Résumé du projet

Le dossier présente le développement de **Randomi GO**, une adaptation web en 3D du jeu de cartes Randomi, réalisée en neuf semaines dans le cadre du Titre Professionnel de Concepteur Développeur d'Applications. L'objectif principal consiste à offrir une expérience multijoueur (2 à 6 participants) directement dans le navigateur, sur desktop comme sur mobile, tout en respectant scrupuleusement les règles et mécaniques du jeu original.

L'application se compose de trois volets interdépendants :

1. **Frontend** : site web "from scratch" pour la navigation (inscription, gestion d'amis, création et liste des salons) et zone de jeu 3D sous Three.js. Les interfaces sont conçues en HTML/CSS/JavaScript pur, avec un travail particulier sur l'accessibilité (contraste 4.5:1, RGAA) et la responsivité (media queries pour écrans de 320 px à 1920 px, adaptation du hover sur interfaces tactiles via le filtre pointer: coarse). Des custom elements permettent la modularité des panels (profil, amis, liste de salons).

2. **API & Backend** : serveur REST sécurisé, développé en Rust avec Actix Web. La station de travail inclut le hachage bcrypt des mots de passe et l'émission de JSON Web Tokens. Les middlewares gèrent CORS, journalisation et authentification, tandis qu'Utoipa génère la documentation Swagger.

3. **Service de jeu** : module dédié aux échanges temps réel via WebSocket, isolé par "hubs" par salon, chargé de la logique métier des cartes (trait Card, PlayAction, PlayInfo). Les Server-Sent Events informent l'interface d'accueil des changements de statut (nouvelle partie, demande d'ami).

La persistance combine :

- **PostgreSQL** (migrations Diesel, conventions SCREAMING_SNAKE_CASE, indexation des FK) pour les comptes, amis, statistiques, historique de parties et tokens de réinitialisation

- **PoloDB** pour le stockage NoSQL embarqué des paramètres dynamiques des salons (JSONB) et la conservation de l'historique de chat

Les tests couvrent chaque couche : tests unitaires Rust (coverage 100 % pour la logique de cartes), tests d'intégration Postman sur l'API, tests E2E sur les flux critiques, et benchs de montée en charge locale. Un workflow GitHub Actions orchestre linting, compilation, tests et build Docker. Les conteneurs (API, base, frontend) sont orchestrés via Docker Compose et déployés en rolling update sur DigitalOcean, avec protection de la branche principale et blocage en cas de régression.

Ce résumé constitue une synthèse claire et structurée des choix techniques, de l'architecture et des résultats obtenus pour le projet Randomi GO.