# API

Une API (Application Programming Interface) est une interface pour que les applications s’interagissent entre eux-mêmes.

# API REST

REST (Representational State Transfer) est une API qui imite la façon dont le web marche dans les échanges entre un client et un serveur. Une API REST est :

* Orienté client-serveur
* Sans état (le serveur n’a aucune idée de l’état du client entre deux requêtes. Du point de vue du serveur, chaque requête est une entité distincte des autres)
* Cacheable (un client doit être capable de garder en mémoire des informations sans avoir constamment besoin de demander tout au serveur).
* Avec une interface uniforme
* Avec un système de couche
* Un code à la demande (optionnel).

# Service

Un service consiste en une fonction ou fonctionnalité bien définie. C'est aussi un composant autonome qui ne dépend d’aucun contexte ou service externe.

Un service est une entité de traitement qui respecte les caractéristiques suivantes :

* Large granularité : les opérations proposées par un service encapsulent plusieurs fonctions et opèrent sur un périmètre de données large.
* Interface : un service peut implémenter plusieurs interfaces, et aussi plusieurs services peuvent implémenter une interface commune.
* Localisable : avant d’appeler un service, il faudra le trouver.
* Instance unique : un service a une instance unique. Il correspond au design pattern Singleton.
* Couplage faible : les services sont connectés aux clients et aux autres services via des standards. Ces standards assurent le découplage, c'est-à-dire la réduction des dépendances. Ces standards sont des documents XML comme dans les web services.
* Synchrone ou asynchrone : attente de réponse après l’invocation d'un service ou non.

# Service web

### Définition

Un service Web est un service basé sur le modèle SOA accessible au moyen d'Internet, qui utilise un système de messagerie standard XML, et n'est lié à aucun système d'exploitation ou langage de programmation. Le service Web est un moyen rapide de distribution de l'information entre clients, fournisseurs, partenaires commerciaux et leurs différentes plates-formes.

### Les principaux avantages d'un service Web :

* Son interface décrite d'une manière interprétable par les machines, qui permet aux applications clientes d'accéder aux services de manière automatique ;
* Son utilisation de langages et protocoles indépendants des plates-formes d'implantation, qui renforcent l'interopérabilité entre services ;
* Son utilisation des normes actuelles du Web, qui permettent la réalisation des interactions faiblement couplées et favorisent aussi l'interopérabilité.

### L'intérêt d'un Service Web

Les services Web fournissent un lien entre applications. Ainsi, des applications utilisant des technologies différentes peuvent envoyer et recevoir des données au travers de protocoles compréhensibles par tout le monde.

La technologie des services Web repose essentiellement sur une représentation standard des données (interfaces, messageries) au moyen du langage XML. Cette technologie est devenue la base de l'informatique distribuée sur Internet et offre beaucoup d'opportunités au développeur Web. :)

### Les caractéristiques d'un service Web

Un service Web possède les caractéristiques suivantes :

* Il est accessible via le réseau ;
* Il dispose d'une interface publique (ensemble d'opérations) décrite en XML ;
* Ses descriptions (fonctionnalités, comment l'invoquer et où le trouver ?) sont stockées dans un annuaire ;
* Il communique en utilisant des messages XML, ces messages sont transportés par des protocoles Internet (généralement HTTP, mais rien n'empêche d'utiliser d'autres protocoles de transfert tels : SMTP, FTP, BEEP... ) ;
* L’intégration d'application en implémentant des services Web produit des systèmes faiblement couplés, le demandeur du service ne connaît pas forcément le fournisseur.

Ce dernier peut disparaître sans perturber l'application cliente qui trouvera un autre fournisseur en cherchant dans l'annuaire.

### Architecture d'un service Web

Les services Web reprennent la plupart des idées et des principes du Web (HTTP, XML), et les appliquent à des interactions entre machines. Comme pour le World Wide Web, les services Web communiquent via un ensemble de technologies fondamentales qui partagent une architecture commune. Ils ont été conçus pour être réalisés sur de nombreux systèmes développés et déployés de façon indépendante. Les technologies utilisées par les services Web sont HTTP, WSDL, REST, XML-RPC, SOAP et UDDI.

Le service Web utilise WSDL (un métalangage XML) comme langage de description, un annuaire UDDI pour en permettre la localisation et un protocole de transport comme HTTP dans l'architecture REST ou SOAP pour l'architecture SOA.

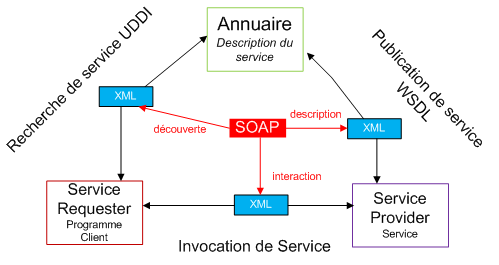
WSDL (Web Services Description Language) est un langage de description standard. C'est l'interface présentée aux utilisateurs. Il indique comment utiliser le service Web et comment interagir avec lui. WSDL est basé sur XML et permet de décrire de façon précise les détails concernant le service Web tels que les protocoles, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.

UDDI (Universal Description, Discovery and Integration) est un annuaire de services. Il fournit l'infrastructure de base pour la publication et la découverte des services Web. UDDI permet aux fournisseurs de présenter leurs services Web aux clients.

REST (Representational State Transfer) est une architecture de services Web. REST est une manière de construire une application pour les systèmes distribués comme le World Wide Web.

### Fonctionnement des services Web

Le fonctionnement des services Web s'articule autour de trois acteurs principaux illustrés par le schéma suivant :



**Service provider service**

Le fournisseur de service met en application le service Web et le rend disponible sur Internet.

**Service requester programme client**

C'est n'importe quel consommateur du service Web. Le demandeur utilise un service Web existant en ouvrant une connexion réseau et en envoyant une demande en XML (REST, XML-RPC, SOAP).

**Annuaire service registry**

Le registre de service est un annuaire de services. Le registre fournit un endroit central où les programmeurs peuvent publier de nouveaux services ou en trouver. Les interactions entre ces trois acteurs suivent plusieurs étapes :

* La publication du service : le fournisseur diffuse les descriptions de ses services Web dans l'annuaire.
* La recherche du service : le client cherche un service particulier, il s'adresse à un annuaire qui va lui fournir les descriptions et les URL des services demandés afin de lui permettre de les invoquer.
* L'invocation du service : une fois que le client récupère l'URL et la description du service, il les utilise pour l'invoquer auprès du fournisseur de services.

# SOA

## Définition

Une architecture orientée service se conforme à divers principes de gestion des services influençant directement le comportement intrinsèque d’une solution logicielle et le style de sa conception :

* L’encapsulation des services.
* Le faible couplage des services avec la maintenance d’une relation réduisant les dépendances.
* Le contrat de service adhère à un accord de communication, collectivement défini avec un ou plusieurs documents de description.
* L’abstraction des services dissimulant la logique du service à l’extérieur.
* La réutilisation des services partageant la logique entre plusieurs services avec l’intention de promouvoir la réutilisation.
* La composition des services.
* L’autonomie des services.
* L’optimisation des services.
* La découverte des services depuis leur description extérieure.

L’architecture orientée service représente un moyen technique d’intégration des divers systèmes d’information de l’entreprise considérant chaque ressource informatique comme un service. Elle permet de construire les buildings blocks qui composeront l'urbanisme du système d'information.

La notion d’interface est importante dans l’approche orientée service. En effet, elle représente le point d’entrée unique vers les fonctionnalités de la solution logicielle et assure la communication grâce à l’échange de messages. Chaque message est porteur de la sémantique particulière à la solution logicielle. De plus ce message est rédigé dans un langage compréhensible aux deux parties en présence. Les services proposés d’une architecture agile décrivent la structure des messages qu’ils attendent du client.

L’architecture orientée service est une solution logicielle distribuée. Elle propose un mécanisme d’échange de messages sécurisé entre les systèmes d’informations sous-jacents en employant des protocoles de communication standardisés. Cette approche offre à l’architecture une opportunité d’ouverture sur un large éventail de solutions logicielles existantes.

# Microservice

## Définition

Les microservices sont une architecture logicielle où une application est décomposée en plusieurs services indépendants et faiblement couplés, souvent spécialisés dans une seule tâche. Ces services communiquent les uns avec les autres en utilisant une API commune. Des API REST sont souvent employées pour relier chaque microservice aux autres.

## Caractéristiques communes des microservices

Caractéristiques de son organisation : Leur organisation contraste avec l’environnement monolithique. Puisqu’ils prennent en compte des aspects tels que les capacités, les besoins et les préférences de l’entreprise ou du client où il sera mis en œuvre. Quant à l’architecture, des modules multifonctionnels sont utilisés pour créer un module commun pour tous, offrant un service spécifique. Sans aucun doute, le grand avantage est le gain de temps et de confort dans les tâches de maintenance, empêchant le reste de l’équipe de terminer sa journée lors de la vérification d’un module.

Caractéristiques de son architecture : Chaque module est indépendant car chacun d’eux possède sa propre base de données, c’est-à-dire qu’ils ne vont pas tous au même. De cette façon, on évite de surcharger et de laisser tomber l’application.

Caractéristiques de leurs systèmes d’alerte et d’action : Comme plusieurs services sont connectés, on a besoin de systèmes d’alerte et d’action en cas de défaillance de ces services. C’est-à-dire qu’il nous donnerait un avertissement, enverrait un courriel à l’assistance, etc. Ce système est positif car il favorise une bonne gestion parmi les modules fonctionnels restants.

### Les avantages

* Mise à l'échelle indépendante ;
* Possibilité d'utiliser plusieurs socles techniques par service ;
* Liberté du développeur de développer et de déployer des services de manière indépendante ;
* Equipe de travail minimale ;
* Évolutivité;
* Le code pour différents services peut être écrit dans différentes langues ;
* Facile à comprendre et à modifier pour les développeurs, peut donc aider un nouveau membre de l'équipe à devenir productif rapidement ;
* Les développeurs peuvent utiliser les dernières technologies ;
* Lorsqu'une modification est requise dans une certaine partie de l'application, seul le service concernant peut être modifié et redéployé: il n'est pas nécessaire de modifier et de redéployer l'application entière ;
* Meilleure isolation des pannes: si un microservice échoue, l'autre continuera à fonctionner ;
* Facile à dimensionner et à intégrer à des services tiers ;
* Pas d'engagement à long terme envers la pile de technologie ;
* Utilisation de conteneurs permettant un déploiement rapide et le développement de l’application.

Les inconvénients

* La communication inter-service passe par des appels réseau, plus couteux qu'en mémoire ;
* Les appels entre les services peuvent avoir besoin d'authentification, surtout s’ils sont dans des réseaux différents, d’autorisation ou d’encryption. Cela augmente évidemment la complexité.
* Le debugging devient compliqué, surtout si ça vient de problèmes réseaux ou matériels. Il faut une bonne gestion de logs, de monitoring, etc.

Consommation de mémoire élevée

Besoin de temps pour fragmenter les différents microservices

Complexité de la gestion d’un grand nombre de services.

Besoin pour les développeurs de résoudre des problèmes tels que la latence du réseau ou l’équilibrage de charge.

Tests compliqués de déploiement distribué.

Un avantage important est que lors d'un besoin critique de mise à jour d'une ressource, seul le microservice contenant cette ressource sera mis à jour. L'ensemble de l'application reste compatible avec cette modification. Contrairement dans l’architecture monolithique, une modification apportée à une petite partie d'une application peut nécessiter la construction et le déploiement d'une version entièrement nouvelle.

L’architecture microservices est une approche d’architecture logicielle où une application est décomposée en plusieurs petits services qui sont souvent spécialisés dans une seule distincte tâche.

Un microservice est un petit service spécialisé dans une tâche spécise.

L’architecture répond aux problèmes rencontrées par les applications monolithes.

Une application monolithe est celle qui a pour ambition de traiter toutes les demandes possibles et répondre à un maximum de cas d’usage.

Avec le temps, elle a tendance à grossir car elle intègre toujours plus de nouvelles fonctionnalités et elle supprime rarement les anciennes fonctionnalités même si ces fonctionnalités deviennent inutiles ou obsolètes. Et avec le temps, des interdépendances entre les composants de l’application augmentent. L’application devient de plus en plus complexe, difficile à maîtriser et à faire évoluer.

Plus un projet est gros, plus il est critique pour l’entreprise. Il est risqué de tester des nouveautés. On va donc préférer la stabilité à l’innovation.

La réponse est de découper l’application en différents microservices.

Chaque microservice exécute une partie spécifique et unique de l’application. Et ces services peuvent être accédés par le client via leur API. Le but de l’architecture microservices est de redonner un maximum d’agilité et d’évolutivité à une application.

Côté technique l’architecture microservices est très souvent couplé à la technologie des conteneurs, et notamment Docker.

Quant aux conteneurs, Le principe est d’avoir un conteneur par unité de microservices. Ainsi, Chaque microservice a son propre environnement d’exécution pour héberger son code. Avec les conteneurs, Il est donc très facile en fonction du besoin d’adapter le nombre de microservices, à la hausse comme à la baisse.

SI un service est plus demandé lors d’un pic de charge, il suffit de créer plus de conteneurs fournissant ce microservice. A contrario, quand la demande de ce service diminue, il suffit de ne garder que le minimum de microservices pour couvrir la demande. Dans l’architecture microservices, il suffit juste d’adapter le nombre de ce microservice au besoin alors que dans les applications monolithes, il aurait fallu modifier la totalité de l’application à cause des interdépendances.

Et c’est en étant couplé avec l’utilisation des conteneurs que chaque microservice devient autonome vis à vis des autres microservices un microservice peut être alors modifiée, supprimée ou déployée sans impacter les autres microservices.

Ce qui veut dire qu’avec une application découpée en microservices, il est plus simple de cibler les différentes parties impactées d’une application lors d’une évolution. Ce qui facilite la modification des parties concernées sans risque pour le reste de l’application non impliqué.

Du coup, les mises à jour et nouvelles fonctionnalités peuvent être déployées beaucoup plus facilement et rapidement, rendant le déploiement continu possible.

Concernant l’organisation humaine, la taille des projets est limitée à une équipe de quelques personnes. Chaque équipe est autonome et va donc gérer son organisation et sa base de code.

Une équipe est alors libre des choix techniques en fonction de leurs besoins propres.

Terminons par des points d’attention à propos des microservices

Et tout ce que l’on vient de voir doit permettre aux entreprise d’avoir un atout très important et c’est d’ailleurs ça qui va pousser à l’adoption des microservices.

Cette adoption permet d’avoir un temps au marché plus rapide. Aujourd’hui un temps au marché rapide est devenu un facteur stratégique majeur, il permet à une entreprise de prendre un avantage concurrentiel décisif sur ces concurrents en intégrant la toute dernière innovation dans ses produits ou d’adapter ses produits aux nouveaux besoins du marché.

Comme on l’a vu tout à l’heure les microservices permettent justement de déployer plus rapidement et facilement. Ce qui veut dire être capable de s’adapter plus rapidement à la demande du marché et aux nouveautés.

Par contre et il est important de souligner :

Plus les itérations sont nombreuses et rapides (notamment dans le cadre d’un déploiement continu), plus cela va demander aux équipes de se synchroniser rapidement.

Cela va demander une organisation mature pour la communication et coordination entre les équipes et une synchronisation de leurs priorités et objectifs pour qu’ils soient alignés avec les enjeux business

# DevOps

DevOps est un ensemble de pratiques qui favorisent une meilleure communication et collaboration entre l’équipe des développeurs et l’équipe de d’exploitation pour avoir les tests et la diffusion de logiciels peuvent être réalisés rapidement, fréquemment et efficacement

Un développement constant

Des tests constants

Une intégration constante

Une mise en œuvre constante

Une surveillance constante

, en automatisant le processus de livraison de logiciels et les changements d’infrastructure.

DevOps vise à créer une culture et un environnement dans lesquels la conception,

En raison d’un développement orienté client et l’adoption de DevOps, la double compétence: développeur et ingénieur système est de plus en plus recherchée.

### Les démarches fondatrices de DevOps

Le mouvement DevOps s’appuie sur l’adoption et l’intégration de trois principales démarches ou méthodes actuelles :

Les méthodes Agile de développement logiciel telles que Scrum.

La gestion des Services IT (ITSM) liée aux bonnes pratiques préconisées par ITIL.

Lean qui permet d’optimiser le travail et améliorer la qualité de la production

# Les phases de développement

Le développement informatique se décompose en quelques phases :

### Expression de votre projet

Vous décrivez votre projet tel que vous l’imaginez : son usage, son mode de fonctionnement, sa présentation. Notre équipe d’ingénieurs peut être amenée à rencontrer les acteurs de votre société aux différents niveaux hiérarchiques, afin d’intégrer la logique et l’organisation de l’entreprise. Leur expertise permettra de déceler les points d’amélioration informatique afin d’apporter un gain de productivité.

### Le cahier des charges

Lorsque toutes les informations nécessaires à la réalisation du projet ont été récoltées, la rédaction du cahier des charges peut commencer.

Il se structure de cette manière :

### Les spécifications

Cette première partie décrit de façon la plus précise possible, le fonctionnement de l’application. Chaque fonctionnalité de l’application y est décrite en détail.

### Les conditions techniques

Ce sont les moyens matériels nécessaires au fonctionnement de l’application. Il s’agit de définir la configuration matérielle requise pour chaque composant de l’application. Par exemple :

* La machine client et la machine serveur dans le cas d’une application client/serveur, ou le type d’hébergement dans le cas d’une application en ligne.
* Le système d’exploitation requis, ou l’indépendance dans le cas d’une application multi-plateformes.
* La configuration réseau et les protocoles nécessaires.

### Le délai

Plus le cahier des charges est détaillé, et plus notre estimation des délais est précise. Nous nous engageons à respecter les délais donnés.

Lorsque nous recevons votre validation du cahier des charges, nous débutons le développement.

## Phase de développement

Pendant la période de développement nous restons très proche de vous. Vous avez accès à l’interface client de notre site pour suivre l’avancée de votre projet, et nous contacter quand vous le désirez.

Lorsque le développement de votre projet arrive à un stade suffisamment avancé pour être utilisé (toutes les fonctionnalités ont été développées), nous vous confions une pré-version de l’application, afin que vous puissiez vérifier l’implémentation des fonctionnalités spécifiées dans le cahier des charges, et nous faire part de vos remarques.

Pendant ce temps, nos développeurs passent en phase de test.

## Phase de test

Nous développons des simulateurs spécifiques afin de tester votre application dans différentes conditions. Votre application subit alors des batteries de test d’utilisation en condition normale et extrême.

Nous vérifions par exemple la réactivité de l’application en cas de surcharge ou coupure réseaux, mais aussi au niveau sécurité en testant différentes techniques d’attaque informatique. Durant cette phase les bugs initiaux sont corrigés, la sécurité est renforcée et les performances générales de l’application sont optimisées.

Cette phase est primordiale lors du développement de logiciels. Elle assure la livraison d’une application de qualité et limite l’effort nécessaire à la maintenance future du logiciel.

## Livraison, intégration et tests

Lorsque l’application est estimée prête à fonctionner, elle vous est livrée, et installée en condition réelle d’utilisation. À ce stade, nous faisons avec vous le tour de l’application, cahier des charges à la main, afin de vérifier que l’application est conforme à vos exigences. Vous pouvez alors débuter vos propres tests avec un nombre limité d’utilisateurs.

## Mise en production

Une fois l’épreuve des tests validée, l’application peut être mise en production en toute sécurité.