Introduction BackEnd

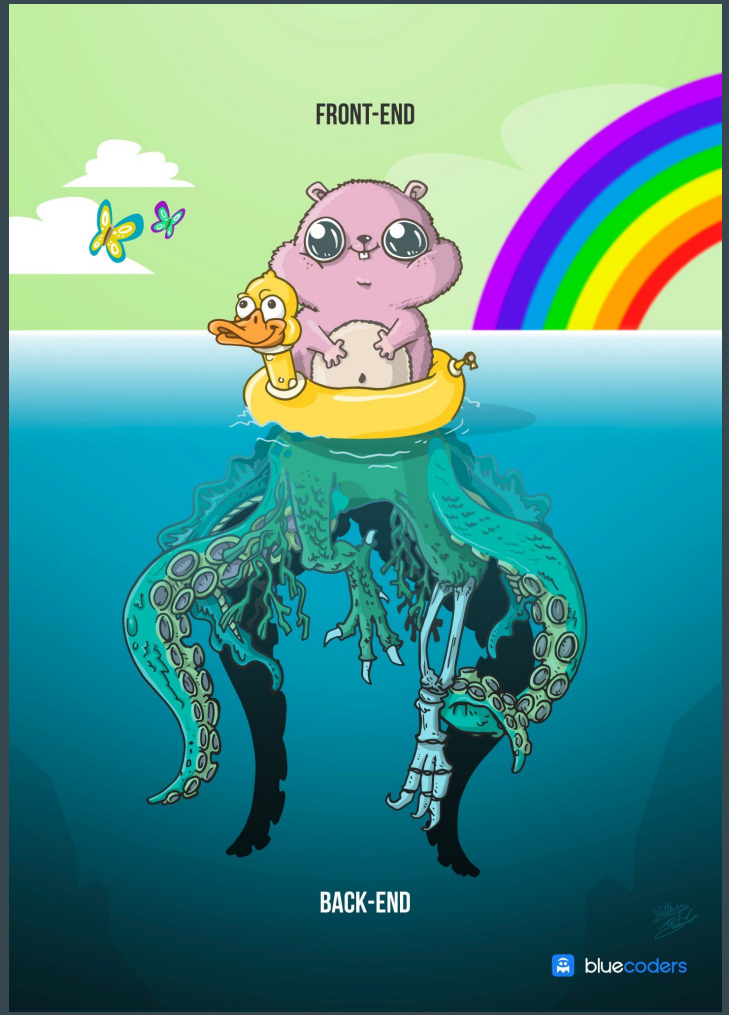
Mathieu Nibas
B3 - ECV 2023

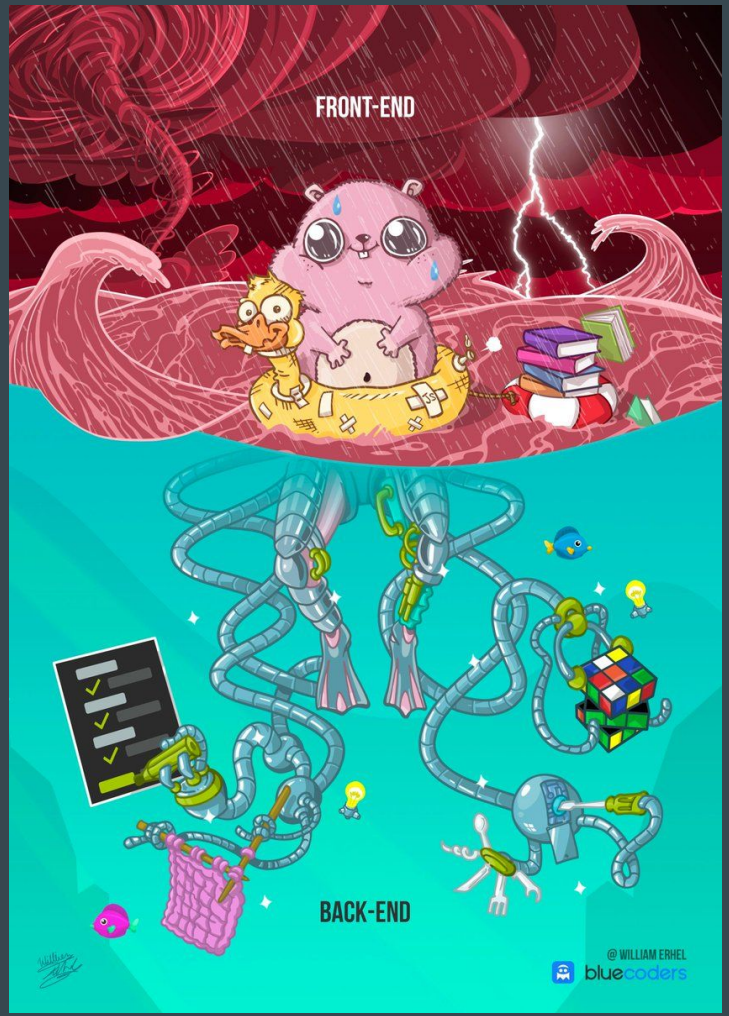
Présentation
-
tour de tables

Sommaire

- Frontend VS Backend
- DevOps
- Agilité VS Cycle en V
- Généralités Techniques

Frontend vs BackEnd

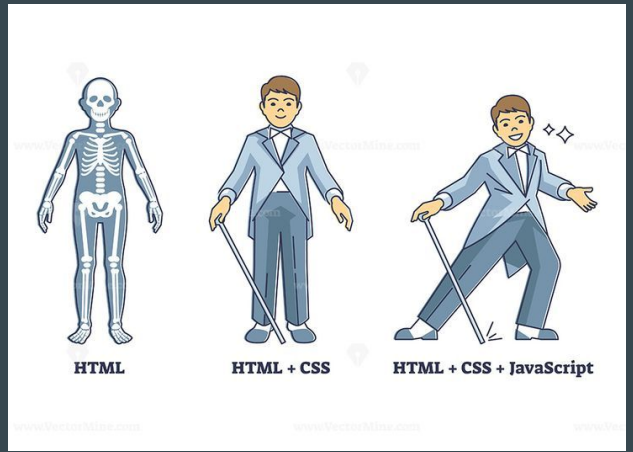

FRONT-END
BACK-END
bluecoders


FRONT-END
BACK-END
@ WILLIAM ERHEL
bluecoders

Le Frontend

Aussi appelé développement client
Html / CSS / Javascript
Création d'un site web ou d'une application

- ➔ L'interface utilisateur
- ➔ La partie visible et accessible par l'utilisateur.
  - ◆ Conception de l'interface utilisateur,
  - La mise en page
  - La présentation de l'information,
  - La navigation
  - Les intéractions avec le site ou l'application.



La partie qui est la plus visible et la plus interactive pour les utilisateurs.

Le Backend

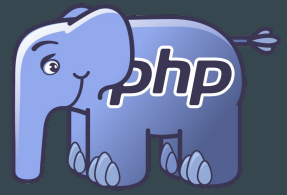
Aussi appelé **développement serveur**

Concerne les aspects techniques et invisibles pour l'utilisateur final.

- ➔ La gestion de la base de données
- ➔ L'intégration des différentes parties du site ou de l'application
- ➔ Mise en place des fonctionnalités avancées
- ➔ La sécurité
- ➔ ...

Le Backend - Langages

- PHP
- Ruby
- Python
- Java
- .NET
- Node.js
- Golang
- Elixir, ...



Le choix du langage de programmation **dépend de la nature du projet** et “des préférences du développeur”

Le Backend - Framework

Une structure de base prédéfinie qui fournit

- ➔ Un ensemble d'outils
- ➔ De bibliothèques
- ➔ De conventions pour faciliter le développement
- ➔ Conçus pour offrir une base commune
- ➔ Une architecture de base pour les applications
- ➔ Améliorer la qualité et la maintenance du code

Le Backend - Framework

Quelques exemples de fonctionnalités courantes

- ➔ Gestion des routes et de la navigation
- ➔ Gestion de l'interface utilisateur
- ➔ Gestion de la persistance des données
- ➔ Gestion des dépendances et des modules

Il existe de nombreux frameworks différents pour différents langages de programmation et pour différents types d'applications (applications web, applications mobiles, applications de bureau, ...)
Chacun d'entre eux a ses propres caractéristiques et ses propres avantages, et **il est important de choisir celui qui convient le mieux aux besoins de votre projet**.

Le Backend - Framework

- **JavaScript**
  - Angular (frontend)
  - React (frontend)
  - Vue.js (frontend)
  - Svelte (frontend)
  - NestJs

- **Python**
  - Django
  - Flask
  - Pyramid

- **PHP**
  - Laravel
  - Symfony
  - CodeIgniter

- **Ruby**
  - Ruby on Rails
  - Sinatra

- **Java**
  - Spring
  - JavaEE (Enterprise Edition)
  - Play

- **C#**
  - ASP.NET
  - NancyFX
  - DotNetty

Le Backend - API

Accronyme de "**Application Programming Interface**" (Interface de programmation d'application)

Fourni un ensemble de fonctionnalités et de conventions qui permettent à des applications de communiquer entre eux.

Définit comment les applications peuvent accéder à des données ou à des fonctionnalités d'un autre logiciel ou application.

On utilise souvent le format **Json** pour générer le payload de la requête. (existe aussi des sorties xml, yaml, texte ou encore binaire)

Le Backend - API

Quelques exemples d'utilisations courantes des APIs :

- ➔ Accéder à des données ou à des fonctionnalités d'un service en ligne
- ➔ Intégrer des fonctionnalités d'un autre logiciel dans votre application
- ➔ Connecter des applications ou des services entre eux

Elles peuvent être développées par les éditeurs de logiciels ou de services en ligne et publiées pour que d'autres développeurs puissent les utiliser dans leurs propres applications.

Le Backend - Exemple sortie API (Json)

Le Backend - SDK

Acronyme de "**Software Development Kit**"

Ensemble d'outils, de bibliothèques et de documentations qui permet de créer des applications ou des intégrations pour un logiciel ou un service en ligne spécifique.

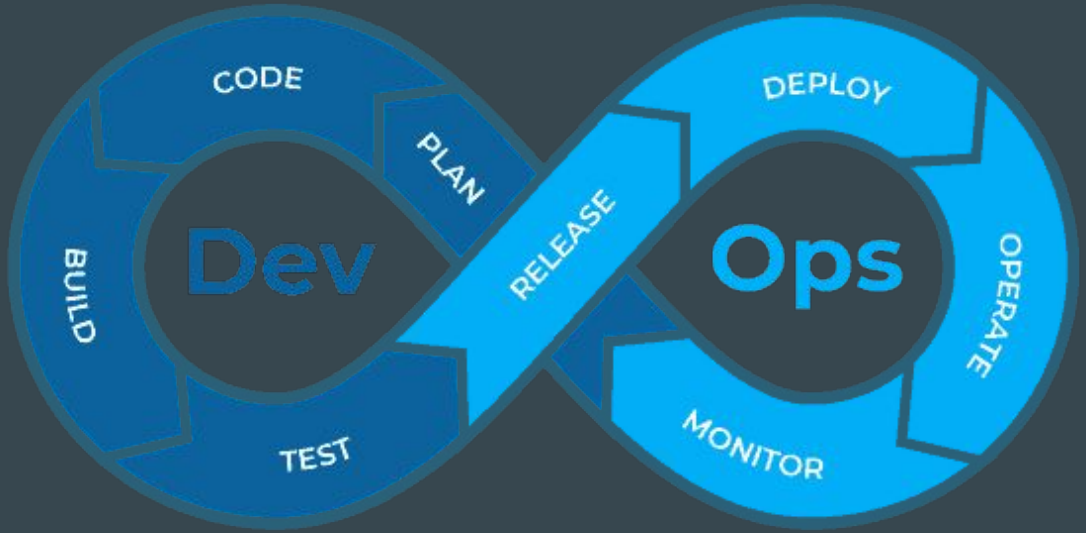
Un SDK inclut généralement des exemples de code, des bibliothèques et des outils pour faciliter le développement d'applications ou d'intégrations pour un logiciel ou un service en ligne donné.

Le Backend - SDK

Voici quelques exemples d'utilisations courantes des SDK :

- ➔ Développer des applications pour un système d'exploitation mobile
- ➔ Développer des intégrations pour un service en ligne
- ➔ Accéder à des données ou à des fonctionnalités d'un service en ligne

DevOps


CODE
PLAN
BUILD
Dev
TEST
RELEASE
DEPLOY
OPERATE
Ops
MONITOR

DevOps

Contraction de "development" et "operations"

Une approche de gestion de projet qui vise à favoriser la collaboration et l'intégration entre les équipes de développement et d'exploitation dans le but d'améliorer la qualité du logiciel, la rapidité de déploiement et la fiabilité des applications.

Inclut également des pratiques de gestion de projet telles que la gestion de configuration, la gestion de la chaîne de livraison et la gestion de la qualité.

DevOps

Quelques éléments clés de DevOps :

- ➔ Collaboration
- ➔ Automatisation
- ➔ Mesure des performances
- ➔ Gestion de la chaîne de livraison
- ➔ Gestion de la qualité

En adoptant une approche DevOps, les équipes de développement et d'exploitation peuvent travailler de manière **plus collaborative** et **efficace** pour **créer et déployer** des applications de manière plus rapide et plus fiable.

DevOps - Docker

Un logiciel de conteneurisation qui permet de **créer, déployer** et **exécuter des applications** dans des conteneurs logiciels.

Un conteneur est une unité de logiciel qui regroupe

- ➔ Le code d'une application
- ➔ Ses dépendances
- ➔ Toutes les ressources nécessaires à son exécution, de manière à pouvoir être exécuté de manière autonome sur n'importe quel serveur.

Similaires aux machines virtuelles, mais plus légers et plus rapides à démarrer (ne nécessitent pas de système d'exploitation séparé)

DevOps - Docker

Permet de créer des applications plus portables et faciles à déployer sur différents environnements (dev, preprod, production,...)

Docker offre également

- ➔ Une plateforme de gestion de conteneurs qui permet de gérer et de déployer
- ➔ Des outils de gestion de conteneurs,
- ➔ De gestion de réseaux et de stockage

DevOps - CI

Aka l'**intégration continue** ("**Continuous Integration**")

Pratique de gestion de projet qui consiste à intégrer **régulièrement** les modifications apportées au code source d'un projet dans une branche principale.

L'intégration continue vise à s'assurer que le code source du projet est **stable et fonctionnel en tout temps** en intégrant **fréquemment** de petites modifications plutôt que de grandes modifications moins fréquemment.

DevOps - CI

Comment fonctionne généralement l'intégration continue ?

1. Les développeurs soumettent leurs modifications
2. L'outil d'intégration continue exécute des tests automatisés
3. Le code est intégré dans la branche principale
4. Le code est déployé

DevOps - CD

Aka **déploiement continu** ("Continuous Deployment")

Pratique de gestion de projet qui consiste à **déployer automatiquement** les modifications apportées au code source d'un projet dans un environnement de production ou de test chaque fois que **le code est prêt à être déployé.**

Le déploiement continu vise à **accélérer** le processus de déploiement et à **réduire les erreurs** en automatisant le processus de déploiement.

DevOps - CD

Comment fonctionne généralement le déploiement continu ?

Les développeurs soumettent régulièrement leurs modifications au code source du projet à l'aide d'un outil de gestion de version de code, comme Git.

1. L'outil de déploiement continu exécute des tests automatisés
2. Le code est déployé

Cependant, il peut être difficile à mettre en œuvre et nécessite une infrastructure et une organisation solides pour fonctionner de manière efficace.

Agilité VS Cycle en V

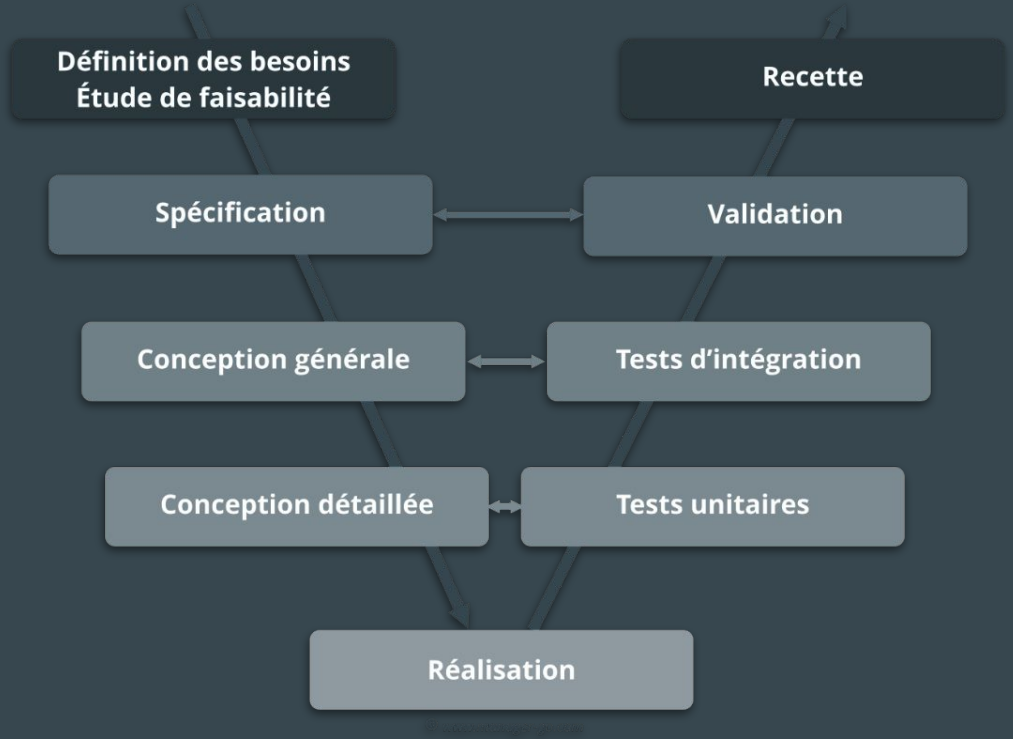
Cycle en V

Un modèle de développement qui consiste à **planifier** et à **réaliser** un projet en suivant un processus en forme de V, qui comprend généralement les étapes suivantes :

1. Conception
2. Développement
3. Test
4. Déploiement

Le souci : **Très linéaire et planifié**, qui vise à définir clairement les objectifs et les fonctionnalités du projet en début de cycle et à s**uivre un processus de développement étape par étape jusqu'au déploiement du projet.**

Cycle en V


Définition des besoins
Étude de faisabilité
Recette
Spécification
Validation
Conception générale
Tests d'intégration
Conception détaillée
Tests unitaires
Réalisation

Agilité

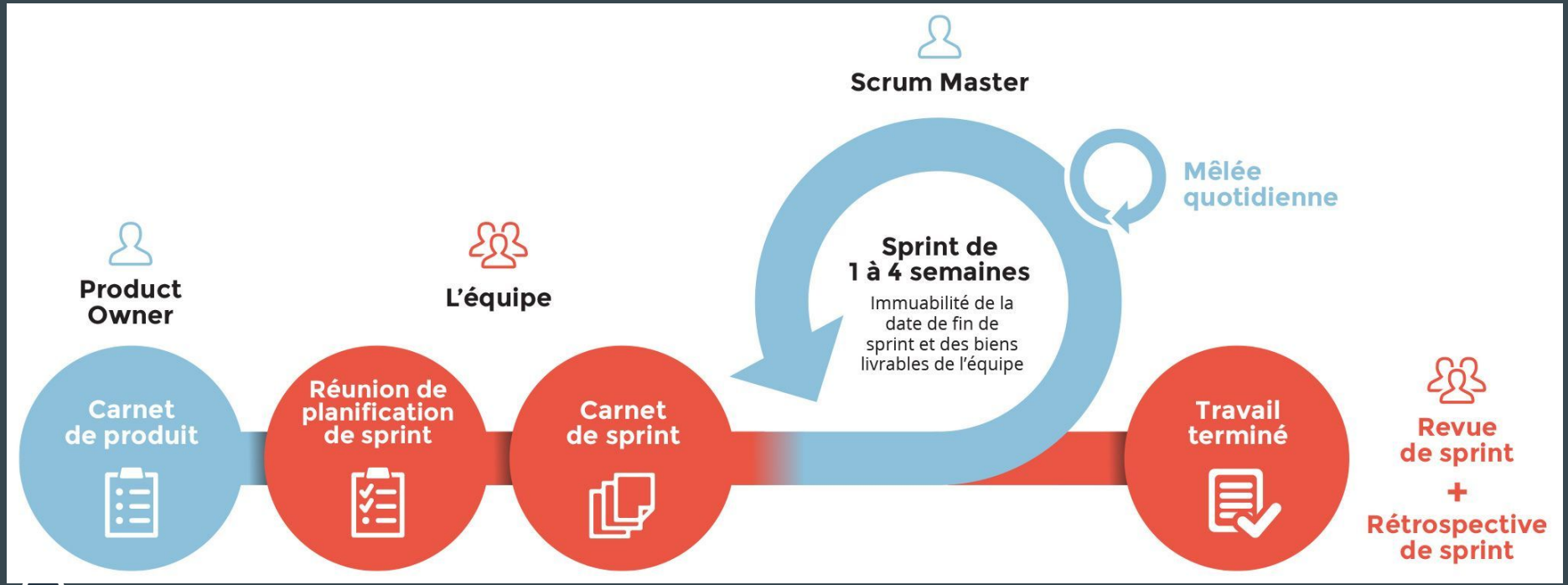
En opposition avec le cycle en V

Une approche de développement de logiciels qui met l'accent sur **la flexibilité** et l'**adaptation aux changements**.

Elle vise à développer des logiciels de manière i**térative et incrémentale,** en travaillant en cycles courts appelés "**sprints**".

Accent mis sur la **collaboration** et **l'interaction** avec les utilisateurs et les clients, et vise à fournir des logiciels de qualité rapidement et de manière continue.

Agilité


Scrum Master
Mêlée quotidienne
Product Owner
L'équipe
Sprint de 1 à 4 semaines
Immuabilité de la date de fin de sprint et des biens livrables de l'équipe
Carnet de produit
Réunion de planification de sprint
Carnet de sprint
Travail terminé
Revue de sprint
+
Rétrospective de sprint

Généralités Techniques

Git

Logiciel de **gestion de version** de code source, permet de suivre et de contrôler les modifications apportées au code source d'un projet.

Chaque modification du code source est stockée sous forme de "**commit**", de manière à pouvoir facilement **retracer l'historique** des modifications et revenir à une version précédente du code si nécessaire.

Git

Permet également de travailler en **collaboration** sur le même projet en utilisant des "**branches**".

Une branche est u**ne copie indépendante** du code source qui permet de travailler sur des fonctionnalités ou des correctifs de bugs sans perturber le code principal du projet.

Les modifications apportées à une branche peuvent être fusionnées (ou "mergées") avec le code principal une fois qu'elles sont prêtes.

Git


Git Version Control
TIME
v0.1
v0.2
v1.0
Main
Hotfix
Release
Develop
Feature
Feature

Monolithique vs Micro-services

Monolithique

- ➔ Toutes les parties d'une application sont regroupées en une seule unité de logiciel.
  - Tous les composants de l'application
  - La base de données
  - Les interfaces utilisateur
  - Les algorithmes de traitement de données
  - Autres

Monolithique vs Micro-services

Micro-services

- ➔ Divisée en plusieurs petits services indépendants qui communiquent entre eux pour réaliser des tâches spécifiques.
- ➔ Chaque service est conçu :
  - De manière à être indépendant
  - Réutilisable
  - De sorte qu'il peut être déployé
  - Exécuté de manière autonome

Monolithique vs Micro-services

Quelques différences clés entre les deux approches :

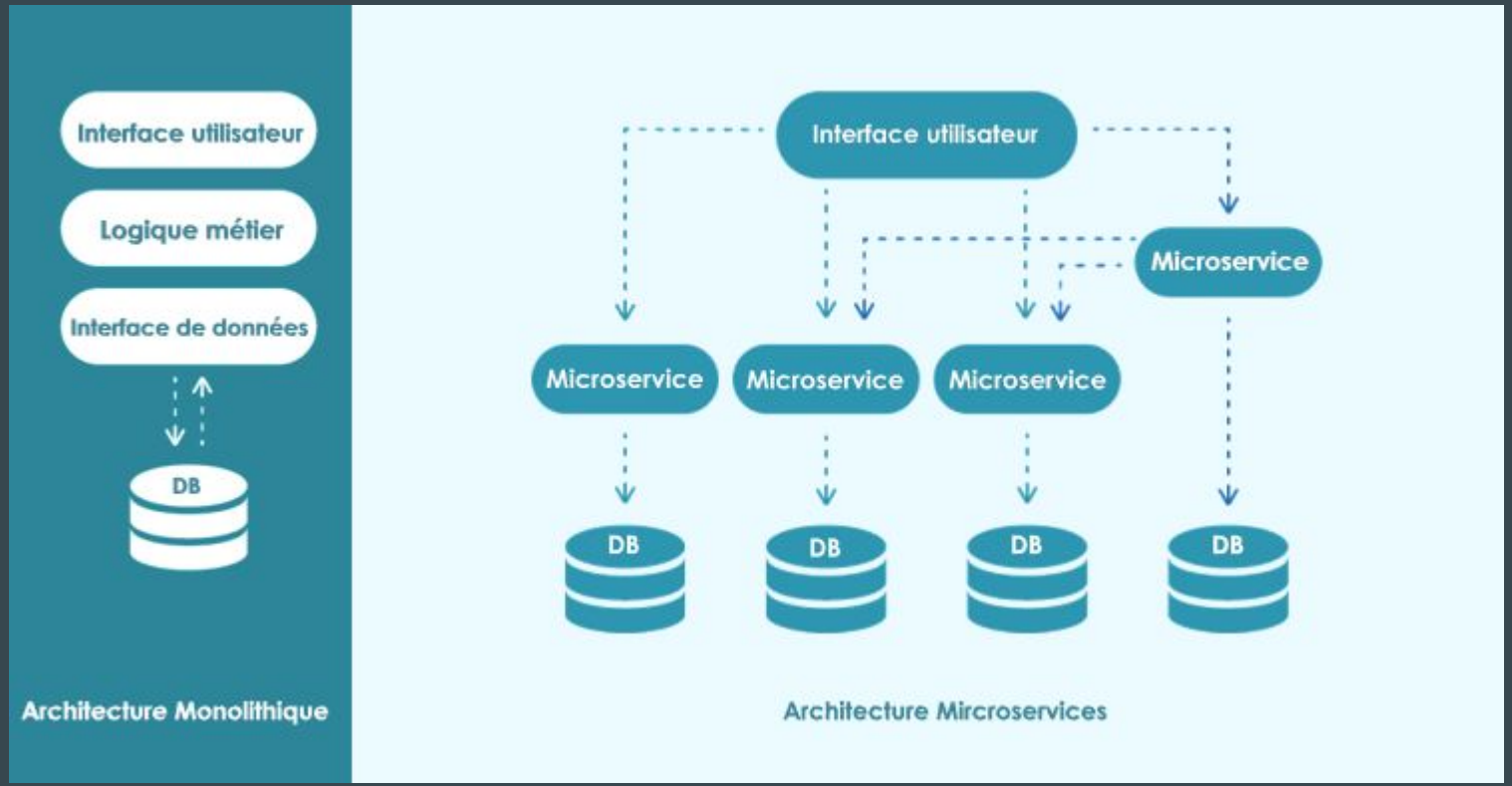
- ➔ Évolutivité
  - **Micro-service** : des services indépendants de manière plus facile et plus rapide, plus évolutive
  - **Monolithique :** peut être plus difficile à maintenir et à évoluer à mesure que l'application grandit et se complexifie.
- ➔ Fiabilité
  - **Micro-service** : mettre à jour des services indépendants de manière plus facile, ce qui peut améliorer la fiabilité de l'application.
  - **Monolithique** : peut être plus sensible aux erreurs et aux pannes, car une erreur dans un composant peut affecter l'ensemble de l'application.

Monolithique vs Micro-services

- ➔ Scalabilité
  - **Micro-service :** mettre à l'échelle des services indépendants de manière plus facile, ce qui peut améliorer la scalabilité de l'application.
  - **Monolithique :** peut être plus difficile à mettre à l'échelle, car tous les composants de l'application doivent être mis à l'échelle en même temps.

Il **n'y a pas de modèle** de développement "meilleur" en soi.
Le choix entre le développement monolithique et le développement micro-service **dépend souvent de la nature du projet** et des **objectifs de l'équipe** de développement.

Monolithique vs Micro-services



Bonnes pratiques

Quelques bonnes pratiques pour découper son code de manière propre et organisée :

- ➔ Respecter la modularité
- ➔ Utiliser des noms de variables et de fonctions explicites
- ➔ Commenter votre code
- ➔ Utiliser des conventions de style de code
- ➔ Utiliser des outils de formatage de code
- ➔ Tester votre code

Code Reviews

Une pratique courante dans le développement qui consiste à **examiner** le code source d'un projet par un ou plusieurs pairs avant de le valider et de le déployer.

Le but est de s'assurer que le code est de **qualité**, qu'il respecte les **conventions de style de code** et qu'il est **conforme** aux spécifications du projet.

Code Reviews

Comment se déroulent généralement les codes reviews ?

1. Un développeur soumet son code pour review
2. Un ou plusieurs pairs examinent le code
3. Le développeur apporte les modifications nécessaires
4. Le code est accepté et déployé

Peer Programming

Une pratique de développement dans laquelle d**eux développeurs travaillent ensemble** sur le même projet en utilisant un seul ordinateur.

Un des développeurs écrit le code, tandis que l'autre examine le code et suggère des modifications ou des améliorations.

Peer Programming

Plusieurs avantages par rapport au développement “traditionnel” :

- ➔ Amélioration de la qualité du code
- ➔ Apprentissage mutuel
- ➔ Réduction des erreurs et des bugs

Le peer programming **n'est pas adapté à tous les projets** et peut être moins efficace dans certains cas, mais il peut être une pratique très utile pour **améliorer la qualité** du code et **favoriser l'apprentissage** mutuel dans une équipe de développement.

Les SGBD

Acronyme de "**Système de Gestion de Bases de Données**" ("Database Management System")

Un système qui permet de

- ➔ Stocker
- ➔ Gérer
- ➔ Récupérer des données de manière structurée et efficace

Les SGBD sont utilisés dans de nombreux contextes, comme la gestion de bases de données pour les entreprises, les sites web, les applications mobiles, etc.

Les SGBD

Quelques exemples de fonctionnalités courantes des SGBD :

- ➔ Stockage de données
- ➔ Sécurité des données
- ➔ Gestion des transactions
- ➔ Interrogation des données

Les types de SGBD

Quelques exemples de types de SGBD courants :

- ➔ SGBD relationnels
  - ◆ MySQL, Oracle, Microsoft SQL Server.
- ➔ SGBD NoSQL
  - ◆ MongoDB, Cassandra, Redis.
- ➔ SGBD en mémoire :
  - ◆ Redis, Memcached, Apache Ignite.

Il existe également d'autres types de SGBD (SGBD objet-relationnels , SGBD orientés colonnes, ...)

Les Tests Unitaires

Les tests unitaires sont des tests de logiciels qui visent à **vérifier le bon fonctionnement d'une unité de code spécifique**, comme une méthode ou une fonction.

Ils sont généralement **exécutés automatiquement** (dans une CI par exemple) et permettent de s'assurer que le code fonctionne comme prévu et ne provoque pas **d'erreurs ou de bugs**.

Il existe aussi les tests fonctionnels qui s'assurent du bon fonctionnement d'une fonctionnalité complète.

Les Tests Unitaires

Quelques exemples de ce que peuvent couvrir les tests unitaires :

- ➔ Vérification que la méthode ou la fonction retourne le résultat attendu pour différentes entrées de données.
- ➔ Vérification que la méthode ou la fonction gère correctement les exceptions ou les erreurs qui pourraient survenir.
- ➔ Vérification que la méthode ou la fonction ne produit pas d'effets de bord indésirables sur d'autres parties du code.

Les tests unitaires sont généralement exécutés dans un environnement de test isolé, qui permet de s'assurer qu'ils ne sont pas affectés par d'autres parties du code ou par des données externes.

Sécurités Informatiques

Quelques exemples de failles de sécurité courantes dans le développement web :

- ➔ Injection SQL
- ➔ Injection de code
- ➔ Cross-Site Scripting (XSS)
- ➔ Déni de service (DoS)

Il est important de prendre en compte ces failles de sécurité lors du développement d'applications web afin de s'assurer qu'elles ne peuvent pas être exploitées par des attaquants.