# Propositions d’architecture de l’application PoneyHiring

Par Johnny YVARD

# Table des matières

# 1 Rappel du contexte

Ceci est la troisième version de l'application "PoneyHiring".
Nous passons à l'international !
Notre business est florissant et nous avons de nouveaux clients autour du monde tous les jours, mais il devient de plus en plus difficile pour nos salariés de travailler avec notre « bon vieux monolithe » pour faire tourner la société.
Nous allons avoir besoin de changement pour nous adapter au contexte international et que nos différents services puissent faire leur travail le plus simplement possible.
Notre compagnie a maintenant 1000 employés, des bureaux à Paris, Londres, New York, Tokyo, Melbourne (qui vient d'ouvrir) et Kuala Lumpur (notre plus gros chiffre d'affaire, le poney c'est sérieux là bas !).
Les services sont divisés en Locations, Comptabilité, Ressources Humaines, Marketing, Gestion des Stocks et la toute nouvelle division Mobile (application de location de poney à la Uber).
Notre mission, proposer 2 designs d'architecture pour aider la société à grandir et fonctionner mais aussi pour « exploser le monolithe ».

# 2 Première proposition

## 2.1 L'architecture n-tier

C'est une architecture client serveur dans laquelle l'application sera exécutée par plusieurs composants logiciels distincts.

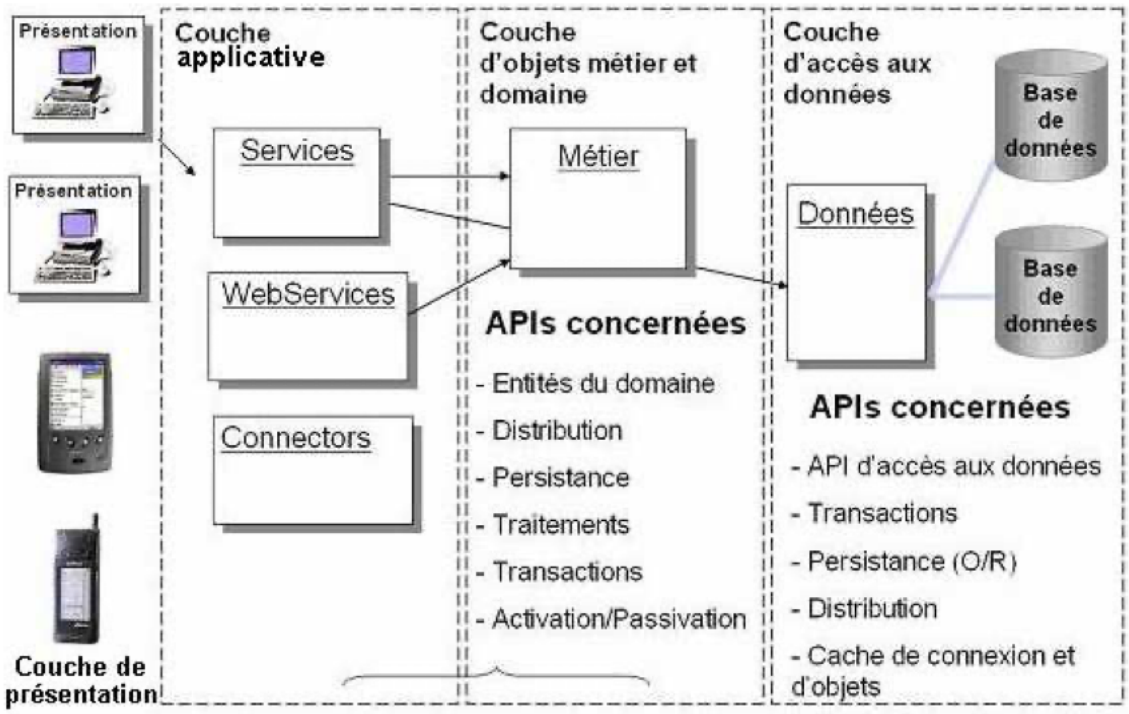
## 2.2 Exemple d'architecture 3-tier

- **Tier de présentation** : interfaces utilisateurs sur un PC qui s'adressent à des applications serveur.
- **Tier des règles de gestion** : applications serveur qui contiennent la logique de gestion et accèdent aux données stockées dans des bases de données.
- **Tier de base de données** : serveurs de bases de données.

## 2.3 Les avantages d'une telle architecture

- Le lien entre les niveaux défini et limité à des interfaces.
- Les interfaces assurent la modularité et l'indépendance technologique et topologique de chaque niveau.

## 2.4 Différentes couches d'une architecture 4-tier




— **La couche de présentation** contient les différents types de clients, léger ou lourd.

— **La couche applicative** contient les traitements représentant les règles métier.

— **La couche d'objets métier** est représentée par les objets du domaine, c'est à dire l'ensemble des entités persistantes de l'application : nos bureaux à Paris, Londres, New York, Tokyo, Melbourne et Kuala Lumpur mais aussi nos services Locations, Comptabilité, Ressources Humaines, Marketing, Gestion des Stocks et la nouvelle division Mobile.

— **La couche d'accès aux données** contient les usines d'objets, c'est à dire les classes chargées de créer des objets métier de manière totalement transparente, indépendamment de leur mode de stockage.

## 2.5 La valeur ajoutée des architectures n-tier

— Cette **séparation par couches de responsabilités** sert à découpler au maximum une couche de l'autre afin d'éviter l'impact d'évolutions futures de l'application.

— Par exemple, si l'on est amené à devoir changer de base de données relationnelle, seule la couche d'accès aux données sera impactée, la couche de service et la couche de présentation ne seront pas concernées car elles auront été découplées des autres.

# 3 Deuxième proposition

## 3.1 L'architecture orientée services

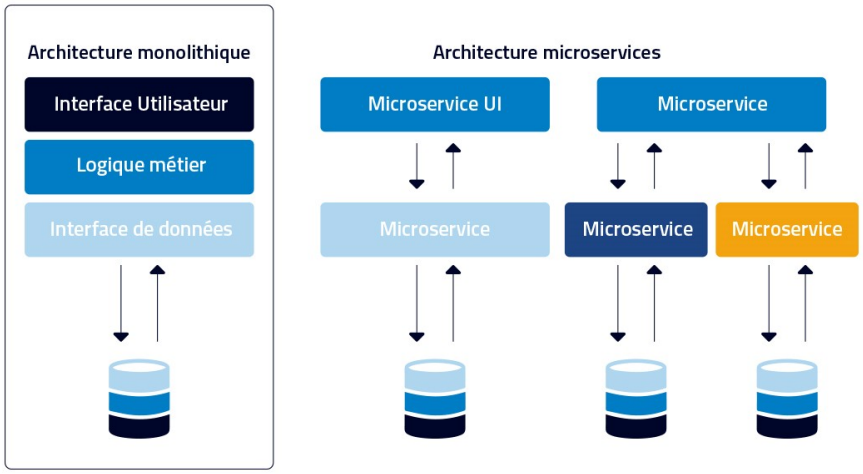
De l'anglais "**Service-Oriented Architecture**" (SOA). C'est un modèle de développement logiciel à base de composants applicatifs distribués et doté de fonctions de découverte, de contrôle d'accès, de mappage de données et de sécurité.
L'architecture SOA a deux grandes fonctions. Tout d'abord, il s'agit de créer un ample modèle d'architecture qui définit les objectifs des applications et les approches pour les atteindre ; ensuite, de définir des caractéristiques de mise en œuvre précises, souvent liées à celles du langage de description de services WSDL (Web Services Description Language) et du **protocole SOAP** (Simple Object Access Protocol).

## 3.2 Principaux objectifs de l'architecture orientée services

On dénombre trois grands objectifs de l'architecture orientée services, chacun axé sur une partie distincte du **cycle de vie applicatif**.
Le premier vise à structurer sous forme de services les procédures ou composants logiciels. Ces services sont conçus pour être faiblement couplés aux applications : ils ne servent qu'en cas de besoin. Ils sont prévus pour que les développeurs, tenus de standardiser la création de leurs applications, les utilisent facilement.
Le deuxième objectif est de fournir un mécanisme de publication des services disponibles qui comprend la fonctionnalité et les besoins d'entrée/sortie (**E/S** ou I/O). Les services sont publiés de manière à faciliter leur intégration aux applications.
Le troisième objectif de l'architecture SOA est de contrôler l'utilisation de ces services pour éviter tout problème de sécurité et de gouvernance. La sécurité de cette SOA est surtout axée sur la sécurité des composants individuels en son sein, sur les procédures d'authentification et d'identification en lien avec ces composants, et la sécurisation des connexions entre les composants de l'architecture.

## 3.3 Architecture SOA et microservices

La tension entre les deux visions, ensemble de principes et mise en oeuvre logicielle spécifique, culmine avec l'arrivée de deux phénomènes : la virtualisation et le **cloud computing**. Combinés, ils vont pousser les développeurs à concevoir des applications à partir de composants fonctionnels plus petits. Les **microservices**, une des tendances logicielles aiguës du moment, ont été l'apogée de ce modèle de développement. Plus il y a de composants, plus il faut d'interfaces et plus la conception logicielle se complique : la tendance a mis au jour la complexité et les défauts de performance de la plupart des mises en oeuvre SOA.



# 4 Conclusion

Finalement, les architectures logicielles à base de microservices ne sont que des mises en oeuvre actualisées du modèle SOA. Les composants logiciels sont conçus comme des services à exposer via des API, comme l'exige la SOA. Un broker d'API fait l'intermédiaire : il donne accès aux composants et garantit l'observation des règles de sécurité et de gouvernance. Par des techniques logicielles, il assure la correspondance entre les différents formats d'E/S des microservices et les applications qui les utilisent.

Mais l'architecture SOA reste valable aujourd'hui comme au premier jour. Ses principes nous ont amenés au cloud et prennent en charge les techniques les plus avancées de développement de logiciels cloud actuellement en usage.