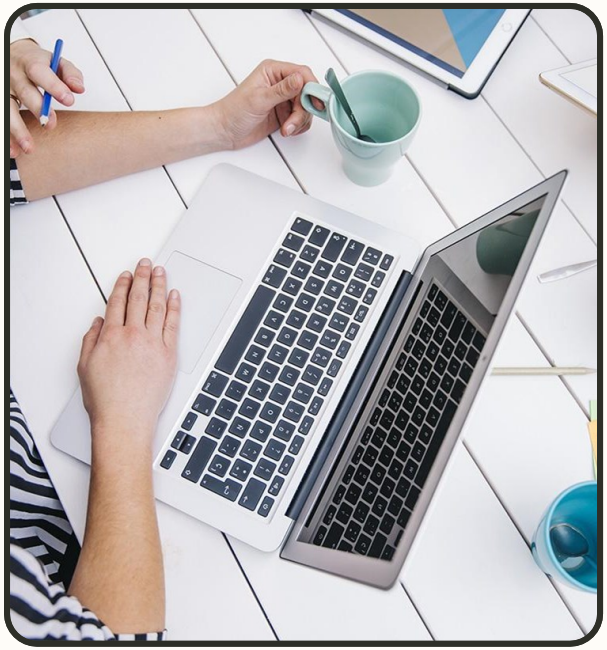
// Architecture Logicielle

ARCHITECTURE LOGICIELLE

Construisez des systèmes logiciels puissants avec une architecture intelligente

Térence FERUT



Styles d'Architecture Logicielle

Explorez les multiples styles d'architecture pour concevoir des logiciels innovants et évolutifs

Plan du cours Styles d'Architecture Logicielle

01 Architecture client-serveur

02 Architecture en couches (Layered Architecture)

03 Model-Vue-Contrôleur et ses Variantes

04 Architecture orientée services (SOA)

Plan du cours Styles d'Architecture Logicielle

Plan du cours Styles d'Architecture Logicielle

09 **Architecture peer-to-peer**

Aperçu du cours

- A travers ce second cours, nous allons explorer les principales familles d'architectures logicielles, ainsi que ce qui les caractérise et fait leurs forces.
- **Vous apprendrez à choisir une architecture adaptée à un besoin grâce à de nombreux exemples historiques**


#03
Modèle-Vue-Contrôleur (MVC) et variantes
15, 2021 - 18H
1 - 15H
JULY 11, 2
AUGUST 8, 2021 -
RIL 15, 2021 - 15H
ting with Company A

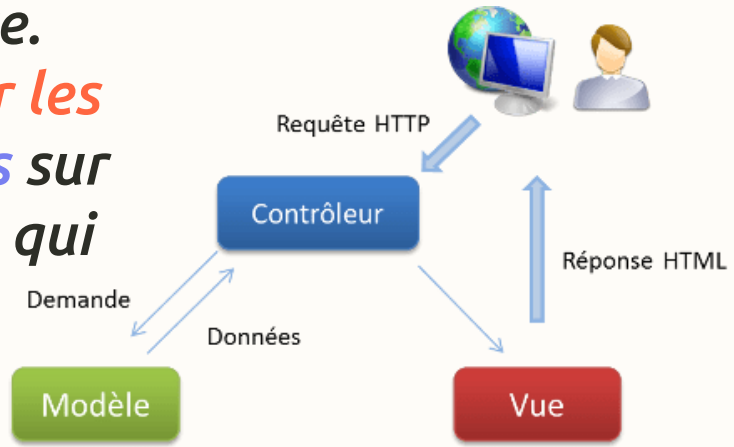
2.3.1 Introduction à l'architecture MVC

- L'architecture MVC, pour Modèle-Vue-Contrôleur, est un modèle d'architecture logicielle couramment utilisé dans le développement de logiciels.
- Il divise une application en trois composants interagissant entre eux :
  - le modèle (la logique de données),
  - la vue (la présentation des données)
  - et le contrôleur (la gestion des interactions).

2.3.1 Introduction à l'architecture MVC

Exemples d'utilisation

- *Pensez à un site web de e-commerce.*
- *Le modèle gère les informations sur les produits, la vue affiche ces produits sur votre écran et le contrôleur gère ce qui se passe lorsque vous ajoutez un produit à votre panier et que vous passez commande.*



2.3.2 Caractéristiques de l'architecture MVC

- L'architecture MVC favorise l'organisation du code, sa réutilisabilité et sa maintenance.
- Chaque composant a une responsabilité spécifique.
- Le modèle gère les données, la vue affiche ces données et le contrôleur gère la logique d'interaction entre le modèle et la vue.

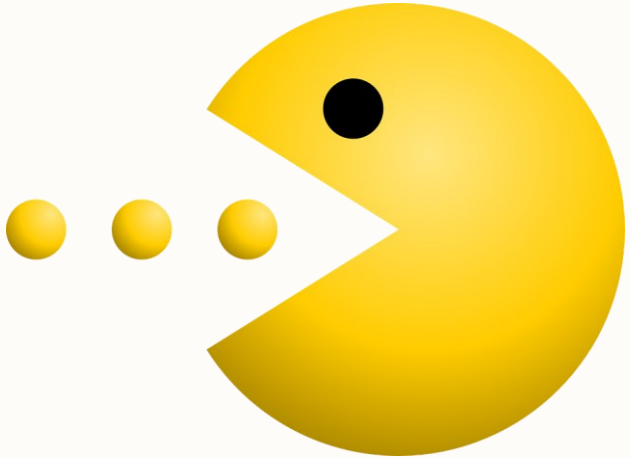
2.3.3 Conception et structure d'une architecture MVC

- La conception d'une architecture MVC implique de diviser le logiciel en trois composants principaux.
  - Le modèle contient les données et la logique liée aux données.
  - La vue affiche les informations à l'utilisateur.
  - Et le contrôleur gère les interactions entre le modèle et la vue.

2.3.3 Conception et structure d'une architecture MVC

Exercice - Réflexion

- *Imaginez que vous développez un jeu vidéo simple, du genre jeu d'arcade comme Pac Man.*
- *Quelles seraient les responsabilités du modèle, de la vue et du contrôleur ?*



2.3.5 Déploiement et évolutivité d'une architecture MVC

- Une application MVC peut être déployée et évoluer facilement.
- Les modifications apportées à un composant n'affectent généralement pas les autres.
- De plus, plusieurs vues peuvent utiliser le même modèle, ce qui favorise le développement d'applications à plusieurs interfaces.

2.3.6 Gestion et maintenance d'une architecture MVC

- La maintenance d'une application MVC est facilitée par sa structure.
- Comme chaque composant a des responsabilités bien définies, il est plus facile de localiser et de corriger les problèmes.

2.3.7 Avantages et inconvénients de l'architecture MVC

- Parmi les avantages de l'architecture MVC, citons la modularité, la réutilisabilité du code et la facilité de maintenance.
- Cependant, elle peut aussi entraîner une complexité accrue et n'est pas toujours la meilleure option pour des applications simples ou de petite taille.

2.3.7 Avantages et inconvénients de l'architecture MVC

Exercice - Réflexion

- *Identifiez une situation où l'utilisation de l'architecture MVC pourrait être contre-productive ?*

2.3.8 Exemples d'utilisation de l'architecture MVC

Exemples d'utilisation

- *L'architecture MVC est utilisée dans de nombreux domaines, notamment dans le développement web.*
- *Par exemple, des frameworks populaires comme Ruby on Rails, Django (Python) et Laravel (PHP) utilisent tous une forme d'architecture MVC.*

2.3.8 Exemples d'utilisation de l'architecture MVC

Exemples d'utilisation

Quelques exemples de boilerplates MVC :

- *En Python/Flask : https://github.com/salimane/flask-mvc*
- *En Python/Bottle : https://github.com/salimane/bottle-mvc*
- *En PHP : https://github.com/mmilanovic4/mvc*
- *En Node.js/Express : https://github.com/oguzhanoya/express-mvc-boilerplate*
- *En Java/Swing : https://github.com/ashiishme/java-swing-mvc*

2.3.9 Introduction aux variantes de l'architecture MVC

- En plus du modèle MVC classique, il existe d'autres variantes, comme le modèle-vue-présentateur (MVP) et le modèle-vue-vue-modèle (MVVM).
- Ces variantes respectent la philosophie MVC mais ajustent les responsabilités et les interactions des trois composants pour s'adapter à différentes situations.

2.3.10 Variante MVP

- L'architecture MVP (**Modèle-Vue-Présentateur**) est une variante de MVC, utilisée pour structurer le code d'une application en séparant les préoccupations logiques.
- Le Modèle s'occupe de la gestion des données et de la logique métier.
- La Vue gère l'interface utilisateur et l'affichage des données.

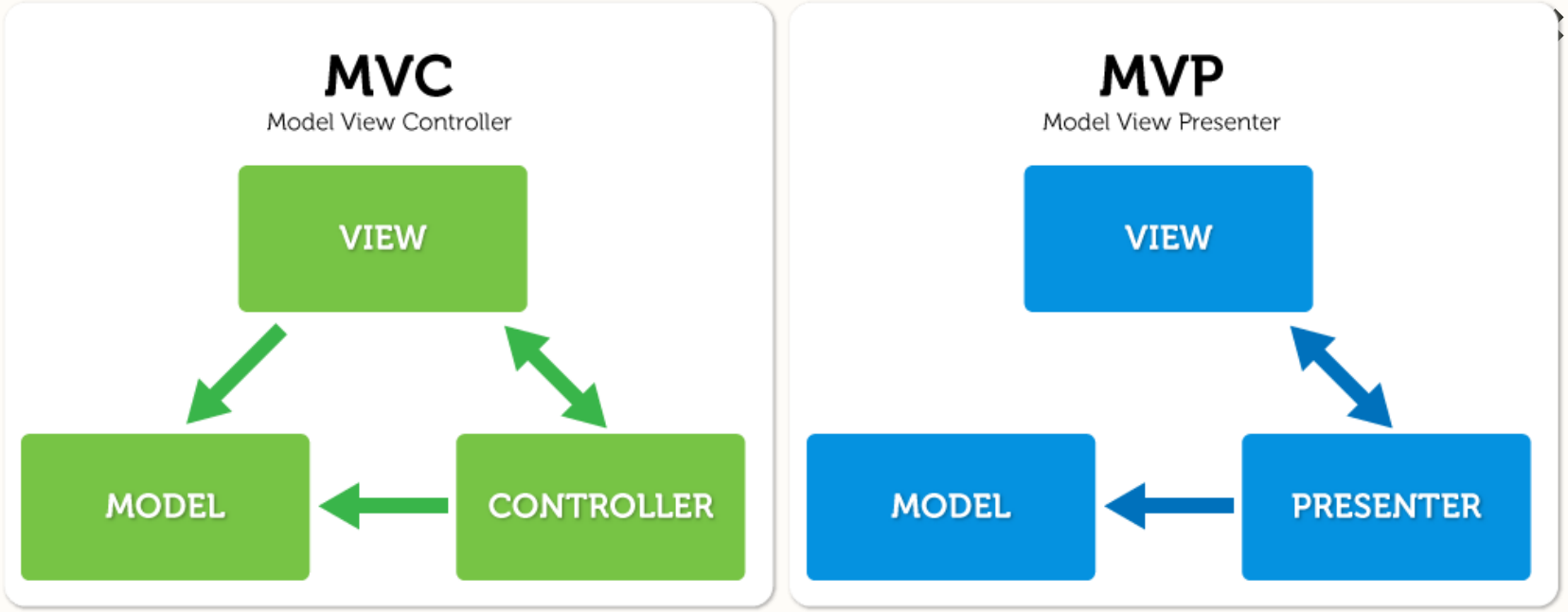
2.3.10 Variante MVP

- Le Présentateur fonctionne comme un intermédiaire, prenant les données du Modèle pour les afficher dans la Vue.
- Contrairement à MVC, où le Contrôleur manipule les données avant qu'elles n'atteignent la Vue, dans MVP, c'est la **Vue qui appelle le Présentateur pour récupérer les données**.

2.3.10 Variante MVP

- La Vue est plus passive, se concentrant uniquement sur l'affichage, tandis que le Présentateur ne se soucie ni de la manière dont les données sont produites (*Modèle*) ni de la manière dont elles sont présentées (*Vue*).
- MVP facilite le test de la logique de présentation et offre une **séparation claire entre la logique d'affichage et la logique métier**.

2.3.10 Variante MVP



2.3.11 Variante MVVM

- L'architecture MVVM (*Modèle-Vue-ViewModel*) est un schéma de conception populaire dans le développement d'applications avec des interfaces utilisateur riches.
- Le Modèle gère les **données et la logique métier**.
- La Vue s'occupe de **l'affichage et de l'interaction** utilisateur.
- Le ViewModel fait le **lien** entre les deux, transformant les données du Modèle en une forme optimale pour la Vue.

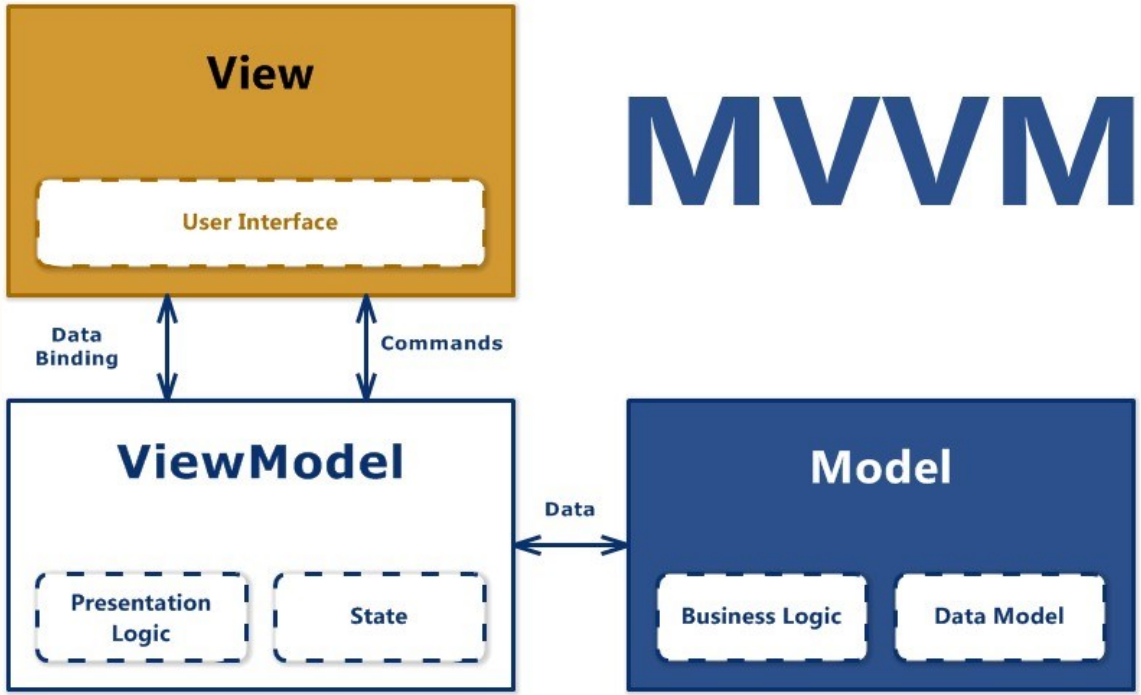
2.3.11 Variante MVVM

- L'avantage réside dans la séparation nette entre la logique de l'interface utilisateur et la logique métier, favorisant ainsi la maintenance et les tests unitaires.
- La liaison de données bidirectionnelle est une caractéristique clé, permettant une communication fluide entre la Vue et le ViewModel.

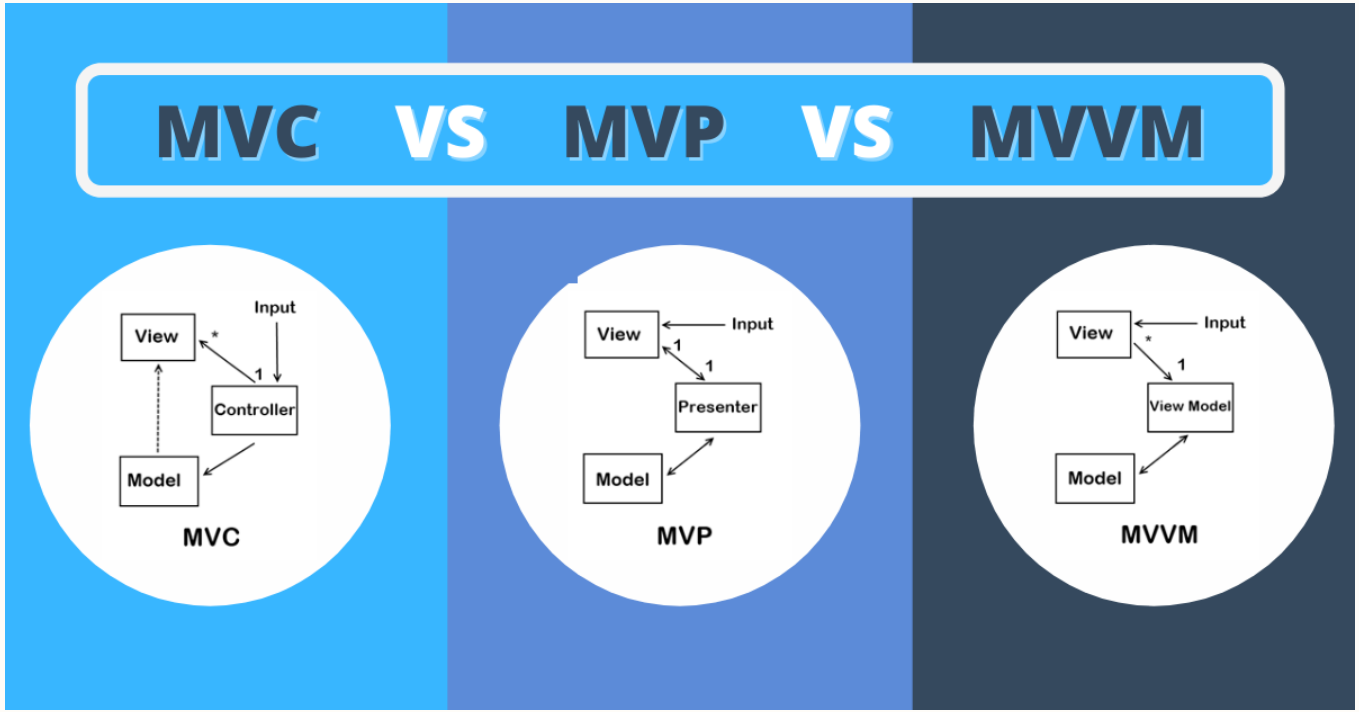
2.3.11 Variante MVVM

- MVVM est particulièrement efficace dans les environnements qui supportent le data-binding, comme les applications utilisant le framework WPF (*Windows Presentation Foundation*) ou des **frameworks front-end** tels qu'Angular ou Vue.js, facilitant ainsi la gestion des états et interactions au sein de l'application.

2.3.11 Variante MVVM



2.3.12 Comparaison des variantes de l'architecture MVC ✖



✖

2.3.12 Comparaison des variantes de l'architecture MVC

Exercice - Réflexion

- *Quelles pourraient être les raisons de choisir l'architecture MVP ou MVVM plutôt que l'architecture MVC traditionnelle ?*

2.3.13 Choisir entre l'architecture MVC et ses variantes

- Chaque variante de l'architecture MVC a ses avantages et inconvénients.
- Le choix dépend des besoins spécifiques du projet.
- *Aucune solution ne peut être la meilleure dans tous les cas*

2.3.13 Choisir entre l'architecture MVC et ses variantes

Choix de l'architecture MVC classique :

- Web traditionnel : MVC est souvent préféré pour les applications web traditionnelles où la **logique serveur** joue un rôle central.
- Simplicité : Optez pour MVC quand vous avez besoin d'une architecture **simple et directe**, particulièrement dans des petits à moyens projets.
- Rapidité du développement : Lorsque **la rapidité de développement** est cruciale, MVC avec son approche établie et sa vaste documentation peut être bénéfique.

2.3.13 Choisir entre l'architecture MVC et ses variantes

Choix de l'architecture MVP:

Testabilité accrue : Si vous avez besoin d'une **forte couverture de tests unitaires**, particulièrement sur la **logique de présentation**, MVP est souvent un meilleur choix.

Séparation stricte : Lorsqu'une **séparation claire entre la logique de présentation et l'UI** est essentielle, pour des raisons de testabilité ou de clarté dans le code.

Technologies sans data-binding : Dans les environnements technologiques qui ne supportent pas le *data-binding bidirectionnel*, MVP pourrait être plus approprié.

2.3.13 Choisir entre l'architecture MVC et ses variantes

Choix de l'architecture MVVM:

Data-binding : MVVM est particulièrement efficace dans les technologies qui supportent le **data-binding bidirectionnel**, comme WPF ou Angular.

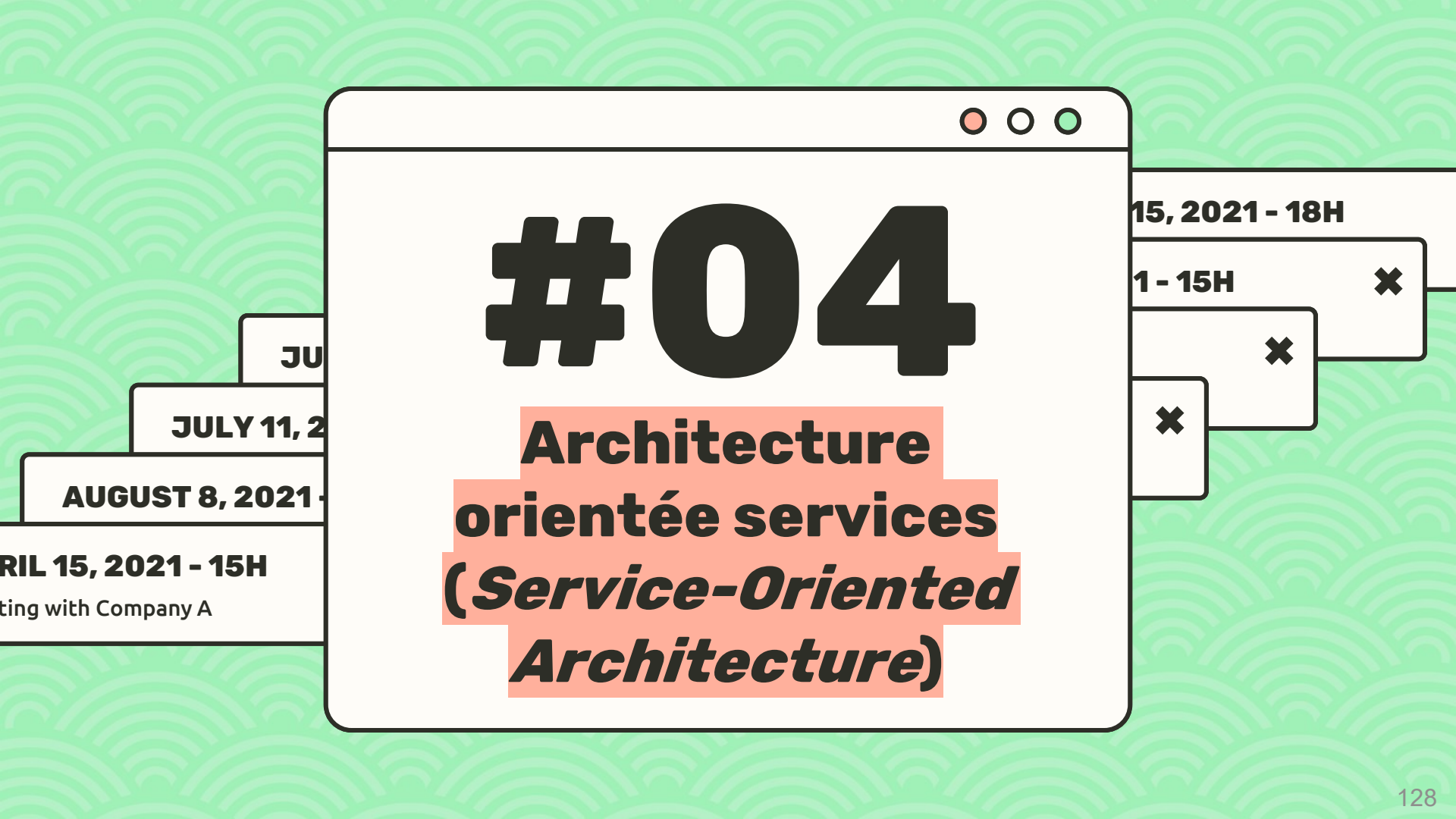
Applications riches : Pour les **applications riches client lourd** où une séparation nette des préoccupations et une **interaction UI dynamique** sont nécessaires.

Évolutions futures : Si votre application pourrait s'étendre ou se compliquer davantage, MVVM permet une **évolutivité plus aisée** avec une meilleure structuration.

2.3.13 Choisir entre l'architecture MVC et ses variantes

Cas spécifiques

- *Applications mobiles : MVVM est souvent favorisé dans le développement d'applications mobiles avec **Xamarin** ou en utilisant des frameworks comme **ReactiveUI**.*
- *Applications web modernes (SPA) : MVVM peut également être un choix judicieux pour les applications web monopage (SPA) utilisant des frameworks comme **Angular** ou **Vue.js**, qui tirent parti du data-binding bidirectionnel.*
- *Applications d'entreprise : MVP pourrait être privilégié pour des applications d'entreprise où la **testabilité** et la **maintenance** sont primordiales.*


#04
Architecture orientée services (Service-Oriented Architecture)
15, 2021 - 18H
1 - 15H
JULY 11, 2
AUGUST 8, 2021 -
RIL 15, 2021 - 15H
ting with Company A

2.4.1 Introduction à l'architecture orientée services

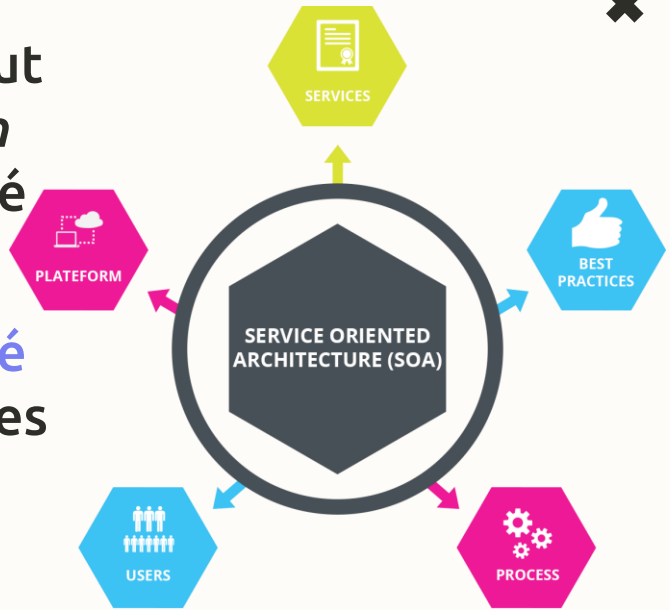
- L'architecture orientée services (*SOA*) est un style d'architecture logicielle qui favorise des systèmes modulaires et flexibles basés sur des **services**.
- Elle émerge dans les années 1990 et gagne en popularité depuis.
- SOA repose sur des principes tels que la modularité, le découplage, la réutilisabilité et l'interopérabilité, permettant aux organisations de créer des systèmes agiles capables de s'adapter rapidement.

2.4.1 Introduction à l'architecture orientée services

- Dans l'architecture orientée services, les services représentent des fonctionnalités spécifiques offertes par des composants logiciels.
- Ils interagissent en utilisant des protocoles standardisés.

2.4.1 Introduction à l'architecture orientée services

- Par exemple, un *service de paiement* peut communiquer avec un *service de gestion des comptes* pour vérifier la disponibilité des fonds avant une transaction.
- Cette approche favorise la réutilisabilité et la modularité, permettant aux services d'être développés, testés et déployés indépendamment.



2.4.1 Introduction à l'architecture orientée services ✖

Exercice - Réflexion

- *Identifiez trois fonctionnalités clés d'une application de réservation de voyages et imaginez comment elles pourraient être développées en tant que services autonomes.*

2.4.2 Caractéristiques clés de l'archi. orientée services

- L'architecture orientée services se caractérise par le découplage, la modularité, la réutilisabilité et l'interopérabilité.
- Le découplage permet à chaque service de fonctionner de manière indépendante, facilitant les modifications ou les remplacements.

2.4.2 Caractéristiques clés de l'archi. orientée services

- La modularité divise les fonctionnalités en services autonomes, simplifiant ainsi le développement, le test et le déploiement.
- La réutilisabilité permet l'utilisation de services dans différentes applications, réduisant les efforts de développement.

2.4.2 Caractéristiques clés de l'archi. orientée services

- L'interopérabilité, enfin, assure la communication entre services, indépendamment des langages ou des plates-formes utilisés.

2.4.2 Caractéristiques clés de l'archi. orientée services

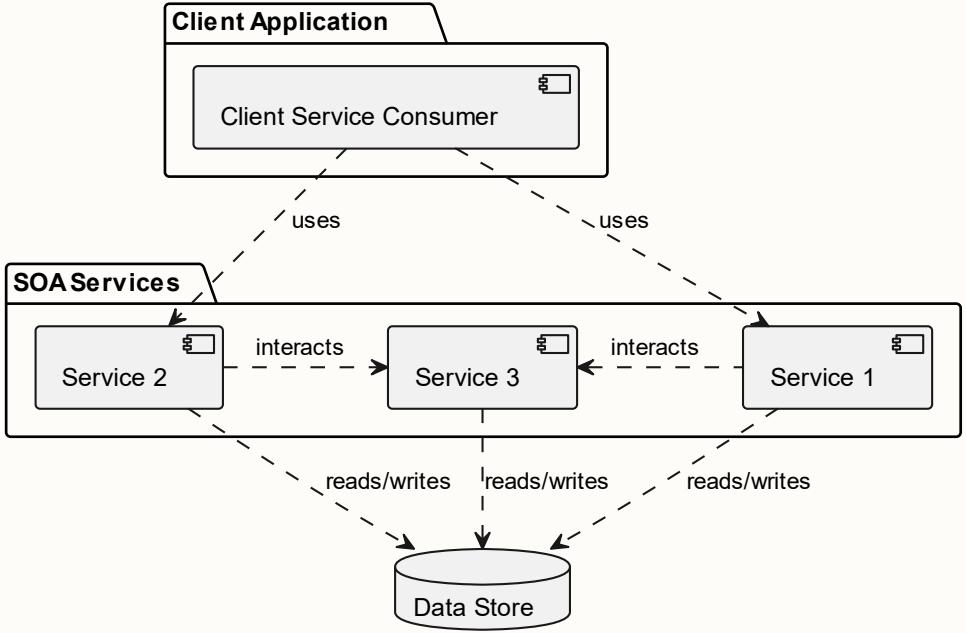


Diagramme de composants représentant une SOA en UML

2.4.2 Caractéristiques clés de l'archi. orientée services

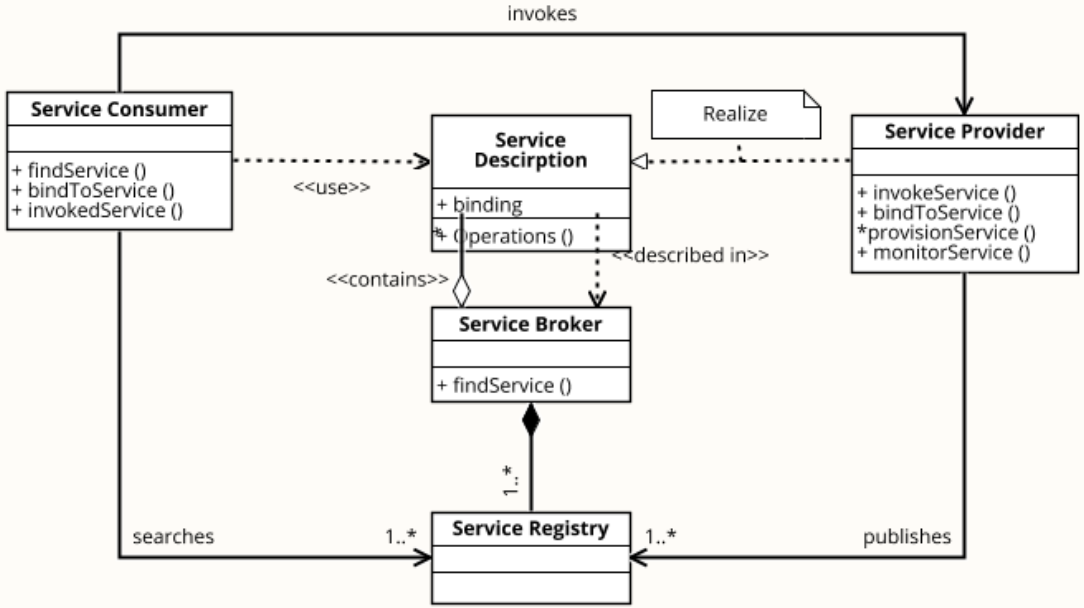


Diagramme de classes représentant une SOA en UML

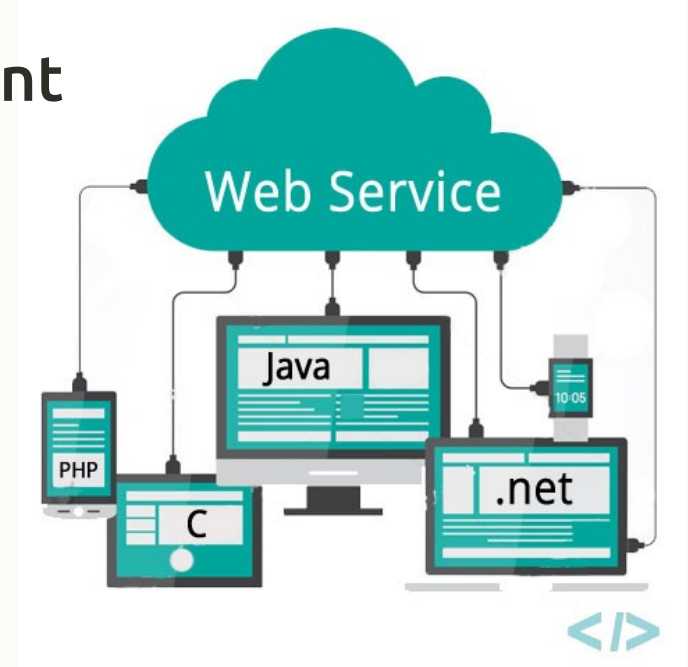
2.4.2 Caractéristiques clés de l'archi. orientée services

Exercice - Réflexion

- *Identifiez une autre caractéristique de l'architecture orientée services et expliquez comment elle peut bénéficier au développement de logiciels basés sur une SOA*

2.4.3 Services web et protocoles

- Les services web sont couramment utilisés pour mettre en œuvre l'architecture orientée services.
- Ils permettent aux services de communiquer via Internet en utilisant des protocoles standardisés.



2.4.3 Services web et protocoles

- Les services web utilisent généralement le protocole HTTP pour la communication et le langage XML pour représenter les données échangées.
- Cette approche facilite la communication transparente entre les services développés dans différents langages et exécutés sur différentes plates-formes.

2.4.3 Services web et protocoles

- Deux méthodes couramment utilisées pour les services web sont SOAP (*Simple Object Access Protocol*) et REST (*REpresentational State Transfer*).
- SOAP est un protocole basé sur XML qui offre un formalisme strict et complexe.

2.4.3 Services web et protocoles

- REST, quant à lui, repose sur les principes du web et utilise les méthodes HTTP (*GET*, *POST*, *PUT*, *DELETE*) pour interagir avec les services.
- REST est plus simple, léger et largement adopté pour les services web.

2.4.3 Services web et protocoles

Exemple : création d'un service web RESTful

- *Supposons que vous deviez créer un service web RESTful pour récupérer des données d'un **système de gestion des clients**.*
- *Vous pouvez définir une API REST avec une **URI** spécifiant l'adresse du service et les ressources à récupérer.*
- *Par exemple, /clients pour récupérer tous les clients ou /clients/{id} pour récupérer un client spécifique en fonction de son identifiant.*
- *Les méthodes HTTP sont utilisées pour interagir avec le service, offrant une interface simple et intuitive pour récupérer les données du système de gestion des clients.*

2.4.3 Services web et protocoles

Exercice - Réflexion

- *Identifiez plusieurs différences entre les approches SOAP et REST.*

2.4.4 Composition de services

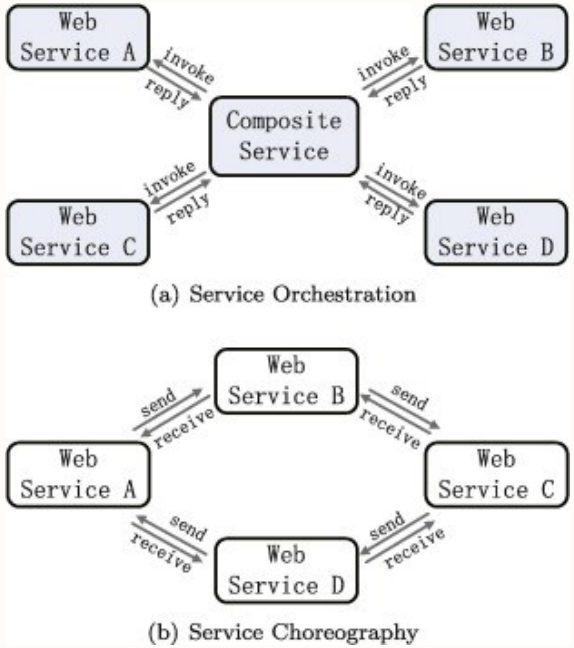
- La composition de services consiste à combiner plusieurs services existants pour créer des **fonctionnalités complexes**.
- Au lieu de développer une fonctionnalité complexe à partir de zéro, la **composition de services** permet de tirer parti des services existants et de les orchestrer pour répondre à un besoin spécifique.
- *Par exemple, pour créer un processus de réservation de billets en ligne, vous pouvez combiner des services tels que la recherche de vols, la réservation d'hôtels et le paiement.*

2.4.4 Composition de services

- Il existe deux approches principales pour la composition de services : l'orchestration et la chorégraphie.
- Dans l'orchestration, un service principal (**l'orchestrateur**) contrôle et coordonne l'exécution des autres services.
- **L'orchestrateur** définit un flux d'exécution et appelle séquentiellement les services nécessaires pour accomplir une tâche.

2.4.4 Composition de services

- Dans la chorégraphie, chaque service connaît ses propres responsabilités et interagit directement avec les autres services pour accomplir une tâche.
- Les services communiquent entre eux en utilisant des messages échangés selon un **protocole** défini.



2.4.4 Composition de services

Exemple : Composition de services pour un système de réservation

- *Imaginons que vous souhaitiez créer un flux de processus de réservation de billets en ligne.*
- *Vous pouvez utiliser l'orchestration pour contrôler les étapes du processus.*
- *L'orchestrateur appelle successivement les services de recherche de vols pour trouver des options, de réservation d'hôtels pour réserver un hébergement et de paiement pour effectuer la transaction.*

2.4.4 Composition de services

Exemple : Composition de services pour un système de réservation

- *L'orchestrateur gère les erreurs éventuelles et garantit le bon déroulement du processus.*
- *En composant ces services existants, vous créez un système complet de réservation de billets en ligne.*

2.4.4 Composition de services

Exercice - Réflexion

- *Identifiez une différence majeure entre l'orchestration et la chorégraphie dans la composition de services et expliquez comment chaque approche peut être utilisée pour créer un flux de processus de réservation de billets en ligne.*

2.4.5 Gestion des services

- La gestion des services dans une architecture orientée services englobe plusieurs aspects importants.
- La découverte des services permet aux développeurs de trouver rapidement les services nécessaires à leurs applications sans avoir à connaître tous les détails de chaque service existant.
- Les services peuvent être découverts à l'aide de registres, de moteurs de recherche ou d'annuaires de services.

2.4.5 Gestion des services

- La publication des services consiste à les rendre accessibles à d'autres applications en exposant leurs **interfaces** et leurs **fonctionnalités**.
- Un annuaire de services agit comme une base de données centralisée où les services peuvent être enregistrés et recherchés.
- La surveillance des services implique de suivre leur disponibilité, leurs performances et de détecter d'éventuelles erreurs.

2.4.5 Gestion des services

Exemple : Utilisation d'un annuaire de services

- *Supposons que vous utilisiez un annuaire de services pour rechercher et accéder à des services disponibles.*
- *L'annuaire contient des informations sur les différents services, tels que leurs descriptions, leurs interfaces et leurs emplacements.*
- *Vous pouvez utiliser l'annuaire pour trouver un service spécifique, obtenir les détails nécessaires et établir une connexion pour l'utiliser dans votre application.*

2.4.5 Gestion des services

Exemple : Utilisation d'un annuaire de services

- *L'annuaire facilite la découverte et l'intégration des services, simplifiant ainsi le processus de développement et d'utilisation des services dans une architecture orientée services.*

2.4.6 Sécurité dans une SOA

- La sécurité est un aspect essentiel de l'architecture orientée services.
- Les services doivent être protégés contre les accès non autorisés et les attaques potentielles.
- Les protocoles de sécurité tels que SSL/TLS (*HTTPS*) sont utilisés pour sécuriser les échanges de données entre les services.
- Les considérations de sécurité incluent :
  - l'authentification des utilisateurs et des services,
  - l'autorisation pour contrôler les accès,
  - la confidentialité des données échangées
  - et l'intégrité des messages.

2.4.6 Sécurité dans une SOA

1. L'authentification garantit que seules les entités légitimes peuvent accéder aux services en vérifiant leur identité.
2. L'autorisation contrôle les actions autorisées pour chaque entité après l'authentification.
3. La confidentialité assure que les données échangées entre les services sont protégées contre les regards indiscrets.
4. L'intégrité garantit que les messages ne sont pas modifiés lors de leur transmission, en utilisant des mécanismes de vérification de l'intégrité des données.

2.4.6 Sécurité dans une SOA

Exemple : sécurisation des échanges de données

- *Prenons l'exemple de la sécurisation des échanges de données entre les services à l'aide de protocoles de sécurité tels que SSL/TLS.*
- *Ces protocoles permettent d'établir des canaux de communication chiffrés entre les services, assurant ainsi la confidentialité des données.*

2.4.6 Sécurité dans une SOA

Exemple : sécurisation des échanges de données

- *Ils utilisent des certificats pour vérifier l'identité des services et garantir l'authenticité.*
- *De plus, des mécanismes de vérification d'intégrité, tels que les fonctions de hachage, sont utilisés pour s'assurer que les données n'ont pas été altérées lors de leur transit.*

SSL/TLS

2.4.7 Avantages et défis de la SOA

- L'architecture orientée services offre de nombreux avantages, notamment la réutilisabilité des services qui permet de développer plus rapidement en tirant parti des services existants.
- Elle favorise également la flexibilité en permettant aux services d'être modifiés ou remplacés indépendamment, sans affecter l'ensemble du système.
- De plus, l'interopérabilité facilite l'intégration entre différentes applications et plates-formes.

2.4.7 Avantages et défis de la SOA

- Malgré ses avantages, l'architecture orientée services présente également des défis potentiels.
- La complexité peut augmenter en raison de **l'orchestration** ou de la **composition** de plusieurs services.
- La gestion des services, la coordination des flux de données et la gestion des erreurs peuvent également être des défis.
- De plus, la performance peut être affectée par la communication entre les services et le traitement des messages.

2.4.7 Avantages et défis de la SOA

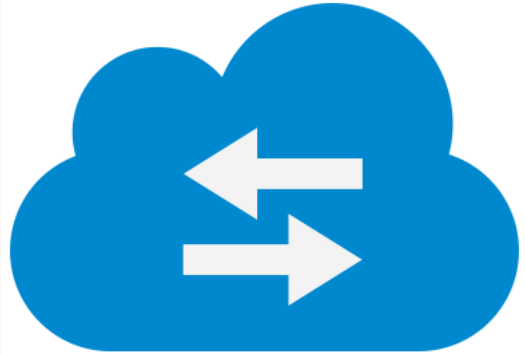
Exemple : Utilisation d'une SOA pour faciliter l'intégration de services

- *Un exemple concret de l'utilisation de l'architecture orientée services est la facilitation de l'intégration entre différentes applications d'entreprise.*
- *En adoptant une approche orientée services, les entreprises peuvent exposer des services spécifiques qui peuvent être utilisés par d'autres applications internes ou externes.*

2.4.7 Avantages et défis de la SOA

Exemple : Utilisation d'une SOA pour faciliter l'intégration de services

- *Par exemple, un service de gestion des clients peut être partagé entre un système de facturation, un système de support client et une application mobile.*
- *Cela permet une meilleure collaboration et une meilleure évolutivité des systèmes d'entreprise.*



2.4.7 Avantages et défis de la SOA

Exemple : Utilisation d'une SOA

- *Exemple d'une architecture SOA sur AWS : https://github.com/jcolemorrison/foundational-soa*

2.4.7 Avantages et défis de la SOA

Exercice - Réflexion

- *Identifiez d'autres avantages de l'architecture orientée services et expliquez comment ils peuvent bénéficier au développement de systèmes complexes.*

Travaux Pratiques #2

Vous pouvez faire le nouveau TP de ce cours qui porte particulièrement sur les styles architecturaux que nous venons de voir.

Merci pour votre attention !

Avez-vous des questions ?

contact@astroware-conception.com

www.astroware-conception.com

www.linkedin.com/in/terence-ferut/