**📌 Synthèse du projet Realtime Quiz**

**1. Objectif du projet**

* Développer une application web de quiz **multijoueur en temps réel**.
* Fonctionnalités cibles :
  + Création et rejoint d’une partie avec un code.
  + Gestion d’un lobby de joueurs.
  + Démarrage du jeu par l’hôte.
  + Questions à choix multiples.
  + Système de réponses en temps réel.
  + Comptage des scores.
  + Timer par question + enchaînement automatique des rounds.
  + (À venir) gestion d’images dans les questions.

**2. Stack technique**

* **Frontend** : React + Vite + TypeScript (SPA).
* **Backend** : Node.js + Fastify + Socket.IO + TypeScript.
* **Base de données** : PostgreSQL + Prisma ORM.
* **Infra dev actuelle** : tout en local sur Windows, PostgreSQL installé nativement (pas de Docker car Virtualization non activée).
* **Déploiement futur** : VPS Ubuntu avec Caddy (HTTPS + reverse proxy). Docker est possible mais pas obligatoire.

**3. Organisation des dossiers**

quiz-realtime/

├─ web/ # frontend React

└─ server/ # backend Fastify + Socket.IO + Prisma

**4. Configuration & fichiers importants**

**4.1 Backend .env (dans server/.env)**

PORT=3000

CLIENT\_URL=http://localhost:5173

DATABASE\_URL=postgresql://dev:dev@localhost:5432/quiz?schema=public

ROUND\_MS=15000 # durée d’un round (ms) — optionnel

GAP\_MS=2500 # délai entre rounds (ms) — optionnel

**4.2 Prisma schema.prisma (dans server/prisma/schema.prisma)**

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model Game {

id String @id @default(cuid())

code String @unique

createdAt DateTime @default(now())

state String

players Player[]

questions Question[]

answers Answer[]

}

model Player {

id String @id @default(cuid())

name String

score Int @default(0)

game Game @relation(fields: [gameId], references: [id])

gameId String

answers Answer[]

}

model Question {

id String @id @default(cuid())

text String

choices Choice[]

correctId String

game Game @relation(fields: [gameId], references: [id])

gameId String

order Int

answers Answer[]

imageUrl String? // prévu pour images

}

model Choice {

id String @id @default(cuid())

label String

question Question @relation(fields: [questionId], references: [id])

questionId String

}

model Answer {

id String @id @default(cuid())

player Player @relation(fields: [playerId], references: [id])

playerId String

question Question @relation(fields: [questionId], references: [id])

questionId String

game Game @relation(fields: [gameId], references: [id])

gameId String

choiceId String

createdAt DateTime @default(now())

@@unique([playerId, questionId]) // empêche plusieurs réponses d’un joueur à une même question

}

**4.3 Seed script (dans server/prisma/seed.ts)**

* Génère un jeu avec un **code aléatoire**.
* Crée deux questions de démo :
  + *2+2 ?* (réponse : 4).
  + *Capitale de la France ?* (réponse : Paris).
* Affiche en console le code de jeu (ABCD etc.).

Commande pour lancer :

npx prisma db seed

**5. Backend (Fastify + Socket.IO)**

**Fonctionnalités actuelles**

* Endpoint /health pour vérifier que le serveur tourne.
* Gestion de la connexion socket : join\_game, start\_game, submit\_answer.
* Timer par question + enchaînement automatique :
  + round\_begin (avec question et horodatage endsAt).
  + round\_end (avec la bonne réponse + leaderboard).
  + game\_over (fin du quiz avec scores finaux).

**Stockage en mémoire**

* gameStates: conserve l’état courant d’une partie (questions, index, timers).
* clientMap: mappe un socket → joueur en DB.

**6. Frontend (React)**

**Fonctionnalités actuelles**

* Connexion au serveur WebSocket (io("http://localhost:3000")).
* Formulaire pour rejoindre une partie (code + name).
* Bouton **Start (host)** pour lancer le jeu.
* Affichage en temps réel :
  + Question et choix multiples.
  + Timer (compte à rebours).
  + Mise en évidence de la bonne réponse à la fin d’un round.
  + Scoreboard mis à jour après chaque question.
  + Message **Game over** à la fin.

**7. Postgres**

* Installé en natif sous Windows.
* Utilisateur : dev (mdp dev).
* Base : quiz.
* Prisma se connecte correctement via DATABASE\_URL.

**8. Problèmes rencontrés et résolus**

* 🚫 **Docker ne fonctionne pas** (virtualization non activée). Solution : installation native de PostgreSQL.
* 🚫 Erreurs de validation Prisma → corrigées en ajoutant les relations manquantes.
* 🚫 Oubli du prisma:seed script → corrigé avec npx prisma db seed.
* ✅ Ajout d’un **timer + auto-advance** pour enchaîner les questions.

**9. État actuel**

* ✅ Backend tourne (npm run dev dans server).
* ✅ Frontend tourne (npm run dev dans web).
* ✅ Jeu complet jouable avec plusieurs onglets :
  + Rejoint par code.
  + Questions s’affichent automatiquement avec timer.
  + Correction affichée après chaque round.
  + Scoreboard mis à jour en live.
  + Fin de partie avec classement final.

**🔮 10. Prochaines étapes possibles**

1. **Gestion d’images pour les questions**
   * Actuellement champ imageUrl prévu en DB.
   * À implémenter : upload local ou sur S3/R2 (recommandé).
   * Frontend : afficher l’image si imageUrl est présent.
2. **UI améliorée**
   * Styles (Tailwind, animations).
   * Feedback visuel sur le timer, barre de progression.
3. **Sécurité**
   * Validation des payloads avec Zod.
   * Auth simple (token par joueur ?).
4. **Admin panel**
   * CRUD questions / jeux via Prisma + API REST/GraphQL.
5. **Déploiement VPS**
   * Ubuntu + Caddy (reverse proxy, HTTPS).
   * Postgres installé via apt (ou Docker si virtualization possible).
   * Systemd services pour backend/frontend.