

# QCM — Architectures logicielles

---

## Question 1 — Monolithe

---

Dans une architecture monolithique, quel est le **principal risque** lors de l'évolution du projet ?

- ☐ A. Une faible performance due aux appels réseau fréquents
  - ☐ B. Une dette technique accrue à cause du couplage fort
  - ☐ C. La difficulté d'assurer la cohérence transactionnelle
  - ☐ D. L'absence de compatibilité avec une base de données relationnelle
- 

## Question 2 — Architecture n-tiers

---

Dans une architecture 3-tiers classique (présentation / métier / données), à quoi sert la **couche métier** ?

- ☐ A. À stocker les données de manière persistante
  - ☐ B. À fournir une interface graphique réactive
  - ☐ C. À encapsuler la logique métier et les règles de gestion
  - ☐ D. À gérer la scalabilité horizontale
- 

## Question 3 — Dépendances en n-tiers

---

Dans une architecture n-tiers, les dépendances doivent généralement aller :

- ☐ A. De la couche présentation vers la couche métier, puis vers la couche données
  - ☐ B. Dans les deux sens, afin de garder la flexibilité
  - ☐ C. De la base de données vers les services, puis vers l'UI
  - ☐ D. Directement de l'UI vers la base de données pour éviter les surcouches inutiles
- 

## Question 4 — Architecture Clean

---

Selon l'architecture Clean, où doivent se trouver les **Entities** ?

- ☐ A. Dans l'infrastructure, aux côtés des repositories
  - ☐ B. Dans la couche la plus externe, proche des frameworks
  - ☐ C. Au cœur du domaine, indépendantes de toute technologie
  - ☐ D. Dans les cas d'usage, pour être testées facilement
- 

## Question 5 — Cas d'usage (Clean)

---

Quel est le rôle principal d'un **Use Case** dans l'architecture Clean ?

- ☐ A. Orchestrer une logique métier à partir d'entrées externes
  - ☐ B. Gérer la persistance des données dans la base
  - ☐ C. Fournir des composants graphiques pour l'IHM
  - ☐ D. Contrôler les threads utilisés par l'application
- 

## Question 6 — Architecture hexagonale

---

Dans une architecture hexagonale, un **port OUT** correspond à :

- ☐ A. Une API publique exposée aux clients
  - ☐ B. Une dépendance dont le core a besoin (ex. repository, mailer)
  - ☐ C. Un adaptateur de présentation (UI, REST, CLI)
  - ☐ D. Une entité métier utilisée par un cas d'usage
-

## Question 7 – Adaptateurs (Hexagonale)

---

Dans l'architecture hexagonale, quel est le rôle des **adapters** ?

- ☐ A. Traduire entre le core et le monde extérieur (infra, UI, DB...)
  - ☐ B. Simplifier la logique métier en supprimant les règles de gestion
  - ☐ C. Optimiser les performances réseau
  - ☐ D. Remplacer les entités métiers par des DTO
- 

## Question 8 – Microservices

---

Quelle est la **principale différence** entre une architecture microservices et une architecture n-tiers ?

- ☐ A. Les microservices imposent toujours l'usage de bases NoSQL
  - ☐ B. Les microservices séparent le code par couches, pas par domaine
  - ☐ C. Les microservices isolent chaque contexte métier avec sa persistance
  - ☐ D. Les microservices ne permettent pas de tests unitaires
- 

## Question 9 – Contrôles et cohérence

---

Dans une architecture microservices, comment assurer la **cohérence des données** entre services ?

- ☐ A. En partageant une seule base de données centrale
  - ☐ B. En utilisant des transactions globales synchrones
  - ☐ C. En s'appuyant sur des mécanismes de communication (événements, messages)
  - ☐ D. En supprimant les contraintes de cohérence
- 

## Question 10 – Dépendances et inversion de contrôle

---

Quel principe permet de garantir que le code métier reste indépendant de la technique (DB, frameworks) ?

- ☐ A. Le principe de Single Responsibility
- ☐ B. L'inversion de dépendances (Dependency Inversion)
- ☐ C. Le polymorphisme d'héritage
- ☐ D. Le principe de Liskov