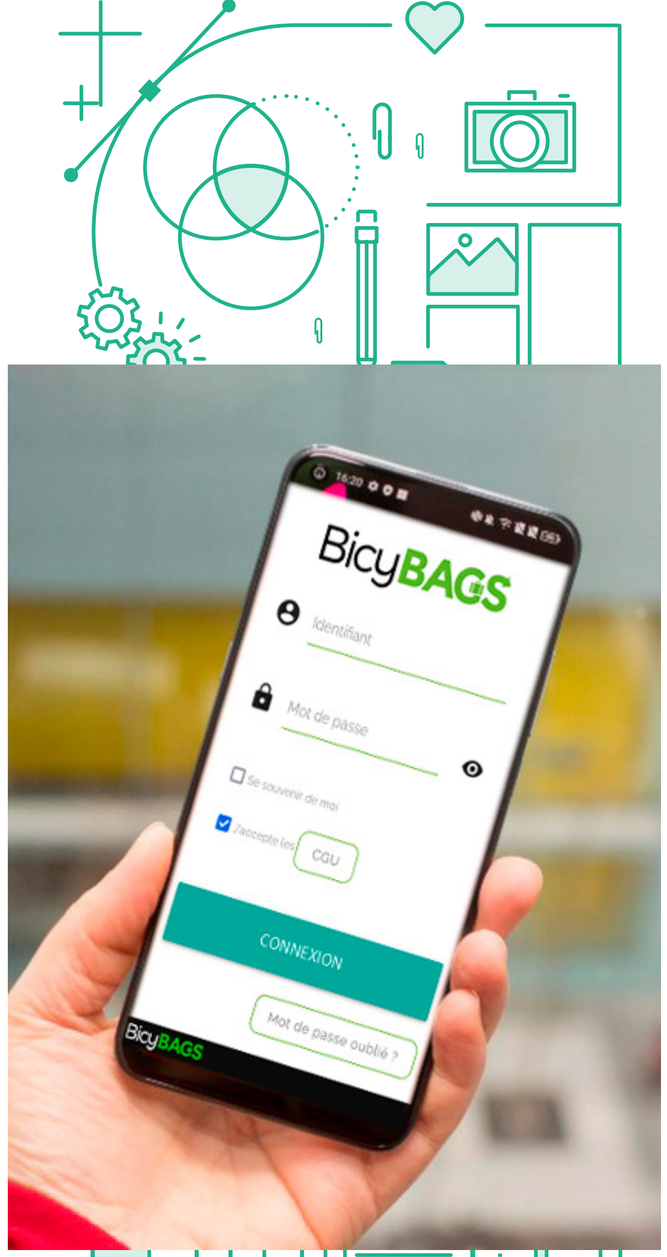
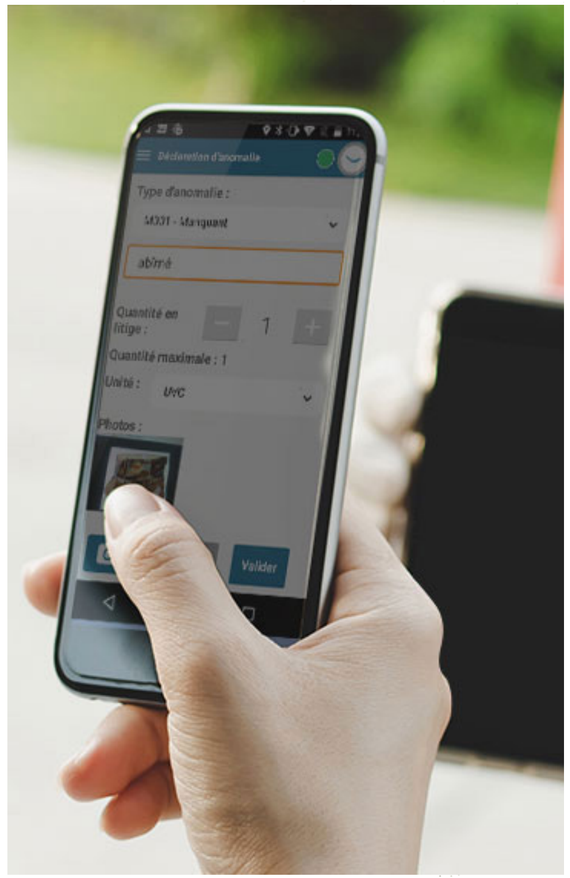
Concevoir une architecture d'application

Bourgeonnier Michaël | PaPrika Studio | 05/01/2023 | Doc v. 1.3



L’importance de l’architecture lors d’un développement logiciel

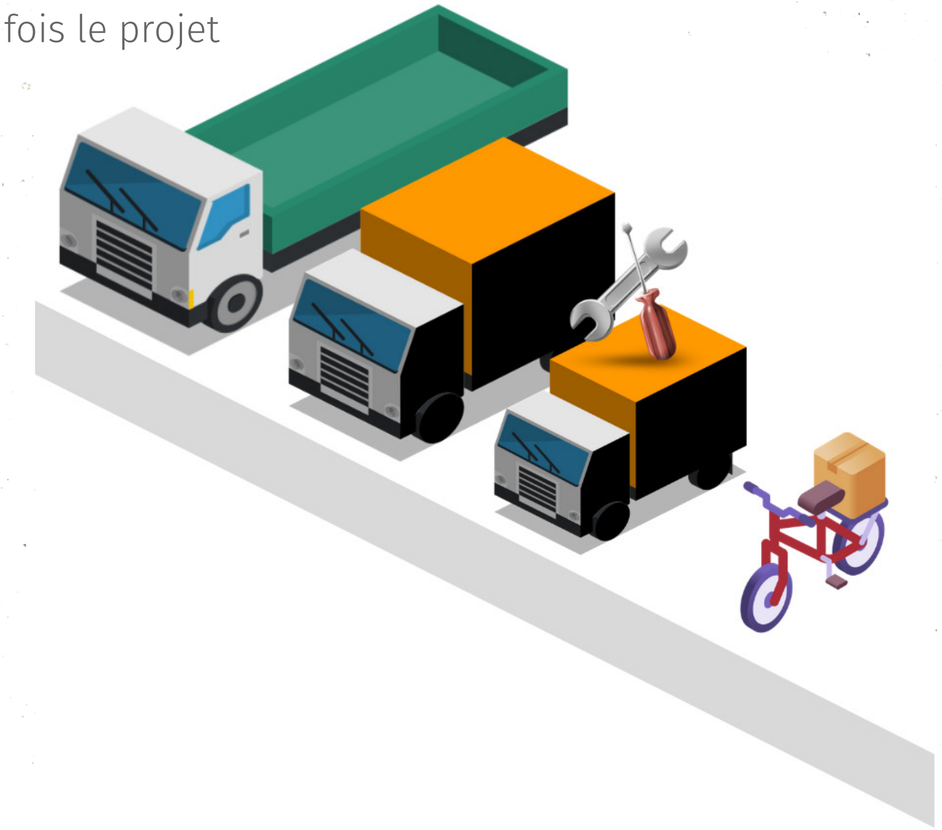


+ La conception de l’architecture est une phase particulièrement importante du développement d’un logiciel. Elle conditionne sa stabilité, son efficacité et sa pérennité.

+ Au contraire, certaines applications peuvent connaitre des faiblesses dues à une architecture mal pensée ou plus adaptée au contexte.

Si la pression du **temps de production/temps de distribution** pèse sur le développement d’un logiciel, elle pèse donc également sur la conception de son architecture.

Il est important de noter que nous l'observons en général, qu’une fois le projet commencé, s’agissant d’un élément aussi structurel,
il est dangereux voire impossible d’en changer.



Ceci étant dit, il n’est pas si fréquent de trouver
de « mauvaises » architectures dans l’absolu, mais on observe souvent des architectures qui ne sont pas parfaitement adaptées au contexte du projet de développement.

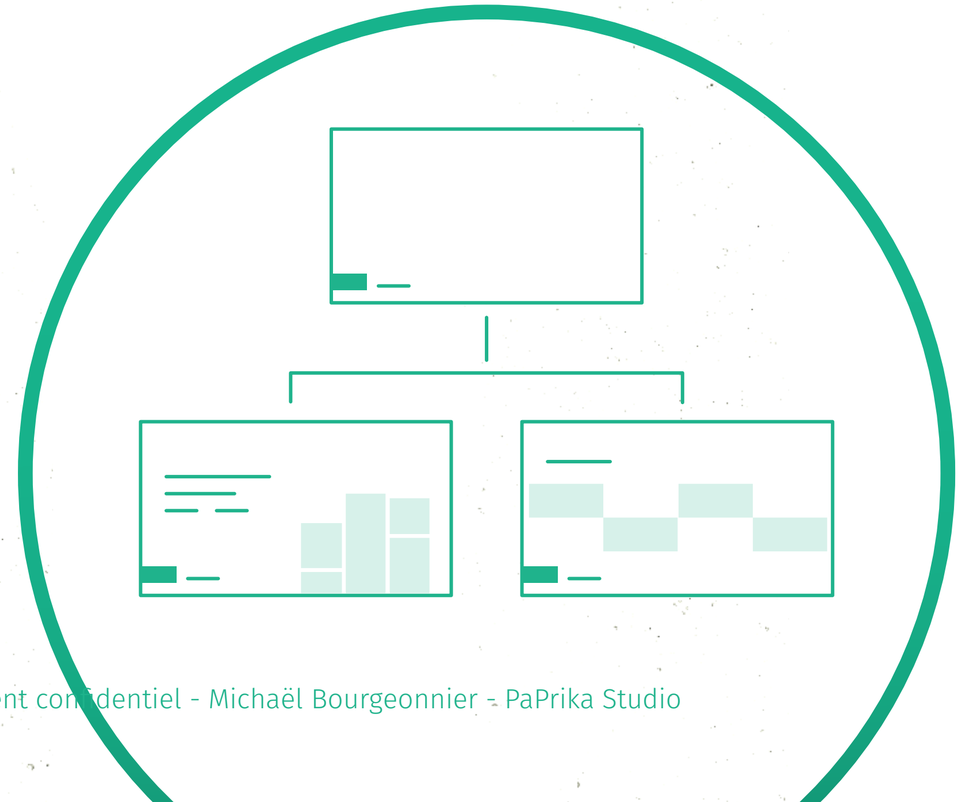
L’architecture logicielle est avant tout issue d’un compromis entre les exigences techniques, opérationnelles et fonctionnelles qui entourent l’application.

Du point de vue pratique des personnes en charge d'une application, l'architecture, c'est surtout, ce qu'il est difficile de changer.

De ce point de vue, l'architecture doit faire l'objet de décisions saines en amont d'un projet afin de minimiser les coûts d'évolution.

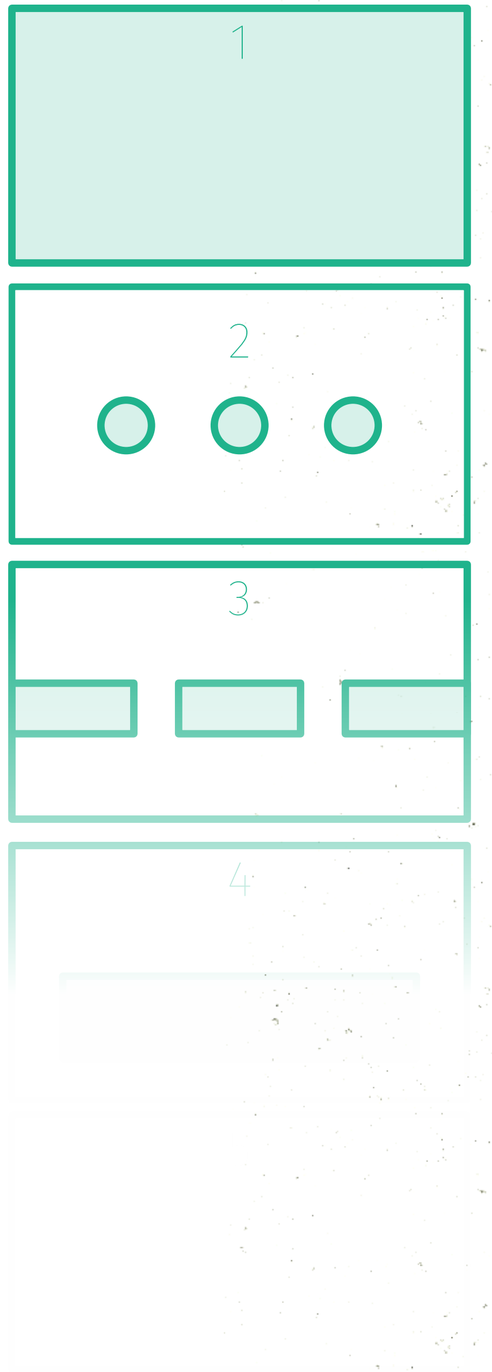
2.Plusieurs critères guident le choix de l’architecture applicative

TOUT D’ABORD, CE CHOIX FAIT SUITE À LA BONNE COMPRÉHENSION DU BESOIN MÉTIER



LA COMPRÉHENSION DU BESOIN MÉTIER DOIT PAR EXEMPLE COUVRIR LES DIFFÉRENTS TYPES D'UTILISATEURS IMPLIQUÉS, LEUR(S) DIFFÉRENT(S) MODE(S) D'ACCÈS À CE LOGICIEL.

+ Cette réflexion doit permettre de répondre entre autres à des questions essentielles en matière d'architecture logicielle :
+ Est-ce que le logiciel doit répondre rapidement ?
+ Quel volume de données doit-il traiter ?
+ Savoir si ces données doivent être centralisées ou réparties ?
+ Comment les utilisateurs doivent-il accéder à l'application du point de vue du réseau ?
+ Sur quel type de matériel ? En quelle langue ou avec quel type de clavier ?
+ ...

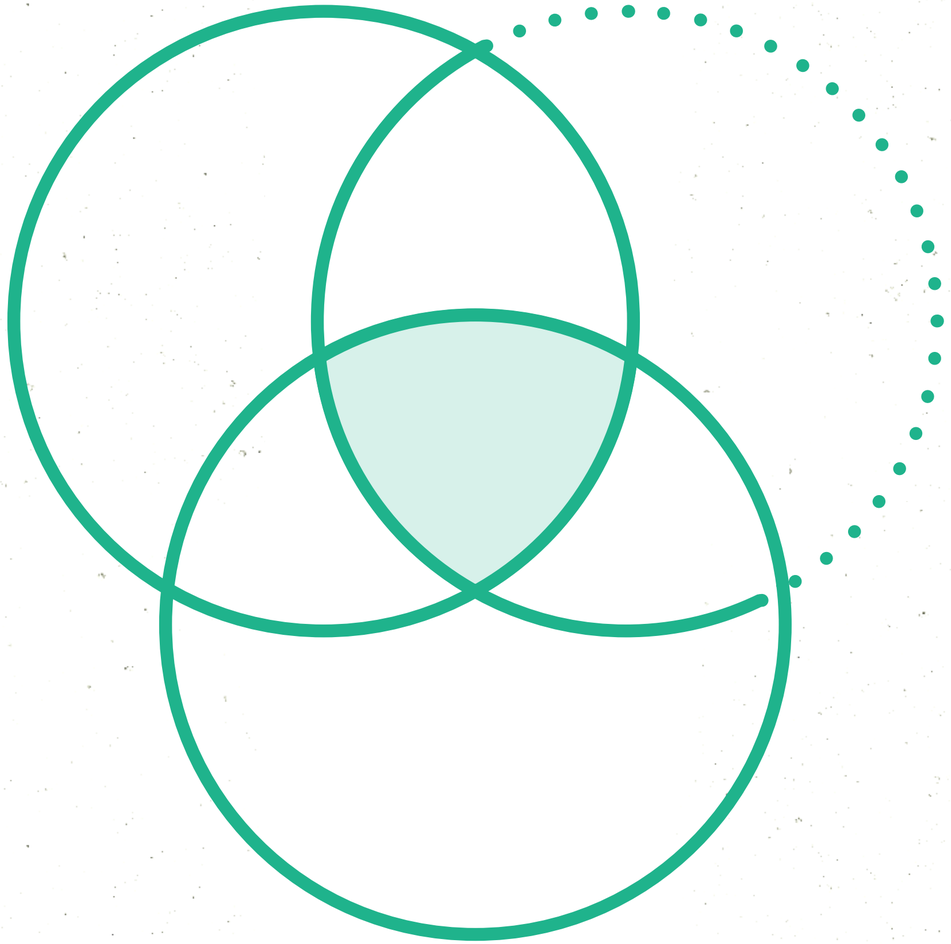


L’ARCHITECTURE DÉPENDRA AINSI DE LA TYPOLOGIE DE L’ENTREPRISE ET DE L’UTILISATION À LAQUELLE ELLE DESTINE L’APPLICATION.

Le nombre d’utilisateurs prévu de l’application rentre également en jeu notamment car cela impacte la fréquence des requêtes du logiciel.

Un logiciel sollicité chaque jour par 3 utilisateurs sera pensé différemment d’un logiciel sollicité chaque jour par 50 000 utilisateurs, même si le volume de données traitées à chaque requête est faible et similaire.

Le choix de l’architecture peut se faire également en fonction d’éléments externes au projet et notamment les compétences de l’équipe technique qui peuvent orienter vers l’une ou l’autre architecture.



Définition d'une bonne architecture

QU’EST-CE QUI FAIT UNE BONNE ARCHITECTURE D'APPLICATIONS

Son évolutivité :

L’architecture doit prendre en compte les évolutions futures du logiciel en fonction du besoin métier. Si on ne peut anticiper les évolutions elles-mêmes, elle doit dans ce cas être assez souple pour qu’elles soient possibles, sans tomber dans le piège de vouloir prévoir démesurément le futur.

Sa simplicité :

Une architecture complexe est souvent source de défaillance et peut créer de la dette technique, impacter les performances ou l’évolution d’une application. Elle est due à une mauvaise conception, une sur-ingénierie initiale ou à l’inverse un manque de conception global qui induit une complexification progressive du logiciel dans le temps.

Sa maintenabilité :

Une bonne architecture intègre aussi l'outillage nécessaire à sa maintenance. Cela permet notamment de récupérer de l'information de manière centralisée lorsqu'il y a une erreur afin de pouvoir la traiter efficacement et d'agir en conséquence.

Sa compatibilité :

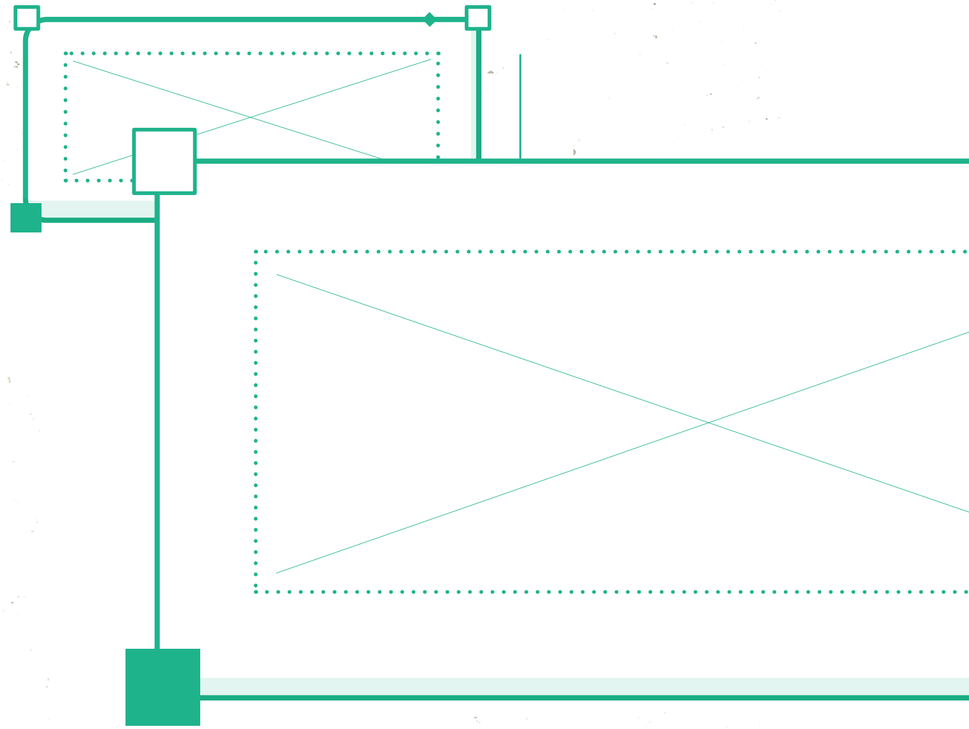
L'architecture doit définir le compatibilité du logiciel avec les différentes plates-formes matérielles, systèmes d'exploitation, navigateur ou taille d'écran qui conviennent à la cible d'utilisation

Son interconnectivité :

Puisque le logiciel évolue dans un certain environnement, son architecture doit permettre son interconnectivité avec d'autres systèmes d'information. Il est de plus en plus rare de trouver un logiciel totalement isolé des autres applications et ne nécessitant pas des interfaces de données ou a minima des exports.

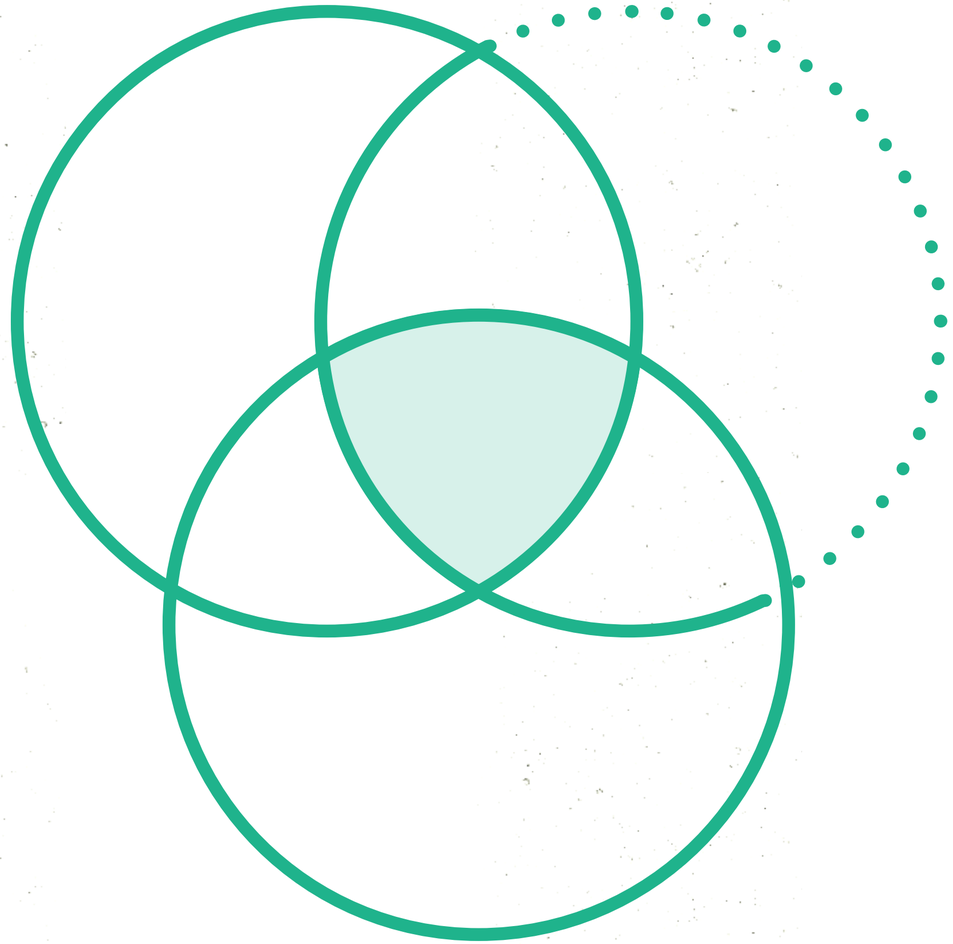
L’ARCHITECTURE D'APPLICATION N’A PAS DE DURÉE DE VIE À PROPREMENT PARLER.

Ce sont principalement les évolutions de l’application qui vont influencer son bon fonctionnement.



Les différentes architectures d'une application

Avant de décider quelle architecture d'application utiliser pour votre nouvelle application, ou avant d'évaluer votre architecture actuelle, commencez par déterminer vos objectifs stratégiques.



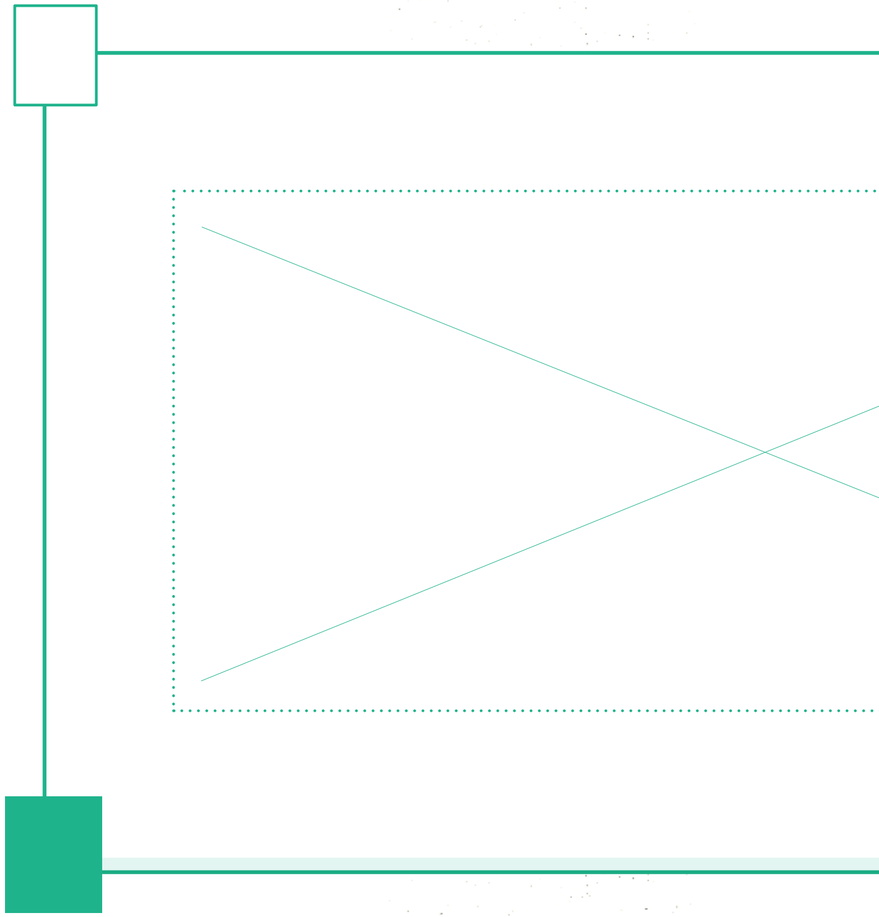
ARCHITECTURE MONOLITHIQUE

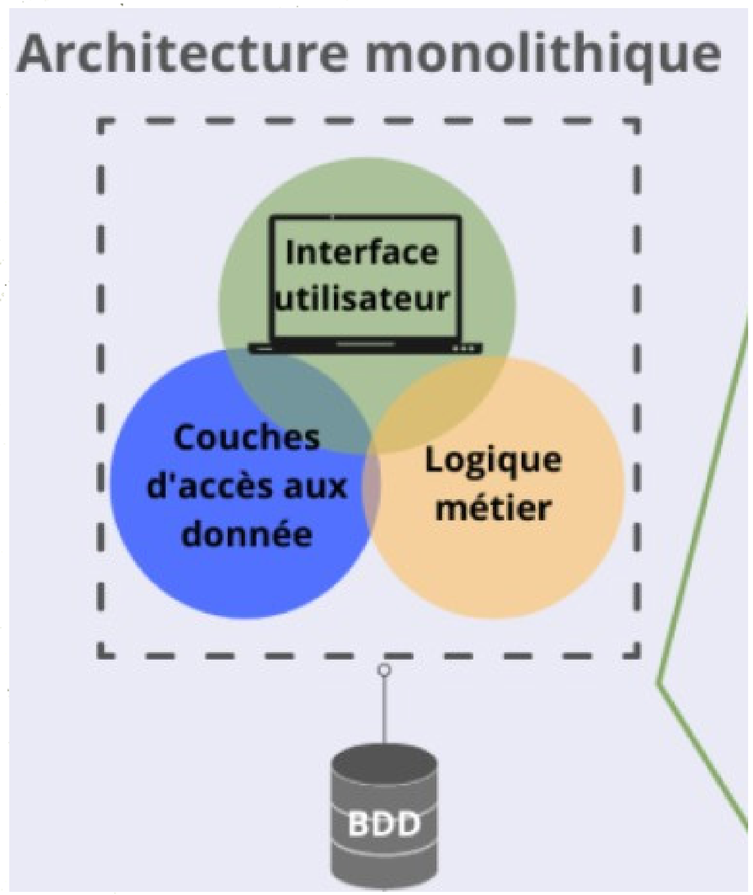
+ Il s'agit d'une unique pile d'applications qui rassemble toutes les fonctionnalités au sein d'une seule et même application.

+ Étroitement couplée, tant au niveau des interactions entre les services qu'au niveau des processus de développement et de déploiement.

+ La mise à jour ou la mise à l'échelle d'un seul composant d'une application monolithique peut avoir des effets sur toute l'application, ainsi que sur son infrastructure sous-jacente.

+ Après chaque changement apporté au code de l'application, il faut publier à nouveau l'application tout entière. De ce fait, les mises à jour et les nouvelles versions ne peuvent être publiées qu'une à deux fois par an et elles n'incluent souvent que de la maintenance générale, à défaut de nouvelles fonctions.





C’est l’approche traditionnelle et historique pour la conception des programmes informatiques.

Encore aujourd’hui, malgré une évolution des modes de développement, l’architecture monolithique reste très utilisée.

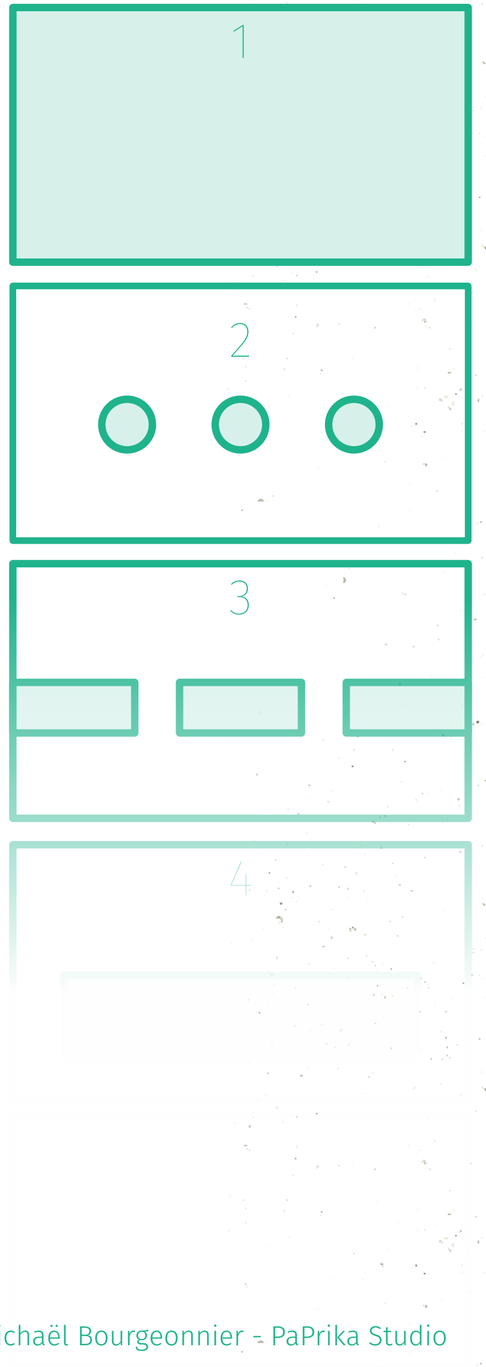
ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

+ Une architecture en couches ou multi-niveaux est une architecture classique, souvent utilisée pour créer des applications d'entreprise ou sur site. Elle est fréquemment associée aux applications d'anciennes générations.

+ Constituée souvent de trois couches, parfois plus, qui forment l'application. Chaque couche a des responsabilités distinctes.

+ Les couches permettent de gérer les dépendances et d'exécuter des fonctions logiques. Elles sont empilées, de sorte qu'elles ne peuvent faire appel qu'aux couches inférieures.

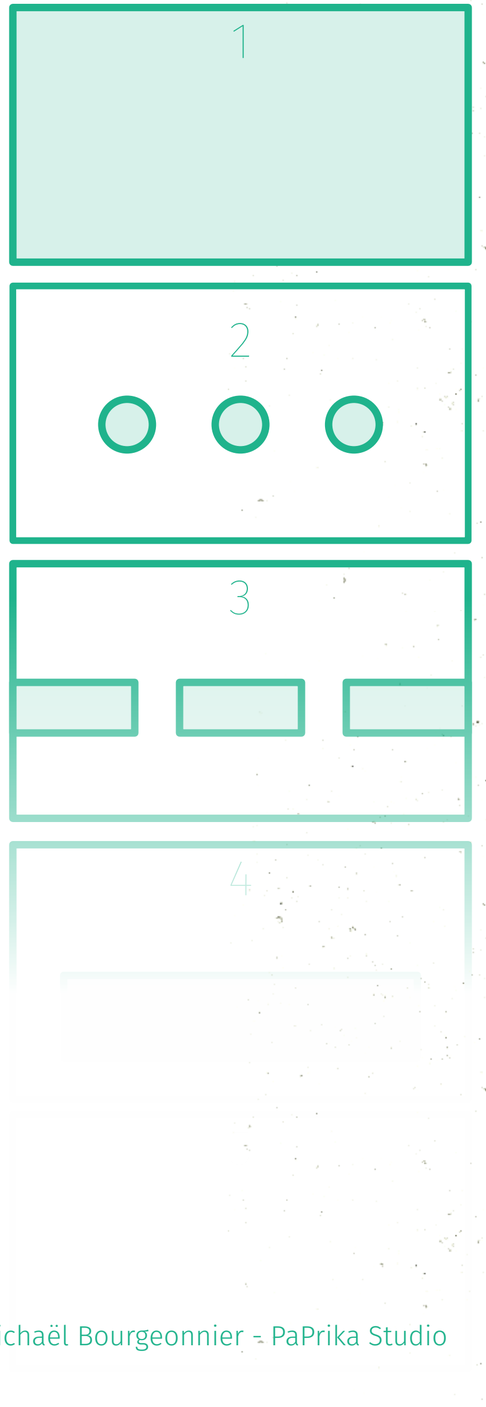
+ Ainsi, chaque couche peut faire appel soit à la couche directement au-dessous d'elle, soit à n'importe quelle autre couche inférieure



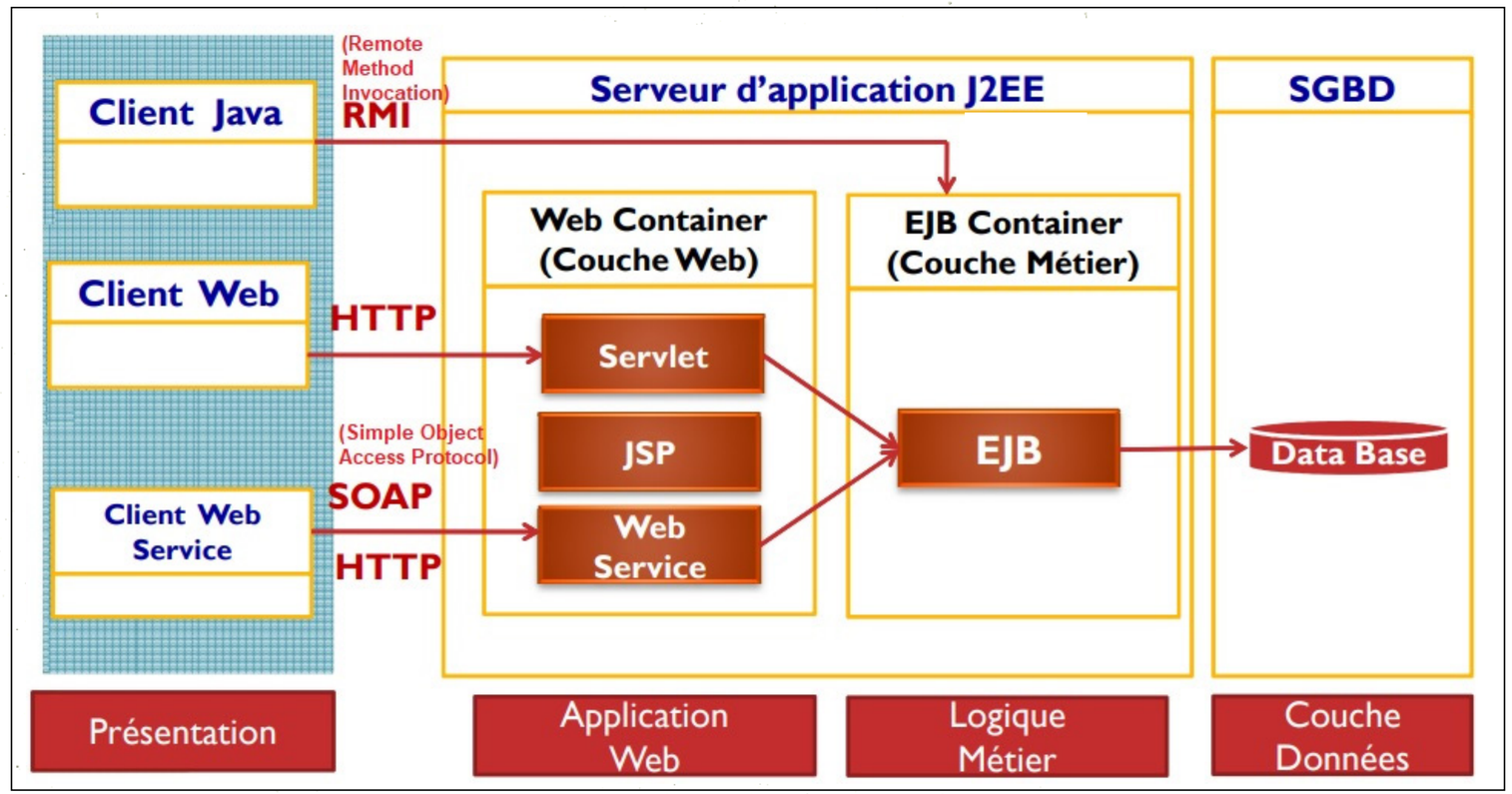
ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

DANS LE CAS D'APPLICATIONS WEB CES COUCHES SONT

+ la logique de présentation,
+ les processus de traitement
+ la gestion de la persistance des données.



ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX



Dans une telle architecture, chaque tier assure une fonction particulière :

+ Le client assure la saisie et l'affichage des données sur le serveur,
+ Un service sert de médiateur entre le client et les objets métiers,
+ Les objets métiers contiennent les traitements.
  Les EJB sont spécialement conçus pour constituer de telles entités.
+ Une base de données assure la persistance des informations.

ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

LA COUCHE PRÉSENTATION

Elle implémente la logique présentation de l'application
La couche présentation est liée au type de client utilisé :

Client Lourd java Desktop:

Interfaces graphiques java SWING, AWT, SWT.

Ce genre de client peut communiquer directement avec les composants métiers déployés dans le conteneur EJB en utilisant le middleware RMI (Remote Method Invocation)

Client Leger Web

HTML, Java Script, CSS.

Un client web communique avec les composants web Servlet déployés dans le conteneur web du serveur d'application en utilisant le protocole HTTP.

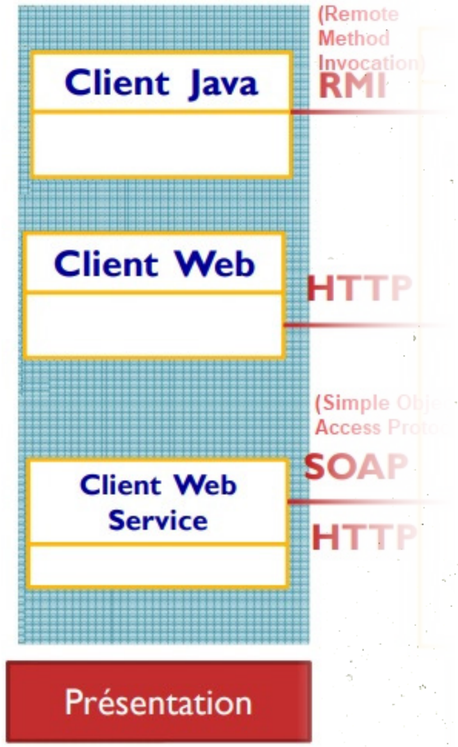
Un client .Net, PHP, C++, ...

Ce genre de clients développés avec un autre langage de programmation autre que java, communiquent généralement avec les composants Web Services déployés dans le conteneur Web du serveur d'application en utilisant le protocole SOAP (HTTP+XML)

Client Mobile

Androide, iPhone, Tablette etc..

Généralement ce genre de clients communique avec les composants Web Services en utilisant le protocole HTTP ou SOAP



ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

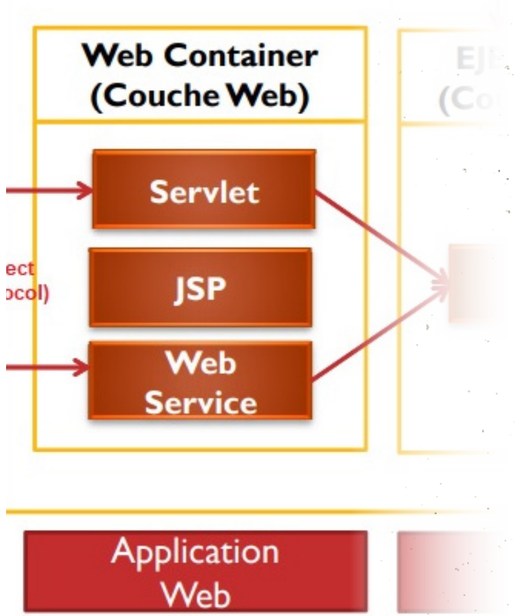
LA COUCHE APPLICATION

Appelée également couche web.

La couche application sert de médiateur entre la couche présentation et la couche métier.

+ Elle contrôle l'enchaînement des tâches offertes par l'application
+ Elle reçoit les requêtes http clientes
+ Assure le suivie des sessions
+ Vérifie les autorisations d'accès de chaque session
+ Assure la validation des données envoyées par le client
+ Fait appel au composants métier pour assurer les traitements nécessaires
+ Génère une vue qui sera envoyée à la couche présentation.
+ Elle utilise les composants web Servlet et JSP
+ Elle respecte le modèle MVC (Modèle Vue Contrôleur)

Des framework sont généralement utilisés dans cette couche.



ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

LA COUCHE MÉTIER

La couche métier est la couche principale de toute application

+ Elle implémente la logique métier d'une entreprise

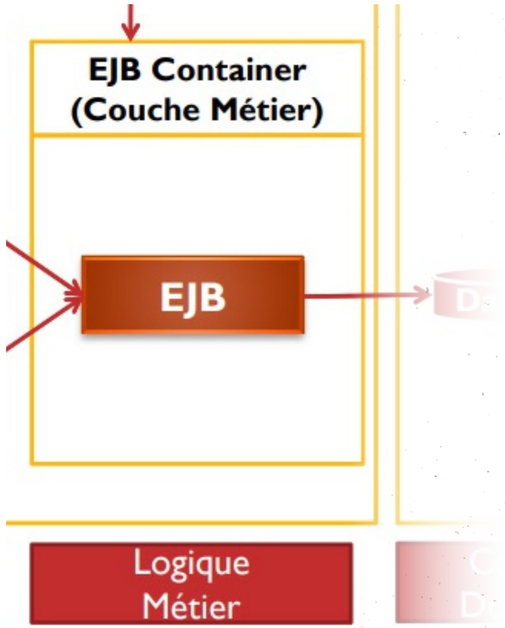
+ Elle se charge de récupérer, à partir des différences sources de données, les données nécessaires pour assure les traitement métiers déclenchés par la couche application.

+ Elle assure la gestion du WorkFlow (Processus de traitement métier en plusieurs étapes)

+ Il est cependant important de séparer la partie accès aux données (Couche DAO) de la partie traitement de la logique métier (Couche Métier) pour les raisons suivantes :

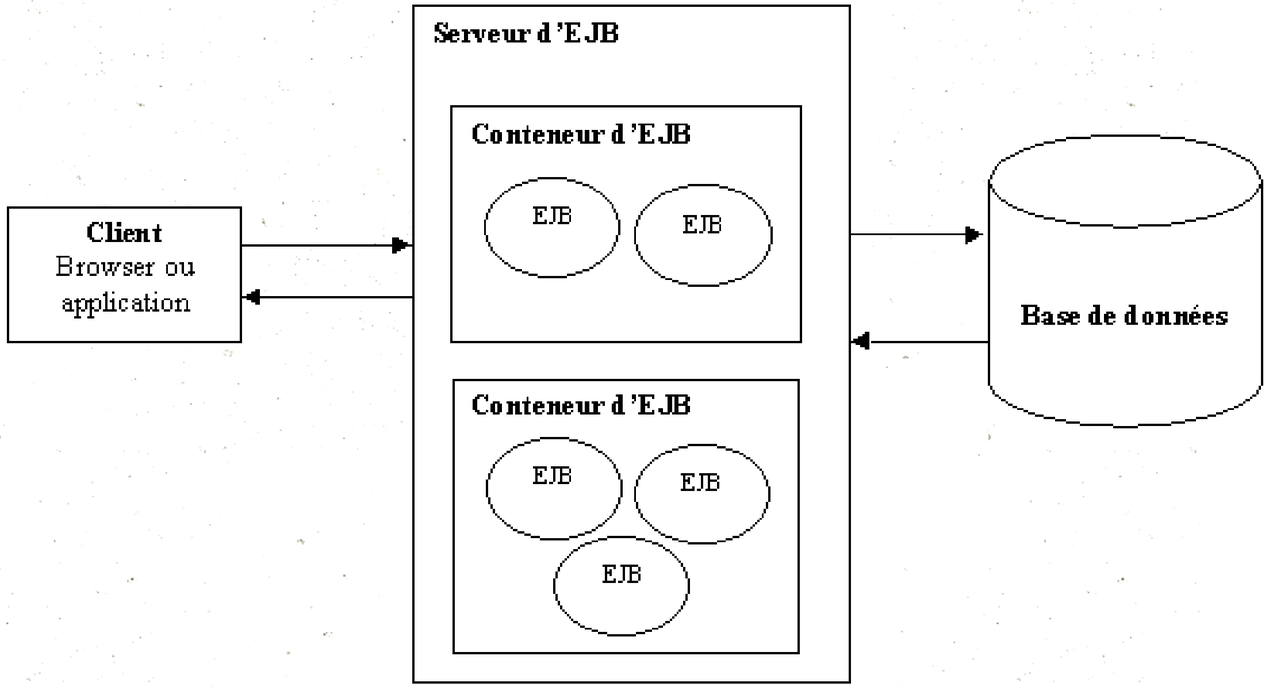
+ Ne pas se perdre entre le code métier, qui est parfois complexe, et le code d'accès aux données qui est élémentaire mais conséquent.

+ Ajouter un niveau d'abstraction sur l'accès aux données pour être plus modulable et par conséquent indépendant de la nature des unités de stockage de données.



ARCHITECTURES EN COUCHES OU MULTI-NIVEAUX

Les EJB sont parfaitement adaptés pour être intégrés dans une architecture trois tiers ou plus.



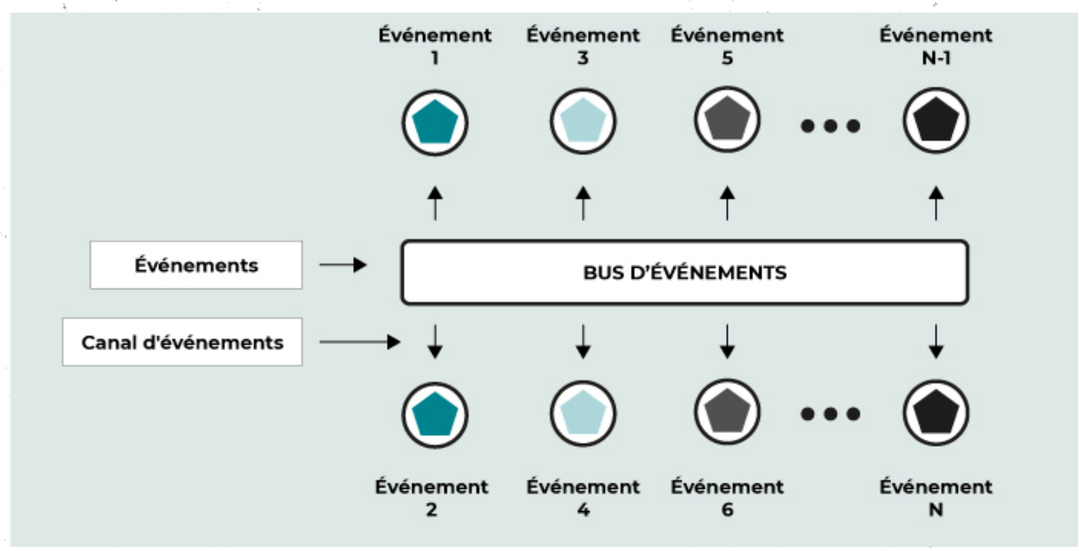
ARCHITECTURE ORIENTÉE ÉVÉNEMENTS (EVENT-DRIVEN ARCHITECTURE)

+ Dans un système orienté événements, la structure centrale de la solution repose sur la capture, la communication, le traitement et la persistance des événements.

+ Un événement désigne tout phénomène ou changement d'état significatif au niveau du matériel ou d'un logiciel système. Les événements peuvent être causés par des actions internes ou externes.

+ Ce type d'architecture implique des producteurs et des consommateurs d'événements. Un producteur d'événements détecte ou reconnaît un événement et le représente sous forme de message. Il ignore quels seront les consommateurs et les conséquences de chaque événement.

+ Lorsqu'un événement a été détecté, il est transmis du producteur d'événements au consommateur via des canaux d'événement, où une plateforme de traitement les prend en charge de façon asynchrone.



L'utilisation d'une architecture pilotée par les événements présente plusieurs avantages :

+ Si vous avez des utilisateurs qui ont besoin d'écouter (de s'abonner à) différents messages sur un sujet spécifique, l'architecture pilotée par les événements est la bonne approche. Cela se produit lorsque le système doit réagir à des événements sans comportement prédictif : par exemple, si je m'abonne à la chaîne « Informations africaines », je n'ai aucun contrôle sur les actualités que je recevrai ni sur leur quantité. Le système n'est pas prédictif ; il réagit aux événements générés par d'autres utilisateurs.

+ Si ces utilisateurs sont en dehors de l'organisation, les événements peuvent être transportés dans un bus d'événements publics et atteindre tous les utilisateurs immédiatement.

+ De nombreux événements peuvent être traités en même temps, voire des millions.

Il existe également quelques inconvénients majeurs :

+ Le bus d'événements peut être surchargé.

+ Si le bus d'événements tombe en panne, tout le système tombe en panne.

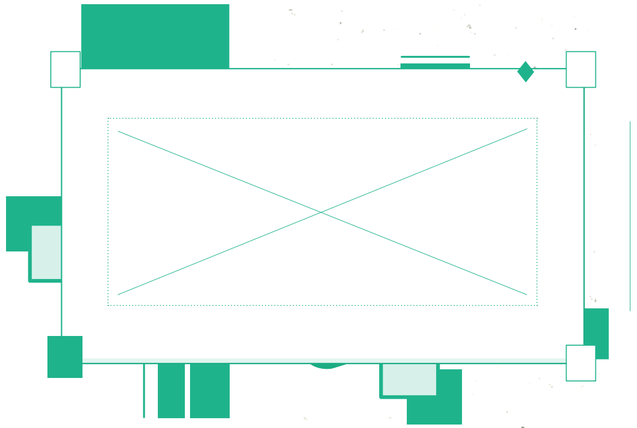
+ Il n'y a pas de contrôle du flux d'événements : de nombreux événements peuvent se produire en même temps, créant un véritable chaos pour les utilisateurs.

ARCHITECTURE ORIENTÉE SERVICES (SOA)

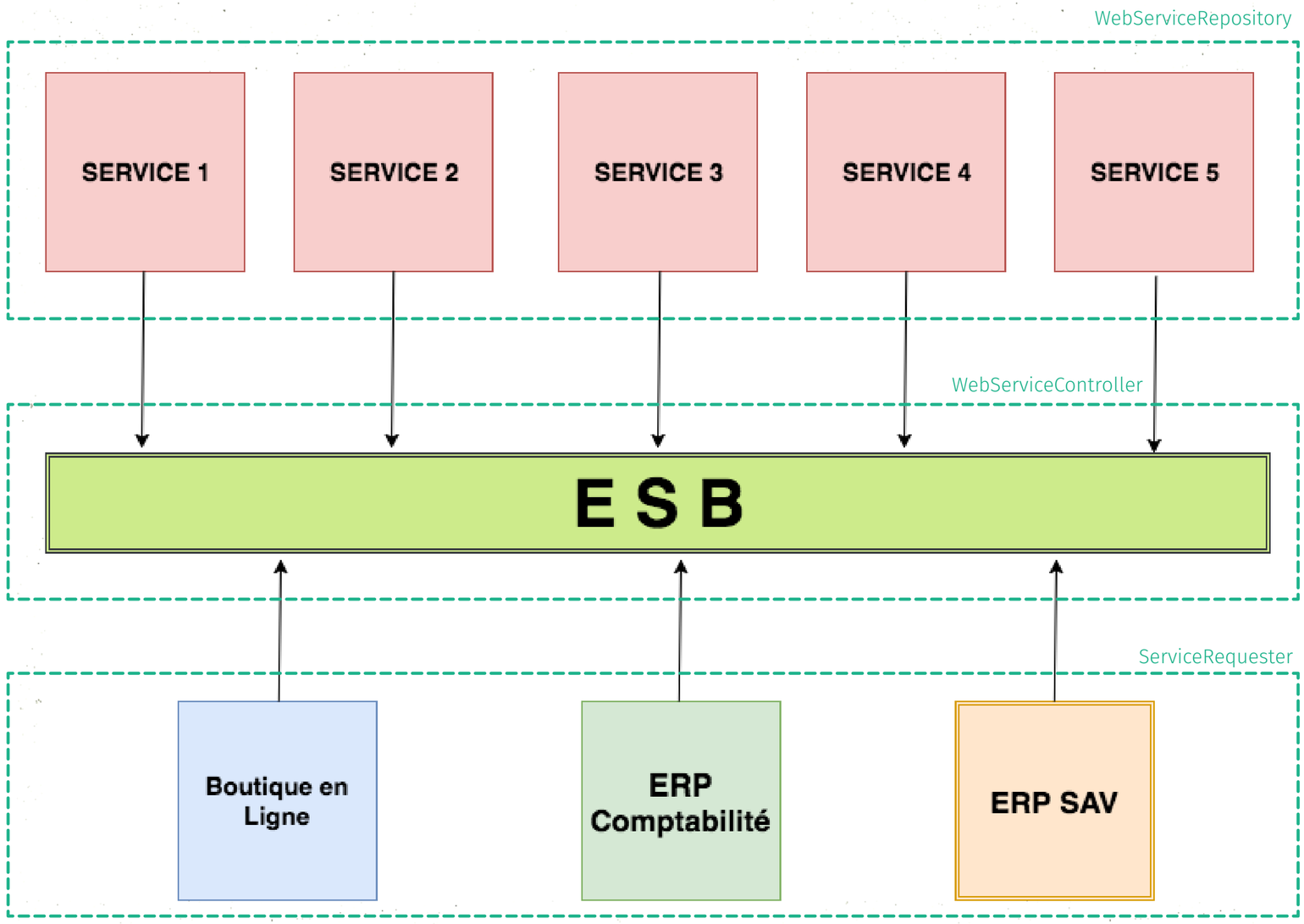
+ L'architecture orientée services (SOA) est un modèle de conception largement utilisé, similaire au modèle d'architecture de micro-services.

+ L'architecture orientée services structure les applications en services individuels et réutilisables qui communiquent via un ESB. (L'enterprise service bus. Son but est avant tout de permettre la communication des applications qui n'ont pas été conçues pour fonctionner ensemble (par exemple deux progiciels de gestion intégrés provenant d'éditeurs différents)

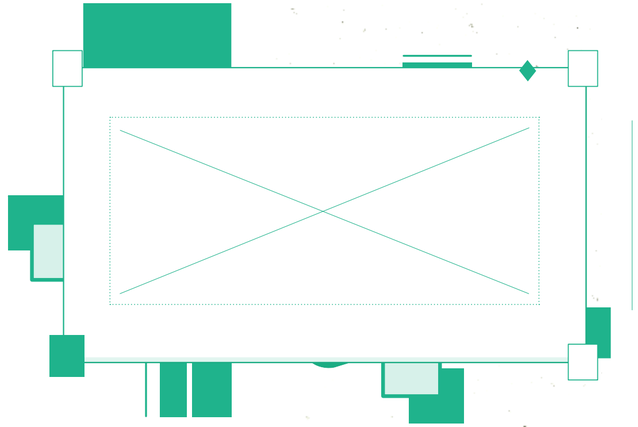
+ Au sein de cette architecture, chacun des services individuels (organisés autour d'un processus métier spécifique) suit un protocole de communication pour « s'exposer » via la plateforme d'un ESB. L'entreprise ou le client se sert d'une application front-end pour utiliser cette suite de services.



ARCHITECTURE ORIENTÉE SERVICES (SOA)



exemple d'une SOA



ARCHITECTURE DE MICRO-SERVICES (MSA)

Cette architecture est une extension du concept de SOA

Certaines applications d'entreprise s'avèrent laborieuses et trop coûteuses à faire évoluer.

Ce type de dette technologique est un constat, une difficulté majeure, que rencontrent à terme de nombreuses entreprises.

L'essor des architectures MICRO-SERVICES (micro-services architectures - MSA) répond à cette problématique, et propose des moyens de la résoudre.

ARCHITECTURE DE MICRO-SERVICES (MSA)

Introduction :

De nombreux micro-services
utilisent des API pour communiquer entre eux.

Tout d'abord, il faut insister sur le fait que les micro-services et les APIs sont deux choses différentes.

Un micro-service est une architecture qui sépare une application en plusieurs petits services web autonomes.

L'API (Application programming Interface) est, comme son nom l'indique une interface de programmation. Elle constitue le cadre à travers lequel un développeur peut interagir avec une application. Les APIs sont un ensemble de classes, méthodes, fonctions et constantes qui sert d'interface par laquelle un logiciel peut offrir ses services à d'autres logiciels. Elles servent concrètement à accéder aux données d'une application et à utiliser ses fonctionnalités.

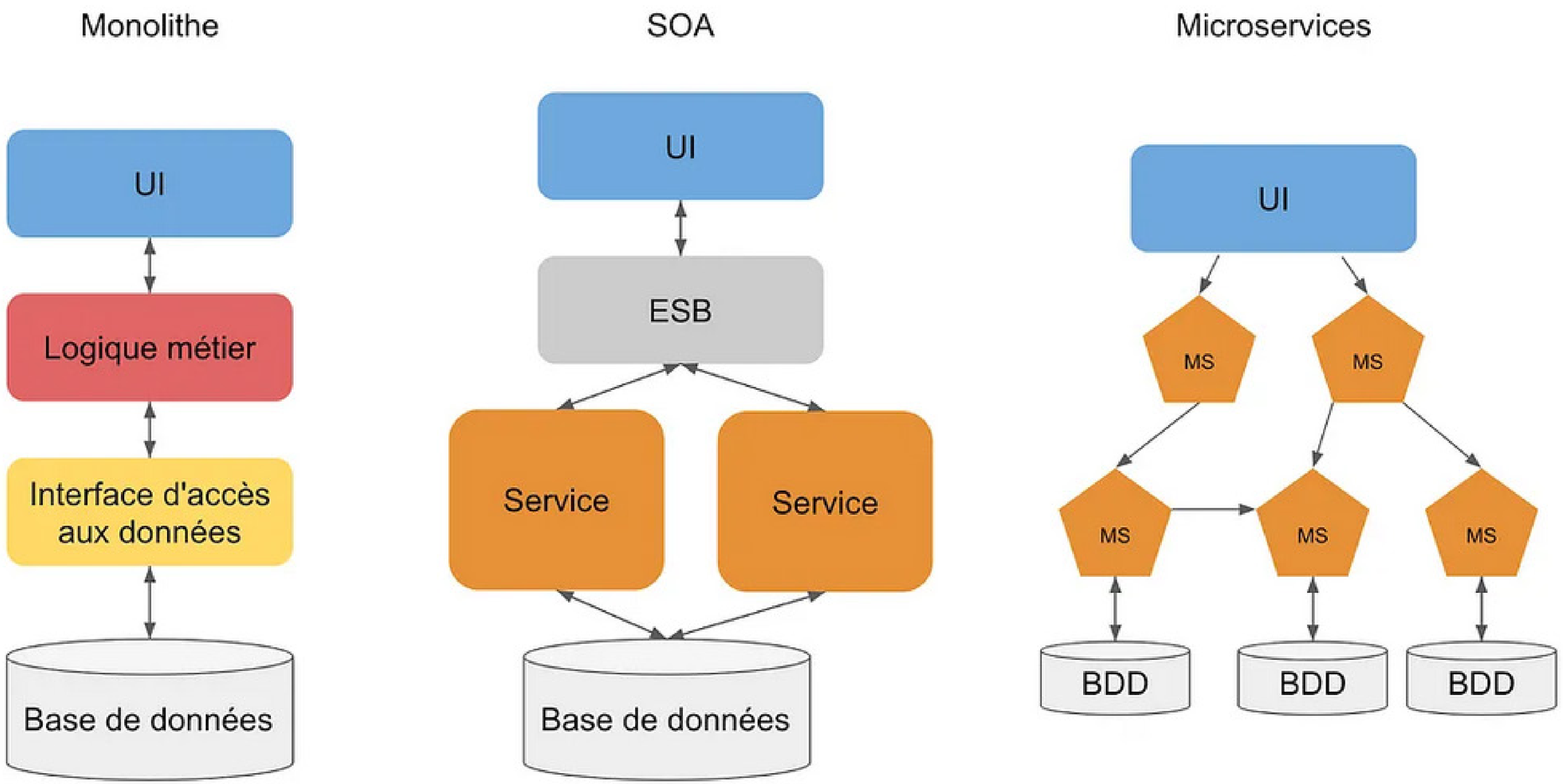
La confusion se produit car il existe un chevauchement entre ces deux concepts informatiques. De nombreux micro-services utilisent en effet des API pour communiquer entre eux.

+ Les **micro-services** désignent à la fois une architecture et une approche de développement logiciel, qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Chacun de ces composants ou processus est un **micro-service**.

+ Les **micro-services** sont distribués et faiblement couplés, ils n'ont donc pas d'incidence les uns sur les autres. Ces caractéristiques offrent des avantages au niveau de l'évolutivité dynamique et de la tolérance aux pannes : il est possible de mettre à l'échelle des services individuels sans nécessiter une infrastructure lourde, et d'effectuer leur basculement sans affecter les autres services.

+ Avec une architecture de **micro-services**, l'objectif est de fournir rapidement des logiciels de qualité. Il est possible de développer plusieurs **micro-services** simultanément. De plus, puisque les services sont déployés indépendamment, vous n'avez pas à recréer ou redéployer toute l'application après chaque modification.

+ Plusieurs développeurs peuvent ainsi travailler sur leurs services individuels en même temps, sans qu'il y ait besoin de mettre à jour toute l'application.



ARCHITECTURE ORIENTÉE SERVICES

Modèle d'architecture	Description	Avantages	Inconvénients	Quand l'utiliser
Orienté services	Un modèle d'architecture basé sur les services qui permet à un système externe d'utiliser une bibliothèque de fonctionnalités sans accéder aux systèmes internes.	Communication simple : fonctionne à 100 % sur Internet. Normes de sécurité strictes : aucun client ne peut accéder aux systèmes internes.	Les clients ont besoin d'Internet pour utiliser la bibliothèque. Le contrôleur de services web peut être surchargé et subir des problèmes de performance. Pensez au nombre de demandes que le service web de suivi de FedEx reçoit par seconde.	Lorsque vous avez de nombreux clients pour un service web. Lorsque vous devez communiquer à distance avec ces clients.

EN RÉSUMÉ

ARCHITECTURE ORIENTÉE ÉVÉNEMENTS (EVENT-DRIVEN ARCHITECTURE)

Modèle d'architecture	Description	Avantages	Inconvénients	Quand l'utiliser
Pilotage par les événements	Les événements sont produits, transportés et interprétés dans un bus d'événements. Les clients s'abonnent à un groupe d'événements (appelé *canal*) et agissent à la réception des messages.	Capacité à gérer des millions d'événements en même temps. Capacité à faire communiquer différentes technologies et plateformes avec le même bus d'événements.	Si le bus d'événements tombe en panne, le système ne fonctionne plus du tout. Le bus d'événements peut être surchargé et subir des problèmes de performance.	Lorsque vous avez plusieurs événements en même temps. Lorsque vous devez agir sur un événement en temps réel (exécution synchronisée). Lorsque vous avez des plateformes différentes.

EN RÉSUMÉ

ARCHITECTURE EN COUCHE

Modèle d'architecture	Description	Avantages	Inconvénients	Quand l'utiliser
En couches	Les logiciels fonctionnent en couches qui permettent à tous les composants d'être indépendants les uns des autres.	Encapsulation du matériel, des logiciels et des fonctionnalités. Si une couche est modifiée, les autres couches restent les mêmes.	Pour les petites applications, de nombreuses couches créent un problème de performance et sont très difficiles à maintenir.	Uniquement pour les grandes applications.

ARCHITECTURE MONOLITHIQUE

Elle comporte en effet plusieurs avantages dont :

+ une grande simplicité de mise en œuvre. Basiquement si aucune architecture n'est appliquée, le résultat sera probablement proche du monolithique ;
+ des tests rapides : il est facile de déployer ce type d'application en local par exemple pour la débugger ;
+ une simplicité de déploiement : il suffit de copier-coller l'application packagée sur un serveur ;
+ moins de latence : dans la structure monolithique, tous les appels sont locaux, les temps de traitement et de réponse sont ainsi réduits ;
+ des intégrations simplifiées : des éléments comme les frameworks, les bibliothèques ou les scripts peuvent être ajoutés rapidement.

Cependant, l'approche monolithique comporte aussi plusieurs inconvénients :

+ une maintenance compliquée : les applications deviennent souvent volumineuses et complexes. Bien comprendre et tester l'ensemble des dépendances avant d'apporter des modifications est donc indispensable ;
+ des lenteurs de chargement : toujours à cause de la taille du code source, l'environnement de développement peut être surchargé et les applications longues à lancer ;
+ un potentiel manque de fiabilité : un bug au niveau de n'importe quel composant peut interrompre tout le processus et donc le fonctionnement de l'application ;
+ des redéploiements intégraux : à chaque fois qu'un correctif ou une évolution est apporté, il faut redéployer l'ensemble du programme ;
+ des difficultés pour travailler simultanément : à cause des forts liens et dépendances entre les différents composants, les développeurs peuvent créer de nombreux conflits lorsqu'ils modifient en même temps le code source ;

EN RÉSUMÉ

RESSOURCES

https://www.free-work.com/fr/tech-it/blog/actualites-informatiques/architecture-monolithique-une-approche-a-abandonner

https://sebastien-bouttier.medium.com/architecture-microservices-de-a-%C3%A0-z-5626809cd4f

https://apcpedagogie.com/les-couches-dune-application/

https://blog.nicolashachet.com/developpement-php/larchitecture-rest-expliquee-en-5-regles/

https://openclassrooms.com/fr/courses/7210131

https://learn.microsoft.com/fr-fr/azure/architecture/guide/architecture-styles/n-tier

https://www.oracle.com/fr/cloud/definition-web-service/

https://www.sqli.com/fr-fr/insights-news/blog/architectures-micro-services-objectifs-benefices-et-defis-partie-1

https://www.talend.com/fr/resources/guide-microservices/

L'architecture REST

REST (representational state transfer) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web.

"wikipedia"

Un **Web Service** est une interface permettant la communication et l'échange des données avec d'autres applications web (environnements distribués), même si ces dernières sont construites dans des langages de programmation différents. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, de manière synchrone ou asynchrone.

REST est un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière.

L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web.

APPEL DE PROCÉDURE

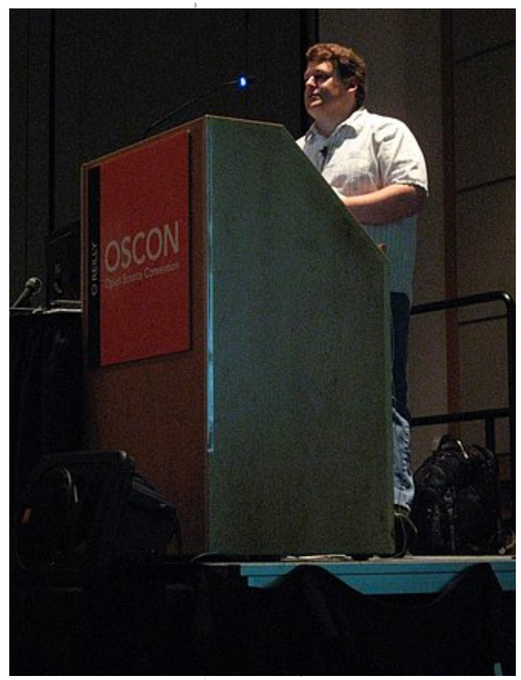
Avec Rest, nous sommes dans un monde **distribués**.

L'objectif de Rest c'est d'offrir un moyen pour manipuler et effectuer des actions sur ces différentes ressources aux travers du protocole HTTP.

L'architecture Rest s'articule autour de quatres principes clés

+ Une ressource distribuée sur un serveur distant
+ Un identifiant de la ressource
+ Des « verbes » HTTP permettant d'agir sur la ressource
+ Une représentation de la ressource

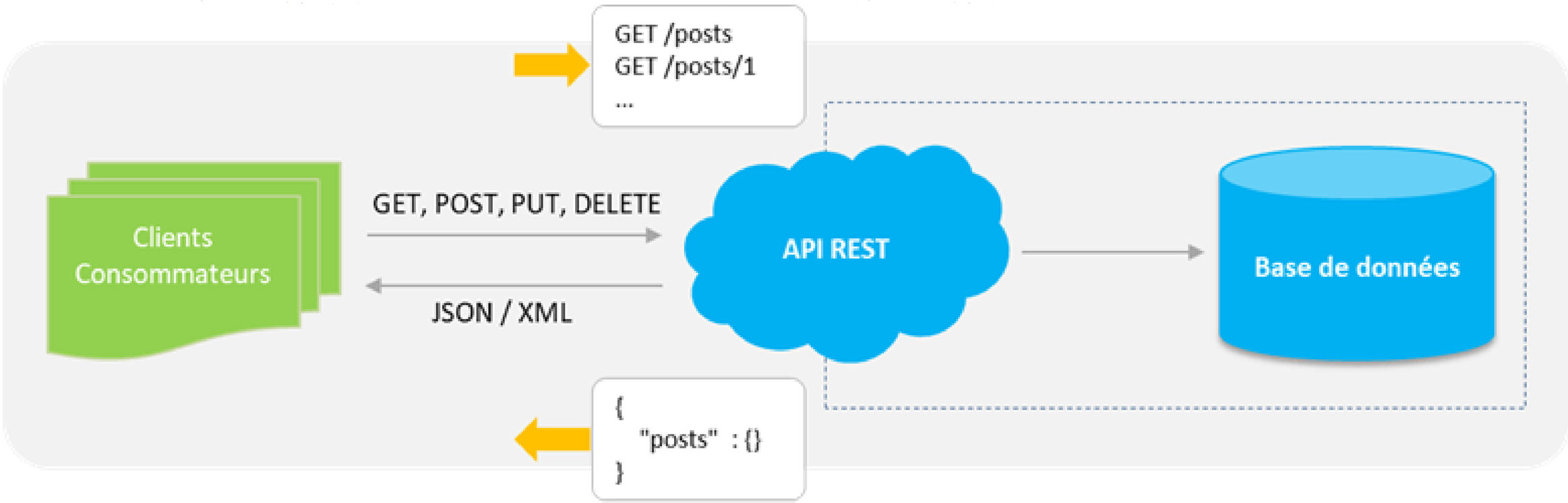
Roy Fielding a défini REST en 2000 dans sa thèse de doctorat Architectural Styles and the Design of Network-based Software Architectures à l'université de Californie à Irvine. Il y a développé le style d'architecture REST en parallèle du protocole HTTP 1.1 de 1996 à 1999, basé sur le modèle existant de HTTP 1.0 de 1996.



APPEL DE PROCÉDURE

Une "Action" Rest s'effectue en plusieurs étapes.

+ Tout d'abord le client veut accéder à une ressource à travers les différents verbes HTTP.
+ Il envoie donc une requête HTTP au serveur en précisant une URL particulière qui correspond à la ressource.
+ Le serveur traite la requête et génère une représentation de la ressource qu'il va renvoyer au client accompagné d'un code HTTP.



API REST

Une API REST est une application qui expose des ressources via les URL avec lesquelles il est possible d'interagir.

+ **GET, POST, PUT et DELETE** permettront de récupérer, créer, modifier ou supprimer une ressource.

Ainsi, par exemple, les actions suivantes vers une API REST d'une application de bibliothèque en ligne permettront d'opérer des actions sur les enregistrements en base de données.

GET /books -> Récupère tous les livres

POST /books -> Crée un livre

GET /books/1402 -> Récupère le livre identifié par 1402

PUT /books/1402 -> Modifie le livre 1402

DELETE /books/1402 -> Supprime le livre 1402

LE WORLD WIDE WEB Présentation

+ Le **World Wide Web**, littéralement la « toile (d'araignée) mondiale », communément appelé le Web,

+ **Le web** parfois **la Toile** ou le **WWW**, est un système hypertexte public fonctionnant sur Internet qui permet de consulter, avec un navigateur, des pages accessibles sur des sites. L'image de la toile d'araignée vient des hyperliens qui lient les pages web entre elles.

+ Le **Web** n'est qu'une des applications d'Internet. D'autres applications sont par exemple le courrier électronique, la messagerie instantanée etc. Le Web a été inventé plusieurs années après Internet.