



ARCHITECTURES LOGICIELLES

Architectures Micro-Services

Ibtihel FEKIH

E-mail: ibtihel.fekih.alaya@gmail.com



Plan du cours

Architecture Micro-Services

- Définition
- Domaine d'application
- Objectifs
- Avantages & Bénéfices
- Inconvénients
- Modélisation de l'architecture Micro-services
- Documentation de son architecture

Le métier d'ARCHITECTE système

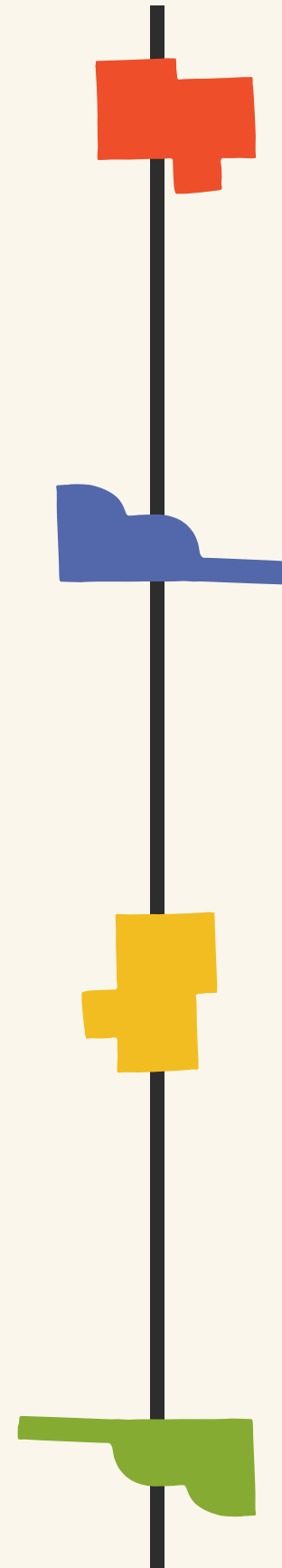
- Description du métier
- Compétence architecte logiciel

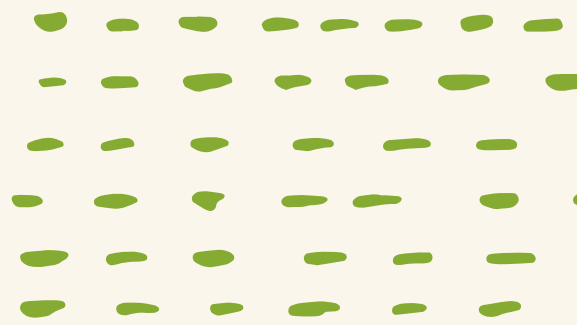
Architectures logicielles

- Définition & Contexte
- Critères de qualité logicielle
- Types d'architectures
- L'importance de l'architecture
- Quelle architecture pour son système
- Caractéristiques d'une bonne architecture

Cas Pratique

- Concevoir une architecture micro-service d'une plateforme E-commerce
- Documenter l'architecture
- Définir le backlog des différents micro-services à développer sur Trello
- Travailler en mode Squad teams pour développer les micro-services
- Documentation du code





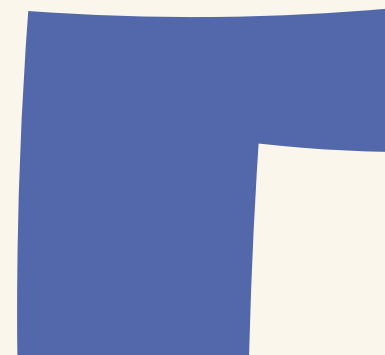
Architecture logicielle

Définition & Contexte

Définition

L'architecture logicielle décrit les différents éléments d'une application et leurs interactions.

La conception de l'architecture est donc une étape particulièrement cruciale du développement logiciel. De cette phase, va dépendre la stabilité, la robustesse ou encore la scalabilité d'une application.





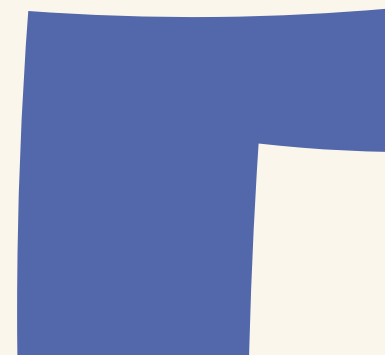
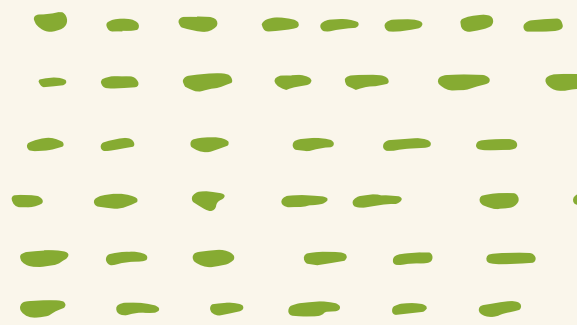
Architecture logicielle

Définition & Contexte

Contexte

L'architecture logicielle est conçue en prenant en considération les besoins du futur utilisateur de l'application. Sa définition réside dans la description générale et détaillée d'un logiciel et précisant chaque sous-système.

L'architecture logicielle constitue un compromis entre les exigences de l'utilisateur, qui s'attend probablement à avoir entre ses mains un logiciel facile d'utilisation, performant, et pouvant interagir avec d'autres applications, et des exigences en termes de fiabilité et de sécurité, le tout en proposant le meilleur rapport qualité de conception - Coût.



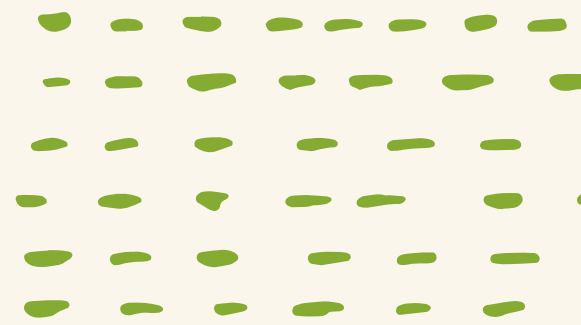


Architecture logicielle

Critères de qualité logicielle

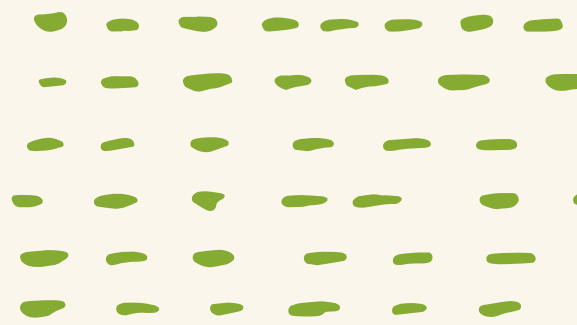
Partie 1

Critères de qualité logicielle



- L'**interopérabilité extrinsèque** exprime la capacité du logiciel à communiquer et à utiliser les ressources d'autres logiciels comme les documents créés par une certaine application.
- L'**interopérabilité intrinsèque** exprime le degré de cohérence entre le fonctionnement des commandes et des modules à l'intérieur d'un système ou d'un logiciel.
- La **portabilité** exprime la possibilité de compiler le code source et/ou d'exécuter le logiciel sur des plates-formes (machines, systèmes d'exploitation, environnements) différentes.





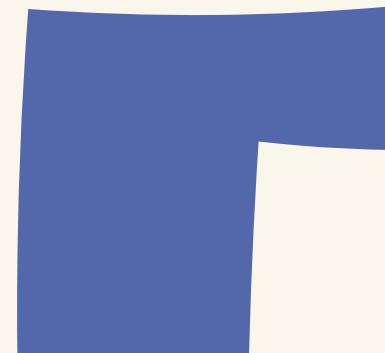
Architecture logicielle

Critères de qualité logicielle

Partie 2

Critères de qualité logicielle

- **La compatibilité** exprime la possibilité, pour un logiciel, de fonctionner correctement dans un environnement ancien (compatibilité descendante) ou plus récent (compatibilité ascendante).
- **La validité** exprime la conformité des fonctionnalités du logiciel avec celles décrites dans le cahier des charges.
- **La vérifiabilité** exprime la simplicité de vérification de la validité.
- **L'intégrité** exprime la faculté du logiciel à protéger ses fonctions et ses données d'accès non autorisés.
- **La fiabilité** exprime la faculté du logiciel à gérer correctement ses propres erreurs de fonctionnement en cours d'exécution.





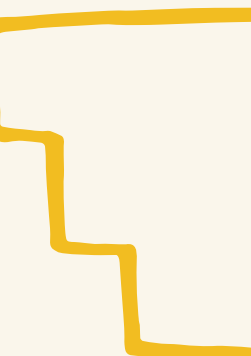
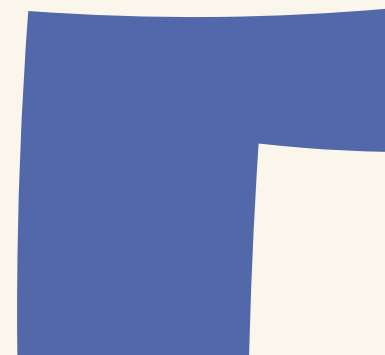
Architecture logicielle

Critères de qualité logicielle

Partie 3

Critères de qualité logicielle

- **La maintenabilité** exprime la simplicité de correction et de modification du logiciel, et même, parfois, la possibilité de modification du logiciel en cours d'exécution.
- **La réutilisabilité** exprime la capacité de concevoir le logiciel avec des composants déjà conçus tout en permettant la réutilisation simple de ses propres composants pour le développement d'autres logiciels.
- **L'extensibilité** exprime la possibilité d'étendre simplement les fonctionnalités d'un logiciel sans compromettre son intégrité et sa fiabilité.





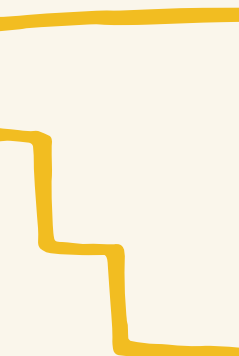
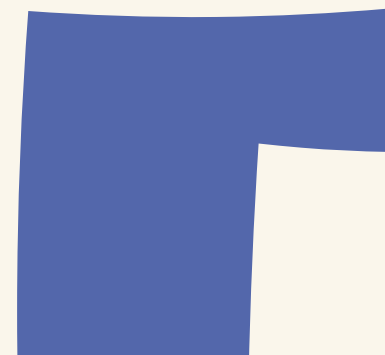
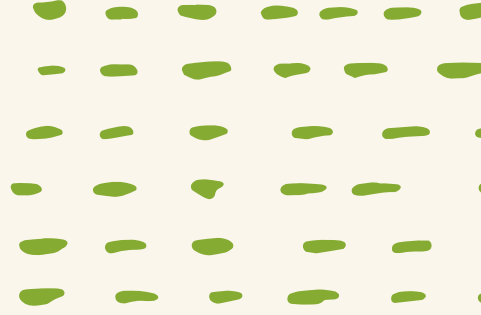
Architecture logicielle

Critères de qualité logicielle

Partie 4

Critères de qualité logicielle

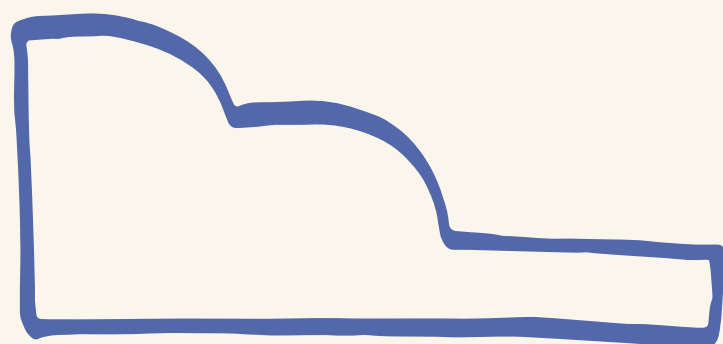
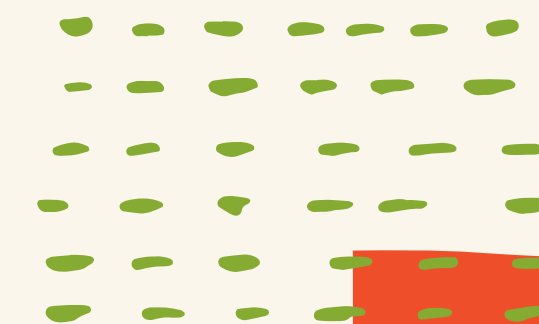
- **L'efficacité** exprime la capacité du logiciel à exploiter au mieux les ressources offertes par la ou les machines où le logiciel sera implanté.
- **L'autonomie** exprime la capacité de contrôle de son exécution, de ses données et de ses communications.
- **La transparence** exprime la capacité pour un logiciel de masquer à l'utilisateur (humain ou machine) les détails inutiles à l'utilisation de ses fonctionnalités.
- **La composabilité** exprime la capacité pour un logiciel de combiner des informations provenant de sources différentes.
- La convivialité décrit la facilité d'apprentissage et d'utilisation du logiciel par les usagers.





Motivés pour la suite ?

On découvre les types des architectures ..



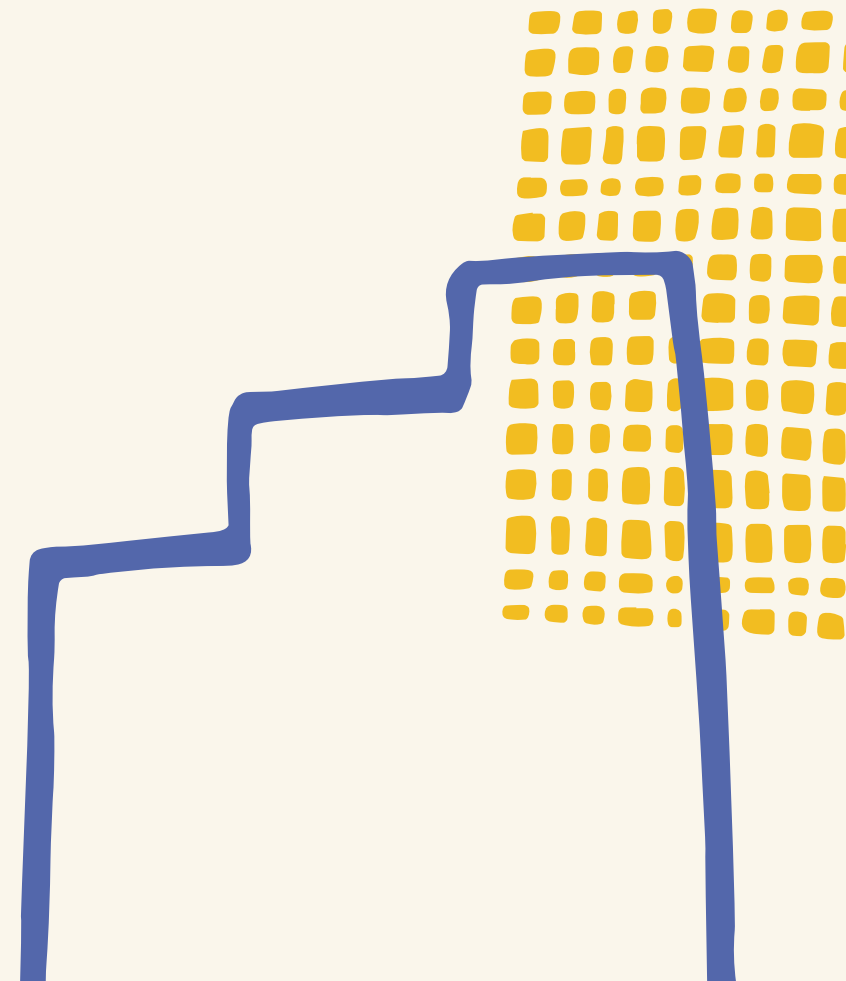
Types des architectures

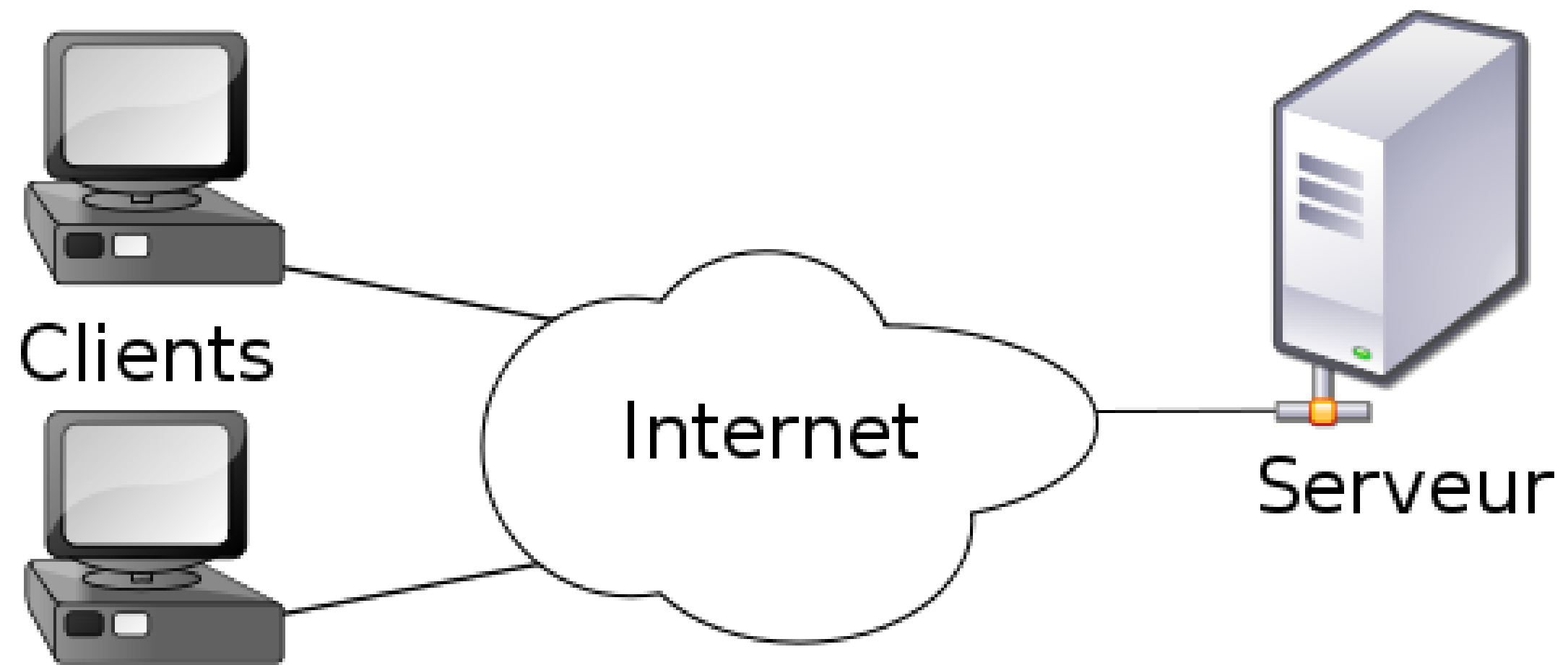
Architecture client/serveur

L'architecture client/serveur caractérise un système basé sur des échanges réseau entre des clients et un serveur centralisé, lieu de stockage des données de l'application.

On peut classer les clients d'une architecture client/serveur en plusieurs types :

- **client léger**, destiné uniquement à l'affichage (exemple : navigateur web) ;
- **client lourd**, application native spécialement conçue pour communiquer avec le serveur (exemple : application mobile) ;
- **client riche** combinant les avantages des clients légers et lourds (exemple : navigateur web utilisant des technologies évoluées pour offrir une expérience utilisateur proche de celle d'une application native).





Types des architectures

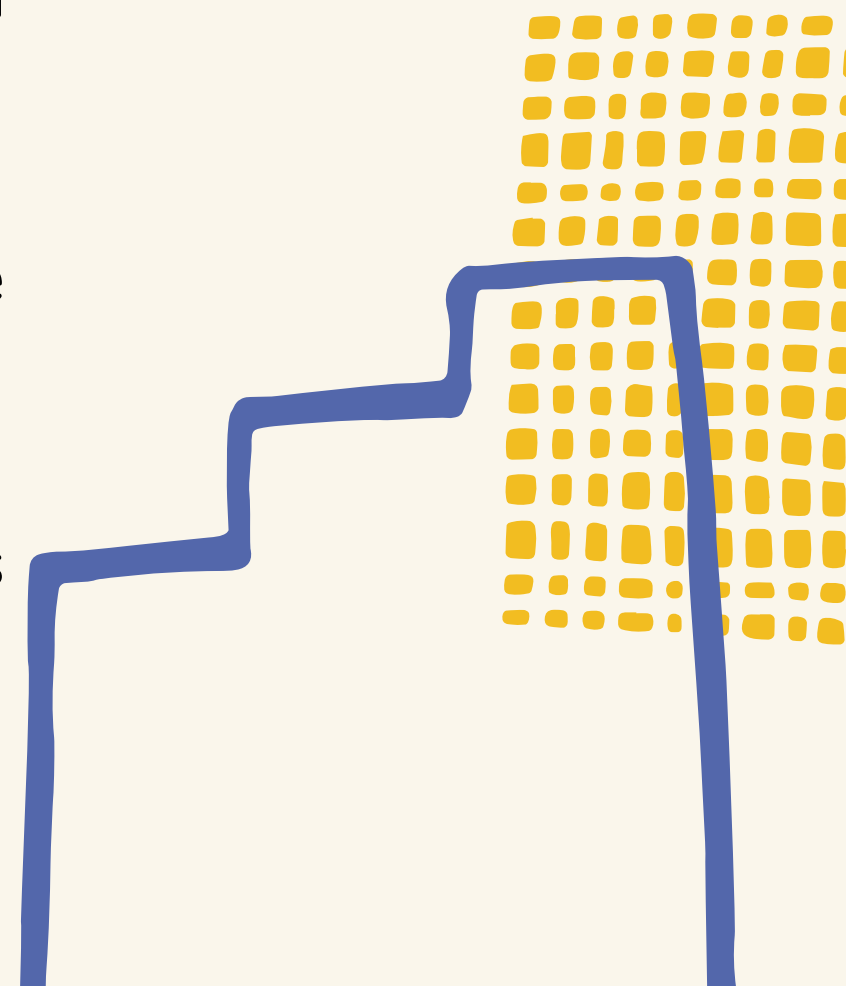
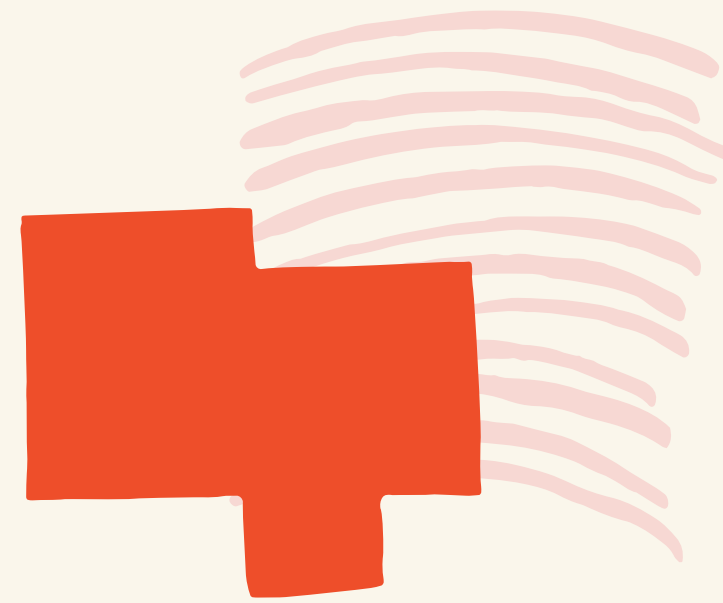
Architecture en couches

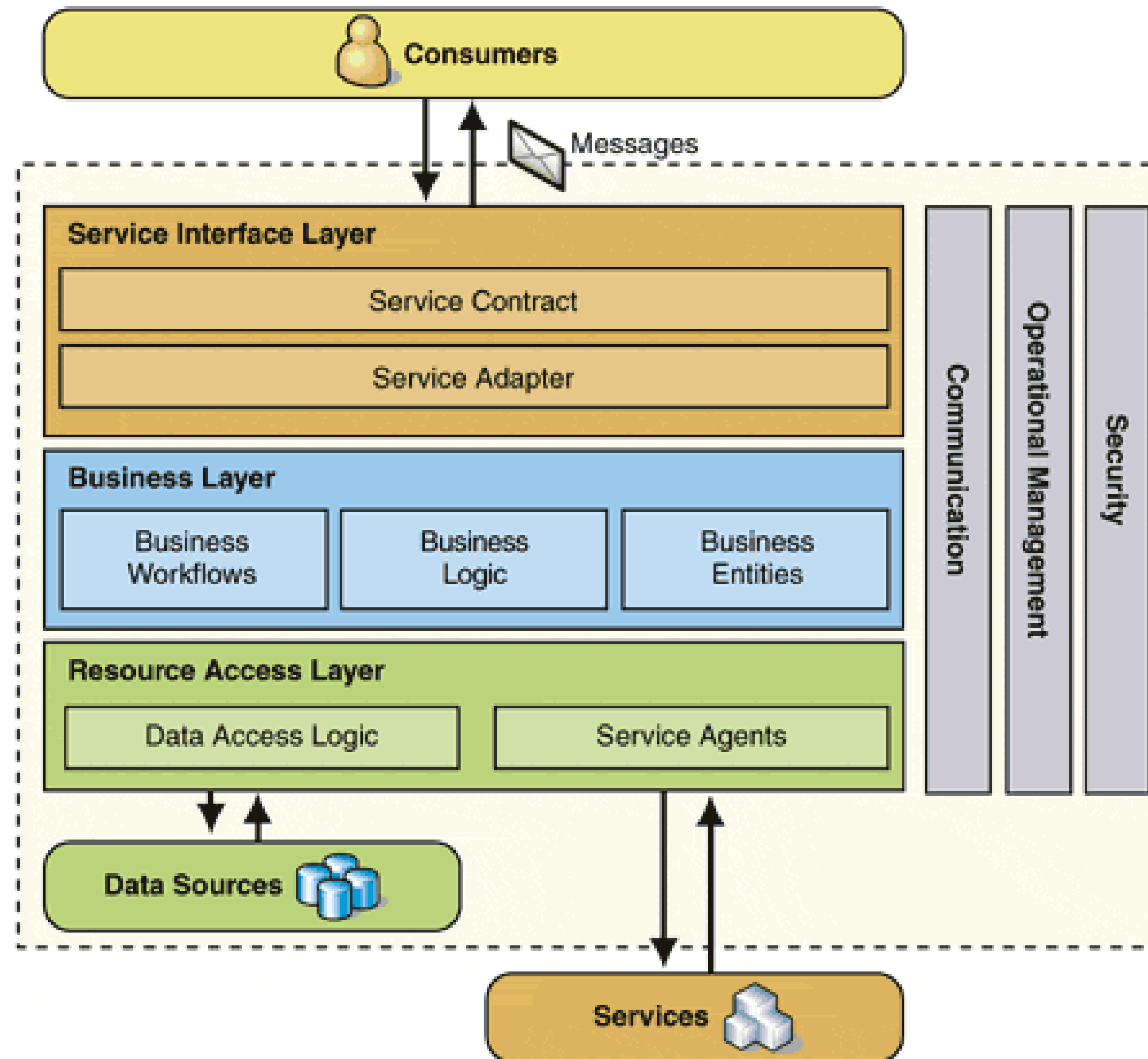
Une architecture en couches organise un logiciel sous forme de couches (layers). Chaque couche ne peut communiquer qu'avec les couches adjacentes.

Cette architecture respecte le principe de séparation des responsabilités et facilite la compréhension des échanges au sein de l'application.

Lorsque chaque couche correspond à un processus distinct sur une machine, on parle d'architecture n-tiers, n désignant le nombre de couches.

Un navigateur web accédant à des pages dynamiques intégrant des informations stockées dans une base de données constitue un exemple classique d'architecture 3-tiers.





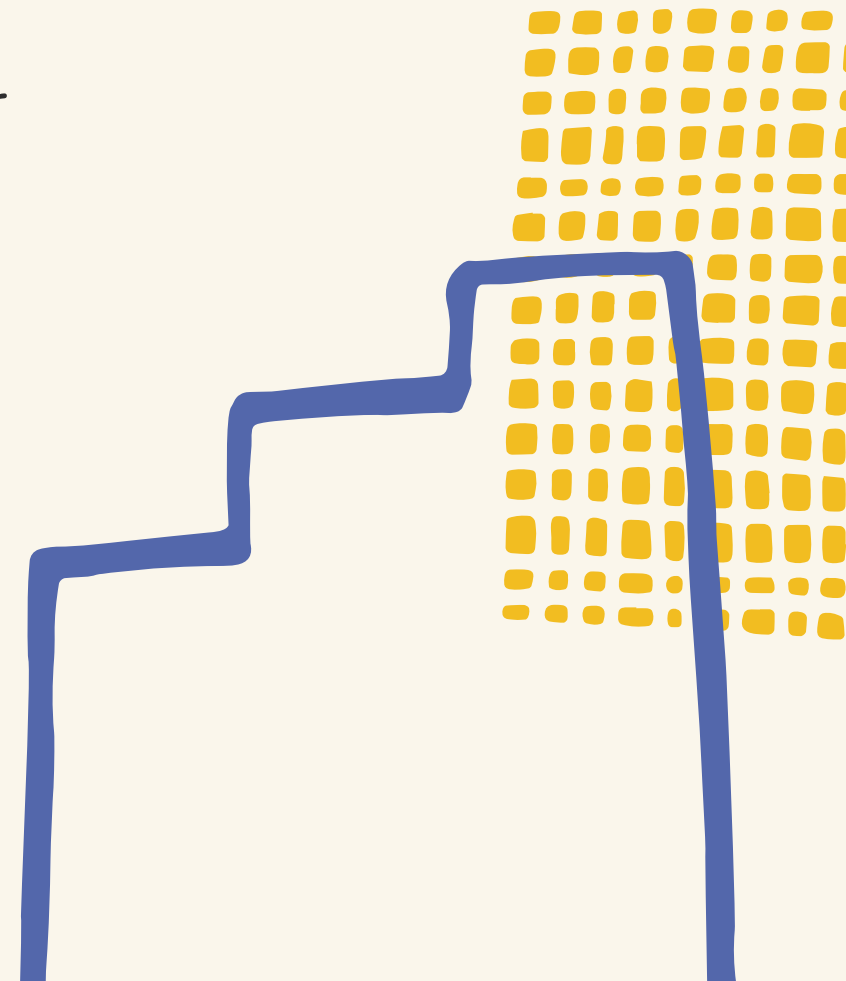
Types des architectures

Architecture orientée services

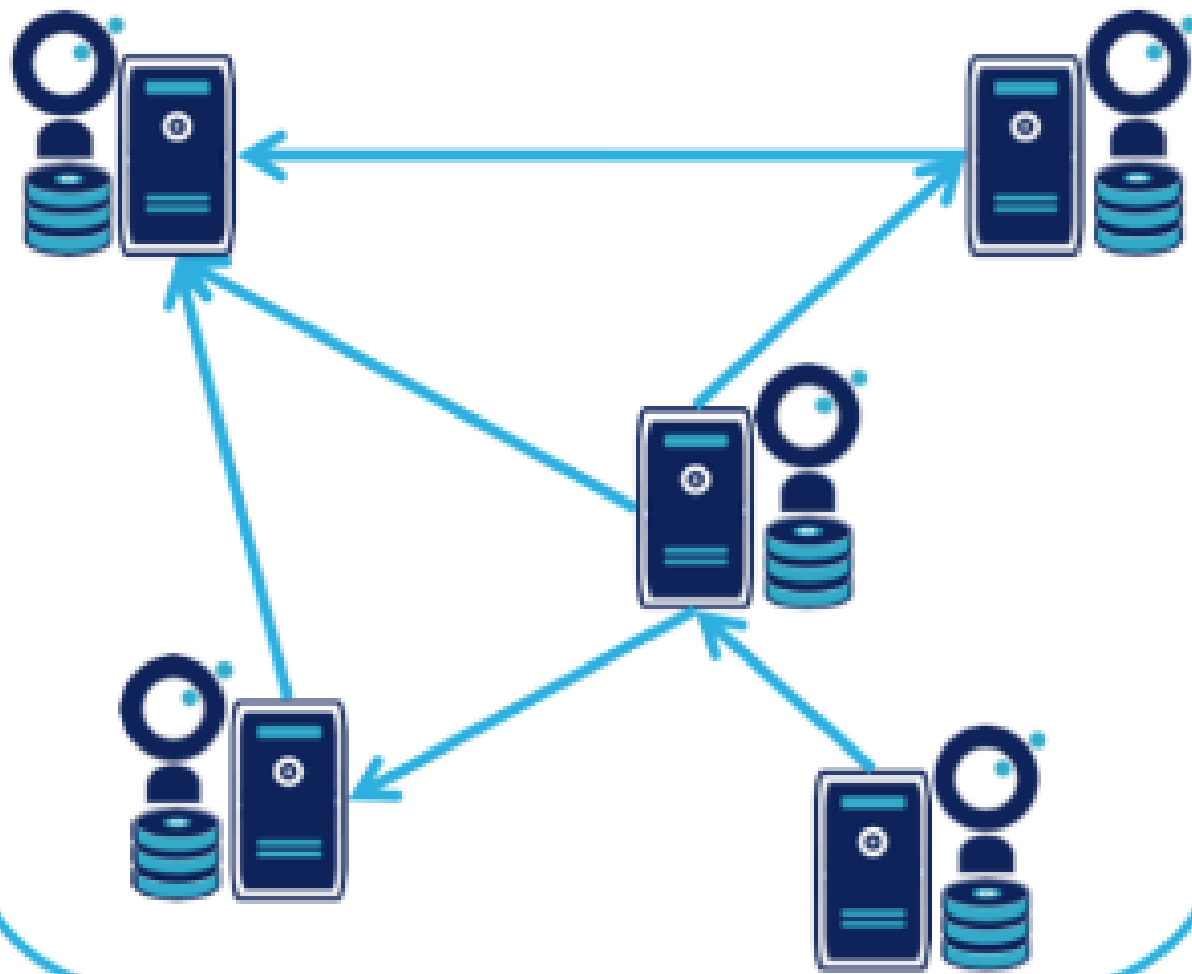
Une architecture orientée services (SOA, Service-Oriented Architecture) décompose un logiciel sous la forme d'un ensemble de services métier utilisant un format d'échange commun, généralement XML ou JSON.

Une variante récente, l'architecture microservices, diminue la granularité des services pour leur assurer souplesse et capacité à évoluer, au prix d'une plus grande distribution du système.

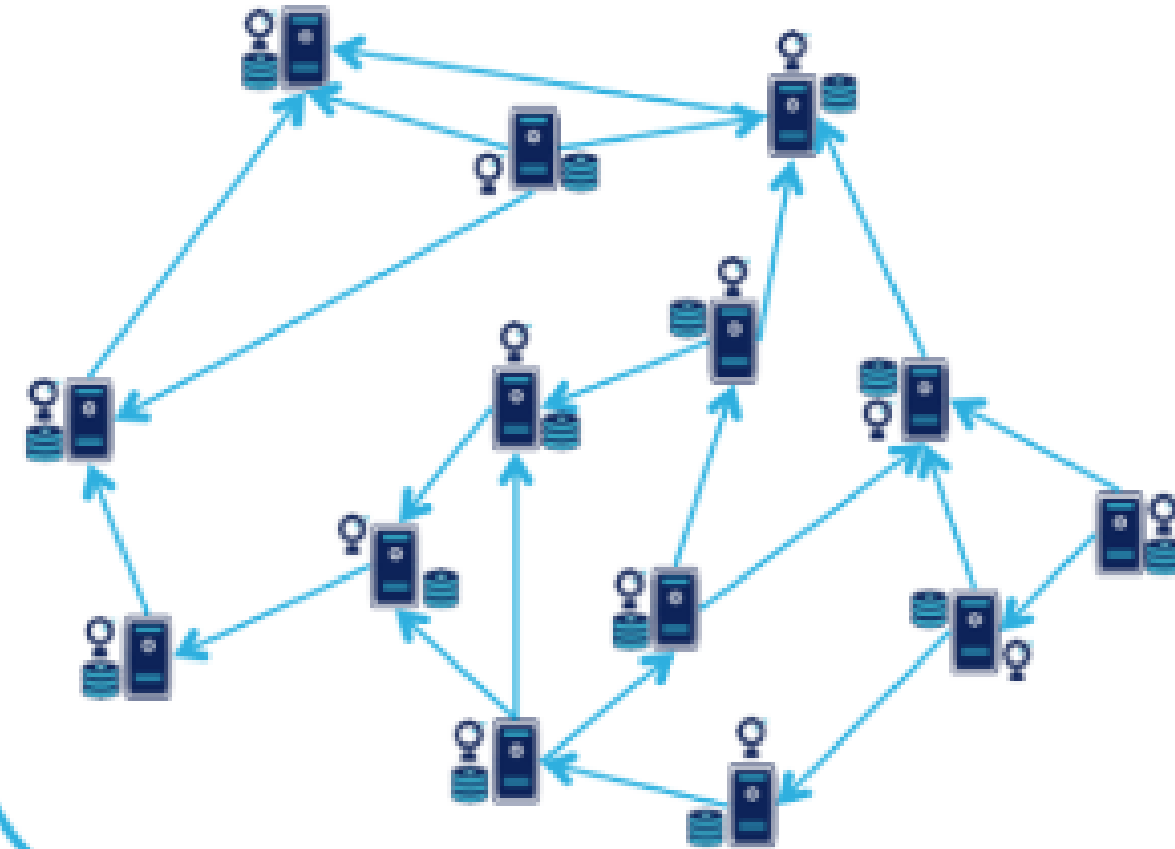
L'image ci-après illustre la différence entre ces deux approches.



Architecture de Services



Architecture Microservices



Types des architectures



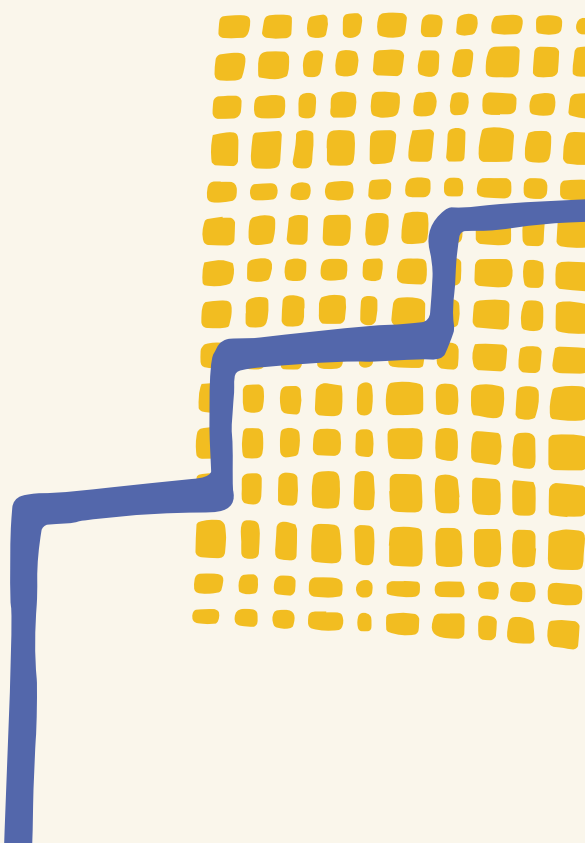
Architecture Modèle–Vue–Contrôleur

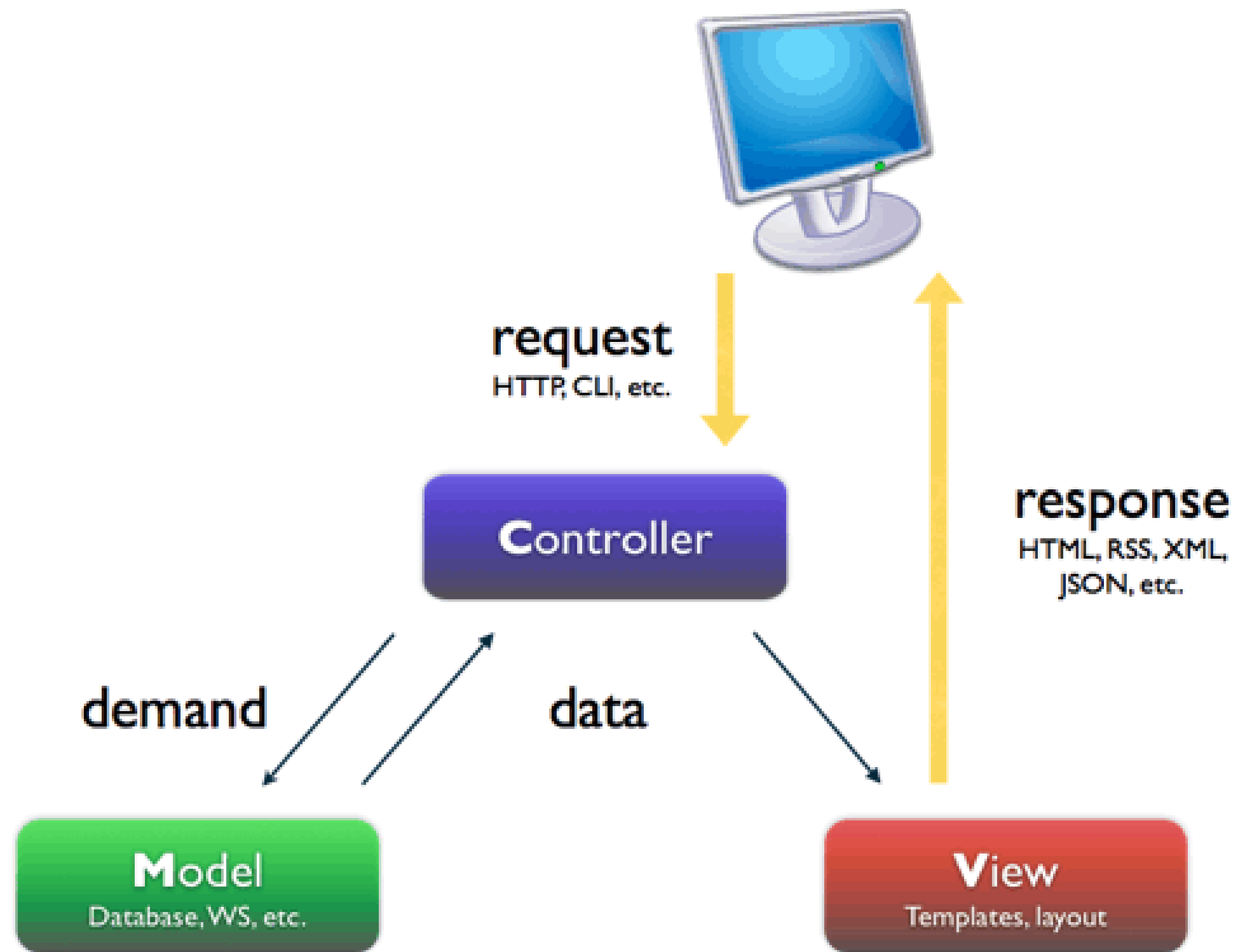
L'architecture Modèle-Vue-Contrôleur, ou MVC, décompose une application en trois sous-parties :

- la partie **Modèle** qui regroupe la logique métier (« business logic ») ainsi que l'accès aux données. Il peut s'agir d'un ensemble de fonctions (Modèle procédural) ou de classes (Modèle orienté objet) ;
- la partie **Vue** qui s'occupe des interactions avec l'utilisateur : présentation, saisie et validation des données ;
- la partie **Contrôleur** qui gère la dynamique de l'application. Elle fait le lien entre les deux autres parties.

Ce patron permet de bien séparer le code de l'interface graphique de la logique applicative. On le retrouve dans de très nombreux langages : bibliothèques Swing et Model 2 (JSP) de Java, frameworks PHP, ASP.NET MVC, etc.

Le diagramme ci-après (extrait de la documentation du framework PHP Symfony) résume les relations entre les composants d'une architecture web MVC.





Types des architectures

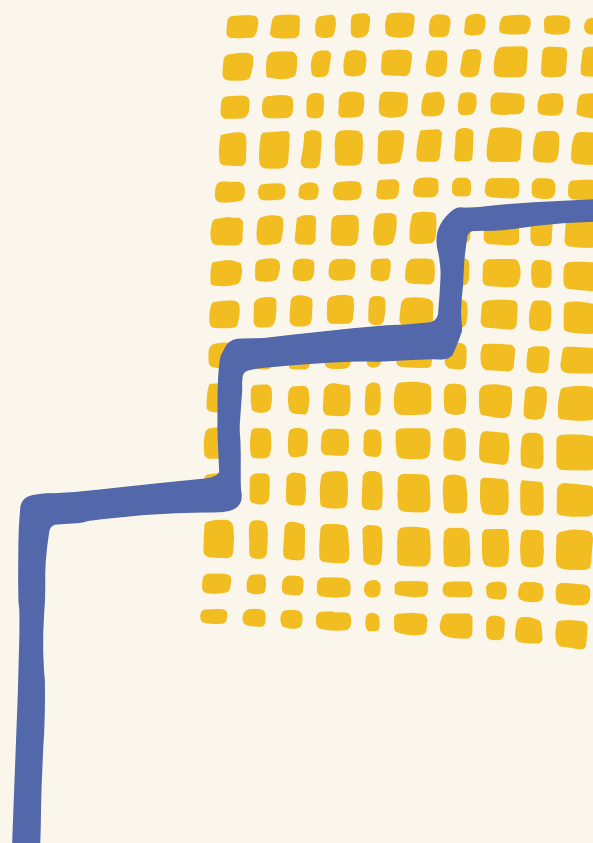


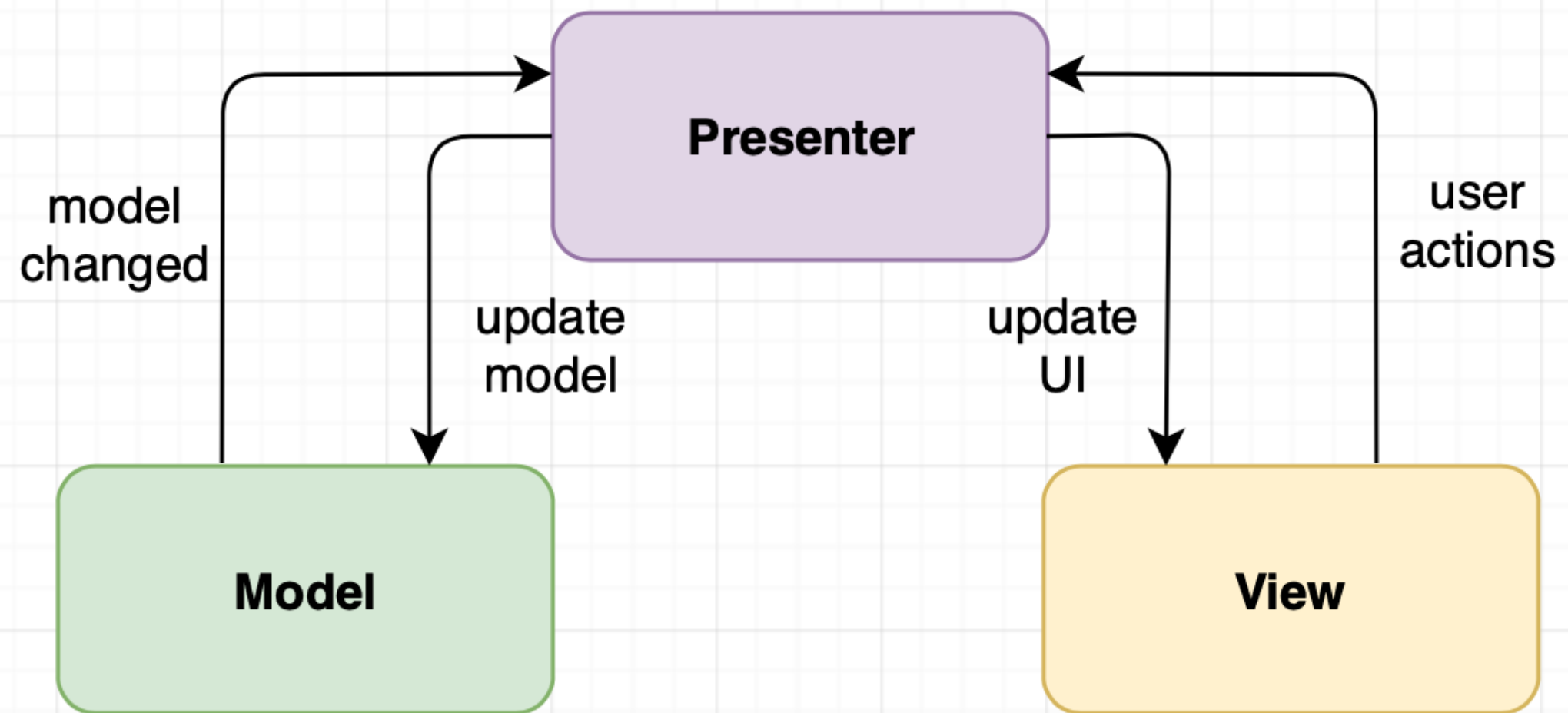
Architecture Modèle-Vue-Présentation

La patron Modèle-Vue-Présentation, ou MVP, est un proche cousin du patron MVC surtout utilisé pour construire des interfaces utilisateurs (UI).

Dans une architecture MVP, la partie Vue reçoit les événements provenant de l'utilisateur et délègue leur gestion à la partie Présentation. Celle-ci utilise les services de la partie Modèle puis met à jour la Vue.

Dans la variante dite Passive View de cette architecture, la Vue est passive et dépend totalement du contrôleur pour ses mises à jour. Dans la variante dite Supervising Controller, Vue et Modèle sont couplés et les modifications du Modèle déclenchent la mise à jour de la Vue.

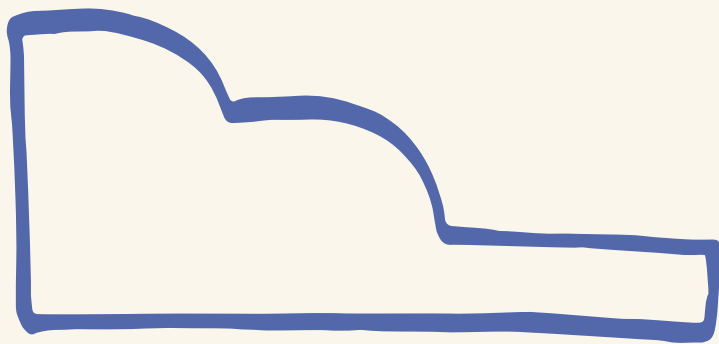




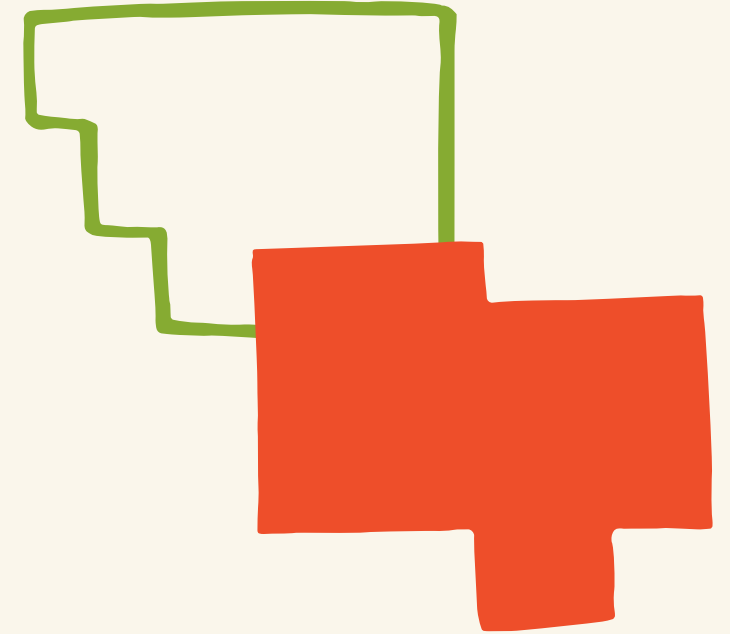


Maintenant.. Quelle importance ?

On découvre pourquoi on développe une architecture logicielle !

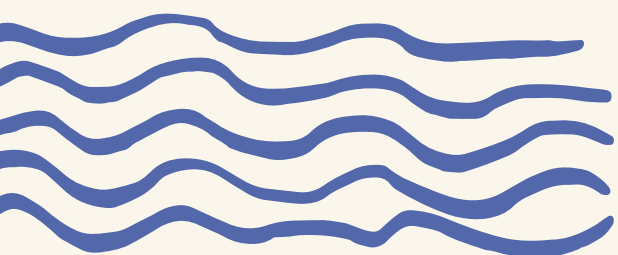


L'importance de l'architecture



Compréhension

Pour permettre à tous de mieux comprendre le système



Collaboration

Pour permettre aux développeurs de travailler sur des parties individuelles du système en isolation

Extensibilité

Pour préparer les extensions du système

Réutilisabilité

Pour faciliter la réutilisation et la réutilisabilité





Architecture logicielle

Quelle architecture pour son
système ?

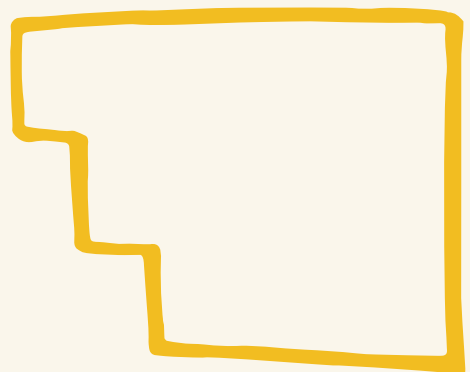
Quelle architecture pour son système ?

Le choix dépend de:

- La **compréhension** du besoin métier et des contraintes fonctionnelles et non fonctionnelles du logiciel.
- 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.

Les contraintes non fonctionnelles comprennent par exemple les aspects :

- De performance
- De sécurité





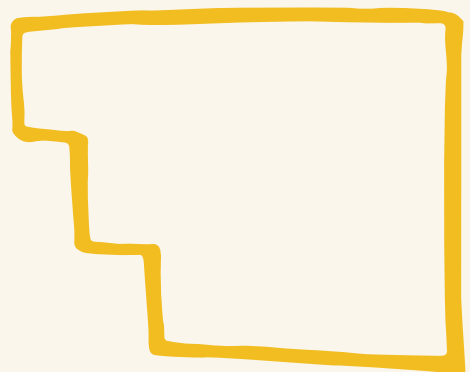
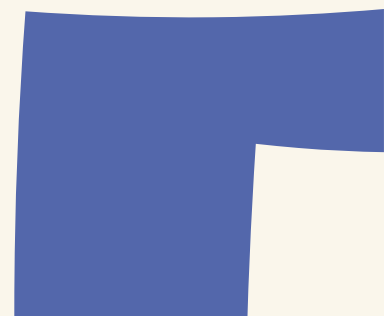
Architecture logicielle

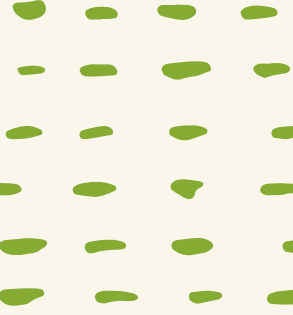
Quelle architecture pour son
système ?

Quelle architecture pour son système ?

Mais également les contraintes d'exécution et d'exploitation incluant :

- l'hébergement attendu : Cloud privé et/ou public,...
- les systèmes d'exploitation.
- le choix des technologies
- le besoin de résilience du système.



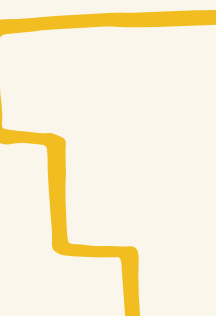
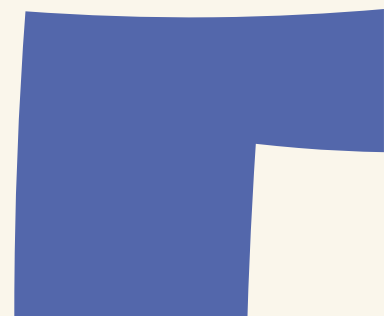


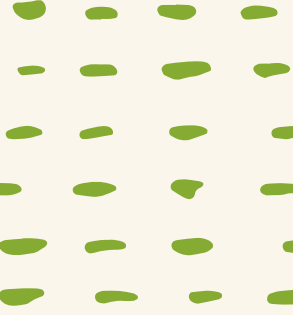
Caractéristiques d'une bonne architecture

Architecture logicielle

Caractéristiques d'une bonne architecture !

- Son **évolutivité** : doit prendre en compte les évolutions futures du logiciel en fonction du besoin métier
- Sa **simplicité**: une architecture complexe est souvent source de défaillance et peut créer de la dette technique.
- Sa **maintenabilité** : qui intègre aussi l'outillage nécessaire à sa maintenance



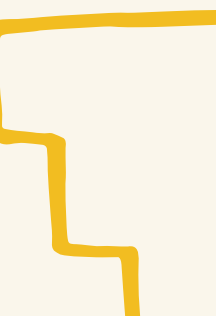
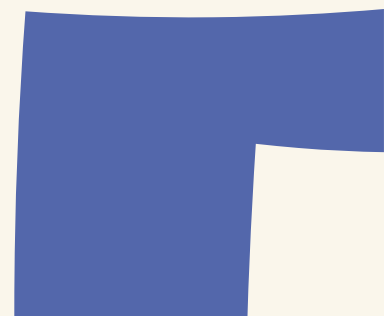


Caractéristiques d'une bonne architecture

Architecture logicielle

Caractéristiques d'une bonne
architecture !

- Sa **compatibilité** : avec les différentes plates-formes matérielles,
- Son **interconnectivité** : Puisque le logiciel évolue dans un certain environnement, son architecture doit permettre son interconnectivité avec d'autres systèmes d'information





Architecture Micro-Services

Définition

Définition

L'architecture Microservices propose une solution en principe simple : **découper** une application en petits services, appelés des **Microservices**, parfaitement autonomes qui exposent une **API REST** que les autres Microservices pourront consommer.







Architecture Micro-Services

Définition

Définition

Chaque Microservice est parfaitement autonome : il a sa propre base de données, son propre serveur d'application, ses propres librairies et ainsi de suite.

La plupart du temps ces Microservices sont chacun dans un container Docker, ils sont donc totalement indépendants y compris vis-à-vis de la machine sur laquelle ils tournent.






Architecture Micro-Services

Domaine d'application



Domaine d'application

Les services dans l'architecture Microservices sont très spécialisés. Ils s'occupent d'une, et d'une seule, fonctionnalité, alors que dans la SOA, la pratique est de créer des services qui gèrent un domaine, par exemple un service de gestion des utilisateurs.





Architecture Micro-Services


Domaine d'application



Domaine d'application

Dans la MSA vous aurez plutôt un Microservice de gestion des rôles des utilisateurs , un autre pour l'ajout/suppression/suspension des utilisateurs, etc.

La taille des services dans une MSA est souvent beaucoup plus petite que dans une SOA.





Architecture Micro-Services

Avantages et Bénéfices




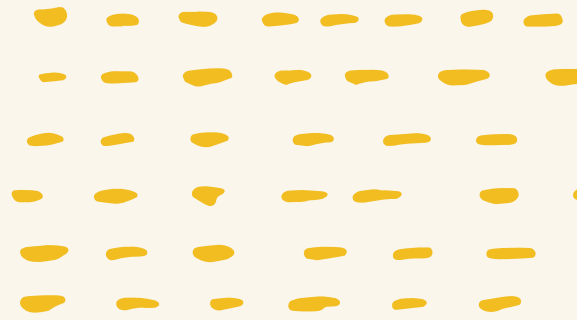
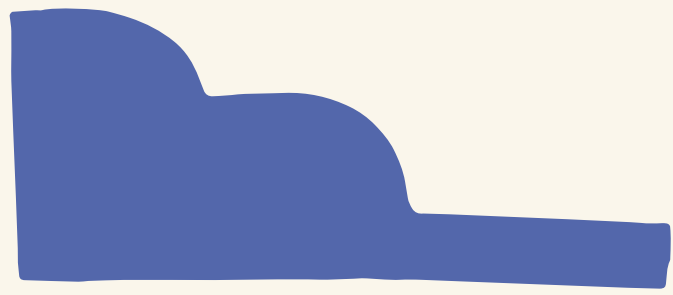
Avantages et Bénéfices



Les microservices stimulent vos équipes et vos routines grâce à un développement distribué. Vous pouvez aussi développer plusieurs microservices simultanément.

Ainsi, davantage de développeurs peuvent travailler en même temps, sur la même application, ce qui réduit la durée du développement.





Architecture Micro-Services

Avantages et Bénéfices

Avantages et Bénéfices

Mise sur le marché plus rapide:

Comme les cycles de développement sont plus courts, l'architecture de microservices permet des déploiements et mises à jour plus agiles.





Architecture Micro-Services


Avantages et Bénéfices



Avantages et Bénéfices

Haute évolutivité:

À mesure que la demande pour certains services augmente, vous pouvez étendre les déploiements sur plusieurs serveurs et infrastructures pour répondre à vos besoins.





Architecture Micro-Services

Avantages et Bénéfices





Avantages et Bénéfices

Résilience:

Lorsqu'ils sont développés correctement, ces services indépendants n'ont aucun impact les uns sur les autres.

Cela signifie que, lorsqu'un élément tombe en panne, l'ensemble de l'application ne cesse pas de fonctionner comme c'est le cas avec le modèle monolithique.





Architecture Micro-Services



Avantages et Bénéfices

Avantages et Bénéfices

Facilité de déploiement:

Les applications basées sur des microservices sont plus modulaires et légères que les applications monolithiques classiques.

Aussi, vous pouvez les déployer plus sereinement.





Architecture Micro-Services


Avantages et Bénéfices



Avantages et Bénéfices

Accessibilité:

Vu que l'application est décomposée en plusieurs éléments, les développeurs peuvent plus facilement comprendre, mettre à jour et améliorer chacun de ces éléments. Résultat : des cycles de développement plus courts, surtout s'ils sont associés à des méthodes de développement agiles.





Architecture Micro-Services


Avantages et Bénéfices



Avantages et Bénéfices

Ouverture:

Grâce aux API qui utilisent plusieurs langages, les développeurs ont la liberté de choisir la technologie et le langage qui conviennent le mieux à chaque fonction.





Architecture Micro-Services

Inconvénients



Inconvénients

Le terme « micro » est relatif:

L'objectif est de décomposer les applications en blocs gérables, mais les blocs doivent également être de taille raisonnable.

Un équilibre parfait peut être difficile à trouver !







Architecture Micro-Services

Inconvénients

Inconvénients

Une architecture fondée sur des bases de données cloisonnées pose des problèmes de mise à jour:

Chaque mise à jour nécessite de faire le tour sur les différents micro-services.





Architecture Micro-Services

Inconvénients




Inconvénients

Des restrictions:

Une architecture distribuée nécessite un déploiement, un dimensionnement et des tests approfondis et ciblés.

Ce caractère distribué, constitue également un défi considérable en matière de test, déploiement et dimensionnement des applications.





Architecture Micro-Services

Inconvénients




Inconvénients

Les changements peuvent être compliqués:

Les dépendances entre les services peuvent être source de problèmes lors de la mise en place de changements.

Les développeurs doivent soigneusement planifier et coordonner le déploiement des changements pour chacun des services.





Architecture Micro-Services


Inconvénients



Inconvénients

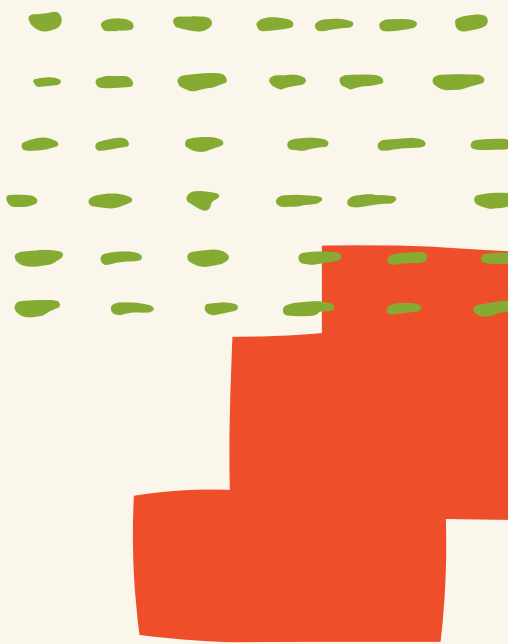
Une architecture couteuse quand il s'agit d'une multitude des micro-services:

C'est une architecture couteuse au niveau de l'infrastructure, les compétences et les technologies à mettre en place.

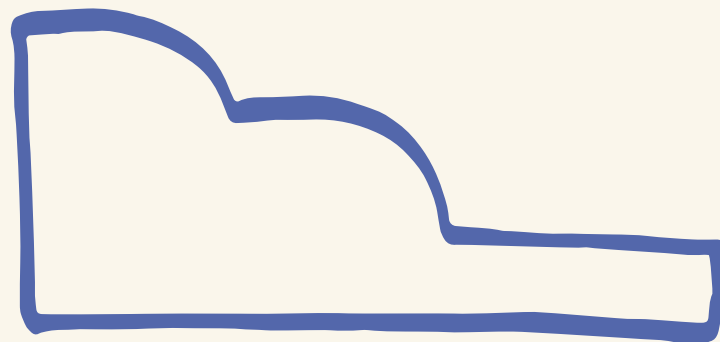




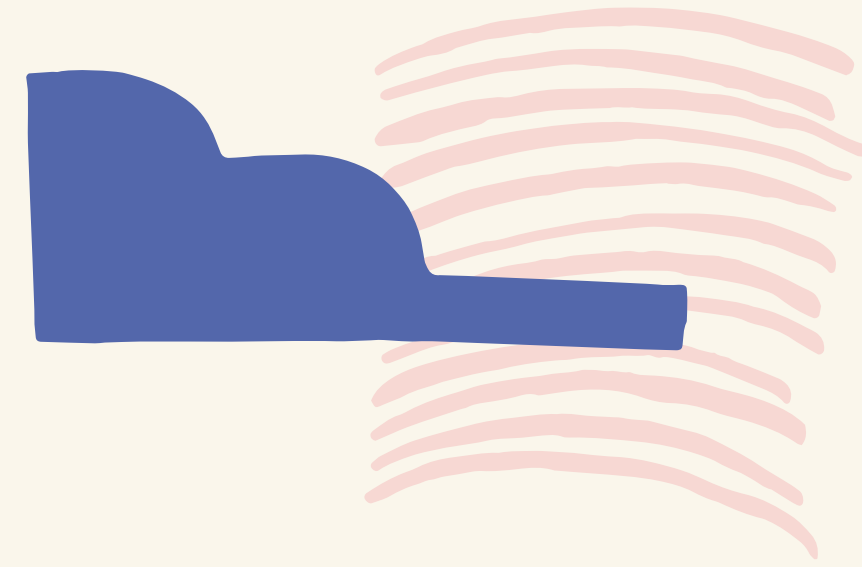
Maintenant reste à modéliser une architecture micro-services !



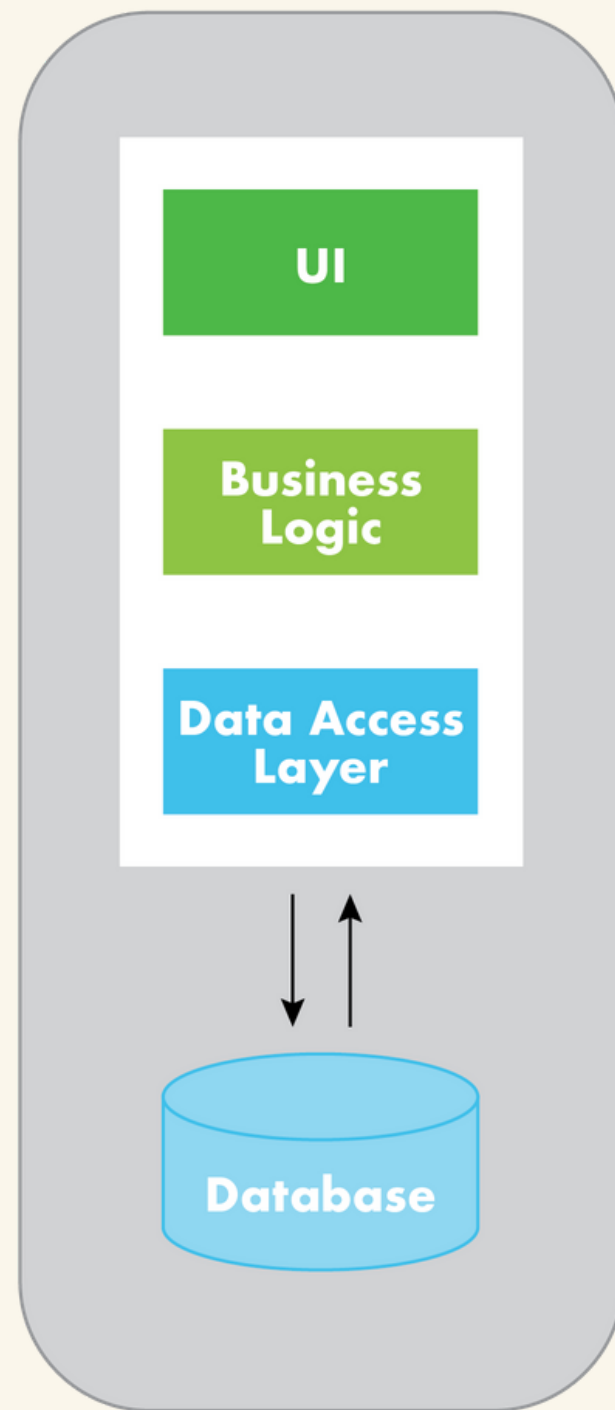
On découvre ceci ensemble ;)



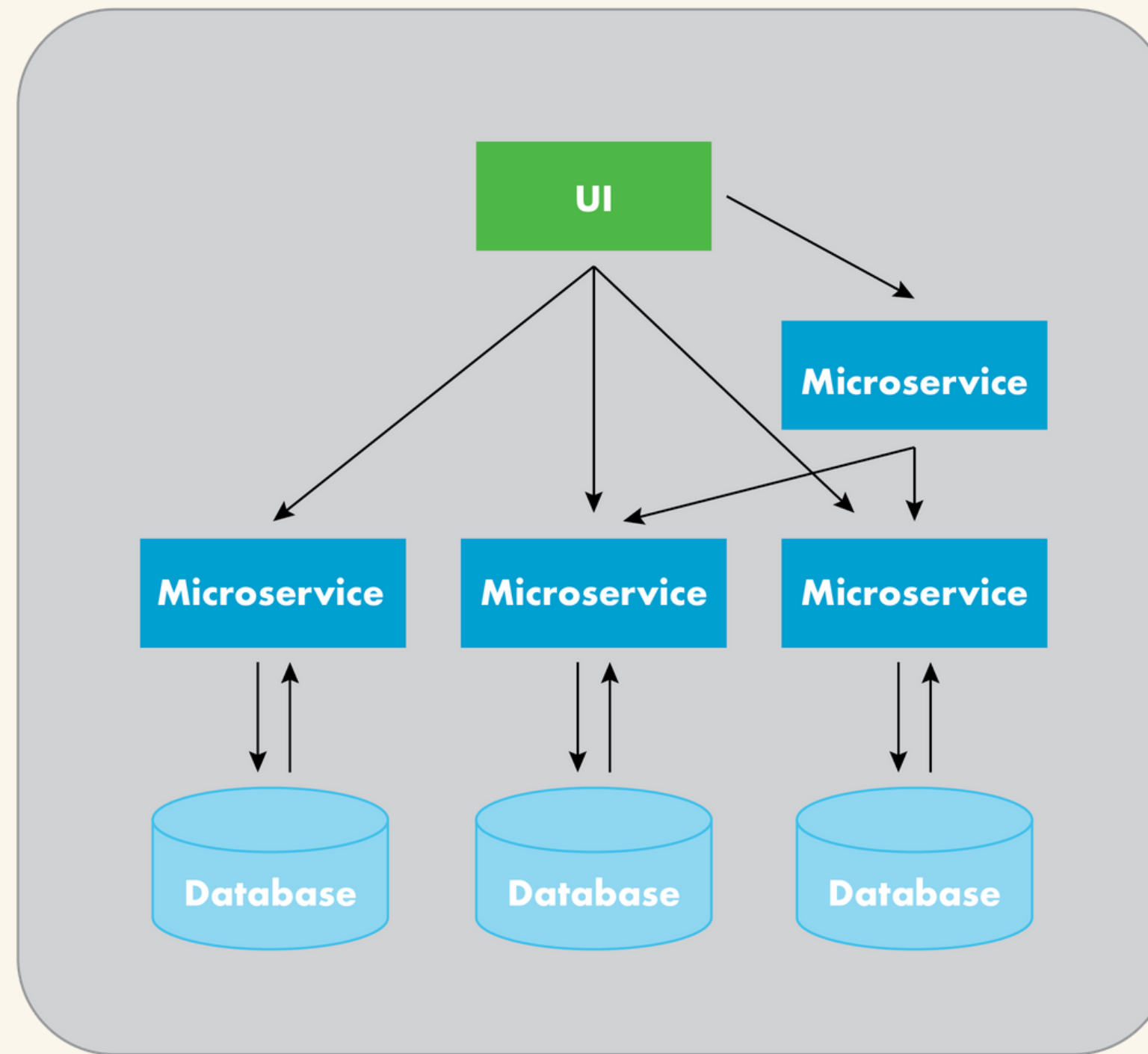
Modélisation d'une MSA



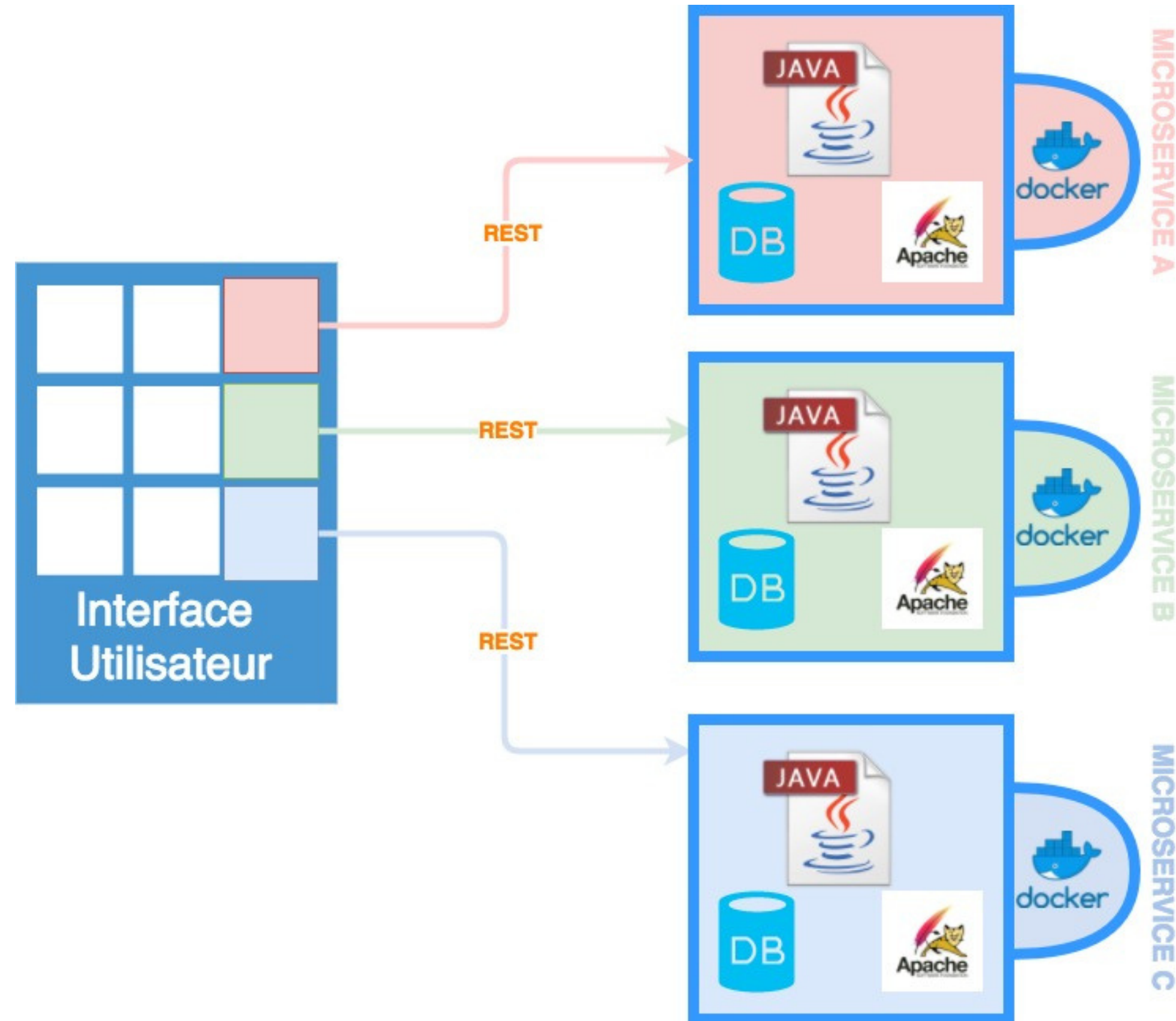
**Monolithic
Architecture**

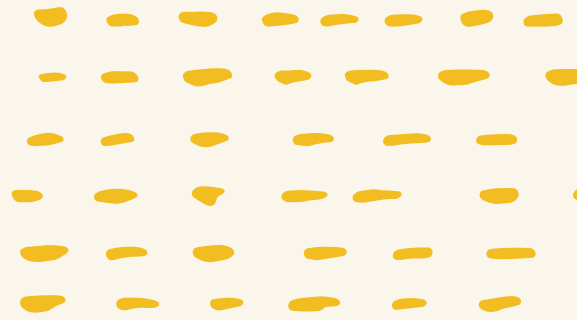
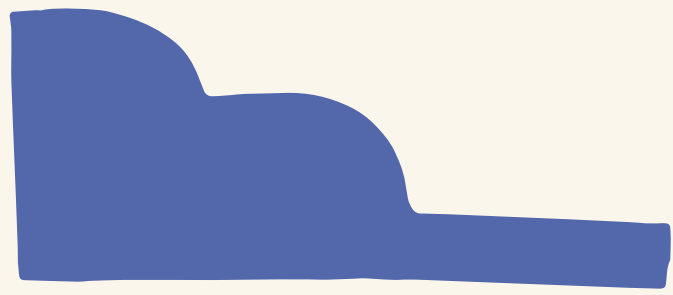


**Microservices
Architecture**



Modélisation d'une MSA





Architecture Micro-Services

Documentation de son micro-service


Documentation de son micro-service

À partir de votre code, Swagger est capable de générer une documentation détaillée au format JSON, répondant aux spécifications OpenAPI.

Il vous offre également la possibilité de visualiser cette documentation dans un format HTML élégant.



Documentation de son micro-service

 **swagger**

Select a spec default

Api Documentation ^{1.0}

[Base URL: localhost:9090/]
<http://localhost:9090/v2/api-docs>

Api Documentation

[Terms of service](#)

[Apache 2.0](#)

basic-error-controller Basic Error Controller >

product-controller Product Controller ▾

GET

/Produits

listeProduits

POST

/Produits

ajouterProduit

PUT

/Produits

updateProduit

GET

/Produits/{id}

afficherUnProduit

DELETE

/Produits/{id}

supprimerProduit

GET

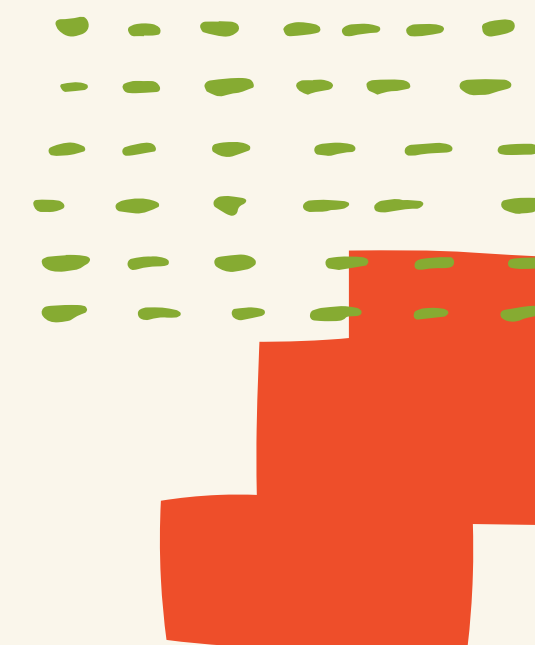
/test/produits/{prix}

testeDeRequetes

Models >



**Nous avons atteint
notre objectif pour
aujourd'hui ;)**



Merci pour votre attention !

