clean-gym-manager

Examen - Conncevoir une application de gestion de sport (Entity, Use Case, Question de réflexion)

Partie 2 :

1. Port: MemberRepository (cas d'utilisation "badge d'entrée")

Pour permettre à un membre de badger et d'entrer, le domaine a besoin d'un port de persistance simple :

```
export interface MemberRepository {
    findById(id: EntityId): Promise<Member | null>; // charger le membre par son identifiant
    save(member: Member): Promise<void>;           // persister l'état recalculé (isActive, dates)
}
```

- `findById` : récupérer l'adhérent quand il présente son badge (id).
- `save` : enregistrer l'état après les règles métier (`checkAccess` , éventuel recalcul d'activité).

Implémentations possibles (adapters) : en mémoire pour les tests, base SQL/NoSQL pour la prod.

Partie 3 : Architecture et Réflexion

1. Arborescence propre pour la clean architecture

```
src/
    domain/               # coeur métier (pur TS, sans dépendances externes)
        entities/           # Member, value objects
        errors/             # erreurs métier (EntityValidationError, ...)
        repositories/       # ports de persistance (MemberRepository.ts)
        services/           # autres ports (GymGate.ts)
        usecases/           # cas d'usage (EnterGym.ts)

    adapters/             # implémentations concrètes des ports
        repositories/
            sql/              # SqlMemberRepository.ts (via un driver/ORM)
            memory/           # InMemoryMemberRepository.ts (tests, dev)
        services/

    frameworks/
        express/            # routes/controllers
        container/
        config/             # env, log, etc.
```

- L'interface `MemberRepository` se met dans `src/domain/repositories` car elle fait partie du contrat métier.

- L'implémentation SQL `SqlMemberRepository` se met dans `src/adapters/repositories/sql/SqlMemberRepository.ts`, car elle dépend de la technologie.
- Les ports restent dans le domaine ; les implémentations concrètes vont dans les adapters ; la couche frameworks assemble le tout.

2. Tester EnterGym sans DB ni portique physique

On remplace les dépendances réelles par des versions factices. On injecte un `InMemoryMemberRepository` qui stocke les membres dans une map tout en respectant l'interface `MemberRepository`, et un `FakeGymGate` qui enregistre les appels à `open` dans un tableau sans piloter de matériel. On crée un membre actif et on le place dans le repo mémoire, puis on exécute `EnterGym.execute(memberId)` avec ces substituts. On vérifie que `FakeGymGate.open` a bien été appelée avec le bon `EntityId` ou qu'une `AccessDeniedError` ou `MemberNotFoundError` est levée dans les scénarios négatifs. De cette façon, le cas d'usage reste entièrement testable sans base de données ni portique physique.