Cours Inf3522 – Développement d'Applications N–tiers

Cours Inf3523 – Architecture et Génie des Logiciels

Cours_1 : Architecture Web

Une architecture web implique plusieurs composants tels qu'une base de données, une file de messages, un cache, une interface utilisateur, etc., tous fonctionnant conjointement pour former un service en ligne.

Qu'est-ce que l'architecture web ?

Il s'agit d'une architecture d'application web typique utilisée dans la plupart des applications en ligne. Si vous comprenez les composants impliqués dans ce diagramme, vous pouvez construire sur cette architecture pour répondre à des exigences complexes.

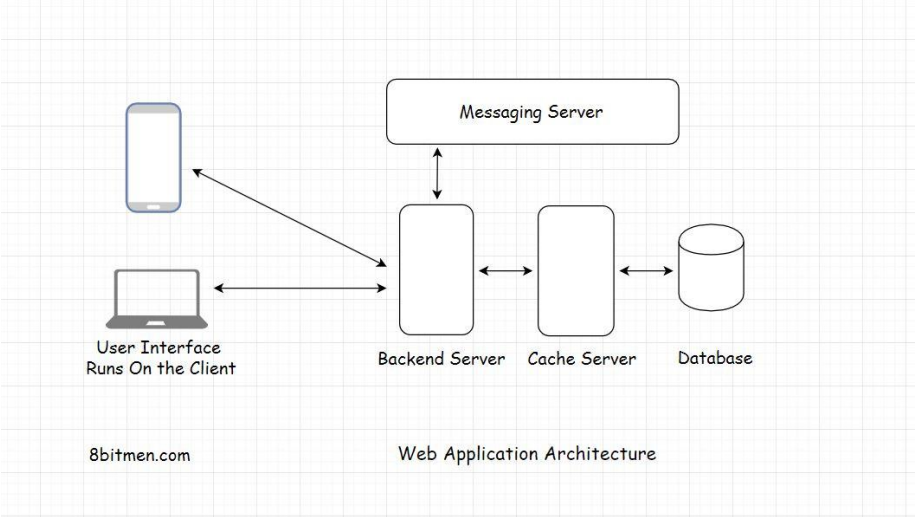

Illustration 1.6 : Architecture d'une application web

Architecture client-serveur

Vous avez déjà appris un peu sur l'architecture client-serveur lorsque nous avons discuté de l'architecture à deux, trois et n-tiers. Maintenant, nous allons l'examiner en détail.

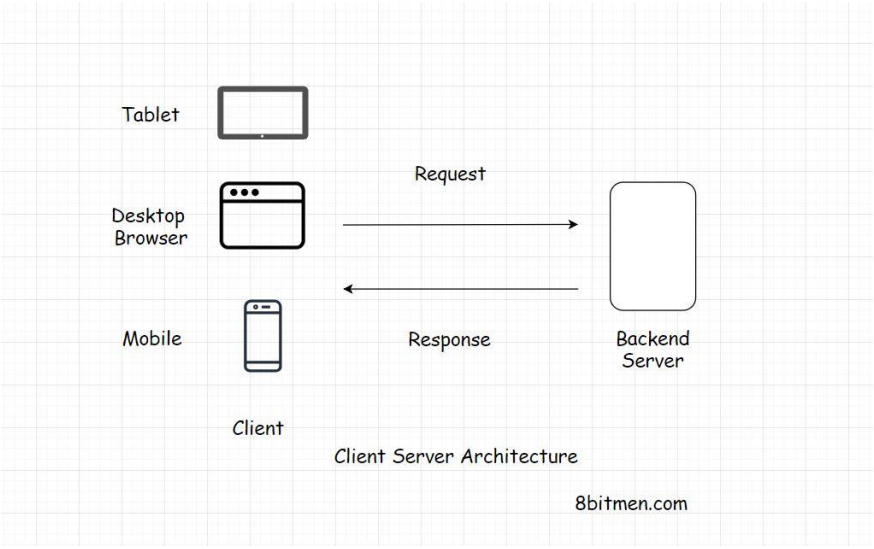

Illustration 1.7: Architecture client-serveur

L'architecture client-serveur est le bloc de construction fondamental du Web.

L'architecture fonctionne sur un modèle de requête-réponse. Le client envoie la requête au serveur pour obtenir des informations et le serveur y répond.

Tous les sites Web que vous parcourez, qu'il s'agisse d'un blog WordPress, d'une application comme Facebook, Twitter ou votre application bancaire, sont construits sur l'architecture client-serveur.

Un très petit pourcentage de sites Web et d'applications d'entreprise utilisent l'architecture peer-to-peer, qui diffère de l'architecture client-serveur.

Nous en discuterons plus tard dans le cours. Pour l'instant, concentrons-nous sur le client.

Client

Le client contient notre interface utilisateur. L'interface utilisateur est la partie présentation de l'application. Elle est écrite en HTML, JavaScript, CSS et est responsable de l'apparence de l'application.

L'interface utilisateur s'exécute sur le client. En termes simples, un client est une passerelle vers notre application. Il peut s'agir d'une application mobile, d'un ordinateur de bureau ou d'une tablette comme un iPad. Il peut également s'agir d'une console basée sur le web, exécutant des commandes pour interagir avec le serveur backend.

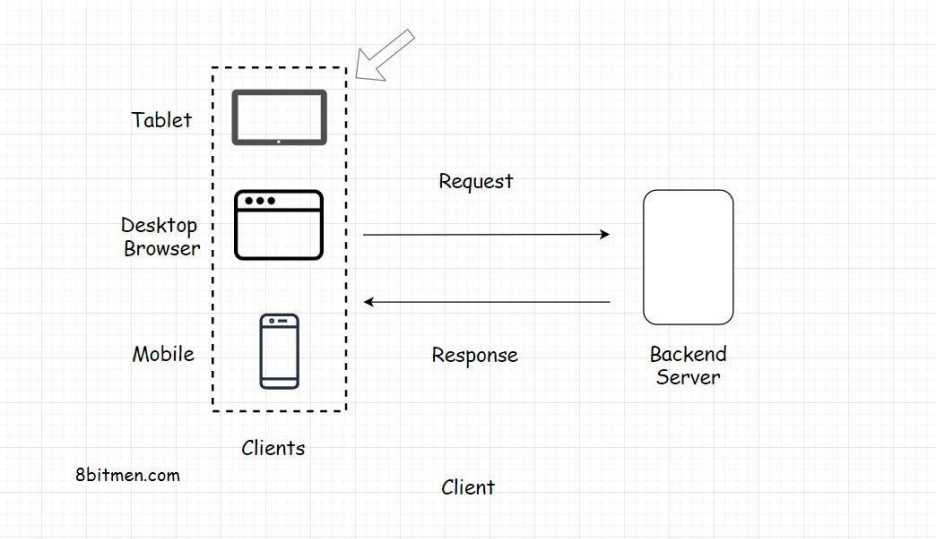

Illustration 1.8: Client

Technologies utilisées pour implémenter des clients dans les applications web

Les technologies open-source populaires pour écrire l'interface utilisateur basée sur le web dans l'industrie sont Vanilla JavaScript, jQuery, React, Angular, Vue, Flutter, etc. Toutes ces bibliothèques sont écrites en JavaScript, sauf Flutter écrit avec le langage Dart développé par Google.

Il existe une pléthore d'autres technologies qui peuvent également être exploitées pour écrire la partie frontend.

Différentes plateformes nécessitent des frameworks et des bibliothèques différents pour écrire la partie frontend de notre application. Par exemple, les téléphones mobiles fonctionnant sous Android ont besoin d'un ensemble d'outils différents de ceux fonctionnant sous Apple ou Windows OS.

Types de clients

Il y a principalement deux types de clients :

Client léger

Client lourd (parfois appelé client riche)

Client léger

Un client léger est un client qui ne contient que l'interface utilisateur de l'application. Il ne contient aucune logique métier de quelque sorte que ce soit. Pour chaque action, le client envoie une requête au serveur backend, tout comme dans une application à trois niveaux.

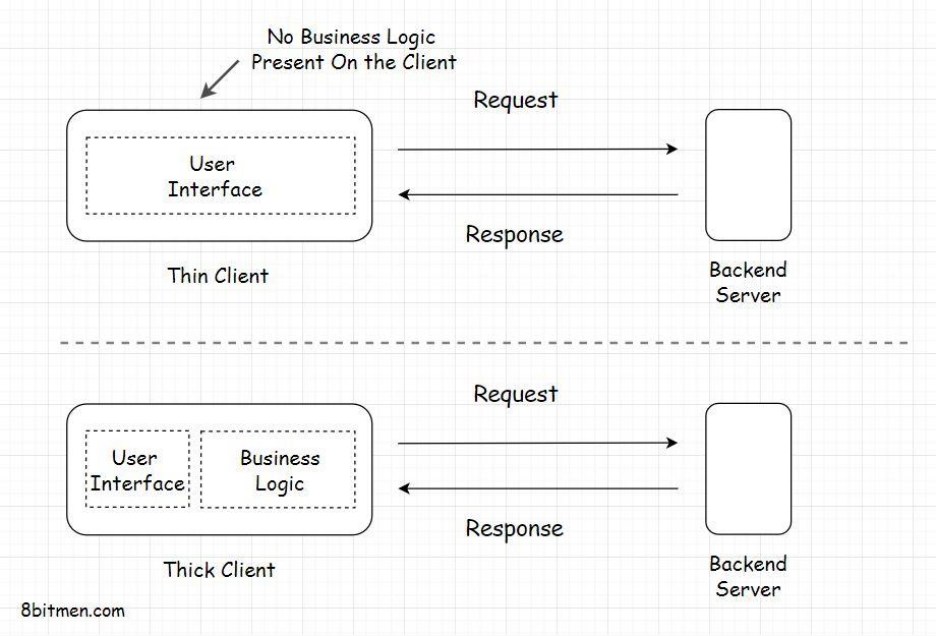

Illustration 1.9 : Client léger - Client lourd

Client lourd

En revanche, le client lourd contient toute ou une partie de la logique métier. Ce sont des applications à deux tiers. Nous en avons déjà parlé.

Les exemples typiques de clients lourds sont les applications utilitaires, les jeux en ligne, et ainsi de suite.

Serveur

Qu'est-ce qu'un serveur web ?

La tâche principale d'un serveur web est de recevoir les requêtes du client et de fournir la réponse après l'exécution de la logique métier en fonction des paramètres de la demande reçue du client.

Chaque service en ligne a besoin d'un serveur pour fonctionner. Les serveurs exécutant des applications web sont couramment appelés serveurs d'application.

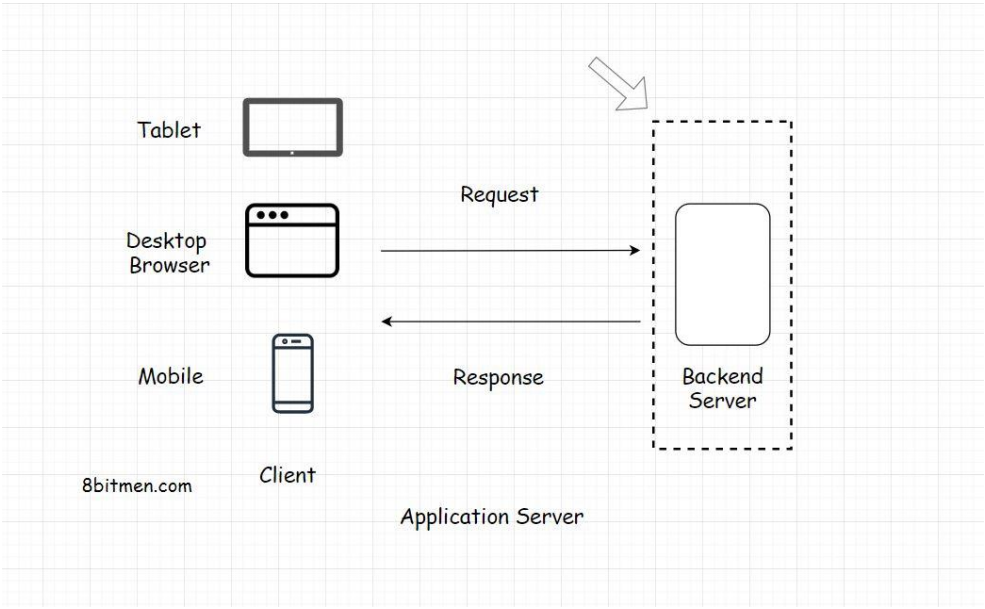

Illustration 1.10 : Serveur d'application

Outre les serveurs d'application, il existe également d'autres types de serveurs avec des tâches spécifiques attribuées. Ceux-ci comprennent :

Serveur proxy

Serveur de messagerie

Serveur de fichiers

Serveur virtuel

Serveur de stockage de données

Serveur de tâches en batch, etc.

La configuration et le type de serveur peuvent différer en fonction du cas d'utilisation. Par exemple, si nous exécutons un code d'application backend écrit en Java, nous choisirions Apache Tomcat ou Jetty. Pour des cas d'utilisation simples tels que l'hébergement de sites web, nous choisirions le serveur HTTP Apache.

Dans ce cours, nous nous en tiendrons au serveur d'application.

Tous les composants d'une application web ont besoin d'un serveur pour fonctionner, qu'il s'agisse d'une base de données, d'une file d'attente de messages, d'un cache ou de tout autre composant. Dans le développement d'applications modernes, même l'interface utilisateur est hébergée séparément sur un serveur dédié.

Rendu côté serveur

Souvent, les développeurs utilisent un serveur pour rendre (afficher) l'interface utilisateur côté serveur et ensuite envoyer les données générées au client. Cette technique est connue sous le nom de rendu côté serveur. Je discuterai des avantages et des inconvénients du rendu côté client par rapport au rendu côté serveur plus loin dans le cours.

Maintenant que nous avons une compréhension fondamentale du client et du serveur, plongeons dans les concepts impliqués dans la communication entre eux.

Communication entre le client et le serveur

Modèle de requête-réponse

Le client et le serveur fonctionnent selon un modèle de requête-réponse. Le client envoie la requête et le serveur répond avec les données.

S'il n'y a pas de requête, il n'y a pas de réponse.

Protocole HTTP

Toute la communication se fait via le protocole HTTP. C'est le protocole d'échange de données sur le World Wide Web. Le protocole HTTP est un protocole de requête-réponse qui définit la manière dont l'information est transmise sur le Web.

C'est un protocole sans état, et chaque processus sur HTTP est exécuté indépendamment et n'a aucune connaissance des processus précédents.

Pour aller plus loin sur ce sujet : https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

API REST et points de terminaison API

En parlant du contexte des applications Web modernes n-tiers, chaque client doit envoyer une requête à un point de terminaison API pour récupérer les données du backend.

Note : Si vous n'êtes pas familiarisé avec l'API REST et les points de terminaison API, nous y reviendrons plus tard.

Le code de l'application backend a une API REST implémentée. Cela sert d'interface pour les requêtes du monde extérieur. Chaque demande, que ce soit du client écrit par l'entreprise ou des développeurs tiers qui consomment nos données API, doit envoyer une requête aux points de terminaison REST pour récupérer les données.

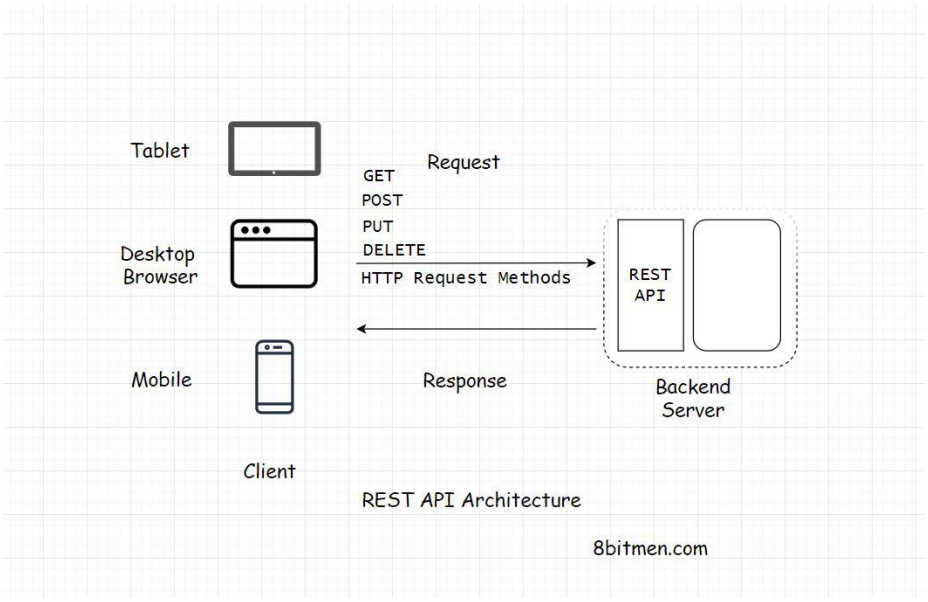

Illustration 1.11 : Architecture de l'API REST

Exemple réel d'utilisation d'une API REST

Disons que nous voulons écrire une application pour suivre les anniversaires de tous nos amis Facebook. L'application nous enverrait ensuite un rappel quelques jours avant l'anniversaire d'un certain ami.

Pour mettre en œuvre cela, la première étape consisterait à obtenir les données d'anniversaire de tous nos amis Facebook. Nous écririons un client pour frapper l'API Facebook Social Graph, qui est une API REST, pour obtenir les données, puis exécuter notre logique métier sur les données.

La mise en œuvre d'une API basée sur REST présente plusieurs avantages dont nous parlerons plus loin.