



Cursus JAVA

M2I Formations 2021



Objectifs de la formation

- ▶ Etre capable de faire une démonstration de l'application CRM en ligne.
- ▶ Etre capable de montrer le découpage du projet et la gestion de projet sur Jira.
- ▶ Être capable de présenter le projet (prototypes, designs, spécifications fonctionnelles et détaillées)
- ▶ Etre capable d'expliquer le code (Back et Front) et de justifier les choix (bdd, technos, langages, api et intégration continue avec Jenkins).
- ▶ Être capable d'expliquer les cycles de livraisons et la mise en production.
- ▶ Être capable de parler des notions présentes dans chaque plan de cours et pour chaque module.
- ▶ Etre capable d'expliquer les processus de montée en compétence appliqués durant la formation (pair programming, peer correcting et revues de code).
- ▶ Montrer aux futurs employeurs que vous êtes aptes à intégrer des projets d'entreprise, que vous êtes à l'aise avec les processus, technos et outils, que vous avez acquis les bases pour une montée en compétence à la fois en autodidacte et en entreprise.



MODULE AGILE

Comprendre la démarche Agile



1.

Introduction à l'agilité



Qu'est ce que l'agilité

L'agilité est né au début du XXIème siècle, lorsque les industries ont commencé à découper la production en micro-tâche.

Toyota met en place en premier la gestion de projet se rapprochant de l'agilité d'aujourd'hui en introduisant :

- une culture de l'amélioration en continu et d'autonomisation des salariés (**le kaizen**),
- l'étiquetage (**kanban**) pour la gestion des pièces à produire ou à livrer et éviter ainsi le gaspillage,
- l'adaptation face au changement,
- le respect des salariés,
- la force du travail d'équipe,
- l'importance d'une philosophie à long terme plutôt que les objectifs financiers à court terme.

En 2001 le manifeste agile est créé



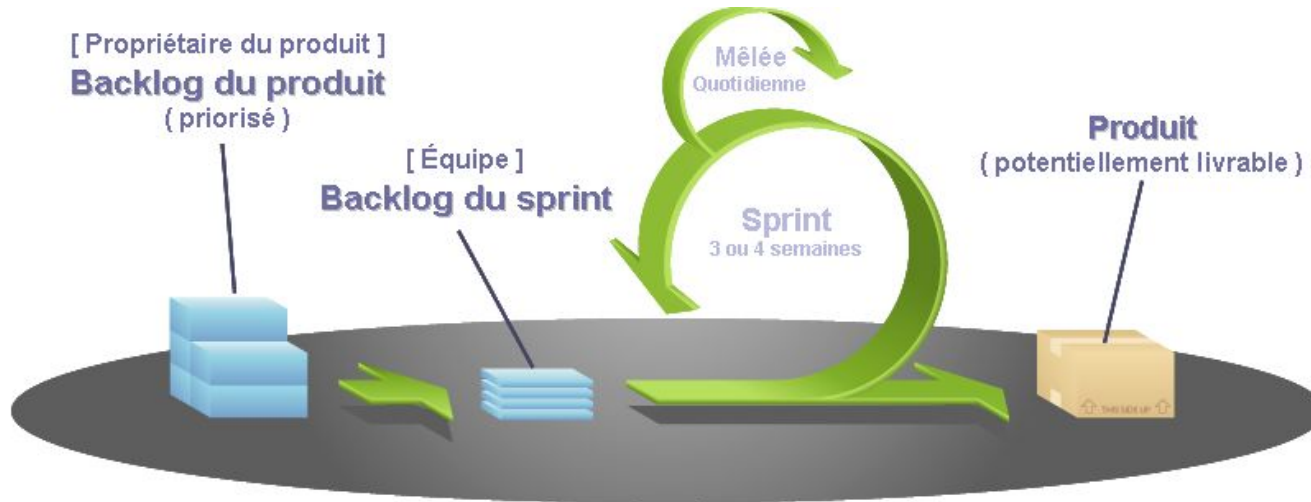
Qu'est ce que l'agilité : les 4 valeurs

- Les individus et leurs interactions plus que les processus et les outils :
Communiquer à travers des processus et des outils nécessite un formatage des messages ainsi que le respect de leurs rythmes particuliers : structurons plutôt un contexte où les individus interagissent en direct de manière plus efficace.
- Du logiciel qui fonctionne plus qu'une documentation exhaustive :
Documenter l'action de façon bureaucratique est généralement une perte de temps : réduisons cela au strict minimum nécessaire et choisissons plutôt l'avancée opérationnelle de notre produit comme repère.
- La collaboration avec les clients plus que la négociation contractuelle :
Impliquons l'utilisateur/client à chaque étape du développement – et pas seulement avant et après : c'est l'unique solution pour accompagner dynamiquement ses besoins réels.
- L'adaptation au changement plus que le suivi d'un plan :
Procédons par itérations courtes plutôt que sur la base d'un plan : cela permettra de mieux nous adapter

L'agilité : SCRUM

SCRUM est un framework de la méthode Agile. Il décrit un cadre de travail permettant la mise en place des méthodes agiles.

Ce framework se base sur un découpage du projet en boîte de temps (Time-Box) appelé **Sprint**.





L'agilité : eXtrem Programming

eXtrem Programming est une méthode agile permettant l'amélioration de la qualité du code produit grâce au travail en binôme. L'idée est de pousser chaque concept à l'extrême :

- La revue de code est une bonne pratique, elle sera faite en permanence (par un binôme) ;
- Les tests sont utiles, ils seront faits systématiquement avant chaque mise en œuvre ;
- La conception est importante, le code sera retravaillé tout au long du projet (*refactoring*) ;
- La simplicité permet d'avancer plus vite, la solution la plus simple sera toujours celle qui sera retenue;
- La compréhension est importante, des métaphores seront définies et évolueront en concomitance;
- L'intégration des modifications est cruciale, celles-ci seront faites plusieurs fois par jour ;
- Les besoins évoluent vite, des cycles de développement très rapides faciliteront l'adaptation au changement.

L'agilité : kanban

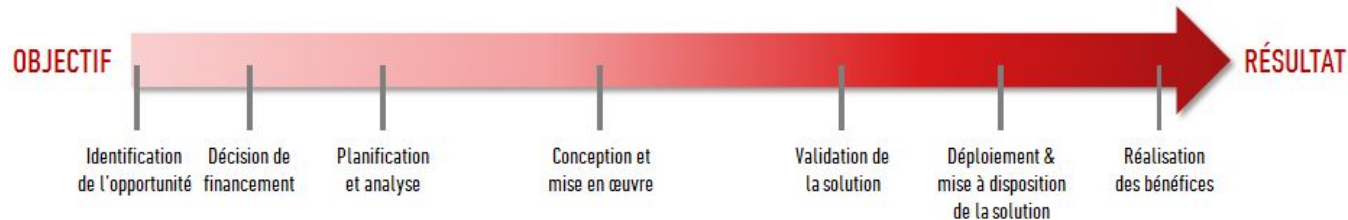
Le **Kanban** est un système visuel de gestion des processus qui indique quoi produire, quand le produire et en quelle quantité.



L'agilité et le devops

LES ENTREPRISES DOIVENT ÊTRE CAPABLE DE MODERNISER LEURS PROCESSUS DE GESTION DE PROJET

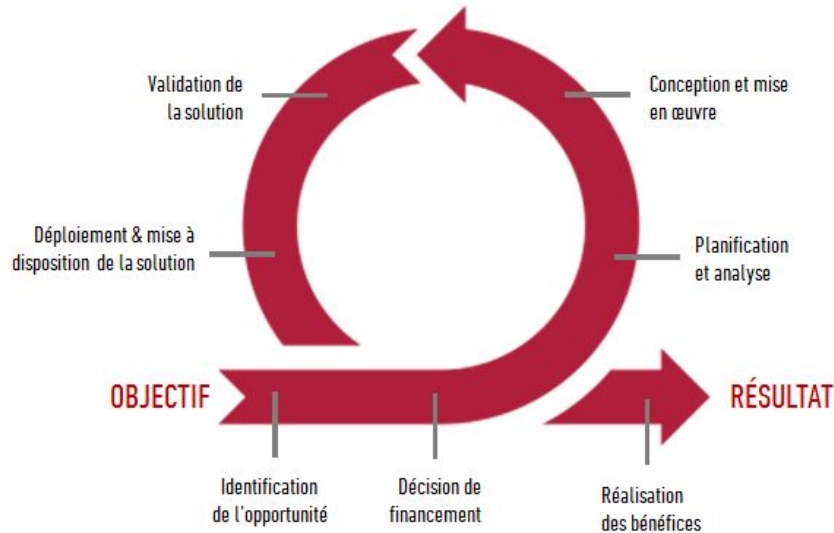
HIER



L'agilité et le devops

LES ENTREPRISES DOIVENT ÊTRE CAPABLE DE MODERNISER LEURS PROCESSUS DE GESTION DE PROJET

DEMAIN





2.

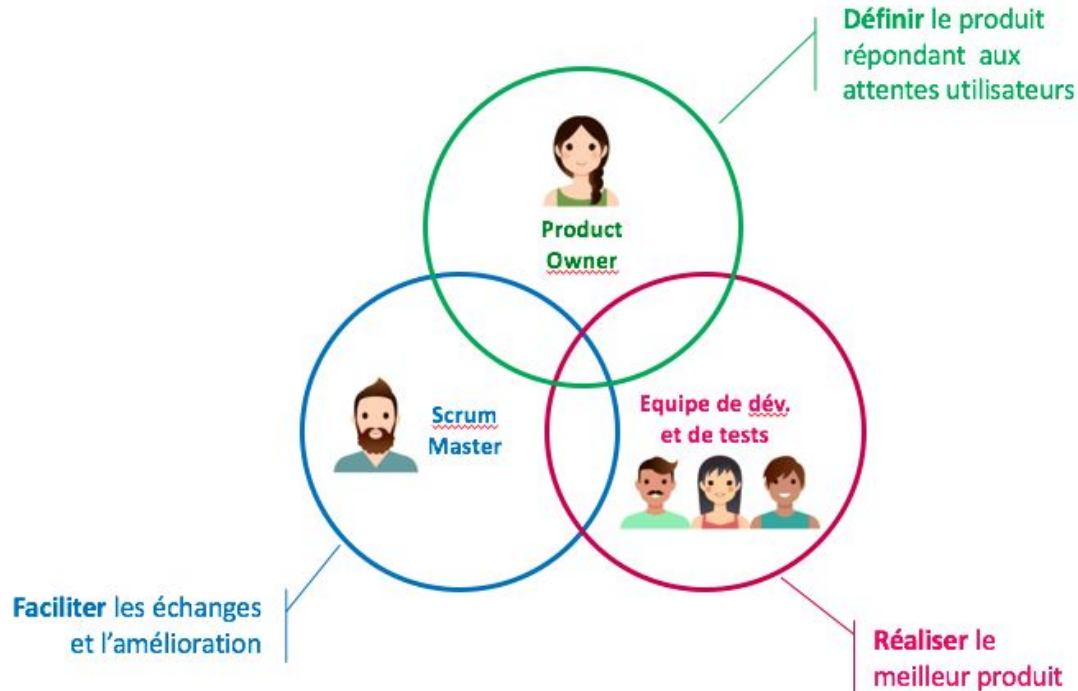
L'agilité : les pratiques via SCRUM



SCRUM : les rôles & responsabilités

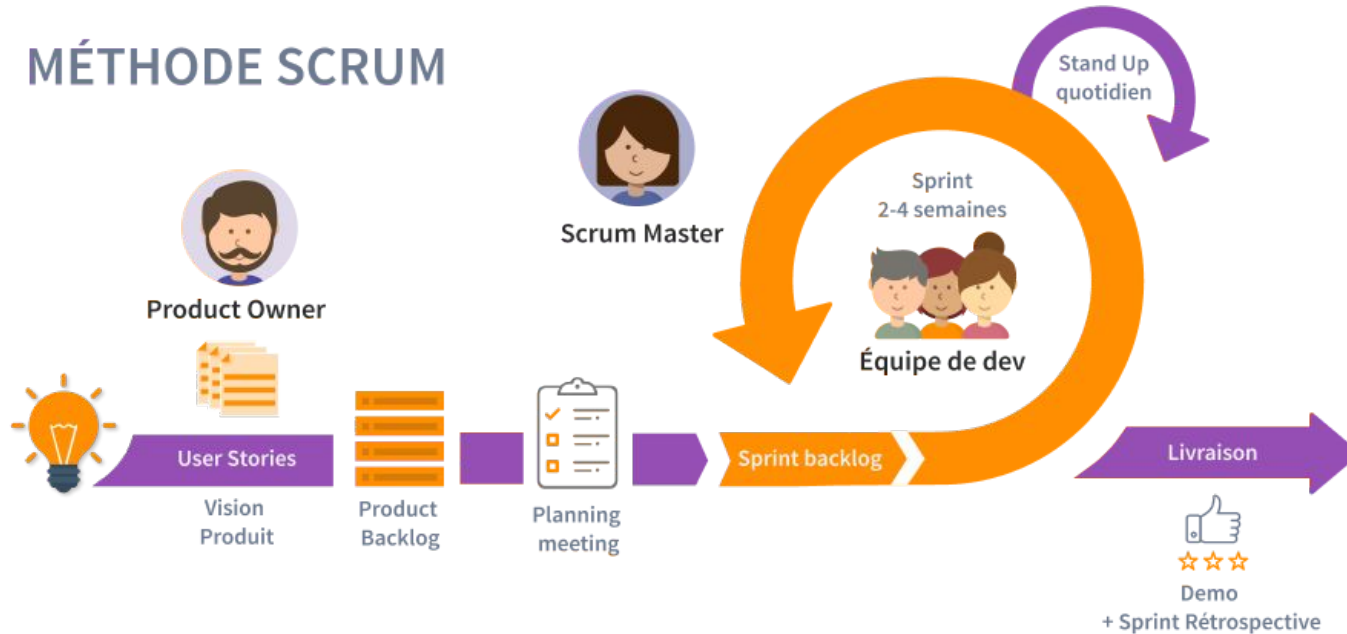
Le Product Owner (PO)	Le stratège. Il recueille les besoins, estime leur valeur. Il porte la vision du produit . Il est en charge de la construction et la mise à jour du backlog. Il est garant du produit et de sa valeur .
Le Scrum Master	Le facilitateur. Il élimine les obstacles dûent à l'environnement de travail. Il est garant de l'application des principes et pratiques SCRUM .
L'équipe de développement	Les réalisateurs. Ils analysent et estiment les user stories . Ils déterminent la meilleure façon de réaliser leur travail . Ils sont garants de la qualité du livrable produit .

SCRUM : les rôles & responsabilités



SCRUM : au quotidien

MÉTHODE SCRUM





L'agilité : les cérémonies

Le Sprint Planning

Objectif : **Organiser le sprint qui débute**

Déroulement :

Le Product Owner définit les objectifs du sprint

L'équipe estime les users stories à réaliser

Le Product Owner propose une liste de tâches à réaliser durant le sprint



L'agilité : les cérémonies

Le Dayli Scrum

Objectif : **Faire un état des lieux de l'avancement et relever les points bloquants**

Déroulement :

Chaque membre doit expliquer ce qu'il a fait depuis le dernier daily

Chaque membre doit expliquer ce qu'il va faire avant le prochain daily

Chaque membre doit lever les alertes s'il y en a



L'agilité : les cérémonies

Le Sprint Review

Objectif : **Point final d'un sprint, utilisé également pour présenter une itération aux parties prenantes**

Déroulement :

Le Product Owner présente le travail livré pendant le Sprint

Démonstration d'une nouvelle fonctionnalité

Le SCRUM Master présente les indicateurs d'avancée du projet

Echange avec les parties prenantes



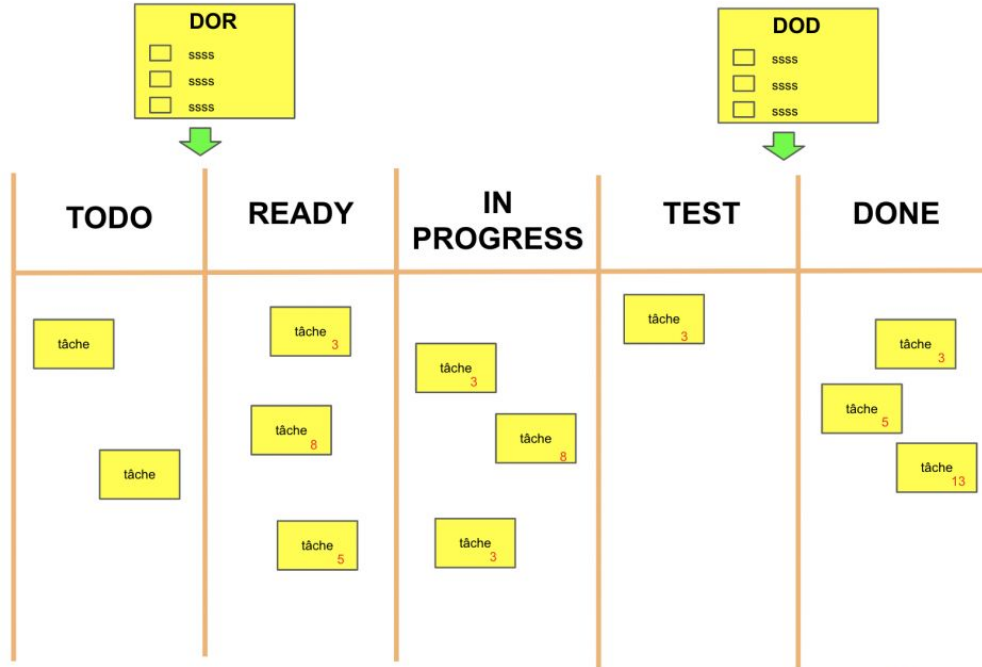
L'agilité : les cérémonies

Le Sprint Retrospective

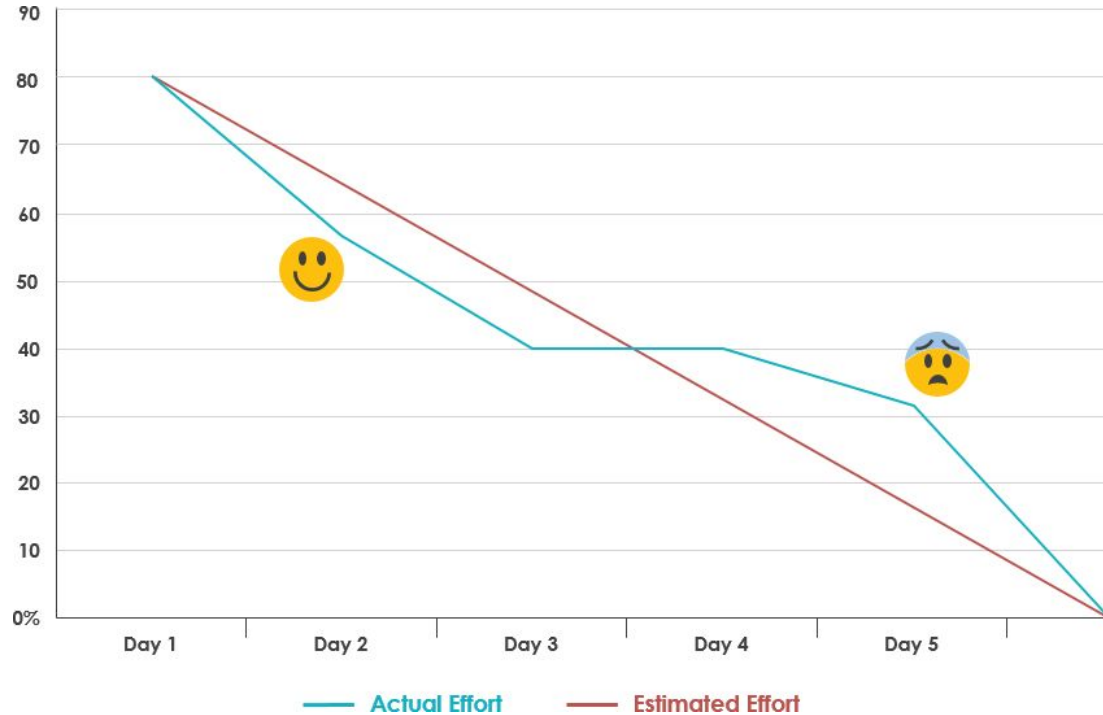
Objectif : **Travailler sur l'amélioration continue des process de l'équipe**

Déroulement : à définir en équipe !

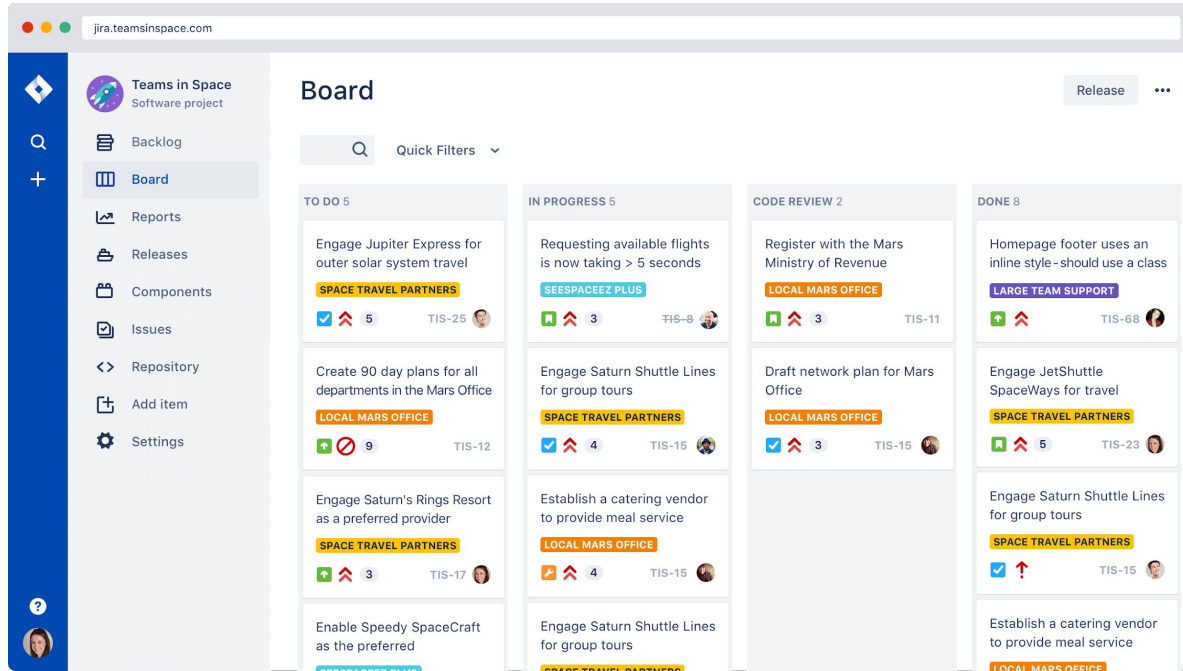
L'agilité : la Definition of Done (DoD)



L'agilité : Burndown



SCRUM : outil JIRA



The screenshot shows the JIRA Scrum Board for the 'Teams in Space' project. The interface includes a left sidebar with navigation options: Backlog, Board (selected), Reports, Releases, Components, Issues, Repository, Add item, and Settings. The main area displays a Kanban-style board with four columns: TO DO (5 items), IN PROGRESS (5 items), CODE REVIEW (2 items), and DONE (8 items). Each column contains task cards with details such as the task description, assignee, due date, and status. For example, in the 'TO DO' column, the first task is 'Engage Jupiter Express for outer solar system travel' assigned to 'TIS-25' with a due date of 'TIS-25'. The 'IN PROGRESS' column shows 'Requesting available flights is now taking > 5 seconds' assigned to 'TIS-8'. The 'CODE REVIEW' column has 'Register with the Mars Ministry of Revenue' assigned to 'TIS-11'. The 'DONE' column includes 'Homepage footer uses an inline style - should use a class' assigned to 'TIS-68'.

MODULE SQL

Langage SQL ou PL/SQL

1.

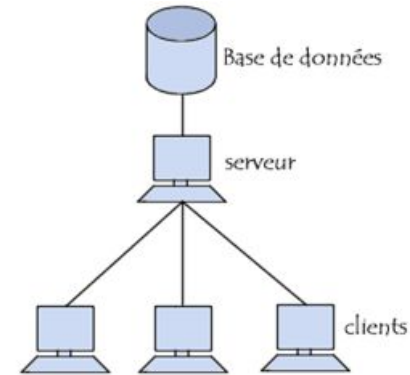
Base de données : les concepts



Base de données

- **Ensemble structuré d'informations** (*d'une entreprise ou organisation*), **mémorisé** sur une machine (*serveur*).
- **Données stockées** et **organisées** sous forme de fichiers ou ensemble de fichiers.
- Une BD sert à **créer, enregistrer, récupérer** et **manipuler** des **données communes**.

- **Système de Gestion de Base de Données** (ou DBMS: Data Base Management System)
- **Ensemble cohérent de services** (*logiciels*) permettant aux utilisateurs **d'accéder, mettre à jour ou administrer** une DB
- Fonctionne sur le modèle **client/serveur** (requêtes/traitements)





SGBD

Pourquoi utiliser un SGBD ? Quels objectifs ?

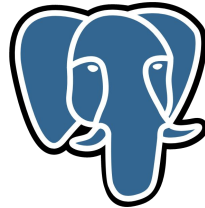
- Indépendance physique
- Indépendance logique
- Accès / partage des données
- Administration centralisée
- Non redondance des données
- Cohérence des données
- Sécurité des données
- Résistance aux pannes



SGBD

- Installation de PostgreSQL 13.4 (+ pgAdmin)

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>



- Docs SQL : <https://sql.sh/>



2.

Base de données : modélisation



Modélisation

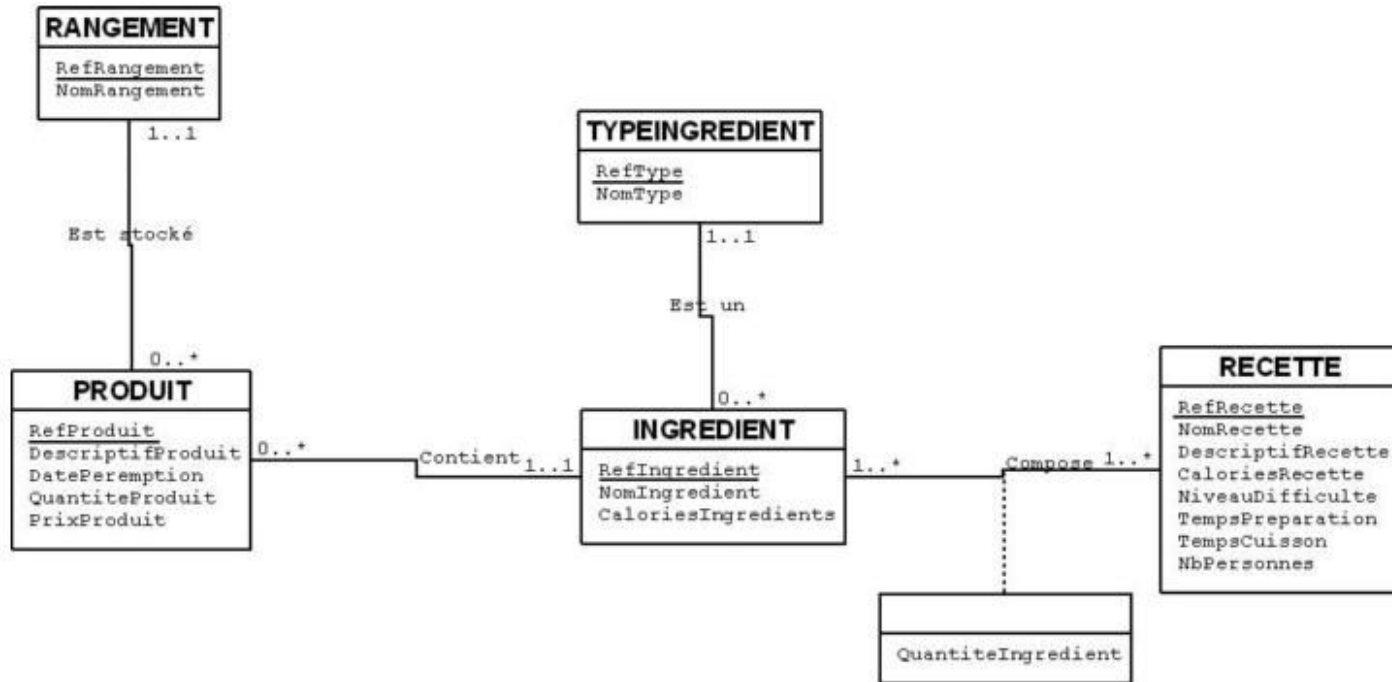
- **Pourquoi modéliser ?**
 - Avoir une représentation graphique de la structure
 - Connaître les propriétés attendues d'une données
 - Connaître les relations entre les données
- **Comment modéliser ?**
 - Effectuer un design conceptuel
 - Insérer des cardinalités
 - Effectuer un modèle logique



Modélisation : les règles à respecter

- **Normalisation des tables**
 - Ne contient pas d'espace, d'accents ni de caractères spéciaux
 - Tout doit être écrit en lowercase (minuscule)
 - Les espaces sont remplacés par des underscores : “_”
- **Les relations**
 - Les relations “1,n” : l'identifiant de la table mère est dupliqué en clef étrangère de la table fille
 - Les relations “n, n”:
 - Effectuer un modèle logique
- **Un identifiant**
 - Un champs d'identification unique est obligatoire, la clef primaire (souvent nommé “id”).

Modélisation : Exemple UML





3.

Base de données : Le SQL



SQL: Les requêtes

- **Création d'une base**

- CREATE DATABASE <nom_colonne>;

- **Création d'une table**

- CREATE TABLE <nom_table>(<nom_colonne> <type_colonne> <contraintes>, ...)

- **Modification d'une table**

- ALTER TABLE <nom_table> ALTER COLUMN <nom_colonne> TYPE <type_colonne>
<contraintes>



SQL: Les requêtes (CRUD)

- **Lire des données**
 - `SELECT <nom_colonne> FROM <nom_table>`
 - `SELECT <nom_colonne> FROM <nom_table> WHERE <condition>`
- **Ajouter des données**
 - `INSERT INTO <nom_table> VALUES ('valeur1', 'valeur2',)`
- **Modifier des données**
 - `UPDATE <nom_table> SET <nom_colonne> = 'valeur' WHERE <condition>`
- **Supprimer des données**
 - `DELETE FROM <nom_table> WHERE <condition>`

Les outils Devops

Git, Jenkins, Docker, Ansible



1.

DevOps : Git



Git : C'est quoi, à quoi ça sert ?

- Logiciel de gestion des versions des sources
- Fonctionne via un système de client (local) et serveur (distant)
- Permet le partage et la sauvegarde de chaque version du code
- Permet à un grand nombre de collaborateurs de travailler sur un même projet, en offrant une bonne gestion des modifications.
- Permet de synchroniser un projet entre plusieurs collaborateurs, en ayant l'assurance que tous les fichiers soient à jour.
- Permet d'avoir un historique précis de tous les changements et modifications d'un projet. Cela permet de clarifier les questions récurrentes: "Où est la dernière version du fichier X?" et "Qu'est-ce qui a été changé entre les révisions 41 et 42"?



Git : Terminologie

- **Versioning** : Le terme versioning fait référence à l'action de contrôler les versions. Donc, GIT va nous permettre de faire du versioning.
- **Repository** (Repo) : c'est le dépôt, le dossier de travail contenant les fichiers d'un projet
- **Commit** : action d'enregistrer vos modifications dans l'historique du projet. Il est accompagné d'un commit message.
- **Push** : permet d'envoyer les commits locaux vers le serveur (remote)
- **Pull** : récupérer les commits depuis le serveur en local
- **Checkout** : permet de voyager dans le temps, en retournant vers un commit précis
- **Stash** : Permet de mettre de côté vos modifications, sans commiter
- **Merge** : action de fusionner deux versions d'un projet
- **Rebase** : action de rembobiner pour modifier des commits précédents
- **Branch** : version alternative du projet



Où utiliser GIT

Techniquement, git peut être utilisé partout, c'est-à-dire qu'il peut être utilisé pour gérer des version des fichiers word, excel, Text,etc.

Mais, son objectif principale est orienté vers le versioning des fichiers faisant partie du code source d'un projet de développement(software).



Où utiliser GIT

Un gestionnaire de version nous permettra de faire principalement garder en mémoire 3 choses:

- ☐ Chaque modification de chaque fichier
- ☐ La raison de sa modification
- ☐ L'auteur de la modification



Où utiliser GIT

Git peut être utilisé dans de manière individuelle ou dans un environnement en équipe car dans les 2 cas il permet de :

- ☐ Revenir à une version précédente de votre code en cas de problème.
- ☐ Suivre l'évolution de votre code étape par étape.
- ☐ Travailler sur des petits incréments, ce qui facilite le retour en arrière en ayant connaissance de la dernière version stable du code.

En équipe, avec la particularité d'avoir

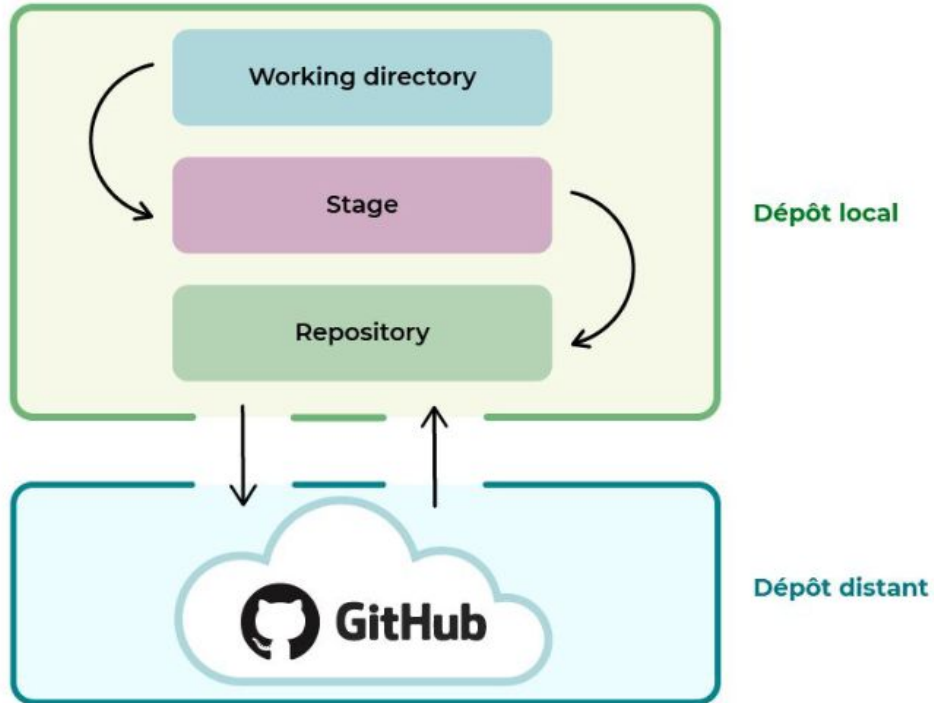
- ☐ plusieurs développeurs dans un même projet qui écrivent du code sans toucher ou supprimer les modifications apportées par d'autres
- ☐ Possibilité de mettre en commun le travail effectué par plusieurs membres de l'équipe où chacun travail sur des fonctionnalité différentes.



Git : Interface



Fonctionnement du dépôt local





Working directory

Cette zone correspond au dossier du projet sur votre ordinateur.



Stage ou Index

C'est une zone intermédiaire entre le working directory et le repository local git.

C'est là que se trouve tous les fichiers modifiés que vous souhaitez voir apparaître dans votre prochaine version de code.



Le repository

C'est la zone où sont stockées les changements validés, c'est-à-dire, nos versions.



Installation GIT

Installer GIT : <https://git-scm.com/>

Configurer GIT :

```
git config --global user.name "Alexandre Cases"
```

```
git config --global user.email "a.cases@cleverdev.fr"
```

```
git config --list : Pour voir toutes vos configurations
```




Revoir les base de la ligne de commande

Avec git, vous allez beaucoup manipuler la ligne de commande, donc il est nécessaire de revenir sur les notions de base de la ligne de commande.

- ❑ `ls` : Elle permet de lister le contenu d'un répertoire
- ❑ `mkdir <nom_du_repertoire>` : permet de créer un répertoire ou dossier
- ❑ `rm <nom_du_fichier>` : Supprimer un fichier
- ❑ `rm -d <nom_du_dossier>` : Supprimer un dossier vide
- ❑ `rm -r <nom_du_dossier>` : Effectue une suppression récursive et permet la suppression des dossiers sous dossiers.
- ❑ `touch <nom_fichier>` : Permet de créer un fichier
- ❑ `cat <nom_fichier>` : Permet de voir ou lire le contenu d'un fichier
- ❑ `cd` : Permet de naviguer dans notre file system
- ❑ `pwd` : Permet de voir le chemin absolu vers le dossier où l'on se trouve
- ❑ `du -h` : Taille occupée par les dossiers (l'option -h c'est pour avoir un format d'affichage facile à comprendre).



Git : Premier pas

- Création d'un dépôt
- Récupération du dépôt en local

Exercice 1 : Créer un répertoire sandwich. Créer un fichier à l'intérieur 'burger.txt' qui contient la liste des ingrédients d'un burger (un par ligne), par exemple :

Steak

Salade

Tomate

Cornichon

Fromage



Git : Premier pas

Exercice 2 :

- Vérifier l'état de votre dépôt local : `git status`
- Préparer `burget.txt` pour le commit : `git add`
- Commitez les modifications : `git commit -m "message de commit"`
- Vérifier l'état du dépôt local
- Vérifier la liste des changements prêt à l'envoi : `git log`



Git : Premier pas

Exercice 3 :

- Créez quelques autres sandwiches (hot_dog.txt, jambon_beurre.txt, etc.), modifiez également le contenu de burger.txt. **Attention** : chaque modification doit être placée dans un commit dédié. Effectuez au moins 5 modifications en reprenant les étapes de l'exercice 2.
- Testez `git log --graph --pretty=short`



Git : Premier pas

Exercice 4 :

- Réalisez une modification dans une de vos recettes. **Attention** ne commitez pas
- Vérifier les modification via un git status puis git diff
- Réalisez un git reset <nom_du_fichier>
- Vérifier les modification via un git status puis git diff
- Réalisez un git checkout <nom_du_fichier>
- Réalisez un git status



Git : Premier pas

Exercice 5 :

- Réalisez un push : `git push`
- Allez constater la mise à jour sur le repository

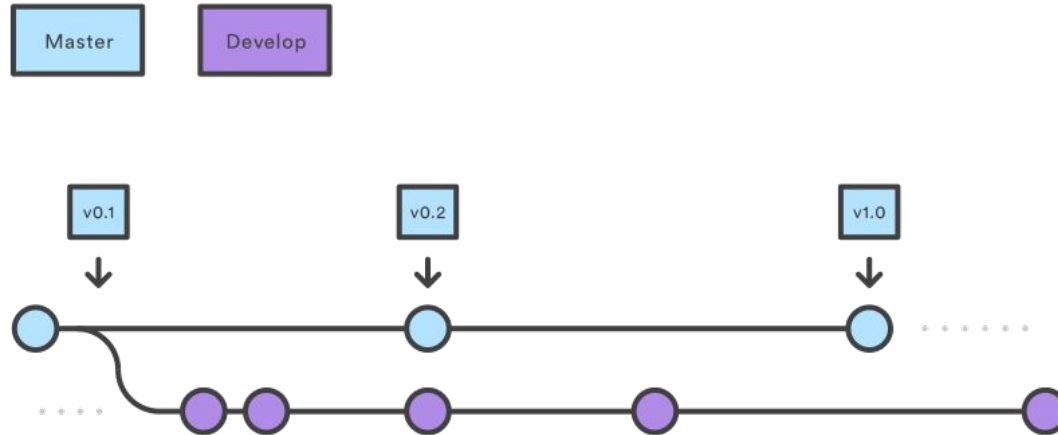


Git : Le Gitflow

- Le Gitflow est un workflow de travail. Il décrit les processus d'utilisation de git dans une équipe (créé par Vincent Driessen).
- Le Gitflow représente un framework particulièrement efficace et adapté aux projets avec un cycle de livraison planifié et régulier.
- Ce workflow n'ajoute aucun concept ni aucune commande en dehors de ce qui est exigé pour le workflow d'organisation des branches

Gitflow : Develop & Master

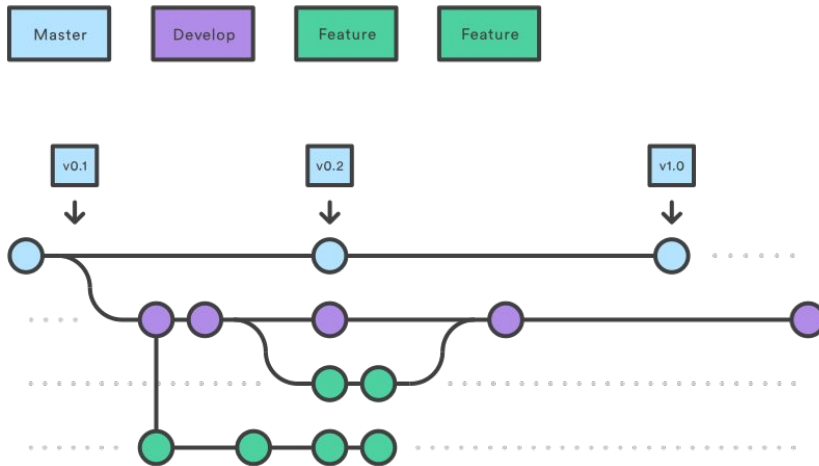
- Le Gitflow utilise deux branches principales au lieu d'une seule.
- La branche master stocke l'historique officiel des versions
- La branche develop sert de branche d'intégration pour les fonctionnalités



- Reprendre le projet des exercices précédent et créer une branche develop depuis master et pusher

Gitflow : Branche de fonctionnalité

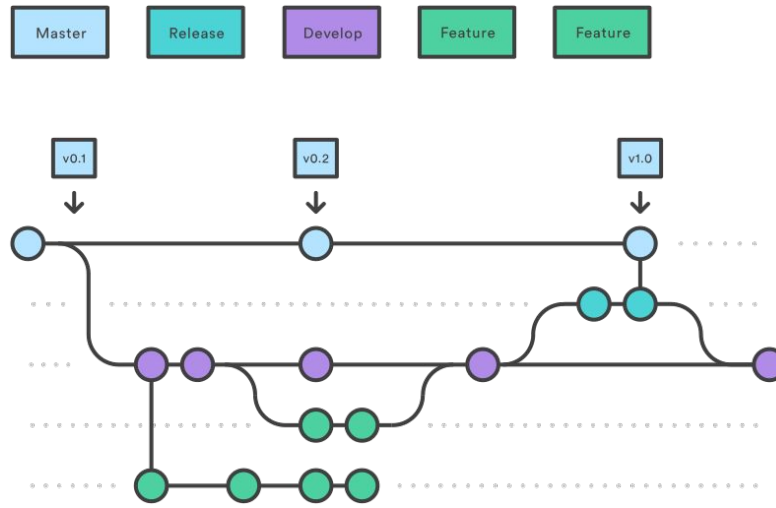
- Le Gitflow utilise une branche par fonctionnalité à développer
- Ces branches sont créées à partir de la branche develop
- Ces branches seront ensuite pushées en vue d'une intégration sur develop



- Créer une branche feature/xxx sur le projet, ajouter un sandwich et pushez sur le repository

Gitflow : Branche de livraison

- Lorsque vous souhaitez préparer une livraison d'une version, vous allez créer une branche de livraison, souvent nommée release, depuis develop
- La création de cette branche débute un cycle de livraison :
 - Aucune fonctionnalité supplémentaire ne pourra être ajoutée à cette branche
 - Dédiée à la corrections de bugs, documentation
- Une fois la branche prête elle sera mergée dans master et taguée d'un numéro de version





-



Gitflow : Le résumé

- Une branche de développement (develop) est créée à partir de master.
- Une branche de version (release) est créée à partir de develop.
- Les branches de fonctionnalité (feature) sont créées à partir de develop.
- Lorsqu'une branche de fonctionnalité (feature) est terminée, elle est mergée dans la branche de développement (develop).
- Lorsque la branche de version (release) est terminée, elle est mergée dans les branches de développement (develop) et principale (master).
- Si un problème est identifié dans la branche principale (master), une branche de maintenance (hotfix) est créée à partir de master.
- Une fois la branche de maintenance (hotfix) terminée, elle est mergée dans les branches de développement (develop) et principale (master).



2.

DevOps : Ansible



Ansible : présentation



Ansible est un moteur d'automatisation informatique open source, qui peut éliminer les corvées de votre vie professionnelle, et améliorer également considérablement l'évolutivité, la cohérence et la fiabilité de votre environnement informatique.

Vous pouvez utiliser Ansible pour automatiser trois types de tâches :

- Provisioning : configurer les différents serveurs dont vous avez besoin dans votre infrastructure
- Configuration : modifier la configuration d'une application, d'un système d'exploitation ou d'un périphérique, démarrer et arrêter les services, installer ou mettre à jour des applications, mettre en œuvre une politique de sécurité, ou effectuer une grande variété d'autres tâches de configuration.
- Déploiement d'applications : adopter une démarche DevOps en automatisant le déploiement d'applications développées en interne sur vos environnements de production.

En gros, c'est l'outil de scripting idéal pour un devOps.

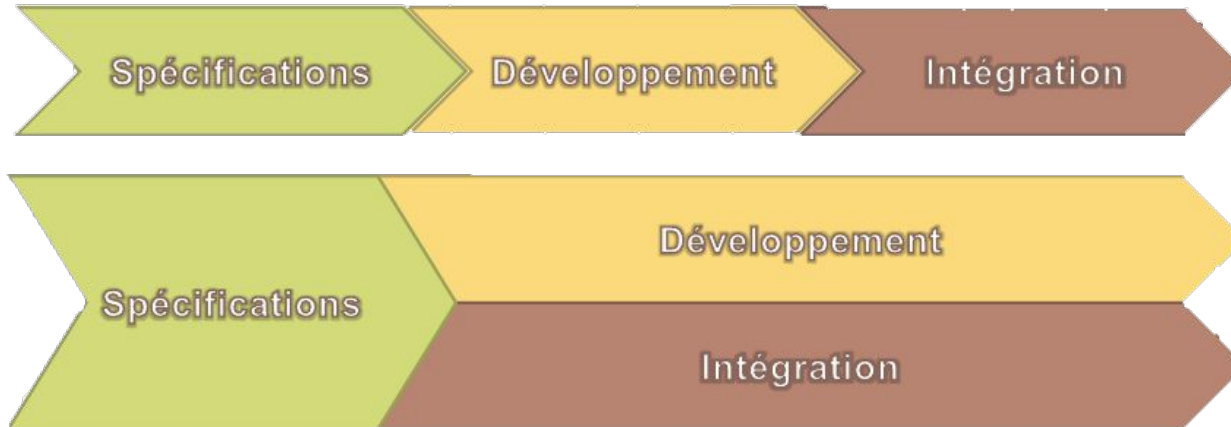


3.

DevOps : Jenkins

Jenkins : c'est quoi ?

- Serveur d'intégration continue (CI)
- Utilisé pour automatiser des tâches comme :
 - La compilation du projet
 - L'exécution des tests
 - Le déploiement





Pourquoi utiliser Jenkins ?

- Dédié aux DevOps, Jenkins est un outil d'intégration continue open source (sous licence MIT) développé en Java.
- A chaque modification de code d'une application dans le gestionnaire de configuration, Jenkins se charge automatiquement de la recompiler, et de la tester.
Pour cette seconde étape, Jenkins intègre le framework de test open source JUnit. En cas d'erreur détectée, Jenkins alerte le développeur afin qu'il résolve le problème. Un process évidemment des plus avantageux dans le cadre d'un projet de développement.
- Fork de l'outil Hudson, Jenkins s'adosse à un serveur de servlets comme Apache Tomcat ou peut reposer sur son propre serveur web embarqué. Accessible via un navigateur web, il est compatible avec les systèmes de gestion de versions les plus populaires comme Git ou Subversion. De manière standard, il supporte les pipelines d'intégration continue (CI) basés sur les outils de build Apache Ant et Apache Maven.



L'écosystème Jenkins

- Via son Update Center, Jenkins propose + de 1 500 plugins permettant d'étendre son environnement d'intégration continue.
- Exemple :
 - Automatisation des tests
 - D'un déploiement Docker
 - Ou même serveur
 - etc...
- <https://www.jenkins.io/>.



4.

DevOps : Docker



Docker : présentation

- **Docker** permet de créer des environnements (appelées containers) de manière à isoler des applications.
- Il repose sur le kernel **Linux** et sur une fonctionnalité : les **containers**.
- Le principe est d'exécuter du code dans un environnement isolé.
- Docker a également d'un autre composant : **cgroups**. Ce composant a pour objectif de gérer les ressources (utilisation RAM, CPU, etc.).

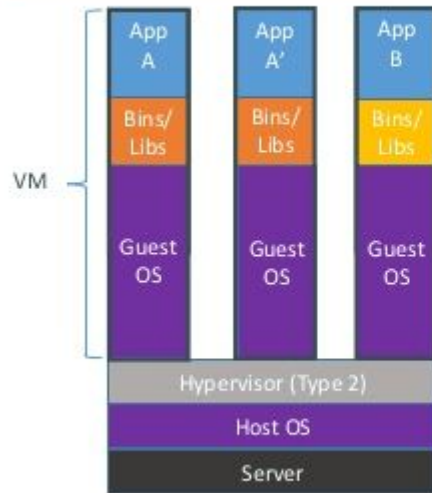


Docker vs Machine virtuel (VM)

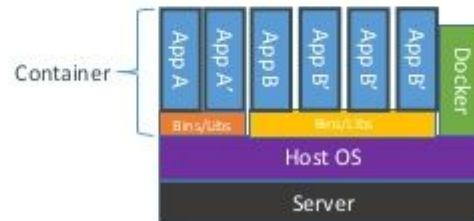
- Une machine virtuelle isole tout un système (son OS), et dispose de ses propres ressources.
- Dans le cas de **Docker**, le kernel va partager les ressources du système hôte et interagir avec les containers. Techniquement, **Docker n'est pas une VM**, mais en terme d'utilisation, **Docker** peut-être apparenté à une **VM**.
- Lancer un environnement, et isoler les composants de ce container avec les composants de mon hôte, voilà ce que **Docker** sait faire ! Il le fait d'ailleurs très bien, et reste une alternative bien plus performante que les **VMs** (à utilisation équivalente).

Docker vs Machine virtuel (VM)

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries





Docker : cas d'utilisation

Une entreprise lyonnaise que j'appellerai à tout hasard **Wanadev**, souhaite développer un projet Java comme ils savent le faire. Leur équipe est composée de deux personnes.

- Manu est un vieux de la vieille. Lui, Debian 7, c'est parfait. De plus, il est ISO avec les serveurs de production.
- Baptiste lui est un vrai hippie. Il dispose de la dernière distribution exotique de debian. Lui, **JAVA**, c'est la toute dernière version ou rien.

Cependant il va falloir réaliser des tests unitaires, ce qui va poser problèmes aux deux développeurs...

En effet, ça pose problème que deux développeurs ne travaillent pas sur les mêmes environnements. Grâce à Docker, on peut faire en sorte que Manu et Baptiste travaillent sur les mêmes versions Linux sans craindre des problèmes de compatibilité entre leurs codes respectifs. Dans ce cas, le plus simple est de mettre en place un **Dockerfile**, document "chef d'orchestre", qui permettra à Manu et à Baptiste de monter une image similaire. Ce **Dockerfile** sera calqué sur les éléments présents en production. Du coup, Manu et Baptiste en plus de travailler sur un environnement identique, seront sur un environnement similaire à celui de la production !



Docker : présentation technique

- Démonstration des containers, les lister, les arrêter etc.
- Présentation d'un dockerfile et de son utilité
- Explication du docker-compose

<https://www.docker.com/get-started>

Introduction à la programmation



1.

Introduction à la programmation



Introduction à la programmation

- La programmation permet de résoudre un problème de manière automatisée grâce à l'application d'un algorithme :

Programme = Algorithme + Données

- Un algorithme est une suite d'instructions qui sont évaluées par le processeur sur lequel tourne le programme.
- Les instructions utilisées dans le programme représentent le **code source**.
- Pour que le programme puisse être exécuté, il faut utiliser un langage que la machine peut comprendre : un **langage de programmation**.



Introduction à la programmation

Ça doit faire mal à la tête des 0 et des 1 toute la journée, non ?



Introduction à la programmation

- Langage bas niveau vs langage de haut niveau
 - Un langage de bas niveau est un langage qui est considéré comme plus proche du langage machine (binaire) plutôt que du langage humain. Il est en général plus difficile à apprendre et à utiliser mais offre plus de possibilité d'interactions avec le hardware de la machine.
 - Un langage de haut niveau est le contraire, il se rapproche plus du langage humain et est par conséquent plus facile à appréhender. Cependant les interactions se voient limiter aux fonctionnalités que le langage met à disposition

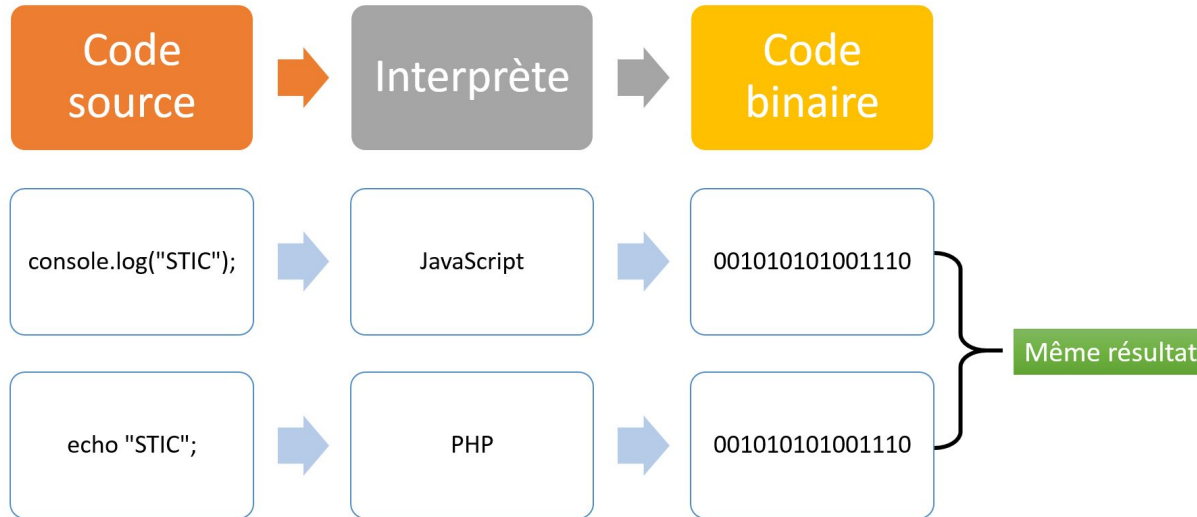


Introduction à la programmation

- Compilation vs Interprétation
 - La compilation d'un programme consiste à transformer toutes les instructions en langage machine avant que le programme puisse être exécuté. Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source.
 - Si un langage n'est pas compilé, il est nécessaire d'utiliser un interprète qui traduit les instructions en temps réel (on run time). Dans ce cas le code source est lu à chaque exécution et par conséquent les changements apportés au code seront pris en compte directement. La contrainte réside dans le fait que la machine faisant tourner le programme doit disposer de l'interprète de celui-ci.

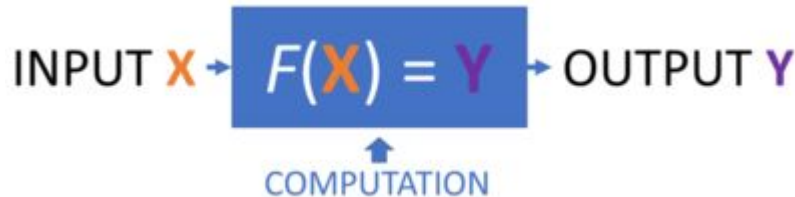
Introduction à la programmation

- Alors c'est pas des 0 et des 1 le code ?



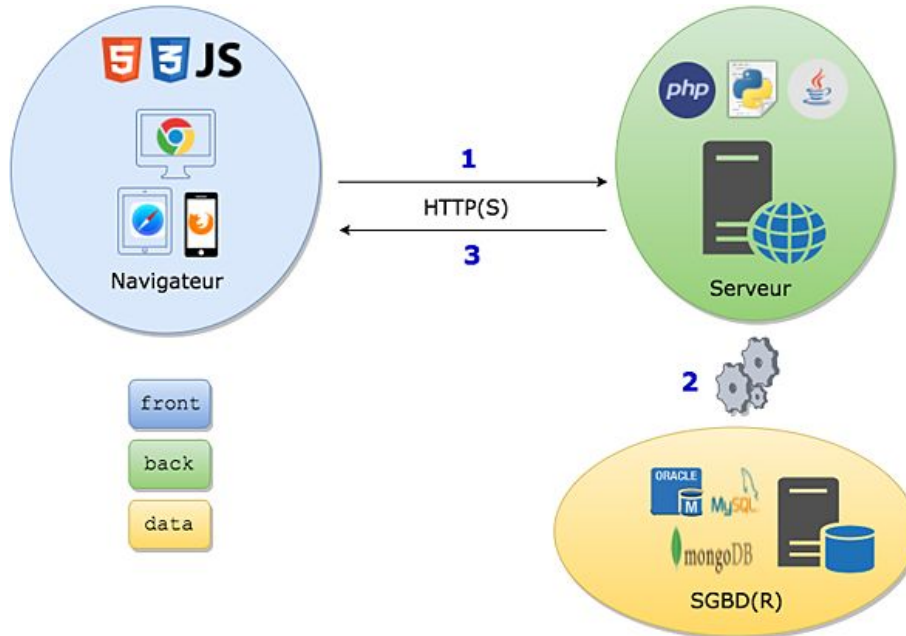
Introduction à la programmation

- Et les mathématiques dans tout ça ?
- Développer ne consiste pas à écrire des formules mathématiques (sauf si vous travailler sur des logiciels destinées aux sciences évidemment). On parle plutôt **d'algorithme**.
- Même si les langages sont différents, l'objectif principal d'un langage de programmation reste celui d'instruire la machine pour qu'elle produise des outputs conformes aux objectifs de l'application.
- On résume souvent par :
 - $F(X) = Y$, où ::
 - X représente l'input
 - Y représente l'output
 - $F()$ représente la fonction qui permet de transformer X en Y



Introduction à la programmation

- Anatomie d'une application interactive





Introduction à la programmation

- **Les éléments fondamentaux :**
 - Les variables
 - Les opérateurs
 - Les structures de contrôle (Conditions)
 - Les boucles
 - Les fonctions
 - Les tableaux (ou array)
 - Les objets



Introduction à la programmation

- **Les variables :**

Une variable permet d'identifier une valeur avec un nom.

On peut imaginer une boîte avec une étiquette. Une boîte peut contenir plusieurs éléments et l'étiquette sert en tant que référence de cette boîte. Il est possible de mettre des boîtes dans des boîtes.

Utilité des variables :

- Déclaration d'une variable, sert à insérer pour une référence dans l'application
- Affectation d'une variable, sert à associer une valeur à une variable

Les types de variables :

- Les **suites de caractères** (string) : elles sont utilisées pour représenter du texte ;
- Les **chiffres** (nombre entier, à virgule flottante, etc.) : ils sont utilisés surtout avec des opérateurs mathématiques ;
- Les **valeurs booléennes (en anglais : booleans)** : elles sont des valeurs dichotomiques (soit vrai, soit faux) ;
- Les **tableaux** (array) : ils sont utilisés pour créer des listes avec des indices qui permettent de récupérer la valeur associée;
- Les **objets** : ils sont des conteneurs qui peuvent inclure souvent tout type de données, y compris de sous-objets, des variables (i.e. des propriétés), ou des fonctions (i.e. méthodes)



Introduction à la programmation

- **Les opérateurs :**

Les opérateurs permettent de manipuler ou comparer des valeurs, notamment des variables. Parmi les opérateurs utilisés fréquemment dans tout langage de programmation vous trouverez :

- Les opérateurs **mathématiques** : addition, soustraction, multiplication, division, etc.
- Les opérateurs de **comparaison** (égalité, différence, majeur ou mineur)
- Les opérateurs **logiques** (AND et OR)



Introduction à la programmation

- **Les structures de contrôle :**

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la vérification d'une ou plusieurs conditions. La version sémantique la plus répandue des structures de contrôle est « **si... alors...** ». Le type de mécanisme de contrôle peut concerner différents niveaux de granularité. Par exemple :

- Si la note d'un examen est mineur de 4, alors la note est insuffisante
- Si le joueur n'a pas débloqué une porte, alors il ne peut pas accéder au niveau supérieur
- Si le mot de passe choisi contient moins de 6 caractères, alors il est trop court
- Si l'extension du fichier n'est pas .pdf, alors ne le téléverse pas sur le serveur

En général, elles sont utilisées pour déterminer si certaines instructions doivent s'exécuter ou non :

- Si cette condition s'avère, alors...
Exécute l'instruction #1
Exécute l'instruction #2
Exécute l'instruction #9
- Si non...
Exécute l'instruction #7
Exécute l'instruction #10



Introduction à la programmation

- **Les boucles :**

Les boucles sont à la base d'un concept très utile en programmation : l'**itération**.

L'itération permet d'exécuter de manière récursive (plusieurs fois) des instructions. Elles peuvent être très utiles par exemple pour appliquer un traitement à une liste d'éléments.

Exemple d'utilisation :

Tant que la liste n'est pas totalement parcourue :

 Modifier un élément de la liste

Ou encore :

Tant que ce chiffre ne dépasse pas 16 :

 Réalise un calcul



Introduction à la programmation

- **Les fonctions :**

Les fonctions représentent une sorte de "programme dans le programme", car elles sont la première forme d'organisation du code. On utilise des fonctions pour regrouper des instructions et les appeler sur demande : chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions. Pour accomplir ce rôle, le cycle de vie d'une fonction se divise en deux :

1. **Une phase unique dans laquelle la fonction est déclarée**

On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité

2. **Une phase qui peut être répétée une ou plusieurs fois dans laquelle la fonction est exécutée**

On demande à la fonction de mener à bien toutes les instructions dont elle se compose à un moment donné dans la logique de notre application



Introduction à la programmation

- **Les tableaux ou listes :**

Les tableaux (*array*) sont des listes indexées d'éléments qui partagent normalement une certaine relation sémantique pour appartenir à la liste. Un exemple tout simple d'array est la liste des courses : on peut indexer cette liste en fonction de l'ordre des articles à acheter :

1. Lait
2. Farine
3. Pommes
4. Fromage

Attention : l'index d'un tableau commence à 0. En prenant l'exemple précédent on dira que l'élément "Lait" est dans la première case du tableau qui est à l'index 0.

De la même façon l'élément "Fromage" est l'élément dans la 4^{ième} case du tableau qui se trouve à l'index 3.



Introduction à la programmation

- **Les tableaux ou listes :**

Les tableaux (*array*) sont des listes indexées d'éléments qui partagent normalement une certaine relation sémantique pour appartenir à la liste. Un exemple tout simple d'array est la liste des courses : on peut indexer cette liste en fonction de l'ordre des articles à acheter :

1. Lait
2. Farine
3. Pommes
4. Fromage

Attention : l'index d'un tableau commence à 0. En prenant l'exemple précédent on dira que l'élément "Lait" est dans la première case du tableau qui est à l'index 0.

De la même façon l'élément "Fromage" est l'élément dans la 4^{ième} case du tableau qui se trouve à l'index 3.



Programmation : apprendre via Robotmind

- Le principe de RobotMind est de manipuler des algorithmes en développant sans contrainte de langage complexe
- Fonctionnement :
 - Le robot à des instructions de base pour :



Se déplacer



Regarder



Peindre



Saisir



Programmation : apprendre via Robotmind

Exercices : A vous de jouer !



Introduction à la programmation

- **Les objets :**

On dit souvent en informatique qu'un langage est "orienté objets". Un **objet** est un élément qui possède des caractéristiques (**propriétés**) et qui peut normalement exécuter certaines fonctions (**méthodes**). Par exemple dans un jeu comme Super Mario Bros. Les différents ennemis qu'il faut éviter pendant le jeu sont des objets, et selon le type d'ennemi (c'est-à-dire le type d'objet), ils varient en fonction :

- De leur caractéristiques (propriétés) : la couleur, la forme, les coups nécessaires pour les faire disparaître, etc.
- Des actions (méthodes) qu'ils peuvent effectuer: sauter, lancer des objets, etc.

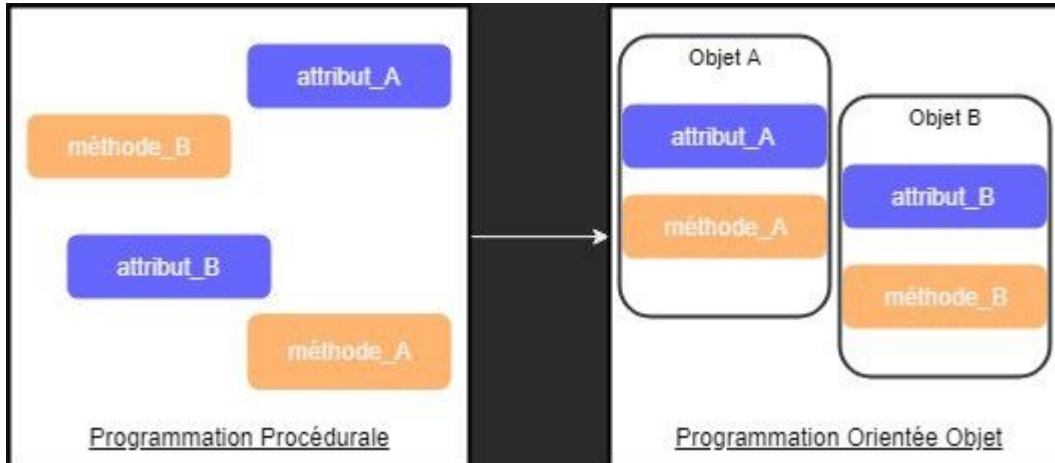
Chaque fois qu'un ennemi (objet) d'un certain type se présente à l'écran, il est une **instance** du type d'ennemi auquel il appartient. En tant qu'instance, il partage avec les autres ennemis de son type les mêmes propriétés et méthodes, mais il est en même temps unique, car lorsque vous le tuez, c'est seulement lui (et pas tous les ennemis de ce type) qui va disparaître.

Sur le même exemple, on peut identifier Mario et Luigi comme appartenant au même type d'objet : ils partagent les mêmes méthodes (sauter, courir, ...) mais ils sont différents dans certaines propriétés (couleur de la casquette, etc.).

Introduction à la programmation

- L'encapsulation

C'est ce qu'on appelle l'**encapsulation**. Cela permet de protéger l'information contenue dans notre objet et de le rendre manipulable uniquement par ses actions ou propriétés



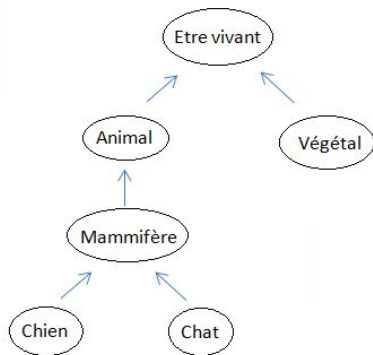


Introduction à la programmation

- L'héritage

Un objet dit « père » peut transmettre certaines de ses caractéristiques à un autre objet dit « fils ».

Pour cela, on pourra définir une relation d'héritage entre eux. S'il y a une relation d'héritage entre un objet père et un objet fils, alors **l'objet fils hérite de l'objet père**. On dit également que l'objet fils est une **spécialisation** de l'objet père ou qu'**il dérive de l'objet père**.





MODULE JAVA



1.

Présentation

Mettre en place le côté « Back » d'une application

- Utiliser le langage **Java**
- Savoir mettre en place des **Tests Unitaires** (TU)
- Savoir mettre en place une **connexion** avec une base de données
- Comprendre et mettre en œuvre **JEE** (Java Entreprise Edition)
- Comprendre et mettre en œuvre **JSF** (Java Server Faces) → **MVC**
- Comprendre et mettre en œuvre **JPA** (Java Persistence API)
- Comprendre et mettre en œuvre les **Web Services** (SOAP et REST)
- Comprendre et mettre en œuvre **Spring**



Cursus Java

Utiliser des outils

- Environnement de développement (IDE) : **Eclipse / VS Code**
- Base de données : **PostgreSQL**
- Utilitaire de gestion de projet Java : **Maven**
- Serveur d'application : **Tomcat**
- Utilitaire de test des Web Services : **SoapUI** et **Postman**
- Conteneurisation : **Docker**



Cursus Java

Utiliser des « frameworks »

- Accès aux données
- Tests
- Web
- Utilitaires
- ...



Cursus Java

Un projet **fil rouge** : l'application **CRM**

- Gestion de **Clients**
- Gestion de **Commandes**
- Liens entre les **Clients** et les **Commandes**
- Gestion des utilisateurs de l'application

→ En s'appuyant sur la base de données créée dans le module précédent



Cursus Java

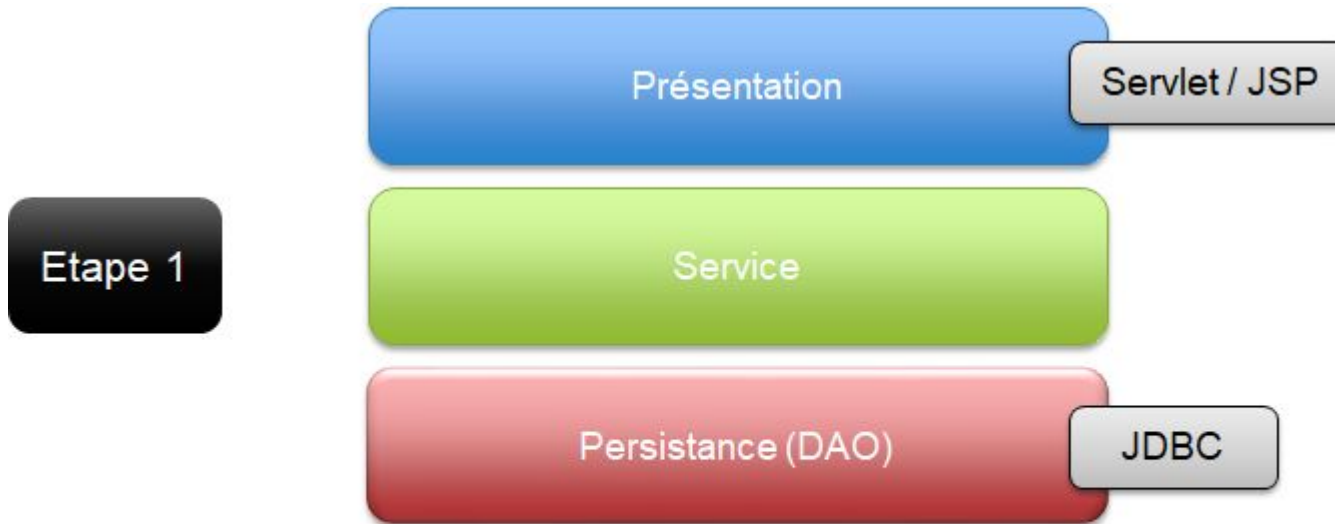
Un projet **fil rouge** : l'application **CRM**





Cursus Java

Un projet **fil rouge** : l'application **CRM**



Etape 1

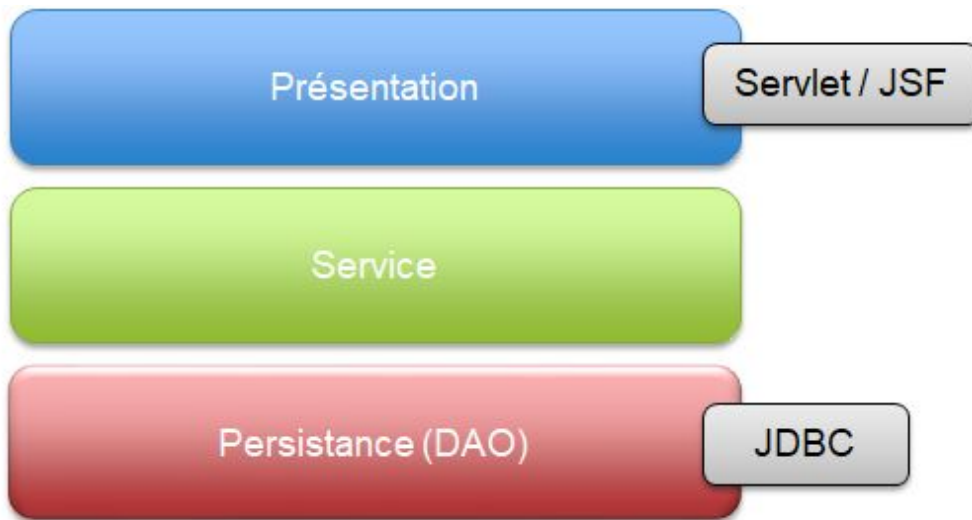




Cursus Java

Un projet **fil rouge** : l'application **CRM**

Etape 2

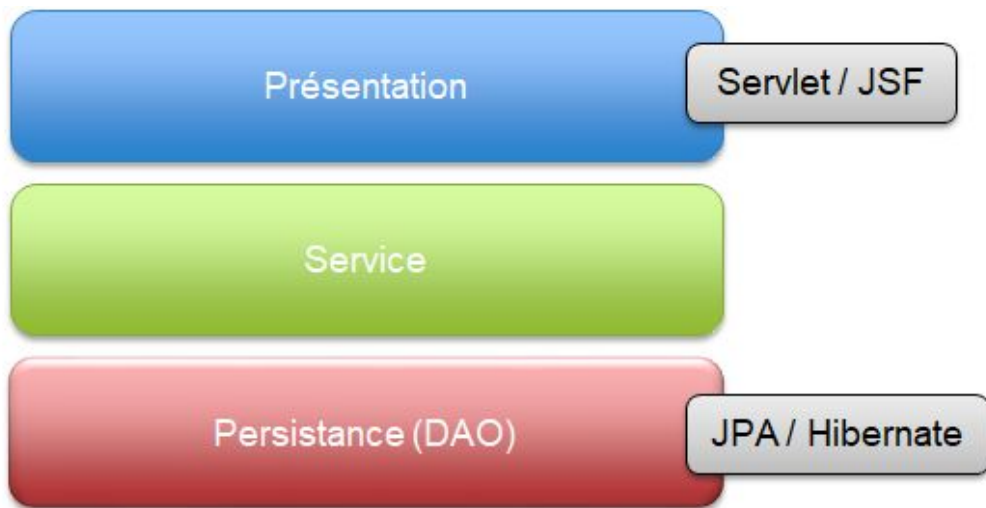




Cursus Java

Un projet **fil rouge** : l'application **CRM**

Etape 3

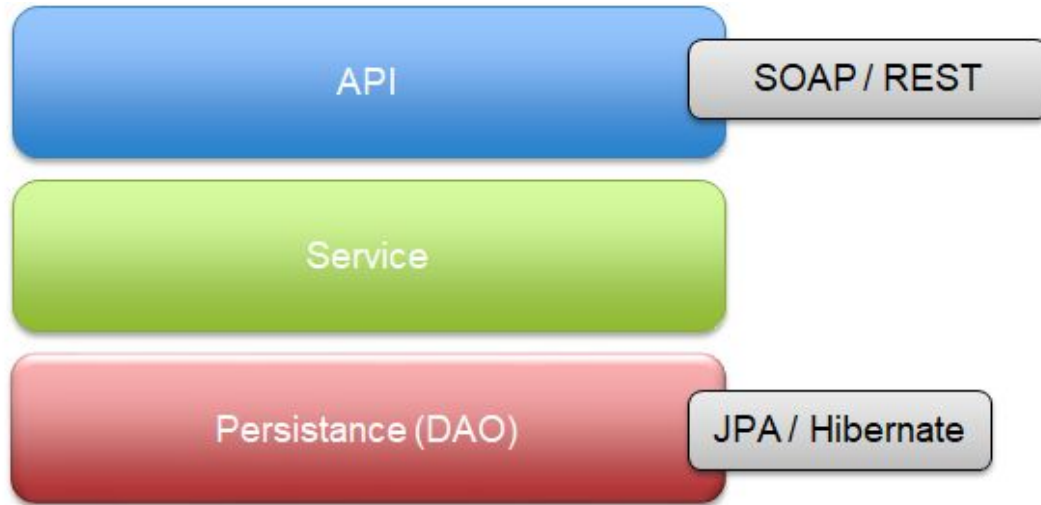




Cursus Java

Un projet **fil rouge** : l'application **CRM**

Etape 4

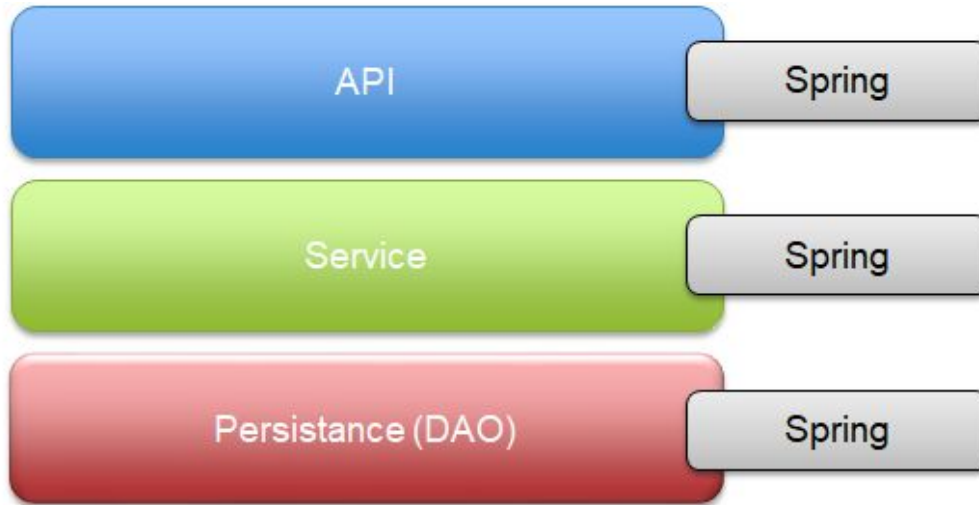




Cursus Java

Un projet **fil rouge** : l'application **CRM**

Etape 5





Cursus Java

Prérequis Java : JDK

- Installer une version **11** de la JDK en version **OpenJDK**
- La version Oracle est **payante** pour une utilisation en production depuis la version 11 (coût du support Oracle)
- **Préconisation** : privilégier la version **OpenJDK** qui est **gratuite**
- **Préconisation** : privilégier une version **LTS** (« Long Term Support »)



Cursus Java

Prérequis Java : JDK

- **Préconisation** : privilégier une version **LTS** (« Long Term Support »)
- Une version LTS est maintenue jusqu'à la sortie de la prochaine LTS (environ tous les 3 ans) → **patches correctifs et de sécurité !**
- Une version non-LTS n'est **plus maintenue** dès lors que la version suivante est mise à disposition



Cursus Java

Prérequis Java : IDE

- Utilisation de l'IDE Eclipse pour la formation
- D'autres IDE sont disponibles : IntelliJ, VS Code, ...
- Utiliser la dernière version d'Eclipse JEE : **Eclipse IDE for Enterprise Java Developers**

<https://www.eclipse.org/downloads/packages/>