Cours Inf3522 – Développement d'Applications N–tiers

Cours Inf3523 – Architecture et Génie des Logiciels

Cours_0 : Introduction

Pensez à un tiers comme une séparation logique et physique des composants dans une application ou un service. Cette séparation est au niveau des composants, pas au niveau du code.

Que veut dire composants ?

Base de données

Serveur d'application en backend

Interface utilisateur

Messagerie

Caching etc.

Ce sont les composants qui constituent un service Web.

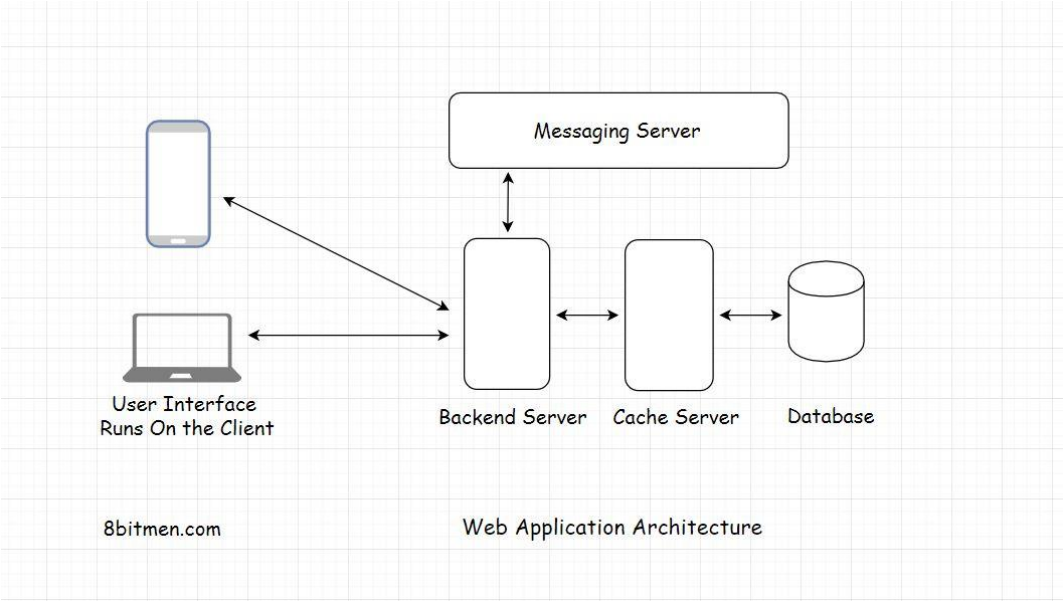

Illustration 1.1 : Architecture d'application web

Maintenant, examinons les différents types de tiers ou niveaux et leurs exemples concrets.

Un tiers

Dans une application à un seul tiers, l'interface utilisateur, la logique métier backend et la base de données résident sur la même machine.

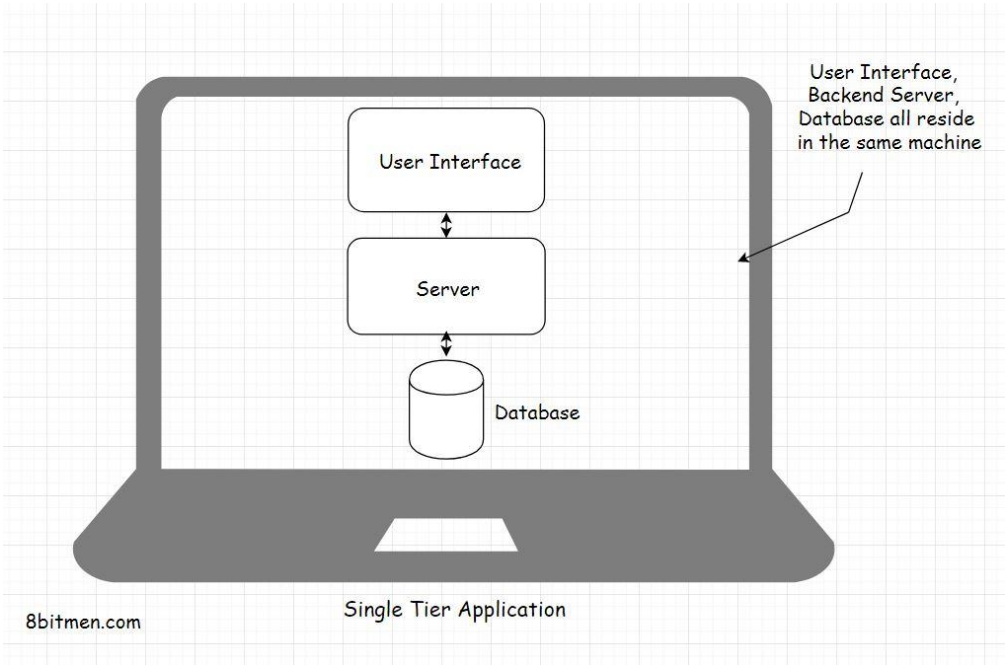

Illustration 1.2 : Application un tiers

Des exemples typiques d'applications à un seul tiers sont les applications de bureau telles que MS Office, les jeux pour PC, les logiciels de retouche d'images tels que Gimp, Photoshop, etc.

Avantages des applications à un seul tiers

Le principal avantage des applications à un seul tiers est qu'elles n'ont pas de latence réseau car chaque composant est situé sur la même machine. Cela se traduit par des performances accrues du logiciel.

Les applications à deux, trois et plusieurs tiers doivent souvent envoyer des demandes de données au serveur backend. Cela ajoute une latence réseau au système, ce qui rend l'expérience utilisateur plus lente par rapport aux applications à un seul tiers. Dans les applications à un seul tiers, les données sont facilement disponibles car tous les composants sont situés sur la même machine.

Cependant, la performance réelle d'une application à un seul tiers dépend largement des exigences matérielles de l'application et de la puissance de la machine sur laquelle elle s'exécute.

De plus, en ce qui concerne la confidentialité et la sécurité des données, elle est de la plus haute importance dans les applications à un seul tiers car les données de l'utilisateur restent toujours sur sa machine et n'ont pas besoin d'être transmises sur un réseau pour être persistantes.

Inconvénients des applications à un seul tiers

Le gros inconvénient des applications à un seul tiers est que l'éditeur de l'application n'a aucun contrôle sur l'application. Une fois que le logiciel est livré, aucun code ou fonctionnalité ne peut être mis à jour jusqu'à ce que le client le mette à jour manuellement en se connectant au serveur distant ou en téléchargeant et en installant un correctif.

En raison de cela, dans les années 90, les éditeurs de jeux vidéo ne pouvaient rien faire si leur jeu était expédié avec un code bogué. Ils ont finalement dû faire face à beaucoup de critiques de la part des joueurs en raison de la nature boguée de leur logiciel. Cela a rendu les tests de produits essentiels,

responsables de faire ou de défaire une entreprise. Les tests de logiciels devaient être approfondis car il n'y avait pas de place pour les erreurs.

Le code dans les applications à un seul tiers est également vulnérable à être modifié et inversé. La sécurité du produit pour l'éditeur de l'application est minimale. Une personne malveillante peut, avec un peu d'effort, accéder au code source de l'application, le modifier ou le copier à des fins lucratives. Cela est peu probable dans une architecture où l'entreprise contrôle le serveur d'application et met en œuvre une sécurité pour se protéger des pirates informatiques.

Enfin, les performances, l'apparence et la convivialité des applications à un seul tiers peuvent être inconsistantes car le rendu de l'application dépend largement de la configuration de la machine de l'utilisateur.

Deux tiers

Une application à deux tiers implique un client et un serveur. Le client contient l'interface utilisateur avec la logique métier sur une machine. Pendant ce temps, le serveur backend comprend la base de données qui s'exécute sur une machine différente. Le serveur de base de données est hébergé par l'entreprise qui en a le contrôle.

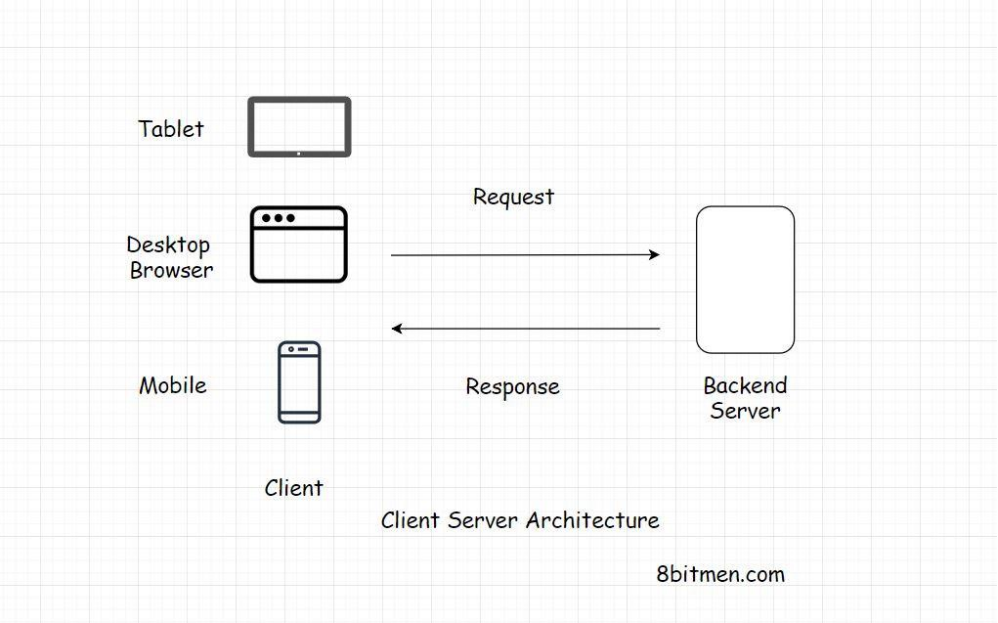

Illustration 1.3 : Application client serveur

Pourquoi avons-nous besoin d'applications à deux tiers? Pourquoi ne pas héberger la logique métier sur une machine différente et en avoir également le contrôle?

Aussi, le code de l'application n'est-il pas vulnérable à l'accès par une tierce personne?

La nécessité d'applications à deux tiers

Eh bien, oui ! Le code est vulnérable. Cependant, il existe des cas d'utilisation où les applications à deux tiers sont pratiques, par exemple, une application de liste de tâches ou un planificateur similaire ou une application de productivité.

Dans ces scénarios, même si le code est accessible par une tierce personne, cela ne causera pas beaucoup de préjudice à l'entreprise. Au contraire, puisque la logique métier et l'interface utilisateur

résident sur la même machine, il y a moins d'appels réseau vers le serveur backend. Cela maintient la latence de l'application faible, ce qui constitue un avantage.

Prenons une application de liste de tâches comme exemple. L'application appelle le serveur de base de données uniquement lorsque l'utilisateur a fini de créer sa liste et souhaite persister les données.

Un autre bon exemple d'applications à deux tiers est les jeux basés sur le navigateur et l'application mobile. Les fichiers de jeu sont assez volumineux, et ils ne sont téléchargés sur le client qu'une seule fois lorsque l'utilisateur utilise l'application pour la première fois. Et ils effectuent les appels réseau vers le backend uniquement pour persister l'état du jeu.

De plus, moins d'appels de serveur signifient moins d'argent dépensé pour maintenir les serveurs en fonctionnement, ce qui est naturellement économique. Cependant, le choix de ce type de tiers pour notre service dépend largement de nos exigences commerciales et du cas d'utilisation. Lors de la conception de notre système, nous pouvons choisir de garder l'interface utilisateur et la logique métier sur le client ou de déplacer la logique métier vers un serveur backend dédié, en faisant une application à trois tiers.

Trois tiers

Les applications à trois tiers sont assez populaires et largement utilisées sur le Web. Presque tous les sites Web simples comme les blogs, les sites d'actualités, etc., font partie de cette catégorie.

Dans une application à trois tiers, l'interface utilisateur, la logique métier et la base de données résident toutes sur des machines différentes et ont donc des niveaux différents. Ils sont physiquement séparés.

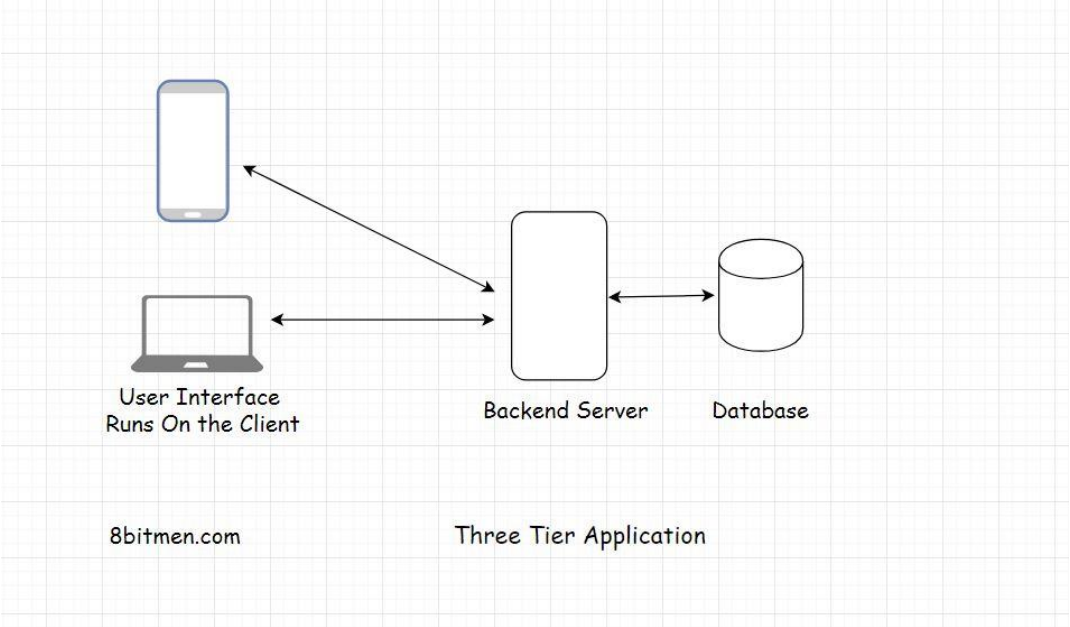

Illustration 1.4 : Application trois tiers

Prenons l'exemple d'un simple blog. L'interface utilisateur sera écrite en HTML, JavaScript et CSS, la logique d'application côté serveur sera exécutée sur un serveur comme Apache et la base de données sera MySQL. Une architecture à trois tiers fonctionne mieux pour les cas d'utilisation simples.

N tiers

Une application en plusieurs tiers est une application qui comprend plus de trois composants (interface utilisateur, serveur backend, base de données) dans son architecture.

Quels sont ces composants ?

Cache

Files de messages pour un comportement asynchrone

Équilibreurs de charge

Serveurs de recherche pour effectuer des recherches dans des quantités massives de données

Composants impliqués dans le traitement de quantités massives de données

Composants exécutant une technologie hétérogène communément appelée services web, microservices, etc.

Toutes les applications sociales telles qu'Instagram, Facebook, TikTok, les services grand public à grande échelle tels qu'Uber, Airbnb, les jeux en ligne massivement multijoueurs tels que Pokémon Go, Roblox, etc. sont des applications en plusieurs tiers. Les applications en plusieurs tiers sont plus couramment connues sous le nom de systèmes distribués.

Maintenant, comprenons la nécessité de tant de tiers.

Quel est le besoin de tant de tiers?

Deux principes de conception de logiciels clés pour expliquer cela sont le *principe de responsabilité unique et la séparation des préoccupations*.

Principe de responsabilité unique

Le principe de responsabilité unique signifie donner une responsabilité dédiée à un composant et le laisser l'exécuter sans faille, que ce soit pour enregistrer des données, exécuter la logique de l'application ou assurer la livraison des messages dans tout le système.

Cette approche nous donne beaucoup de flexibilité, rendant la gestion facile. Nous pouvons avoir des équipes et des référentiels de code dédiés pour les composants individuels, en gardant les choses plus propres.

Avec le principe de responsabilité unique, les composants sont faiblement couplés en termes de responsabilité et apporter des modifications à l'un d'entre eux n'affecte pas la fonctionnalité des autres composants. Par exemple, la mise à niveau d'un serveur de base de données avec un nouveau système d'exploitation ou un correctif n'affectera pas les autres composants de service. Même si quelque chose se passe mal pendant l'installation du système d'exploitation, seul la base de données sera hors ligne. L'application dans son ensemble restera en ligne et seuls les fonctionnalités nécessitant la base de données seront impactées.

Procédures stockées

Les procédures stockées nous permettent d'ajouter de la logique métier à la base de données, ce qui ne respecte pas le principe susmentionné. Car si, à l'avenir, nous voulons introduire une base de données différente ? Où prenons-nous la logique métier ? La prenons-nous dans la nouvelle base de

données ? Ou révisons-nous le code de l'application et insérons-nous la logique de la procédure stockée quelque part ?

Une base de données ne doit pas contenir de logique métier. Elle doit seulement s'occuper de la persistance des données. C'est cela le principe de responsabilité unique, c'est pourquoi nous avons des tiers séparés pour des composants séparés.

Séparation des préoccupations

La séparation des préoccupations signifie, de manière générale, se préoccuper uniquement de son travail et arrêter de se soucier du reste. Ces principes s'appliquent à tous les niveaux du service, que ce soit au niveau des tiers ou du code.

Le fait de séparer les composants les rend réutilisables. Différents services peuvent utiliser la même base de données, serveur de messagerie ou tout autre composant tant qu'ils ne sont pas étroitement liés les uns aux autres.

Avoir des composants faiblement couplés nous permet de faire évoluer facilement notre service lorsque les choses dépassent une certaine échelle à l'avenir.

Différence entre les couches et les tiers

Note : Ne confondez pas les tiers avec les couches de l'application. Certains préfèrent les utiliser de manière interchangeable. Cependant, dans l'industrie, les couches de l'application représentent généralement les couches d'interface utilisateur, de logique métier, de service et d'accès aux données.

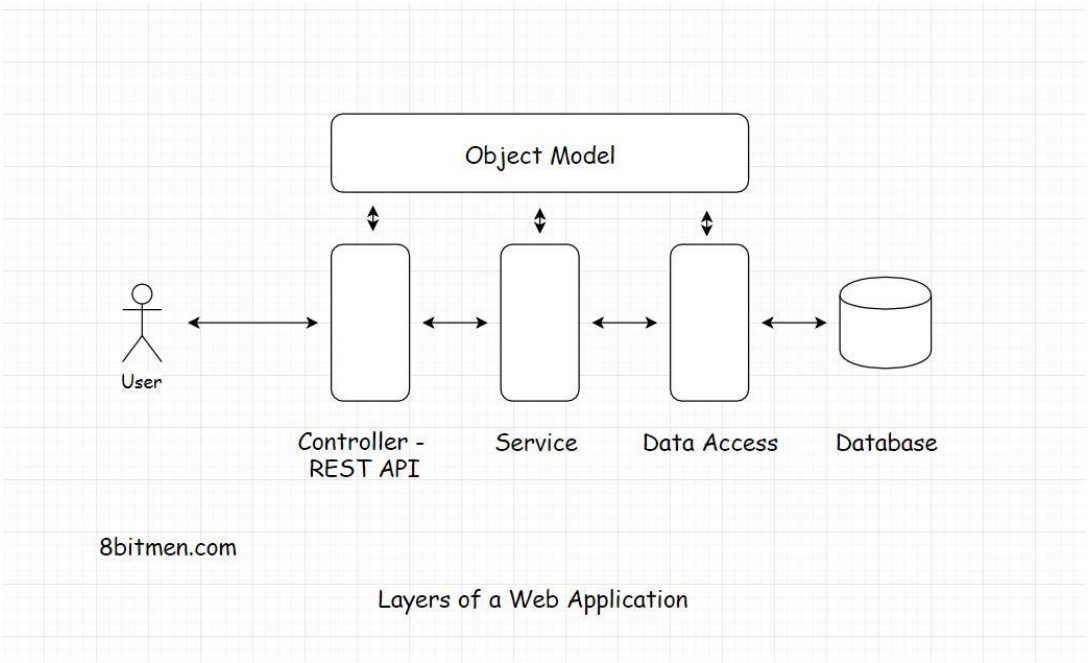

Illustration 1.5 : Application trois tiers

Ces couches se situent au niveau du code. La différence entre les couches et les tiers est que les couches représentent l'organisation conceptuelle/logique du code, tandis que les tiers représentent la séparation physique des composants.

Toutes ces couches peuvent être utilisées dans n'importe quelle application à tiers. Qu'elle soit à un tiers, deux tiers, trois tiers ou n tiers.