ft_transcendence

Sommaire

Répartition des tâches

Rôle :

- **Maxime** ➡️ Lead – User/Git Management
- **Lylou** ➡️ DevOps Backbone
- **Jeanne** ➡️ Front Backbone
- **Mehdi** ➡️ Cybersecurity
- **Abdul** ➡️ Gameplay & UX

Répartition des modules :

Modules	Lead	Support
User Management	Maxime	Mehdi
DevOps	Lylou	Maxime
Accessibility	Jeanne	Abdul
Cybersecurity	Mehdi	Maxime
Gameplay & UX	Abdul	Jeanne
Web	Jeanne	Lylou
AI/Algo	Mehdi	Abdul
Graphics	Abdul	Jeanne
Server-Side Pong	Lylou	Maxime

Roadmap

🗓️ Semaine 1 – Reprise & Environnement

🎯 Stabiliser le dépôt et les environnements Docker.
🧩 Objectifs :
Nettoyage du repo & branches Git (Maxime)
Mise en place Docker complet (Lylou)
Base Front + Routing (Jeanne)
Audit sécurité initial (Mehdi)
Refactor WebSocket client (Abdul)
✅ Livrable : Projet propre, compilable, environnement stable

🗓️ Semaine 2 – Authentification & Utilisateurs

🎯 Connecter front ↔ back avec gestion des comptes.
🧩 Objectifs :
Login / Register / Logout (Maxime)
Microservices auth et user (Lylou)
Sécurité JWT / 2FA (Mehdi)
Interfaces Login / Profil (Jeanne)
Intégration WS côté user (Abdul)
✅ Livrable : Auth complète, profils utilisateurs persistants

🗓️ Semaine 3 – Chat & Interactions

🎯 Permettre la communication entre utilisateurs.
🧩 Objectifs :
Backend Chat & invitations (Maxime)
Gestion sockets & persistance (Lylou)
UI Chat & notifications (Jeanne)
Sécurité messages (Mehdi)
Animations notifications (Abdul)
✅ Livrable : Chat fonctionnel (DM, rooms, invitations)

🗓️ Semaine 4 – Tournois, Matchmaking & IA

🎯 Ajouter les bonus majeurs.
🧩 Objectifs :
Backend tournois (Maxime)
Matchmaking & multirooms (Lylou)
IA adaptative (Mehdi)
UI brackets & tournois (Jeanne)
Intégration IA côté client (Abdul)
✅ Livrable : Tournois, IA et matchmaking opérationnels

🗓️ Semaine 5 – Sécurité & Monitoring

🎯 Renforcer la fiabilité et l'observabilité.
🧩 Objectifs :
2FA, OAuth42, audit complet (Mehdi)
Monitoring Grafana / Prometheus (Lylou)
Gestion des erreurs & logs (Maxime)
Accessibilité & polish UI (Jeanne)
Effets graphiques & finition (Abdul)
✅ Livrable : Projet stable, sécurisé, bonus validés

🗓️ Semaine 6 – Finalisation & Présentation

🎯 Préparer la version finale et la démonstration.
🧩 Objectifs :
Merge final & release v1.0 (Maxime)
CI/CD finale & docker-compose prod (Lylou)
Documentation UI & accessibilité (Jeanne)
Tests sécurité & conformité (Mehdi)
Démo finale & polish gameplay (Abdul)
✅ Livrable : Version stable + documentation + démo prête

Détails du travail

(Par personne et par tâche)

🎯 Objectif global

- **Un `docker-compose` unique** pour le dev (hot-reload), compatible prod (build multi-stage).
- **Reverse-proxy Nginx** (HTTP/2 + WebSocket) devant tous les services.
- **Réseaux séparés** (public ↔ interne), **volumes** pour données et artefacts.
- **Healthchecks** + dépendances explicites.
- **Observabilité** (Prometheus + Grafana).
- **Makefile** ergonomique.

🧱 Structure cible

```
/docker
  /nginx
    default.conf
  /prometheus
    prometheus.yml
  /grafana
    provisioning/...
/services
  /frontend  (SPA TypeScript)
  /auth      (Fastify/Node)
  /user      (Fastify/Node)
  /chat      (WS server)
  /game      (WS server)
  /matchmaking (Node worker)
.env
docker-compose.yml
Makefile
```

> Base de données : le sujet listait **SQLite**. On part donc sur **SQLite** par service avec **volume monté** (simple et conforme). On pourra garder un profil Postgres optionnel plus tard.

🧩 `docker-compose.yml` (squelette solide)

```
version: "3.9"

name: transcendence
x-common-env: &common_env
  NODE_ENV: development
  TZ: Europe/Paris

networks:
  public:
  internal:

volumes:
  v_front_build:
  v_auth_db:
  v_user_db:
  v_chat_db:
  v_game_db:
  v_match_db:

services:
  gateway:
    image: nginx:1.27-alpine
    depends_on:
      - frontend
      - auth
      - user
      - chat
      - game
    volumes:
      - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf:ro
      - v_front_build:/usr/share/nginx/html:ro
    ports:
      - "80:80"
    networks: [public, internal]
    healthcheck:
      test: ["CMD-SHELL", "wget -qO- http://localhost/health || exit 1"]
      interval: 10s
      timeout: 2s
      retries: 6
```

```
frontend:
  build:
    context: ./services/frontend
    target: dev
  environment:
    <<: *common_env
    VITE_API_BASE: http://gateway
  volumes:
    - ./services/frontend:/app
    - v_front_build:/app/dist
  command: ["npm","run","dev"] # ou build → artefacts dans v_front_build
  networks: [internal]
  healthcheck:
    test: ["CMD","node","-e","process.exit(0)"]

auth:
  build:
    context: ./services/auth
    target: dev
  environment:
    <<: *common_env
    PORT: "3001"
    SQLITE_FILE: "/data/auth.sqlite"
    JWT_SECRET: "change-me"
  volumes:
    - ./services/auth:/app
    - v_auth_db:/data
  command: ["npm","run","dev"]
  networks: [internal]
  healthcheck:
    test: ["CMD","curl","-fsS","http://localhost:3001/health"]
    interval: 10s
    timeout: 2s
    retries: 6

user:
  build:
    context: ./services/user
    target: dev
  environment:
    <<: *common_env
    PORT: "3002"
    SQLITE_FILE: "/data/user.sqlite"
  volumes:
    - ./services/user:/app
    - v_user_db:/data
  command: ["npm","run","dev"]
  networks: [internal]
```

```
chat:
  build:
    context: ./services/chat
    target: dev
  environment:
    <<: *common_env
    PORT: "3003"
    SQLITE_FILE: "/data/chat.sqlite"
  volumes:
    - ./services/chat:/app
    - v_chat_db:/data
  command: ["npm","run","dev"]
  networks: [internal]

game:
  build:
    context: ./services/game
    target: dev
  environment:
    <<: *common_env
    PORT: "3004"
    SQLITE_FILE: "/data/game.sqlite"
  volumes:
    - ./services/game:/app
    - v_game_db:/data
  command: ["npm","run","dev"]
  networks: [internal]

matchmaking:
  build:
    context: ./services/matchmaking
    target: dev
  environment:
    <<: *common_env
    SQLITE_FILE: "/data/match.sqlite"
  volumes:
    - ./services/matchmaking:/app
    - v_match_db:/data
  command: ["npm","run","worker"]
  networks: [internal]

prometheus:
  image: prom/prometheus
  volumes:
    - ./docker/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro
  networks: [internal]
  ports: ["9090:9090"]
```

```
  grafana:
    image: grafana/grafana
    environment:
      GF_SECURITY_ADMIN_USER: admin
      GF_SECURITY_ADMIN_PASSWORD: admin
    networks: [internal]
    ports: ["3000:3000"]
```

🌐 Nginx (Reverse-proxy + WebSocket)

`docker/nginx/default.conf`

```
server {
  listen 80;
  server_name _;

  # Static SPA (build copié/monté)
  location / {
    root /usr/share/nginx/html;
    try_files $uri /index.html;
  }

  # API / services (proxy interne)
  location /api/auth/ {
    proxy_pass http://auth:3001/;
    include /etc/nginx/proxy_params;
  }
  location /api/user/ { proxy_pass http://user:3002/; include /etc/nginx/proxy_params; }
  location /api/chat/ { proxy_pass http://chat:3003/; include /etc/nginx/proxy_params; }
  location /api/game/ { proxy_pass http://game:3004/; include /etc/nginx/proxy_params; }

  # WebSocket upgrade
  map $http_upgrade $connection_upgrade { default upgrade; '' close; }
  location /ws/ {
    proxy_pass http://chat:3003/ws/; # exemple
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_read_timeout 60s;
  }

  # Health
  location = /health { return 200 "ok\n"; add_header Content-Type text/plain; }
}
```

En prod : utiliser Nginx **devant** ce conteneur (ou Traefik/Caddy) avec **TLS**. En dev, rester en HTTP local.

🐳 Dockerfiles Node (multi-stage)

`services/auth/Dockerfile`

```
# --- base ---
FROM node:20-alpine AS base
WORKDIR /app
COPY package*.json ./

# --- dev (hot reload) ---
FROM base AS dev
RUN npm ci
COPY . .
EXPOSE 3001
CMD ["npm","run","dev"]

# --- prod (build + run) ---
FROM base AS deps
RUN npm ci --omit=dev
FROM node:20-alpine AS runner
WORKDIR /app
ENV NODE_ENV=production
COPY --from=deps /app/node_modules ./node_modules
COPY . .
USER node
EXPOSE 3001
CMD ["node","dist/index.js"]
```

Frontend : même logique, mais **build SPA → copie dans volume** `v_front_build` (ou COPY vers runner et bind par Nginx).

📊 Prometheus minimal

`docker/prometheus/prometheus.yml`

```
global:
  scrape_interval: 10s
scrape_configs:
  - job_name: 'auth'
    static_configs: [{ targets: ['auth:3001'] }]
  - job_name: 'user'
    static_configs: [{ targets: ['user:3002'] }]
  - job_name: 'chat'
    static_configs: [{ targets: ['chat:3003'] }]
  - job_name: 'game'
    static_configs: [{ targets: ['game:3004'] }]
```

Exposez un `/metrics` dans chaque service (prom-client pour Node).

🧰 Makefile ergonomique

```
SHELL := /bin/bash
.ONESHELL:

up:
\tdocker compose up -d --build

down:
\tdocker compose down -v

logs:
\tdocker compose logs -f --tail=200

re:
\tdocker compose down -v
\tdocker compose build --no-cache
\tdocker compose up -d

ps:
\tdocker compose ps

shell-%:
\tdocker compose exec $* sh
```

🔐 Sécurité & hygiène

- **Users non-root** dans les conteneurs Node (`USER node`).
- `.dockerignore` propre (node_modules, dist, .git, etc.).
- **Secrets/env** via `.env` (Jamais committer secrets).
- **Read-only FS** possible en prod (`read_only: true` + `tmpfs` si nécessaire).
- **Limites** de ressources (CPU/mem) optionnelles pour éviter les dérapages.

✅ DoD (Definition of Done)

- `make up` → tous les services healthy, **SPA servie**, **API OK**, **WS OK**.
- Artefacts front présents dans `v_front_build` (ou dans runner).
- Healthchecks passent, **restarts** ne bouclent pas.
- Prometheus & Grafana accessibles et collectent des métriques.
- Doc rapide dans le README : commandes de base, ports, variables env.

🧪 Tests rapides (checklist)

- `curl http://localhost/health` → ok
- `curl http://localhost/api/auth/health` → ok
- WebSocket upgrade via Nginx (connexion depuis front) → ok
- SQLite fichiers persistants dans volumes → ok
- Prometheus `http://localhost:9090/targets` → UP
- Grafana `http://localhost:3000` (admin/admin) → dashboards créés

🎯 Objectif global

- **Auth**: inscription, login, refresh, logout, 2FA, OAuth42 (bonus), gestion des sessions JWT (access + refresh), protection brute-force.
- **User**: profils, avatars, stats de jeu, préférences, recherche utilisateurs, relations (blocklist), endpoints lisibles et rapides.
- **Sécurité**: hashage robuste, validation stricte, rate-limit, CORS, scopes JWT.
- **Opé**: migrations SQLite, tests unitaires + d'intégration, logs + métriques, docs OpenAPI.

🧱 Architecture (Fastify + SQLite par service)

```
/services
  /auth
    src/
      index.ts           # bootstrap Fastify
      routes/*.ts        # endpoints
      domain/*.ts        # services métier (hash, token, 2FA)
      adapters/sqlite.ts  # accès DB
      plugins/*.ts       # cors, rate-limit, openapi, metrics
      schemas/*.ts       # zod/schemas JSON
    prisma/ or migrations/   # migrations SQLite
    Dockerfile
  /user
    src/
      index.ts
      routes/*.ts
      domain/*.ts
      adapters/sqlite.ts
      plugins/*.ts
      schemas/*.ts
    prisma/ or migrations/
    Dockerfile
/shared
  auth-types.ts    # types communs: claims JWT, userId
  errors.ts        # mapping erreurs
```

- **Fastify** (rapide, typé) + **zod** pour valider payloads.
- **prom-client** pour exporter `/metrics` (Prometheus).
- **pino** pour logs structurés (JSON).
- **argon2id** pour hash (ou bcrypt coûteux).

🔐 Service Auth — Fonctions & Endpoints

Données (SQLite)

```
-- auth.db
CREATE TABLE users (
  id           TEXT PRIMARY KEY,              -- uuid
  login        TEXT UNIQUE NOT NULL,          -- "mechard"
  email        TEXT UNIQUE,                   -- optionnel
  pass_hash    TEXT,                          -- null si OAuth-only
  twofa_secret TEXT,                          -- base32, null si inactif
  twofa_enabled INTEGER DEFAULT 0,
  created_at   INTEGER NOT NULL,
  updated_at   INTEGER NOT NULL
);

CREATE TABLE refresh_tokens (
  id           TEXT PRIMARY KEY,              -- uuid
  user_id      TEXT NOT NULL,
  token_hash   TEXT NOT NULL,                 -- hashé en base
  created_at   INTEGER NOT NULL,
  expires_at   INTEGER NOT NULL,
  revoked      INTEGER DEFAULT 0,
  FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE INDEX idx_refresh_user ON refresh_tokens(user_id);
CREATE INDEX idx_refresh_valid ON refresh_tokens(expires_at, revoked);
```

Endpoints (proposition)

- `POST /auth/register` → créer compte (login, email?, password)
- `POST /auth/login` → `accessToken` (JWT court) + `refreshToken` (opaque, long)
- `POST /auth/refresh` → nouveau couple (rotation + invalidation de l'ancien)
- `POST /auth/logout` → révoque refresh côté serveur
- `POST /auth/2fa/setup` → génère secret + otpauth URI (QR) (protégé)
- `POST /auth/2fa/enable` → vérifie code TOTP puis active
- `POST /auth/2fa/verify` → vérifie TOTP lors du login si activé
- `GET /auth/me` → renvoie claims de l'utilisateur courant (via access token)
- `GET /auth/health` → healthcheck (Prometheus scrappeur ok)

JWT Access: 5–15 min.
Refresh: 7–30 jours (en base, rotatifs).
Scopes/claims: `{ sub, login, roles, scopes, iat, exp }`.

Flux login standard

1. `POST /auth/login {login, password}`
2. Si `twofa_enabled == 1`, renvoyer `requires2FA: true` → client envoie `POST /auth/2fa/verify { code }`.
3. Réponse finale: `{ accessToken, refreshToken }`.
4. Client garde `accessToken` en mémoire, `refreshToken` en cookie httpOnly *ou* storage sécurisé (selon contraintes du sujet).

Sécurité

- **Hash**: `argon2id` (ou bcrypt cost ~12).
- **Rate-limit** sur `/auth/login` (ex. 5/min/IP + 5/min/login).
- **Refresh rotation**: chaque refresh invalide l'ancien; vol détecté → révoquer tous pour l'utilisateur.
- **TOTP**: `speakeasy` ou `otplib`, skew 1 pas (30s).
- **CORS** précis via Nginx/gateway.
- **WS Auth**: le client passe le `accessToken` en query `?token=` au handshake; serveur valide.

👤 Service User — Profils & Relations

Données (SQLite)

```
-- user.db
CREATE TABLE profiles (
  user_id      TEXT PRIMARY KEY,              -- same id as auth
  display_name TEXT,
  avatar_url   TEXT,                          -- ou stockage local
  bio          TEXT,
  country      TEXT,
  prefs_json   TEXT,                          -- JSON string
  created_at   INTEGER NOT NULL,
  updated_at   INTEGER NOT NULL
);

CREATE TABLE stats (
  user_id      TEXT PRIMARY KEY,
  games_played INTEGER DEFAULT 0,
  games_won    INTEGER DEFAULT 0,
  elo_rating   INTEGER DEFAULT 1200,
  last_match   INTEGER,
  FOREIGN KEY(user_id) REFERENCES profiles(user_id)
);

CREATE TABLE relations (
  owner_id     TEXT NOT NULL,
  target_id    TEXT NOT NULL,
  kind         TEXT NOT NULL,  -- 'block' | 'friend' (si désiré)
  created_at   INTEGER NOT NULL,
  PRIMARY KEY(owner_id, target_id, kind)
);
```

Endpoints (proposition)

- `GET /user/me` → profil courant (via access token)
- `PUT /user/me` → update profil (displayName, bio, avatar_url, prefs)
- `GET /user/:id` → profil public
- `GET /user/search?q=...` → recherche par login/displayName (limit 20)
- `POST /user/block/:id` / `DELETE /user/block/:id`
- `GET /user/blocked` → liste blockés
- `GET /user/stats/:id` → stats publiques (ou `/user/me/stats`)
- `POST /user/stats/increment` → endpoint interne (ou via event) pour MAJ suite à un match

Règles d'accès

- Les endpoints `/user/*` (sauf lecture publique) exigent `accessToken`.
- Respecter **blocklist** (ex. chat/game ne doivent pas lister un utilisateur bloqué côté UI).

🧪 Validation & Schémas (zod)

- Chaque route a un **schema d'entrée** et un **schema de sortie**.
- Réponses uniformes `{ ok: true, data }` ou `{ ok: false, error: { code, message } }`.
- Erreurs mappées (ex. `AUTH_INVALID`, `AUTH_2FA_REQUIRED`, `USER_NOT_FOUND`).

🔗 Intégration Nginx (gateway)

- `GET/POST /api/auth/*` → `auth:3001`
- `GET/POST /api/user/*` → `user:3002`
- Ajouter headers de sécurité (HSTS, no sniff, etc.) au niveau gateway.

🧰 Exemples d’API (curl)

```
# register
curl -X POST http://localhost/api/auth/register \
  -H 'content-type: application/json' \
  -d '{"login":"mechard","password":"P@ssw0rd42"}'

# login
curl -X POST http://localhost/api/auth/login \
  -H 'content-type: application/json' \
  -d '{"login":"mechard","password":"P@ssw0rd42"}'

# me (access token)
curl http://localhost/api/auth/me \
  -H "authorization: Bearer <ACCESS_TOKEN>"

# update profile
curl -X PUT http://localhost/api/user/me \
  -H "authorization: Bearer <ACCESS_TOKEN>" \
  -H 'content-type: application/json' \
  -d '{"display_name":"Maxime","bio":"Lead & Git"}'
```

📈 Observabilité

- `/metrics` exposé dans **auth** et **user** (ex. `http_requests_total`, `auth_logins_total`, `auth_login_failures_total`, `user_profile_update_total`).
- **Logs**: pino JSON avec `reqId`, route, durée, code HTTP, userId si dispo.

🧯 Sécurité & hygiène

- **Désactiver** X-Powered-By.
- **Limiter** taille de requête (ex. 1–2 Mo).
- **Sanitiser** champs texte (bio) → pas de HTML interprété côté UI.
- **Consistance** des dates: timestamps en **ms** (number) ou ISO.
- **Scopes** JWT (ex. `scopes: ["user.read", "user.write"]`) si besoin de granularité.
- **Mots de passe**: force minimale (longueur, classes), mais pas de logs en clair.

🧪 Tests — Critères d’acceptation

- **Auth**
  - login/register OK, 2FA setup/enable/verify OK
  - refresh rotation OK, ancien refresh invalide
  - rate-limit login fonctionne
- **User**
  - CRUD profil OK, recherche OK (limite + tri)
  - stats MAJ après match (endpoint ou event) OK
  - blocklist appliquée (non listée dans recherche si bloqué)

🧩 DoD (Definition of Done)

- `docker-compose up` → **auth** et **user** healthy, OpenAPI exposé.
- Schémas de validation en place, erreurs standardisées.
- JWT + refresh + 2FA opérationnels.
- Profils & stats accessibles et modifiables avec ACL correcte.
- Métriques Prometheus + logs pino.
- Migrations SQLite reproductibles.
- README des services avec endpoints clés.

🎯 Objectif global

- **WS robuste** : auth JWT au handshake, reconnexion (backoff), heartbeat, backpressure, rate-limit.
- **Sémantique de livraison** : au moins *“at-least-once + idempotence”* (ACK + `msgId`).
- **Multiplexage** sur une seule connexion (topics : `chat:room#<id>`, `game:match#<id>`, `lobby`, `notify`).
- **Persistance** : historique messages/événements durable en SQLite, pagination par curseur, index performants.
- **Offline ready** : rattrapage depuis un `cursor` (ou `sinceId`) à la reconnexion.
- **Observabilité** : métriques Prometheus + logs structurés.

🧱 Archi serveur (Fastify + ws / uWS)

- **Fastify** pour HTTP + upgrade WS ; plugin `@fastify/websocket` (ou uWebSockets.js si besoin de très haut débit).
- **Un seul service “chat”** côté WS pour l’instant, qui route par *topic*. Plus tard on pourra séparer “game WS” si nécessaire.
- **Nginx** en reverse proxy (HTTP/2, upgrade WS) comme déjà prévu.

Handshake (auth)

- Le client inclut `?token=<JWT>` (ou protocole `Sec-WebSocket-Protocol: bearer,<token>` si supporté).
- Le serveur **valide le JWT** (signature + exp) → charge les *claims* (userId, login, scopes).
- En cas d’échec : close code `4401` (unauthorized).

💬 Protocole messages (schéma commun)

```
// Outbound et inbound côté client/serveur (JSON)
{
  "v": 1,
  "topic": "chat:room#42",
  "type": "message.create",        // ex: "state.update", "notify", "pong"…
  "msgId": "01JABC...ULID",        // idempotence client
  "ts": 1730123456789,             // ms
  "payload": { /* typed */ }
}
```

Types usuels (chat)

- `message.create` → création d'un message.
- `message.ack` → accusé de réception (serveur → client).
- `message.received` / `message.read` → accusés de lecture (optionnel).
- `room.join` / `room.leave` → membership live.
- `notify` → invitations, alertes.
- `ping` / `pong` → heartbeat application.

Idempotence : le serveur **doit** ignorer un `msgId` déjà traité (journal d'ACK court-terme en mémoire + unique constraint en base).

🧵 Multiplexage par *topic*

- **Un socket / plusieurs topics.** Le client "s'abonne" logiquement via `room.join` (conserve les droits côté serveur).
- `topic` = `<domain>:<entity>#<id>` (ex. `chat:room#abc123`, `notify:user#uid`).

🛡️ Qualité de service côté socket

- **Heartbeat**: serveur envoie un `ping` toutes 5–10 s ; si pas de `pong` en 15 s → close.
- **Backpressure**: surveiller `ws.bufferedAmount`; si > 5 MB → ralentir / drop non essentiels (ex : “typing”).
- **Rate-limit** (token-bucket par connexion) : p.ex. 30 msg/3s, burst 50. Si dépassement → `429` + cooldown.
- **Taille maximale** message : p.ex. 64 KB.
- **Filtrage HTML** côté affichage (XSS) + encodage strict côté UI.
- **ACL** : vérifier à chaque action que l’utilisateur **a le droit** d’écrire/lire dans le `topic` (membre du room, pas bloqué, etc.).

🗃️ Persistance (SQLite + WAL)

Activer WAL et quelques pragma dev :

PRAGMA journal_mode=WAL;
PRAGMA synchronous=NORMAL;
PRAGMA foreign_keys=ON;

Schéma minimal Chat

```
-- Rooms
CREATE TABLE rooms (
  id          TEXT PRIMARY KEY,       -- ulid/uuid
  kind        TEXT NOT NULL,          -- 'dm' | 'group'
  created_at  INTEGER NOT NULL
);

-- Room membership
CREATE TABLE room_members (
  room_id     TEXT NOT NULL,
  user_id     TEXT NOT NULL,
  role        TEXT NOT NULL DEFAULT 'member', -- 'member'|'admin'
  joined_at   INTEGER NOT NULL,
  PRIMARY KEY (room_id, user_id),
  FOREIGN KEY(room_id) REFERENCES rooms(id)
);

-- Messages (append-only)
CREATE TABLE messages (
  id          TEXT PRIMARY KEY,       -- ulid
  room_id     TEXT NOT NULL,
  author_id   TEXT NOT NULL,
  seq         INTEGER NOT NULL,       -- séquence monotone par room
  content     TEXT NOT NULL,
  created_at  INTEGER NOT NULL,
  edited_at   INTEGER,
  deleted     INTEGER NOT NULL DEFAULT 0,
  FOREIGN KEY(room_id) REFERENCES rooms(id)
);

CREATE UNIQUE INDEX idx_messages_room_seq ON messages(room_id, seq);
CREATE INDEX idx_messages_room_ts ON messages(room_id, created_at);

-- Delivery receipts (optionnel)
CREATE TABLE receipts (
  room_id   TEXT NOT NULL,
  msg_id    TEXT NOT NULL,
  user_id   TEXT NOT NULL,
  kind      TEXT NOT NULL,            -- 'received' | 'read'
  at        INTEGER NOT NULL,
  PRIMARY KEY (room_id, msg_id, user_id, kind)
);
```

Génération `seq` par room

- Utiliser une **table compteur** ou calcul `MAX(seq)+1` **dans une transaction**:

```
BEGIN IMMEDIATE;
  SELECT COALESCE(MAX(seq),0)+1 AS next FROM messages WHERE room_id=?;
  -- insérer le message avec seq=next
COMMIT;
```

Ça garantit l'ordre strict **par room**.

Requêtes courantes

- **Page d'historique** (descendant, pagination par curseur `seq` ou `created_at`) :

```
SELECT * FROM messages
 WHERE room_id = ?
   AND seq < ?
 ORDER BY seq DESC
 LIMIT 50;
```

Rattrapage (offline)

- Le client envoie `room.sync { roomId, sinceSeq }` → retourne les messages `seq > sinceSeq`.
- Alternative : `sinceTs` si tu préfères un curseur temporel.

🔄 Flux serveur (message.create)

1. **Réception** du WS message (valider schéma JSON via zod).
2. **ACL** : l'auteur est-il membre du room ? non bloqué ?
3. **Idempotence** : si `msgId` déjà vu → renvoyer `message.ack` existant.
4. **Transaction** : calcul `seq`, insertion, commit.
5. **Broadcast** aux membres connectés (`topic = chat:room#id`).

6. **ACK** à l'auteur (`message.ack { msgId, persisted: true, seq }`).

   En cas d'échec (quota, ACL, taille) → `error { code, message, ref: msgId }`.

🔔 Notifications & invitations

- Canal dédié `notify:user#<userId>` pour pousser
  - **invites** (match / room),
  - **mentions**,
  - **système** (maintenance, etc.).
- Persistées dans `notifications(user_id, kind, payload_json, created_at, read_at)` pour **historique**.

🔗 Intégration avec Auth / User

- **Auth** : valider le JWT et charger `userId` au handshake.
- **Blocklist** (dans `user` service) : enregistrement local (cache) côté "chat" mis à jour via **webhook** ou **pull** périodique.
- **Scopes** : uniquement publier/lire si scopes autorisent.

📈 Observabilité (Prometheus + logs)

Métriques (ex. via `prom-client`) :

- `ws_connections_current` (gauge)
- `ws_messages_in_total`, `ws_messages_out_total`
- `ws_backpressure_drops_total`
- `chat_messages_persisted_total`

- `chat_room_joins_total`
- `request_duration_ms_bucket` (histogram Fastify)

Logs pino (JSON) : `reqId`, `userId`, `topic`, `type`, `roomId`, `seq`, durée, statut.

🧪 Tests (acceptation)

Unitaires

- Validation schema (zod) / ACL / calcul `seq`.
- Idempotence : même `msgId` → un seul enregistrement.
- Pagination : `LIMIT 50` + curseur fonctionne.

Intégration

- 2 clients → envois simultanés → `seq` strictement croissant.
- Coupure réseau 5 s → reconnexion → rattrapage `sinceSeq` OK.
- Backpressure : injecter 10 MB → pas de crash, drops contrôlés.

E2E

- DM et room group → envoi/lecture, mentions, invites ;
- Blocklist : l'utilisateur bloqué ne reçoit plus l'event, ni mention.

🔧 Snippet Fastify (squelette WS)

```
// chat/ws.ts
import { WebSocket } from 'ws';
import { z } from 'zod';
import type { FastifyInstance } from 'fastify';

const msgSchema = z.object({
  v: z.number().int(),
  topic: z.string(),
  type: z.string(),
  msgId: z.string().min(8),
```

```
  ts: z.number(),
  payload: z.any().optional(),
});

export async function registerWs(server: FastifyInstance) {
  server.get('/ws', { websocket: true }, (conn, req) => {
    const user = validateJwt(req); // throw si invalide
    const ws: WebSocket = conn.socket;

    const state = {
      userId: user.sub,
      topics: new Set<string>(),
    };

    ws.on('message', async (buf) => {
      let msg;
      try { msg = msgSchema.parse(JSON.parse(String(buf))); }
      catch { return ws.send(JSON.stringify(err('bad_request'))); }

      // route par topic
      if (msg.type === 'room.join') {
        if (!canJoin(state.userId, msg.payload.roomId)) return ws.send(err('forbidden'));
        state.topics.add(`chat:room#${msg.payload.roomId}`);
        return ws.send(ok('room.joined', { roomId: msg.payload.roomId }));
      }

      if (msg.type === 'message.create') {
        const roomId = parseRoomId(msg.topic);
        if (!state.topics.has(`chat:room#${roomId}`)) return
ws.send(err('not_in_room',{roomId}));
        const result = await persistMessage({ roomId, authorId: state.userId, content:
msg.payload.text, msgId: msg.msgId });
        broadcast(roomId, {
          v: 1, topic: `chat:room#${roomId}`, type: 'message.created',
          ts: Date.now(), msgId: result.msgId, payload: { id: result.id, seq: result.seq, text:
msg.payload.text, authorId: state.userId }
        });
        ws.send(ok('message.ack', { msgId: msg.msgId, seq: result.seq }));
      }
    });

    ws.on('close', () => {
      // cleanup
    });
  });
}
```

Les fonctions `validateJwt`, `canJoin`, `persistMessage`, `broadcast`, `ok/err` sont à implémenter proprement (service + repo + cache).

🧯 Sécurité & hygiène

- **Sanitisation** stricte des contenus côté affichage (UI) + escape.
- **TTL** configurable pour rétention messages (si besoin) ; archivage éventuel.
- **Quotas** : taille message, nombre de rooms par user, membres/room.
- **Audit** : journaliser les actions sensibles (invites, kicks).
- **DoS** : limiter connexions/concurrence par IP et par user.

✅ Definition of Done (DoD)

- Connexion WS avec auth JWT + heartbeat + reco OK.
- Multiplexage par `topic`, join/leave par room.
- `message.create` persiste, diffuse, ACK avec idempotence.
- Historique paginé (`sinceSeq`/`cursor`) + rattrapage après reco.
- Métriques Prometheus exposées, logs JSON.
- Tests d’intégration validés (ordre, reco, pagination, quotas).

🎯 Objectif global

- **File d'attente** robuste (1v1 au minimum), avec critères : ELO/MMR, région/latence, statut (dispo), blocklist.
- **Algorithme de matching** progressif (fenêtre de tolérance qui s'agrandit dans le temps).
- **Création automatique de room** (game + chat) et **assignation** des joueurs.
- **Multiplexage WS** par *topics* (`game:match#id`, `chat:room#id`, `notify:user#id`) sur **une seule connexion**.
- **Tolérance aux pannes** (reco, timeouts, abandon), **persistance** minimale pour reprendre l'état.
- **Observabilité** (métriques + logs) pour diagnostiquer file/latence/qualité des matches.

🧱 Architecture logique

```
[auth]      [user]      [matchmaking]      [game]      [chat]      [gateway/nginx]
  |           |              |               |           |
JWT/claims  profils/stats  queues, pairing  state server  room msgs  WS/HTTP proxy
                              └─> create room (game+chat), notify players
```

- **Service `matchmaking`** (worker Node) responsable de :
  - Maintenir une **queue en mémoire** + **persistence** (SQLite) pour reprise.
  - Lancer le **pairing loop** (ex. toutes les 250 ms).
  - **Créer la room de jeu** (via `game`) et la room de chat (via `chat`) ou les référencer.
  - **Notifier** les joueurs (`notify:user#<id>`) avec la room choisie, + timeout de confirmation.
- **Service `game`** instancie l'état pour `match#<id>` ; **service `chat`** gère `room#<id>`.

🗃️ Données (SQLite) — `matchmaking.db`

```
CREATE TABLE queue (
  user_id     TEXT PRIMARY KEY,
  elo         INTEGER NOT NULL,
  region      TEXT NOT NULL,         -- 'eu' | 'us' | ...
  enqueued_at INTEGER NOT NULL,      -- ms epoch
  wanted_mode TEXT NOT NULL DEFAULT '1v1',
  status      TEXT NOT NULL DEFAULT 'queued'  -- 'queued' | 'matched' | 'cancelled'
);

CREATE TABLE matches (
  id          TEXT PRIMARY KEY,      -- ulid/uuid
  p1_id       TEXT NOT NULL,
  p2_id       TEXT NOT NULL,
  p1_elo      INTEGER NOT NULL,
  p2_elo      INTEGER NOT NULL,
  region      TEXT NOT NULL,
  created_at  INTEGER NOT NULL,
  state       TEXT NOT NULL DEFAULT 'pending'  -- 'pending'|'active'|'finished'|'cancelled'
);

CREATE TABLE match_rooms (
  match_id    TEXT PRIMARY KEY,
  game_topic  TEXT NOT NULL,         -- 'game:match#<id>'
  chat_topic  TEXT NOT NULL          -- 'chat:room#<id>'
);
```

Option : table `queue_history` pour audit (entrée/sortie de file, raisons).

🧠 Algorithme de matchmaking

Critères

- **Écart ELO** (ΔE): commence serré, s'élargit avec le temps d'attente.
  Exemple :
    - T < 10s → ΔE ≤ 50
    - 10–30s → ΔE ≤ 100
    - 30–60s → ΔE ≤ 200
    - 60s → ΔE ≤ 400

- **Région/latence** : prioriser la même région ou latence estimée < 80 ms.
- **Blocklist** : exclure paires bloquées (via service `user` / cache local).
- **Rematch immédiat** : empêcher de rematcher les mêmes joueurs 3 fois d'affilée (variante anti-smurf/frustration).

Boucle de pairing (pseudo)

```
// every 250ms
const now = Date.now();
const players = getQueueOrderedByEnqueueTime(); // FIFO

for each p in players:
  const window = deltaElo(now - p.enqueued_at); // fonction de temps d'attente
  const candidates = players
    .filter(q => q !== p)
    .filter(q => abs(q.elo - p.elo) <= window)
    .filter(q => regionCompatible(p, q))
    .filter(q => !blocked(p, q))
    .sort(by |elo_diff| asc, enqueue_time asc);

  if (candidates[0]) {
    const q = candidates[0];
    createMatch(p, q);
    removeFromQueue(p, q);
  }
```

Création du match

- `match_id = ulid()`
- `game_topic = 'game:match#' + match_id`
- `chat_topic = 'chat:room#' + ulid()` (ou DM room dédiée)
- Appel au service **game** pour *instancier l'état* (serveur autoritaire).
- Appel au service **chat** pour créer la room (ou réutiliser DM).

Notifier les joueurs via `notify:user#<id>`:

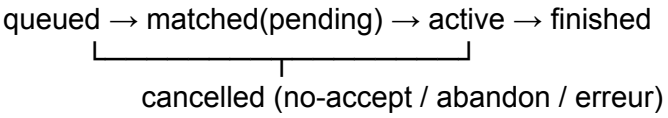

```
{
 "v":1, "topic":"notify:user#<uid>", "type":"match.found",
 "payload":{"matchId":"...", "gameTopic":"game:match#...", "chatTopic":"chat:room#...",
"expiresIn":10000}
}
```


-
- **Timeout de confirmation** (ex. 10 s). Chaque joueur envoie `match.accept` (sur notify ou HTTP).
    - Si deux *accept* → `state = active` et *launch*.
    - Sinon → `cancelled` + *requeue* joueur non fautif.

🧵 Multirooms (une connexion, plusieurs topics)

- Un **unique WebSocket** par client, avec **subscriptions** :
    - `notify:user#<id>` (toujours abonné)
    - `chat:room#<id>` (rejoindre/quitter)
    - `game:match#<id>` (pendant une partie)
- Le client **join**/**leave** explicitement chaque topic (room) pour recevoir les messages ciblés.
- Un **match** ouvre généralement **2 rooms** : `game` et `chat` (room de match).

🔁 États & transitions (match)



- **pending** : en attente d'acceptation des 2 joueurs.
- **active** : partie démarrée (service `game` autoritaire).
- **finished** : résultat enregistré (stats ELO via service `user`).
- **cancelled** : libération de slots, requeue éventuelle.

⏱️ Timeouts & robustesse

- **acceptTimeout** (10–15 s) : sinon cancel.
- **readyTimeout** (5 s) : client doit se connecter à `game:match#id` rapidement.
- **abandon** : si un joueur se déconnecte > N secondes dès le départ → victoire/défaite technique.
- **reco** : si client revient < X secondes → reprendre la room (via `sinceSeq`/`state sync`).

🔐 Sécurité & fair-play

- **ACL** par topic : seuls `p1/p2` peuvent publier/recevoir `game:match#id`.
- **Anti-replay** : `msgId` + *idempotence* côté `game`.
- **Rate-limit** actions (inputs) → 60 Hz max côté client.
- **Anti-ghosting** : pas de double connexion active au même match pour un même user, ou seconde en *spectate* restreint si bonus spectateur prévu.
- **Blocklist** : empêcher un match si l'un a bloqué l'autre.

🔗 APIs (HTTP + WS)

HTTP (matchmaking)

- `POST /mm/enqueue { mode, region? }` → enfile l'utilisateur courant.
- `POST /mm/dequeue` → sort de la file.
- `GET /mm/status` → statut : `queued/matched/none`.
- `POST /mm/accept { matchId }` → accepte le match.
- `POST /mm/decline { matchId }` → refuse le match.

WS (notify)

- `match.found` → serveur → client
- `match.accepted` / `match.cancelled` → serveur → client
- `match.countdown` (optionnel avant start)

🧮 ELO/MMR (simple et efficace)

- Base 1200, K-factor 32 (ou 24), standard Elo.

Stockage dans `user.stats` (`elo_rating`), MAJ en fin de match :

Ea = 1 / (1 + 10^((Rb - Ra)/400))
Ra' = Ra + K * (Sa - Ea) // Sa = 1 win, 0.5 draw, 0 loss

-
- **Protection Newbie** : bloquer pertes Elo sur les 3 premières parties, ou K plus élévé au début.
- **Anti-smurf** : seuil minimal d'écart Elo selon historique; drapeau si perf anormale.

📈 Observabilité (Prometheus + logs)

Métriques côté matchmaking

- `mm_queue_size` (gauge)
- `mm_match_created_total`
- `mm_match_cancelled_total` (by reason: no_accept, disconnect, error)
- `mm_time_to_match_seconds` (histogram)
- `mm_elo_delta_bucket` (histogram)
- `mm_accept_rate` (ratio)

Logs (pino JSON)

- `userId`, `action`, `matchId`, `elo`, `region`, `reason`, durées.

🧪 Tests d’acceptation

Unitaires

- Fenêtre ΔE croissante avec le temps.
- Exclusion blocklist.
- Pairing FIFO équitable.

Intégration

- 100 joueurs en file → temps moyen de match mesurable (< N s).
- Pannes réseau : cancel non-accept, requeue propre.

E2E

- Enqueue → match.found → accept → join rooms (`game`,`chat`) → finish → stats MAJ.
- Decline → requeue logique.
- Abandon → victoire technique + MAJ ELO.

✅ Definition of Done (DoD)

- Enqueue/dequeue/accept/decline fonctionnels et **idempotents**.
- Création de match + topics (`game:match#id`, `chat:room#id`) + notifications OK.
- Multirooms opérationnels sur **une seule connexion WS**.
- Timeouts/annulations gérés proprement.
- Métriques Prometheus + logs en place.
- MAJ ELO/stats en fin de match via service `user`.
- Documentation concise (flux + endpoints + schémas).

🎯 Objectif global

- Un **scrape unique** Prometheus qui collecte tous les services (`auth`, `user`, `chat`, `game`, `matchmaking`, `gateway/nginx`).
- Des **métriques applicatives** exposées par chaque service (`/metrics`), + métriques système basiques (CPU/RAM du conteneur).
- **Dashboards Grafana** par domaine (API, WS, MM, DB) + un **overview**.
- **Alertes** minimales mais utiles (erreurs 5xx, WS down, queue matchmaking gonflée, temps de réponse > SLO).

🧱 Mise en place (stack Docker)

Prometheus (compose)

```
prometheus:
  image: prom/prometheus
  command:
    - --config.file=/etc/prometheus/prometheus.yml
    - --storage.tsdb.retention.time=15d
  volumes:
    - ./docker/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml:ro
  networks: [internal]
  ports: ["9090:9090"]
```

Grafana (compose)

```
grafana:
  image: grafana/grafana
  environment:
    GF_SECURITY_ADMIN_USER: admin
    GF_SECURITY_ADMIN_PASSWORD: admin
  volumes:
    - ./docker/grafana/provisioning:/etc/grafana/provisioning
  networks: [internal]
  ports: ["3000:3000"]
```

> Option : provisionner automatiquement **datasource** + **dashboards** via fichiers YAML/JSON (voir plus bas).

📡 Scrape Prometheus

`docker/prometheus/prometheus.yml`

```
global:
  scrape_interval: 10s
  scrape_timeout: 5s
  evaluation_interval: 10s

scrape_configs:
  - job_name: 'auth'
    static_configs: [{ targets: ['auth:3001'] }]
    metrics_path: /metrics

  - job_name: 'user'
    static_configs: [{ targets: ['user:3002'] }]

  - job_name: 'chat'
    static_configs: [{ targets: ['chat:3003'] }]

  - job_name: 'game'
    static_configs: [{ targets: ['game:3004'] }]

  - job_name: 'matchmaking'
    static_configs: [{ targets: ['matchmaking:3005'] }]

  - job_name: 'gateway'
    static_configs: [{ targets: ['gateway:80'] }]
    metrics_path: /nginx_status  # via exporter ou stub_status (voir plus bas)
```

Nginx → métriques

Deux options :

- **nginx-prometheus-exporter** (recommandé) :
    - expose `/metrics` en scrappant `stub_status`.
- **stub_status** à la mano (métriques limitées).

Exporter (compose rapide) :

```
nginx-exporter:
  image: nginxinc/nginx-prometheus-exporter:0.11
  command: ["-nginx.scrape-uri=http://gateway/stub_status"]
  networks: [internal]
  ports: ["9113:9113"]
```

Et ajoute:

```
- job_name: 'nginx'
  static_configs: [{ targets: ['nginx-exporter:9113'] }]
```

Dans `gateway` (nginx):

```
location /stub_status {
  stub_status; access_log off; allow 127.0.0.1; deny all;
}
```

(ou autoriser l’exporter via réseau interne).

🧪 Exposer des métriques côté Node/Fastify

Dans chaque service Node:

```
// metrics.ts
import client from 'prom-client';
import fastify from 'fastify';

const app = fastify();
client.collectDefaultMetrics({ prefix: 'svc_' }); // CPU/mem process-level

export const httpDuration = new client.Histogram({
  name: 'http_request_duration_ms',
  help: 'Durée des requêtes HTTP côté service',
  labelNames: ['method', 'route', 'code'],
  buckets: [10, 25, 50, 100, 250, 500, 1000, 2000, 5000],
});

export const httpCount = new client.Counter({
  name: 'http_requests_total',
  help: 'Nombre de requêtes',
  labelNames: ['method', 'route', 'code'],
});

app.addHook('onResponse', async (req, reply) => {
  const route = req.routeOptions?.url || 'unknown';
  httpCount.inc({ method: req.method, route, code: reply.statusCode });
  httpDuration.observe({ method: req.method, route, code: reply.statusCode },
reply.getResponseTime());
});

app.get('/metrics', async (req, reply) => {
  reply.header('Content-Type', client.register.contentType);
  return client.register.metrics();
});
```

Métriques spécifiques par service

auth

- `auth_logins_total{status="ok|fail"}` (Counter)
- `auth_refresh_total{status="ok|revoked|invalid"}` (Counter)
- `auth_2fa_verifications_total{status="ok|fail"}` (Counter)
- `jwt_verify_duration_ms` (Histogram)

user

- `user_profile_update_total` (Counter)
- `user_search_total` + `user_search_duration_ms` (Histogram)

chat / WS

- `ws_connections_current` (Gauge)
- `ws_messages_in_total`, `ws_messages_out_total` (Counter)
- `ws_backpressure_drops_total` (Counter)
- `chat_messages_persisted_total` (Counter)
- `chat_room_joins_total` (Counter)

game

- `game_matches_active` (Gauge)
- `game_tick_duration_ms` (Histogram)
- `game_desync_events_total` (Counter)

matchmaking

- `mm_queue_size` (Gauge)
- `mm_match_created_total` (Counter)
- `mm_time_to_match_seconds` (Histogram) — observe (matchedAt - enqueuedAt)
- `mm_match_cancelled_total{reason="no_accept|disconnect|error"}` (Counter)

Noms : format `snake_case`, unités dans le nom (`_ms`, `_seconds`), **labels stables** (éviter label cardinality explosive).

📊 Dashboards Grafana (proposés)

Overview

- *Uptime par service*, *5xx rate*, *latence p95/p99*, *ws_connections_current*, *mm_queue_size*, *matches_active*.
- *SingleStat/Stat + sparklines*.

API (auth/user)

- Latence `http_request_duration_ms` p50/p95/p99 par route.
- Taux 4xx/5xx, QPS par service.
- Logins success/fail (stacked bar), refresh success/invalid.

WebSocket (chat/game)

- `ws_connections_current` (par service)
- Messages in/out par seconde
- Backpressure drops
- Heartbeat RTT (si mesuré côté client → `ws_rtt_ms`)

Matchmaking

- `mm_queue_size` + `mm_match_created_total` (rate)
- `mm_time_to_match_seconds` (heatmap/histogram + p50/p95)
- Cancelled (par reason)

Nginx / Gateway

- Requêtes/s, active connections, 5xx.
- Upstreams (si exporter avancé).

Provisionne via `./docker/grafana/provisioning/dashboards/*.json` pour les avoir dès `make up`.

🚨 Alertes (Prometheus rules)

Ajouter `rule_files` dans `prometheus.yml`:

```
rule_files:
  - /etc/prometheus/alerts.yml
```

`docker/prometheus/alerts.yml`

```
groups:
- name: transcendence_alerts
  rules:
  - alert: APIHighErrorRate
    expr: sum(rate(http_requests_total{code=~"5.."}[5m]))
          / sum(rate(http_requests_total[5m])) > 0.05
    for: 10m
    labels: {severity: warning}
    annotations:
      summary: "Taux d'erreurs API élevé (>5% depuis 10m)"

  - alert: WSNoConnections
    expr: ws_connections_current == 0
    for: 10m
    labels: {severity: warning}
    annotations:
      summary: "Plus aucune connexion WS (chat/game)"

  - alert: MatchmakingQueueBloated
    expr: mm_queue_size > 50
    for: 5m
    labels: {severity: critical}
    annotations:
      summary: "File d'attente matchmaking > 50"

  - alert: MatchTimeToFirstHighP95
    expr: histogram_quantile(0.95, sum(rate(mm_time_to_match_seconds_bucket[5m])) by
(le)) > 30
    for: 10m
    labels: {severity: warning}
    annotations:
      summary: "P95 temps pour matcher > 30s"
```

> Pour recevoir les alertes, ajoute **Alertmanager** (Slack, Discord, mail). Si pas le temps, garde au moins les règles pour consultation dans l'UI de Prometheus.

🔐 Sécurité & hygiène

- Restreindre Grafana/Prometheus aux **réseaux internes** (pas d'exposition Internet brute).
- Créer un **compte Grafana** non-admin pour l'équipe.
- Sanitize les **labels** (pas d'ID utilisateur en clair).
- Conserver l'historique **15–30 jours** (retention) pour garder un poids raisonnable.

🧰 Provisioning Grafana (optionnel mais pro)

`docker/grafana/provisioning/datasources/ds.yml`

```
apiVersion: 1
datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  url: http://prometheus:9090
  isDefault: true
```

`docker/grafana/provisioning/dashboards/dash.yml`

```
apiVersion: 1
providers:
- name: 'Transcendence'
  folder: 'Transcendence'
  type: file
  options:
    path: /etc/grafana/provisioning/dashboards
```

Dépose tes JSON de dashboards dans ce dossier, Grafana les charge auto.

✅ Definition of Done (DoD)

- `make up` → Prometheus et Grafana **UP**, datasource configurée.
- Chaque service expose `/metrics`; Prometheus les **scrape** (targets **UP**).
- Dashboards **Overview**, **API**, **WS**, **MM** visibles et lisibles.
- 3–4 métriques clés par domaine **renseignées** (pas de graphes vides).
- Règles d'alerte basiques chargées (même sans Alertmanager).
- Un **mini guide README** (URL Grafana, login, dashboards, SLOs).

⚡ SLOs de départ (réalistes)

- **API** p95 < **200 ms**, erreur < **1 %**.
- **WS** uptime > **99 %**, drops backpressure ≈ **0**.
- **Matchmaking** p95 *time-to-match* < **20–30 s** aux heures pleines.

🎯 Objectif global

- **Branches** : `main` (prod), `dev` (préprod), `feature/*`.
- **Tags & versions** : `vX.Y.Z` (SemVer) + images `:vX.Y.Z`, `:latest`.
- **Build** multi-stage + **cache** (Buildx), **scan** vulnérabilités (option Trivy).
- **Registry** : GHCR (`ghcr.io/<org>/<image>`).
- **Déploiement** : `docker compose -f compose.prod.yml up -d --pull always`.
- **Santé** : healthchecks + dépendances, **rollback** 1 commande.
- **Secrets** via GitHub Actions + `.env.prod` sur le serveur (non commité).

🧭 Flux global

1. PR → tests + lint + build images **sans push** (CI rapide).
2. Merge sur `dev` → push images `:dev-<sha>` + déploiement **préprod**.
3. Tag `vX.Y.Z` sur `main` → push images `:vX.Y.Z :latest` + **déploiement prod**.
4. Rollback = re-lancer `compose` avec l'image précédente.

🗂️ Registry & noms d'images

- `ghcr.io/mechard-organization/ft-front`
- `ghcr.io/mechard-organization/ft-auth`
- `ghcr.io/mechard-organization/ft-user`
- `ghcr.io/mechard-organization/ft-chat`
- `ghcr.io/mechard-organization/ft-game`
- `ghcr.io/mechard-organization/ft-matchmaking`
- `ghcr.io/mechard-organization/ft-gateway` (nginx)

⚙️ Workflows GitHub Actions

CI sur PR (lint + tests + build)

`.github/workflows/ci.yml`

```
name: CI
on:
  pull_request:
    branches: [ dev, main ]
jobs:
  test-and-build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-node@v4
        with: { node-version: 20 }

      - name: Install root tools
        run: npm ci || true

      - name: Lint & unit tests (frontend)
        working-directory: services/frontend
        run: npm ci && npm run lint && npm test -- --ci

      - name: Lint & unit tests (auth)
        working-directory: services/auth
        run: npm ci && npm test -- --ci

      # … idem pour user/chat/game/matchmaking

      - name: Build images (no push)
        uses: docker/build-push-action@v6
        with:
          context: .
          file: services/auth/Dockerfile
          push: false
          tags: ghcr.io/mechard-organization/ft-auth:ci-${{ github.sha }}
```

On ne **push** pas dans la CI PR : build rapide pour vérifier que ça compile.

CD préprod (branche dev)

`.github/workflows/cd-dev.yml`

```yaml
name: CD-Dev
on:
  push:
    branches: [ dev ]
jobs:
  build-push-and-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    env:
      REG: ghcr.io/mechard-organization
      SHA: ${{ github.sha }}
    steps:
      - uses: actions/checkout@v4

      - uses: docker/setup-qemu-action@v3
      - uses: docker/setup-buildx-action@v3

      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      # Build & push toutes les images (tag dev-<sha>)
      - name: Build & Push auth
        uses: docker/build-push-action@v6
        with:
          context: services/auth
          push: true
          tags: |
            ${{ env.REG }}/ft-auth:dev-${{ env.SHA }}

      # … répéter pour user/chat/game/matchmaking/frontend/gateway

      # (Option) Scan Trivy
      - name: Security scan
        uses: aquasecurity/trivy-action@0.20.0
        with:
          image-ref: ${{ env.REG }}/ft-auth:dev-${{ env.SHA }}
          format: table
          exit-code: 0

      # Déploiement préprod via SSH
      - name: Deploy to staging
        uses: appleboy/ssh-action@v1.0.3
        with:
```

```
        host: ${{ secrets.STAGING_HOST }}
        username: ${{ secrets.STAGING_USER }}
        key: ${{ secrets.STAGING_SSH_KEY }}
        script: |
          set -e
          cd /opt/transcendence
          export IMAGE_TAG=dev-${{ env.SHA }}
          docker compose -f compose.prod.yml pull
          docker compose -f compose.prod.yml up -d
          docker compose -f compose.prod.yml ps
```

Release prod (tag `vX.Y.Z` sur `main`)

`.github/workflows/release.yml`

```
name: Release
on:
  push:
    tags: ['v*.*.*']
jobs:
  build-push-and-deploy:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    env:
      REG: ghcr.io/mechard-organization
      VERSION: ${{ github.ref_name }}  # ex: v1.2.0
    steps:
      - uses: actions/checkout@v4
        with: { fetch-depth: 0 }

      - uses: docker/setup-qemu-action@v3
      - uses: docker/setup-buildx-action@v3

      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      # Build & push avec tags :vX.Y.Z et :latest
      - name: Build & Push auth
        uses: docker/build-push-action@v6
        with:
          context: services/auth
          push: true
```

```
    tags: |
      ${{ env.REG }}/ft-auth:${{ env.VERSION }}
      ${{ env.REG }}/ft-auth:latest

# … répéter pour les autres services

- name: Deploy to production
  uses: appleboy/ssh-action@v1.0.3
  with:
    host: ${{ secrets.PROD_HOST }}
    username: ${{ secrets.PROD_USER }}
    key: ${{ secrets.PROD_SSH_KEY }}
    script: |
      set -e
      cd /opt/transcendence
      export IMAGE_TAG=${{ env.VERSION }}
      docker compose -f compose.prod.yml pull
      docker compose -f compose.prod.yml up -d
      docker compose -f compose.prod.yml ps
```

Secrets requis : `STAGING_HOST`, `STAGING_USER`, `STAGING_SSH_KEY`, `PROD_HOST`, `PROD_USER`, `PROD_SSH_KEY`.
Option : utiliser **Docker context** ou un runner self-hosted.

🐳 `compose.prod.yml` (prod, images taggées)

```
version: "3.9"
name: transcendence

services:
  gateway:
    image: ghcr.io/mechard-organization/ft-gateway:${IMAGE_TAG:-latest}
    ports: ["80:80"]
    depends_on:
      - frontend
      - auth
      - user
      - chat
      - game
    healthcheck:
      test: ["CMD-SHELL","wget -qO- http://localhost/health || exit 1"]
      interval: 10s
      timeout: 2s
      retries: 6
    environment:
      - TZ=Europe/Paris
    restart: unless-stopped

  frontend:
    image: ghcr.io/mechard-organization/ft-front:${IMAGE_TAG:-latest}
    environment:
      - NODE_ENV=production
    restart: unless-stopped

  auth:
    image: ghcr.io/mechard-organization/ft-auth:${IMAGE_TAG:-latest}
    env_file: [.env.prod]
    environment:
      - PORT=3001
      - SQLITE_FILE=/data/auth.sqlite
    volumes:
      - auth_db:/data
    healthcheck:
      test: ["CMD","curl","-fsS","http://localhost:3001/health"]
      interval: 10s
      timeout: 2s
      retries: 6
    restart: unless-stopped
```

```
  user:
    image: ghcr.io/mechard-organization/ft-user:${IMAGE_TAG:-latest}
    env_file: [.env.prod]
    environment:
      - PORT=3002
      - SQLITE_FILE=/data/user.sqlite
    volumes: [ "user_db:/data" ]
    restart: unless-stopped

  chat:
    image: ghcr.io/mechard-organization/ft-chat:${IMAGE_TAG:-latest}
    env_file: [.env.prod]
    environment:
      - PORT=3003
      - SQLITE_FILE=/data/chat.sqlite
    volumes: [ "chat_db:/data" ]
    restart: unless-stopped

  game:
    image: ghcr.io/mechard-organization/ft-game:${IMAGE_TAG:-latest}
    env_file: [.env.prod]
    environment:
      - PORT=3004
      - SQLITE_FILE=/data/game.sqlite
    volumes: [ "game_db:/data" ]
    restart: unless-stopped

  matchmaking:
    image: ghcr.io/mechard-organization/ft-matchmaking:${IMAGE_TAG:-latest}
    env_file: [.env.prod]
    environment:
      - SQLITE_FILE=/data/match.sqlite
    volumes: [ "match_db:/data" ]
    restart: unless-stopped

volumes:
  auth_db:
  user_db:
  chat_db:
  game_db:
  match_db:
```

Note : pas de bind-mount en prod. **Images immuables**, data en volumes. `IMAGE_TAG` injecté par le job (ex : `v1.2.0`).

🔐 `.env.prod` (sur le serveur)

```
NODE_ENV=production
JWT_SECRET=change-me
OAUTH42_CLIENT_ID=...
OAUTH42_CLIENT_SECRET=...
MAIL_SMTP_URL=...
```

> Géré **côté serveur**. Ne jamais commiter.
> GitHub Actions n'a pas besoin de ce fichier pour builder/pusher.

🔄 Stratégie de déploiement & rollback

- **Déploiement** : `pull` puis `up -d` → Compose recrée uniquement ce qui change.
- **Santé** : healthchecks bloquent l'expo via Nginx si non healthy.
- **Rollback** :
  - réexporter `IMAGE_TAG` vers un tag précédent (ex. `v1.1.1`),
  - `docker compose -f compose.prod.yml up -d`
  - vérifier `ps`, logs.

> Si vous voulez du *rolling* plus fin : passer sur **Swarm** ou **Kubernetes**. En Compose, on reste sur du **recreate** service-par-service, mais c'est robuste pour votre scope.

🧪 Check de smoke post-déploiement

- `curl http://host/health` → `ok`
- `curl http://host/api/auth/health` → `ok`
- Connexion WS depuis front → OK, `chat`/`game` reçoivent.
- Prometheus cible les endpoints `/metrics` → UP.
- Grafana montre du trafic réel (<5 min).

🧯 Sécurité & hygiène

- **Users non-root** dans les images Node (`USER node`).
- **Labels** d'image avec SBOM (option `docker/buildx bake` + `--sbom`).
- **Scan** Trivy en *non-bloquant* au début, puis en bloquant si politique fixée.
- **Secrets** uniquement via Actions secrets + `.env.prod`.
- **Logs** : pino JSON → `docker logs` parseables (ou Loki plus tard).

📌 Règles Git & protections

- **Branch protections** sur `main` & `dev` :
  - PR obligatoire, 1 review minimum, CI verte.
  - Pas de push direct.
- **Release** via **tags** SemVer.
- **Changelog** lors du tag (auto-génération possible via `release-please`).

✅ Definition of Done (DoD)

- CI PR : lint + tests + build **OK**.
- `dev` : images `dev-<sha>` poussées + **déploiement préprod**.
- `main` taguée : images `:vX.Y.Z` + `:latest` poussées + **déploiement prod**.
- `compose.prod.yml` fonctionnel, healthchecks **OK**, rollback documenté.
- Secrets en place, métriques **scrapées** après déploiement.
- Guide rapide pour l'équipe (`DEPLOY.md`) avec commandes essentielles.
-

Jeanne

🗓️ Semaine 1 – Reprise & Environnement

🎯 Objectif global

Mettre en place **le squelette complet du front-end** :

- un projet TypeScript moderne, bien structuré,
- un système de **routing SPA (Single Page Application)** clair,
- une **navigation sécurisée** (auth, redirections),
- des **layouts réutilisables** (header, sidebar, content),
- et une **base de design stable** (composants, thème, Tailwind).

Le but : qu'à la fin de cette tâche, tout le monde puisse brancher ses modules front (auth, chat, gameplay, etc.) sur une base commune et robuste.

🧱 Stack & technologies retenues

- **Framework :** React + Vite (ou Next.js si SSR prévu, mais Vite est parfait pour SPA 42).
- **Langage :** TypeScript.
- **CSS :** TailwindCSS (rapide, cohérent, responsive).
- **Router :** React Router v6 (ou équivalent si Next).
- **State global :** Zustand ou Redux Toolkit (léger, prévisible).
- **Sécurité :** gestion des tokens, routes privées, logout automatique sur 401.
- **Accessibilité :** focus trap, navigation clavier, thème contrasté (Jeanne 👑).

🗂️ Structure du projet (exemple standard)

```
/frontend
  /src
    /api
      auth.ts         # fonctions fetcher API (login, register, refresh)
      user.ts         # fetch profil, update, etc.
    /components
      Layout.tsx      # header, sidebar, etc.
      Loader.tsx
      ProtectedRoute.tsx
    /hooks
      useAuth.ts
      useFetch.ts
    /pages
      /auth
        Login.tsx
        Register.tsx
      /user
        Profile.tsx
      /chat
        ChatRoom.tsx
      /game
        GameView.tsx
      Home.tsx
      NotFound.tsx
    /routes
      router.tsx
    /store
      authStore.ts
      themeStore.ts
    /utils
      constants.ts
      helpers.ts
    App.tsx
    main.tsx
  index.html
  tailwind.config.js
  vite.config.ts
```

Cette organisation permet à chaque membre de travailler dans **son module (`/pages/…`)** sans casser les autres.

🔗 Routing – le cœur du front

Utilisation de **React Router v6** (compatible avec Vite, performant et simple).

```
// src/routes/router.tsx
import { createBrowserRouter } from "react-router-dom";
import AppLayout from "@/components/Layout";
import Home from "@/pages/Home";
import Login from "@/pages/auth/Login";
import Register from "@/pages/auth/Register";
import Profile from "@/pages/user/Profile";
import ChatRoom from "@/pages/chat/ChatRoom";
import GameView from "@/pages/game/GameView";
import ProtectedRoute from "@/components/ProtectedRoute";
import NotFound from "@/pages/NotFound";

export const router = createBrowserRouter([
 {
   element: <AppLayout />,
   children: [
    { path: "/", element: <Home /> },
    {
      path: "/login",
      element: <Login />,
    },
    {
      path: "/register",
      element: <Register />,
    },
    {
      path: "/profile",
      element: (
        <ProtectedRoute>
          <Profile />
        </ProtectedRoute>
      ),
    },
    {
      path: "/chat/:roomId",
      element: (
        <ProtectedRoute>
          <ChatRoom />
        </ProtectedRoute>
      ),
    },
    {
      path: "/game/:matchId",
```

```
        element: (
          <ProtectedRoute>
            <GameView />
          </ProtectedRoute>
        ),
      },
      { path: "*", element: <NotFound /> },
    ],
  },
]);
```

> **Avantage :** Jeanne peut valider dès maintenant la navigation et la structure — avant même que les modules soient remplis.

🧩 Layout principal (AppLayout)

```
// src/components/Layout.tsx
import { Outlet, NavLink } from "react-router-dom";

export default function AppLayout() {
  return (
    <div className="min-h-screen bg-gray-950 text-white flex">
      <aside className="w-64 bg-gray-900 p-4 flex flex-col">
        <h1 className="text-xl font-bold mb-6 text-blue-400">🏓 Transcendence</h1>
        <NavLink to="/" className="py-2 hover:text-blue-300">Accueil</NavLink>
        <NavLink to="/chat" className="py-2 hover:text-blue-300">Chat</NavLink>
        <NavLink to="/profile" className="py-2 hover:text-blue-300">Profil</NavLink>
      </aside>

      <main className="flex-1 p-6 overflow-y-auto">
        <Outlet />
      </main>
    </div>
  );
}
```

🛡️ Routes protégées (ProtectedRoute)

Jeanne devra mettre en place un composant qui **bloque l'accès** aux pages privées si l'utilisateur n'est pas authentifié.

```
// src/components/ProtectedRoute.tsx
import { Navigate } from "react-router-dom";
import { useAuthStore } from "@/store/authStore";

export default function ProtectedRoute({ children }: { children: JSX.Element }) {
  const { isAuth } = useAuthStore();
  return isAuth ? children : <Navigate to="/login" replace />;
}
```

🔐 Auth & Store global

```
// src/store/authStore.ts
import { create } from "zustand";
import { persist } from "zustand/middleware";

interface AuthState {
  token: string | null;
  user: { id: string; login: string } | null;
  isAuth: boolean;
  login: (token: string, user: any) => void;
  logout: () => void;
}

export const useAuthStore = create<AuthState>()(
  persist(
    (set) => ({
      token: null,
      user: null,
      isAuth: false,
      login: (token, user) => set({ token, user, isAuth: true }),
      logout: () => set({ token: null, user: null, isAuth: false }),
    }),
    { name: "auth-storage" }
  )
);
```

Ce store sera commun à tous les modules (chat, game, profil).
Il permet à `ProtectedRoute` et `Layout` d'accéder directement à l'état d'auth.

🌈 Thème & accessibilité

Jeanne doit poser les **bases d'un design cohérent et accessible** :

- **Contraste AA minimum** sur tous les textes.
- **Composants interactifs** focusables au clavier (`tabindex`).
- **Dark mode** natif Tailwind (`class: dark`).

Couleurs définies dans `tailwind.config.js` :

```
 colors: {
  bg: '#0f172a',
  accent: '#3b82f6',
  text: '#e2e8f0'
}
```

-
- Polices lisibles, tailles relatives (`rem`).
- Utiliser les labels ARIA (`aria-label`, `role="button"`, etc.).

> Bonus : intégration de `@headlessui/react` ou `radix-ui` pour les composants accessibles (modals, menus, dropdowns).

🧩 Intégration future

Module	Rôle	Intégration front
Auth (Maxime)	Login/Register	Pages `/login`, `/register`
User (Maxime)	Profil, stats	`/profile`
Chat (Abdul)	Interface WS	`/chat/:roomId`
Game (Abdul)	Canvas Pong	`/game/:matchId`
Tournois (Lylou)	Brackets	`/tournament/:id`

Jeanne assure la **cohérence UX** : transitions, layouts, routes uniformes.

📈 Observabilité front

Facile à ajouter dès le départ :

```
// src/main.tsx
import { register } from 'promjs'; // mini Prometheus client (optionnel)
register({ name: 'frontend_load_time_ms', help: 'Time to first paint' });
```

Mais l'essentiel pour Jeanne : un **console.log clair des routes** au début (debug), puis suppression avant prod.

✅ Definition of Done (DoD)

- Projet React + TS + Tailwind initialisé.
- Router fonctionnel (`/`, `/login`, `/register`, `/profile`, `/chat/:id`, `/game/:id`).
- Layout global opérationnel.
- `ProtectedRoute` implémenté.
- State global (`authStore`) persistant.
- Thème Tailwind cohérent (dark mode, contraste AA).
- Base d'accessibilité (focus visible, aria-labels).
- Documentation minimale (`README_front.md`) : structure + commandes.

💡 Étape suivante pour Jeanne

Une fois la base et le routing livrés :

- Intégrer les pages Auth (avec Maxime).
- Préparer le composant ChatShell pour Abdul.
- Connecter le store Auth avec le backend via `fetcher.ts`.
- Ajouter un composant "Loading" global (`<Loader />`) pour les transitions entre routes.

🎯 Objectif global

- **Login**: email/login + password, support **2FA TOTP**, gestion d'erreurs propre, "remember me".
- **Profil**: lecture/édition des infos (display name, bio, avatar, pays, préférences), **blocklist** minimale, stats en lecture.
- **UX fiable**: formulaires typés, validations immédiates, tokens gérés proprement, redirections cohérentes.

🧱 Pages & fonctionnalités

Login

- Champs: `login` (ou email), `password`.
- Options: "Remember me", lien "Mot de passe oublié ?" (placeholder si non prévu), **2FA step** conditionnel.
- États: `idle` → `submitting` → `error/success`.
- Redirection: vers `redirectTo` query ou `/`.

Flow 2FA

1. `POST /auth/login {login, password}` → `requires2FA: true` ?
2. Si oui: afficher champ `code` (6 digits) → `POST /auth/2fa/verify { code }`.
3. Enregistrer `{accessToken, refreshToken}` puis `useAuthStore.login()`.

Profil

- **Lecture**: `/user/me` (profil + stats).
- **Édition**: `display_name`, `bio`, `country`, `prefs` (JSON simple), **avatar upload**.
- **Blocklist**: lister/supprimer un blocage (CRUD minimal).
- Afficher stats (`games_played`, `games_won`, `elo_rating`).

Avatar upload

- Input file (PNG/JPG ≤ 1–2 Mo), aperçu local, envoi multipart → reçoit `avatar_url`.
- Alt text = “Avatar de {display_name}” (a11y).

🔒 Sécurité côté front

- **Tokens** : `accessToken` mémoire + storage (persist via Zustand) ; `refreshToken` via cookie httpOnly si possible, sinon stockage sécurisé + rotation côté API.
- **Intercepteur fetch**: sur 401 → tenter `POST /auth/refresh`, rejouer la requête, sinon `logout()` + redirect `/login`.
- **CSRF**: limité si tout en JWT + même domaine ; sinon `SameSite=Lax` sur cookies.
- **Validation stricte** côté client (zod) + côté serveur (déjà prévu).

🔌 Contrats API (rappel rapide)

- `POST /auth/login` → `{ requires2FA?: true }` ou `{ accessToken, refreshToken, user }`.
- `POST /auth/2fa/verify` → `{ accessToken, refreshToken, user }`.
- `GET /auth/me` → `{ user }`.
- `GET /user/me` → `{ profile, stats }`.
- `PUT /user/me` → update profil (JSON).
- `POST /user/avatar` → multipart form → `{ avatar_url }`.
- `GET /user/blocked` / `POST /user/block/:id` / `DELETE /user/block/:id`.

✅ Validation (zod)

- **Login**: `login` 3–50, `password` ≥ 8.
- **2FA**: 6 chiffres strict.
- **Profil**: `display_name` 2–32, `bio` ≤ 280, `country` ISO (ou liste), `prefs` JSON parseable.
- **Avatar**: type `image/png|jpeg`, taille ≤ 2 Mo, ratio carré conseillé.

🧩 Composants

- `FormField` (label, input, message d'erreur).
- `PasswordInput` (toggle visible/masqué, aria-controls).
- `TwoFAInput` (6 cases ou input unique validé).
- `AvatarUploader` (drag&drop, preview, bouton "Enregistrer").
- `StatBadge` (ELO, winrate).
- `BlockList` (tableau + bouton Unblock).

🦾 State & hooks

- `useAuthStore`: `token`, `user`, `isAuth`, `login()`, `logout()`.
- `useApi()`: wrapper fetch avec intercepteur 401→refresh.
- `useProfile()` : fetch + cache (SWR/tanstack-query optionnel).

♿ Accessibilité

- Labels explicites, `aria-invalid` sur erreurs, `aria-live="polite"` pour messages.
- Focus visible, ordre Tab logique.
- Boutons avec `aria-busy` pendant submit.

🧪 Tests (front)

- Unit: validation zod, hooks `useApi` (mock fetch), rendu des erreurs.
- Intégration: login happy path, 2FA path, refus → message.
- Profil: changement de display name, upload avatar (mock), block/unblock.

✨ Snippets clés

Login (React + RHF + zod)

```
// Login.tsx
import { useForm } from "react-hook-form";
import { z } from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useAuthStore } from "@/store/authStore";
import { useState } from "react";

const schema = z.object({
  login: z.string().min(3, "Identifiant trop court"),
  password: z.string().min(8, "Mot de passe trop court"),
});
type Form = z.infer<typeof schema>;

export default function Login() {
  const { register, handleSubmit, formState:{ errors, isSubmitting } } =
    useForm<Form>({ resolver: zodResolver(schema) });
  const auth = useAuthStore();
  const [needs2FA, setNeeds2FA] = useState(false);
  const [twofa, setTwofa] = useState("");

  const onSubmit = async (data: Form) => {
    const res = await fetch("/api/auth/login", { method:"POST", headers:{
"content-type":"application/json" }, body: JSON.stringify(data) });
    const json = await res.json();
    if (json.requires2FA) { setNeeds2FA(true); return; }
    auth.login(json.accessToken, json.user);
  };

  const onVerify2FA = async () => {
    const res = await fetch("/api/auth/2fa/verify", { method:"POST", headers:{
"content-type":"application/json" }, body: JSON.stringify({ code: twofa }) });
    const json = await res.json();
    auth.login(json.accessToken, json.user);
  };

  return (
    <div className="max-w-sm mx-auto p-6">
      {!needs2FA ? (
        <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
          <div>
            <label className="block mb-1">Identifiant</label>
            <input {...register("login")} className="w-full input" aria-invalid={!!errors.login}/>
            <p className="text-red-400 text-sm">{errors.login?.message}</p>
```

```
          </div>
          <div>
            <label className="block mb-1">Mot de passe</label>
            <input type="password" {...register("password")} className="w-full input"
aria-invalid={!!errors.password}/>
            <p className="text-red-400 text-sm">{errors.password?.message}</p>
          </div>
          <button disabled={isSubmitting} className="btn btn-primary w-full">{isSubmitting ?
"Connexion..." : "Se connecter"}</button>
        </form>
      ) : (
        <div className="space-y-3">
          <label className="block mb-1">Code 2FA</label>
          <input value={twofa}
onChange={e=>setTwofa(e.target.value.replace(/\D/g,"").slice(0,6))} inputMode="numeric"
className="w-full input"/>
          <button onClick={onVerify2FA} className="btn btn-primary w-full">Vérifier</button>
        </div>
      )}
    </div>
  );
}
```

Profil (édition + avatar)

```
// Profile.tsx (extrait)
import { useForm } from "react-hook-form";
import { z } from "zod"; import { zodResolver } from "@hookform/resolvers/zod";
const schema = z.object({
  display_name: z.string().min(2).max(32),
  bio: z.string().max(280).optional(),
  country: z.string().max(2).optional(),
});
type Form = z.infer<typeof schema>;

export default function Profile() {
  const { data } = useProfile(); // GET /user/me
  const { register, handleSubmit, reset, formState:{ isSubmitting } } =
    useForm<Form>({ resolver: zodResolver(schema), values: data?.profile });

  const onSubmit = async (payload: Form) => {
    await fetch("/api/user/me",{ method:"PUT", headers:{ "content-type":"application/json" },
body: JSON.stringify(payload) });
  };

  const onAvatar = async (file: File) => {
    const fd = new FormData(); fd.append("avatar", file);
    const r = await fetch("/api/user/avatar",{ method:"POST", body: fd });
```

```
    // rafraîchir profil
  };

  return (
    <div className="grid md:grid-cols-[240px_1fr] gap-8">
      <section>
        <AvatarUploader onPick={onAvatar} url={data?.profile.avatar_url}/>
      </section>
      <section>
        <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
          <div>
            <label>Nom affiché</label>
            <input {...register("display_name")} className="input w-full"/>
          </div>
          <div>
            <label>Bio</label>
            <textarea {...register("bio")} rows={4} className="input w-full"/>
          </div>
          <div>
            <label>Pays</label>
            <input {...register("country")} placeholder="FR" className="input w-24"/>
          </div>
          <button disabled={isSubmitting} className="btn btn-primary">Enregistrer</button>
        </form>
        <div className="mt-8">
          <h3 className="font-semibold mb-2">Stats</h3>
          <div className="flex gap-4 text-sm opacity-90">
            <span>Parties : {data?.stats.games_played}</span>
            <span>Victoires : {data?.stats.games_won}</span>
            <span>ELO : {data?.stats.elo_rating}</span>
          </div>
        </div>
      </section>
    </div>
  );
}
```

🔁 Erreurs & messages

- Login: “Identifiants invalides”, “2FA incorrect”, “Compte bloqué n tentatives”.
- Profil: “Nom déjà pris” (si public), “Type d’image non supporté”, “Fichier trop lourd”.
- Toasts non bloquants (`aria-live="polite"`), pas de modals pour des erreurs simples.

🧯 Cas limites

- **Token expiré** pendant edit profil → intercepteur refresh → rejouer requête.
- **Upload annulé** → pas de crash, état remis à neuf.
- **Réseau offline** → bannière “Hors ligne”, formulaires désactivés.
- **Multi-onglets**: logout depuis un onglet → event `storage` pour déconnecter les autres.

📦 Styling rapide (Tailwind)

Classes utilitaires standard:

- `.input`: `px-3 py-2 rounded bg-gray-800 border border-gray-700 focus:outline-none focus:ring-2 focus:ring-blue-500`

- `.btn.btn-primary`: `px-4 py-2 rounded bg-blue-600 hover:bg-blue-500 disabled:opacity-50`

✅ Definition of Done (DoD)

- Login + 2FA fonctionnels, redirection post-login OK.
- Profil: lecture/édition + upload avatar OK, stats affichées.
- Validations zod côté client, messages d’erreurs accessibles.
- Intercepteur 401→refresh opérationnel.
- Tests min: 2FA happy/failed, update profil, upload avatar (mock).
- Respect du thème et de l’accessibilité (focus, aria, contrastes).

🎯 Objectif global

Créer une **interface de chat moderne, fluide et accessible**, intégrée à l'application principale :

- **Vue conversationnelle** claire (DMs, salons publics, tournois).
- **Notifications en temps réel** (nouveau message, défi, match, connexion d'un ami).
- Gestion **multirooms WebSocket**, cohérente avec le backend.
- Thème, raccourcis clavier, animations douces et feedback visuel immédiat.

> Lylou fournit les WebSockets et la persistance ; Jeanne conçoit et implémente toute la **couche visible**, en TypeScript + React + Tailwind.

🧱 Architecture Front – Chat

```
/src
  /modules/chat
    ChatShell.tsx        ← wrapper principal (layout + socket)
    ChatSidebar.tsx      ← liste de salons + recherche
    ChatRoom.tsx         ← messages + input
    ChatMessage.tsx      ← composant message (text + meta)
    ChatInput.tsx        ← zone d'écriture (mentions, emoji)
    ChatNotifications.tsx ← popup toast + icône header
  /store/chatStore.ts    ← state global (messages, currentRoom, socket)
  /hooks/useChatSocket.ts ← abstraction WebSocket
  /hooks/useNotifications.ts
```

💬 Fonctionnalités clés

Sidebar (liste de rooms / DMs)

- Liste scrollable, triée par activité récente.
- Icône + titre (room name ou pseudo).
- États : *online/offline*, *unread count*.
- Bouton "+" → créer ou rejoindre room (modale).
- Recherche : filtrage instantané local.

```
<aside className="w-72 bg-gray-900 border-r border-gray-800 p-3 flex flex-col">
  <div className="flex justify-between items-center mb-4">
    <h2 className="text-lg font-semibold">Chats</h2>
```

```
   <button className="text-blue-400 hover:text-blue-300">＋</button>
  </div>
  <input type="text" placeholder="Rechercher…" className="input mb-3" />
  <ul className="space-y-1 overflow-y-auto flex-1">
   {rooms.map(r => (
    <li key={r.id}
       className={`p-2 rounded ${r.id===current?'bg-gray-800':'hover:bg-gray-800'}`}
       onClick={()=>setCurrent(r.id)}>
     <span className={r.unread?'text-blue-400 font-bold':''}>{r.name}</span>
    </li>
   ))}
  </ul>
</aside>
```

ChatRoom (vue des messages)

- Flux vertical inversé, autoscroll vers le bas.
- Messages : bulles alignées gauche/droite selon auteur.
- Timestamp + pseudo au survol.
- Markdown léger (`bold`, `italic`, liens cliquables).
- Réactions (emoji) facultatives.
- Séparateurs “Aujourd’hui”, “Hier”.

1 message = { id, user, text, ts, reactions? }
Stockage dans Zustand `chatStore`.

```
<div ref={scrollRef} className="flex-1 overflow-y-auto space-y-1 p-4">
  {messages.map(m => (
   <div key={m.id}
      className={`flex ${m.self?'justify-end':'justify-start'}`}>
    <div className={`rounded-lg px-3 py-2 max-w-[70%] text-sm
     ${m.self?'bg-blue-600 text-white':'bg-gray-800 text-gray-100'}`}>
     {m.text}
     <span className="block text-xs opacity-60 mt-1">{formatTime(m.ts)}</span>
    </div>
   </div>
  ))}
</div>
```

Input & Envoi

- Input text + bouton send + raccourci `Enter` (avec `Shift+Enter` pour saut de ligne).
- Détection mentions `@pseudo` avec autocomplete.
- Option emojis (`:smile:` ou menu).
- Envoi WS : `socket.emit('chat:message', {room, text})`.

```
<form onSubmit={sendMessage} className="flex items-center gap-2 p-3 border-t
border-gray-800">
  <textarea ref={ref} rows={1} placeholder="Écrire un message..."
    className="flex-1 bg-gray-900 rounded resize-none focus:ring-2 focus:ring-blue-500"/>
  <button className="btn btn-primary px-4 py-2">Envoyer</button>
</form>
```

Notifications

Trois niveaux :

1. **Toast visuel** (message entrant, invitation, victoire, etc.)
2. **Badge** sur l'icône "Notifications" dans la topbar.
3. **Popup système** (optionnelle) via API Notification (si autorisée).

→ Hook `useNotifications()` gère :

```
showToast("Nouveau message dans #general");
playSound("notif.mp3");
maybePushNotification("Abdul t'invite à jouer !");
```

Utilisation :

```
import { useNotifications } from "@/hooks/useNotifications";
const { showToast } = useNotifications();
showToast("🎮 Nouveau défi reçu !");
```

⚡ Gestion temps réel

- **Connexion WS** gérée dans `useChatSocket`.
- Écoute :
    - `chat:message`
    - `chat:user_joined`
    - `chat:user_left`
    - `invite:match`
    - `system:announcement`

```
socket.on("chat:message", (msg) => chatStore.addMessage(msg));
socket.on("invite:match", (invite) => showToast(`🎮 ${invite.from} t'invite à jouer !`));
```

> Le state global maintient la cohérence, même si Jeanne démonte/remonte le composant.

🪄 Expérience utilisateur (UX)

- Auto-scroll intelligent : vers le bas sauf si l'utilisateur lit l'historique.
- Animation d'apparition (Framer Motion, `animate-fadeIn`).
- Loader lors du join initial de la room.
- Préservation du brouillon local par room (`localStorage`).
- Shortcut clavier : `Ctrl+K` pour chercher un utilisateur ou room.

♿ Accessibilité

- Rôles ARIA :
    - `<ul role="log" aria-live="polite">` pour messages.
    - `aria-label` sur boutons, “Envoyer le message”.
- Contraste AA, focus visible.
- Lecture screen reader possible du dernier message.

🔔 Intégration avec le reste du front

- Notifications dans la **topbar globale** (ex : cloche 🔔).
- Compteur sur icône de chat dans la sidebar principale.
- Les “défis” (matchmaking) utilisent le même système de notification.

🎨 Style visuel

- Palette sombre (`bg-gray-900`, `text-gray-100`, `accent-blue-500`).
- Bulles arrondies, ombres subtiles, police monospace pour timestamps.
- Transitions `transition-colors` sur hover.
- Toasts en bas-droite : `position: fixed; bottom: 16px; right: 16px;`.

✅ Definition of Done (DoD)

- Sidebar + rooms dynamiques.
- ChatRoom opérationnelle (envoi/réception WS).
- Scroll auto intelligent.
- Input ergonomique + raccourcis clavier.
- Notifications visuelles + toast.
- Hooks `useChatSocket`, `useNotifications` fonctionnels.
- Aucune régression accessibilité (focus, ARIA).
- Tests : affichage d'un message entrant, toast, changement de room.

🧩 Extensions possibles (bonus)

- Mentions `@pseudo` → highlight + ping sonore.
- Indicateur "X est en train d'écrire…" (événement WS `typing`).
- Filtres (muted words / spam).
- Historique paginé via API REST.
- Web Push Notifications (service worker).

🎯 Objectif global

- **Types**: Single Elim (SE) d'abord. (Hooks prêts pour Double Elim / Round-Robin plus tard.)
- **Phases**: inscription → seeding → bracket → matches → podium.
- **Rôles**: owner/admin (crée/édite), joueur (rejoint, voit ses matchs), spectateur (lecture seule).
- **Live**: scores & états de matchs en temps réel via WS (`tournament:*`, `game:match#id`).

🧱 Architecture Front (proposée)

```
/src/modules/tournament
  BracketShell.tsx        // wrapper : charge données, écoute WS
  BracketView.tsx         // rendu SVG des rounds/liaisons
  MatchCard.tsx           // carte d'un match (players, score, CTA)
  SidebarPanel.tsx        // liste joueurs, seeds, réglages
  ControlsBar.tsx         // actions (start, shuffle, export)
  useBracketLayout.ts     // algorithmes layout (coords SVG)
  useTournamentWS.ts      // abonnement WS (notify + updates)
  types.ts                // types forts
```

Types (TS)

```
export type PlayerRef = { id: string; name: string; elo?: number };
export type MatchState = 'pending'|'live'|'done'|'walkover'|'bye';
export type Match = {
```

```
  id: string;
  round: number;        // 1 = quarts (ex)
  seedA?: number;       // têtes de série
  seedB?: number;
  a?: PlayerRef;
  b?: PlayerRef;
  score?: { a: number; b: number };
  state: MatchState;
  next?: { id: string; slot: 'a'|'b' }; // routage vers match suivant
};
export type Bracket = { rounds: Match[][]; size: 4|8|16|32; };
export type Tournament = {
  id: string; name: string; mode:'SE';
  players: PlayerRef[]; bracket: Bracket; ownerId: string;
  status:'signup'|'seeding'|'running'|'finished';
};
```

🔌 Contrats backend (résumé, côté UI)

- `GET /tournaments/:id` → `{ tournament }`
- `POST /tournaments/:id/seed` → fixe ordre/seed (admin)
- `POST /tournaments/:id/start` → génère bracket initial (avec byes si besoin)
- `POST /matches/:id/score {a,b}` → maj score (admin/arbitre)
- **WS** `tournament:updates#<id>` → `match.updated`, `player.joined`, `status.changed`
- **WS** `game:match#<id>` → pour spectate/joindre le match réel

🖼️ Rendu du bracket (SVG + layout stable)

Approche

- **SVG** pour la qualité et les liens “propres”.
- Layout entièrement **calculé côté front** (pixels) pour éviter les sauts.
- **Virtualisation légère** si >16 joueurs (optionnel).

Hook de layout

```
// useBracketLayout.ts (extrait)
export function useBracketLayout(bracket: Bracket, opts = { colW: 260, rowH: 90, gutterX: 80 }) {
  // calcule x/y de chaque match selon round/position
  // relie next.id pour dessiner les connecteurs
  return { nodes: /* {matchId, x, y, w, h}[] */, edges: /* {fromId,toId, path: string}[] */ };
}
```

BracketView

- Chaque `MatchCard` est placé via `transform: translate(x,y)`.
- **Edges**: path cubic Bézier pour lisibilité.
- **Responsive**: zoom/scroll (wheel+Ctrl), drag pan (maintien bouton).

🧩 MatchCard UX

- Compacte, lisible, focusable clavier.
- États visuels : `pending` (gris), `live` (accent animé), `done` (muted + score).
- Boutons contextuels (si admin):
    - “Start” (ouvre match → crée/associe `game:match#id`)
    - “Edit score” (modal accessible)
    - “Open chat” (ouvre `/chat/:roomId` du match)
    - “Spectate” (navigue `/game/:matchId`)

```
function MatchCard({ m, onEdit }: { m: Match; onEdit:()=>void }) {
  return (
    <div role="group" aria-label={`Match ${m.id}`} tabIndex={0}
      className={`rounded-lg border p-2 w-60 bg-gray-900 ${m.state==='live'?'border-blue-500 shadow':''}`}>
```

```
      <Row player={m.a} score={m.score?.a} highlight={m.state==='done' &&
(m.score!.a>m.score!.b)} />
      <Row player={m.b} score={m.score?.b} highlight={m.state==='done' &&
(m.score!.b>m.score!.a)} />
      <div className="flex gap-2 mt-2">
        {m.state==='pending' && <button className="btn-xs">Start</button>}
        {m.state!=='live' && <button className="btn-xs" onClick={onEdit}>Score</button>}
        {m.state==='live' && <button className="btn-xs">Spectate</button>}
      </div>
    </div>
  );
}
```

🧭 Sidebar & contrôles

- **Players**: liste + état (inscrit / seed / bye), drag-and-drop pour admin (clavier: déplacer via flèches + Entrée).
- **Actions**:
    - Shuffle seeds (Fisher-Yates)
    - Lock seeding (écrit côté backend)
    - Start tournament
    - Export PNG/SVG du bracket

Drag-and-drop doit être **accessible** : on ajoute une alternative clavier (focus + “Déplacer en haut/bas”).

⚡ Temps réel & cohérence

- `useTournamentWS(tournamentId)` :
    - `match.updated` → met à jour `bracket.rounds[round][i]` + recalcul edges si besoin.
    - `status.changed` → transition de `seeding` → `running` → `finished`.
    - `player.joined/left` (si inscriptions ouvertes).
- Optimiser en **immutabilité fine** (updates ciblées, pas de reflow complet).

🧮 Règles SE (Single Elimination) et byes

- Taille de bracket = **puissance de 2** supérieure (4,8,16,32).
- Si joueurs < taille → **byes** auto (match marqué `bye`, l'adversaire avance).
- Semis (seeding) : standard 1 vs (n), 2 vs (n-1)… ou **snake seeding**.
- "Walkover" si absence; score `WO` (UI le stylise et avance l'adversaire).

🔒 Permissions

- Admin/owner uniquement :
    - modifier seeds, lancer tournoi, éditer scores manuels, annuler match.
- Joueur : voit ses prochains matchs mis en avant (badge "Next for you").
- Spectateur : lecture seule + "Spectate" quand `live`.

♿ Accessibilité (indispensable)

- **SVG accessible** : chaque `MatchCard` a un `aria-label` clair.
- **Focus visible** pour cartes et boutons.
- **Navigation clavier** dans le bracket (flèches gauche/droite = round; haut/bas = match).
- **Modale score** avec **focus trap** et `aria-modal`.

🚀 Performance & confort

- Calcul layout **O(n)**, mémoïsé par `tournamentId + size`.
- Pas d'images lourdes ; 100% SVG/CSS.
- **Pan & zoom** lisses (transform GPU), limiter le repaint (container overflow).
- "Mini-map" optionnelle si 32 joueurs.

🔗 Intégrations

- **Chat du match**: bouton ouvre `/chat/:roomId` associé.
- **Game**: `Start` crée/associe `game:match#id`, puis "Spectate"/"Join".

- **Stats**: à la fin, UI affiche **podium** + MAJ ELO (lecture via `/user/stats/:id`).

🧪 Tests à prévoir (front)

- Génération bracket SE à 4/8/16 joueurs (incl. byes).
- Mise à jour `match.updated` (score → next slot correct).
- Accessibilité modale score (tab, ESC).
- Seeding: drag-and-drop + alternative clavier → backend mis à jour.
- Export SVG/PNG produit un rendu complet.

✅ Definition of Done (DoD)

- Bracket SE affiché (4→32).
- Seeding manipulable (admin), sauvegardé.
- Démarrage tournoi → matches `pending` puis `live/done` via WS.
- MatchCard ergonomique + modale score accessible.
- Sidebar joueurs, actions (shuffle, lock, start, export).
- Pan/zoom, responsive, focus clavier.
- Tests UI clés passants.

🎁 Bonus (si temps)

- **Double Elim** (tree losers + reset final).
- **Round-Robin** avec table des rencontres.
- **Timeline** des matchs (gantt mini).
- **Badges** : “you’re up in 2 matches”.

🗓️ Semaine 5 – Sécurité & Monitoring

🎯 Objectif global

Garantir que tout le front — login, profil, chat, tournois, gameplay — est :

1. **Accessible** à tous (clavier, lecteur d'écran, daltonisme, motion sensibility, etc.).
2. **Visuellement cohérent** (espacement, animations, couleurs, responsive).
3. **Techniquement propre** (HTML sémantique, aria-labels, focus management, contrastes AA/AAA).
4. **Emotionnellement agréable** : polices équilibrées, transitions fluides, micro-interactions, cohérence typographique.

Bref, Jeanne transforme le "ça marche" en "c'est agréable et universel".

🧩 Accessibilité universelle (A11y)

🧭 Navigation clavier

- **Tout élément interactif** (bouton, lien, input) est focusable (`tabindex=0` par défaut).
- Ordre **tab** logique et intuitif (de haut en bas, gauche à droite).

Focus visible : halo clair et contrasté, par ex.

:focus-visible { outline: 2px solid #3b82f6; outline-offset: 2px; }

-
- Pas de focus "sauté" après fermeture de modale (Jeanne remet le focus sur le bouton qui l'a ouverte).
- Raccourcis :
    - `Esc` → ferme modale/toast.
    - `Ctrl+K` → ouvre la recherche globale (chat ou profil).
    - `Enter` → valide les formulaires.
    - `Alt+←` / `→` → navigation historique.

📢 Lecteurs d'écran

- Tous les **éléments non textuels** ont un `aria-label` explicite :
    - `<button aria-label="Envoyer le message">`

    - `<img alt="Avatar de Lylou">`

- Les changements d'état importants déclenchent des **annonces** via `aria-live="polite"` (par exemple, "message reçu", "tournoi commencé").
- Utilisation cohérente des **rôles ARIA** :
    - `role="dialog"` pour modales,
    - `role="log"` pour flux de messages,
    - `role="navigation"`, `role="main"`, `role="status"`.
- Éviter les éléments **visuellement cachés mais lisibles** (`sr-only`) pour titres secondaires ou descriptions.

🌈 Contraste & daltonisme

- Vérifier les couleurs avec **contrast ratio ≥ 4.5:1** (AA minimum).
  Outils : `@tailwindcss/forms`, `@tailwindcss/typography`, plugin `a11y-check`.
- Ne pas coder le sens uniquement par la couleur (ex. : vert/rouge → icônes ou texte explicite).
- Mode **"Daltonisme"** optionnel (palette à saturation réduite et accent neutre).

  accent-blue-400 → accent-sky-300

  bg-gray-900 → bg-neutral-900

🌀 Réduction des animations

Respect du media query :

```
@media (prefers-reduced-motion: reduce) {
  * { animation: none !important; transition: none !important; }
}
```

- Les animations non essentielles (hover, fade, scale) restent **subtiles** et non-intrusives.
- Les transitions importantes (modales, toasts, chat pop) < 150ms.

🎨 Cohérence visuelle et design system

Jeanne harmonise **tous les composants** selon un **système visuel stable** :

Élément	Couleur / Style	Exemple
Boutons principaux	`bg-blue-600 hover:bg-blue-500 text-white`	CTA (“Envoyer”, “Jouer”)
Boutons secondaires	`bg-gray-800 hover:bg-gray-700 text-gray-200`	Annuler, Fermer
Inputs	`border-gray-700 focus:ring-blue-500`	Login, Chat
Modales	`bg-gray-950/95 backdrop-blur`	Score match, Invite
Textes	`text-gray-100` titres / `text-gray-400` secondaires	Général

Les **espacements** (marges, paddings) sont doublés entre sections (`2rem`) pour respirer visuellement.
Grille responsive : breakpoints Tailwind standard (`sm`, `md`, `lg`) avec réorganisation fluide.

✨ Micro-interactions & feedbacks

Le “polish” vient des **micro-réponses du front** :

- Bouton qui vibre légèrement au clic (`scale-95` sur 100ms).
- Apparition douce des messages (`fade-up` 150ms).

- Spinner minimal mais visible (`animate-spin text-blue-500`).
- Toasts qui glissent depuis la droite avec ombre légère.

Survol de carte → ombre + accent lumineux :

hover:shadow-[0_0_10px_#3b82f6]

-
- Animations synchronisées avec les sons (`/sounds/send.wav`, `/sounds/notif.mp3`).

Chaque interaction doit donner **un retour visuel ou sonore immédiat**.

📱 Responsive & adaptatif

- **Mobile-first** : Jeanne valide d'abord l'UX sur 360px (téléphone), puis monte jusqu'à 1920px.
- Les éléments critiques (chat, bouton match, profil) s'adaptent :
  - Sidebar → drawer mobile (slide-in).
  - Grille tournois → scroll horizontal ou zoom.
  - Modales → full-screen sur mobile.
- `useMediaQuery()` (hook custom) pour adapter densité d'infos :
  - moins de texte, icônes plus grandes en mobile.

🧰 Nettoyage et uniformisation technique

Code

- Noms de classes et fichiers cohérents (`ButtonPrimary`, `ChatInput`, `ModalWrapper`).
- Suppression des duplications de styles (`btn`, `input`, `card` via utilitaires Tailwind).
- Composants isolés réutilisables (`<Modal>`, `<Toast>`, `<Tooltip>`, `<Badge>`).
- Passage en TypeScript strict (`strict: true`).

Performance

- Suppression des images inutiles, compressions SVG.
- Lazy loading des routes lourdes (`React.lazy` + `Suspense`).
- Préchargement intelligent des pages fréquentées (`<link rel="prefetch">`).

♿ Checklist A11y (avant livraison)

Catégorie	Test	Statut
Navigation	Tout navigable au clavier	✅
Focus	Focus visible et non perdu	✅
Labels	Aria-labels sur chaque bouton/image	✅
Couleurs	Contraste AA/AAA	✅
Motion	Respect prefers-reduced-motion	✅
Alt text	Présents sur toutes les images	✅
Polices	Lisibles, tailles relatives	✅
Screen Reader	Navigation logique et structurée	✅

🧪 Vérification outils

Jeanne passe un audit avec :

- **Lighthouse** (onglet Accessibility → viser >95%).
- **axe DevTools** (Chrome extension).
- **NVDA / VoiceOver** (navigation clavier).
- **Color Oracle** (daltonisme).
- **Keyboard-only test** : zéro clic souris → app utilisable à 100 %.

🧩 Harmonisation UX (fin de sprint)

À ce stade, Jeanne coordonne :

- **Abdul** → cohérence des transitions Gameplay/Chat.
- **Lylou** → feedback réseau clair sur erreurs WS/API.
- **Maxime** → uniformiser boutons et toasts sur tout le front.
- **Mehdi** → conformité RGPD + consentement 2FA (form accessible).

✅ Definition of Done (DoD)

- Navigation 100 % clavier + screen reader.
- Contrast ratio conforme WCAG 2.1 AA.
- Motion respect “prefers-reduced-motion”.
- UI cohérente, sans décalages visuels.
- Lighthouse Accessibility ≥ 95 %.
- Micro-interactions homogènes (boutons, modales, chat).
- Tests réels NVDA/VoiceOver concluants.
- Audit final validé par l’équipe (design & dev).

🗓️ Semaine 6 – Finalisation & Présentation

🎯 Objectif global

Créer une **documentation claire, illustrée et exploitable** qui :

1. Définit le **système visuel** (composants, couleurs, typographie, espaces, feedbacks).
2. Décrit **toutes les règles d'accessibilité (A11y)** appliquées au projet.
3. Détaille les **bonnes pratiques de conception** et de contribution UI.
4. Sert de **référence unique** pour tout ajout ou refonte future.

> En somme : un *design system vivant* et une *charte d'accessibilité technique* réunis dans un guide.

🗂️ Structure proposée du document

Nom du fichier : `docs/UI_ACCESSIBILITY_GUIDE.md`
Format : Markdown GitHub, lisible directement dans le repo.

UI_ACCESSIBILITY_GUIDE.md
├── 🪶 Introduction
├── 🎨 Système visuel (Design System)
│ ├── Palette & contrastes
│ ├── Typographie & espacement
│ ├── Composants communs
│ └── États et feedbacks
├── ♿ Accessibilité (A11y)
│ ├── Navigation clavier
│ ├── Aria roles & labels
│ ├── Focus & hiérarchie visuelle
│ └── Gestion des médias & animations
├── 💡 Bonnes pratiques UX
│ ├── Formulaires
│ ├── Micro-interactions
│ ├── Erreurs & messages
│ └── Responsive & mobile
├── 🧪 Tests & audits
│ ├── Lighthouse
│ ├── axe DevTools
│ ├── Screen readers
│ └── Checklist finale
└── 👩‍💻 Guide de contribution UI

🪶 Introduction

Jeanne y précise le **but du guide**, les **principes de conception** et le **scope** du front.
Exemple :

> Ce guide définit les règles visuelles, ergonomiques et d'accessibilité du projet **ft_transcendence**.

Il garantit la cohérence, la maintenabilité et l'inclusivité de l'interface utilisateur sur l'ensemble des modules (auth, chat, profil, tournois, gameplay).

Chaque composant doit être utilisable :

- sans souris,
- sans couleur,
- sans animation obligatoire,
- et avec un lecteur d'écran.

🎨 Système visuel (Design System)

Palette & contrastes

Palette principale en **mode sombre**, validée WCAG 2.1 niveau AA :

Couleur	Usage	Code	Contraste
Bleu (accent)	Boutons, focus	`#3b82f6`	6.9:1
Gris foncé	Fond principal	`#0f172a`	—
Gris clair	Texte secondaire	`#94a3b8`	5.2:1
Rouge	Alerte / Erreur	`#ef4444`	4.8:1
Vert	Succès	`#10b981`	5.3:1

Mode *reduced motion* et *high contrast* : couleurs légèrement désaturées, animations coupées.

Typographie

- Police : **Inter** ou **Roboto** (sans-serif lisible).
- Hiérarchie : `text-3xl` titres, `text-xl` sous-titres, `text-sm` corps.
- Line-height entre 1.4 et 1.6 pour lecture fluide.

- Pas de texte < 12px.

Exemple :

h1 { font-size: 1.875rem; font-weight: 700; }
h2 { font-size: 1.5rem; font-weight: 600; }
p { font-size: 0.875rem; color: #e2e8f0; }

Espacement & grilles

Jeanne formalise la **règle des 8** (espacements en multiples de 8px).
Exemple : padding global = `p-8`, marge inter-sections = `my-16`.

Composants réutilisables

Définir les **composants standardisés** :

- `Button` → primary, secondary, danger
- `Input` → label + helper text + focus visible
- `Card` → conteneur UI commun
- `Modal` → trap de focus, `aria-modal`, fond semi-transparent
- `Toast` → notification non bloquante, `aria-live="polite"`

Chaque composant inclut :

1. Un code d'exemple.
2. Une note d'accessibilité.
3. Son comportement en cas d'erreur.

États et feedbacks

Jeanne consigne les **règles de transitions et d'interaction** :

- Hover = légère accentuation (`brightness(110%)`).

- Focus = halo bleu.
- Disabled = opacité réduite, non focusable.
- Erreur = message clair + bordure rouge accessible.

Exemple :
❌ “Erreur 500” → ✅ “Une erreur est survenue, réessayez.”

♿ Accessibilité (A11y)

Navigation clavier

- `Tab` / `Shift+Tab` pour circuler logiquement.
- `Enter` ou `Space` = actionner.
- `Esc` = fermer modales.
- Focus visible sur tous les éléments interactifs.

Aria roles & labels

- Tous les boutons ont un texte ou un `aria-label`.
- Modales = `role="dialog"` + `aria-modal="true"`.
- Chat = `role="log" aria-live="polite"`.
- Navigation = `role="navigation" aria-label="Menu principal"`.

Focus management

- Focus conservé au retour après fermeture de modale.

- Aucun “focus perdu” sur changement de page.

Médias & Motion

- Respect de `prefers-reduced-motion`.
- Aucune animation clignotante > 3Hz.
- Alternatives textuelles (`alt`, `caption`) sur chaque média.

💡 Bonnes pratiques UX

Jeanne y met ses **principes directeurs d’expérience utilisateur** :

- Toujours un **feedback visuel** après action.
- Les erreurs doivent être **compréhensibles** et **non techniques**.
- Les transitions ne doivent **jamais bloquer** la navigation.
- **Simplicité > densité** : une idée par écran.
- Utiliser **les mots des utilisateurs**, pas ceux des développeurs.

🧪 Tests & audits

Outils recommandés

- **Lighthouse** : score A11y > 95.
- **axe DevTools** : vérifie ARIA roles.
- **NVDA** (Windows) ou **VoiceOver** (Mac) : lecture clavier.
- **Color Oracle** : test daltonisme.
- **Keyboard-only audit** : 100 % des actions faisables sans souris.

Checklist finale

Catégorie	Critère	OK
Navigation	100 % clavier	✅

Focus	Toujours visible	✅
Aria	Présents et pertinents	✅
Contraste	≥ 4.5:1	✅
Motion	Réduit si demandé	✅
Texte alternatif	Images, icônes	✅
Responsivité	Mobile / desktop	✅

👩‍💻 Guide de contribution UI

Pour tout nouveau composant ou correctif :

1. **Vérifier le contraste** des nouvelles couleurs.
2. **Ajouter les aria-labels** requis.
3. **Tester au clavier** avant commit.
4. Inclure une capture Lighthouse dans la PR.
5. Ne jamais casser la structure des classes `btn`, `input`, `modal`.

Jeanne peut aussi y inclure :

```
# Vérification accessibilité rapide
npm run lint:a11y
npm run test:ui
```

✅ Definition of Done (DoD)

- Fichier `UI_ACCESSIBILITY_GUIDE.md` complet et versionné.
- Composants et règles documentés (visuels + code).

- Procédure de test Lighthouse & axe incluse.
- Checklist A11y exhaustive.
- Section “Contribuer” claire pour futurs devs.
- Score final Lighthouse ≥ 95.

💡 Bonus possible

- Intégration d’un **mini StyleGuide interactif** (route `/styleguide` en React).
  → Permet de visualiser et tester chaque composant directement dans l’app.
- Génération automatique du guide via **Storybook** + MDX.

Abdul

🗓️ Semaine 1 – Reprise & Environnement

🎯 Objectif global

- **Connexion WSS** stable (Nginx TLS) avec **reco auto** (exponential backoff, jitter).

- **Auth solide**: handshake avec JWT, refresh transparent, refus propre si token invalide.
- **Heartbeat**: ping/pong app-level (ou WS native si dispo) → détection de liens zombies < 10 s.
- **File d'attente**: messages sortants **bufferisés** avant `open` et durant reco, avec **flush** ordonné.
- **Idempotence**: `msgId` + `ack` pour éviter doubles actions après reco.
- **Multiplexage**: un **seul socket** partagé (game, chat, lobby) avec `topic`/`type`.
- **Schéma** de messages versionné (`v`, `type`, `topic`, `ts`, `payload`) + validation (zod).
- **Observabilité**: hooks d'événements, métriques (connects, retry count, RTT, drops), logs niveau debug.
- **API propre**: wrapper TS typé, callbacks ou event emitter, facile à mocker en tests.
- **Perf**: permessage-deflate activée, JSON compact (ou binaire plus tard), batch des updates quand utile.

🧱 Architecture proposée (client TS)

- `WsClient` (singleton) gère la vie du socket.
- `Channel`/`Subscription` pour topic (ex. `game:match#123`, `chat:dm#<id>`).
- `MessageBus` interne (EventEmitter) pour distribuer les events aux modules.
- `AuthProvider` (injecté) pour obtenir/renouveler le JWT.
- `Schema` zod pour valider `inbound/outbound`.

Schéma de message (JSON)

```
{
  "v": 1,
  "topic": "game:match#123",
  "type": "state_update",
  "msgId": "c2f6a...e1",
  "ts": 1730123456789,
  "payload": { "ball": {"x": 10, "y": 42}, "p1": 0.62, "p2": 0.34 }
}
```

🔌 Wrapper TypeScript (extrait)

```
// types.ts
export type Outbound = {
  v: 1;
  topic: string;
  type: string;
  msgId: string;
  ts: number;
  payload?: unknown;
};
export type Inbound = Outbound & { ack?: string; err?: string };

// wsClient.ts
import { z } from "zod";

const inboundSchema = z.object({
  v: z.number().int(),
  topic: z.string(),
  type: z.string(),
  msgId: z.string(),
  ts: z.number(),
  payload: z.any().optional(),
  ack: z.string().optional(),
  err: z.string().optional(),
});

type Handler = (msg: Inbound) => void;

export class WsClient {
  private ws?: WebSocket;
  private url: string;
  private getToken: () => Promise<string>;
  private handlers = new Map<string, Set<Handler>>();
  private queue: Outbound[] = [];
  private retry = 0;
  private hbTimer?: number;
  private lastPong = 0;
  private connected = false;

  constructor(url: string, getToken: () => Promise<string>) {
    this.url = url;
    this.getToken = getToken;
    window.addEventListener("online", () => this.ensure());
    document.addEventListener("visibilitychange", () => {
      if (document.visibilityState === "visible") this.ping();
    });
```

```
  }

  on(topic: string, h: Handler) {
    if (!this.handlers.has(topic)) this.handlers.set(topic, new Set());
    this.handlers.get(topic)!.add(h);
    return () => this.handlers.get(topic)!.delete(h);
  }

  async ensure() {
    if (this.connected || (this.ws && (this.ws.readyState === WebSocket.CONNECTING)))
return;
    const token = await this.getToken();
    const url = `${this.url}?token=${encodeURIComponent(token)}`;
    this.ws = new WebSocket(url);
    this.ws.onopen = () => this.onOpen();
    this.ws.onclose = () => this.onClose();
    this.ws.onerror = () => this.onError();
    this.ws.onmessage = (e) => this.onMessage(e.data);
  }

  send(msg: Omit<Outbound,"v"|"msgId"|"ts">) {
    const full: Outbound = {
      v: 1,
      msgId: crypto.randomUUID(),
      ts: Date.now(),
      ...msg,
    };
    if (this.connected && this.ws?.readyState === WebSocket.OPEN) {
      this.ws!.send(JSON.stringify(full));
    } else {
      this.queue.push(full);
    }
    return full.msgId;
  }

  private flush() {
    while (this.queue.length && this.ws?.readyState === WebSocket.OPEN) {
      this.ws!.send(JSON.stringify(this.queue.shift()));
    }
  }

  private onOpen() {
    this.connected = true;
    this.retry = 0;
    this.startHeartbeat();
    this.flush();
  }
```

```
private onClose() {
  this.connected = false;
  this.stopHeartbeat();
  this.backoffReconnect();
}

private onError() {
  // log minimal, la reco gère
}

private async onMessage(raw: string) {
  let parsed: Inbound;
  try {
    parsed = inboundSchema.parse(JSON.parse(raw));
  } catch {
    return;
  }
  if (parsed.type === "pong") {
    this.lastPong = Date.now();
    return;
  }
  const set = this.handlers.get(parsed.topic);
  set?.forEach(h => h(parsed));
}

private ping() {
  if (!this.connected) return;
  const now = Date.now();
  if (now - this.lastPong > 15000) {
    // link zombie → force reconnect
    this.ws?.close();
    return;
  }
  this.ws?.send(JSON.stringify({ v: 1, type: "ping", ts: now, topic: "hb" }));
}

private startHeartbeat() {
  this.lastPong = Date.now();
  this.hbTimer = window.setInterval(() => this.ping(), 5000);
}
private stopHeartbeat() { if (this.hbTimer) clearInterval(this.hbTimer); }

private backoffReconnect() {
  const base = 500; // ms
  const max = 5000;
  const jitter = Math.random() * 250;
  const delay = Math.min(max, base * 2 ** this.retry) + jitter;
  this.retry++;
```

```
    setTimeout(() => this.ensure(), delay);
  }
}
```

Utilisation côté modules (ex.) :

```
const ws = new WsClient("wss://example.com/ws", getJwtToken);
await ws.ensure();

const off = ws.on("game:match#123", (msg) => renderState(msg.payload));
ws.send({ topic: "game:match#123", type: "input", payload: { up: true } });
```

🔐 Auth & sécurité

- **Handshake** via querystring ou header `Sec-WebSocket-Protocol` (si serveur supporte).
- **Renouvellement**: si 401 côté serveur → client tente `getToken()` puis **reconnecte**.
- **Scopes**: limiter les topics autorisés par le JWT (claims).
- **Anti-replay**: `msgId` + horodatage; serveur peut ignorer doublons < fenêtre T.
- **Rate-limit** côté serveur (par IP/user/topic) et côté client (debounce input jeu à 60 Hz max).

🧪 Tests & critères d’acceptation

Unitaires

- Parser/validator (zod) refuse les messages invalides.
- Queue sortante se flush bien après `open`.
- Backoff calcule des délais bornés, avec jitter.

Intégration (mock WS ou serveur de test)

- Reco après `close` volontaire et après coupure réseau.
- Heartbeat: pas de “lien zombie” > 15s sans fermeture.
- Auth: token expiré → refresh → reconnect → reprise des topics.

Manuel (QA)

- Ouvrir 2 onglets, jouer 15 min; couper le wifi 5 s → reco OK, pas de double actions.
- Changer d’onglet 30x → pas de fuite de timers (profil mémoire).
- Flood chat (100 msgs/s sur 2 s) → pas de freeze UI, messages ordonnés.

📈 Observabilité côté client

- Compteurs: `ws_connects_total`, `ws_retries_total`, `ws_rtt_ms` (estimation: `ts`/pong).
- Logs debug activables via `localStorage.DEBUG="ws:*"`.

🔧 Intégration Nginx (rappel)

Proxy WS:

```
 proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "Upgrade";
proxy_read_timeout 60s;  # ajuster pour hb
```

- **WSS** obligatoire en prod, **permessage-deflate** activée si supportée.

🎮 Spécifique Gameplay

- Envoyer **inputs** (petits, fréquents), pas des frames lourdes.
- **Horloges**: inclure `serverTime` dans updates pour **clock-sync** (drift client < 50 ms).
- **Lag compensation** simple: côté client, prédire paddle local; côté serveur, autorité.

✅ Definition of Done (DoD)

- API `WsClient` stable, typée, documentée.
- Reco + heartbeat + queue + idempotence validés en tests.
- Auth/refresh intégré.
- Multiplexage par `topic` en place (jeu, chat, lobby utilisent la même connexion).
- Instrumentation minimale (compteurs) et logs debug.
- Remontée d'un **guide d'usage** pour les autres modules (2–3 exemples).

🗓️ Semaine 2 – Authentification & Utilisateurs

🎯 Objectif global

- **Auth WS** via JWT (avec 2FA déjà validé côté HTTP).
- **Gestion de session** : reconnexion, backoff, reprise d'abonnement.

- **Canaux/événements user** : présence, profil, amis, invites (match), notifications, sécurité (force-logout).
- **Interop** avec le reste du front : chat, tournois, game.
- **Observabilité** : métriques, logs client, tracing d'événements (léger).

🔌 Contrat réseau (proposé)

Endpoint WS (derrière Nginx)
`wss://<host>/ws?v=1` avec **JWT en sub-protocol** (plus propre que query) :

- Header `Sec-WebSocket-Protocol: bearer, <JWT>`

Handshake côté serveur

- Valide le JWT (exp, scope, 2FA).
- Attache `userId` au contexte et renvoie un `hello`:

{ "t": "hello", "userId": "u_123", "serverTs": 1730123456, "cap": ["presence","invite","notif","profile"] }

Keepalive

- `ping` (client → serveur) toutes 25s, serveur répond `pong`.
- Si 2 pings sans `pong` → reconnect.

🧱 Événements à supporter (côté user)

Type	Payload (ex)	Usage UI
`user:profile.updated`	`{ fields: ["avatar","display_name"] }`	Refresh profil
`user:presence`	`{ friendId, status: "online" | "offline"	

`invite:match`	`{ from, mode, roomId, expiresAt }`	Toast + CTA
`invite:accept/reject`	`{ matchId, by }`	Historique
`notif:new`	`{ id, kind, title, body, link }`	Cloches/badges
`tournament:updated`	`{ id, change:"match.updated", matchId }`	Brackets
`security:force-logout`	`{ reason: "session_revoked" }`	Logout immédiat
`system:announcement`	`{ title, body, severity }`	Bandeau

Commandes (client → serveur)

- `presence.set { status }`
- `invite.send { to, mode }`
- `invite.answer { inviteId, accept }`
- `notif.read { id }` (accusé de lecture)

🧰 Client WS côté front (TypeScript)

Gestionnaire unique (singleton) + Zustand

```
// wsClient.ts
type WSMsg = { t: string; [k: string]: any };
type Listener = (m: WSMsg) => void;

export class WSClient {
  private ws?: WebSocket;
  private url: string;
  private tokenProvider: () => Promise<string>;
  private listeners = new Set<Listener>();
  private hb?: number; private backoff = 1000; // ms (1s→10s)
  private alive = false;

  constructor(url: string, tokenProvider: () => Promise<string>) {
    this.url = url; this.tokenProvider = tokenProvider;
```

```
  }

  on(fn: Listener) { this.listeners.add(fn); return () => this.listeners.delete(fn); }
  emit(msg: WSMsg) { this.ws?.readyState === 1 && this.ws.send(JSON.stringify(msg)); }

  async connect() {
    const jwt = await this.tokenProvider();
    this.ws = new WebSocket(this.url, ["bearer", jwt]);

    this.ws.onopen = () => {
      this.alive = true;
      this.backoff = 1000;
      this.startHeartbeat();
    };

    this.ws.onmessage = (ev) => {
      const msg = JSON.parse(ev.data) as WSMsg;
      if (msg.t === "pong") { this.alive = true; return; }
      this.listeners.forEach(fn => fn(msg));
    };

    this.ws.onclose = () => this.scheduleReconnect();
    this.ws.onerror = () => { try { this.ws?.close(); } finally { /* noop */ } };
  }

  private startHeartbeat() {
    clearInterval(this.hb);
    this.hb = window.setInterval(() => {
      if (!this.ws || this.ws.readyState !== 1) return;
      this.alive = false;
      this.ws.send(JSON.stringify({ t: "ping", ts: Date.now() }));
      setTimeout(() => { if (!this.alive) this.ws?.close(); }, 6000);
    }, 25000);
  }

  private scheduleReconnect() {
    clearInterval(this.hb);
    setTimeout(() => this.connect(), Math.min(this.backoff, 10000));
    this.backoff *= 2;
  }
}
```

Intégration store (extrait) :

```
// userWsStore.ts
import { create } from "zustand";
type Presence = Record<string, "online"|"offline"|"ingame">;
```

```
type Notif = { id: string; kind: string; title: string; body: string; link?: string; unread?: boolean };

type State = {
  presence: Presence;
  notifs: Notif[];
  invites: any[];
  connect: () => void;
  send: (m: any) => void;
};

export const useUserWs = create<State>((set, get) => ({
  presence: {},
  notifs: [],
  invites: [],
  connect: () => {
    const client = new WSClient("wss://host/ws?v=1", async () => /* refresh JWT ici */ "");
    client.on((msg) => {
      switch (msg.t) {
        case "user:presence":
          set(s => ({ presence: { ...s.presence, [msg.friendId]: msg.status }})); break;
        case "notif:new":
          set(s => ({ notifs: [{...msg, unread:true}, ...s.notifs]})); break;
        case "invite:match":
          set(s => ({ invites: [msg, ...s.invites]})); break;
        case "security:force-logout":
          // call authStore.logout(); show toast; redirect
          break;
      }
    });
    set({ send: (m) => client.emit(m) });
    client.connect();
  },
  send: () => {}
}));
```

Dans `AppLayout` (dès login) :

```
const { isAuth } = useAuthStore();
const { connect } = useUserWs();
useEffect(() => { if (isAuth) connect(); }, [isAuth]);
```

Rafraîchissement JWT

- **Ne jamais** réutiliser un JWT expiré en boucle.

- `tokenProvider()` doit appeler `/auth/refresh` si le token va expirer < 60s.
- En cas d'échec → **logout** + notification claire.

🔒 Sécurité & durcissement

- **Subprotocol "bearer"** (évite JWT en query string).
- **Scopes** embarqués dans le JWT (`ws:user`, `ws:chat`…), le serveur filtre les topics.
- **Rate-limit** côté serveur sur commandes (`invite.send`, etc.).
- **Force-logout** si refresh token révoqué (événement dédié).
- **CORS/Upgrade** bien réglé sur Nginx (headers `Upgrade`, `Connection`).
- **No PII** dans les payloads de diffusion (masquer ce qui n'est pas nécessaire).

Nginx (rappel)

```
location /ws {
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
  proxy_pass http://ws_upstream;
}
```

🔔 UX côté user

- **Présence temps réel** des amis (pastilles, status textuels).
- **Toasts** non bloquants pour `notif:new` et `invite:match`.
- **Badge** compteur sur l'icône cloche.
- **"Ne pas déranger"** (status local) → limite les toasts/sons.
- **Persist brouillons** (ex. message d'invite) si reconnexion en plein flux.

🧪 Tests (abordables et utiles)

1. **Connexion** → **hello** reçu, capteurs initialisés.

2. **Heartbeat** : si `pong` non reçu, le client **reconnecte**.
3. **Backoff** : 1s → 2 → 4 → 8 → max 10s, puis stabilise.
4. **Presence** : réception `user:presence` met à jour UI immédiatement.
5. **Invite flow** : `invite:match` → toast + CTA → `invite.answer`.
6. **Force-logout** : déclenche logout + redirection `/login`.
7. **Token expire** : reconnexion automatique avec nouveau JWT.
8. **Pas de leak** : démonter/remonter le layout n'ajoute pas d'handlers fantômes.

> Mocks : utiliser `mock-socket` (ou équivalent) pour simuler serveur WS en tests UI.

📊 Observabilité

- **Compteur local** : nombre de reconnexions (`ws_reconnects_total`).
- **Dernier RTT** (ping→pong) pour diagnostiquer lenteurs.
- **Log niveau "debug"** activable via query `?debug=ws`.
- **Event audit** minimal : dernier 20 événements (mémoire) consultables dans un panneau Dev.

✅ Definition of Done (DoD)

- Connexion WS authentifiée (JWT subprotocol) + **heartbeat** OK.
- **Reconnexion résiliente** avec backoff et reprise des subscriptions.
- Événements **presence / invite / notif / security** intégrés au store.
- UI réactive : toasts, badges, pastilles statut, CTA "Accepter/Refuser".
- **No memory leaks** (handlers unique, cleanup correct).
- Tests clés passants (connexion, reconnexion, force-logout, invite).
- Docs rapides (`/docs/WS_USER.md`) : événements supportés + exemples.

🗓️ Semaine 3 – Chat & Interactions

🎯 Objectif global

Mettre en place un **système d’animations cohérent et réactif** pour les notifications du front :

- **Entrée / sortie douce**,
- **Feedback visuel contextuel** (succès, erreur, invite, message),
- **Respect de l’accessibilité** (durée, contraste, motion-reduce),
- **Uniformité UX** sur toutes les pages.

Cette tâche vient **après** que le système de WebSocket côté user soit prêt (car il déclenche les toasts et notifications dynamiques).

🧩 Types de notifications concernées

Type	Contexte	Couleur	Durée	Rôle
Succès	Action validée, match accepté, sauvegarde OK	Vert `#10B981`	3s	`aria-live="polite"`
Erreur	Connexion perdue, échec WS, conflit de merge	Rouge `#EF4444`	5s	`aria-live="assertive"`
Info	Nouveau message, événement système	Bleu `#3B82F6`	4s	`aria-live="polite"`
Invite / action	Invitation à un match, demande d’ami, tournoi	Jaune `#FACC15`	reste jusqu’à action	`aria-live="polite"`

🧠 Fonctionnement général

Chaque notification est **un composant React animé** avec Framer Motion (ou équivalent Tailwind Motion + transitions CSS).
Abdul doit intégrer :

1. Un **contexte global de notifications** (store Zustand ou context React).
2. Un **container dédié**, fixe en haut à droite (ou en bas si mobile).
3. Un système d'**animations fluides** :
    - slide-in, fade-out, stack vertical dynamique.
4. **Temps contrôlé** : durée configurable selon le type.
5. **Accessibilité visuelle & clavier** :
    - dismissible (`Esc` ou bouton fermer).
    - Focus trap si bloquant.
    - Respect de `prefers-reduced-motion`.

💻 Exemple d'implémentation technique

Store global (Zustand)

```
// notifStore.ts
import { create } from "zustand";
type Notif = { id: string; type: "success"|"error"|"info"|"invite"; title: string; msg?: string;
timeout?: number; };

export const useNotifStore = create<{
  list: Notif[];
  push: (n: Omit<Notif, "id">) => void;
  remove: (id: string) => void;
}>((set, get) => ({
  list: [],
  push: (n) => {
    const id = crypto.randomUUID();
    set(s => ({ list: [...s.list, { ...n, id }] }));
    if (n.timeout !== 0)
      setTimeout(() => get().remove(id), n.timeout ?? 4000);
  },
  remove: (id) => set(s => ({ list: s.list.filter(n => n.id !== id) })),
}));
```

Composant `<Notifications />`

```
import { motion, AnimatePresence } from "framer-motion";
import { useNotifStore } from "@/store/notifStore";

export function Notifications() {
  const { list, remove } = useNotifStore();
```

```
  return (
    <div className="fixed top-4 right-4 z-50 flex flex-col gap-3">
      <AnimatePresence>
        {list.map((n) => (
          <motion.div
            key={n.id}
            initial={{ opacity: 0, x: 50 }}
            animate={{ opacity: 1, x: 0 }}
            exit={{ opacity: 0, x: 50 }}
            transition={{ type: "spring", stiffness: 200, damping: 20 }}
            role="status"
            aria-live={n.type === "error" ? "assertive" : "polite"}
            className={`rounded-lg shadow-md px-4 py-3 text-white flex justify-between
items-center ${
              n.type === "success" ? "bg-emerald-600" :
              n.type === "error" ? "bg-rose-600" :
              n.type === "info" ? "bg-blue-600" : "bg-yellow-500"
            }`}
          >
            <div>
              <p className="font-semibold">{n.title}</p>
              {n.msg && <p className="text-sm opacity-90">{n.msg}</p>}
            </div>
            <button onClick={() => remove(n.id)} className="ml-3 opacity-70
hover:opacity-100">
              ✖
            </button>
          </motion.div>
        ))}
      </AnimatePresence>
    </div>
  );
}
```

Intégration dans l’app

```
// App.tsx
import { Notifications } from "@/components/Notifications";

function App() {
  return (
    <>
```

```
      <MainRouter />
      <Notifications /> {/* toujours monté */}
    </>
  );
}
```

🌬️ Transitions & comportements recommandés

Transition	Détails
Entrée	`slide-in-right` + `fade-in` sur 200–300 ms
Sortie	`fade-out` sur 150 ms
Stack	Espacement fluide (`gap-3`), max 4 visibles
Invites	pas de timer automatique, requièrent action
Réduction mouvement	si `prefers-reduced-motion`, passer en `opacity` simple

♿ Accessibilité

- Chaque toast possède un `role="status"` (ou `"alert"` pour les erreurs).
- Les messages ne volent jamais le focus.
- Bouton “fermer” accessible au clavier.
- Respect de la durée (ne jamais forcer un message court à disparaître avant 2 s).
- Animation désactivée si l’utilisateur a activé “réduire les animations”.

Exemple :

```
@media (prefers-reduced-motion: reduce) {
  * {
    transition-duration: 0.01ms !important;
    animation-duration: 0.01ms !important;
  }
}
```

🧪 Tests attendus

1. **Affichage d’une notif** lors d’un événement WS (`notif:new`).
2. **Disparition automatique** après la durée configurée.

3. **Action manuelle** ferme immédiatement.
4. **Empilement** (max 4) → la plus ancienne disparaît d'abord.
5. **Mode "reduce motion"** désactive transitions.
6. **Erreur WS** affiche une notification "assertive".
7. **Aucune fuite** de listeners quand le composant est démonté.

🔧 Bonus possibles (si temps)

- Ajout d'un **panneau d'historique** des notifs (persistées localement).
- **Animation sonore** courte (non bloquante) à la réception.
- Apparition contextuelle différente (ex : côté bas si mode jeu plein écran).
- Gestion d'un **canal "tournament"** séparé visuellement.

✅ Definition of Done (DoD)

- Composant `<Notifications />` intégré globalement.
- Animations d'entrée/sortie fluides (Framer Motion ou équivalent).
- Support WS temps réel (`notif:new`).
- Durées et couleurs adaptées au type de notif.
- Respect complet de l'accessibilité (aria, clavier, reduce motion).
- Aucun overlap, aucune fuite mémoire.
- Tests manuels validés (apparition, disparition, interaction).

🗓️ Semaine 4 – Tournois, Matchmaking & IA

🎯 Objectif global

- Jouer contre un **bot local** (offline) et **remplaçant** (si l'adversaire drop en ligne).
- IA **déterministe par niveau** (Easy/Medium/Hard/Insane) → reproductible.
- **Aucune saccade UI** : calcul IA dans un **Web Worker** (ou WASM plus tard).
- **Fair-play** : plafonner la vitesse/accélération de la raquette IA, bruit contrôlé.
- **Instrumentation** (fps IA, temps de décision, erreurs) pour tuning rapide.

🧱 Architecture côté client

```
/src/ai
  ai.worker.ts        ← boucle décisionnelle (off-main-thread)
  ai-controller.ts    ← façade TS (start/stop, setLevel, tick)
  physics.ts          ← modèles simples (rebonds, friction, clamp)
  difficulty.ts       ← presets (réaction, bruit, biais)
  pid.ts              ← contrôleur PID (ou PD) pour la raquette
```

Principe runtime

- Le **render** (Canvas/WebGL) reste sur le main thread.
- À chaque **tick** (ou à 60 Hz), on envoie au Worker un **état compressé** du jeu (positions, vitesses, tailles).
- Le Worker renvoie une **commande** "targetY / velocity" pour la raquette IA.
- Option : **SharedArrayBuffer** + Atomics pour zéro-alloc (si COOP/COEP activés). Sinon `postMessage`.

🔌 Contrat de messages (main ⇄ worker)

```
// -> worker
type AiInput = {
  t: number;              // timestamp
  seed: number;           // seed niveau (determinisme)
  level: "easy"|"medium"|"hard"|"insane";
```

```
  ball: { x:number; y:number; vx:number; vy:number };
  paddle: { y:number; h:number; speedMax:number };
  arena: { w:number; h:number; paddleX:number };
};

// <- main
type AiOutput = {
  t: number;              // echo
  targetY: number;        // position visée (centre raquette)
  v: number;              // vitesse désirée (pour lerp côté main)
};
```

🧠 Logique IA (simple, efficace)

1. **Prédiction d'impact**: extrapoler la balle jusqu'à `paddleX` avec **rebonds verticaux**.
2. **Contrôle**: un **PID** (ou PD) pousse la raquette vers `impactY`, limité par `speedMax`.
3. **Humanisation**: bruit gaussien léger sur `impactY` + **temps de réaction** (retard) et **zone morte** (ne bouge pas si delta<ε).
4. **Anti-omniscience**: sur *easy/medium*, la prédiction n'intègre pas instantanément un futur changement de spin/angle (biais).

Presets de difficulté (exemples à ajuster) :

- *Easy*: reactionDelay=120ms, noise=±18px, clamp speed 70%, "overshoot" léger.
- *Medium*: 60ms, ±10px, clamp 85%.
- *Hard*: 30ms, ±5px, clamp 95%, anticipation spin.
- *Insane*: 10ms, ±2px, clamp 100%, anticipation agressive (à réserver aux tests).

🧪 Exemple Worker minimal (TypeScript)

```
// ai.worker.ts
import { predictImpactY } from "./physics";
import { pidCreate } from "./pid";
import { levelPreset } from "./difficulty";

let pid = pidCreate({ kp: 0.9, kd: 0.22, ki: 0 });
let lastDecision = 0;
```

```
self.onmessage = (e: MessageEvent) => {
  const s = e.data as AiInput;
  const L = levelPreset[s.level];

  // throttle: simuler temps de réaction
  if (s.t - lastDecision < L.reactionMs) return;

  // prédiction (rebonds jusqu'à paddleX)
  const impactY = predictImpactY(
    s.ball, s.arena, s.arena.paddleX
  );

  // bruit contrôlé (gaussien tronqué)
  const noisy = impactY + gaussian(L.noisePx);

  // consigne PID sur position (erreur = cible - centre raquette)
  const center = s.paddle.y + s.paddle.h / 2;
  const vCmd = pid.update(noisy - center);

  // clamp vitesse et retour
  const v = Math.max(Math.min(vCmd, s.paddle.speedMax * L.speedFactor),
                     -s.paddle.speedMax * L.speedFactor);

  lastDecision = s.t;
  const out: AiOutput = { t: s.t, targetY: noisy, v };
  (self as any).postMessage(out);
};

// utils
function gaussian(std: number) {
  if (std <= 0) return 0;
  // Box-Muller
  const u = 1 - Math.random(), v = 1 - Math.random();
  return Math.sqrt(-2 * Math.log(u)) * Math.cos(2 * Math.PI * v) * std;
}
```

Main thread (extrait)

```
// ai-controller.ts
export class AIController {
  private w?: Worker;
  private lastOut?: AiOutput;

  start() {
    this.w = new Worker(new URL("./ai.worker.ts", import.meta.url), { type: "module" });
    this.w.onmessage = (e: MessageEvent<AiOutput>) => this.lastOut = e.data;
  }
```

```
  stop() { this.w?.terminate(); }

  tick(state: AiInput) {
    this.w?.postMessage(state);
  }

  // applique la vitesse à la raquette côté main
  apply(paddle: { y:number }, dt: number) {
    if (!this.lastOut) return;
    paddle.y += this.lastOut.v * dt;
  }
}
```

🧩 Physique & prédiction (rebonds simples)

- On extrapole `y` de la balle vers `paddleX` en tenant compte des rebonds haut/bas (miroir).
- Si spin/effets existent côté serveur, **ne pas** tricher : sur easy/medium on **n'anticipe pas** les changements tardifs.

🔒 Fair-play & anti-cheat

- **Client-bot** uniquement pour **offline** et **remplacement** en ligne si le **serveur l'autorise**.
- En mode en ligne, le **serveur reste autoritaire** (positions officielles). Le bot local propose une commande, mais le serveur peut la limiter/refuser.
  Clamp systématique de la vitesse/accélération de la raquette côté client **et** côté serveur.
- **Pas d'accès** du bot à des infos non visibles (ex. décision future de l'adversaire).

⚡ Performance & stabilité

- Worker dédié (et unique) pour l'IA → pas de blocage du thread UI.
- **Budget** : < 0.5 ms / tick sur laptop moyen (profilage à vérifier).
- Option **WASM** si besoin (Rust/C++) si on ajoute une IA plus lourde (recherche trajectoire/simulation).

- **Fallback** : si Worker indispo (environnements stricts), basculer en mode “cadencé” (moins de ticks IA).

🔔 UX & contrôle

- Sélecteur de **difficulté** dans l’écran “Solo” (persisté localStorage).
- Indicateur visuel discret “Bot: Easy/Medium/Hard”.
- Bouton **Pause** interrompt la boucle IA proprement.
- Sur *matchmaking*, si bot remplace : toast **“L’adversaire est temporairement remplacé par un bot (Medium)”**.

📊 Instrumentation (dev-only)

- Overlay dev (`?debug=ai`) :
  - fps IA, temps par décision, `impactY`, `targetY`, erreur PID.
- Logs throttlés : “decision @t=… v=… noise=…”
- Métriques envoyées côté serveur en **mode QA** (compter la difficulté choisie).

🧪 Tests (à automatiser au minimum)

1. **Déterminisme** : même seed + même inputs → même sorties par niveau.
2. **Budget temps** : 10k décisions < 100 ms en CI.
3. **Clamp** : `|v|` ≤ `speedMax * factor`.
4. **Prédiction** : impactY ∈ [0, arena.h].
5. **Stabilité** : pas d’oscillation PID (tune `kp/kd`).
6. **Fallback** : Worker tué → reprise propre sans crash.

✅ Definition of Done (DoD)

- IA fonctionnelle en **Worker**, intégrée au loop de rendu.
- **4 niveaux** documentés, reproductibles (seed).
- **Aucune saccade** mesurable sur UI.

- **Fair-play** garanti (clamp, pas d’omniscience).
- **Instrumentation** accessible (overlay dev).
- Tests unitaires clés verts.
- Petite doc `/docs/AI_CLIENT.md` (contrat messages + presets).

🗓️ Semaine 5 – Sécurité & Monitoring

🎯 Objectif global

- Donner au jeu et à l’UI une **cohérence visuelle** (ombres, lueurs, transitions).

- Ajouter des **effets temps réel** (particules, post-traitements légers) qui renforcent le feedback.
- Garantir **60fps stables** (mobile/desktop) et respecter `prefers-reduced-motion`.
- Zéro “glitter” superflu : chaque effet sert la **compréhension** de l’action.

🧱 Périmètre & architecture

```
/src/graphics
  fx-engine.ts        // orchestration des effets (horloge, layers, perf)
  particles.ts        // système particules (GPU-friendly)
  postfx.ts           // post-traitements légers (bloom soft, vignette)
  shaders/            // GLSL si WebGL, sinon Canvas2D fallback
  timing.ts           // easings, synchronisation sons
  presets.ts          // styles (neo→sobre ; arcade→punchy)
```

- **Rendu**: Canvas2D robuste par défaut. Si WebGL activé → activer shaders optionnels.
- **Layers** : `background` (grille/gradient) → `playfield` (balle/raquettes) → `fx-top` (particules, trails) → `UI`.
- **Mode safe** (mobile bas de gamme) : désactive postfx, garde particules low-cost.

✨ Catalogue d’effets (priorisés)

Feedback balle/raquette (gameplay)

- **Trail discret** de la balle (2–4 ghosts alpha dégressifs).
- **Impact sparks** (10–16 particules trianglaires, 120 ms) sur collision raquette/mur.
- **Shake micro** du playfield (8–12 px, 90 ms) sur point marqué.
- **Highlight** raquette active (glow doux) quand elle touche la balle.

Tous ces effets doivent être **désactivables** et réduits si `prefers-reduced-motion`.

Effets UI (cohérence app)

- **Transitions** de panneaux (chat, tournois, profil) : fade/slide 150–200 ms.
- **Boutons** : press `scale-95` + shadow-lift au hover.
- **Toasts/notifications** : slide-in right + shadow soft (cf. animations notifs).

- **Loaders** : skeletons pulsants (opacity) plutôt que spinners agressifs.

Post-traitement léger (optionnel WebGL)

- **Bloom très soft** sur la balle (seulement en zoom spectateur).
- **Vignette subtile** pour focaliser le regard (2–4 %).
- **Chromatic aberration** interdite (illisible). **Motion blur** limité à 1–2 samples si 120fps.

🧠 Règles d'ergonomie (non négociables)

- Les effets **n'aident jamais un seul joueur** (pas d'advantage visuel).
- Contraste garanti : la balle reste **toujours** la forme la plus lumineuse du playfield.
- Aucune animation > 150 ms pour l'UI (sauf modales 200–240 ms max).
- Respect strict de `prefers-reduced-motion`: passer en **fade only**.

```
@media (prefers-reduced-motion: reduce) {
  * { animation: none !important; transition: none !important; }
}
```

🧮 Perfs & budgets

- **60 fps** cible. Budget frame **< 4 ms** pour les effets cumulés.
- Particules : **pooling** (pas d'alloc runtime), max ~150 simultanées.
- Canvas2D : dessiner **triangles/segments** (pas d'images lourdes).
- WebGL :
  - **VAO/VBO** persistants, instancing pour particules,
  - **MSAA** si possible, sinon FXAA shader léger.
- Textures UI en **sprite atlas** (pas de multiples requêtes).
- **Frame-skipping** des trails sur devices lents (1 trail/2).

🧪 Système particules (Canvas2D ou WebGL)

API simple :

```
fx.spawn({
  kind: "spark",
  x, y,
  dir: angle, spread: Math.PI/3,
  speed: [120, 220], lifeMs: 160,
  color: "rgba(255,255,255,0.85)"
});
```

Boucle :

- Update (dt) : vie, vitesse (drag), gravité optionnelle (0).
- Draw : batch par kind (limiter les `fillStyle` changes).
- Recycle quand `life <= 0`.

🧪 PostFX (WebGL)

- **Bloom soft** = blur separable (2 passes) + add.
- **Vignette** = simple radial factor `f = smoothstep(1, 0.7, r)`.
- **Gamma** clampée (pas de whites blow-out).

Basculer dynamiquement en **Canvas-only** si WebGL indisponible.

🔊 Synchronisation audio (polish sensoriel)

- Impact → **click** court synchronisé (≤ 20 ms de latence).
- Point gagné → **thump** + shake court.
- Victoire → **sting** musical ≤ 1 s, volume maîtrisé.
- Volume des SFX **lié** aux préférences notifications.

♿ Accessibilité visuelle

- Mode **“réduit”** : pas de shake, pas de trails, seulement **fade**.
- Contrastes testés (AA) sur fond sombre (UI). Playfield = palette dédiée.
- Pas de flash > 3 Hz, pas de patterns stroboscopiques.

🔧 Intégration code (exemples)

Framer Motion (UI)

```
<motion.div
  initial={{ opacity: 0, y: 8 }}
  animate={{ opacity: 1, y: 0, transition: { duration: 0.18 } }}
  exit={{ opacity: 0, y: 8, transition: { duration: 0.12 } }}
/>
```

Shake contrôlé (Canvas)

```
const shake = { t:0, amp:0 };
function triggerShake(a=8, d=0.1){ shake.t=d; shake.amp=a; }
function applyShake(ctx, dt){
  if (shake.t<=0) return;
  shake.t -= dt;
  const k = shake.t<=0 ? 0 : (shake.t/0.1);
  ctx.translate((Math.random()-0.5)*shake.amp*k, (Math.random()-0.5)*shake.amp*k);
}
```

🧰 Panneau Dev (QA)

- Toggle **WebGL/Canvas**, particules on/off, postfx on/off.
- FPS, frame time, draw calls, particules actives.
- Screenshot rapide pour comparer.

✅ Definition of Done (DoD)

- Effets gameplay (trail, sparks, shake) implémentés + toggle **reduced motion**.
- UI transitions homogènes (durées/curves) + cohérence toasts/modales.
- 60fps sur laptop moyen et mobile récent ; fallback Canvas-only OK.

- Aucun flicker ni flash > 3 Hz ; balle toujours lisible.
- Panneau QA et presets d'intensité (low/medium/high).
- Tests manuels : point, collision, victoire, ouverture/fermeture UI.
- Doc courte `/docs/FX_GUIDE.md` (paramètres, budgets, toggles).

⚠️ Risques & garde-fous

- Overdesign → lisibilité en jeu diminue. **Remède** : tests utilisateurs rapides.
- Perfs WebGL variables. **Remède** : fallback Canvas + toggles.
- Motion sickness chez certains. **Remède** : `prefers-reduced-motion` + option in-app.
- Désynchronisation audio/visuel. **Remède** : planifier sons sur `timeOrigin` (WebAudio).

🗓️ Semaine 6 – Finalisation & Présentation

🎯 Objectif global

Créer une **expérience de jeu fluide, cohérente et présentable** pour la soutenance.
La "démo finale" doit :

- **fonctionner sans bug visible**, même hors connexion serveur,
- **montrer le savoir-faire** technique et UX de l'équipe,
- **donner envie de jouer** : son, feedbacks, transitions, rythme,
- être **autoportée** (lancement rapide, pas de setup complexe).

En clair : la *"touch finale Abdul"* — du gameplay net, rythmé et lisible.

🧱 Structure de la tâche

```
/src/game
  game-loop.ts          ← boucle principale stable (rafraîchissement, synchro)
  gameplay-tuning.ts    ← constantes physiques, vitesse, timing
  effects/              ← FX, sons, feedbacks
  ui/hud.tsx            ← score, round, countdown, overlay
  demo.ts               ← script de démo, automation des actions
```

⚙️ Les sous-tâches principales

Boucle de jeu lissée et stable

- Rafraîchissement fixe (`dt=1/60`) même si le `requestAnimationFrame` saute une frame.
- Delta clampé pour éviter les "sauts de balle" ou les accélérations incohérentes.
- Priorité : cohérence de la physique → visuel suit, jamais l'inverse.

  Exemple de stratégie :

  - `update()` = logique (position, collisions) avec `dt` fixe (physique pure).
  - `render(interp)` = interpolation visuelle pour lisser.

```
let acc = 0, last = performance.now();
function loop(now:number){
  acc += now - last; last = now;
  while (acc >= 1000/60) { update(1/60); acc -= 1000/60; }
  render(acc / (1000/60));
  requestAnimationFrame(loop);
}
```

Polish gameplay

C’est la *danse des détails*, invisible mais ressentie.

Rythme et inertie

- Ajustement fin des vitesses :
    - balle : pas plus rapide que la lisibilité humaine (~500–650 px/s)
    - raquettes : inertie légère (0.1–0.2 s de délai max)
- La balle accélère légèrement à chaque échange (gain de tension).

Feedbacks immédiats

- Raquette touche balle → son + flash court (10 ms).
- Point → *shake* + son “pop”.
- Fin de round → écran gris clair + “Ready?” + compte à rebours 3–2–1.

Équilibre IA / joueur

- Sur solo mode, Abdul vérifie que l’IA n’a **pas d’avantage de précision**.
- Si trop forte → ajout de micro-erreurs (latence ou bruit).

Caméra / cadrage

- Centrée sur la balle avec **limites** pour éviter le mal de mer.
- Zoom-out doux sur échanges longs ou rebonds extrêmes.

Collisions propres

- Correction des arrondis :
    - rebond latéral : renvoi symétrique (pas d’angle impossible).
    - clamp des overlaps → éviter le “téléport” de la balle dans la raquette.

HUD et transitions

Éléments essentiels :

- Score live (joueur gauche / droite).
- Nom ou pseudo affiché.
- Chrono ou compteur de rounds.
- Overlay “Pause”, “Match point”, “Victory”.
- Transitions fondues (`opacity 0.3s`, pas de jump cut).

Polish visuel :

- Couleurs cohérentes avec les FX (voir tâche “Effets graphiques”).
- Lueurs discrètes sur le score gagnant.
- Sons courts, compressés (`.ogg`) pour latence < 20 ms.

Démo automatisée (mode “Spectator”)

La démo doit pouvoir se **jouer toute seule**, pour la présentation :

- Une IA joue contre une autre IA (niveau medium/hard).
- Démarre automatiquement après 3 s sur l’écran titre.
- Relance un match dès qu’un score final est atteint.
- Si l’utilisateur touche une touche ou clique → reprend le contrôle.

Objectif :
Permettre de montrer le projet **sans interaction**, utile à la soutenance ou aux tests de performance.

Exemple :

```
function startDemo() {
  aiLeft.setLevel("medium");
  aiRight.setLevel("hard");
  game.reset();
  game.loop();
}
```

Nettoyage code et réglages finaux

- Supprimer les logs console / traces WS.
- Réduire la charge CPU (profil Chrome Performance).
- Vérifier `pause/resume` sans fuite mémoire (stopper tous les intervals).
- Vérifier compatibilité mobile (taille canvas auto-ajustée).
- Validation avec `prefers-reduced-motion` et `reduced-contrast`.

Tests & validation finale

Test	Description	Résultat attendu
FPS stable	60fps ±3 sur laptop milieu de gamme	✅
Inputs	Pas de “lag” visible entre mouvement raquette et visuel	✅
Collisions	Aucun rebond incohérent	✅
Sons	Volume et timing cohérents	✅
Restart	Reset d’un match = clean total des states	✅
Demo	Boucle de démo sans crash pendant 10 min	✅
Accessibilité	motion réduite / son coupé respectés	✅

Livrable final

- Dossier `/demo` contenant :
    - `index.html` → version standalone (no login, no serveur).
    - `assets/` (sons, shaders, sprites).
    - `demo-loop.ts` (boucle autonome).

    - `README_demo.md` (comment lancer localement).

- Script `npm run demo` → lance le mode spectateur automatique.

🧠 Bonus possibles

- Ajout d'un **mode ralenti** (`Shift` maintenu) pour montrer la précision.
- Un **effet "replay"** des 3 derniers points (slow + overlay score).
- Easter egg discret (logo 42 qui s'allume à 42 points gagnés 🧠).

✅ Definition of Done (DoD)

- Gameplay stable (aucune incohérence physique).
- 60fps constants sur laptop moyen.
- HUD complet (score, round, transition).
- Démo autonome (IA vs IA, restart automatique).
- Feedback audio/visuel complet.
- Accessibilité respectée (motion, audio).
- Prêt pour la soutenance (présentation fluide, impact immédiat).

Mehdi

🗓️ Semaine 1 – Reprise & Environnement

🎯 Objectifs de l'audit

1. **Menaces & périmètre** (threat model) : qui attaque quoi, comment, pourquoi.

2. **Hygiène immédiate** : dépendances, secrets, headers, CORS, JWT, rate-limit.
3. **Chaîne d'approvisionnement** (supply chain) : dépendances, images Docker, CI.
4. **Plan de remédiation** chiffré (impact/effort) + checklists récurrentes.

🧭 Périmètre à couvrir

- **Frontend** (SPA TS + Tailwind) : stockage local, XSS, CSP.
- **Backend** (Fastify/Node + SQLite) : auth, validation, injection, erreurs.
- **WebSockets** : auth, sujets, rate-limit, heartbeat.
- **Nginx** (reverse proxy TLS).
- **Docker / Compose** : isolation, users non-root, secrets.
- **CI/CD** : fuite de tokens, scans auto.
- **Données** : RGPD minimal (logs, rétention).

🧠 Threat modeling express (STRIDE)

- **S**poofing : JWT volé, 2FA contourné → sign subprotocol WS + rotation.
- **T**ampering : messages WS altérés → signature serveur / validation schéma.
- **R**epudiation : logs d'audit (who/when/from).
- **I**nformation disclosure : CORS laxiste, erreurs verbeuses, buckets publics.
- **D**enial of Service : spam WS/HTTP → rate-limit + quotas.

- **E**levation of Privilege : scopes OAuth/roles mal gérés.

Livrable : **diagramme simple** (contexte, flux HTTP/WS, trust boundaries) + table risques (Probabilité × Impact).

⚡ Contrôles prioritaires (concrets)

Secrets & config

- **Interdiction** de secrets en repo.
- `.env.example` versionné, **.env non versionné**.
- En prod : **Docker secrets** / variables chiffrées.

```
# vérif rapide dans le repo
git ls-files | xargs grep -nE '(AKIA|SECRET|BEGIN RSA|password=)' || true
```

Dépendances & SBOM

- **npm audit** + **pnpm audit** (si pnpm).
- **Semgrep** (SAST) + **Trivy** (deps & images) + **Syft** (SBOM).

```
npx semgrep --config p/owasp-top-ten .
trivy fs --severity HIGH,CRITICAL .
trivy image --severity HIGH,CRITICAL mechard/ft_transcendence:* || true
syft . -o json > sbom.json
```

Nginx / TLS / headers

- TLS 1.2+ uniquement, HSTS, redirection 80→443.
- **Security headers** (CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy).

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-Frame-Options "DENY" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Permissions-Policy "camera=(), microphone=(), geolocation=()" always;
```

```
# CSP minimale pour SPA (à durcir)
add_header Content-Security-Policy "default-src 'self'; img-src 'self' data:; script-src 'self';
style-src 'self' 'unsafe-inline'; connect-src 'self' wss:;" always;
```

Auth & JWT (HTTP & WS)

- **Algorithme** : `RS256` / `EdDSA`, pas de `none`.
- **Durée courte** (15–30 min), **refresh** séparé (httpOnly, Secure, SameSite=strict).
- **Scopes**/roles dans le token.
- WS : **JWT dans le subprotocol** (pas en query), rate-limit commandes.
- 2FA : TOTP (time-based), backup codes, **anti-bruteforce**.

Validation & sérialisation (Backend)

- **Schéma zod/yup** pour toutes les entrées (HTTP & WS).
- **Sanitization** et **output encoding**.
- Messages d'erreur **ne** divulguent **rien** de technique en prod.

CORS & cookies

- `Access-Control-Allow-Origin` = **liste blanche** (pas `*`).
- Cookies : `Secure`, `HttpOnly`, `SameSite=strict`.
- Pas de **LocalStorage** pour tokens d'accès (risque XSS) → cookies httpOnly.

Rate-limiting & anti-abuse

- HTTP : **/auth**, **/login**, **/reset** → limite agressive (ex. 5/min IP + user).

- WS : **invite.send**, **chat.send** → token bucket par user.
- **Slowdown** sur tentatives répétées.

Logs & audit

- **Corrélation** (request id), IP, userId, endpoint, statut, durée.
- **Audit events** : login, 2FA change, reset, rôle modifié, match create/join.
- Rétention courte en dev, conforme en prod (RGPD).

Docker & images

- Base **slim/alpine**, **USER non-root**.
- Read-only FS, pas d'écriture hors volumes.
- `HEALTHCHECK` défini.
- **Trivy** à chaque build CI.

CI/CD

- Secrets CI dans **vault/vars protégées**.
- **Scan automatique** : Semgrep, Trivy, npm audit → PR "Security" avec rapport.
- **Refuse** le merge si CRITICAL non accepté.

🧪 DAST (smoke test)

- **OWASP ZAP Baseline** sur l'instance dev.

docker run -t owasp/zap2docker-stable zap-baseline.py -t https://dev.ft -r zap.html

- Vérifier : pages publiques sans fuites, cookies, headers, formulaires.

🧩 Spécifiques WebSocket

- Auth handshake : `Sec-WebSocket-Protocol: bearer, <JWT>`.
- **Ping/pong** + timeout → fermeture + backoff.
- **Filtrage serveur** par scope (pas d'événement global).
- **Schema validation** des messages entrants + **taille max** (éviter DoS).

🔐 Mots de passe & stockage

- **Argon2id** (ou scrypt) avec params conservateurs (temps ~100ms).
- **Pepper** côté serveur (secret) + salt unique.
- Aucune possibilité de récupérer un mot de passe → **reset token** court (10–15 min), **one-time use**.

📄 Livrables de l'audit

1. **SECURITY_AUDIT.md**
    - Contexte, diagramme, STRIDE, risques priorisés.
    - Liste contrôles appliqués, non-conformités, plan de remédiation (Tableau RICE/ICE : Impact × Effort).
2. **Rapports** : `semgrep.sarif`, `trivy.txt`, `zap.html`, `sbom.json`.

3. **Policies** : `SECURITY_HEADERS.md`, `JWT_POLICY.md`, `CORS_POLICY.md`, `RATE_LIMITING.md`.

4. **Issues GitHub** créées par priorité (P0/P1/P2) avec owner + échéance.

✅ Definition of Done (DoD)

- Threat model documenté + risques classés.
- Scans SAST/DAST/Deps exécutés et archivés en CI.
- Nginx headers + TLS + CSP en place.
- Auth JWT (durées, scopes), 2FA, cookies sûrs.
- CORS whitelist, rate-limit HTTP/WS.
- Zéro secret en repo, Docker non-root + scans images.
- Checklists “pre-merge” & “pre-release” ajoutées.

🧰 Snippets utiles

Fastify helmet (headers)

```
await app.register(import('@fastify/helmet'), {
  contentSecurityPolicy: false, // géré par Nginx
  referrerPolicy: { policy: 'strict-origin-when-cross-origin' }
});
```

Rate-limit

```
await app.register(import('@fastify/rate-limit'), {
  max: 100, timeWindow: '1 minute', allowList: ['127.0.0.1']
});
```

Validation schéma (zod)

```
const InviteSchema = z.object({
  to: z.string().uuid(), mode: z.enum(['classic','tournament'])
});
```

Cookie sécurisé

```
reply.setCookie('rt', token, { httpOnly:true, secure:true, sameSite:'strict', path:'/auth/refresh'
});
```

🗓️ Semaine 2 – Authentification & Utilisateurs

🎯 Objectif global

- **JWT d’accès court** (stateless) + **refresh rotatif** (stateful, révoquable).
- **2FA TOTP** (RFC 6238) avec **codes de secours** hachés.

- Cookies **httpOnly + Secure + SameSite=strict**, CORS en liste blanche.
- Auth **WebSocket** via **subprotocol bearer** (pas de token en query).
- Outils d'admin : **révocation**, **liste d'appareils**, **journal d'audit**.

🧱 Modèle d'auth (vue d'ensemble)

Tokens

- **Access JWT** (15 min) : signé **RS256** ou **EdDSA** ; claim minimal (`sub`, `iat`, `exp`, `scope`, `sid`, `jti`).
- **Refresh rotatif** (7–30 jours) : opaque **random 256 bits**, stocké DB (hashé) avec **rotation** à chaque refresh (reuse-detection).

États

- Table **sessions** (par appareil) + table **refresh_tokens** (par rotation).
- 2FA activé → **step-up** : login = 1) mot de passe → 2) TOTP ou code secours.

Flux

1. `POST /auth/login` → set cookie `rt` (refresh, httpOnly) + renvoie `access` (header/body).
2. `POST /auth/2fa/verify` (si requis) → même sortie.
3. `POST /auth/refresh` → rotation refresh + nouveau access.
4. `POST /auth/logout` → révoque session (invalidate refresh chain).

🗄️ Schéma de données (exemple minimal)

users(id, email, pwd_hash, twofa_enabled, twofa_secret_enc, created_at)
backup_codes(user_id, code_hash, used_at)
sessions(id, user_id, ua, ip_hash, created_at, revoked_at)
refresh_tokens(id, session_id, token_hash, prev_id, created_at, used_at, revoked_at)
audit(id, user_id, event, ip, ua, ts, meta_json)

- `twofa_secret_enc` chiffré (AES-GCM) avec une **clé serveur**.
- `backup_codes` : 8–10 codes, **bcrypt/argon** hashés, usage unique.

🔐 Détails de sécurité

Mots de passe

- **Argon2id** (temps ~100–200 ms), pepper serveur.
- Politique : longueur ≥ 10, rejet des plus communes.

JWT d’accès

- Signature **asymétrique** (clé privée côté backend, publique partageable).
- Revendications :
  - `sub`: userId
  - `sid`: sessionId (lie l’access à une session révoquable)
  - `jti`: id unique (anti-rejeu ciblé)
  - `scope`: ex. `["user:read","ws:user"]`
  - `exp`: +15 min
- **Pas de PII** dans le JWT.

Refresh rotatif + reuse detection

- Refresh = **random 32 bytes**, **jamais** en JWT; stocké **hashé** (bcrypt/argon).
- À chaque `/auth/refresh` :
  - Invalider l’ancien, créer un **nouveau** (chaînage `prev_id`).

    - Si un **ancien** refresh est présenté → **révoquer la session entière** (signe d'exfiltration).

Cookies & CORS

- Cookie `rt`: `httpOnly`, `Secure`, `SameSite=strict`, `Path=/auth/refresh`.
- CORS : `Origin` en **whitelist** uniquement (dev/prod séparés).
- CSRF : le refresh est **même-site** + endpoint dédié → risque réduit. Si besoin, `Double-Submit` CSRF pour actions sensibles.

2FA TOTP (RFC 6238)

- Algo : **SHA-1**, période 30s, digits 6 (compatibilité large).
- **Enrollment** :
    - Générer `secret` (Base32), stocker **chiffré**.
    - Afficher QR (otpauth URI).
    - Exiger 1 code **valide** pour activer.
    - Générer 10 **codes de secours** (hashés).
- **Vérif** : fenêtre ±1 pas (30–60s) max (synchro temps).
- **Step-up** : exiger TOTP lors :
    - login,
    - actions sensibles (changer 2FA, e-mail, mot de passe),
    - création de tokens longues durées.

WebSockets

- Handshake :
    - Client : `Sec-WebSocket-Protocol: bearer, <ACCESS_JWT>`

    - Serveur : vérifie signature/exp, que `scope` inclut `ws:*`, lie `sid`.
- Heartbeat + **fermeture** si expiré/non rafraîchi.
- **Interdire** le JWT dans l'URL (logs, leaks).

Anti-abus & verrouillages

- Login/2FA rate-limit : p.ex. 5/min par IP + 5/min par `email`.
- **Progressive delay** (backoff) au-delà.
- **Lock** court (5–10 min) si 20+ échecs consécutifs, **sans** divulguer si email existe.
- **Device list** : sessions actives visibles + bouton **Revoquer**.

Journaux & alertes

- Journaliser : login ok/ko, 2FA ok/ko, refresh reuse, revoke, reset.
- Option : e-mail d'alerte si connexion d'un nouvel appareil.

🧪 Endpoints (Fastify/Node – pseudo-impl)

```
// login
app.post('/auth/login', async (req, reply) => {
  const { email, password } = req.body;
  const user = await findUserByEmail(email);
  await verifyPasswordOrFail(user, password); // argon2id + pepper

  const session = await createSession(user.id, req);
  const need2fa = user.twofa_enabled;

  if (need2fa) {
    return reply.send({ step: '2fa_required', sessionId: session.id });
  }

  const { access, refresh } = await issueTokens(user, session.id);
  setRefreshCookie(reply, refresh);
  reply.send({ access });
});
```

```
// 2FA verify
app.post('/auth/2fa/verify', async (req, reply) => {
  const { sessionId, code } = req.body;
  const session = await getSession(sessionId);
  const user = await getUser(session.user_id);

  await verifyTOTPOrBackup(user, code); // window ±1, backup hash
  const { access, refresh } = await issueTokens(user, session.id);
  setRefreshCookie(reply, refresh);
  reply.send({ access });
});

// refresh (rotation + reuse detection)
app.post('/auth/refresh', async (req, reply) => {
  const rt = req.cookies.rt;
  const rec = await getRefreshRecordByHash(hash(rt));
  if (!rec || rec.revoked_at) return reply.code(401).send();

  if (rec.used_at) {
    // reuse detected → revoke whole session
    await revokeSession(rec.session_id, 'refresh_reuse');
    clearRefreshCookie(reply);
    return reply.code(401).send();
  }

  await markUsed(rec.id);
  const { user, session } = await ctxFrom(rec.session_id);
  const { access, refresh } = await issueTokens(user, session.id);
  rotateRefresh(rec.id, refresh); // create new, link prev_id
  setRefreshCookie(reply, refresh);
  reply.send({ access });
});

// logout
app.post('/auth/logout', async (req, reply) => {
  const sid = getSidFromAccess(req); // ou via cookie si stateful
  await revokeSession(sid, 'logout');
  clearRefreshCookie(reply);
  reply.send({ ok: true });
});
```

Helpers — cookies

```
function setRefreshCookie(reply, token) {
  reply.setCookie('rt', token, {
    httpOnly: true, secure: true, sameSite: 'strict',
    path: '/auth/refresh', maxAge: 30*24*3600
```

```
  });
}
function clearRefreshCookie(reply) {
  reply.clearCookie('rt', { path: '/auth/refresh' });
}
```

TOTP (enrollment minimal)

```
// generate
const secret = crypto.randomBytes(20);              // 160 bits
const base32 = base32Encode(secret);
const uri =
`otpauth://totp/ft_transcendence:${email}?secret=${base32}&issuer=ft_transcendence`;

// verify
function verifyTotp(secretBytes, code) {
  const nowStep = Math.floor(Date.now()/1000/30);
  for (const step of [nowStep-1, nowStep, nowStep+1]) {
    if (totp(secretBytes, step) === code) return true;
  }
  return false;
}
```

🔒 Cas particuliers importants

- **“Trust this device 30j”** : ce n’est **pas** ignorer 2FA ; c’est marquer la **session** comme 2FA-verified et limiter step-up aux actions sensibles.
- **Reset 2FA** : nécessite preuve forte (email + délai de sécurité + codes secours).
- **Heure système** : s’assurer d’un NTP correct (drift TOTP sinon).
- **Rotation des clés JWT** : planifier (kid + JWKS), rouler sans downtime.

✅ Definition of Done (DoD)

- Access JWT 15 min (RS256/EdDSA) ; refresh rotatif + reuse detection.
- Cookies `rt` httpOnly/Secure/SameSite=strict ; CORS whitelist.

- 2FA TOTP complet : enrollment, vérif, codes secours (hashés).
- WS via subprotocol `bearer` ; fermeture si token expiré.
- Rate-limit login/2FA ; verrouillage temporaire.
- Pages **Appareils** (sessions actives) + **Révoquer**.
- Logs d'audit (login, 2FA, refresh, revoke) + alertes basiques.
- Jeux de tests : success, 2FA on/off, refresh normal, **refresh reuse**, logout, WS.

🧪 Plan de tests (essentiels)

1. **Login sans 2FA** → access + cookie rt.
2. **Login avec 2FA** → step-up requis ; codes secours OK.
3. **Refresh normal** → rotation OK, ancien marqué `used_at`.
4. **Refresh reuse** → 401 + session totalement révoquée.
5. **WS** : connexion avec access valide → OK ; expiré → fermeture.
6. **Rate-limit** : 10 tentatives → backoff/lock.
7. **Révocation session** → access futurs refusés (via `sid`).

🧯 Menaces & parades (résumé)

- **Vol de refresh** → *reuse-detection + revoke chain*.
- **XSS** → pas de token en `localStorage`, cookies httpOnly.

- **Leaking JWT** → pas d'URL, pas de logs ; `kid` + rotation planifiée.
- **Brute force TOTP** → rate-limit + lock doux.
- **CSRF refresh** → cookie same-site strict + endpoint dédié.

📎 Mini-checklist "PR prête à merger"

- `JWT_POLICY.md` et `/auth/README.md` documentés.
- Clés privées **hors repo**, gestion via secrets/vars CI.
- TOTP secret **chiffré au repos** ; codes secours **hashés**.
- Tests d'intégration passent (incluant *reuse detection*).
- Lint Semgrep sur auth OK.

🎯 Objectif global

- Garantir que **tous les messages** (chat, WS, notifications, tournois, invites, logs) soient :

    1. Authentifiés (signés par un utilisateur légitime),
    2. Validés et filtrés (contenu sûr, taille limitée, typé),
    3. Non falsifiables (pas d'usurpation d'identité),
    4. Non exploitables (aucune injection, XSS, spam flood),
    5. Journalisés (audit trail minimal et conforme).

En clair : **zéro message "pourri" ne doit polluer le système**, que ce soit via le chat, une API REST, ou le canal WebSocket.

🧱 Périmètre technique

```
/src/backend/ws/
  message.gateway.ts      ← point d'entrée WebSocket (auth + routing)
  message.schema.ts       ← schémas zod/yup pour validation stricte
  message.sanitizer.ts    ← nettoyage HTML, liens, scripts
  message.store.ts        ← persistance / audit
  message.rate-limit.ts   ← anti-spam & throttle
  message.policy.ts       ← autorisations & contexte (room, dm, role)
```

🔐 Authentification & Contexte

- Chaque message WebSocket **doit être signé** par un JWT d'accès encore valide.
  → Le serveur lie chaque socket à un `user_id` via `sid` (session_id).

- Le JWT n’est **jamais** transporté dans le message lui-même (évite spoof).
  → Authentification faite **au handshake** (`Sec-WebSocket-Protocol: bearer, <ACCESS_JWT>`).
- Ensuite, tout message est reçu dans un **contexte utilisateur** déjà validé :

wsContext = { userId, sessionId, roles, scopes, rooms: Set<string> }

Aucune donnée “userId” envoyée par le client n’est crue.

🧩 Validation stricte des messages (schémas)

Fastify + Zod pour les types WebSocket :

```
import { z } from "zod";

export const MessageInSchema = z.object({
  t: z.literal("chat:send"),
  room: z.string().uuid(),
  content: z.string().trim().min(1).max(500),
});

export const InviteSchema = z.object({
  t: z.literal("invite:send"),
  to: z.string().uuid(),
  mode: z.enum(["classic", "tournament"]),
});
```

→ Toute payload reçue est validée **avant exécution** :

```
const result = schema.safeParse(message);
if (!result.success) return socket.close(4001, "invalid schema");
```

Cette validation évite :

- **Prototype pollution**,
- **Command injection**,
- **Data overflow** (flood de 100 Mo),
- **Structure altérée** (t falsifié).

🧼 Sanitization du contenu

Le **contenu textuel** (messages chat) est nettoyé pour empêcher le XSS ou le markdown malicieux.

Exemples :

- `<script>` supprimé,
- `javascript:` URI bloquée,
- HTML réduit à un *subset sûr* (`<b>`, `<i>`, `<code>`, `<a href>` avec rel=noopener).

Code typique :

```
import sanitizeHtml from 'sanitize-html';

export function sanitizeMessage(raw: string) {
  return sanitizeHtml(raw, {
    allowedTags: ['b', 'i', 'strong', 'em', 'a', 'code'],
    allowedAttributes: { 'a': ['href'] },
    allowedSchemes: ['https', 'mailto'],
  });
}
```

Et avant stockage :

```
msg.content = sanitizeMessage(msg.content);
```

⛔ Filtrage et modération

Mehdi mettra un système de **préfiltrage** :

- **Anti-spam** : messages identiques ou trop rapides bloqués.
  - max 5 messages / 5 s → “slow mode” automatique.
- **Anti-mention flood** : max 3 mentions par message.
- **Blacklist de mots** configurable (`bad_words.txt`).
- **Lien scanning** (regex ou API) : détecte URLs extérieures pour warning.
- **Flagging automatique** : message suspect → log modération + notif admin.

```
if (msg.content.length > 500) reject("too_long");
if (recentMsgs.sameUser > 5/5s) reject("spam");
if (containsBannedWord(msg.content)) flag(msg);
```

⚙️ Politique d’accès / Autorisations

- **Room** : utilisateur doit appartenir à la room.
  → `if (!context.rooms.has(roomId)) reject("unauthorized_room")`.
- **DM** : les deux utilisateurs doivent être “amis” ou avoir autorisation mutuelle.

- **Admin** / **System** messages : seulement émis par backend signé (`t:system:…`).
- **Tournoi** / **match** : seuls les joueurs concernés peuvent poster dans le salon.

🚧 Rate limiting & DoS

Niveau WebSocket

Limiter la fréquence des messages par socket :

```
const last = rateMap.get(userId) || { t:0, n:0 };
if (Date.now() - last.t < 2000 && ++last.n > 5) close(userId, "rate_limit");
else rateMap.set(userId, { t:Date.now(), n:last.n });
```

Niveau serveur

Limiter le nombre de connexions WS simultanées / IP :

- max 3 sockets/user, 10/IP.
- `maxPayload: 8 KB` dans la config WS → anti-flood binaire.

🧾 Journalisation et traçabilité

Chaque message stocké inclut :

messages(id, room_id, user_id, content, created_at, edited_at, flagged, ip_hash)

- `ip_hash` = SHA256(IP + pepper).
- `flagged` = booléen, pour modération.
- Log en parallèle dans `audit` pour les tentatives rejetées (type, raison).

🔒 Intégrité et authentification inter-services

Si les messages transitent par plusieurs microservices (ex. `chat-service` → `notif-service`) :

- Chaque message signé avec une **clé HMAC interne** (`X-Service-Signature`).
- Vérif côté récepteur (évite spoof interne).

```
const sig = crypto.createHmac("sha256", SHARED_KEY).update(body).digest("hex");
if (req.headers["x-service-signature"] !== sig) return 401;
```

📡 Chiffrement & stockage

- Pas de chiffrement bout-à-bout (non demandé par le sujet 42), mais :
    - **Transport sécurisé** : HTTPS + WSS only.
    - **Logs et DB** chiffrés au repos (volume-level, Docker secret).
- Si un futur “mode privé” apparaît → prévoir clé publique utilisateur (hybride RSA/AES).

🧪 Tests & vérifications

Cas	Attendu
Message valide	200 / diffusé
Message >500 chars	rejet (payload too large)
HTML/script	purgé / échappé
6 messages/2s	socket close rate-limit
Non-auth WS	handshake refused
User hors room	403 reject
Banned word	message flaggué
Lien suspect	alert modération
X-Service spoof	401 reject

🗒️ Documentation & livrables

Docs à fournir :

1. `MESSAGE_SECURITY_POLICY.md`

    - règles de validation, filtrage, stockage, logs.
2. `WS_SECURITY_CHECKLIST.md`
    - handshake, auth, rate-limit, payload, size.
3. `SANITIZATION_TESTS.md`
    - exemples d'entrée brute → sortie nettoyée.
4. `MODERATION_GUIDE.md`
    - gestion flags / suppression / audit.

Rapports CI :

- `semgrep --config p/owasp-top-ten .`
- `zap-baseline.py -t https://localhost:8080/chat`

✅ Definition of Done (DoD)

- Validation stricte (zod/yup) sur chaque message entrant.
- Sanitization complète (XSS-safe).
- Auth handshake WS obligatoire, `userId` injecté par serveur.
- Rate-limit global + par utilisateur.
- Journaux des refus / flagging modération.
- Tests unitaires XSS / spam / flood.
- Docs & policies versionnées dans `/docs/security/`.

⚠️ Menaces contrées

Attaque	Paré par
XSS via message HTML	`sanitize-html` strict
Flood / DoS	Rate limit + taille max

Spoof userId	Contexte JWT serveur-only
Injection WS	Validation zod + schema
Cross-room leak	Room policy stricte
Spam / mention abuse	Slow mode + anti-duplicate
Data exfil (service interne)	HMAC inter-service
Stockage persistant XSS	sanitize avant save

🗓️ Semaine 4 – Tournois, Matchmaking & IA

🎯 Objectif global

- Maintenir une **zone de flow** (ni trop facile, ni injuste) en ajustant l'IA.

- S’adapter **à court terme** (pendant le match) et **à long terme** (profil joueur).
- Rester **vérifiable** (télémétrie, logs), **fair-play** (pas d’avantage caché), **RGPD-safe**.

🧭 Cadre : deux boucles d’adaptation

1. **In-match DDA (Dynamic Difficulty Adjustment)** – pilotage du bot à chaud, via paramètres IA (réaction, bruit, clamp vitesse).
2. **Entre matchs (profil joueur)** – mise à jour d’un rating (type Elo/TrueSkill) + sélection de preset via **bandit contextuel**.

L’in-match est **lent et borné** (petits pas, limites strictes).
Le hors-match est **plus ample** (on repositionne la difficulté de départ).

📡 Signaux mesurés (sans PII)

Fenêtre glissante 30–60 secondes :

- Diff d’**écart au score** (Δscore).
- **Rally count / échanges moyens** (qualité du point).
- **Hit accuracy** IA vs joueur (contacts utiles/ratés).
- **Temps de réaction** estimé joueur (latence entrée → contact).
- **Vitesse moyenne** de la balle, durée points, fautes directes.

Entre matchs :

- Résultats (W/L), marge, durée, abandon.
- Série récente (streak), variance performances.

🔧 Leviers d’ajustement (IA côté client)

Paramètres IA (exposés et logués) :

- `reactionMs` (temps de réaction simulé)

- `noisePx` (bruit sur la prédiction d'impact)
- `speedFactor` (pourcentage de la vitesse max de la raquette)
- `anticipation` (prend en compte/ignore certains spins)
- `deadZonePx` (zone morte : ne bouge pas si delta faible)

Bornes strictes (exemple) :

- `reactionMs ∈ [10, 150]`
- `noisePx ∈ [0, 20]`
- `speedFactor ∈ [0.70, 1.00]`
- Pas plus de **1 cran** d'ajustement toutes les **10 s**.

🧠 Algorithmes (simple → robuste)

Score de tension (in-match)

On calcule un **Flow Score** sur une fenêtre glissante :

```
flow = w1*norm(rally_len) + w2*norm(ball_speed) + w3*norm(time_per_point)
     - w4*norm(score_diff) - w5*norm(unforced_errors)
cible: flow ≈ 0.6  (entre 0 et 1)
```

- Si `flow < 0.45` → match trop facile pour l'IA → **augmenter** difficulté très légèrement.
- Si `flow > 0.75` → match trop dur → **baisser** difficulté légèrement.

Contrôleur PI (lent) :

```
err = targetFlow - flow
integral = clamp(integral + err, -Icap, Icap)
delta = Kp*err + Ki*integral
apply(delta)  // convertit en petits pas sur reaction/noise/speed
```

Sélection de preset au coup d’envoi (entre matchs)

- On maintient un **rating joueur** (Elo ou **TrueSkill**).
- On a 4–6 **presets** IA (`easy, medium, hard, pro, insane`), chacun = vecteur param.
- On choisit via **bandit contextuel** (ex. LinUCB/Thompson) avec contexte : rating, dernière marge, device (mobile/desktop), latence réseau.
- Reward = “qualité du match” (flow moyen + résultat serré).

🎯 Effet : démarrage déjà cohérent, l’in-match fait des **affinages fins**.

🧱 Architecture

```
/src/ai
  ai.worker.ts          ← logique IA (déjà en worker)
  adaptive.ts           ← contrôleur DDA (PI + bornes)
  presets.ts            ← presets IA (vectors param)
  rating.ts             ← Elo/TrueSkill
  bandit.ts             ← LinUCB simple (sélection preset)
  telemetry.ts          ← collecte + export métriques (agrégées)
```

Client

- Mesure signaux, calcule flow, pilote DDA → ajuste `reactionMs/noise/speedFactor` **dans les limites**.
- En fin de match : envoie au serveur **métriques agrégées** (anonymisées).

Serveur

- Met à jour **rating** + **bandit** (profil joueur).
- Renvoie preset recommandé pour le prochain match.
- Stocke uniquement agrégats (pas de trajectoires brutes), **opt-out** possible.

🔒 Sécurité, fair-play, anti-abus

- **Transparence** : curseur “Difficulté dynamique” ON/OFF (par défaut ON), label “IA adaptative active”.

- **Garde-fous** : pas de “triche magique” (la balle/physique restent identiques); on **limite uniquement** les capacités de la raquette IA.

- **Anti-sandbagging** : si le joueur perd volontairement pour baisser la difficulté (marques incohérentes, séries anormales), on **gèle** l’adaptation 60 s et on remonte légèrement la base au match suivant.

- **Vie privée** : on ne stocke **aucun input cru** ; seulement des agrégats numériques (durées, moyennes).

- **Reproductibilité** : seed loguée, presets versionnés (pour démos et QA).

🧩 Snippets (TypeScript)

Contrôleur d’adaptation in-match

```
// adaptive.ts
type IAParams = { reactionMs:number; noisePx:number; speedFactor:number;
anticipation:boolean; };

export class AdaptiveController {
  private target = 0.6;
  private integ = 0;
  constructor(
    private params: IAParams,
    private apply: (p: IAParams)=>void,
  ) {}

  step(flow: number, dtSec: number) {
    const err = this.target - flow;
    this.integ = clamp(this.integ + err*dtSec, -0.5, 0.5);
    const delta = 0.35*err + 0.05*this.integ; // Kp, Ki doux

    // mapping doux → petits pas bornés
    const dReact = -delta * 40;   // flow bas -> +difficulté -> -reactionMs
    const dNoise = -delta * 6;    // flow bas -> -noise
    const dSpeed =  delta * 0.04; // flow bas -> +speedFactor

    const next: IAParams = {
      reactionMs: clamp(this.params.reactionMs + dReact, 10, 150),
      noisePx:    clamp(this.params.noisePx    + dNoise,  0, 20),
      speedFactor:clamp(this.params.speedFactor+ dSpeed, 0.70, 1.00),
      anticipation: this.params.anticipation,
    };

    // hysteresis: n'applique que toutes les 10 s max
```

```
    if (shouldApplyNow()) {
      this.params = next;
      this.apply(this.params);
    }
    return this.params;
  }
}

function clamp(x:number,a:number,b:number){ return Math.max(a, Math.min(b, x)); }
function shouldApplyNow(){ return performance.now()%10000 < 50; } // ex: ~10s
```

Choix de preset (début de match)

```
// bandit.ts (LinUCB simplifié)
export function pickPreset(ctx: number[], arms: IAParams[]): number {
  // ctx: [rating_norm, mobile_flag, avg_margin_norm, latency_norm]
  // … conserver θ par bras en mémoire; calculer upper confidence bound
  // renvoyer l'index du bras gagnant
  return argmax(arms.map((_,i) => estReward(i,ctx) + alpha*uncertainty(i,ctx)));
}
```

🧪 Tests attendus

1. **Stabilité** : pas de yo-yo des paramètres (variation < 20% par minute).
2. **Efficacité** : partant d'un preset trop dur/facile, le flow converge vers 0.6 en < 90 s.
3. **Bornes** : jamais hors limites (réaction, bruit, speed).
4. **Éthique** : la physique **ne change pas** (balle identique).
5. **Anti-sandbagging** : perte volontaire → adaptation gelée.
6. **Replay QA** : seed + presets reproduisent un match identique.
7. **Opt-out** : désactivée → l'IA reste au preset fixe, aucune adaptation.

📊 Télémétrie (dev & prod)

- `flow_avg`, `flow_std`, `delta_params/min`, `rally_mean`, `margin`, `ai_difficulty_at_start/end`.
- Dashboard Grafana simple pour tuning (Mehdi + Abdul).

- Export **agrégé** toutes les 60 s (ou fin de match) → serveur.

✅ Definition of Done (DoD)

- In-match DDA **lent et borné** (PI), appliqué aux **seuls** paramètres IA.
- Sélection **entre matchs** via rating + bandit contextuel.
- Garde-fous : anti-sandbagging, opt-out, transparence UI.
- Télémétrie agrégée + dashboard minimal.
- Tests unitaires + scénarios QA (convergence, bornes, stabilité).
- Doc `/docs/AI_ADAPTIVE.md` (signaux, algos, bornes, privacy).

🗓️ Semaine 5 – Sécurité & Monitoring

🎯 Objectif global

- **2FA TOTP** (avec codes de secours), **cookies sûrs**, **rate-limit**.

- **OAuth2 “42”** en **Authorization Code** (état/PKCE, callback sécurisé, liaison de compte).
- **Audit sécurité complet** (SAST/DAST, headers, CSP, supply chain, CI bloquante sur CRITICAL).
- **Docs & dashboards** : procédures, politiques, métriques.

🧱 Architecture (vue d’ensemble)

```
frontend (SPA)
  └─ /auth → déclenche OAuth42 → redirige vers backend /auth/42/login

backend (Fastify/Node)
  ├─ /auth/42/login      (redirige vers provider 42 + state/PKCE)
  ├─ /auth/42/callback   (échange code→token, récup userinfo, link)
  ├─ /auth/2fa/enroll    (QR, secret, activation)
  ├─ /auth/2fa/verify    (TOTP / codes de secours)
  ├─ /auth/refresh       (refresh rotatif + reuse detection)
  ├─ /auth/logout        (révocation session)
  └─ middleware WS       (subprotocol bearer, scopes)
```

Sessions :

- **Access JWT** (15 min, RS256/EdDSA, `sub`, `sid`, `scope`, `exp`),
- **Refresh opaque rotatif** en cookie **httpOnly+Secure+SameSite=strict** (`/auth/refresh`).
- **2FA** comme **step-up** après OAuth si l’utilisateur l’a activée (ou si policy de sécurité l’exige).

🔐 2FA (TOTP + codes secours)

Flux

1. **Enroll** : user demande 2FA → serveur génère `secret` (160 bits), affiche **QR** (URI `otpauth://…`).

2. **Activation** : user entre 1 code valide → on **active** + on génère **10 codes secours** (hashés).

3. **Login** : si 2FA actif → après mot de passe/OAuth, on exige **TOTP** ou **backup code**.

4. **Step-up** : TOTP requis pour actions sensibles (changer mdp, désactiver 2FA, exporter données).

Contraintes

- TOTP : 6 digits, 30 s, fenêtre ±1 pas.
- `twofa_secret_enc` **chiffré** (AES-GCM, clé serveur).
- **Backup codes** hashés (argon2/bcrypt), usage unique.
- **Rate-limit** TOTP : p.ex. 5/min par session.
- **Audit** : journaliser activation/désactivation, tentatives, usages backup.

Exemple (Fastify, pseudo-code)

```
// Enroll
app.post('/auth/2fa/enroll', requireAuth, async (req, reply) => {
  const secret = crypto.randomBytes(20);
  const enc = encryptAESGCM(secret);
  await saveTwofaSecret(user.id, enc);
  const otpUri = buildOtpUri('ft_transcendence', user.email, base32(secret));
  reply.send({ otpUri, qrPng: qrFrom(otpUri) });
});

// Verify
app.post('/auth/2fa/verify', requireAuth, async (req, reply) => {
  const { code } = req.body;
  const secret = decryptAESGCM(await getTwofaSecret(user.id));
  if (!verifyTotp(secret, code)) return reply.code(401).send();
  await markTwofaEnabled(user.id);
  reply.send({ ok: true });
});
```

🔑 OAuth “42” (Authorization Code, state/PKCE)

Principes

- **Flow Authorization Code** côté **backend** (client secret **jamais** exposé).
- **State** aléatoire anti-CSRF, **PKCE** (S256) si provider le permet ; sinon, au minimum **state** + callback strict.

- **Redirect URIs** en **liste blanche** (prod/dev).
- **Account linking** : si mail existe → lier au compte ; sinon proposer création.
- **Scopes** : commencer minimal (lecture profil), élargir si nécessaire.

Flux

1. `/auth/42/login` → génère `state` (+ `code_verifier/challenge`) → redirige vers 42.
2. Callback `/auth/42/callback?code&state` : vérifier **state**, échanger code→token.
3. Appeler l'API 42 pour récupérer l'identité (id, mail).
4. **Lier** ou **créer** l'utilisateur local → créer **session** + émettre `access` + cookie `refresh`.
5. Si 2FA actif → **step-up** TOTP avant d'émettre `access`.

Exemple (lib type `simple-oauth2` ou `openid-client`)

```
app.get('/auth/42/login', async (req, reply) => {
  const state = randomBase64(32);
  const { verifier, challenge } = pkcePair();
  await saveAuthState({ state, verifier });
  const url = buildAuthUrl({ state, challenge }); // include scope, redirect_uri
  reply.redirect(url);
});
app.get('/auth/42/callback', async (req, reply) => {
  const { code, state } = req.query;
  const rec = await getAuthState(state);
  if (!rec) return reply.code(400).send('bad state');
  const token = await exchangeCodeForToken(code, rec.verifier);
  const profile = await fetch42Profile(token.access_token);
  const user = await findOrCreateUserFrom42(profile);
  const session = await createSession(user.id, req);
  if (user.twofa_enabled) return reply.redirect('/2fa'); // step-up
  const { access, refresh } = await issueTokens(user, session.id);
  setRefreshCookie(reply, refresh);
  reply.redirect('/app');
});
```

Sécurité

- **State** stocké côté serveur (avec TTL court).
- **PKCE** si possible ; sinon, forcer stricte vérification `state` + **client_secret** uniquement au backend.
- **Timeouts** réseau courts, **retries** bornés.

- **Logs** de l'échec d'échange/CSRF.
- **Désactiver** "login CSRF" en bloquant toute redirection vers un `redirect_uri` non listé.

🧮 Modèle de données (ajouts)

users(id, email, pwd_hash NULLABLE, twofa_enabled, twofa_secret_enc, provider, provider_id, created_at)
backup_codes(user_id, code_hash, used_at)
sessions(id, user_id, ua, ip_hash, created_at, revoked_at)
refresh_tokens(id, session_id, token_hash, prev_id, created_at, used_at, revoked_at)
audit(id, user_id, event, ip, ua, ts, meta_json)
oauth_states(state, verifier, created_at, used) // TTL court

🧰 Hardening & politiques

- **Cookies** : `rt` = httpOnly, Secure, SameSite=strict, `Path=/auth/refresh`.
- **CORS** : whitelist stricte des origines.
- **Headers** (via Nginx + Helmet) : HSTS, `X-Frame-Options=DENY`, `Referrer-Policy=strict-origin-when-cross-origin`, `Permissions-Policy`, **CSP** adaptée SPA.
- **Rate-limit** : `/auth/*` (login, 2FA verify, refresh) + **backoff**.
- **Rotation clés JWT** : `kid` + JWKS, planifié.
- **RGPD** : minimiser les données (pas de PII dans JWT), droit à la suppression.

🔎 Audit complet (automatisé)

Scans & supply chain

- **Semgrep (SAST)** : `npx semgrep --config p/owasp-top-ten .`

- **Trivy** : `trivy fs .` + `trivy image <image>`
- **npm audit** (ou pnpm) : hauts/critique.
- **Syft SBOM** : `syft . -o json > sbom.json`

DAST (smoke)

- **OWASP ZAP baseline** sur env dev :
  `docker run -t owasp/zap2docker-stable zap-baseline.py -t https://dev.app -r zap.html`

CI

- Job “Security” déclenché sur PR + nightly.
- **Fail pipeline** si **CRITICAL** non waivé (fichier `SECURITY_EXCEPTIONS.md` daté, owner, échéance).

Docs

- `JWT_POLICY.md`, `OAUTH42_FLOW.md`, `TWOFA_POLICY.md`, `CSP_POLICY.md`.
- `SECURITY_AUDIT.md` : périmètre, risques, résultats, remédiations (priorisées P0/P1/P2).

📊 Observabilité & traçabilité

- **Audit log** : login ok/ko, 2FA ok/ko, refresh reuse, revoke, OAuth link, unlink.
- **Métriques** (Prom/Grafana) : tentatives login/2FA (ok/ko), taux reuse, erreurs OAuth, latence callback.
- **Alertes** : pic d’échecs 2FA, pic de refresh reuse, 401/403 anormaux.

✅ Definition of Done (DoD)

- 2FA TOTP opérationnel (enroll, verify, backup), chiffré au repos, rate-limit.
- OAuth “42” en Authorization Code avec **state** (+ **PKCE** si dispo), callbacks sûrs, **account linking**.
- Access JWT 15 min, refresh rotatif + reuse detection, cookies sûrs, CORS whitelist.
- WS authentifié via **subprotocol bearer**, fermeture à expiration.
- Headers de sécu + **CSP** en place côté Nginx/Helmet.
- Scans SAST/DAST/SBOM intégrés CI, pipeline **bloquant** sur CRITICAL.
- Docs & politiques versionnées, logs d’audit actifs, métriques exposées.

🧪 Snippets utiles

Subprotocol WS (client)

```
new WebSocket(url, ['bearer', accessJwt]);
```

Helmet (Fastify)

```
await app.register(import('@fastify/helmet'), {
  contentSecurityPolicy: false, // déporté Nginx
  referrerPolicy: { policy: 'strict-origin-when-cross-origin' }
});
```

Cookie refresh

```
reply.setCookie('rt', token, {
  httpOnly:true, secure:true, sameSite:'strict', path:'/auth/refresh', maxAge:30*24*3600
});
```

🗓️ Semaine 6 – Finalisation & Présentation

🎯 Objectif global

- Définir un **référentiel** (OWASP ASVS L2 + bonnes pratiques 42 + RGPD minimal).

- Outiller le repo pour que **chaque PR** passe des tests **SAST/DAST/SCA/Container/IaC**.
- Produire des **preuves** (rapports archivé CI) et des **policies** (docs) qui passent l'audit.

🧭 Périmètre de test

- **Code**: backend Node/Fastify, frontend SPA TS, WS.
- **Infra**: Nginx/TLS, Dockerfiles, docker-compose.
- **Supply chain**: dépendances npm, images Docker.
- **Auth**: JWT/refresh/2FA/WS subprotocol.
- **Données**: cookies, CORS, logs, traces (RGPD minimal).
- **Réseau**: headers, CSP, CORS, rate-limit.

🔨 Outils & catégories

Catégorie	Outil	Ce qu'on cherche
SAST (code)	**Semgrep** (p/owasp-top-ten + custom)	XSS, SQLi, SSRF, JWT mauvais usages

SCA (deps)	**npm audit/pnpm audit** + **Trivy fs**	CVE HIGH/CRITICAL
SBOM	**Syft**	Inventaire deps (JSON)
Conteneurs	**Trivy image**, **Hadolint**, **Dockle**	CVE images, mauvaises pratiques Docker
DAST (runtime)	**OWASP ZAP Baseline** (HTTP)	En-têtes, cookies, formulaires, liens
WS fuzz	**zod-fuzz/fast-check** + tests e2e WS	payload invalides, taille/DoS
Secrets	**Gitleaks**	clés/API en repo
IaC	**kics** (si k8s plus tard) / validation compose	défauts conf
Perf-sécu	k6 (optionnel)	rate-limit & crash test léger

🧩 Structure repo (proposée)

```
/security
  semgrep/              # règles custom
  zap/                  # config ZAP
  reports/              # artefacts CI (html, json, sarif)
  policies/             # JWT_POLICY.md, CSP_POLICY.md, etc.
/.github/workflows/
  security.yml          # pipeline sécurité
```

🧪 Jeux de tests (exemples ciblés)

HTTP & Headers (Nginx + Fastify)

- HSTS (1 an, includeSubDomains), X-Frame-Options=DENY, X-Content-Type-Options=nosniff, Referrer-Policy=strict-origin-when-cross-origin, Permissions-Policy minimal, **CSP** SPA stricte.

- Cookies : `Secure`, `HttpOnly`, `SameSite=strict`, `Path=/auth/refresh` pour refresh.

- CORS : **whitelist** (pas `*`), pas de credentials larges.

Test automatisé (node)

```
import fetch from "node-fetch";
const r = await fetch("https://dev.ft/health");
if (r.headers.get("strict-transport-security")?.includes("31536000") !== true) process.exit(1);
```

Auth / JWT / Refresh / 2FA

- Access 15 min, Refresh rotatif (reuse detection).

- 2FA step-up actif, backup codes hashés.

- WS handshake via **subprotocol bearer**, **jamais** dans l'URL.

Test refresh reuse (e2e)

1. Login → récupérer `rt1`.

2. Refresh → `rt2` et `rt1` marqué `used`.

3. Réutiliser `rt1` → doit **révoquer** la session (401 + audit log).

WebSocket

- Taille `maxPayload` (ex. 8–16 KB), **validation zod** sur chaque message, **rate-limit**/slow mode.

- Fuzz: objets malformés, champs inattendus, répétitions rapides (5/s).

Property-based (fast-check)

```
fc.assert(fc.property(fc.string(1000), async (s) => {
  const msg = { t:"chat:send", room:"not-a-uuid", content:s };
  const res = await sendWS(msg);
  return res.code === 4001; // invalid schema
}));
```

XSS / Sanitization

- Messages contenant `<script>`, `onerror`, `javascript:` → doivent être purgés/échappés avant save & affichage.

Docker & images

- `USER non-root`, `HEALTHCHECK`, pas de `latest`, pas de secrets dans image.
- Trivy image → CRITICAL bloque le merge.

Secrets & Supply Chain

- Gitleaks = 0 findings (ou exceptions tracées).
- Syft SBOM généré et archivé.
- npm audit HIGH/CRITICAL → PR de bump auto (dependabot/renovate conseillé).

⚙️ Pipeline GitHub Actions (exemple minimal)

```
name: security
on:
  pull_request:
  push:
    branches: [ main ]
jobs:
  sast:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: semgrep/semgrep-action@v1
```

```
        with:
          config: p/owasp-top-ten
      - run: npx semgrep --config ./security/semgrep --error || true
  sca:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: npm ci
      - run: npm audit --audit-level=high || true
      - run: trivy fs --severity HIGH,CRITICAL --exit-code 1 .
  container:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: docker build -t ft/app:ci .
      - run: trivy image --severity HIGH,CRITICAL --exit-code 1 ft/app:ci
      - run: hadolint Dockerfile
  dast:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      # suppose un env dev up (preview) ou service lancé en job séparé
      - run: docker run --network host -t owasp/zap2docker-stable zap-baseline.py -t
http://localhost:8080 -r security/reports/zap.html || true
  secrets:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: gitleaks/gitleaks-action@v2
        with: { args: "--redact --exit-code 1" }
  sbom:
    runs-on: ubuntu-latest
    steps:
      - uses: anchore/syft-action@v0.17.0
        with: { output: "json", path: ".", artifact-name: "sbom.json" }
```

Ajuste `--exit-code` pour bloquer le merge (CRITICAL/HIGH), et consigne toute exception dans `SECURITY_EXCEPTIONS.md` (owner + échéance).

📑 Conformité & Policies (docs à livrer)

- `SECURITY_TEST_STRATEGY.md` (cette stratégie, périmètre, outils).

- `JWT_POLICY.md`, `TWOFA_POLICY.md`, `CSP_POLICY.md`, `RATE_LIMITING.md`, `WS_SECURITY_CHECKLIST.md`.
- `DATA_MINIMIZATION.md` (RGPD minimal : quelles données, durée, droits).
- `RELEASE_CHECKLIST.md` (pré-prod → prod : scans à ZÉRO, headers ok, backups).

📊 Observabilité (preuve continue)

- Exposer métriques : tentatives login ok/ko, 2FA ok/ko, refresh reuse, 4xx/5xx rate, WS close codes, latence auth.
- Dashboard Grafana rapide + alertes (pics 401/403, reuse > baseline).

✅ Definition of Done (DoD)

- Pipeline CI **exécute** SAST/SCA/DAST/Container/Secrets à chaque PR + nightly.
- **CRITICAL** bloquants, **HIGH** exigent ticket + plan (ou bump).
- WS : tests e2e (fuzz + limites) passent ; XSS nettoyé ; rate-limit vérifié.
- Headers/CSP conformes ; cookies sûrs ; CORS whitelistée.
- SBOM générée et archivée ; rapports ZAP/Semgrep/Trivy stockés.
- Policies rédigées et référencées depuis le README “Security”.

⚠️ Pièges & garde-fous

- **Faux positifs** : documenter dans `SECURITY_EXCEPTIONS.md` avec *owner* + *review date*.
- **ZAP** en “baseline” n’auth pas : prévoir un job e2e auth séparé si besoin.
- **Preview env** : idéal pour DAST (GitHub Environments).
- **WS** : ZAP ne couvre pas → tests maison obligatoires (fast-check + e2e).

Maxime

🎯 Objectif global

- **Réduire l'entropie** : branches mortes, doublons, historiques lourds.
- **Standardiser** : conventions de branches/commits/tags.
- **Sécuriser** : protections, droits, pas de réécritures dangereuses.
- **Documenter & automatiser** : scripts + CI qui maintiennent l'ordre.

🧭 Modèle de branches (simple et robuste)

GitHub Flow+ (adapté à votre équipe) :

- `main` : **protégée**, toujours déployable.
- Branches courtes par tâche :
  `feat/<module>-<slug>`, `fix/<module>-<slug>`, `chore/…`, `docs/…`, `sec/…`
  - Exemples :
    `feat/auth-oauth42`, `fix/ws-reconnect`, `sec/jwt-rotation`.
- Tags de release : `vMAJ.MIN.PATCH` (ex : `v0.5.0`).

Règles :

- Merge par **Pull Request** avec **squash** + **Conventional Commits** dans le titre.
- **Aucune** réécriture d'historique sur `main` (pas de `push --force`).
- Suppression automatique des branches **après merge** (GitHub setting).

🧼 Plan de nettoyage (ordre sûr)

Garde-fous (une fois)

- Protéger `main` : review requise (1), status checks verts, interdiction **force-push**, suppression interdite.
- Activer “Delete head branches” après merge (Settings → General → Pull Requests).
- Activer “Require linear history” (optionnel si squash).

Cartographier

```
# Branches locales vs remote
git fetch --all --prune
git branch -vv
git branch -r

# Branches distantes déjà mergées dans main
git branch -r --merged origin/main | sed 's| *origin/||' | sort | uniq > merged.txt

# Branches distantes non mergées
git branch -r --no-merged origin/main | sed 's| *origin/||' | sort | uniq > unmerged.txt
```

Archiver ce qui doit vivre (avant suppression)

- Si une branche est “historique” ou une “release” : **tag d’archive** puis suppression.

```
git checkout <branch-ancienne>
git tag archive/<branch-ancienne>
git push origin archive/<branch-ancienne>
```

Supprimer les branches mergées (sûr)

```
# (dry-run mental) lister
cat merged.txt

# supprimer côté remote (hors branches protégées)
for b in $(cat merged.txt); do
  case "$b" in main|HEAD) continue;; esac
  git push origin --delete "$b"
done
```

Traiter les branches non mergées

- Pour chaque entrée de `unmerged.txt` :

- **Comparer** : `git log --oneline origin/main..origin/<branch>`
- Si c'est un travail utile → **ouvrir une PR**, ou faire un **cherry-pick** ciblé.
- Sinon → **tag d'archive + suppression** (cf. étape 2 & 3).

Astuce rapide pour prioriser :

dernières modifs (qui mérite une revue)
git for-each-ref --format='%(committerdate:short) %(refname:short)' refs/remotes/origin | sort -r | head -n 30

Nettoyage local

Prune des références mortes
git remote prune origin

Supprimer branches locales mergées
git branch --merged main | egrep -v '^*|main$' | xargs -r git branch -d

🧱 Conventions & linting Git

Nommage des branches

feat/<zone>-<slug>
fix/<zone>-<slug>
docs/<zone>-<slug>
chore/<zone>-<slug>
sec/<zone>-<slug>

`<zone>` ∈ {web, ws, game, auth, user, chat, devops, sec, ui, a11y, db}

Messages de commit (Conventional Commits)

feat(auth): oauth42 flow with state+pkce
fix(ws): prevent double connect on reconnect
chore(devops): add trivy to security job

Tags

- Releases : `vX.Y.Z`

- Archives : `archive/<old-branch>`

🧯 Poids du repo & hygiène

Fichiers lourds / Binary

- Si >50 Mo en historique : **BFG** (ou `git filter-repo`) — *à utiliser seulement si nécessaire* (et **jamais** sur `main` public sans coordination).

```
# Exemple BFG pour purger des fichiers
bfg --delete-files '*.{mp4,zip,psd}'  # puis git gc, push --force (coordonner!)
```

- Pour l'avenir : **git-lfs** si besoin (assets larges).

`.gitignore` minimal utile

```
node_modules/
dist/
.env
*.log
.DS_Store
.idea/
.vscode/
coverage/
```

`.gitattributes` (diff/texte)

```
* text=auto eol=lf
*.png binary
*.jpg binary
*.woff binary
*.woff2 binary
```

🔐 Sécurité & qualité côté repo

- `CODEOWNERS` — routes par domaine :

/src/auth/ @mel-yand @mechard
/src/ws/ @ajamshid @mechard
/src/ui/ @jealefev
/.github/ @abutet @mechard

- PR template : checklist (tests, docs, sécurité).
- **Branch protection** sur `main` :
    - Require PR review (≥1)
    - Require status checks (build, tests, lint, security)
    - Dismiss stale approvals on new commits
    - Linear history + auto-delete branches

🤖 Automatisations utiles

Script “prune-merged-remote.sh”

```
#!/usr/bin/env bash
set -euo pipefail
git fetch --all --prune
for b in $(git branch -r --merged origin/main | sed 's| *origin/||'); do
  case "$b" in main|HEAD) continue;; esac
  echo "Deleting remote branch: $b"
  git push origin --delete "$b" || true
done
```

GH CLI — fermer PRs inactives (optionnel)

```
gh pr list --state open --search "updated:<2025-01-01" | awk '{print $1}' | xargs -n1 -I{} gh pr close {}
```

Action GitHub — auto-delete stale branches (sécurité)

- Option 1 : activer “Delete head branches” (recommandé).
- Option 2 (si besoin spécial) : action programmée qui supprime branches **mergées** via API (sans toucher aux PR ouvertes).

📄 Documentation à ajouter

- `docs/GIT_WORKFLOW.md` : modèle de branches, naming, squash, tags.
- `docs/REPO_HYGIENE.md` : routine mensuelle (scripts + commandes).
- PR Template : `/ .github/PULL_REQUEST_TEMPLATE.md`
- `CODEOWNERS` + règles de review.

✅ Definition of Done (DoD)

- `main` protégée (no force-push, checks requis, delete interdit).
- Branches **mergées** remote supprimées.
- Branches **non mergées** traitées (PR, cherry-pick, ou archivées).
- Conventions de nommage & Conventional Commits actées.
- “Delete head branches after merge” activé.
- `.gitignore` & `.gitattributes` à jour.
- `CODEOWNERS` + PR template en place.
- Script de prune + doc d’exploitation commités.

🗓️ Routine de maintenance (15 min / semaine)

1. `git fetch --all --prune`
2. Supprimer branches mergées (script).

3. Revoir `unmerged.txt` → ouvrir PR ou archiver.
4. Vérifier que les protections de branche sont intactes.
5. Faire un **tag** si milestone franchie.

🧠 Conseils pratiques

- **Jamais** de `push --force` sur `main`.
- Si vous devez purger l'historique (BFG) : **branch de maintenance**, période d'info, **freeze**, puis re-push coordonné avec l'équipe.
- **Squash & merge** garde un historique lisible, parfait pour un projet 42 multi-branches.
- Garde **1 owner** pour la sérénité (toi), mais délègue via `CODEOWNERS`.

🗓️ Semaine 2 – Authentification & Utilisateurs

🎯 Objectif global

- **Inscription** sûre (e-mail vérifié, mot de passe robuste, consentements).
- **Connexion** avec **JWT d'accès 15 min** + **refresh rotatif** en cookie httpOnly.
- **Déconnexion** qui **révoque la session** (et invalide la chaîne de refresh).
- UX fluide (erreurs utiles, pas de fuites d'infos), hooks prêts côté Front & WS.

🧱 Modèle fonctionnel (flux)

Register

1. Form : `email`, `password`, `username` (unique), consentements.
2. Création user → envoi **e-mail de vérification** (token court).
3. Tant que non vérifié : accès restreint (lecture seule / bannière).
4. Option : *magic link* en plus du mot de passe (facultatif).

Login

1. `POST /auth/login` (email + password).
2. Si user 2FA → **step-up** (`/auth/2fa/verify`).
3. Retour : `access` (JSON), `refresh` en **cookie httpOnly**.

Logout

- `POST /auth/logout` → **révocation** de la session + effacement cookie.

🗄️ Schéma de données (minimal)

```
users(
  id UUID PK,
  email TEXT UNIQUE,
  email_verified_at TIMESTAMP NULL,
  username TEXT UNIQUE,
  pwd_hash TEXT,
```

```
  created_at TIMESTAMP
)

sessions(
  id UUID PK,
  user_id UUID FK,
  ua TEXT, ip_hash TEXT,
  created_at TIMESTAMP, revoked_at TIMESTAMP NULL
)

refresh_tokens(
  id UUID PK,
  session_id UUID FK,
  token_hash TEXT, prev_id UUID NULL,
  created_at TIMESTAMP, used_at TIMESTAMP NULL, revoked_at TIMESTAMP NULL
)

email_verify_tokens(
  user_id UUID, token_hash TEXT, exp TIMESTAMP
)
```

🔐 Sécurité (exigences)

- **Hash** : Argon2id (≈100–200 ms), **pepper** serveur.
- **Access JWT** : RS256/EdDSA, 15 min, claims : `sub`, `sid`, `scope`, `iat`, `exp`.
- **Refresh** : **opaque**, 32 octets random, **rotation** + **reuse-detection** (si un ancien est réutilisé → révoquer toute la session).
- **Cookies** : `rt` = `HttpOnly`, `Secure`, `SameSite=strict`, `Path=/auth/refresh`.
- **Rate-limit** : login/register (ex. 5/min par IP + 5/min par email), backoff progressif.
- **E-mail** : token de vérif **court** (15–30 min), usage unique, stocké **hashé**.

🧪 Validation & messages d’erreur

- `email` valide (RFC), `password` ≥ 10 chars (au moins 3 classes), `username` `[a-z0-9_-]{3,20}`.
- Erreurs **non révélatrices** :

    - “Identifiants invalides” (ne pas dire si l’email existe).
    - “Lien de vérification invalide/expiré”.
- Front : afficher **recommandations** (barre de robustesse), pas la politique brute.

🔌 Endpoints (Fastify/Node – pseudo-impl)

```
// POST /auth/register
// body: { email, username, password }
createUser(); sendVerifyEmailLink(); return 204;

// GET /auth/verify?token=...
// vérifie token, marque email vérifié, 302 → /login?verified=1

// POST /auth/login
// body: { email, password }
verifyPassword(); createSession(); issueTokens();
setRefreshCookie(rt); return { access };

// POST /auth/refresh
// cookie: rt ; rotation + reuse-detection
rotate(); setRefreshCookie(newRt); return { access };

// POST /auth/logout
// header: Bearer access (pour récupérer sid) ou cookie
revokeSession(); clearRefreshCookie(); return 204;
```

Helpers cookies :

```
reply.setCookie('rt', token, {
  httpOnly:true, secure:true, sameSite:'strict',
  path:'/auth/refresh', maxAge:30*24*3600
});
reply.clearCookie('rt', { path:'/auth/refresh' });
```

🎨 UI/UX (Front Jeanne + toi)

- **Register** : fields + checklist robustesse; message “Vérifie tes mails” + bouton “Renvoyer e-mail”.

- **Login** : e-mail, password; liens “mot de passe oublié”, “se connecter avec 42” (si activé); gestion 2FA si step-up.
- **Banner** si e-mail non vérifié : CTA renvoyer.
- **Logout** : bouton visible (menu user), feedback instantané.
- **États** accessibles : focus visibles, erreurs lisibles, annonce ARIA.

🔄 WebSocket (Abdul ↔ toi)

- Client ouvre WS avec `Sec-WebSocket-Protocol: bearer, <access>`.
- Si `401 / exp` → front déclenche `/auth/refresh` (silencieusement), relance WS.
- À `logout` → **fermer** toutes les connexions WS.

📧 E-mails (templates)

- `verify_email.html` : bouton “Confirmer mon adresse”, lien alternatif.
- `device_new_login.html` (option) : nouvel appareil détecté → alerte.
- DKIM/SPF à valider si SMTP externe (ou service type Mailgun/SES).

🧰 Tests à prévoir

- **Unit** : validation inputs, hash, génération tokens, cookies.
- **Intégration** :
  - Register → verify → login → refresh → logout.
  - Refresh reuse → session révoquée.
  - Tentatives rate-limit (codes 429).
  - Login avec e-mail non vérifié (selon policy : autoriser lecture seule ou bloquer).
- **E2E** : parcours utilisateur complet + WS (connect/expire/refresh/reconnect).

✅ Definition of Done (DoD)

- Endpoints `register/login/refresh/logout/verify` stables & testés.

- Hash Argon2id + policies de complexité.
- Access 15 min, refresh rotatif httpOnly + reuse-detection.
- E-mail de vérification fonctionnel + renvoi.
- Rate-limit & messages non-affirmatifs.
- Hooks front (auth context, refresh silencieux, fermeture WS au logout).
- Doc `AUTH_FLOW.md` + `EMAIL_TEMPLATES.md`.

🗓️ Plan d’exé (3–4 jours)

1. **Jour 1** : schémas DB, register + verify mail, validations.
2. **Jour 2** : login + refresh rotatif + cookies + rate-limit.
3. **Jour 3** : logout + intégration WS + E2E de base.
4. **Jour 4** : UI polish, messages accessibles, docs & tests restants.

🔎 Bonus utiles

- “Rester connecté 30 jours” = **session marquée** 2FA-verified, pas d’allègement de sécurité.
- “Magic link” en option (limité, révoquable, TTL court).
- Page **“Mes appareils”** (liste de sessions, bouton “Révoquer”).

🎯 Objectif global

- **Chat** : DM et rooms (public/privé), pagination, édition/suppression soft, pièces jointes légères (liens), read-receipts basiques.
- **Invitations de jeu** : créer, accepter/refuser/expirer, transformer en **match** (ou rejoindre file de matchmaking).
- **Modération** : block/mute, bad-words, rate-limit, flagging.
- **Sécurité** : auth WS via subprotocol, validation stricte, sanitization, quotas.
- **Observabilité** : logs d'audit, métriques Prometheus.

🗄️ Modèle de données (SQLite → compatible Postgres)

```
-- Utilisateurs
users(id TEXT PK, username TEXT UNIQUE NOT NULL);
-- Relations sociales
blocks(blocker_id TEXT, target_id TEXT, PRIMARY KEY(blocker_id,target_id));
mutes(muter_id TEXT, target_id TEXT, room_id TEXT NULL, until TIMESTAMP NULL);
-- Rooms
rooms(id TEXT PK, kind TEXT CHECK(kind IN ('public','private','dm')) NOT NULL,
    name TEXT NULL, created_by TEXT, created_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP);
room_members(room_id TEXT, user_id TEXT, role TEXT CHECK(role IN
('owner','mod','member')),
          joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, PRIMARY
KEY(room_id,user_id));
-- Messages
messages(id TEXT PK, room_id TEXT, user_id TEXT, content TEXT, sanitized TEXT,
       created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, edited_at TIMESTAMP
NULL, deleted_at TIMESTAMP NULL,
       flagged INTEGER DEFAULT 0, meta_json TEXT);
```

reads(room_id TEXT, user_id TEXT, last_msg_id TEXT, read_at TIMESTAMP, PRIMARY KEY(room_id,user_id));
-- Invitations de jeu
invites(id TEXT PK, from_user TEXT, to_user TEXT, room_id TEXT NULL,
mode TEXT CHECK(mode IN('classic','tournament')), status TEXT CHECK(status IN('pending','accepted','declined','expired')),
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, expires_at TIMESTAMP);

DM = `rooms.kind='dm'` avec 2 membres.
`sanitized` = contenu purgé XSS.
`meta_json` = URLs détectées, assets légères, etc.

🔌 API & WS : surfaces

REST minimal

- `POST /chat/rooms` créer room (private/public).
- `GET /chat/rooms/:id/messages?before=<id>&limit=50` pagination.
- `POST /chat/rooms/:id/messages` (optionnel si tu veux aussi HTTP).
- `POST /chat/invites` créer une invite `to, mode, room?`.
- `POST /chat/invites/:id/{accept|decline}`.
- `GET /chat/invites` list perso (filtre `status`).

WebSocket (canal temps réel)

Subprotocol: `Sec-WebSocket-Protocol: bearer, <ACCESS_JWT>`

Événements In → Out (types)

```
// Client → Serveur (In)
{ t:'room.join', room:string }
{ t:'room.leave', room:string }
{ t:'chat.send', room:string, content:string }          // texte
{ t:'msg.edit', id:string, content:string }
{ t:'msg.delete', id:string }
{ t:'invite.send', to:string, mode:'classic'|'tournament', room?:string }
{ t:'invite.act', id:string, action:'accept'|'decline' }

// Serveur → Client (Out)
```

{ t:'room.join.ok', room, members:[...] }
{ t:'chat.msg', room, msg:{id, user, content, sanitized, created_at, meta} }
{ t:'msg.edited', id, sanitized, edited_at }
{ t:'msg.deleted', id }
{ t:'invite.recv', invite:{id, from, mode, room, expires_at} }
{ t:'invite.update', id, status } // accepted/declined/expired
{ t:'error', code, reason }
{ t:'notice', code, message } // ex: rate-limit

🧰 Validation & Sanitization (stricte)

- **Zod** pour schémas de messages WS (types + tailles), **reject** sinon.
- Longueur `content` ≤ **500** chars ; `maxPayload` WS 8–16 KB.
- **sanitize-html** whitelist minimale (b, i, code, a[href] https/mailto).
- **Anti-spam** : 5 messages / 5 s / user / room (slow mode + avertissement).
- **Block/Mute** avant diffusion (ne pas livrer au muter / respecter block).

🔐 Contrôles d’accès

- **DM** : seuls les 2 membres.
- **Private** : membre requis.
- **Public** : lecture ouverte si connecté ; écriture selon policy (membre/whitelist).
- **Block** : si `A` bloque `B`, messages de `B` non livrés à `A` (et pas de DM auto).
- **Moderation** : owner/mod peuvent `msg.delete`, `member.kick`, `room.invite`.

♻️ Invitations de jeu

- `invite.send` → créer en DB (TTL par défaut 2–5 min).
- Notifier `to_user` par WS (`invite.recv`).
- `invite.act: accept` → statut `accepted` + émettre **hook** “match:create”

    - soit **appel matchmaking** (`queue.add({p1, p2, mode})`),
    - soit **création room de match** (si server-side Pong dédié).
- Expiration **auto** via job (cron/TTL) → `invite.update: expired`.

🧪 Snippets (Fastify + WS)

Zod schémas

```
import { z } from 'zod';
export const ChatSend = z.object({
  t: z.literal('chat.send'),
  room: z.string().uuid(),
  content: z.string().trim().min(1).max(500),
});
export const InviteSend = z.object({
  t: z.literal('invite.send'),
  to: z.string().uuid(),
  mode: z.enum(['classic','tournament']),
  room: z.string().uuid().optional(),
});
```

Sanitizer

```
import sanitizeHtml from 'sanitize-html';
const sanitize = (s:string)=> sanitizeHtml(s, {
  allowedTags: ['b','i','strong','em','a','code'],
  allowedAttributes: { 'a':['href'] },
  allowedSchemes: ['https','mailto'],
});
```

Gestion WS (esquisse)

```
type Ctx = { userId:string, rooms:Set<string> };
ws.on('connection', (socket, req) => {
  const access = subprotocolJwt(req); // récup depuis 'bearer'
  const user = verifyAccess(access);  // sub, scopes, sid
  const ctx:Ctx = { userId:user.sub, rooms:new Set() };
  socket.on('message', async (raw) => {
    let msg:any; try{ msg = JSON.parse(raw.toString()) } catch{ return sendErr('bad_json') }
    // routeur
    switch(msg.t){
```

```
      case 'room.join': return joinRoom(ctx, msg.room);
      case 'chat.send': return onChatSend(ctx, ChatSend, msg);
      case 'invite.send': return onInviteSend(ctx, InviteSend, msg);
      // ...
      default: return sendErr('unknown_type');
    }
  });
  function send(toRoom:string, payload:any){
    // diffuser seulement aux membres & non-bloqués & non-mute
  }
});
```

onChatSend (sécurité incluse)

```
async function onChatSend(ctx:Ctx, schema:any, msg:any){
  const { success, data } = schema.safeParse(msg);
  if (!success) return sendErr('schema');

  if (!await isMember(data.room, ctx.userId)) return sendErr('not_member');
  if (await isMuted(ctx.userId, data.room))   return sendNotice('muted');

  if (rateLimitHit(ctx.userId, data.room))    return sendNotice('slow_mode');

  const sanitized = sanitize(data.content);
  const id = uuid();
  await db.insert('messages',{ id, room_id:data.room, user_id:ctx.userId,
content:data.content, sanitized });
  broadcast(data.room, { t:'chat.msg', room:data.room, msg:{ id, user:ctx.userId,
content:data.content, sanitized, created_at:Date.now() }});
}
```

Invites

```
async function onInviteSend(ctx:Ctx, schema:any, msg:any){
  const { success, data } = schema.safeParse(msg);
  if (!success) return sendErr('schema');
  if (await isBlocked(data.to, ctx.userId))   return sendErr('blocked');

  const invId = uuid();
  const expires = Date.now() + 2*60_000;
  await db.insert('invites',{ id:invId, from_user:ctx.userId, to_user:data.to, room_id:msg.room
?? null, mode:data.mode, status:'pending', expires_at:new Date(expires) });

  pushToUser(data.to, { t:'invite.recv', invite:{ id:invId, from:ctx.userId, mode:data.mode,
room:msg.room ?? null, expires_at:expires }});
}
```

📦 Pagination & index

- **Pagination par curseur** (`before=<msg_id>`) + index `(room_id, created_at DESC)`.
- Lire 50 messages par page ; **read-receipts** via `reads(room_id,user_id,last_msg_id)`.

🚨 Modération & conformité

- **Flagging auto** : bad-words → `flagged=1` + log modération.
- **Delete soft** : messages “deleted” restent en DB (audit), `sanitized='[deleted]'`.
- **Audit log** séparé : join/leave, msg.reject (raison), invite actions.
- **RGPD minimal** : pas de PII dans messages système ; `ip_hash` si nécessaire (pepper).

📊 Observabilité (Prom/Grafana)

- `chat_msgs_total{room,kind}`
- `chat_reject_total{reason}` (schema, ratelimit, blocked, mute)
- `invites_total{status}`
- `ws_connections{state}`
- Latences handler (histogrammes).

🧪 Tests à prévoir

1. **Validation** : messages invalides → rejet 4001.
2. **Sanitization** : `<script>` purgé, `javascript:` bloqué.

3. **Rate-limit** : 6 msgs/5s → notice + pas de diffusion.
4. **Block/Mute** : blocage respecte sens correct.
5. **DM** : non-membre refuse join/send.
6. **Invites** : send → recv → accept → hook match ; expiration auto.
7. **Pagination** : stable (ordre, curseur).
8. **Delete/Edit** : droits (owner/mod/self), diffusion `msg.deleted/edited`.

✅ Definition of Done (DoD)

- Endpoints REST basiques + **canal WS** opérationnel (join/send/invite).
- Validation **zod**, **sanitization** en place, **rate-limit** & block/mute fonctionnels.
- Pagination par curseur, read-receipts, delete soft.
- Invitations avec TTL, accept/decline, hook vers matchmaking.
- Logs d'audit + métriques exposées.
- Suite de tests verte (unit + e2e WS).
- Docs : `CHAT_BACKEND.md` (schémas, événements), `INVITES_FLOW.md`.

Intégrations

- **Frontend (Jeanne)** : hooks `useChat`, `useRoom`, `useInvites`, toasts sur `invite.recv`.
- **Gameplay/UX (Abdul)** : `invite.accept` → navigation match / queue.
- **DevOps (Lylou)** : exposer `/metrics`, variables Ratios rate-limit.
- **Sécurité (Mehdi)** : brancher ses checklists WS (auth, quotas, headers).

🎯 Objectif global

- Création/édition de tournois (single-elim, round-robin → single-elim “playoffs”).
- Inscriptions/check-in, seeding (aléatoire/Elo), gestion des **BYE**.
- Génération du **bracket** (arbre) + progression auto des gagnants.
- Planification des matchs, délais, forfaits (WO), re-scheduling minimal.
- Scores, tie-breakers (BO1/BO3), validation croisée, audit.
- Événements temps réel (WS) + endpoints REST, pagination.
- Observabilité Prometheus + logs d’audit structurés.

🗄️ Modèle de données (SQLite → Postgres-friendly)

```
tournaments(
  id TEXT PK, name TEXT, mode TEXT CHECK(mode IN ('se','rr_se')),
  best_of INT DEFAULT 1, status TEXT CHECK(status IN
('draft','reg','checkin','ongoing','done','canceled')),
  seed_policy TEXT CHECK(seed_policy IN ('random','elo')), owner_id TEXT, created_at
TIMESTAMP, starts_at TIMESTAMP, checkin_until TIMESTAMP
);

tournament_players(
  tid TEXT, user_id TEXT, seed INT NULL, checked_in INT DEFAULT 0,
  elo INT NULL, PRIMARY KEY(tid, user_id)
);

matches(
  id TEXT PK, tid TEXT, round INT, pos INT,  -- pos = index dans la round
  best_of INT, state TEXT CHECK(state IN
('pending','ready','playing','reported','confirmed','wo','canceled')),
  p1 TEXT NULL, p2 TEXT NULL, winner TEXT NULL, scheduled_at TIMESTAMP NULL,
  parent_next TEXT NULL, parent_slot INT NULL,          -- où reporter le gagnant
  chat_room_id TEXT NULL, created_at TIMESTAMP
);

games( -- si BO3 etc.
  id TEXT PK, match_id TEXT, index INT, p1_score INT DEFAULT 0, p2_score INT
DEFAULT 0
);

reports(  -- rapports de score (anti-litige)
  id TEXT PK, match_id TEXT, reporter TEXT, p1 INT, p2 INT, created_at TIMESTAMP
);

audit(id PK, tid TEXT, evt TEXT, actor TEXT, ts TIMESTAMP, meta_json TEXT);
```

```
      mode:
```

- `se` = **single elimination** pur.
- `rr_se` = **phase de poules (round-robin) → playoffs single-elim**.

🔀 Flux fonctionnels

Création / configuration

- `POST /tournaments` → `draft` (nom, mode, best_of, starts_at, checkin_until, seed_policy).
- `PATCH /tournaments/:id` tant que `draft`/`reg`.

Inscriptions

- `status = reg` → `POST /tournaments/:id/join` (vérifie blocklist/dup).
- `checkin`: fenêtre `checkin_until` → `/checkin` marque `checked_in=1`.
- À la fin du check-in → **purge** les non-checkés (optionnel), **gèle** la liste.

Seeding

- **random** (Fisher-Yates) ou **elo** (descendant).
- Remplir `seed` par joueur.
- Construire un bracket **puissance de 2**, insérer **BYE** si nécessaire (haut de seed).

Génération bracket (single-elim)

- Nombre de rounds = `ceil(log2(N))`.
- Créer `matches` par round + liaisons `parent_next, parent_slot`.
- Marquer `ready` les matches dont les deux slots (`p1/p2`) connus (ou `wo` si BYE).

Phase de poules (`rr_se`)

- Regrouper joueurs en poules (k≈4). Round-robin dans chaque poule.

- Classement par **W-L**, puis **diff score**, puis **head-to-head**.
- Qualifiés → génération du bracket **playoffs** (comme ci-dessus).

Planification / notifications

- `scheduled_at` calculé (fenêtres par round, ex: R1 à T0, R2 à T0+30 min).
- Créer salon chat de match (`chat_room_id`) + WS events : `match.ready`, `match.scheduled`.

Score & validation

- Client envoie `POST /matches/:id/report {p1, p2}`.
- **Double validation** :
  - 1er report → `state=reported`, stocker `reports`.
  - 2e (adverse) **concordant** → `confirmed` + `winner`.
  - Conflit → `dispute` (flag modération, arbitre = owner/mod).
- BO3 : actualiser `games`, déterminer **majorité**.

Progression & fin

- Sur `confirmed`: propager `winner` vers `parent_next/slot`, marquer match suivant `ready`.
- Lorsque dernière finale `confirmed` → `status=done`, **podium** calculé.

Forfaits / retards

- Si un joueur ne se présente pas avant `grace_period` → `wo` (walk-over).
- `wo` assigne auto `winner` l'autre joueur + progression.

🔌 API (REST) & événements WS

REST (exemples)

POST /tournaments
PATCH /tournaments/:id
POST /tournaments/:id/open-reg → status=reg
POST /tournaments/:id/open-checkin → status=checkin
POST /tournaments/:id/lock → fin checkin → seeding + bracket
GET /tournaments/:id → détails + rounds
GET /tournaments/:id/matches?round=1&cursor=...&limit=50
POST /tournaments/:id/join
POST /tournaments/:id/checkin
POST /matches/:id/report → {p1, p2} (ou détail BO3)
POST /matches/:id/confirm → confirme le report adverse
POST /matches/:id/wo → admin/mod force forfait

WS (émis)

{ t:'tournament.update', tid, status }
{ t:'bracket.ready', tid }
{ t:'match.ready', tid, match_id }
{ t:'match.scheduled', tid, match_id, scheduled_at }
{ t:'match.state', tid, match_id, state } // playing/reported/confirmed/wo
{ t:'match.result', tid, match_id, winner, score }
{ t:'notice', code, message }

Auth WS par **subprotocol bearer** (access JWT), scoping : `/tournaments/:id` rooms.

🔐 Sécurité & règles métier

- **Auth** obligatoire (JWT).
- **Owner/mod** seuls peuvent : ouvrir/fermer inscriptions, verrouiller, WO, re-schedule, annuler.
- **Rate-limit** sur `report/confirm` pour éviter le spam.
- Validation stricte (zod) : scores entiers ≥0, somme cohérente pour BOx.
- **Anti-triche** :

- Pas de report si match pas `playing/ready`.
- Pas de `confirm` par le même reporter.
- **Horodatage** + IP (hashée) dans `reports` pour l'audit.

🧠 Seeding & BYE (détails)

- Nombre d'entrants **M**, prochaine puissance de 2 **P** (ex: M=10 → P=16).
- **P-M** BYE placés de façon à **protéger les seeds hauts** (standards de bracket).
- Un match avec un BYE → opposant **auto-qualifié** (state=wo→confirmed).

⚙️ Intégration gameplay & chat

- **Chat** : à la création du match → `chat_room_id`.
- **Gameplay** : `match.ready` → Abdul déclenche "salle de jeu" avec `match_id`.
- À la fin d'un match joué côté jeu, le service gameplay peut **publier** un score qui **pré-remplit** le `report` (toujours validé par un joueur ou arbitre).

📈 Observabilité (Prom/Grafana)

- `tournaments_total{status}`
- `matches_total{state, round}`
- `reports_conflicts_total`
- Durée moyenne par round, temps de lock→bracket, no-show rate.
- Logs d'audit : transition d'état, WO, disputes.

🧪 Tests (unit + intégration)

1. **Seeding** random/elo → ordre et intégrité bracket.
2. **BYE** → propagation auto correcte.

3. **Report/Confirm** → OK, conflit → `dispute` (bloque progression).
4. **BO3** → 2-1/2-0 cohérents, ordre des games.
5. **rr_se** → classement poules, tie-breakers.
6. **Lock** → pas de join après lock.
7. **WO** → délais respectés, progression correcte.
8. **Permissions** → owner/mod vs player.

✅ Definition of Done (DoD)

- CRUD tournoi + transitions d'état **sûres** (`draft→reg→checkin→ongoing→done`).
- Seeding + génération bracket (SE) + poules (`rr_se`), BYE gérés.
- Matches jouables, BOx supporté, reports double-validés, conflits traités.
- WS événements émis, endpoints REST paginés.
- Observabilité + audit en place.
- Suite de tests verte (incl. cas limites 3, 5, 9, 17 joueurs).
- Docs : `TOURNAMENTS_BACKEND.md` (schémas, algos, endpoints).

🧩 Squelettes (TS/Fastify – indications rapides)

```
// bracket.ts
export function buildSingleElim(seeds:number[]): MatchNode[] { /* ... */ }
export function nextPowerOfTwo(n:number){ /* ... */ }

// seeding.ts
export function seedRandom(players:Player[]): Player[] { /* ... */ }
export function seedElo(players:Player[]): Player[] { /* ... */ }

// progress.ts
export function confirmMatch(db, matchId, winner){ /* maj match + push parent */ }

// tiebreak.ts (rr)
export function rankRoundRobin(group:PlayerStats[]): Ranking { /* W-L, diff, h2h */ }
```

🗓️ Plan d’exé (2 semaines focus, emboîtable dans vos 6 sem.)

- **J1–J2** : modèles, états, endpoints base (create/join/checkin/lock).
- **J3–J4** : seeding + bracket SE + BYE + WS `bracket.ready`.
- **J5–J6** : reports/confirm/BOx + progression auto + conflits.
- **J7** : poules (rr) + qualifs → playoffs.
- **J8** : WO, re-schedule, audits & métriques.
- **J9–J10** : tests lourds + docs + polish (perfs & pagination).

🎯 Objectif global

- **Taxonomie d'erreurs** commune (HTTP, WS, jobs), codes stables, messages propres.
- **Logs structurés JSON** (niveau, service, reqId, userId, route, latences, tags).
- **Corrélation** bout-à-bout (X-Request-ID + traceId).
- **Redaction** de secrets/PII, rotation & rétention.
- **Métriques** (Prometheus) + **alertes** (SLO de base).
- **Pages d'erreur** et **payloads WS** cohérents, dev vs prod séparés.
- **Docs + tests de chaos** (scénarios d'échec réalistes).

🧩 Taxonomie des erreurs

Couche	Type	Exemple	Rendu
Val. entrée	`E_SCHEMA`	zod yup	400 + détails champs
Auth	`E_AUTH`, `E_2FA`, `E_SCOPE`	JWT invalide/expiré	401/403
Métier	`E_DOMAIN`	invite expirée/room close	409/422
Dépendance	`E_DEP_TIMEOUT`, `E_DB`, `E_WS_BROKER`	DB down	503
Quota/DoS	`E_RATE_LIMIT`	flood chat	429
Interne	`E_INTERNAL`	bug non capturé	500

Convention code : `FT_<module>_<erreur>` (ex. `FT_CHAT_E_SCHEMA`), **message stable**, **hint** optionnel.

🧱 Architecture logs & tracing

- **Format** : JSON ligne (ndjson).
- **Niveaux** : `debug` (dev), `info`, `warn`, `error`, `fatal`.
- **Clés obligatoires** :
  - `ts` (ISO), `level`, `service`, `env`, `version`
  - `reqId` (X-Request-ID), `traceId`, `spanId`
  - `route`/`handler`, `method`, `status`, `latency_ms`
  - `userId` (si auth), `ip_hash`
  - `err.code`, `err.msg`, `err.stack?` (prod: stack masquée par défaut)
  - `tags` (ex. `["ws","chat"]`)
- **Corrélation** :
  - Ingress : si `X-Request-ID` absent → générer.
  - WS : propager `reqId` à l'upgrade puis stocker en contexte socket.
  - Jobs : `traceId` généré par tâche, propagé aux sous-ops.

🔐 Redaction & vie privée

- **Jamais** loguer : mots de passe, secrets, tokens, cookies, TOTP, payloads sensibles.
- **Masquer** les e-mails (`a***@d***.tld`) et **hash IP** (SHA256 + pepper).
- **Sample** messages volumineux (max 1 kB) avec suffixe `truncated=true`.
- **Rétention** : dev 7 j, prod 14–30 j (logs applicatifs), rotation quotidienne.

⚙️ Implémentation (Fastify/Node + WS)

Logger structuré (pino)

```
import pino from 'pino';
export const logger = pino({
  level: process.env.LOG_LEVEL ?? 'info',
  base: { service: 'ft-backend', env: process.env.NODE_ENV },
  redact: ['req.headers.authorization', 'req.headers.cookie', 'res.headers["set-cookie"]']
});
```

Middleware reqId + timing

```
app.addHook('onRequest', async (req, reply) => {
  const reqId = req.headers['x-request-id'] ?? crypto.randomUUID();
  reply.header('x-request-id', reqId);
  // attacher au logger
  (req as any).log = logger.child({ reqId, route: req.routerPath, method: req.method });
  (req as any).start = performance.now();
});

app.addHook('onResponse', async (req, reply) => {
  const ms = Math.round(performance.now() - (req as any).start);
  (req as any).log.info({ status: reply.statusCode, latency_ms: ms }, 'http_end');
});
```

Gestionnaire global d’erreurs (HTTP)

```
app.setErrorHandler((err, req, reply) => {
  const code = mapErrorToCode(err);   // FT_CHAT_E_SCHEMA, FT_AUTH_E_SCOPE, …
  const http = mapErrorToHttp(err);   // 400/401/403/409/422/429/500
  const safe = makeSafeMessage(err);  // message sans PII

  req.log.error({ err: { code, msg: safe, stack: isProd()?undefined:err.stack } }, 'http_error');
  reply.code(http).send({
    error: { code, message: safe, hint: hintFor(code) }
  });
});
```

WS : try/catch + codes fermetures

```
function safeHandle(socket, ctx, handler){
  return async (msg) => {
    try {
      await handler(msg);
    } catch (err) {
      ctx.log.error({ err: { code: mapErrorToCode(err), msg: makeSafeMessage(err) }},
'ws_error');
      // notifier le client
      socket.send(JSON.stringify({ t:'error', code: mapErrorToCode(err), reason:
makeSafeMessage(err) }));
      // fermer si critique
      if (isFatal(err)) socket.close(4000, 'fatal_error');
    }
  };
}
```

Rate-limit & “safe fallbacks”

- Sur `E_RATE_LIMIT` → répondre `429` + header `Retry-After`, côté WS envoyer `notice` et ignorer la commande.
- Sur `E_DEP_TIMEOUT` (DB/queue) → **circuit breaker** court (ex. 5 s) + `503`.

📊 Métriques & alertes

Prometheus exposé :

- `http_requests_total{route,method,status}`
- `http_request_duration_ms_bucket{route}` (histogram)
- `ws_messages_total{type,code}`
- `errors_total{code,module}` ← clé pour alertes
- `rate_limit_hits_total{route}`
- `dependency_errors_total{kind}` (db, cache, mail, oauth)

Alertes basiques (Grafana/Alertmanager) :

- `errors_total{code=~"FT_.*"}` ↑ > X/min pendant Y min.
- `5xx rate` > 1% sur 5 min.
- `ws close{reason="fatal_error"}` > seuil.

- **SLO** : p90 latence HTTP < 200 ms (chat/invites), < 100 ms (auth handshake).

🧪 Tests & chaos

- **Unit** : mapErrorToCode/HTTP, safe message, redaction.
- **E2E** :
  - schema invalide → 400 + `FT_*_E_SCHEMA`
  - rate-limit → 429 + Retry-After / notice WS
  - dépendance indisponible → 503 + log `E_DEP_TIMEOUT`
- **Chaos ciblé** (local/dev) : couper DB 30 s → vérifier 503, pas de panique.
- **Snapshot logs** : formats JSON valides, champs obligatoires présents.

🖥️ Dev vs Prod

- **Dev** : `LOG_LEVEL=debug`, stack complète, pretty-print (pino-pretty).
- **Prod** : `info`/`warn`/`error`, stack masquée par défaut, sortie **JSON** vers stdout.
- **Feature flags** : `LOG_HTTP_BODY=false` (ne jamais loguer les bodies par défaut).

📄 Docs & conventions

- `LOGGING.md` : structure, niveaux, clés, exemples.
- `ERRORS.md` : table **code** ↔ **http** ↔ **message** (stable), with hints.
- `RUNBOOKS.md` : que faire sur `E_DB`, `E_WS_BROKER`, `5xx spike`, `rate-limit storm`.
- **Exemples** d'erreurs (copier/coller stack synthétique + reqId).

✅ Definition of Done (DoD)

- Logger pino branché partout (HTTP/WS/jobs) + reqId/traceId.
- Gestionnaire d'erreurs global (HTTP) + wrapper WS.
- Taxonomie `FT_<module>_<erreur>` mappée HTTP + messages "safe".
- Redaction sensible + limites de taille/sampling.
- Métriques Prom exposées + 3 alertes de base.
- Tests unit/e2e sur cas critiques + chaos DB court.
- Docs `LOGGING.md`, `ERRORS.md`, `RUNBOOKS.md` ajoutées au repo.

🗓️ Plan d'exé (2 jours focus)

- **J1** : pino + hooks reqId + errorHandler HTTP + mapping erreurs + métriques HTTP.
- **J2** : wrapper WS + codes/close + métriques WS + redaction + docs + tests.

Bonus "confort dev"

- Middleware **"echo-reqId"** dans les réponses → copy/paste en support.
- `X-Request-ID` logué côté Nginx aussi (corrélation proxy ↔ app).
- Un **"sandbox error"** `/dev/boom?code=FT_CHAT_E_SCHEMA` pour valider la chaîne log/alert sur demande.

🎯 Objectif global

- Geler la base de code, **stabiliser**, merger proprement.
- Produire des **artefacts versionnés** (images Docker, build front).
- **Tagger** `v1.0.0` + notes de release utiles.
- Déployer en **prod** via `docker-compose`/stack, vérif auto, **plan de rollback** éprouvé.
- Observabilité & communication post-release.

🧊 Pré-freeze (T-3 jours)

1. **Gel des branches**
   - Geler `main` : n'accepter que des PR **bugfix** marquées `release-blocker`.
   - Basculer toutes les PR "feature" vers `v1.1` (milestone).
2. **Labels & board**
   - Label `release-blocker`, board "v1.0 Stabilization".
3. **Versions**
   - Fixer `VERSION=1.0.0` dans repo (fichier unique `VERSION` ou `package.json` app).
4. **CI stricte**
   - Gate : build, tests, e2e, security (Semgrep/Trivy/ZAP baseline) → **vert** sinon PR bloquée.
5. **Doc "Release Checklist"** à jour dans `docs/RELEASE_CHECKLIST.md`.

🔀 Intégration finale

1. **Rebase/squash** des PR bugfix sur `main`
   - `Squash & merge` avec style Conventional Commits.
2. **Vérif dépendances**
   - `npm ci && npm audit --audit-level=high`
   - `trivy fs --severity HIGH,CRITICAL` (OK).
3. **Tests & e2e**
   - Back : unit + integration
   - Front : unit + cypress-smoke (login, chat, match, tournois)
   - WS : connect/refresh/reconnect + chat + invite.
4. **Freeze** : tag candidat `v1.0.0-rc.1`

🧪 Pré-release (staging)

1. **Build & push images**
   - `docker build -t ghcr.io/<org>/ft_api:1.0.0-rc1 .`
   - `docker build -t ghcr.io/<org>/ft_front:1.0.0-rc1 ./front`
   - `docker push ghcr.io/<org>/...`

2. **Compose staging**
   - `docker compose -f compose.staging.yml up -d`
   - Migrations DB → `docker compose exec api npm run migrate`

3. **Smoke tests** (automatisés)
   - Health API, login, OAuth42, 2FA step-up, chat send/recv, invite accept, création tournoi, métriques /metrics.

4. **Sécurité rapide**
   - ZAP baseline sur staging (report archivé).

5. **Go/No-Go** (toi + Lylou + Jeanne + Mehdi + Abdul). Si OK → prod.

🏷️ Versioning, tag & changelog

1. **Bump final** si nécessaire : `1.0.0`
2. **Changelog** (auto + retouche humaine) :
   - `feat:` visibles, `fix:` listés, “breaking changes” (s’il y en a)
   - Merci, highlights, captures.
3. **Tag signé**

git checkout main
git pull
git tag -s v1.0.0 -m "ft_transcendence v1.0.0"
git push origin v1.0.0

4. **GitHub Release**

   - Titre: `v1.0.0 — First stable`
   - Corps: changelog + “How to deploy” + hashes images
   - Assets: `compose.prod.yml`, `.env.template`, SBOM, ZAP/Trivy reports résumés.

🚀 Déploiement prod

Option A — Blue/Green (recommandé)

- Deux stacks `ft_blue` et `ft_green`. On déploie la nouvelle (`green`), on bascule le trafic (Nginx/upstream), on surveille, on éteint l'ancienne.

Option B — Rolling compose

Images finales

docker pull ghcr.io/<org>/ft_api:1.0.0
docker pull ghcr.io/<org>/ft_front:1.0.0

1.
2. **Secrets & env**
    - `.env` prod avec `JWT_KEYS`, OAuth42, DB_URL, 2FA, `CSP`, `LOG_LEVEL=info`
3. **Migrations**
    - `docker compose -f -f compose.prod.yml up -d db`
    - `docker compose -f compose.prod.yml run --rm api npm run migrate`
4. **Up**
    - `docker compose -f compose.prod.yml up -d --remove-orphans`
5. **Checks**
    - `curl -f https://app/health`
    - Front : login → chat message → invite → début de match → métriques ok
    - Grafana : 5xx < 1%, latences p90, erreurs WS, taux 401/403.

🧯 Rollback (documenté, testé)

- **Principe** : images immuables + tags versionnés → rollback = redeployer `v1.0.0-rc1` ou `v0.9.x`.

- **DB** : migrations **réversibles** (up/down). Si breaking non réversible → **hold** & hotfix; sinon restauration snapshot (rpo < 5 min).

Commandes :

```
 docker compose -f compose.prod.yml pull \
  ghcr.io/<org>/ft_api:1.0.0-rc1 \
  ghcr.io/<org>/ft_front:1.0.0-rc1
sed -i 's/1.0.0/1.0.0-rc1/' compose.prod.yml
docker compose -f compose.prod.yml up -d
# DB down migration si prévu
```

📊 Post-release (24–48 h)

- **Monitoring resserré** : alertes 5xx, WS close (fatal), latence auth, erreurs OAuth42/2FA, rate-limit hits anormaux.
- **Sentry/Logs** : top 5 erreurs avec reqId + reproduction.
- **Feedback** UX (Jeanne+Abdul) : friction login, chat, tournois.
- **Security** : revue ZAP/Trivy post-release, backlog `v1.0.x`.
- **Hotfix policy** : `hotfix/*` → `v1.0.1` tag + déploiement ciblé.

📑 Fichiers à livrer/mettre à jour

- `VERSION` → `1.0.0`
- `CHANGELOG.md` (section v1.0.0)
- `docs/RELEASE_CHECKLIST.md` (pas-à-pas ci-dessus)
- `docs/ROLLBACK.md` (2 scénarios : images, DB migrations)
- `compose.prod.yml` (images taggées `1.0.0`, healthchecks, ressources)
- `SECURITY_AUDIT.md` (résumé scans)
- `AUTH_FLOW.md`, `CHAT_BACKEND.md`, `TOURNAMENTS_BACKEND.md` (stables)

✅ Definition of Done (DoD)

- `main` verte + **tag signé** `v1.0.0` + Release GitHub publiée.
- Images `ft_api:1.0.0` / `ft_front:1.0.0` **poussées** & référencées en prod.
- Déploiement **réussi** + smoke tests prod OK.
- **Monitoring** & **alertes** actives, logs propres.
- **Rollback** testé au moins une fois (staging).
- Changelog & docs à jour, comms envoyées.

🧠 Conseils d’artisan

- Toujours **tagger** les images avec la version + un digest (sha256) dans la release notes.
- Ne jamais lier `latest` en prod.
- Healthchecks **stricts** dans compose (API + WS ping).
- Une personne **aux commandes**, une **à l’observation** (dashboards), une **au support** (Slack/Discord).

Détails des Rôles :

Lead – User/Git Management

🧭 Rôle de Lead (technique & coordination)

Tu es le **chef d'orchestre**. Pas celui qui joue tous les instruments, mais celui qui assure que chacun entre au bon moment, sur le bon tempo.

Concrètement :

- **Planification** : tu découpes les objectifs du projet, définis la roadmap, attribues les priorités et répartis les tâches selon les compétences.
- **Cohérence technique** : tu garantis que les modules (front, back, game, sécurité, etc.) convergent dans la même direction architecturale.
- **Décisions structurantes** : stack technique, conventions de nommage, format des commits, structure des services et interfaces d'API.
- **Code reviews de référence** : tu valides les branches critiques (auth, user, intégration WS, etc.) et assures un niveau de qualité homogène.
- **Support global** : quand quelqu'un bloque (merge conflict, dépendance, bug profond), tu interviens pour débloquer, pas pour faire à la place.

C'est un rôle d'équilibriste : il faut **rester technique**, mais **savoir déléguer intelligemment**. Tu poses la méthode, tu ne micromanages pas.

🧑‍💻 User Management (module principal)

Ce module fait de toi le garant de **l'identité** et **de la sécurité des interactions utilisateur**.

Ton périmètre inclut :

- **Authentification complète** (login/register/logout, JWT, 2FA, OAuth42).
- **Gestion des rôles et permissions** (user/admin/mod).
- **Profils utilisateurs** (avatar, alias, stats, historique, matchmaking rating).
- **Sécurité** (hashing, 2FA, validation d'entrée, régénération tokens, sessions).
- **Relations entre users** : blocage, invitations, historique chat/matchs.
- **Lien avec le frontend et le jeu** : endpoints REST + WS (profil sync, status online/offline).
- **Audit & logs** : chaque action user significative doit laisser une trace claire.

Bref, c'est la **colonne vertébrale de la cohérence côté utilisateur**, celle sur laquelle tous les autres modules s'appuient.

🌿 Git Management (discipline collective)

Tu es aussi le **gardien du dépôt Git**, autrement dit : celui qui fait en sorte que le chaos du code reste maîtrisé.

Tes responsabilités :

- **Branches propres** : chaque membre a sa branche nominative (comme tu l'as prévu), + branches de features propres (`feature/`, `fix/`, `hotfix/`).
- **Intégrations progressives** : merges réguliers vers `main`, pas de PR de 400 commits d'un coup.

- **Gestion des conflits** : tu arbitres, tu merges, tu assistes ceux qui se perdent dans les rebase.
- **Revue de qualité** : tu vérifies la lisibilité, la cohérence, et le respect des conventions avant merge.
- **Tags & versions** : tu définis les releases (`v1.0.0`, `v1.1.0`, etc.), tu gères le changelog.
- **CI/CD** : tu supervises que les pipelines GitHub Actions tournent proprement (builds, tests, lint, déploiement).
- **Formation** : tu aides les membres à mieux comprendre Git (pull, rebase, merge propre, résolution de conflits).

En bref : **chaque commit sur `main` passe sous ton radar**, et chaque problème de version ou de synchronisation doit te remonter.

⚙️ Environnement du Lead

Tu es celui qui garde **la vision globale** :

- Quand Jeanne touche au front, tu t'assures que les endpoints user soient déjà prêts côté back.

- Quand Lylou restructure le docker-compose, tu t'assures que la migration n'impacte pas les modules user/auth.

- Quand Mehdi change une politique de sécurité, tu vérifies que les tokens et permissions restent cohérents avec ton module.

- Quand Abdul améliore le gameplay ou le matchmaking, tu garantis que les profils users répercutent correctement les stats et victoires.

Tu es donc le **point de contact central** entre les sous-systèmes.

🔒 Compétences clés

- Maîtrise Git (branches, merge, conflits, tags, CI/CD).
- Solide logique backend (auth, sécurité, JWT, sessions).
- Vision architecture (modules indépendants, microservices, cohérence API).
- Bonne communication : ton rôle est aussi diplomatique que technique.
- Rigueur documentaire : chaque règle ou processus doit être écrite et suivie.

🧠 En résumé

Tu es :

> **le garant de la stabilité technique, de la cohérence du code et de la synchronisation humaine.**

En clair :

- Lylou fait tourner les serveurs,
- Jeanne rend le tout beau,
- Mehdi rend le tout sûr,
- Abdul rend le tout vivant,
- **et toi, tu fais en sorte que tout ça vive ensemble sans exploser.**

DevOps Backbone

⚙️ Rôle global : la colonne technique invisible

Le DevOps Backbone, c'est **celle qui relie tous les mondes entre eux** :
le code de Maxime (backend/auth), les interfaces de Jeanne, les sockets d'Abdul, et la sécurité de Mehdi.
Elle ne code pas "pour l'utilisateur" — elle construit **l'infrastructure qui permet aux autres de coder**.

En clair : elle s'occupe de **l'écosystème technique du projet** — celui qui garantit que tout fonctionne, partout, de manière stable, reproductible et mesurable.

🧱 Responsabilités principales

🐳 Conteneurisation & Environnements

- Créer et maintenir **les images Docker** (API, Front, DB, Reverse Proxy, Prometheus, Grafana…).
- Configurer les **Dockerfile** et le **docker-compose.yml** de développement et de production.
- S'assurer que **chaque membre** peut lancer le projet localement avec un simple `make up`.
- Mettre en place les **réseaux internes Docker** (communication inter-services : API ↔ Front ↔ DB ↔ Auth).
- Automatiser les migrations, seeds et volumes persistants (par ex. SQLite ou Postgres).

👉 Objectif : **un environnement unique, cohérent, portable** entre dev, test et prod.

⚙️ Intégration Continue (CI)

- Gérer les **GitHub Actions** (ou GitLab CI) :
  - Lint, Build, Tests automatiques.
  - Analyse statique (ESLint, Trivy, ZAP, etc.).
  - Push automatique des images Docker sur GHCR (`ghcr.io`).
- Empêcher le merge de code cassé ou non testé.
- Automatiser les PR checks (tests unitaires, e2e, lint + format).

👉 Objectif : **chaque commit testé, chaque PR vérifiée**, aucun "ça marche chez moi".

🚀 Déploiement Continu (CD)

- Construire le pipeline **build → test → deploy → notify** :
  - Sur staging → pull automatique des dernières images, relance propre.
  - Sur prod → déploiement versionné (tags Docker + release).

- Maintenir les fichiers `compose.staging.yml` et `compose.prod.yml`.

- Gérer les **secrets**, les `.env` et les **volumes sensibles** (clé JWT, OAuth, 2FA…).

- Surveiller les **migrations DB** avant déploiement (jamais casser la prod !).

👉 Objectif : **zéro déploiement manuel** et **zéro "ah j'ai oublié de rebuild"**.

📊 Observabilité & Monitoring

- Installer **Prometheus** pour collecter les métriques backend (latences, erreurs, WS, etc.).

- Configurer **Grafana** pour afficher :
  - CPU/RAM/latence des conteneurs.
  - Taux d'erreur HTTP / WS.
  - Statistiques utilisateurs & matchmaking.

- Intégrer des **alertes** (mail ou Discord) :
  - Si 5xx > 1% sur 5 min,
  - Si WS crash > seuil,
  - Si DB > 80% usage.

- Ajouter un `/health` endpoint pour chaque service.

👉 Objectif : **savoir ce qui se passe avant que l'utilisateur ne s'en rende compte**.

🧠 Microservices & Architecture réseau

- Définir les **frontières entre services** (auth, chat, matchmaking, game).
- Configurer les **reverse proxies Nginx** : HTTPS, redirection, CORS, WebSocket upgrade.
- S'assurer que chaque service communique via les bons ports & protocoles.
- Structurer le projet en microservices **scalables et isolés** (pas de dépendance directe entre backends).

👉 Objectif : un système modulaire, capable de croître sans tout casser.

🔒 Sécurité DevOps

Même si Mehdi est le responsable sécurité, **le DevOps** est celle qui applique concrètement les bonnes pratiques :

- Clés SSH & tokens GitHub : rotation régulière, stockage dans GitHub Secrets.
- Variables d'environnement chiffrées (`.env.example` propre, `.env` ignoré).
- Accès réseau limités (`expose` précis, `depends_on` maîtrisé).
- HTTPS en local via Nginx + certificats auto.
- Scan de vulnérabilités images Docker (Trivy).
- Vérification des images (signature, digest).

👉 Objectif : **sécurité by design**, sans ralentir le développement.

🪄 Outils et stack typiques

Domaine	Outil principal	Exemple
Conteneurisation	Docker / docker-compose	`make up`
CI/CD	GitHub Actions	`.github/workflows/build.yml`
Reverse proxy	Nginx	`nginx.conf`
Monitoring	Prometheus / Grafana	`localhost:9090`, `:3000`
Tests automatisés	Jest / Vitest / Cypress	CI
Linter & format	ESLint / Prettier	CI pre-merge
Alertes	Discord webhook	erreur 5xx
Scans sécurité	Trivy / Semgrep	CI quotidienne

🧩 Collaboration avec le reste de l’équipe

- Avec **Maxime (Lead)** : elle aligne les pipelines Git avec les branches, s’assure que les merges déclenchent les bons builds, et valide les déploiements finaux.
- Avec **Jeanne (Front)** : elle fournit les environnements (`VITE_API_URL` par exemple) et simplifie le lancement du front dans Docker.
- Avec **Mehdi (Sécurité)** : elle applique ses recommandations dans la CI/CD (scan deps, chiffrage clés, audit images).
- Avec **Abdul (Gameplay)** : elle gère les conteneurs WS / Game pour que les connexions soient stables.

👉 C’est littéralement **le liant technique** entre tous.

📜 Definition of Done (DoD)

- Chaque service (auth/chat/game/front) fonctionne **indépendamment** via Docker.
- CI/CD : build, tests et déploiement automatisés.
- Observabilité : métriques + dashboard Grafana.
- Secrets & `.env` gérés proprement.
- Scripts Makefile & README clairs (`make up`, `make stop`, etc.).
- Déploiement stable prod/staging validé.

🧠 En résumé

Le **DevOps Backbone**, c’est :

> “Celle qui ne fait pas de magie, mais sans qui plus rien ne marche.”

Elle :

- donne un **squelette** au projet,
- assure la **fluidité entre le code et l’infrastructure**,
- **automatise tout ce qui est répétitif**,
- et transforme un dossier GitHub en **produit déployable, traçable et observable.**

Front Backbone

🎯 Mission principale : cohérence et expérience

Le rôle de **Front Backbone** consiste à :

> Concevoir, structurer et maintenir toute l'architecture du frontend — du routage aux composants réutilisables — tout en assurant une UX fluide et accessible.

Ce n'est pas seulement "faire des pages jolies" :
c'est **rendre l'ensemble du projet utilisable, intuitif et cohérent** pour tout le monde, du simple visiteur au joueur le plus actif.

🧱 Structure et base technique

Jeanne est responsable de la **structure frontale complète**, c'est-à-dire :

- **Initialisation du projet** (Vite, React, TypeScript, TailwindCSS).
- **Arborescence claire** : `components/`, `pages/`, `hooks/`, `contexts/`, `services/`.
- Mise en place du **routage principal** (React Router) :
  - `/login`, `/register`, `/profile`, `/chat`, `/tournament`, `/play`…
- **Gestion du state global** : Context API ou Zustand pour synchroniser les données utilisateur, socket, et jeu.
- **Layout unifié** : header, sidebar, toasts, modales — tout doit suivre le même thème visuel et comportement UX.
- Gestion de la **navigation en SPA** fluide et accessible (pas de rechargement complet, animations, transitions).

👉 En résumé : elle construit **le squelette React** dans lequel tout le reste va s'emboîter.

🧩 Interfaces utilisateurs principales

Sous sa responsabilité directe :

🔐 Authentification

- Écrans de **login / register / logout / reset password**.
- Gestion des états (`loading`, `error`, `success`).
- Connexion OAuth42 et 2FA.
- Interaction propre avec les endpoints de Maxime (User Management).

👤 Profil utilisateur

- Page profil : avatar, statistiques, historique, paramètres.
- Interface d'édition du compte (changement de mot de passe, suppression du compte).
- Intégration avec le WebSocket pour statut "online/offline".

💬 Chat et notifications

- Intégration du **système de chat temps réel** : channels, DM, blocklist.
- Notifications visuelles (pop-up, toasts, icônes de badges).
- Système de **modales interactives** (invitation à jouer, alerte tournoi, etc.).
- Interaction directe avec le backend WS géré par Abdul.

🏆 Tournois et brackets

- Affichage graphique des brackets et matchs.
- Interface fluide et réactive pour suivre les résultats.
- Synchronisation en temps réel avec le backend (via WebSocket).
- Gestion de l'état en direct (victoire, défaite, mise à jour des rounds).

🕹️ Jeu intégré

- Affichage du Pong dans une zone canvas intégrée.
- Interface responsive (adaptée aux différentes tailles d'écran).
- Gestion des événements (pause, score, timer, retour au menu).
- Animation légère et soignée, sans gêner les performances.

🧠 Design System & UI Consistency

Le **Front Backbone** pose les **normes visuelles et ergonomiques** du projet :

- Définition de la **palette de couleurs**, des polices et des marges globales.
- Mise en place d'un **Design System** (boutons, inputs, modales, cartes…).
- Gestion des **états d'interaction** : hover, focus, disabled, etc.
- Création de composants réutilisables :
  `Button`, `Input`, `Modal`, `Card`, `Toast`, `Avatar`, etc.
- Suivi de la cohérence : pas de "patch CSS local" ou de composants bricolés dans un coin.

👉 En clair, tout doit "respirer le même air visuel".

🧩 Accessibilité (A11Y)

C'est un point souvent négligé — ici, c'est **une responsabilité centrale** de Jeanne :

- Respect des **normes ARIA** (roles, labels, focus, navigation clavier).
- **Contraste des couleurs** et tailles de police suffisantes.
- Validation de l'arborescence DOM pour les lecteurs d'écran.
- Tests manuels via outils comme Lighthouse, axe DevTools, etc.
- Sensibilité inclusive : cohérence entre visuel, son, et accessibilité cognitive.

👉 Le front doit être **aussi agréable à voir qu'à utiliser**, quelle que soit la personne.

⚙️ Collaboration avec les autres pôles

Collaborateur	Interaction principale
Maxime (Lead/User)	Consommation d'API auth, profil, sessions, erreurs cohérentes
Lylou (DevOps)	Configuration des environnements front (`VITE_API_URL`, build Docker, staging)
Abdul (Gameplay)	Intégration WebSocket, rendu Canvas, gestion des états du jeu
Mehdi (Cybersécurité)	Vérification des entrées front, CSP, sanitization, prévention XSS

👉 Jeanne est **le point d'entrée utilisateur** — elle traduit techniquement tout ce que les autres construisent.

🧩 Outils et technologies

Domaine	Outils / libs
Framework	React + TypeScript
Style	TailwindCSS + clsx + variables CSS
Routing	React Router
State management	Zustand / Redux Toolkit / Context API
API calls	Axios / Fetch avec interceptors JWT
Notifications	react-hot-toast / custom hooks
Charts & brackets	Recharts / custom SVG Canvas
Tests	Vitest + Testing Library + Cypress
Accessibilité	ARIA + Lighthouse + axe DevTools

✅ Definition of Done (DoD)

- Code front **typé**, **documenté**, **cohérent**.
- Design System appliqué à tout le projet.
- Composants **accessibles**, testés et réactifs.
- **Routing fluide** sans rechargement complet.
- **Dark mode** et adaptabilité (mobile, desktop).
- Intégration stable avec API et WS.
- Tests UI (unit + e2e) et **scores Lighthouse ≥ 90**.

🌟 En résumé

Le **Front Backbone**, c'est :

> **La garante de l'expérience utilisateur, de la cohérence visuelle et de la fluidité du projet.**

Elle :

- construit **l'ossature visuelle et logique** du projet,
- garantit une **interface stable, fluide et accessible**,
- veille à la **cohérence UX/UI** sur tous les écrans,
- et collabore avec tout le monde pour que **le produit final ait une âme**.