ESGI
école supérieure de génie informatique

Document d'Architecture Logicielle

Gym Management Project

ESGI3 – 2024/2025

MORISSET Guilliam

LE BRAS Elouan

ESGI
école supérieure de génie informatique

Contents

1. Vue d’ensemble du système

1.1 Architecture générale

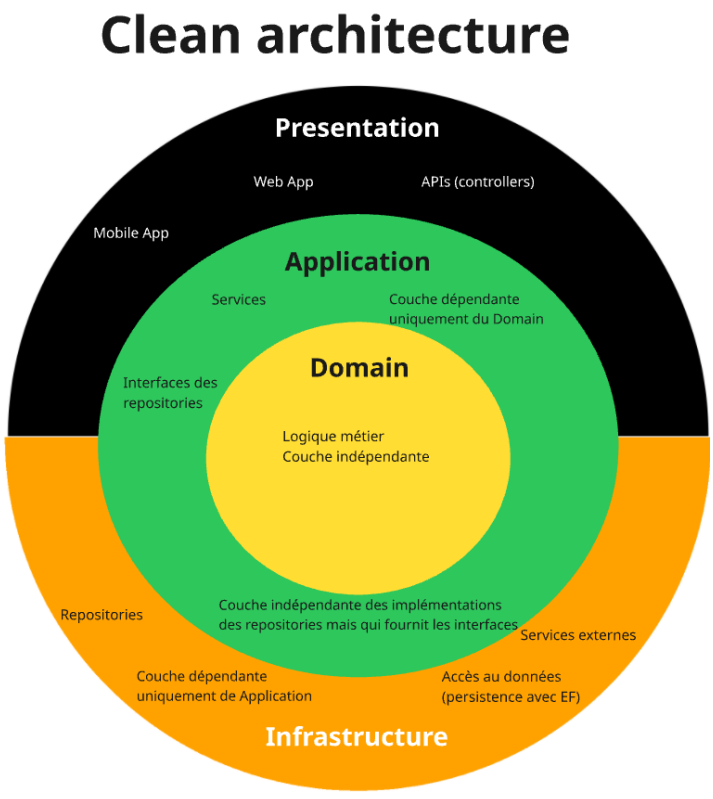
Le système Gym Management est basé sur une architecture Clean Architecture, garantissant :

- **Séparation des responsabilités** : Chaque couche a une responsabilité spécifique
- **Indépendance technologique** : Le cœur métier est indépendant des technologies externes
- **Testabilité** : Architecture facilitant les tests unitaires et d’intégration
- **Évolutivité** : Facilité d’ajout de nouvelles fonctionnalités

1.2 Acteurs du système

- **Administrateurs** : Gestion complète du système via interface web
- **Clients** : Accès aux fonctionnalités via applications web et mobile
- **Staff** : Gestion opérationnelle via interface dédiée

2. Architecture Clean Architecture



2.1 Présentation du modèle

L'architecture suit les principes de Clean Architecture avec quatre couches principales :

2.1.1 Couche Domain (Domaine)

- **Responsabilité** : Contient la logique métier pure
- **Composants** :
  - Entités métier (Club, Membership, Coaching, …)
  - Règles de gestion
  - Services de domaine
- **Caractéristiques** :
  - Indépendante de toute technologie
  - Aucune dépendance vers les couches externes
  - Contient les invariants métier

2.1.2 Couche Application

- **Responsabilité** : Orchestration des cas d'usage
- **Composants** :
  - Services applicatifs
  - Interfaces des repositories
  - DTOs (Data Transfer Objects)
  - Handlers de commandes et requêtes
- **Caractéristiques** :
  - Dépendante uniquement du domaine
  - Implémente les cas d'usage métier
  - Coordonne les appels aux services

2.1.3 Couche Infrastructure

- **Responsabilité** : Implémentation des accès aux données et services externes
- **Composants** :
  - Repositories (Entity Framework)
  - Services externes (Azure Key Vault, APIs tierces)
  - Configurations de persistence
- **Caractéristiques** :
  - Dépendante uniquement de l'application
  - Implémente les interfaces définies par l'application
  - Gère la persistance et les communications externes

2.1.4 Couche Présentation

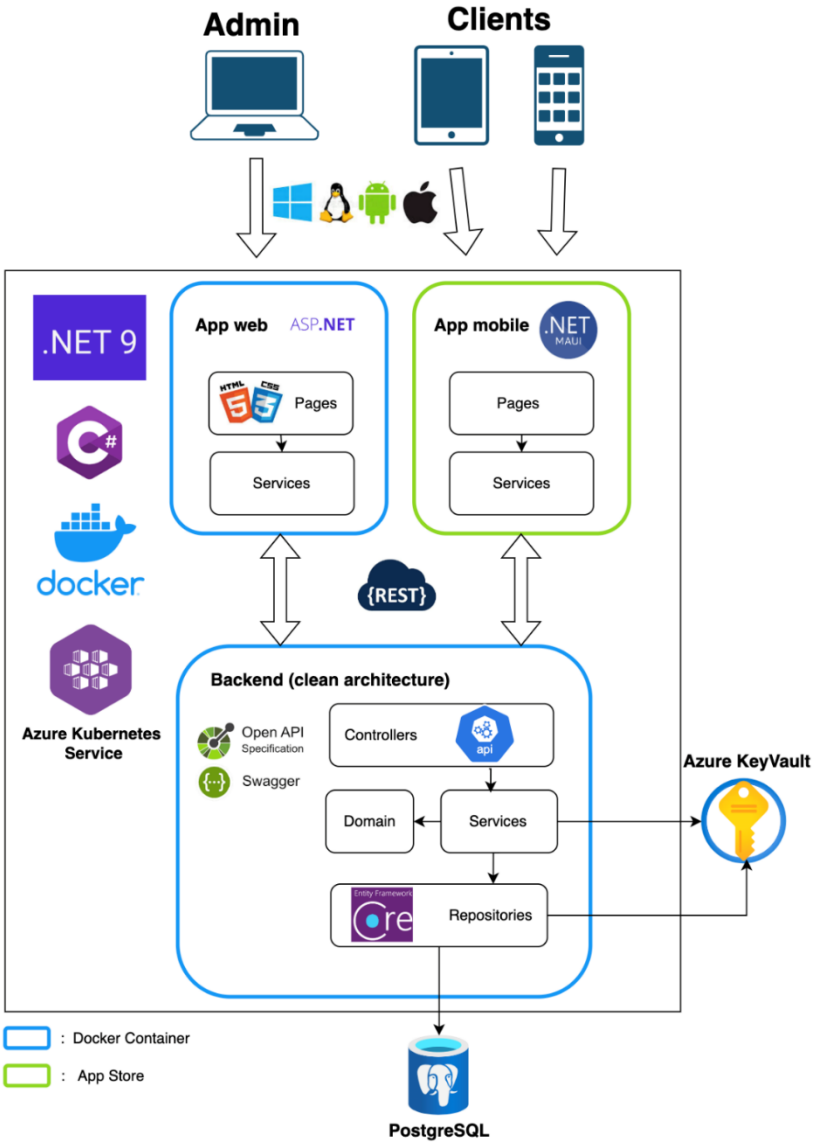
- **Responsabilité** : Interfaces utilisateur et APIs
- **Composants** :
  - Contrôleurs API REST
  - Pages web (ASP.NET)
  - Application mobile (.NET MAUI)
- **Caractéristiques** :
  - Point d'entrée du système
  - Gestion de l'authentification et autorisation
  - Transformation des données pour l'affichage

2.2 Flux de données

Le flux de données suit le principe de dépendance inversée :

1. **Requête** : Présentation → Application → Domain
2. **Persistance** : Domain → Application → Infrastructure
3. **Réponse** : Infrastructure → Application → Présentation

3. Architecture technique



3.1 Stack technologique

3.1.1 Backend

- **.NET 9** : Framework principal pour le développement
- **ASP.NET Core** : Framework web pour les APIs REST
- **Entity Framework Core** : ORM pour l'accès aux données
- **PostgreSQL** : Base de données relationnelle
- **Docker** : Conteneurisation des services

3.1.2 Frontend

- **ASP.NET Core MVC** : Application web administrative
- **.NET MAUI** : Application mobile cross-platform
- **HTML5/CSS3/JavaScript** : Technologies web standard

3.1.3 Infrastructure

- **Azure Kubernetes Service (AKS)** : Orchestration des conteneurs
- **Azure Database for PostgreSQL** : Base de données managée
- **Azure Key Vault** : Gestion sécurisée des secrets
- **Docker Hub** : Registry des images de conteneurs

3.2 Architecture des services

3.2.1 Services principaux

- **gym-api** : API principale exposant les services métier
- **gym-auth-api** : Service d'authentification et autorisation
- **gym-web-app** : Application web pour l'administration

3.2.2 Communication inter-services

- **REST APIs** : Communication synchrone entre services
- **HTTPS** : Chiffrement des communications
- **JWT Tokens** : Authentification et autorisation

3.3 Patterns architecturaux

3.3.1 CQRS (Command Query Responsibility Segregation)

- **Commandes** : Opérations de modification des données
- **Requêtes** : Opérations de lecture des données
- **Avantages** : Optimisation des performances, séparation des responsabilités

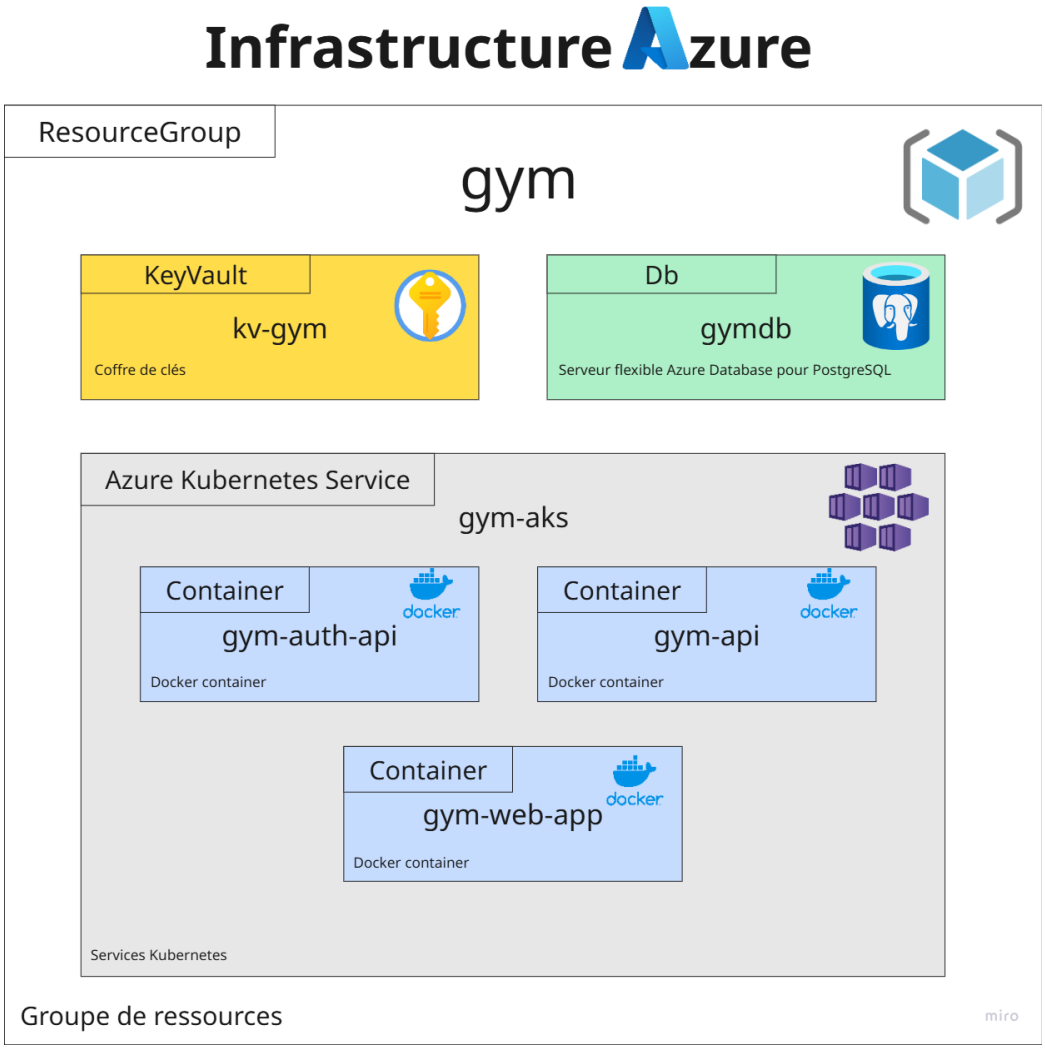
3.3.2 Repository Pattern

- **Abstraction** : Interface pour l'accès aux données
- **Implémentation** : Entity Framework avec PostgreSQL
- **Avantages** : Facilite les tests, permet le changement de technologie

3.3.3 Dependency Injection

- **Container** : ASP.NET Core DI Container
- **Avantages** : Faible couplage, facilite les tests, configuration centralisée

4. Infrastructure Azure



4.1 Groupe de ressources “gym”

L’infrastructure Azure est organisée autour d’un groupe de ressources unique contenant :

4.1.1 Azure Key Vault (kv-gym)

- **Fonction** : Stockage sécurisé des clés et secrets
- **Utilisation** :
  - Chaînes de connexion à la base de données
  - Clés de chiffrement JWT
  - Secrets des services externes
- **Avantages** : Sécurité renforcée, audit des accès

4.1.2 Azure Database for PostgreSQL (gymdb)

- **Fonction** : Base de données managée
- **Configuration** :
  - Serveur flexible pour l'optimisation des coûts
  - Sauvegardes automatiques
  - Monitoring intégré
- **Avantages** : Haute disponibilité, maintenance automatique

4.1.3 Azure Kubernetes Service (gym-aks)

- **Fonction** : Orchestration des conteneurs
- **Conteneurs déployés** :
  - gym-api : API principale
  - gym-auth-api : Service d'authentification
  - gym-web-app : Application web
- **Avantages** : Scalabilité automatique, gestion des déploiements

4.2 Sécurité infrastructure

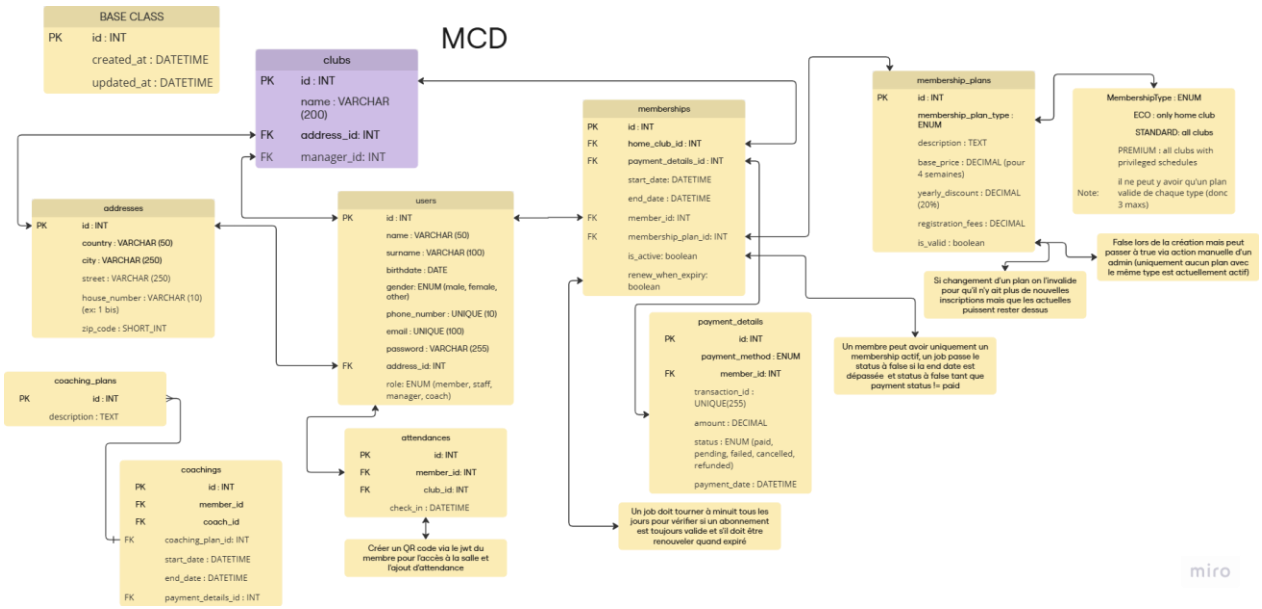
4.2.1 Network Security

- **Private Endpoints** : Accès sécurisé aux services Azure
- **Network Security Groups** : Filtrage du trafic réseau
- **Azure Firewall** : Protection périmétrique

4.2.2 Identity and Access Management

- **Azure Active Directory** : Authentification des utilisateurs
- **Role-Based Access Control (RBAC)** : Gestion des permissions
- **Managed Identity** : Authentification des services

5. Modèle conceptuel de données



5.1 Entités principales

5.1.1 Gestion des utilisateurs

- **users** : Utilisateurs du système (admin, staff, clients)
- **clubs** : Établissements de fitness
- **memberships** : Abonnements Utilisateurs

5.1.2 Gestion des abonnements

- **membership_plans** : Plans d'abonnement disponibles
- **payment_details** : Détails de paiement des abonnements
- **MembershipType** : Types d'abonnement (Standard, Premium, VIP)

5.1.3 Gestion des séances

- **coachings** : Séances d'entraînement
- **coaching_plans** : Plans de coaching
- **attendances** : Présences aux séances

5.2 Relations et cardinalités

5.2.1 Relations principales

- **users (1) ←→ (n) memberships** : Un utilisateur peut avoir plusieurs adhésions
- **clubs (1) ←→ (n) memberships** : Un club peut avoir plusieurs membres
- **memberships (1) ←→ (n) payment_details** : Une adhésion peut avoir plusieurs paiements
- **coachings (1) ←→ (n) attendances** : Une séance peut avoir plusieurs participants

5.2.2 Contraintes métier

- **Intégrité référentielle** : Clés étrangères avec contraintes CASCADE
- **Contraintes de domaine** : Validation des données (emails, dates)
- **Contraintes d'unicité** : Prévention des doublons

5.3 Optimisations base de données

5.3.1 Indexation

- **Index primaires** : Clés primaires sur toutes les tables
- **Index secondaires** : Colonnes fréquemment utilisées dans les requêtes
- **Index composites** : Optimisation des requêtes multi-colonnes

5.3.2 Partitioning

- **Partitioning par date** : Tables d'historique (paiements, présences)
- **Partitioning par club** : Données spécifiques aux établissements

6. Avantages de l'architecture

6.1 Découpage en couches

Avantage : Séparation claire des responsabilités

- **Domain** : Logique métier pure
- **Application** : Cas d'usage et orchestration
- **Infrastructure** : Accès aux données et services externes
- **Presentation** : Interfaces utilisateur

Bénéfice : Maintenance facilitée, compréhension du code améliorée

6.2 Facilité de test

Avantage : Architecture testable par design

- **Tests unitaires** : Logique métier isolée
- **Tests d'intégration** : APIs et bases de données
- **Tests end-to-end** : Parcours utilisateur complets

Bénéfice : Qualité du code, réduction des régressions

6.3 Indépendance des services externes

Avantage : Isolation des dépendances externes

- **Interfaces** : Abstraction des services externes
- **Dependency Injection** : Implémentations interchangeables
- **Mocking** : Facilite les tests sans dépendances

Bénéfice : Flexibilité, résilience aux changements externes

6.4 Indépendance de l'UI et des APIs

Avantage : Séparation entre logique métier et présentation

- **Multiple UI** : Web, mobile, desktop possibles
- **API First** : Design orienté API
- **Versioning** : Évolution indépendante des interfaces

Bénéfice : Réutilisabilité, évolutivité des interfaces

6.5 Domain Driven Design (DDD)

Avantage : Conception centrée sur le métier

- **Ubiquitous Language** : Vocabulaire commun équipe/métier
- **Bounded Context** : Délimitation claire des domaines
- **Aggregate** : Cohérence des données métier

Bénéfice : Alignement avec les besoins métier, maintenabilité

6.6 Facilité de modification

Avantage : Changement d'implémentation sans impact

- **Exemple** : Passage de PostgreSQL à MySQL
- **Processus** : Modification de la couche Infrastructure uniquement
- **Impact** : Aucun changement dans les couches supérieures

Bénéfice : Adaptabilité technologique, réduction des coûts de migration

6.7 Évolutivité

Avantage : Architecture préparée pour la croissance

- **Horizontal scaling** : Ajout de nouvelles fonctionnalités
- **Vertical scaling** : Amélioration des performances
- **Microservices** : Évolution possible vers une architecture distribuée

Bénéfice : Pérennité de l'investissement, croissance soutenue