Examen Architecture Logiciel

Exercice 1:

- Définissons les concepts suivants:

Architecture:

Une architecture dans un projet informatique désigne la structure globale du projet.
C'est -à -dire comment sont organisés les différents composants du projet et comment ils interagissent entre eux.

Architecture par couches:

C’est un modèle de conception logicielle qui divise un système en différentes couches fonctionnelles distinctes. Chaque couche est responsable d'une partie spécifique du système et communique avec les couches adjacentes selon un ensemble prédéfini de règles et d'interfaces.

Exemple:

- Couche de présentation (interface utilisateur) :
- Couche de logique métier (ou couche applicative)
- Couche d'accès aux données (ou couche de persistance)

Couplage Fort:

Le couplage fort fait référence à une dépendance forte entre deux composants ou modules d'un système. Lorsqu'il y a un couplage fort entre deux éléments, un changement dans l'un des éléments a un impact direct sur l'autre, ce qui rend les deux éléments fortement interdépendants.

Couplage Faible:

Le couplage faible fait référence à une dépendance limitée entre deux composants ou modules d'un système. Lorsqu'il y a un couplage faible entre deux éléments, ils sont moins interdépendants et ont une connaissance limitée des détails de mise en œuvre spécifiques de l'autre.

Injection de dépendances:

Dans le cadre de l'injection de dépendance, les dépendances d'un objet sont injectées ou fournies par un autre objet plutôt que d'être créées ou gérées par l'objet lui-même. Cela permet de rendre les objets indépendants des détails d'implémentation de leurs dépendances et facilite leur réutilisation et leur testabilité.

Faisons une étude comparative entre une architecture monolithique et celle en micro-service.

Architecture Monolithique :

- Dans une architecture monolithique, l'application est construite comme une seule unité logicielle, où tous les composants sont regroupés et déployés ensemble.
- Tous les modules et fonctionnalités de l'application sont intégrés dans un seul code source, partageant la même base de données et le même environnement d'exécution.
- La communication entre les différentes parties de l'application se fait généralement par des appels de fonction internes.
- Le déploiement de l'application monolithique se fait en une seule fois, ce qui rend les mises à jour et les déploiements plus lents et plus risqués.
- L'évolutivité de l'application monolithique peut être limitée, car toute modification ou ajout de fonctionnalité nécessite la recompilation et le redéploiement de l'ensemble de l'application.
- Les tests unitaires sont généralement plus complexes, car il est difficile d'isoler les différentes parties de l'application pour les tester individuellement.
- En cas de panne ou de problème, toute l'application peut être affectée, car une seule instance est en cours d'exécution.

Architecture Microservices :

- Dans une architecture microservices, l'application est divisée en un ensemble de petits services indépendants, chacun avec sa propre logique métier et son propre déploiement.
- Chaque service est développé, déployé et mis à l'échelle de manière indépendante. Ils peuvent être écrits dans différents langages de programmation et utiliser différentes technologies.
- Les services communiquent généralement entre eux via des API (interfaces de programmation d'application) ou des messages asynchrones.

- Les mises à jour et les déploiements peuvent être plus rapides et moins risqués, car seuls les services concernés par les modifications doivent être déployés.
- L'évolutivité est améliorée, car chaque service peut être mis à l'échelle indépendamment en fonction de la demande.
- Les tests unitaires sont plus faciles à réaliser, car chaque service peut être testé individuellement et isolément.
- En cas de panne ou de problème, seuls les services touchés sont affectés, les autres continuent de fonctionner normalement.
- La complexité opérationnelle peut augmenter, car il est nécessaire de gérer plusieurs services et de gérer la communication et la coordination entre eux.
- Une bonne conception et une gestion efficace des interfaces et de la communication entre les services sont essentielles pour garantir la cohérence et la fiabilité de l'application.