pinia gere accessToken en memoire, login(), refresh(), initialize(), logout() et fetchMe(). Axios utilise withCredentials: true et ajoute Authorization si accessToken est present.

Cote backend (FastAPI)

/auth/login valide l'admin, genere access + refresh, et pose le cookie HttpOnly. /auth/refresh relit le cookie, valide le JWT (type=refresh), regenere access + refresh et repose le cookie. /auth/logout supprime le cookie.

Cookies et CORS (important)

Pour le cross-origin : withCredentials cote front, allow_credentials cote backend, allow_origins specifique (pas '*'). En cross-site, SameSite=None et Secure=true.

Sequence simplifiee

1) LOGIN : Front -> POST /auth/login, Backend -> Set-Cookie refresh + JSON access_token, Front -> GET /auth/me.
2) REFRESH : Front -> POST /auth/refresh (cookie), Backend -> Set-Cookie refresh + JSON access_token, Front -> GET /auth/me.
3) LOGOUT : Front -> POST /auth/logout, Backend -> supprime le cookie, Front -> purge accessToken + user.

Schema sequence (texte)

Client(Web) -> POST /auth/login { email, password }
API -> Set-Cookie: refresh_token=...; HttpOnly; SameSite=...; Secure?
API -> 200 { access_token, token_type }
Client -> GET /auth/me (Authorization: Bearer access_token)
API -> 200 { user }

Client (au demarrage) -> POST /auth/refresh (cookie refresh)
API -> Set-Cookie: refresh_token=... (rotation)
API -> 200 { access_token, token_type }
Client -> GET /auth/me (Authorization: Bearer access_token)

Client -> POST /auth/logout
API -> Set-Cookie: refresh_token=; Max-Age=0 (suppression)
Client -> purge access_token + user

Details techniques (exemples)

Login :
POST /auth/login
Content-Type: application/json
{ "email": "admin@neurales.com", "password": "admin123" }
Reponse :
Set-Cookie: refresh_token=; HttpOnly; Path=/; SameSite=Lax
{ "access_token": "", "token_type": "bearer" }

Refresh :
POST /auth/refresh
Cookie: refresh_token=
Reponse :
Set-Cookie: refresh_token=; HttpOnly; Path=/; SameSite=Lax
{ "access_token": "", "token_type": "bearer" }

Me :
GET /auth/me
Authorization: Bearer
Reponse :
{ "user_id": 1, "prenom": "Admin", "nom": "NeuralES", "email": "admin@neurales.com", "role": "admin"

}

Logout :
POST /auth/logout
Reponse :
Set-Cookie: refresh_token=; Max-Age=0; Path=/
{ "ok": true }

Notes

- L'access token n'est jamais sauvegarde sur disque.
- Le refresh token est inaccessible au JS (HttpOnly).
- Le WebSocket /eeg/stream doit accepter l'auth par cookie si besoin.